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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка 3D-гри “FANTASY POLY SURVIVORS” у жанрі vampire like"»

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

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

_____ Віталій КАСЯНЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Віталій КАСЯНЕНКО

Керівник: Максим КУКЛІНСЬКИЙ
канд. техн. наук

Рецензент: _____

Київ 2026

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

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Касяненку Віталію Валерійовичу

1. Тема кваліфікаційної роботи: «Розробка 3D-гри “FANTASY POLY SURVIVORS” у жанрі vampire like"»
керівник кваліфікаційної роботи канд. тех. наук Максим КУКЛІНСЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.
2. Строк подання кваліфікаційної роботи «29» травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку комп’ютерних ігор жанру survival-like, принципи побудови ігрових механік та систем прогресії персонажа, документація ігрового рушія Unity, мови програмування C#, середовищ розробки Microsoft Visual Studio та Visual Studio

Code, а також програмного забезпечення Blender для створення тривимірних моделей ігрових об’єктів.

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

1. Аналіз предметної області та конкурентів на ринку гри

2. Засоби реалізації застосунку

3. Програмна реалізація та опис функціонування застосунку Fantasy Poly Survivors

4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма прецедентів.
5. Діаграмі діяльності.
6. Діаграма послідовності.
7. Діаграма класів.
8. Відео демонстрація застосунку.
9. Апробація результатів дослідження.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд існуючих ігор та засобів їх реалізації, методів і технологій жанру гри vampire-like	08.03-14.03.2026	
4	Проектування програмного застосунку	15.03-29.03.2026	
5	Програмна реалізація застосунку	30.03-23.04.2026	
6	Тестування застосунку	24.04-30.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.05-07.05.2026	
8	Розробка демонстраційних матеріалів	08.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Віталій КАСЯНЕНКО

(підпис)

Керівник

кваліфікаційної роботи _____ Максим КУКЛІНСЬКИЙ

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: __68__ стор., ___1__ табл., __9__ рис., __20__ джерел.

Мета роботи – підвищення користувацького досвіду в survival-іграх, шляхом розроблення 3D гри «Fantasy Poly Survivors» на рушії Unity з використанням low-poly графіки, системи хвиль ворогів та RPG-механік

Об'єкт дослідження – процес виживання в іграх стилі vampire-like

Предмет дослідження – методи та засоби реалізації ігрових механік, системи прогресії персонажа, low-poly графіки, архітектури гри та бойової взаємодії на рушії Unity із використанням мови програмування C#.

Короткий зміст роботи: У роботі проаналізовано існуючі ігри жанру survival-like та їх ключові механіки в контексті розробки гри "Fantasy Poly Survivors". Розглянуто основні підходи до побудови ігрових систем, систем прогресії та взаємодії гравця з ігровим світом.

Розроблено концепцію та алгоритми роботи гри, а також реалізовано основні функціональні можливості, зокрема: керування персонажем, бойову систему, систему хвиль ворогів, отримання досвіду, вибір покращень та відображення ігрового інтерфейсу.

Проведено функціональне та модульне тестування розробленого програмного продукту, яке підтвердило коректність роботи основних ігрових механік та відповідність реалізації поставленим вимогам.

У процесі розробки використано ігровий рушій Unity та мову програмування C#, а також Microsoft Visual Studio як основне середовище програмування.

Сферою використання застосунку є мобільні платформи, освітні проєкти (тренажери реакції), тренінги для вивчення рушіїв Unity.

КЛЮЧОВІ СЛОВА: survival-like, LOW POLY, Survivor-like, Brotato, Megabon, Fantasy poly survivors, Fantasy Poly Survivors, survivors, Survivors.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі комп’ютерні ігри є однією з найпоширеніших форм інтерактивного дозвілля. Вони виконують не лише розважальну функцію, але й сприяють розвитку когнітивних навичок, таких як швидкість реакції, концентрація уваги та прийняття рішень у динамічних умовах.

Окреме місце в сучасній індустрії займають ігри жанру Survivor-like, які сформувалися як самостійний напрям після значного комерційного успіху гри Vampire Survivors. Цей жанр поєднує елементи екшену, рольової прогресії та виживання в умовах постійного збільшення складності та кількості противників.

Ігровий процес у таких проєктах базується на коротких, але насичених сесіях, у яких гравець поступово розвиває персонажа, адаптуючись до дедалі складніших умов. Це формує інтенсивний та динамічний ігровий досвід.

Разом із тим, Survivor-like ігри часто характеризуються певною повторюваністю базових сценаріїв, що може впливати на зниження інтересу гравців після тривалого використання. У зв’язку з цим важливого значення набуває підхід до покращення загального ігрового досвіду, зокрема через візуальне оформлення, варіативність персонажів та зручність взаємодії з грою.

Метою даної дипломної роботи є розробка ігрового програмного продукту «Fantasy Poly Survivor», який реалізує базові принципи жанру Survivor-like та орієнтований на покращення ігрового досвіду користувача через систему класів персонажів, візуальний стиль Low Poly та збалансований ігровий процес.

У межах роботи розглядаються сучасні підходи до створення ігрових застосунків, використання рушія Unity та мови програмування C#, а також реалізація ключових компонентів гри, таких як ігрова логіка, система прогресії та поведінка противників.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення та особливості ігор жанру Survivor-like

Ігри жанру Survivor-like є сучасним піджанром екшен-ігор, що сформувався на перетині аркадних шутерів, рольових ігор та механік виживання. Даний напрямок отримав активний розвиток у результаті спрощення базового керування та одночасного ускладнення ігрового процесу через збільшення кількості противників і варіативності їхньої поведінки.

Однією з ключових особливостей жанру є автоматизована бойова система. У таких іграх гравець, як правило, не керує безпосередньо кожною атакою персонажа, а зосереджується на переміщенні, виборі позиціонування та стратегічному розвитку свого героя. Це дозволяє зменшити поріг входження для нових гравців і водночас створити глибокий ігровий процес через систему прогресії.

Основна мета Survivor-like ігор полягає у виживанні протягом якомога довшого часу в умовах безперервного збільшення складності. З кожною хвилею противників інтенсивність ігрового процесу зростає, що змушує гравця постійно адаптуватися до нових умов та приймати швидкі рішення щодо розвитку персонажа.

Важливим етапом у формуванні жанру стала гра Vampire Survivors, яка продемонструвала ефективність поєднання мінімалістичного керування, автоматичних атак та глибокої системи випадкових покращень. Незважаючи на просту візуальну складову, проєкт зміг забезпечити високий рівень залученості гравців завдяки динаміці та постійному відчуттю прогресу.

Система розвитку персонажа є центральним елементом жанру. Вона базується на отриманні досвіду, підвищенні рівнів та виборі покращень, які впливають на бойові характеристики, швидкість пересування, радіус атак або інші

параметри. Комбінація цих покращень формує унікальний стиль проходження для кожної ігрової сесії.

Ще однією важливою характеристикою є висока щільність противників на ігровій карті. Вороги з’являються у великій кількості та поступово оточують гравця, створюючи постійний тиск і необхідність підтримувати баланс між атакою та виживанням.

Окремо слід відзначити високу реіграбельність таких ігор. Завдяки випадковому набору покращень, різним комбінаціям здібностей та змінним умовам кожне проходження має унікальний характер, що стимулює повторне проходження гри.

Разом із цим, Survivor-like проєкти часто мають обмежену глибину довготривалого контенту. Через повторюваність базового циклу “виживання – отримання покращень – збільшення складності” інтерес гравців може знижуватися після тривалого періоду гри. Саме тому важливим аспектом розробки є покращення загального ігрового досвіду через різноманітність персонажів, візуальні рішення та баланс ігрових систем.

У рамках даної дипломної роботи гра «Fantasy Poly Survivor» реалізує основні принципи жанру Survivor-like, доповнюючи їх класовою системою персонажів, фентезійним сетингом та тривимірною Low Poly графікою, що спрямовано на підвищення якості сприйняття та різноманітності ігрового процесу.

1.2 Історія та еволюція жанру Survivor-like

Жанр Survivor-like є відносно новим явищем в індустрії відеоігор, однак його формування базується на тривалому еволюційному розвитку суміжних жанрів, зокрема екшен-ігор, рольових ігор та проєктів із хвильовою структурою противників.

Перші елементи, що згодом стали основою жанру, можна простежити в класичних аркадних іграх, де гравець протистояв великій кількості ворогів у межах обмеженої ігрової області. Основний акцент у таких проєктах робився на швидкість

реакції та виживання, однак через технічні обмеження того часу кількість об’єктів та складність механік була суттєво обмежена.

З розвитком апаратного забезпечення та ігрових рушіїв у 2000-х роках з’явилася можливість створювати більш складні ігрові системи з великою кількістю одночасних об’єктів та взаємодій. У цей період активно розвивалися рольові екшени та hack-and-slash ігри, які сформували базові принципи розвитку персонажа, накопичення досвіду та використання різних типів озброєння.

Важливим етапом у становленні жанру став вихід гри Vampire Survivors, яка запропонувала новий підхід до побудови ігрового процесу. Основною ідеєю стало поєднання автоматичних атак, виживання в умовах постійного тиску ворогів та поступового посилення персонажа через систему випадкових покращень.

Цей проєкт продемонстрував, що навіть за умов мінімалістичної графіки можливо створити глибокий та захоплюючий ігровий досвід, що призвело до формування окремого піджанру Survivor-like (також відомого як “Vampire Survivors-like”).

Після цього жанр почав активно розвиватися серед інді-розробників. З’явилися численні ігри, які використовують подібну базову формулу, але розширюють її за рахунок нових механік, таких як класи персонажів, системи активних і пасивних здібностей, боси та процедурна генерація елементів ігрового процесу.

Важливу роль у розвитку жанру також відіграли рольові та екшн-RPG ігри, які вплинули на формування системи прогресії, луту та побудови білдів персонажа. Саме з цих жанрів Survivor-like проєкти запозичили ідею поступового посилення персонажа та формування різних стилів проходження.

Сучасні представники жанру все частіше використовують тривимірну графіку, зокрема стиль Low Poly, який дозволяє поєднати візуальну привабливість із високою продуктивністю навіть при великій кількості ворогів на екрані. Це особливо важливо для ігор, де одночасно можуть бути активними десятки або сотні противників.

Разом із цим слід відзначити, що подібні ігри часто мають відносно короткий цикл активного інтересу аудиторії. Після початкового періоду популярності частина гравців може втрачати мотивацію через повторюваність базових ігрових сценаріїв. Це підкреслює важливість роботи над загальним ігровим досвідом, зокрема через візуальну різноманітність, баланс прогресії та зручність взаємодії з грою.

У межах даної дипломної роботи проєкт «Fantasy Poly Survivor» використовує напрацювання жанру Survivor-like та адаптує їх до фентезійного середовища, роблячи акцент на покращенні ігрового досвіду користувача через поєднання класової системи, візуального стилю та збалансованої прогресії.

1.3 Основні елементи та механіки Survivor-like гри у проєкті «Fantasy Poly Survivor»

Ігровий процес у жанрі Survivor-like базується на циклі безперервного виживання гравця в умовах поступового зростання складності. У межах розробленого проєкту «Fantasy Poly Survivor» реалізовано базову структуру даного жанру з акцентом на динамічний бій, автоматизовану систему атак та розвиток персонажа в процесі проходження.

Основною механікою гри є цикл виживання, який полягає у протистоянні гравця хвилям противників. Гравець керує персонажем у тривимірному просторі, уникає зіткнень із ворогами та намагається вижити якомога довше. З часом кількість ворогів збільшується, що призводить до поступового ускладнення ігрового процесу та підвищення вимог до реакції та позиціонування.

Важливою складовою геймплею є система автоматичної атаки. Персонаж самостійно завдає ударів по ворогах у межах своєї зони атаки, без необхідності прямого керування кожною дією. Таким чином, основна увага гравця зосереджується на переміщенні, ухиленні та виборі оптимальної позиції на ігровій карті.

У грі реалізовано систему прогресії персонажа, яка базується на отриманні досвіду за знищення противників. Після накопичення певної кількості досвіду гравець отримує підвищення рівня та можливість обрати покращення, які впливають на ефективність персонажа. Це дозволяє поступово підсилювати героя та адаптувати його до зростаючої складності гри.

Єдиним реалізованим класом персонажа у грі є Berserker. Даний клас орієнтований на ближній бій та характеризується підвищеною витривалістю і здатністю завдавати масових пошкоджень у ближній зоні. Такий підхід дозволяє гравцю активно взаємодіяти з великою кількістю противників та підтримувати динамічний стиль проходження.

Ще одним важливим елементом є система хвиль противників. Вороги з’являються поступово, а їх кількість і щільність збільшуються з плином часу. Це створює постійний тиск на гравця та формує динамічний темп гри, який є характерною ознакою жанру Survivor-like.

Основною метою ігрового процесу є виживання протягом максимально можливого часу. Результат гравця безпосередньо залежить від ефективності його переміщення, вибору моментів для ризику та правильного використання доступних покращень персонажа.

Таким чином, реалізовані механіки формують цілісний ігровий цикл, що відповідає базовим принципам жанру Survivor-like та забезпечує поступове ускладнення ігрового процесу з акцентом на динаміку, виживання та розвиток персонажа.

1.4 Аналіз існуючих ігор жанру Survivor-like на ринку

Гра Vampire Survivors є найбільш впливовим проєктом у формуванні сучасного вигляду жанру Survivor-like. Саме вона заклала базову формулу геймплею, яка сьогодні використовується великою кількістю інді-ігор: автоматичні атаки, нескінченні хвилі противників та поступовий розвиток персонажа в межах одного ігрового забігу.

Ігровий процес побудований таким чином, що гравець не контролює атаки напряму. Замість цього основна увага приділяється переміщенню персонажа по карті та виживанню в умовах постійно зростаючої кількості ворогів. Такий підхід створює унікальну комбінацію простоти керування та високої складності ситуаційного вибору.

Графічно гра виконана у стилі 2D піксель-арту, який навмисно спрощений. Це дозволяє одночасно відображати сотні ворогів без суттєвого навантаження на систему. Локації представлені як відносно великі, але мінімалістичні арени, де основна увага гравця зосереджена не на оточенні, а на динаміці бою.

Важливою складовою є система випадкових покращень. Після підвищення рівня гравець отримує вибір із кількох варіантів розвитку, які можуть змінювати характеристики атак, додавати нові ефекти або підсилювати виживаність. Саме випадковість формує унікальність кожного проходження, оскільки одна й та сама комбінація покращень майже ніколи не повторюється.

Окремо варто відзначити високу інтенсивність пізніх етапів гри. Коли кількість ворогів значно зростає, екран заповнюється великою кількістю ефектів та моделей, що створює хаотичну, але контрольовану ігрову ситуацію.

Рис. 1.1 Приклад інтерфейсу та геймплею гри Vampire Survivors

Гра Brotato є розвитком формули Survivor-like, але з більш структурованим та аркадним підходом до геймплею. На відміну від інших представників жанру, вона робить акцент на коротких, але інтенсивних ігрових сесіях.

Ігровий процес відбувається на невеликій замкненій арені, де гравець протистоїть хвилям ворогів. Основною метою є виживання до кінця хвилі, після чого гравець отримує можливість обрати нову зброю або покращення. Це формує чітку структуру прогресії між бойовими фазами.

Графіка гри виконана у простому 2D стилі з видом зверху**,** але при цьому має кращу читабельність порівняно з багатьма іншими Survivor-like проєктами. Візуальні ефекти чітко розділяють ворогів, атаки та середовище, що позитивно впливає на сприйняття динамічних сцен.

Однією з ключових особливостей є система білдів персонажа. Гравець постійно комбінує зброю та характеристики, створюючи різні стилі гри — від агресивного ближнього бою до дистанційного контролю арени. При цьому випадковість відіграє важливу роль, оскільки доступні покращення змінюються після кожної хвилі.

Також варто відзначити баланс складності. Гра поступово збільшує тиск на гравця, але при цьому зберігає можливість адаптації завдяки швидким ігровим сесіям. Це робить її більш “аркадною” у порівнянні з класичними Survivor-like іграми.

Рис. 1.2 Приклад інтерфейсу та геймплею гри Brotato

Гра Megabonk є менш масовим, але показовим прикладом розвитку жанру Survivor-like у напрямку більш хаотичного та фізично орієнтованого геймплею. Вона демонструє інший підхід до побудови бойових ситуацій, де важливу роль відіграє не лише кількість ворогів, але й взаємодія фізики та простору.

Ігровий процес будується навколо виживання на відкритих аренах, де гравець постійно перебуває під тиском великої кількості противників. Основний акцент робиться на динаміці пересування та реакції, оскільки бойові ситуації часто стають хаотичними через щільність ворогів та ефекти взаємодії.

Графічно гра використовує спрощений 3D стиль, що наближає її до сучасних інді-екшенів. Це дозволяє створювати більш об’ємне середовище та краще відчуття простору в порівнянні з 2D-аналогами.

Локації в Megabonk зазвичай представлені як відкриті арени з мінімальною кількістю перешкод. Це робить геймплей більш швидким і дозволяє гравцю концентруватися на ухиленні та позиціонуванні.

Система випадковості також відіграє значну роль. Вона впливає на появу бонусів, ворогів та можливостей розвитку персонажа, що забезпечує різноманітність кожного проходження.

Особливістю гри є більш “хаотичний” стиль бою, де результат часто залежить не тільки від вибору покращень, але й від ситуаційної реакції гравця в умовах високої щільності ворогів.

Рис. 1.3 Приклад інтерфейсу та геймплею гри Megabonk

Таблиця 1.1

Зведені результати аналізу survival-like

Назва	Vampire Survivors	Brotato	Megabonk	Fantasy Poly Survivors
Тип графіки	2D Pixel Art	2D Cartoon	3D	Low-poly 3D
Тип камери	Top-down 2D	Top-down 2D	Top-down 3D	Top-down 3D
3D-простір	-	-	+	+
Low-poly стиль	-	-	-	+
Система класів персонажа	-	-	-	+
Основні недоліки	2D-простір, перевантаження ефектами	Одноманітність та слабка атмосфера	Проблеми балансу та різноманітності	Відсутні у межах концепції
Інші особливості	Простий pixel-art стиль	Гумористичний стиль та аркадність	Акцент на хаосі та RPG-елементах	Авторський low-poly fantasy стиль, 3D арена, система класів

1.5 Популярність та комерційний успіх ігор жанру Survivor-like

Ігри жанру Survivor-like за короткий період часу стали одним із найбільш помітних напрямків в інді-ігровій індустрії. Їх популярність значною мірою зумовлена простотою базових механік, високою динамікою ігрового процесу та можливістю швидкого залучення гравця без необхідності тривалого навчання.

Поштовхом до комерційного успіху жанру став вихід гри Vampire Survivors, яка продемонструвала, що навіть проєкт із мінімалістичною графікою може досягти значного комерційного результату за рахунок правильно побудованого ігрового циклу. Гра швидко набула популярності на цифрових платформах

розповсюдження, що сприяло формуванню нового піджанру та появі великої кількості подібних проєктів.

Однією з причин комерційного успіху Survivor-like ігор є їхня доступність для широкої аудиторії. Завдяки простому керуванню та зрозумілій меті, а саме виживанню якомога довше, такі ігри не вимагають від гравця попереднього досвіду, або складного навчання. Це дозволяє швидко залучати нових користувачів та забезпечує високий рівень "входу в гру".

Крім того, важливу роль відіграє коротка тривалість ігрових сесій. Більшість Survivor-like ігор побудовані таким чином, що один запуск триває від кількох до десятків хвилин, що робить їх зручними для "швидкої гри". Такий формат добре підходить для сучасного ритму життя гравців і сприяє повторному запуску гри.

Іншим фактором популярності є високий рівень реіграбельності. Завдяки системі випадкових покращень, різним комбінаціям здібностей та змінним умовам кожна ігрова спроба відрізняється від попередньої. Це створює ефект "нескінченного прогресу", який утримує інтерес гравця протягом тривалого часу.

Разом із цим слід зазначити, що Survivor-like ігри часто стикаються з проблемою швидкого насичення контентом. Після певного періоду активної гри базовий ігровий цикл може ставати передбачуваним, що призводить до зниження інтересу частини аудиторії. Саме тому комерційний успіх таких проєктів часто залежить від здатності розробників утримувати увагу гравців через оновлення, новий контент або розширення ігрових механік. Окремо варто відзначити вплив інді-сегменту на розвиток жанру. Більшість Survivor-like ігор створюються невеликими командами або навіть окремими розробниками, що дозволяє швидко експериментувати з формулами геймплею та випускати нові продукти без значних фінансових витрат. Це сприяє постійному росту жанру та появі нових варіацій ігрового процесу.

Таким чином, жанр Survivor-like займає важливе місце в сучасній індустрії відеоігор, поєднуючи комерційну успішність, простоту входу та високу реіграбельність. Розроблена гра «Fantasy Poly Survivor» використовує ці переваги

жанру, адаптуючи їх у вигляді тривимірного Low Poly проєкту з класовою системою та фокусом на покращенні ігрового досвіду.

1.6 Виклики та тенденції у розробці ігор жанру Survivor-like

У процесі розробки ігор жанру Survivor-like виникає ряд специфічних викликів, які безпосередньо впливають на якість ігрового досвіду та загальну сприйнятність проєкту. Незважаючи на відносну простоту базової концепції, реалізація подібних ігор вимагає ретельного балансування ігрових систем та правильного формування темпу гри.

Одним із основних викликів є баланс складності. У Survivor-like іграх гравець постійно стикається зі зростаючою кількістю противників, тому важливо забезпечити плавне підвищення складності. Якщо складність зростає надто швидко, гравець втрачає можливість адаптації, що призводить до фрустрації. У протилежному випадку занадто повільне зростання складності знижує інтерес до гри через відсутність виклику.

Іншим важливим аспектом є утримання різноманітності ігрового процесу. Оскільки базовий цикл Survivor-like ігор є повторюваним (виживання, отримання досвіду, покращення, нові хвилі ворогів), існує ризик швидкого “звикання” гравця до ігрової формули. Саме тому значна увага приділяється створенню різних типів ворогів, варіацій поведінки та системи прогресії, яка дозволяє кожному проходженню відрізнятися від попереднього.

Важливим викликом також є оптимізація продуктивності, оскільки на екрані одночасно може перебувати велика кількість ворогів, ефектів та взаємодій. Це особливо актуально для тривимірних реалізацій, таких як проєкт «Fantasy Poly Survivor», де використання Low Poly стилю допомагає зменшити навантаження на систему без втрати візуальної читабельності.

Серед сучасних тенденцій жанру слід відзначити розвиток системи прогресії персонажа. Ігри все частіше зосереджуються не лише на виживанні, але й на формуванні унікального стилю проходження через комбінацію покращень. Це

дозволяє гравцю відчувати постійний розвиток і створює мотивацію до повторних проходжень.

Ще однією тенденцією є візуальне та стилістичне різноманіття. Розробники намагаються відійти від однотипних арен і впроваджують різні середовища — від фентезійних локацій до науково-фантастичних світів. У випадку «Fantasy Poly Survivor» використовується фентезійне Low Poly оточення, що підсилює візуальну унікальність проєкту.

Також варто відзначити тенденцію до спрощення входу в гру. Більшість Survivor-like проєктів орієнтуються на короткі ігрові сесії та інтуїтивне керування, що дозволяє швидко залучати нових гравців без складного навчання.

Окремо слід згадати проблему довгострокового утримання інтересу аудиторії. Через повторюваність базового ігрового циклу багато ігор жанру стикаються зі зниженням активності гравців після початкового періоду популярності. Це підкреслює важливість якісного ігрового досвіду, візуального різноманіття та грамотної побудови прогресії.

Ще однією тенденцією є активне використання 3D технологій та Low Poly стилістики, що дозволяє поєднати продуктивність і привабливий зовнішній вигляд. Це особливо важливо для ігор, де одночасно обробляється велика кількість ворогів та ефектів.

Таким чином, розробка Survivor-like ігор вимагає балансу між простотою геймплею, візуальною різноманітністю та стабільною продуктивністю. У проєкті «Fantasy Poly Survivor» ці аспекти враховані через використання Low Poly графіки, системи хвиль ворогів та прогресії персонажа, що спрямовано на покращення загального ігрового досвіду.

ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Огляд ігрового рушія Unity

Unity є одним із найпопулярніших сучасних ігрових рушіїв та інтегрованих середовищ розробки (IDE), який використовується для створення двовимірних і тривимірних комп’ютерних ігор. Рушій розроблений компанією Unity Technologies і активно застосовується як інді-розробниками, так і великими студіями завдяки своїй універсальності, доступності та великому набору вбудованих інструментів.

Unity надає розробнику повноцінне середовище для створення ігрових проєктів, включаючи роботу з графікою, фізикою, анімаціями, звуком, штучним інтелектом та інтерфейсом користувача. Завдяки цьому рушій дозволяє реалізовувати складні ігрові механіки без необхідності створення великої кількості низькорівневих систем із нуля.

Однією з ключових переваг Unity є його кросплатформеність. Розроблені на цьому рушії ігри можуть бути експортовані на різні платформи, включаючи персональні комп’ютери, мобільні пристрої та ігрові консолі. Це значно розширює потенційну аудиторію проєктів і дозволяє розробникам ефективніше масштабувати свої продукти.

Важливою особливістю Unity є використання мови програмування C#, яка забезпечує зручну та зрозумілу структуру коду, а також високу продуктивність у реалізації ігрової логіки. Це робить рушій доступним для навчання студентам і початківцям, одночасно залишаючи достатньо можливостей для професійної розробки складних ігрових систем.

У контексті розробки гри «Fantasy Poly Survivor» Unity використовується як основна платформа для реалізації ігрового процесу жанру Survivor-like. Рушій дозволяє ефективно працювати з великою кількістю об’єктів на сцені, що є критично важливим для даного жанру, де одночасно може бути активною велика кількість ворогів, ефектів та взаємодій.

Окрему роль відіграє система фізики Unity, яка забезпечує коректну взаємодію об’єктів у тривимірному просторі. Це особливо важливо для реалізації переміщення персонажа, зіткнень із ворогами та обробки атак у реальному часі.

Також Unity містить потужну систему компонентів, яка дозволяє будувати ігрові об’єкти на основі модульного підходу. Це спрощує процес розробки та підтримки коду, оскільки кожен елемент гри може бути реалізований як окремий компонент із власною логікою.

Завдяки вбудованому редактору сцен розробник отримує можливість візуально налаштовувати рівні, розміщувати об’єкти та тестувати ігровий процес у режимі реального часу. Це значно прискорює процес розробки та дозволяє швидко вносити зміни в ігровий баланс і структуру рівнів.

Ще однією важливою перевагою Unity є велика екосистема готових рішень, включаючи Asset Store, де можна знайти моделі, анімації, системи AI та інші інструменти. Це дозволяє суттєво скоротити час розробки та зосередитися на реалізації унікальних механік гри.

Таким чином, Unity є ефективним інструментом для створення сучасних ігрових проєктів, зокрема у жанрі Survivor-like. У рамках даної дипломної роботи цей рушій забезпечує технічну основу для реалізації гри «Fantasy Poly Survivor», включаючи графіку, логіку ігрового процесу та взаємодію об’єктів у тривимірному середовищі.

2.1.1 Unity та його особливості

Однією з ключових особливостей ігрового рушія Unity Technologies є його кросплатформеність. Це означає, що проєкти, створені в Unity, можуть бути експортовані на різні платформи, включаючи персональні комп’ютери, ігрові консолі, мобільні пристрої та вебсередовища. Така можливість дозволяє значно розширити потенційну аудиторію гри та забезпечує її доступність для користувачів із різними типами пристроїв і операційних систем.

Окрім цього, Unity підтримує основні операційні системи, зокрема Windows, macOS та Linux, що робить його універсальним інструментом для розробки

незалежно від середовища програміста. Це особливо важливо для інді-розробки, де команда або навіть один розробник часто працює на різних платформах.

Важливою перевагою Unity є широкий набір вбудованих інструментів і компонентів, які значно спрощують процес створення гри. Розробник отримує доступ до готових рішень для роботи з фізикою, анімацією, освітленням, аудіо та штучним інтелектом. Це дозволяє зосередитися не на низькорівневій реалізації базових систем, а на створенні ігрового процесу та покращенні користувацького досвіду.

У контексті розробки гри «Fantasy Poly Survivor» це має особливе значення, оскільки жанр Survivor-like передбачає велику кількість одночасних об’єктів на сцені, зокрема ворогів, ефектів та взаємодій. Unity забезпечує достатню продуктивність та оптимізаційні можливості для стабільної роботи таких систем.

Ще однією сильною стороною Unity є підтримка роботи з різними форматами мультимедійних файлів. Рушій дозволяє легко імпортувати 3D-моделі, текстури, анімації, аудіо- та відеофайли, що значно пришвидшує процес наповнення гри контентом. Це особливо важливо для проєктів у стилі Low Poly, де велика кількість візуальних елементів використовується для створення атмосфери гри.

Unity також активно використовує мову програмування C#, яка є основною мовою для створення ігрової логіки. Вона дозволяє реалізовувати поведінку персонажів, систему бою, генерацію хвиль ворогів, систему досвіду та інші ключові механіки гри. Завдяки зрозумілій структурі C# розробка стає більш організованою та зручною для масштабування проєкту.

Окрему роль відіграє візуальний редактор Unity, який дозволяє створювати ігрові сцени без необхідності постійного написання коду. Розробник може розміщувати об’єкти, налаштовувати камери, світло, фізичні взаємодії та тестувати результат у реальному часі. Це значно прискорює процес розробки та дозволяє швидко вносити зміни у геймплей і баланс гри.

У випадку «Fantasy Poly Survivor» редактор Unity є ключовим інструментом для побудови ігрових арен, розміщення ворогів, налаштування хвиль та тестування поведінки персонажа берсерка в умовах постійного натиску противників.

Також Unity має добре розвинену екосистему та велику спільноту розробників. Це означає, що під час створення гри можна використовувати готові рішення, отримувати підтримку, знаходити приклади реалізації механік та швидко вирішувати технічні проблеми. Це особливо корисно в інді-розробці, де швидкість прийняття рішень має велике значення.

Окремої уваги заслуговує підтримка мобільної розробки. Unity дозволяє оптимізувати ігри для Android та iOS, адаптуючи керування, продуктивність та графіку під мобільні пристрої. Це відкриває можливість розширення аудиторії та робить гру доступною для користувачів смартфонів і планшетів.

Таким чином, Unity є універсальним та потужним рушієм, який поєднує зручність використання, високу продуктивність та гнучкість у розробці. У межах даної дипломної роботи він використовується як основна технологічна база для реалізації гри «Fantasy Poly Survivor», забезпечуючи стабільність роботи, гнучкість у розробці та можливість реалізації складних ігрових сценаріїв жанру Survivor-like.

2.1.2 Можливості рушія Unity для розробки ігор жанру Survivor-like

Однією з ключових переваг ігрового рушія Unity Technologies є його компонентна архітектура, яка базується на принципі розділення ігрових об’єктів на окремі логічні модулі. У Unity кожен об’єкт сцени складається з набору компонентів, що відповідають за певні функції, такі як рух, фізика, колізії, візуалізація та логіка поведінки.

Такий підхід є особливо ефективним для ігор жанру Survivor-like, де на ігровій сцені одночасно присутня велика кількість об’єктів: персонаж гравця, вороги, ефекти, снаряди та елементи оточення. Завдяки компонентній системі можна легко масштабувати проєкт і керувати великою кількістю взаємодій між об’єктами без значного ускладнення коду.

У контексті гри «Fantasy Poly Survivor» це дозволяє окремо реалізувати логіку гравця (клас берсеркера), систему ворогів, механіку хвиль та систему отримання досвіду, зберігаючи при цьому модульність ігрової архітектури.

Ще однією важливою перевагою Unity є наявність великої кількості вбудованих інструментів для роботи з фізикою, анімацією, звуком та графікою. Це дозволяє створювати більш живий і динамічний ігровий процес, що є критично важливим для Survivor-like ігор, де постійний рух і взаємодія об’єктів формують основний геймплейний цикл.

Також Unity надає можливості для базової реалізації штучного інтелекту, що дозволяє створювати поведінку ворогів, їх пересування до гравця та реакцію на ігрові події. Це особливо важливо для жанру Survivor-like, де велика кількість противників повинна діяти одночасно та створювати постійний тиск на гравця.

Окремою перевагою рушія є підтримка кросплатформеності. Ігри, створені в Unity, можуть бути експортовані на різні платформи, включаючи ПК, мобільні пристрої та консолі. Це забезпечує широке охоплення аудиторії та дозволяє адаптувати гру під різні пристрої без необхідності повного переписування проєкту.

Для інді-розробки це має особливе значення, оскільки дозволяє швидко тестувати та розповсюджувати ігрові прототипи. У випадку «Fantasy Poly Survivor» це дає можливість зосередитися на ігровому досвіді, балансі та візуальній складовій, не витрачаючи зайві ресурси на платформозалежні реалізації.

Ще однією сильною стороною Unity є розвинена екосистема розширень та готових рішень. Завдяки цьому розробник може використовувати вже створені інструменти для роботи з UI, анімаціями, генерацією контенту та оптимізацією продуктивності. Це значно пришвидшує процес розробки і дозволяє зосередитися на ключових механіках гри.

Важливу роль відіграє також активна спільнота розробників Unity. Вона забезпечує доступ до великої кількості навчальних матеріалів, прикладів реалізації механік та рішень типових проблем. Це суттєво спрощує процес розробки, особливо на етапі створення складних систем взаємодії об’єктів у грі.

Таким чином, Unity надає всі необхідні інструменти для ефективної розробки ігор жанру Survivor-like. У межах проєкту «Fantasy Poly Survivor» цей рушій використовується як основна технологічна база для реалізації ігрової логіки, системи хвиль, поведінки ворогів та загальної архітектури гри.

2.1.3 Переваги та обмеження використання Unity для розробки ігор жанру Survivor-like

Використання ігрового рушія Unity Technologies для створення ігор жанру Survivor-like має низку суттєвих переваг, які безпосередньо впливають на якість фінального продукту та ефективність процесу розробки.

Однією з головних переваг є потужні можливості Unity у сфері графічного та візуального оформлення. Рушій дозволяє створювати якісні тривимірні сцени з використанням освітлення, тіней, постобробки та анімацій. Це особливо важливо для Survivor-like ігор, де візуальна читабельність великої кількості об’єктів на екрані є критичною для комфортного ігрового процесу.

У межах проєкту «Fantasy Poly Survivor» це дозволяє реалізувати Low Poly стилістику, яка поєднує простоту моделей із достатнім рівнем візуальної привабливості та високою продуктивністю. Завдяки цьому гра залишається зрозумілою навіть у моменти великої кількості ворогів і ефектів на екрані.

Ще однією важливою перевагою є підтримка аудіосистеми. Unity дозволяє інтегрувати фонову музику, звукові ефекти ударів, дій персонажа та подій у грі. Це підсилює атмосферу напруженого виживання та робить ігровий процес більш емоційно насиченим.

Також важливою перевагою є гнучкість рушія у роботі з великою кількістю об’єктів. Survivor-like ігри передбачають одночасну активність великої кількості ворогів, снарядів та ефектів. Unity дозволяє ефективно керувати такими системами, однак потребує грамотної оптимізації для підтримки стабільної продуктивності.

Водночас використання Unity має і певні обмеження. Одним із основних є необхідність ретельної оптимізації продуктивності. При великій кількості об’єктів на сцені можливе зниження частоти кадрів, особливо на слабших пристроях. Це вимагає від розробника продуманого підходу до створення ігрових систем та контролю навантаження.

Окремим викликом є обмеженість вбудованих рішень для складних систем штучного інтелекту. Хоча Unity надає базові інструменти для реалізації поведінки

об’єктів, складні логічні системи ворогів та їх взаємодії з гравцем потребують додаткової розробки або використання сторонніх рішень. У жанрі Survivor-like це особливо важливо, оскільки велика кількість противників повинна діяти одночасно та створювати відчуття постійного тиску.

Також слід враховувати обмеження, пов’язані з масштабуванням проєкту. У випадку значного збільшення кількості об’єктів, ефектів та систем взаємодії необхідно ретельно контролювати використання ресурсів пам’яті та процесора. Без оптимізації це може призвести до зниження стабільності гри.

Незважаючи на ці обмеження, Unity залишається одним із найбільш ефективних рушіїв для створення ігор жанру Survivor-like. Його переваги у вигляді гнучкості, візуальних можливостей та підтримки кросплатформеності значно переважають технічні складності, особливо для інді-проєктів.

У рамках розробки гри «Fantasy Poly Survivor» ці переваги дозволяють зосередитися на головному — створенні якісного ігрового досвіду, балансу складності та атмосферного виживання, що є ключовими елементами жанру.

2.2 Огляд мови програмування C#

Мова програмування C# є однією з основних мов, що використовується для розробки ігрових додатків у середовищі ігрового рушія Unity Technologies. Вона широко застосовується в індустрії завдяки своїй простоті, структурованості та потужним можливостям об’єктно-орієнтованого програмування.

Синтаксис C# має певну схожість із мовами C та C++, що полегшує його освоєння для розробників, які вже мають базове розуміння цих мов. При цьому C# є більш безпечною та високорівневою мовою, що зменшує кількість помилок при розробці складних програмних систем.

Однією з ключових переваг C# є підтримка повноцінного об’єктно-орієнтованого програмування. Мова дозволяє використовувати класи, об’єкти, наслідування, інкапсуляцію та поліморфізм. Це особливо важливо при розробці

ігор, оскільки дозволяє структурувати код у вигляді логічних модулів, наприклад: гравець, вороги, система хвиль, система досвіду та бойова система.

У контексті гри «Fantasy Poly Survivor» використання C# дозволяє розділити логіку гри на окремі компоненти, такі як управління персонажем (клас берсеркера), поведінка ворогів, система нанесення шкоди та система прогресії. Такий підхід забезпечує зрозумілу архітектуру проєкту та спрощує його подальшу підтримку і розширення.

Важливою перевагою C# є його глибока інтеграція з Unity через Unity Scripting API. Це дає можливість напряму взаємодіяти з усіма системами рушія, включаючи роботу зі сценами, ігровими об’єктами, фізикою, анімацією, UI та звуковими ефектами. Завдяки цьому розробник може реалізовувати повний цикл ігрової логіки без необхідності використання зовнішніх інструментів.

Окремо слід відзначити підтримку подієвої моделі програмування, яка активно використовується в C#. Вона дозволяє ефективно обробляти ігрові події, такі як отримання шкоди, смерть ворога, підвищення рівня або активація здібностей. Це робить код більш гнучким і зменшує залежність між окремими системами гри.

Також C# підтримує багатопоточність та асинхронні операції, що може бути корисним для оптимізації деяких процесів у грі, наприклад завантаження ресурсів або обробки складних обчислень. Хоча в більшості випадків у Unity використовується основний ігровий потік, ці можливості дозволяють підвищити загальну продуктивність проєкту.

Таким чином, C# є ефективною та зручною мовою програмування для розробки ігор жанру Survivor-like. У рамках дипломної роботи вона використовується як основний інструмент реалізації ігрової логіки гри «Fantasy Poly Survivor», забезпечуючи гнучкість, масштабованість та структурованість коду.

2.2.1 Основні характеристики мови програмування C#

Мова програмування C# є однією з ключових технологій, що використовуються у сучасній ігровій розробці, особливо в поєднанні з рушієм Unity. Вона належить до сучасних мов високого рівня і поєднує в собі простоту синтаксису та потужні можливості для створення складних програмних систем. Завдяки цьому C# активно використовується для реалізації ігрової логіки, взаємодії об’єктів, обробки подій та побудови архітектури ігрових додатків.

Однією з базових характеристик C# є строга типізація. Це означає, що кожна змінна, параметр або об’єкт мають чітко визначений тип даних. Такий підхід дозволяє зменшити кількість помилок на етапі компіляції, покращує читабельність коду та робить програму більш передбачуваною у виконанні. Завдяки цьому розробник отримує більше контролю над структурою даних і взаємодією між компонентами програми.

Важливою особливістю мови є повна підтримка об’єктно-орієнтованого програмування. C# дозволяє використовувати класи та об’єкти як основні будівельні блоки програми, реалізовувати спадкування, поліморфізм та інкапсуляцію. Такий підхід особливо ефективний у розробці ігор, оскільки дає змогу логічно структурувати великі системи: окремі класи можуть відповідати за персонажів, ворогів, предмети, бойові системи або інтерфейс користувача. Це значно спрощує подальше масштабування проєкту та його супровід.

Ще однією сильною стороною C# є автоматичне керування пам’яттю за допомогою механізму garbage collection. Розробнику не потрібно вручну звільняти пам’ять, як у деяких низькорівневих мовах, оскільки система сама визначає та очищає невикористані ресурси. Це зменшує ризик витоків пам’яті та підвищує стабільність роботи застосунку, що особливо важливо для ігрових проєктів із великою кількістю об’єктів.

Окремо слід відзначити інтеграцію C# із платформою .NET, яка надає великий набір готових бібліотек і рішень. Завдяки цьому розробники можуть використовувати вже реалізовані інструменти для роботи з файлами, математичними обчисленнями, колекціями даних, мережею та іншими системними

задачами. Це значно прискорює процес розробки і дозволяє зосередитися на ігровій логіці, а не на низькорівневих деталях реалізації.

Також C# підтримує асинхронне програмування, що реалізується через механізм async/await. Ця можливість є особливо корисною в іграх, оскільки дозволяє виконувати тривалі операції (наприклад, завантаження сцен або роботу з мережею) без блокування основного ігрового потоку. У результаті гра працює більш плавно, а взаємодія з користувачем залишається стабільною та комфортною.

Таким чином, C# є універсальною та ефективною мовою програмування, яка поєднує простоту використання з потужними функціональними можливостями. У контексті розробки ігор на Unity вона забезпечує необхідний інструментарій для створення структурованих, продуктивних та масштабованих ігрових проєктів.

2.2.2 Використання мови C# у розробці ігор на рушії Unity

Мова програмування C# є основним інструментом реалізації ігрової логіки в середовищі Unity і фактично виступає стандартом для створення ігрових проєктів на цьому рушії. Вона використовується для опису поведінки об’єктів, керування ігровими подіями, реалізації систем взаємодії та побудови загальної архітектури гри. Завдяки глибокій інтеграції Unity з C# розробники отримують змогу ефективно поєднувати візуальну розробку сцен із програмною реалізацією функціоналу.

У процесі створення гри на Unity C# дозволяє реалізовувати широкий спектр задач, починаючи від базової логіки ігрового процесу і завершуючи складними системами поведінки об’єктів. Зокрема, за допомогою цієї мови програмування можна обробляти введення користувача, керувати рухом персонажів, реалізовувати бойові механіки, систему взаємодії з оточенням, а також контролювати стан ігрового світу. Така гнучкість робить C# універсальним рішенням для різних жанрів ігор, включаючи покрокові стратегії та survival-проєкти.

Важливою перевагою використання C# у Unity є те, що мова органічно інтегрована в архітектуру рушія. Усі ігрові об’єкти можуть бути пов’язані зі скриптами C#, які керують їхньою поведінкою через компоненти. Це відповідає компонентно-орієнтованому підходу Unity, де кожен об’єкт сцени складається з

набору незалежних модулів. Така структура дозволяє легко масштабувати проєкт, повторно використовувати код і розділяти відповідальність між різними частинами системи.

Крім того, середовище Unity надає зручні інструменти для роботи з C#, зокрема інтеграцію з редактором коду, можливість миттєвого тестування скриптів та налагодження безпосередньо в ігровій сцені. Це значно скорочує цикл розробки, оскільки розробник може одразу бачити результат змін у коді та швидко коригувати поведінку ігрових об’єктів.

Окремо слід відзначити кросплатформені можливості C# у поєднанні з Unity. Один і той самий програмний код може бути використаний для збірки гри на різних платформах, включаючи персональні комп’ютери, мобільні пристрої та ігрові консолі. Це дозволяє значно зменшити витрати часу на адаптацію проєкту під різні пристрої та забезпечує широкий охоплення аудиторії без необхідності переписування логіки гри.

Таким чином, використання C# у Unity є ключовим фактором ефективної розробки ігор, оскільки поєднує в собі гнучкість програмування, зручність інтеграції та можливість створення масштабованих ігрових систем із підтримкою різних платформ.

2.2.3 Переваги та обмеження використання мови C# у розробці покрокових стратегій

Використання мови програмування C# у розробці покрокових стратегій на базі рушія Unity має низку суттєвих переваг, які роблять її одним із найоптимальніших інструментів для створення ігрових застосунків. Перш за все, C# характеризується відносною простотою синтаксису та зрозумілою структурою коду, що значно знижує поріг входження для нових розробників. Це дозволяє швидше опановувати мову та переходити до практичної реалізації ігрових механік без необхідності глибокого занурення у складні низькорівневі концепції.

Додатковою перевагою є наявність великої кількості стандартних бібліотек та вбудованих інструментів, які значно спрощують процес програмування.

Розробник може використовувати готові рішення для роботи з колекціями даних, математичними операціями, обробкою подій та іншими базовими задачами. Це дозволяє зосередитися безпосередньо на реалізації ігрової логіки та механік, а не на написанні допоміжного функціоналу з нуля.

Важливим фактором є також висока інтеграція C# із середовищем Unity та зовнішніми інструментами розробки. Зокрема, середовище Microsoft Visual Studio забезпечує розширені можливості налагодження, автодоповнення коду та аналізу помилок, що значно підвищує продуктивність розробника. У поєднанні з редактором Unity це створює зручне середовище для швидкої розробки, тестування та вдосконалення ігрових систем у режимі реального часу.

Однак, попри значні переваги, використання C# має і певні обмеження. Одним із них є залежність від екосистеми Unity, оскільки основна інтеграція мови відбувається саме в межах цього рушія. Це може певною мірою обмежувати гнучкість у виборі альтернативних технологій або нестандартних рішень, які не підтримуються Unity напряму.

Ще одним аспектом є те, що C# є мовою високого рівня, що іноді може впливати на продуктивність у порівнянні з більш низькорівневими мовами програмування. У проєктах із великою кількістю одночасних обчислень або складною симуляцією це може вимагати додаткової оптимізації коду для забезпечення стабільної роботи гри. Крім того, при неправильному використанні ресурсів можливе навантаження на систему, що особливо важливо для мобільних платформ або слабших пристроїв.

Таким чином, C# є збалансованим інструментом для розробки покрокових стратегій, який поєднує зручність, функціональність та широкі можливості інтеграції, однак потребує грамотного підходу до оптимізації та архітектури проєкту для досягнення максимальної ефективності.

2.3 Огляд середовища тривимірного моделювання Blender

Blender є безкоштовним і відкритим програмним середовищем для створення тривимірної графіки, яке широко використовується у розробці ігор, анімації та візуалізацій. Завдяки своїй універсальності Blender дозволяє реалізовувати повний цикл створення 3D-контенту: від базового моделювання об’єктів до текстурування, ріґгінгу, анімації та підготовки моделей до використання в ігрових рушіях, зокрема Unity.

У межах даної дипломної роботи Blender використовується як основний інструмент створення low-poly графіки, що відповідає стилістиці гри. Такий підхід дозволяє досягти візуальної простоти, високої продуктивності та швидкої оптимізації ігрових сцен без втрати читаємості об’єктів у геймплеї.

2.3.1 Основні можливості Blender

Blender підтримує широкий набір інструментів для роботи з 3D-графікою. Основними можливостями є полігональне моделювання, робота з примітивами, скульптинг, UV-розгортка, текстурування та запікання матеріалів.

Однією з ключових переваг Blender є зручність створення low-poly моделей. Завдяки простій роботі з вершинами, ребрами та гранями розробник може швидко формувати базові форми об’єктів без зайвої деталізації. Це особливо важливо для ігор, орієнтованих на продуктивність та велику кількість об’єктів на сцені.

Blender також дозволяє легко працювати з матеріалами та простими текстурами, що дає можливість створювати стилізовану графіку без потреби у складних шейдерах. У даній роботі це стало важливою перевагою, оскільки основний акцент зроблено на читабельності ігрового світу та швидкості рендерингу.

2.3.2 Використання Blender у розробці гри

У процесі розробки гри Blender використовується для створення всіх основних ігрових об’єктів: персонажів, ворогів, оточення та декоративних

елементів. Усі моделі створюються з урахуванням обмеженої кількості полігонів, що відповідає стилю low-poly та забезпечує стабільну продуктивність гри.

Особлива увага приділяється авторському візуальному стилю. Використання спрощених форм і мінімалістичних текстур дозволяє створити впізнавану стилістику гри, яка не перевантажує гравця деталями, але зберігає атмосферність і читабельність ігрового процесу.

Експорт моделей здійснюється у форматі, сумісному з Unity (наприклад, FBX), що дозволяє без втрати якості інтегрувати об’єкти в ігрові сцени. Це забезпечує швидкий та зручний пайплайн між моделюванням та реалізацією геймплею.

2.3.3 Переваги використання Blender у розробці ігор

Основною перевагою Blender є його повна безкоштовність та відкритий вихідний код, що робить його доступним для інді-розробників і студентських проєктів. Це дозволяє створювати повноцінний ігровий контент без фінансових витрат на ліцензії.

Другою важливою перевагою є швидкість створення low-poly контенту. У контексті даної гри це критично важливо, оскільки дозволяє швидко наповнювати ігровий світ об’єктами без тривалого виробничого циклу. Простота інструментів моделювання дає можливість швидко змінювати геометрію моделей та адаптувати їх під потреби геймплею.

Також важливою перевагою є гнучкість Blender як інструменту для авторського дизайну. У межах цієї роботи він дозволив реалізувати унікальне візуальне бачення гри, де акцент зроблено не на фотореалізмі, а на стилізованій low-poly естетиці.

2.3.4 Обмеження використання Blender

Попри широкі можливості, Blender має певні обмеження. Перш за все, він потребує часу на освоєння, оскільки містить велику кількість інструментів і

режимів роботи. Це може ускладнити початковий етап розробки для нових користувачів.

Також Blender не є ігровим рушієм, тому не підтримує інтерактивну логіку або фізику в реальному часі — його роль обмежується лише створенням контенту. Усі ігрові механіки реалізуються вже в середовищі Unity.

Ще одним обмеженням є необхідність правильної оптимізації моделей перед експортом. Невірно налаштована геометрія або текстури можуть негативно вплинути на продуктивність гри, особливо на слабших пристроях.

2.3.5 Роль Blender у формуванні візуального стилю гри

У даному проєкті Blender виконує ключову роль у формуванні загального художнього стилю гри. Вибір low-poly підходу був свідомим дизайнерським рішенням, яке дозволяє досягти балансу між візуальною простотою та атмосферністю.

Такий стиль підкреслює ігрову читабельність, що особливо важливо для жанру survival-like, де гравець повинен швидко орієнтуватися в оточенні. Простота форм також сприяє кращому сприйняттю динаміки бою та взаємодії з ворогами.

Blender у цьому випадку виступає не лише інструментом моделювання, а й засобом реалізації авторської ідеї — створення мінімалістичного, але впізнаваного світу, який підтримує загальну концепцію гри.

2.4 Огляд середовища розробки Microsoft Visual Studio

Середовище розробки Microsoft Visual Studio є одним із найпотужніших та найпоширеніших інтегрованих середовищ розробки (IDE), яке використовується для створення програмного забезпечення різного рівня складності, включаючи комп’ютерні ігри, корпоративні системи, веб-додатки та мобільні застосунки. У контексті розробки ігор на рушії Unity Visual Studio виступає основним інструментом для написання, налагодження та оптимізації коду мовою програмування C#.

Microsoft Visual Studio було вперше представлено компанією Microsoft у 1997 році як універсальне середовище для розробки програм під операційну систему Windows. З того часу IDE постійно розвивалося, розширюючи підтримку мов програмування, платформ та технологій. Сучасні версії Visual Studio підтримують такі мови, як C#, C++, F#, Python, JavaScript, а також технології .NET, ASP.NET, Xamarin та інші. Завдяки цьому Visual Studio стало універсальним інструментом для розробників у різних галузях програмування.

Основною метою створення Visual Studio було забезпечення розробників комплексним середовищем, яке об'єднує редактор коду, компілятор, засоби налагодження, систему керування проєктами та інтеграцію з базами даних і зовнішніми сервісами. Такий підхід дозволяє значно прискорити процес розробки програмного забезпечення та зменшити кількість помилок на етапі реалізації.

У сучасній розробці ігор Visual Studio найчастіше використовується разом із рушієм Unity, оскільки між ними реалізована глибока інтеграція. Unity автоматично підключає Visual Studio як основний редактор скриптів C#, що дозволяє розробнику відкривати будь-який скрипт безпосередньо з редактора Unity, редагувати його та миттєво повертатися до ігрового середовища. Такий підхід значно спрощує робочий процес і зменшує час на перемикання між різними інструментами.

2.4.1 Основні можливості Microsoft Visual Studio

Microsoft Visual Studio надає широкий набір інструментів, які роблять процес розробки більш ефективним і контрольованим. Однією з ключових можливостей є інтелектуальне автодоповнення коду IntelliSense. Цей інструмент дозволяє автоматично підказувати розробнику методи, класи, властивості та синтаксис мови програмування. У контексті Unity це особливо важливо, оскільки рушій має велику кількість API, і швидкий доступ до них значно прискорює розробку ігрових механік.

Ще однією важливою функцією Visual Studio є потужна система налагодження (debugging). Вона дозволяє встановлювати точки зупинки у коді,

відслідковувати значення змінних у реальному часі, аналізувати стек викликів та знаходити логічні помилки у програмі. Для ігрової розробки це критично важливо, оскільки ігрові системи часто складаються з великої кількості взаємопов'язаних компонентів.

Visual Studio також підтримує систему рефакторингу коду, яка дозволяє покращувати структуру програми без зміни її функціональності. Наприклад, розробник може автоматично перейменовувати класи, методи або змінні у всьому проєкті, що значно зменшує ризик помилок при масштабних змінах.

Окремо слід відзначити підтримку системи контролю версій Git, яка інтегрована безпосередньо у середовище Visual Studio. Це дозволяє відслідковувати зміни у коді, працювати в команді та повертатися до попередніх версій проєкту у разі необхідності.

2.4.2 Інтеграція Microsoft Visual Studio з Unity

Однією з ключових переваг Visual Studio у контексті розробки ігор є його глибока інтеграція з ігровим рушієм Unity. При створенні нового Unity-проєкту середовище автоматично налаштовує Visual Studio як основний редактор скриптів. Це означає, що всі C# файли відкриваються безпосередньо у Visual Studio, а зміни одразу синхронізуються з Unity.

Під час роботи розробник може створювати скрипти, які керують ігровими об'єктами, фізикою, штучним інтелектом, UI та іншими системами гри. Visual Studio надає повну підтримку Unity API, включаючи підказки для класів MonoBehaviour, методів життєвого циклу (Start, Update, Awake) та інших компонентів рушія.

Також важливою особливістю є можливість налагодження Unity-проєкту безпосередньо через Visual Studio. Розробник може запускати гру в режимі відлагодження, встановлювати точки зупинки у коді та аналізувати поведінку гри в реальному часі. Це значно спрощує процес пошуку помилок у складних ігрових системах.

Інтеграція між Unity та Visual Studio також включає автоматичну генерацію проєктних файлів, що дозволяє IDE коректно розпізнавати всі залежності, скрипти та бібліотеки Unity. Завдяки цьому забезпечується стабільна робота середовища та відсутність конфліктів між компонентами проєкту.

2.4.3 Переваги використання Microsoft Visual Studio

Використання Microsoft Visual Studio у розробці ігор має низку суттєвих переваг. Перш за все, це висока продуктивність середовища та оптимізація для роботи з великими проєктами. IDE здатне обробляти складні кодові бази без значних затримок, що є важливим при створенні ігор середнього та великого масштабу.

Другою важливою перевагою є зручність роботи з мовою C#. Visual Studio забезпечує повну підтримку всіх сучасних можливостей мови, включаючи асинхронне програмування, LINQ, делегати та події. Це дозволяє створювати чисту та структуровану архітектуру ігрового коду.

Також варто відзначити велику кількість розширень (extensions), які дозволяють адаптувати середовище під конкретні потреби розробника. Наприклад, можна додати інструменти для аналізу продуктивності, автоматичного форматування коду або розширеної роботи з Unity.

Ще однією перевагою є активна підтримка з боку Microsoft та регулярні оновлення, які покращують стабільність, продуктивність та функціональність середовища. Це гарантує довгострокову актуальність інструменту.

2.4.4 Обмеження та недоліки

Попри численні переваги, Visual Studio має і певні обмеження. Основним недоліком є висока вимогливість до системних ресурсів. Для комфортної роботи середовище потребує достатньо великого обсягу оперативної пам’яті та продуктивного процесора, особливо при роботі з великими Unity-проєктами.

Також варто зазначити, що початкове налаштування середовища може бути складним для новачків. Visual Studio має велику кількість функцій і налаштувань, що може ускладнювати швидке освоєння.

У деяких випадках також можуть виникати проблеми з синхронізацією Unity та Visual Studio, особливо при некоректних налаштуваннях проєкту або відсутності необхідних пакетів .NET SDK.

2.4.5 Роль Visual Studio у розробці гри “Fantasy Poly Survivors”

У межах розробки гри “Fantasy Poly Survivors” середовище Microsoft Visual Studio використовується як основний інструмент для написання ігрової логіки на мові C#. Саме у Visual Studio реалізуються ключові системи гри, включаючи керування гравцем, систему хвиль ворогів, бойову систему, систему покращень та інтерфейс користувача.

Завдяки інтеграції з Unity розробка відбувається у єдиному робочому процесі: зміни у коді миттєво відображаються у грі, що дозволяє швидко тестувати механіки та балансувати геймплей. Використання Visual Studio також забезпечує зручне відлагодження складних систем, таких як взаємодія між класами PlayerController, EnemyController та WaveManager.

Таким чином, Microsoft Visual Studio виступає ключовим елементом у процесі розробки, забезпечуючи ефективність, стабільність та зручність створення програмного продукту.

2.5 Огляд середовища розробки Visual Studio Code

Visual Studio Code (VS Code) є сучасним легким кросплатформеним редактором вихідного коду, розробленим компанією Microsoft. Він широко використовується для редагування, аналізу та швидкої модифікації коду в різних мовах програмування, включаючи C#, JavaScript, Python, HTML, XML та інші. На відміну від повноцінного інтегрованого середовища розробки, Visual Studio Code

орієнтований на швидкість роботи, простоту інтерфейсу та високу гнучкість налаштування.

Visual Studio Code був вперше представлений у 2015 році як легка альтернатива повноцінним IDE. Його основною метою було створення інструменту, який поєднує можливості редактора коду та розширюваність середовища розробки, але при цьому не перевантажує систему. Завдяки цьому VS Code швидко набув популярності серед розробників, особливо у сферах веб-розробки, ігрової індустрії та роботи з конфігураційними файлами.

Середовище побудоване на базі Electron, що дозволяє йому працювати на операційних системах Windows, macOS та Linux. Однією з ключових переваг Visual Studio Code є його модульна архітектура, яка дозволяє встановлювати розширення (extensions) для підтримки різних мов програмування, фреймворків та форматів файлів. Це робить його універсальним інструментом для широкого кола задач.

2.5.1 Основні можливості Visual Studio Code

Visual Studio Code надає широкий набір можливостей для роботи з кодом і текстовими файлами. Однією з ключових функцій є підсвічування синтаксису, що дозволяє зручно працювати з різними мовами програмування та швидко орієнтуватися у структурі файлів.

Також важливою функцією є підтримка IntelliSense, яка забезпечує автодоповнення коду, підказки параметрів функцій та швидкий доступ до документації. Це значно прискорює процес редагування файлів і зменшує кількість синтаксичних помилок.

Visual Studio Code підтримує вбудовану систему терміналу, що дозволяє виконувати команди безпосередньо в середовищі, а також інтеграцію з системами контролю версій Git. Це забезпечує зручну роботу з проєктами та дозволяє відстежувати зміни у файлах.

Окремою перевагою є велика бібліотека розширень, які дозволяють адаптувати середовище під конкретні задачі, включаючи роботу з XML-файлами,

JSON, YAML та іншими форматами даних, що є важливим у процесі налаштування ігрових проєктів.

2.5.2 Використання Visual Studio Code у розробці ігор на Unity

У процесі розробки ігор на рушії Unity Visual Studio Code може використовуватися як альтернативний або допоміжний редактор коду. Він особливо ефективний для швидкого редагування окремих скриптів, конфігураційних файлів та службових даних проєкту.

Unity підтримує інтеграцію з Visual Studio Code через встановлення відповідних пакетів та розширень, що дозволяє відкривати C# скрипти безпосередньо в редакторі. Хоча основним середовищем для повноцінної розробки і відлагодження зазвичай є Visual Studio, VS Code використовується як легкий інструмент для швидких змін і редагування структури проєкту.

У межах даного проєкту Visual Studio Code застосовувався переважно для редагування XML-файлів, які містять конфігураційні дані та параметри системи гри. Це дозволило значно спростити процес налаштування ігрових об’єктів, зменшити час внесення змін та підвищити гнучкість роботи з даними.

2.5.3 Переваги використання Visual Studio Code

Однією з головних переваг Visual Studio Code є його легкість та висока швидкість роботи. Середовище запускається значно швидше за повноцінні IDE, що робить його зручним для оперативного редагування файлів.

Другою важливою перевагою є гнучкість налаштування. Завдяки великій кількості розширень користувач може адаптувати середовище під будь-які потреби, включаючи роботу з ігровими проєктами, веб-технологіями або структурованими даними.

Також важливою перевагою є кросплатформеність, що дозволяє використовувати Visual Studio Code на різних операційних системах без втрати функціональності.

2.5.4 Обмеження використання Visual Studio Code

Попри свої переваги, Visual Studio Code має і певні обмеження. Основним недоліком є відсутність повноцінного вбудованого середовища налагодження для складних Unity-проєктів без додаткових налаштувань.

Також він не замінює повноцінну IDE у випадках великомасштабної розробки, оскільки не містить деяких розширених інструментів для аналізу архітектури проєкту, профілювання та глибокого дебагу.

2.5.6 Роль Visual Studio Code у проєкті «Fantasy Poly Survivors»

У межах розробки гри “Fantasy Poly Survivors” Visual Studio Code використовувався як допоміжний інструмент для швидкого редагування XML-конфігурацій та службових файлів проєкту. Це дозволило значно спростити процес налаштування ігрових параметрів та забезпечило більш гнучке керування структурою даних.

Завдяки використанню Visual Studio Code розробка стала більш зручною у частині внесення швидких змін без необхідності запуску повноцінного середовища Visual Studio. Це позитивно вплинуло на швидкість ітераційного процесу розробки та тестування гри.

ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до програмного продукту

Одним із найважливіших етапів розробки будь-якого програмного продукту є формування вимог до його функціональності та характеристик роботи. Вимоги визначають перелік можливостей, які повинна забезпечувати система, а також критерії якості, стабільності та продуктивності програмного забезпечення.

У межах даної дипломної роботи розробляється комп'ютерна гра «Fantasy Poly Survivor», виконана у жанрі survival-like з використанням тривимірної low-poly графіки. Основною метою гри є виживання персонажа протягом визначеного часу шляхом знищення хвиль супротивників, отримання досвіду та покращення характеристик героя.

Під час визначення вимог до програмного продукту враховувалися особливості жанру, технічні можливості обраних інструментів розробки, а також необхідність забезпечення комфортного ігрового процесу для користувача. Сформовані вимоги поділяються на функціональні та нефункціональні.

Функціональні вимоги описують можливості, які повинна надавати гра користувачу, а нефункціональні визначають характеристики якості, продуктивності та стабільності роботи програмного продукту.

3.1.1 Функціональні вимоги

До функціональних вимог програмного продукту належать такі можливості:

1. Головне меню

1.1. Користувач може розпочати нову гру.

1.2. Користувач може завершити роботу програми.

1.3. Користувач може перейти до ігрової сцени через головне меню.

2. Ігровий процес

2.1. Гравець керує персонажем класу Берсерк.

2.2. Персонаж може переміщуватися ігровою локацією.

2.3. Персонаж автоматично атакує супротивників за допомогою доступної зброї.

2.4. На ігровій карті відбувається генерація хвиль ворогів.

2.5. Вороги переслідують персонажа та завдають йому шкоди при контакті.

2.6. За знищення ворогів гравець отримує досвід.

2.7. Після накопичення достатньої кількості досвіду персонаж підвищує рівень.

2.8. Після підвищення рівня користувач отримує можливість обрати одне із запропонованих покращень.

2.9. Покращення змінюють характеристики персонажа або його зброї.

2.10. Гравець може підбирати кристали досвіду, що залишаються після загибелі ворогів.

2.11. Під час гри відображається таймер виживання.

2.12. Гра завершується після втрати всіх одиниць здоров’я персонажа.

3. Інтерфейс користувача

3.1. Інтерфейс відображає поточний рівень персонажа.

3.2. Інтерфейс відображає шкалу здоров’я.

3.3. Інтерфейс відображає кількість отриманого досвіду.

3.4. Інтерфейс відображає час виживання.

3.5. Інтерфейс відображає меню вибору покращень після підвищення рівня.

4. Графічне та звукове оформлення

4.1. Гра використовує тривимірну low-poly графіку.

4.2. У грі реалізовані анімації персонажа та супротивників.

4.3. Гра містить звукові ефекти атак, отримання досвіду та взаємодії з інтерфейсом.

4.4. Гра містить фоновий музичний супровід.

3.1.2 Нефункціональні вимоги

До нефункціональних вимог програмного продукту належать такі характеристики:

1. Продуктивність

1.1. Програма повинна забезпечувати плавний ігровий процес без помітних затримок.

1.2. Частота кадрів повинна залишатися в межах 50–60 кадрів за секунду на рекомендованій конфігурації комп’ютера.

1.3. Система повинна підтримувати одночасне відображення великої кількості ворогів без критичного зниження продуктивності.

2. Надійність

2.1. Гра повинна стабільно працювати протягом тривалих ігрових сесій.

2.2. Під час роботи не повинно виникати критичних помилок або аварійного завершення програми.

2.3. Система повинна коректно обробляти зіткнення між персонажем, ворогами та зброєю.

3. Сумісність

3.1. Програмний продукт повинен працювати на операційній системі Windows.

3.2. Гра повинна підтримувати стандартні роздільні здатності моніторів.

3.3. Інтерфейс користувача повинен коректно масштабуватися залежно від роздільної здатності екрана.

4. Зручність використання

4.1. Керування повинно бути інтуїтивно зрозумілим для користувача.

4.2. Основні ігрові показники повинні бути постійно доступними на екрані.

4.3. Навігація між меню повинна здійснюватися без додаткових налаштувань.

5. Системні вимоги

Мінімальні системні вимоги:

1. Операційна система: Windows 10;
2. Процесор: Intel Core i3 або аналогічний;
3. Оперативна пам’ять: 4 GB RAM;
4. Відеокарта: Intel HD Graphics 4000;
5. Вільне місце на диску: 1 GB.

Рекомендовані системні вимоги:

1. Операційна система: Windows 10/11;
2. Процесор: Intel Core i5 або аналогічний;
3. Оперативна пам’ять: 8 GB RAM;
4. Відеокарта: NVIDIA GeForce GTX 970 або новіша;
5. Вільне місце на диску: 2 GB.

3.2 Розробка діаграми класів

Діаграма класів є одним із найважливіших інструментів об'єктно-орієнтованого проєктування програмного забезпечення. Вона дозволяє наочно відобразити структуру програмного продукту, взаємозв'язки між окремими компонентами системи, а також їх основні властивості та функції. Використання діаграми класів на етапі проєктування дає можливість спростити подальшу реалізацію програмного продукту та забезпечити зрозумілу архітектуру системи.

Для побудови діаграми класів було використано мову моделювання UML (Unified Modeling Language), яка є міжнародним стандартом для візуального опису програмних систем. UML дозволяє відобразити структуру програмного продукту у вигляді взаємопов’язаних класів, що значно полегшує аналіз архітектури проєкту та процес його подальшої підтримки.

Під час розробки гри «Fantasy Poly Survivor» було виділено ряд основних класів, які відповідають за реалізацію окремих підсистем гри. Центральним класом системи є GameManager, який виконує функції загального керування грою. Даний клас відповідає за запуск ігрової сесії, її призупинення та завершення. Крім того, він забезпечує взаємодію між іншими компонентами системи та координує їх роботу під час виконання гри.

Для організації переходів між ігровими сценами використовується клас SceneController. Він забезпечує завантаження головного меню, ігрового лобі та основної арени. Наявність окремого класу для керування сценами дозволяє розділити логіку навігації між екранами від основного ігрового процесу.

Одним із ключових компонентів гри є клас PlayerController, який відповідає за керування персонажем. Даний клас реалізує пересування гравця ігровою картою, виконання атак та обробку отримання шкоди від ворогів. Через цей клас користувач взаємодіє з ігровим світом, тому він є одним із центральних елементів архітектури проєкту.

Інформація про поточний стан персонажа зберігається у класі PlayerStats. До його складу входять такі характеристики, як кількість здоров’я, поточний рівень та накопичений досвід. Також цей клас містить методи для підвищення рівня персонажа після отримання необхідної кількості очок досвіду.

Для реалізації бойової системи використовується клас WeaponSystem, який відповідає за використання та покращення зброї. Окремий клас Weapon містить параметри конкретної зброї, зокрема величину шкоди та швидкість атаки. Такий підхід дозволяє легко розширювати арсенал гри та додавати нові типи озброєння без зміни основної структури проєкту.

Важливу роль у геймплеї відіграють супротивники. Їх поведінка реалізована за допомогою класу EnemyController, який відповідає за пересування ворогів до гравця, виконання атак та обробку загибелі противника. Характеристики ворогів винесені до окремого класу EnemyStats, де зберігаються значення здоров’я, швидкості пересування та сили атаки.

Для організації появи хвиль супротивників використовується клас WaveManager. Його основними функціями є запуск хвилі, створення ворогів та завершення поточного етапу. Безпосереднє створення ворогів на ігровій арені виконує клас SpawnManager, який використовується іншими підсистемами для генерації ігрових об’єктів.

Окремою складовою гри є система покращень, реалізована класом UpgradeManager. Даний клас відповідає за відображення доступних покращень після підвищення рівня персонажа та застосування обраного користувачем бонусу. Система покращень взаємодіє як із характеристиками героя, так і з параметрами зброї.

Для відображення інформації користувачу використовується клас UIManager. Він забезпечує оновлення показників здоров’я, досвіду та інформації про поточну хвилю ворогів. Наявність окремого класу інтерфейсу дозволяє відокремити логіку відображення даних від основного ігрового процесу.

Клас CameraController відповідає за роботу камери та її переміщення відносно персонажа. Основним його завданням є забезпечення постійного спостереження за гравцем під час проходження рівня.

Між класами присутні зв’язки типу асоціації та композиції, які забезпечують взаємодію окремих підсистем між собою. Наприклад, клас PlayerController використовує класи PlayerStats та WeaponSystem, а система хвиль ворогів взаємодіє зі SpawnManager для створення нових противників. Подібна структура дозволяє досягти високої модульності програмного продукту, спрощує підтримку коду та забезпечує можливість подальшого розширення функціоналу гри. Діаграму класів програмного продукту зображено на рисунку 3.1.

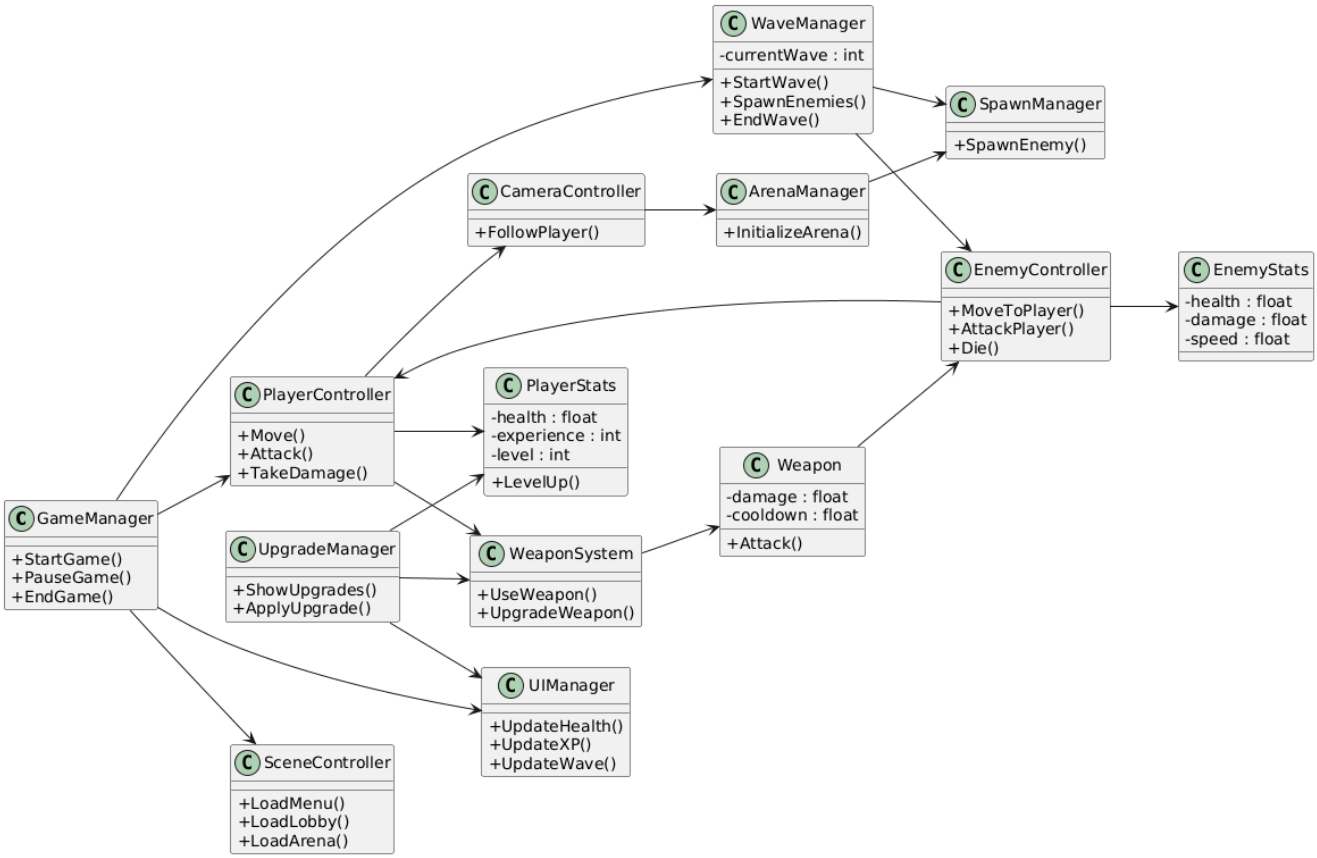


Рис. 3.1 Діаграма класів проєкту

3.3 Розробка основних механік

Розробка основних механік є одним із найважливіших етапів створення гри «Fantasy Poly Survivors», оскільки саме вони формують основу ігрового процесу та визначають взаємодію користувача з віртуальним середовищем. У межах даного проєкту було реалізовано набір базових механік, характерних для жанру survival-like, включаючи переміщення персонажа, автоматичну систему атак, хвильову генерацію ворогів, накопичення досвіду та систему покращень.

Центральним елементом програмної архітектури виступає клас GameManager. Він відповідає за загальне керування грою, координацію роботи інших підсистем та контроль основних станів ігрової сесії. Після запуску гри GameManager виконує ініціалізацію необхідних компонентів, завантажує арену, створює початкові параметри персонажа та запускає першу хвилю противників.

Для керування персонажем використовується клас PlayerController, який забезпечує обробку переміщення гравця по ігровій арені та взаємодію з іншими об'єктами середовища. Характеристики персонажа, такі як кількість здоров'я, поточний рівень та накопичений досвід, зберігаються у класі PlayerStats. Такий підхід дозволяє розділити логіку керування персонажем та його характеристики, що спрощує подальший розвиток проєкту.

Бойова система реалізована за допомогою класів WeaponSystem та Weapon. Особливістю жанру survival-like є автоматичне використання зброї без необхідності виконання атак вручну. Гравець концентрується на переміщенні та виборі оптимальної позиції на арені, тоді як система зброї автоматично здійснює атаки по найближчих або доступних противниках. Такий підхід робить ігровий процес більш динамічним та дозволяє зосередити увагу на виживанні серед великої кількості ворогів.

Для реалізації противників використовуються класи EnemyController та EnemyStats. Вони відповідають за поведінку ворогів, їх пересування до персонажа, нанесення шкоди та зберігання основних характеристик. Кожен ворог має власний

запас здоров'я, швидкість руху та показник шкоди, що дозволяє створювати різні типи противників і поступово збільшувати складність проходження.

Однією з ключових механік гри є система хвиль ворогів. Для її реалізації використовується клас WaveManager, який контролює поточний номер хвилі, запускає нові етапи та визначає момент їх завершення. Створення ворогів на арені виконується класом SpawnManager. З кожною новою хвилею кількість супротивників збільшується, що забезпечує поступове зростання складності та підтримує інтерес гравця протягом усієї сесії.

Важливою складовою ігрового процесу є система розвитку персонажа. Після знищення ворогів гравець отримує досвід, який накопичується та використовується для підвищення рівня. При досягненні нового рівня активується система покращень, реалізована за допомогою класу UpgradeManager. Гравцю пропонується вибір одного з доступних покращень, що дозволяє адаптувати стиль проходження відповідно до власних вподобань та робить кожну ігрову сесію частково унікальною.

Для відображення важливої інформації використовується клас UIManager. Він відповідає за оновлення показників здоров'я персонажа, кількості отриманого досвіду, поточного рівня та номера хвилі. Завдяки цьому гравець постійно отримує актуальну інформацію про стан персонажа та перебіг ігрової сесії.

Загалом усі реалізовані механіки працюють як єдина взаємопов'язана система. Гравець переміщується ареною, знищує ворогів, отримує досвід, покращує характеристики персонажа та намагається протриматися якомога довше під час безперервного збільшення складності гри. Саме така структура є характерною для сучасних survival-like проєктів та лежить в основі розробленої гри «Fantasy Poly Survivors». Алгоритм роботи основного ігрового циклу наведено на рисунку 3.2.

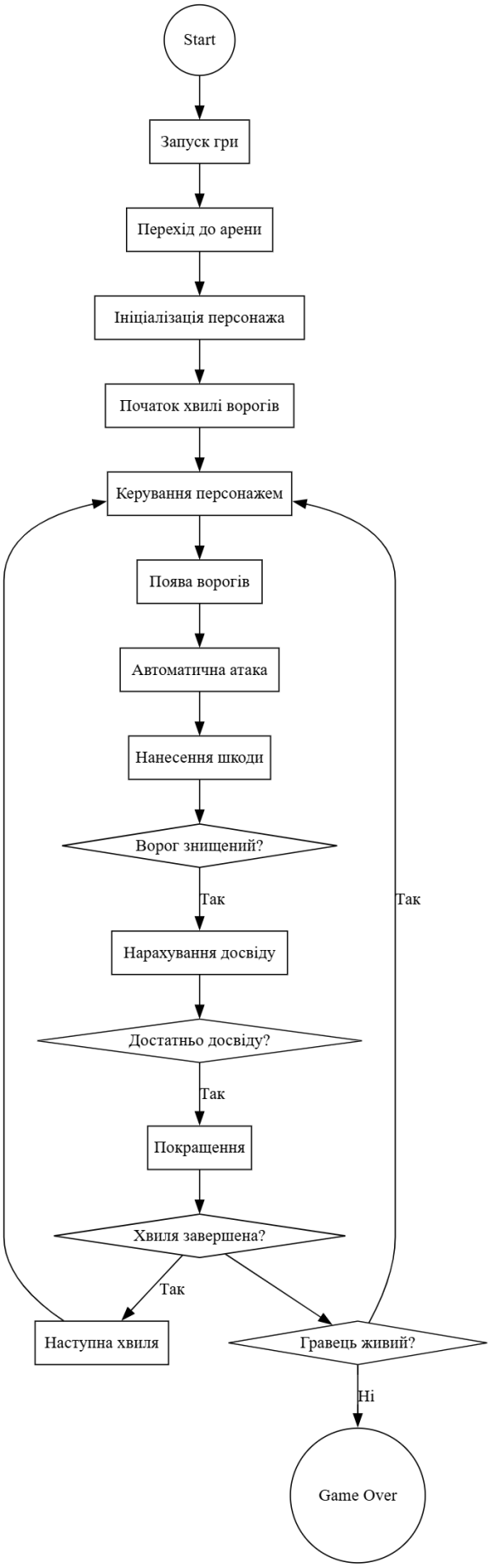


Рис. 3.2Діаграма послідовності

Рис. 3.3 Діаграма прецедентів

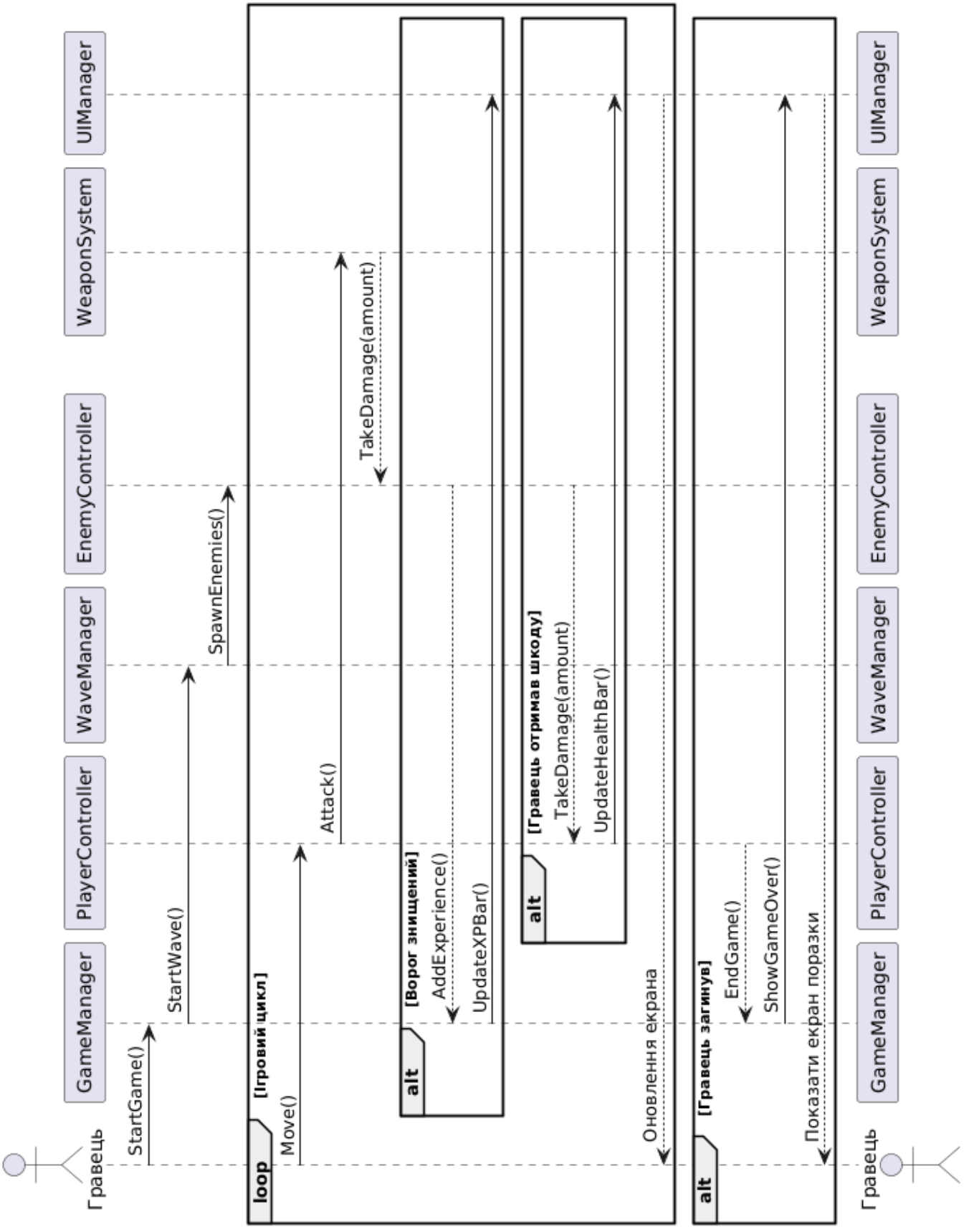


Рис. 3.4 Діаграма послідовності

3.4. Реалізація графічного інтерфейсу користувача

Графічний інтерфейс користувача (GUI) у грі «Fantasy Poly Survivors» є одним із ключових елементів, що забезпечує взаємодію гравця з ігровими системами та формує загальне сприйняття ігрового процесу. Саме інтерфейс дозволяє отримувати інформацію про стан персонажа, хвилі ворогів, прогрес розвитку та доступні покращення, а також швидко реагувати на зміни в ігровій ситуації.

Реалізація графічного інтерфейсу виконувалась із використанням вбудованої UI-системи рушія Unity, що дозволяє створювати інтерактивні елементи, такі як текстові індикатори, панелі стану, кнопки керування та спливаючі меню. Інтерфейс тісно інтегрований з ігровою логікою, що забезпечує його динамічне оновлення в реальному часі.

Після запуску гри користувач потрапляє в головне меню, яке виконує функцію стартового екрану та дозволяє розпочати нову ігрову сесію або вийти з гри. Головне меню є першою точкою взаємодії гравця з проєктом і формує початкове враження про гру. Приклад головного меню наведено на рисунку 3.3.

Рис. 3.5 Головне меню

Основна ігрова сцена являє собою арену, на якій відбувається боротьба гравця з хвилями ворогів. Саме тут реалізується основний ігровий цикл: переміщення персонажа, атака ворогів, отримання досвіду та поступове посилення гравця через систему покращень. Приклад ігрової сцени наведено на рисунку 3.4.

Рис. 3.6 Ігрова сцена (арена бою)

Інтерфейс під час гри побудований у вигляді HUD (Head-Up Display), який відображає основні показники стану гравця. Серед них рівень здоров’я (HP), отриманий досвід (XP), поточна хвиля ворогів, а також інші важливі параметри, що дозволяють оцінювати ситуацію в реальному часі.

Окрему важливу роль відіграє система покращень, яка з’являється після накопичення достатньої кількості досвіду. Вона дозволяє гравцю обирати один із запропонованих варіантів розвитку персонажа, що безпосередньо впливає на стиль проходження та подальшу складність гри. Це створює елемент варіативності та підсилює реіграбельність.

ТЕСТУВАННЯ

Тестування є процесом перевірки та оцінки програмного забезпечення з метою виявлення помилок, недоліків та невідповідностей вимогам. Це один із ключових етапів розробки, який дозволяє переконатися, що ігровий продукт працює стабільно, коректно та відповідає задуму розробника.

У процесі тестування перевіряється функціональність гри, стабільність роботи, продуктивність, коректність взаємодії ігрових систем, а також загальна якість користувацького досвіду. Особлива увага приділяється перевірці бойової системи, системи хвиль ворогів, прогресії персонажа та інтерфейсу користувача.

Метою тестування є забезпечення стабільної роботи гри, виявлення критичних помилок до релізу та підтвердження того, що всі ігрові механіки працюють відповідно до задуманої логіки. Також тестування дозволяє перевірити баланс складності, коректність взаємодії систем та відсутність критичних збоїв під навантаженням.

Процес тестування включає створення тестових сценаріїв, їх виконання, фіксацію результатів та подальше виправлення виявлених помилок. Для перевірки гри використовуються як ручні тести, так і спостереження за поведінкою системи під час ігрових сесій.

4.1 Типи тестів

У процесі тестування гри «Fantasy Poly Survivors» використовуються різні типи тестів, кожен з яких спрямований на перевірку окремих аспектів функціонування системи.

1. Функціональні тести.

Перевіряють коректність роботи основних механік гри: рух персонажа, атака ворогів, отримання досвіду, підвищення рівня, вибір покращень та проходження хвиль.

2. Інтеграційні тести.

Перевіряють взаємодію між системами гри, такими як бойова система, система спавну ворогів, UI та система прогресії персонажа.

3.Юніт-тести.

Використовуються для перевірки окремих компонентів, наприклад логіки отримання шкоди, роботи характеристик персонажа або розрахунку шкоди зброї.

4.Стрес-тести.

Перевіряють стабільність гри при великій кількості ворогів на екрані, інтенсивних боях та високому навантаженні системи спавну.

5. Тести продуктивності.

Оцінюють FPS, навантаження на процесор і пам’ять під час активного ігрового процесу, особливо при пізніх хвилях ворогів.

6. Тести стабільності.

Перевіряють відсутність критичних помилок, зависань або аварійного завершення гри під час тривалих сесій.

7. Тести сумісності.

Перевіряють коректну роботу гри на різних роздільних здатностях екранів та конфігураціях ПК.

4.2 Тестові сценарії

Тестові сценарії визначають послідовність дій, які виконуються для перевірки основних механік гри «Fantasy Poly Survivors». Вони дозволяють перевірити коректність роботи ігрових систем у реальних умовах геймплею та виявити можливі помилки у взаємодії компонентів.

Тестовий сценарій – “Запуск гри та початок ігрового процесу” Кроки:

1. Запуск гри з головного меню.
2. Перехід до ігрової сцени (Arena).
3. Ініціалізація гравця та основних компонентів гри.
4. Запуск першої хвилі ворогів через WaveManager.

5. Перевірка коректного відображення інтерфейсу користувача (HP, XP, Wave).

Цей тестовий сценарій дозволяє перевірити правильність ініціалізації гри, завантаження сцени та старту основного ігрового процесу.

Тестовий сценарій – “Бойова система”

Кроки:

1. Початок ігрової сесії на арені.
2. Поява ворогів у межах ігрової зони.
3. Автоматична атака гравця по ворогах.
4. Нанесення шкоди ворогам та зменшення їх HP.
5. Отримання шкоди гравцем при контакті з ворогами.
6. Перевірка знищення ворогів після втрати всього HP.

Цей тестовий сценарій дозволяє перевірити коректність бойової системи, включаючи взаємодію гравця та ворогів, а також систему нанесення та отримання шкоди.

Тестовий сценарій – “Отримання досвіду та підвищення рівня”

Кроки:

1. Знищення одного або декількох ворогів.
2. Нарахування гравцю очок досвіду (XP).
3. Перевірка заповнення шкали досвіду.
4. Досягнення необхідного значення XP для підвищення рівня.
5. Відображення меню вибору покращень.
6. Вибір та застосування одного з доступних покращень.

Цей тестовий сценарій дозволяє перевірити коректність системи прогресії персонажа, включаючи отримання досвіду та систему покращень.

Тестовий сценарій – “Система хвиль ворогів”

Кроки:

1. Запуск першої хвилі ворогів.
2. Спавн ворогів через SpawnManager.
3. Активна взаємодія гравця з ворогами.

4. Знищення всіх ворогів поточної хвилі.
5. Автоматичний перехід до наступної хвилі.
6. Збільшення складності наступної хвилі.

Цей тестовий сценарій дозволяє перевірити коректність роботи системи хвиль, включаючи спавн ворогів та масштабування складності.

Тестовий сценарій – “Завершення гри”

Кроки:

1. Зменшення значення HP гравця до нуля.
2. Виклик події смерті персонажа.
3. Зупинка ігрової логіки.
4. Відображення екрану завершення гри.
5. Показ результату гри та статистики.
6. Перехід до головного меню.

Цей тестовий сценарій дозволяє перевірити правильність завершення гри та обробку події смерті гравця.

Тестовий сценарій – “Інтерфейс користувача”

Кроки:

1. Запуск гри та відображення HUD-інтерфейсу.
2. Перевірка відображення HP, XP та номера хвилі.
3. Оновлення показників у реальному часі під час гри.
4. Перевірка меню покращень після підвищення рівня.
5. Перевірка меню паузи та основної навігації.
6. Перевірка коректності відображення всіх UI елементів.

Цей тестовий сценарій дозволяє перевірити коректність роботи графічного інтерфейсу та його взаємодію з ігровими системами.

Тестовий сценарій – “Продуктивність”

Кроки:

1. Запуск гри з максимальною кількістю активних ворогів.
2. Відстеження частоти кадрів (FPS) під час бою. ○
3. Перевірка стабільності FPS у різних ігрових ситуаціях.

4. Аналіз навантаження на процесор та пам’ять.
5. Перевірка відсутності критичних просідань продуктивності.
6. Тестування гри на різних конфігураціях пристроїв.

Цей тестовий сценарій дозволяє оцінити продуктивність гри та її стабільність при високому навантаженні.

Після проведення тестування було встановлено, що всі основні механіки гри працюють коректно. Ігрові системи, включаючи бойову систему, систему хвиль ворогів, прогресію персонажа та інтерфейс користувача, функціонують стабільно. Виявлені незначні помилки були усунені в процесі доопрацювання проєкту.

ВИСНОВКИ

Під час дослідження та аналізу сучасних ігор у жанрі survival-like було визначено їхні ключові механіки, особливості прогресії персонажа та принципи побудови хвильового геймплею. Це дозволило сформувати чіткі вимоги до розробки власного ігрового проєкту *Fantasy Poly Survivors*, з урахуванням сучасних підходів до створення ігрових систем.

Використання ігрового рушія Unity та мови програмування C# забезпечило можливість реалізації основних механік гри, включаючи систему керування персонажем, бойову систему, систему хвиль ворогів, прогресію рівня та систему покращень. Завдяки цьому було досягнуто стабільної роботи гри та узгодженої взаємодії всіх ігрових компонентів.

Проєктування архітектури гри здійснювалося на основі компонентного підходу Unity. Було розроблено ключові системи, зокрема GameManager, WaveManager, SpawnManager, систему контролю гравця, систему ворогів та систему інтерфейсу користувача. Така структура забезпечила гнучкість, масштабованість та зручність подальшого розширення функціоналу гри.

Розробка графічного інтерфейсу користувача дозволила забезпечити зручну взаємодію гравця з ігровими механіками. Інтерфейс відображає основні параметри персонажа, такі як здоров’я, досвід та поточна хвиля, а також надає доступ до системи покращень і керування грою. Візуальна складова гри виконана у стилі low-poly, що відповідає загальній концепції проєкту та підкреслює його стилізовану графіку.

Тестування гри було проведено відповідно до розроблених тестових сценаріїв, які охоплюють основні аспекти ігрового процесу. У результаті тестування підтверджено коректну роботу бойової системи, системи хвиль ворогів, прогресії персонажа та інтерфейсу користувача. Виявлені незначні помилки були усунуті на етапі доопрацювання, що підвищило стабільність ігрового продукту.

У результаті виконання дипломного проєкту було створено функціональну гру *Fantasy Poly Survivors* у жанрі survival-like з елементами прогресії та хвильового

виживання. Досягнуто поставленої мети — реалізації стабільного, розширюваного та іграбельного проєкту на основі Unity та C#.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Касяненко В.В., Куклінський М.В. створення 3d гри на рушію unity “Fantasy Poly Survivors”. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 393-395.

2. Касяненко В.В., Куклінський В.М. Розробка 3D-гри «Fantasy Poly Survivors» на рушії unity. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій, Збірник тез. К.: ДУІКТ, 2026. С. 154-157.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unity Technologies. Unity Manual. San Francisco : Unity Technologies, 2025. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 02.06.2026).

2. Unity Technologies. Unity Scripting API. San Francisco : Unity Technologies, 2025. URL: https://docs.unity3d.com/ScriptReference/ (дата звернення: 02.06.2026).

3. Hardman C. Game Programming with Unity and C#: A Complete Beginner's Guide. 2nd ed. Berkeley : Apress, 2024. 434 p.

4. Ferrone H. Learning C# by Developing Games with Unity. 7th ed. Birmingham : Packt Publishing, 2022. 466 p.

5. Nystrom R. Game Programming Patterns. Boston : Genever Benning, 2021. 354 p.

6. Rabin S. Game AI Pro 360: Guide to AI for Computer Games. Boca Raton : CRC Press, 2021. 818 p.

7. Gregory J. Game Engine Architecture. 4th ed. Boca Raton : CRC Press, 2023. 1056 p.

8. Rogers S. Level Up! The Guide to Great Video Game Design. 3rd ed. Hoboken : Wiley, 2024. 560 p.

9. Totten C. An Architectural Approach to Level Design. 2nd ed. Boca Raton : CRC Press, 2024. 528 p.

10. Microsoft Corporation. C# Documentation. Redmond : Microsoft, 2025. URL: https://learn.microsoft.com/dotnet/csharp/ (дата звернення: 02.06.2026).

11. Microsoft Corporation. .NET Documentation. Redmond : Microsoft, 2025. URL: https://learn.microsoft.com/dotnet/ (дата звернення: 02.06.2026).

12. Microsoft Corporation. Visual Studio Documentation. Redmond : Microsoft, 2025. URL: https://learn.microsoft.com/visualstudio/ (дата звернення: 02.06.2026).

13. Microsoft Corporation. Visual Studio Code Documentation. Redmond : Microsoft, 2025. URL: https://code.visualstudio.com/docs (дата звернення: 02.06.2026).

14. Blender Foundation. Blender Manual. Amsterdam : Blender Foundation, 2025. URL: https://docs.blender.org/manual/en/latest/ (дата звернення: 02.06.2026).

15. Hess R. Blender Foundations: The Essential Guide to Learning Blender 4.0. New York : Routledge, 2024. 402 p.

16. Williams J. Blender 4 for Beginners. Birmingham : Packt Publishing, 2024. 520 p.

17. Madhav S. Game Design with Unity and C#. Birmingham : Packt Publishing, 2023. 412 p.

18. Chandler H. The Game Production Handbook. 4th ed. Burlington : Jones & Bartlett Learning, 2023. 430 p.

19. Schell J. The Art of Game Design: A Book of Lenses. 4th ed. Boca Raton : CRC Press, 2024. 640 p.

20. Novak J. Game Development Essentials: An Introduction. 4th ed. New York : Delmar Cengage Learning, 2022. 720 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Розробка 3D-гри “FANTASY POLY SURVIVORS” у жанрі vampire like

Виконав студент 4 курсу
групи ПД-41
Касяненко Віталій Валерійович
Керівник роботи
Професор кафедри, канд.техн.наук. Куклінський Максим Володимирович

Київ - 2026

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення користувацького досвіду в survival-іграх, шляхом розроблення 3D гри «Fantasy Poly Survivors» на рушії Unity з використанням low-poly графіки, системи хвиль ворогів та RPG-механік.

Об’єкт дослідження: процес виживання в іграх стилі vampire-like.

Предмет дослідження: методи та засоби реалізації ігрових механік, системи прогресії персонажа, low-poly графіки, архітектури гри та бойової взаємодії на рушії Unity із використанням мови програмування C#.

КОНЦЕПТ ГРИ

Сетинг - фентезійний світ, у якому герой змушений виживати серед нескінченних хвиль монстрів, що з'являються з магічних розломів.

Сюжет - гравець бере на себе роль воїна, який протистоїть силам Темряви та намагається якомога довше вижити на арені, знищуючи ворогів та стаючи сильнішим.

Ключові механіки:
1. Керування персонажем у 3D-просторі.
2. Автоматичні атаки та використання зброї.
3. Система хвиль ворогів.
4. Отримання досвіду та підвищення рівня.
5. Вибір покращень персонажа.
6. Різні типи зброї та атак.
7. Система здоров'я гравця та ворогів.

Ключові особливості:
1. Low-poly 3D-графіка.
2. Top-down камера.
3. Фентезійна атмосфера.
4. Велика кількість ворогів на арені.
5. Система прогресії персонажа.
6. Поєднання механік survival та RPG.
7. Розробка на рушії Unity з використанням мови програмування C#.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної області survival-ігор жанру vampire like.
2. Дослідити особливості розробки 3D survival-ігор на рушії Unity.
3. Провести порівняльний аналіз існуючих аналогів: Vampire Survivors, Brotato та Megabonk.
4. Визначити функціональні та нефункціональні вимоги до програмного забезпечення.
5. Обрати стек технологій
6. Спроєктувати архітектуру гри.
7. Реалізувати систему керування персонажем, бойову систему, систему хвиль ворогів, прогресії персонажа та користувацький інтерфейс.
8. Провести тестування гри та програмного забезпечення.

АНАЛІЗ АНАЛОГІВ

Назва	Vampire Survivors	Brotato	Megabonk	Fantasy Poly Survivors
Тип графіки	2D Pixel Art	2D Cartoon	3D	Low-poly 3D
Тип камери	Top-down 2D	Top-down 2D	Top-down 3D	Top-down 3D
3D-простір	-	-	+	+
Low-poly стиль	-	-	-	+
Система класів персонажа	-	-	-	+
Основні недоліки	2D-простір, перевантаження ефектами	Одноманітність та слабка атмосфера	Проблеми балансу та різноманітності	Відсутні у межах концепції
Інші особливості	Простий pixel-art стиль	Гумористичний стиль та аркадність	Акцент на хаосі та RPG-елементах	Авторський low-poly fantasy стиль, 3D арена, система класів

5

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Керування персонажем у 3D-просторі.
2. Автоматична атака на ворогів.
3. Система хвиль ворогів та їх поява.
4. Система здоров'я гравця і противників.
5. Нарахування досвіду, ресурсів і винагороди.
6. Вибір покращень персонажа між хвилями.
7. Відображення показників: HP, XP, таймер, хвиля.
8. Перехід між сценами: меню, лобі, арена.

Нефункціональні вимоги:

1. Стабільна робота гри без критичних помилок(>100 fps 0 критичних помилок).
2. Оптимізована продуктивність на ПК середнього рівня(rtx 2050 4 gb).
3. Швидке завантаження сцен і ресурсів(<1 секунди).
4. Коректна робота UI на різних роздільних здатностях екрана(full HD, QHD).
5. Мінімальні та рекомендовані системні вимоги до ПК (Intel Core i3 / Ryzen 3, 8 ГБ RAM, GTX 1050 — мінімальні; Intel Core i5 / Ryzen 5, 16 ГБ RAM, RTX 2050 і вище — рекомендовані).

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Unity

Blender

VISUAL STUDIO CODE

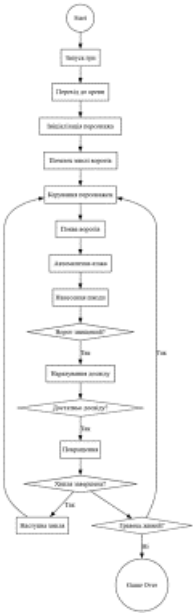
VISUAL STUDIO

7

ДІАГРАМА ПРЕЦЕНДЕНТІВ

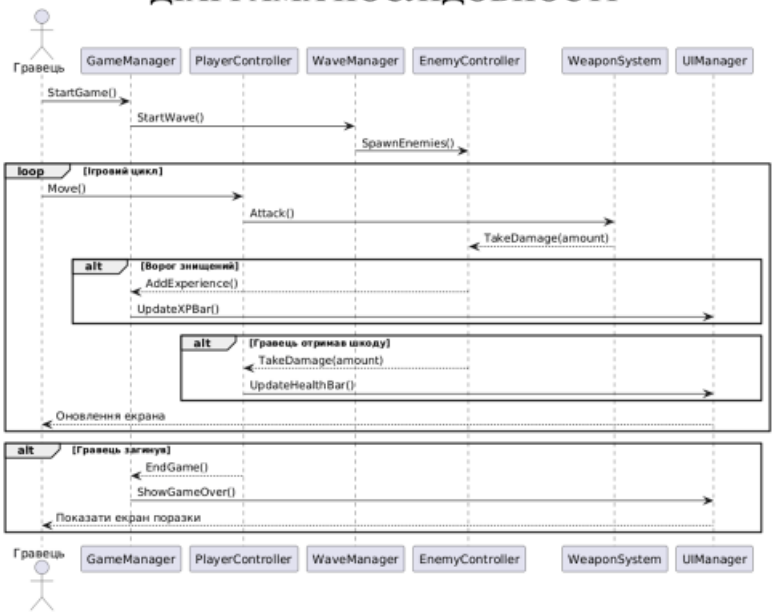
8

ДІАГРАМА ДІЯЛЬНОСТІ



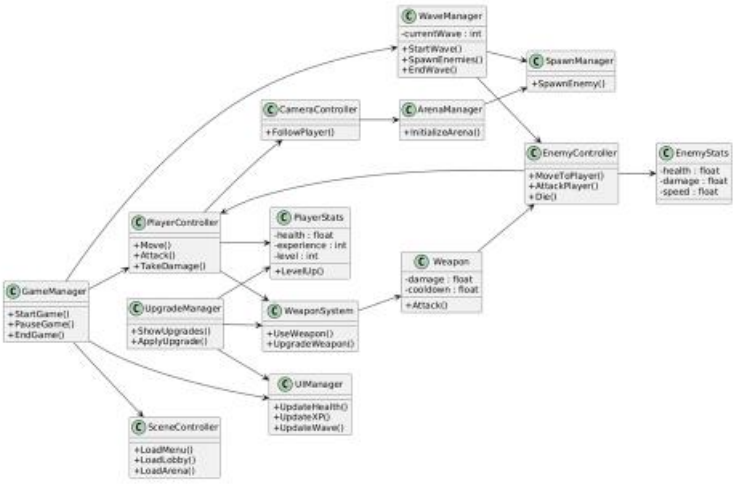
9

ДІАГРАМА ПОСЛІДОВНОСТІ



10

UML-ДІАГРАМА КЛАСІВ



11

ВІДЕО ДЕМОНСТРАЦІЯ ЗАСТОСУНКУ

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Касяненко В.В., Куклінський В.М. Розробка 3D-гри «Fantasy poly survivors» на рушії unity. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», Київ, ДУІКТ, 23.04.2026. Збірник тез. с. 6-154
2. Касяненко В.В., Куклінський М.В. СТВОРЕННЯ 3D ГРИ НА РУШІЮ UNITY "FANTASY POLY SURVIVORS. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», Київ, ДУІКТ, 26.11.2025. Збірник тез. с. 10

13

ВИСНОВКИ

1. Проведено аналіз предметної області survival-ігор жанру vampire like та визначено основні проблеми сучасних інді-проєктів.
2. Виконано порівняльний аналіз існуючих аналогів Vampire Survivors, Brotato та Megabonk, їх переваг і недоліків.
3. Сформовано функціональні та нефункціональні вимоги до програмного продукту.
4. Спроєктовано архітектуру гри «Fantasy Poly Survivors».
5. Реалізовано систему керування персонажем, бойову систему та систему хвиль ворогів.
6. Реалізовано систему прогресії персонажа, low-poly 3D-середовище та користувацький інтерфейс.
7. Проведено тестування гри та аналіз стабільності роботи основних систем.
8. Підготовлено звіт, презентацію та матеріали до захисту кваліфікаційної роботи.

14

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
using System.Collections;
using UnityEngine;

public class WaveManager : MonoBehaviour
{
    public WaveUI waveUI;
    public EnemySpawner enemySpawner;
    public EnemyRegistry enemyRegistry;
    public WaveJsonLoader waveJsonLoader;
    public SpawnGateRegistry gateRegistry;

    [Header("Wave Timing")]
    public float prepareTime = 2f;
    public float waveCompleteDelay = 2f;
    public float endlessWaveCooldown =
10f;

    private WaveDatabase waveDatabase;
    private int currentWaveIndex = -1;
    private bool waveInProgress = false;
    private bool endlessModeStarted =
false;

    void Start()
    {
        waveDatabase =
waveJsonLoader.LoadDatabase();

        if (waveDatabase == null ||
waveDatabase.waves == null ||
waveDatabase.waves.Length == 0)
        {
            Debug.LogError("No waves
loaded!");
            return;
        }


StartCoroutine(StartNextWaveRoutine());
    }

    void Update()
    {
        if (waveInProgress)
        {
            if (waveUI != null &&
enemyRegistry != null)
            {

waveUI.SetEnemiesLeft(enemyRegistry.TotalA
ctiveThreats);
            }

            if (enemyRegistry != null &&
enemyRegistry.TotalActiveThreats <= 0)
            {
                waveInProgress = false;

StartCoroutine(EndWaveRoutine());
            }
        }
    }

    IEnumerator StartNextWaveRoutine()
    {
        currentWaveIndex++;

        if (currentWaveIndex >=
waveDatabase.waves.Length)
        {
            endlessModeStarted = true;

StartCoroutine(StartEndlessLastWaveRoutine
());
            yield break;
        }

        WaveDefinition wave =
waveDatabase.waves[currentWaveIndex];

        if (waveUI != null)
        {

waveUI.SetWaveNumber(wave.waveNumber);
            waveUI.SetStatus("Prepare for
Battle");
            waveUI.SetEnemiesLeft(0);
        }

        if (GameManager.Instance != null)
        {

GameManager.Instance.RegisterWaveReached(w
ave.waveNumber);
        }

        yield return new
WaitForSeconds(prepareTime);

        if (waveUI != null)
        {
            waveUI.SetStatus("Wave
Started");
        }

        yield return
StartCoroutine(RunWave(wave));

        waveInProgress = true;
    }

    IEnumerator
StartEndlessLastWaveRoutine()
    {
        WaveDefinition lastWave =
waveDatabase.waves[waveDatabase.waves.Leng
th - 1];
        int endlessWaveNumber =
lastWave.waveNumber;

        while (true)
        {
            endlessWaveNumber++;

            if (waveUI != null)
            {

waveUI.SetWaveNumber(endlessWaveNumber);
                waveUI.SetStatus("Next
Endless Wave In 10s");
```

```
                waveUI.SetEnemiesLeft(0);
            }

            if (GameManager.Instance !=
null)
            {

GameManager.Instance.RegisterWaveReached(e
ndlessWaveNumber);
            }

            yield return new
WaitForSeconds(endlessWaveCooldown);

            if (waveUI != null)
            {
                waveUI.SetStatus("Endless
Wave Started");
            }

            yield return
StartCoroutine(RunWave(lastWave));

            waveInProgress = true;

            while (waveInProgress)
            {
                yield return null;
            }
        }
    }

    IEnumerator RunWave(WaveDefinition
wave)
    {
        foreach (WaveSpawnEntry entry in
wave.entries)
        {
            yield return new
WaitForSeconds(entry.startDelay);

            for (int i = 0; i <
entry.count; i++)
            {

enemySpawner.SpawnEnemy(entry.enemyId,
entry.gateId, entry.level);
                yield return new
WaitForSeconds(entry.spawnInterval);
            }
        }
    }

    IEnumerator EndWaveRoutine()
    {
        if (waveUI != null)
        {
            waveUI.SetStatus("Wave
Complete");
            waveUI.SetEnemiesLeft(0);
        }

        yield return new
WaitForSeconds(waveCompleteDelay);

        if (!endlessModeStarted)
        {

StartCoroutine(StartNextWaveRoutine());
        }
    }
}
using UnityEngine;

public class EnemyStats : MonoBehaviour
{
    public string enemyId;
    public string displayName;

    public int currentLevel = 1;

    public int maxHealth;
    public int damage;
    public float moveSpeed;

    public int xpReward;
    public int goldReward;
    public int essenceReward;

    public Color enemyColor = Color.white;

    public void
ApplyDefinition(EnemyDefinition
definition, int level)
    {
        if (definition == null) return;

        enemyId = definition.enemyId;
        displayName =
definition.displayName;
        currentLevel = level;

        maxHealth =
definition.baseMaxHealth + (level - 1) *
5;
        damage = definition.baseDamage +
(level - 1);
        moveSpeed =
definition.baseMoveSpeed + (level - 1) *
0.2f;

        xpReward = definition.xpReward;
        goldReward =
definition.goldReward;
        essenceReward =
definition.essenceReward;

        enemyColor = new
Color(definition.colorR,
definition.colorG, definition.colorB);

        ApplyColor();
    }

    void ApplyColor()
    {
        Renderer renderer =
GetComponent<Renderer>();
        if (renderer != null)
        {
            renderer.material.color =
enemyColor;
        }
    }
}
using UnityEngine;

public class WeaponUnlockManager :
MonoBehaviour
{
```

```
    public WeaponUnlockConfig
unlockConfig;

    public bool IsUpgradeAvailable(string
upgradeId)
    {
        if (unlockConfig == null) return
true;

        switch (upgradeId)
        {
            case "sword_size":
                return
unlockConfig.IsWeaponUnlocked("sword");

            case "axe_speed":
                return
unlockConfig.IsWeaponUnlocked("axe");

            case "circle_radius":
                return
unlockConfig.IsWeaponUnlocked("circle");

            case "chakram_bounce":
                return
unlockConfig.IsWeaponUnlocked("chakram");

            default:
                return true;
        }
    }

    public string
GetRequiredWeaponUnlockText(string
upgradeId)
    {
        if (unlockConfig == null) return
"Locked";

        switch (upgradeId)
        {
            case "sword_size":
                return "Unlocked at Level
" +
unlockConfig.GetRequiredLevel("sword");

            case "axe_speed":
                return "Unlocked at Level
" + unlockConfig.GetRequiredLevel("axe");

            case "circle_radius":
                return "Unlocked at Level
" +
unlockConfig.GetRequiredLevel("circle");

            case "chakram_bounce":
                return "Unlocked at Level
" +
unlockConfig.GetRequiredLevel("chakram");

            default:
                return "Locked";
        }
    }
}
using System.Collections.Generic;
using UnityEngine;

public class WeaponManager : MonoBehaviour
{
    public List<BaseWeapon> allWeapons =
new List<BaseWeapon>();

    void Start()
    {
        foreach (BaseWeapon weapon in
allWeapons)
        {
            if (weapon != null)
            {
                weapon.enabled = false;
            }
        }
    }

    public void UnlockWeapon(int
weaponIndex)
    {
        if (weaponIndex < 0 || weaponIndex
>= allWeapons.Count) return;
        if (allWeapons[weaponIndex] ==
null) return;

        allWeapons[weaponIndex].enabled =
true;
        Debug.Log("Unlocked weapon: " +
allWeapons[weaponIndex].name);
    }
}
using UnityEngine;

public class PlayerStats : MonoBehaviour
{
    public PlayerProgressUI progressUI;

    void Start()
    {
        SyncFromGameManager();
        RefreshUI();
    }

    void SyncFromGameManager()
    {
        if (GameManager.Instance == null)
return;

    }

    public void AddXP(int amount)
    {
        if (GameManager.Instance == null)
return;


GameManager.Instance.AddXP(amount);
        RefreshUI();
    }

    public void AddGold(int amount)
    {
        if (GameManager.Instance == null)
return;


GameManager.Instance.AddGold(amount);
        RefreshUI();
    }

    public void AddEssence(int amount)
    {
```

```
        if (GameManager.Instance == null)
return;

GameManager.Instance.AddEssence(amount);
        RefreshUI();
    }

    void RefreshUI()
    {
        if (progressUI != null)
        {
            progressUI.UpdateUI();
        }
    }
}
using UnityEngine;

public class PlayerWeaponController :
MonoBehaviour
{
    [Header("Weapon Scripts")]
    public MonoBehaviour swordWeapon;
    public MonoBehaviour axeWeapon;
    public MonoBehaviour circleWeapon;
    public MonoBehaviour chakramWeapon;

    public WeaponUnlockConfig
unlockConfig;

    void Start()
    {
        ApplyWeaponUnlocks();
    }

    public void ApplyWeaponUnlocks()
    {
        if (unlockConfig == null)
        {
            Debug.LogError("UnlockConfig
missing on PlayerWeaponController!");
            return;
        }

        SetWeaponState(swordWeapon,
"sword");
        SetWeaponState(axeWeapon, "axe");
        SetWeaponState(circleWeapon,
"circle");
        SetWeaponState(chakramWeapon,
"chakram");
    }

    void SetWeaponState(MonoBehaviour
weaponScript, string weaponId)
    {
        if (weaponScript == null) return;

        bool unlocked =
unlockConfig.IsWeaponUnlocked(weaponId);
        weaponScript.enabled = unlocked;

        Debug.Log("Weapon " + weaponId + "
unlocked = " + unlocked);
    }
}
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneLoader : MonoBehaviour
{
    public void LoadMainMenu()
    {

SceneManager.LoadScene("MainMenu");
    }

    public void LoadLobby()
    {
        SceneManager.LoadScene("Lobby");
    }

    public void LoadBattle()
    {
        SceneManager.LoadScene("Battle");
    }

    public void QuitGame()
    {
        Debug.Log("Quit Game");
        Application.Quit();
    }
}
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class UpgradeState
{
    public string upgradeId;
    public int level;
}

public class GameManager : MonoBehaviour
{
    public static GameManager Instance;

    [Header("Player Progress")]
    public int currentLevel = 1;
    public int currentXP = 0;
    public int xpToNextLevel = 20;

    public int gold = 0;
    public int essence = 0;

    [Header("Wave Progress")]
    public int highestWaveReached = 0;

    [Header("Upgrade States")]
    public List<UpgradeState> upgrades =
new List<UpgradeState>();

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

        Instance = this;
        DontDestroyOnLoad(gameObject);

        SaveSystem.LoadGame(this);
    }

    void OnApplicationQuit()
    {
        SaveSystem.SaveGame(this);
```

```
    }

    public void AddXP(int amount)
    {
        currentXP += amount;

        while (currentXP >= xpToNextLevel)
        {
            currentXP -= xpToNextLevel;
            LevelUp();
        }

        RefreshAllProgressUI();
    }

    void LevelUp()
    {
        currentLevel++;
        xpToNextLevel += 10;

        Debug.Log("LEVEL UP! Level: " +
currentLevel);
    }

    public void AddGold(int amount)
    {
        gold += amount;
        RefreshAllProgressUI();
    }

    public void AddEssence(int amount)
    {
        essence += amount;
        RefreshAllProgressUI();
    }

    public void RegisterWaveReached(int
waveNumber)
    {
        if (waveNumber >
highestWaveReached)
        {
            highestWaveReached =
waveNumber;
            SaveSystem.SaveGame(this);
            RefreshAllProgressUI();

            Debug.Log("New highest wave
reached: " + highestWaveReached);
        }
    }

    public int GetUpgradeLevel(string
upgradeId)
    {
        foreach (var upgrade in upgrades)
        {
            if (upgrade.upgradeId ==
upgradeId)
                return upgrade.level;
        }

        return 0;
    }

    public void
IncreaseUpgradeLevel(string upgradeId)
    {
        foreach (var upgrade in upgrades)
        {
            if (upgrade.upgradeId ==
upgradeId)
            {
                upgrade.level++;
                return;
            }
        }

        upgrades.Add(new UpgradeState
        {
            upgradeId = upgradeId,
            level = 1
        });
    }

    public void SaveGame()
    {
        SaveSystem.SaveGame(this);
    }

    public void DeleteSave()
    {
        SaveSystem.DeleteSave();

        currentLevel = 1;
        currentXP = 0;
        xpToNextLevel = 20;
        gold = 0;
        essence = 0;
        highestWaveReached = 0;

        upgrades.Clear();

        RefreshAllProgressUI();
    }

    public void RefreshAllProgressUI()
    {
        PlayerProgressUI[] allUI =
FindObjectsByType<PlayerProgressUI>(FindOb
jectsSortMode.None);

        foreach (PlayerProgressUI ui in
allUI)
        {
            if (ui != null)
            {
                ui.UpdateUI();
            }
        }

        LobbyUpgradeUI[] allLobbyUI =
FindObjectsByType<LobbyUpgradeUI>(FindObje
ctsSortMode.None);

        foreach (LobbyUpgradeUI ui in
allLobbyUI)
        {
            if (ui != null)
            {

ui.RefreshUpgradeButtons();
            }
        }
    }
}
```