НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

_____комп'ютерних наук_____

(назва кафедри)

____ _____Голуб Б.Л._____

(підпис) (ПІБ)

"__" червня 2025р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення VR-гри у жанрі RPG з використанням технологій NET**»**

Спеціальність 121 – «Інженерія програмного забезпечення» ОПП – «Інженерія програмного забезпечення»

Гарант освітньої програми

_____к.т.н, доцент_____ _____ __Вайганг Г.О._____

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____асистент_____ _____ __Баранова Т.А.

_____к.т.н. доцент_____ _____ ___ Даков С.Ю.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _Гаврилюк Денис Юрійович____

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОВИКОРИСТАННЯУКРАЇНИ

Факультетінформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

комп'ютерних наук

(назва кафедри)

_____/ Голуб Б.Л. доцент к.т.н./

(підпис)

"__" _____2025р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Гаврилюку Денису Юрійовичу

Спеціальність 121 – «Інженерія програмного забезпечення»

1. Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення VR-гри у жанрі RPG з використанням технологій NET» затверджена наказом ректора НУБіП України від 16.12.2024 № 2248 «С».
2. Термін подання завершеної роботи на кафедру 2025 . 06 . __

(рік, місяць, число)

3. Вихідні дані для роботи:
4. Перелік питань, що розглядаються:
    1. Системний аналіз предметної області
    2. Реалізація ключових ігрових механік
    3. Оптимізація та тестування продуктивності
    4. Висновки

Дата видачі завдання "__16___"_____12_____2024__ р.

Керівник бакалаврської кваліфікаційної роботи_____/Баранова Т.А., асистент /

Консультант бакалаврської кваліфікаційної роботи _____/Даков С.Ю., доцент, к.т.н/

(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:_____/ Гаврилюк Д.Ю. /__

(підпис) (прізвище та ініціали)

ЗМІСТ

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс програмування додатків)

DOF – Degrees of Freedom (ступені свободи)

HMD – Head-Mounted Display (шолом віртуальної реальності)

IK – Inverse Kinematics (зворотна кінематика)

LOD – Level of Detail (рівень деталізації)

NPC – Non-Player Character (неігровий персонаж)

RPG – Role-Playing Game (рольова гра)

SDK – Software Development Kit (набір інструментів розробки програмного забезпечення)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (досвід користувача)

VR – Virtual Reality (віртуальна реальність)

XR – Extended Reality (розширена реальність)

ВСТУП

Віртуальна реальність (VR) за останні роки перетворилася з експериментальної технології на повноцінну платформу для створення ігрових та інтерактивних проєктів. Особливий інтерес становить використання VR у жанрі рольових ігор (RPG), адже поєднання глибокого занурення та взаємодії робить ігровий процес більш захоплюючим для гравця. Використання сучасних

VR-пристроїв зі зростаючою продуктивністю та точним відстеженням рухів дозволяє створювати нові ігрові механіки, які відрізняються від традиційних ігор на інших платформах.

Актуальність теми. На нашу думку, розробка VR-RPG гри є актуальною з огляду на стрімке зростання ринку VR-технологій та потребу гравців у нових захоплюючих враженнях. По-перше, аналітики прогнозують стабільне зростання ринку VR-ігор (за деякими оцінками – до $40+ млрд до 2025 року), що зумовлено доступнішими шоломами (Meta Quest 3, Valve Index, PS VR2 тощо) та розширенням аудиторії. По-друге, сучасні гравці прагнуть отримати максимально реалістичний і захоплюючий досвід, який можливий саме у VR. Жанр RPG, з його фокусом на сюжет і розвиток персонажа, ідеально відповідає цій потребі. По-третє, технічний прогрес (повідомлення про точне позиціювання, відстеження жестів, висока роздільна здатність дисплеїв, частота оновлення 90–120 Гц) дозволяє реалізувати природні взаємодії гравця з ігровим світом. VR дає можливість впроваджувати інноваційні геймплей-механіки: реалістичну роботу зі зброєю, фізичну взаємодію з предметами та динамічні соціальні взаємодії з неігровими персонажами. Нарешті, попри великий потенціал, наявні VR-RPG-проєкти поки що не забезпечують одночасно повноцінний сюжет, систему прогресії і глибоку VR-взаємодію. Це створює нішу для нової розробки.

Мета та завдання роботи. Метою цієї роботи є створення повноцінної VR-ігри у жанрі RPG на базі .NET, що пропонуватиме гравцям іммерсивний досвід із природними механіками взаємодії, розвиненою системою прогресії персонажа та цікавим сюжетом. Для досягнення мети поставлено такі завдання:

- Проаналізувати предметну область: дослідити сучасний стан VR-технологій та характерні ознаки RPG-ігор.
- Визначити функціональні та нефункціональні вимоги до системи (перелік функцій, VR-комфорт, продуктивність тощо).
- Побудувати концептуальну і структуральну модель майбутньої системи за допомогою UML-діаграм (прецеденти, послідовності, активності, класи, пакети).

- Розробити логічну модель даних та фізичну архітектуру проєкту.
- Реалізувати програмні модулі гри з використанням Unity та .NET/C#, включаючи основні ігрові системи (рух, бій, інвентар, квести, діалоги тощо).
- Провести тестування ігрового додатку на відповідність вимогам (стабільний FPS, зручність VR-взаємодії тощо) та підготувати систему до запуску на цільових платформах.

Методи та інструменти. Для реалізації проєкту будуть використані сучасні технології: ігровий рушій Unity (звернення до C# та .NET), набір SDK для VR (OpenXR для більшості PC/VR пристроїв), бібліотеки для фізики та звуку. Процес розробки організовується за методологією Agile з ітеративним прототипуванням і тестуванням. Використані засоби: Visual Studio 2022 як середовище програмування, Git для контролю версій, системи відстеження помилок та задач. Для моделювання системи застосовано набір UML-діаграм, а інформаційну базу спроєктовано з використанням підходу ER-діаграм.

Апробація

Тези - Розробка VR-RPG гри з використанням технологій .NET та Unity.

Теоретичні та прикладні аспекти розробки комп’ютерних систем". Науково-практична конференція студентів і аспірантів.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області

Аналіз сучасного стану VR-технологій

Віртуальна реальність пройшла довгий шлях розвитку від концептуальних експериментів до масового споживчого продукту.

1.1.1 Станом на 2024-2025 роки ринок VR-технологій характеризується такими особливостями:

1. Стан VR-технологій. Сучасні VR-системи досягли високого рівня зручності та ефективності. Останні моделі шоломів (Meta Quest 3, Apple Vision Pro, Valve Index тощо) пропонують роздільну здатність дисплеїв понад 2K на око, широке поле зору (~120°) і точне відстеження руху (six degrees of freedom – 6DOF). Поява автономних (standalone) пристроїв (Meta Quest, Pico) спростила доступ до VR, а також продовжують розвиватися високопродуктивні PC VR платформи та консольні рішення (PlayStation VR2). Прогрес відстеження: від зовнішніх базових станцій до inside-out трекінгу, що дозволяє легко налаштовувати систему. Сучасні VR-шоломи відстежують не лише рухи голови й рук, а й пальців та очей, підвищуючи природність взаємодії. Важливими чинниками імерсивності є: затримка менше 20 мс (для комфортного відгуку), частота оновлення кадрів 72–120 Гц, субміліметрова точність трекінгу, висока роздільна здатність для реалістичної графіки. Завдяки стандарту OpenXR забезпечується сумісність з різними пристроями, що значно спрощує розробку кросплатформених VR-додатків.

2. Різноманітність платформ: Ринок VR-пристроїв можна розділити на три основні категорії:

- Автономні (standalone) пристрої — не потребують зовнішнього комп’ютера (Meta Quest 3, Pico 4)

- PC VR — вимагають підключення до потужного ПК (Valve Index, HTC Vive Pro)
- Консольні — інтегровані з ігровими консолями (PlayStation VR2)

Кожна з цих категорій має свої технічні обмеження та переваги, що впливають на процес розробки.

3. Технології відстеження: Еволюціонували від зовнішніх датчиків (external tracking) до відстеження «зсередини-назовні» (inside-out tracking), що суттєво спростило налаштування та використання VR-систем. Сучасні пристрої здатні відстежувати не лише рухи голови та рук, але й пальців, очей та навіть виразу обличчя користувача.

4. Чинники імерсивності: Ключовими параметрами, що впливають на якість VR-досвіду, є:

- Латентність (має бути менше 20 мс для комфортного досвіду)
- Частота оновлення (мінімум 60 Гц, оптимально 72-90 Гц)
- Точність відстеження (субміліметрова точність для природної взаємодії)
- Роздільна здатність (від 1832×1920 на око у сучасних пристроях)

5. Стандартизація: З появою OpenXR — відкритого стандарту для XR-додатків — індустрія рухається в напрямку більшої сумісності між різними пристроями, що спрощує розробку кросплатформених додатків.

1.1.2. Особливості жанру RPG у VR

- Персонаж і прогресія: головний герой з набором характеристик і навичок; система збору досвіду та підвищення рівня; наявність класів чи спеціалізацій; дерево талантів з новими здібностями.
- Наратив: нелінійний сюжет із гілками вибору; діалоги з вибором реплік; система основних і побічних квестів; ретельно опрацьований світ (лором) і персонажі.

- Ігрові системи: інвентар для зберігання предметів; економіка і торгівля; бойові системи (ввід покрокових та action); дослідження світу з пошуком скарбів.
- Соціальна взаємодія: відносини з NPC ; можливі супутники; наслідки вибору гравця для світу й персонажів.
- Світ: відкритий або локаційний ігровий світ; різноманітні середовища (міста, підземелля, біоми); наприклад, динамічна система дня/ночі і погоди для більшої правдоподібності.

1.1.3. Вимоги VR-розробки

Розробка для VR накладає специфічні вимоги на проєкт. Головне – забезпечити присутність гравця у світі та природність дій:

- Імерсивність: об’єкти мають реалістичний масштаб; рухи і взаємодії мають бути природними (наприклад, рух корпусом для переміщення, справжні жести для атаки чи захисту); звуковий дизайн (просторовий звук) підсилює орієнтацію у віртуальному просторі.
- Інтерфейс: традиційні 2D-меню не завжди зручні у VR. Переважають дієгетичні UI (інтерфейс вбудований у світ гри, наприклад, віртуальні панелі або планшети). Необхідно розробляти інтуїтивні 3D-інтерфейси, які гравець може «брати» та рухати. Тактильний відгук (вібрація контролерів) підвищує реалістичність.
- Технічні обмеження: VR вимагає високої продуктивності: 60–90 FPS без падінь і затримок. Потрібна оптимізація графіки та фізики (мінімізація складних шейдерів, подвійний рендеринг для двох очей). Враховуються обмеження різних пристроїв (від мобільних до потужних ПК).
- Комфорт: необхідно уникати різких рухів камери (щоб запобігти нудоті або кинетозу). Використовують альтернативні методи переміщення

(телепортація, плавний рух) та можливість налаштування параметрів (ширина поля зору, швидкість руху). Інтерфейси мають бути ергономічними, щоб зменшити фізичне навантаження при тривалій грі.

- Взаємодія: гравець маніпулює об’єктами фізично: хапає, кидає, відкриває. Для цього потрібно реалізувати реалістичну фізику взаємодії і природні жести (наприклад, захоплення об’єкта за реальну руку). Система відгуку через контролери доповнює занурення.

Таким чином, для успішного VR-RPG проєкту потрібно поєднати класичні механіки RPG з вимогами VR: глибока система розвитку персонажа та сюжет, насичений ігровий світ, а також зрозуміла та комфортна VR-взаємодія.

1.1.4. Ключові компоненти VR-RPG гри

Для успішної реалізації рольової гри у віртуальній реальності необхідно приділити особливу увагу таким ключовим компонентам:

1. Взаємодія з віртуальним світом:

- Інтуїтивне захоплення та маніпуляція об’єктами
- Природні способи взаємодії з елементами світу (двері, механізми, тощо)
- Тактильний зворотний зв’язок при взаємодії
- Можливість взаємодії з об’єктами різними способами (штовхання, кидання, обертання)

2. Система розвитку персонажа:

- Адаптація традиційних RPG-механік до VR-середовища
- Інтуїтивні та іммерсивні способи представлення характеристик персонажа
- Система навичок, орієнтована на фізичні дії користувача
- Візуальне відображення прогресу та розвитку персонажа

3. Сюжет та оповідь:

- Адаптація наративних прийомів для VR-середовища

- Система діалогів з природним вибором реплік
- Інтеграція сюжетних елементів у ігровий процес

4. Бойова система:

- Фізична взаємодія зі зброєю та спорядженням
- Реалістичні механіки бою, що використовують рухи користувача
- Баланс між реалізмом та ігровою умовністю
- Системи прицілювання та блокування атак через фізичні рухи
- Різноманітні типи зброї з унікальною механікою використання

1.2. Аналіз вимог до програмної системи

1.2.1. Функціональні вимоги:

Система повинна реалізувати ключовий ігровий процес ігрового світу у VR:

- Рух гравця: передбачено декілька режимів переміщення: телепортація (для зменшення дискомфорту) та плавний рух (для досвідчених користувачів). Повинна бути можливість перемикатися між режимами та налаштовувати їх.
- Взаємодія з об’єктами: реалізується природна взаємодія – захоплення предметів з реалістичною фізикою, можливість обертання, кидання об’єктів. Інтерактивні механізми (важелі, кнопки, двері) повинні реагувати на дії гравця; зброя та інструменти використовуються відповідно до їх фізичних властивостей.
- Інтерактивне оточення: середовища містять руйнівні елементи (об’єкти, що можуть бути знищені), реагують на дії гравця (наприклад, зміна звуків). Передбачено динамічне освітлення й тіні, фізику рідин і взаємодії з навколишнім світом.
- Система розвитку персонажа: присутня прогресія –очки характеристик для розподілу при зростанні рівня; різні типи зброї та їх механік

- Квестова система: є основна сюжетна лінія з гілками, а також побічні завдання з власними історіями. Система відстежує прогрес квестів, винагороджує гравця досвідом, предметами.
- Інвентар та екіпірування: гравець має інвентар.. Є можливість носіння спорядження на персонажі (зміна зброї).
- Система предметів : різні категорії предметів (зброя, броня, витратні матеріали). Є особливі унікальні предмети з унікальними властивостями. Передбачено торгівлю з NPC (купівля/продаж предметів).
- Діалогова система: гравець може спілкуватися з NPC. Інтерфейс діалогів представлений у VR-просторі (віртуальні панелі або текст у світі). Діалоги мають гілки з вибором реплік, варіанти діалогів залежать попередніх рішень.
- Взаємодія з NPC: NPC мають набір анімації під час діалогів; реагують на дії гравця у світі; NPC доступні для торгівлі та інших взаємодій поза діалогом.

1.2.2.Нефункціональні вимоги:

- Продуктивність: гра повинна підтримувати стабільну високу частоту кадрів (мінімум 60 FPS на цільових пристроях). Потрібна оптимізація для різних VR-платформ (ПК VR, автономні шоломи), мінімізація затримки процесора для плавного відтворення.
- Відгук системи: затримка між дією гравця та реакцією системи не повинна перевищувати 20–30 мс. Анімації мають бути плавними без ривків. Фізичний рушій повинен коректно та стабільно обробляти зіткнення та взаємодії.
- Комфорт VR-досвіду: система повинна мінімізувати ризик виникнення VR-хвороби: уникати непотрібних різких рухів камери, гарантувати стабільну частоту кадрів, підтримувати плавні переходи (наприклад, телепортація, плавне прискорення). Має бути можливість налаштування параметрів комфорту (ширина поля зору, швидкість руху, тип пересування).

- Ергономіка: інтерфейси мають бути інтуїтивними та природними. Мінімізувати фізичне навантаження при тривалій грі (наприклад, передбачити опори або паузи для відпочинку). Враховувати різні фізичні можливості користувачів (регулювання висоти персонажа).
- Надійність: система обробляє збереження/завантаження прогресу без втрат даних. Використовуються механізми резервування даних.
- Мінімальні вимоги: визначено мінімальні характеристики обладнання: шолом з відстеженням 6DOF, контролери з позиційним трекінгом, частота оновлення ≥ 60 Гц; для ПК – процесор Intel i5 (9-го покоління) або аналогічний AMD, 16 GB RAM, відеокарта не нижче GTX 1660.

Ця класифікація вимог закладає основу для проєктування архітектури системи та подальшої розробки. Враховані технологічні можливості VR та специфіка жанру RPG забезпечать повноцінне покриття функціональності.

1.3. Моделювання предметної області

Моделювання предметної області VR RPG-гри є ключовим етапом у процесі розробки програмного забезпечення, що дозволяє структурувати та візуалізувати складні взаємозв'язки між компонентами системи. Комплексний підхід із використанням різних типів UML-діаграм забезпечує всебічне розуміння архітектури, функціональності та динаміки проєктованої системи.

1.3.1. Функціональне моделювання системи

1.3.1.1. Діаграма прецедентів (Use Case)

Діаграма прецедентів відображає взаємодію користувача (гравця) із системою та визначає основні функціональні можливості VR RPG-гри. На рис. 1.1 представлена діаграма прецедентів, яка демонструє ключові функції, доступні користувачеві.

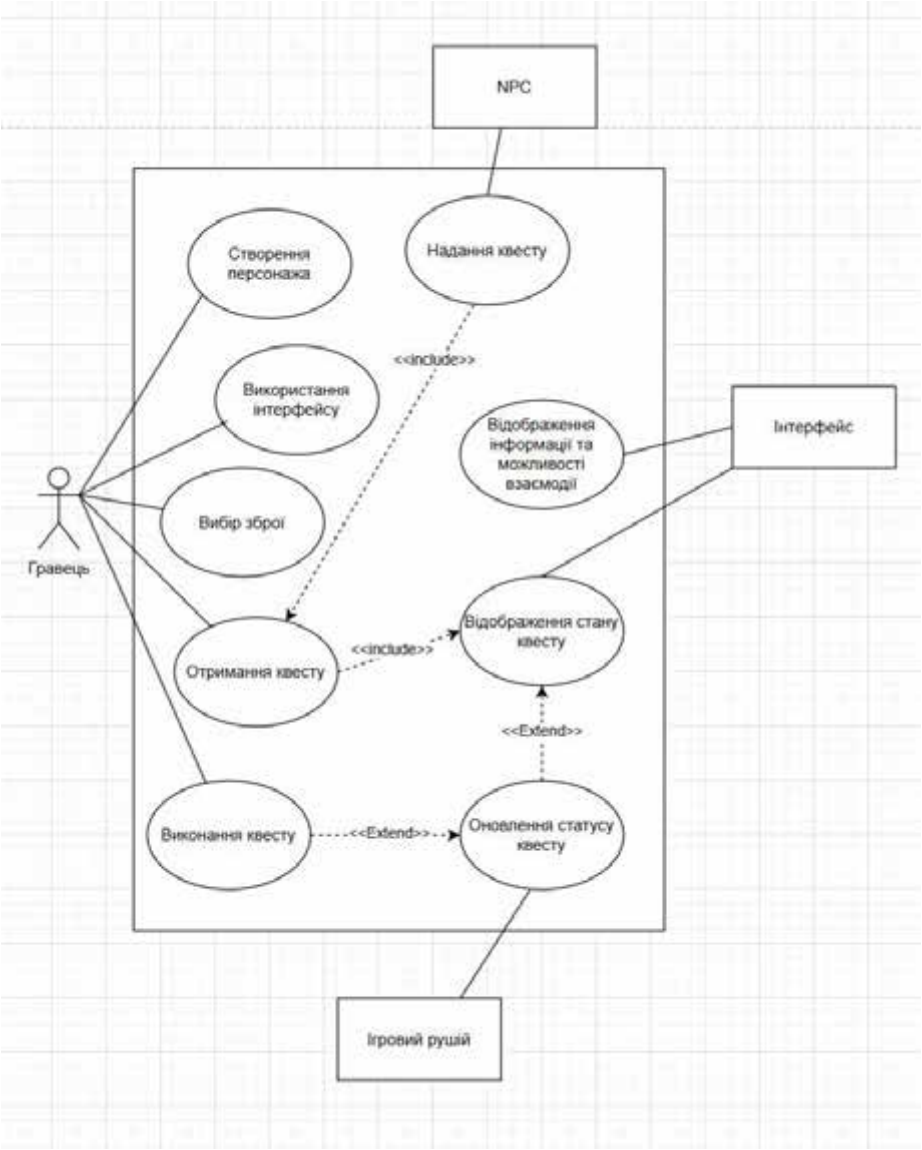


Рис. 1.1. Діаграма прецедентів VR RPG-гри

Актори:

- Гравець — створює персонажа, обирає зброю, приймає та виконує квести через UI.
- NPC — видає квест.
- Інтерфейс — показує опис квесту, прогрес та опції взаємодії.
- Ігровий рушій — оновлює статус квесту за подіями в грі.

Основні прецеденти:

- Створення персонажа, Вибір зброї, Використання інтерфейсу.
- Надання квесту і отримання квесту.
- Відображення інформації — опис, умови й нагороди.
- Виконання квесту — дії гравця в світі гри.
- Оновлення стану квесту (рушій) та його відображення (UI) — розширює ланцюг виконання.

1.3.1.2. Діаграма послідовності (Sequence Diagram)

Для більш детального аналізу поведінки системи розроблено діаграму послідовності, що моделює процес взаємодії гравця з NPC та отримання квесту (рис. 1.2).

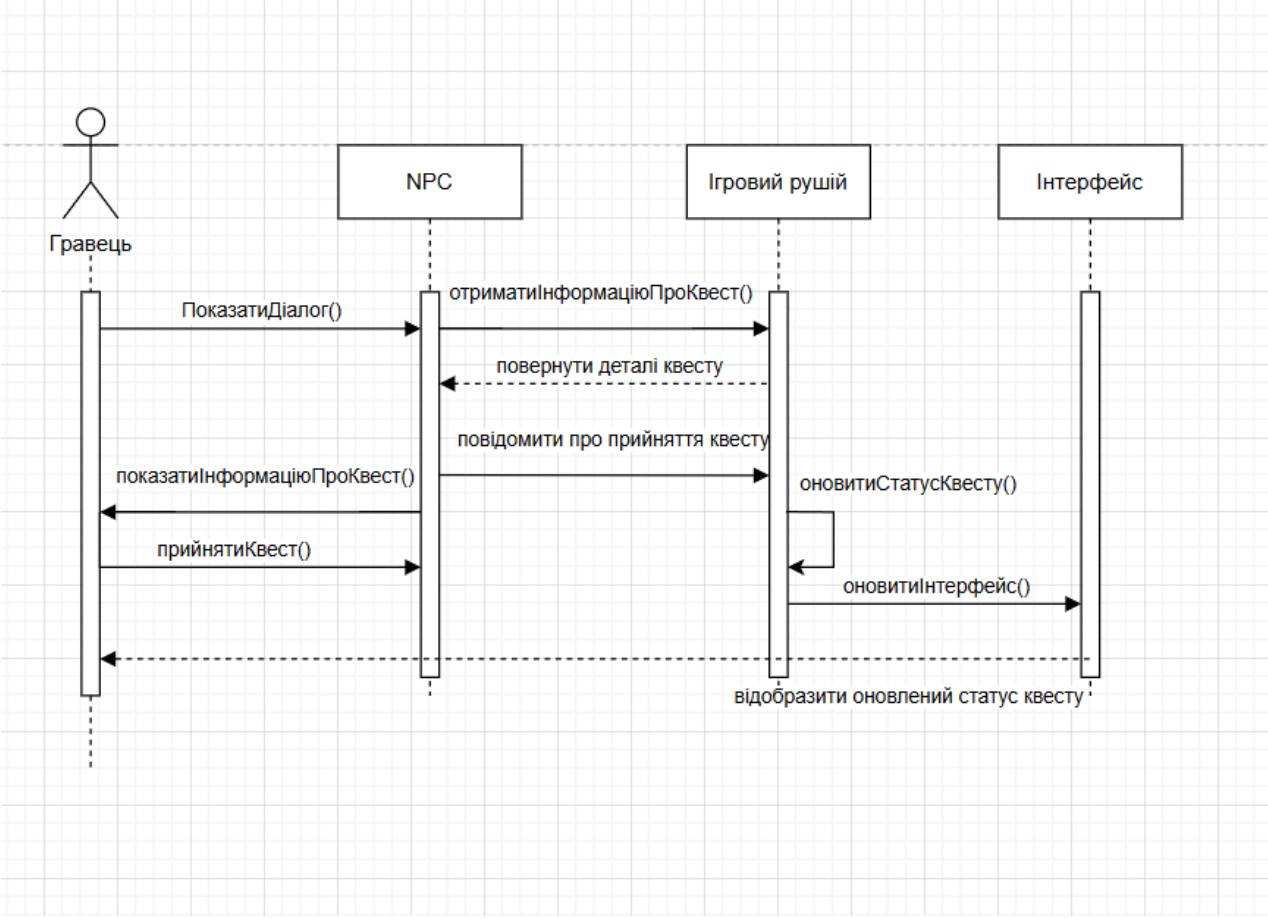


Рис. 1.2. Діаграма послідовності процесу отримання квесту

Діаграма послідовності демонструє хронологічний порядок обміну повідомленнями між об’єктами системи:

1. Гравець ініціює діалог з NPC
2. NPC надає доступні квести
3. Гравець обирає квест
4. Система реєструє квест у списку активних завдань гравця
5. NPC оновлює свій статус, позначаючи квест як виданий
6. Інтерфейс користувача відображає інформацію про новий квест

Важливим елементом діаграми є часова шкала, що дозволяє відстежити послідовність взаємодій та оцінити ефективність процесу. Потік повідомлень відображає як синхронні виклики (очікування відповіді), так і асинхронні події (коли відповідь не очікується негайно).

Використання діаграми послідовності дозволяє:

- Виявити потенційні проблеми у взаємодії компонентів
- Оптимізувати порядок викликів для підвищення продуктивності
- Забезпечити правильну синхронізацію дій у багатокомпонентній системі

1.3.1.3. Діаграма активності (Activity Diagram)

Діаграма активності описує динамічні аспекти поведінки системи, зокрема робочий процес проходження квесту в VR RPG-грі (рис. 1.3).

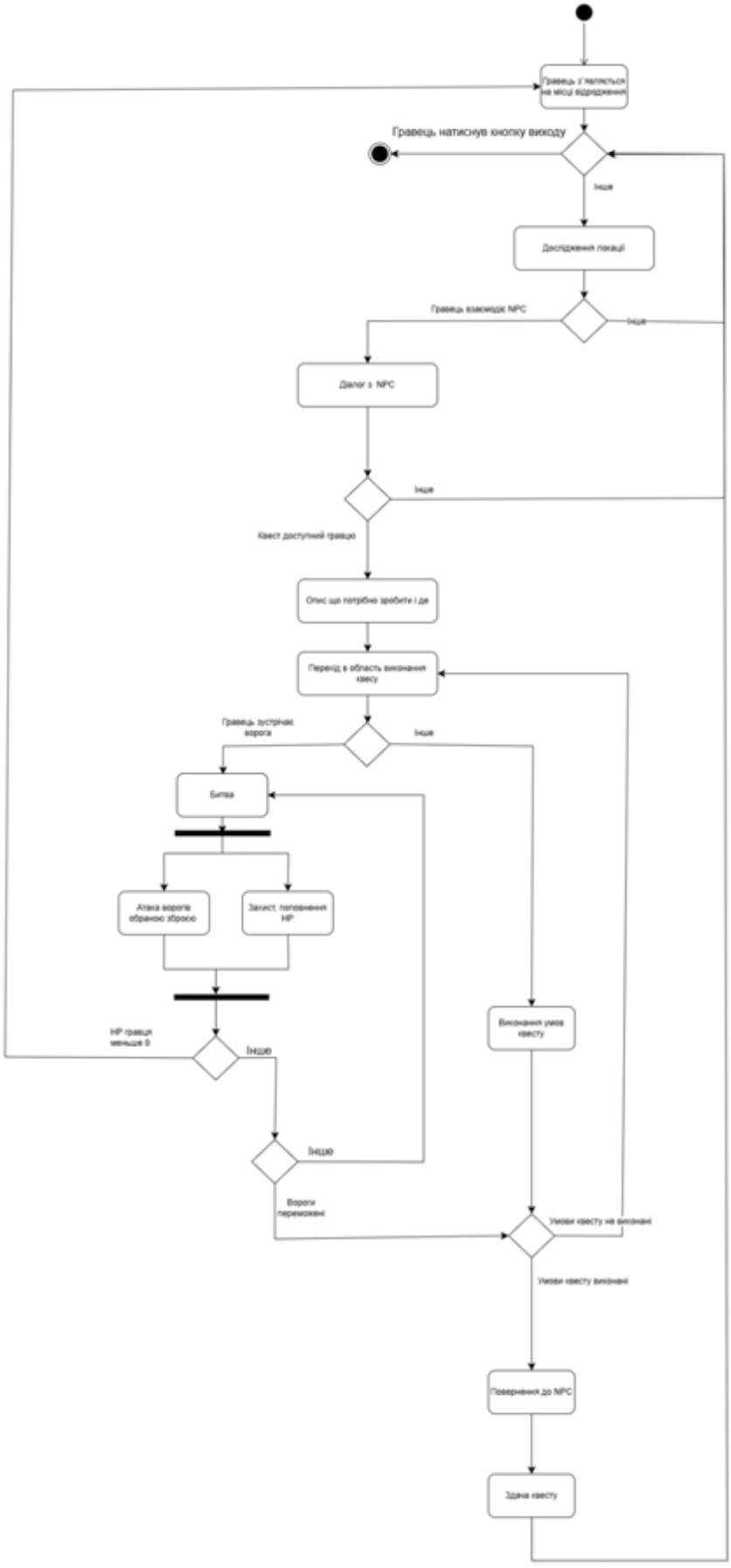


Рис. 1.3. Діаграма активності проходження квесту

Діаграма активності візуалізує послідовність дій, починаючи з отримання завдання і закінчуючи його виконанням:

1. Початок процесу – отримання квесту від NPC
2. Аналіз завдання – визначення мети та умов виконання
3. Переміщення до локації квесту
4. Розгалуження логіки залежно від типу квесту:
   - Знайти предмет
   - Перемогти ворога
   - Дослідити локацію
5. Виконання необхідних дій
6. Повернення до NPC для завершення квесту
7. Отримання винагороди

Особливістю діаграми є наявність точок прийняття рішень, що дозволяють моделювати різні сценарії виконання завдань. Також відображено паралельні процеси, наприклад, одночасний моніторинг здоров'я персонажа та прогресу квесту.

Діаграма активності надає цілісне представлення про логіку квестової системи та забезпечує основу для розробки алгоритмів і їх подальшої реалізації в коді.

1.3.2. Структурне моделювання системи

1.3.2.1. Визначення абстракцій предметної області

На основі аналізу предметної області VR RPG-гри були визначені ключові абстракції, що формують концептуальну основу системи (рис. 1.4).

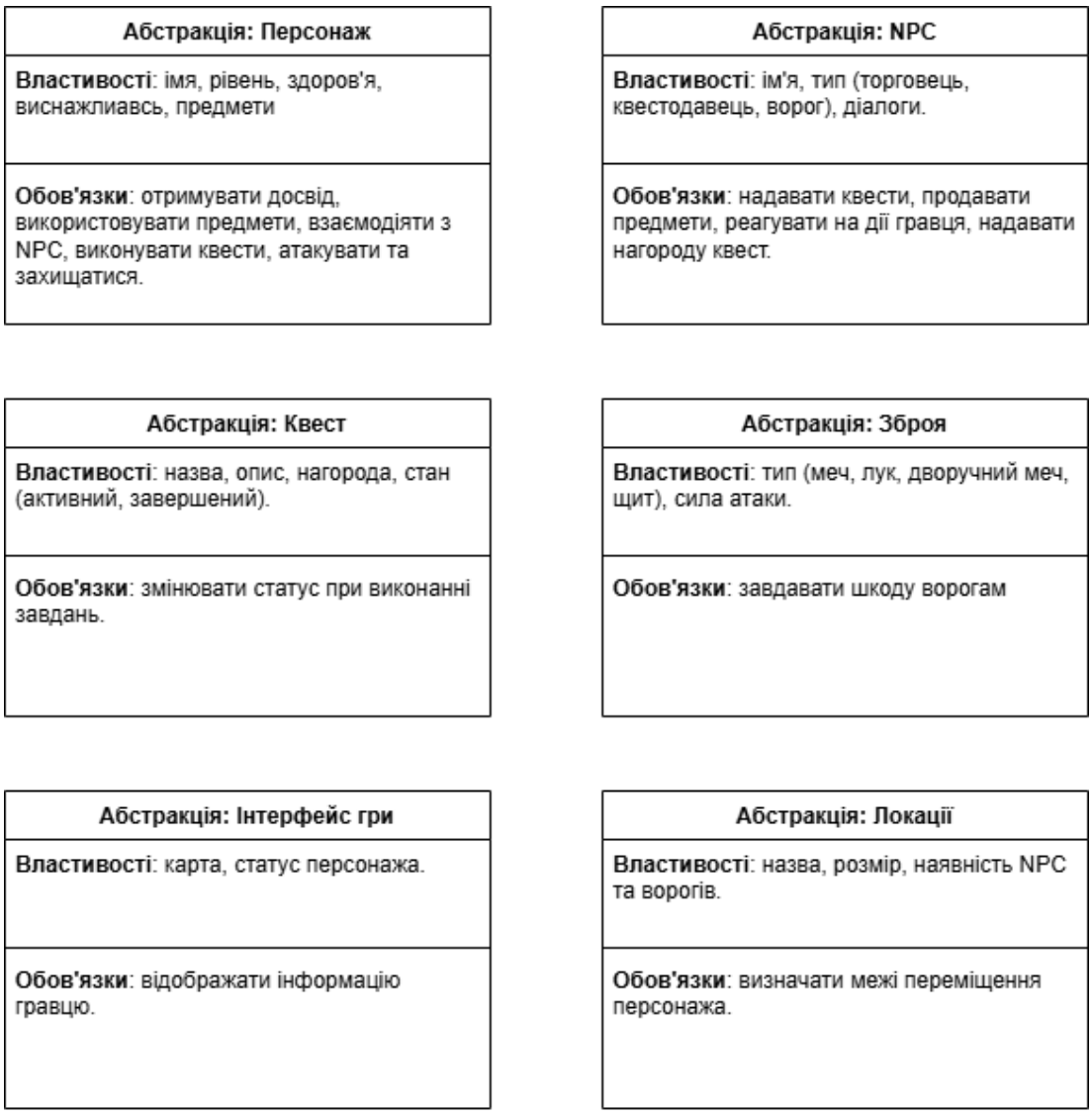


Рис. 1.4. Діаграма абстракцій предметної області

Виділені основні абстракції:

1. Персонаж – центральна сутність, керована гравцем, що характеризується показниками здоров'я, рівня та наявністю інвентаря
2. NPC – неігрові персонажі, які взаємодіють з гравцем, надаючи квести та інформацію
3. Квести – структуровані завдання з умовами виконання та винагородою
4. Зброя – предмети для бою, що мають різні характеристики (меч, лук, посох, тощо)

5. Інтерфейс – система відображення ігрової інформації та взаємодії з гравцем

6. Локації – просторові області ігрового світу з унікальними характеристиками

Визначені абстракції відображають ключові концепти домену VR RPG-ігор та слугують основою для побудови детальної об'єктної моделі системи.

1.3.2.2. Діаграма класів

Діаграма класів деталізує абстракції предметної області та представляє їх взаємозв'язки у вигляді структурованої моделі (рис. 1.5).

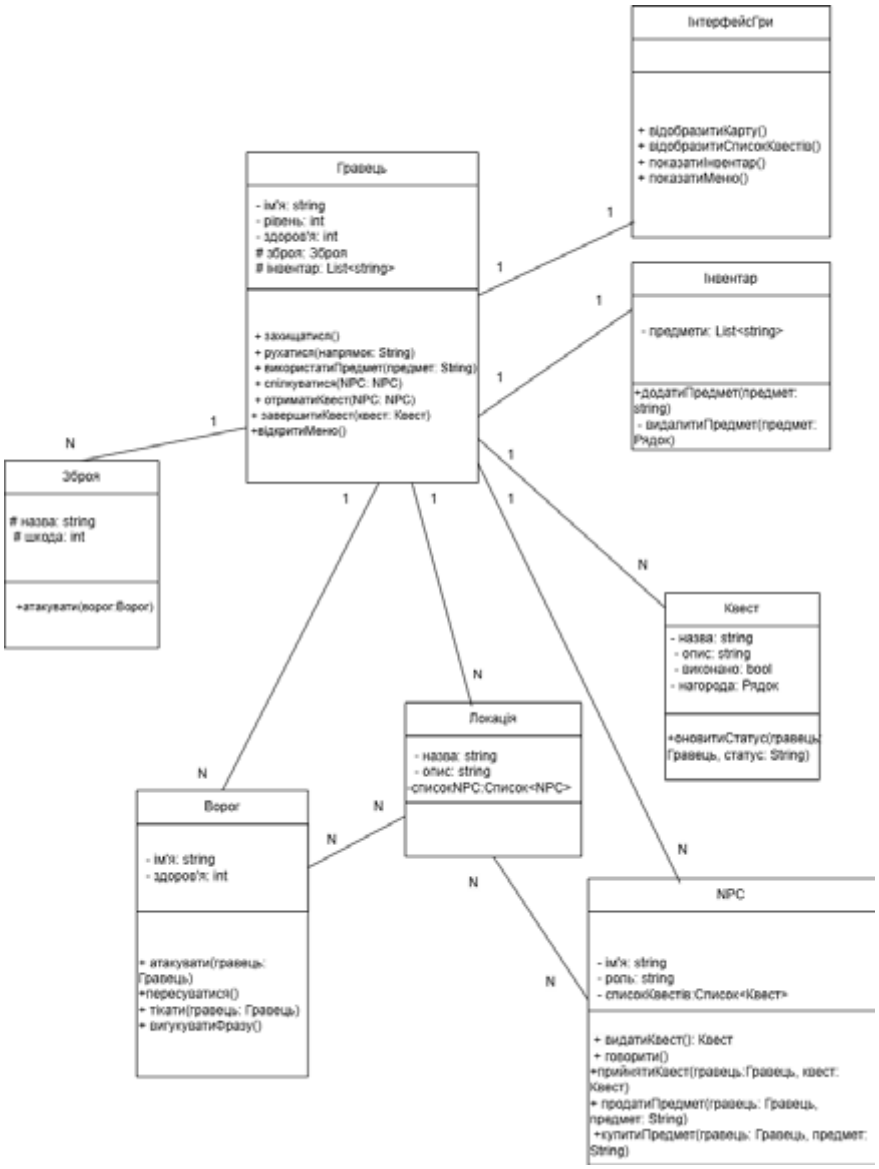


Рис. 1.5. Діаграма класів VR RPG-гри

Основні класи системи та їх взаємозв'язки:

1. Гравець (Player):
   - Атрибути: ім'я, рівень, здоров'я, мана, інвентар
   - Методи: атака(), захист(), переміщення(), взаємодія(), отриматиКвест(), завершитиКвест()
   - Зв'язки: композиція з Інвентарь, агрегація з Зброя, асоціація з Квест
2. NPC:
   - Атрибути: ім'я, роль, доступніКвести
   - Методи: видатиКвест(), розмовляти(), передатиПредмет()
   - Зв'язки: асоціація з Quest
3. Квест (Quest):
   - Атрибути: назва, опис, статус, нагорода
   - Методи: оновитиСтатус()
   - Зв'язки: асоціація з Player і NPC
4. Інвентар (Inventory):
   - Атрибути: список предметів
   - Методи: додатиПредмет(), видалитиПредмет(), використатиПредмет()
   - Зв'язки: композиція з Player
5. Зброя (Weapon):
   - Атрибути: назва, шкода, тип
   - Методи: атакувати()
   - Зв'язки: агрегація з Player, узагальнення різних типів зброї

1.3.2.3. Діаграма пакетів

Для забезпечення модульності та організації коду розроблено діаграму пакетів, що відображає логічне групування класів системи (рис. 1.6).

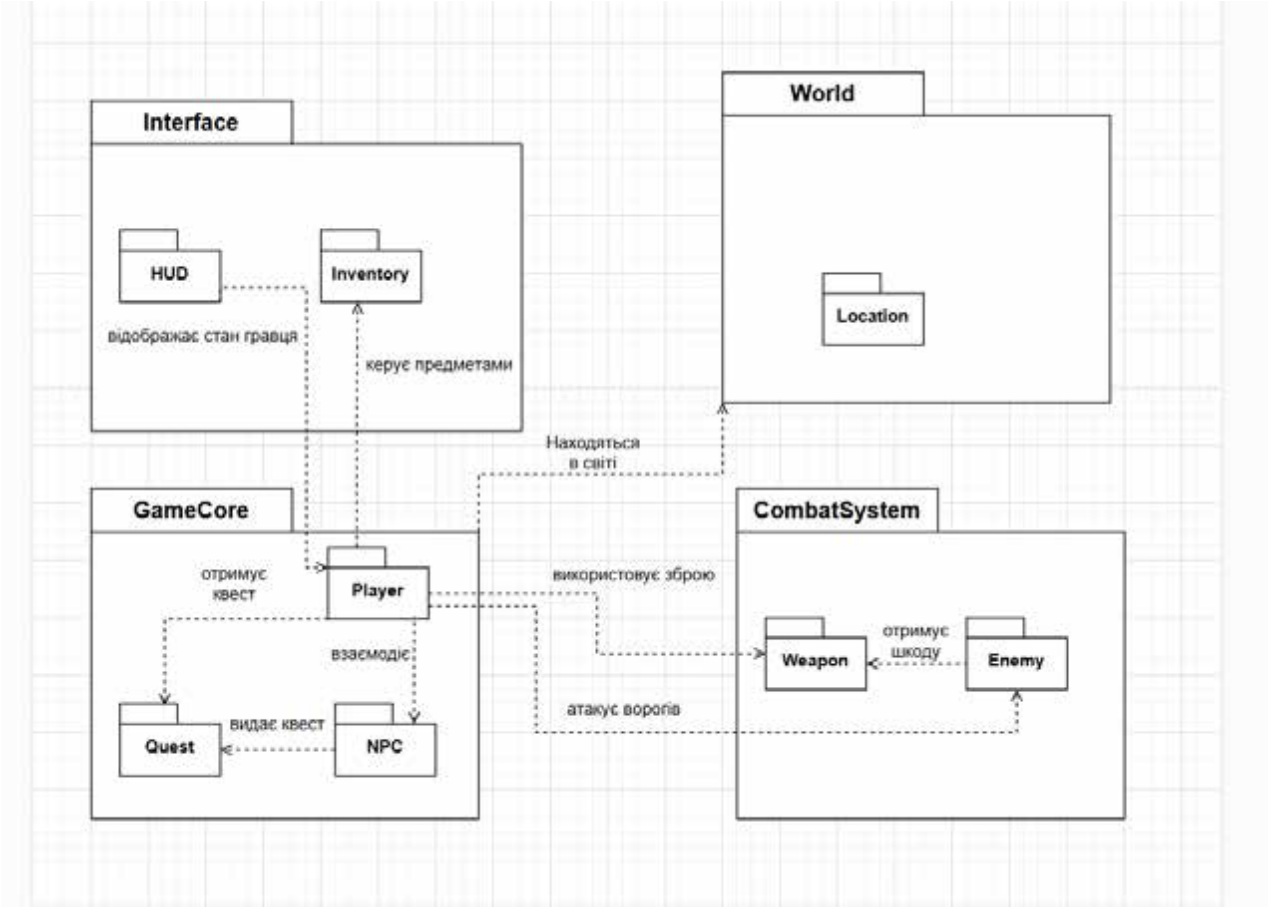


Рис. 1.6. Діаграма пакетів VR RPG-гри

Основні пакети системи:

1. GameCore – центральний пакет, що містить:
   - Player – клас гравця
   - NPC – класи неігрових персонажів
   - Quest – система квестів
2. CombatSystem – бойова система:
   - Weapon – класи зброї
   - Enemy – вороги та їх логіка
3. GameInterface – користувацький інтерфейс:
   - HUD – відображення статусу гравця
   - Inventory – система управління інвентарем
4. World – ігрове середовище:
   - Location – локації та області світу

Взаємозалежності між пакетами реалізовані за принципом мінімальної зв'язаності, що забезпечує незалежну розробку та тестування окремих модулів системи. Діаграма пакетів сприяє кращій організації проєкту та полегшує процес розробки через чітке розмежування відповідальності.

1.3.3 Висновки щодо моделювання предметної області

Проведене моделювання предметної області VR RPG-гри з використанням різних типів UML-діаграм забезпечує комплексне розуміння структури та поведінки системи. Розроблені моделі дозволяють:

1. Визначити функціональні вимоги через діаграми прецедентів, послідовності та активності, що описують взаємодію користувача з системою та потік операцій.
2. Структурувати архітектуру системи за допомогою діаграм класів і пакетів, що забезпечують модульність та слабку зв'язаність компонентів.
3. Спроєктувати фізичну реалізацію через діаграми компонентів і розгортання, які визначають організацію програмного коду та його розподіл на апаратних вузлах.

Розроблені моделі формують міцну основу для подальшої імплементації системи, забезпечуючи чітке розуміння її структури, функціональності та взаємозв'язків між елементами. Використання UML-діаграм також сприяє покращенню комунікації між розробниками та іншими зацікавленими сторонами проєкту.

Наступним етапом розробки буде конкретизація архітектурних рішень та імплементація описаних функцій і структур у програмному коді, з урахуванням специфіки VR-технологій та особливостей RPG-жанру.

1.4. Огляд інформаційних джерел та існуючих рішень

У цьому підрозділі проведено аналіз існуючих VR-ігор у жанрі RPG, розглянуто основні технології та інструменти для розробки VR-додатків, а також проаналізовано переваги та недоліки існуючих рішень.

Аналіз існуючих VR-ігор у жанрі RPG

На сучасному ринку VR-ігор представлено кілька помітних проєктів у жанрі RPG, які демонструють різні підходи до адаптації традиційних рольових механік для віртуальної реальності. Розглянемо найбільш значущі з них:

1. The Elder Scrolls V: Skyrim VR (Bethesda Game Studios, 2017) Skyrim VR є адаптацією популярної RPG для віртуальної реальності. Гра зберігає всі основні елементи оригіналу, включаючи відкритий світ, систему розвитку персонажа та квестову структуру.

Переваги:

- Величезний відкритий світ з сотнями годин ігрового контенту
- Повноцінна система розвитку характеристик та навичок
- Різноманітність ігрових активностей (бої, крафт, дослідження)

Недоліки:

- Обмежена VR-взаємодія (багато механік перенесено без адаптації)
- Застарілі візуальні ефекти
- Недостатня оптимізація для VR-платформ

2. Asgard's Wrath (Sanzaru Games, 2019) Asgard's Wrath є однією з найамбітніших VR-RPG, розроблених спеціально для віртуальної реальності. Гра пропонує повноцінну кампанію з елементами екшену та головоломок.

Переваги:

- Розроблено спеціально для VR з відповідними механіками

- Високоякісна графіка та візуальні ефекти
- Інноваційна система бою з фізичними взаємодіями

Недоліки:

- Лінійна структура рівнів (відсутній повноцінний відкритий світ)
- Обмежена система розвитку персонажа
- Доступна лише на платформі Oculus

3. Until You Fall (Schell Games, 2020) Until You Fall представляє собою рогалайк-RPG з процедурно-генерованими рівнями та акцентом на бойову систему.

Переваги:

- Інноваційна бойова система з фізичними рухами
- Висока реіграбельність завдяки процедурній генерації
- Відмінна оптимізація для різних VR-платформ

Недоліки:

- Обмежений наратив та розвиток сюжету
- Спрощена система прогресії персонажа
- Відсутність соціальних взаємодій з NPC

4. Blade & Sorcery (WarpFrog, 2018) Blade & Sorcery є симулятором бою з елементами RPG, що фокусується на реалістичній фізиці взаємодії зі зброєю та магією.

Переваги:

- Передова система фізичної взаємодії зі зброєю
- Глибокі механіки бою з урахуванням фізики
- Активна спільнота моддерів, що розширюють гру

Недоліки:

- Відсутність повноцінної сюжетної кампанії
- Обмежена система прогресії персонажа
- Фокус на боях замість комплексного RPG-досвіду

Порівняльний аналіз існуючих VR-RPG представлено в таблиці 1.1, де оцінено ключові аспекти кожної гри за 10-бальною шкалою.

Таблиця 1.1.

Порівняльний аналіз існуючих VR-RPG рішень

Гра	VR-взаємодія	Графіка	Бойова система	Прогресія персонажа	Сюжет	Відкритий світ
Skyrim VR	5	6	5	9	8	10
Asgard's Wrath	9	9	8	7	8	6
Until You Fall	8	7	9	6	4	3
Blade & Sorcery	10	7	10	5	2	5

Аналіз існуючих рішень демонструє, що наразі на ринку відсутні VR-RPG, які б одночасно пропонували глибоку систему розвитку персонажа, відкритий світ для дослідження та повноцінно використовували можливості VR-взаємодії. Це підтверджує актуальність розробки нової VR-RPG, що поєднає ці елементи.

Порівняльний аналіз рушіїв для розробки VR-ігор

Вибір ігрового рушія є критично важливим етапом при розробці VR-гри. Для аналізу було обрано два найпопулярніших рушія: Unity та Unreal Engine, які широко використовуються для створення VR-додатків.

Таблиця 1.2.

Порівняння ігрових рушіїв для розробки VR-ігор

Критерій	Unity	Unreal Engine
Мова програмування	C#	C++, Blueprint
Інтеграція з .NET	Нативна	Обмежена
Підтримка VR-платформ	Відмінна	Відмінна
Продуктивність VR	Добра	Відмінна
Візуальна якість	Добра	Відмінна
Крива навчання	Помірна	Висока
Інструменти для VR	Вичерпні	Вичерпні
Розмір спільноти	Дуже велика	Велика
Документація	Вичерпна	Добра
Ліцензування	Безкоштовно до певного порогу доходу	Роялті від доходу

На основі проведеного аналізу для розробки нашої VR-RPG було обрано рушій Unity з наступних причин:

- Нативна підтримка C# та екосистеми .NET
- Багатий набір інструментів для розробки VR
- Більш доступна крива навчання
- Відмінна документація та велика спільнота розробників
- Гнучкіша модель ліцензування для невеликих проєктів

Огляд VR SDK та фреймворків

Для забезпечення взаємодії з різними VR-пристроями існує кілька спеціалізованих SDK та фреймворків. В таблиці 1.3 представлено порівняльний аналіз основних VR SDK, доступних для інтеграції з Unity та .NET.

Таблиця 1.3.

Порівняльний аналіз VR SDK та фреймворків

SDK/Фреймворк	Підтримувані платформи	Інтеграція з Unity	Інтеграція з .NET	Простота використання	Активність розробки
OpenXR	Універсальний	Відмінна	Добра	Середня	Висока
SteamVR/OpenVR	SteamVR-сумісні	Відмінна	Добра	Середня	Висока
Oculus Integration	Oculus/Meta	Відмінна	Добра	Висока	Висока
Mixed Reality Toolkit	Windows MR, різні	Відмінна	Відмінна	Середня	Висока

На основі аналізу для розробки нашої VR-RPG було обрано технологію:

OpenXR як базовий API для забезпечення сумісності з різними VR-пристроями

Така комбінація забезпечить максимальну сумісність з різними VR-платформами при збереженні високої продуктивності та зручності розробки.

Аналіз технологій взаємодії у віртуальній реальності

Однією з ключових проблем при розробці VR-ігор є створення інтуїтивних та комфортних механік взаємодії користувача з віртуальним світом. В таблиці 1.4 проаналізовано основні технології та підходи до реалізації такої взаємодії.

Таблиця 1.4.

Аналіз технологій VR-взаємодії

Технологія	Переваги	Недоліки	Застосування в RPG
Захоплення об'єктів (Grab)	Інтуїтивність, реалістичність	Вимагає точного відстеження	Інвентар, предмети, зброя
Телепортація	Комфорт, запобігання кінетозу	Менш іммерсивна, ніж вільний рух	Переміщення на великі відстані
Вільний рух (Locomotion)	Максимальна іммерсивність	Ризик VR-хвороби	Дослідження світу
Вказівка променем (Ray casting)	Взаємодія на відстані	Менш реалістична	Діалоги, UI, вибір цілей
Фізична взаємодія	Високий рівень іммерсивності	Технічно складна реалізація	Бойова система, головоломки

На основі аналізу для нашої VR-RPG було обрано комбінований підхід, що включає:

- Фізичне захоплення для взаємодії з об'єктами ближньої зони
- Основний метод переміщення режим вільного руху
- Фізичну взаємодію для бойової системи з реалістичним відчуттям зброї

Переваги та недоліки існуючих рішень

Аналіз існуючих VR-RPG виявив ряд загальних проблем та обмежень, які потребують вирішення при розробці нового проєкту:

Загальні недоліки існуючих VR-RPG:

1. Недостатня адаптація RPG-механік для VR:

- Багато ігор просто переносять традиційні інтерфейси та механіки в VR

- Меню та інвентар часто не використовують потенціал 3D-простору
- Бойові системи не повністю адаптовані до природних рухів користувача

2. Технічні обмеження:
- Компроміс між візуальною якістю та продуктивністю
- Обмежена інтерактивність світу через ресурсні обмеження
- Спрощені фізичні моделі для підтримки стабільного FPS

3. Проблеми з комфортом користувача:
- Недостатня увага до запобігання VR-хворобі
- Втома від тривалого фізичного навантаження
- Непродумана ергономіка інтерфейсів

4. Обмеження RPG-систем:
- Спрощені системи прогресії порівняно з традиційними RPG
- Обмежені можливості соціальної взаємодії з NPC
- Менша глибина квестів та ігрового світу

Виявлені недоліки допомогли сформулювати вимоги та підходи до розробки нашої VR-RPG, спрямовані на подолання цих обмежень:

Пропоновані вдосконалення:

1. Інноваційні VR-інтерфейси:
- Розробка дієгетичних інтерфейсів, вбудованих у світ гри
- Використання фізичних репрезентацій для меню та інвентарю
- Природні жести для доступу до ігрових функцій

2. Адаптація RPG-механік для VR:
- Система розвитку персонажа з візуальною репрезентацією у VR
- Фізична бойова система з урахуванням навичок персонажа
- Інтуїтивні механіки крафту та взаємодії з предметами

3. Технологічні вдосконалення:
- Використання адаптивного LOD для оптимізації продуктивності
- Розробка ефективної системи фізики об'єктів
- Застосування оптимізаційних технік для стабільного FPS

4. Фокус на комфорт користувача:

- Розробка системи налаштувань для різних рівнів комфорту
- Впровадження механік, що запобігають VR-хворобі
- Зменшення фізичного навантаження для тривалих ігрових сесій

1.5. Постановка завдання

На основі проведеного аналізу предметної області, вивчення існуючих рішень та визначення вимог до програмної системи можна сформулювати конкретні завдання даної кваліфікаційної роботи.

Мета розробки

Розробити повноцінну VR-гру у жанрі RPG з використанням технологій .NET, що пропонуватиме гравцям імерсивний ігровий досвід з інтуїтивними механіками взаємодії, розвиненою системою прогресії персонажа та захоплюючим наративом.

Вимоги до розроблюваної VR-гри

1. Загальні вимоги:

- Цільові платформи: PC VR (SteamVR-сумісні пристрої) та Oculus/Meta Quest
- Підтримка контролерів руху з 6DOF відстеженням
- Мінімальна частота кадрів: 60 FPS
- Мінімальна тривалість проходження: 6-8 годин

2. Системні вимоги:

- Ігровий рушій: Unity 2022.1 або новіший
- Мова програмування: C# (.NET)
- VR SDK: OpenXR з інтеграцією VRTK
- Мінімальні вимоги до ПК: процесор Intel i5 9-го покоління або аналогічний AMD, 16 GB RAM, GPU NVIDIA GTX 1660 або аналогічна

3. Механіки та функціональність:

- Система пересування: телепортація та опціональний вільний рух

- Бойова система: фізична взаємодія зі зброєю
- Інвентар: фізична взаємодія з предметами, реалістичне екіпірування
- Система розвитку: характеристики, навички, таланти
- Діалогова система: інтерактивні розмови з NPC
- Квестова система: основні та побічні завдання
- Система крафту: створення та вдосконалення предметів

Обсяг та розділи роботи

В рамках кваліфікаційної роботи було створено функціональний прототип VR-гри у жанрі RPG на платформі Unity з використанням технологій .NET та OpenXR. Важливо відзначити суттєві обмеження, які вплинули на обсяг реалізованого функціоналу:

1. Часові обмеження: розробка здійснювалась в межах одного навчального семестру.
2. Ресурсні обмеження: проєкт реалізовувався одноосібно, без залучення команди спеціалістів
3. Технічна складність: VR-розробка вимагає інтеграції специфічних технологій та постійного тестування на цільовому обладнанні

Варто підкреслити, що розробка повноцінної комерційної VR-гри у жанрі RPG зазвичай потребує:

- Команди з 10-20+ спеціалістів (програмісти, дизайнери, 3D-моделери, аніматори, тестувальники)
- Термінів розробки від 12 до 36 місяців
- Значного бюджету для покриття всіх виробничих витрат

Запланований функціонал (згідно розділу 2)

Проєктне рішення передбачало створення комплексної VR-RPG системи з наступними компонентами:

- Система прогресії персонажа: рівні, характеристики, дерево навичок
- Діалогова система: розгалужені діалоги з NPC з впливом на ігровий світ

- Система квестів: нелінійна логіка, залежні та незалежні завдання
- Інвентар та економіка: збір, зберігання, торгівля предметами
- Бойова система: фізична взаємодія зброї, різні типи ворогів, тактичні елементи
- Система збереження прогресу: збереження стану персонажа та світу
- VR-навігація та взаємодія: телепортація, плавне пересування, фізична взаємодія
- AI та анімації NPC: поведінкова логіка, реакції на гравця, анімаційні системи

Реалізовані компоненти в межах дипломної роботи

Фокус розробки був зосереджений на створенні функціонального ядра гри та найбільш критичних VR-механік, що формують основу для подальшого розвитку:

1. Фізично-реалістична VR-взаємодія:
- Імплементація XR-підсистеми для захоплення та маніпуляції об'єктами
- Реалістичні анімації рук з динамічною адаптацією до контролерів
- Фізична взаємодія об'єктів з навколишнім світом
2. Інтерфейс та інвентар користувача:
- Адаптований для VR інтерфейс інвентаря з просторовою взаємодією
- Логіка управління предметами (додавання, стекування, видалення)
- Взаємодія з об'єктами інвентаря через природні рухи в VR
3. Система ближнього бою:
- Фізична механіка володіння холодною зброєю (меч, сокира)
- Система виявлення зіткнень та нанесення пошкоджень
- Компонент SliceObject для взаємодії зброї з руйнованими об'єктами
4. Система дистанційного бою:
- Реалістична механіка стрільби з лука

- Фізичне моделювання польоту стріл
- Система прицілювання та натягування тятиви

5. Інтерактивні активності:

- Міні-ігри для демонстрації VR-взаємодії
- Інтерактивні елементи навколишнього середовища

6. Базовий віртуальний світ:

- Оптимізований ландшафт з використанням Unity Terrain
- Інтеграція води, рослинності та природних елементів
- Оптимізація для стабільної продуктивності у VR

7. OpenXR-інтеграція та конфігурація:

- Налаштування XR Interaction Toolkit для різних пристроїв
- Конфігурація XR Origin та систем відстеження
- Імплементація комфортних VR-механік пересування

Обґрунтування обсягу реалізації

Реалізовані компоненти формують функціональне ядро VR-гри, демонструючи технічну спроможність до створення повноцінного продукту. Основний акцент було зроблено на:

1. Технічній реалізації ключових VR-механік, що є фундаментом для будь-якої VR-гри у жанрі RPG
2. Забезпеченні оптимальної продуктивності для комфортного VR-досвіду
3. Створенні архітектури, що дозволяє масштабувати проєкт у майбутньому

Висновок

Поточна реалізація демонструє професійний підхід до VR-розробки та створює міцний фундамент для подальшого розвитку проєкту. Всі реалізовані компоненти відповідають архітектурі та концепції, описаній у розділі 2, а обсяг реалізації є адекватним для:

1. Одноосібної розробки в обмежених часових рамках
2. Демонстрації технічної спроможності до створення VR-проєкту
3. Підтвердження життєздатності концепції та обраних технологічних рішень

Результати роботи можуть слугувати основою для подальшого розвитку проєкту в межах комерційної розробки або наукових досліджень у сфері VR-технологій та інтерактивних розважальних систем.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Логічна модель даних у вигляді ER-діаграми

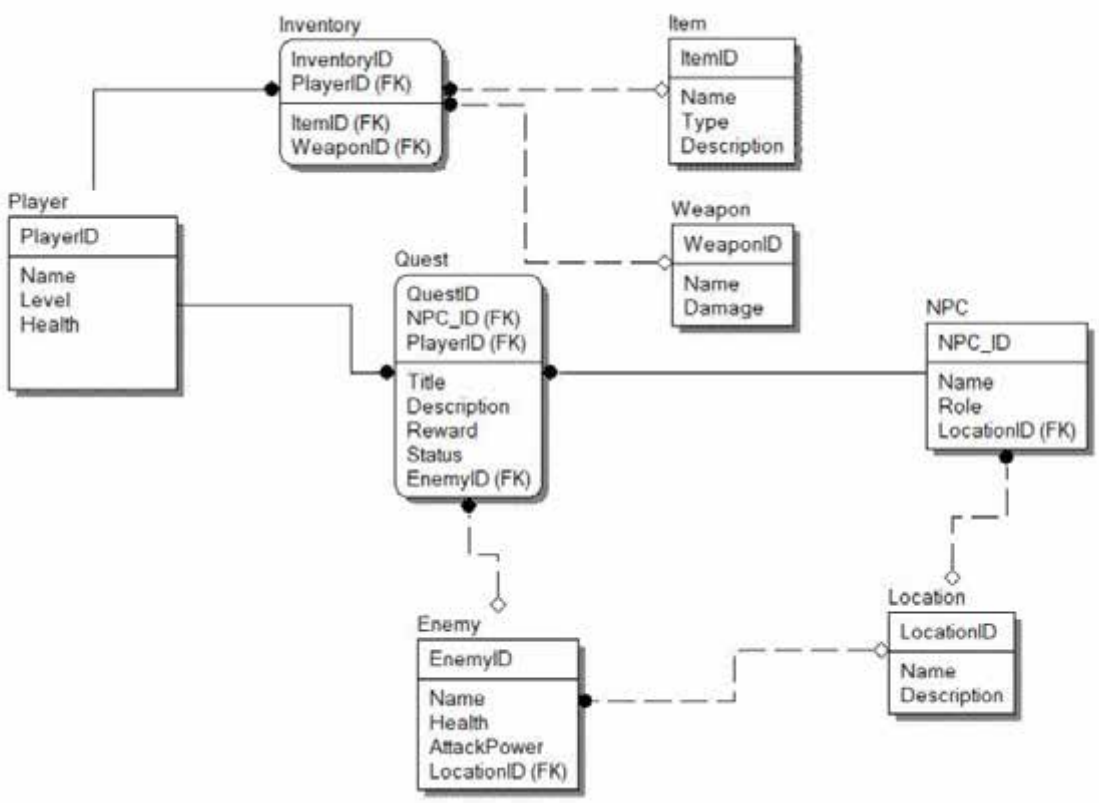


Рис. 2.1. Логічна модель даних VR RPG-гри

Логічна модель даних для VR RPG-гри представлена у вигляді ER-діаграми (Entity-Relationship Diagram), яка відображає основні сутності системи та зв'язки між ними.

Основні сутності:

1. Гравець – центральна сутність, що містить інформацію про персонажа користувача:

- Ідентифікатор
- Ім'я

- Рівень
- Здоров’я

2. NPC – неігрові персонажі, які взаємодіють з гравцем:
- Ідентифікатор
- Ім’я
- Тип (торговець, квестодавець, ворог)
- Локація

3. Квест – завдання для гравця:
- Ідентифікатор
- Назва
- Опис
- Статус (активний, виконаний, провалений)
- Винагорода

4. Предмет – об’єкти, які можуть бути у гравця в інвентарі:
- Ідентифікатор
- Назва
- Тип (зброя, броня, витратний матеріал)
- Опис

5. Локація – ігрові простори:
- Ідентифікатор
- Назва
- Опис
- Тип (місто, підземелля, ліс)

Ця логічна модель даних забезпечує структуровану основу для проєктування інфомаційної системи та взаємодії компонентів у системі.

2.2. Діаграма класів та кооперацій

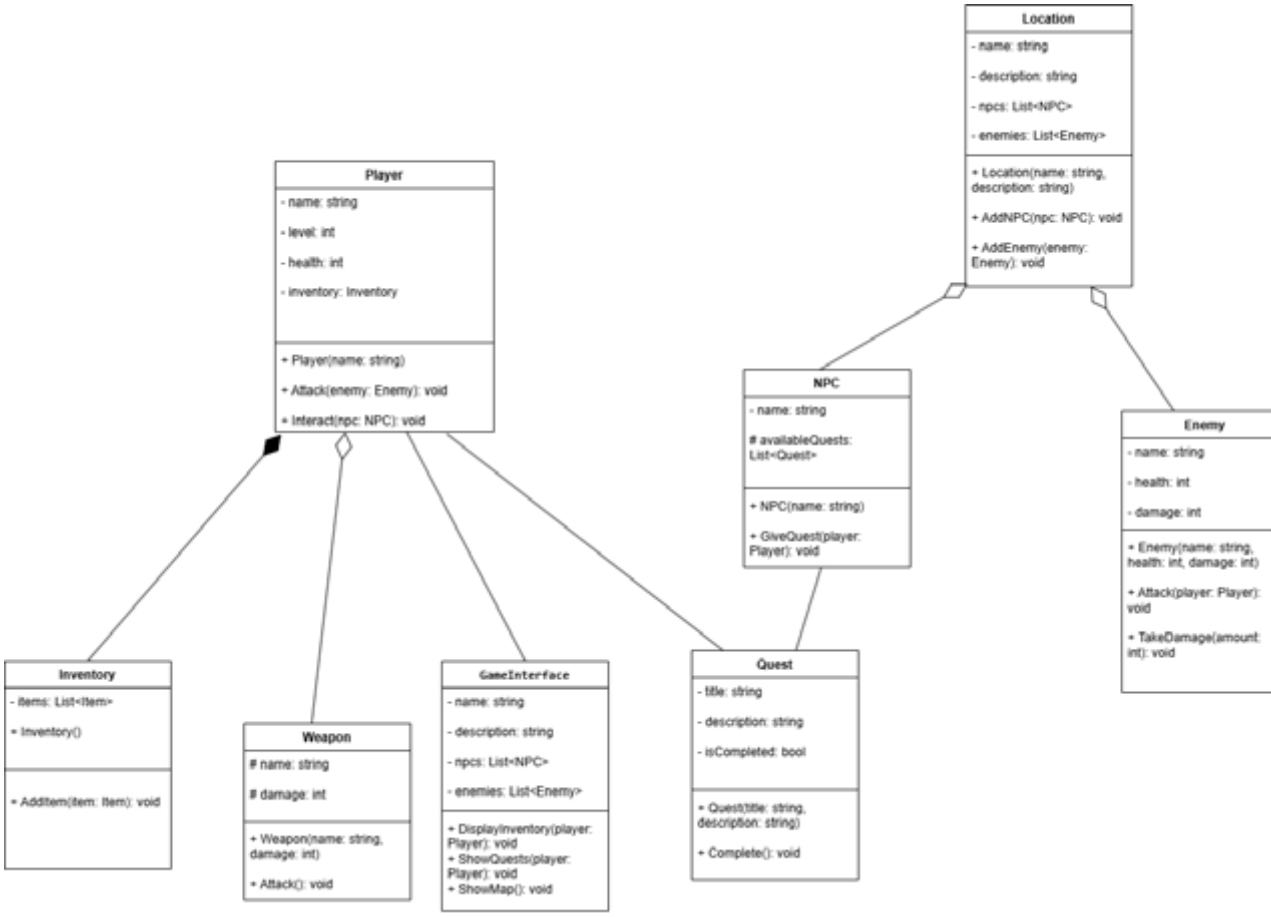


Рис. 2.2. Діаграма класів та кооперацій VR RPG-гри

Діаграма класів є фундаментальною для проєктування програмного забезпечення VR RPG-гри, оскільки відображає структуру системи через класи, їх атрибути, методи та взаємозв’язки.

Основні класи:

1. Гравець (Player)
   - Атрибути: ім’я, рівень, здоров’я, інвентар
   - Методи: атака(), захист(), переміщення(), взаємодіяЗПредметами(), взаємодіяЗNPC(), отриманняКвесту(), завершенняКвесту()
2. NPC
   - Атрибути: ім’я, роль, доступніКвести
   - Методи: видачаКвесту(), розмова(), передачаПредмету()
3. Квест (Quest)

- Атрибути: назва, опис, статус, нагорода
- Методи: оновленняСтатусу()

4. Інвентар (Inventory)
   - Атрибути: списокПредметів
   - Методи: додатиПредмет(), видалитиПредмет(), використатиПредмет()
5. Зброя (Weapon)
   - Атрибути: назва, шкода, тип
   - Методи: використати()
6. Ворог (Enemy)
   - Атрибути: ім’я, здоров’я, рівень
   - Методи: атака(), захист()
7. Локація (Location)
   - Атрибути: назва, опис, списокNPC
   - Методи: додатиNPC(), видалитиNPC()
8. ІнтерфейсГри (GameInterface)
   - Атрибути: карта, списокКвестів, інтерфейсІнвентаря
   - Методи: відобразитиКарту(), відобразитиСписокКвестів(), відобразитиІнвентар()

Взаємозв’язки між класами:

1. Композиція (Player – Inventory)
   - Інвентар є невід’ємною частиною гравця і не може існувати окремо від нього
2. Агрегація (Player o–Weapon)
   - Гравець може мати зброю, але зброя може існувати незалежно від гравця
3. Асоціація (Player – Quest)
   - Гравець взаємодіє з квестами, приймаючи та виконуючи їх
4. Асоціація (NPC – Quest)
   - NPC може видавати квести гравцеві

5. Узагальнення (Enemy extends NPC)

o Ворог є специфічним типом NPC з додатковими властивостями

Діаграма класів наочно демонструє структуру системи та взаємодію її компонентів, що є важливим для подальшої імплементації.

2.3 Діаграма компонентів

Діаграма компонентів відображає організацію та залежності між програмними компонентами системи (рис. 1.8).

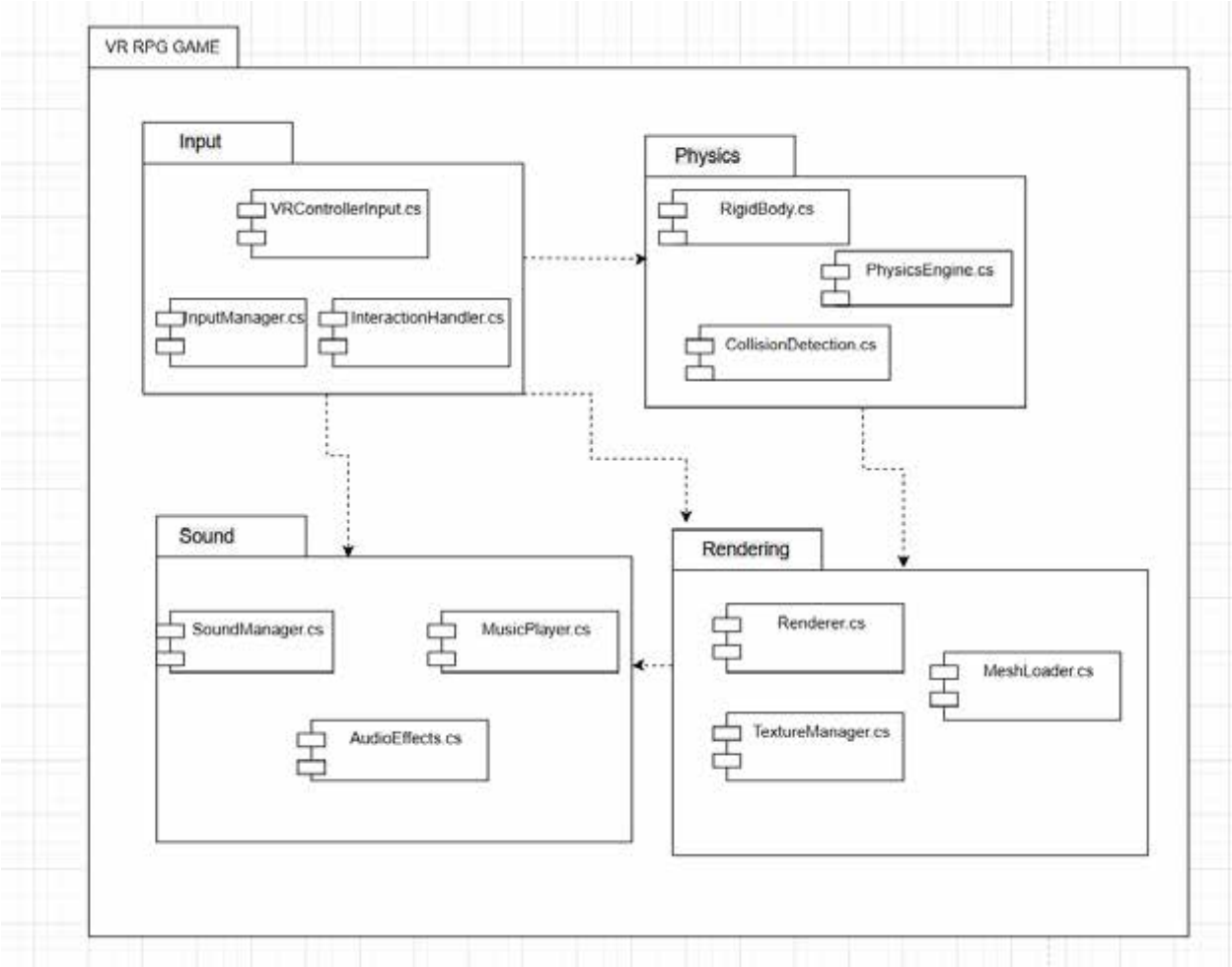


Рис. 2.3. Діаграма компонентів VR RPG-гри

Основні компоненти системи:

1. InputComponent – обробка вводу користувача:

- VRControllerInput.cs
- InputManager.cs

- InteractionHandler.cs

2. PhysicsComponent – фізичний рушій:

- RigidBody.cs
- PhysicsEngine.cs
- CollisionDetection.cs

3. SoundComponent – звукова підсистема:

- SoundManager.cs
- MusicPlayer.cs
- AudioEffects.cs

4. RenderingComponent – візуалізація:

- Renderer.cs
- TextureManager.cs
- MeshLoader.cs

Компоненти взаємодіють через визначені інтерфейси, що забезпечує слабку зв'язаність та можливість незалежної заміни реалізацій. Ключові взаємозв'язки:

- InputComponent → PhysicsComponent: передача подій користувача для фізичної обробки
- RenderingComponent → SoundComponent: синхронізація звукових ефектів з візуальними подіями

Діаграма компонентів демонструє модульність системи та забезпечує основу для організації вихідного коду проєкту.

2.4 Діаграма розгортання

Діаграма розгортання (рис. 1.9) відображає фізичну архітектуру системи та розподіл програмних артефактів на апаратних вузлах.

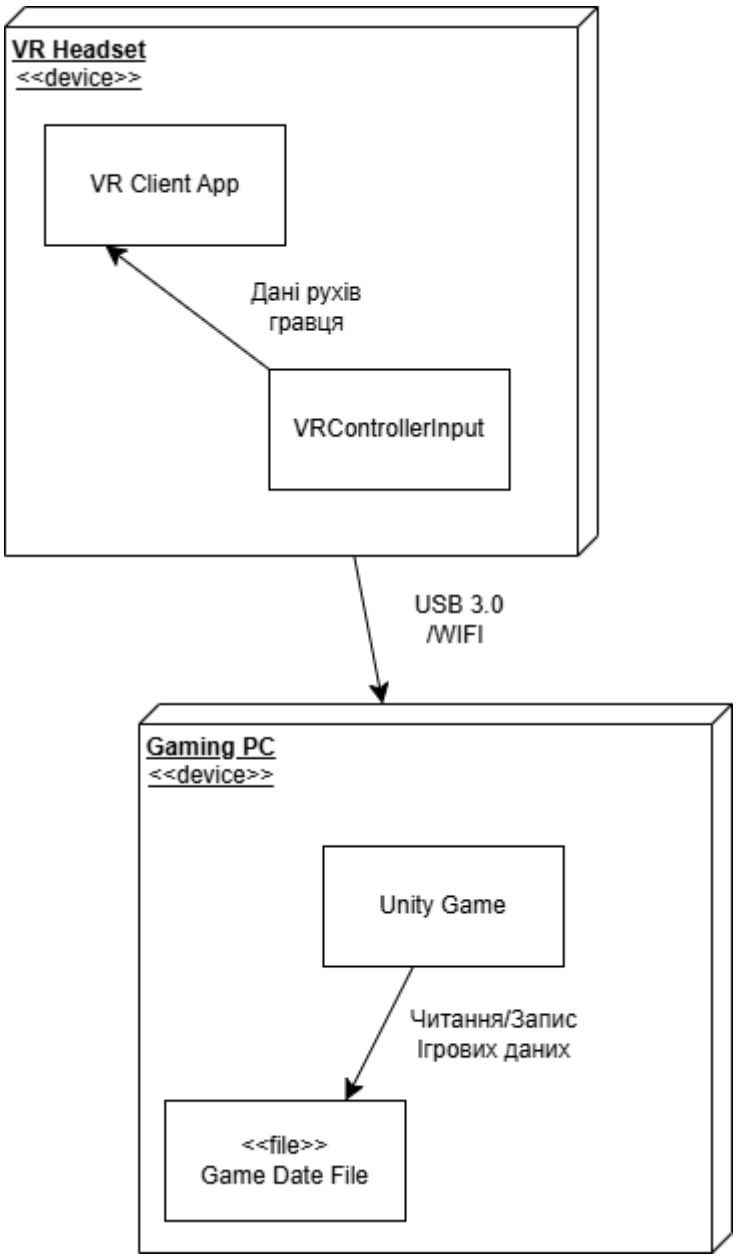


Рис. 2.4. Діаграма розгортання VR RPG-гри

Основні вузли системи:

1. VR Headset (VR-шолом та контролер):

- Артефакт: VR Client App, VRControllerInput

- Забезпечує візуальне відтворення ігрового середовища та відповідає за збір даних від користувача (положення рук, орієнтація, натискання кнопок)

2. Gaming PC (Ігровий комп'ютер):

- Артефакти: Unity Game, Game Data File
- Виконує основні обчислення, рендеринг, фізику та логіку гри

Діаграма також відображає фізичні зв'язки між вузлами:

- З'єднання VR-шолома з ігровим комп'ютером через USB 3.0 або бездротовий зв'язок

Діаграма розгортання демонструє фізичну архітектуру системи та взаємодію між різними апаратними компонентами, що є важливим для розуміння повної картини функціонування VR RPG-гри.

Компоненти, визначені в діаграмі компонентів, розгортаються на відповідних фізичних пристроях, забезпечуючи цілісність та ефективність роботи системи в цілому.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛЬНИХ МЕХАНІК VR- ГРИ У ЖАНРІ RPG

3.1 Створення базового ландшафту у середовищі Unity

Формування основного ігрового середовища є критичним етапом розробки VR-проєкту. У даній роботі для побудови ландшафту було обрано компонент Terrain середовища Unity, який надає потужні та інтуїтивні інструменти для моделювання віртуального світу.

За допомогою набору інструментів Terrain Tools було створено рельєф із різноманітними географічними елементами: горами, долинами та водоймами, що формують основу ігрового світу (рис. 3.1). Поверхня рельєфу була вручну оброблена шляхом "фарбування" відповідними текстурами – травою, камінням, піском та іншими природними матеріалами, що забезпечує реалістичний вигляд ландшафту.

Під час створення ландшафту було використано знання, отримані з навчального відео *Build a beautiful 3D open world in 5 minutes | Unity* [1], що дало змогу швидко освоїти базові техніки генерації відкритого світу, зокрема роботу з шумами висоти, плавні переходи між типами поверхні та базову оптимізацію сцени. Це дозволило ефективно створити візуально привабливе й придатне для VR-досвіду середовище.

Рис. 3.1. Базовий ландшафт VR-проєкту

Весь ландшафт був сформований вручну з використанням вбудованих можливостей редагування Terrain в Unity, з подальшим детальним нанесенням текстур для досягнення реалістичного зовнішнього вигляду (рис. 3.2).

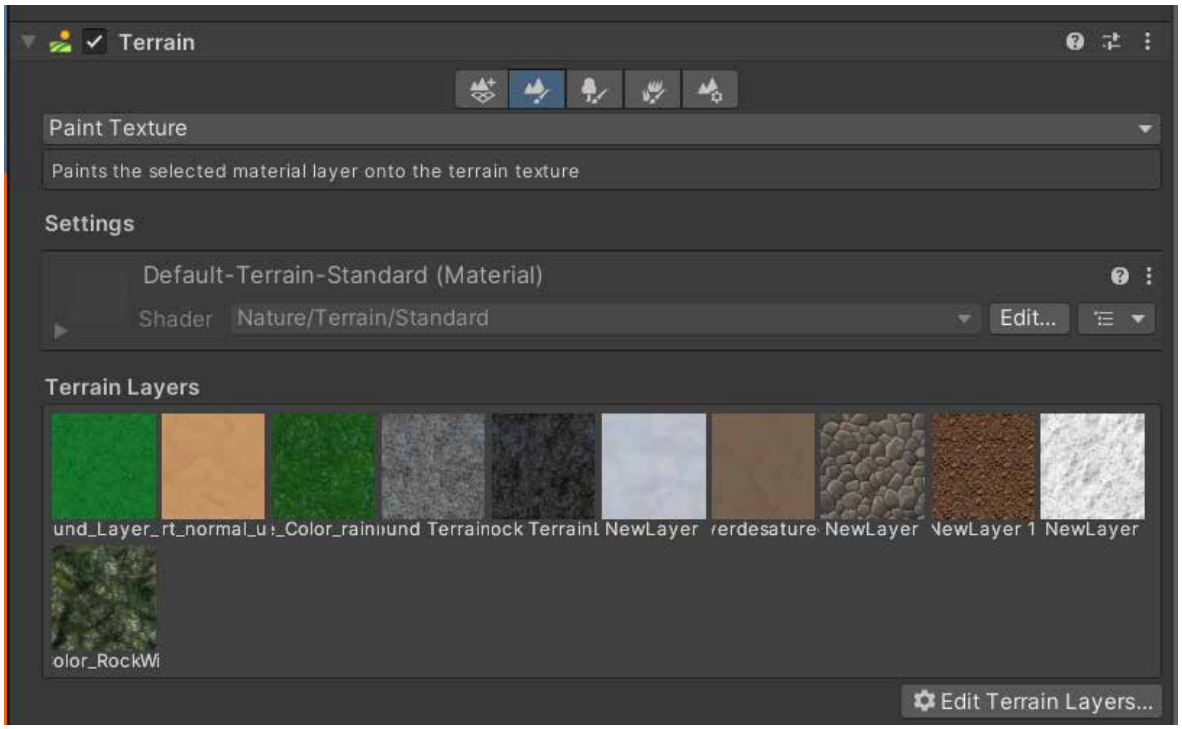


Рис. 3.2. Текстурований ландшафт

Для збагачення середовища на створений рельєф були розміщені елементи природного оточення: дерева, кущі, трава та інші рослинні об'єкти. Це було

реалізовано з використанням компонентів Terrain (Tree, Detail) та імпортованих 3D-моделей (рис. 3.3, 3.4).

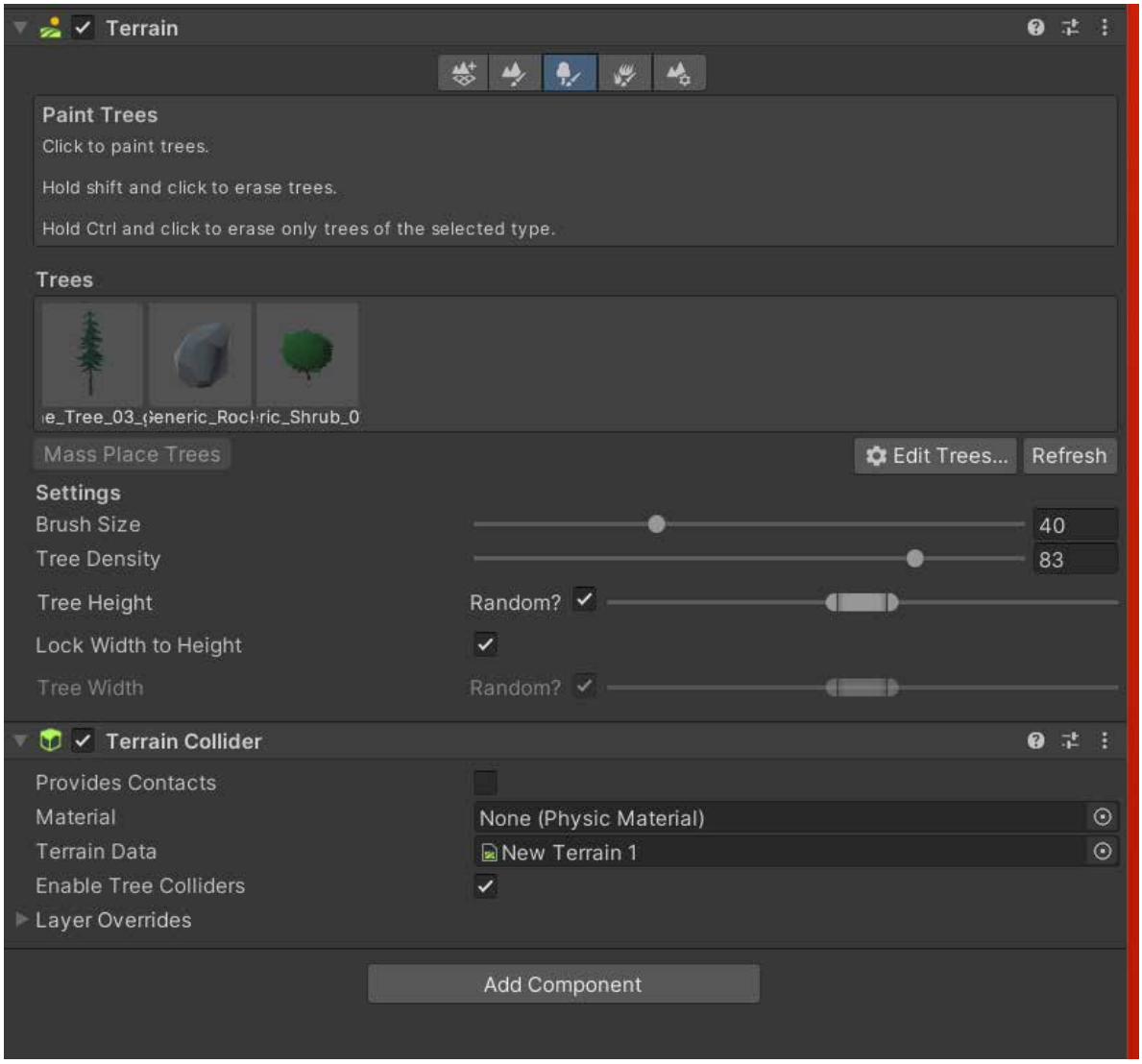


Рис. 3.3. Рослинність на ландшафті: дерева та трава

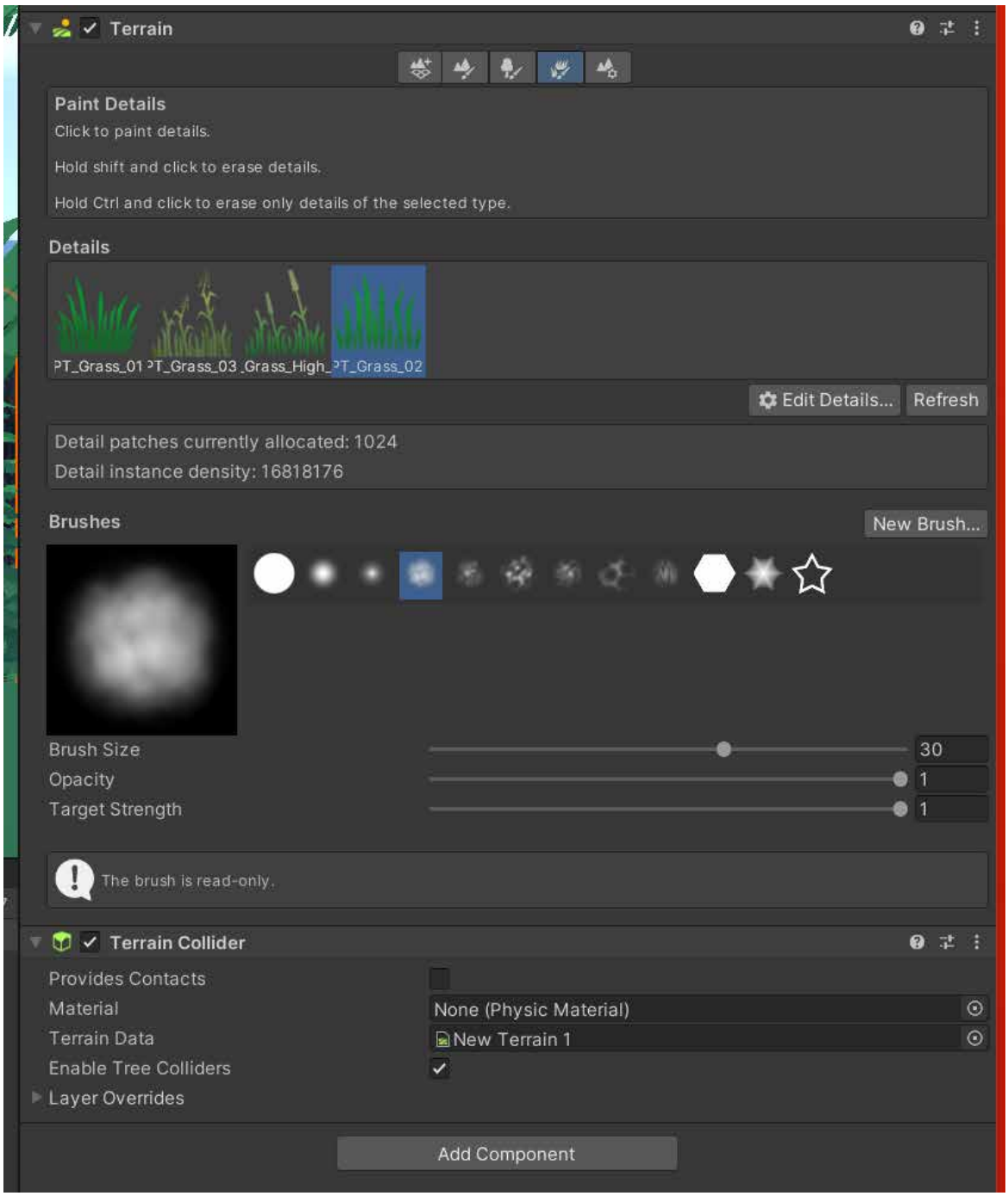


Рис. 3.4. Детальний вигляд рослинності

Рис. 3.5. Загальний вигляд ландшафту з рослинністю

Водні об'єкти реалізовано як плоскі поверхні (Plane) з налаштованим шейдером води, що створює реалістичний ефект водної поверхні (рис. 3.6). Такий підхід є оптимальним з точки зору продуктивності системи.

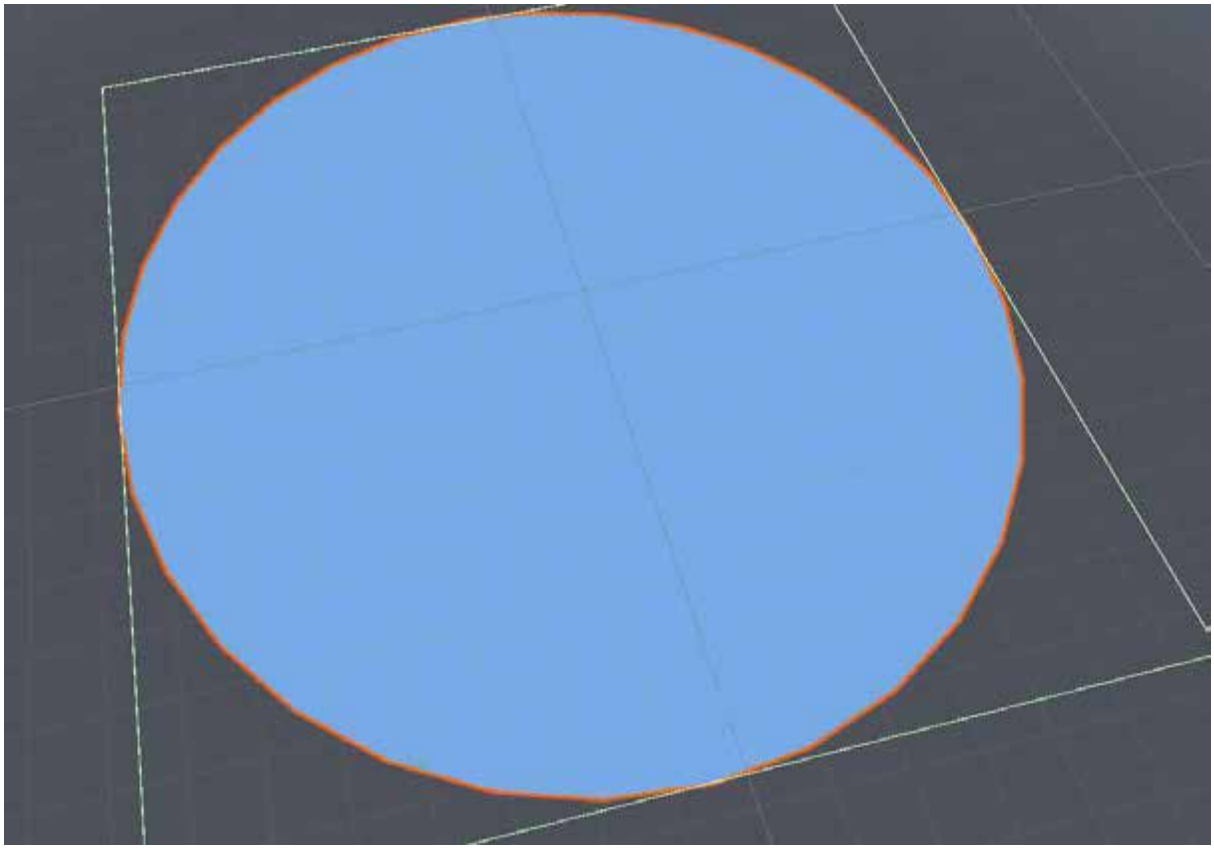


Рис. 3.6. Водна поверхня з налаштованим шейдером

Для забезпечення високої продуктивності VR-сцени було впроваджено ряд технічних оптимізацій:

- Налаштовано рівні деталізації (LOD) для всіх складних 3D-моделей
- Відключено колайдери для віддалених об'єктів

- Використано прості плоскі поверхні з шейдерами замість складних рідинних симуляцій для водойм

Такі методи оптимізації критично важливі для VR-середовища, оскільки вони забезпечують стабільний фреймрейт (частоту кадрів), необхідний для комфортного перебування користувача у віртуальній реальності та запобігання явищу VR-хвороби, що може виникати при низькій швидкості відображення.

3.2 Налаштування VR-проєкту та інтеграція OpenXR

Після побудови базового ландшафту наступним важливим етапом стала конфігурація проєкту для віртуальної реальності. Для забезпечення сумісності з різними VR-пристроями було обрано стандарт OpenXR, який забезпечує універсальний інтерфейс взаємодії з широким спектром VR-гарнітур.

У XR Plugin Management Unity було активовано плагін OpenXR, що гарантує крос-платформену сумісність з різними VR-системами, включаючи Oculus Quest, HTC Vive та інші. У налаштуваннях OpenXR було обрано відповідні профілі контролерів та підключено необхідні розширення, зокрема підтримку Oculus Touch (рис. 3.7).

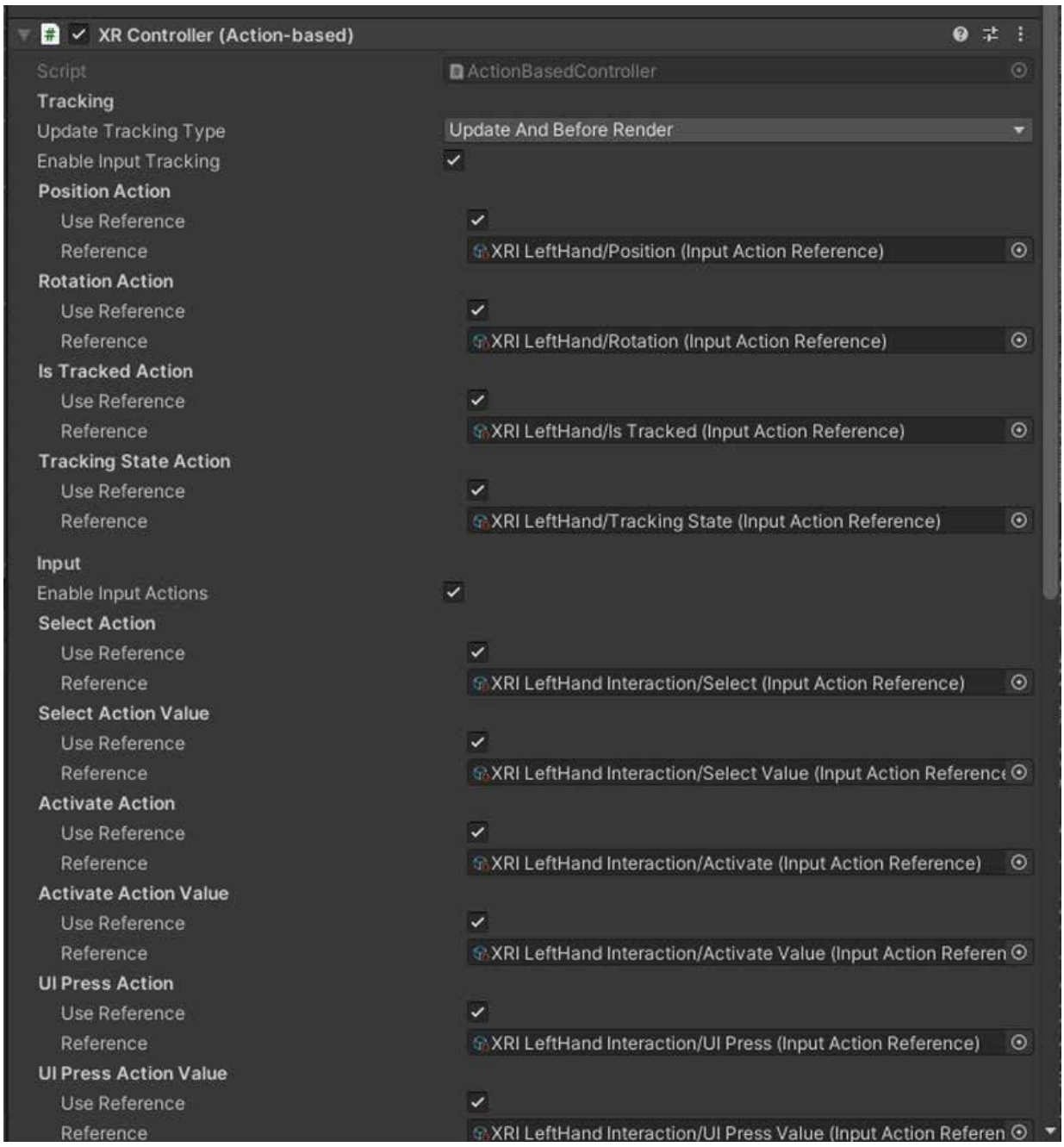


Рис. 3.7. Налаштування OpenXR у Unity

Для розширення функціональності VR-взаємодії було імпортовано XR Interaction Toolkit – набір компонентів Unity, що надає стандартизовані інструменти для VR-взаємодій. Основні використані компоненти включають:

- XR Origin (Rig) – базовий компонент, що містить камеру та моделі "рук-контролерів" гравця
- XR Controller – компонент управління VR-контролерами
- XR Interactor – система взаємодії з об'єктами

- XR Grab Interactable – компонент, що дозволяє об'єктам бути захопленими у VR

У сцені було налаштовано префаб XR Origin (Rig), що забезпечує правильне позиціонування камери відповідно до рухів користувача у фізичному просторі. Для забезпечення комфортного VR-досвіду особливу увагу було приділено параметрам графіки та продуктивності:

- Встановлено цільову роздільну здатність для VR-рендерингу
- Налаштовано цільову частоту кадрів (72-90 FPS) для стабільної роботи
- Оптимізовано рендеринг сцени для зменшення навантаження на систему

Для навігації у віртуальному світі були імплементовані стандартні VR-механіки переміщення, що забезпечують зручне пересування гравця ландшафтом, враховуючи обмеження фізичного простору користувача.

Застосування OpenXR у поєднанні з XR Interaction Toolkit забезпечило єдиний інтерфейс для різних VR-пристроїв, що значно прискорило розробку стандартних механік взаємодії з об'єктами та середовищем. Використання цих перевірених компонентів гарантує оптимізовану продуктивність та стабільну роботу VR-сцени, що є необхідною умовою комфортного занурення гравця у віртуальний світ.

3.3 Моделювання та анімація рук у VR

Для забезпечення реалістичної взаємодії користувача з віртуальним середовищем розроблено систему моделювання та анімації рук у VR. На відміну від абстрактних контролерів, використання моделей рук значно підвищує відчуття присутності та інтуїтивність взаємодії з віртуальними об'єктами.

Для відображення рук персонажа використано 3D-моделі лівої та правої руки з налаштованими скелетними анімаціями.

До цих моделей додано компонент Animator, у якому реалізовано два основні параметри управління (рис. 3.8):

- *Grip* – відповідає за стискання кулака
- *Trigger* – контролює згинання вказівного пальця

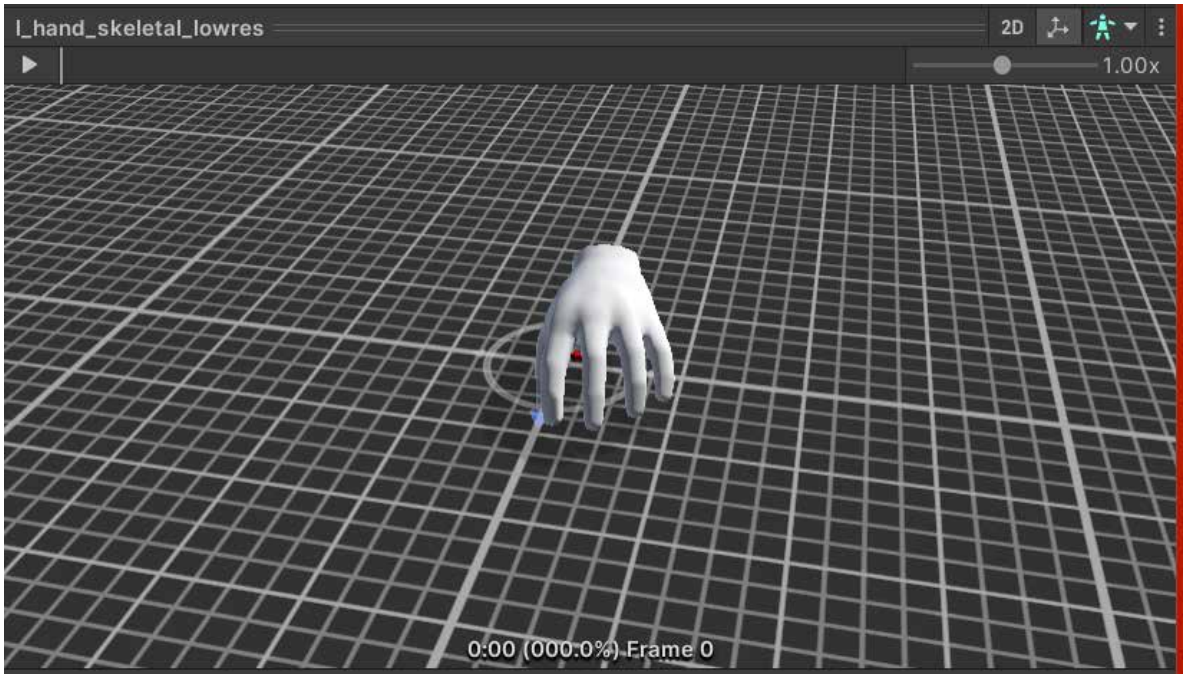


Рис. 3.8. Налаштування анімації рук

Анімаційні кліпи (Idle, Grab та інші) задано за допомогою стандартної системи Unity Animation (рис. 3.9, 3.10).

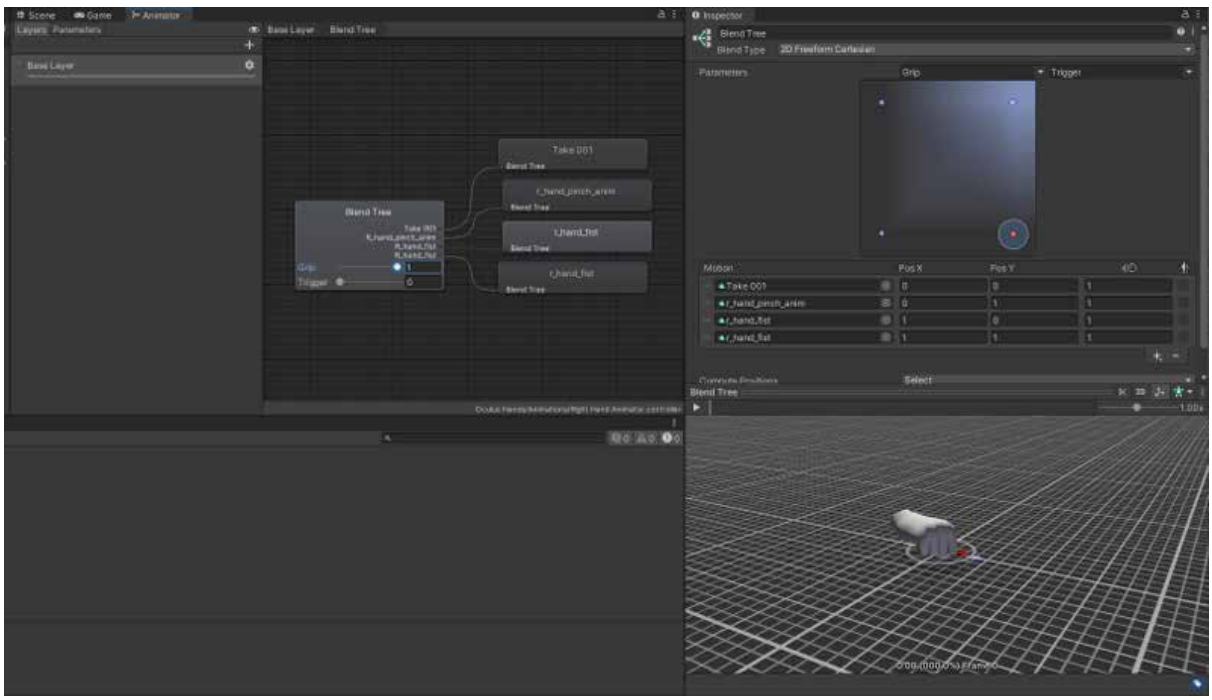


Рис. 3.9. Анімаційні стани рук

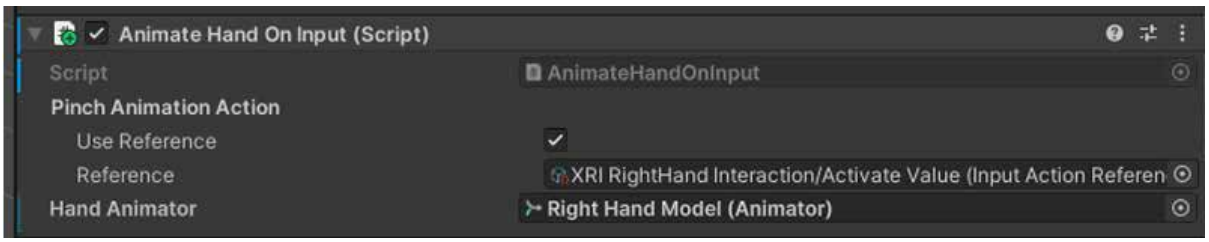


Рис. 3.10. Параметри анімації рук

Для зв'язку фізичних рухів контролерів з анімацією віртуальних рук реалізовано спеціалізовані C#-скрипти, що зчитують сигнали з тригерів та кнопок VR-контролерів і передають їх у систему анімації рис. 3.11.

```
public class AnimateHandOnInput : MonoBehaviour
{
    public InputActionProperty pinchAnimationAction;
    public InputActionProperty gripAnimationAction;

    public Animator handAnimator;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        float triggerValue = pinchAnimationAction.action.ReadValue<float>();
        handAnimator.SetFloat("Trigger", triggerValue);

        float gripValue = gripAnimationAction.action.ReadValue<float>();
        handAnimator.SetFloat("Grip", gripValue);


    }
}
```

Рис. 3.11. Код обробки даних з контролерів для правої руки

У скрипті *HandAnimatorController* (див. Додаток А) реалізовано механізм зчитування значень аналогових тригерів контролерів та передачі їх до системи анімації. Наприклад, значення натискання тригера передається через метод animator.SetFloat("Grip", value), де value – ступінь натискання. Таким чином, згортання пальців у моделі руки точно відповідає фізичним діям користувача.

Моделі рук також оснащено компонентами XRDirectInteractor і XRGrabInteractable, які забезпечують автоматичну анімацію захоплення при взаємодії з предметами. Це дозволяє моделям рук природно адаптуватись до форми захоплюваних об'єктів.

Розроблена система забезпечує плавну та узгоджену анімацію рук у VR відповідно до рухів користувача, що суттєво підвищує відчуття присутності у віртуальному світі. Візуалізація реалістичних рук замість абстрактних контролерів значно полегшує користувачу орієнтацію у просторі та забезпечує інтуїтивне маніпулювання віртуальними об'єктами.

3.4 Розробка механіки холодної зброї

Важливим елементом ігрової механіки розробленої VR-гри є система холодної зброї, яка реалізує реалістичну фізичну взаємодію гравця з об'єктами середовища через зброю. Розроблено кілька типів холодної зброї з різними характеристиками та способами використання.

3.4.1 Фізичні властивості зброї

Кожен екземпляр холодної зброї (меч, сокира тощо) спроектовано з урахуванням реалістичних фізичних властивостей. Структурно зброя складається з двох основних частин:

1. Руків'я (рукоятка) – частина для утримання
2. Лезо – ріжуча/ударна частина

Руків'я оснащено компонентом XRGrabInteractable, що дозволяє гравцеві захоплювати зброю у VR. На лезо і руків'я додано окремі колайдери (BoxCollider або MeshCollider) для коректної реєстрації зіткнень (рис. 3.12).

Рис. 3.12. Модель одноручного меча з колайдерами

У фізичних налаштуваннях зброї особлива увага приділена центру маси (RigidBody.centerOfMass), що забезпечує природну поведінку зброї при розмаху (рис. 3.13).

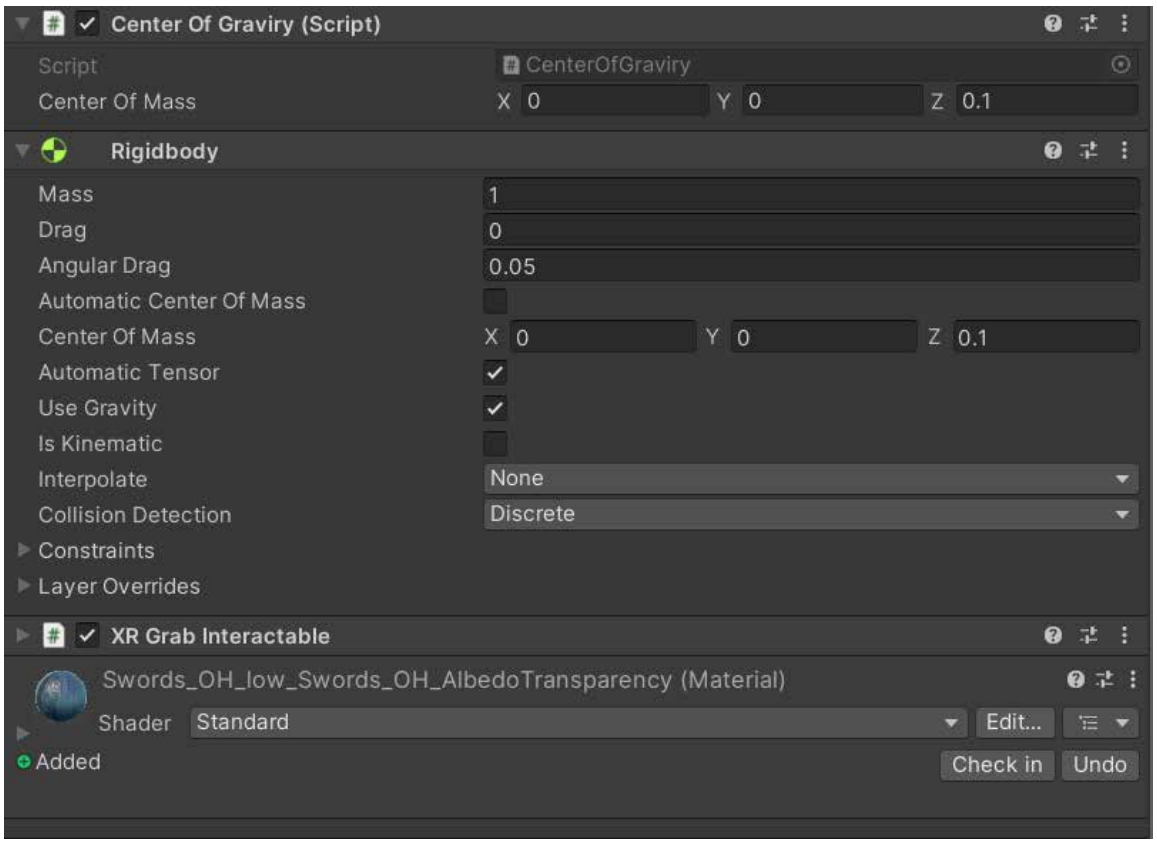


Рис. 3.13. Налаштування фізичних властивостей меча (жовтим позначено центр тяжіння)

3.4.2 Різновиди реалізованої холодної зброї

Структура одноручної зброї

Одноручна зброя (мечі, невеликі сокири) має одну точку захвату, що дозволяє утримувати її однією рукою (рис. 3.14, 3.15).

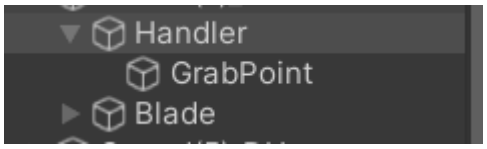


Рис. 3.14. Структура рукоятки одноручного меча

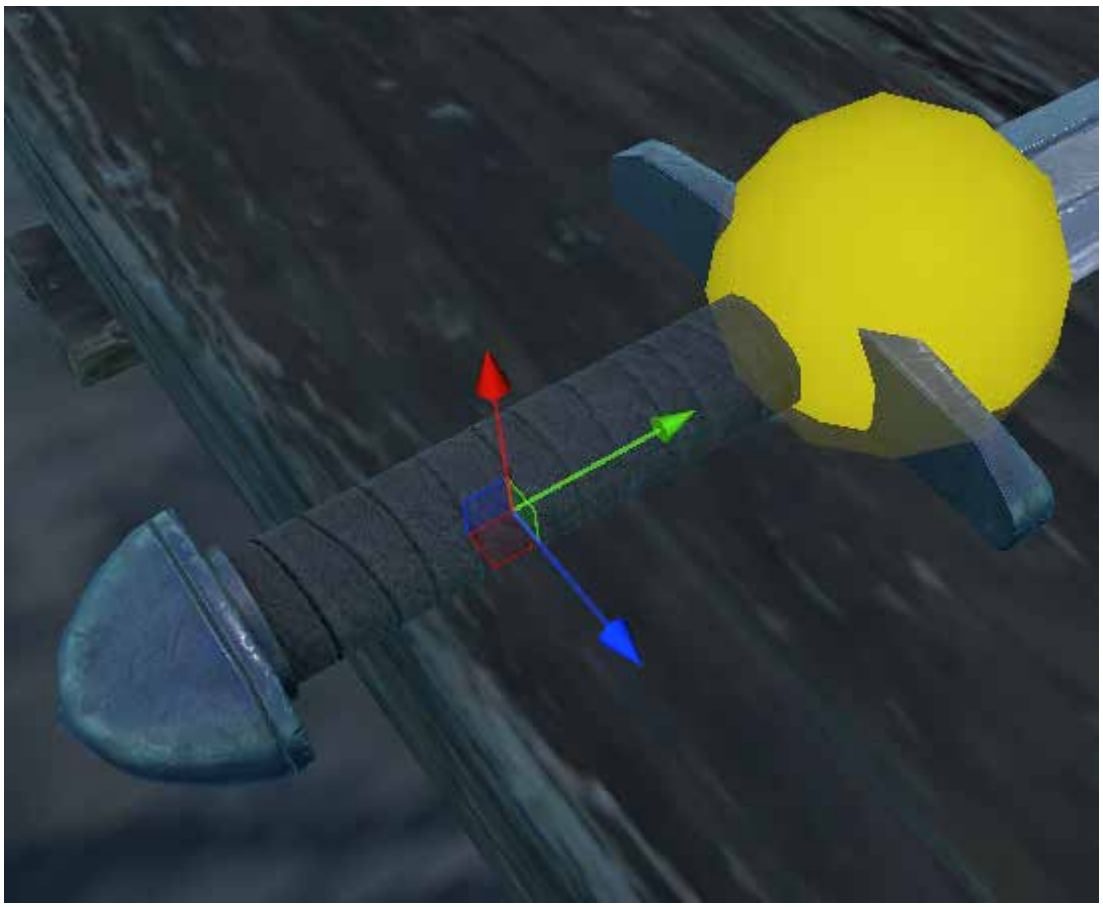


Рис. 3.15. Точка захвату (GrabPoint) одноручного меча з BoxCollider

Важливим елементом конструкції є точка захвату (GrabPoint), яка має власний BoxCollider, що визначає зону взаємодії з рукою гравця (рис. 3.16, 3.17).

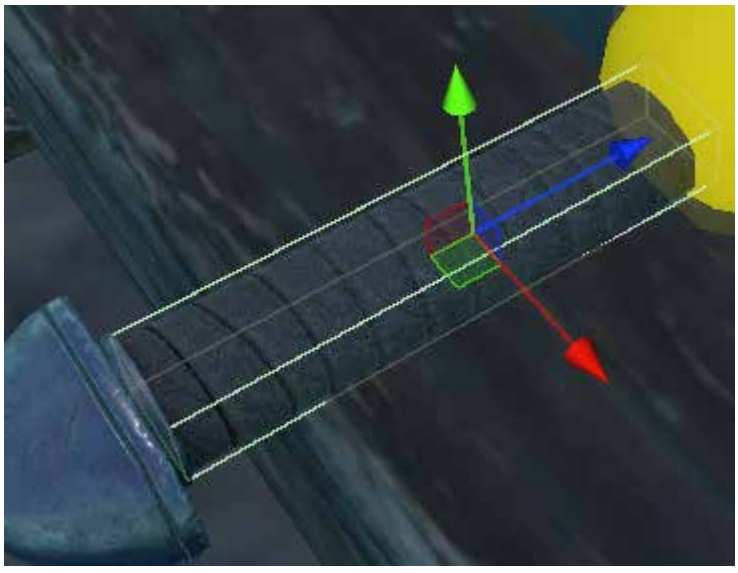


Рис. 3.16. Колайдер точки захвату

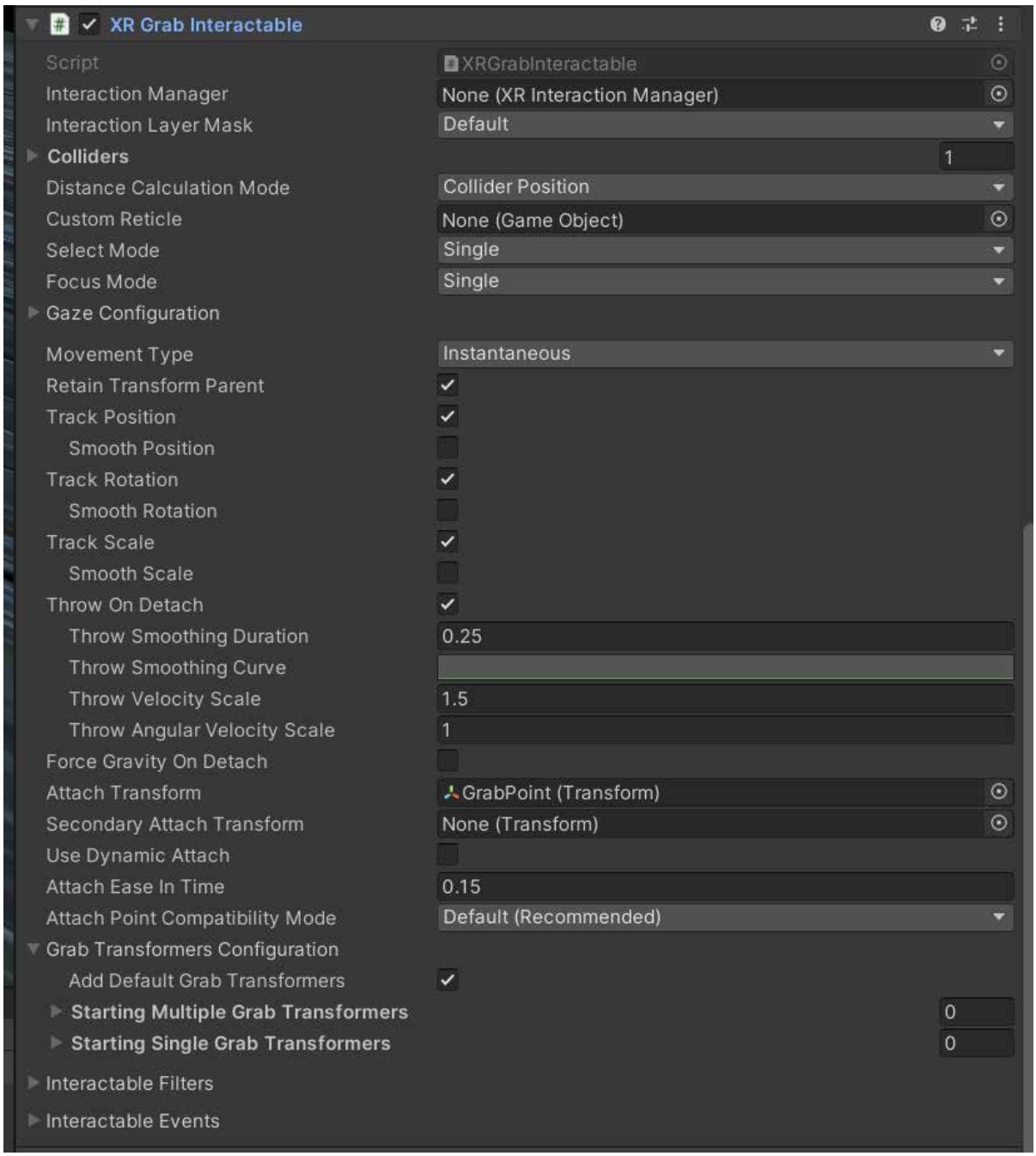


Рис. 3.17. Налаштування точки захвату

Система реєстрації ударів

Для реєстрації ударів та реалізації механіки розрізання об'єктів лезо зброї оснащено спеціальним колайдером та точками відстеження (рис. 3.18, 3.19).

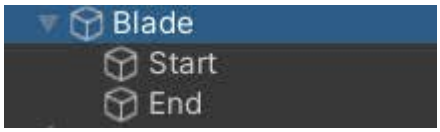


Рис. 3.18. Структура леза меча

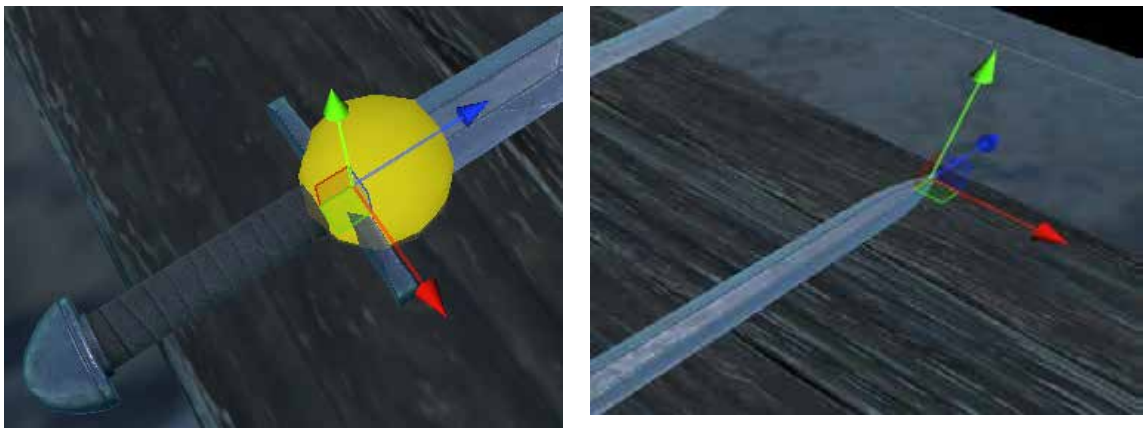


Рис. 3.19. Точки відстеження початку та кінця леза для визначення вектора удару

Дворучна зброя

Для дворучної зброї (двуручні мечі, великі сокири) створено дві точки захвату на руків'ї, що дозволяють утримувати предмет двома руками (рис. 3.20, 3.21).

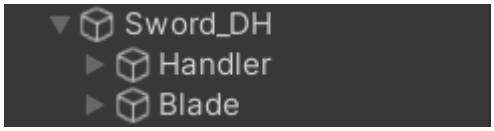


Рис. 3.20. Структура дворучного меча

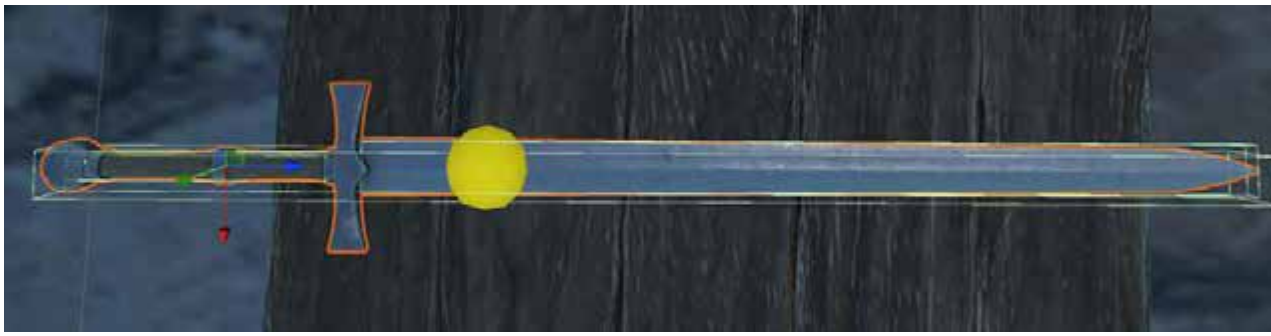
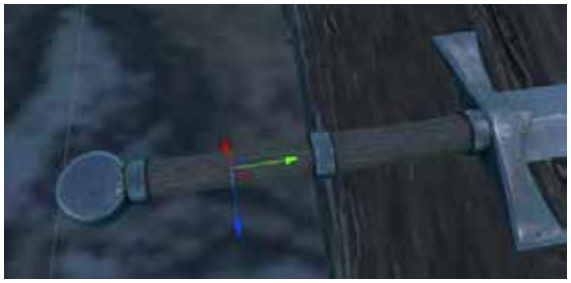


Рис. 3.21. Дворучний меч з системою колайдерів

Рукоятка дворучної зброї містить дві точки захвату, що розташовані на оптимальній відстані для дворучного утримання (рис. 3.22, 3.23).

Рис. 3.22. Точки захвату дворучного меча



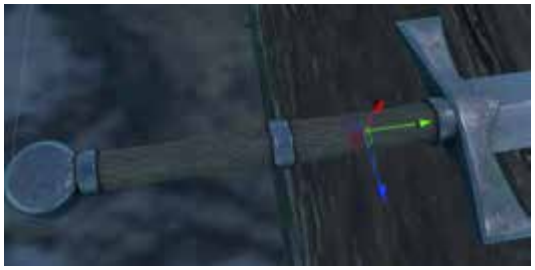


Рис. 3.23. Розташування точок захвату на дворучному мечі

Одноручний топір

Топір має довгу рукоятку та масивне лезо складної форми (рис. 3.24). Його можна використовувати як однією, так і двома руками.

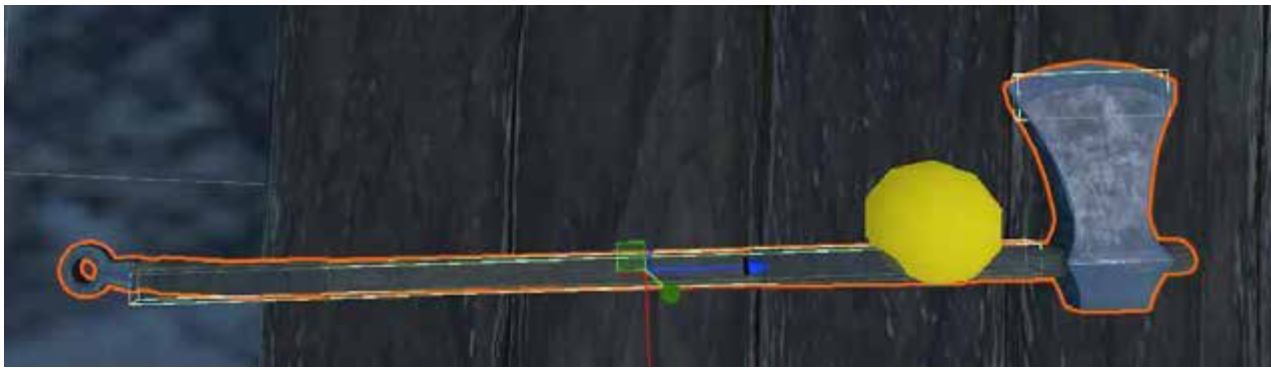


Рис. 3.24. Модель одноручного топора

Структура топора включає рукоятку та лезо з оптимізованим колайдером (рис. 3.25, 3.26).

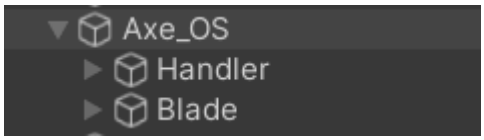


Рис. 3.25. Структура одноручного топора

Рис. 3.26. Оптимізований BoxCollider для складної форми леза топора

Довга рукоятка топора забезпечує можливість зручного тримання як однією, так і двома руками (рис. 3.27).

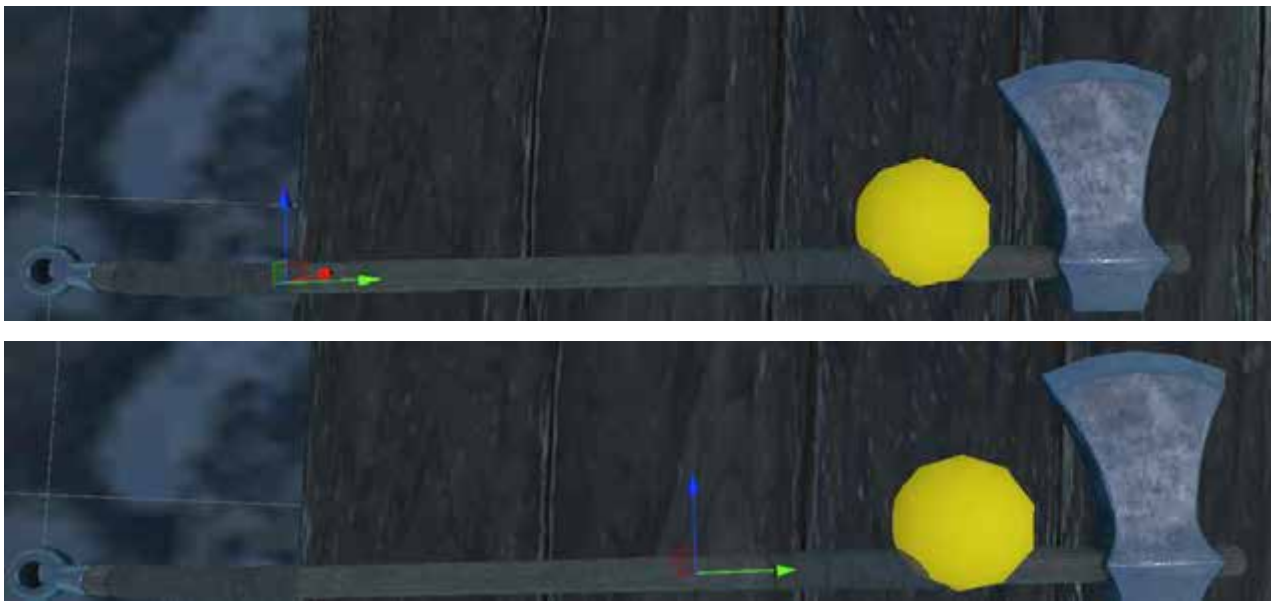


Рис. 3.27. Рукоятка топора для дворучного тримання

Дворучний топір

Дворучний топір має масивну рукоятку та подвійне лезо складної геометрії (рис. 3.28).

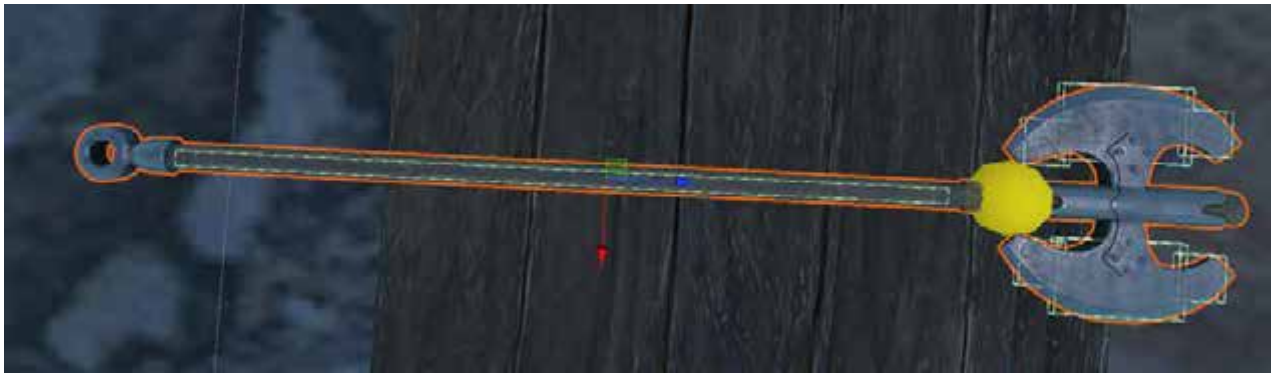


Рис. 3.28. Модель дворучного топора

Рукоятка обладнана двома точками захвату (рис. 3.31).

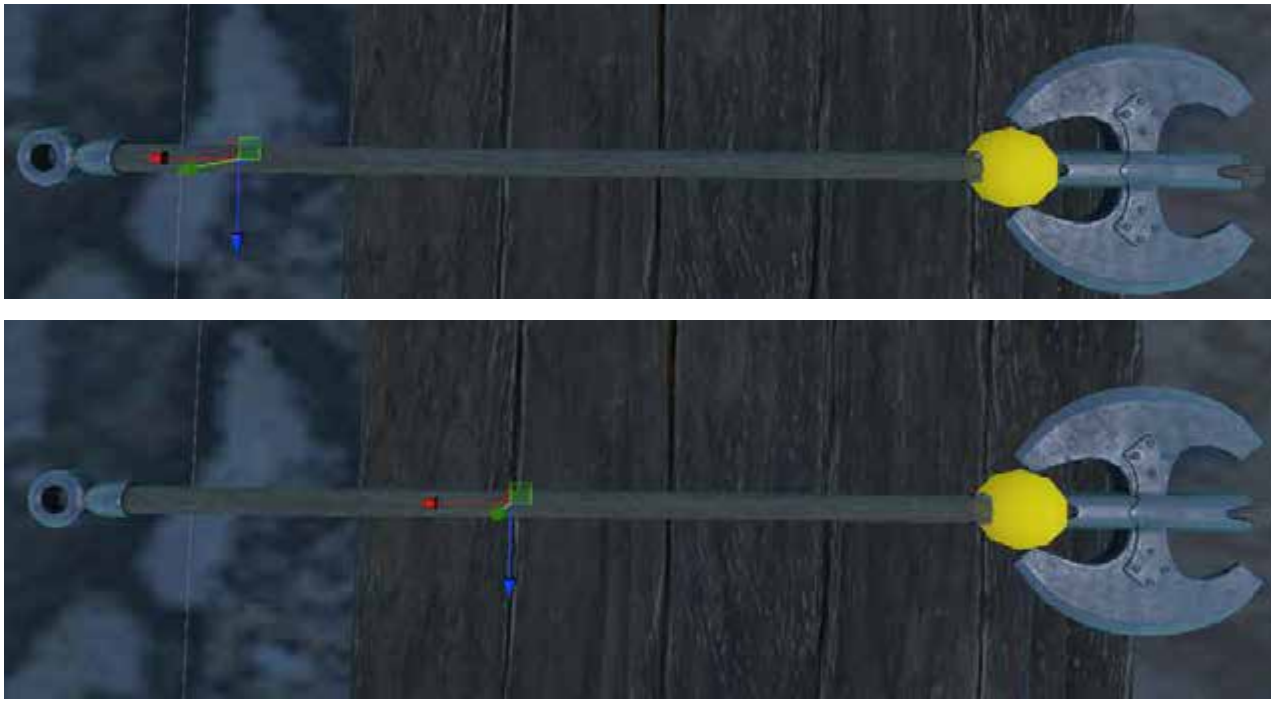


Рис. 3.29. Рукоятка дворучного топора з двома точками захвату

Лезо дворучного топора складається з двох окремих частин з індивідуальними колайдерами (рис. 3.30, 3.31).

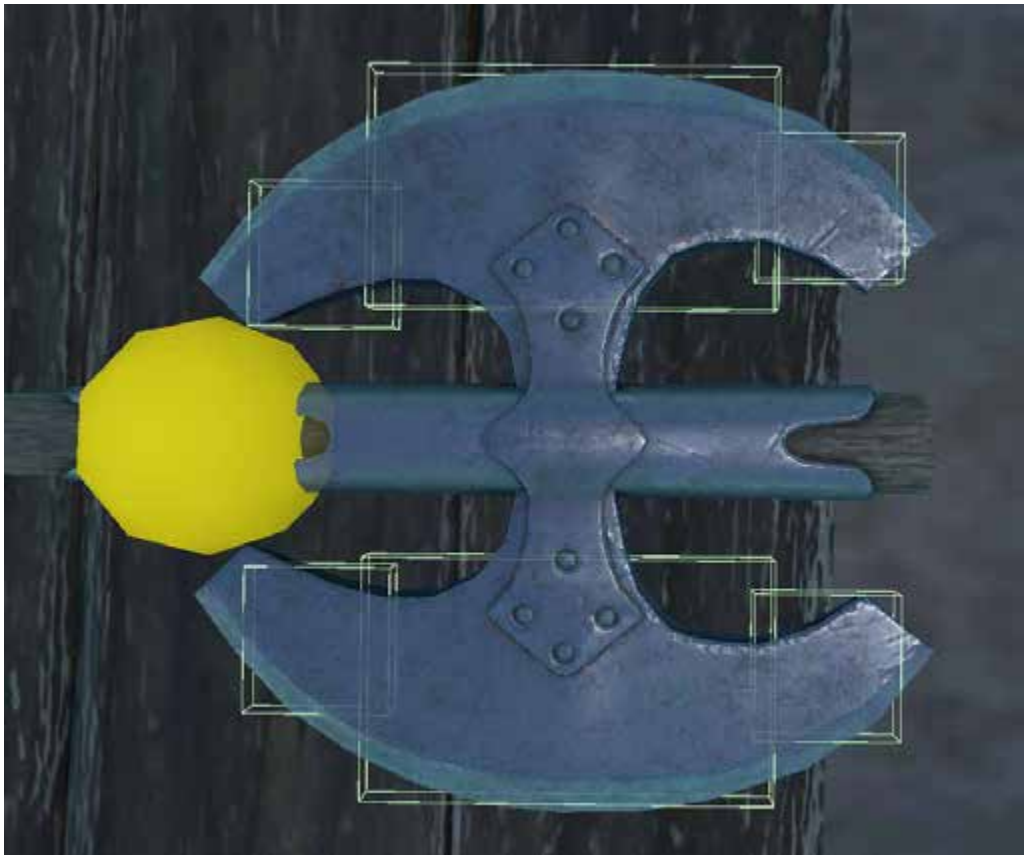


Рис. 3.30. Подвійне лезо дворучного топора

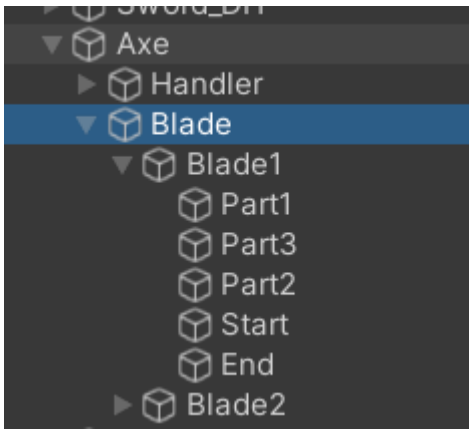


Рис. 3.31. Детальна структура одного з лез топора

Для оптимізації фізичної взаємодії кожна частина леза має окремий BoxCollider, що відповідає його формі (рис. 3.32).

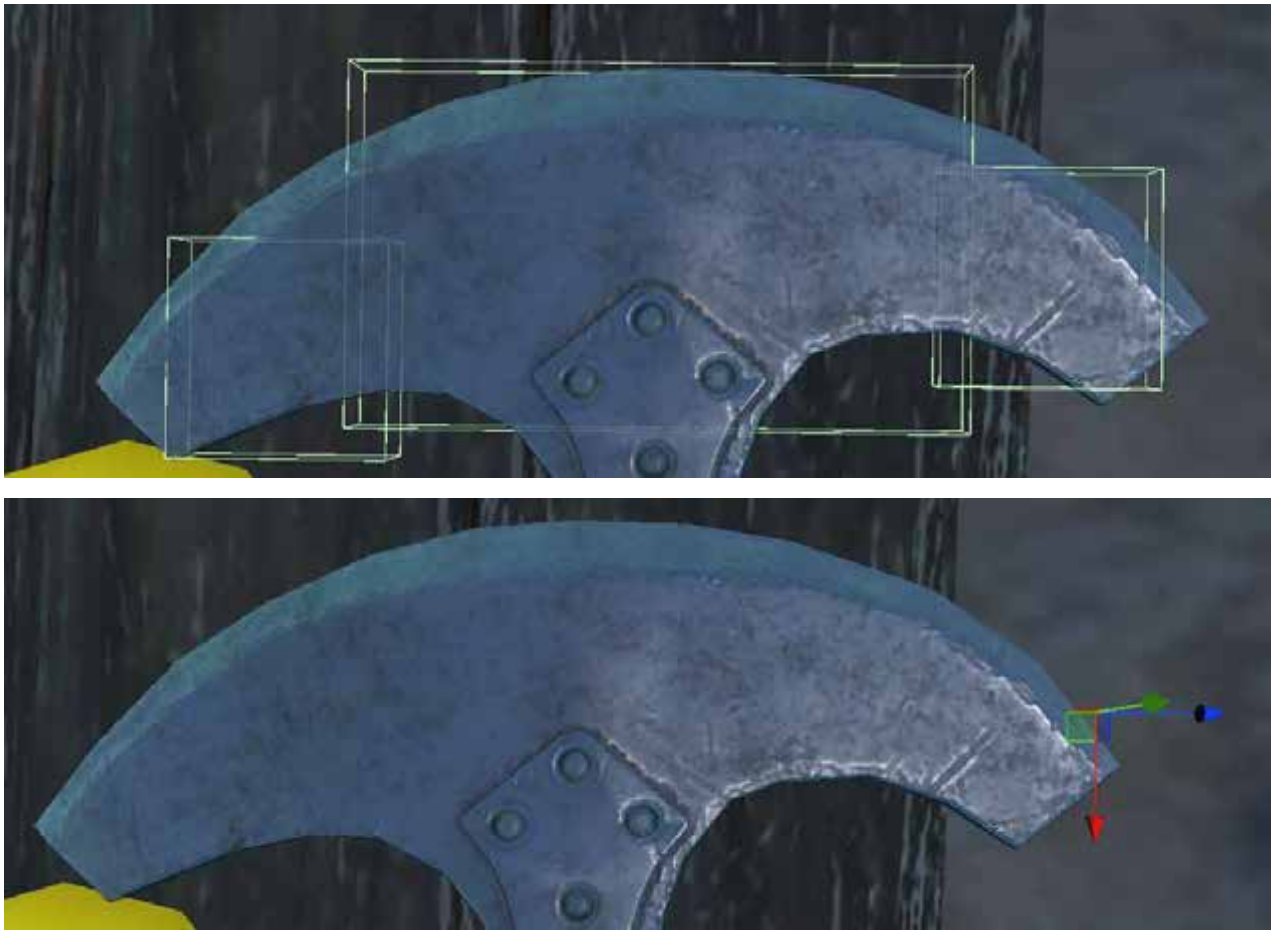


Рис. 3.32. Колайдери частин леза та точки відстеження для вектора удару

Альтернативний дворучний меч

Реалізовано також альтернативний варіант дворучного меча з іншими візуальними та фізичними характеристиками (рис. 3.33).

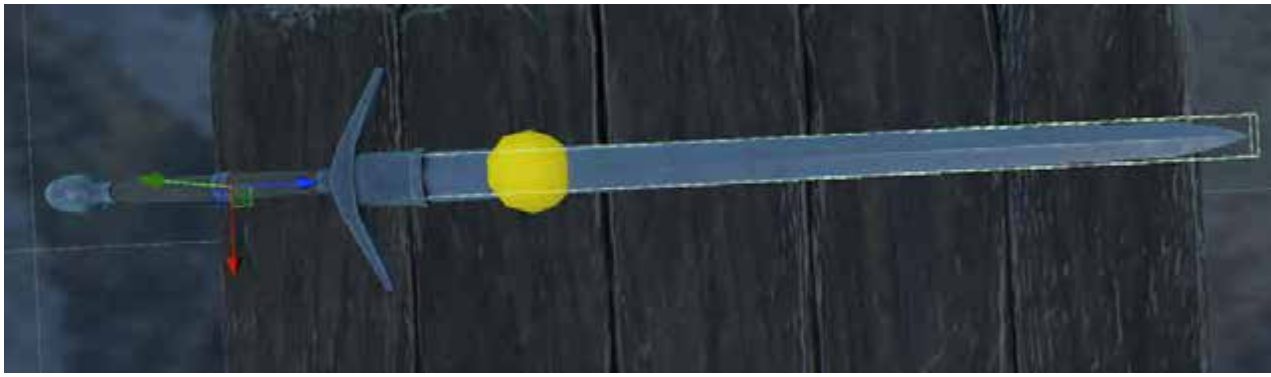


Рис. 3.33. Альтернативний дворучний меч

Для організації механіки розрізання до зброї додано компонент VelocityEstimator (із XR Interaction Toolkit), який обчислює швидкість руху двох точок: початкової (біля руків'я) і кінцевої (на вершині леза). Під час розмаху, якщо ці точки формують прямолінійний шлях через підрізаний об'єкт, запускається процес розрізання (детальний опис у розділі 3.6).

Розроблена система холодної зброї забезпечує реалістичну поведінку зброї при взаємодії з об'єктами середовища, при цьому залишаючись оптимізованою для стабільної роботи у VR з необхідною частотою кадрів.

3.5 Реалізація лука та системи стрільби

Для розширення ігрових механік було розроблено систему дистанційної зброї, зокрема лук зі стрілами. Ця система забезпечує інтуїтивне та реалістичне використання лука у віртуальній реальності з урахуванням фізичних принципів.

3.5.1 Конструкція лука

Модель лука складається з двох основних частин (рис. 3.34):

1. Основна частина лука з руків'ям
2. Тетива, реалізована як незалежний об'єкт

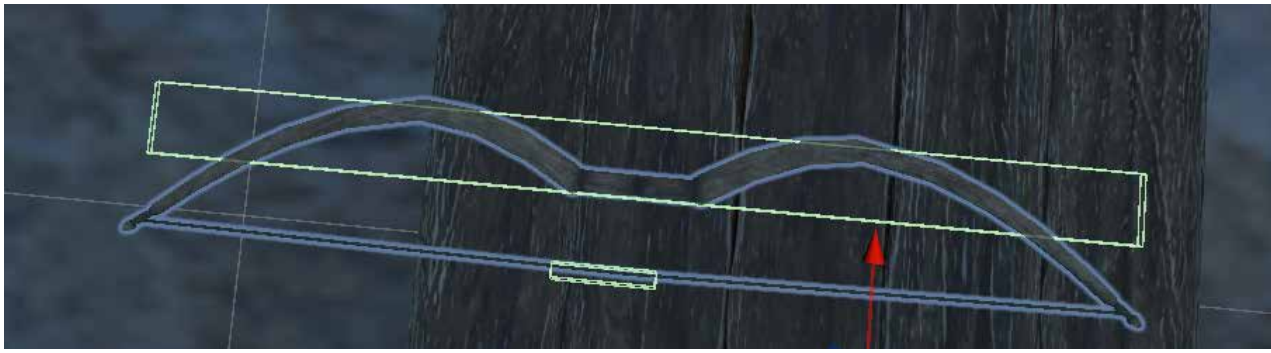


Рис. 3.34. Модель лука з тетивою

Основна частина лука має BoxCollider, компонент фізики та точку захвату GrabPoint (рис. 3.35, 3.36).

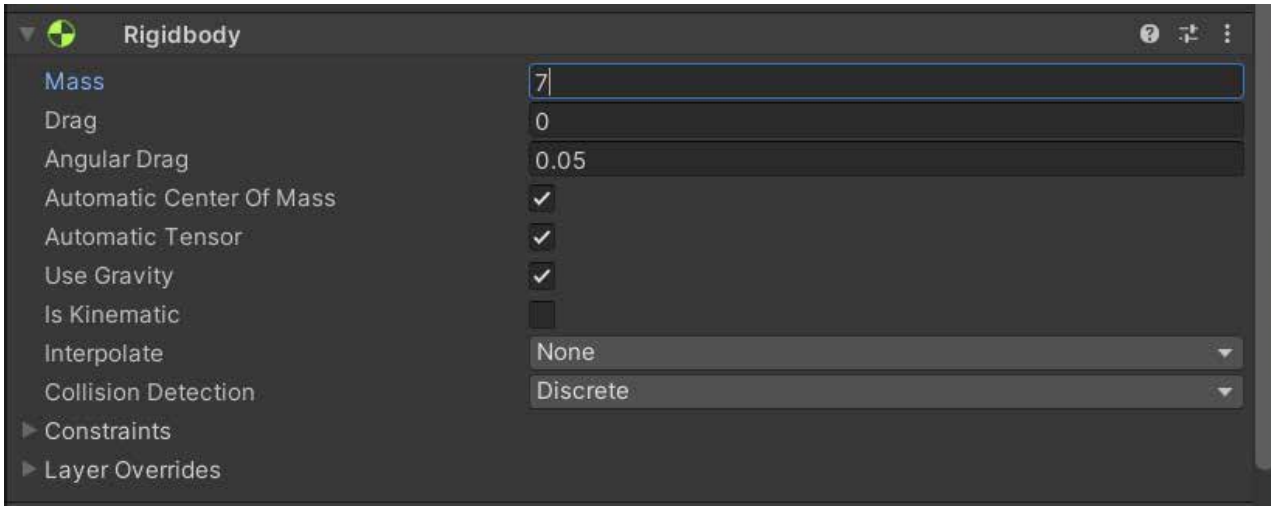


Рис. 3.35. Фізичні компоненти основної частини лука

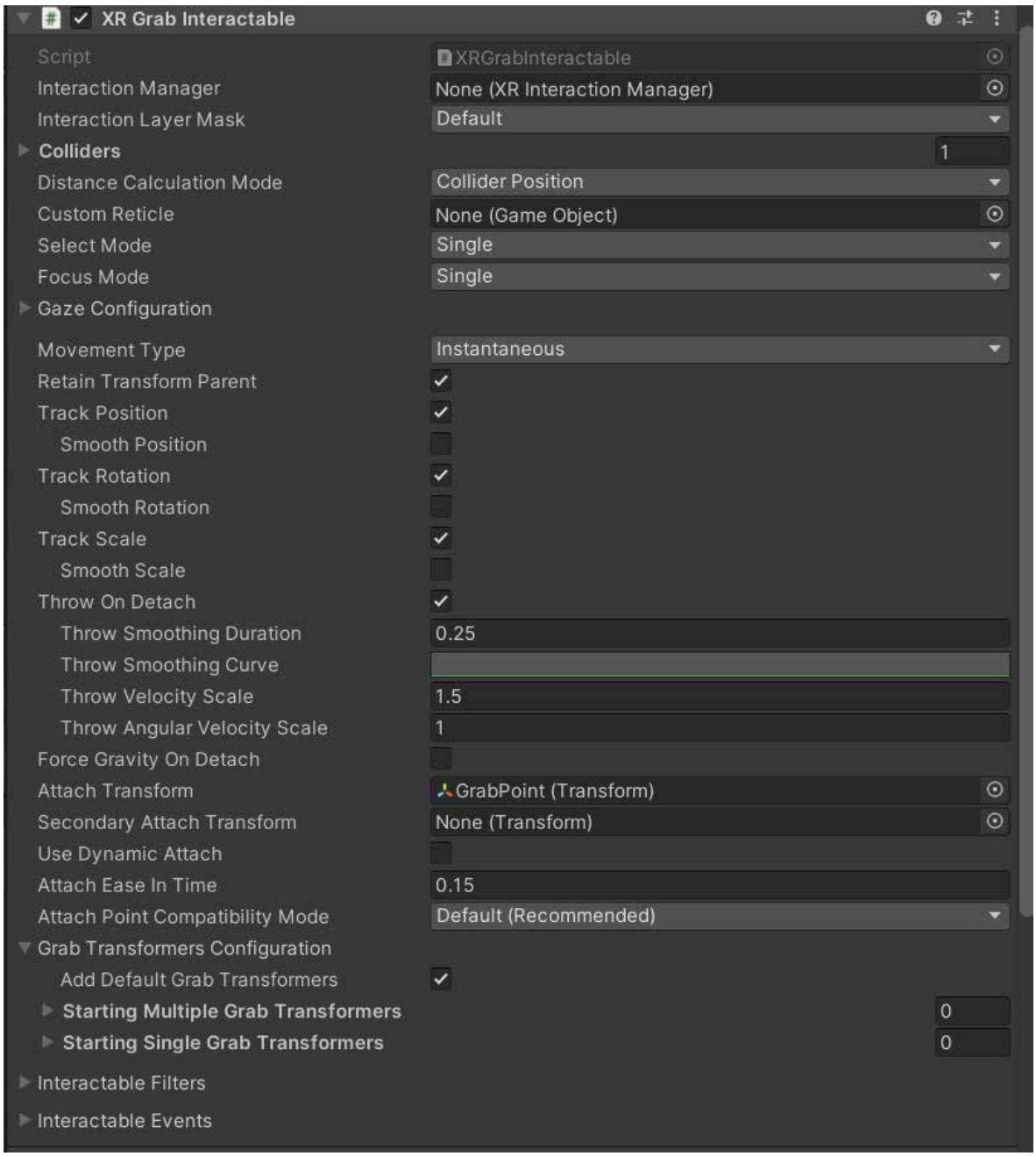


Рис. 3.36. Налаштування взаємодії з основною частиною лука

3.5.2 Система тетиви

Тетива реалізована як окремий GameObject з власним колайдером і системою взаємодії (рис. 3.37).

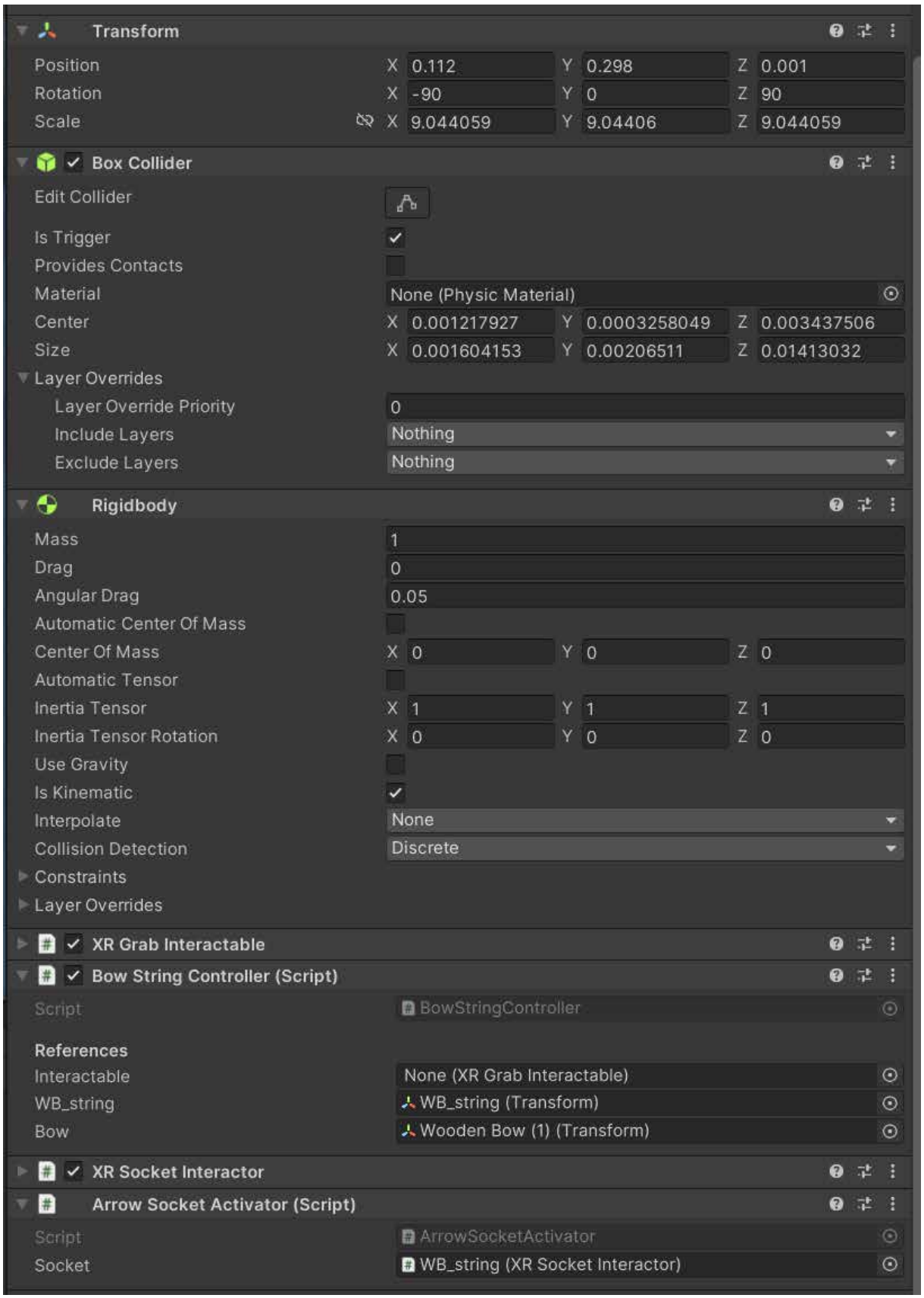


Рис. 3.37. Налаштування тетиви з компонентом взаємодії

Тетива закріплена на моделі лука в двох точках і має можливість розтягування в будь-якому напрямку. Вона оснащена такими компонентами:

- Колайдер для взаємодії з рукою гравця
- XRGrabInteractable для захоплення і натягування
- XRSocketInteractable для вставлення стріли
- Спеціалізований скрипт BowStringController для керування поведінкою тетиви

Скрипт BowStringController відповідає за керування натягом тетиви у VR-середовищі та випуск стріли при відпусканні тетиви (рис. 3.38).

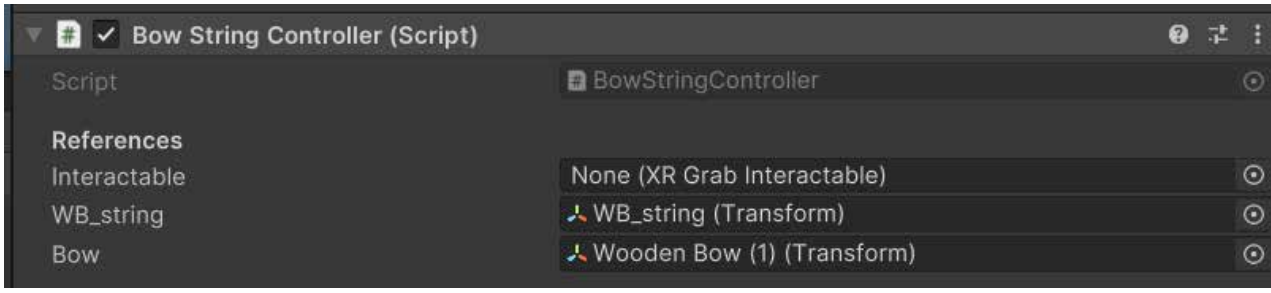


Рис. 3.38. Скрипт BowStringController для керування тетивою

Основні функції скрипта BowStringController:

- Обробка взаємодії гравця з тетивою через VR-контролери
- Реалізація механіки натягування з фізичним відгуком
- Керування випуском стріли залежно від сили натягу
- Розрахунок траєкторії стріли на основі напрямку та сили натягу

3.5.3 Система стріл

Стріла реалізована як окремий об'єкт з компонентами для взаємодії з гравцем та луком (рис. 3.39).

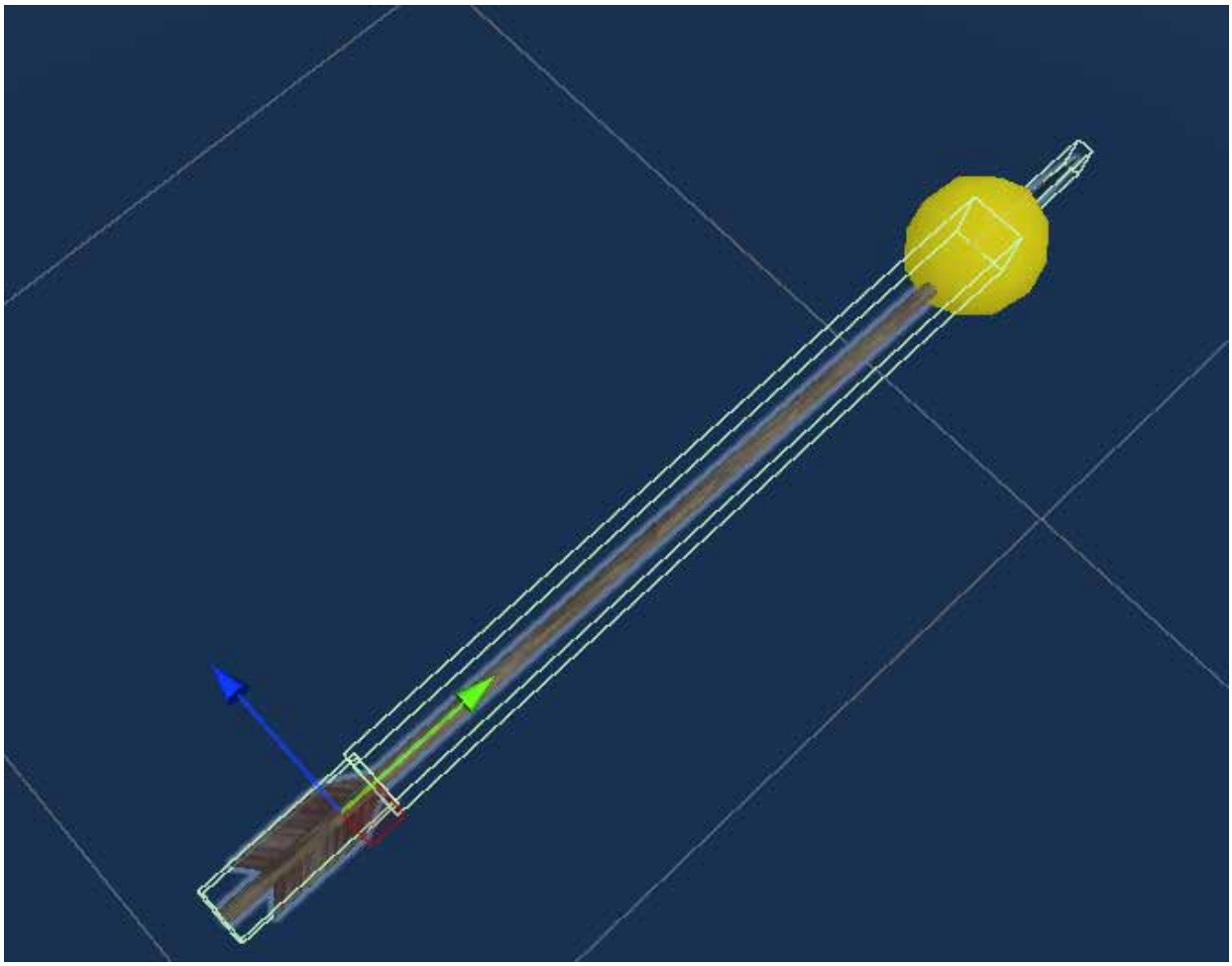


Рис. 3.39. Модель стріли з компонентами взаємодії

Стріла складається з трьох структурних елементів:

1. Наконечник – бойова частина
2. Древко – основна частина
3. Хвіст – стабілізатор для польоту

Стріла оснащена компонентами XRGrabInteractable для підбору руками та скриптом ArrowController (рис 3.40) для керування її життєвим циклом.

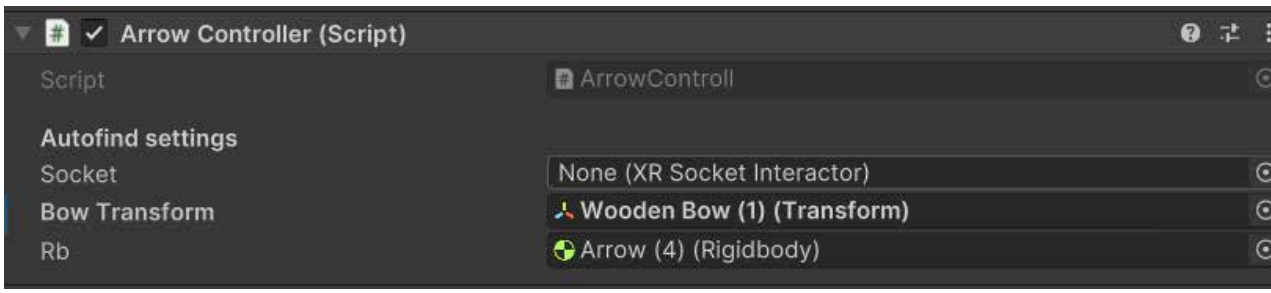


Рис. 3.40. Налаштування ArrowControlle

Скрипт ArrowController забезпечує:

1. Автоматичне підключення до елементів:

- Знаходження XRSocketInteractor (сокет лука) за тегом BowSocket
- Визначення об'єкта, до якого кріпиться стріла (лук)
- Керування фізикою стріли через Rigidbody

2. Взаємодію через VR-контролери:

- Обробку підняття стріли гравцем (OnGrab())
- Обробку відпускання стріли (OnRelease())

3. Вставлення в лук:

- Виявлення зіткнення з луком (OnTriggerEnter())
- Позиціонування стріли в точну позицію на луці
- Вимкнення фізики при закріпленні на луці

4. Натяг та політ:

- Слідування за тетивою при натягу
- Запуск стріли з відповідною силою (Fire(force))
- Увімкнення фізики для реалістичного польоту

5. Взаємодію з цілями:

- Виявлення влучення в об'єкт (OnCollisionEnter())
- Закріплення стріли в цілі (StickToTarget())
- Виклик реакції цілі на влучення (OnHit())

3.5.4 Реалізація сагайдака

Для зберігання стріл розроблено модель сагайдака з системою слотів (рис. 3.41).

Рис. 3.41. Модель сагайдака для зберігання стріл

Сагайдак має компонент XRGrabInteractable для перенесення та набір з 9 сокетів для стріл (рис. 3.42, 3.43).

Рис. 3.42. Структура сагайдака з набором сокетів

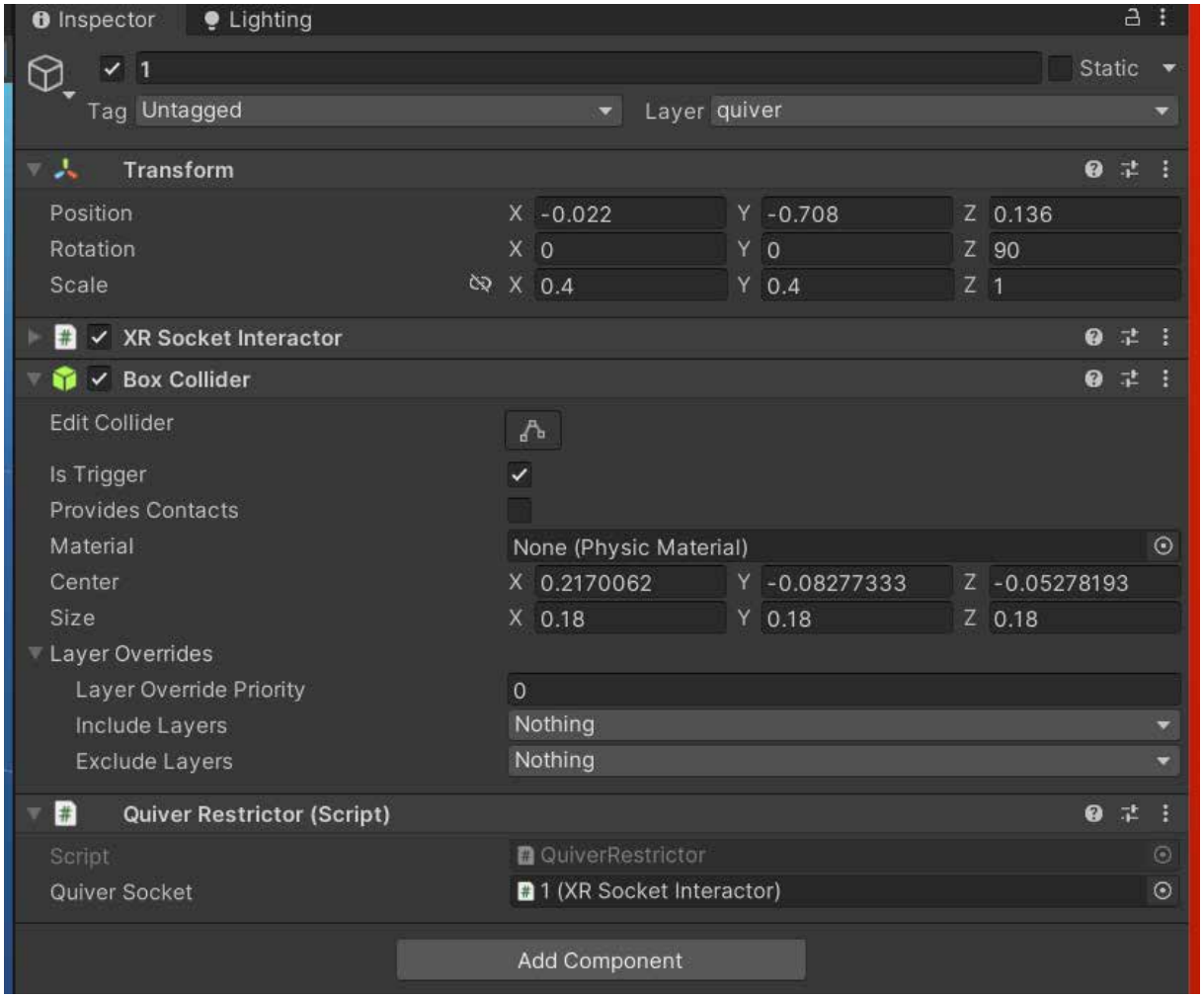


Рис. 3.43. Налаштування взаємодії з сагайдаком

Кожен із 9 сокетів має BoxCollider та SocketInteractor, налаштований на взаємодію зі стрілами (рис. 3.44).

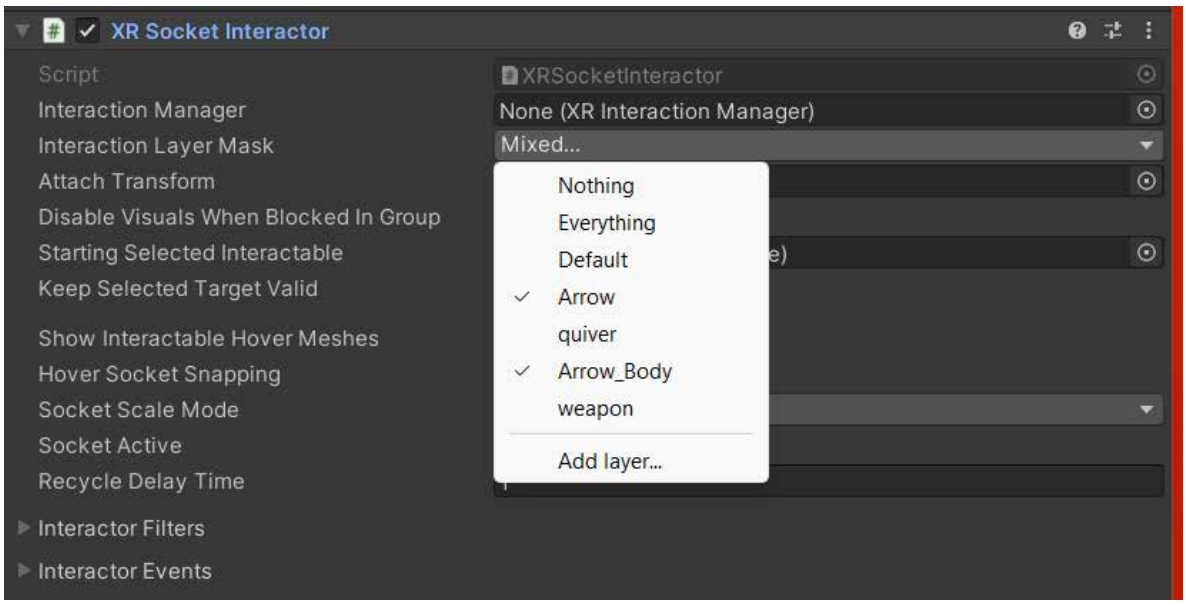


Рис. 3.44. Налаштування сокетів сагайдака для стріл

Розроблена система лука і стріл створює реалістичний VR-досвід: гравець натягує тетиву власною рукою й випускає стрілу з фізичною силою, залежною від натягу. Використання XRSocketInteractor забезпечує точне позиціонування стріли у луці, а фізичний рушій Unity гарантує реалістичний політ стріли. Таке поєднання робить процес стрільби інтуїтивно зрозумілим і плавним, що підсилює занурення гравця в ігровий процес.

3.6 Розрізання об'єктів та механіка SliceObject

Однією з ключових ігрових механік розробленої VR-гри є динамічне розрізання об'єктів, що забезпечує реалістичну взаємодію холодної зброї з об'єктами віртуального середовища. Ця механіка подібна до систем, реалізованих у таких іграх як Fruit Ninja, але адаптована для VR-середовища з урахуванням особливостей фізичної взаємодії у віртуальній реальності.

Для реалізації цієї функції використано бібліотеку EzySlice, яка дозволяє динамічно розділяти 3D-меші на частини в режимі реального часу без значного навантаження на систему. Вся логіка розрізання інкапсульована в скрипті SliceObject, який є одним із ключових компонентів розробленої системи.

Логіку реалізації базовано на підході, продемонстрованому в навчальному відео «How to Slice in VR – Unity XR Tutorial» (автор – Valem Tutorials) [2], де описано реалізацію розрізання об'єктів у VR за допомогою Unity XR. У результаті

вся функціональність була інкапсульована в скрипті SliceObject, який є одним із ключових компонентів розробленої системи.

3.6.1 Технічна реалізація механіки розрізання

До кожного ріжучого предмета (меча, сокири) прикріплено два спеціальні порожні GameObject'и — startSlicePoint та endSlicePoint. Ці точки визначають геометричну лінію удару леза та є критично важливими для точного визначення площини розрізу. Вони розташовані таким чином, щоб формувати пряму лінію, яка проходить через найгострішу частину леза (рис. 3.5).

На кожному кадрі фізичного розрахунку (метод FixedUpdate()) виконується операція трасування (raycast) між цими двома точками:

C#

Physics.Linecast(startSlicePoint.position, endSlicePoint.position, out hit, sliceableLayer)

Ця операція перевіряє, чи перетинає лінія удару будь-які об'єкти, що знаходяться в спеціально визначеному шарі sliceableLayer. Використання окремого шару для об'єктів, які можна розрізати, дозволяє оптимізувати перевірки зіткнень та уникнути непотрібних розрахунків для об'єктів, що не підлягають розрізанню.

3.6.2 Визначення площини розрізу

Якщо система виявляє перетин, виконується обчислення параметрів площини розрізу. Для цього використовується компонент VelocityEstimator, який обчислює вектор швидкості кінчика зброї в момент перетину з об'єктом. Цей вектор швидкості є ключовим для визначення напряму удару.

Нормаль площини розрізу обчислюється за допомогою векторного множення (крос-продукту) між вектором швидкості руху зброї та напрямком лінії удару:

C#

Vector3 planeNormal = Vector3.Cross(sliceDirection, velocityVector).normalized;

Така математична операція дозволяє отримати вектор, перпендикулярний до площини, утвореної напрямком руху та лінією леза, що в результаті дає нам ідеальну площину розрізу, яка відповідає геометрії руху зброї в просторі. Цей підхід забезпечує реалістичну поведінку системи, коли зброя може розрізати об'єкт лише при відповідному куті удару з достатньою швидкістю.

3.6.3 Процес розрізання об'єкта

Після визначення параметрів площини розрізу виконується сам процес розрізання за допомогою методу з бібліотеки EzySlice:

C#

SlicedHull hull = target.Slice(planeWorldNormal, planeWorldPosition, crossSectionMaterial);

Цей метод повертає структуру SlicedHull, яка містить дві частини мешу — верхню та нижню відносно площини розрізу (upperHull і lowerHull). На гранях розрізу застосовується спеціально розроблений матеріал crossSectionMaterial, який забезпечує візуалізацію внутрішньої структури розрізаного об'єкта.

3.6.4 Фізична поведінка розрізаних частин

Після геометричного розрізання оригінальний об'єкт деактивується, а на його місці створюються два нові об'єкти, що відповідають отриманим частинам. До кожної з цих частин додаються необхідні компоненти для забезпечення фізичної взаємодії в межах допоміжного методу SetupSliceComponent():

C#

```
private GameObject SetupSliceComponent(GameObject slicedObject) {
// Додаємо Rigidbody для фізичної взаємодії
Rigidbody rb = slicedObject.AddComponent<Rigidbody>();
rb.interpolation = RigidbodyInterpolation.Interpolate;

// Додаємо MeshCollider на основі розрізаного мешу
```

```
MeshCollider collider = slicedObject.AddComponent<MeshCollider>();
collider.convex = true;

return slicedObject;
}
```

Кожна частина отримує компоненти Rigidbody і MeshCollider (з опцією convex = true для оптимізації фізичних розрахунків). Для досягнення більш реалістичного ефекту "розрізання" до обох частин застосовується невелика імпульсна сила через метод AddExplosionForce():

```
C#
rb.AddExplosionForce(explosionForce, hit.point, explosionRadius);
```

Це створює ефект розбризкування фрагментів під час розрізання, що значно підвищує реалістичність взаємодії та забезпечує візуальний фідбек гравцю.

3.6.5 Оптимізація та управління ресурсами

Щоб уникнути захаращення сцени великою кількістю розрізаних об'єктів, до кожної утвореної частини додається компонент DestroyAfterTime, який знищує їх після певного проміжку часу:

```
C#
slicedObject.AddComponent<DestroyAfterTime>().timeToDestroy = destroyAfterSeconds;
```

Цей підхід забезпечує правильне управління пам'яттю та підтримує стабільну продуктивність VR-додатку навіть при інтенсивному використанні механіки розрізання.

3.6.6 Ігрова логіка та нарахування очків

Якщо розрізаний об'єкт мав певний тег (наприклад, "Beat" або "Enemy" для ворогів), скрипт SliceObject викликає відповідні події та нараховує гравцю очки:

```
C#
```

```
if (hit.transform.CompareTag("Beat")) {
GameManager.Instance.AddScore(pointsPerSlice);
OnBeatSliced?.Invoke();
}
```

Ця система дозволяє інтегрувати механіку розрізання з іншими ігровими системами, такими як система підрахунку очок, прогресу гравця або тригерів ігрових подій.

3.6.7 Результати та відчуття присутності

Реалізована система динамічного розрізання об'єктів забезпечує високий рівень інтерактивності та значно підвищує відчуття присутності у віртуальному світі. На відміну від традиційних ігор, де розрізання часто імітується заздалегідь підготовленими анімаціями або простим зникненням об'єктів, ця система забезпечує справжню динамічну інтерактивність, де результат залежить від конкретних дій гравця (швидкості, напрямку та кута удару).

Гравець отримує миттєвий і очевидний фізичний відгук на свої дії — об'єкти буквально розрізаються перед його очима відповідно до руху зброї (рис. 3.7). Такий підхід значно посилює відчуття реальної взаємодії з віртуальним світом, що є одним із ключових аспектів успішного VR-досвіду.

3.7 Міні-гра "Розрізання кубів"

Для демонстрації практичного застосування розробленої системи розрізання об'єктів було створено повнофункціональну міні-гру у стилі слешера, концептуально подібну до популярної VR-гри Beat Saber. Дана міні-гра дозволяє користувачу використовувати віртуальну зброю для розрізання геометричних об'єктів, що рухаються у його напрямку із заданою швидкістю та траєкторією.

3.7.1 Концепція та геймплей

Основною метою міні-гри є тестування реакції та точності рухів гравця у віртуальному середовищі. Користувач, екіпірований VR-шоломом та контролерами, взаємодіє з віртуальною зброєю та намагається перерізати кубічні об'єкти, що летять у його напрямку.

Ігровий процес базується на наступних ключових механіках:

- динамічна генерація кубів з різноманітними траєкторіями руху;
- поступове підвищення складності через збільшення швидкості та частоти появи об'єктів;
- система підрахунку очок та відображення стану здоров'я гравця;
- механізм покарання за пропущені куби (зменшення здоров'я).

3.7.2 Архітектура та технічна реалізація

Вся ігрова логіка міні-гри інкапсульована в центральному скрипті FireOnCube, який керує генерацією, рухом та взаємодією кубів з гравцем. Основні компоненти системи включають:

Ключові параметри конфігурації:

- bullet — префаб куба для створення ігрових об'єктів;
- spawnPoint та EndPoint — точки початку та закінчення траєкторії руху;
- fireSpeed та fireInterval — швидкість руху та інтервал між запусками;
- параметри випадкового зміщення позиції та напрямку руху;
- компоненти UI для відображення здоров'я та рахунку гравця.

3.7.3 Система генерації та руху об'єктів

Основою ігрового процесу є алгоритм динамічної генерації кубів з випадковими параметрами руху. Принцип роботи системи полягає у створенні об'єктів з випадковими зміщеннями позиції та напрямку руху:

C#

// Створення випадкових зміщень позиції та напрямку

```
Vector3 randomPositionOffset = new Vector3(
    Random.Range(-positionOffsetRange, positionOffsetRange),
    Random.Range(-positionOffsetRange, positionOffsetRange),
    Random.Range(-positionOffsetRange, positionOffsetRange)
);

// Встановлення швидкості з урахуванням напрямку руху
spawnedBullet.GetComponent<Rigidbody>().velocity =
    spawnedBullet.transform.forward * fireSpeed;
```

Така реалізація забезпечує різноманітність геймплею через непередбачуваність траєкторій руху кубів та їх початкових позицій.

3.7.4 Механізми управління станом гри

Система оновлення стану гри функціонує за принципом покадрового моніторингу та включає три основні підсистеми:

Контроль генерації об'єктів: Система відстежує час між створенням кубів та динамічно регулює параметри складності:

```
C#
if (timeSinceLastFire >= fireInterval) {
    FireBullet();
    fireSpeed += 0.5f;       // Збільшення швидкості
    fireInterval -= 0.1f;    // Зменшення інтервалу
}
```

Детекція пропущених об'єктів: Алгоритм перевіряє позицію всіх активних кубів відносно кінцевої точки та автоматично знищує об'єкти, що пройшли повз гравця, з відповідним зменшенням здоров'я.

Управління життєвим циклом гри: При досягненні критичного рівня здоров'я система автоматично скидає параметри гри до початкових значень, забезпечуючи можливість повторного проходження.

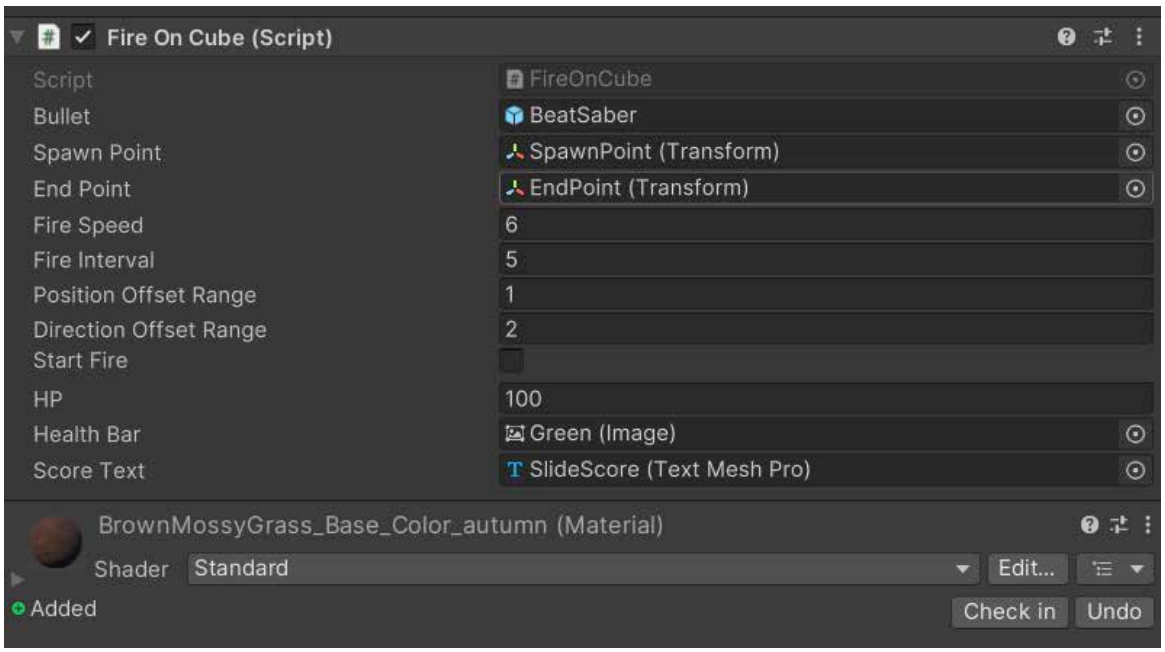


Рис 3.35 – Параметри скрипта для міні гри "Розрізання кубів" в інтеракторі

3.7.5 Інтеграція з системою розрізання об'єктів

Взаємодія між міні-грою та основною системою розрізання реалізована через механізм тегування об'єктів. При успішному розрізанні куба з тегом "Beat" активується логіка нарахування очок:

```
C#
if (hit.transform.CompareTag("Beat")) {
   GameManager.Instance.AddScore(pointsPerSlice);
   OnBeatSliced?.Invoke();
}
```

Такий підхід забезпечує тісну інтеграцію міні-гри з основними механіками проекту та дозволяє централізовано керувати системою винагород.

3.7.6 Візуальний інтерфейс та користувацький досвід

Інтерфейс міні-гри розроблено з урахуванням специфіки VR-середовища та включає інтуїтивно зрозумілі елементи керування станом гри.

Візуальне представлення включає:

- Шкалу здоров'я гравця (healthBar), що зменшується при пропуску кубів
- Лічильник очок (scoreText), що відображає поточний результат гравця
- Візуалізацію кубів, що летять на гравця з різних напрямків

Рис 3.36 – Локація міні гри "Розрізання кубів"

Рисунок 3.37 - Демонструє активну фазу гри з відображенням шкали здоров'я, лічильника очок та кубів, що рухаються у напрямку гравця.

Ключові елементи інтерфейсу включають:

- індикатор здоров'я гравця з візуальним зменшенням при пропуску кубів;
- лічильник очок з динамічним оновленням при успішних розрізаннях;
- просторове відображення кубів з реалістичною фізикою руху.
-

3.7.7 Оптимізація продуктивності

Для забезпечення стабільної роботи міні-гри у VR-середовищі реалізовано комплекс оптимізаційних рішень:

Управління життєвим циклом об'єктів: Автоматичне знищення кубів через визначений час або при досягненні кінцевої точки запобігає накопиченню непотрібних об'єктів у пам'яті.

Ефективний пошук об'єктів: Використання системи тегів замість пошуку за іменами або компонентами значно підвищує продуктивність операцій детекції.

Контрольована складність: Обмеження мінімального інтервалу між генерацією кубів запобігає критичному перевантаженню системи при високих рівнях складності.

3.7.8 Результати та перспективи розвитку

Реалізована міні-гра успішно демонструє практичне застосування системи розрізання об'єктів та створює захоплюючий ігровий досвід у VR-середовищі. Ключові досягнення включають:

- високий рівень інтерактивності з миттєвим відгуком на дії користувача;
- динамічну систему регулювання складності;
- ефективну інтеграцію з основними механіками проекту;
- оптимізовану продуктивність для VR-застосувань.

Потенційні напрямки подальшого розвитку включають розширення типології об'єктів, інтеграцію з аудіосистемою для ритмічної синхронізації, додавання візуальних ефектів та розробку системи прогресії з різними рівнями складності.

Створена міні-гра є не лише технічною демонстрацією можливостей системи розрізання об'єктів, а й самостійним ігровим продуктом, що надає користувачеві повноцінний VR-досвід взаємодії з віртуальною зброєю.

3.8 Розробка міні-ігор з використанням системи лука

Для демонстрації ефективності розробленої системи лука та стрільби було створено дві міні-гри, які дозволяють користувачу випробувати свої навички

стрільби з лука у віртуальній реальності. Ці ігри служать не лише практичним прикладом застосування розробленої системи, але й надають можливість оцінити якість взаємодії з віртуальним середовищем у розважальному форматі.

3.8.1 Міні-гра "Стаціонарні мішені"

Перша міні-гра являє собою класичну стрілецьку галерею з трьома мішенями різних розмірів. Основна мета гри полягає у перевірці точності прицілювання користувача та його здатності влучати в динамічні мішені.

3.8.1.1 Ігрова механіка

Як показано на рисунку 3.48, ігровий процес базується на взаємодії з рухомими мішенями, які створюють додатковий виклик для гравця. Система включає наступні ключові елементи:

Система руху мішеней: Кожна мішень виконує періодичні вертикальні коливання з індивідуальними параметрами амплітуди та швидкості, що створює різний рівень складності для кожної цілі.

Механіка взаємодії: При влученні стріли мішень реагує згідно з фізичними законами – падає під дією гравітації та через визначений інтервал часу автоматично повертається у вихідне положення для продовження гри.

Рисунок 3.48 – Ігровий процес міні-гри "Стаціонарні мішені" з трьома мішенями різного розміру

3.8.1.2 Реалізація системи мішеней

Архітектура мішеней побудована на принципі модульності, де кожна мішень представляється окремим GameObject із набором компонентів для обробки колізій та керування поведінкою.

Основу функціональності забезпечує скрипт TargetMover, який реалізує три ключові аспекти поведінки мішені:

Коливальний рух: Використовується синусоїдальна функція для створення плавного вертикального руху з можливістю налаштування амплітуди та швидкості для кожної мішені індивідуально.

Реакція на влучення: При детекції зіткнення зі стрілою система вимикає кінематичний режим Rigidbody та застосовує фізичну силу для реалістичного падіння мішені.

Система відновлення: Через заданий інтервал часу мішень автоматично повертається у початкове положення та відновлює свій рух.

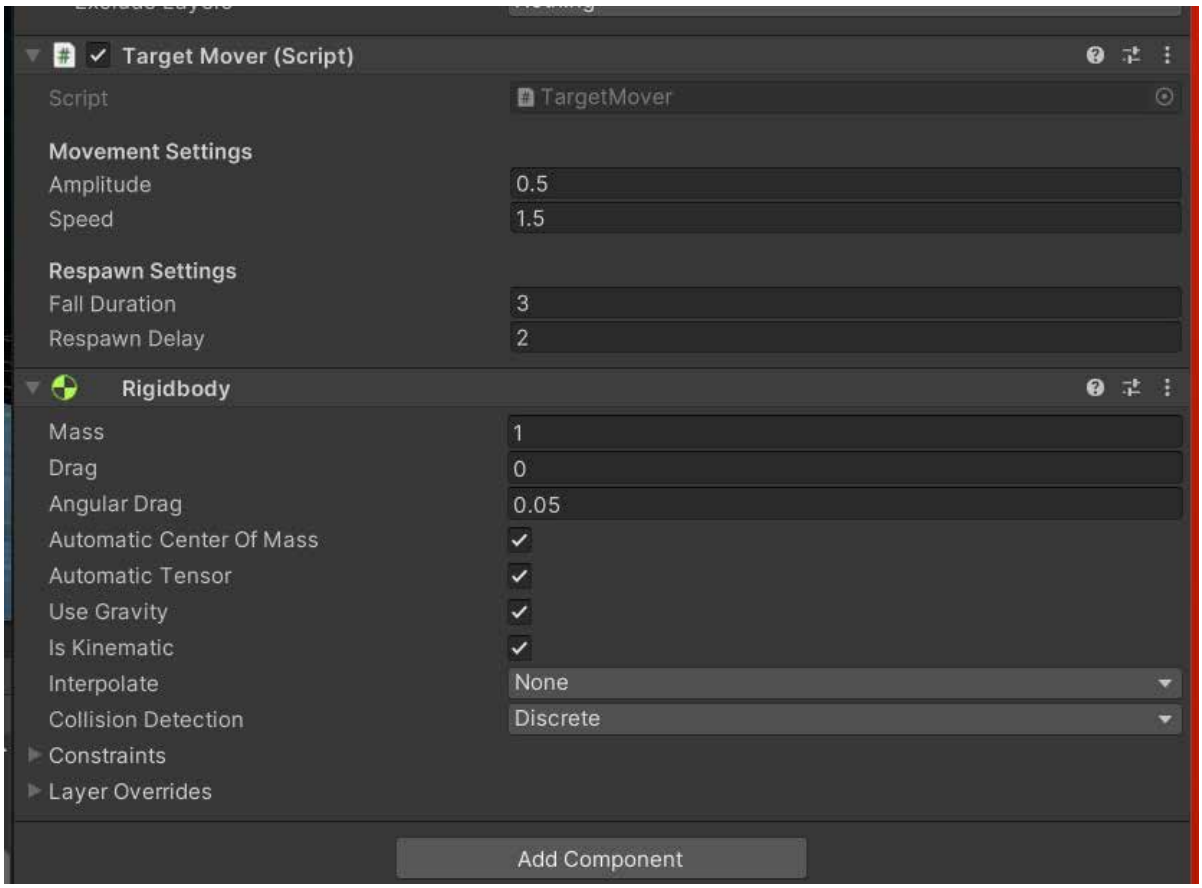


Рисунок 3.49 – Налаштування компонентів мішені в Inspector Unity

Важливою особливістю реалізації є керування обмеженнями обертання Rigidbody. У нормальному стані мішень обмежена в обертанні по осях X та Z для забезпечення стабільного коливального руху, але при влученні всі обмеження знімаються для природного падіння:

```
C#
// Обмеження обертання під час нормального руху
rb.constraints = RigidbodyConstraints.FreezeRotationX |
                 RigidbodyConstraints.FreezeRotationZ;

// Зняття обмежень при влученні
rb.constraints = RigidbodyConstraints.None;
```

3.8.2 Міні-гра "Хвильовий стрілець"

Друга міні-гра представляє більш складну та динамічну версію стрілецької галереї з прогресивним наростанням складності та системою обмеженого часу, що створює напружений ігровий процес.

3.8.2.1 Ігрова механіка

Концепція гри базується на системі хвиль із поступовим збільшенням складності (рисунок 3.50):

Прогресивна система хвиль: Гра розпочинається з однієї великої мішені, з кожною наступною хвилею кількість мішеней подвоюється, а їх розмір пропорційно зменшується.

Часові обмеження: Гравець має обмежений час на завершення всіх хвиль, що додає елемент стратегічного планування.

Система винагород: За кожне влучення нараховуються очки, а також надається бонусний час, розмір якого збільшується з кожною новою хвилею.

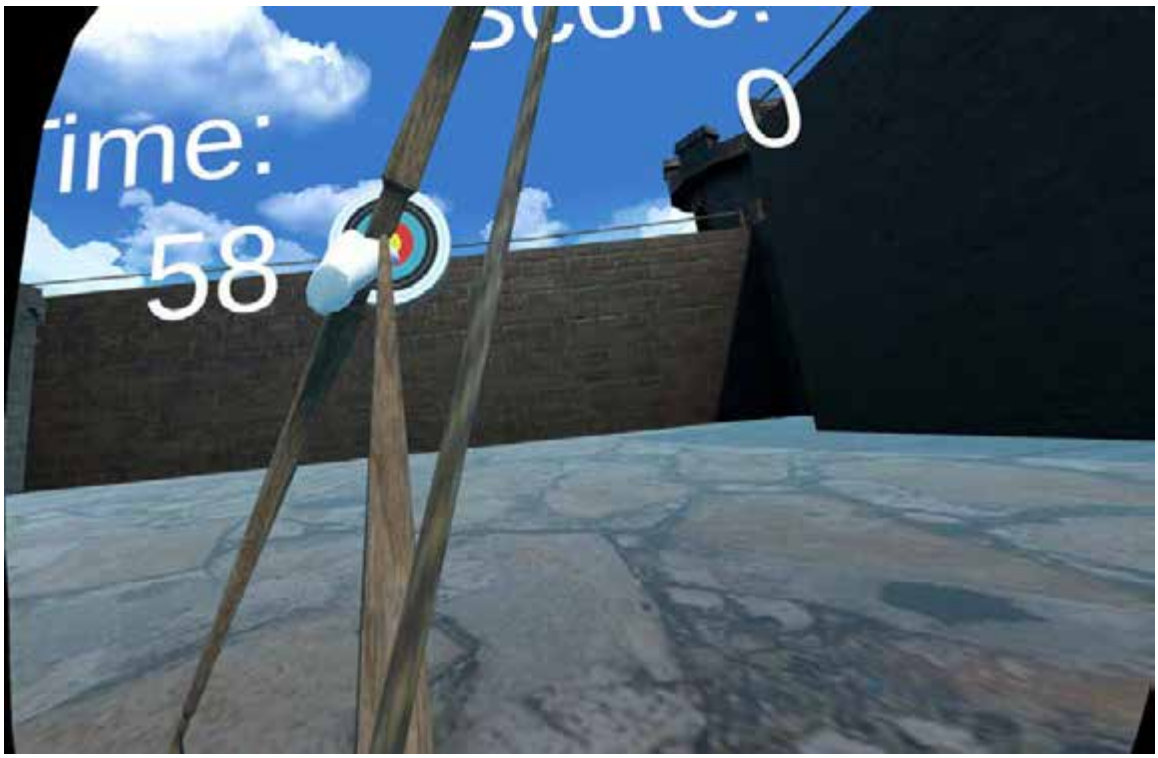


Рисунок 3.50 – Інтерфейс гри "Хвильовий стрілець" з відображенням поточної хвилі та часу

3.8.2.2 Архітектура ігрової логіки

Центральним елементом системи є скрипт ArcherGameController, який координує всі аспекти ігрового процесу:

Керування станом гри: Контролер відстежує поточний стан гри, час що залишився, рахунок гравця та оновлює відповідні елементи користувацького інтерфейсу в реальному часі.

Система генерації мішеней: Алгоритм створення нових мішеней враховує номер поточної хвилі для розрахунку кількості, розміру та швидкості руху цілей. Використовується спеціальний алгоритм розміщення для запобігання перекриттю мішеней у просторі.

Прогресія складності: З кожною новою хвилею система автоматично збільшує кількість мішеней, підвищує швидкість їх руху та зменшує розмір для

створення прогресивного виклику.

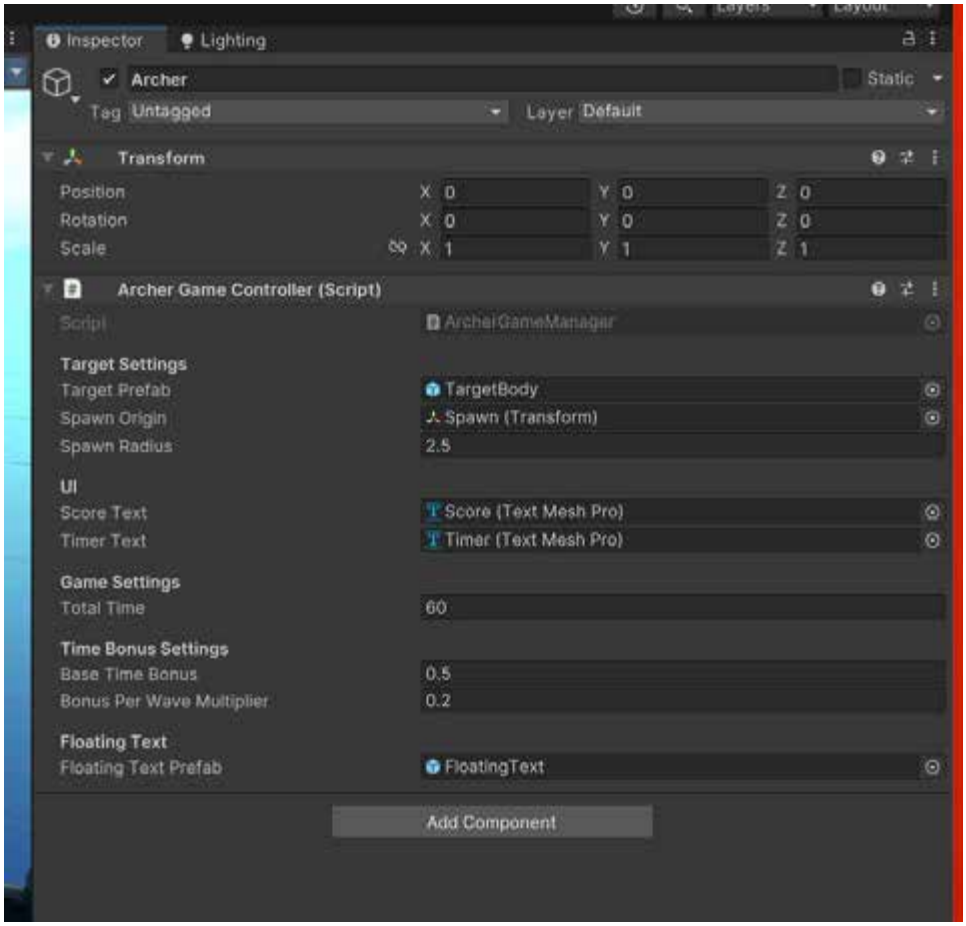


Рисунок 3.51 – Параметри скрипта "Хвильовий стрілець" у інтеракторі

3.8.2.3 Система часових бонусів

Унікальною особливістю гри є адаптивна система часових винагород, яка додає стратегічний елемент до ігрового процесу. Розмір бонусу розраховується за формулою:

C#

```
float bonusTime = baseTimeBonus + bonusPerWaveMultiplier *
        Mathf.Log(currentWaveSize, 2);
```

Логарифмічне масштабування забезпечує збалансоване зростання винагороди відповідно до складності хвилі, мотивуючи гравця до точної стрільби на пізніх етапах гри.

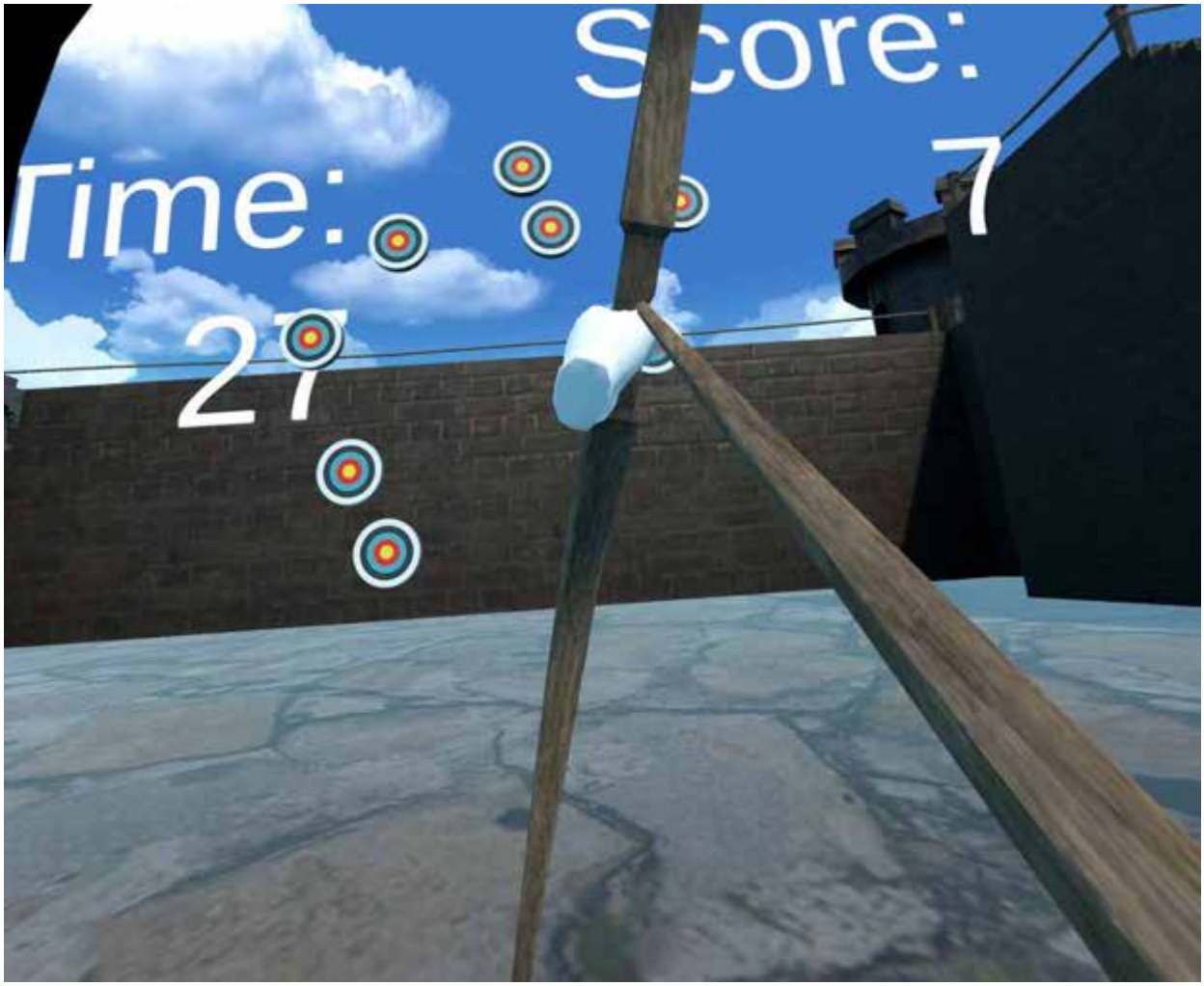


Рис 3.52 - Ігровий процес на пізніх етапах гри "Хвильовий стрілець"

3.8.3 Оптимізація продуктивності

Для забезпечення стабільної роботи системи при збільшенні кількості мішеней на пізніх хвилях було реалізовано кілька оптимізаційних рішень:

Алгоритм розміщення без перекриття: Система використовує ефективний алгоритм перевірки відстаней між позиціями для запобігання накладанню мішеней, що покращує як візуальну складову, так і ігровий досвід.

Уніфікований інтерфейс мішеней: Створено загальний інтерфейс для всіх типів мішеней, що дозволяє легко адаптувати систему для різних ігрових сценаріїв без дублювання коду.

Ефективне керування життєвим циклом: Мішені автоматично видаляються після влучення з невеликою затримкою, що запобігає накопиченню непотрібних об'єктів у сцені.

3.8.3 Результати та висновки

Розроблені міні-ігри успішно демонструють функціональні можливості системи лука у віртуальній реальності та підтверджують ефективність обраних технічних рішень:

Інтуїтивність взаємодії: Природність керування луком у VR-середовищі забезпечує швидке освоєння системи користувачами різного рівня підготовки.

Масштабованість складності: Адаптивні механіки дозволяють створювати ігровий досвід, що підходить як для новачків, так і для досвідчених користувачів VR.

Технічна стабільність: Система демонструє стабільну роботу навіть при високому навантаженні з великою кількістю динамічних об'єктів.

Потенціал розширення: Модульна архітектура створює основу для розробки більш складних ігрових сценаріїв з використанням луку у майбутніх проектах.

Створені міні-ігри не тільки служать демонстрацією технічних можливостей розробленої системи, але й формують базис для подальшого розвитку VR-додатків з використанням традиційної зброї у віртуальному просторі.

3.9 Реалізація інтерактивної VR-кнопки

Для забезпечення запуску розроблених міні-ігор ("Розрізання кубів" і "Хвильовий стрілець") було створено спеціалізований компонент віртуальної кнопки, що відповідає за обробку взаємодії користувача та активацію відповідних ігрових сценаріїв.

3.9.1 Структура та компоненти VR-кнопки

Розроблена VR-кнопка складається з трьох основних елементів, як показано на рис. 3.53:

Рис. 3.53. Ієрархічна структура VR-кнопки в інспекторі Unity

Компоненти кнопки включають:

1. База кнопки — статичний компонент, що служить основою для рухомої частини
2. Рухома частина кнопки — інтерактивний елемент, що реагує на взаємодію
3. Колайдер-тригер — компонент для виявлення зіткнень з рукою користувача або іншими об'єктами

Така трикомпонентна структура забезпечує реалістичне відтворення фізичної взаємодії з кнопкою у віртуальному просторі, створюючи відчуття натискання реального фізичного об'єкта.

3.9.2 Візуальне представлення та дизайн кнопки

В проекті використано дизайн кнопки, що відповідає загальному стилістичному рішенню віртуального середовища. Як показано на рис. 3.54, кнопка має загальний вигляд, що інтуїтивно підказує користувачеві можливість взаємодії:

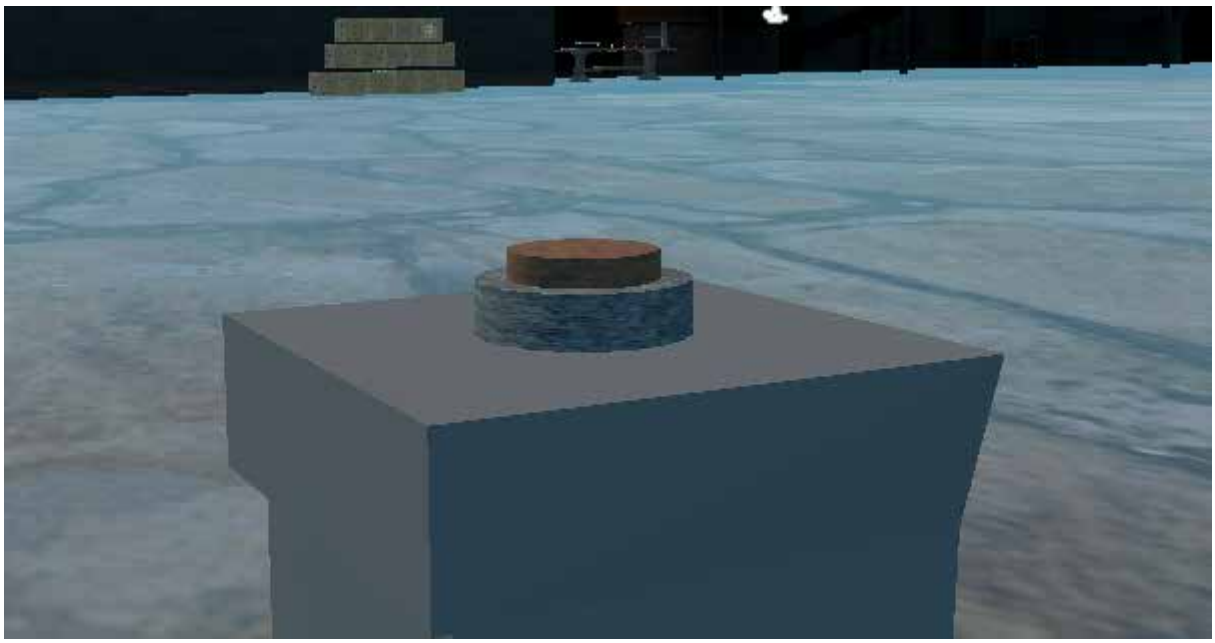


Рис. *3.54. Загальний вигляд VR-кнопки у сцені*

База кнопки виділена помаранчевим кольором для привернення уваги користувача (рис. 3.55):

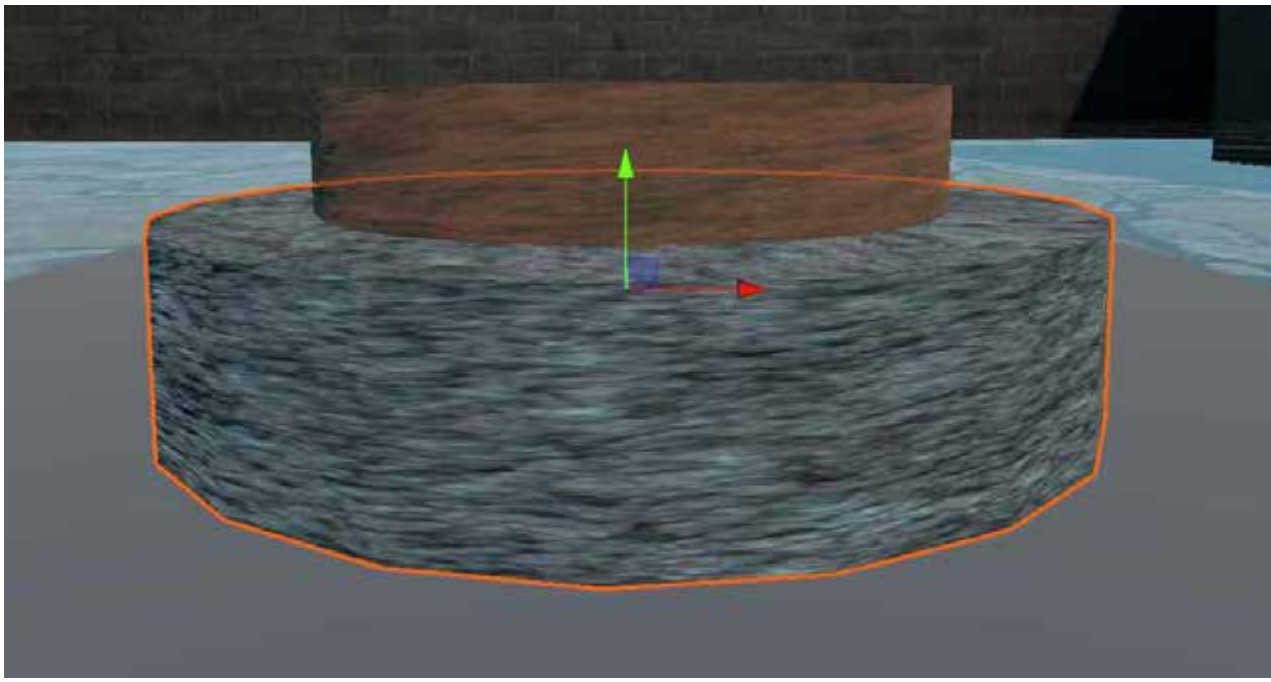


Рис. 3.55. База кнопки з помаранчевим підсвічуванням

Рухома частина кнопки виконана в нейтральному кольорі для чіткого візуального розмежування інтерактивного елемента (рис. 3.56):

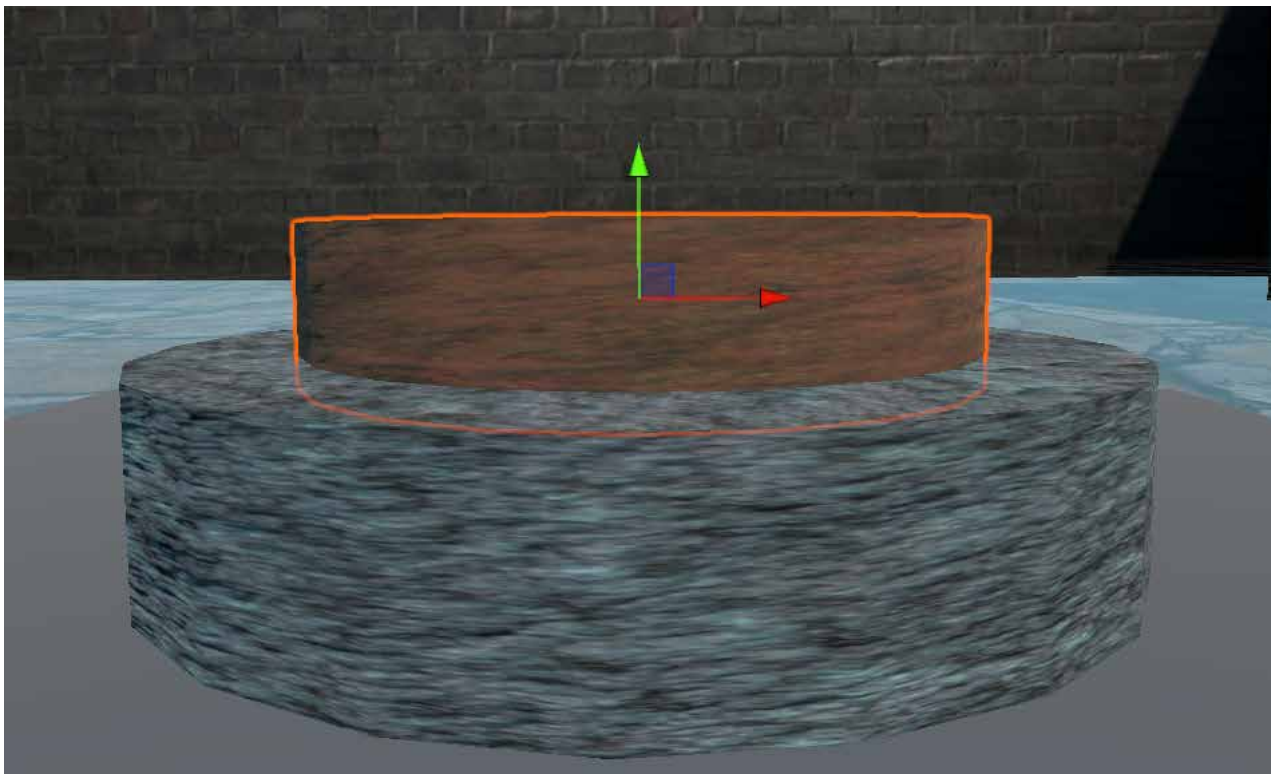


Рис. 3.56. Рухома частина кнопки

Для забезпечення інтерактивності, на кнопку встановлено невидимий колайдер-тригер, що відповідає за реєстрацію взаємодій (рис. 3.57):

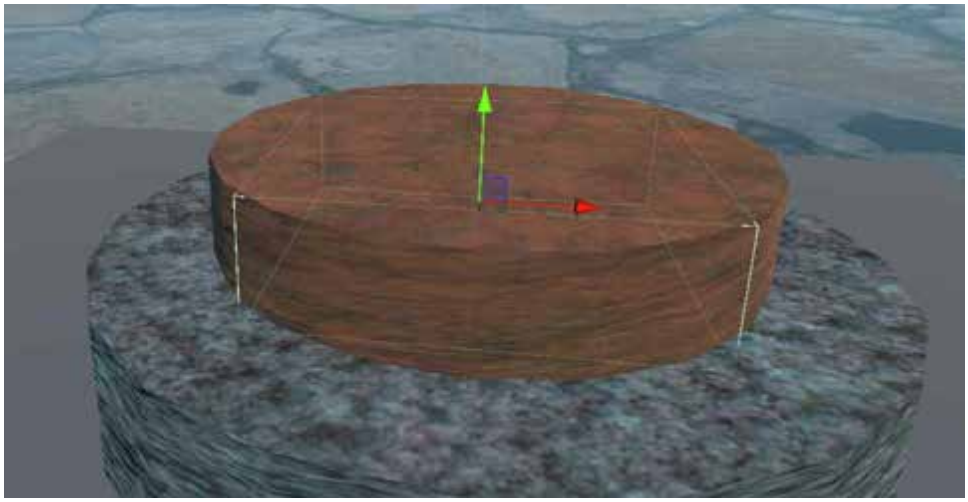


Рис. 3.57. Візуалізація колайдера-тригера кнопки

3.9.3 Конфігурація компонентів кнопки

Для забезпечення всіх функціональних можливостей, на колайдер-тригер кнопки встановлено компонент ButtonVR та відповідні налаштування (рис. 3.58):

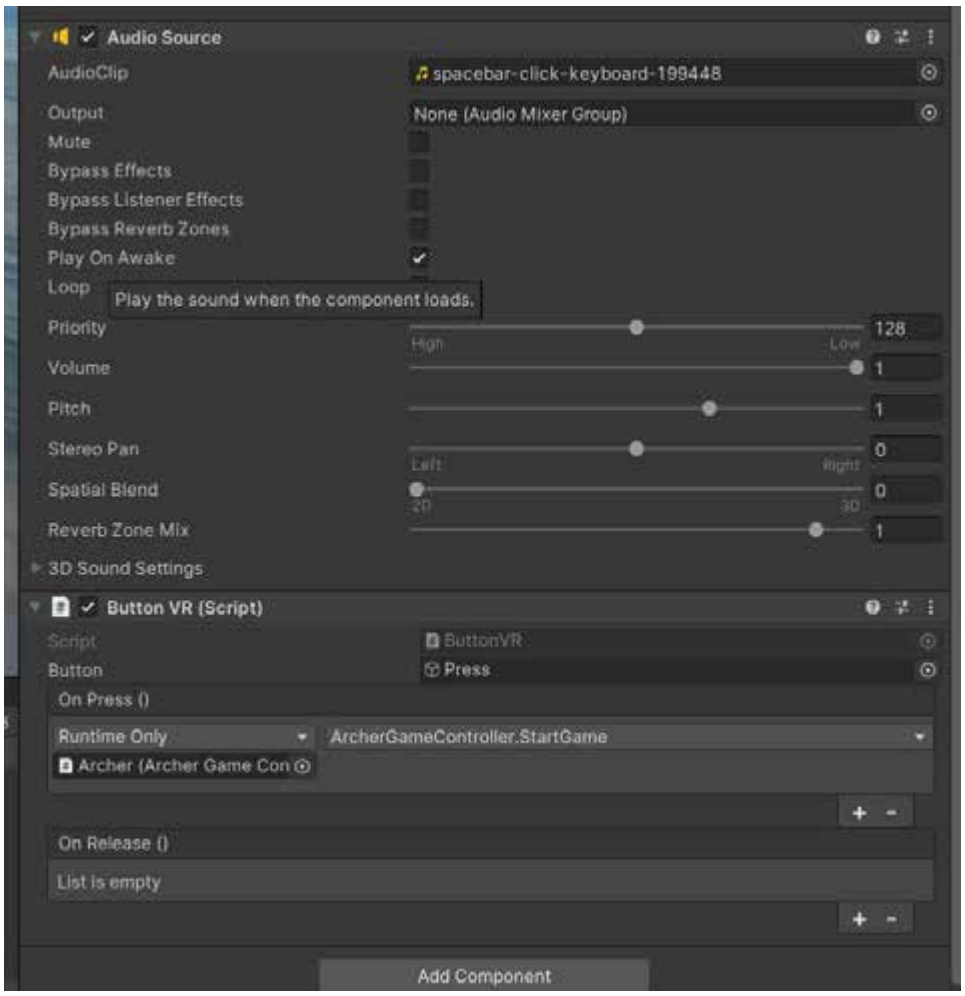


Рис. 3.58. Компоненти та налаштування колайдера кнопки в інспекторі

Як видно з рисунка, кнопка містить:

- Посилання на рухому частину кнопки
- Події onPress та onRelease, що активуються при натисканні та відпусканні
- Компонент AudioSource для звукового супроводу
- Компонент Box Collider з увімкненим режимом Is Trigger

3.9.4 Функціональні можливості кнопки

VR-кнопка реалізована через скрипт ButtonVR, який виконує наступні основні функції:

1. Відстеження стану натискання
2. Візуальна анімація руху кнопки при взаємодії
3. Звуковий супровід натискання

4. Виклик подій при натисканні та відпусканні

Ключовою особливістю реалізації є використання системи подій Unity (UnityEvent), що дозволяє гнучко прив'язувати різні функції до взаємодії з кнопкою без необхідності модифікації коду компонента.

3.9.5 Технічна реалізація механіки кнопки

Механіка роботи кнопки базується на системі тригерів Unity. Коли об'єкт (наприклад, контролер VR або віртуальна рука користувача) входить у зону тригера кнопки, активується механізм натискання:

```
C#
private void OnTriggerEnter(Collider other)
{
if (!isPressed)
{
button.transform.localPosition = new Vector3(0, 0.187f, 0);
presser = other.gameObject;
onPress.Invoke();
sound.Play();
isPressed = true;
}
}
```

При цьому відбуваються наступні дії:

1. Зміна візуальної позиції кнопки (імітація натискання)
2. Запам'ятовування об'єкта, що натиснув кнопку
3. Виклик події onPress для активації прив'язаних функцій
4. Відтворення звуку натискання
5. Встановлення прапорця стану кнопки

Так само при виході об'єкта із зони тригера відбувається зворотна послідовність дій:

C#

```
private void OnTriggerExit(Collider other)
{
button.transform.localPosition = new Vector3(0, 0.197f, 0);
onRelease.Invoke();
isPressed = false;
}
```

3.9.6 Інтеграція з міні-іграми

Кнопка інтегрується з розробленими міні-іграми через систему подій Unity. Для кожної міні-гри налаштовано відповідні події, що активуються при натисканні на кнопку:

1. Для міні-гри "Розрізання кубів":
   - Виклик методу StartFire() зі скрипта FireOnCube, що ініціює генерацію та рух кубів
2. Для міні-гри "Хвильовий стрілець":
   - Виклик методу запуску гри у скрипті ArcherGameController, що починає генерацію мішеней та відлік часу

Такий підхід дозволяє використовувати кнопки як універсальні активатори для різних ігрових сценаріїв без необхідності створення специфічних контролерів для кожної міні-гри.

3.9.6 Звуковий супровід взаємодії

Для посилення реалістичності взаємодії, VR-кнопка оснащена звуковим компонентом AudioSource, що відтворює звук натискання. Звук активується в момент входу об'єкта в зону тригера кнопки:

```
C#
sound.Play();
```

Реалістичний звуковий відгук підвищує занурення користувача у віртуальне середовище, створюючи додатковий канал сенсорного зворотного зв'язку.

3.9.7 Підвищення стійкості до помилок

Для забезпечення надійної роботи кнопки в різних ситуаціях, реалізовано захист від помилкових спрацьовувань:

1. Перевірка стану кнопки перед обробкою взаємодії:

C#

```
if (!isPressed)
{
// Обробка натискання
}
```

2. Ініціалізація початкового стану кнопки при запуску:

C#

```
void Start()
{
sound = GetComponent<AudioSource>();
isPressed = false;
}
```

Ці заходи запобігають багаторазовій активації подій кнопки при одному натисканні та забезпечують коректний початковий стан системи.

3.9.8 Висновки щодо механізму VR-кнопок

Впроваджений механізм VR-кнопок значно розширює функціональні можливості розробленої системи, забезпечуючи:

1. Інтуїтивність взаємодії — користувач може запускати міні-ігри природним рухом віртуальної руки або контролера
2. Універсальність — механізм кнопки можна використовувати для активації різних функцій системи
3. Зворотний зв'язок — візуальний та звуковий відгук підвищує занурення користувача
4. Модульність — завдяки використанню системи подій Unity, функціональність кнопки легко розширюється

Розроблена VR-кнопка служить не лише інструментом для запуску міні-ігор, але й демонструє загальний підхід до створення інтерактивних елементів управління у віртуальному середовищі. Цей підхід може бути розширений в майбутньому для реалізації більш складних інтерфейсних компонентів, таких як перемикачі, слайдери або віртуальні панелі управління.

3.10 Локації та додаткові інтерактивні об'єкти віртуального середовища

У межах проєкту було розроблено комплекс взаємопов'язаних локацій, що формують цілісне та імерсивне VR-середовище. Кожна локація не лише забезпечує унікальний ігровий досвід, але й демонструє різні функціональні можливості розробленої системи. Спроєктовані віртуальні простори дозволяють користувачеві повноцінно взаємодіяти з навколишнім світом, досліджувати його та відпрацьовувати різноманітні ігрові механіки в контексті віртуальної реальності.

3.10.1 Тренувальна арена

Початковою локацією для користувача є тренувальна арена — спеціально спроєктований відкритий простір, що слугує для ознайомлення з базовими елементами геймплею. Ця локація виконує роль інтерактивного навчального середовища, де користувач може безпечно опанувати основні механіки взаємодії з VR-світом.

Центральним елементом арени є стенд зі зброєю, де представлено колекцію мечів різних типів, луків та стріл (Рис. 3.59). Кожен предмет озброєння оснащено компонентом XRGrabInteractable, що забезпечує можливість інтуїтивного захоплення та маніпуляції за допомогою VR-контролерів.

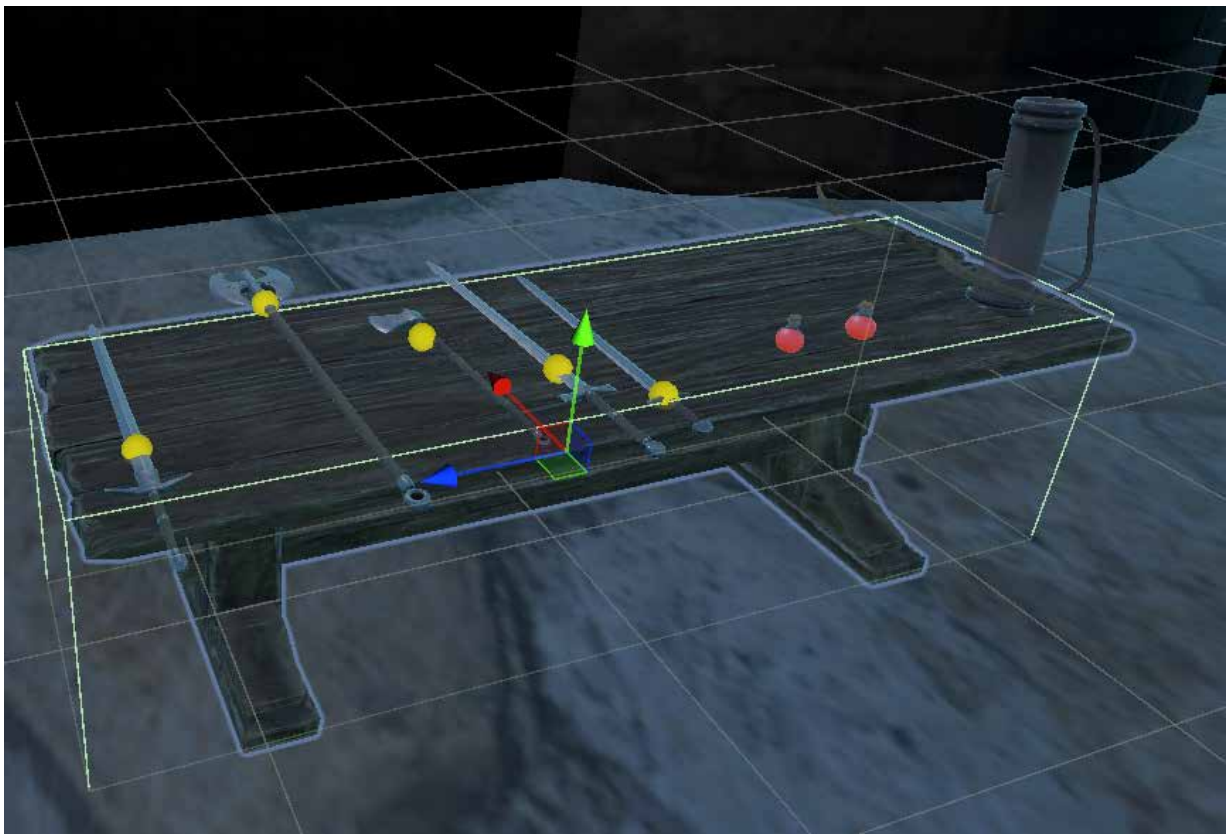


Рис. 3.59. Стенд із колекцією зброї на тренувальній арені

Мінімалістичний дизайн локації з обмеженою кількістю відволікаючих елементів дозволяє користувачеві сконцентруватися на освоєнні базових ігрових механік. Такий підхід не лише покращує процес навчання, але й забезпечує стабільну продуктивність VR-сцени на етапі початкового знайомства з грою, що є критичним фактором для позитивного першого враження користувачів.

3.10.2 Тематичні середовища

Для створення цілісного ігрового світу, окрім тренувальної арени, у проєкті реалізовано кілька тематичних локацій, що розширюють ігровий простір та збагачують наративний потенціал гри. Серед них варто відзначити:

1. Середньовічне місто (Рис. 3.60) — деталізоване міське середовище з характерною архітектурою, вуличними елементами та атмосферою середньовічного поселення.

Рис. 3.60. Панорамний вигляд середньовічного міста

2. Село (Рис. 3.61) —локація з традиційними будівлями, що доповнює міське середовище та забезпечує контраст між різними типами поселень середньовічного світу.

Рис. 3.61. Загальний вигляд середньовічного села

Ці локації містить інтерактивні елементи, специфічні для відповідного середовища, що підвищує рівень імерсивності та реалістичності ігрового світу.

3.10.3 Інтерактивний сільськогосподарський симулятор

Для демонстрації розширених можливостей взаємодії з віртуальним середовищем розроблено спеціалізовану локацію із сільськогосподарською тематикою. Центральним елементом цієї локації є система вирощування та збору різноманітних овочевих культур, реалізована у вигляді чотирьох тематичних грядок (Рис. 3.62).

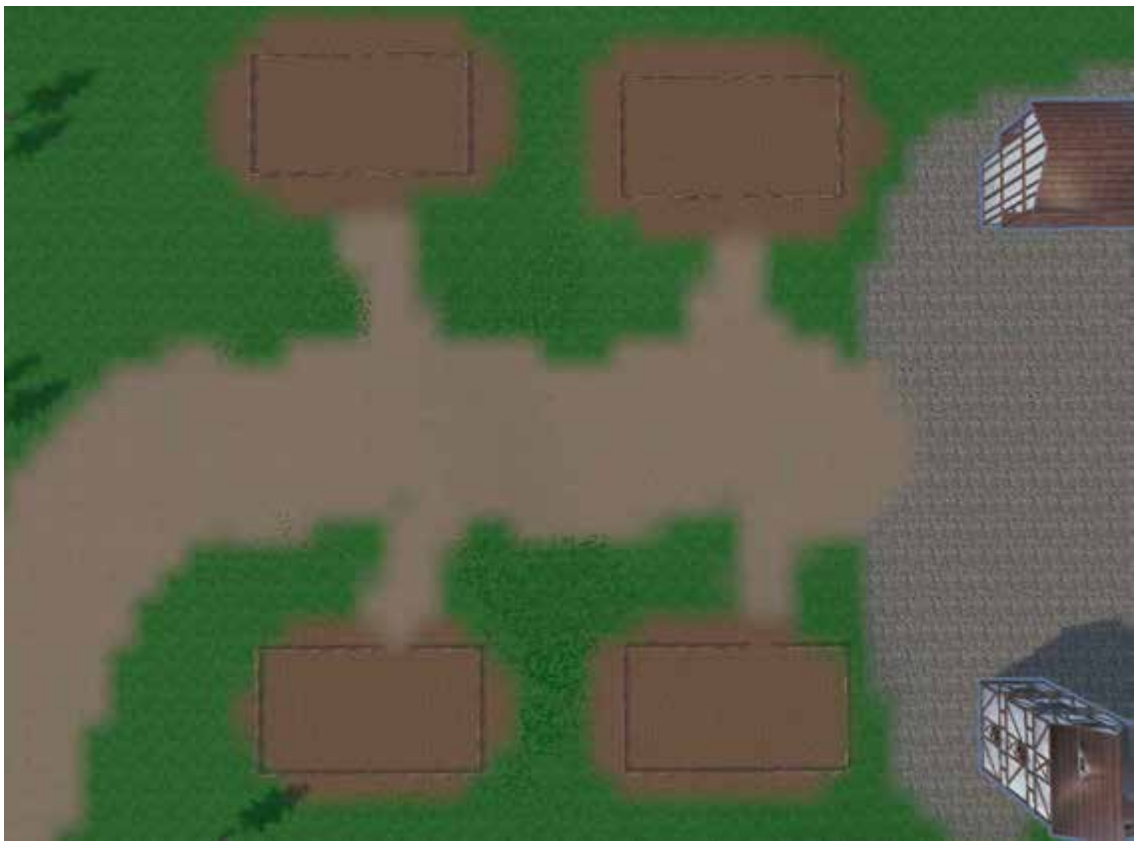


Рис. 3.62. Інтерактивні грядки з овочевими культурами

Для забезпечення ефективного та гнучкого створення грядок розроблено спеціалізований скрипт GardenSpawner, що автоматизує процес розміщення рослин у віртуальному просторі. Графічний інтерфейс налаштування скрипта представлено на Рис. 3.63.

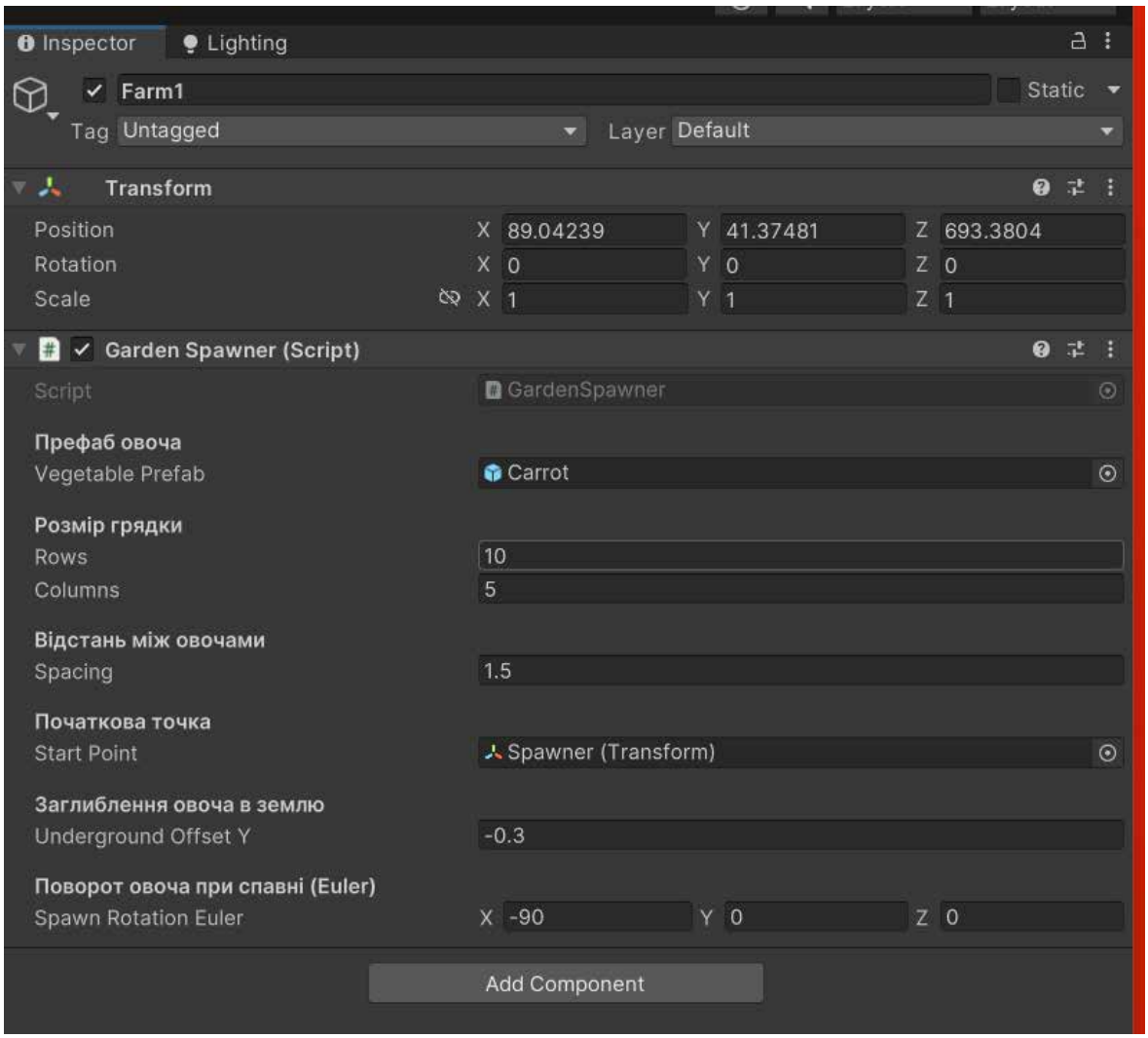


Рис. 3.63. Параметри конфігурації скрипта GardenSpawner в інспекторі Unity

Функціональність скрипта GardenSpawner

Скрипт GardenSpawner забезпечує програмне створення грядок із гнучкими параметрами налаштування:

- Формування впорядкованої сітки рослин у рядах і колонках
- Налаштування відстані між окремими рослинами
- Регулювання глибини "посадки" (заглиблення об'єктів під поверхню ґрунту)
- Налаштування випадкових кутів розміщення рослин для підвищення візуального різноманіття

Важливим елементом системи є механізм інтерактивності, що дозволяє користувачеві "збирати врожай" — взаємодіяти з рослинами, виймаючи їх із

ґрунту. Для реалізації цієї функціональності кожен об'єкт рослини налаштовується наступним чином:

Така система не лише демонструє можливості механіки взаємодії з об'єктами, але й створює основу для розширеного ігрового циклу, пов'язаного з отриманням, обробкою та використанням ресурсів у VR-середовищі.

3.10.4 Фізичні симуляції: інтерактивна рідина

Для підвищення реалістичності віртуального середовища в деяких локаціях впроваджено просунуті фізичні симуляції. Одним із найбільш технічно складних елементів є система інтерактивної рідини, реалізована у вигляді склянки з рідиною, що реагує на рухи користувача (рис. 3.64). Підхід до реалізації цієї системи базується на методиці, продемонстрованій у навчальному відео *«Liquid Physics for VR in Unity»* (автор – **Valem Tutorials**) [3].

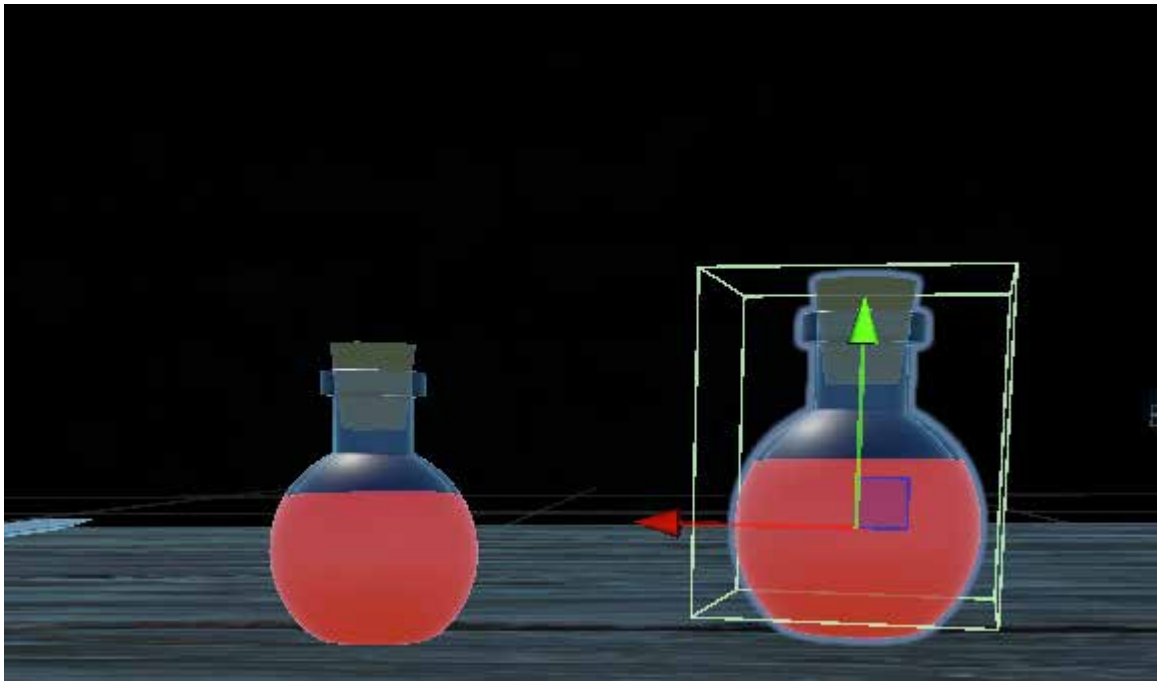


Рис. 3.64. Візуалізація інтерактивної рідини у склянці

Шейдер для симуляції рідини

Ключовою складовою системи є спеціалізований шейдер, що візуалізує поведінку рідини (Рис. 3.65, 3.65, 3.67).

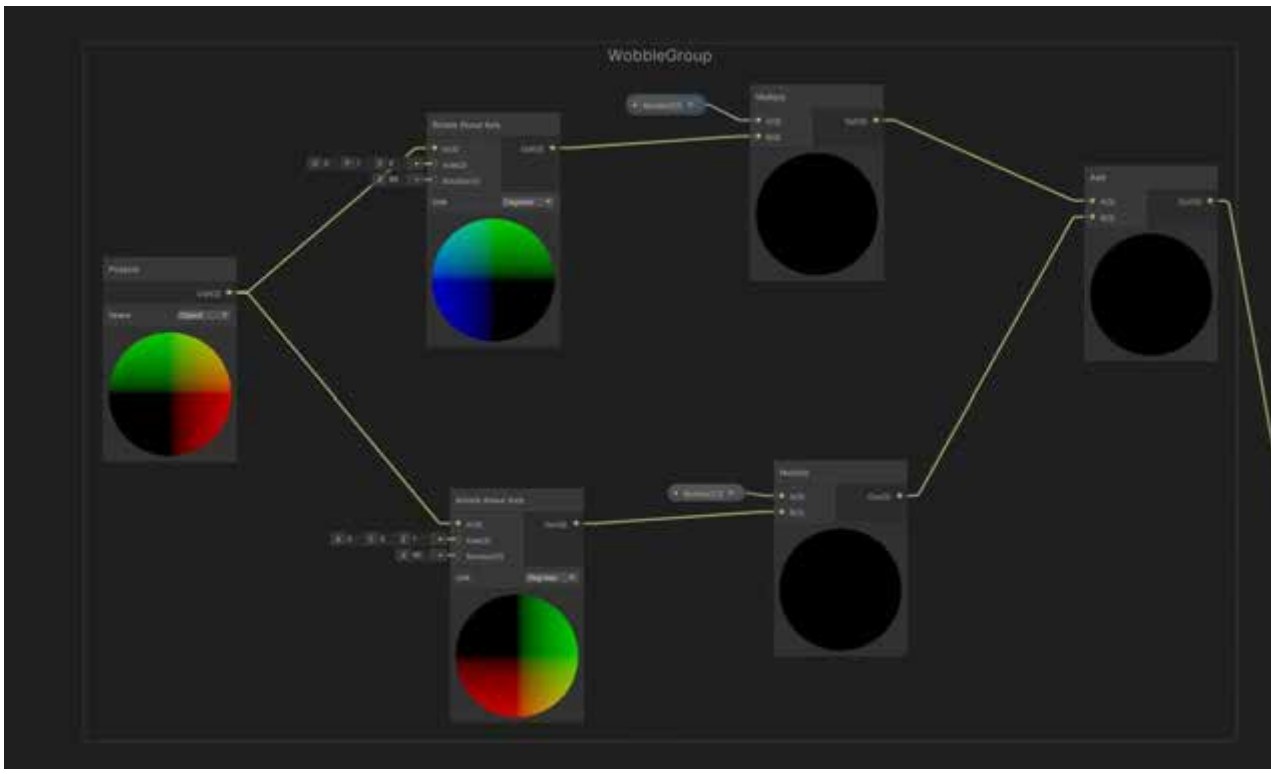


Рис. 3.65. Параметри шейдера рідини в інспекторі Unity

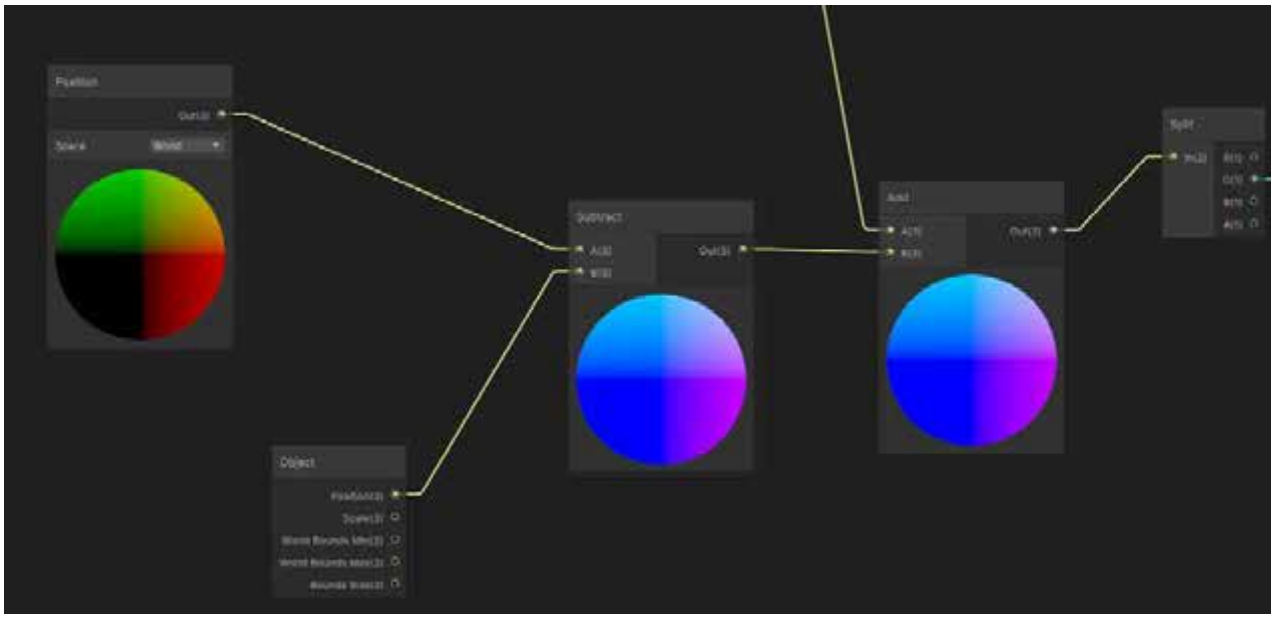


Рис. 3.66. Візуалізація роботи шейдера при різних рівнях заповнення

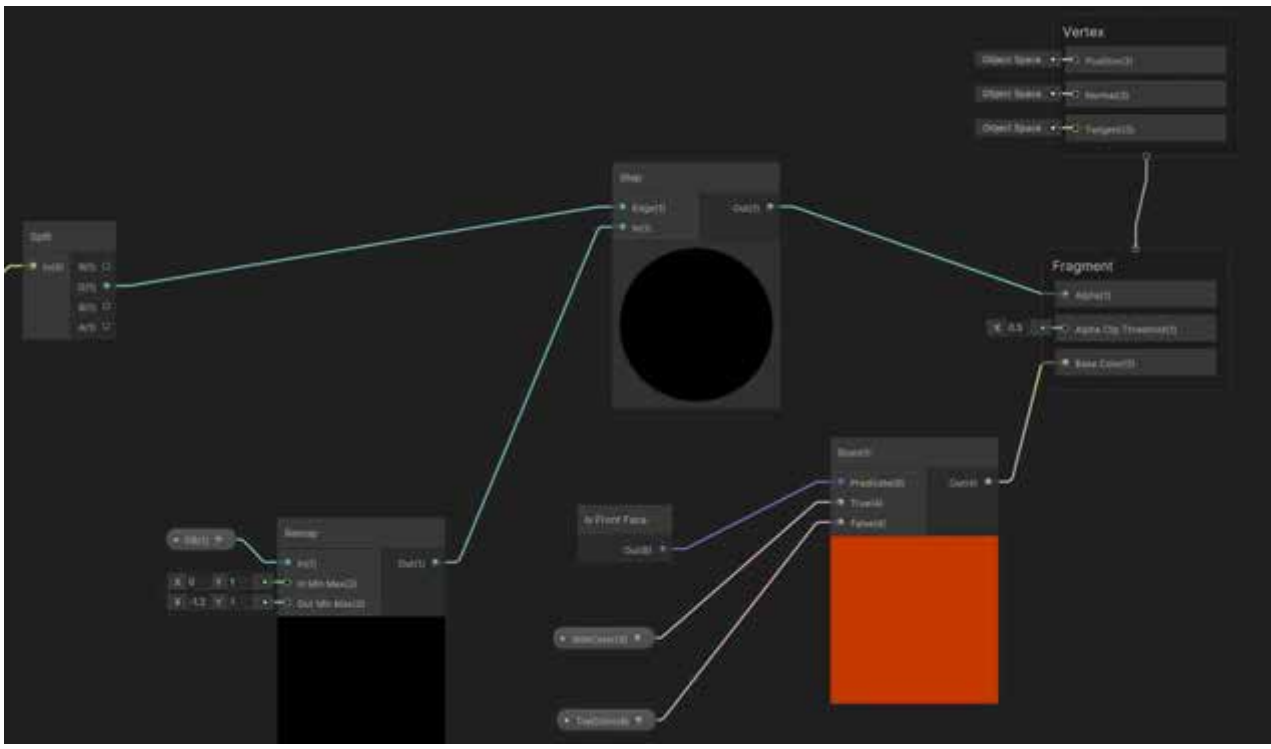


Рис. 3.67. Конфігурація матеріалу з шейдером рідини

Архітектура системи інтерактивної рідини

Симуляція рідини реалізована за допомогою композиції наступних компонентів:

1. Фізичний контейнер — модель склянки з компонентом XRGrabInteractable для забезпечення можливості захоплення та маніпуляції

2. Візуальний шар рідини — адаптований дублікат внутрішньої частини склянки з застосованим спеціалізованим шейдером

3. Скрипт Wobble — компонент, що відповідає за розрахунок параметрів коливання поверхні рідини залежно від руху контейнера

Параметри налаштування скрипта Wobble доступно в інспекторі Unity (Рис. 3.68), що дозволяє гнучко підлаштовувати поведінку рідини під різні типи контейнерів та різні властивості симульованих рідин.

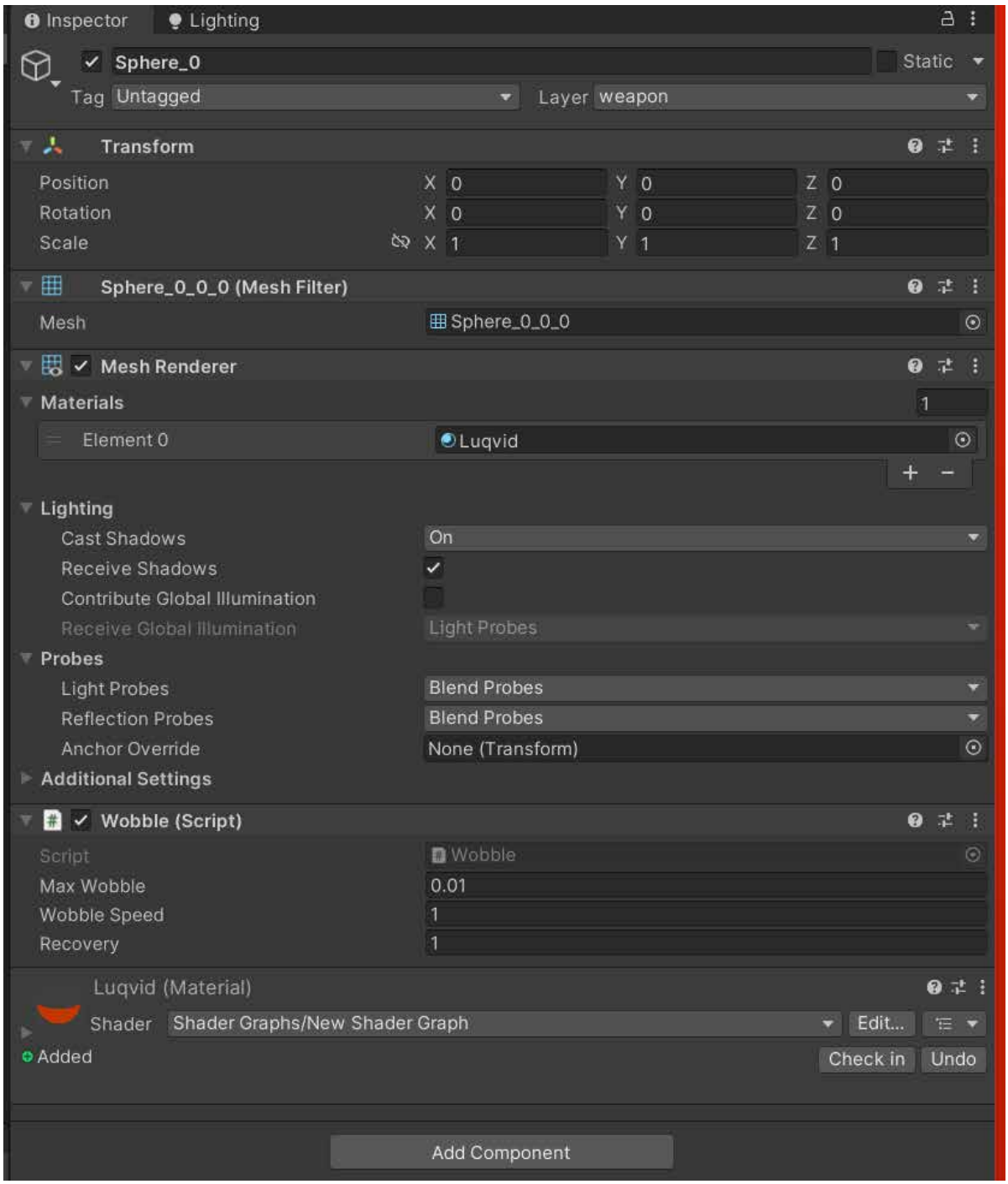


Рис. 3.68. Параметри конфігурації компонента Wobble

Основна логіка обчислення коливань поверхні рідини реалізована в методі Update() скрипта Wobble:

C#

```
void Update()
{
// Обчислення зміщення відносно попередньої позиції/обертання
Vector3 angleVelocity = transform.rotation.eulerAngles – lastRot;
```

```
Vector3 velocity = transform.position – lastPos;

// Застосування параметрів коливання до шейдера
wobbleAmountX = Mathf.Lerp(wobbleAmountX,
angleVelocity.z * wobbleMultiplier, Time.deltaTime * recovery);
wobbleAmountZ = Mathf.Lerp(wobbleAmountZ,
angleVelocity.x * wobbleMultiplier, Time.deltaTime * recovery);

// Оновлення параметрів шейдера
renderer.material.SetFloat(«_WobbleX», wobbleAmountX);
renderer.material.SetFloat(«_WobbleZ», wobbleAmountZ);

// Збереження попередніх значень для наступного кадру
lastPos = transform.position;
lastRot = transform.rotation.eulerAngles;
}
```

Для зручності експериментування з параметрами симуляції в режимі реального часу, реалізовано спеціалізований інтерфейс налаштування (Рис. 3.69), доступний безпосередньо в Інспекторі.

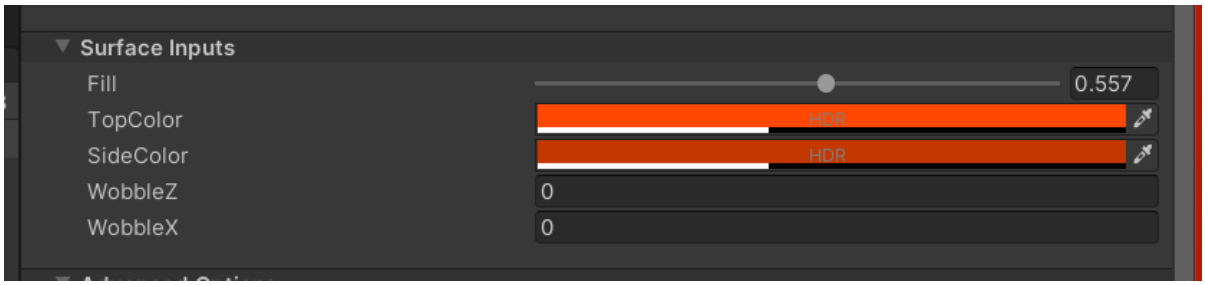


Рис. 3.69. Інтерфейс налаштування параметрів рідини в VR-середовищі

3.10.5 Інтерактивні об’єкти для розрізання

Важливим елементом інтерактивності віртуального середовища є система розрізання об’єктів, що дозволяє взаємодіяти з віртуальними предметами за допомогою холодної зброї. Об’єкти, призначені для розрізання (ящики, фрукти

та інші предмети), належать до спеціального шару SliceLayer, з яким взаємодіє система виявлення колізій для холодної зброї (Рис. 3.70).

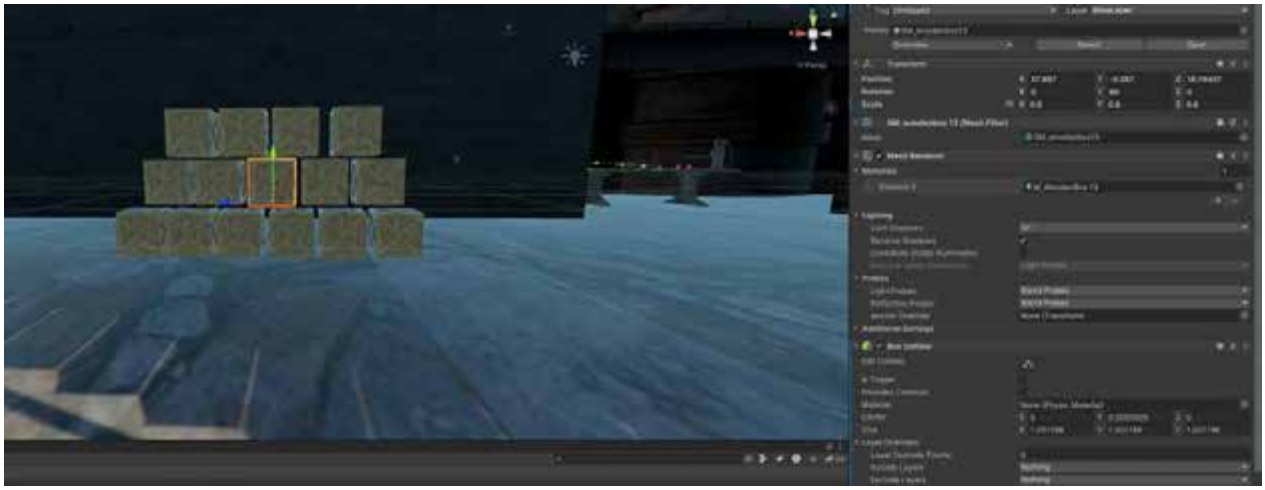


Рис. 3.70. Об’єкти, призначені для розрізання у віртуальному середовищі

Технічна реалізація об’єктів для розрізання

Об’єкти, придатні для розрізання, характеризуються наступними компонентами:

- MeshFilter та MeshRenderer для візуального представлення об’єкта
- BoxCollider або MeshCollider, що визначає межі об’єкта для фізичних взаємодій
- Належність до шару SliceLayer, що дозволяє системі розпізнавати їх як об’єкти, придатні для розрізання

Наявність таких об’єктів у різних локаціях дозволяє користувачеві експериментувати з механікою розрізання в різних умовах та середовищах, забезпечуючи як тренування навичок поводження зі зброєю, так і підвищуючи загальний рівень інтерактивності віртуального світу.

3.10.6 Система анімації водного транспорту

Для підвищення реалістичності водного середовища у віртуальному світі було розроблено спеціалізовану систему анімації лодок, що імітує природне похитування на воді. На пірсі розміщено три моделі водного транспорту, кожна з яких оснащена індивідуальною системою анімації коливальних рухів (Рис. 3.71).

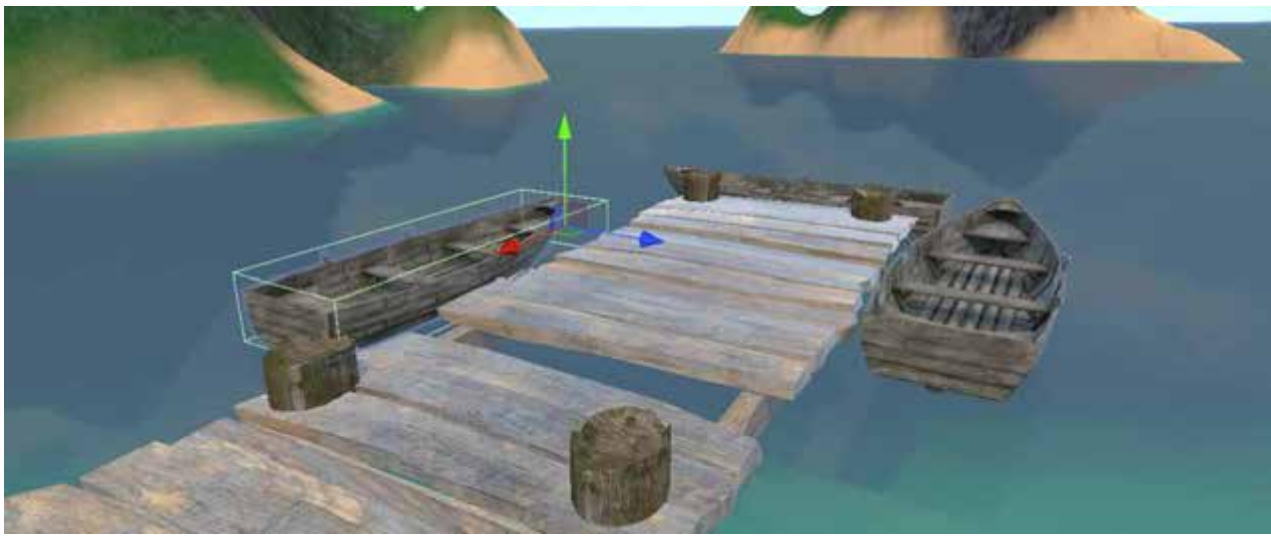


Рис. 3.71. Розміщення водного транспорту на пірсі

Технічна реалізація системи похитування

Для створення реалістичного ефекту похитування лодок на воді розроблено спеціалізований компонент TrayWobble, що забезпечує програмне управління коливальними рухами об'єктів. Система базується на математичних функціях синуса для створення плавних та природних рухів.

Основними параметрами налаштування системи є (Рис. 3.72):

- **Амплітуда коливання** (angleAmplitude) — максимальний кут відхилення об'єкта від початкового положення
- **Швидкість коливання** (wobbleSpeed) — частота коливальних рухів
- **Вісь обертання** (rotationAxis) — напрямок, вздовж якого відбувається похитування

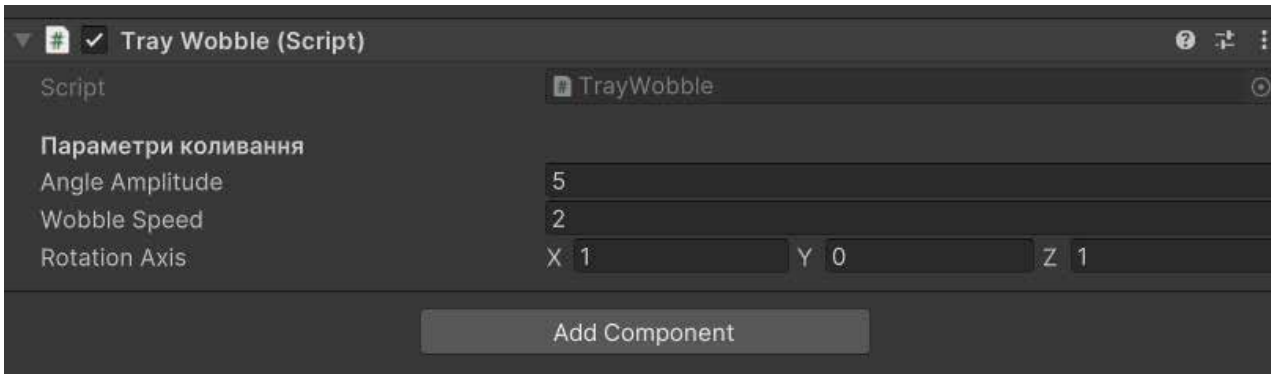


Рис. 3.72. Параметри конфігурації компонента анімації похитування

Принцип роботи системи

Алгоритм роботи системи базується на обчисленні кутового зміщення об'єкта відносно початкової позиції з використанням тригонометричних функцій. В кожному кадрі відтворення система розраховує поточний кут відхилення на основі часового параметра та заданих характеристик коливання.

Для досягнення максимальної реалістичності ефекту кожна лодка налаштована з індивідуальними параметрами коливання по декількох осях одночасно, що імітує складний характер природного похитування на водній поверхні під впливом хвиль та течії.

3.10.7 Модульна структура локацій

Всі розроблені локації об’єднані в єдину систему з використанням модульного підходу (Рис. 3.73), що забезпечує значні переваги з точки зору розробки, оптимізації та масштабованості проєкту.

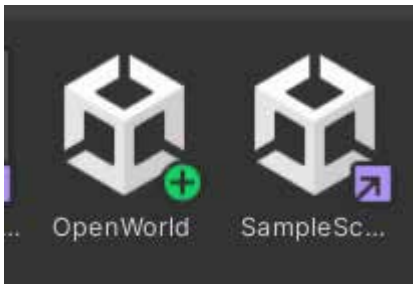


Рис. 3.73. Схема модульної організації локацій у проєкті

Переваги модульного підходу

Обраний підхід до організації локацій надає наступні можливості:

1. Динамічне управління ресурсами — локації можуть завантажуватися та вивантажуватися за потреби через систему управління сценами Unity (Scene Management) або через активацію/деактивацію модулів всередині однієї сцени

2. Оптимізація використання обчислювальних ресурсів — віддалені об’єкти автоматично спрощуються (зменшення кількості полігонів, спрощення колайдерів) за допомогою системи рівнів деталізації (LOD)

3. Плавні переходи між локаціями — реалізовано систему порталів, що забезпечує безшовні переходи між різними частинами віртуального світу

Модульний підхід до організації локацій значно підвищує масштабованість проєкту, дозволяючи поступово розширювати ігровий світ без суттєвого впливу на продуктивність системи. Він також забезпечує ефективне управління пам’яттю, що є критичним фактором для VR-застосунків, особливо тих, що працюють на автономних пристроях з обмеженими обчислювальними ресурсами.

3.10.8 Система навігації між локаціями

Для забезпечення плавного переміщення користувача між різними локаціями віртуального світу розроблено спеціалізовану систему телепортації, що базується на принципі тригерних зон. Ця система дозволяє користувачеві інтуїтивно переходити між різними сценами проєкту шляхом наближення до спеціально визначених точок переходу.

Рис. 3.74. Візуалізація зони телепортації між локаціями

Конфігурація системи управління сценами

Для коректного функціонування системи переходів необхідно налаштувати ієрархію сцен у параметрах збірки проєкту. Кожній сцені присвоюється унікальний числовий індекс, що використовується системою управління сценами Unity для ідентифікації цільової локації (Рис. 3.75).

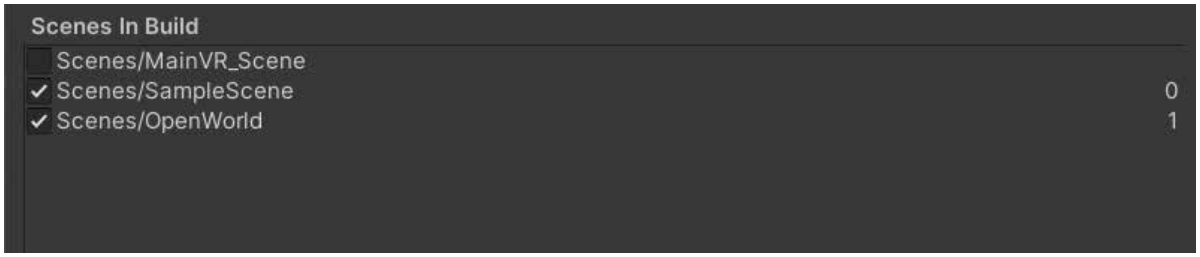


Рис. 3.75. Конфігурація індексів сцен у налаштуваннях збірки

Архітектура компонента переходу

Система переходу між локаціями реалізована за допомогою спеціалізованого компонента NextScene, що прикріплюється до об'єктів-телепортів у віртуальному просторі. Компонент забезпечує гнучке налаштування параметрів переходу через інспектор Unity (Рис. 3.76).

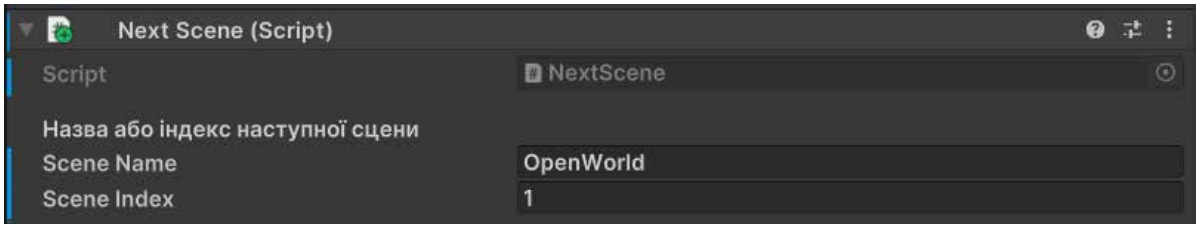


Рис. 3.76. Параметри конфігурації компонента переходу між сценами

Принцип роботи системи телепортації

Механізм переходу активується при перетині користувачем (об'єктом з тегом "player") зони тригера телепорта. Система підтримує два способи ідентифікації цільової сцени:

1. **Ідентифікація за назвою сцени** — використання текстового ідентифікатора сцени для більш інтуїтивного налаштування
2. **Ідентифікація за індексом** — використання числового індексу сцени для більш ефективного управління у випадку великої кількості локацій

При активації тригера система автоматично ініціює процес завантаження цільової сцени через вбудований менеджер сцен Unity, забезпечуючи плавний перехід між локаціями без розриву ігрового процесу.

Така архітектура системи навігації забезпечує масштабованість проєкту та дозволяє легко додавати нові локації без необхідності модифікації існуючого коду.

3.11 Реалізація системи інвентаря у VR середовищі

У даному підрозділі розглянуто процес проектування та реалізації системи інвентаря для віртуальної реальності, що є ключовим елементом розроблюваної RPG гри. Система інвентаря забезпечує зберігання, групування та використання внутрішньоігрових предметів, що суттєво впливає на механіки геймплею та загальний користувацький досвід.

За основу було взято загальну концепцію інвентаря, представлену у навчальному відео *«Unity VR Development for Oculus Quest: Inventory»* (автор – **RealaryVR**) [4]. Однак реалізація в даному проєкті є **авторською**, оскільки оригінальний підхід орієнтовано на іншу платформу з іншим SDK. Під час розробки було здійснено адаптацію до обраного середовища, змінено архітектуру взаємодії з об’єктами та вдосконалено логіку управління інвентарем. У результаті реалізовано власну модифіковану систему, яка враховує специфіку обраного VR-шолома та потреби гри.

3.11.1. Проектування користувацького інтерфейсу інвентаря

Проектування користувацького інтерфейсу інвентаря розпочалося з розробки візуального макета у програмному середовищі Figma. Цей етап дозволив визначити основні елементи інтерфейсу та їх розташування, перш ніж переходити до безпосередньої імплементації в Unity.

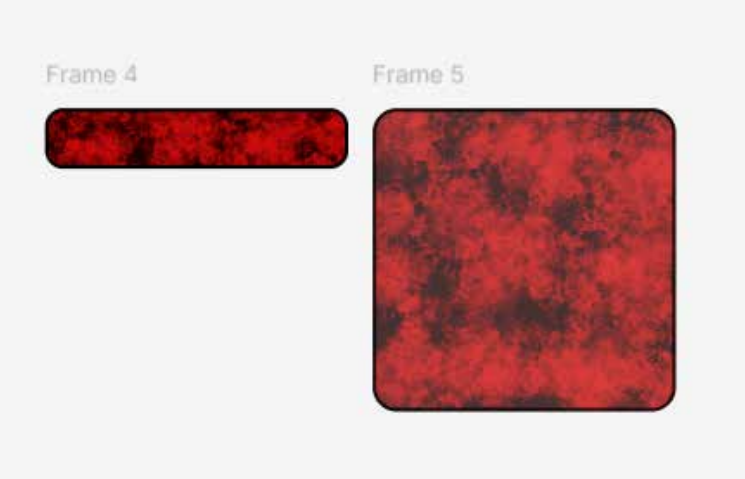


Рисунок 3.77 - Макет інтерфейсу інвентаря, розроблений у Figma

Після завершення розробки макету, елементи інтерфейсу були експортовані для подальшого використання в ігровому рушії Unity.

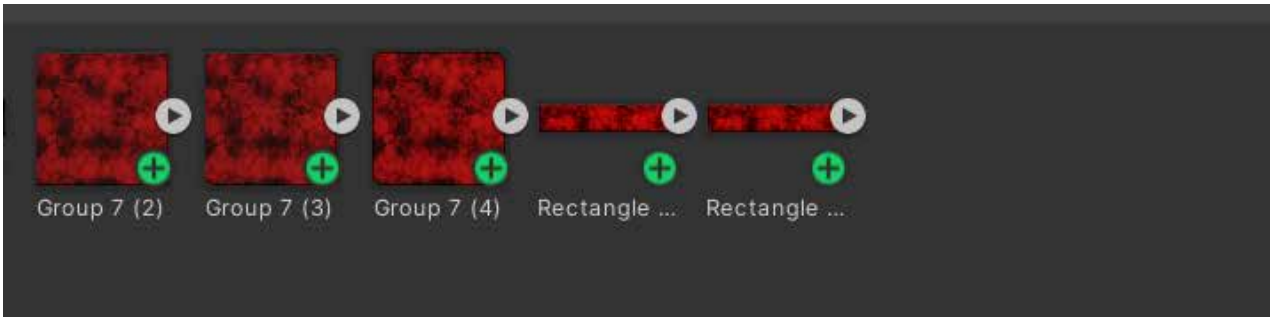


Рисунок 3.78 - Експортовані елементи інтерфейсу в Unity

Ключову увагу в процесі проектування було приділено таким аспектам:

- Ергономічність взаємодії в VR середовищі
- Інтуїтивно зрозуміле представлення вмісту інвентаря
- Візуальна індикація стану слотів (зайнятий/вільний)
- Візуальне відображення кількості згрупованих предметів

Після завершення проектування макету в Figma, текстури та елементи інтерфейсу були експортовані у формат, сумісний з Unity, і далі інтегровані в проект.

3.11.2. Структура системи інвентаря

Розроблена система інвентаря складається з наступних ключових компонентів:

1. Панель інвентаря - основний контейнер, що містить слоти

2. Слоти - окремі комірки для зберігання предметів

3. Система відстеження предметів - механізм, що дозволяє визначати, які предмети знаходяться у зоні взаємодії зі слотами

4. Механізм активації інвентаря - система, що відповідає за появу та зникнення інтерфейсу інвентаря через визначені користувацькі дії

У реалізованій системі інвентар складається з чотирьох слотів. Кожен слот функціонує як окремий контейнер, здатний зберігати предмети та групувати подібні об'єкти.

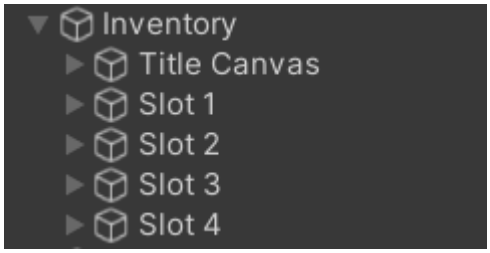


Рисунок 3.79 - Структура інвентаря з чотирма слотами

Структура слоту інвентаря

Слот інвентаря включає наступні компоненти:

- Canvas (полотно) - основний компонент для відображення UI елементів
- Image (зображення) - візуальне представлення слоту
- TextMeshPro Text - текстовий елемент для відображення кількості предметів у слоті

Рисунок 3.80 - Детальна структура слоту інвентаря

C#

```
[Header("UI References")]
public Image slotImage;
public TMP_Text countText;
private Color originalColor;
```

private HashSet<Item> blockedItems = new HashSet<Item>();

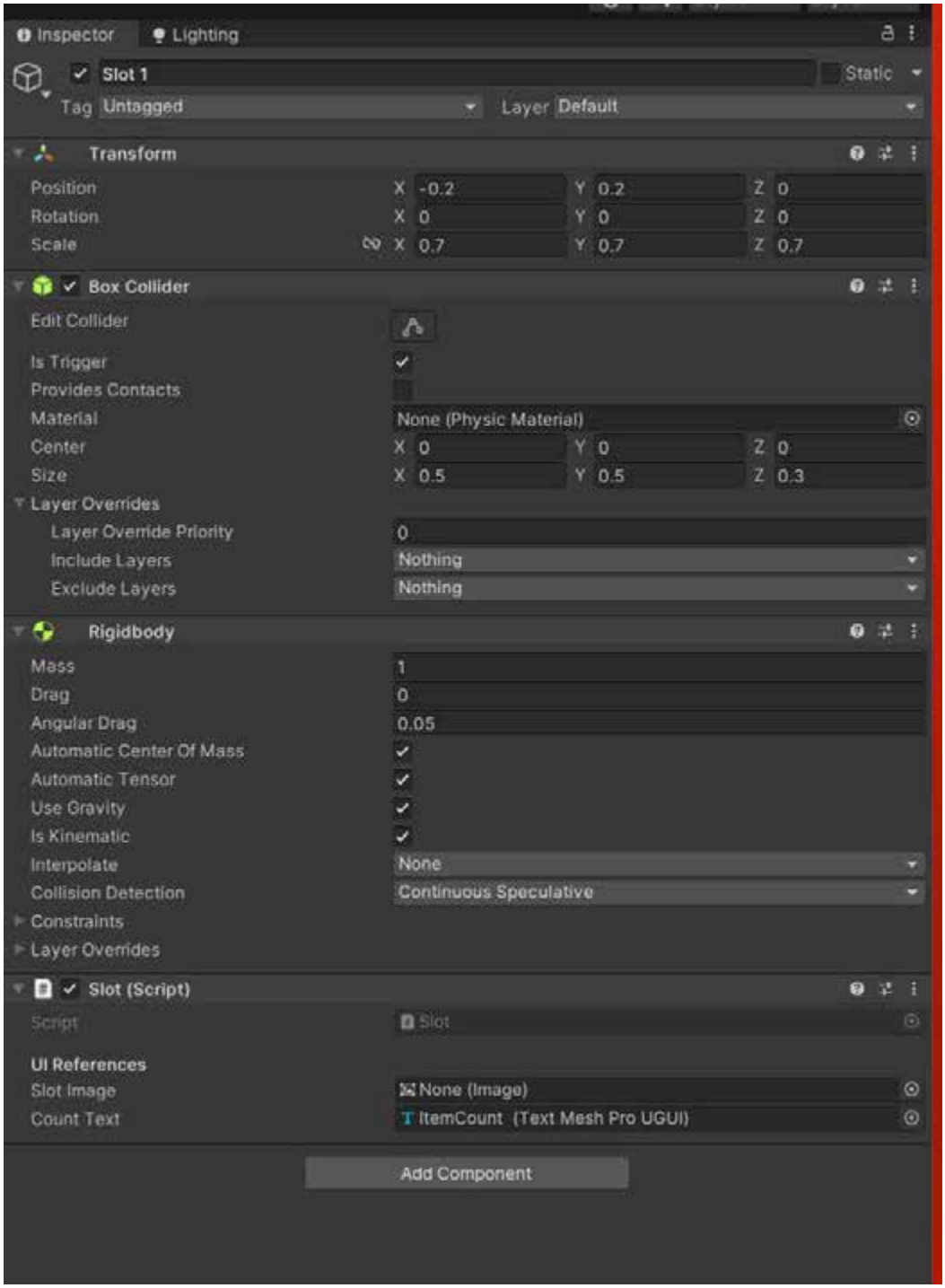


Рисунок 3.81 - Компоненти скрипта слота в інспекторі

Слоти виконують кілька важливих функцій:

- Визначення можливості розміщення предмета в слоті
- Візуальне відображення стану (зайнятий/вільний)
- Підрахунок і відображення кількості предметів одного типу

- Управління фізичними властивостями предметів при розміщенні в слоті

3.11.3. Алгоритм управління предметами

Розроблений алгоритм управління предметами базується на принципі автоматичного розпізнавання типу об'єкта та його відповідності умовам розміщення у конкретному слоті. Система здійснює валідацію можливості розміщення предмета, враховуючи його тип та поточний вміст слота.

Ключовою особливістю алгоритму є інтелектуальне управління фізичними властивостями предметів при їх розміщенні в інвентарі. При потраплянні до слота предмет автоматично втрачає фізичну активність та фіксується у визначеній позиції, що забезпечує стабільність системи та запобігає небажаним фізичним взаємодіям.

Процес вилучення предмета з інвентаря супроводжується відновленням його фізичних властивостей та від'єднанням від системи слотів. Це дозволяє користувачу знову взаємодіяти з предметом у віртуальному просторі природним чином.

3.11.4. Система групування предметів

Реалізована система групування (стекування) однотипних предметів оптимізує використання обмеженого простору інвентаря. Алгоритм групування працює на основі порівняння типів предметів, визначених через відповідні властивості компонента Item.

Принцип роботи системи полягає у автоматичному розпізнаванні предметів одного типу та їх об'єднанні в межах одного слота. При цьому візуально відображається тільки один екземпляр предмета з числовим індикатором кількості. Така організація дозволяє ефективно використовувати обмежену кількість слотів інвентаря.

Для предметів, які не підлягають групуванню (унікальні або спеціальні предмети), система автоматично блокує можливість стекування, забезпечуючи індивідуальне зберігання кожного екземпляра.

3.11.5. Інтеграція інвентаря з VR контролерами

Інтеграція системи інвентаря з VR контролерами реалізована через спеціалізований модуль, що забезпечує інтуїтивну взаємодію користувача з інтерфейсом. Активація інвентаря здійснюється через призначену кнопку X на контролері, що відповідає загальноприйнятим стандартам VR інтерфейсів.

Рисунок 3.82 - Схематика VR контролера з кнопкою активації інвентаря

Панель інвентаря динамічно позиціонується відносно лівої руки користувача, забезпечуючи оптимальний кут огляду та зручність взаємодії. Система автоматично коригує положення панелі залежно від рухів користувача, підтримуючи стабільну видимість інтерфейсу.

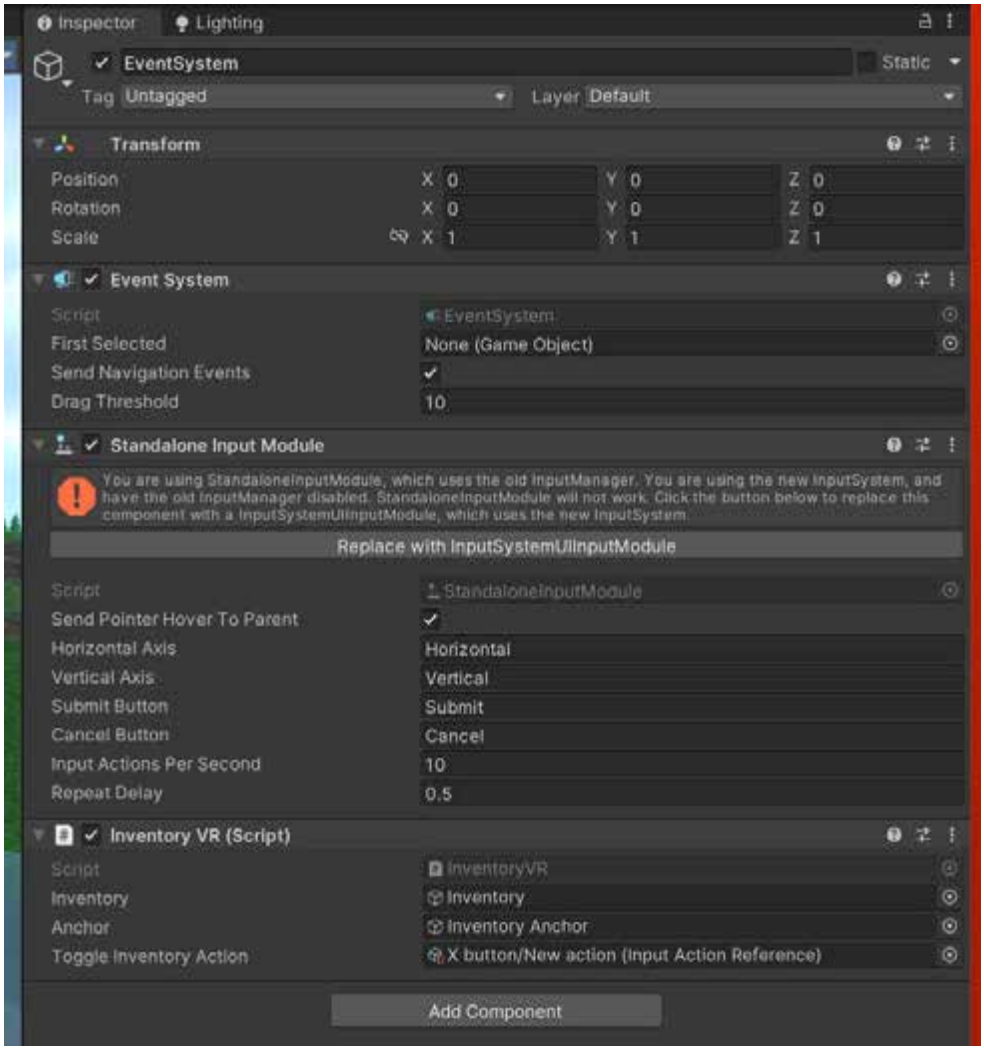


Рисунок 3.83 - Налаштування реагування на кнопку контролера

Особливістю реалізації є використання системи прив'язки (anchor system), яка забезпечує стабільне позиціонування інтерфейсу відносно руки користувача навіть при активних рухах. Це критично важливо для забезпечення комфортного користування інвентарем під час динамічного геймплею.

3.11.6. Система управління станами предметів

Розроблений компонент Item служить основою для управління станами всіх предметів, що можуть взаємодіяти з системою інвентаря. Компонент відстежує поточний стан предмета, його розташування та приналежність до конкретного слота.

Ключовою функціональністю компонента є обробка подій захоплення та відпускання предмета користувачем. При взаємодії з предметом система

автоматично оновлює його стан, забезпечуючи коректну роботу всіх пов'язаних механізмів.

Основні властивості системи включають:

C#

public string itemType;

[HideInInspector] public bool inSlot = false;

[HideInInspector] public Slot currentSlot;

public Vector3 slotRotation = Vector3.zero;

Така структура дозволяє системі ефективно відстежувати стан кожного предмета та забезпечувати коректну взаємодію між різними компонентами системи інвентаря.

3.11.7. Система групування предметів

У розробленій системі інвентаря реалізовано функціональність групування (стекування) однотипних предметів для оптимізації використання слотів. Алгоритм групування базується на перевірці типу предмета через властивість itemType.

Принцип роботи системи групування:

1. Предмети без визначеного значення itemType не підлягають групуванню
2. Предмети з однаковим значенням itemType можуть бути згруповані в одному слоті
3. При групуванні предметів відображається числовий індикатор кількості

C#

private void OnTriggerStay(Collider other)

{

var obj = other.gameObject;

var item = obj.GetComponent<Item>();

```
if (item == null || item.inSlot) return;

// Якщо itemType порожній => не стекати більше одного
bool isNonStackable = string.IsNullOrEmpty(item.itemType);

// Якщо вже є предмети в слоті
if (itemsInSlot.Count > 0)
{
var firstItem = itemsInSlot[0].GetComponent<Item>();
bool firstNonStackable = string.IsNullOrEmpty(firstItem.itemType);
if (firstNonStackable || isNonStackable || item.itemType != firstItem.itemType)
return;
}

if (blockedItems.Contains(item)) return;

var grab = obj.GetComponent<XRGrabInteractable>();
if (grab.isSelected) return;

InsertItem(obj);
}
```

Для оновлення відображення кількості предметів у слоті використовується метод:

```
C#
private void UpdateCountUI()
{
if (itemsInSlot.Count <= 1)
countText.text = string.Empty;
else
countText.text = itemsInSlot.Count.ToString();
```

}

3.11.8. Результати реалізації та тестування

Розроблена система інвентаря успішно інтегрована у віртуальне середовище та демонструє ефективну роботу при взаємодії з різними типами предметів. Під час тестування підтверджено коректну роботу наступних аспектів:

1. Активація та деактивація інтерфейсу інвентаря
2. Розміщення предметів у слотах
3. Групування однотипних предметів
4. Вилучення предметів зі слотів
5. Відображення кількості згрупованих предметів

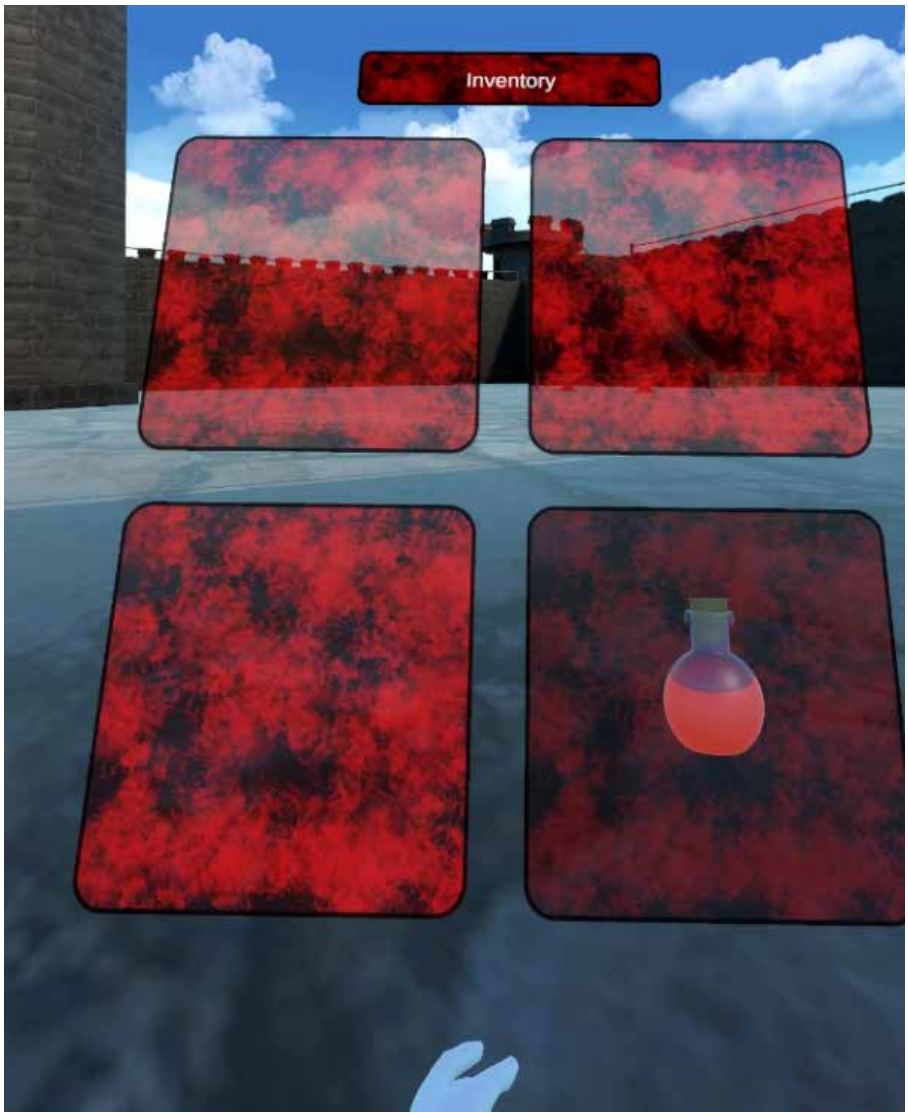


Рисунок 3.84 - Інвентар з розміщеним предметом

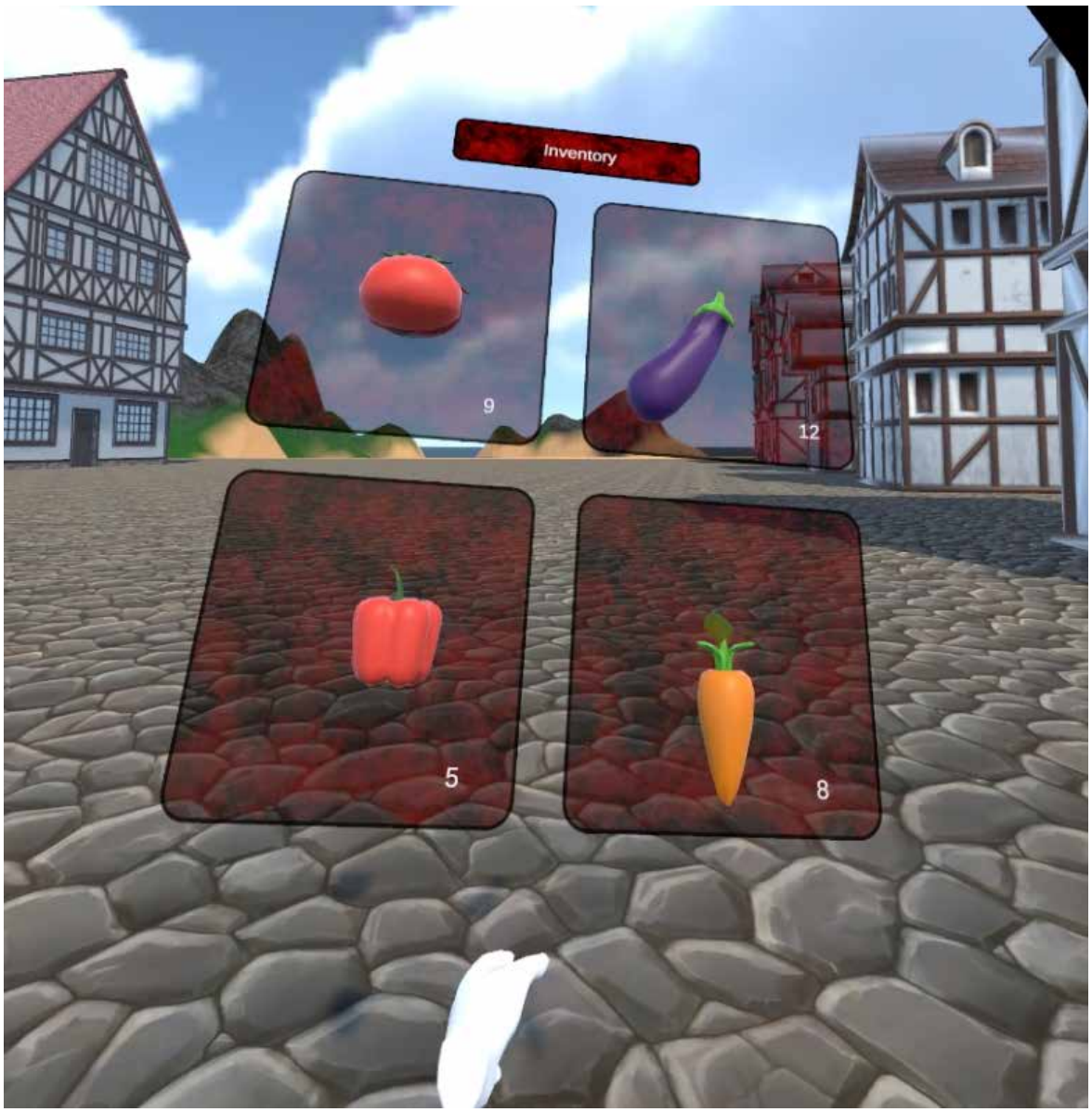


Рисунок 3.85 - Приклад групування однотипних предметів (овочів) в інвентарі

Система демонструє стабільну роботу в різних сценаріях використання та забезпечує інтуїтивно зрозумілий механізм взаємодії для користувача.

3.11.9. Висновки

Розроблена система інвентаря для VR RPG гри забезпечує ефективний механізм управління ігровими предметами з урахуванням особливостей взаємодії у віртуальній реальності. Ключовими перевагами реалізованої системи є:

1. Інтуїтивно зрозумілий інтерфейс, адаптований для VR середовища

2. Ефективне використання ігрового простору завдяки механізму групування предметів

3. Гнучкість у налаштуванні взаємодії для різних типів предметів

4. Оптимізоване позиціонування відносно користувача для забезпечення зручного доступу

Подальший розвиток системи може включати розширення функціональності через додавання категорій предметів, реалізацію системи швидкого доступу до часто використовуваних предметів та оптимізацію процесу взаємодії для підвищення комфорту користувача.

3.12. Реалізація системи Body Inventory для VR середовища

У даному підрозділі розглянуто процес проектування та реалізації системи Body Inventory (інвентаря на тілі персонажа) для віртуальної реальності, що є додатковим елементом розроблюваної RPG гри. Система Body Inventory забезпечує більш іммерсивний досвід взаємодії з предметами у грі, дозволяючи гравцю розміщувати та отримувати предмети безпосередньо з віртуальних "слотів" на тілі персонажа.

3.12.1. Концепція та особливості Body Inventory

На відміну від класичного інвентаря, описаного в підрозділі 3.11, система Body Inventory створює віртуальні слоти, розташовані навколо тіла гравця у VR середовищі, що забезпечує більш природну та інтуїтивну взаємодію з ігровими предметами. Основні переваги такого підходу:

- Підвищення рівня іммерсивності за рахунок природних рухів для доступу до предметів
- Швидкий доступ до часто використовуваних предметів без виклику меню інвентаря
- Реалістичне відчуття носіння спорядження на тілі персонажа

Body Inventory поділено на кілька спеціалізованих зон, кожна з яких призначена для зберігання певного типу предметів:

1. Пояс (сагайдак) - для зберігання стріл
2. Спина - для зберігання зброї (мечів, сокир)

Рисунок 3.86 - Структура Body Inventory з компонентами для поясу та спини

3.12.2. Архітектура системи Body Inventory

Система Body Inventory базується на концепції "body sockets" (слотів на тілі), які динамічно позиціонуються відносно положення VR шолома (HMD - Head-Mounted Display) користувача. Кожен слот має визначене співвідношення висоти відносно положення HMD, що дозволяє системі адаптуватися під зріст та положення користувача.

Структура системи включає:

1. Клас BodySocketInventory - основний компонент, що керує системою інвентаря на тілі
2. Клас bodySocket - структура даних для опису окремого слота на тілі
3. Механізм відстеження положення HMD для коректного позиціонування слотів

3.12.3. Реалізація сагайдаку на поясі

Одним із ключових елементів Body Inventory є сагайдак на поясі користувача, реалізований у вигляді сферичного колайдера, що відстежує положення камери та розташовується на віртуальному поясі гравця. Цей компонент призначений для зберігання стріл та інших дрібних предметів, що мають бути легкодоступними під час гри.

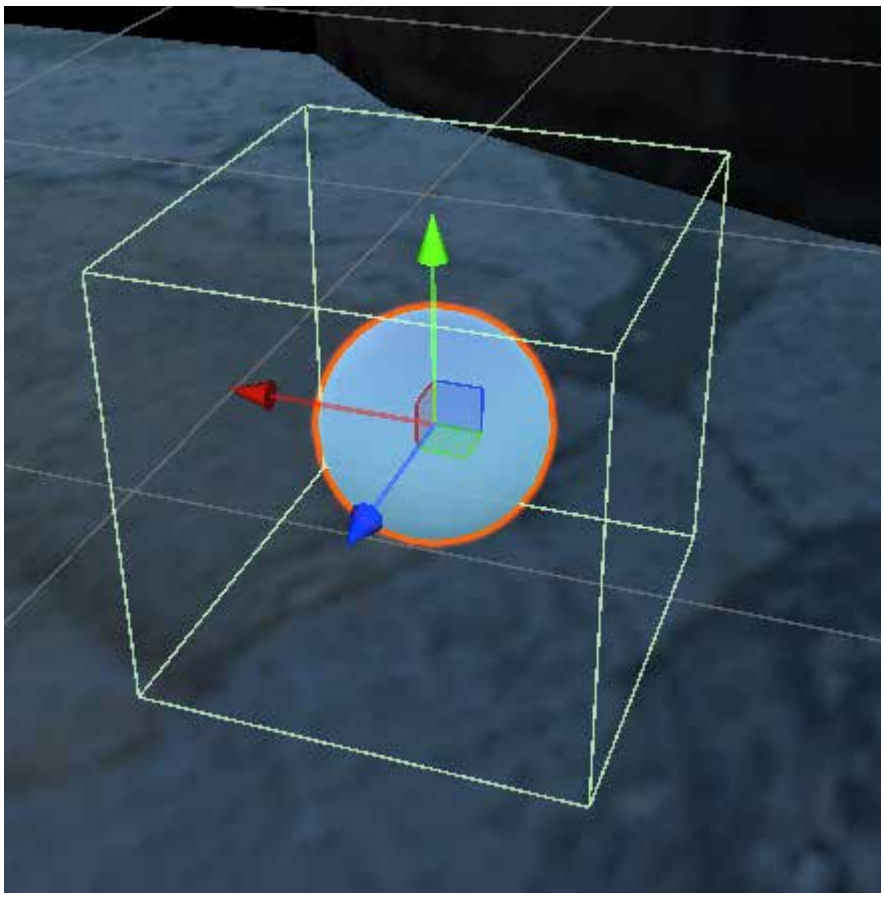


Рисунок 3.87 - Сферичний колайдер для реалізації сагайдаку на поясі

Налаштування сагайдаку в інспекторі Unity включає наступні параметри:

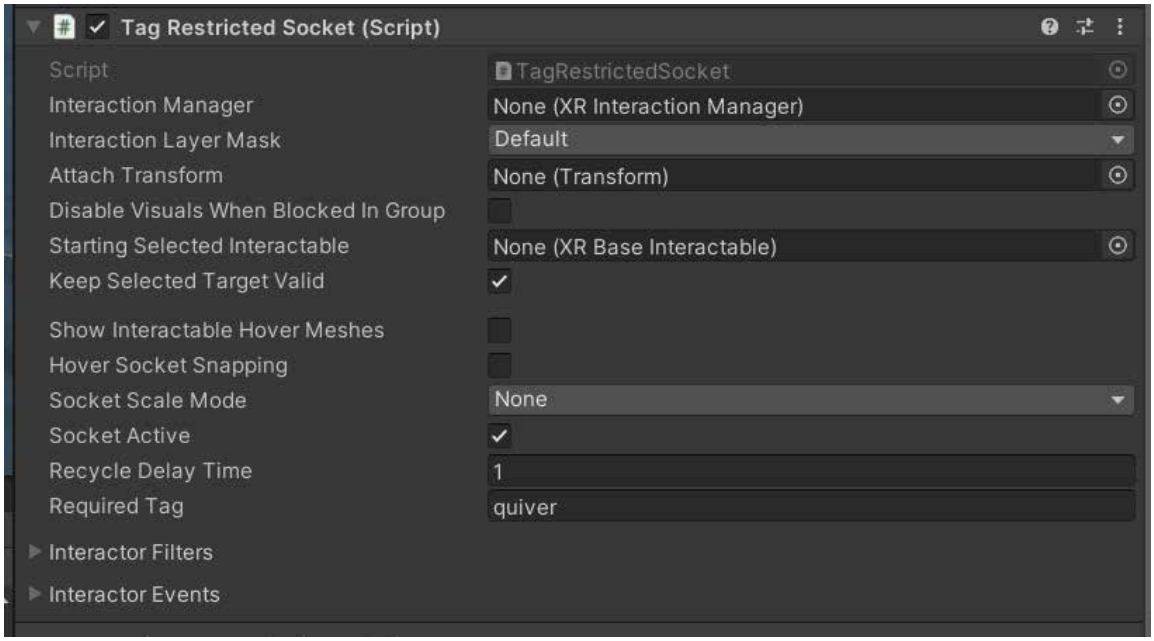


Рисунок 3.88 - Налаштування місця для сагайдаку в інспекторі Unity

Основні властивості сагайдаку:

- Динамічне позиціонування відносно висоти користувача
- Автоматичне обертання разом із поворотом користувача
- Можливість вилучення предметів природним рухом руки до поясу

Рисунок 3.89 – Вигляд реалізації у грі

3.12.4. Реалізація слота для зброї на спині

Для розміщення зброї на спині персонажа реалізовано спеціалізований слот у вигляді куба з колайдером. Цей слот розташовується за спиною користувача та забезпечує зручне зберігання великої зброї.

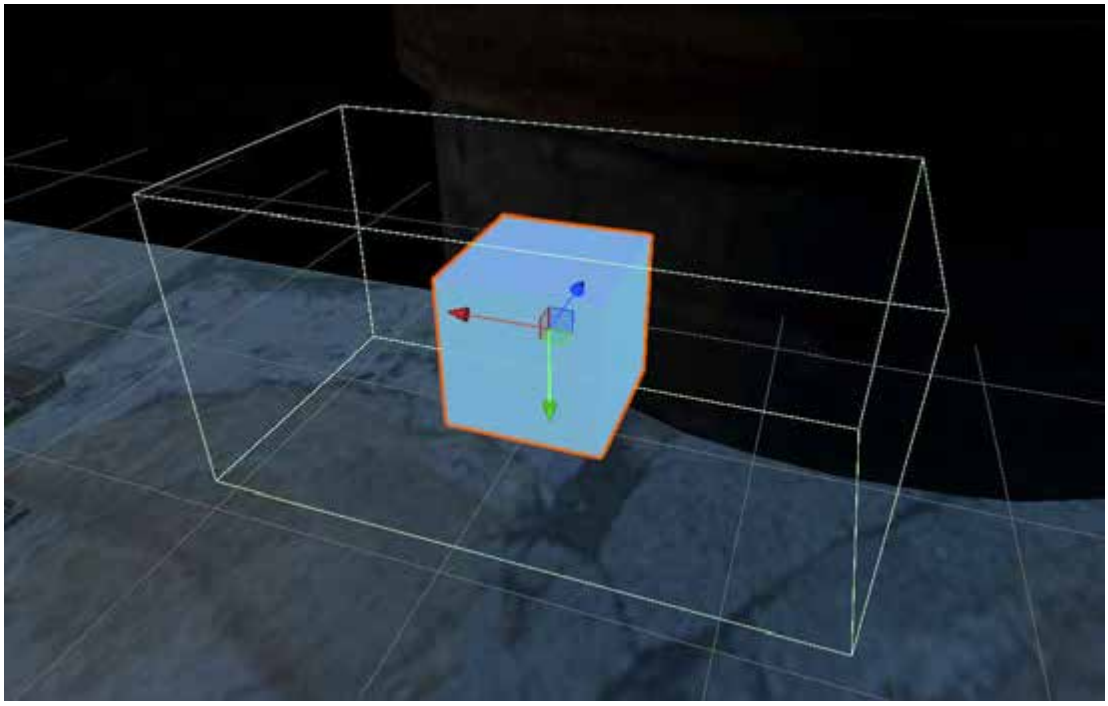


Рисунок 3.90 - Загальний вигляд куба для зберігання зброї за спиною

Налаштування слота для зброї в інспекторі:

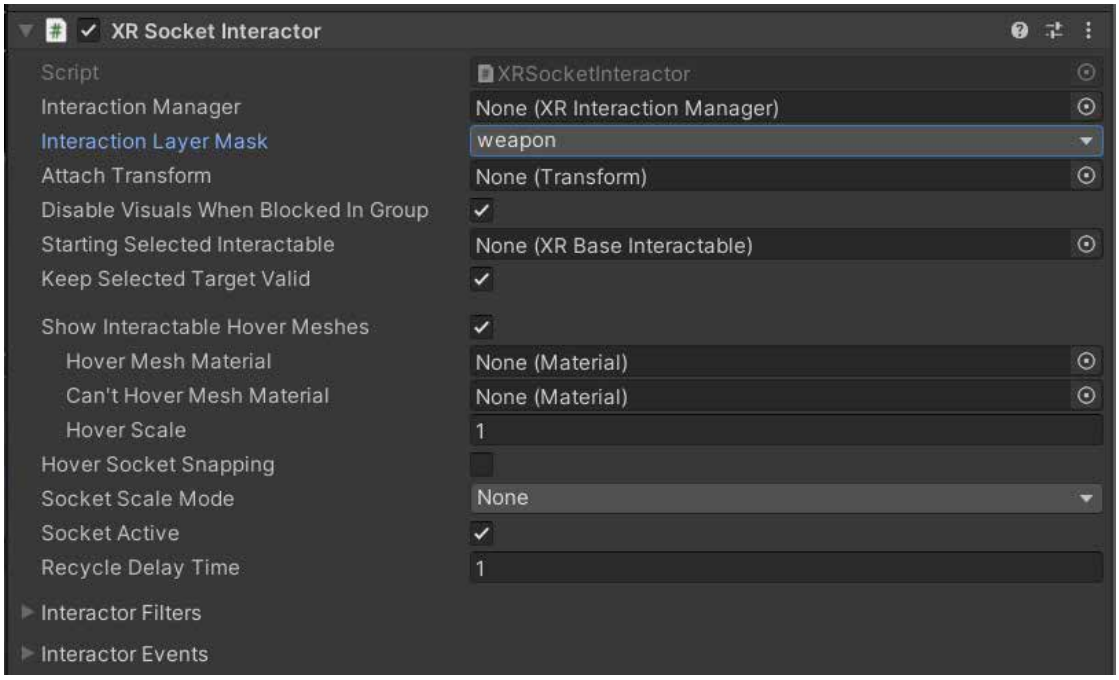


Рисунок 3.91 - Налаштування слота для зброї в інспекторі Unity

Особливості слота для зброї:

- Розташування за спиною користувача для реалістичного відчуття носіння зброї
- Адаптація позиції відповідно до зросту та повороту користувача
- Система колайдерів для правильного визначення моменту розміщення зброї

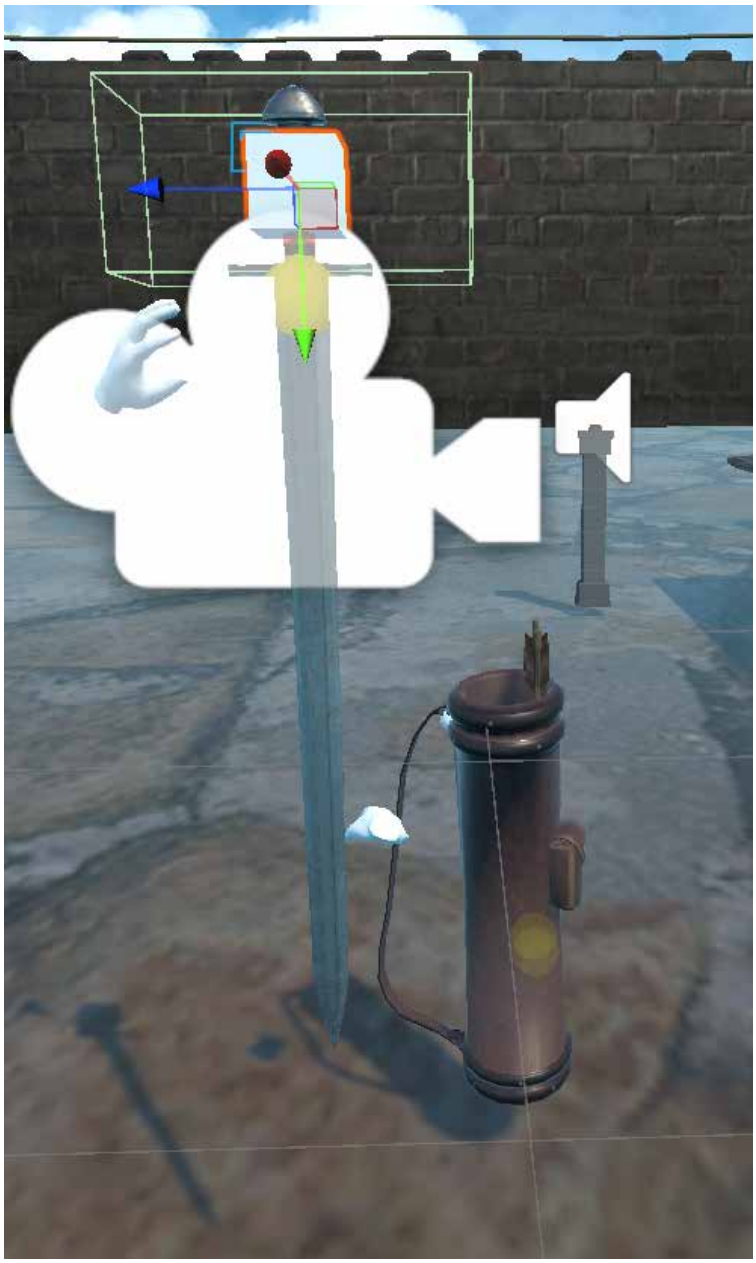


Рис. 3.92. Під час гри активований слот зброї

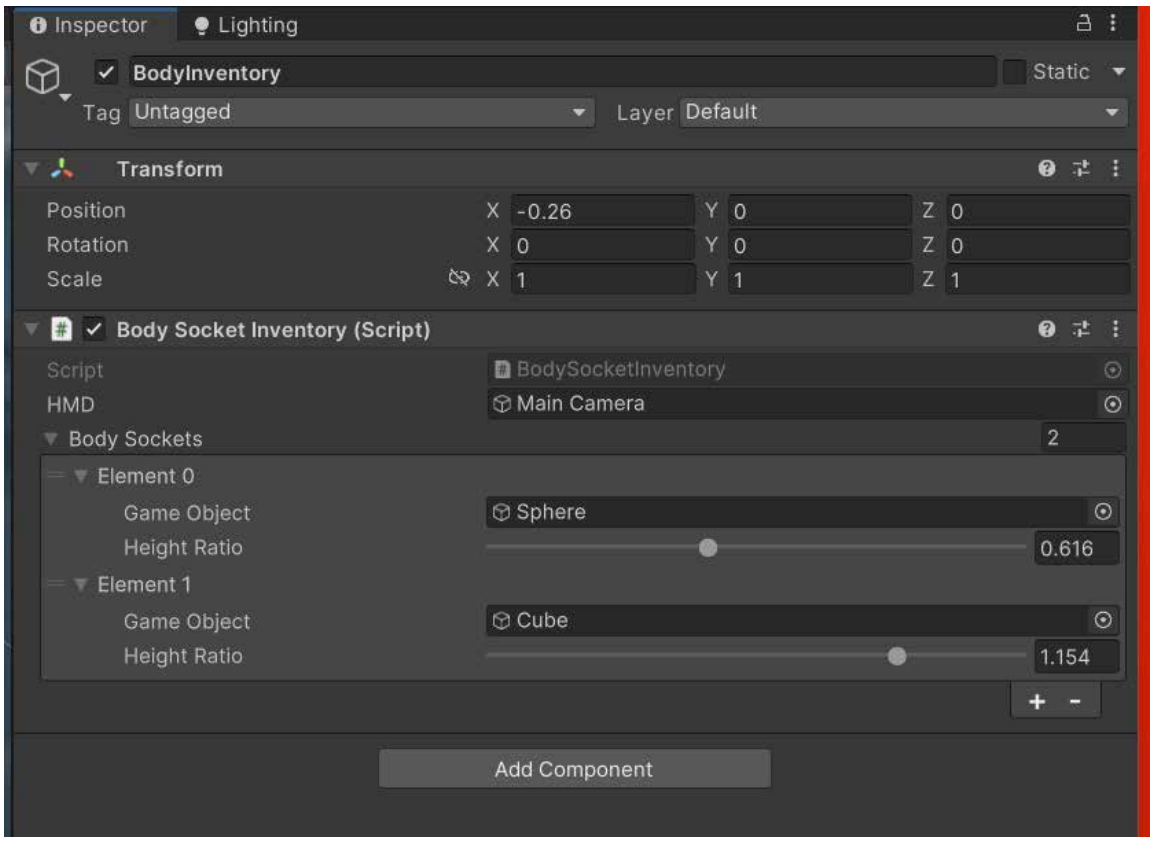


Рисунок 3.93 - Загальний вигляд системи Body Inventory у інспекторі

3.12.5. Алгоритм взаємодії користувача з Body Inventory

Для забезпечення зручної взаємодії з Body Inventory розроблено алгоритм, що дозволяє природним чином розміщувати та вилучати предмети з відповідних слотів:

1. Розміщення предмета:
   - Користувач наближає предмет до відповідного слота на тілі
   - Система визначає входження предмета у зону колайдера слота
   - При відпусканні кнопки захоплення предмет автоматично позиціонується у слоті
2. Вилучення предмета:
   - Користувач наближає руку до слота з предметом
   - При натисканні кнопки захоплення користувач бере предмет
   - Предмет переходить у стан захоплення та відокремлюється від слота

Цей алгоритм забезпечує інтуїтивний та іммерсивний досвід взаємодії, що відповідає природним людським рухам у реальному світі.

3.13 Проектування та реалізація інтерфейсу користувача

3.13.1 Концептуальні рішення дизайну інтерфейсу

Розробка інтерфейсу користувача для VR-RPG гри потребувала особливого підходу, який враховує специфіку віртуальної реальності та жанрові особливості рольових ігор. Основною концепцією стало створення іммерсивного інтерфейсу, який органічно інтегрується у віртуальний світ гри.

Візуальна стилістика

Для створення цілісного візуального стилю було обрано підхід, що поєднує класичні елементи фентезійного жанру з сучасними можливостями VR-технологій. Основні принципи дизайну включають:

Текстурний підхід до оформлення фону: Використання детальних текстур створює відчуття матеріальності та глибини, що особливо важливо у віртуальній реальності для досягнення ефекту присутності.

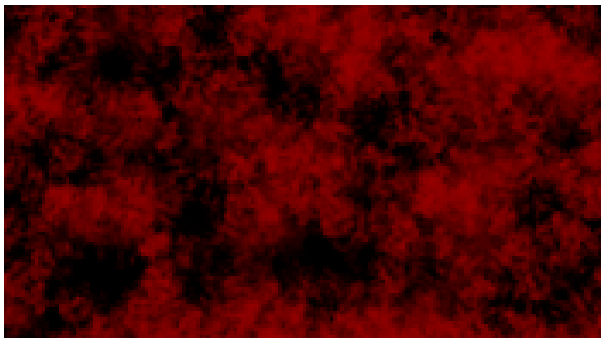


Рисунок 3.94 - Текстурний задній фон інтерфейсу

Стилізована рамкова система: Розроблено унікальну систему декоративних рамок, які використовуються для обрамлення інтерфейсних елементів. Ці рамки:

- Створюють візуальну єдність всіх елементів інтерфейсу
- Підкреслюють фентезійну атмосферу гри
- Забезпечують чітке відокремлення інтерфейсних елементів від ігрового середовища

Рисунок 3.95 - Декоративна рамка для інтерфейсних елементів

Модульна побудова інтерфейсних елементів

Для забезпечення гнучкості та масштабованості інтерфейсу було впроваджено модульний підхід до створення UI-елементів:

1. Сегментація декоративних елементів: Складні рамки розділяються на окремі компоненти, що дозволяє:
   - Адаптувати розміри під різний контент
   - Створювати нові комбінації існуючих елементів
2. Композиційний підхід: Збирання фінальних інтерфейсних блоків з підготовлених модулів забезпечує консистентність дизайну та спрощує процес розробки.

Рисунок 3.96 - Збірка фінального інтерфейсного елементу з модульних компонентів

3.13.2 Інтеграція з VR-середовищем

Система взаємодії

Для забезпечення інтуїтивної взаємодії з інтерфейсом у VR-середовищі було концептуально реалізовано:

Контролери з лучовою навігацією: Система променевого наведення дозволяє точно вибирати інтерфейсні елементи на відстані, що є критично важливим для комфортної взаємодії у віртуальній реальності.

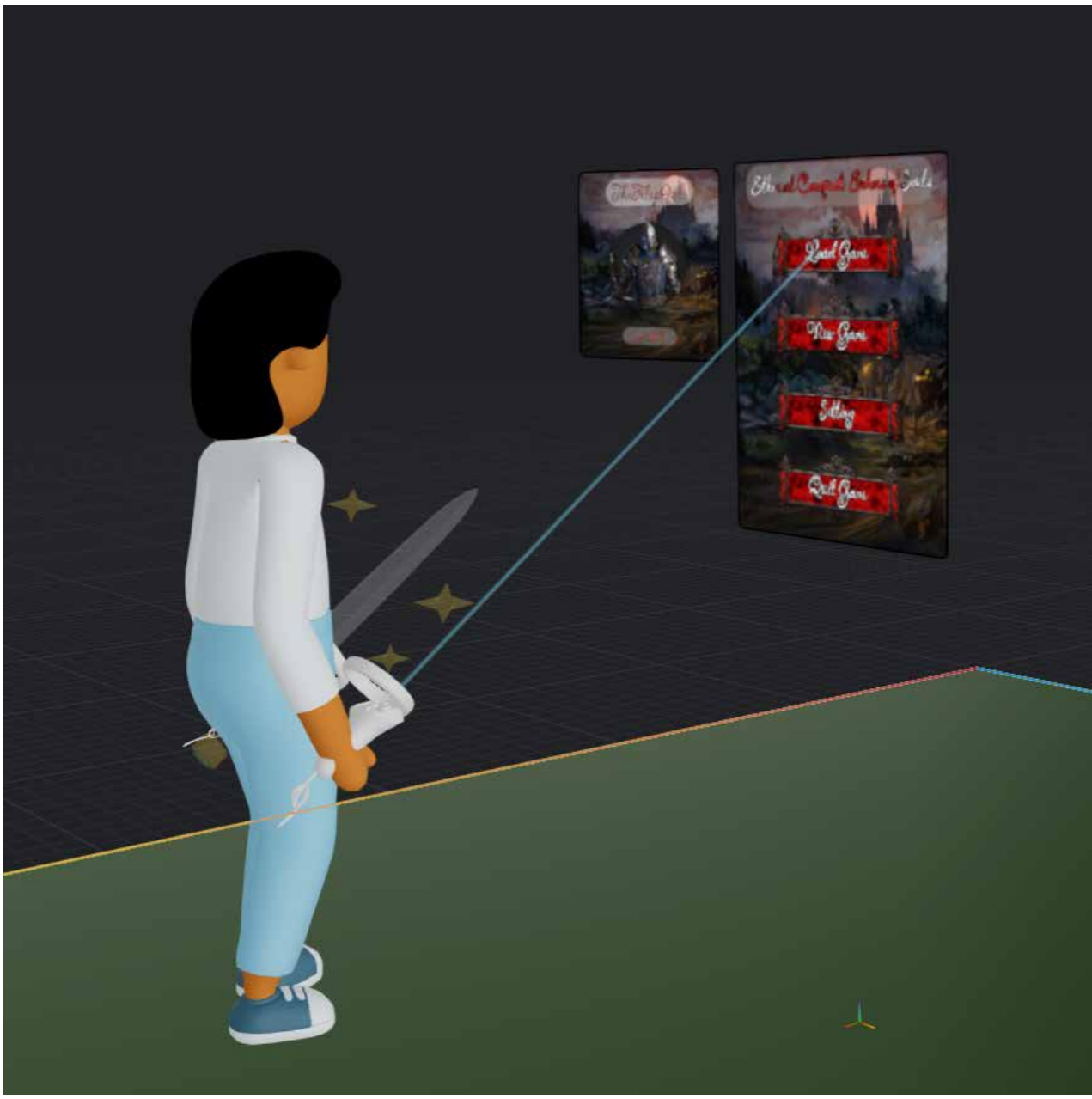


Рисунок 3.97 - Система контролерів з лучовим наведенням та псевдо-мечем

Тактильний фідбек: Використання вібрації контролерів для підтвердження вибору та навігації по меню підвищує відчуття взаємодії з віртуальними об'єктами.

3.13.3 Типи діалогових інтерфейсів

У контексті VR-RPG було визначено, що найбільш доречним є змішаний тип діалогу, який включає:

Меню-орієнтований діалог

Цей підхід передбачає:

- Представлення варіантів реплік у вигляді інтерактивного меню
- Візуальне виділення доступних опцій
- Інтуїтивну навігацію за допомогою VR-контролерів

Переваги меню-орієнтованого підходу у VR:

- Чіткість доступних варіантів дій
- Зменшення когнітивного навантаження на гравця
- Простота реалізації локалізації

Інтеграція з ігровим світом

Важливим аспектом проектування стала органічна інтеграція інтерфейсних елементів у світ гри:

- Діалогові вікна стилізовані під сувої або магічні екрани
- Анімації появи та зникнення відповідають фізичним законам віртуального світу

3.13.4 Технічна реалізація

Інструменти розробки

Для створення та інтеграції інтерфейсу використовувалися:

Figma: Для проектування статичних макетів та створення дизайн-системи.

Базель (Bezel): Веб-платформа для прототипування VR-сцен, що дозволила:

- Швидко тестувати розміщення інтерфейсу у просторі
- Оцінювати зручність взаємодії
- Ітерувати дизайн без необхідності повної збірки проекту

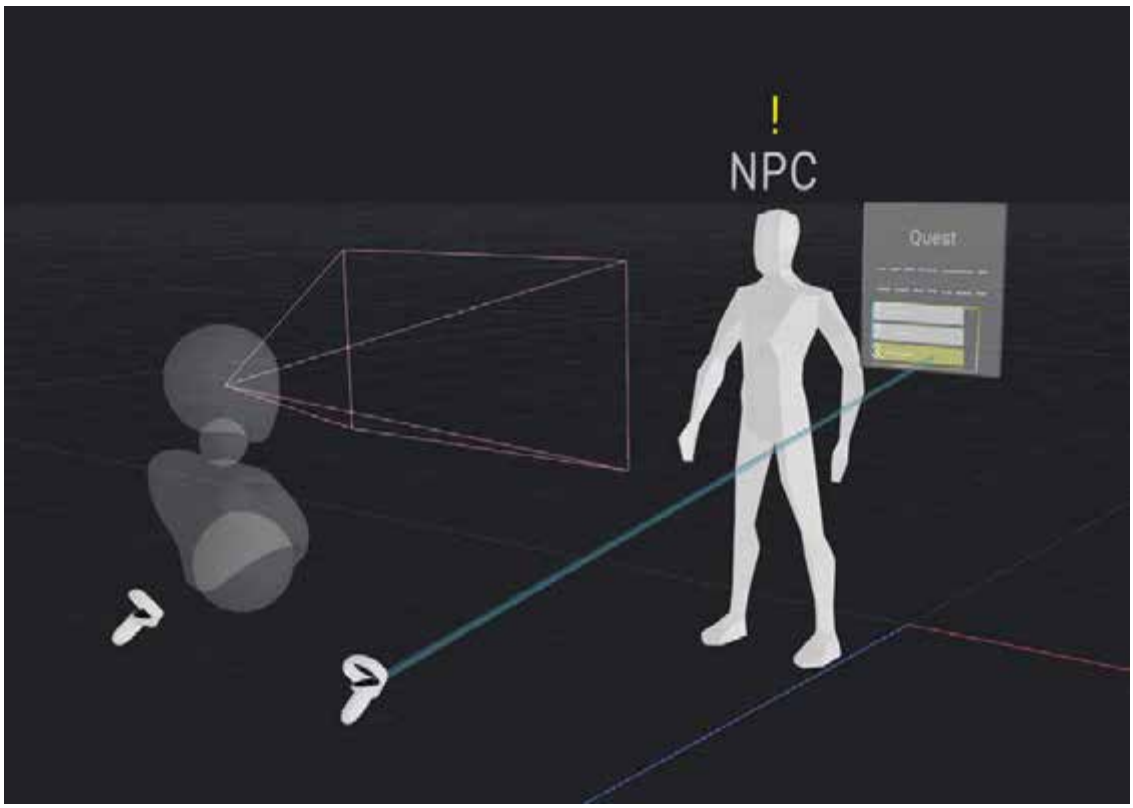


Рисунок 3.98 – Концепт меню орієнтованого діалгу в Basel

Рисунок 3.99 – Концепт меню орієнтованого діалгу в Basel від першого лиця

3.14 Висновки до розділу

Усі розроблені компоненти VR-системи формують цілісну екосистему, де окремі механіки (переміщення у VR-просторі, взаємодія з об'єктами, бойові системи, міні-ігри) взаємодіють між собою, створюючи комплексний ігровий досвід. Для забезпечення ефективної інтеграції компонентів було застосовано ряд архітектурних рішень та підходів до розробки.

Архітектурні рішення для інтеграції компонентів

1. Уніфікована система взаємодії

Усі інтерактивні об'єкти в проєкті використовують компоненти XR Interaction Toolkit (XRGrabInteractable, XRSocketInteractor тощо), що забезпечує єдиний інтерфейс взаємодії для різних типів об'єктів. Це дозволяє користувачеві застосовувати однакові жести та дії для взаємодії з різними елементами віртуального світу, підвищуючи інтуїтивність управління та спрощуючи процес навчання.

2. Модульна архітектура

Кожен функціональний елемент (зброя, мішені, інтерактивні об'єкти) реалізовано як незалежний компонент, що може бути повторно використаний у різних контекстах. Модульний підхід значно спрощує процес розробки нових функціональних елементів та інтеграції їх у існуючу систему без необхідності суттєвих змін в архітектурі проєкту.

3. Подієва модель комунікації

Взаємодія між компонентами здійснюється переважно через систему подій Unity (UnityEvent), що зменшує зв'язаність коду та підвищує гнучкість системи. Такий підхід дозволяє компонентам комунікувати між собою без жорстких залежностей, що значно спрощує процес тестування, відлагодження та модифікації окремих елементів системи.

4. Багаторівнева оптимізація

На всіх рівнях розробки — від окремих механік до цілісних сцен — застосовано різноманітні техніки оптимізації, що забезпечують стабільну

продуктивність у VR-середовищі. Особлива увага приділяється оптимізації фізичних розрахунків, рендерингу та управлінню ресурсами, що є критичними факторами для забезпечення комфортного досвіду користувача у віртуальній реальності.

Переваги обраного підходу до інтеграції

Описаний вище підхід до інтеграції компонентів дозволяє не лише підтримувати наявну функціональність, але й забезпечує значний потенціал для подальшого розширення проєкту. Модульність, слабка зв'язаність компонентів та уніфіковані інтерфейси взаємодії створюють фундамент для додавання нових механік, локацій та ігрових режимів без необхідності суттєвої переробки існуючого коду.

РОЗДІЛ 4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення оптимального досвіду середньовічної RPG у віртуальній реальності необхідно врахувати наступні вимоги до апаратного та програмного забезпечення:

4.1.1 Вимоги до ПК (для режиму PC VR)

Мінімальні вимоги:

- Процесор: Intel Core i5-9400 / AMD Ryzen 5 3600
- Оперативна пам'ять: 16 ГБ RAM
- Відеокарта: NVIDIA GeForce GTX 1660 / AMD Radeon RX 5600 XT (6 ГБ VRAM)
- Місце на диску: 2 ГБ вільного простору (SSD рекомендовано)
- Операційна система: Windows 10 (64-біт)
- Програмне забезпечення: .NET 8 Runtime, OpenXR-сумісний драйвер

Рекомендовані вимоги:

- Процесор: Intel Core i7-10700K / AMD Ryzen 7 5800X або потужніший
- Оперативна пам'ять: 16 ГБ RAM і більше
- Відеокарта: NVIDIA RTX 3070 / AMD RX 6800 XT (8+ ГБ VRAM)
- Місце на диску: 2 ГБ вільного простору (SSD обов'язково)
- Додаткове ПЗ: Останні драйвери GPU, Meta XR Plugin (для Oculus Link)

4.1.2 Вимоги до автономного використання (Meta Quest 3)

- Внутрішня пам'ять: мінімум 2 ГБ вільного простору
- Версія прошивки: остання доступна для Meta Quest 3
- Режим розробника: активований (для тестування)

- Бездротове з'єднання: Wi-Fi 6 (802.11ax) рекомендовано для Oculus Air Link
- Oculus Developer Hub: встановлений на ПК для розробки

4.1.3 Вимоги до VR-гарнітур

- Підтримувані ПК-гарнітури: Meta Quest 2/3 (через Link), Valve Index, HTC Vive (Pro), Windows Mixed Reality
- Автономні гарнітури: Meta Quest 2, Meta Quest 3
- Контролери: VR-контролери з 6DOF відстеженням, підтримка хапальних жестів
- Мінімальний рівень відстеження: 6DOF (шість ступенів свободи) для повноцінного занурення
- Додаткові аксесуари: зовнішні трекери для покращення відстеження рухів тіла (опціонально)

4.1.4 Специфічні вимоги для середньовічної RPG

- Ігровий простір: мінімум 2×2 метри для повноцінного відтворення бойової механіки
- Додаткові вимоги для графічних ефектів: підтримка DirectX 12 або Vulkan для реалістичного рендерингу середньовічних локацій
- Аудіо: рекомендовано стереонавушники з просторовим звуком для повного занурення
- Latency: максимум 20 мс для комфортних бойових сцен без симптомів VR-хвороби

4.2 Процес встановлення та розгортання системи

4.2.1 Встановлення на ПК платформі (Steam)

1. Підготовка до встановлення:
   - Перевірка відповідності системним вимогам
   - Оновлення драйверів відеокарти до останньої версії
   - Інсталяція .NET 8 Runtime (якщо відсутній)

- Встановлення та налаштування Steam VR

2. Процес інсталяції:

- Завантаження гри через клієнт Steam
- Автоматична перевірка наявності залежностей:
  - Visual C++ Redistributable 2019 або новіше
  - .NET 8 Runtime
  - OpenXR Runtime
- Налаштування параметрів VR-системи через інсталятор

3. Перше налаштування:

- Автоматичне визначення типу VR-гарнітури
- Калібрування ігрового простору для оптимальної взаємодії
- Встановлення початкових графічних налаштувань відповідно до характеристик системи
- Генерація файлів ігрового профілю

4.2.2 Встановлення на Meta Quest 3

1. Підготовка гарнітури:

- Оновлення прошивки Quest 3 до останньої версії
- Активація режиму розробника через мобільний додаток Meta
- Включення "Невідомих джерел" у налаштуваннях

2. Процес інсталяції:

- Варіант 1 (через Oculus Developer Hub):
  - Підключення Quest 3 до ПК
  - Завантаження APK через Oculus Developer Hub
  - Перевірка встановлення та запуск програми
- Варіант 2 (через SideQuest):
  - Встановлення SideQuest на ПК
  - Підключення Quest 3 через USB
  - Встановлення APK через SideQuest інтерфейс
- Варіант 3 (через Meta App Lab):

- Пошук гри у каталозі App Lab через мобільний додаток Meta
- Завантаження та встановлення за стандартною процедурою

3. Перше налаштування:
   - Автоматичне визначення характеристик гарнітури
   - Налаштування зони Guardian для безпечного геймплею
   - Оптимізація графічних параметрів під продуктивність Quest 3

4.2.3 Особливості розгортання RPG-компонентів

- Початкові ресурси: генерація початкового інвентарю та предметів при першому запуску
- Ігрові світи: поетапне завантаження великих середньовічних локацій для оптимізації пам'яті
- RPG-система: налаштування складності та балансу під технічні можливості пристрою

4.3 Оновлення та технічна підтримка

4.3.1 Стратегія випуску оновлень

- Версіонування: використання семантичного версіонування (MAJOR.MINOR.PATCH)
  - MAJOR: значні зміни геймплею, нові RPG-механіки
  - MINOR: додавання нового контенту (квести, зброя, локації)
  - PATCH: виправлення помилок, оптимізація
- Канали релізів:
  - Стабільний: перевірені оновлення для всіх користувачів
  - Бета: тестування нового контенту для добровольців
  - Експериментальний: тестування нових RPG-механік (опціонально)
- Частота оновлень:
  - Патчі (PATCH): за необхідністю, 2-4 рази на місяць
  - Контентні оновлення (MINOR): щоквартально
  - Великі оновлення (MAJOR): 2-3 рази на рік

4.3.2 Технічні засоби розповсюдження оновлень

- Платформа Steam:
  - Використання Steamworks для автоматичних оновлень
  - Диференційовані патчі для зменшення обсягу завантажень
  - Система відкату до попередніх версій у разі проблем
- Платформа Meta Quest:
  - Оновлення через Meta App Lab/Store
  - Резервне копіювання збережень перед оновленням
  - Поетапне розгортання для виявлення потенційних проблем
- CI/CD інструменти:
  - Використання Jenkins/GitHub Actions для автоматизації збірки
  - Автоматичне тестування сумісності з різними VR-платформами
  - Налаштування системи сповіщень про критичні помилки

4.3.3 Моніторинг і зворотний зв'язок

- Система логування:
  - Реєстрація помилок з контекстною інформацією
  - Анонімізована телеметрія продуктивності (FPS, завантаження CPU/GPU)
  - Відстеження прогресу в RPG-квестах для виявлення дисбалансу
- Взаємодія з користувачами:
  - Інтегрована система звітів про помилки
  - Форум спільноти для обговорення геймплею та балансу
  - Регулярні опитування щодо вподобань та очікувань
- Аналітика геймплею:
  - Збирання анонімних даних про проходження RPG-контенту
  - Аналіз взаємодії з ігровими механіками
  - Відстеження використання різних класів персонажів та зброї

4.4 Безпека та конфіденційність користувачів

4.4.1 Захист персональних даних

- Типи даних, що збираються:
  - Обов'язкові: обліковий запис, налаштування, ігровий прогрес
  - Опціональні: телеметрія, статистика використання VR-контролерів
  - Фізіологічні: відстеження рухів тіла (анонімізовано)
- Безпечне зберігання:
  - Шифрування даних користувача (AES-256)
  - Ізоляція персональних даних від телеметрії
  - Локальне зберігання чутливої інформації
- Передача даних:
  - Використання HTTPS/SSL для всіх мережевих комунікацій
  - Мінімізація обсягу даних, що передаються
  - Анонімізація перед агрегацією для аналітики

4.4.2 Відповідність законодавству

- GDPR:
  - Чітке інформування про збір та використання даних
  - Можливість експорту та видалення персональних даних
  - Механізми отримання згоди на збір різних типів даних
- Українське законодавство:
  - Відповідність Закону України "Про захист персональних даних"
  - Адаптація політики конфіденційності до місцевих вимог
  - Забезпечення прав користувачів на керування своїми даними
- Вікові обмеження:
  - Реалізація перевірки віку відповідно до вимог PEGI/ESRB
  - Обмеження контенту для неповнолітніх користувачів
  - Додаткові налаштування батьківського контролю

4.4.3 Безпека програмного забезпечення

- Захист від несанкціонованого доступу:
  - Цифрове підписування програмного коду
  - Обфускація критичних компонентів
  - Захист від модифікації ігрових даних (античит)
- Тестування безпеки:
  - Регулярний аудит коду на вразливості
  - Penetration testing перед великими оновленнями
  - Code review критичних компонентів безпеки
- Реагування на загрози:
  - План оперативного випуску патчів безпеки
  - Система сповіщення користувачів про критичні оновлення
  - Процедура ескалації інцидентів безпеки

4.5 Адаптація до VR-платформ та розповсюдження

4.5.1 Платформа SteamVR (ПК)

- Технічна адаптація:
  - Підтримка OpenXR та SteamVR Runtime
  - Оптимізація під різні VR-гарнітури (Valve Index, HTC Vive, Oculus Rift)
  - Налаштування високоякісних графічних пресетів для потужних ПК
- Інтеграція з екосистемою Steam:
  - Реалізація досягнень, пов'язаних з RPG-прогресією
  - Підтримка Steam Workshop для користувацьких модифікацій
  - Використання Steam Cloud для синхронізації збережень
- Особливості розповсюдження:
  - Створення привабливої сторінки гри з середньовічною тематикою
  - Підготовка маркетингових матеріалів (трейлери, скріншоти)
  - Участь у фестивалях та акціях Steam
- Адаптація RPG-механік:

- Налаштування управління для різних типів VR-контролерів
- Масштабування складності бойової системи під технічні можливості
- Деталізація графіки для занурення у середньовічний світ

4.5.2 Платформа Meta (Quest)

- Технічна оптимізація:
  - Зменшення полігонального навантаження на 30-40% порівняно з ПК-версією
  - Використання ефективних методів рендерингу (MSAA замість SSAA)
  - Оптимізація шейдерів та текстур для мобільного GPU
- Адаптація управління:
  - Перебудова інтерфейсу під Oculus Touch контролери
  - Спрощення деяких RPG-механік для зручності управління
  - Використання вбудованих жестів Quest для швидкого доступу до меню
- Розповсюдження через Meta екосистему:
  - Підготовка до рев'ю у Meta App Lab/Store
  - Чітке дотримання вимог Meta щодо контенту
  - Використання соціальних функцій Meta для мультиплеєра (опціонально)
- Автономні можливості:
  - Незалежність від ПК при збереженні ключових RPG-елементів
  - Синхронізація прогресу між ПК і Quest-версіями
  - Оптимізація великих середньовічних локацій для автономного режиму

4.5.3 Крос-платформена сумісність

- Єдина система акаунтів:
  - Синхронізація прогресу між усіма підтримуваними платформами
  - Збереження інвентарю та характеристик персонажа

- Крос-платформенні досягнення
- Адаптивні налаштування:
- Автоматичне визначення VR-платформи при запуску
- Масштабування якості графіки відповідно до характеристик пристрою
- Автоматичне налаштування схеми управління
- Підтримка соціальних функцій:
- Крос-платформенний мультиплеєр (якщо передбачений)
- Спільна таблиця лідерів між різними платформами
- Сумісність збережень та прогресу

4.5.4 Майбутні напрямки розвитку

- Підтримка нових VR-платформ:
- Готовність до адаптації під нові гарнітури від Sony, Pico
- Моніторинг тенденцій ринку VR для раннього входу на перспективні платформи
- Модульна архітектура для спрощення портування
- Розширення RPG-функціоналу:
- Планування DLC з новими регіонами середньовічного світу
- Впровадження нових класів персонажів та бойових механік
- Розширення системи крафту та економіки
- Технологічні вдосконалення:
- Впровадження технологій штучного інтелекту для NPC
- Покращення фізики взаємодії з середньовічними об'єктами
- Розробка додаткових систем для підвищення іммерсивності

4.6 Результати тестування

Система була протестована на наступних конфігураціях:

Тестова конфігурація 1:

- Windows 10 Pro 22H2
- AMD Ryzen 5 3600, 3.6GHz
- 32,0 ГБ (4x8), DDR4-3600МГц
- AMD Rx 5600 XT
- Шолом Lenovo Windows Mix Reality

Результат тестування показав стабільну продуктивність і плавність роботи при середніх налаштуваннях графіки. Система успішно працює з усіма ключовими функціями VR-інтерфейсу та RPG-механіками.

Рис. 4.1 – Результати тестування конфігурації 1

Рис. 4.2 – Результати тестування конфігурації 1

Тестова конфігурація 2:

- AMD Ryzen 7800X3D
- NVIDIA RTX 4080 Super
- 32 ГБ DDR5 6400MHz
- Шолом Oculus Quest 3:

На даній конфігурації система працює з обмеженням на 72 FPS відповідно до частоти оновлення VR-гарнітури.

Рис. 4.3 – Результати тестування конфігурації 2

Процесор загружений на 20%, відеокарта на 43%, що свідчить про значний запас продуктивності системи і можливість роботи з подвійною частотою кадрів при необхідності.

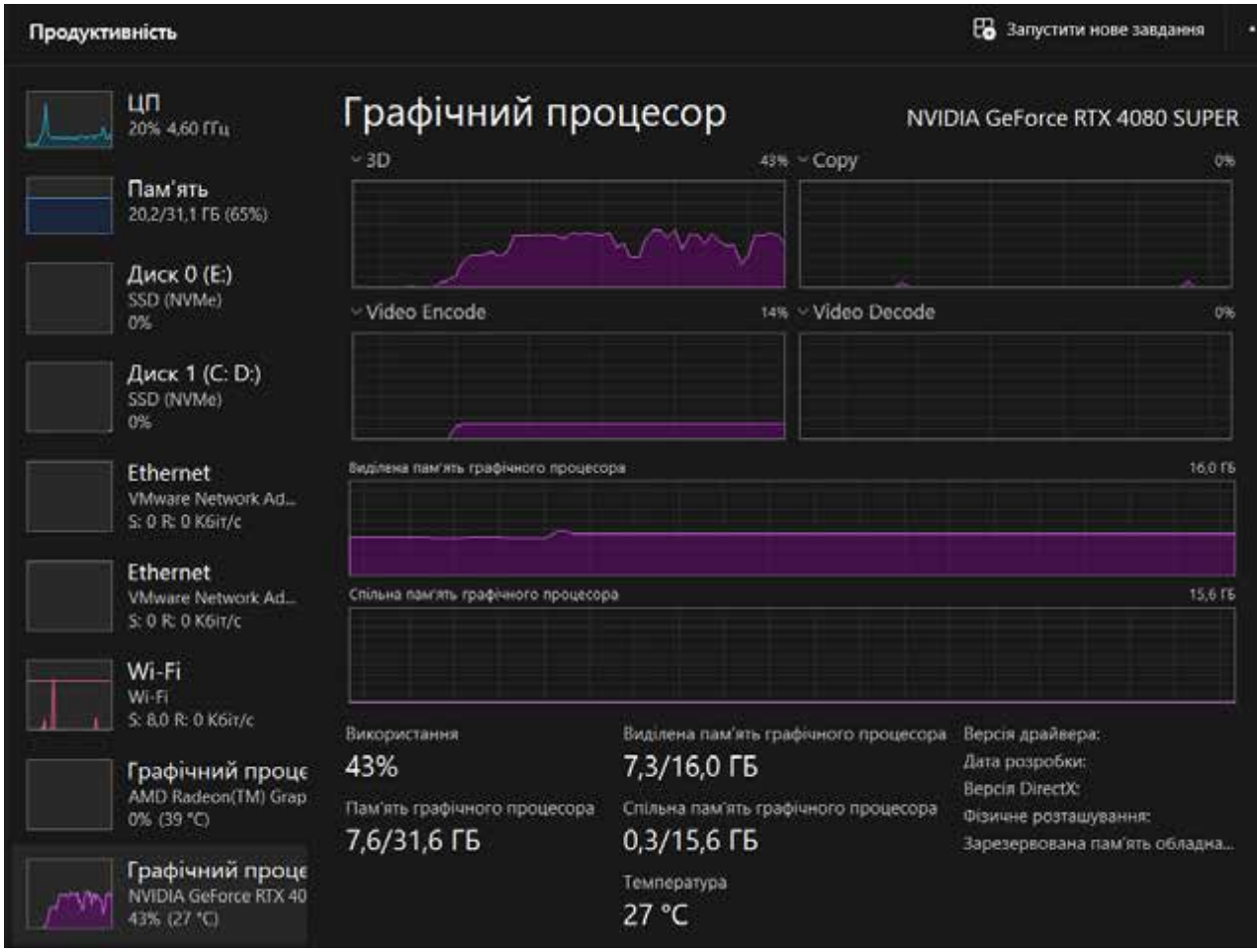


Рис. 4.4 – Результати тестування конфігурації 2

Тестова конфігурація 3:

Intel i7-10700kf

DDR 4 – 16g

AMD-RX7600

Шолом Oculus Quest 2:

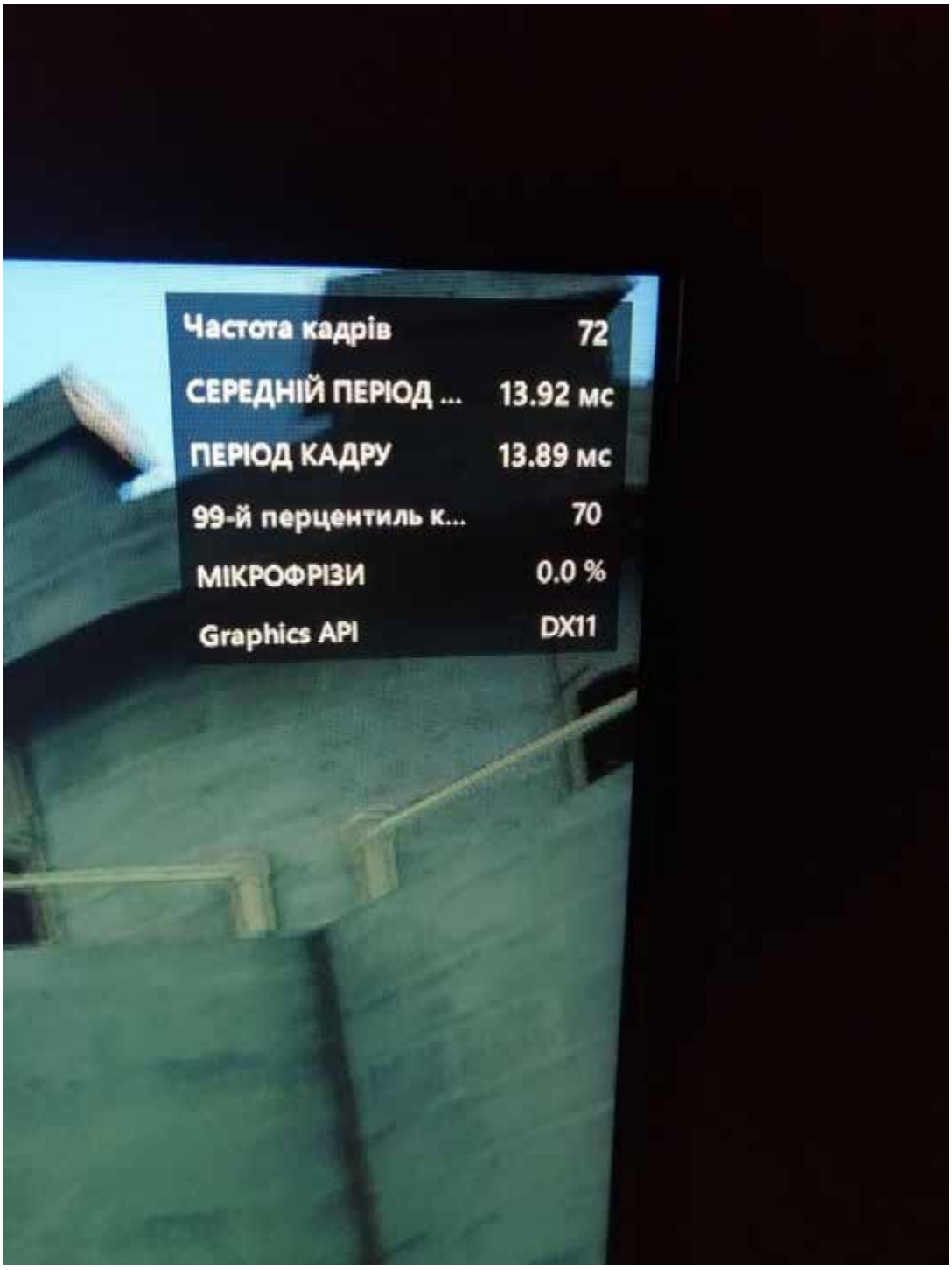


Рис. 4.5 – Результати тестування конфігурації 3

4.7 Висновки до розділу

У розділі 4 було проаналізовано та визначено необхідні вимоги до апаратного та програмного забезпечення для успішного функціонування розробленої VR-системи з середньовічною RPG-тематикою. Детально описано процеси встановлення та розгортання системи на різних платформах, а також рекомендації щодо оновлення та технічної підтримки.

Особлива увага була приділена питанням безпеки та конфіденційності користувачів, відповідності законодавчим вимогам та захисту програмного забезпечення від несанкціонованого доступу. Розроблено стратегію адаптації системи до різних VR-платформ, включаючи ПК з SteamVR та автономні пристрої Meta Quest.

Результати тестування на різних апаратних конфігураціях продемонстрували стабільність та продуктивність системи, що підтверджує правильність прийнятих технічних рішень та оптимізації.

Запропоновані рекомендації забезпечують ефективне впровадження та експлуатацію розробленої системи, її масштабування та адаптацію до майбутніх технологічних змін у галузі віртуальної реальності.

ВИСНОВКИ

У рамках виконаної роботи було розроблено програмне забезпечення VR-ігри жанру RPG на основі технологій .NET. Реалізовано основні функціональні механіки гри: системи руху гравця та взаємодії з оточенням у VR (відстеження рухів голови й рук для захоплення й переміщення предметів), бойову систему, інтерактивні міні-ігри, систему інвентаря та інші ігрові елементи. Для цього створено повноцінний прототип гри. Досягнуто комплексної реалізації ключових аспектів VR-ігрового досвіду – від іммерсивної взаємодії з об'єктами віртуального світу до звичних елементів гри. Особливу цінність цієї роботи становить синтез специфічних VR-механік із традиційними ігровими системами жанру RPG і створення на цій основі дієвого прототипу. Пристрої віртуальної реальності забезпечують просторове відстеження положення голови, контролерів та кистей, що підвищує очікування тісної фізичної взаємодії з навколишнім середовищем: користувачі можуть піднімати та маніпулювати віртуальними об'єктами. Отриманий прототип забезпечує 360°-досвід: користувач повністю занурюється в цифрове середовище і може природно досліджувати його, маніпулюючи предметами, ніби був фізично присутній у грі. Це відповідає доведеному в галузі уявленню, що імерсивна VR-графіка дозволяє гравцям «повністю зануритися» в віртуальний світ та активно брати участь у дослідженні світу. Результати тестування підтвердили універсальність рішення. Прототип було перевірено на трьох різних апаратних конфігураціях (власна тестова система та дві сторонні), що продемонструвало сумісність гри з різними моделями VR-шоломів. Гра стабільно працює незалежно від особливостей обладнання, що свідчить про високу адаптивність розробленого ПЗ до різних VR-платформ та середовищ. Розроблений прототип має відчутну прикладну значущість.

Подальші перспективи розвитку проєкту пов'язані з розширенням ігрового наповнення та функціоналу. Можливі напрямки включають:

- Додавання ворогів та NPC для оживлення віртуального світу.

- Впровадження системи торгівлі між персонажами та NPC.
- Розробку сюжетної кампанії та різноманітних активностей (квестів, випробувань) на ігровій мапі.
- Систему рівнів персонаджа.
- Інтеграцію мультиплеєрного режиму для спільної взаємодії декількох гравців.

Отже, у виконаній роботі реалізовано комплексне програмне забезпечення VR-гри в жанрі RPG, поєднано сучасні VR-механіки з ігровими системами та підтверджено працездатність готового прототипу. Ці результати підкреслюють актуальність поєднання VR-технологій з класичними ігровими жанрами та створюють базу для подальшого розвитку VR-проєктів у розважальній сферах

ДЖЕРЕЛА

[1] RealaryVR. *spaderdabomb. Build a beautiful 3D open world in 5 minutes*. YouTube, 2022. URL: https://www.youtube.com/watch?v=nCDGjLRecrs

[2] Valem Tutorials. *How to Slice in VR – Unity XR Tutorial*. YouTube, 2021. URL: https://www.youtube.com/watch?v=gfVR1rWhjxk

[3] Gabriel Aguiar Prod.. *Unity Shader Graph - Liquid Effect Tutorial*. YouTube, 2022. URL: https://www.youtube.com/watch?v=tI3USKIbnh0

[4] RealaryVR. *Unity VR Development for Oculus Quest: Inventory*. YouTube, 2022. URL: https://www.youtube.com/watch?v=gAz_SeDUQBk

[5] Linowes J. Unity Virtual Reality Projects. Packt Publishing, 2021.

[6] Unity Technologies. Unity Manual – XR Development. Unity Docs, 2023. URL: https://docs.unity3d.com/Manual/XR.html

[7] Valem Tutorials. XR Interaction Toolkit Series. YouTube Playlist, 2022. URL:https://www.youtube.com/playlist?list=PLpPqplz6dKxX8t6Q73f91QShpGCU9Q3iC

[8] Oculus Developer. VR Performance Optimization Guide. Oculus Blog, 2021. URL: https://developer.oculus.com/blog/tech-note-vr-performance-optimization/

[9] Valve. SteamVR – Best Practices. Steamworks Documentation, 2022. URL: https://partner.steamgames.com/doc/features/vr/best_practices

[10] Brackeys. How to make Terrain in Unity. YouTube, 2020. URL: https://www.youtube.com/watch?v=8P0p-Mf5z5Y

[11] Slater M., Sanchez-Vives M. V. Enhancing Our Lives with Immersive Virtual Reality. Frontiers in Robotics and AI, Vol. 3, 2016. DOI: 10.3389/frobt.2016.00074

[12] Sketchfab. Platform for Downloading 3D Models. Sketchfab, 2023. URL: https://sketchfab.com

[13] Omogonix. How to Change Scenes in Unity Using C#. YouTube, 2021. URL: https://www.youtube.com/watch?v=gtpXc_9MR6g

ArcherGameController

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;
using System.Collections.Generic;
public class ArcherGameController : MonoBehaviour
{
   [Header("Target Settings")]
   public GameObject targetPrefab;
   public Transform spawnOrigin; // Зазвичай це сам Acher (цей об'єкт)
   public float spawnRadius = 2.5f;
   [Header("UI")]
   public TMP_Text scoreText;
   public TMP_Text timerText;
   [Header("Game Settings")]
   public float totalTime = 60f;
   private float timeRemaining;
   private int score;
   private bool isGameRunning;
   private List<TargetController> currentTargets = new List<TargetController>();
   private int currentWaveSize = 1;
   [Header("Time Bonus Settings")]
   public float baseTimeBonus = 0.5f;
   public float bonusPerWaveMultiplier = 0.2f; // бонус зростає на кожну нову хвилю
   [Header("Floating Text")]
   public GameObject floatingTextPrefab;
   public void StartGame()
```

```
{
    Debug.Log("Game Started!");
    isGameRunning = true;
    timeRemaining = totalTime;
    score = 0;
    currentWaveSize = 1;
    ClearAllTargets();
    SpawnTargets(currentWaveSize, true);
    StartCoroutine(GameTimer());
    UpdateUI();
}
private IEnumerator GameTimer()
{
    while (timeRemaining > 0f)
    {
        timeRemaining -= Time.deltaTime;
        UpdateUI();
        yield return null;
    }
    EndGame();
}
private void EndGame()
{
    isGameRunning = false;
    ClearAllTargets();
    Debug.Log("Game Over! Score: " + score);
}
private void UpdateUI()
{
    if (scoreText != null)
```

```
        scoreText.text = "Score: " + score;
    if (timerText != null)
        timerText.text = "Time: " + Mathf.CeilToInt(timeRemaining);
}
private void ClearAllTargets()
{
    foreach (var target in currentTargets)
    {
        if (target != null)
            Destroy(target.gameObject);
    }   currentTargets.Clear();
}
public void OnTargetHit(TargetController target)
{
    if (!isGameRunning) return;
    score += 1;
    currentTargets.Remove(target);
    float bonusTime = baseTimeBonus + bonusPerWaveMultiplier *
Mathf.Log(currentWaveSize, 2);
    timeRemaining += bonusTime;
    UpdateUI();
    if (currentTargets.Count == 0)
    {
        currentWaveSize *= 2;
        SpawnTargets(currentWaveSize, false);
    }
    if (floatingTextPrefab != null && target != null)
    {
        Vector3 spawnPos = target.transform.position + Vector3.up * 0.5f;
        GameObject textObj = Instantiate(floatingTextPrefab, spawnPos, Quaternion.identity);
```

```
        FloatingText ft = textObj.GetComponent<FloatingText>();
        if (ft != null)
            ft.SetText("+" + bonusTime.ToString("0.0") + "s");
    }
}
private void SpawnTargets(int count, bool isFirstWave)
{
    ClearAllTargets();
    float amplitude = 0.3f + 0.2f * Mathf.Log(count, 2);
    float speed = 1.5f + 0.5f * Mathf.Log(count, 2);
    float scale = Mathf.Clamp(1f - 0.2f * Mathf.Log(count, 2), 0.3f, 1f);
    float minDistance = scale * 1.2f;
    List<Vector3> usedPositions = new List<Vector3>();
    for (int i = 0; i < count; i++)
    {
        Vector3 localPos;
        int attempts = 0;
        do
        {           if (isFirstWave && i == 0)
            {
                localPos = Vector3.zero;
                break;
            }
            // РАДІУС СПАВНУ (напр. множимо на 1.5f)
            localPos = (Vector3)(Random.insideUnitCircle * spawnRadius * 1.5f);
            attempts++;
            if (attempts > 100)
            {
                Debug.LogWarning("Could not find non-overlapping position after 100 attempts.");
                break;
```

```
            }
        } while (IsTooClose(localPos, usedPositions, minDistance));
        usedPositions.Add(localPos);
        Vector3 worldPos = spawnOrigin.TransformPoint(localPos);
        // СПАВН ІЗ РОЗВОРОТОМ по X
        Quaternion rotation = Quaternion.Euler(180f, 0f, 0f);
        GameObject targetObj = Instantiate(targetPrefab, worldPos, rotation);
        TargetController tc = targetObj.GetComponent<TargetController>();
        if (tc != null)
        {
            tc.Init(this, amplitude, speed, Vector3.one * scale);
            currentTargets.Add(tc);
        }
    }
  }
  private bool IsTooClose(Vector3 newPos, List<Vector3> existing, float minDist)
  {
    foreach (var pos in existing)
    {
      if (Vector3.Distance(newPos, pos) < minDist)
          return true;
    }
    return false;
  }
}
```

ArrowController

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;
using System.Collections;
```

```
public class ArrowController : MonoBehaviour
{
   [Header("Autofind settings")]
   [SerializeField] private XRSocketInteractor socket;
   [SerializeField] private Transform bowTransform;
   [SerializeField] private Rigidbody rb;
   private bool isInserted = false;
   private bool wasFired = false;
   private bool isHeld = false;
   private Vector3 initialLocalPosition;
   private Quaternion initialLocalRotation;
   private Transform stringTransform;
   private void Awake()
   {  if (rb == null) rb = GetComponent<Rigidbody>();
      if (socket == null)
      {var sockObj = GameObject.FindWithTag("BowSocket");
         socket = sockObj != null ? sockObj.GetComponent<XRSocketInteractor>() :
FindObjectOfType<XRSocketInteractor>();}
      if (socket == null)
         Debug.LogError("ArrowController: No socket found!");
      stringTransform = socket.transform;
      if (bowTransform == null)
         bowTransform = stringTransform.parent;
      if (bowTransform == null)
         Debug.LogError("ArrowController: No bowTransform assigned!");
      rb.isKinematic = false;
      if (socket != null)
         socket.socketActive = false;
      XRGrabInteractable grab = GetComponent<XRGrabInteractable>();
      if (grab != null)
```

```
    {grab.selectEntered.AddListener(OnGrab);
        grab.selectExited.AddListener(OnRelease);} }
private void Update()
{if (!isInserted)
        return;
    Vector3 lp = initialLocalPosition;
    lp.x = stringTransform.localPosition.x;
    transform.localPosition = lp;
    transform.rotation = bowTransform.rotation * initialLocalRotation; }
private void OnGrab(SelectEnterEventArgs args)
{
    isHeld = true;
    if (socket != null)
        socket.socketActive = true;
    var activator = FindObjectOfType<ArrowSocketActivator>();
    if (activator != null)
        activator.EnableForArrow(GetComponent<XRGrabInteractable>());
}
private void OnRelease(SelectExitEventArgs args)
{isHeld = false;
    if (!isInserted && socket != null)
        socket.socketActive = false;
    var activator = FindObjectOfType<ArrowSocketActivator>();
    if (activator != null)
        activator.DisableSocket();
}
private void OnTriggerEnter(Collider other)
{if (isInserted || wasFired || !isHeld) return;
    if (other.GetComponent<XRSocketInteractor>() == socket)
    {InsertIntoSocket();
```

```
    }
}
private void InsertIntoSocket()
{isInserted = true;
    wasFired = false;
    isHeld = false;
    var grab = GetComponent<XRGrabInteractable>();
    if (grab != null)
        grab.enabled = false;
    transform.SetParent(bowTransform, false);
    transform.localPosition = new Vector3(0f, 0.31f, -0.01f);
    transform.localRotation = Quaternion.Euler(0f, 0f, 270f);
    initialLocalPosition = transform.localPosition;
    initialLocalRotation = transform.localRotation;
    rb.isKinematic = true;
    rb.velocity = Vector3.zero;
    rb.angularVelocity = Vector3.zero;
    BowStringController bowStringController = FindObjectOfType<BowStringController>();
    if (bowStringController != null)
    {
        bowStringController.SetCurrentArrow(this);
    }
    Debug.Log("Arrow inserted into socket successfully!");
}
public void Fire(Vector3 force)
{
    if (!isInserted) return;
    rb.collisionDetectionMode = CollisionDetectionMode.ContinuousDynamic;
    rb.isKinematic = false;
    isInserted = false;
```

```
    wasFired = true;
    transform.SetParent(null);
    var grab = GetComponent<XRGrabInteractable>();
    if (grab != null)
    {
        grab.enabled = false;
        grab.interactionLayers = 0;
        Destroy(grab);
    }
    if (socket != null)
    {
        var selected = socket.firstInteractableSelected;
        if (selected != null)
        {
            socket.EndManualInteraction();
            socket.interactionManager.SelectExit(socket, selected);
        }
        socket.socketActive = false;
        Invoke(nameof(ReenableSocket), 1f);
    }
    float minForceThreshold = 1f;
    if (force.magnitude < minForceThreshold)
        force = force.normalized * minForceThreshold;
    StartCoroutine(ApplyForceNextFrame(force));
    Invoke(nameof(ReenableArrowInteraction), 1f);
}
private IEnumerator ApplyForceNextFrame(Vector3 force)
{
    yield return null;
    rb.velocity = Vector3.zero;
```

```
    rb.angularVelocity = Vector3.zero;
    rb.AddForce(force, ForceMode.Impulse);
}
private void ReenableArrowInteraction()
{
    Debug.Log("Reenabling arrow interaction...");

    var grab = GetComponent<XRGrabInteractable>();
    if (grab != null)
    {
        grab.interactionLayers = InteractionLayerMask.GetMask("Default");
        grab.enabled = true;
    }
    wasFired = false;
}
private void ReenableSocket()
{
    if (socket != null)
        socket.socketActive = true;
}

private void OnCollisionEnter(Collision collision)
{
    if (wasFired && collision.gameObject.CompareTag("Target"))
    {
        StickToTarget(collision);
    }
}
private void StickToTarget(Collision collision)
{
```

```
    rb.isKinematic = true;
    rb.velocity = Vector3.zero;
    rb.angularVelocity = Vector3.zero;
    Rigidbody targetRb = collision.rigidbody;
    if (targetRb != null)
    {
        targetRb.isKinematic = false;
    }
    transform.SetParent(collision.transform);
    TargetMover target = collision.gameObject.GetComponent<TargetMover>();
    if (target != null)
    {
        target.OnHit();
    }
    ContactPoint contact = collision.contacts[0];
    transform.position = contact.point;
    Vector3 direction = rb.velocity.normalized;
    if (direction != Vector3.zero)
        transform.rotation = Quaternion.LookRotation(direction);
}}
```

BowStringController

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;
public class BowStringController : MonoBehaviour
{
   [Header("References")]
   [SerializeField] private XRGrabInteractable interactable;
   [SerializeField] private Transform WB_string, Bow;
   private Transform interactor;
   private Vector3 initialPosition;
   private float tensionForce = 0f;
   private float lastTensionForce = 0f;
   private ArrowController currentArrow;
   private void Awake()
   {interactable = WB_string.GetComponent<XRGrabInteractable>();
      initialPosition = WB_string.localPosition;
   }
   private void Start()
   {interactable.selectEntered.AddListener(PrepareBowString);
      interactable.selectExited.AddListener(ReleaseBowString);
   }
   private void PrepareBowString(SelectEnterEventArgs arg0)
   {interactor = arg0.interactorObject.transform;
      WB_string.SetParent(Bow, true);
      tensionForce = 0f;
   }
   private void ReleaseBowString(SelectExitEventArgs arg0)
   {
      interactor = null;
```

```
    WB_string.SetParent(Bow, true);
    WB_string.localPosition = initialPosition;
    FireArrow();
  }
  private void Update()
  {
    if (interactor != null)
    {WB_string.SetParent(Bow, true);
      Vector3 localInteractorPosition = Bow.InverseTransformPoint(interactor.position);
      localInteractorPosition.x = Mathf.Clamp(localInteractorPosition.x, -0.4f, 0f);
      Vector3 newLocalPosition = initialPosition;
      newLocalPosition.x = localInteractorPosition.x;
      if (localInteractorPosition.x < -0.01f)
      {tensionForce = Mathf.Abs(localInteractorPosition.x) * 8f;
        lastTensionForce = tensionForce;
        Debug.Log($"Tensioning! localX: {localInteractorPosition.x}, tensionForce:
{tensionForce}");
      }
      else
      {tensionForce = 0f;
      }
      WB_string.localPosition = newLocalPosition;
      if (currentArrow != null)
      {Vector3 arrowPosition = currentArrow.transform.localPosition;
        arrowPosition.x = newLocalPosition.x; / / Прив'язуємо рух стріли до тетиви
        currentArrow.transform.localPosition = arrowPosition;
      }
    }
  }
  private void FireArrow()
```

```
  {
     if (currentArrow == null)
     {currentArrow = FindObjectOfType<ArrowController>();
     }
     if (currentArrow != null && lastTensionForce > 0f)
     {
        Debug.Log("Firing current arrow with force " + lastTensionForce);
        currentArrow.Fire(Bow.right * lastTensionForce);
        currentArrow = null;
     }
 else
     {Debug.LogWarning("No arrow to fire or bow not tensioned!");
     }
  }
    public void SetCurrentArrow(ArrowController arrow)
  {currentArrow = arrow;
   Debug.Log("Arrow set as current in BowStringController!");
  }
}
```

ButtonVR;

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.Events;
public class ButtonVR : MonoBehaviour
{public GameObject button;
   public UnityEvent onPress;
   public UnityEvent onRelease;
   GameObject presser;
   AudioSource sound;
   bool isPressed;
   void Start()
   {sound = GetComponent<AudioSource>();
      isPressed = false;
   }
   private void OnTriggerEnter(Collider other)
   {if (!isPressed)
      {button.transform.localPosition = new Vector3(0, 0.187f, 0);
         presser  = other.gameObject;
         onPress.Invoke();
         sound.Play();
         isPressed = true;
      }
   }
   private void OnTriggerExit(Collider other)
   { button.transform.localPosition = new Vector3(0, 0.197f, 0);
         onRelease.Invoke();
         isPressed = false;}}
```

FireOnCube

```
using System.Collections;
using System.Collections.Generic;
using System.Net;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.XR.Interaction.Toolkit;
public class FireOnCube : MonoBehaviour
{
   public GameObject bullet;
   public Transform spawnPoint;
   public Transform EndPoint;
   public float fireSpeed = 10;
   public float fireInterval = 5.0f;
   public float positionOffsetRange = 0.5f;
   public float directionOffsetRange = 5f;
   private float timeSinceLastFire;
   public bool startFire=false;
   public int HP = 100;
   public Image healthBar;
   public TextMeshPro scoreText;
   void Start()
   {
    timeSinceLastFire = 0f;
   }
   void Update()
   {
      timeSinceLastFire += Time.deltaTime;
```

```
if (timeSinceLastFire >= fireInterval)
{
   if(startFire==true)
   {
      FireBullet();
      fireSpeed += 0.5f;
      timeSinceLastFire = 0f;
      if(fireInterval>1.5)
         fireInterval -= 0.1f;
   }
}
GameObject[] beatObjects = GameObject.FindGameObjectsWithTag("Beat");
foreach (GameObject obj in beatObjects)
{
   if (Mathf.Abs(obj.transform.position.z) < Mathf.Abs(EndPoint.position.z))
   {
                Destroy(obj);
      HP -= 10;
      healthBar.fillAmount = HP / 100f;
   }
}


if (HP <= 0)
{
   scoreText.text = "0";
   startFire = false;
   fireSpeed = 10;
   fireInterval = 5.0f;
   HP = 100;
   healthBar.fillAmount = HP / 100f;
```

```
    }
  }
  public void StartFire ()
  {
      startFire = true;
  }
  public void FireBullet()
  {
    Vector3 randomPositionOffset = new Vector3(
      Random.Range(-positionOffsetRange, positionOffsetRange),
      Random.Range(-positionOffsetRange, positionOffsetRange),
      Random.Range(-positionOffsetRange, positionOffsetRange)
    );
    Quaternion randomDirectionOffset = Quaternion.Euler(
      Random.Range(-directionOffsetRange, directionOffsetRange),
      Random.Range(-directionOffsetRange, directionOffsetRange),
      Random.Range(-directionOffsetRange, directionOffsetRange)
    );
    GameObject spawnedBullet = Instantiate(bullet);
    spawnedBullet.transform.position = spawnPoint.position + randomPositionOffset;
    spawnedBullet.transform.rotation = spawnPoint.rotation * randomDirectionOffset;
    spawnedBullet.GetComponent<Rigidbody>().velocity = spawnedBullet.transform.forward
* fireSpeed;
    spawnedBullet.tag = "Beat";
    Destroy(spawnedBullet, 10);   }}
```

GardenSpawner

```
using UnityEngine;
```

```
using UnityEngine.XR.Interaction.Toolkit;
public class GardenSpawner : MonoBehaviour
{ [Header("Префаб овоча")]
   public GameObject vegetablePrefab;
   [Header("Розмір грядки")]
   public int rows = 3;
   public int columns = 4;
   [Header("Відстань між овочами")]
   public float spacing = 1.5f;
   [Header("Початкова точка")]
   public Transform startPoint;
   [Header("Заглиблення овоча в землю")]
   public float undergroundOffsetY = -0.3f;
   [Header("Поворот овоча при спавні (Euler)")]
   public Vector3 spawnRotationEuler = new Vector3(-90f, 0f, 0f);
   void Start()
   {if (vegetablePrefab == null)
      {Debug.LogError("Префаб овоча не призначено!");
         return;
      }
      if (startPoint == null)
      {
         startPoint = transform;
      }SpawnGarden();
   }void SpawnGarden()
   {for (int x = 0; x < columns; x++)
      {for (int z = 0; z < rows; z++)
         {Vector3 spawnPos = startPoint.position + new Vector3(x * spacing,
undergroundOffsetY, z * spacing);
            Quaternion rotation = Quaternion.Euler(spawnRotationEuler);
```

```
            GameObject vegetable = Instantiate(vegetablePrefab, spawnPos, rotation, transform);
            SetupVegetable(vegetable);
        }
    }
  }
  void SetupVegetable(GameObject vegetable)
  {
    Rigidbody rb = vegetable.GetComponent<Rigidbody>();
    if (rb == null)
    {
      rb = vegetable.AddComponent<Rigidbody>();
    }
    rb.isKinematic = true;
    XRGrabInteractable grab = vegetable.GetComponent<XRGrabInteractable>();
    if (grab == null)
    {
      grab = vegetable.AddComponent<XRGrabInteractable>();
    }
    grab.selectEntered.AddListener((args) =>
    {Vector3 pos = vegetable.transform.position;
      pos.y = 0f; // Піднімаємо овоч на поверхню
      vegetable.transform.position = pos;
    });
  }
}
```

InventoryVR

```
using UnityEngine;
using UnityEngine.InputSystem;
```

```
public class InventoryVR : MonoBehaviour
{public GameObject Inventory;
   public GameObject Anchor;
   public InputActionReference toggleInventoryAction;
   private bool UIActive;
   private void OnEnable()
   {toggleInventoryAction.action.performed += OnToggleInventory;
      toggleInventoryAction.action.Enable(); }
   private void OnDisable()
   { toggleInventoryAction.action.performed -= OnToggleInventory;
      toggleInventoryAction.action.Disable();
   }
   private void Start()
   {Inventory.SetActive(false);
      UIActive = false;
      Inventory.transform.SetParent(Anchor.transform, false);
   }
   private void Update()
   {if (UIActive)
      {Inventory.transform.localPosition = new Vector3(0f, 0.5f, 0f);
         Inventory.transform.localEulerAngles = new Vector3(2f, 0f, 0f);
      }
   }
   private void OnToggleInventory(InputAction.CallbackContext context)
   {UIActive = !UIActive;
      Inventory.SetActive(UIActive); }}
```

itemType

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;
[RequireComponent(typeof(XRGrabInteractable))]
public class Item : MonoBehaviour
{public string itemType;
  [HideInInspector] public bool inSlot = false;
  [HideInInspector] public Slot currentSlot;
  public Vector3 slotRotation = Vector3.zero;
  private XRGrabInteractable grabInteractable;
  private Rigidbody rb;
  private void Awake()
  {  grabInteractable = GetComponent<XRGrabInteractable>();
     rb = GetComponent<Rigidbody>();
  }
  private void OnEnable()
  {grabInteractable.selectEntered.AddListener(OnGrab);
     grabInteractable.selectExited.AddListener(OnRelease);
  }
  private void OnDisable()
  {grabInteractable.selectEntered.RemoveListener(OnGrab);
     grabInteractable.selectExited.RemoveListener(OnRelease);
  }
  private void OnGrab(SelectEnterEventArgs args)
  {if (inSlot && currentSlot != null)
     {currentSlot.RemoveOneItemFromStack(this);
        inSlot = false;
        currentSlot = null;
     }
```

```
    DetachAndEnablePhysics();
  }
  private void OnRelease(SelectExitEventArgs args)
  {DetachAndEnablePhysics();
  }
  private void DetachAndEnablePhysics()
  {transform.SetParent(null, true);
    if (rb != null)
    {rb.isKinematic = false;
      rb.WakeUp();
    } }}
```

NextScene

```
using UnityEngine;
using UnityEngine.SceneManagement; // потрібно для завантаження сцен
public class NextScene : MonoBehaviour
{
  [Header("Назва або індекс наступної сцени")]
  public string sceneName;
  public int sceneIndex = -1;
  private void OnTriggerEnter(Collider other)
  {
    // Перевіряємо, що гравець увійшов у тригер
    if (other.CompareTag("player"))
    {
      Debug.Log("Гравець увійшов у зону переходу");

      if (!string.IsNullOrEmpty(sceneName))
```

```
        {
            SceneManager.LoadScene(sceneName);
        }
        else if (sceneIndex >= 0)
        {
            SceneManager.LoadScene(sceneIndex);
        }
        else
        {
            Debug.LogWarning("Не вказано сцену для завантаження!");
        }
    }
}}
```

SliceObject

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using EzySlice;
using TMPro;
using System.Net;
public class SliceObject : MonoBehaviour
{
    public Transform startSlicePoint;
    public Transform endSlicePoint;
    public VelocityEstimator velocityEstimator;
    public LayerMask sliceableLayer;
    public TextMeshPro scoreText;
```

```
    private int scoreCounter=0;
    public Material crossSectionMaterial;
    public float cutForce = 2000;
    // Start is called before the first frame update
    void Start()
    {
        if(int.TryParse(scoreText.text, out int initialScore))
        {scoreCounter = initialScore;
        }
        else
        {scoreCounter = 0;
            Debug.LogWarning("Failed to parse scoreText. Initialized scoreCounter to 0.");
        }
    }
    void FixedUpdate()
    {bool hasHit = Physics.Linecast(startSlicePoint.position, endSlicePoint.position,out
RaycastHit hit, sliceableLayer);
        if (hasHit)
        {GameObject target = hit.transform.gameObject;
            Slice(target);
        }
        else
        {
            ///scoreCounter = 0;
            Debug.LogWarning("Failed to parse scoreText. Initialized scoreCounter to 0.");
        }
    }


    public void Slice(GameObject target)
    {
```

```
    Vector3 velocity = velocityEstimator.GetVelocityEstimate();
    Vector3 planeNormal= Vector3.Cross(endSlicePoint.position - startSlicePoint.position,
velocity);
    planeNormal.Normalize();
    SlicedHull hull = target.Slice(endSlicePoint.position, planeNormal);
    if (hull != null)
    {GameObject upperHull = hull.CreateUpperHull(target,crossSectionMaterial);
      SetupSliceComponent(upperHull);
      GameObject lowerHull = hull.CreateLowerHull(target, crossSectionMaterial);
      SetupSliceComponent(lowerHull);
      Destroy(target);
      if (target.CompareTag("Beat"))
      {scoreCounter++;
        scoreText.text = scoreCounter.ToString();
      }
      Destroy(upperHull, 20f);
      Destroy(lowerHull, 20f);
    }
  }
  public void SetupSliceComponent(GameObject slicedObject)
  {Rigidbody rb = slicedObject.AddComponent<Rigidbody>();
    MeshCollider collider = slicedObject.AddComponent<MeshCollider>();
    collider.convex = true;
    rb.AddExplosionForce(cutForce, slicedObject.transform.position, 1); }}
```

Slot

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.XR.Interaction.Toolkit;
using System.Collections.Generic;
```

```
using TMPro;
public class Slot : MonoBehaviour
{
   private List<GameObject> itemsInSlot = new List<GameObject>();

   [Header("UI References")]
   public Image slotImage;
   public TMP_Text countText;

   private Color originalColor;
   private HashSet<Item> blockedItems = new HashSet<Item>();
   void Start()
   {
      slotImage = GetComponentInChildren<Image>();
      originalColor = slotImage.color;
      UpdateCountUI();
   }

   private void UpdateCountUI()
   {
      if (itemsInSlot.Count <= 1)
         countText.text = string.Empty;
      else
         countText.text = itemsInSlot.Count.ToString();
   }
   private void OnTriggerStay(Collider other)
   {
      var obj = other.gameObject;
      var item = obj.GetComponent<Item>();
      if (item == null || item.inSlot) return;
```

```
    // Якщо itemType порожній => не стекати більше одного
    bool isNonStackable = string.IsNullOrEmpty(item.itemType);
    // Якщо вже є предмети в слоті
    if (itemsInSlot.Count > 0)
    {
        var firstItem = itemsInSlot[0].GetComponent<Item>();
        bool firstNonStackable = string.IsNullOrEmpty(firstItem.itemType);
        if (firstNonStackable || isNonStackable || item.itemType != firstItem.itemType)
            return;
    }
    if (blockedItems.Contains(item)) return;
    var grab = obj.GetComponent<XRGrabInteractable>();
    if (grab.isSelected) return;
    InsertItem(obj);
}

private void OnTriggerExit(Collider other)
{
    var item = other.GetComponent<Item>();
    if (item != null)
        blockedItems.Remove(item);
}

void InsertItem(GameObject obj)
{
    // 1) Grab the item's current *world* scale
    Vector3 worldScale = obj.transform.lossyScale;

    // 2) Attach it
    obj.transform.SetParent(transform, worldPositionStays: false);
```

```
    // 3) Reset position & rotation
    obj.transform.localPosition = Vector3.zero;
    obj.transform.localEulerAngles = obj.GetComponent<Item>().slotRotation;

    // 4) Compute the proper *local* scale so that:
    //    (localScale * parent.lossyScale) == worldScale
    var parentScale = transform.lossyScale;
    obj.transform.localScale = new Vector3(
        worldScale.x / parentScale.x,
        worldScale.y / parentScale.y,
        worldScale.z / parentScale.z
    );
    var rb = obj.GetComponent<Rigidbody>();
    rb.isKinematic = true;
    var itemComp = obj.GetComponent<Item>();
    itemComp.inSlot = true;
    itemComp.currentSlot = this;
    itemsInSlot.Add(obj);
    slotImage.color = Color.gray;
    UpdateCountUI();
}
public void RemoveOneItemFromStack(Item item)
{
    if (!itemsInSlot.Contains(item.gameObject)) return;
    itemsInSlot.Remove(item.gameObject);
    blockedItems.Add(item);
    if (itemsInSlot.Count == 0)
        slotImage.color = originalColor;
    UpdateCountUI();
}
```

```
}
```

TargetMover

```
using UnityEngine;


public class TargetMover : MonoBehaviour
{
   [Header("Movement Settings")]
   public float amplitude = 0.5f;
   public float speed = 1.5f;
   [Header("Respawn Settings")]
   public float fallDuration = 3f;
   public float respawnDelay = 2f;
   private Vector3 startPos;
   private Quaternion startRot;
   private Rigidbody rb;
   private bool isHit = false;
   private float timer;
   private void Start()
   {
      startPos = transform.position;
      startRot = transform.rotation;
      rb = GetComponent<Rigidbody>();
      if (rb == null)
      {
         rb = gameObject.AddComponent<Rigidbody>();
      }
      rb.isKinematic = true;
      rb.useGravity = true;
```

```
    rb.constraints = RigidbodyConstraints.FreezeRotationX |
RigidbodyConstraints.FreezeRotationZ;
  }
  private void Update()
  {
    if (!isHit)
    {
      float offsetY = Mathf.Sin(Time.time * speed) * amplitude;
      transform.position = startPos + new Vector3(0f, offsetY, 0f);
    }
    else
    {
      timer += Time.deltaTime;
      if (timer >= fallDuration + respawnDelay)
      {
        ResetTarget();
      }
    }
  }
  public void OnHit()
  {
    if (isHit) return;
    isHit = true;
    timer = 0f;

    rb.isKinematic = false;
    rb.constraints = RigidbodyConstraints.None; / /  Дозволяємо всі обертання
    rb.angularVelocity = Vector3.zero;
    rb.velocity = Vector3.zero;
```

```
        Vector3 forceDir = (-transform.forward + Vector3.down).normalized;
        rb.AddForce(forceDir * 5f, ForceMode.Impulse);
    }

    private void ResetTarget()
    {
        isHit = false;

        rb.isKinematic = true;
        rb.velocity = Vector3.zero;
        rb.angularVelocity = Vector3.zero;

        rb.constraints = RigidbodyConstraints.FreezeRotationX |
RigidbodyConstraints.FreezeRotationZ;

        transform.position = startPos;
        transform.rotation = startRot;
    }
}
```