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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка гри в жанрі Top-Down Shooter»

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

_____ Денис СІКОРСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Денис СІКОРСЬКИЙ

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____

Київ 2025

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
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сікорському Денису Вадимовичу

1. Тема кваліфікаційної роботи: «Розробка гри в жанрі Top-Down Shooter» керівник кваліфікаційної роботи доктор філософії (PhD) Олесь ДІБРІВНИЙ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

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

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

1. Огляд та аналіз існуючих ігор та рушіїв для розробки.
2. Проектування гри для адаптації ігрового процесу в грі жанру Top-Down Shooter
3. Програмна реалізація та опис функціонування гри для адаптації ігрового процесу в грі жанру Top-Down Shooter.
4. Тестування гри.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих ігор та методів їх розробки	05.03-13.03.2025	
4	Проектування архітектури гри	14.03-18.03.2025	
5	Програмна реалізація гри	19.03-18.04.2025	
6	Тестування гри	19.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Денис СІКОРСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 55 стор.,1 табл., 20 рис., 20 джерел.

Мета роботи – адаптація ігрового процесу в грі жанру Top-Down Shooter.

Об'єкт дослідження – процес гри в жанрі Top-Down Shooter.

Предмет дослідження – ігровий застосунок в жанрі Top-Down Shooter.

Короткий зміст роботи: У роботі представлено повний цикл розробки двовимірної гри у жанрі Top-Down Shooter з використанням рушія Unity. Висвітлено історичний розвиток жанру, його ключові особливості, а також сучасні тенденції в ігровій індустрії. Проведено огляд існуючих інструментів для створення ігор, зокрема рушіїв, графічних та звукових редакторів, мов програмування й середовищ розробки.

В аналітичній частині розглянуто приклади відомих ігор жанру, виділено їхні переваги та недоліки. Сформульовано цілі й завдання дослідження, а також розроблено структурну модель гри, що включає архітектуру класів, діаграми та підсистеми.

У практичній частині описано реалізацію основного функціоналу: управління гравцем, стрільба, система ворогів, обробка зіткнень та UI. Також приділено увагу налагодженню, тестуванню та оптимізації гри.

Сферою використання гри є новачки, котрі бажають ознайомитись з жанром ігор Top-Down Shooter.

КЛЮЧОВІ СЛОВА: TOP-DOWN SHOOTER, 2D-ГРА, UNITY, ПРОТИВНИК, ГРА BLOOD AND BULLETS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI (User Interface) — інтерфейс користувача; графічні елементи, через які гравець взаємодіє з грою (кнопки, меню, індикатори тощо).

HUD (Heads-Up Display) — інформація, яка відображається на екрані поверх геймплею (здоров’я, боєприпаси, карта тощо).

RPG (Role-Playing Game) — рольова гра; жанр, де гравець керує персонажем, розвиває його характеристики та бере участь у сюжеті.

FPS (First-Person Shooter) — шутер від першої особи; гра, де гравець бачить світ очима персонажа і веде бойові дії.

TPS (Third-Person Shooter) — шутер від третьої особи; камера розташована позаду персонажа, що дозволяє бачити його повністю.

RTS (Real-Time Strategy) — стратегія в реальному часі; гравець керує юнітами/базами без пауз, одночасно з противником.

AI (Artificial Intelligence) — штучний інтелект; поведінка комп’ютерних ворогів або NPC, запрограмована розробником.

NPC (Non-Playable Character) — персонаж, яким не керує гравець (торговці, союзники, вороги).

HP (Health Points) — очки здоров’я; показують життєздатність персонажа.

DLC (Downloadable Content) — завантажуваний контент; додаткові матеріали до гри (рівні, зброя, сюжет тощо).

FPS (Frames Per Second) — кількість кадрів за секунду; показник плавності гри.

SDK (Software Development Kit) — набір інструментів для створення або модифікації гри.

API (Application Programming Interface) — інтерфейс програмування; набір команд для взаємодії з іншими програмами чи сервісами.

ВСТУП

Обґрунтування вибору теми та її актуальність: З огляду на стрімке зростання популярності відеоігор як форми культурного, освітнього й комерційного продукту, зростає й інтерес до розробки ігор різних жанрів, здатних задовольнити запити різних аудиторій. Жанр Top-Down Shooter вирізняється динамічним геймплеєм, високою реіграбельністю та стратегічними елементами, однак далеко не всі існуючі продукти відповідають сучасним стандартам якості, балансу та зручності. Це створює запит на створення нових, функціонально досконалих ігор з урахуванням актуальних технічних можливостей та ігрових трендів.

Ступінь вивчення проблеми: Існує чимало відомих представників жанру, таких як Hotline Miami, Nuclear Throne, Enter the Gungeon, однак вони або надто складні для новачків, або вимагають високого рівня технічного оснащення. Більшість аналогів створені раніше і мають застарілий UI, обмежену підтримку платформ, або не дають можливості подальшої адаптації. Таким чином, існує потреба у розробці нової гри, яка поєднуватиме простоту освоєння, стратегічну глибину та привабливу візуальну складову.

Об'єктом дослідження є процес гри в жанрі Top-Down Shooter.

Предметом роботи є ігровий застосунок в жанрі Top-Down Shooter.

Метою роботи є адаптація ігрового процесу в грі жанру Top-Down Shooter.

Завданням роботи є реалізація технічно стабільного прототипу гри з основними ігровими підсистемами, провести тестування працездатності та оптимізації, а також задокументувати архітектуру проєкту.

Методика дослідження полягала у попередньому аналізі аналогічних продуктів, виборі доцільного інструментарію, створенні архітектурної моделі, розробці функціоналу та проведенні багаторівневого тестування із подальшим аналізом отриманих результатів. Для реалізації було використано середовище Unity, мову програмування C#, інструменти дизайну інтерфейсу та аудіо, систему

контролю версій Git.

Гру було розроблено з орієнтацією на ОС Windows, з перспективою майбутньої адаптації під Android та WebGL.

Наукова новизна полягає у створенні повнофункціонального ігрового прототипу з сучасною архітектурною структурою та оптимізованим ігровим циклом.

Практичне значення результатів полягає в тому, що створений продукт може бути використаний як основа для майбутніх комерційних чи навчальних проєктів у сфері геймдизайну, а також бути інтегрований у курси програмування та розробки ігор як демонстраційна розробка.

1. ОГЛЯД ІСНУЮЧИХ ПРОДУКТІВ В ЖАНРІ

1.1. Різноманіття жанрових класифікацій у відеоіграх

Ігрова індустрія постійно розвивається, породжуючи дедалі більше нових жанрів та піджанрів. Жанр гри – це класифікація, яка описує її механіку, цілі, тип взаємодії гравця з ігровим середовищем і стиль подачі контенту. Вибір жанру має вирішальне значення на етапі розробки, оскільки визначає структуру геймплею, набір функціональностей та очікування аудиторії.

Екшн (Action) — це жанр, у якому головне — швидкість реакції та координація гравця. До нього належать платформери, файтинги та шутери. У Top-Down Shooter гравець одночасно контролює приціл і пересування, уникає ворожих атак і швидко реагує на зміну ситуації. Action-ігри часто мають високу динаміку подій і не прощають помилок, що змушує гравця бути постійно зосередженим. Вони можуть включати бої на ближній дистанції, стрілянину, паркур або швидке переміщення крізь небезпечні зони. Часто в таких іграх присутні боси з унікальними механіками, які вимагають особливої стратегії та навичок. Багато екшн-ігор мають яскравий візуальний стиль і напружений музичний супровід, що підсилює атмосферу бою. Успішність у цьому жанрі зазвичай залежить від майстерності, рефлексів і здатності швидко приймати рішення.

Пригодницькі ігри (Adventure) — це ігри з акцентом на сюжет, дослідження локацій і взаємодію з об’єктами. Гравець розв’язує головоломки, спілкується з NPC і впливає на розвиток історії, часто досліджуючи відкритий світ. Основна мета пригодницьких ігор — занурити гравця в захопливу історію та створити емоційний зв’язок із персонажами. Часто гравець стикається з моральними виборами, які можуть змінити хід подій або фінал гри. У деяких проєктах геймплей включає елементи стелсу, виживання або бойових дій, але завжди на другому плані після сюжету. Важливу роль відіграє атмосфера: ретельно пророблене середовище, музичний супровід і візуальний стиль. Пригодницькі ігри можуть бути як лінійними, так і нелінійними з кількома варіантами проходження.

Рольові ігри (RPG) — це ігри, де гравець розвиває персонажа, проходить квести та бере участь у діалогах. Вибір у розмовах і дії змінюють сюжет, а бій може бути покроковим або в реальному часі. Часто є крафт, інвентар і детальна система прокачування. У RPG велике значення має занурення у вигаданий світ, який часто має власну історію, фракції та складну політичну систему. Гравець може обирати клас персонажа, його риси характеру та стиль гри — від дипломатії до жорсткої сили. Важливою складовою є система лута: зброя, броня, магічні артефакти. Багато рольових ігор мають відкритий світ, який гравець вивчає у власному темпі. Деякі RPG також включають елементи симуляції, керування базою або взаємодію з іншими персонажами у стилі життя.

Стратегії (Strategy) — це жанр, у якому гравець керує військом, ресурсами або містом. На рисунку 1.1 представлено стратегію Civiliztion VII.

Рис. 1.1 Стратегія Civiliztion VII

У покрокових стратегіях дії відбуваються по черзі, а в RTS — у реальному часі. Важлива аналітика, планування та швидке ухвалення рішень, особливо в мультиплеєрі. У стратегічних іграх гравець часто виступає в ролі лідера: генерала, правителя або менеджера, відповідального за успіх своєї фракції чи економіки. Значну роль відіграють технологічне дерево, дипломатія, будівництво та розвиток інфраструктури. У багатьох проєктах реалізована система прогресу через епохи або

кампанії з історичним чи фантастичним контекстом. Стратегії вимагають глибокого розуміння механік і здатності передбачати дії супротивника. Саме тому вони приваблюють гравців, які цінують інтелектуальні виклики й тривале планування.

Симулятори (Simulation) моделюють реальні процеси або діяльність, наприклад, пілотування літака, водіння авто, фермерство чи управління містом. Їх відрізняє високий рівень деталізації, реалістична фізика та можливість глибокого налаштування. Деякі симулятори виконують освітню функцію, а інші створені для розваги широкої аудиторії. У багатьох симуляторах користувачі занурюються в рутину обраної професії або сфери діяльності, що створює ефект повного занурення. Гравці можуть контролювати кожен аспект процесу — від технічних характеристик до погодних умов чи економічних показників. Популярність таких ігор часто ґрунтується на їхній здатності передавати відчуття реального досвіду без ризику. Існують як хардкорні симулятори з великою кількістю параметрів, так і спрощені версії для новачків. Завдяки цьому жанр охоплює як професійних ентузіастів, так і випадкових гравців.

Головоломки (Puzzle) побудовані на завданнях, які вимагають логічного мислення, просторового сприйняття чи креативного підходу. Вони можуть включати механіки переставляння об'єктів, побудову маршрутів або розв'язання числових задач. Часто мають сотні рівнів зі зростаючою складністю, ідеально підходять для коротких сесій гри. Головоломки можуть мати як мінімалістичний, так і яскравий візуальний стиль, що сприяє зосередженню або створює унікальну атмосферу. У деяких іграх додаткову глибину надає сюжет, який розгортається через розв'язання завдань. Puzzle-ігри часто використовуються як тренажери для розвитку мислення та пам’яті. Їх популярність зросла завдяки мобільним платформам і доступності для широкої аудиторії. Крім того, вони нерідко включають систему підказок або навчальні рівні для новачків.

Спортивні ігри відтворюють популярні види спорту — футбол, баскетбол, гонки тощо. У них важливі знання правил гри, стратегія, командна взаємодія й майстерність управління. Залежно від рівня реалізму, управління може бути

аркадним або симуляційним. Часто мають ліцензовані команди, режими кар’єри й онлайн-турніри. Спортивні ігри приваблюють як фанатів реального спорту, так і гравців, які шукають змагальний досвід у віртуальному середовищі. Регулярні оновлення складів, сезонні івенти та онлайн-рейтинг сприяють підтримці активного ком’юніті. Багато з них мають глибоку систему прокачування спортсменів або менеджменту клубу. Крім того, кіберспортивні змагання з таких ігор, як FIFA або NBA 2K, здобули міжнародну популярність. Рисунок 1.2 демонструє вигляд гри FC24.

Рис. 1.2 Спортивна гра FC24

Це робить спортивні симулятори не лише засобом розваги, а й платформою для професійного суперництва.

Казуальні ігри — це легкі, інтуїтивно зрозумілі ігри, розраховані на широку аудиторію. Вони мають просте керування, короткі ігрові сесії та часто орієнтовані на мобільні платформи. Найпопулярніші — клікери, “три в ряд”, ранери. Часто використовують монетизацію через рекламу або внутрішньоігрові покупки. Казуальні ігри швидко захоплюють увагу завдяки яскравому дизайну, простим завданням і системі винагород. Вони рідко потребують попереднього досвіду в

іграх, що робить їх привабливими для новачків. Часто в них впроваджують щоденні бонуси, обмежені за часом події та соціальні функції на кшталт змагань із друзями. Завдяки мінімальному порогу входу, такі ігри ідеально підходять для короткого дозвілля в транспорті чи під час перерви. Казуальний жанр відіграє ключову роль у мобільному геймінгу й приносить значну частку доходів індустрії.

First-Person Shooter (FPS) — шутери від першої особи, де гравець бачить світ очима персонажа. Основна мета — точна стрільба, швидка реакція та виживання в бойових ситуаціях. Відомі приклади — Call of Duty, Battlefield, Counter-Strike. Камера від першої особи підсилює ефект присутності й адреналін під час гри. FPS-ігри часто мають режими одиночної кампанії зі сценарієм і багатокористувацькі матчі з різними завданнями: захоплення точки, командний бій, ескорт. Велике значення мають баланс зброї, дизайн карт і система прогресу — від прокачки навичок до відкриття нового спорядження. У кіберспорті FPS займає чільне місце завдяки своїй динаміці та змагальному потенціалу. Деякі ігри також інтегрують елементи тактики, наприклад, важливість позиціонування чи командної взаємодії. Постійне оновлення контенту та сезонні події підтримують інтерес гравців у довгостроковій перспективі.

Third-Person Shooter (TPS) — шутери з виглядом від третьої особи, де гравець бачить свого героя ззаду. Такий ракурс дозволяє краще орієнтуватися в просторі та використовувати укриття. Приклади: Gears of War, Uncharted, The Division, Call Of Duty. Також у TPS-іграх часто реалізовано покращену анімацію персонажа, оскільки гравець постійно бачить свого героя. Це дозволяє краще передати характер і стиль бою персонажа. Розміщення камери позаду сприяє глибшому зануренню в навколишнє середовище та взаємодії з ним. Багато TPS включають елементи стелсу або тактичного бою, що додає геймплею різноманітності. Шутери від третьої особи часто поєднують бойові дії з платформерами або елементами паркуру, що робить ігровий процес динамічнішим. Завдяки широкому огляду гравець може ефективніше планувати свої дії та виявляти загрози. У сучасних TPS також активно використовуються сюжетні вставки та кінематографічні сцени, що підсилюють емоційне занурення. На рисунку 1.3 зображено гру Call Of Duty.

Рис. 1.3 Відеогра Call of Duty

TPS часто дозволяє кастомізувати персонажа та поєднує стрілянину з елементами тактики. У TPS-іграх важливу роль відіграє позиціонування гравця на полі бою та взаємодія з навколишнім середовищем. Багато таких ігор акцентують увагу на сюжеті, персонажах і кінематографічній подачі. Завдяки видимості героя, гравець може краще відчути його емоції та стиль гри. Часто TPS включають стелс-елементи, паркур або ближній бій, що урізноманітнює геймплей. Також цей піджанр популярний у кооперативних режимах, де взаємодія між гравцями має вирішальне значення.

Top-Down Shooter — це піджанр шутерів з виглядом згори, який дає гравцю широке поле огляду. Управління зазвичай поєднує клавіші руху з прицілюванням мишею. Типові приклади: Hotline Miami, Enter the Gungeon, Nuclear Throne. Ці ігри вимагають швидкої реакції та тактичного мислення, а також поєднують елементи екшену, аркади й іноді RPG. Top-Down Shooter ідеально підходить для інтенсивного геймплею з великою кількістю ворогів і ефектів на екрані. Завдяки вигляду згори, гравець може краще контролювати ситуацію навколо персонажа й планувати переміщення. Часто такі ігри мають рівні, що генеруються процедурно,

що підвищує реіграбельність. Візуальний стиль може варіюватися від мінімалістичного до ретро-піксельного або сучасного з візуальними ефектами. У багатьох Top-Down Shooter'ах також реалізовані кооперативні режими, що дозволяють проходити гру з другом.

1.2. Розвиток ігрової індустрії та її значущість у глобальному контексті

Індустрія відеоігор за останні кілька десятиліть перетворилася з нішового захоплення ентузіастів на одну з провідних галузей розважального бізнесу. Згідно з численними аналітичними звітами, за обсягом доходів ігрова індустрія вже давно обійшла кіно та музичний бізнес, а її розвиток продовжує набирати обертів у всіх регіонах світу. Рисунок 1.4 ілюструє доходи ігрової індустрії за долями ринку.

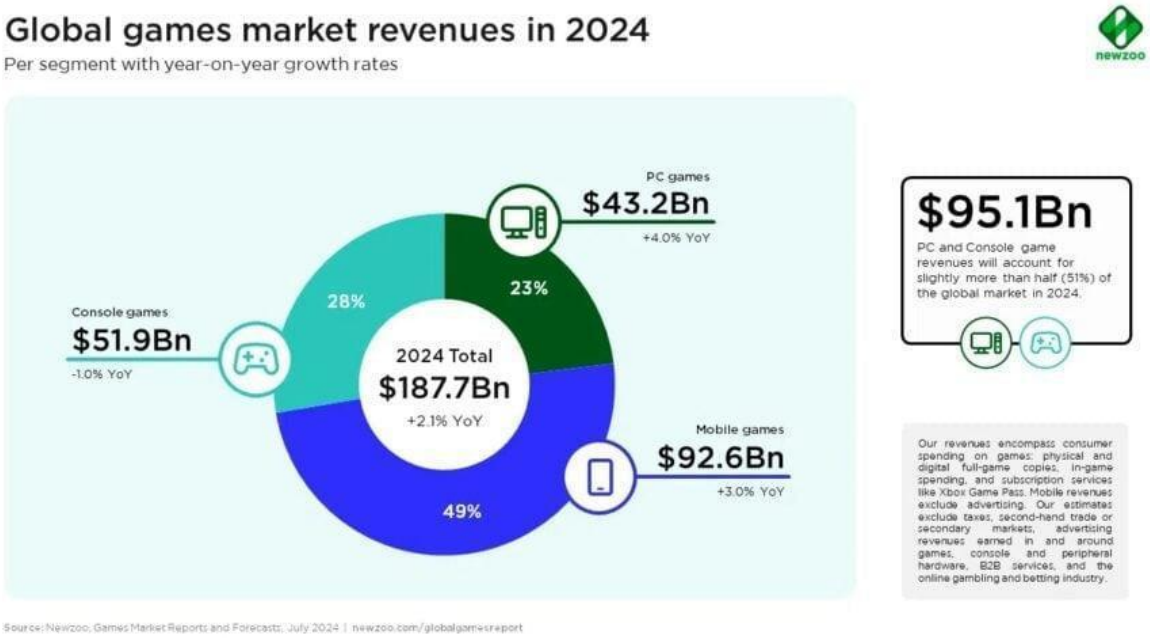


Рис. 1.4 Загальний дохід ігрової індустрії у світі

Перші відеоігри з'явилися у 70-х роках XX століття, коли компанії на кшталт Atari почали випускати аркадні автомати. У 80-х роках на ринку домінували домашні консолі, такі як Nintendo Entertainment System, що започаткували масовий інтерес до ігор. Згодом відбувся перехід до персональних комп’ютерів та

геймінгових платформ нового покоління. Кожна нова епоха супроводжувалась як технічними проривами, так і еволюцією жанрового розмаїття.

З розвитком Інтернету та мобільних технологій ігрова індустрія отримала друге дихання. З'явилися онлайн-ігри, багатокористувацькі середовища, free-to-play моделі, хмарні сервіси та кросплатформні рішення, які суттєво розширили ринки та змінили спосіб сприйняття ігор як продукту.

На сьогодні відеоігри є одним з ключових секторів цифрової економіки. За даними звітів Newzoo та Statista, обсяг світового ринку відеоігор у 2024 році перевищив $200 мільярдів. Основними рушіями зростання стали мобільні ігри, eSports, хмарні сервіси та стримінг-платформи, як-от Twitch і YouTube Gaming.

Крім доходів від продажу ігор, прибутковими напрямами стали мікротранзакції, підписки, внутрішньоігрова реклама, розробка контенту для кіберспорту та створення інфраструктури для гравців. Компанії-гіганти, як-от Sony, Microsoft, Nintendo, Tencent, Ubisoft і Activision Blizzard, виступають не лише виробниками контенту, але й потужними технологічними корпораціями.

Окрім чисто розважального аспекту, відеоігри дедалі більше сприймаються як повноцінний міждисциплінарний арт-проект. Поєднуючи графічний дизайн, сценарну майстерність, композиторське мистецтво, анімацію й програмування, вони формують унікальний «загальнокультурний» простір, де сюжет, візуальний стиль і музичний супровід працюють як єдине художнє полотно. Інтерактивність дозволяє гравцеві взяти участь у творенні історії: вибір діалогів і дій реально змінює розвиток сюжету або поведінку персонажів, а інструменти для створення модифікацій відкривають шлюз до спільнот розробників-аматорів, що розширюють і переосмислюють ігровий світ.

У сфері освітніх застосунків ігри служать потужним візуальним і практичним тренажером. Аерофлотні і симулятори реального польоту забезпечують безпечне відпрацювання навичок пілотування, а медичні симулятори з точністю імітують операційний процес, даючи студентам-хірургам можливість відпрацювати алгоритми дій та розвинути моторні навички без ризику для пацієнта.

Психотерапевтичні та соціальні ігри використовують механіки, що знижують рівень тривоги та стресу через керовану взаємодію й поступове ускладнення завдань. Такі кейси застосовують у реабілітації пост-травматичного стресового розладу, у програмах для людей із соціальною фобією або ASD, де поступове занурення в ігрові сценарії сприяє адаптації та відпрацюванню комунікативних навичок.

У шкільному та позашкільному навчанні освітні ігри й «серйозні» симулятори (serious games) стимулюють розвиток критичного мислення, математичного апарату та креативності через гейміфікацію завдань: від побудови віртуального міста з економічним балансом до вирішення екологічних головоломок із динамікою екосистем. Таким чином, відеоігри стають не тільки джерелом емоцій і вражень, а й інструментом трансформації знань і навичок у різних сферах життя.

Відеоігри об’єднують людей незалежно від національності, віку, статі чи соціального статусу. У сучасному світі гравці утворюють спільноти, беруть участь у турнірах, переглядають трансляції, створюють фан-контент. Завдяки розвитку кіберспорту, ігри стали не лише формою розваги, а й професійною діяльністю для мільйонів людей.

Особливо важливою є інтеграція відеоігор у культурний простір. З’являються екранізації ігор, тематичні виставки, конференції розробників (GDC, E3), а деякі ігрові персонажі набувають статусу ікон. Відеоігри дедалі частіше розглядаються як форма мистецтва, що поєднує в собі графіку, музику, сценарну майстерність та інноваційні технології. Вони стають предметом дослідження в академічних колах і темою обговорення в медіа. Ігрова естетика впливає на моду, дизайн та інші сфери креативної індустрії. На рисунку 1.5 показано візуальний стиль екранізації гри Minecraft. Завдяки цьому відеоігри набувають ширшого визнання не лише як розвага, а як значущий культурний феномен. Їхній вплив можна простежити в кіно, літературі, музиці та навіть у візуальному мистецтві сучасності.

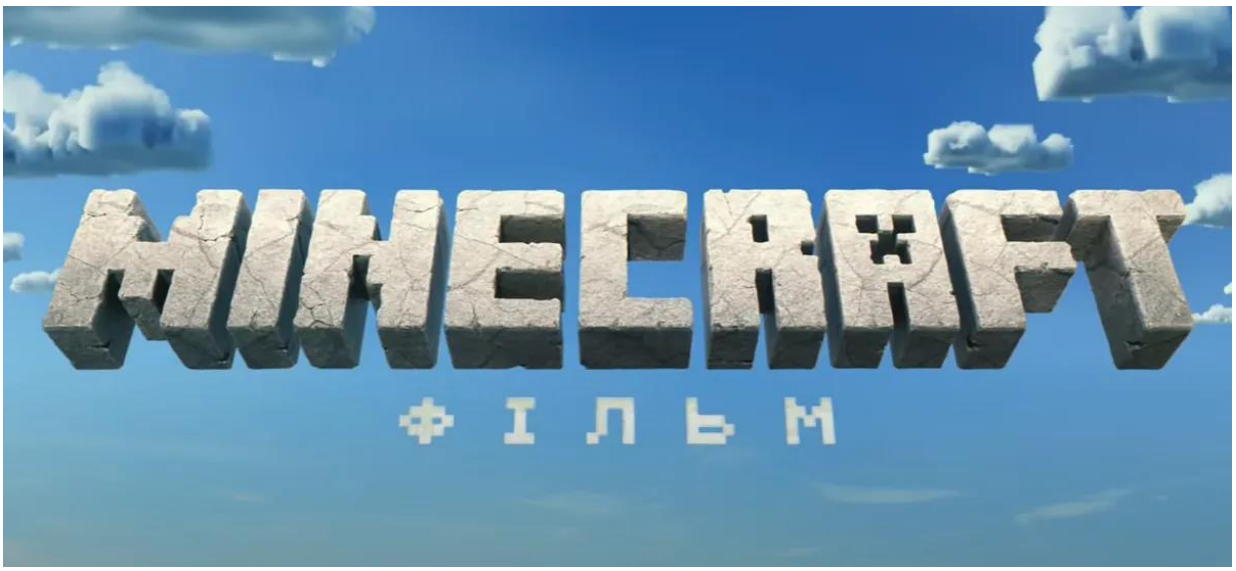


Рис. 1.5 Екранізація гри Minecraft

Відеоігри вже давно перестали бути просто способом розваги для підлітків — сьогодні це одна з найбільших і найвпливовіших форм культурного, соціального й економічного феномену. Вони об’єднують людей незалежно від національності, віку, статі чи соціального статусу, створюючи спільноти, що виходять далеко за межі окремих країн чи мовних бар’єрів. Через спільну гру в кооперативному чи змагальному режимі люди знайомляться, здобувають навички командної роботи, комунікації, а іноді навіть будують кар’єру.

Геймерська культура сьогодні представлена у всіх соціальних прошарках. Онлайн-ігри об'єднують мільйони гравців у масштабних віртуальних світах, таких як World of Warcraft, Final Fantasy XIV, Fortnite, GTA Online, Valorant, де гравці не лише проходять квести або змагаються, а й відвідують віртуальні концерти, фестивалі, дискутують на форумах, створюють і підтримують внутрішні соціальні структури — гільдії, клани, альянси. Це створює унікальне середовище для соціалізації, в якому легко знайти однодумців або сформувати власну групу за інтересами. Такі віртуальні спільноти часто виходять за межі самої гри, переносячись у соціальні мережі, стрімінгові платформи та навіть реальне життя. Геймерська культура поступово формує власні традиції, термінологію й події, що об'єднують людей по всьому світу. У рисунку 1.6 представлено гру GTA Online.

Рис. 1.6 GTA Online

Завдяки розвитку кіберспорту, ігри трансформувалися в професійну діяльність для мільйонів людей по всьому світу. Професійні гравці тренуються по кілька годин щодня, укладають контракти зі спонсорами, беруть участь у турнірах із мільйонними призовими фондами — як наприклад, The International з Dota 2, CS:GO Major, League of Legends World Championship або Valorant Champions Tour. Подібно до традиційного спорту, у кіберспорті є тренери, аналітики, менеджери, стрімери та навіть коментатори, що формує цілу індустрію з величезними прибутками. Багато гравців, які розпочинали з хобі, стали інфлюенсерами, відомими далеко за межами геймерської спільноти.

Геймплей та трансляції на платформах типу Twitch, YouTube Gaming чи Kick стали основою для нового виду дозвілля — стрімінгу. Мільйони людей не тільки грають, але й регулярно дивляться, як грають інші. Це породило величезну аудиторію, яка активно споживає контент, бере участь у чатах, підтримує улюблених стрімерів донатами та субскрипціями. Сучасний стрімер — це водночас гравець, ведучий, комік, технічний спеціаліст і медіаобличчя.

Фандом відеоігор — ще одне підтвердження їхнього культурного впливу. Гравці створюють фан-арт, фанфіки, косплеї, модифікації (моди) до ігор, власні рівні, навіть цілі неофіційні продовження. Наприклад, ігри як Skyrim, Minecraft чи Half-Life мають тисячі фанатських доповнень, які іноді дорівнюють або навіть перевершують за якістю офіційний контент.

Інтеграція відеоігор у культурний простір стала особливо помітною останніми роками. Екранізації за мотивами ігор — The Last of Us, Arcane (за мотивами League of Legends), Cyberpunk: Edgerunners, Super Mario Bros. Movie — отримують визнання як критиків, так і глядачів. Деякі з них навіть здобувають престижні нагороди, що піднімає статус ігор як джерела якісної історії й драматургії. Окремі персонажі, як-от Лара Крофт, Майстер Чіф, Кратос, Маріо чи Джералт з Рівії, стали справжніми культурними іконами, знайомими навіть тим, хто ніколи не грав у відповідні ігри.

Також варто згадати ігрові виставки та конференції, такі як E3 (Electronic Entertainment Expo), Gamescom, GDC (Game Developers Conference), Tokyo Game Show, які стали центральними подіями в ігровій індустрії. Тут оголошуються нові проєкти, демонструються технологічні інновації, відбувається обмін досвідом між розробниками, інвесторами й фанатами. Такі заходи формують пульс індустрії, задають тренди й стають приводом для бурхливих обговорень у соцмережах.

Таким чином, відеоігри стали повноцінним і потужним медіаформатом, що впливає на культуру, економіку, суспільство. Вони формують нові способи взаємодії, розваг, навчання й самореалізації. Гра вже не є втечею від реальності — це новий спосіб її переживати, змінювати й навіть створювати.

1.3. Кіберзмагання як сучасна еволюційна стадія індустрії відеоігор

Змагання у відеоіграх, відомі як кіберспорт (eSports), стали ключовим явищем у розвитку сучасної ігрової індустрії. Від простих турнірів у локальних клубах до арен з десятками тисяч глядачів і мільйонними онлайн-аудиторіями —

кіберспорт за останнє десятиліття набув статусу глобального соціального і комерційного феномену.

Перші організовані змагання у відеоігри відбулися ще у 1980-х роках, коли компанія Atari провела турнір із гри Space Invaders, який зібрав понад 10 000 учасників. Проте справжній розвиток відбувся у 1990-х завдяки стрімкому поширенню персональних комп'ютерів, локальних мереж і перших багатокористувацьких ігор. Такі ігри, як Quake, StarCraft, Counter-Strike і Warcraft III, стали основою перших кіберспортивних сцен. Рисунок 1.7 демонструє кіберспортивний турнір та його масштаби.

Рис. 1.7 Кіберспортивні змагання

З появою Інтернету змагання вийшли за межі локальних клубів. Було створено перші професійні організації, турнірні оператори та трансляційні сервіси. Сьогодні кіберспорт охоплює десятки дисциплін, серед яких найвідоміші — League of Legends, Dota 2, Counter-Strike 2, Valorant, Fortnite, PUBG, Overwatch, Rocket League та інші.

Сучасний кіберспорт побудований за принципом професійної спортивної ліги, у якій ключовими гравцями є змагальні організації з добре налагодженою структурою. Кожна професійна команда має свою кадрову базу – гравців, які

постійно вдосконалюють навички, тренерів та аналітиків, що розробляють стратегії й аналізують ігрові дані, а також PR-фахівців і менеджерів, які відповідають за контрактні питання, комунікацію зі спонсорами та просування в медіа.

Організацією турнірів займаються спеціалізовані оператори на кшталт ESL, BLAST, DreamHack, PGL та самої Riot Games, котрі створюють глобальні серії змагань із розгалуженою сіткою кваліфікацій, плей-оф і фіналів у великих аренах. У процесі підготовки та трансляції вони залучають професійні студії, де коментатори та аналітики проводять попередній розбір матчів, беруть інтерв'ю в гравців і забезпечують глядачам глибоке розуміння подій.

Глядацька аудиторія кіберспорту налічує сотні мільйонів фанатів усього світу, які стежать за матчами онлайн через платформи Twitch, YouTube Gaming та Trovo, а також відвідують живі події в спеціально обладнаних залах. Це створює великий попит на професійну трансляцію, високоякісну режисуру та інтерактивні сервіси, які дозволяють уболівальникам голосувати за MVP, брати участь у конкурсах ігрових прогнозів тощо.

Важливим елементом екосистеми є партнерства з відомими брендами: Red Bull, Intel, BMW, Logitech, Monster, Nike та іншими, які інвестують у команди, спонсорують турніри та використовують кіберспорт як канал для просування своєї продукції серед молодої та високововлеченої аудиторії. Усе це підкріплюється розвиненою інфраструктурою – від спеціалізованих кіберспортивних арен і тренувальних академій до ліцензованих ліг і аналітичних центрів, що забезпечують стабільні джерела доходу для учасників і створюють умови для професійного зростання як гравців, так і організаторів. У результаті кіберспорт функціонує як окрема галузь із власними законами ринку, стандартами підготовки кадрів і механізмами взаємодії з суміжними індустріями.

Кіберспорт вже став важливою частиною світової цифрової економіки. За оцінками Newzoo, дохід від кіберспорту у 2024 році перевищив $1,5 мільярда, а загальна глядацька база сягнула понад 500 мільйонів людей. Офіційні турніри збирають аудиторії, співмірні з великими спортивними подіями: фінали The

International (Dota 2) чи League of Legends Worlds переглядають мільйони глядачів одночасно.

Важливою особливістю є інтеграція кіберспорту в молодіжну культуру. Гравці стають медійними особистостями, інфлюенсерами, зірками нового покоління. Освітні установи в деяких країнах вже впроваджують програми підготовки кіберспортсменів або аналітиків, а самі змагання включають до культурних та спортивних заходів національного рівня.

Кіберспорт висуває до розробників особливі вимоги, адже для організації справді конкурентних змагань необхідне чітке віддзеркалення навичок гравця та балансу між учасниками.

По-перше, у багатьох eSports-проєктах впроваджуються рейтингові системи, які оцінюють вміння гравців і дозволяють формувати матчі між приблизно рівними суперниками. Завдяки цьому кожен поєдинок стає максимально напруженим і справедливим, а гравці отримують зрозумілу мотивацію підвищувати свій рейтинг через перемоги та демонстрацію індивідуальної майстерності.

По-друге, критично важливими є точний таймінг і баланс ігрових механік. Розробники повинні ретельно калібрувати швидкодію зброї, час відновлення навичок, шкоду від атак та інші параметри, аби будь-яка стратегічна помилка чи блискавична рефлекторна реакція відчувались і обговорювались як заслуга гравця, а не як наслідок технічних неточностей. Це вимагає глибокого тестування на різних рівнях гри, а також можливості оперативно виправляти будь-які дисбаланси через патчі чи оновлення.

Щоб зробити змагання максимально глядацькими та аналітичними, сучасні ігри пропонують розвинений набір інструментів для спостереження та аналізу матчів. Коментатори й аналітики мають доступ до спеціальних камер, повторів і телеметрії, що дозволяє відтворювати ключові моменти гри в уповільненому режимі та витягувати статистичні дані про рухи гравців, їх ефективність і прийняті рішення.

Окремим напрямом є підтримка кастомних матчів і турнірного контролю, які дають змогу організаторам швидко налаштовувати правила, обмеження зброї,

карту або інші параметри змагання. Це спрощує створення внутрішніх ліг, офіційних кваліфікаційних раундів чи благодійних турнірів — достатньо одного кліка, щоб підготувати арену відповідно до формату івенту.

У відповідь на ці потреби провідні ігрові рушії, такі як Unity та Unreal Engine, інтегрують спеціалізовані SDK та сервіси. Вони надають готові модулі для організації мережевих сесій із низькою затримкою, хостинг-рішення, системи автентифікації гравців та лідерборди. Завдяки цьому команди розробників можуть зосередитися на балансі й геймдизайні, не витрачаючи ресурси на побудову інфраструктури «з нуля». Такий підхід дозволяє створювати мультиплеєрні та eSports-орієнтовані проєкти швидше та з мінімальними технічними бар’єрами.

Очікується, що розвиток віртуальної реальності, доповненої реальності та хмарного геймінгу відкриє нові горизонти для кіберспорту. Уже зараз тестуються турніри у VR, а мобільний кіберспорт активно розвивається в Азії та Південній Америці.

Не менш важливою є поява національних збірних, участь кіберспорту в спортивних змаганнях (як-от Азійські ігри або потенційно Олімпіада) та офіційне визнання його як виду спорту в різних країнах.

1.4. Інструментарій для створення ігор

Розробка сучасних відеоігор потребує застосування широкого спектра інструментів, які забезпечують проєктування, програмування, створення графіки, анімації, озвучення, тестування й фінальну публікацію продукту. Завдяки стрімкому розвитку індустрії, сьогодні існує чимало програмних рішень, які адаптовані під потреби інді-розробників, середніх студій і великих AAA-компаній.

Інструментарій для створення ігор можна умовно поділити на такі основні категорії:

1. Ігрові рушії (game engines) – центральний компонент розробки, який об'єднує рендеринг, фізику, логіку, звуки та інші системи.

2. Графічні редактори – використовуються для створення 2D- і 3D-елементів.

3. Середовища розробки (IDE) – необхідні для написання коду.

4. Системи контролю версій – допомагають керувати змінами у проєкті.

5. Інструменти для створення анімацій та UI.

6. Системи аналітики та тестування.

Успішна розробка Top-Down Shooter вимагає не лише вмілого геймдизайну, а й правильного поєднання рушіїв, графічних та аудіоінструментів, середовищ програмування й сервісів для командної роботи й аналітики.

Однією з найпопулярніших платформ для створення 2D-проєктів є Unity. Він пропонує потужний візуальний редактор, у якому розробник може з легкістю розмістити спрайти, налаштувати фізичні властивості об’єктів через компоненти Rigidbody2D та Collider2D, а також реалізувати систему анімацій Mecanim для плавних переходів між станами персонажа. Asset Store дає змогу швидко інтегрувати готові пакети—від ефектів частинок до готових шаблонів меню—що значно пришвидшує роботу над прототипом. Для великих фотореалістичних або 3D-орієнтованих проєктів часто вибирають Unreal Engine, який із системою Blueprints дозволяє створювати складні механіки без глибокої роботи з кодом, а також забезпечує доступ до рушія PhysX і просунутих рендерингових технологій, таких як Lumen і Nanite. Натомість Godot приваблює легкістю освоєння та відкритим вихідним кодом: його GDScript дає змогу швидко писати логіку, а вбудовані інструменти для 2D та 3D допомагають не лише малим інді-студіям, а й початківцям–самоукам.

Графічна складова будь-якої гри формується в спеціалізованих редакторах. Для створення та обробки 2D-спрайтів найчастіше використовують Adobe Photoshop або безкоштовний аналог GIMP, які дають можливість малювати текстури, експортувати атласи спрайтів і обробляти їх кривими кольору. Aseprite — це інструмент, заточений саме під піксель-арт та покадрову анімацію, де кожен кадр легко коригувати за допомогою шарів і тайлмапів. Якщо ж гра вимагає тривимірних об’єктів, Blender пропонує повний цикл: від моделювання й UV-

розгортки до анімації та рендерингу. Для створення складних 2D-анімацій із кістяковою системою використовують Spine або DragonBones, де ключові точки анімації прив’язуються до “кісток” персонажа, що дозволяє створювати плавні цикли руху без зайвого малювання кожного кадру. На рисунку 1.8 зображено графічний редактор Aseprite.

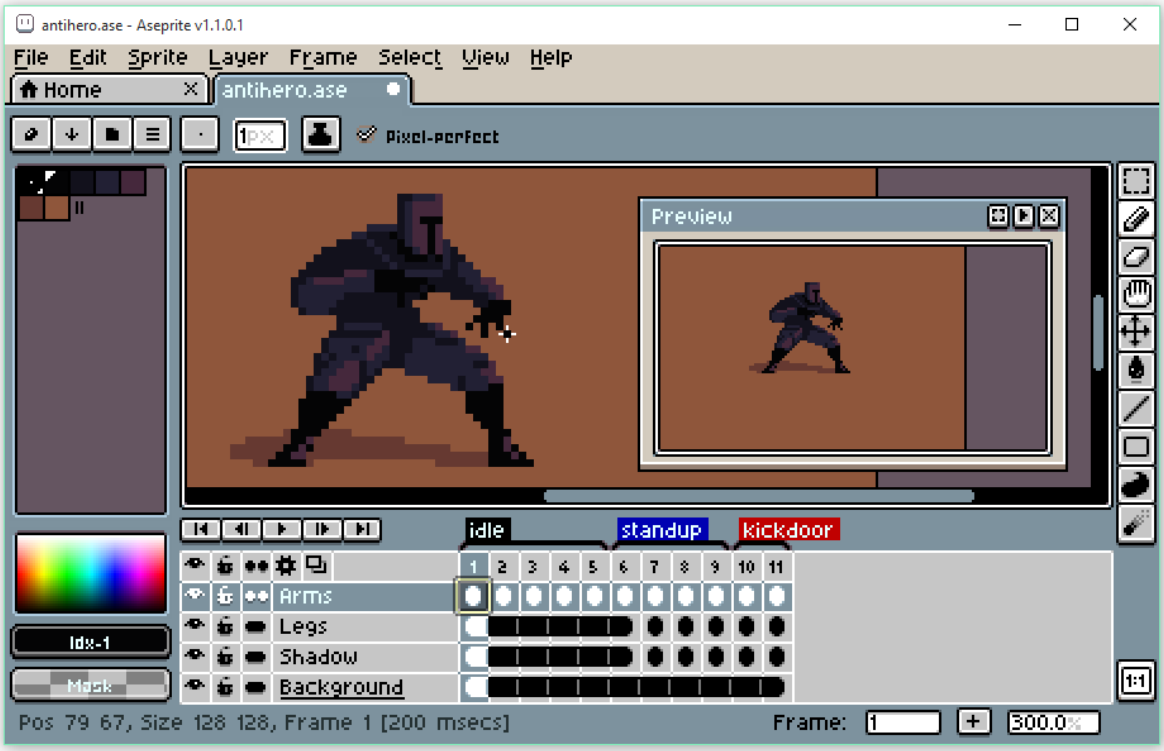


Рис. 1.8 Графічний редактор Aseprite

Кодування та налагодження відбувається в середовищах Visual Studio або JetBrains Rider, що тісно інтегруються з Unity і дозволяють відлагоджувати C#-скрипти в реальному часі, переглядати змінні й виклики методів. Легкий текстовий редактор VS Code також може слугувати для швидких правок конфігурацій, шейдерів чи JSON-файлів з параметрами рівнів. У Godot найчастіше працюють безпосередньо в його вбудованому редакторі з підтримкою GDScript і C#, а Unreal Engine пропонує Visual Studio для C++ та власний Blueprint Editor.

Під час командної роботи або навіть при власній розробці важливо відстежувати історію змін, тому застосовують Git з хостингом репозиторіїв на

GitHub, GitLab чи Bitbucket. Для великих Unity-проєктів популярний також Plastic SCM, який краще справляється з великими бінарними файлами та дозволяє ставити «замки» на сцени.

Управління завданнями та документацією зазвичай ведуть у Trello, Notion або Jira, де створюють дошки із беклогом, спринтами та статусами виконання. А для реалізації онлайн-функцій—лідербордів, аутентифікації, зберігання прогресу—розробники звертаються до PlayFab або Firebase, а для відстеження поведінки гравців і аналітики найзручнішими виявляються Unity Analytics і GameAnalytics, які збирають метрики запусків рівнів, часу сесій і точності стрільби, що допомагає оптимізувати баланс і покращувати UX.

Таким чином, грамотне поєднання рушіїв, редакторів і сервісів створює повний технологічний стек, необхідний для якісної, стабільної та масштабованої розробки Top-Down Shooter.

1.5. Можливості рушія Unity у створенні інтерактивних ігор

На сучасному етапі розвитку ігрової індустрії рушій Unity займає провідне місце серед інструментів для створення інтерактивних ігор. Його популярність пояснюється гнучкістю, кросплатформеністю, великою кількістю вбудованих функцій і відкритістю до інтеграції сторонніх рішень. Unity активно використовується як інді-розробниками, так і великими студіями для створення 2D та 3D ігор, симуляторів, AR/VR-додатків і навіть інтерактивного навчального ПЗ.

Однією з головних переваг Unity є підтримка великої кількості платформ: розробник має змогу створювати одну гру й експортувати її на Windows, macOS, Linux, Android, iOS, WebGL, а також на консолі (PlayStation, Xbox, Nintendo Switch) і VR/AR-шоломи. Ця особливість значно спрощує процес мультиплатформенного розвитку без необхідності суттєво змінювати код проєкту.

Редактор Unity має інтуїтивно зрозумілий інтерфейс і забезпечує візуальне середовище для створення сцен, управління об’єктами, налаштування матеріалів, анімацій і фізики. Компонентна архітектура рушія дозволяє швидко додавати до

об'єктів у сцені нову функціональність, наприклад скрипти, колайдери або фізичні властивості. Цей підхід є гнучким і масштабованим, що особливо важливо під час роботи в невеликих командах або при швидкому прототипуванні.

Скриптування у Unity здійснюється за допомогою мови C#. Вона поєднує простоту синтаксису з широкими можливостями, що дозволяє ефективно реалізовувати геймплейну логіку, штучний інтелект, UI та інші підсистеми. Крім того, Unity підтримує подієву модель програмування, яка ідеально підходить для розробки інтерактивних застосунків.

Система анімацій Animator (Mecanim) дозволяє створювати складні анімаційні стани, зв’язки між ними, використання blend trees, інверсну кінематику та контроль анімацій у реальному часі. Для Top-Down Shooter цей функціонал є надзвичайно корисним: наприклад, можна реалізувати плавні переходи між станами персонажа — біг, стрільба, перезарядка тощо.

Unity також забезпечує широкі можливості для створення 2D-ігор. За допомогою Sprite Renderer, Tilemap, Rigidbody2D та інших інструментів можна створювати ефективні 2D-сцени з фізикою та взаємодією. Система UI заснована на Canvas і підтримує адаптивність для різних роздільностей екранів. Створення HUD-елементів, таких як індикатори здоров’я, патрони або меню — є інтуїтивно зрозумілим і гнучким.

Фізичні рушії Unity включають підтримку як 2D (Box2D), так і 3D (PhysX) симуляцій. Це дозволяє реалізовувати зіткнення, гравітацію, сили, тригери, а також поведінку об’єктів у реальному часі. Ці можливості легко інтегруються з геймплейними елементами, такими як кулі, вороги чи пастки.

Unity надає інструменти для інтеграції з онлайн-сервісами: аналітика, реклама, хмарне збереження, база даних, мультиплеєр через Photon або Mirror. Також доступна інтеграція з Unity Services, які дозволяють швидко реалізовувати backend-функціонал без потреби у створенні серверної частини з нуля.

Asset Store — це ще одна сильна сторона Unity. Він дозволяє завантажити як безкоштовні, так і платні ресурси: моделі, звуки, анімації, UI-набори, фреймворки та навіть повноцінні прототипи. Це пришвидшує розробку та знижує поріг входу.

У Unity використовується система пакетів (Package Manager), яка дозволяє підключати необхідні модулі — від URP (Universal Render Pipeline) до інструментів для роботи зі штучним інтелектом або редакторськими розширеннями. Завдяки цьому можна адаптувати рушій до потреб конкретного проєкту без навантаження зайвими модулями.

Важливою перевагою Unity є швидкий цикл розробки: редагування — запуск — тестування. Це дозволяє оперативно вносити зміни, перевіряти нові механіки й адаптувати ідеї до потреб гравця. З цієї причини Unity ідеально підходить для створення прототипів і експериментів із геймдизайном.

У контексті створення гри жанру Top-Down Shooter Unity забезпечує всі необхідні інструменти. Камера з видом зверху реалізується простим налаштуванням ортографічної камери. Управління персонажем здійснюється через Rigidbody2D або CharacterController з підтримкою Input System. Механіка стрільби реалізується через Bullet-префаби або Raycast з обробкою зіткнень. AI ворогів будується на FSM-моделі або через патерни навігації з використанням тригерів і станів. Інтерфейс гри включає всі базові елементи: здоров’я, боєприпаси, меню, підказки. Звуковий супровід реалізовується за допомогою аудіо-кліпів, які активуються при певних подіях. Системи прогресії реалізуються через PlayerPrefs або зовнішні сервіси.

Unity також легко розширюється сторонніми плагінами, наприклад, Cinemachine (динамічна камера), TextMeshPro (текстова система), DOTween (анімації), Mirror або Photon (мультиплеєр), NavMesh або A* Pathfinding Project (навігація ворогів).

Якщо порівнювати Unity з Unreal Engine, то кожен з них має свої переваги. Unreal Engine, зокрема, відомий високоякісною графікою, що досягається завдяки рушію Lumen, Nanite та іншим сучасним рендерами. Він більше підходить для створення AAA-проєктів, шутерів від першої особи або ігор із фотореалістичною графікою. Unreal використовує мову C++ та Blueprints — візуальну систему програмування, яка дозволяє створювати ігрову логіку без глибоких знань коду. Проте це також робить Unreal складнішим у вивченні й вимагає більше ресурсів на

підтримку проєкту. Для 2D-ігор або невеликих проєктів Unreal є менш зручним, ніж Unity, через брак адаптованих інструментів і складність розгортання. Рисунок 1.9 ілюструє рушій Unreal Engine.

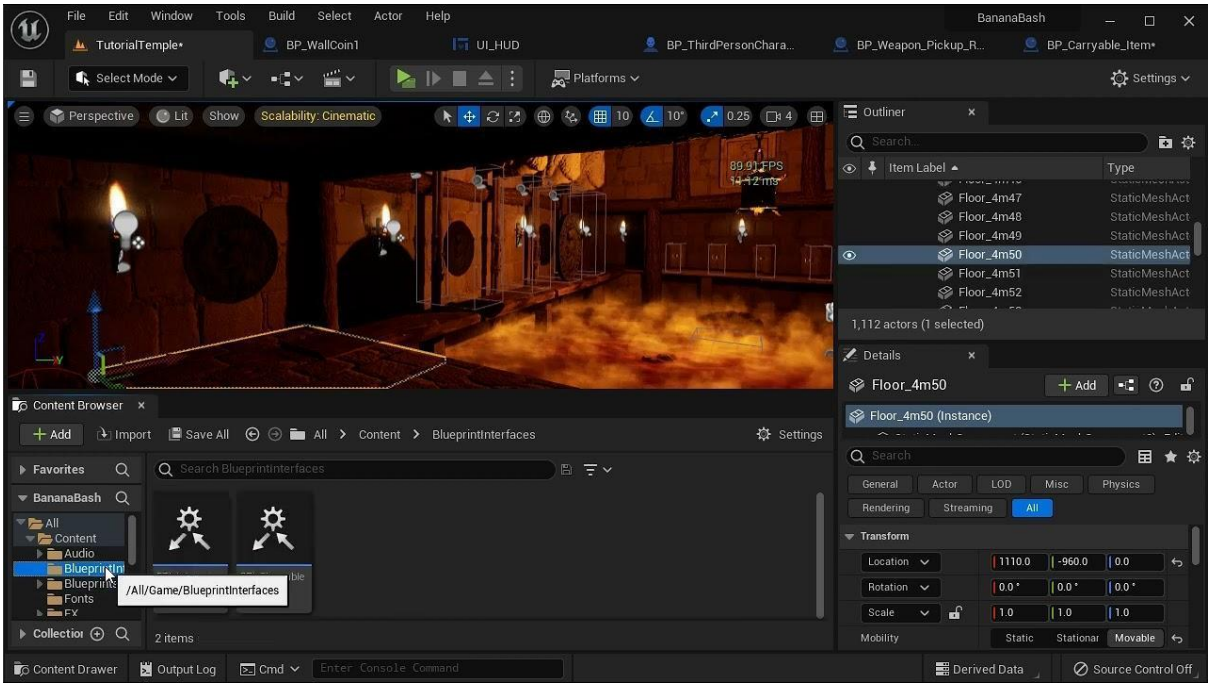


Рис. 1.9 Інтерфейс рушія Unreal Engine 5

Unity, натомість, оптимальніший для інді-розробників, мобільних ігор, AR/VR-проєктів, 2D-ігор і проєктів середнього масштабу. Його менші вимоги до ресурсів, простота освоєння, багата документація й активна спільнота роблять його ідеальним вибором для широкого спектра ігор, зокрема Top-Down Shooter.

1.6. Аналіз існуючих ігор жанру Top-Down Shooter

Hotline Miami. Це ультра-динамічний інді-шутер із піксель-артом і яскравими неоновими кольорами, в якому кожне влучання означає миттєву смерть — і для гравця, і для ворога. Гра спонукає до вкрай швидких рефлексів і вимагає досконального знання мап, бо навіть одна помилка може завершити раунд. Сюжет побудований на психологічних алюзіях із нарративними епізодами між рівнями, що створюють відчуття сюрреалізму та напруги. Саундтрек у стилі

synthwave синхронізується з темпом бою, додаючи грі відчуття пульсуючого хаосу. Незважаючи на мінімалістичний HUD, інтерфейс підсилює занурення, не відволікаючи від критичного моменту. На рисунку 1.14 показано гру Hotline Miami.

Рис. 1.10 Відеогра Hotline Miami

Недоліки: незважаючи на блискавичний темп, різка складність окремих рівнів може відлякати гравців: одиничне влучання — миттєва смерть штовхає до багатьох поспіль спроб без поступового навчання. Велика кількість механік (від зміни зброї до хитромудрих схем охорони) іноді перевантажує швидкий геймплей, що знижує задоволення від напруги.

Переваги: кожен рівень просякнутий стильним піксель-артом і чіткими контурами ворогів, а динамічна синхронізація з ретро-саундтреком загострює відчуття хаосу бою. Інтерфейс мінімалістичний — лише таймер і назви масок, що не відволікають. Розхідників, як аптечок чи боєприпасів, тут немає — усе спрощено до голого влучання та ухилення, що додає горор-відтінку безжального «один ворог — один крок до кінця». Фізика куль спрощена (постріл миттєвий, без куленепробивних матеріалів), а AI ворогів обмежується простими патрульними маршрутами.

Enter the Gungeon. Цей rogue-lite Top-Down Shooter вирізняється барвистим піксель-артом, величезним арсеналом зброї та гумором у кожній пастці й капості. Космічна станція “Gungeon” процедурно генерує кімнати, тому кожна спроба відрізняється розташуванням ворогів, стін і секретних засідок. Унікальна фізика куль дозволяє їм рикошетити від стін, створюючи веселий і непередбачуваний хаос у бою. Запаси боєприпасів, аптечки й заряд ліхтаря додають тактичного планування — особливо в темних коридорах із горор-атмосферою. Завдяки допоміжним предметам і перкам гравець може адаптуватися до будь-якого стилю гри й шукати найоптимальніші комбінації озброєння. Рисунок 1.11 демонструє вигляд гри Enter the Gungeon під час проходження.

Рис. 1.11 Відеогра Enter the Gungeon

Недоліки: процедурно згенеровані кімнати та випадкові ключі іноді створюють непрохідні конфігурації, роблячи деякі сесії занадто складними навіть для досвідчених гравців. Велика кількість різноманітної зброї та механік «перекочування» може заплутати новачків, що знижує початкову привабливість.

Переваги: гумористичний стиль, барвисті піскельні спрайти та оригінальні боси роблять кожну гру унікальною. Інтерфейс із великими іконками зброї, лічильниками здоров’я та гранат легко читається, а запас розхідників (банджі, аптечки) й заряд ліхтаря (у темних кімнатах) додають тактичного шару. Фізика куль у Gungeon досить аркадна — кулі рикошетять від стін та перешкод, що породжує непередбачувані ситуації. AI ворогів використовує прості схемні шаблони атаки, але в сукупності з випадковим розташуванням приміщень створює гарні виклики.

Alien Shooter. Класичний ізометричний шутер початку 2000-х, де гравець протистоїть натовпам інопланетян у закритих арених рівнях. Графіка в стилі раннього 3D разом із масивними хвилями ворогів створює невпинний екшен із постійним напором. Фізика куль проста й прямолінійна, але велика кількість розхідників (патрони та аптечки) дозволяє випробувати різні класи зброї та тактики. Інтерфейс містить докладну статистику, однак місцями може відволікати від основної дії. Горор-елементи проявляються в напрузі, що виникає, коли поруч з’являються орди вибухових «пушокатарів» або швидких мутантів. На рисунку 1.12 зображено фрагмент ігрового процесу Alien Shooter.

Рис. 1.12 Відеогра Alien Shooter

Недоліки: Ізометрична камера іноді ускладнює оцінку відстані до ворогів, а проста фізика куль (кулі не мають інерції, одразу вилітають прямолінійно) знижує глибину стрільби. Інтерфейс перевантажений панелями із статистикою, що відволікає від дії. Горор-елементи мінімальні — переважно масивні натовпи інопланетян без атмосферних спецефектів. Іноді відсутня чітка індикація отриманого урону, що ускладнює контроль за здоров'ям гравця. Через високий темп подій новачкам важко адаптуватися до механік і керування.

Переваги: Гра вражає масштабністю зіткнень: орди ворогів створюють справжній екшн-хаос. Розхідники (патрони, аптечки) розкидані щедро, що дозволяє експериментувати з різними видами зброї. Флешлайт у грі відсутній, однак підсвічування мінімальних темних коридорів додає напруженості. AI ворогів — простіші скрипти переслідування, але в поєднанні з великими групами створюють відчуття постійного тиску. Система модифікацій дозволяє кастомізувати стиль гри під себе, підвищуючи варіативність проходження. Гнучке керування та швидкий респаун після смерті сприяють збереженню темпу ігрового процесу.

Nuclear Throne. Постапокаліптичний rogue-like у піксель-арті, де гравець керує мутованими персонажами в пошуках легендарного «Трону». Кожна смерть повертає героя на початок із частковим збереженням накопичених перків, що створює глибоку систему прогресії. Кожен персонаж має унікальні здібності, що кардинально змінюють стиль гри. Вороги, зброя й рівні генеруються випадково, що забезпечує непередбачуваність і високу реіграбельність. Динамічний геймплей і хардкорна складність роблять Nuclear Throne викликом навіть для досвідчених гравців. Графічний стиль гри, хоч і мінімалістичний, влучно передає атмосферу хаосу та руйнування постапокаліптичного світу. Саундтрек і звукові ефекти підсилюють відчуття напруги та драйву під час бою. Завдяки своїй глибокій механіці та постійним викликам, Nuclear Throne здобула культовий статус серед фанатів інді-ігор. На рисунку 1.13 можна подивитись гру Nuclear Throne.

Рис. 1.13 Відеогра Nuclear Throne

Хардкорні перестрілки з процедурно згенерованим арсеналом змушують постійно адаптуватися, а кожен елемент рівня може слугувати як прикриття чи засідка. Деяким гравцям Nuclear Throne може здатися надто повторюваною через постійні смерті й необхідність проходити все заново. Відсутність чіткого навчання на початку ускладнює вхід для новачків, а хаотичність боїв іноді зводить нанівець стратегічне планування. Баланс між персонажами не завжди ідеальний — деякі мутанти значно ефективніші за інших, що може обмежувати варіативність стилів гри. На рисунку 1.14 ми бачимо гру та її ігровий процес Nuclear Throne.

Рис. 1.14 Ігровий процес Nuclear Throne

Недоліки: висока складність і хардкорний rogue-like аспект призводять до частих смертей, що може відлякати менш досвідчених гравців. Велика кількість зброї та перків створює круту криву навчання, вимагаючи часу на адаптацію.

Переваги: автентичний піксель-арт і динамічний саундтрек формують унікальну атмосферу постапокаліпсису. Лаконічний інтерфейс — шкала здоров'я та патрони займають мінімум простору, залишаючи екран вільним для бою. Горор-елементи проявляються в агресивній поведінці мутантів та напруженому звуковому оформленні. Хоча фізика куль доволі проста, вона задовольняє вимоги хардкорної аудиторії. Штучний інтелект ворогів базується на простих патрульних і агресивних алгоритмах, що створює напружений хаос у вузьких коридорах.

Таблиця 1.1

Порівняння характеристик аналогів та розроблюваної гри

Критерій / Гра	Nuclear Throne	Enter the Gungeon	Alien Shooter	Hotline Miami	Blood and Bullets
Фізика кулі	+	+	-	-	+
Відображення стану гравця (здоров'я)	-	+	-	-	+
Горор елементи	-	-	+	-	+
Наявність розхідників	+	-	+	-	+
Наявність ліхтаря	-	-	-	-	+
Кілька типів ворогів	-	+	-	+	+

1.7. Формулювання цілей дослідження та постановка завдань

У межах даної кваліфікаційної роботи ставиться завдання здійснити всебічне дослідження жанру Top-Down Shooter із подальшим практичним застосуванням

отриманих знань для створення власного прототипу. Це передбачає не лише аналіз історії та розвитку жанру, його ключових ігрових механік і візуальних особливостей, а й вивчення сучасних тенденцій ігрової індустрії, що впливають на формування попиту та технологічні підходи до розробки. Особливу увагу буде приділено оцінці інструментів створення ігор, зокрема можливостям рушія Unity, і порівнянню з іншими проєктами жанру, щоб виокремити найбільш вдалі рішення. Завершальною фазою стане розробка архітектури програмного продукту, імплементація базових механік і користувацького інтерфейсу, а також всебічне тестування для підтвердження технічної стійкості та ігрової привабливості прототипу.

У процесі реалізації прототипу буде використано модульний підхід до побудови коду, що дозволить досягти високої гнучкості та масштабованості. Основні компоненти гри — рух персонажа, система стрільби, поведінка ворогів, інтерфейс користувача — будуть реалізовані як окремі підсистеми, що взаємодіють між собою через чітко визначені інтерфейси. Особливу увагу буде приділено оптимізації продуктивності, зокрема зменшенню навантаження на процесор під час великої кількості одночасних об’єктів на екрані. У межах візуального оформлення прототипу планується використати мінімалістичну, але виразну стилістику, що акцентує увагу на геймплейній динаміці. Для цього будуть застосовані стандартні інструменти Unity — система анімацій, частинки, освітлення та постобробка. З метою підвищення глибини ігрового досвіду реалізується система поступового ускладнення рівнів і адаптивної поведінки ворогів. На етапі тестування проводитиметься аналіз юзабіліті, виявлення помилок і збір зворотного зв’язку від тестувальників. Для досягнення мети дослідження необхідно виконати низку послідовних завдань:

1. Провести аналітичний огляд жанру Top-Down Shooter, включаючи його історію, етапи розвитку, характерні ігрові механіки та візуальні особливості.
2. Дослідити сучасний стан ігрової індустрії, її вплив на культуру та економіку, а також тенденції розвитку, включаючи кіберспорт, інді-розробку та мобільний сегмент.

3. Оцінити існуючі інструменти для створення ігор, порівнявши їх функціональність, доступність, зручність для початківців і підтримку ком'юніті.

4. Вивчити можливості рушія Unity як найпопулярнішого середовища для 2D-розробки, включаючи управління анімацією, фізикою, UI, системою частинок та імплементацією коду.

5. Провести порівняльний аналіз відомих ігор жанру Top-Down Shooter, виокремивши ефективні рішення, які можуть бути використані в розробці власного прототипу.

6. Розробити архітектуру проєкту, включаючи структуру сцен, об'єктів, скриптів, а також взаємозв’язки між ними.

7. Реалізувати прототип гри з базовими механіками: управлінням гравцем, стрільбою, рухом ворогів, системою зіткнень, візуальними ефектами.

8. Створити користувацький інтерфейс, який відображає необхідну інформацію: здоров’я, боєзапас, очки або прогрес.

9. Здійснити тестування реалізованого функціоналу, виявити й усунути технічні недоліки.

Очікувані результати

У результаті виконання кваліфікаційної роботи має бути створено:

- Робочий прототип 2D-гри у жанрі Top-Down Shooter;
- Документовану технічну та дизайнерську базу гри;
- Аналіз аналогів і обґрунтований вибір технологій;
- Запропоновані напрями для подальшої розробки продукту.

2. РОЗРОБКА СТРУКТУРНОЇ СКЛАДОВОЇ ГРИ

2.1. Цілі розробки програмного продукту

На сучасному етапі розвитку індустрії відеоігор ринок демонструє стрімке зростання: за підсумками 2023 року його обсяг сягнув 385 млрд доларів, що перевищує за доходами як кіно-, так і спортивну індустрію. Це свідчить про те, що ігри займають незамінне місце в житті мільйонів гравців, які шукають усе більш динамічні й видовищні проєкти. Водночас така популярність породжує серйозні виклики для розробників: ринок перенасичений, створення нових механік ускладнюється, а генерувати успішні ідеї стає дедалі складніше. Незважаючи на це, аудиторія не втрачає інтересу й очікує від ігор не лише вражаючої графіки, а й збалансованого геймплею, зрозумілого інтерфейсу та оптимальної тривалості сесій.

Моя кваліфікаційна робота присвячена саме жанру Top-Down Shooter. Мета роботи – розробити прототип гри в жанрі Top-Down Shooter із фокусом на оптимізації ключових механік бою для підвищення динаміки й користувацького задоволення.

2.2. Структурна модель гри та побудова архітектури програмного забезпечення

Розробка будь-якої системи чи гри спрямована на задоволення потреб користувачів і допомогу їм у виконанні ключових завдань. Звичайно, можна зібрати перелік усіх можливих побажань із форумів та оглядів — але це рідко призводить до успіху, адже без чіткої структури вимог процес «ідеального» продукту перетворюється на хаотичний перелік забаганок. Саме тому на етапі аналізу та проєктування важливо сформувати обмежений набір ключових функціональних вимог і підібрати технології, які дозволять реалізувати їх максимально ефективно.

Метод Use Case (варіанти використання) допомагає описати, як майбутня гра жанру Top-Down Shooter реагуватиме на дії гравця, і зрозуміти, які сценарії є основними для користувача.

У контексті нашої гри це означає:

- чітко визначити всі ключові дії (наприклад, “Рух персонажа”, “Вистріл”, “Вибір зброї” тощо);
- встановити зв’язки між гравцем (актором) та системою гри;
- простежити, як кожна дія користувача перетворюється на запит до внутрішніх компонентів двигуна Unity, AI ворогів чи інтерфейсу.

Use Case-діаграма складається з:

- акторів, тобто ролей, які взаємодіють із грою (насамперед Гравець, а можливо — Система збереження або Система лідерборду);
- варіантів використання (овали), кожен із яких описує певну корисну функцію: від базових “Стріляти” та “Рухатися” до “Перезарядження” чи “Виклику меню паузи”;
- зв’язків (стрілок), що показують, який актор викликає кожен Use Case.

Ретельно спроєктована Use Case-діаграма дозволяє:

1. Визначити та зафіксувати ключові функціональні вимоги проєкту.
2. Спростити керування процесом розробки, розділивши її на логічні підсистеми.
3. Зрозуміти реальні потреби гравців і переконатися, що кожна важлива дія відображена в архітектурі.

Забезпечити основу для подальшого тестування — кожен Use Case може стати окремим тестовим сценарієм.

Таким чином, Use Case-діаграма є невід’ємним елементом аналізу та проєктування нашої Top-Down Shooter-гри: вона гарантує, що розробка зосередиться на найважливішому і надасть користувачеві стабільний і продуманий ігровий досвід. На рисунку 2.1 ми бачимо Use Case діаграму

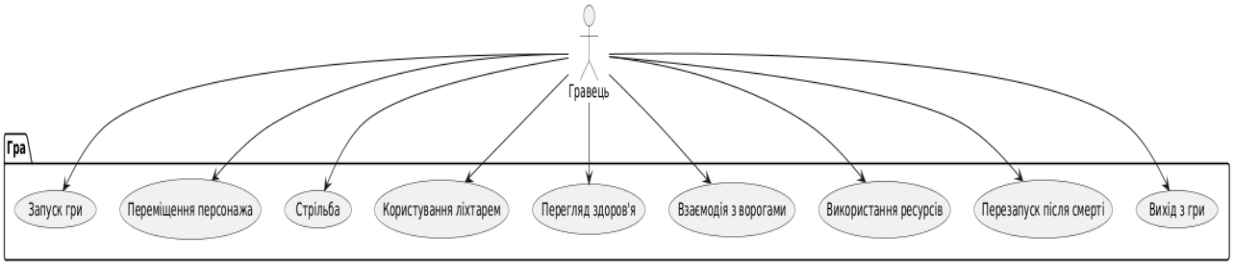


Рис. 2.1 Use Case Діаграма

Таким чином, Use Case-діаграма є невід’ємним елементом аналізу та проєктування нашої Top-Down Shooter-гри: вона гарантує, що розробка зосередиться на найважливішому і надасть користувачеві стабільний і продуманий ігровий досвід.

Діаграма класів у UML зображує структуру програми як набір взаємопов’язаних класів, кожен із яких позначений прямокутником із трьома секціями: ім’я класу, перелік його атрибутів та набір методів. Зв’язки між класами позначаються різними типами стрілок і вказують, чи один клас використовує інший (асоціація), чи містить його екземпляри у собі (агрегація чи композиція).

У нашій діаграмі виділено ключові класи, що відповідають за загальне керування грою, обробку вводу, роботу зі зброєю, керування життям об’єктів, генерацію ворогів та відображення інтерфейсу. Кожен із них має чітко визначені атрибути (наприклад, параметри здоров’я чи швидкість руху) та методи (наприклад, ініціалізація, оновлення стану або обробка подій).

Зв’язки між цими класами відображають механізми взаємодії: асоціації показують, які класи звертаються один до одного, а композиції означають, що один клас містить екземпляри іншого й не може функціонувати без них. Така візуалізація допомагає побачити, як різні компоненти проєкту організовані усередині загальної архітектури, і визначити, які з них є незалежними чи тісно пов’язаними. Діаграма також дозволяє швидко ідентифікувати потенційні точки розширення, де можна безпечно додавати новий функціонал без порушення вже наявної структури. Вона демонструє, які об'єкти обмінюються даними, які методи викликаються найчастіше, і де можливі вузькі місця в логіці взаємодії. Особливо корисно це при

роботі в команді, коли кілька розробників взаємодіють із однією кодовою базою — UML-діаграма забезпечує спільне розуміння архітектури. Також вона дає змогу перевірити дотримання принципів ООП, таких як інкапсуляція, спадкування й поліморфізм. Якщо структура побудована правильно, зміни в одному класі матимуть мінімальний вплив на інші, що полегшує супровід і модифікацію проєкту. На рисунку 2.2 можемо бачити діаграму класів гри.

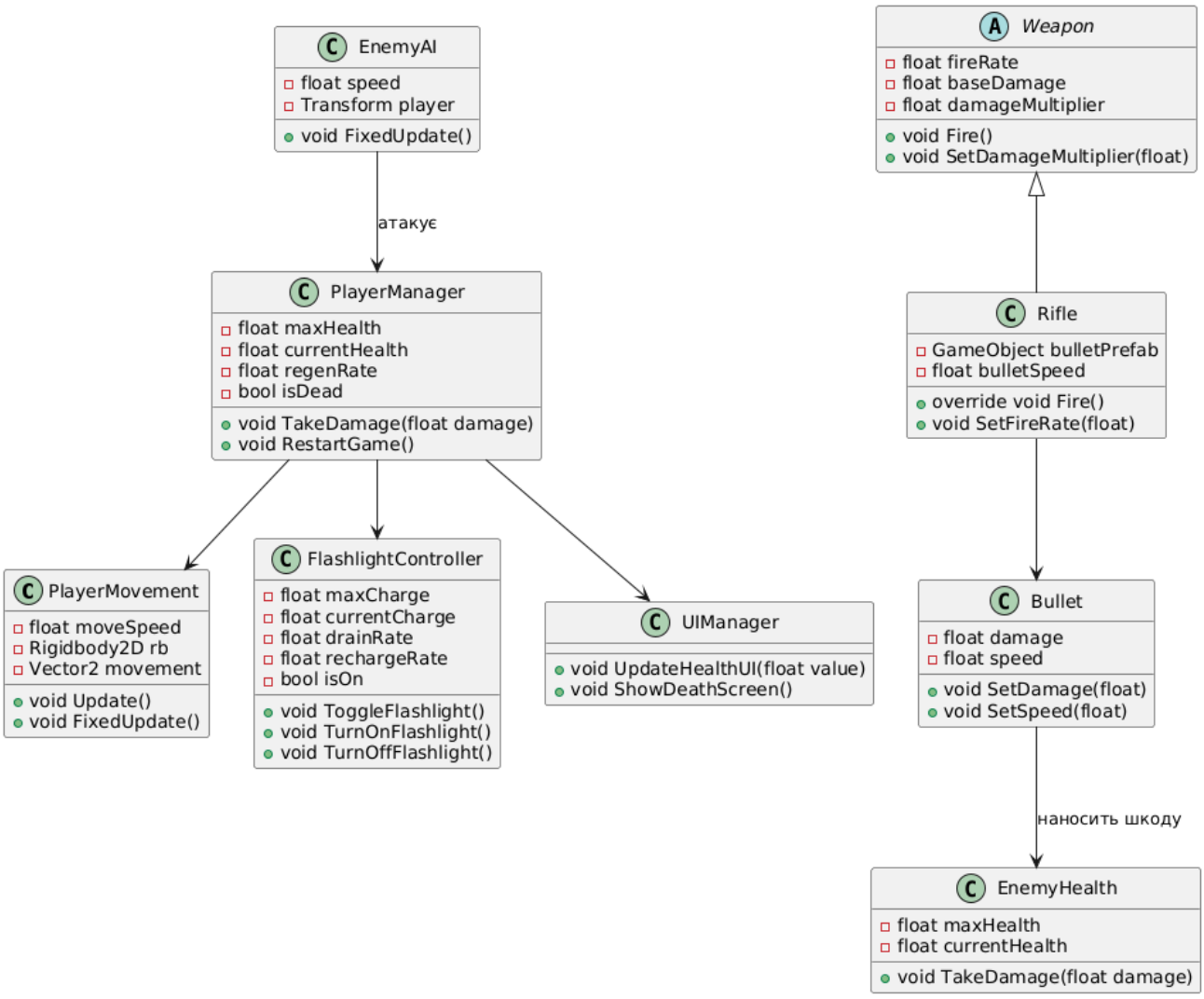


Рис 2.2 Діаграма класів

Використання UML-діаграми класів значно спрощує аналіз, підтримку та подальше розширення системи, оскільки дозволяє миттєво оцінити структуру проекту й знайти місця для модернізацій чи оптимізацій без необхідності заглиблюватися в деталі реалізації.

У нашій грі всі ключові компоненти згруповані в окремі підсистеми, кожна з яких представлена власним класом із чітко розмежованою відповідальністю. Відповідальність за збереження стану персонажа (здоров’я, боєзапас, активні перки) лежить на PlayerManager, тоді як безпосередня обробка вводу та переміщення героя винесена в клас PlayerMovement, що читає натискання клавіш та перетворює їх на коректні вектори руху в ігровому просторі. Окремо виділено FlashlightController, який керує зарядом і промінням ліхтаря — його класифікація як окремого компоненту дає змогу масштабувати механіку освітлення, додаючи елементи виживання.

У протилежному таборі стоїть EnemyAI, що забезпечує поведінку ворогів: пошук гравця, патрулювання, атаки та ухиляння. Кожен ворог володіє власною системою EnemyHealth, яка відслідковує шкоду, опрацьовує ефекти влучання та ініціює анімації чи події загибелі. Бойовий процес організовано через класи Weapon та його конкретну реалізацію Rifle: загальні методи перезарядки, дозвіл на стрільбу та облік боєприпасів містяться в абстрактному Weapon, а Rifle додає специфічні параметри темпу вогню та розльоту куль. Згенеровані об’єкти Bullet рухаються зі швидкістю й траєкторією, заданими класом Weapon, та при колізії викликають EnemyHealth.TakeDamage().

Нарешті, весь зв’язок із гравцем через меню, індикатори стану й повідомлення забезпечує UIManager. Цей клас оновлює смугу здоров’я, лічильник патронів, індикатори заряду ліхтаря й час до наступної хвилі, керуючи видимістю панелей і паузою, коли це необхідно. Така розбивка на класи дозволяє чітко відокремити механіку пересування від логіки керування персонажем, штучний інтелект від обробки шкоди, а також узагальнити зброю й боєприпаси, спростивши майбутні розширення та тестування кожного компонента.

Для реалізації архітектури було використано підхід ECS-lite (Entity-Component-System), де кожен об'єкт — це Entity, що складається з Component'ів, які реалізують певну поведінку (MoveComponent, HealthComponent, ShootComponent тощо).

Це дозволяє:

- Уникнути дублювання коду.
- Гнучко змінювати логіку без змін у базовому класі.
- Швидко додавати нові фічі.

Основні класи:

- PlayerManager, PlayerMovement — обробка стрільби, руху.
- EnemyAI— логіка руху ворогів, переслідування гравця.
- Bullet — поведінка снарядів.
- HealthSystem — втрата здоров’я, смерть.
- UIManager — керування елементами інтерфейсу.
- GameManager — загальний контроль станів гри.

Кожен компонент тестувався окремо. В Unity використовувалися Gizmos для візуального представлення зон атаки, пересування тощо. Також реалізовані debug-логи для ключових подій (втрата HP, смерть ворога). Тестування та дебаг у процесі розробки гри є критично важливими етапами, які дозволяють переконатися, що всі механіки працюють стабільно, логіка не містить помилок, а гравець отримує очікуваний ігровий досвід. У Unity цей процес поєднує як візуальні, так і кодові інструменти, що робить діагностику проблем гнучкою й ефективною.

Перш за все, слід використовувати вбудований інструмент Console, який показує повідомлення, попередження та помилки, що виникають під час виконання гри. За допомогою функцій Debug.Log(), Debug.Warning() або Debug.Error() розробник може вручну виводити потрібну інформацію під час виконання скриптів: значення змінних, повідомлення про вхід у певні методи, результат обчислень або інформацію про зіткнення.

Для більш глибокого аналізу варто активно застосовувати режим гри в редакторі (Play Mode). Це дозволяє бачити поведінку об'єктів у реальному часі: перевірити рух персонажа, точність стрільби, відгук інтерфейсу, появу ворогів. У поєднанні з вкладкою Scene, можна стежити за невидимими компонентами (наприклад, колайдерами, радіусами зон, напрямком руху), переміщуючи камеру вручну, зупиняючи або сповільнюючи час виконання.

Також в Unity надзвичайно корисна функція "Breakpoints" та Step-by-step виконання через Visual Studio, яка дозволяє поставити зупинку в коді, де саме розробник хоче перевірити стан об’єкта або значення змінних. Це особливо важливо для відлову складних логічних помилок або для налагодження складних анімаційних або фізичних взаємодій.

Крім цього, важливо враховувати юніт-тестування – хоча воно не так поширене в інді-іграх, воно дозволяє перевіряти окремі частини логіки незалежно від загальної сцени. Наприклад, можна створити тести на перевірку точності нарахування очок, шкоди, математичних розрахунків або логіки взаємодії між об’єктами.

У фінальних етапах розробки корисно проводити ручне тестування різними користувачами – гравці можуть звернути увагу на дрібниці, які неочевидні для самого розробника. Після цього помилки та незручності фіксуються й усуваються в наступних ітераціях.

І, нарешті, варто також звертати увагу на оптимізацію продуктивності: використовувати профайлери, які показують, скільки ресурсів споживають скрипти, ефекти або фізичні взаємодії. Це дозволяє уникнути непередбачуваних падінь FPS або перевантаження девайсу, особливо на мобільних платформах.

Таким чином, процес тестування та дебагу – це не лише пошук помилок, а й постійний аналіз якості гри, її стабільності, продуктивності та зручності для користувача. Це творча частина розробки, що допомагає перетворити технічний прототип на справжній, глянцевий ігровий продукт.

Ретельно спроєктована архітектура дозволила створити гнучкий, модульний і надійний фундамент гри, який можна розширювати без ризику порушити базову функціональність. Компонентний підхід у поєднанні з подіями та менеджерами ресурсів сприяє легкій підтримці коду, а також адаптації до нових ідей.

Завдяки такому підходу команда розробників може швидко вносити зміни або додавати новий контент без необхідності кардинального переписування існуючого коду. Також зменшується ризик виникнення критичних багів, оскільки кожен модуль відповідає лише за свою частину логіки.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СКЛАДОВОЇ ГРИ

3.1. Використані інструменти та технології у процесі реалізації

У процесі створення гри в жанрі Top-Down Shooter було використано низку інструментів, програмного забезпечення та технологічних рішень, які забезпечили ефективну розробку, налагодження та тестування продукту. Правильний вибір інструментарію є ключовим фактором у досягненні цілей проєкту та забезпеченні високої якості кінцевого продукту.

Unity було обрано як основний рушій для реалізації гри завдяки його гнучкості, широким можливостям, підтримці 2D- ігор та великій кількості документації.

Серед головних причин вибору Unity:

- Підтримка C# як основної мови програмування.
- Візуальний редактор сцени, що значно пришвидшує дизайн рівнів.
- Можливість роботи з Physics2D, що необхідна для обробки зіткнень та фізики об’єктів.
- Інтеграція з анімаційною системою Animator.
- Зручна реалізація UI за допомогою Canvas.
- Наявність Asset Store для швидкого додавання графіки, звуків, інструментів розробки.

Unity також підтримує кросплатформенність, що дозволяє легко адаптувати гру під Windows, Android, WebGL тощо.

Однією з ключових причин вибору стала підтримка C# як основної мови програмування. C# — це сучасна, потужна та зручна для навчання мова, яка дозволяє створювати чистий, модульний код. Вона підтримує об’єктно-орієнтовану парадигму, що є особливо важливим у великих проєктах із багатьма взаємодіючими об’єктами. Unity має зручну інтеграцію з Visual Studio та Rider, що забезпечує зручне середовище для написання, відлагодження і тестування скриптів.

Інструментом, що суттєво прискорює процес створення контенту, є візуальний редактор сцени (Scene View). Завдяки ньому розробник може в реальному часі додавати об’єкти, змінювати їхні властивості, позиції, компоненти та компонувати рівні за принципом drag-and-drop. Це дозволяє швидко ітеративно тестувати й змінювати ідеї без необхідності постійного запуску гри. Редактор також забезпечує зручну систему навігації, групування об’єктів у ієрархії, масштабування та роботи з декількома камерами.

Ще однією перевагою Unity є повноцінна підтримка Physics2D — фізичного рушія, спеціально розробленого для 2D-ігор. Він включає зіткнення, сили, імпульси, гравітацію, тригери, фізичні матеріали, обробку контактів тощо. Physics2D забезпечує реалістичну взаємодію між об’єктами, що є критично важливим у платформерах, шутерах, головоломках та інших жанрах.

Unity також має потужну анімаційну систему Animator, яка підтримує створення складних анімацій на основі state machine. Через Animation Controller можна налаштовувати переходи між станами (наприклад, ходьба, стрибок, атака), керувати параметрами (bool, int, float, trigger) і прив’язувати логіку до них через код. Це відкриває широкі можливості для створення реалістичних рухів персонажів, ворогів, об’єктів середовища та UI-елементів.

Для створення інтерфейсів Unity надає систему Canvas, яка дозволяє легко реалізовувати меню, індикатори здоров’я, магазини, підказки, а також будь-які інші елементи користувацького інтерфейсу. Canvas підтримує адаптивну розмітку, якірну систему, компоненти ScrollView, кнопки, слайдери та інші інтерактивні елементи. Усе це легко кастомізується, анімується та інтегрується зі скриптами.

Ще однією перевагою є Unity Asset Store — офіційний маркетплейс для ресурсів, де можна знайти як безкоштовні, так і платні 2D та 3D-асети, шейдери, скрипти, анімації, фреймворки, звукові ефекти, музичні пакети, UI-набори та навіть повноцінні шаблони ігор. Це особливо корисно на ранніх етапах розробки або під час створення прототипів, коли важливо перевірити геймплейну ідею без витрат на створення унікального контенту.

Окрім того, Unity має активну спільноту, тисячі туторіалів, відеоуроків, форумів та документацію, що охоплює як базові питання, так і складні технічні рішення. Це знижує поріг входу для новачків і дозволяє швидше розв'язувати проблеми, що виникають у процесі розробки.

У сукупності всі ці фактори роблять Unity ідеальним вибором для реалізації інді-гри: він поєднує в собі технічну потужність, гнучкість, кросплатформенність і доступність, що дозволяє втілити проєкт будь-якої складності — від простих аркад до масштабних RPG.

Усі скрипти були реалізовані мовою **C#**, яка надає широкі можливості об'єктно-орієнтованого програмування, включно з інкапсуляцією, наслідуванням, поліморфізмом та делегатами.

Ця мова дозволяє:

- Створювати модульні та розширювані класи.
- Використовувати шаблони проєктування (наприклад, Singleton, Event System).
- Працювати з Unity API на повному рівні.

У середовищі Unity основною мовою програмування служить C#. Він поєднує простоту синтаксису зі всіма перевагами .NET-екосистеми, що дозволяє розробникам створювати читабельний, модульний та швидко розвивати код. Кожен скрипт у Unity зазвичай наслідує MonoBehaviour, завдяки чому рушій сам піклується про виклик життєвих методів — від Awake і Start до Update та FixedUpdate. Це дає змогу зосередитися виключно на реалізації ігрової логіки, не витрачаючи час на налаштування циклів ініціалізації чи оновлення.

Статична типізація C# гарантує, що більшість помилок виявляються ще на етапі компіляції, а об'єктно-орієнтований підхід дає змогу чітко розділити функціональні блоки гри. Використовуючи класи й інтерфейси, розробник може створити, наприклад, базовий інтерфейс IWeapon або абстрактний клас EnemyBase і потім додавати різні реалізації, не змінюючи вже налагоджений код. Делегати та події, які є невід'ємною частиною C#, дозволяють організувати подійну модель взаємодії між компонентами — наприклад, система здоров'я може сповіщати

UI-менеджер про зміну показників, а інші класи реагувати на це без прямого зв’язку.

Для обробки колекцій у C# доступні дженерики та LINQ, що значно спрощує фільтрацію, сортування й групування списків об’єктів. Наприклад, знайти всіх ворогів із здоров’ям нижче половини максимального можна однією лінією коду з Where-виразом. Такі можливості знижують кількість шаблонного коду й покращують читабельність.

Хоча Unity працює в одному головному потоці, стандартні async/await-конструкції C# корисні при завантаженні зовнішніх ресурсів або обробці мережевих запитів через UnityWebRequest. Це дозволяє уникати фризів інтерфейсу й підтримувати плавність геймплею.

Через Package Manager розробник може підключати офіційні пакети Unity, такі як нові Input System, Addressables або Data-Oriented Technology Stack, не втручаючись у ядро рушія і отримуючи сучасні API та інструменти для великих проєктів.

Для налагодження C#-скриптів застосовують Visual Studio або JetBrains Rider, де можна встановлювати точки зупину, відстежувати стек викликів і дивитися змінні в реальному часі. Unity Console допомагає фіксувати події через Debug.Log-повідомлення, а Assert-перевірки захищають ключові припущення в коді.

Під час розробки коду основним інструментом слугує Visual Studio, тісно інтегрована з Unity: вона автоматично підсвічує синтаксис C#, миттєво повідомляє про помилки й пропонує виправлення на льоту. Її ключова перевага—система IntelliSense, що надихає на швидке написання коду, підказуючи методи та властивості API Unity без необхідності звертатися до документації. Відлагодження у реальному часі дозволяє встановлювати точки зупину в скриптах, відстежувати значення змінних і негайно виявляти помилки під час виконання в редакторі. Для зручності спільної роботи та відстеження змін у версіях Visual Studio сприймає Git-репозиторії, підсвічуючи конфлікти merge та допомагаючи їх вирішувати прямо в середовищі. У великих проєктах альтернативою виступає JetBrains Rider: він

досить легкий на пам’ять, має покращену індексацію Unity-компонентів і комплексний набір інструментів рефакторингу, що особливо цінується при обробці тисяч рядків одночасно.

У процесі розробки гри важливою складовою стала організація зберігання, відстеження змін і спільної роботи над проєктом. Для цього було обрано систему контролю версій Git у поєднанні з хмарним сервісом GitHub, який надає зручний інтерфейс для ведення репозиторіїв, перегляду комітів, роботи з гілками та командної взаємодії. Завдяки Git можна зберігати історію змін у проєкті, що дозволяє в будь-який момент повернутися до попереднього стабільного стану, якщо нові зміни викликають баги або нестабільну поведінку гри. Це особливо корисно на етапах експериментального впровадження нових механік, коли ризик помилок високий.

Особливу увагу було приділено налаштуванню файлу .gitignore, який був адаптований під Unity. До нього внесено всі тимчасові файли, кеші, автоматично згенеровані бібліотеки та інші артефакти, які не мають стосунку до вихідного коду гри. Завдяки цьому в репозиторій потрапляють тільки дійсно важливі ресурси: скрипти, сцени, префаби, текстури та налаштування, які прямо впливають на геймплей або зовнішній вигляд гри. Такий підхід забезпечує чистоту структури проєкту, спрощує навігацію та пришвидшує клонування/оновлення репозиторію.

Також GitHub дозволяє працювати з гілками — кожна фіча чи виправлення може реалізовуватись у власній гілці, незалежно від головної. Це дозволяє кільком розробникам працювати паралельно, не блокуючи один одного. Після завершення роботи зміни проходять перевірку та об'єднуються через pull request. У разі конфліктів, Git надає інструменти для їх вирішення з можливістю зберегти важливі частини обох версій.

Щодо графічного оформлення, то важливу роль відіграли графічні редактори, які використовувалися на різних етапах створення візуального контенту. Для складних графічних об’єктів, фонів, обробки кольорів, створення шарів, ефектів освітлення та накладень застосовувався Adobe Photoshop — він дозволяє працювати з високою точністю, використовуючи професійні інструменти

ретуші, пензлів і масок. У тих випадках, коли потрібно було виконати швидке редагування або працювати з відкритим ПЗ, використовувався GIMP — безкоштовна альтернатива з потужним функціоналом, яка зручно інтегрується у робочий процес, особливо коли потрібне легке редагування PNG або створення масок для альфа-каналів.

Для створення піксельної графіки та анімації спрайтів використовувався спеціалізований онлайн-інструмент Piskel. Він дозволив зручно працювати з тайл-сетами, створювати покадрові анімації, зберігати анімаційні цикли у вигляді sprite sheet, що легко імпортуються в Unity. Інтерфейс Piskel розрахований саме на pixel-art стиль, що прискорювало створення графіки для персонажів, ворогів і об'єктів середовища.

На початковому етапі, для економії часу та швидкого прототипування, активно використовувалися безкоштовні ресурси від Kenney.nl. Це популярна колекція 2D та 3D-асетів з відкритими ліцензіями, яка включає інтерфейсні елементи, тайли, персонажів, іконки та звуки. Kenney Assets дозволили швидко зібрати перші версії сцени, не витрачаючи ресурси на розробку графіки з нуля. Це дало змогу зосередитися на механіці гри, геймплейних рішеннях та логіці взаємодії без потреби одразу залучати художника.

Для моделювання руху та виявлення зіткнень застосовувалися компоненти Rigidbody2D і Collider2D (використовувалися як Box, так і Circle). Rigidbody2D відповідав за коректне переміщення об’єктів під дією сил і гравітації, а Collider2D — за точне визначення контакту між персонажем, ворогами та кулями. Для перевірки попадання куль у ворогів використовувався метод Raycasting, який відправляє «промені» від ствола Weapon у напрямку прицілу й миттєво фіксує колізію без необхідності інстанціювати багато фіктивних об’єктів.

У реалізації HUD та меню головним контейнером слугував Canvas, до якого прив’язувалися окремі елементи TextMeshPro для виводу чітких шрифтів без пікселізації. Піктограми здоров’я, патронів і панелі навігації створювалися як UI-зображення (Image), а інтерактивні елементи типу Button та Slider дозволяли користувачеві змінювати гучність звуку, регулювати складність ігрового процесу

та перезапускати рівень без виходу з гри. Все це працює в режимі адаптивного масштабування, щоб інтерфейс виглядав коректно на будь-якій роздільній здатності.

Щоденне тестування супроводжувалося переглядом консолі Unity — вона акумулювала повідомлення про помилки, попередження та власні логи, розставлені в коді для відстеження життєвих подій. Щоб бачити траєкторію пострілів або зони виявлення ворогів, використовувалися Debug.DrawLine та Gizmos, які під час роботи в редакторі малюють допоміжні лінії й контури. Для більш глибокого аналізу продуктивності застосовувався Profiler, що дозволив виявити «вузькі місця» в спрайтовому рендерингу, скриптах або викликах API Unity та вчасно оптимізувати їх до рівня комфортного FPS. Додатково активно використовувались breakpoints і покрокове виконання (step-by-step) у вбудованому дебагері, що допомагало точно визначити момент виникнення помилки.

Це особливо корисно при виявленні логічних помилок або неправильного порядку виклику методів. Аналіз стеку викликів (call stack) дозволяв простежити, з якого саме місця в коді прийшов проблемний сигнал. Також проводилось тестування на різних розширеннях екрана, щоб перевірити адаптивність UI та стабільність розміщення елементів. На пізніших етапах були задіяні тести з емуляцією слабших пристроїв, аби переконатися в іграбельності на мінімальних конфігураціях. Логи регулярно експортувались для подальшого аналізу, а критичні помилки маркувались для швидкого реагування.

Важливу роль відіграло й тестування на багатокористувацький сценарій — це дозволило виявити проблеми синхронізації та затримок. Для автоматизації окремих перевірок створювались невеликі утиліти, що спрощували перевірку сценаріїв відтворення помилок. Всі знайдені баги документувались у системі трекінгу, що забезпечувало прозорість у командній роботі та дозволяло контролювати прогрес їх виправлення. Такий підхід дозволив досягти високого рівня стабільності проєкту ще до фінального етапу релізного тестування.

3.2. Планування, етапи розробки та управління проєктом

У нашому проєкті ми повністю перейшли на роботу в Trello, у якому кожен етап розробки відображається як своя колонка ("Backlog", "In Progress", "Review", "Done") на одній дошці. Початкова вкладка – огляд дошки – має містити скріншот із загальним виглядом усіх списків, щоб будь-хто міг одразу побачити поточний статус задач.

У колонці Backlog зберігаються картки з описом усіх завдань: аналіз жанру та аналогів, прототипування руху, налаштування бізнес-логіки тощо. Тут добре розмістити скріншот із розгорнутою карткою (Card Detail), де видно чек-ліст підзадач та коментарі, що пояснюють критерії прийняття.

Коли ми починаємо спринт, задачі перетягуються в колонку In Progress. Для цієї фази доцільно додати скріншот із Kanban-режиму, аби було видно рух карток відліва та вправо. В описі картки варто додати чек-ліст “Development → Code review → Testing” і в коментарі закріпити лінк на відповідну гілку GitHub.

Після кодування і базового тестування картки переходять до Review. Тут скріншот покаже, як виглядає коментар з проханням про рев’ю, а також чек-ліст задач з позначками “Done” навпроти пунктів, які пройшли перевірку.

І нарешті в колонці Done ми робимо фінальний скріншот із календарним виглядом (Calendar Power-Up), щоб можна було проілюструвати, які завдання в які дні були закриті.

Окремо рекомендується додати на дошку графік прогресу (Burndown Chart Power-Up): скріншот самої дашборд-вкладки з діаграмою допоможе візуально продемонструвати, як ми укладаємося в заплановані терміни.

Таким чином, чотири ключові скріншоти (загальний вигляд дошки, картка Backlog, Kanban-стан у спринті, Burndown-діаграма) дадуть повне уявлення про наш процес: від формування списку завдань до їхнього успішного завершення. Це не тільки спрощує комунікацію в команді, а й робить управління розвитком гри максимально прозорим.

3.3. Ключові функціональні можливості розробленої гри

Розроблений прототип жанру Top-Down Shooter поєднує низку взаємопов’язаних систем, що забезпечують гнучкий, динамічний ігровий досвід, водночас зберігаючи простоту й зрозумілість для гравця.

У центрі геймплею лежить керування персонажем, яке побудоване на C#-скриптах PlayerManager і PlayerMovement. Клас PlayerManager відповідає за збереження стану героя, облік здоров’я, боєзапасу та активних перків, тоді як PlayerMovement перетворює вхідні дані (Клавіші W/A/S/D) у плавний рух по всіх напрямах із одночасним незалежним прицілюванням за курсором миші або другим стиком геймпада. Такий розподіл ролей дозволяє масштабувати підсистеми окремо й легко адаптувати поведінку персонажа під нові потреби.

Система озброєння, представлена абстрактним класом Weapon та його конкретною реалізацією Rifle, забезпечує контроль над темпом вогню, розподілом патронів та розльотом кулі. Після натискання «вистрілити», об’єкт Bullet створюється з параметрами швидкості, діапазону дії та шкоди, а метод Bullet.OnCollision викликає EnemyHealth.TakeDamage для завдання ушкоджень ворогу. Це розділення дозволяє додавати нові типи зброї (пістолети чи гранатомети) без змін у загальній архітектурі.

Поведінка супротивників реалізовано у класі EnemyAI разом із його модулем EnemyHealth. Вороги патрулюють задані точки, переходять в режим переслідування при виявленні гравця й атакують із заданою частотою стрільби або влученням у ближньому бою. Коли EnemyHealth фіксує критичну шкоду, запускається анімація загибелі та подія знищення, що передається GameManager для контролю хвиль.

Безперебійний і зрозумілий інтерфейс користувача опікується UIManager: він оновлює HUD у реальному часі, відображаючи смугу здоров’я, індикатор боєзапасу, лічильник хвиль і час до наступної генерації супротивників. Меню паузи та екран завершення гри також обслуговуються цим же класом, що гарантує узгодженість вигляду елементів і знижує дублювання коду.

Графічні ефекти працюють синхронно: піксель-артові спрайти супроводжуються спалаючими частинками при пострілах. Для цього застосовано Unity Particle System.

Завдяки такому поділу на модулі й чіткому розмежуванню відповідальностей, кожна підсистема може розвиватися незалежно: додавання нових видів ворогів, спеціальних здібностей гравця або режимів гри не вимагатиме кардинальної перебудови існуючої архітектури. Це гарантує не лише стабільність базового функціоналу, а й високу гнучкість для подальших ітерацій і поліпшень. Такий підхід особливо корисний у командній розробці, де кілька розробників можуть працювати над різними частинами проєкту паралельно, не заважаючи одне одному.

Завдяки ізольованості модулів знижується ризик виникнення побічних ефектів при внесенні змін. Крім того, повторне використання коду стає значно простішим — уже реалізовані компоненти можна легко адаптувати для нових рівнів або механік. Це також спрощує процес тестування, оскільки кожну систему можна перевіряти окремо, використовуючи мок-об’єкти або спеціальні сценарії. З часом, коли гра розростається, модульна структура дозволяє уникнути перетворення проєкту на "моноліт", який важко підтримувати. Упровадження нових функцій або UX-покращень не вимагає значних витрат часу на перебудову логіки. Більше того, це відкриває шлях до реалізації розширень або DLC, що можуть інтегруватися без конфліктів зі старими елементами. Завдяки цьому гру можна довше підтримувати, розвивати та масштабувати. Важливо й те, що така архітектура полегшує адаптацію під різні платформи, змінюючи лише окремі підсистеми (наприклад, керування або графіку). У підсумку модульність стає не лише технічною перевагою, а й стратегічною основою довготривалого життєвого циклу гри.

4. ДЕМОНСТРАЦІЯ ТА ТЕСТУВАННЯ

4.1. Особливості ігрового процесу та інтерфейсу користувача

На рисунку 4.1 – показано загальний перелік створених класів, які в свою чергу мають опис функціоналу гри.

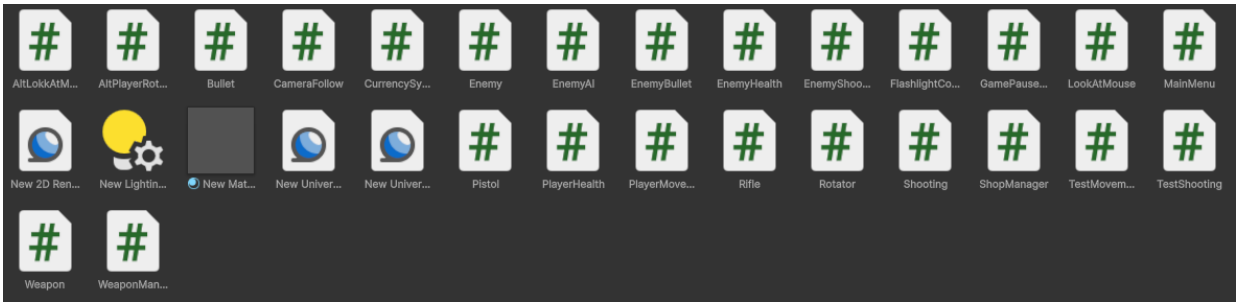
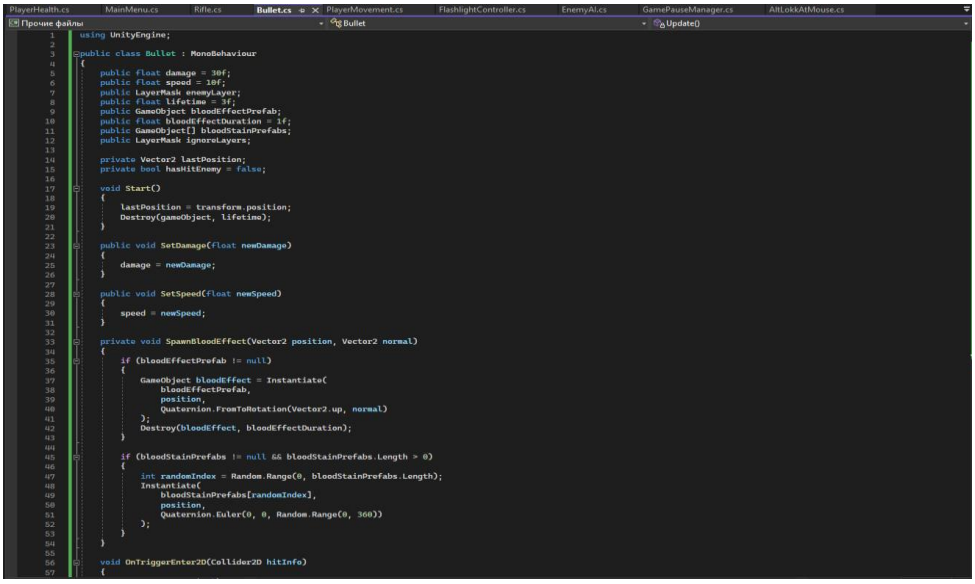


Рис. 4.1 Список класів

Клас Bullet у Unity відповідає за керування життєвим циклом та поведінкою кулі в нашому Top-Down Shooter. Після створення снаряд відразу запам’ятовує своє стартове положення та запускає відлік часу свого існування – через заданий інспектором інтервал він автоматично знищується, щоб уникнути нескінченного нагромадження об’єктів у сцені. На рисунку 4.2 можна побачити клас Bullets.



```
using UnityEngine;

public class Bullet : MonoBehaviour
{
    public float damage = 30f;
    public float speed = 10f;
    public LayerMask enemyLayer;
    public float lifetime = 3f;
    public GameObject bloodEffectPrefab;
    public float bloodEffectDuration = 1f;
    public GameObject[] bloodStainPrefabs;
    public LayerMask ignoreLayers;

    private Vector2 lastPosition;
    private bool hasHitEnemy = false;

    void Start()
    {
        lastPosition = transform.position;
        Destroy(gameObject, lifetime);
    }

    public void SetDamage(float newDamage)
    {
        damage = newDamage;
    }

    public void SetSpeed(float newSpeed)
    {
        speed = newSpeed;
    }

    private void SpawnBloodEffect(Vector2 position, Vector2 normal)
    {
        if (bloodEffectPrefab != null)
        {
            GameObject bloodEffect = Instantiate(
                bloodEffectPrefab,
                position,
                Quaternion.FromToRotation(Vector2.up, normal)
            );
            Destroy(bloodEffect, bloodEffectDuration);
        }

        if (bloodStainPrefabs != null && bloodStainPrefabs.Length > 0)
        {
            int randomIndex = Random.Range(0, bloodStainPrefabs.Length);
            Instantiate(
                bloodStainPrefabs[randomIndex],
                position,
                Quaternion.Euler(0, 0, Random.Range(0, 360))
            );
        }
    }

    void OnTriggerEnter2D(Collider2D hitInfo)
    {
```

Рис. 4.2 Клас Bullet.cs

Снаряд рухається вперед уздовж власної локальної осі X із заданою швидкістю, а при зустрічі з будь-яким колайдером, що не належить до ігнорованих шарів, реагує на зіткнення.

Якщо в зоні зіткнення виявляється компонент EnemyHealth, куля відразу застосовує до нього зазначену шкоду, після чого на місці попадання створюється ефект крові: із першим шару ми відтворюємо частинковий префаб bloodEffectPrefab, орієнтуючи його за нормаллю поверхні, а під ним – випадкову пляму крові з масиву bloodStainPrefabs, повертаючи її на випадковий кут. Далі куля одразу знищується, щоб запобігти повторному завданню шкоди. Якщо ж зіткнення трапляється з об’єктом без EnemyHealth, куля просто зникає без додаткових ефектів.

Клас дозволяє динамічно змінювати його основні властивості – damage і speed – через публічні методи SetDamage та SetSpeed, що робить його універсальним для будь-яких типів зброї. Маски шарів enemyLayer і ignoreLayers легко налаштовуються в Інспекторі, дозволяючи контролювати, з якими об’єктами куля може взаємодіяти. Завдяки такій модульній структурі компонент забезпечує плавний політ снаряда, коректне визначення попадань і видовищне візуальне підкріплення, не навантажуючи систему зайвою логікою чи зайвими перевірками.

У нашій грі жанру Top-Down Shooter ігровий процес сконцентрований навколо плавної взаємодії між рухом персонажа, веденням прицільного вогню та швидкою реакцією на хвилі супротивників. Як тільки гравець опиняється в арені, він миттєво занурюється в динаміку: управління поєднує клавіші W/A/S/D для переміщення з незалежним наведенням курсора миші, що дає можливість рухатися та вести вогонь у різних напрямках одночасно. Система контролю над рухом забезпечує відчуття високої майстерності. Завдяки інерційній фізиці та плавному прискоренню руху, гравець відчуває природну реакцію персонажа на введення команд. На рисунку 4.3 видно голвне меню гри.

Рис. 4.3 Головне меню гри

Вороги очікують на різних позиціях карти й поводяться за штучно-інтелектуальними шаблонами: вони патрулюють, відступають при тісному контакті, кидаються у переслідування при необхідності. Такий підхід гарантує безперебійну роботу без надлишкового навантаження на систему й дозволяє підтримувати високий FPS навіть у найгарячіших перестрілках.

Інтерфейс користувача реалізовано з прагненням залишити максимум простору під екшн, але при цьому надати всю необхідну інформацію. Угорі зліва під шапкою екрану знаходиться індикатор поточного здоров’я з плавним спадом кольору, що сигналізує про отримані ушкодження, а під ним індикатор енергії ліхтаря. Таке поєднання продуманої поведінки ворогів і лаконічного, функціонального інтерфейсу сприяє глибшому зануренню в ігровий процес. Гравець постійно відчуває динаміку бою, водночас не відволікаючись на зайву інформацію або складні HUD-елементи. Завдяки зручному розташуванню індикаторів важливі дані сприймаються миттєво, що особливо критично в напружених ситуаціях. Візуальні ефекти пошкоджень і витрати енергії доповнюють геймплей, підсилюючи емоційний вплив. Усе це створює відчуття злагодженої та зручної системи, в якій технічна реалізація тісно пов’язана з

геймдизайнерськими рішеннями. У результаті формується якісний ігровий досвід, де кожен елемент виконує чітку функцію і працює на спільну мету — забезпечити захопливу, але зрозумілу гру. На риснуку 4.4 видно індикатори здоровя та заряду ліхтаря та геймплей гри.

Рис. 4.4 Робота ліхтаря та геймплей гри

Меню паузи та екран завершення оформлені в тому ж стильовому ключі й з’являються над напівпрозорим фоном бою, дозволяючи не втрачати відчуття простору та атмосфери гри. Тут можна перезапустити рівень, вийти з гри, продовжити гру. Завдяки такому поєднанню чіткого зворотного зв’язку, інтуїтивного навчання й мінімалістичного, але виразного HUD гравець отримує максимально ясний і приємний досвід взаємодії з грою. Усі елементи інтерфейсу розміщені логічно й не перевантажують екран, зберігаючи фокус на основній дії. Кольорова палітра меню гармоніює з загальним візуальним стилем гри, підсилюючи цілісність сприйняття. Анімації відкриття та закриття меню плавні й непомітно інтегруються в ігровий процес, не вибиваючи гравця з ритму. Також враховано зручність керування — навігація по меню легко здійснюється як мишею, так і з геймпада. Це забезпечує універсальність і комфорт для різних категорій користувачів.

4.2. Перспективи вдосконалення та майбутні напрями розвитку гри

Нинішній прототип Top-Down Shooter закладає основи геймплею, але його можна зробити значно багатшим і динамічнішим. По-перше, варто розширити арсенал (лазери, керовані ракети) і типи ворогів (невидимки, броньовані, із розгалуженими хвилями), а також додати потужних «босів» і різні карти з унікальними перешкодами.

Щоб підсилити емоційне занурення, доцільно впровадити мінімальну сюжетну лінію: вступну катсцену, проміжні завдання та NPC, які направляють гравця. Мультиплеєр (кооператив і PvP-арени) разом із онлайн-лідербордами підвищать змагальний дух і доведуть гру до етапу соціальної взаємодії.

Технічну сторону слід оптимізувати через налаштування графічних пресетів, постобробку (bloom, motion blur) і порт на мобільні/консольні платформи із адаптованим управлінням. Анімації можна зробити плавнішими, додавши переходи між станами.

Економічна система з внутрішньою валютою, магазином апгрейдів та досягненнями підтримає інтерес у довготривалій перспективі й відкриє шлях до free-to-play або DLC.

Штучний інтелект ворогів — із патрулюванням, укриттями та тактичним відступом — зробить кожен раунд непередбачуваним. І нарешті, багатомовний інтерфейс та інклюзивні налаштування (шрифти, кольори, субтитри) дозволять охопити ширшу аудиторію й показати турботу про різні потреби гравців.

ВИСНОВКИ

1. Виконано ґрунтовний аналітичний огляд жанру Top-Down Shooter, у який увійшли дослідження його історичного становлення, ключових ігрових механік та візуальних особливостей — це дозволило виокремити ті рішення, які забезпечують високу динаміку та залученість гравця.

2. Проаналізовано сучасний стан ігрової індустрії, зокрема вплив кіберспорту на баланс ігрових механік, роль інді-розробки в генерації нових ідей і вимоги мобільного сегмента до коротких сесій та простого інтерфейсу.

3. Оцінено наявні інструменти для розробки: порівняння Unity, Unreal Engine та Godot показало, що Unity є оптимальним вибором завдяки розвиненим 2D-інструментам, швидкому циклу прототипування та великому Asset Store.

4. Розроблено архітектуру проекту з допомогою UML-діаграм: класи PlayerManager, PlayerMovement, EnemyAI, Weapon, Rifle, Bullet, EnemyHealth та UIManager чітко відокремлені за відповідальністю, що забезпечує модульність і полегшує подальшу підтримку коду.

5. Створено робочий прототип гри з реалізацією базових механік: плавного руху персонажа, незалежного прицілювання, різних типів ворогів із AI, системи стрільби та колізій, а також простого, але зручного інтерфейсу користувача.

6. Проведено тестування функціоналу та виявлено відсутність критичних помилок, що підтверджує ефективність оптимізацій (object pooling, зведення викликів Update, стиснення ресурсів).

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

Сікорський Д. В., Дібрівний О.А. Реалізація системи штучного інтелекту ворогів у грі жанру Top-Down Shooter. Матеріали VI Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в ІКТ”. Збірник тез. 24.04.2025, ДУІКТ, м. Київ., 2025. С.279-280.

Сікорський Д. В., Дібрівний О.А. Використання шаблонів проєктування при створенні гри в жанрі Top-Down Shooter. Матеріали VI Всеукраїнської Науково-Технічної конференції “Застосування програмного забезпечення в ІКТ”. Збірник тез. 24.04.2025, ДУІКТ, м. Київ., 2025. С.280-282.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unity Learn — навчальні курси та туторіали [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: Unity Learn – Дата звернення: 28.05.2025

2. Дія.Освіта — Освітні серіали з геймдизайну [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: Дія.Освіта – Дата звернення: 28.05.2025

3. Розробка 2D платформної гри з використанням Unity [Електронний ресурс]: [PDF] – електронні дані. – Режим доступу: КМА – Дата звернення: 28.05.2025

4. Методичні матеріали з Unity — НТУ «Дніпровська політехніка» [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: it.nmu.org.ua – Дата звернення: 28.05.2025

5. Lviv IT Cluster — освітні проєкти та аналітика [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: Lviv IT Cluster – Дата звернення: 28.05.2025

6. GameDev DOU — статті про Unity та геймдев [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: DOU – Дата звернення: 28.05.2025

7. Розробка комп'ютерної гри "Jump and Move" в жанрі 2D платформер [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: Репозиторій ЛНАУ – Дата звернення: 28.05.2025

8. Основи програмування на C# | Prometheus [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: courses.prometheus.org.ua – Дата звернення: 28.05.2025

9. Професія геймдизайнер — хто це, чим займається, і як ним стати [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: SWP School – Дата звернення: 28.05.2025

10. Як зробити переклад UI Unity за 8 хвилин [Електронний ресурс]: [Відео] – електронні дані. – Режим доступу: YouTube – Дата звернення: 28.05.2025

11. Матеріали конференції «Комп'ютерні ігри та мультимедіа як інструмент навчання» [Електронний ресурс]: [PDF] – електронні дані. – Режим доступу: otfk.od.ua – Дата звернення: 28.05.2025

12. Game Station Academy — Основи геймдизайну (базовий курс) [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: gamestation.academy – Дата звернення: 28.05.2025

13. Augmented Reality — Доповнена реальність | Prometheus [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: Прометей – Дата звернення: 28.05.2025

14. Unity Documentation [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: https://docs.unity3d.com/Manual/ – Дата звернення: 28.05.2025

15. Game Programming Patterns — Robert Nystrom [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: https://gameprogrammingpatterns.com/ – Дата звернення: 28.05.2025

16. Історія розвитку комп'ютерних і відео ігор: Від Пакмена До Кіберспорту. [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: https://computerrgameshistory.blogspot.com/ - Дата звернення: 15.05.2025

17. Кращі ігри в жанрі Tower Defense для смартфонів [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: https://igate.com.ua/news/18170-luchshie-igry-v-zhanre-tower-defence-dlyasmartfonov - Дата звернення: 16.05.2024

18. Gamasutra (Game Developer) — аналітика геймдизайну [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: https://www.gamedeveloper.com/ – Дата звернення: 28.05.2025

19. Найвідоміші відеоігри, створені вітчизняними розробниками. [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: https://www.redbull.com/ua-uk/best-ukrainian-games - Дата звернення: 28.04.2022

20. Unity Learn — навчальні курси та туторіали [Електронний ресурс]: [Веб-сайт] – електронні дані. – Режим доступу: Unity – Дата звернення: 28.05.2025

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка гри в жанрі Top-Down Shooter

Виконав студент 4 курсу
групи ПД-43
Сікорський Денис Вадимович
Керівник роботи
доктор філософії (PhD), доцент кафедри ІПЗ Дібрівний Олесь Андрійович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: адаптація ігрового процесу в грі жанру Top-Down Shooter.

Об'єкт дослідження: процес гри в жанрі Top-Down Shooter.

Предмет дослідження: ігровий застосунок в жанрі Top-Down Shooter.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз існуючих рішень жанру Top Down Shooter та виявити їхні переваги та недоліки.
2. Сформулювати вимоги до розроблюванї гри, визначити основні функціональні компоненти та можливості.
3. Розробити архітектуру додатку, обрати відповідні інструменти, мову програмування, рушій, бібліотеки та формати зберігання даних.
4. Реалізувати основний функціонал гри, зокрема: управління персонажем, логіку ворогів, систему зброї, фізику кулі.
5. Забезпечити інтерфейс користувача.
6. Провести тестування функціоналу гри, виявити помилки та оптимізувати продуктивність, зокрема при роботі з великою кількістю об'єктів на сцені.

КОНЦЕПТ ГРИ

Жанр:

Top-Down Shooter з елементами виживання

Цільова аудиторія:

Гравці віком 16–35 років, які віддають перевагу динамічним шутерам із виглядом зверху, цікавляться стилістикою неону, кібертехно та постапокаліптичним світом. Основна аудиторія — хардкорні гравці та інді-фанати, які шукають швидкий, реіграбельний геймплей з поступовим розвитком складності.

Сетинг:

Події гри розгортаються у темних і занедбаних інтер'єрах невідомого підземного складу, який став епіцентром техногенної катастрофи. Колись це було секретне сховище, де проводилися дослідження біотехнологічної зброї, однак витік вірусу спричинив мутацію персоналу. Атмосфера постійної загрози, замкненість простору та мерехтливе аварійне освітлення створюють відчуття тривоги та виживання у пастці.

Короткий огляд ідеї гри:

Гравець опиняється всередині закритого складу, де внаслідок витоку вірусу було втрачено контроль над персоналом. Основна мета — вижити, досліджуючи приміщення, знищуючи мутованих ворогів та орієнтуючись у темряві за допомогою ліхтарика. Бойова система базується на реалістичній фізиці кулі, що вимагає точного прицілювання. У грі представлено кілька типів ворогів, які змінюють геймплейну динаміку. Інтерфейс мінімальний, що дозволяє повністю зосередитися на атмосфері напруги та загрози.

Платформи:

Windows (основна платформа)

Ключові особливості гри:

Гра має реалістичну фізику кулі — кожен постріл враховує напрямок і швидкість, що додає динаміки. Орієнтація в темному просторі здійснюється за допомогою ліхтарика, який освітлює лише частину зони видимості, посилюючи напруження. Вороги різноманітні: від швидких мутантів до витривалих охоронців, що вимагає адаптації тактики. Інтерфейс мінімалістичний — лише індикатори здоров'я та заряду ліхтаря, що підсилює ефект занурення.

Інші характеристики:

Керування здійснюється клавіатурою (WASD — рух) та мишею (прицілювання, стрільба). Розробка ведеться в Unity із використанням мови C# та спрайтової 2D-графіки.

АНАЛІЗ АНАЛОГІВ

Критерій / Гра	Nuclear Throne	Enter the Gungeon	Alien Shooter	Hotline Miami	Blood and Bullets
Фізика кулі	+	+	-	-	+
Відображення стану гравця (здоров'я)	-	+	-	-	+
Горор елементи	-	-	+	-	+
Наявність розхідників	+	-	+	-	+
Наявність ліхтаря	-	-	-	-	+
Кілька типів ворогів	-	+	-	+	+

5

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. реалізація управління рухом ігрового персонажа;
2. впровадження штучного інтелекту для керування поведінкою ворогів;
3. реалізація прицілювання за допомогою миші;
4. інформування гравця про стан здоров'я персонажа;

Нефункціональні вимоги:

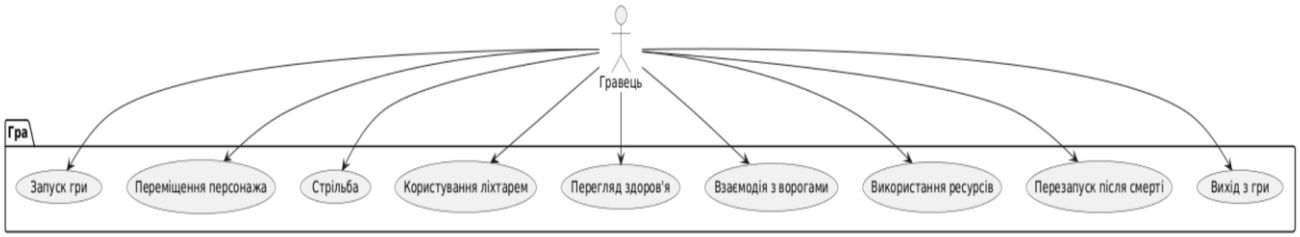
1. підтримка платформи Windows;
2. частота кадрів не менше 60 FPS;
3. час завантаження сцени не повинен перевищувати 5 секунд;
4. гра не повинна аварійно завершуватись або "вилітати" під час ігрового процесу;

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

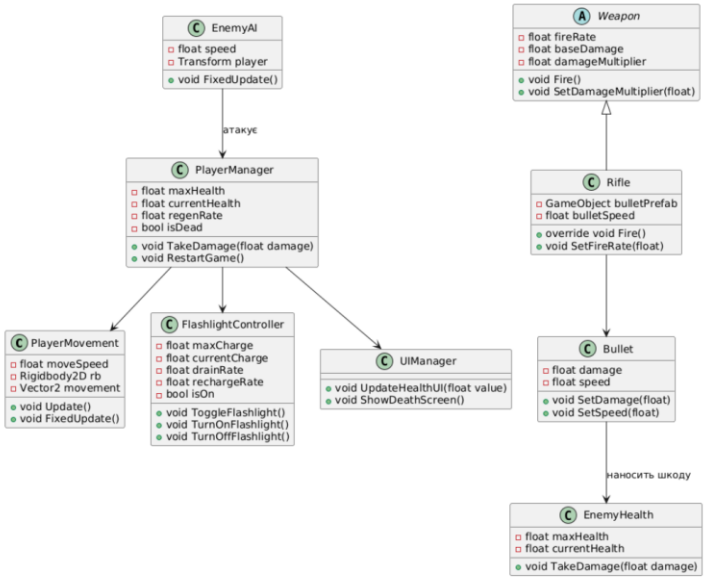
7

ДІАГРАМА ВИКОРИСТАННЯ



8

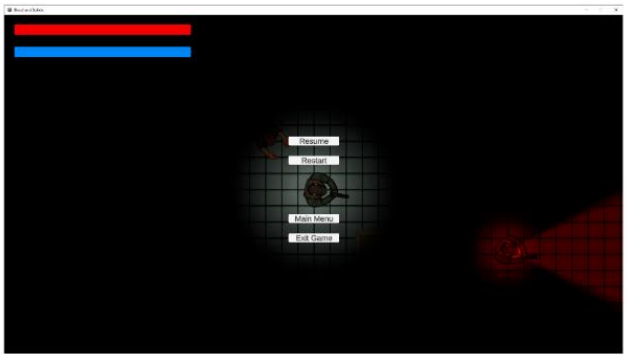
ДІАГРАМА КЛАСІВ



9

ЕКРАННІ ФОРМИ ГРИ

Головне меню гри



Меню паузи гри

10

ІГРОВИЙ ПРОЦЕС

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Сікорський Д. В., Дібрівний О.А. Реалізація системи штучного інтелекту ворогів у грі жанру Top-Down Shooter. Матеріали VI Всеукраїнської Науково-Технічної конференції “Застосування програмного забезпечення в ІКТ”. Збірник тез. 24.04.2025, ДУІКТ, м. Київ., 2025. С.279-280.
2. Сікорський Д. В., Дібрівний О.А. Використання шаблонів проєктування при створенні гри в жанрі Top-Down Shooter. Матеріали VI Всеукраїнської Науково-Технічної конференції “Застосування програмного забезпечення в ІКТ”. Збірник тез. 24.04.2025, ДУІКТ, м. Київ., 2025. С.280-282.

12

ВИСНОВКИ

1. Проаналізовано типові переваги та недоліки подібних ігор: зокрема, було звернуто увагу на реалізацію фізики куль, систем ворогів, візуальні ефекти та варіативність озброєння. Це дало змогу сформувати чітке бачення функціоналу, який має бути реалізований у власному проекті.

2. Обрано рушій Unity, мову програмування C#, базові UI-компоненти Unity для побудови інтерфейсу, використано префаби, коллайдери та Rigidbody2D для фізики.

3. Визначено функціональні і нефункціональні вимоги до гри, що дало змогу сформувати чітке уявлення про очікувану поведінку системи та рівень її якості, забезпечивши системний підхід до розробки.

4. Розроблено архітектуру гри з чітким розділенням логіки на окремі класи відповідно до принципів модульності та повторного використання коду.

5. Реалізовано механіку польоту кулі з урахуванням напряму, швидкості та взаємодії з оточенням, забезпечено обробку зіткнень з ворогами та перешкодами. У майбутньому це дозволить розширити бойову систему додаванням ефектів віддачі, зон ураження, типів снарядів (наприклад, вибухових), а також можливістю використання фізичних об'єктів як укриттів або перешкод.

6. Забезпечено модульний інтерфейс користувача, виведено слайдер здоров’я, кнопку перезапуску, індикатор заряду ліхтаря.

7. Проведено тестування функціоналу, яке дозволило виявити та усунути критичні помилки, перевірити стабільність гри та підтвердити відповідність реалізованих можливостей поставленим вимогам.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
using UnityEngine;

public class Bullet : MonoBehaviour
{
    public float damage = 30f;
    public float speed = 10f;
    public LayerMask enemyLayer;
    public float lifetime = 3f;
    public GameObject bloodEffectPrefab;
    public float bloodEffectDuration = 1f;
    public GameObject[] bloodStainPrefabs;
    public LayerMask ignoreLayers;

    private Vector2 lastPosition;
    private bool hasHitEnemy = false;

    void Start()
    {
        lastPosition = transform.position;
        Destroy(gameObject, lifetime);
    }

    public void SetDamage(float newDamage)
    {
        damage = newDamage;
    }

    public void SetSpeed(float newSpeed)
    {
        speed = newSpeed;
    }

    private void SpawnBloodEffect(Vector2 position, Vector2
normal)
    {
        if (bloodEffectPrefab != null)
        {
            GameObject bloodEffect = Instantiate(
                bloodEffectPrefab,
                position,
                Quaternion.FromToRotation(Vector2.up, normal)
            );
            Destroy(bloodEffect, bloodEffectDuration);
        }

        if (bloodStainPrefabs != null &&
bloodStainPrefabs.Length > 0)
        {
            int randomIndex = Random.Range(0,
bloodStainPrefabs.Length);
            Instantiate(
                bloodStainPrefabs[randomIndex],
                position,
                Quaternion.Euler(0, 0, Random.Range(0, 360))
            );
        }
    }

    void OnTriggerEnter2D(Collider2D hitInfo)
    {
        if (((1 << hitInfo.gameObject.layer) & ignoreLayers) != 0)
            return;

        if (hasHitEnemy)
            return;

        EnemyHealth enemyHealth =
hitInfo.GetComponent<EnemyHealth>();
        if (enemyHealth != null)
        {
            hasHitEnemy = true;
            enemyHealth.TakeDamage(damage);

SpawnBloodEffect(hitInfo.ClosestPoint(transform.position),
hitInfo.transform.up);
            Destroy(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }

    void Update()
    {
        Vector2 direction = transform.right;
        Vector2 nextPosition = (Vector2)transform.position +
direction * speed * Time.deltaTime;
        transform.position = nextPosition;
        lastPosition = nextPosition;
    }
}
          isRefreshing = false;
          refreshSubject.next(false);
          tokenService.revokeToken();
          router.navigateByUrl('/login');
          return throwError(() => new Error("Authentication Error:
Redirected"));
        })
      );

    } else {
      return refreshSubject.pipe(
        filter(isRefreshed => isRefreshed),
        take(1),
        switchMap(() => {
          const clonedRequest = req.clone({
            setHeaders: {
              Authorization: `Bearer
${tokenService.getAccessToken()}` // Отримуємо оновлений
токен
            }
          });
          return next(clonedRequest);
        })
      );
    }
  }

  router.navigateByUrl('/login');
  return throwError(() => error);
}using UnityEngine;

public class WeaponManager : MonoBehaviour
{
    public enum WeaponType { Pistol, AutomaticRifle }
```

```
    public WeaponType currentWeapon;


    public float pistolDamageMultiplier = 1f;
    public float automaticRifleDamageMultiplier = 1.5f;


    public GameObject pistolPrefab;
    public GameObject automaticRiflePrefab;

    private GameObject currentWeaponObject;
    private Weapon currentWeaponScript;

    private void Start()
    {

        SetWeapon(currentWeapon);
    }

    private void Update()
    {

        if (Input.GetMouseButtonDown(0) && currentWeapon ==
WeaponType.Pistol)
        {
            if (currentWeaponScript != null)
            {
                currentWeaponScript.Fire();
            }
        }


        if (Input.GetMouseButton(0) && currentWeapon ==
WeaponType.AutomaticRifle)
        {
            if (currentWeaponScript != null)
            {
                currentWeaponScript.Fire();
            }
        }


        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            ChangeWeapon(WeaponType.Pistol);
        }
        else if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            ChangeWeapon(WeaponType.AutomaticRifle);
        }
    }

    public void ChangeWeapon(WeaponType newWeapon)
    {
        currentWeapon = newWeapon;
        SetWeapon(newWeapon);
    }


    private void SetWeapon(WeaponType weaponType)
    {

        if (currentWeaponObject != null)
        {
            Destroy(currentWeaponObject);
        }


        switch (weaponType)
        {
            case WeaponType.Pistol:
                currentWeaponObject = Instantiate(pistolPrefab,
transform);
                currentWeaponScript =
currentWeaponObject.GetComponent<Pistol>();

currentWeaponObject.GetComponent<Pistol>().SetDamageMu
ltiplier(pistolDamageMultiplier);
                Debug.Log("Pistol Damage Multiplier встановлено:
" + pistolDamageMultiplier);
                break;

            case WeaponType.AutomaticRifle:
                currentWeaponObject =
Instantiate(automaticRiflePrefab, transform);
                currentWeaponScript =
currentWeaponObject.GetComponent<Rifle>();

currentWeaponObject.GetComponent<Rifle>().SetDamageMul
tiplier(automaticRifleDamageMultiplier);


                ((Rifle)currentWeaponScript).SetFireRate(0.1f);

                Debug.Log("Rifle Damage Multiplier встановлено: "
+ automaticRifleDamageMultiplier);
                break;
        }
        if (currentWeaponScript == null)
        {
            Debug.LogError("currentWeaponScript не був
встановлений!");
        }
    }


    public float GetDamageMultiplier()
    {
        switch (currentWeapon)
        {
            case WeaponType.Pistol:
                return pistolDamageMultiplier;
            case WeaponType.AutomaticRifle:
                return automaticRifleDamageMultiplier;
            default:
                return 1f;
        }
    }
}using UnityEngine;

public class Shooting : MonoBehaviour
{
    public GameObject bulletPrefab;
    public Transform firePoint;
    public float bulletSpeed = 10f;
    public float shootRate = 0.5f;

    private float nextShootTime = 0f;

    void Update()
    {
        if (Input.GetMouseButton(0))
        {
            Shoot();
        }
    }

    void Shoot()
    {
        if (Time.time >= nextShootTime)
```

```
    {

        nextShootTime = Time.time + shootRate;
        Vector3 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        mousePosition.z = 0;

        Vector3 shootDirection = (mousePosition - firePoint.position).normalized;

        GameObject bullet = Instantiate(bulletPrefab, firePoint.position, Quaternion.identity);
        Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();
        rb.linearVelocity = shootDirection * bulletSpeed;

        float angle = Mathf.Atan2(shootDirection.y, shootDirection.x) * Mathf.Rad2Deg;
        bullet.transform.rotation = Quaternion.Euler(0, 0, angle);

        Destroy(bullet, 2f);

    }
  }
}using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
  public float moveSpeed = 5f;
  private Rigidbody2D rb;
  private Vector2 movement;
  private Animator animator;

  void Start()
  {
    rb = GetComponent<Rigidbody2D>();
    animator = transform.Find("player_sprite").GetComponent<Animator>();
  }

  void Update()
  {
    movement.x = Input.GetAxisRaw("Horizontal"); // A / D
    movement.y = Input.GetAxisRaw("Vertical");   // W / S


    //animator.SetFloat("MoveX", movement.x);
    //animator.SetFloat("MoveY", movement.y);
    //animator.SetFloat("Speed", movement.sqrMagnitude);
  }

  void FixedUpdate()
  {
    rb.linearVelocity = movement.normalized * moveSpeed;
  }
}using UnityEngine;

public class Rifle : Weapon
{
  public GameObject bulletPrefab;
  public float bulletSpeed = 15f;
  public GameObject muzzleFlash;
  public GameObject muzzleFlashPrefab;
  private float nextFireTime = 0f;
  public float muzzleFlashDuration = 0.1f;
  //private float lastMuzzleFlashTime = 0f;
  public float fireRateMuzzle = 0.5f;
  private bool isFiring = false;

  private void Update()
  {

    if (Input.GetMouseButton(0))
    {

      if (!muzzleFlash.activeSelf)
      {
        muzzleFlash.SetActive(true);
      }


      Fire();
    }
    else
    {

      if (muzzleFlash.activeSelf)
      {
        muzzleFlash.SetActive(false);
      }
    }
  }


  public override void Fire()
  {

    if (Input.GetMouseButton(0))
    {
      isFiring = true;
      if (Time.time >= nextFireTime)
      {
        nextFireTime = Time.time + fireRate;


        Vector3 firePointGlobalPosition = firePoint.TransformPoint(Vector3.zero);
        Debug.Log("FirePoint global position: " + firePointGlobalPosition);


        GameObject bullet = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation * Quaternion.Euler(0, 0, 0));
        Bullet bulletScript = bullet.GetComponent<Bullet>();
        bulletScript.SetDamage(baseDamage * damageMultiplier);
        bulletScript.SetSpeed(bulletSpeed);

        Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();
        rb.linearVelocity = firePoint.right * bulletSpeed;


      }
    }
    else
    {
      isFiring = false;
    }
  }




  public override void SetDamageMultiplier(float multiplier)
  {
    damageMultiplier = multiplier;
  }
```

```
    public void SetFireRate(float newFireRate)
    {
        fireRate = newFireRate;
    }

}using UnityEngine;
using UnityEngine.UI;

public class Enemy : MonoBehaviour
{
    public float maxHealth = 100f;
    private float currentHealth;
    public float damage = 1f;
    public float moneyReward = 500f;
    public ShopManager shopManager;

    public float detectionRadius = 5f;
    public float moveSpeed = 2f;
    private Transform player;
    Rigidbody2D rb;

    void Start()
    {
        currentHealth = maxHealth;
        player = GameObject.FindGameObjectWithTag("Player")?.transform;
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (player != null)
        {
            float distanceToPlayer = Vector2.Distance(transform.position, player.position);

            if (distanceToPlayer <= detectionRadius)
            {
                MoveTowardsPlayer();
            }
        }
    }

    void MoveTowardsPlayer()
    {
        Vector2 direction = (player.position - transform.position).normalized;
        rb.MovePosition(rb.position + direction * moveSpeed * Time.fixedDeltaTime);


        float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.Euler(0, 0, angle);
    }

    public void TakeDamage(float damage)
    {
        currentHealth -= damage;

        if (currentHealth < 0)
        {
            currentHealth = 0;
        }

        if (currentHealth <= 0)
        {
            Die();
        }
    }

    void Die()
    {
        Debug.Log("Enemy died!");


        shopManager.AddMoney(moneyReward);

        Destroy(gameObject);
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {

collision.gameObject.GetComponent<PlayerHealth>().TakeDamage(damage);
        }
    }
}
```