Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ГРИ «TSOKOTARA» З ВИКОРИСТАННЯМ UNITY З ІМПЛЕМЕНТАЦІЄЮ АСИМЕТРИЧНОГО VR КООПЕРАТИВУ

DEVELOPMENT OF THE GAME "TSOKOTARA" USING UNITY WITH ASYMMETRIC VR CO-OP IMPLEMENTATION

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗс-21
Вахрамеєв Богдан Вячеславович

Керівник:
Бойко Лев Степанович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Вахрамєєву Богдану Вячеславовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка гри «Tsokotara» з використанням Unity з імплементацією асиметричного VR кооперативу»*
Керівник роботи: *доцент Бойко Л. С.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *ігровий рушій Unity, методичні вказівки до виконання кваліфікаційної роботи бакалавра*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз та огляд сучасного стану проблеми, формулювання вимог і архітектури проекту та патернів розробки, вибір інструментів розробки, розробка гри, тестування роботи і функціональності проекту, дослідження результатів проведених тестувань та аналіз шляхів розвитку та рекомендації для вдосконалення проекту.*
5. Перелік графічного матеріалу: *6 рисунків, 8 таблиць, 3 додатки*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Бойко Л. С.*		
Специфікація вимог до розробленої системи	*Бойко Л. С.*		
Розробка об'єкта проектування	*Бойко Л. С.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	0,70 %		
Академічна доброчесність	*Бойко Л. С.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Богдан ВАХРАМЕЄВ

Керівник кваліфікаційної роботи _____ Лев БОЙКО

АНОТАЦІЯ

Вахрамеєв Б. В. Розробка гри «Tsokotara» з використанням Unity з імплементацією асиметричного VR кооперативу. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз ринку відеоігор за жанровою належністю до платформерів, Hack'N'Slash, метроідванія та асиметричний VR кооператив і сформульовано завдання. В другому розділі проведено дослідження проблематики предмету роботи, які включають у себе адаптивність схем керування та визначення вимог до архітектури системи. У третьому розділі здійсено опис практичної реалізації проекту гри, з поясненням процесу створення елементів для гри, а також оглянуто результати тестування гри групою користувачів. У висновках підсумовано результати досліження процесу розробки гри на Unity, роботу що була проведена під час роботи та описано перспективи розвитку проекту у майбутньому.

Ключові слова: Unity, VR, Unreal Engine, Godot, GameMaker, Bevy, Krita, Blender, гра на Unity, асиметричний VR кооператив, Hack'N'Slash, платформер, метроідванія.

ABSTRACT

Vakhramieiev B. Development of the Game "Tsokotara" Using Unity with Asymmetric VR Co-op ImplementationManuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents presents an analysis of the video game market by genre, focusing on platformers, Hack’N’Slash, Metroidvania, and asymmetric VR co-op, and formulates the objectives. The second chapter explores the subject-related issues, including the adaptability of control schemes and the definition of system architecture requirements. The third chapter describes the practical implementation of the game project, explains the process of creating game elements, and reviews the results of game testing by a user group . The conclusions summarize the results of the game development process in Unity, the work carried out during the study, and outline the prospects for future project development .

Keywords: Unity, VR, Unreal Engine, Godot, GameMaker, Bevy, Krita, Blender, Unity game, asymmetric VR co-op, Hack’N’Slash, platformer, Metroidvania.

ЗМІСТ

ВСТУП

Актуальність теми. Індустрія відеоігор стрімко розвивається, щорічно пропонуючи гравцям нові формати взаємодії та ігрового досвіду. Одним із трендів останніх років є віртуальна реальність (VR), який забезпечує глибоке занурення в процес гри, пропонуючи найбільш інтерактивну та живу взаємодію з віртуальним простором. Завдяки розвитку цієї технології, однією перспективною тенденцією постав асиметричний кооператив форма багатокористувацької взаємодії, де гравці виконують різні ролі у грі та мають відмінні ігрові можливості. Поєднання ігрового процесу VR та традиційних методів керування, відкриває нові горизонти у проєктуванні пригодницьких ігор, зокрема класичних жанрів, накшталт як метроїдванія та 3D платформер. Розробка гри за такими рішеннями дозволяє не лише створити інноваційний досвід гри, а й дослідити нові підходи до геймдизайну, UI/UX, оптимізації VR-систем і взаємодії між гравцями настільки різних платформ.

Метою даної кваліфікаційної роботи є розробка пригодницької 3D гри з елементами метроїдванії, Hack’n’Slash і платформера, та імплементацією асиметричного VR кооперативу на рушії Unity.

Об’єктом роботи є процес створення інтерактивних програмних продуктів, зокрема відеоігор.

Предметом роботи є технології, інструменти та підходи до реалізації пригодницької гри з асиметричним кооперативом у VR-середовищі.

Головними завданнями кваліфікаційної роботи постає проектування та розробка пригодницької гри на рушії Unity. Це завдання передбачає:

– аналіз ринку ігор, зокрема жанрів 3D платформер, Hack’N’Slash, метроїванія та асиметричний VR кооператив. Аналіз інструментів для розробки;

– аналіз та розробка схеми керування;

– розробка архітектури проекту;

– створення систем, необхідних для функціоналу гри. Створення геймплейних механік, ігрових предметів;

– створення 3D моделей, їх оптимізація, імпорт та накладення текстур;

– реалізація ігрового UI/UX, створення ігрових меню та написання логіки переходу між меню;

– проведення плейтестів, виправлення виявлених багів, завершення циклу розробки проекту.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У наш час, індустрія відеоігор пропонує широкий вибір засобів для самореалізації, будучи однією з найбільш креативних індустрій у світі сьогодні. Щодня появляються та знаходять свій успіх як представники проектів ААА рівня (проекти що розроблені студіями, які залучають до своїх проектів великі грошові інвестиції), так і ентузіасти початкового рівня, що презентують самобутні та незалежні проекти. Завдяки стірмкому технологічному прогресу, підвищенню доступності інструментів розробки а також комерційному успіху невимірної кількості інді проектів за останні десятиліття, ринок ігрової індустрії активно насичують новачки, що вивчають, розробляють і впроваджують рішення, які раніше могли використовувати в більшості лише великі гравці індустрії. Інді розробники сьогодні навчаються досвіду як таких же ентузіастів свого часу, так і досвіду великих корпоративних проектів, переймаючи інструменти та знання.

Левовою частиною сфери однокористувацьких ігор в наш час є різного роду ігри які передбачають проходження певної прогресії дій. Логічний розподіл таких ігор можна провести за різними жанрами, накшталт платформер, RPG, шутер, Beat’em’up, puzzle і тд. Самі жанри в першу чергу визначаються за такими аспектами:

– основні механіки, керування гравцем або ігровими юнітами, головні механіки залучення гравця, баланс механік згідно ігрового процесу;

– керівні системи гри, по суті ігрові правила в класичному розумінні, у поєднанні з автоматичним ігровим «ведучим»;

– основні візуальні рішення, ракурси камери, центральні ідеї естетичних рішень та ігровий сеттинг;

– архітектура гри, глобальні рішення геймдизайну, структура побудови ігрових рівнів, адаптація пристроїв керування.

Напевне одним з найпопулярніших видів синглплеєрних ігор сьогодні є пригодницькі ігри, оскільки глобально вони поєднують у собі широкий спектр жанрових елементів, так за останні декілька років можна спостерігати рух трендів індустрії в напрямку Souls like новоявленому жанру ігор, головні аспекти якого походять від серії ігор Dark Souls. Проте найбільш визначним жанром у контексті формування різноманітних механік бойової системи у таких іграх є Hack'N'Slash. Цей жанр є еталонним показником комплексної та цікавої системи механік. Найбільш надихаючими для нашого проекту стали ігри:

– NieR:Automata [18] це гра від японської ігрової студії PlatinumGames у співробітництві з видавництвом Square Enix, за авторством Йоко Таро. Гра оповідає зворушливу філософську історію, супроводжуючи її геймплеєм, побудованим еталонною в жанрі слешерів студією;

– Devil May Cry [7] серія ігор за авторством Хідекі Камія, розробляється та видається компанією Capcom. Ігри серії славляться візуальним стилем, харизматичною історією та головне комплексною та цікавою бойовою системою, що у свій час стала революцією в жанрі та продовжує бути одним з найяскравішим прикладом для наслідування;

– V.A.Proxy [24] гра у розробці інді-команди PyroLith, що завдяки незвичайному візуальному стилю та зразковому ігровому процесу у жагрі, отримала доволі високу популярність ще до свого повноцінного релізу.

Одним з жанрів що в найбільшій мірі зробив власний вклад у механіки переміщення у іграх сьогодні, є платформер, ігри якого історично славляться механіками та засобами дослідження ігрових рівнів. Ці ігри чудово себе почувають як у двовимірному, так і трьохвимірному середовищі завдяки гнучкості визначних аспектів жанру. З цих ігор, найбільш надихаючими для проекту стали:

– Marvel's Spider-Man [13] це гра розроблена студією Insomniac Games, під видавництвом Sony Interactive Entertainment, у свій час стала

прикладом як хорошої гри про бої, так і напевне найкращим в жанрі на момент виходу прикладом втілення механік подорожей відкритим світом гри;

– Spark the Electric Jester 3 [17] цей нішевий інді-проект розроблений командою Feperd Games, наслідує канонічному шляху швидкісного платформеру, основу якого у свій час заклала Sonic Adventure. Проект реалізує усі основні аспекти класичної гри в обгортці нового якісного проекту;

– The Big Catch [19] проект авторства Filet Group, це ще один проект від інді-команди, що знайшов своє визнання перед тим як вийти у повноцінний реліз, що сталось завдяки яскравому візуальному стилю та унікальним ігровим механікам що будуються навколо вудки головного персонажа гри.

З усіх жанрів що фокусуються на створенні найбільш цікавого ігрового світу, метроїдванія більше усього ставить за ціль побудову переплетеного та унікального світу, дослідження якого розкривається по частинам з отриманням різних нових ігрових інструментів, які залучають гравця до дослідження усіх закутків ігрового світу, планомірно надаючи гравцеві доступ до все більшої кількості ігрових механік та сильно заохочуючи backtracking (дослідження проходів чи локацій, повз які гравець проходив раніше). Цей жанр надихає до створення якісного ігрового світу, і найбільшими його представниками є:

– Hollow Knight [10] проект незалежної студії Team Cherry, у рік свого виходу став свіжим ковтком для жанру, ставши мастадонтом і однією з перших асоціацій у контексті розмови про метроїдванії;

– Batman: Arkham series [1] серія ігор Rocksteady Studios, видавництва WB Games (підрозділ Warner Bros. що володіють DC Comics, які є правовласниками франшизи). Ці ігри стали прикладом побудови ігрового світу, який є надбудовою для захоплюючого ігрового процесу та історії;

– Pseudoregalia [16] проект незалежної команди розробників rittzler, що став популярним представником жанру 3D платформерів у інді сегменті, проте славиться також структурою світу що заоохочує гравця досліджувати локації детальніше.

Напевне найбільш інтерактивною та імерсивною (занурюючою) нішею сучасного ринку відеоігор є VR проекти. Будучи доволі самодостатньою нішею ігор, VR проекти пропонують ще більш цікавий досвід дослідження ігрового світу у компанії з гравцями що приєднуються до ігрових сесій з традиційних ігрових платформ. Ця ідея стала ядром жанру асиметричних VR ігор, даруючи гравцям і розробникам сферу для ігор як PVP (гравець проти гравця) так і Co-op (взаємодія в команді проти комп'ютерних ворогів в одиночній грі), які поєднують два різні ігрові досвіди. Найбільш цікавими джерелами для натхнення стали:

– Carly and the Reaperman [6] гра розроблена командою Odd Raven Studios, є найбльшим прикладом досвіду що змушує гравця у VR обладнанні співпрацювати з гравцем що грає з класичного устаткування, створюючи унікальні ігрові ситуації;

– Panoptic [14] розроблений Team Panoptes, проект пропонує досвід ігрової конфронтації, де гравець з класичним керуванням мусить уникати погляду гігантського аватару гравця у VR шоломі, з ціллю покинути локацію неспійманим.

Визначення жанру, це рішення що на етапах початку розробки супроводжується вибором інструментів реалізації проекту, і в середовищі без напрацьованого фреймворку постає питання вибору інструментів:

– який обрати рушій? Серед найбільш доступних та якісних варіантів на ринку доступні Unity, Unreal Engine та Godot;

– в якому середовищі розробляти 3D asset'и? Обрати можна наприклад стандарт індустрії 3DS Max на комерційній основі, або почати роботу у популярному та прогресучому Blender на базі open source;

– де текстури малювати? Тут вибір стоїть між Photoshop, Krita, та безліччю інших графічних рішень;

– де обробляти звук? Таку змогу надають Audacity, FL Studio та чимало інших рішень.

Серед рішень ігрових рушіїв, у той час як розробка на Unreal Engine нативно направляє фокус на найкращі графічні рішення на ринку, Godot будучи open source продуктом надає можливість глибшого рівня кастомізації, проте з значно меншою кількістю загальних інструментів для розробки, Unity пропонує баланс цих рішень, надаючи як потужні інструменти для створення якісної графіки, так і високий рівень гнучкості робочого процесу у поєднанні з великою кількістю інструментів безпосередньо у рушії та у вигляді додаткових package рішень. І попри контраверсійну ситуацію через зміну економічної політики Unity у 2023 році, рушій зберігає статус одного з найбільш популярних і якісних рішень для розробки ігор.

Unity серед усіх наявних інструментів, є найбільш цікавим рішенням в нашому контексті оскільки містить в собі потужний інструментар що дозволяє створювати проекти для більшості сучасних платформ, завдяки нативній підтримці більшості сучасних графічних API, таких як OpenGL або Vulkan, а також якісними рішеннями редактора, що дозволяють адаптувати гру для різних схем керування без жодних проблем [23].

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Головним завданням кваліфікаційної роботи постає проектування та розробка пригодницької гри на рушії Unity. Це завдання передбачає:

– аналіз ринку ігор, зокрема жанрів 3D платформер, Hack’N’Slash, метроїванія та асиметричний VR кооператив. Аналіз інструментів для розробки;

– аналіз та розробка схеми керування;

– розробка архітектури проекту;

– створення систем, необхідних для функціоналу гри. Створення геймплейних механік, ігрових предметів;

– створення 3D моделей, їх оптимізація, імпорт та накладення текстур;

– реалізація ігрового UI/UX, створення ігрових меню та написання логіки переходу між меню;

– проведення плейтестів, виправлення виявлених багів, завершення циклу розробки проекту.

Побудова цього завдання передбачала аналіз прикладів стандартизованих фреймворків в індустрії, планування проекту заздалегідь, базуючись на існуючій ідеї створення такого проекту. Даний план є найбільш логічним у послідовності реалізації проекту гри, адже завершення кожного попереднього його кроку підштовхує прогрес наступного.

Висновки до розділу 1

Було здійснено поверхневий аналіз ринку відеоігор, зокрема увагу загалом отримали проекти суміжної з моїм проектом жанрової належності. На огляді було взято ігри, які власними рішеннями стали джерелом для натхнення та/або прикладом для наслідування.

Жанри виду пригодницьких ігор що було оглянуто, це ті жанри та проекти, які фокусують увагу під час ігрового процесу на власні вигідно виразні механіки, і напрямком для мого проекту було обрано наслідування та напрацювання саме на засадах цих механік у їх найкращих проявленнях. Так, створюючи систему взаємодій з ігровим світом, зокрема бойову систему, сильний ухил при проектуванні гри покладався саме на аспекти притаманні іграм жанру Hack’N’Slash. Розробка ж рішень щодо організації переміщення персонажа ігровим світом, покладається на принципи побудови платформерів. А проектування ігрового світу, для кращого залучення гравця до повторного дослідження пройдених локацій та в цілому створення цікавого користувацького досвіду, покладається на філософію жанру metroidvania. Стосовно елементів для VR гравців, геймдизайн будується навколо ідеї супроводження та підтримки гравця за традиційною схемою керування, що є стержнем молодого жанру VR ігор з асиметричним кооперативом.

Також на завершення аналізу ринку, було проведено коротку звірку з обранимим інструментами для розробки, що допоможе нам на наступних етапах з формуванням власного фреймворку. Порівняння та детальніший огляд самих інструментів описано в наступних розділах.

На основі аналізу цих рішень, було сформовано мету розробки. Було окреслено завдання для розробки рішення, які містять в собі огляд аналогів рішень та інструментів, вибір інструментів для стеку проекту, закладено план проекту, окреслено аспекти передбаченні як під час так і після побудови проекту.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Після проведення аналізу ринку, а також проектування власної розробки, було прийнято велику кількість рішень, які в результаті комбінуються у архітектуру проекту. Будь-яка система у цифровому просторі вимагає створення логічної послідовності алгоритмів, які в результаті зможуть видавати результат отримуючи лише дані у зручному форматі. Проте якщо система нашого проекту і потребує додавання нових механік чи алгоритмів, архітектура проекту мусить бути максимально зручною та відкритою до доповнення, що досягається наприклад додаванням коду до уже існуючих функцій у вигляді наслідуючих класів [2].

Також при розробці проекту важливо врахувати аспекти пристосованості схем керування грою, що дозволяє легко адаптувати гру для різних девайсів. Найбільш простим методом для усуспільнення різних елементів різного призначення є абстракція, наприклад натиснення символу квадрат на ігровому контролері Dualshock або Dualsense та натиснення лівої клавіші миші у грі мусять репрезентувати одну і ту саму дію, це важливо запам’ятати при створренні схем керування та при написанні ігрових алгоритмів.

Важливо також використовувати методи абстракції і при створенні алгоритмів спільної дії для різних об’єктів. Об’єкт що є персонажем гравця використовує певну логіку для обробки дій відповідно до оточення об’єкта та натиснутих клавіш. Логіка за якою обробляється дія гравця може легко використовуватись і для об’єктів НІПів (не ігрових персонажів – NPC), а визначення натиснутої клавіші можна зробити абстарктним значенням алгоритму дії «головна дія», «другорядна дія», «дія скасування» і тд [22].

При проведені аналізу, було визначено і складено схему керування, і вибором для прототипу стали девайси керування: Dualshock 4 (рис 2.1),

класична клавіатура з мишею. Обрано саме ці девайси саме тому що вони є найбільш популярними традиційними методами керування на ринку, і при потребі адаптації схеми для інших пристроїв, наприклад геймпаду від Xbox, можна налаштувати уже готову схему що використовується для Dualshock. Контролери Sony та Microsoft Xbox мають дуже схожу структуру, і в цілому за виключенням розташування деяких елементів та символіки що використовують їх кнопки, є по суті ідентичними девайсами.

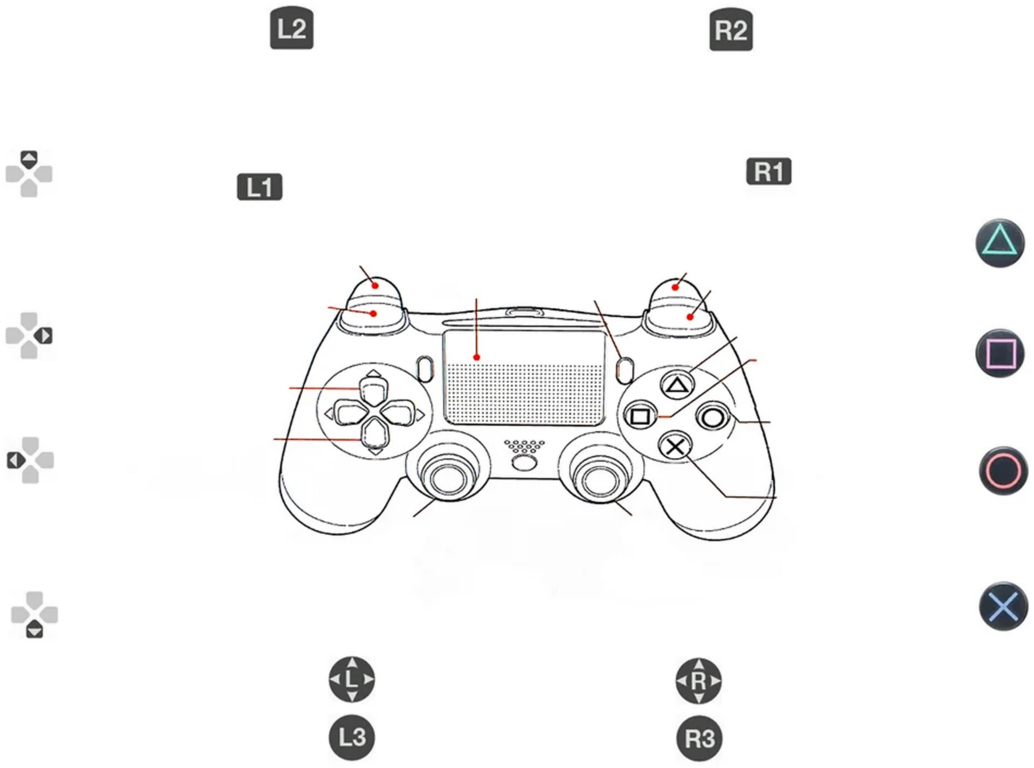

Рисунок 2.1 – Схема керування Dualshock 4 за результатом пошуку [8]

У проекті передбачено можливість додавання та введення до 3х клавіш дії одночасно, і ці клавіші для усіх девайсів мають таку абстрактну розкладку:

– MainAction – головна дія, у грі це «легка атака» або «головна дія» залежно від контексту предмету у руках персонажа. На геймпаді це зазвичай кнопка з символом квадрату, а на схемі комп’ютерного керування це ліва кнопка миші;

– HeavyAction – другорядна дія, зазвичай це «важка атака» або «другорядна дія» залежно від контексту предмета. На геймпаді це клавіша з символом трикутника, в той час як на комп’ютері це права кнопка миші;

– CancelAction – дія скасування, у іграх це зазвийчай щось накшталт «захвату», проте залежно від контексту гри це третьорядна дія. У класичній схемі керування Dualshock, зазвичай ця кнопка в меню використовується в якості кнопки для дії «скасування», що зазвичай здійснює повернення в попереднє меню або закриття наявного меню. На клавіатурі ця дія встановлена на кнопку колесика миші або клавішу E;

– JumpAction – дія стрибок, відповідно до назви ця дія в класичній схемі керування дозволяє здійснювати стрибок або схожу по контексту дію. На схемі контролера Sony це зазвичай кнопка з символом X. Для клавіатури ж це пробіл;

– ShootAction – дія постріл, це дія що в контексті нашого проекту використовується як дія для здійснення пострілу (ідея для цього була взята з схеми керування поширеної у іграх Platinum Games, зокрема їх проект «Transformers: Devastation»). На геймпаді це клавіша R1 (права кнопка). На клавіатурі обрано клавішу Q;

– TargetAction – дія прицілення, відповідно використовується в грі для прицілення зі зброї у другій руці, або наприклад для блокування шкоди щитом у другій руці (контекст передбачає що ця клавіша має бути використана для взаємодії з предметом у другій руці, яка вимагає затримання клавіші). На геймпаді для цього обрано клавішу L2 (лівий тригер). На клавіатурі обрано клавішу C, оскільки для ігор у яких передбачено використання певного роду дії приближення для погляду на об’єкти в далечінь, саме ця клавіша використовується, і в контексті прицілювання вона має схоже призначення;

– DodgeAction – дія ухилення, яка в іграх зазвичай репрезентує маневр накшталт перекид або ривок, що допомагає персонажу ухилитись від атаки або приблизитись до бажаної точки. На контролері зазвичай це роль

клавіші R2. На клавіатурі це клавіша лівий Alt або Ctrl, залежно від контексту використання та зручності;

– UltAction – спеціальна дія, вона зазвичай використовуються в іграх у якості якоїсь спеціальної атаки, або певної другорядної дії. На Dualshock це кнопка L1, а на клавіатурі клавіша F.

Решту клавіш, зокрема для вибору предмету, призначено на хрестовину геймпаду, крутіння колесика миші та ряд цифр на клавіатурі. Відкриття меню паузи на геймпаді це мала клавіша зправа від сенсорної панелі, та Esc на клавіатурі. Відкриття інвентарю та суміжних GUI елементів здійснюється малою кнопкою зліва від сенсорної панелі та натиском на саму сенсорну панель на геймпаді, клавіші Tab та інші на клавіатурі, оскільки користуючись клавіатурою зручніше перемикати вікна меню GUI.

Відповідно використовуючи описані абстрактні дії, ми в подальшому формуємо дії та рухи що персонаж у грі може виконати за умови натискання комбінації клавіш, такий підхід до систем взаємодії, зокрема бойової системи гри називають системою Combo, від слова combination – виконання атак у певній послідовності, з натисненням певних клавіш, з дотриманням ритму гри. Така система є основою майже усіх ігор жанру Hack’N’Slash що є одним з ідейних натхень для бойової системи проекту.

Після формування схеми керування, найбільш комплексним завданням постає архітектура проекту. Побудова якісної структури проекту вимагає логічної відповіді на питання:

1) яку дію виконує певний елемент;
2) чи можна використати один елемент двічі чи частіше;
3) чи можливо та чи просто додати до готового елементу нову логіку;
4) наскільки вимогливим до обчислювальних потужностей є елемент.

Відповіді на ці питання важливо знайти тому що від отриманих висновків можна визначити якість реалізації проекту. Від відповіді на перше питання ми розуміємо доцільність використання обраної частини архітектури, друге питання пояснює наскільки елемент дійсно інтегрований в контекст системи і

чи його зміна може вирішити проблему глобально. Зазвичай найлегше використовувати повторно один елемент якщо він репрезентує одну технічно однакову дію для двох різних об’єктів, доприкладу вважається недоцільним писати логіку для обрахування параметра «Здоров’я» об’єкта більше ніж один раз, адже в результаті обидва об’єкта однаково потребують показник здоров’я, і обидва об’єкта мусять виконати якусь дію коли здоров’я вичерпується. З вирішенням цього питання допомагають методи наслідування та композиції, перший метод дозволяє власним класам для унікальних об’єктів наслідувати функції та параметри «батьківського» класу, тоді як другий метод передбачає розбиття незалежних фукцій класу на декілька окремих класів (розподіл відповідальностей), та використання цих окремих класів на одному об’єкті. Відповідь на третє питання важливе в контексті не лише підтрики проекту, а і в цілому є демонстративним показником якості написаної логіки, адже scaleability (можливість до нарощення) є основним показником можливості проекту до збільшення кількості контенту. Четверте питання відповідає важливому аспекту будь-якого проекту навіть з елементарними ігровими алгоритмами, наскільки продуктивними є обчислення алгоритмів. Проекти в яких по певній причині алгоритми обчислюються повільно та непрактично створюють проблему з затримками (lag spikes) обчислень, а також допускають часто критичні помилки, і ці проблеми є одними з найбільш впливаючих факторів поганого користувацького досвіду. Відповідаючи на ці питання ми формуємо базу для архітектури нашого проекту, яку можна використовувати в майбутньому для виправлення помилок, підтримки та розширення проекту.

Взаємозв’язок компонентів комплексної системи, простіше всього продемонструвати за допомогою діаграми класів. Зокрема, діаграма класів (рис. 2.3) демонструє зпроектовану структуру для нашого проекту. Структура проекту була складена послуговуючись принципами логічної послідовності, розділення відповідальностей та наслідування повторюваних елементів. Створення такої архітектури дозволяє нам отримати у керування систему, що є не лише легкою у розумінні, а також є дуже модульним інструментом .

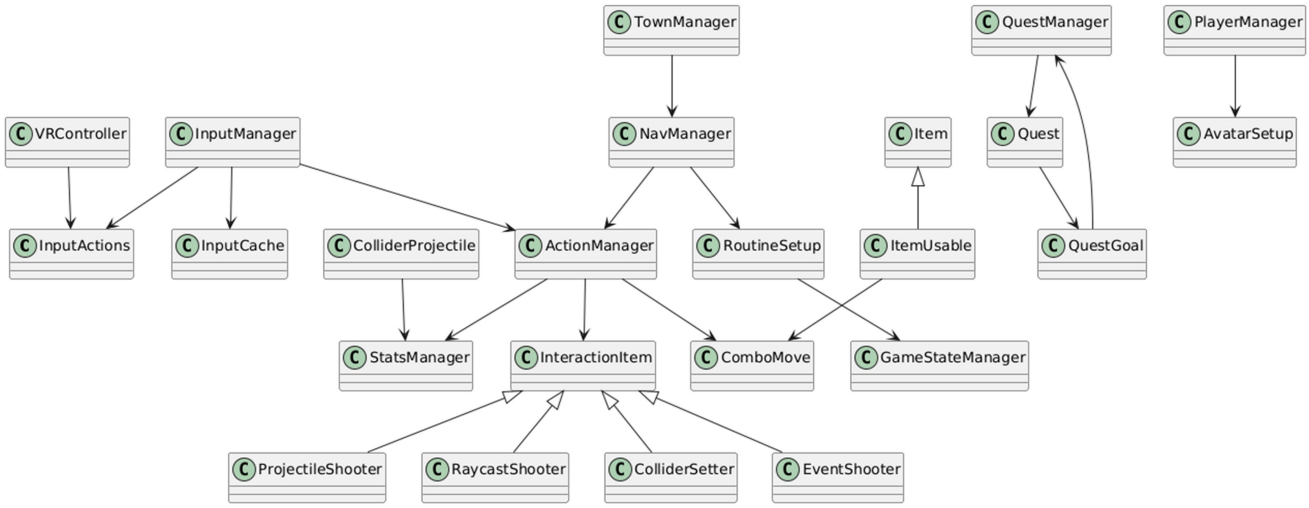

Рисунок 2.3 Діаграма класів архітектури гри

У додатку А представлено та описано ці класи, їх призначення, можливості для повторного використання, розширення та доповнення, а також продуктивність кожного класу у контексті використання. З таблиці виходить, що головним аспектом для повторного використання логіки є простота її алгоритмів, що гарно впливає на більшу кількість об’єктів що послуговуються логікою, а також її уніфікація що дозволяє використовувати аспекти або навіть увесь клас на дуже різних по своїй суті об’єктах [5, 15]. Розширення та доповнення логіки класу є можливим у тому випадку якщо описані алгоритми послуговуються усуспільненими параметрами, адже нова логіка може користуватись уже наявними параметрами і нам не потрібно будувати код для отримання недостатніх параметрів, а також базовий алгоритм в ідеалі мусить будуватись з розрахунком на те що його основою будуть користуватись наслідуючі класи. І головним чинником для визначення продуктивності є спільний з повторюваністю фактор більш простих алгоритмів на одному об’єкті, а також цикл викликів цього класу, адже наприклад ми не хочемо щоб складні алгоритми і операції для гри виконувались у коді циклічно, а тимбільше циклічно у декількох об’єктах одночасно.

Таким чином, проект передбачає реалізацію описаної архітектури систем, що потрібно для формування максимально простого у використанні

фреймворку який дозволяє буквально перетягувати необхідні asset’и та дата контейнери на кінцевий об’єкт, і зфокусуватись на написанні сценарію для гри та геймдизайні. Прийняття геймдизайнерських рішень зазвичай супроводжується документуванням ідей для рівнів, предметів, персонажів, сценарію та квестів. Цей документ допомагає організувати більш прямолінійний процес реалізації прийнятих рішень і запобігає проблемі коли прийняття рішень у ланцюгу збивається з потоку.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Ринок ігрових рушіїв сьогодні широкий як ніколи, адже власні рішення пропонують як Open Source ентузіасти, так і компанії які пропонують набір інструментів з різною комерційною моделлю. Так, при виборі рушія перед проектом постали такі вирішальні аспекти:

- наявність хорошого інструментарію;
- зручний фреймворк «з коробки»;
- якісна підтримка і розвиток від творців та власників рушія;
- популярна та хороша мова програмування.

Кандидатів у проекту було багато, проте обрати важливо одне рішення, адже залежно від того наскільки хорошим був початковий вибір, складається якісний та прямолінійний потік роботи над проектом. У порівняльній характеристиці було взято такі рушії: Unreal Engine 4/5, Unity, Godot, GameMaker та Bevy.

Unreal Engine це рушій що розробляється Epic Games приблизно останні 27 років (перша версія вийшла у травні 1998 року), і досі активно підтримується та комерційно розвивається. Головний фокус цього рушія це створення високобюджетних ігор з реалістичною графікою. Завдяки своєму потужному набору графічних інструментів таких як Lumen (система динамічного глобального освітлення) та Nanite (система віртуалізованої геометрії).

Unity це рушій що розвивається приблизно уже 20 років з моменту виходу, і напевно на ринку рушіїв немає зараз ресурсу який отримував би таку активну публічну підтримку як цей рушій. Головна ідея цього рушія «бути рушієм для інді» появилась не просто так. Рушій з самого початку моменту свого існування послуговується філософією максимальної доступності для усіх користувачів, його комерційна модель завжди будувалась на основі безкоштовного використання та системі royalty, тоді як Unreal змінив свою модель на схожу лише у травні 2015го року. Найбільш привабливими фішками Unity є його доступність, зручний інструментар та високий рівень підтримки користувачів та власників.

Godot це open source рушій що побачив світ у 2014 році. З самих початків фокусом Godot було створення 2D проектів, проте з роками розвитку рушій виріс в можливостях і тепер є привабливим рішенням для роботи над 3D проектами також. Головною фішкою Godot є його open source природа, що дозволяє будь-кому робити власні доповнення, і не просто кастомізувати рушій самостійно, а і також пропонувати власні доповнення до офіційної версії рушія [9].

GameMaker це один з найбільш технічно простих рушіїв для створення ігор, що робить його найкращим рішенням для початківців справи. Він пропонує доволі якісний інструментар для створення 2D проектів, та використовує дуже прості методи програмування. Його простота і є його фішкою у конкуренції з іншими.

Bevy це рішення що розробляється з 2020 року, є потужним open source інструментом. Оскільки проект є доволі молодим то на разі візуальний редактор досі знаходиться в розробці, що може відлякати стандартного користувача. Проте головна фішка Bevy сьогодні компенсує багато недоліків для досвідчених працівників у індустрії, адже архітектура Bevy буквально побудована для максимальної кастомізації фреймворку, кожен елемент в вашому проекті є повністю опціональним. Також головним інструментом рушія

є сам його фреймворк що в базі своїй націлений на ECS системи компонентів [3].

У таблиці 2.1. складено усуспільнену характеристику усіх рушіїв в контексті порівняння їх інструментарю, фреймворку, підтримки та мови програмування.

Таблиця 2.1 – Порівняння рушіїв за описаними критеріями

Рушій	Інструментарій	Фреймворк	Підтримка	Мова скриптів
Unreal Engine	Потужний візуальний редактор, Blueprint, інструменти для 3D, VR, анімацій, матеріалів, Niagara VFX	Unreal Framework: ECS-подібна архітектура, Actor/Component система	Активна спільнота, офіційна документація, Unreal Marketplace, Epic Games Support	C++, Blueprints (візуальне програмування)
Unity	Візуальний редактор, інспектор, Shader Graph, Animation, Timeline, URP/HDRP, пакети	MonoBehaviour система, ScriptableObject, ECS (через DOTS), Unity Framework	Дуже велика спільнота, Unity Asset Store, офіційна документація, навчальні ресурси, часті оновлення	C#, ShaderLab, візуальні графи
Godot	Lightweight редактор, сцени та ноди, 2D/3D підтримка, вбудований UI редактор, Visual Script	Власна сцено-орієнтована архітектура (Node/Scene), Component-підхід	Активна open-source спільнота, документація, GitHub, часті оновлення, модульна структура	GDScript, C#, C++, Visual Script
GameMaker	2D-орієнтований редактор, Sprite/Tile редактор, Drag-and-Drop, Room Editor	Простий об’єктно-орієнтований фреймворк	Спільнота, документація, Marketplace, добре підходить для початківців	GML (GameMaker Language), Drag-and-Drop
Bevy	Без редактора, повністю код-орієнтований, ECS, плагіни	ECS-архітектура (Entity-Component-System), Bevy App, модульна структура	Open-source проєкт, активна Rust-спільнота, документація на сайті, швидкий розвиток, спільнота на Discord/GitHub	Rust

Порівнявши наявні опції, було складено вердикт обраний рушій Unity. Хоча Unreal Engine і пропонує потужний інструментар, в контексті нашого проекту було виділено такі недоліки:

– інструменти Unreal гарно адаптовані для реалістичної та деталізованої графіки, проте проект ставить в пріорітеті більш прості та стилістично вивірені рішення що не потребують таких інструментів для роботи з освітленням та графікою достатньо базових шейдерів та asset’ів з низькою кількістю полігонів та деталізації в цілому;

– графіка Unreal вимагає значно потужнішого заліза для розробки та запуску гри на рушії, цей недолік відсікає не лише користувачів слабших систем, а і також відсікає перспективні платформи по причині того що основні

фішки рушія досягаються лише завдяки DirectX 12 що є графічним API суто для Windows та Xbox консолей [20];

– Unreal хоч і підтримує кросплатформні графічні API (Vulkan, OpenGL, Metal), кінцевий продукт на цільових платформах за межами ПК не може використовувати усіх графічних можливостей рушія;

– фреймворк Unreal Engine покладається на інструменти Blueprints та мову C++, і хоч мова програмування дуже якісна і легка в компіляції, фокус та найбільшу підтримку виділяють саме на Blueprint, а він доволі часто обмежує певні аспекти написання логіки і часто може навіть ускладнити роботу розробнику з досвідом що звик користуватись алгоритмами на рівні мови програмування;

– шаблонність проектів Unreal Engine дещо відштовхує проекти, бачення яких фокусується не на використанні реалістичної графіки, а також тих хто зацікавлений в написанні більш комплексної логіки для фізики чи інших механічних елементів проекту. Через це в певних колах Unreal має репутацію шаблону для шутерів від першої та 3ї особи, і це змушує ігри що були зроблені на рушії викликати відчуття дежавю.

Що стосується недоліків Unity, в контексті проекту, їх було виділенно значно менше, рушій по загальній характеристиці краще підходить для цієї гри. Проте згадати їх варто, це:

– контраверсійність комерційної моделі Unity. У вересні 2023 року Unity оголосили зміни до своєї комерційної моделі, пропонуючи замінити традиційну роялті модель на модель що використовуватиме телеметрію даних про встановлення ігор, і використовувати ці дані беручи комісію за кожне навіть повторне встановлення гри на девайс. Ця подія підірвала довіру багатьох розробників до Unity, проте ці зміни було перероблено і тепер «Runtime fee» це опція на рівні з традиційним відсотком royalty;

– рушій хоч і пропонує потужні та якісні графічні інструменти, він все ще не може повноцінно конкурувати з інструментами Unreal через їх новітні рішення Nanite та Lumen що дійсно стали проривом для індустрії, і це

дійсно важливий аспект для великих студій, та для тих проектів що фокусуються на реалістичній графіці у великих відкритих локаціях;

– основний фреймворк Unity працює на базі Monobehaviour, клас що є наслідуючим для більшості інших класів у проекті. Наслідуючі компоненти цього класу зазвичай покладаються на однопотокові процеси оновлення (кількість потоків важко змінити для Monobehaviour), що робить масштабованість об’єктів з такими компонентами доволі ризикованою, адже появляється не лише ризик створення системи з «спагетті коду», а також наткнутись на проблеми з продуктивністю проекту. Проте Unity активно розробляє також систему DOTS яка набуває популярності та функціоналу, і включає в себе інструменти ECS (Entity component system), Burst Compiler та Jobs system [21].

Godot будучи open source рішенням у порівнянні хоч і має багато технічних переваг, головними недоліками для нього є:

– підтримка, яка тримається суто на команді ентузіастів та ком’юніті користувачів, і хоч це дозволяє кожному вільно допомагати з розробкою рішень для рушія, це також означає що кількість і доступність інструментів солідно менша в порівнянні з комерційними рішеннями;

– графічні інструменти Godot є «молодими» і значно слабшими у графічному потенціалі, в порівнянні з Unity у цього рушія немає настільки хороших інструментів для створення шейдерів;

– фізичний рушій у Godot є слабшим за той що використовується у Unity, через що велика кількість фізичнийх об’єктів значно сповільнює гру.

Про GameMaker потрібно сказати коротко, якщо інші рушії є складними для використання у порівнянні інструментами, якість та глибина кастомізації у них ні в яке порівняння з цим рушієм не стає, GameMaker занадто простий та фокусований суто на 2D проекти.

У Bevy ж ситуація складніша. Рушій виглядає дуже цікаво з точки зору створення гарно оптимізованих проектів, використовує хорошу мову програмування та має доволі зручний фреймворк. Але негативні моменти цього

рушія нажаль не дають йому можливості зараз конкурувати з іншими рушіями у контексті мого проекту. Ці негативні аспекти включають:

– відсутність графічного інтерфейсу. Bevy дуже молодий open source рушій, що відбивається на зручності використання його інструментарю, адже більша частина роботи відбувається у консолі, рушій не має графічного редактора та інспектора «з коробки»;

– інтеграція користувацьких розробок, через відсутність графічного інтерфейсу робить фреймворк значно більш комплексним для не досвідчених програмістів;

– графічні інструменти рушія, хоч і включають у себе як можливості 2D та 3D, навіть близько не можуть конкурувати в цьому аспекті з іншими рушіями.

Опираючись на ці порівняльні характеристики було обрано Unity. Попри ситуацію з контраверсією, компанія-власник рушія все ж дослуховується до ком'юніті та впроваджує зміни від нині більш прозоро. Графічні інструменти що надає рушій ідеально підходять для стилізованої гри що покладається на екперементальні та загалом не звичайні артистичні рішення. Та навіть звертаючи увагу на те, що базою фреймворка Unity є компоненти Monobehaviour, ці компоненти є значно зручнішими та зрозумілішими у побудові та використанні у проектах, додатково до цього компоненти Monobehaviour можна доволі гарно комбінувати разом з інструментами DOTS, що дозволяє користуватись бенефітами керування централізованими компонентами та одночасно користуватись компонентами ECS та Jobs що дозволяє реалізовувати значно більш комплексні обрахування для великої кількості різноманітних об'єктів [8].

Тепер, обравши рушій для розробки, потрібно обрати інші частини фреймворку. У виборі IDE для проекту, вибір керується такими аспектами: легкість програми, інтегрованість обраним рушієм, наявність допоміжних для фреймворку плагінів, доступність на платформі розробки. Порівняння було складено у порівняльну таблицю 2.2.

Таблиця 2.2 – Порівняння різних IDE

Критерій	Visual Studio	VS Code	IntelliJ IDEA	NeoVim
Легкість програми (RAM, CPU)	Важкий (2–3 ГБ+)	Дуже легкий (300–600 МБ)	Середній (1–2 ГБ)	Надлегкий (менше 100 МБ)
Інтеграція з рушіями	Ідеальна для Unreal/C#	Дуже хороша для Unity, Godot, Bevy	Добра для Java/Android, можна адаптувати	Вимагає налаштування
Плагіни для фреймворків	Переважно для Windows	Величезна кількість (Unity, Bevy, Godot, Web)	Потужна екосистема для JVM + веб	Є плагіни, але вручну через lua/Vimscript
Підтримка системи:	Лише Windows	Повна підтримка Linux (Snap/Deb)	Повна підтримка Linux	Повна підтримка Linux

Для роботи над проектом було обрано VS Code, завдяки тому що ця IDE є доволі легкою на споживання ресурсів системи, має потужну всебічну інтеграцію з різними фреймворками, зокрема Unity, та підтримується системою на якій розробляється проект Linux дистрибутиву Mint 21.3.

Оскільки роль розробки asset'ів, зокрема 3D моделей падає на мої плечі, потрібно було додатково обрати інструменти для створення моделей, анімації, створення текстур. В контексті цього вибору конкурують декілька рішень, для яких було проведено порівняльну характеристику у таблицях. У таблиці 2.3 проведено порівняння інструментів для 3D моделювання.

Таблиця 2.3 – Порівняння 3D редакторів

Інструмент	Плюси	Мінуси
Blender	Безкоштовний і відкритий; Потужні інструменти моделювання (SubD, Sculpt, Hard Surface); Активна спільнота	UI може здатись незвичним; Важкий для слабких ПК
3Ds Max	Стандарт у AAA; Добре інтегрується з Unreal Engine	Платний (дорожчий для інді); Лише Windows
Maya	Потужна анімація + моделювання; Часто в кіноіндустрії	Дорогий; Високий поріг входу
Blockbench	Простий для low-poly і voxel; Добре для Minecraft-like	Не підходить для складного hi-poly
ZBrush	Лідер у скульптингу; Ідеальний для деталізації персонажів	Дуже дорогий; Не підходить для технічного моделювання

У таблиці 2.4 проведено порівняння інструментів для створення анімацій, за позитивними та негативними аспектами. Головними аспектами стали інтуїтивність інтерфейсу, якість самих інструментів та ціна ліцензійної копії програми.

Таблиця 2.4 – Порівняння редакторів анімацій

Інструмент	Плюси	Мінуси
Blender	Потужний анімаційний стек	Rigging не такий інтуїтивний, як у інших
Cascadeur	AI-підтримка пози; Зручно для бойової анімації	Обмежена екосистема; Менше гнучкості;
Maya	AAA-рівень для анімацій (кінематограф, ігри)	Платний; Ускладнене освоєння
Mixamo	Швидка генерація motion capture; Безкоштовно	Лімітована кастомізація;

У таблиці 2.5 проведено порівняння інструментів для створення текстур, за позитивними та негативними аспектами.

Таблиця 2.5 – Порівняння редакторів зображення

Інструмент	Плюси	Мінуси
Krita	Відкритий і безкоштовний; Спеціалізований на цифровому малюванні (і текстурах)	Менш зручний для ретуші фото
GIMP	Вільна альтернатива Photoshop; Плагіни	Не інтуїтивний UI; Погана підтримка CMYK
Photoshop	Ідеальний для 2D-текстур, ретуші; Багато ресурсів	Платний (підписка); Важкий для слабких ПК
Substance Painter	Лідер у 3D-текстуруванні (PBR); Реалтайм viewport	Дуже дорогий; Немає 2D-піксельної деталізації

Для розробки нашого проекту було обрано стек Blender та Krita [11], оскільки обидва редактори працюють в ОС базованих на Linux, мають безкоштовну основу, Blender за роки розвитку став потужним універсальним інструментом для моделювання, rigging процесу, анімації, накладення текстур та навіть створення повноцінних кастомних матеріалів (гнучке налаштування параметрів освітлення, об’ємів, кольору та загалом рендеру об’єкта) за допомогою інструментам Nodes [4]. Krita ж до всього є одним з найбільш якісним графічним редактором, є популярною у колах арт-дизайнерів, та є легким у використанні інструментом.

Висновки до розділу 2

Проаналізувавши першорядні аспекти під час розробки проекту, було визначено принципи за якими буде формуватись архітектура систем, зокрема такі складові як система керування ігровим процесом. Для схеми керування

було складено та класифіковано набір абстрактних дій, назви та значення для яких було обрано на основі поширених геймдизайнерських рішень щодо систем керування грою. Було сформовано та описано архітектуру класів системи та аргументовано короткими тезами та порівняннями чому та як були прийняті певні рішення під час розробки.

Для зображення моделі системи використано UML діаграму класів, оскільки вона дозволяє графічно пояснити прості зв’язки між класами, зберігаючи у своїй простоті увесь необхідний інформаційний вміст. Більшість порівняльних та аналітичних аргументацій було окреслено у вигляді таблиць, зокрема порівняння рушіїв, порівняння IDE програм, аналітика для вибору програми для 3D моделювання, анімацій та роботи з растровою графікою.

По закінченню аналітичного огляду ігрових рушіїв, для розробки проекту було обрано Unity, через те що цей рушій пропонує найбільш підходящий набір інструментів для фреймворку гри, зокрема зручний у розробці Monobehaviour клас, зручні інструменти створення шаблонів для дата контейнерів, інструменти збереження та відтворення створених ігрових об’єктів, графічні інструменти для створення шейдерів (Shader Graph), можливість інтеграції сучасної системи оптимізації DOTS (Data oriented technology stack) що містить в собі систему мультипроцесових алгоритмів Jobs, інструмент створення для найкраще оптимізованих для використання пам’яті комп’ютера сутностей (Entity component system), та оптимізований для швидких операцій Burst compiler.

Також у доповнення стеку розробки було обрано легкий IDE, з хорошою інтеграцією в фреймворк, це Virtual Studio Code. Для створення моделей та анімацій обрано Blender завдяки своїй універсальності, зручності та безкоштовній основі, у додаток до нього графічним редактором було обрано Krita за його легкість та безкоштовну основу.

РОЗДІЛ 3
РОЗРОБКА ВІДЕОГРИ «TSOKOTARA» З ІМПЛЕМЕНТАЦІЄЮ АСИМЕТРИЧНОГО VR КООПЕРАТИВУ НА UNITY

3.1 Практична реалізація об’єкта проектування

Початком роботи у рушії стає створення проекту. У вікні завантаженого UnityHub додатку обираємо необхідну нам LTS (Long term support) версію редактора, важливо врахувати також наявність у версії потрібних нам у фреймворку інструментів (певні package’і доступні лише в новіших версіях редактора), у налаштуванні обираємо стек для білда (системи на яких гра буде запускатись, додаткові мови), проходимо процес встановлення редактора. По закінченню завантаження обраної версії ми створюємо новий проект, де вводимо ім’я проекту та обираємо шаблон з доступних опцій, це навчальний шаблон, 2D та 3D варіації шаблонів, для яких обираємо також варіації Render pipeline. З доступних нам опцій Render Pipeline є Built-inRP (вбудований, підтримує лише базові графічні інструменти нових версій рушія з коробки), URP (Універсальний, найкраща опція для всебічної розробки адже дозволяє використовувати майже усі новітні графічні інструменти розробки, а також дозволяє глибоко кастомізувати ці графічні інструменти) та HDRP (High Definition, це render pipeline що містить в собі усі найбільш новітні та потужні інструменти графіки що пропонує Unity). Для проекту було обрано URP, адже інструментів BRP для проекту є недостатньо (крім того що у BRP «з коробки» не йде Shader graph, навіть якщо цей інструмент встановити то він містить в собі доволі обмежений функціонал для цього render pipeline), а HDRP є занадто вимогливим та в цілому не є потрібним для нашого проекту.

Після встановлення та відкриття проекту в редакторі, потрібно додати по можливості усі необхідні packages (від нині плагіни), для того щоб продовжувати роботу. Одним із перших плагінів що вам може кинутись у процесі роботи з UI в майбутньому, скоріше за все стане Text Mesh Pro, це плагін що додає кращі можливості редагування тексту, у яких можна

налаштувати шрифти та загальне форматування для блоку тексту. Другим плагіном рекомендується встановити Input Actions, це новітні інструменти схеми керування що прийшли на заміну вбудованій системі Input Manager. Цей плагін є потрібним оскільки наявність саме такої системи керування дозволяє створювати систему керування грою для кросплатформи без жодних проблем. Третім плагіном який рекомендовано встановити саме для проектів у 3D середовище є Cinemachine. Це зручний плагін для налаштування та керування камерою, він є доволі потужним інструментом у поєднанні з Timeline плагіном, що дозволяє створювати катсцени та загалом динамічні сцени всередині редактора рушія. Також, якщо розробка проекту передбачає використання великої кількості об’єктів, необхідно встановити набір плагінів DOTS, це Entities (головний плагін ECS, який містить всередині вікна package manager усі необхідні для його роботи плагіни), Entities Graphics (сутності користуються окремою від стандартних об’єктів графікою) та Unity Physics (це плагін фізики від Unity що було створено для використання з ECS, оскільки у сутностей також окрема фізична модель).

По закінченню встановлення плагінів проект по суті є готовим для роботи, проте в цілому при потребі в майбутньому ви можете повернутись до вікна Package manager оскільки у Unity Asset Store є величезна кількість різного роду плагінів для усіх елементів розробки (графічні asset’и, збірки систем, звукові asset’и та багато інших).

Для нашого проекту було встановлено усі описані плагіни, що дуже спрощує розробку проекту, проте певні плагіни використовуються в проекті доволі обмежено. Через те що певні аспекти ECS по суті вимагають окремих систем як власна фізика та власна графіка, створення компонентів для сутностей є доволі комплесним та обмежуючим у роботі завданням, оскільки ECS до прикладу досі немає підтримки інструментів pathfinding накшталт плагіну AI Navigation, через що прийнято рішення написання базових Monobehaviour компонентів, проте попри цей недолік плагін Entities знаходить своє використання у вигляді створення subscenes, які використовуються для

«запікання» об’єктів у сутності. Головною перевагою «сабсцен» над звичайними сценами є швидкість їх завантаження, адже навіть без використання привілеїв сегментованого провантаження, «сабсцени» легше підвантажити. Ще одною перевагою для «сабсцен» є їх структура, через те що «сабсцена» технічно є ігровим об’єктом вона за замовчуванням зберігає у собі компонент Transform, що дозволяє легко визначити її місцезнаходження що буде дуже корисно при розробці систем для world streaming.

Нарешті по закінченню встановлення усіх необхідних плагінів, варто коротко розібратись з базовою структорою вікон у редакторі. На рисунку 3.1 зображено та пронумеровано для подальших пояснень стандартну розкладку вікон у редакторі.

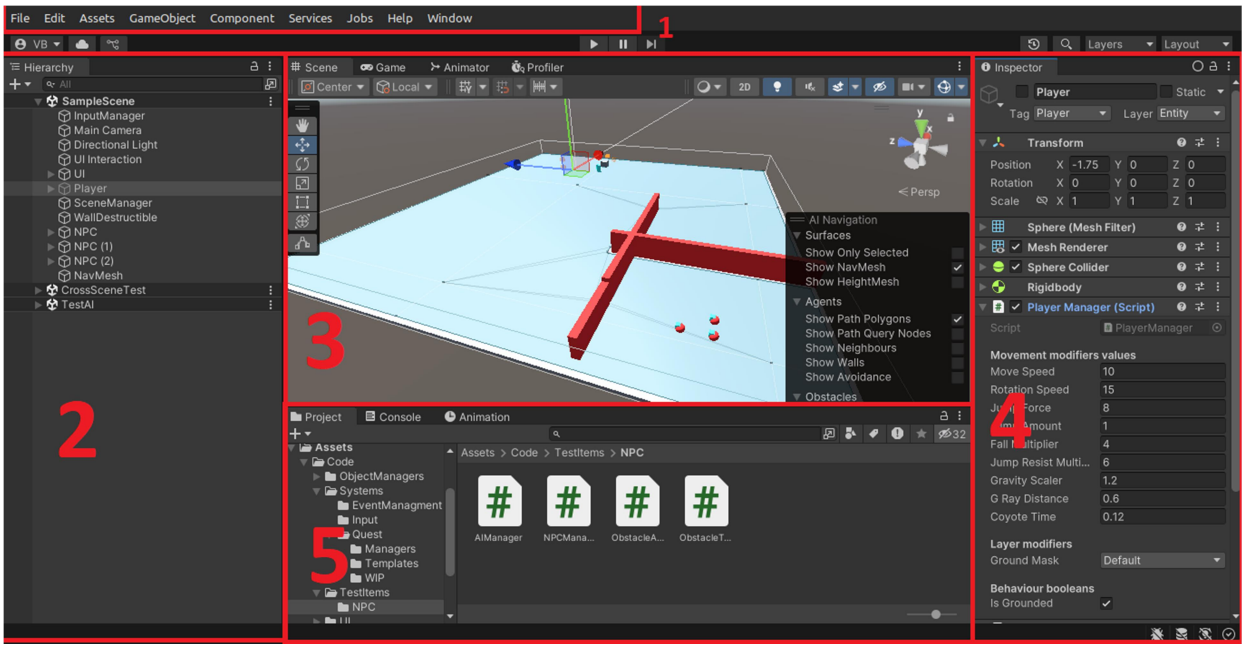

Рисунок 3.1 – Структура вікна редактора Unity

Відповідно до рисунка, у таблиці 3.1 описано усі вікна у відміченій послідовності. Першим елементом є Панель керування, другим елементом стоїть вікно ієрархії, третім позачено головне вікно у якому відображається ігрова сцена та сама гра, у четвертому позначенні знаходиться вікно «Інспектор», та у п’ятому елементі відмічено положення вікон «Проєкт» та «Консоль».

Таблиця 3.1 – Опис елементів у вікні редактора Unity

№	Вікна	Призначення
1	Панель керування	Тут розміщені контекстні меню редактора для різних додаткових дій, налаштувань та для відкриття додаткови вікон
2	Hierarchy (Ієрархія)	Це вікно розстановлення в певному порядку об'єктів на сцені. Використовується також для створення об'єктів на сцені та корректного розстановлення наслідуючих об'єктів
3	Scene/Game	Вікно редактора сцен та вікно ігрового процесу (перемикання табів зверху). Перше це незакріплена камера редактора, друге це вигляд на сцен під час гри з активованої камери
4	Inspector	Вікно огляду компонентів обраного об'єкта
5	Project/Console	Вікно файлової організації проекту та вікно консолі компілятора рушія

На панелі керування редактором знаходяться опції для налаштування файлу, налаштування редактора, опції asset’ів, створення нових об’єктів, налаштування компонентів, опції сервісів, опції для нових вікон редактора та опції довідника допомоги. Ці опції використовуються відносно не часто тому не виникає проблеми запам’ятати потрібні пункти.

Вікно Ієрархії у редакторі є одним з найважливіших вікон в редакторі, оскільки тут формується структура об’єктів для кожної сцени і відповідно підвантажуються самі сцени та «сабсцени». Залежно від розміщення одних об’єктів всередині інших, здійснюється їх переміщення на сцені, адже компонент Transform батьківського об’єкта по суті є орієнтовним центром для локальних координат для об’єкта наслідувача. Також структура ієрархії допомагає звертатись до компонентів як батьківського так і наслідуючого компонента за допомогою виклику «GetComponentInChildren».

Вікно сцени відповідно є головним вікном для розстановки об’єктів у сценті, в ньому розміщені інструменти для редагування позиції, обертання та розмірів об’єкта. Також з цього вікна зручно знаходити та переглядати конкретні об’єкти на сцені. Вікно гри ж перемикається автоматично за замовчуванням коли запускається режим гри. У цьому вікні фактично демонструється ігровий процес відкритої сцени сцени можна завантажувати з заміною та асинхронно в процесі режима гри, тобто реальні умови роботи кінцевого проекту.

У інспекторі відбувається головна робота по організації ігрових об’єктів встановлення та налаштування компонентів. Ці компоненти фактично являють собою Monobehaviour класи написані мовою C#, і встановлення кожного з них передбачає додавання патернів поведінки для об’єкта на сцені. Фактично написання цих компонентів є основною активністю по створенню механік, логіки та систем для гри.

Вікно «Проект» фактично є спрощеною файловою системою у редакторі, використовується для управління та органзацією файлів коду, зовнішніх asset’ів, дата контейнерів наслідуючих клас Scriptable Object, колекцією prefub’ів (копія об’єкта з ігрової сцени у вигляді файлу) та плагінів. Ну а вікно консолі в іншому табі зверху вікна виводить логи розробника, попередження та помилки що появились під час спроби запуску режиму гри, після збереження файлу скрипта з помилками або під час виникнення помилки у режимі гри.

Для створення першого функціонального компонента клікаємо ПКМ (правою кнопкою миші) на порожнє середовище у вікні «Проект» та у контекстному вікні що появилось обираємо «Create C# script». Після створення та назви скрипта (рекомендовано дотримуватись загальноприйнятих правил найменування для уникнення помилок у організації архітектури проекту пізніше) ми можемо відкрити файл подвійним кліком лівої кнопки миші, після чого відкриється вікно налаштованої IDE, в нашому випадку це VS Code, де буде описано майже порожній клас з назвою файла скрипта, в якому автоматично описано використання середовищ визначень (using …), а також біля назви класу на його початку вписано що він наслідує клас в Unity що зветься Monobehaviour. Чим важливий аспект того що наш клас наслідує Monobehaviour так це тим що:

– компонент наслідуючий Monobehaviour може бути використаний та отримує цикл оновлень лише коли його було прикріплено до ігрового об’єкту;

– цей клас отримує у наслідуванні функції циклів Update, FixedUpdate, LateUpdate, Awake, Start, OnEnable та OnDisable та декілька інших

функцій що працюють від прив’язки з іншими компонентами на об’єкті (наприклад OnTrigger або OnCollision для взаємодії з компонентом Collider);

– для кожного об’єкту з компонентом цього класу використовується цикл оновлень окремо, тобто умовна функція Awake відбувається для двох об’єктів по суті у випадковій черзі адже викликається вона одночасно, але потребує оновлення для кожного Awake на запуску циклу.

Цей компонент класу ми можемо одразу прикріпити до об’єкту на сцені, для відслідковування помилок за допомогою компілятора Unity, зокрема такий метод допомагає знаходити упущені референси в коді, які ми можемо визначити під час виконання однієї з функцій або просто перетягнути у компонент на об’єкті у вікні інспектора. Спершу ми описуємо необхідні значення у коді, оскільки C# вимагає щоб значення що будуть використані далі у коді були описані перед виконанням алгоритму, також важливо знати що значення описане в межах певної функції може використовуватись лише всередині функції, тому значення зазвичай описуються на рівні класу (головним виключенням є функції які для опрацювання отримують значення в аргументі). Коли у нас є абстрактні значення накшталт цілих чисел (int), чисел з плаваючою крапкою (float), вектор (Vector2 для 2х вимірного значення та Vector3 для 3х вимірного значення), або навіть визначення для іншого класу (наприклад ComponentManager як визначник, та componentManager у якості імені для виклику у функціях) та інших, ми можемо на їх основі почати функції. Функція у C# виконує алгоритм описаний всередині неї (наприклад для визначеного числового значення у функції можна описати математичну формулу для знаходження нового числовго значення що може замінити попереднє значення) під час виклику. Функції є різних типів, що важливо враховувати при написанні, оскільки функіця типу void наприклад не вимагає повернення жодних значень оскільки вона сама не декларує значення, а от функція типу bool (визначення хибне чи правдиве значення) вимагає повернення значення boolean у кінці алгоритму (саме в кінці алгоритму оскільки функція завершує своє виконання коли викликається return) [15]. Ще

один важливий елемент всередині класу на даному етапі це співпроцедура (coroutine), цей елемент важливо згадати оскільки він працюючи як функція дозволяє запустити алгоритм дій що працює незалежно від іншох функцій у класі оскільки його виконання не обмежене одним кадром, співпроцедура може викликати таймер всередині себе який може очікувати фіксовану та не фіксовану кількість часу, задаючи послідовність виконання алгоритму у часі (наприклад коли ми викликаємо співпроцедуру вона запускає перший алгоритм, запускає очікування до наступного кадру, після чого викликає інший алгоритм, після чого може запустити наступний таймер який очікує на секунди і тд). Використовуючи ці аспекти ми пишемо перший скрипт логіки для об’єкта.

На прикладі лістингу 3.1. можна розглянути як клас, що прикріплюється до об’єкта у вигляді компонента, працює на практиці. Спершу ми задаємо значення для змінних які будемо надалі використовувати в коді:

– InputManager inputManager, Rigidbody pRigidbody, Transform cameraObject це змінні компонентів ззовні класу PlayerManager, а саме InputManager (клас що було написано попередньо та прікріплено до окремого об’єкта на сцені), Rigidbody (компонент що виконує фізичні обрахування та задає рух закріпленого об’єкта в параметрах фізичного рушія) та Transform (компонент що зберігає та керує позицією, кутами нахилу та розміром об’єкта, прикріплений до всіх об’єктів без виключення) що прикріплено до об’єкта камери у грі;

– private Vector3 moveDirection , це значення 3х вимірного вектора що визначає напрям персонажа, ця змінна має модифікатор «private» оскільки є лише копією для місцевого використання;

– public float moveSpeed, це змінна числа з плаваючою крапкою що визначає множник для задання швидкості руху гравця;

– public float rotationSpeed, ця змінна визначає множник швидкості обертання персонажа з його минулої позиції до нової.

Лістинг 3.1 – Приклад змінних компонента «PlayerManager» для гравця

```
using UnityEngine;

public class PlayerManager : MonoBehaviour
{
      InputManager inputManager;
      Rigidbody pRigidbody;
      Transform cameraObject;

      private Vector3 moveDirection;

      public float moveSpeed;
      public float rotationSpeed;
```

кінець лістингу 3.1

Використовуючи ці змінні ми можемо описати алгоритми в наступних функціях. Для збереження логічної послідовності ми описуємо функції приблизно по порядку їх логічного виклику, до прикладу функція Awake завжди у черзі ініціалізації буде передувати функції Start, навіть попри те що по суті обидві функції мають схожу роль. Приблизно такий порядок функцій у описаному класі (ліст. 3.2).

Лістинг 3.2 – Приклад функцій класу компонента «PlayerManager» для гравця

```
public class PlayerManager : MonoBehaviour
{
      private void Awake()
      {
            pRigidbody = GetComponent<Rigidbody>();
            cameraObject = Camera.main.transform;
            inputManager = InputManager.instance;
      }

      private void FixedUpdate()
      {
            PlayerMovement();
            PlayerRotation();
      }

      private void PlayerMovement()
      {
            moveDirection = cameraObject.forward *
      inputManager.movementInput.y;
```

```
        moveDirection = moveDirection + cameraObject.right *
    inputManager.movementInput.x;
        moveDirection.Normalize();
        moveDirection.y = 0;
        pRigidbody.velocity = new Vector3(moveDirection.x * moveSpeed,
    pRigidbody.velocity.y, moveDirection.z * moveSpeed);

    }

    private void PlayerRotation()
    {
        Vector3 targetDirection = Vector3.zero;
        targetDirection = cameraObject.forward *
    inputManager.movementInput.y;
        targetDirection = targetDirection + cameraObject.right *
    inputManager.movementInput.x;
        targetDirection.Normalize();
        targetDirection.y = 0;

        if (targetDirection == Vector3.zero)
            targetDirection = transform.forward;
        Quaternion targetRotation =
    Quaternion.LookRotation(targetDirection);
        Quaternion playerRotation =
    Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed
    * Time.deltaTime);

        pRigidbody.MoveRotation(playerRotation);   }
}
```

кінець лістингу 3.2

– функція Awake передує іншим функціям оскільки вона ініціалізується першою. Всередині цієї функції ми задаємо значення для змінних компонентів pRigidbody(Rigidbody гравця), inputManager(клас InputManager у формі singleton компонента) та cameraObject (компонент Transform камери). Викликаються алгоритми що віднаходять ці компоненти на об’єкті гравця, менеджера керування та камери відповідно;

– функція FixedUpdate, має відмінність від звичайної функції Update у тому, що ця функція викликається через кожен умовний проміжок часу (приблизно кожні 0,02 сек) на відміну від другої функції яка викликається залежно від частоти кадрів (FPS) у грі. Тому FixedUpdate важливо використовувати для оновлень функцій і компонентів фізики (надання сил або

швидкості через згаданий компонент Rigidbody наприклад). В наші функції описано алгоритм для – оновлення наступних функцій PlayerMovement та PlayerRotation;

– функція PlayerMovement у нашому класі відповідає за виклик змінної Vector3 velocity всередині компонента Rigidbody в об’єкті гравця. Для цього наша функція підтягує значення Vector2 з компонента InputManager, перетворює їх відповідно до позиції камери (щоб напрямок у який рухається персонаж не збивав з пантелику гравця коли камера змінила свою позицію), нормалізує отриманий Vector2 (це необхідно щоб отримати коректну довжину вектора коли отримані значення X=1 та Y=1, адже координата 1,1 видає вектор довжиною більше чим 1 за теоремою Піфагора). Так наприкінці ми використовуємо цей Vector2 для утворення напрямку руху для Vector3 (значення Y для Vector3 ми ігноруємо оскільки персонаж за функцією мусить рухатись лише у горизонтальній площині) та передаємо це значення у компонент Rigidbody;

– функція PlayerRotation у класі відповідає за обертання персонажа у сцені, оскільки використовуючи лише функцію PlayerMovement ми отримуємо персонажа що рухається у сцені але не обертається у напрямку руху. Спочатку ми оголошуємо змінну Vector3 targetDirection що дорівнює нулю (це зроблено для того щоб персонаж обертався лише коли натиснута кнопка руху), після чого встановллюються значення координат X та Z відповідно до положення камери та напряму Vector2 заданого натисненням клавіш WASD (усередині InputManager). Ми знову ігноруємо значення Vector3 осі Y, оскільки нам потрібен напрямок в горизонтальнії площині. Далі ми декларуємо змінну Quaternion (змінна обертання у 3х осях) в якії визначаємо напрям кута для обертання від отриманого в останньому кроці алгоритму значення Vector3 targetDirection, після чого ми визначаємо новий Quaternion playerRotation, але сюди ми знаходимо значення за формулою інтерполяції Quaternion.Slerp, яка визначає значення поміж теперишнім та бажаним кутом повороту, схиляючи його від першого до другого використовуючи значення часу (Time.deltaTime

означає час в секундах між попереднім та цим кадром) помноженого на оголошену нами числову змінну rotationSpeed. Після завершення цих обрахувань, результат формули задає обертання викликаючи змінну MoveRotation всередині компонента Rigidbody, що в результаті обертає персонажа на сцені, залежно від того наскільки довго ви тримаєте кнопку та виставленого множника змінної швидкості обертання, персонаж обертається від свого кута напряму в бік напряму руху, і якщо кнопку відпустити зарано то персонаж буде дивитись у бік між напрямом руху що ви задали та попереднім своїм напрямом погляду.

Повноцінну версію скрипта PlayerManager можна переглянути у додатку Б, а також приклад коду для класу InputManager що прікріплено та використовується у PlayerManager, можна переглянути у додатку В. Таким чином ми змусили об’єкт гравця рухатись на сцені, і приблизно таким чином і описуються функції в подальшому. Наступним кроком в послідовності розробки буде вивчення необхідних вам компонентів, як вони взаємодіють з вбудованими системами гри, яким чином ви можете зберігати різного виду об’єкти та створювати чи видаляти їх на сцені. Важливо для поставлених задач також розуміти принципи роботи логіки C# у цілому, це означає зрозуміти як працюють класи (наслідування, композиції та розділення обов’язків, спілкування між класами, цикли викликів оновлення, тощо), інтерфейси (контекст їх використання), struct (відмінності цього компонента від класів, контекст їх використання).

Компоненти у Unity можна не лише створювати самому, важливо також пам’ятати про наявні компоненти рушія, наприклад компоненти що взаємодіють з фізокою (Collider, Rigidbody, Joint різних типів, вбудований CharachterController, PhysicsMaterial), базові компоненти (Transform, скрипти Monobehaviour), компоненти для рендеру (MeshFilter та MeshRenderer, Sprite 2D, інші UI компоненти), та безліч інших компонентів «з коробки» та велика кількість компонентів що додаються плагінами.

Створення гри вимагає грамотної побудови структури самописних компонентів та в цілому менеджменту файлів проекту. Важливо дотримуватись логічного розподілу файлів за призначенням, за об’єктами для яких існують компоненти, за колекціями asset’ів.

3.2 Створення 3D моделей для гри

Після створення першого ігрового процесу за допомогою нативних asset’ів Unity (моделі геометричних примітивів, базові матеріали, стандартний фізичний матеріал, тощо), наступає момент коли потрібно створити або експортувати готові asset’и за межами проекту в Unity, для цього в конеткслі 3D проекту ми використовуємо платформи дистрибуції цифрових asset’ів (дотримуйтесь умов ліцензійної моделі завантажених робіт) або інструменти для 3D моделювання. У якості головного інструмента для створення моделей для розробників у інді-сегменті являється Blender, він є безкоштовним, open source, і також пропонує потужний інструментар для створення моделей, їх текстурування та анімації.

Щоб почати працювати у Blender, потрібно зрозуміти його інтерфейс та попрактикуватись з моделюванням у цілому. На рисунку 3.2. зображено інтерфейс вікна Blender за замовчуванням.

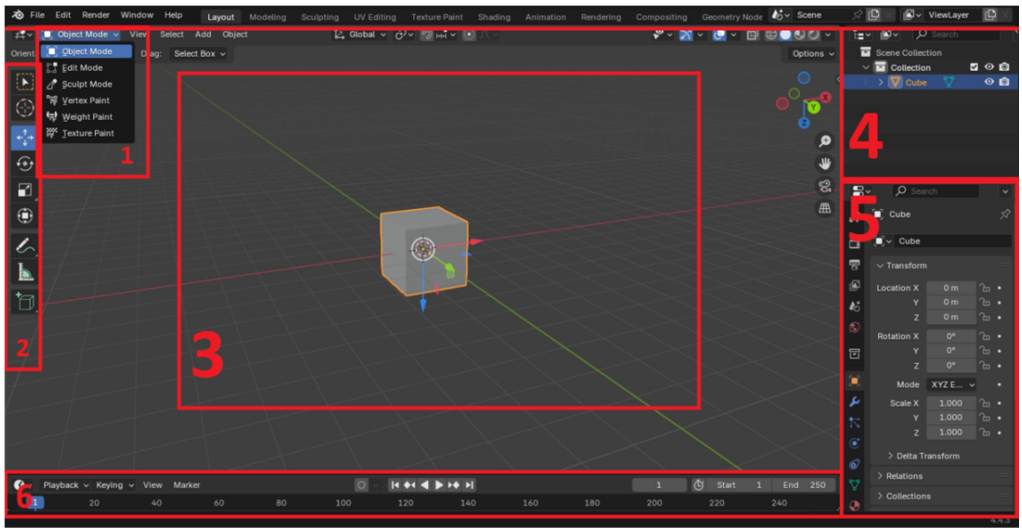

Рисунок 3.2 – Інтерфейс Blender

У таблиці 3.2. описано коротке пояснення кожного основного вікна в інтерфейсі.

Таблиця 3.2 – Опис вікон інтерфейсу Blender

№	Вікна	Призначення
1	Контекстне вікно режиму	Використовується для перемикання режимів редактора
2	Панель інструментів	Панель на якій розміщено набір базових інструментів редактора, відрізняється у різних режимах
3	Середовище з об'єктами	Основне серидовище для маніпуляції об'єктом у вибраному режимі
4	Ієрархія об'єктів	Вікно з впорядкованою структурою об'єктів на сцені
5	Вікно керування об'єктом та сценою	Вікно в якому знаходиться найбільша кількість головних інструментів редактора
6	Вікно timeline стрічки	Використовується в основному у роботі над анімаціями

У контекстному вікні режимів ми обираємо режим роботи редактора:

– Object Mode дозволяє створити окремий об’єкт для подальшого редагування, та розмістити його на сцені;

– Edit Mode це головний режим редагування створених об’єктів, тут ми працюємо безпосередньо з складовими об’єкту (для mesh’ів це грані, вертекси та обличчя, для Armature це кістки що складають rig, тощо), рухаємо, додаємо граней, загалом міняємо геометрію об’єкта;

– Sculpt Mode працює схожим чином до Edit Mode з mesh об’єктами, але пропонує значно ширший набір інструментів для створення геометрії, що дозволяє створювати високу деталізацію, наче ти працюєш зі скульпторою;

– Weight Paint це спеціалізований режим що дозволяє «фарбувати вагу» на окремі вертекси об’єкта, це використовується в основному для прив’язки mesh до анімаційного скелета (rigging) та для створення спеціалізованих матеріалів за допомогою nodes;

– Pose Mode дозволяє маніпулювати створеним та прив’язаним до mesh об’єктом скелета (armature), та за допомогою вікна №6 створювати анімації на часовій лінії (об’єкт виставляється в позу, робиться «ключовий кадр», обирається наступний кадр на лінії, виставляється наступна поза та кадр).

Також у редакторі є декілька інших вікон що працюють безпосередньо з об’єктами спеціального призначення (режим для малювання для Grease Pencil наприклад), проте в контексті нашого фреймворку ці режими не будуть часто використовуватись або працюють дуже схожим чином до уже існуючих режимів.

Панель інструментів та вікно з об’єктом працюють разом, оскільки обраний інструмент маніпулює об’єктом в середовищі. Вікно ієрархії створює уже знайому нам структуру наслідування, але на манер використання об’єктів моделі. А вікно з табами для загальних інструментів використовується дуже часто, під час моделювання нам найбільше будуть корисними вкладка для маніпуляції вертексами (всередині якої знаходяться налаштування Vertex Groups, Shape Keys, UV Maps тощо), вкладка з модифікаторами об’єкта, вкладка з властивостями об’єкта, вкладка з інструментами та вкладка матеріалів.

Розібравшись з процесом створення моделі, UV-розгорткою та накладенням текстур на матеріали моделі, логічним наступним кроком є перехід до анімації моделі (якщо об’єкт передбачає анімацію, не рухомі об’єкти на цьому етапі можна імпортувати у Unity). Для цього ми обираємо які засоби ми будемо використовувати для анімації, оскільки наприклад Shape Key анімація (це метод анімації за допомогою безпосередньо маніпуляцій з геометрією mesh’а) не підтримується в режимі Action Editor (режим що дозволяє розділити анімації на окремі дії), і по суті вимагає використовувати стандартний Timeline Editor що дозволяє імпортувати анімаційні кліпи одним великим кліпом (такий кліп потрібно розділити всередині Unity на менші сегменти), проте якщо Shape Key не використовується і потрібна лише традиційна скелетна анімація то можна використовувати Action Editor. Використовуючи методи створення keyframe анімації ми створюємо анімаційні кліпи для різних дій персонажа (Idle стан, атаки, взаємодії, рухи, тощо), після чого готоий asset можна імпортувати у Unity. Проте лишається один нюанс, матеріали з Blender у Unity неможливо експортувати напряму, оксільки вони

використовують різну логіку, відповідно матеріал у Blender та матеріал у Unity це дві різні речі. Цей момент можна виправити, якщо перенести процеси створення матеріалів у Unity.

Для створення матеріалів для моделі всередині Unity можна скористатись шаблонами готових матеріалів, або створити власний матеріал за допомогою інструментів створення шейдерів – Shader Graph. Цей редактор працює схожим чином до редактора nodes у Blender, він також використовує візуальну логіку. Використовуючи логіку для освітлення, накладення кольору, ефектів, всередині редактора ми створюємо шейдер для матеріалу, після чого створений шейдер можна накласти на імпортовані моделі.

Що стосується імпортовани анімацій, після їх додавання у колекцію файлів проекту та підготовки, ми підготовуємо їх до використання. На рисунку 3.3 зображено 3 інструменти роботи з анімацією.

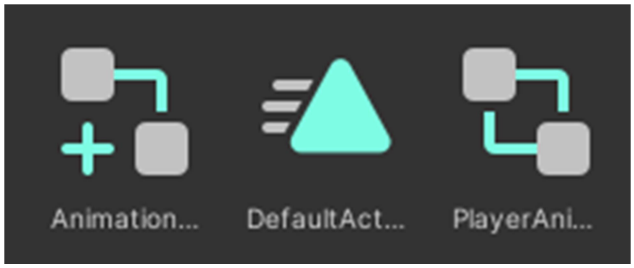

Рисунок 3.3 – Анімаційні інструменти у вікні «Project» Unity

Перший компонент це Animation Override, безпосередньо анімаційний кліп та Animator компонент. Анімаційний кліп містить інформацію про ключові кадри анімації що були імпортовані в рушій, а також у них можна додати анімаційні події (Animation events) всередині редактора рушія, ці події використовуються для того щоб викликати функції з компонентів що прив’язані до того ж ігрового об’єкта на якому відбувається анімація. Компонент Animator використовується для налаштування «станів» анімації, зокрема всередині редактора аніматора можна налаштувати плавність переходу між анімаційними кліпами, умови для програвання стану та налаштування швидкості програвання анімації. Animation Override клас в свою чергу

використовується для заміни анімаційного кліпу всередині Animator’а, що є корисним при створенні логіки для великої кількості анімацій.

Таким чином, поєднуючи імпортовані asset’и у середовищі Unity, створюючи компоненти що будуть взаємодіяти з оточенням, ми створюємо об’єкти для подальшого використання у грі.

3.3 Створення UI та UX для гри

Процес створення інтерфейсу користувача (UI) та побудови користувацького досвіду (UX) є критично важливою складовою у розробці будь-якого програмного продукту, а особливо – у відеогрі, яка передбачає взаємодію з ігровим світом у реальному часі. Оскільки наш проект поєднує традиційне 3D керування з елементами асиметричного VR кооперативу, створення інтуїтивного та адаптивного інтерфейсу є надзвичайно важливим аспектом.

Під час проєктування UI/UX для гри було дотримано низки ключових принципів, які забезпечують комфорт користувача, логічну структуру інтерфейсу та відповідність загальній естетиці гри:

– інтерфейс користувача спроєктовано таким чином, щоб уникати перевантаження екранного простору. Всі елементи розташовані продумано, з урахуванням того, щоб не заважати огляду ігрового світу. Основна увага приділяється ігровому процесу, тому UI виконує лише допоміжну функцію – інформує гравця та забезпечує необхідні механізми взаємодії без зайвих візуальних деталей;

– важливим аспектом стало створення інтерфейсу, що реагує на зміну ситуацій у грі. Наприклад, індикатори здоров’я, ресурсів або підказки з’являються лише тоді, коли це дійсно потрібно: під час бою, взаємодії з об’єктами або при отриманні шкоди. Такий підхід дозволяє уникнути надмірної кількості постійно активних елементів HUD (head-up display), що могло б негативно вплинути на імерсивність гри;

– оскільки проєкт передбачає використання як традиційного керування, так і VR-платформи, UI було розроблено з урахуванням потреб кожної з них. Елементи інтерфейсу масштабуються, переміщуються або змінюють формат відповідно до типу пристрою. Наприклад, у VR режимі використовуються 3D UI об’єкти у просторі сцени, тоді як у класичному режимі застосовується екранний Canvas з двовимірним рендерингом;

– структура меню, логіка взаємодії, розташування кнопок і підказок ґрунтуються на поширених шаблонах, до яких звикли гравці. Це знижує поріг входу для нових користувачів та підвищує загальну ефективність навігації. Наприклад, кнопка «Назад» завжди розташована в одному й тому ж куті меню, а індикатори дій мають уніфіковану стилізацію, що полегшує сприйняття й запам’ятовування їх функцій.

Для реалізації UI в Unity було використано компонент Canvas як основний контейнер для всіх графічних елементів інтерфейсу. У класичному варіанті гри застосовано режим рендерингу «Screen Space – Overlay», що забезпечує накладення інтерфейсу безпосередньо поверх ігрового простору. Це рішення є оптимальним для традиційних платформ, де інтерфейс завжди має бути добре помітним та читабельним незалежно від змісту сцени. Для VR-режиму, де взаємодія з UI потребує фізичної присутності у віртуальному середовищі, було реалізовано World Space UI – тобто UI-елементи, що існують як частина ігрової сцени у вигляді тривимірних об'єктів. Наприклад, інтерфейс інвентарю закріплюється на лівій руці гравця та реагує на його рухи.

Одним із ключових елементів став головний екран меню, який включає типові розділи: «Продовжити», «Нова гра», «Опції» та «Вийти». Для побудови структури меню було використано компоненти Vertical Layout Group, Button, а для текстових елементів – TextMeshPro, який дозволяє застосовувати стилізовані шрифти та створювати більш гнучке форматування. Меню має плавну анімацію переходів між екранами, що реалізовано за допомогою Animator та кастомної системи переходів сцен.

Іншим важливим компонентом став ігровий HUD (Head-Up Display), який інформує гравця про поточний стан персонажа: кількість здоров’я, енергії, активні предмети, індикатори статусу тощо. Для забезпечення гнучкості та динамічного оновлення інтерфейсу було застосовано патерн ScriptableObject для збереження даних і EventSystem для реагування на зміни у цих даних. Завдяки такій структурі, інтерфейс миттєво оновлюється без необхідності прив’язки до конкретних об’єктів сцени.

Окремо було реалізовано VR-меню дій, що є частиною UX-дизайну спеціально для VR-гравця. Це меню побудовано за допомогою XR Interaction Toolkit та включає інтерактивні панелі, які з’являються при певних жестах або натисканні кнопок. Меню розміщується у просторі поблизу руки гравця, зазвичай – на уявному «наручному дисплеї», що створює ефект природної взаємодії. Замість традиційних кнопок миші, взаємодія здійснюється променем (XR Ray Interactor), який дозволяє гравцеві у VR «торкатися» кнопок на відстані.

Завдяки описаним рішенням, інтерфейс гри адаптований для всіх платформ, а користувацький досвід лишається зручним, інтуїтивним і привабливим як у традиційному, так і VR-середовищі.

3.4 Тестування та налагодження інформаційно-комп’ютерної системи

Слідуючи процесу розробки згідно налагодженого фреймворку, тестування нашого проекту постійно відбуваються під час та після закінчення роботи над сегментом проекту. Що відбувається в процесі тестування на цьому етапі, це зазвичай комплексна робота над покращенням ігрового процесу. Будується рівень гри з розроблених об’єктів, відбувається короткий плейтест, після чого ведеться виправлення помилок що виникли під час гри та коригування ігрових механік та рівня.

Таким чином перший плейтест допоміг зкоригувати позиції певних об’єктів на рівні таким чином, щоб гравець не міг потрапити в пастку softlock

(ця проблема ідентифікується неможливістю уникнути класифікованої проблеми традиційними методами, через що гравець змушений перезапускти завантаження гри або навіть усю ігрову сесію).

По завершенню розробки прототипу було проведено плейтест зкомпільованого білда гри, у якому приймали участь 5 користувачів, 2 користувачі провели також тестування ігрового процесу у режимі VR кооперативу, інші користувачі тестували гру без VR шолому. Тестування не виявили ламаючих гру помилок, також не було виявлено помилок що перешкоджали ігровому процесу, проте тестувальниками було зроблено кілька зауважень стосовно балансних рішень у грі, а також певні зауваження стосувались збиваючих з пантелику проблем у дизайні рівня та навчальних підказок.

Виправлення помилок виявлених під час розробки зачасту стосувались цих проблем:

– порядок ініціалізації компонентів. Траплялись ситуації коли через те що певний компонент (до прикладу Action Editor який вимагає зв’язок з класом Interaction Item) ініціалізувався раніше чим компонент що залежить від його попередньої ініціалізації. Виправлення цієї проблеми передбачає локальну реорганізацію порядку ініціалізації, що дозволяє уникнути цієї проблеми;

– змінна що повертає null reference. Ця проблема виникає коли певна змінна в класі що інінціалізується, потребує визначення. Вирішення цієї проблеми передбачає встановлення пропущених значень у редакторі, написання алгоритму ініціалізації всередині функції, або ініціалізація альтернативної опції під час виклику об’єкта якщо повернення null було очікуваним;

– помилка в визначенні правильної функції для ініціалізації. Якщо для змінної або функції має відбуватись виклик лише один раз за визначеної умови, тоді замість використання методів активного оновлення Update, FixedUpdate та LateUpdate не рекомендується, адже ці функції активуються з дуже короткою періодичністю що приводить до помилок накшталт хибної

реєстрації події (коли було передачено що певна змінна boolean буде змінена лише на період одного оновлення або для використання 1 раз всередині кадру, функції оновлення можуть випадково повернути зміну значення кілька разів за декілька швидкоплинних кадрів), виправлення такої помилки передбачає використання співпроцедур з очікуванням наступного кадру або таймеру для повернення кінцевого значення, або ініціалізація функції/значення за межами функцій оновлення.

Щодо виправлення помилок що стосуються балансу та незручностей створених під час проведення плейтестів, більшість з таких помилок передбачає простої підміни чисел або певних доналаштувань всередині рівня. Такий результат можна досягти завдяки грамотній структурі проекту, адже архітектура побудована за допомогою дата-контейнерів майже не вимагає втручань всередину уже написаного коду.

3.5 Перспективи подальшого розвитку проекту

На момент виконання кваліфікаційної роботи бакалавра, проект знаходиться на етапі ранньої розробки, відбуваються перші тестування прототипу, і в цілому процес який можна охарактеризувати як «затвердження ідеї». У період близько середини-кінця червня місяця, поруч з презентацією проекту на захисті кваліфікаційної роботи, запланований випуск проекту у ранній альфа реліз на платформі itch.io, що буде супроводжуватись початком медіа кампанії проекту у соціальних мережах. На момент презентації проекту передбачено показ початкової демо локації гри, у якому передбачено можливості дослідження рівня, знайомство з базовою концепцією та механіками гри, і також альтернативна секція локації для тестування ідей ігрового процесу у режимі асиметричного VR кооперативу.

Перші відгуки отримані під час плейтесту стали і досі стають ідеями що будуть супроводжувати процес розвитку проекту, зокрема в планах є залучення до проекту товариства що стане головним стержнем у процесі аудіо-дизайну

гри. Цільові платформи для релізу проекту на старті передбачають Windows, Mac OS та Linux системи, з перспективою на випуск версій для інших систем.

Також важливо адресувати велику кількість планів на розробку проекту що не буде включено чи продемонстровано на момент презентації проекту, адже робота над проектом в цілому далека від офіційного завершення, і на мою думку висвітлення масштабних нереалізованих ідей може вийти негативними наслідками для проекту, що проявляються у невиконаних обіцянках та зірваних сроках роботи над проектом, тому важливо на початку проекту такі речі уникати. Проте навіть зараз, проект демонструє потенціал для розвитку що можна зрозуміти зрозуміти з проведених тестувань. Реалізація запланованих напрямків розвитку дозволить вивести на ринок багатообіцяючий проект, що дозволить при інтеграції грамотних стратегій розвитку та маркетингу вивести гру на популярні платформи дистрибуції де проект продовжуватиме розвиватись у передбаченому напрямку.

Висновки до розділу 3

Під час розробки проекту було здійснено реалізацію інтерактивного проекту пригодницької гри з елементами метроїдванії, платформера та Hack’n’Slash, з підтримкою асиметричного кооперативного режиму, де один з гравців використовує VR-шолом, а інший – традиційні засоби введення. Гру реалізовано на ігровому рушії Unity з використанням його стандартного інструментарію, що забезпечує належну гнучкість у побудові сцен, реалізації геймплейних механік і UI/UX.

Розроблена архітектура гри передбачає модульний поділ функціональності, зокрема реалізовано незалежні системи керування, взаємодії з об’єктами, бойової системи та системи квестів. Асиметричний кооператив реалізовано в межах одного пристрою, де обидва гравці взаємодіють у спільному віртуальному середовищі з різними засобами керування.

Під час тестувань, проведених перед компіляцією білда гри для плейтесту, було виявлено низку технічних недоліків та аспектів, які було вирішено, проте які заслуговують уваги через ризик їх повторення у майбутньому. Зокрема, зафіксовано наступні типові помилки:

– некоректний порядок ініціалізації залежних компонентів, що іноді призводило до відсутності взаємодії між об’єктами;

– випадки отримання null reference при виклику функцій, пов’язаних з неініціалізованими змінними;

– хибна логіка одноразових викликів подій при використанні функцій типу Update, що вирішувалося через запровадження співпроцедур або таймерів.

Крім технічних аспектів, під час проведення плейтестів було отримано зауваження тестувальників, які в більшості стосувалися недосконалого балансування рівнів, заплутаного місцями дизайну окремих ділянок локації гри, а також недолік інтуїтивних підказок, що вплинуло на загальну зрозумілість ігрового процесу.

Отримані результати свідчать про загальну працездатність реалізованої архітектури та ефективність застосованого технічного підходу, однак також вказують на напрямки подальшого вдосконалення, зокрема щодо:

– оптимізації ігрових процесів;

– покращення інтерфейсних рішень;

– доопрацювання логіки взаємодії з користувачем.

Таким чином, реалізована ігрова система становить завершену функціональну основу для подальшого розвитку проекту.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено програмне забезпечення у вигляді пригодницької гри з елементами метроїдванії, 3D-платформеру та Hack’n’Slash, з реалізацією асиметричного VR-кооперативу. Основним інструментом розробки став рушій Unity, у поєднанні з Blender для створення 3D-моделей персонажів, предметів та середовища. У ході роботи виконано всі поставлені завдання та досягнуто мету дослідження: проведено огляд ігрових жанрів, проаналізовано інструменти реалізації, обґрунтовано вибір технологій, спроєктовано архітектуру гри та виконано повноцінну реалізацію ключових її складових.

На основі проведеного аналітичного огляду можна стверджувати, що технології ігрової взаємодії з залученням VR-елементів, особливо в асиметричному форматі, становлять перспективний напрям у розробці інтерактивного програмного забезпечення, зокрема ігор. Такий підхід відкриває нові можливості для занурення гравців та забезпечує унікальний ігровий досвід, що поєднує фізичну активність та класичні засоби керування.

Концепція реалізації асиметричного кооперативу була побудована на поєднанні різних засобів взаємодії – VR-шолома та традиційних засобів керування (геймпад, клавіатура і миша), з окремою логікою для кожного з гравців. Було враховано вимоги до зручності керування, гнучкості налаштувань, адаптивності до типу гравця та середовища гри.

Геймдизайн спирався на приклади кращих практик жанрів Hack’n’Slash та метроїдванія, з відповідним балансуванням бойової системи, структури рівнів та ігрового прогресу. Функціональність гри охоплює механіки бою, переміщення, збору предметів, виконання квестів, взаємодії з NPC, елементи шифрування логіки переходу між рівнями, побудову ігрового UI та реалізацію взаємодії між VR- і гравцями класичної схеми керування.

Схеми керування було створено за принципом абстракції дій клавіш, таким чином у грі є можливість для адаптації під будь-який сучасний пристрій

керування за потребою. Також абстракція схеми керування дозволяє виробити патерни програмування систем у відповідності до вимог інтуїтивності та комфорту ігрового процесу.

Архітектуру спроєктовано таким чином, щоб забезпечити гнучке масштабування та можливість подальшого вдосконалення модулів. Особливу увагу приділено оптимізації роботи ігрових сцен, налаштуванню колізій, тригерів та логіки подій.

Використовуючи визначені патерни та архітектуру розробки, було реалізовано системи для руху, бойову систему, систему інвентарю, логіку для динамічних об’єктів, логіку для предметів, систему логіки для НІПів та систему взаємодії VR гравця з оточенням. Для реалізації гри використано вбудовані у Unity засоби VR (OpenXR та XR Interaction Toolkit) та набір власних C#-скриптів для управління геймплейною логікою.

Було використано Blender для створення та оптимізації 3D-контенту, зокрема моделей персонажів, предметів та об’єктів у грі. Завдяки використанню методик для low-poly моделювання, топологія моделей для гри не є проблемою для продуктивності гри, розкриваючи також цікаві художні бачення через обрану стилізацію.

За допомогою інструментарю Krita було створенно набір спрайтів та текстур, більшість з яких було використано для створення UI елементів гри. Завдяки дотриманню принципів мінімалізму, контекстності, адаптивності та інтуїтивності, зроблений UI є якісним доповненням ігрового досвіду.

Було проведено апробацію роботи системи через серію плейтестів, під час яких перевірено функціонування ключових механік, взаємодію між гравцями, адекватність керування та стабільність гри. Гру протестовано на одній машині із двома типами гравців, де система показала стабільну роботу та належний рівень взаємодії між учасниками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Batman Arkham Collection. Steam. URL: https://store.steampowered.com/sub/320795/ (date of access: 04.02.2025).

2. Bebko A. O., Troje N. F. Experimental design with Unity game engine. Journal of vision, 2020, Vol. 20, no. 11, p. 810. URL: https://doi.org/10.1167/jov.20.11.810 (date of access: 01.03.2025).

3. Bevy engine. Bevy Engine. URL: https://bevyengine.org/ (date of access: 15.03.2025).

4. Blender.org. Home of the Blender Project. URL: https://www.blender.org/ (date of access: 26.04.2025).

5. C# tutorial. W3Schools. URL: https://www.w3schools.com/cs/index.php (date of access: 26.04.2025).

6. Carly and the Reaperman – Escape from the Underworld. Steam. URL: https://store.steampowered.com/app/547480/Carly_and_the_Reaperman__Escape_from_the_Underworld/ (date of access: 04.02.2025).

7. Devil May Cry. Capcom. URL: https://www.devilmaycry.com/en/ (date of access: 04.02.2025).

8. Dualshock scheme. Reddit. URL: https://preview.redd.it/psstdd45dfo61.jpg?width=1080&crop=smart&auto=webp&s=7be94bee0c927d57ae5cc7a87c853b1f5741fa5a (date of access: 01.03.2025).

9. Godot Engine. Godot Engine. URL: https://godotengine.org/ (date of access: 15.03.2025).

10. Hollow Knight. Team Cherry. URL: https://www.hollowknight.com/ (date of access: 04.02.2025).

11. Krita – цифрове малювання. Творча свобода. Krita. URL: https://krita.org/uk/ (date of access: 26.04.2025).

12. Lee T. Unity engine dissection: improvement points in comparison between Unity engine and open-source engines. Asia-pacific journal of convergent

research interchange. 2024. Vol. 10, no. 11. P. 449–458. URL: https://doi.org/10.47116/apjcri.2024.11.33 (date of access: 15.03.2025).

13. Marvel’s Spider-Man Remastered. Steam. URL: https://store.steampowered.com/app/1817070/Marvels_SpiderMan_Remastered (date of access: 04.02.2025).

14. Panoptic. Steam. URL: https://store.steampowered.com/app/541930/Panoptic/ (date of access: 04.02.2025).

15. Perkett W. C# Programming Reference Library. Independently Published, 2022. (date of access: 26.04.2025).

16. Pseudoregalia. Steam. URL: https://store.steampowered.com/app/2365810/Pseudoregalia/ (date of access: 04.02.2025).

17. Spark the Electric Jester 3. Steam. URL: https://store.steampowered.com/app/1629530/Spark_the_Electric_Jester_3/ (date of access: 04.02.2025).

18. Square Enix – NieR: Automata. URL: https://www.square-enix-games.com/games/nier-automata (date of access: 04.02.2025).

19. The Big Catch. Steam. URL: https://store.steampowered.com/app/2146290/The_Big_Catch/(date of access: 04.02.2025).

20. The most powerful real-time 3D creation tool – Unreal Engine. URL: https://www.unrealengine.com/ (date of access: 15.03.2025).

21. Unity documentation. Unity Docs. URL: https://docs.unity.com/ (date of access: 26.04.2025).

22. Unity Learn. Unity Learn. URL: https://learn.unity.com/ (date of access: 26.04.2025).

23. Unity real-time development platform. 3D, 2D, VR & AR engine. Unity. URL: https://unity.com/ (date of access: 04.02.2025).

24. VA Proxy. Steam. URL: https://store.steampowered.com/app/2063390/VA_Proxy/ (date of access: 04.02.2025).

ДОДАТКИ

Додаток А – Таблиця базової архітектури

Клас	Призначення	Повторне використання	Доповнюваність	Продуктивність
InputActions	Уловлювання клавіш вводу	Унікальний елемент, не потребує дублікатів	Дозволяє додавати нові елементи керування вільно	Вбудоване рішення, продуктивне
VRController	Керування VR аватаром	Унікальний елемент, не потребує дублікатів	Унікальний, нові доповнення отримує сам для себе	Вимагає потужностей для обладнання, сам елемент є продуктивним
InputManager	Обробка отриманого вводу	Унікальний елемент, не потребує дублікатів	У зв'язку з InputActions може вільно доповнюватись	Продуктивний елемент, працює з простими типами даних
InputCache	Тимчасове збереження останньої комбінації клавіш	Допоміжний елемент InputManager, не потребує дублікатів	Доповнює InputManager, від цього ж класу і доповнюється	Продуктивний, перетворює дані InputManager у прийнятниі для інших класів
ActionManager	Обробка отриманої комбінації клавіш, підготовка та запуск відповідної дії для персонажа	Використовується як гравцем так і НІПами, має трохи різну логіку для цих об'єктів завдяки наслідуванню	Зроблений для обробки вхідних запитів на дію, може доповнитись типами дій проте сам є менеджером нових даних	Продуктивний, проте працює з складнішими типами даних (анімаційні кліпи, текстові рядки)
InteractionItem	Отримання запитів від ActionManager, виконання дії викликаної в анімаційному кліпі	Використовується як гравцем так і НІПами, допоміжний елемент ActionManager, репрезентує предмет в руках персонажа	Доповнюється новими методами взаємодії гравця з ігровим світом	Продуктивний, викликається зовні адже не потребує постійного оновлення стану
ProjectileShooter	Здійснює створення об'єкту projectile	Наслідувач InteractionItem	Є доповненням InteractionItem	Імплементує функціонал що зберігає запущені снаряди у неактивному стані коли вони не потрібні, замість знищення яке має наслідки
RaycastShooter	Здійснює запуск Raycast'у у певному напрямку	Наслідувач InteractionItem	Є доповненням InteractionItem	Продуктивний, використовує проту функцію фізичних обрахувань
ColliderSetter	Здійснює активацію Collider компонента предмету у руках персонажа для зчитування колізій	Наслідувач InteractionItem	Є доповненням InteractionItem	Продуктивний, адже керує лише активацією компонента об'єкту
EventShooter	Здійснює активацію Event'у для передачі його "слухачам"	Наслідувач InteractionItem	Є доповненням InteractionItem	Продуктивний, використовує шаблон event'у з динамічними слухачами
ColliderProjectile	"Снаряд" що запускається персонажем, виконує зчитування колізій	Призначений для масштабування, є на кожному об'єкті "Снаряду"	Створений як власний тип об'єкта, є самодостатнім проте може доповнитись додатковими функціями	Може бути проблемним у великих кількостях, оскільки виконує активні фізичні обрахування

Продовження таблиці – Архітектура класів програми

Клас	Призначення	Повторне використання	Доповнюваність	Продуктивність
StatsManager	Зберігає параметри персонажа, в разі зіткнення з colider'ом що передає ефекти, приймає ефекти та проводить обчислення параметрів	Використовується гравцем, НІПами, а також іншими знищенними об'єктами у грі	Може доповнитись новими параметрами персонажа та алгоритмами ефектів	Продуктивний, бо зберігає прості типи даних та виконує активності лише за запитом
Item	Контейнер даних про предмет у грі	Контейнер даних з можливістю створення копій даних з шаблону	Є самодостатнім шаблоном для даних	Простий контейнер даних не створює проблем з продуктивністю
ItemUsable	Зберігає додаткові параметри разом з тими що має Item, для предметів які можна використовувати	Наслідувач Item, розширена версія з тим же призначенням	Є самодостатнім шаблоном для даних	Простий контейнер даних не створює проблем з продуктивністю
ComboMove	Контейнер даних про дію у грі	Контейнер даних з можливістю створення копій даних з шаблону	Є самодостатнім шаблоном для даних	Простий контейнер даних не створює проблем з продуктивністю
QuestManager	Приймає та зберігає квести	Унікальний елемент для гравця, проте технічно може використовуватись іншими об'єктами як сховище даних	Може бути доповнений новими елементами для обробки квестів	Продуктивний, служить лише елементом для збереження та активації квестів
Quest	Виконує логіку самообслуговування, зберігає завдання	Створений для масштабування, є шаблоном	Може бути доповнений новими алгоритмами	Продуктивний бо не виконує складних обчислень
QuestGoal	Слухає обрані event'и, здійснює логіку по виконанню квеста	Створений для масштабування, є шаблоном	Є самодостатнім шаблоном, проте може бути доповнений вихідними взаємодіями	Продуктивний бо динамічно слухає event'и, здійснюючи не важку перевірку вхідних даних що надходять не надто часто
NavManager	Керує поведінкою НІП	Використовується кожним НІПом у грі	Може бути доповнений новими алгоритмами	Може бути проблемним у великих кількостях, оскільки виконує обрахування для кожного об'єкта, проте регулюється системою потоку світу (не активний за межами провантаженої зони)
PlayerManager	Керує рухами гравця	Унікальний для гравця елемент	Може бути доповнений новими алгоритмами, проте створений бути простішим елементом	Продуктивний, здійснює прості фізичні операції на одному об'єкті одночасно

Продовження таблиці – Архітектура класів програми

Клас	Призначення	Повторне використання	Доповнюваність	Продуктивність
AvatarSetup	Зберігає дані про положення об'єктів гравця, потрібних для інших класів	Використовується на усіх персонажах	Може бути доповнений новими типами елементів персонажа	Продуктивний, просто зберігає дані
GameStateManager	Керує станом гри: плином штучного темпу та циклу дня і ночі	Унікальний для об'єкта менеджера елемент	Може бути доповнений алгоритмами для інших класів	Продуктивний для себе бо виконує прості обчислення, проте може створити проблему якщо провантажено забагато НІПів
RoutineSetup	Контейнер даних, зберігає повсякденні завдання для НІПів	Контейнер даних з можливістю створення копій даних з шаблону	Є самодостатнім шаблоном для даних	Простий контейнер даних не створює проблем з продуктивністю
TownManager	Зберігає дані про об'єкти, необхідні НІПам для здійснення повсякденних завдань	Використовується на усіх об'єктах таборів	Може бути доповнений новими механіками та логікою для взаємодій	Продуктивний адже виконує свої алгоритми не часто

Додаток Б – Приклад класу компонента «PlayerManager» для гравця

```
using UnityEngine;

public class PlayerManager : MonoBehaviour
{
      [Header ("External components")]
      InputManager inputManager;
      Rigidbody pRigidbody;
      Transform cameraObject;

      [Header ("Internal values")]
      private Vector3 moveDirection;

      [Header ("Movement modifiers values")]
      public float moveSpeed;
      public float rotationSpeed;

      private void Awake()
      {
            pRigidbody = GetComponent<Rigidbody>();
            cameraObject = Camera.main.transform;
            inputManager = InputManager.instance;
      }

      private void FixedUpdate()
      {
            PlayerMovement();
            PlayerRotation();
      }

      private void PlayerMovement()
      {
            moveDirection = cameraObject.forward *
      inputManager.movementInput.y;
            moveDirection = moveDirection + cameraObject.right *
      inputManager.movementInput.x;
            moveDirection.Normalize();
            moveDirection.y = 0;

            pRigidbody.velocity = new Vector3(moveDirection.x * moveSpeed,
      pRigidbody.velocity.y, moveDirection.z * moveSpeed);

      }

      private void PlayerRotation()
      {
            Vector3 targetDirection = Vector3.zero;
```

```
            targetDirection = cameraObject.forward *
      inputManager.movementInput.y;
            targetDirection = targetDirection + cameraObject.right *
      inputManager.movementInput.x;
            targetDirection.Normalize();
            targetDirection.y = 0;

            if (targetDirection == Vector3.zero)
                  targetDirection = transform.forward;

            Quaternion targetRotation =
      Quaternion.LookRotation(targetDirection);
            Quaternion playerRotation =
      Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed
      * Time.deltaTime);

            pRigidbody.MoveRotation(playerRotation);
      }
}
```

кінець лістингу Б.1

Додаток В – Зразок коду скрипта InputManager

```
using UnityEngine;

public class InputManager : MonoBehaviour
{
      public static InputManager instance {get; private set;}
      private void Awake() => instance = this;

      InputActions inputActions;

      public Vector2 movementInput;
      public bool isJumpHeld;     public bool wasJumpHeld;
      public bool isJumpPressed;

      public bool isInventoryHeld;
      public bool isMainActionHeld; private bool wasMainActionHeld;
      public bool isMainActionPressed;
      public  bool  isHeavyActionHeld;  private  bool  wasHeavyActionHeld;
      public bool isHeavyActionPressed;
      public  bool  isCancelActionHeld;  private  bool  wasCancelActionHeld;
      public bool isCancelActionPressed;

    private void OnEnable()
        {
            if (inputActions == null)
            {
                inputActions = new InputActions();

                inputActions.Movement.WASD.performed     +=     i     =>
movementInput = i.ReadValue<Vector2>(); //WASD

                inputActions.Movement.Jump.performed += i => isJumpHeld =
true; //Jump
                inputActions.Movement.Jump.canceled += i => isJumpHeld =
false;

                inputActions.Interaction.Inventory.performed   +=   i   =>
isInventoryHeld = true; //Inventory
                inputActions.Interaction.Inventory.canceled   +=   i   =>
isInventoryHeld = false;

                inputActions.Interaction.MainAction.performed  +=  i  =>
isMainActionHeld = true; //Main action
                inputActions.Interaction.MainAction.canceled   +=   i   =>
isMainActionHeld = false;

                inputActions.Interaction.HeavyAction.performed  +=  i  =>
isHeavyActionHeld = true; //Heavy action
```

```
                inputActions.Interaction.HeavyAction.canceled  +=  i  =>
isHeavyActionHeld = false;

                inputActions.Interaction.CancelAction.performed  +=  i  =>
isCancelActionHeld = true; //Cancel action
                inputActions.Interaction.CancelAction.canceled  +=  i  =>
isCancelActionHeld = false;
            }

            inputActions.Enable();
        }

    private void OnDisable() => inputActions.Disable();

    private void FixedUpdate()
        {
            isJumpPressed = isJumpHeld && !wasJumpHeld;
            wasJumpHeld = isJumpHeld;

            isMainActionPressed = isMainActionHeld && !wasMainActionHeld;
            wasMainActionHeld = isMainActionHeld;

            isHeavyActionPressed       =       isHeavyActionHeld       &&
!wasHeavyActionHeld;
            wasHeavyActionHeld = isHeavyActionHeld;

            isCancelActionPressed      =      isCancelActionHeld      &&
!wasCancelActionHeld;
            wasCancelActionHeld = isCancelActionHeld;
        }
}
```

кінець лістингу В.1