ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра інформаційних технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Бакалавр»

«Розробка відеогри на основі рушія Unity»

(тема кваліфікаційної роботи українською мовою)

«Development of a Video Game Using the Unity Engine»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання

спеціальності 122 Комп'ютерні науки

(код, назва спеціальності)

Освітня програма Комп'ютерні науки

(назва)

Ковальов Станіслав Володимирович

(прізвище, ім'я, по-батькові здобувача)

Керівник к.т.н., доцент Фразе-Фразенко О.О. (підпис)

(науковий ступінь, вчене звання, прізвище, ініціали)

Рецензент к.т.н., доцент Щербіна Ю.В.

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій .
№ 1 від 09 червня 2024 р.

Завідувачка кафедри

(підпис) КАЗАКОВА Надія .

(прізвище,ім'я)

Захищено на засіданні ЕК № 13 .
протокол № 17 від 20 червня 2024 р.

Оцінка ____ / F / ____.

(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

(підпис) КОПИЧЕНКО Іван .

(прізвище,ім'я)

Одеса 2024

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 5
ВСТУП 6
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ 8
1.1 Дослідження предметної області 8
1.2 Огляд популярних 2D відеоігор в жанрі RPG 14
1.3 Аналіз програмного середовища розробки відеоігри 20
2 ВИБІР АРХІТЕКТУРИ СИСТЕМИ ТА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ 26
2.1 Обґрунтування вибору рушія розробки 26
2.2 Вибір мови програмування 28
2.3 Додаткові засоби для розробки проекту 31
2.4 Жанр Role-Playing Game (RPG) 33
3 ПРОЕКТУВАННЯ ІГРОВОГО ДОДАТКУ 36
3.1 Розробка концепції гри 36
3.2 Планування етапів розробки 38
3.3 Проектування ґеймплею 41
3.4 Архітектура системи і ієрархія класів 45
4 РОЗРОБКА ТЕХНІЧНОГО ПРОЕКТУ 49
4.1 Реалізація механік гри 49
4.2 Збір гри 59
4.3 Тестування гри 60
ВИСНОВКИ 61
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 62

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Unity – рушій для розробки відеоігор.

VS – Visual Studio – IDE – інтегроване середовище розробки.

RPG – Role-Playing Games – рольові ігри.

2D – двовекторний стиль відображення.

3D – тривекторний стиль відображення.

ІВ — інтелектуальна власність.

ЦА — цільова аудиторія.

NPC – non-playable characters – неграбельні персонажі.

Інді, A, AA, AAA — рівні масштабованості фінансування проекту.

VR – віртуальна реальність.

ООП — об'єктно-орієнтоване програмування.

ВСТУП

Сучасні комп'ютерні ігри стали невід'ємною частиною розважальної індустрії, привертаючи мільйони користувачів по всьому світу. Серед різних жанрів ігор особливе місце займають рольові ігри (RPG – Role-Playing Games), які дозволяють гравцям зануритися у фантастичні світи, взяти на себе роль героїв та виконувати різноманітні завдання. Створення RPG гри є складним і багатогранним процесом, який вимагає поєднання знань з різних галузей, включаючи програмування, дизайн, звукорежисуру та сценаристику.

Основною метою даного проекту є розробка керування персонажем та інтерактивності ігрового процесу у 2D RPG грі з використанням рушія Unity. У рамках цього проекту передбачається реалізація кількох ключових аспектів:

– Розробка керування персонажем — забезпечення плавного та інтуїтивного управління головним героєм гри. Це включає рух, ривки, атаки та інші дії, які гравець може виконувати, використовуючи клавіатуру.
– Пропрацювання взаємодії об'єктів — реалізація взаємодії між різними об'єктами в світі, наприклад, зіштовхування персонажа зі стіною чи зустріч персонажів один з одним.
– Розробка штучного інтелекту (AI) — створення ворогів з різними типами поведінки, які будуть реагувати на дії гравця, атакувати його чи триматись на відстані.

Проект розробки 2D RPG гри охоплює кілька етапів, кожен з яких має свої особливості та виклики. На першому етапі необхідно провести аналіз предметної області, визначити основні вимоги до гри, розробити концепцію та створити детальний план реалізації. Наступний етап включає вибір та налаштування інструментів, необхідних для розробки, таких як мова програмування C# та середовище розробки Visual Studio. Особливу увагу потрібно приділити розробці сценарію гри, створенню персонажів, визначенню їхніх характеристик та взаємодії з навколишнім середовищем.

Розробка графічного контенту є одним з ключових аспектів проекту. Використання 2D графіки дозволяє створити унікальний стиль гри, який відрізняється від сучасних 3D проектів. Це вимагає тісної співпраці між програмістами та дизайнерами, щоб забезпечити гармонійне поєднання візуальних ефектів і функціональності. Гра повинна бути не лише візуально привабливою, але й зручною для користувача, тому важливо ретельно опрацювати інтерфейс та систему управління.

Тестування та оптимізація є завершальними етапами розробки гри. Важливо провести детальне тестування для виявлення та усунення можливих помилок, забезпечити стабільну роботу гри на різних платформах. Оптимізація продуктивності дозволить знизити вимоги до апаратного забезпечення та забезпечити комфортну гру на різних пристроях.

Таким чином, розробка 2D RPG гри на рушії Unity є комплексним завданням, яке вимагає глибоких знань та навичок у різних галузях. Цей проект дозволяє не лише створити захоплюючу гру, але й набути цінного досвіду у розробці комп'ютерних ігор, який стане корисним для подальшої професійної діяльності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ

Для розуміння комп'ютерних ігор важливо спершу визначити, що таке ігри взагалі. Гра – це форма інтерактивних розваг, у якій гравець активно бере участь у процесі, взаємодіючи з ігровим середовищем та іншими учасниками. Саме завдяки цій інтерактивності гравці можуть відчути широкий спектр емоцій та отримати унікальний досвід, який неможливо здобути з інших видів медіа. Ігри пропонують не лише розвагу, але й можливість навчатися, розвивати навички та випробовувати різноманітні сценарії, що робить їх особливо цінними та захопливими.

1.1 Дослідження предметної області

У сьогоденні ігрова індустрія, як індустрія розваг, міцно закріпилася в повсякденному житті людей, а ринок ігрової індустрії зараз має неабиякі бюджети та досить близько підібрався до розмаху ринку індустрії кіно. За останні роки глобальний ринок ігор демонстрував стійке зростання. Згідно з останнім звітом Newzoo, доходи від продажів ігор у 2023 році досягли $184 мільярдів, і прогнозується, що вони зростуть до $189 мільярдів у 2024 році. Це вказує на постійне збільшення доходів, хоча темпи зростання сповільнилися порівняно з попередніми роками.[1]

Протягом 2023 року ринок почав відновлюватися після пандемії COVID-19, що також вплинуло на стратегії компаній у галузі. Багато з них змістили акцент на продовження існуючих проектів і використання вже відомих інтелектуальних власностей, уникаючи ризику інвестування в нові великі проекти. Важливу роль у цьому відіграють такі тайтли, як Assassin’s Creed Mirage та Marvel’s Spider-Man: Miles Morales, Resident Evil й інші.[2]

Отже, на фоні стійкого зростання доходів ігрового ринку, компанії адаптуються до нових умов, зосереджуючись на стратегіях, які мінімізують ризики, і на популярних франшизах, що вже зарекомендували себе серед

гравців. Проте варто відмітити, що незважаючи на вектор ігрових компаній, який спрямований на розробку продовжень вже існуючої інтелектуальної власності (ІВ), все ж великі ігрові компанії де-інколи виділяють ресурси на створення нових ІВ, які можуть бути як великими й амбітними, так і малими та скромними.

Пропоную розглянути основні терміни і поняття з області 2D комп'ютерних ігор.

Комп'ютерна гра (відеогра) — це електронна гра, яка відтворюється на комп'ютері, ігровій консолі або іншому електронному пристрої. Відеогри включають зображення, які реагують на дії гравця через натискання клавіш, миші або взаємодію з контролерами. Вони можуть бути однокористувацькими або багатокористувацькими, що дозволяє гравцям взаємодіяти один з одним через мережу. Комп'ютерні ігри варіюються від простих пазлів до складних симуляцій та стратегій, пропонуючи широкий спектр жанрів та стилів.

2D гра — це тип комп'ютерної гри, в якій візуальний простір складається з двовимірних об'єктів. У таких іграх графіка представлена у вигляді плоских зображень, що відображаються у двох вимірах — висоті та ширині. 2D ігри можуть бути як простими, так і складними, пропонуючи гравцям унікальні виклики та досвід. Зазвичай ці ігри характеризуються стилізованою графікою і часто використовуються для платформерів, головоломок та аркадних ігор.

Жанр — це категорія творів мистецтва, в яку відносять певні ігри на основі їх сюжетних і стилістичних ознак. Кожен жанр має свої унікальні правила та механіки, які визначають його особливості. Наприклад, жанр рольових ігор (RPG) фокусується на розвитку персонажів та сюжеті, тоді як жанр Shooter зосереджується на бойових діях та точності стрільби. Також деякі ігри можуть створювати окремі піджанри чи створювати їх, наприклад, існує піджанр Souls-like. Глобально Souls-like ігри відносяться до таких жанрів як Action і RPG, проте вони мають певні шаблони, завдяки яким вони мають певну схожість до ігор Хідетаки Міядзакі — Demon's Souls і Dark Souls (рис. 1.1), від яких і пішла назва піджанру.

Рисунок 1.1 — Скріншот екрану з гри Dark Souls.

Також є і новіші приклади піджанрів, так у 2023-му році світ побачила гра The Lord of The Rings: Gollum (TLoTR: Gollum), Скріншот з якої зображено на рисунку 1.2, в якій усе було дуже жахливо, гравці та критики не могли за щось зачепитись, що похвалити гру, не було за що хвалити, а через певний час після її релізу стало відомо й про інші ігри, які вийшли незабаром після неї, в яких теж було жахливо абсолютно все, тому гравці вирішили відносити такі ігри до новоствореного жартівливого піджанру Gollum-like, який відповідно отримав свою назву від TLoTR: Gollum.

Сетинг — це час і місце, в рамках яких відбувається дія гри. Сетинг визначає атмосферу та стиль гри, впливає на дизайн рівнів, персонажів та історію. Наприклад, гра може відбуватися у фантастичному середньовічному світі, у футуристичному космосі або в сучасному мегаполісі. Сетинг допомагає занурити гравця в ігровий світ і створює контекст для подій, що відбуваються у грі.

Рисунок 1.2 — Скріншот екрану з гри The Lord of The Ring: Gollum.

Розробка відеогри — це процес створення ігор, який може бути виконаний як окремою людиною, так і командою спеціалістів. Цей процес включає кілька ключових етапів: створення концепції, розробка дизайну, програмування, тестування та випуск гри. Розмір команди розробників залежить від складності проекту та його бюджету. Великі ігрові проекти часто потребують участі різних фахівців, таких як ґеймдизайнери, художники, програмісти, тестувальники, звукові інженери та менеджери проектів. Ці фахівці співпрацюють для створення кінцевого продукту, забезпечуючи його функціональність, естетику та якість.

Основні поняття, включно з розробкою відеогри, розглянуто, тому варто розглянути етапи розробки.

Перший етап розробки — ініціація. На етапі ініціації визначається концепція майбутньої відеогри. Команда розробників аналізує ринкові можливості та ідентифікує бізнес-проблему або нішу, яку цей проект може вирішити або заповнити. Встановлюються стратегічні цілі проекту та область його застосування, такі як жанр гри, цільова аудиторія, платформи для випуску, наприклад ПК, консолі, мобільні пристрої, а також ідеї щодо монетизації. Здійснюються перші оцінки ризиків і можливостей, щоб зрозуміти потенційні

перешкоди та переваги, які можуть вплинути на успіх проекту. Також визначаються основні ресурси, необхідні для реалізації проекту, такі як фінансування, людські ресурси, технічна інфраструктура тощо.

Наступний етап — планування. На етапі планування розробляється детальний стратегічний і тактичний план дій. Визначається більш детальний опис завдань, які потрібно виконати для досягнення цілей проекту. Це включає встановлення конкретних задач, призначення відповідальних осіб та встановлення часових рамок для кожного етапу розробки. Проводяться детальні аналізи ризиків, щоб ідентифікувати потенційні проблеми та розробити стратегії їх управління. Також встановлюються механізми контролю якості, які дозволяють забезпечити високу якість кінцевого продукту.

Третій етап — виконання. На етапі виконання реалізується стратегічний план проекту. Кожен член команди розробників виконує свої завдання згідно з визначеним раніше графіком і бюджетом. Художники створюють мистецькі активи, такі як персонажі, оточення, текстури і анімації. Програмісти розробляють код для ігрових механік, систем управління, штучного інтелекту та взаємодії з користувачем. Дизайнери розробляють ґеймплейні механіки, рівні гри, завдання та випробування. Все це відбувається під постійним моніторингом прогресу та звітністю про стан виконання робіт.

Передостанній етап — контроль і моніторинг. Під час цього етапу здійснюється систематичний контроль за прогресом проекту. Вирішуються виниклі проблеми і ризики, що можуть вплинути на успішне завершення проекту. Проводяться регулярні зустрічі з командою проекту для вирішення питань і внесення необхідних корекцій у план. Це дозволяє уникнути затримок і відповісти на зміни у вимогах чи умовах ринку.

І останній етап — завершення. На етапі завершення проекту оцінюється його результативність і відповідність встановленим цілям і вимогам. Проводиться аналіз досягнень, щоб з'ясувати, наскільки успішно були досягнуті поставлені цілі і завдання проекту. Формулюється звіт про проект і його результати, який включає в себе вивчення зібраного досвіду та знань для

майбутніх проектів. Оцінюються сильні і слабкі сторони проекту, а також ідентифікуються можливі шляхи покращення у майбутніх ітераціях або інших проектах. Під час етапу завершення вирішується чи буде випускатись проект на ринок, у випадку ігор, чи готові вони до релізу, чи треба відстрочити час і попрацювати над поліруванням і доробкою гри.

Однією з дуже важливих частин розробки гри, яка вийшла на ринок, є її підтримка й оновлення. Зазвичай, для великих проектів справедливе твердження, що навіть великі тестувальні команди та альфа-тестування, закриті тестування, участь у яких можуть прийняти користувачі за запрошенням розробників, й, навіть, бета-тестування можуть не помітити певні недоліки гри, які потенційно є дуже критичними, і ці недоліки можуть виникнути у кінцевих користувачів. Не випускати гру, для того щоб її доробити на всі 100% досить часто є недоцільним, адже для великих студій це втрата великої суми грошей, часу та потенційна втрата інтересу з боку цільової аудиторії (ЦА), що може призвести до того, що проект не окупиться; а для малих студій це також може бути втратою грошей, часу й інтересу з боку ЦА, проте, на відміну від великих студій, малі ризикують власним закриттям. Тому, щоб не втрачати забагато ресурсів, здебільшого, проекти доводять до певного стану, коли помилок і недоліків, особливо критичних, мінімальна кількість і випускають продукт, а вже під час його експлуатації користувачами розробники роблять оновлення для продукту в якому виправляють помилки.

Кінцем життєвого циклу будь-якого проекту є момент, коли його перестають підтримувати розробники й експлуатувати користувачі. Для відеоігор це також є справедливим, проте на відміну від багатьох інших проектів відеоігри можуть мати неймовірно довгий період експлуатації, не в останню чергу завдяки зацікавленості аудиторією й певним механізмам зберігання. Раніше, коли інтернет ще не був такий розповсюджений, а інтернет-магазинів і подібних веб-сервісів не існувало, відеоігри розповсюджувались шляхом записування на певні носії, накшталт дискет, картриджів або дисків, такий підхід давав розробникам можливість продавати

свої ігри, а користувачам грати в них на своїх пристроях, також такий підхід добре підходить для зберігання відеоігор, адже він зберігання залежить лишень від умов зберігання носія, що при правильній експлуатації призводить до того, що навіть зараз у багатьох людей є робочі носії та пристрої з іграми, які було розроблені у 20-му столітті. У сьогоденні ж відеоігри здебільшого розробляються для розповсюдження шляхом електронних, а не фізичних копій, такий підхід дозволяє не витрачати ресурси на розробку носіїв і записування проекту на них, також завдяки цьому підходу відеоіграми можуть користуватись навіть ті користувачі, які не мають необхідного апаратного забезпечення, наприклад дисководу для зчитування інформації з компакт-дисків. Проте розповсюдження цифрових копій має певні суттєві недоліки, такі як перекладання відповідальності зберігання на третіх осіб або виділення апаратних і людських ресурсів для зберігання цих ігор компаніями-розробниками, що за несправності серверів може призвести до неможливості користувачем отримання доступу до продукту, який він придбав, також цей спосіб дозволяє розробникам диктувати особливі правила користування й позбавляти користувачів їх копії гри за найабсурднішими причинами.

1.2 Огляд популярних 2D відеоігор в жанрі RPG

2D відеоігри в жанрі RPG (role-playing games) мають багату історію і особливий шарм, який приваблює гравців по всьому світу. Незважаючи на розвиток тривимірної графіки і складних ігрових механік, 2D RPG зберігають свою популярність завдяки унікальній атмосфері, деталізованим світам і глибокому сюжету. Ці ігри часто поєднують у собі ретро-естетику з сучасними ігровими елементами, що робить їх привабливими як для ветеранів жанру, так і для новачків. У цьому підрозділі ми розглянемо п’ять популярних 2D RPG, таких як Blasphemous, Enter the Gungeon, Dead Cells, Hollow Knight і Hyper

Light Drifter, кожна з яких має свої унікальні особливості і внесок у розвиток жанру.

Blasphemous — це похмура і атмосферна 2D RPG, розроблена студією The Game Kitchen. Гра відрізняється своєю готичною естетикою, натхненною іспанським фольклором і католицькими ритуалами. Гравці беруть на себе роль Покаяного, останнього живого представника Братства Мовчазного Смутку, і вирушають у жорстоку подорож через прокляті землі Кустодії. Візуальний стиль гри, ретельно промальовані піксель-арт спрайти і складна, але захоплююча бойова система роблять Blasphemous унікальною серед інших 2D RPG. Гравцям доведеться стикнутися з численними босами, вирішувати головоломки і досліджувати розгалужені локації, щоб розкрити таємниці цього темного світу (рис. 1.3).

Рисунок 1.3 — Скріншот екрану з гри Blasphemous.

Blasphemous виділяється своєю атмосферою, яка створюється за допомогою детально опрацьованих візуальних елементів і глибокого звукового супроводу. Гра наповнена символізмом і релігійними мотивами, які пронизують кожен аспект — від дизайну персонажів до структури світу. Покаяний, озброєний своєю мечоподібною зброєю, подорожує через світи, що нагадують середньовічні релігійні картини, зустрічаючись із жорстокими

іспитами і босами, які вимагають від гравця точності і стратегічного мислення. Сюжет гри розкривається поступово, через зустрічі з різними NPC, знахідки та таємничі записи, які розповідають про історію і прокляття Кустодії. Бойова система включає в себе численні комбінації атак, ухилень та блоків, що створює глибоку і складну механіку, яка винагороджує майстерність і терпіння.

Enter the Gungeon — це рогалик-стрілялка з елементами RPG, розроблена студією Dodge Roll. Гра вирізняється швидким темпом і інтенсивними перестрілками, де гравці керують одним з чотирьох мисливців, які намагаються знайти і використовувати легендарну зброю, здатну змінити минуле. Кожен з мисливців має свої унікальні здібності і історію, що додає глибини грі. Enter the Gungeon пропонує процедурно генеровані рівні, насичені ворогами, пастками і скарбами, що забезпечує високу реіграбельність і непередбачуваність кожної сесії. Яскравий піксель-арт, різноманітний арсенал і захоплююча механіка роблять гру видатним представником жанру (рис. 1.4).

Рисунок 1.4 — Скріншот екрану з гри Enter The Gungeon.

Основна привабливість Enter the Gungeon полягає в її динамічному і хаотичному ігровому процесі. Гравці повинні маневрувати через безліч

ворогів, ухиляючись від куль і використовуючи різноманітні види зброї, кожна з яких має свої унікальні властивості. Крім основної зброї, гравці можуть знаходити і використовувати різні активні та пасивні предмети, які змінюють ігровий процес і додають нові стратегії для виживання. Гра також пропонує кооперативний режим, що дозволяє гравцям об'єднувати зусилля для подолання викликів підземелля. Система прогресу в грі винагороджує за дослідження і ризик, дозволяючи гравцям відкривати нові зброї, персонажів і секрети, що збагачують ігровий досвід.

Dead Cells — це динамічна 2D RPG з елементами метроїдванії і roguelike, розроблена Motion Twin. Гравці беруть на себе роль аморфного створіння, яке захоплює тіла мертвих і вирушає у небезпечну подорож через постійно змінювані рівні. Гра відзначається плавною і швидкою бойовою системою, яка дозволяє гравцям використовувати широкий спектр зброї, здібностей і предметів. Dead Cells пропонує інтенсивний екшн з високим рівнем складності, де кожна смерть означає початок нового проходження з іншими умовами. Візуально гра вражає своїм яскравим і деталізованим піксель-артом, а також чудово опрацьованою анімацією (рис. 1.5).

Рисунок 1.5 — Скріншот екрану з гри Dead Cells.

Однією з ключових особливостей Dead Cells є її нелінійна структура рівнів, що дозволяє гравцям досліджувати світ у різних напрямках. Це створює відчуття відкритості і свободи вибору, що є важливим елементом гри. Кожне нове проходження приносить нові виклики і можливості для відкриття, завдяки процедурній генерації рівнів і випадковому розподілу ворогів та предметів. Бойова система гри вимагає від гравців точності і швидкої реакції, а також вміння ефективно використовувати різні види зброї і здібностей. Крім того, Dead Cells постійно отримує оновлення і новий контент, що підтримує інтерес гравців і додає нові елементи ґеймплею. Гра також має унікальну систему прогресу, де гравці можуть відкривати і вдосконалювати постійні поліпшення, що робить кожне нове проходження трохи легшим і цікавішим.

Hyper Light Drifter — це 2D екшн-RPG, розроблена студією Heart Machine. Гра відзначається своїм стилізованим піксель-артом і атмосферною музикою, створеною композитором Disasterpeace. Гравці керують Дрифтером, який досліджує постапокаліптичний світ, знищений стародавньою цивілізацією. Бойова система Hyper Light Drifter поєднує в собі швидкі атаки ближнього бою і стрільбу з різної зброї, що вимагає від гравців точності і тактичного мислення. Гра пропонує нелінійний підхід до проходження, де гравці можуть вільно досліджувати різні області, знаходити приховані секрети і вирішувати складні головоломки (рис. 1.6).

Hyper Light Drifter відзначається своєю унікальною естетикою і глибокою символікою, що надає грі особливої атмосфери. Сюжет гри подається через візуальні елементи і зустрічі з NPC, що дозволяє гравцям самостійно інтерпретувати події і історію світу. Гра також включає елементи виживання і дослідження, де кожне відкриття і здобутий предмет можуть бути ключем до подальшого прогресу. Система прогресу в грі дозволяє гравцям покращувати свої навички і зброю, що додає додатковий шар стратегічного планування і мотивації для дослідження. Hyper Light Drifter отримала високу оцінку за свій унікальний стиль, захоплюючий ґеймплей і емоційну глибину, що робить її видатним представником 2D RPG.

Рисунок 1.6 — Скріншот екрану з гри Hyper Light Drifter.

Hollow Knight — це метроїдванія з елементами RPG, створена студією Team Cherry. Гравці керують маленьким лицарем, який досліджує підземний світ Hallownest, зруйноване королівство, наповнене таємницями, небезпечними ворогами і складними головоломками. Hollow Knight відзначається своєю атмосферною графікою, натхненною ручною анімацією, і глибоким лором, який розкривається через зустрічі з NPC і вивчення світу. Бойова система гри включає в себе різні види зброї і здібностей, що дозволяють гравцям адаптувати свій стиль гри. Гра пропонує великий відкритий світ з численними секретами, альтернативними шляхами і випробуваннями, які роблять кожне проходження унікальним (рис. 1.7).

Рисунок 1.7 — Скріншот екрану з гри Hollow Knight.

Hollow Knight привертає увагу своєю глибокою і складною системою прогресу, яка дозволяє гравцям поступово відкривати нові здібності і доступ до нових областей світу. Ігровий процес нагадує класичні метроїдванії, де дослідження і битви переплітаються, створюючи насичену і захоплюючу пригоду. Кожен куток Hallownest наповнений історіями і секретами, що додає грі глибини і робить кожне відкриття значущим. Бос-битви в грі є справжніми випробуваннями, що вимагають від гравців точності і стратегії. Гра також пропонує численні побічні завдання і випробування, що додають додатковий шар до ігрового досвіду і дозволяють гравцям глибше зануритися у світ Hollow Knight. Музичний супровід, створений композитором Крістофером Ларкіним, підсилює атмосферу гри і додає емоційної глибини кожному моменту.

1.3 Аналіз програмного середовища розробки відеоігри

Розробка відеоігор є складним і багатогранним процесом, що вимагає використання різноманітного програмного забезпечення та інструментів.

Сучасні ігрові проекти можуть включати як інді-розробки, так і масштабні ААА-ігри, що потребують потужних і гнучких середовищ розробки. Програмне середовище розробки відеоігор охоплює різні аспекти, такі як графіка, звук, фізика, штучний інтелект, мережеві функції та інші ключові компоненти. Цей підрозділ детально розгляне основні інструменти та технології, які використовуються у розробці відеоігор, включаючи ігрові рушії, мови програмування, середовища інтегрованої розробки (IDE), системи контролю версій і додаткові інструменти.

Ігрові рушії є основним компонентом програмного середовища розробки відеоігор. Вони надають набір інструментів і бібліотек, що спрощують процес створення ігор, дозволяючи розробникам зосередитися на ігровій механіці та дизайні. Найпопулярнішими ігровими рушіями є Unity, Unreal Engine та Godot.

Unity є одним з найпопулярніших ігрових рушіїв у світі, та широким набором інструментів, які надає розробникам. Unity підтримує безліч платформ, включаючи ПК, консолі, мобільні пристрої та VR. Рушій використовує мову програмування C#, що є достатньо потужною та легкою для вивчення. Завдяки великій спільноті користувачів і безлічі ресурсів, таких як форуми, документація і відеоуроки, розробники можуть швидко вирішувати проблеми та знаходити відповіді на свої питання. Unity також пропонує Asset Store, де можна придбати або завантажити безкоштовно безліч ресурсів, таких як моделі, текстури, скрипти та інші компоненти, що значно прискорює розробку.

Unreal Engine, розроблений Epic Games, є ще одним популярним ігровим рушієм, особливо в сфері ААА-ігор. Unreal Engine відомий своїми потужними графічними можливостями, що дозволяють створювати високоякісні візуальні ефекти та реалістичні сцени. Рушій використовує мову програмування C++, що може бути складнішою у використанні порівняно з C#, але надає більш глибокий контроль над ігровими процесами. Крім того, Unreal Engine пропонує систему Blueprints, що дозволяє створювати ігрову логіку без необхідності написання коду, використовуючи візуальне програмування. Ця

функція робить рушій доступним для дизайнерів та художників, які можуть реалізувати свої ідеї без участі програмістів.

Godot є відкритим рушієм, що набуває все більшої популярності завдяки своїй гнучкості та безкоштовності. Godot підтримує як 2D, так і 3D-ігри, і використовує власну мову програмування GDScript, яка дуже схожа на Python. Рушій також підтримує C# та C++. Godot відзначається своєю простотою у використанні, ієрархічною системою сцен і широкими можливостями для користувацьких налаштувань. Відкритий код рушія дозволяє розробникам модифікувати його відповідно до своїх потреб, що особливо корисно для інді-розробників.

Мови програмування є основним інструментом для написання ігрової логіки та створення інтерактивних елементів у відеоіграх. Найпоширенішими мовами, які використовуються в ігровій розробці, є C#, C++, Python та JavaScript.

C# широко використовується в Unity завдяки своїй простоті та гнучкості. Мова підтримує об’єктно-орієнтоване програмування, що дозволяє створювати модульний і зрозумілий код. Завдяки великій спільноті розробників та обширній документації, розробники можуть легко знайти приклади коду та навчальні матеріали.

C++ є основною мовою програмування для Unreal Engine та інших високопродуктивних ігрових рушіїв. Вона забезпечує високий рівень контролю над системними ресурсами та ефективну роботу з пам’яттю, що є критичним для створення складних і реалістичних ігор. Водночас, C++ є складнішою мовою для вивчення і використання, що може стати викликом для новачків.

Python використовується в Godot через мову GDScript, яка має синтаксис, схожий на Python. Це робить її легкою для вивчення та використання, особливо для новачків. Python також використовується для створення різних інструментів і скриптів, що автоматизують процес розробки.

JavaScript використовується в веб-розробці ігор та для створення інтерактивних елементів у браузерах. Існує безліч бібліотек і фреймворків, таких як Phaser, які полегшують розробку 2D-ігор на JavaScript.

Середовища інтегрованої розробки є важливим компонентом програмного середовища розробки відеоігор, оскільки вони надають розробникам інструменти для написання, налагодження та тестування коду. Найпопулярнішими IDE для розробки ігор є Visual Studio, Rider та VS Code.

Visual Studio є потужним і популярним IDE, особливо серед розробників, які використовують C# та C++. Visual Studio підтримує безліч розширень і плагінів, що робить його дуже гнучким і налаштовуваним під потреби розробника. Інтеграція з Unity і Unreal Engine дозволяє швидко налаштувати середовище розробки та забезпечує зручний процес налагодження і компіляції.

Rider від JetBrains є ще одним популярним IDE для розробки на C#. Він відзначається високою продуктивністю та зручним інтерфейсом, що дозволяє ефективно писати та налагоджувати код. Rider також підтримує інтеграцію з Unity, що робить його чудовим вибором для розробників, які працюють з цим рушієм.

VS Code є легким і розширюваним редактором коду, який підтримує безліч мов програмування та фреймворків. Завдяки великій кількості плагінів і розширень, VS Code можна налаштувати для роботи з будь-яким ігровим рушієм та мовою програмування. Його простота та швидкість роблять його популярним серед розробників, які цінують мінімалізм і ефективність.

Системи контролю версій є критично важливими для управління кодом і співпраці в команді розробників. Вони дозволяють відслідковувати зміни в коді, відновлювати попередні версії та забезпечують координацію між різними членами команди. Найпоширенішою системою контролю версій є Git, яка використовується разом з платформами, такими як GitHub, GitLab та Bitbucket.

Git є децентралізованою системою контролю версій, яка дозволяє кожному розробнику мати повну копію історії проекту. Це забезпечує високу гнучкість і надійність, оскільки зміни можуть бути збережені локально і

синхронізовані з віддаленим репозиторієм пізніше. GitHub, GitLab та Bitbucket надають веб-інтерфейси для управління репозиторіями, спрощують співпрацю та інтегрують інші інструменти, такі як CI/CD, системи управління завданнями та автоматичне тестування.

Окрім основних компонентів, програмне середовище розробки відеоігор включає безліч додаткових інструментів, які полегшують різні аспекти розробки. Це можуть бути інструменти для створення графіки, звуку, анімації, тестування та управління проектами.

Blender є безкоштовним інструментом для створення 3D-графіки, що використовується для моделювання, текстурування та анімації. Blender підтримує експорт моделей у формати, сумісні з більшістю ігрових рушіїв, що робить його популярним серед інді-розробників і професійних студій.

Photoshop та GIMP використовуються для створення та редагування 2D-графіки, текстур та інтерфейсів. Photoshop є комерційним програмним забезпеченням, тоді як GIMP є безкоштовною альтернативою з подібним функціоналом.

FMOD та Wwise є інструментами для створення та інтеграції звукових ефектів і музики в ігри. Вони надають потужні можливості для роботи зі звуком, дозволяючи створювати динамічні і адаптивні аудіоефекти, які реагують на дії гравця та зміни в ігровому середовищі.

JIRA та Trello є популярними системами управління проектами, які дозволяють команді розробників організовувати завдання, відстежувати прогрес і співпрацювати над проектом. Вони надають інструменти для планування спринтів, розподілу завдань та моніторингу виконання робіт.

Automated Testing Tools забезпечують автоматизацію процесу тестування, що значно скорочує час на перевірку якості коду та виявлення помилок. Такі інструменти, як Selenium, JUnit та TestComplete, дозволяють створювати автоматизовані тести, які перевіряють функціональність і продуктивність гри на різних етапах розробки.

Cloud Services використовуються для зберігання даних, управління базами даних та розгортання серверних компонентів. Платформи, такі як AWS, Azure та Google Cloud, надають інфраструктуру та сервіси, що спрощують розробку та підтримку онлайн-ігор, забезпечуючи високу доступність і масштабованість.

Таким чином, програмне середовище розробки відеоігор є комплексним набором інструментів і технологій, що забезпечують ефективний процес створення ігор. Від вибору ігрового рушія та мови програмування до використання IDE, систем контролю версій і додаткових інструментів – кожен аспект цього середовища відіграє важливу роль у забезпеченні якості та успіху кінцевого продукту. Розробники повинні уважно підходити до вибору програмного забезпечення, орієнтуючись на специфіку свого проекту та наявні ресурси, щоб забезпечити максимально ефективний і продуктивний процес розробки.

2 ВИБІР АРХІТЕКТУРИ СИСТЕМИ ТА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

Вибір архітектури системи та програмних засобів для реалізації відеогри є критично важливим етапом у процесі розробки. Від цього вибору залежить якість, продуктивність і майбутня розширюваність гри. Правильно підібрана архітектура дозволяє ефективно втілити ідеї ґеймдизайнерів, забезпечити високий рівень користувацької якості та зручність утримання кодової бази. Розгляд цього підрозділу дозволить зрозуміти ключові аспекти вибору архітектури, технологій та інструментів, які грають важливу роль у створенні сучасних відеоігор. Успішна розробка відеоігри потребує комплексного підходу, що включає вибір правильного ігрового движка, інструментів для створення графіки, звукових редакторів та середовищ для тестування. Кожен з цих компонентів має свої унікальні особливості та переваги, які можуть суттєво вплинути на кінцевий продукт. Вибір оптимальної архітектури дозволяє не тільки реалізувати задумки розробників, але й забезпечити стабільність та масштабованість гри. Цей розділ детально розглядає процес вибору архітектури системи, аналізує різні програмні засоби та їхню сумісність з потребами конкретного проекту, що допоможе забезпечити високий рівень якості та продуктивності майбутньої відеогри.

2.1 Обґрунтування вибору рушія розробки

Обґрунтування вибору рушія розробки є одним з ключових етапів планування і реалізації проекту відеоігри. Рушій гри визначає основні технічні характеристики, можливості та обмеження, з якими доведеться працювати команді розробників. При виборі рушія важливо враховувати кілька факторів, таких як специфікація гри, рівень кваліфікації команди, вартість використання рушія, а також його продуктивність і можливості для майбутнього масштабування.

Розглянемо основні ігрові рушії, які часто використовуються в сучасній індустрії відеоігор: Unity, Unreal Engine, Godot, CryEngine та RPG Maker. Кожен з них має свої унікальні особливості, які роблять їх придатними для певних типів проектів.

Unity є одним з найпопулярніших ігрових рушіїв на ринку. Він відомий своєю доступністю, зручним інтерфейсом і широким спектром можливостей. Unity підтримує розробку для різних платформ, включаючи ПК, мобільні пристрої, консолі і VR. Цей рушій надає великий набір інструментів для створення 2D і 3D ігор, включаючи потужний фізичний движок, можливості анімації та інструменти для розробки користувацького інтерфейсу. Unity також має велику спільноту розробників, що забезпечує доступ до численних ресурсів, навчальних матеріалів і готових рішень. Використання Unity є відносно недорогим, оскільки він має безкоштовну версію для індивідуальних розробників та малих команд, а також гнучку систему ліцензування для комерційних проектів. [3]

Unreal Engine, створений компанією Epic Games, є ще одним потужним рушієм, особливо популярним серед розробників AAA-ігор. Unreal Engine відомий своєю високою продуктивністю та чудовими графічними можливостями. Він підтримує як 2D, так і 3D розробку, надаючи розробникам інструменти для створення високоякісних візуальних ефектів, реалістичної фізики та складних анімацій. Однак, Unreal Engine має більш складний інтерфейс і круту криву навчання, що може стати перешкодою для менших команд або новачків. Крім того, комерційне використання цього рушія може бути дорожчим через роялті-виплати з прибутків гри. [4]

Godot є безкоштовним рушієм з відкритим вихідним кодом, який останнім часом набуває популярності серед незалежних розробників. Godot підтримує 2D та 3D розробку, пропонуючи простий і зрозумілий інтерфейс. Його головною перевагою є відсутність роялті або ліцензійних платежів, що робить його привабливим для невеликих студій та інді-розробників. Проте, Godot ще не має такого великого ком'юніті і кількості ресурсів, як Unity або

Unreal Engine, що може обмежити доступ до навчальних матеріалів та готових рішень. [5]

CryEngine, розроблений компанією Crytek, відомий своєю високою продуктивністю та реалістичними графічними можливостями. Цей рушій підтримує розробку 3D-ігор з акцентом на деталізовані графічні ефекти та реалістичну фізику. CryEngine використовується в основному для створення великих проектів з високим бюджетом, оскільки він вимагає значних ресурсів та технічної експертизи. Незважаючи на свої переваги, CryEngine має складний інтерфейс і круту криву навчання, що може бути складним для новачків. [6]

RPG Maker є спеціалізованим рушієм для створення 2D RPG-ігор. Він відомий своєю простотою у використанні та зручними інструментами для створення сюжетів, персонажів і бойових систем. RPG Maker підходить для розробників, які хочуть швидко створити RPG-гру без глибоких технічних знань. Однак, його можливості обмежені порівняно з універсальними рушіями, такими як Unity або Unreal Engine. [7]

Після розгляду всіх цих варіантів, вибір Unity як основного рушія для розробки нашої гри є обґрунтованим рішенням. Unity поєднує в собі простоту використання, широкі можливості для 2D та 3D розробки, підтримку різних платформ та доступ до великої кількості ресурсів і навчальних матеріалів. Його потужний інструментарій дозволяє реалізувати складні ігрові механіки та високоякісну графіку, а активне ком'юніті розробників забезпечує підтримку і обмін знаннями. Крім того, гнучка система ліцензування Unity дозволяє почати розробку без значних початкових витрат, що є важливим для невеликих студій та інді-розробників. Таким чином, Unity забезпечує оптимальний баланс між можливостями, вартістю та зручністю використання, що робить його ідеальним вибором для нашого проекту.

2.2 Вибір мови програмування

Вибір мови програмування є одним із найважливіших рішень на початковому етапі розробки відеоігри. Мова програмування визначає, як саме

буде реалізована логіка гри, як розробники будуть взаємодіяти з ігровим рушієм та як будуть створюватись ігрові механіки. Вибір мови залежить від багатьох факторів, включаючи технічні можливості мови, її продуктивність, легкість вивчення та використання, підтримку спільноти розробників, а також інтеграцію з обраним ігровим рушієм.

Серед найпопулярніших мов програмування для розробки відеоігор слід зазначити C++, C#, Java, Python, Lua та JavaScript. Кожна з них має свої переваги та недоліки, які варто розглянути детальніше.

C++ є однією з найпоширеніших мов програмування в ігровій індустрії, особливо для розробки AAA-ігор. Вона надає високу продуктивність та гнучкість, що дозволяє розробникам контролювати апаратні ресурси на низькому рівні. C++ використовується в багатьох відомих ігрових рушіях, таких як Unreal Engine і CryEngine. Однак, ця мова має складний синтаксис і круту криву навчання, що може бути складністю для новачків. Крім того, розробка на C++ може займати більше часу через необхідність управління пам'яттю та іншим низькорівневим аспектам.

Java є універсальною мовою програмування, яка відома своєю портативністю і можливістю працювати на різних платформах завдяки використанню Java Virtual Machine (JVM). Вона часто використовується для розробки мобільних ігор на платформі Android. Java забезпечує хорошу продуктивність і зручний синтаксис, однак може бути менш ефективною порівняно з C++ при розробці складних 3D-ігор з високими вимогами до графіки.

Python є популярною мовою програмування завдяки своїй простоті та легкості вивчення. Вона часто використовується для швидкого прототипування та розробки інді-ігор. Python надає безліч бібліотек та фреймворків, таких як Pygame, які полегшують розробку ігор. Проте, Python має обмежену продуктивність порівняно з C++ та іншими мовами, що може бути недоліком для розробки великих та складних ігор.

Lua є легкою і швидкою мовою програмування, яка часто використовується для написання скриптів у відеоіграх. Вона інтегрується з багатьма ігровими рушіями, такими як Unity і CryEngine, та дозволяє розробникам швидко і легко додавати новий функціонал до гри. Lua має простий синтаксис і високу продуктивність, однак її можливості можуть бути обмежені порівняно з більш універсальними мовами, такими як C++ чи C#.

JavaScript є однією з основних мов для розробки веб-ігор завдяки своїй інтеграції з HTML5 та браузерами. Вона забезпечує високу продуктивність і підтримку численних бібліотек і фреймворків для розробки ігор, таких як Phaser та Three.js. JavaScript є чудовим вибором для розробки кросплатформених веб-ігор, однак може бути менш ефективною для розробки великих та складних 3D-ігор.

Оскільки в якості ігрового рушія для нашого проекту обрано Unity, важливо врахувати, які мови програмування він підтримує. Unity переважно використовує C# як основну мову програмування для написання скриптів та реалізації ігрової логіки. C# є високорівневою мовою програмування, розробленою корпорацією Microsoft, яка надає зручний і зрозумілий синтаксис, а також високий рівень продуктивності. Вона поєднує в собі легкість вивчення та використання з потужними можливостями, що робить її ідеальною для розробки ігор.

Однією з ключових переваг C# є його інтеграція з Unity, що дозволяє розробникам легко створювати ігрові механіки, управляти об'єктами та обробляти події. Unity надає потужний редактор та інструменти для роботи з C#, що значно спрощує процес розробки. Крім того, C# має велику спільноту розробників та численні ресурси, що дозволяє швидко знаходити відповіді на питання та отримувати підтримку. [9]

Вибір C# як мови програмування для нашого проекту на рушії Unity є оптимальним рішенням, оскільки вона поєднує в собі простоту використання, високу продуктивність та потужні можливості для розробки ігор.

Використання C# дозволяє максимально ефективно використовувати всі переваги Unity та забезпечує зручність та гнучкість в процесі розробки.

2.3 Додаткові засоби для розробки проекту

Для успішної розробки відеоігри на рушії Unity необхідно обрати відповідні додаткові засоби та інструменти, які допоможуть оптимізувати процес розробки та забезпечити високу якість кінцевого продукту. Основним з таких інструментів є середовище розробки (IDE), яке надає розробникам зручний інтерфейс для написання, тестування та налагодження коду. Серед багатьох доступних середовищ розробки, сумісних з Unity, розглянемо найбільш популярні та зосередимося на Visual Studio, яке було обрано для нашого проекту.

Unity підтримує різні середовища розробки, такі як Visual Studio, Visual Studio Code, JetBrains Rider та MonoDevelop. Кожне з цих середовищ має свої переваги та недоліки, які слід враховувати при виборі інструменту для розробки. Visual Studio є найбільш розповсюдженим та потужним середовищем розробки, яке пропонує широкий спектр функціональності для розробників, зокрема для тих, хто працює з Unity. [10]

Visual Studio Code, полегшена версія Visual Studio, є популярним середовищем розробки завдяки своїй легкості та швидкості. Воно забезпечує базові функції для написання коду, підтримує розширення та налаштування, що робить його зручним для розробки невеликих проектів. Однак, для великих проектів, таких як наша відеогра, Visual Studio Code може бути недостатньо потужним, оскільки не надає всіх необхідних інструментів для складної розробки та налагодження. [12]

JetBrains Rider є ще одним потужним середовищем розробки, яке підтримує Unity. Rider відомий своєю інтеграцією з іншими продуктами JetBrains та надає розробникам широкий спектр інструментів для написання та

налагодження коду. Хоча Rider є потужним інструментом, він вимагає платної підписки, що може бути недоліком для розробників з обмеженим бюджетом.

MonoDevelop, попереднє стандартне середовище розробки для Unity, також підтримує роботу з рушієм. Однак, MonoDevelop поступово втрачає популярність через обмежену функціональність та повільний розвиток порівняно з іншими середовищами.

Основною причиною вибору Visual Studio для нашого проекту є його потужність, надійність та глибока інтеграція з Unity. Visual Studio надає розробникам розширені можливості для написання, налагодження та тестування коду, що значно полегшує процес розробки складних проектів. Крім того, Visual Studio підтримує безліч розширень, які можна використовувати для покращення функціональності та зручності роботи.

Visual Studio забезпечує потужні інструменти для налагодження коду, що дозволяє швидко знаходити та виправляти помилки. Завдяки інтеграції з Unity, розробники можуть використовувати відладчик Visual Studio для налагодження ігрового коду безпосередньо в редакторі Unity. Це значно скорочує час на виявлення та виправлення помилок, що є критично важливим для підтримання високої якості проекту.

Крім того, Visual Studio надає інструменти для управління версіями коду, що дозволяє розробникам ефективно співпрацювати в команді. Інтеграція з системами контролю версій, такими як Git та Subversion, забезпечує зручне зберігання та відстеження змін у коді. Це особливо важливо для великих проектів, де необхідно координувати роботу кількох розробників.

Visual Studio також надає потужні інструменти для аналізу та оптимізації коду. Інтегровані інструменти для профілювання продуктивності допомагають виявити вузькі місця в коді та оптимізувати його для покращення продуктивності гри. Це особливо важливо для розробки відеоігор, де продуктивність може мати вирішальне значення для успішного запуску проекту.

Однією з ключових переваг Visual Studio є велика спільнота розробників та широкий спектр доступних ресурсів для навчання та підтримки. Документація, форуми та інші ресурси допомагають розробникам швидко знаходити відповіді на питання та отримувати підтримку від спільноти. Це робить Visual Studio ідеальним вибором для розробників різного рівня досвіду, від новачків до професіоналів.

З огляду на всі ці фактори, вибір Visual Studio для нашого проекту є оптимальним рішенням. Це середовище розробки забезпечує всі необхідні інструменти для успішної розробки, налагодження та тестування відеоігри, що дозволяє нам зосередитись на створенні високоякісного продукту. Visual Studio поєднує в собі потужність, гнучкість та зручність використання, що робить його найкращим вибором для нашого проекту на рушії Unity.

2.4 Жанр Role-Playing Game (RPG)

Жанр RPG (Role-Playing Game, або ігри зі створенням ролей) є одним з найбільш універсальних і популярних серед геймерської аудиторії, завдяки своїй здатності поглиблювати гравців у фантастичні світи і даючи можливість керувати розвитком персонажів через вибори та взаємодію з ігровим середовищем. Цей жанр знайшов широкий спектр застосувань від класичних медієвальних фентезі історій до науково-фантастичних саг та історичних драм. Завдяки різноманітності піджанрів і стилів, RPG мають змогу задовольнити смаки найрізноманітніших гравців, від тих, хто полюбляє прості та захопливі історії, до тих, хто шукає складні та багатогранні пригоди.

Основними відмінностями RPG є:

- Епічний наратив і глибокий сюжет: Однією з ключових особливостей RPG є можливість глибокого погруження гравця в історію та світ ігри. Від класичних квестів до складних політичних інтриг, RPG пропонують гравцям можливість впливати на події та розвиток сюжету своїми власними виборами. Гравці можуть часто зустрічати моменти, де їхні

рішення мають значні наслідки, що впливають на долю персонажів та світів, які вони досліджують.

- Система прокачки персонажів і розвиток: Гравці можуть налаштовувати своїх персонажів відповідно до своїх уподобань і ігрових стратегій. Від вибору спеціальностей і вмінь до апгрейдів та зброї, система прокачки дозволяє створити унікального героя, який відповідає стилю гри гравця. Це включає розвиток навичок, вивчення нових здібностей та здобуття рідкісного спорядження, що додає глибини та індивідуальності кожному ігровому досвіду.
- Багатогранність ігрового світу: RPG відомі своїми різноманітними світами, що заповнені історіями, легендами та культурними особливостями. Від відкритих світів з безліччю завдань і секретів до обмежених, але деталізованих локацій, цей жанр пропонує безмежні можливості для дослідження і відкриття. Світ RPG може включати різні нації, фракції та соціальні структури, що робить його живим і динамічним.
- Тактичні бої та стратегічні вибори: Великий акцент у RPG робиться на стратегічних аспектах бою та виборів. Гравці повинні розраховувати свої дії, використовувати слабкі сторони ворогів і вміле поєднання вмінь персонажів для досягнення перемоги. Це може включати покрокові битви, де кожен хід має важливе значення, або реальні часом бої, що вимагають швидкого мислення і реакцій.
- Соціальні взаємодії і вплив гравців: Деякі RPG пропонують можливості для соціальних взаємодій між персонажами, включаючи вибори і вплив на ігровий світ. Це може включати відносини з іншими NPC, політичні союзи або ворожість, що має велике значення для ходу історії. Взаємодія з іншими персонажами може відкривати нові квести, змінювати ставлення фракцій до гравця і навіть впливати на кінцівку гри.
- Іммерсивність і атмосфера: RPG часто пропонують глибоку іммерсію в ігрові світи через деталізацію середовища, музику і звукові ефекти, які

сприяють створенню особливої атмосфери і відчуття присутності. Атмосфера може бути посилена через розширені діалогові сцени, унікальні звукові доріжки і ретельно опрацьовані візуальні елементи, що робить кожну гру незабутнім досвідом.

Рольові ігри (RPG) можуть також включати елементи інших жанрів, що робить їх ще більш привабливими для широкої аудиторії. Наприклад, екшн-RPG поєднують динамічні бої з глибокими сюжетами, тоді як тактичні RPG зосереджені на стратегічних битвах і управлінні ресурсами. MMORPG (масові багатокористувацькі онлайн-рольові ігри) дозволяють тисячам гравців взаємодіяти один з одним у спільних віртуальних світах, створюючи величезні і постійно змінювані спільноти.

Жанр RPG продовжує розвиватися, завдяки технологічним інноваціям та творчому підходу розробників, що забезпечує постійний інтерес гравців до нових ігор та їх улюблених класиків. З виходом кожного нового покоління консолей та ПК, RPG стають все більш захоплюючими і вражаючими, надаючи гравцям незабутні пригоди і можливість досліджувати нові, фантастичні світи.

3 ПРОЕКТУВАННЯ ІГРОВОГО ДОДАТКУ

Проектування гри є ключовим етапом у процесі розробки, який визначає, яким буде кінцевий продукт. Незалежно від зростання популярності відеоігор серед різних груп населення, цільовою аудиторією мого проекту залишаються чоловіки віком від 16 до 35 років, які складають значну частину гравців. Гравці цього віку часто шукають у відеоіграх не тільки розвагу, а й захоплюючий ґеймплей із різноманітними механіками та викликами. Для них важлива атмосфера, насичений ґеймплей і можливість активно взаємодіяти з ігровим світом. Аналізуючи психологічні типи гравців за моделлю, запропонованою професором Університету Ессекса Річардом Аланом Бартлом, я можу припустити, що в основному в мою гру гратиме 2 типи гравців: дослідники та відмінники (achievers). Дослідники надають перевагу глибокому вивченню ігрового світу, розкриттю його механік та можливостей. Вони люблять квести, цінують виклики та різноманітність ігрового процесу, досліджуючи кожну деталь гри. Відмінники прагнуть досягнення цілей і виконання завдань, вони отримують задоволення від накопичення очок, завершення місій і підвищення рівнів. Ці гравці цінують ігри, що пропонують чіткі цілі, досягнення та можливості для прогресу, надаючи значення показникам успішності та рейтингу. Враховуючи інтереси та потреби моєї цільової аудиторії, я вирішив, що гра повинна мати насичений і добре продуманий ґеймплей, різноманітні механіки.

3.1 Розробка концепції гри

Розробка концепції гри є критичним етапом у створенні успішного проекту, оскільки саме тут формується основна ідея та визначаються ключові елементи гри. Концептуальна основа гри у жанрі RPG закладає фундамент для багатогранного ігрового досвіду, який включає в себе бойову систему, рівневий дизайн, взаємодію з ворогами та пошук секретів. Основна мета гравця полягає

у просуванні через рівні, кожен з яких є унікальним та має свою атмосферу та виклики.

Сюжетна лінія гри полягає у тому, що гравець повинен просуватися через рівні, борючись з різноманітними ворогами, щоб досягти кінцевої точки кожного рівня, де його очікує бос або міні-бос. Кожен рівень створений вручну, що дозволяє розробникам додавати унікальні деталі та забезпечувати послідовність у наративі та ігровому процесі. Рівні не генеруються процедурно, що дає можливість ретельно продумати кожен аспект, від розташування ворогів до розміщення секретів.

Основна механіка бою включає сутички з різними типами ворогів. Наразі у грі є два види ворогів, кожен з яких має свої унікальні атаки і стратегії. Гравці повинні використовувати свої навички та здібності персонажа, щоб перемогти ворогів і продовжити свій шлях. Боси та міні-боси, які зустрічаються наприкінці кожного рівня, представляють собою особливий виклик. Вони мають більш складні та різноманітні атаки, що вимагає від гравця застосування тактичного мислення і швидкої реакції.

Дослідження є ще одним важливим аспектом гри. На кожному рівні приховані різні секрети, які гравці можуть знайти, використовуючи свою спостережливість і логічне мислення. Відкриття цих секретів може вимагати виконання певних завдань або вирішення головоломок. Секрети можуть включати приховані кімнати, скарби або особливі предмети, які допоможуть гравцеві в подальших битвах або дадуть додаткову інформацію про світ гри.

Для досягнення максимальної глибини занурення, гра використовує ретельно опрацьовану графіку і звук. Візуальні ефекти створюють атмосферу, яка доповнює сюжет і механіку гри. Кожен рівень має свою унікальну естетику, що підкреслює його тему і сюжетні елементи. Звуковий супровід і музика адаптуються під дії гравця, допомагаючи створювати напруженість під час боїв і атмосферу спокою під час досліджень.

Інтеграція системи розвитку персонажа є ще одним ключовим аспектом концепції. Гравці можуть збирати досвід і покращувати свої навички та

здібності, що додає додатковий рівень стратегії. Різноманітні гілки розвитку дозволяють створювати унікальні комбінації умінь, що робить кожну гру неповторною.

Інтерактивність світу також грає важливу роль. Гравці можуть взаємодіяти з різноманітними об’єктами та NPC (неігровими персонажами), що дає можливість отримувати нові квести, інформацію та інші бонуси. Кожна взаємодія може впливати на розвиток сюжету, що сприяє більш глибокому зануренню у світ гри.

Всі ці елементи – битви з ворогами, стратегічні бої з босами, дослідження рівнів та відкриття секретів, система розвитку персонажа та інтерактивний світ – спрямовані на створення захоплюючого ігрового досвіду. Гравці отримують можливість не лише тестувати свої бойові навички, але й занурюватися у глибокий і різноманітний світ, який приховує багато таємниць. Завдяки продуманій концепції, гра пропонує гравцям цікавий та насичений досвід, який тримає увагу і стимулює бажання продовжувати гру, відкриваючи нові аспекти сюжету і ігрового процесу.

3.2 Планування етапів розробки

Планування етапів розробки є критичним аспектом успішного створення відеоігри, оскільки воно допомагає організувати роботу команди, визначити необхідні ресурси та встановити часові рамки для кожного етапу. Для мого проекту розробки RPG-гри, що включає битви з різними ворогами, дослідження рівнів та боротьбу з босами, процес планування включає кілька ключових фаз, кожна з яких має свої завдання та цілі.

Першим етапом є підготовчий етап, який охоплює визначення концепції гри, створення початкового дизайну та документації проекту. На цьому етапі проводиться аналіз ринку, визначення цільової аудиторії та її потреб, а також формулювання основної ідеї гри. Створюються базові документи, такі як Game Design Document (GDD), які включають опис ігрової механіки, сюжетної лінії,

персонажів, рівнів та інших ключових елементів гри. Це також передбачає створення загального бачення гри, де визначається, яка саме RPG буде створена, які механіки будуть використовуватися, і які цілі повинні бути досягнуті для того, щоб гра була цікавою та залучала гравців. Цей етап передбачає активну участь усіх членів команди, щоб переконатися, що всі аспекти гри узгоджені та що в команді є спільне розуміння кінцевої мети.

Наступним етапом є прототипування, де розробники створюють перші прототипи гри для тестування базових механік та концепцій. Це дозволяє команді виявити потенційні проблеми на ранньому етапі та внести необхідні корективи. Прототипи включають основні елементи ґеймплею, такі як бойова система, рухи персонажа та базові взаємодії з оточенням. Тестування прототипів допомагає визначити, які аспекти гри потребують додаткової роботи або змін. Прототипування часто включає створення кількох ітерацій, кожна з яких додає нові функції або вдосконалює існуючі. Цей процес дозволяє швидко виявляти та вирішувати проблеми, що виникають, і забезпечує, що фінальна версія гри буде мати добре продумані та відшліфовані механіки.

Третім етапом є активна розробка, під час якої всі аспекти гри переходять у стадію активного виробництва. На цьому етапі створюються ігрові рівні, розробляються та впроваджуються всі механіки, моделюються персонажі та вороги, створюються текстури, звукові ефекти та музичні композиції. Кожен член команди працює над своєю частиною проекту, зокрема програмісти реалізують ігрову логіку та механіки, художники створюють візуальні елементи, а звукорежисери працюють над аудіо-супроводом. Це найбільш трудомісткий етап, який вимагає координації між різними відділами та постійної комунікації. Зазвичай використовуються системи управління проектами та інструменти для відстеження прогресу та забезпечення того, щоб усі завдання виконувалися вчасно і в рамках бюджету.

Четвертий етап включає тестування та налагодження, що є важливим для забезпечення якості та стабільності гри. Тестувальники проводять ретельне тестування всіх аспектів гри, виявляючи баги, проблеми з продуктивністю та

інші недоліки. На основі їх звітів, розробники виправляють знайдені помилки та вдосконалюють гру. Тестування проводиться в кілька циклів, щоб забезпечити максимальну стабільність та ігровий досвід. Важливо також проводити різні типи тестування, включаючи функціональне тестування, тестування продуктивності, тестування на сумісність та тестування зручності користування. Це допомагає переконатися, що гра працює належним чином на різних платформах та пристроях і що гравці отримають позитивний досвід від гри.

П'ятим етапом є передрелізна підготовка, включаючи фінальне полірування гри, створення маркетингових матеріалів та підготовку до випуску. На цьому етапі завершується робота над усіма аспектами гри, проводяться останні тестування та вносяться фінальні правки. Паралельно з цим команда маркетингу працює над промоцією гри, створюючи трейлери, скріншоти та інші матеріали для залучення уваги потенційних гравців. Важливо також підготувати документацію та ресурси для підтримки гравців після релізу, включаючи керівництва, форуми підтримки та оновлення програмного забезпечення. Передрелізна підготовка також включає налаштування інфраструктури для розповсюдження гри, забезпечення наявності серверів для онлайн-гри (якщо це необхідно) та підготовку до запуску на різних платформах.

Останнім етапом є реліз та підтримка після випуску. Після виходу гри на ринок команда продовжує працювати над підтримкою проекту, випускаючи оновлення, патчі та додатковий контент. Відгуки гравців аналізуються для виявлення можливих покращень та виправлення проблем, що залишилися. Цей етап також включає роботу над розширенням гри через випуск DLC, оновлень та інших додаткових матеріалів. Підтримка після випуску є важливою для збереження інтересу гравців та забезпечення довготривалого успіху гри. Регулярні оновлення та активна комунікація з гравцями допомагають підтримувати позитивний імідж гри та залучати нових користувачів.

Ці етапи розробки забезпечують чітку структуру роботи над проектом, дозволяючи команді ефективно використовувати ресурси та час, забезпечуючи високу якість кінцевого продукту. Планування та чітке дотримання етапів розробки є ключовими для успішного випуску гри та задоволення очікувань гравців.

3.3 Проектування ґеймплею

Розробка ґеймплея є критичною складовою успішного створення відеоігри, оскільки саме тут визначаються основні механіки, які формують взаємодію гравця з ігровим світом. У моїй RPG грі, яка фокусується на битвах з ворогами, просуванні через рівні та відкритті секретів, ґеймплей має забезпечити глибокий і захоплюючий ігровий досвід, підтримуючи баланс між викликами і нагородами.

Основна концепція ґеймплея полягає у тому, що гравець повинен проходити рівні, борючись з різними ворогами, щоб досягти кінцевої точки кожного рівня, де його очікує бос або міні-бос.

Бойова система є центральним елементом ґеймплея. Наразі в грі реалізовано два види ворогів, кожен з яких має свої унікальні атаки і поведінку. Гравець повинен використовувати свої навички, реакцію та стратегічне мислення, щоб ефективно боротися з цими ворогами. Кожен ворог має свої слабкі місця та тактики, що вимагає від гравця адаптації та планування своїх дій.

Босові битви є ключовим елементом кожного рівня. Боси та міні-боси володіють складнішими і різноманітнішими атаками, що вимагає від гравця використання більшої кількості тактик та навичок. Ці битви не лише перевіряють бойові здібності гравця, але й додають додатковий рівень напруги та задоволення від перемоги.

Кожен рівень гри ретельно спроектований, щоб забезпечити багатогранний досвід дослідження. Гравці можуть знаходити приховані

області та секрети, що вимагають використання різних методів для їх відкриття. Наприклад, деякі секрети можуть бути заховані за хитромудрими головоломками, інші - вимагають певних дій або взаємодій з оточенням. Ці секрети можуть включати додаткові ресурси, особливі предмети або інформацію, яка допомагає краще зрозуміти ігровий світ.

Секрети на рівнях є важливим елементом ґеймплея, який додає додатковий рівень викликів та винагород. Гравці можуть знайти приховані кімнати, скарби або особливі предмети, виконуючи певні дії або вирішуючи головоломки. Це стимулює гравців досліджувати кожен куточок рівнів і використовувати свою спостережливість та логічне мислення для досягнення успіху.

Звуковий супровід та візуальні ефекти відіграють ключову роль у створенні атмосфери гри. Музика, звукові ефекти та візуальні елементи синхронізовані з ігровими подіями, що допомагає створити занурення в ігровий світ. Наприклад, музика може змінюватися залежно від ситуації в грі, підкреслюючи напружені моменти під час битви з босом або створюючи відчуття таємничості при дослідженні прихованих областей.

ґеймплей побудований таким чином, щоб постійно підтримувати інтерес гравця через виклики та нагороди. Кожна перемога над ворогами, знайдений секрет чи вирішена головоломка винагороджуються, що стимулює гравця продовжувати гру. Нагороди можуть включати нові предмети, покращення навичок або доступ до нових рівнів.

Розробка ґеймплея в моїй RPG грі спрямована на створення багатогранного і захоплюючого досвіду для гравців, забезпечуючи баланс між викликами та задоволенням від досягнень. Інтеграція різних елементів бою, дослідження, розвитку персонажа та взаємодії з оточенням допомагає створити гру, яка занурює гравця в унікальний і цікавий світ, повний таємниць і пригод.

Базові можливості можна представити наступною діаграмою (див рис. 3.1). До гри поступають команди, які вона оброблює та видає результат

залежно від них. В основному гравець буде оперувати рухом, атакою та ривком персонажа. Вміле володіння та використання цих простих механік дозволить гравцеві досягти успіху в грі. Гравець може рухатись у всіх 4-ох напрям, тобто по двох осях, те саме стосується й ривку, причому, напрям ривку задається клавішами руху. Ривок і атака працюють певний період і мають невелику затримку, також не можна атакувати, коли йде анімація ривку.

Рисунок 3.1 — UML-діаграма основних можливостей персонажу.

Також, у моїй грі, як і в багатьох інших використовуються анімації. Рушій Unity дає зручний інструмент, який дозволяє налаштовувати анімації: їх час виконання, перебіг одних анімацій в інші та його умови. На рисунку 3.1 зображено UML-діаграму анімацій. На цій діаграмі видно, персонаж починає з початкового стану, при якому персонаж представлений спрайтом. Коли все прогрузилось, то персонаж з початкового стану переходить до анімації

стояння, також до цієї анімації персонаж переходить і коли закінчується дія інших анімацій чи не натиснута жодна кнопка. З анімації стояння персонаж може перейти до наступних анімацій:

- анімація атаки, якщо натиснута клавіша атаки;
- анімація лікування, якщо натиснута клавіша лікування;
- анімація бігу, якщо затиснута клавіша бігу;
- анімація отримання пошкодження, якщо натиснута клавіша пошкодження.

У свою чергу, з анімації бігу можна перейти до анімації ривку, якщо натиснути певну клавішу. З анімації отримання пошкодження — до анімації смерті, якщо закінчиться здоров'я. З анімації смерті — до анімації воскресіння, якщо виконати певну умову.

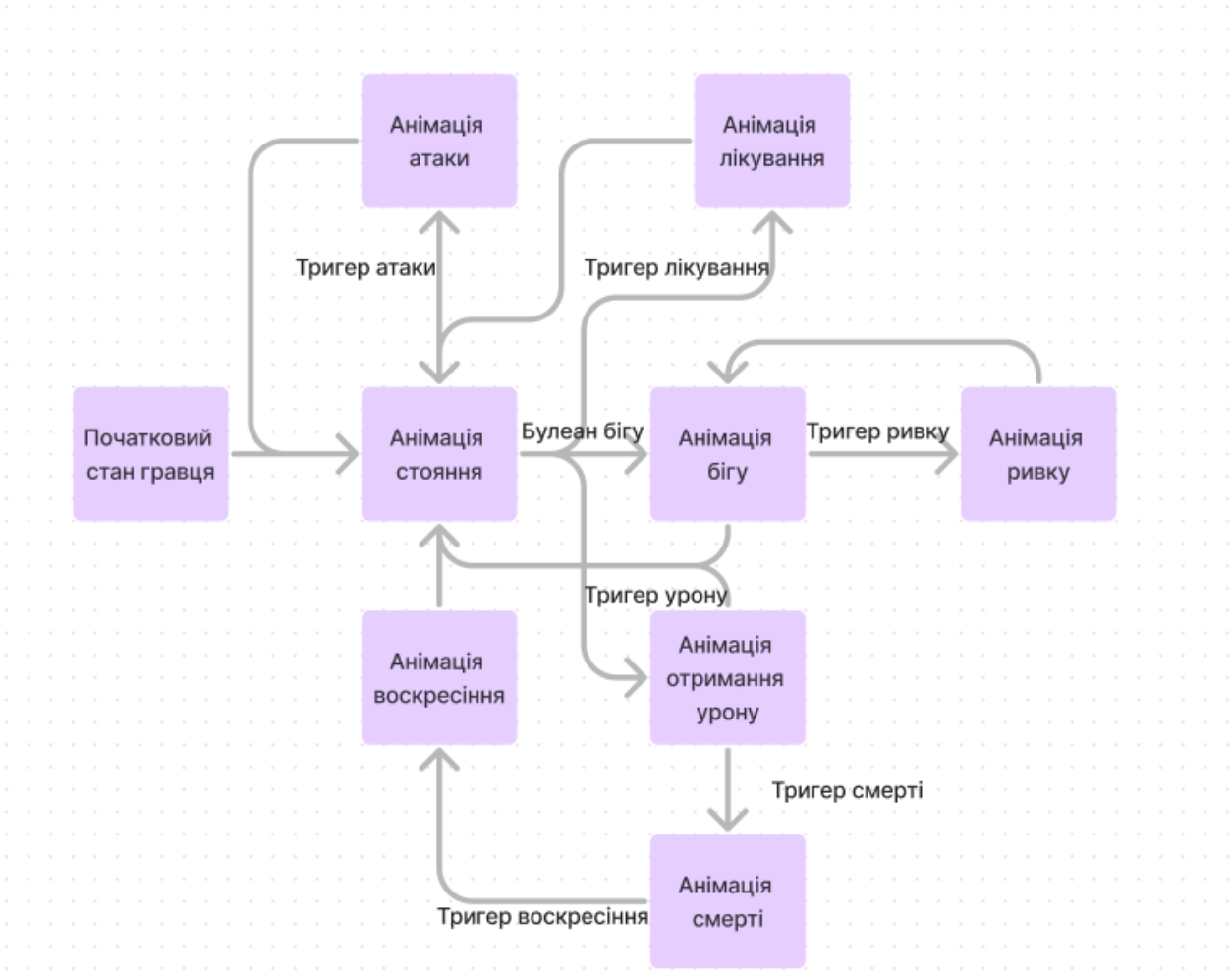

Рисунок 3.2 — UML-діаграма анімацій

3.4 Архітектура системи і ієрархія класів

Основною парадигмою програмування в моєму проекті є об'єктно-орієнтований підхід (ООП). ООП забезпечує ефективну організацію коду в модульну і легко підтримувану структуру, що є критичним для розробки великих програмних комплексів. Об'єктно-орієнтоване проектування полягає в описі структури та поведінки проектованої системи, відповідаючи на два основних питання: з яких частин складається система і в чому полягає відповідальність кожної з частин.

Виділення частин системи проводиться таким чином, щоб кожна частина мала мінімальний за обсягом і точно визначений набір функцій (обов'язків) і при цьому взаємодіяла з іншими частинами якомога менше. Це забезпечує високу модульність системи, де кожен компонент виконує чітко визначені завдання і може бути розроблений, протестований і налагоджений незалежно від інших компонентів. Такий підхід також значно спрощує процес внесення змін до системи, оскільки зміни в одному компоненті мінімально впливають на інші.

Система розділена на кілька основних компонентів, включаючи головний цикл гри, систему управління персонажами, систему ворогів, систему бою, систему рендерингу, систему управління рівнями та систему звукових ефектів. Головний цикл гри відповідає за основний цикл виконання гри, включаючи обробку подій, оновлення стану гри і рендеринг. Система управління персонажами відповідає за рухи і дії гравця, включаючи атаки, захист і взаємодію з об'єктами. Система ворогів містить логіку поведінки ворогів, включаючи атаки і переміщення. Система бою відповідає за обробку боїв між персонажем і ворогами, включаючи розрахунки ушкоджень і спеціальних атак. Система рендерингу відповідає за відображення ігрових об'єктів на екрані. Система управління рівнями відповідає за завантаження і збереження рівнів, а також управління переходами між ними. Система звукових ефектів відповідає за відтворення звукових ефектів і музики.

Кожен з цих компонентів реалізований у вигляді набору класів, які взаємодіють між собою через чітко визначені інтерфейси. Наприклад, система управління персонажами може складатися з класів, які відповідають за основні атрибути і методи, такі як здоров'я, сила атаки і рухи, обробку рухів персонажа і управління інвентарем. Взаємодія між цими класами здійснюється через чітко визначені методи і події.

Система ворогів може містити класи для різних типів ворогів, кожен з яких має свої унікальні властивості і методи, що визначають їх поведінку і атаки. Це дозволяє легко додавати нових ворогів до гри без значних змін у коді.

Система бою складається з класів, які відповідають за різні аспекти бою, включаючи управління боєм, обробку атак і захистів, а також розрахунок ушкоджень. Це забезпечує гнучкість і масштабованість системи бою, дозволяючи легко додавати нові типи атак і захистів.

Система рендерингу включає класи, що відповідають за відображення ігрових об'єктів на екрані, управління анімаціями об'єктів і відображення ігрового світу з певної точки зору. Це забезпечує високу якість графіки і анімацій у грі.

Система управління рівнями включає класи, що відповідають за завантаження і збереження рівнів, управління збереженнями гри та переходами між рівнями. Це забезпечує гнучке і ефективне управління ігровими рівнями, дозволяючи гравцям зберігати свій прогрес і легко переходити між різними частинами гри.

Система звукових ефектів включає класи, що відповідають за управління всіма звуковими ефектами і музикою в грі, включаючи відтворення, зупинку і регулювання гучності. Це забезпечує високу якість звукового супроводу і дозволяє створювати атмосферу гри.

Правильне побудова ієрархії класів має велике значення. Ієрархія класів у проекті структурована таким чином, щоб забезпечити максимальне повторне використання коду і мінімізувати дублювання. Класи організовані у вигляді дерева, де базові класи визначають загальні характеристики і поведінку, а

похідні класи додають спеціалізовані функціональні можливості. Одна з відомих проблем великих систем, побудованих за ООП-технологією, є так звана проблема крихкості базового класу. Ця проблема виникає, коли на пізніх етапах розробки стає важко або навіть неможливо внести зміни в код базових класів, оскільки це може непередбачувано вплинути на похідні класи. У великій системі розробник базового класу може не знати всіх особливостей його використання, що ускладнює внесення змін. Загальна діаграма ієрархії класів у Unity представлена на рисунку 3.3. На якій зображено клас Object, який є батьківським для всіх класів C#. Від класу Object унаслідується клас Component, який є частино простору імен Unity, у свою чергу від нього унаслідується клас Behaviour, від якого унасладіується клас MonoBehaviour, який є батьківським класом для всіх користувацьких класів.

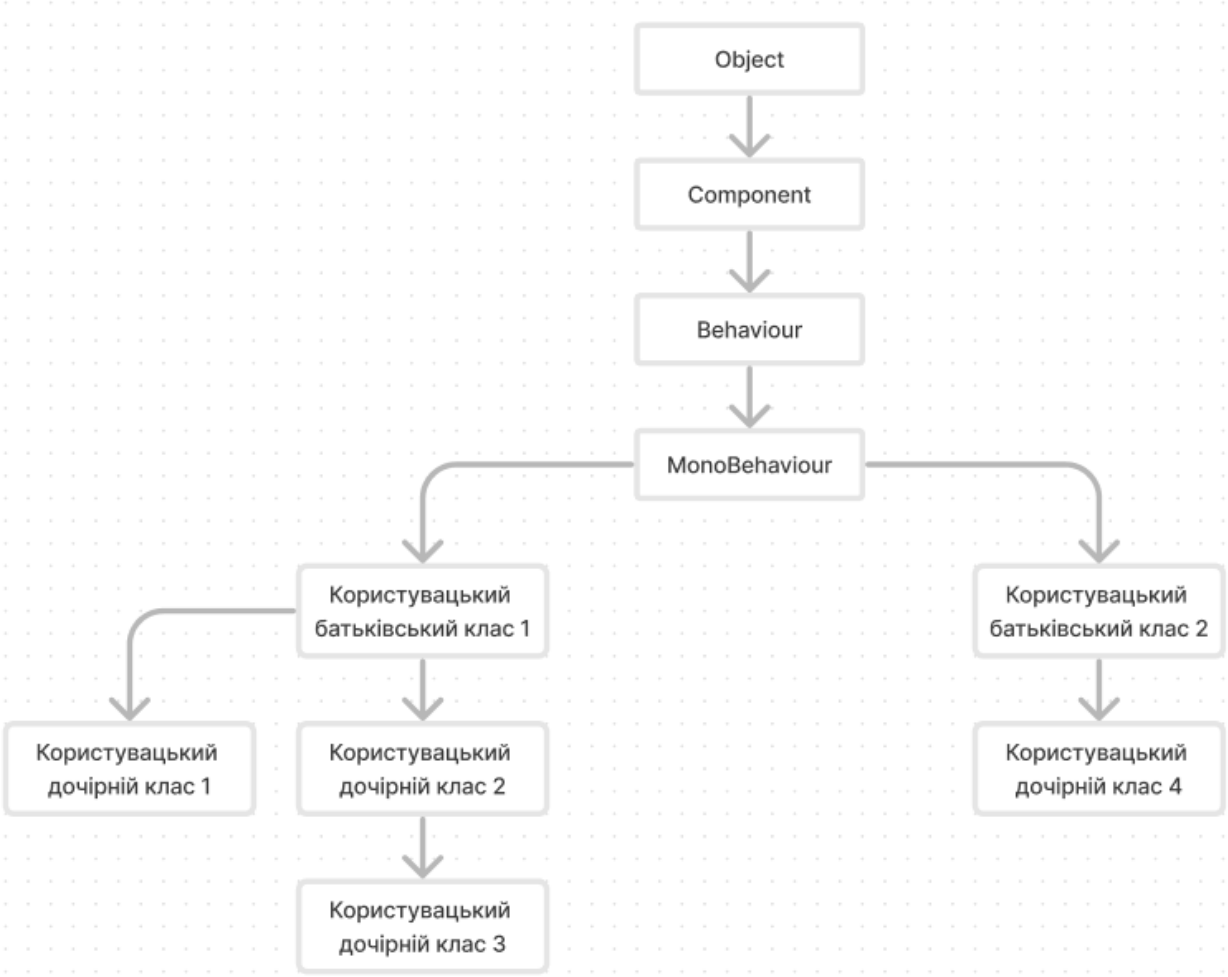

Рисунок 3.3 — Загальна діаграма ієрархії класів

Для забезпечення ефективної взаємодії між компонентами системи використовуються інтерфейси та абстрактні класи. Це дозволяє зменшити зв'язність між класами та забезпечити можливість їх незалежної розробки і тестування. Інтерфейси визначають контракт, який повинні реалізувати класи, а абстрактні класи надають базову реалізацію, яку можуть наслідувати і доповнювати похідні класи.

Архітектура системи та ієрархія класів у моїй грі базується на принципах ООП, що дозволяє створити модульну, легко підтримувану і розширювану систему. Виділення чітких компонентів і відповідальностей, правильне побудова ієрархії класів та використання інтерфейсів і абстрактних класів забезпечують гнучкість і масштабованість проекту. Це особливо важливо для великих і складних систем, де внесення змін може мати значний вплив на інші частини програми.

4 РОЗРОБКА ТЕХНІЧНОГО ПРОЕКТУ

4.1 Реалізація механік гри

Інтерфейс Unity складається з декількох ключових елементів, кожен з яких відповідає за певні функції під час розробки ігор або програм:

Сцени використовуються для організації різних візуальних ігрових об'єктів, які відображаються у грі. Вони дозволяють вам створювати і керувати окремими рівнями або екранами гри. Ієрархія, зі свого боку, відображає структуру об'єктів у поточній сцені, де ви можете переглядати, організовувати та змінювати їхню взаємодію.

Інспектор надає можливість переглядати та редагувати властивості об'єктів, вибраних у сцені або ієрархії. Він є центральним інструментом для налаштування параметрів об'єктів, компонентів та ресурсів, що дозволяє точно настроювати поведінку вашої гри або програми.

Проєкт містить усі активи вашого проєкту, такі як зображення, звуки, моделі, сцени тощо. Вікно Проєкту дозволяє вам організовувати, керувати та імпортувати ресурси, що використовуються у вашому проекті.

Графічні вікна та редактори: Unity надає низку графічних вікон і редакторів, які допомагають у розробці:

- Сцена - для візуального редагування і розміщення об'єктів у просторі.
- Анімаційний редактор - для створення і редагування анімацій об'єктів.
- Кодовий редактор - для програмування логіки гри або програми.
- Інші редактори дозволяють працювати зі скриптами, матеріалами, фізикою та іншими аспектами проєкту.

Консоль відображає відладочні повідомлення, інформаційні повідомлення та помилки, що допомагає відстежувати стан і виконання вашої гри або програми під час розробки і тестування.

Меню містить доступ до різних опцій і налаштувань вашого проєкту, інструменти для імпорту та експорту ресурсів, а також інші інструменти

розробки, які спрощують управління проєктом і налаштування середовища розробки Unity.

Ці елементи, які зображегі на рисунку 4.1, утворюють повний набір інструментів, необхідних для створення, налаштування та відлагодження вашої гри або програми в середовищі Unity.

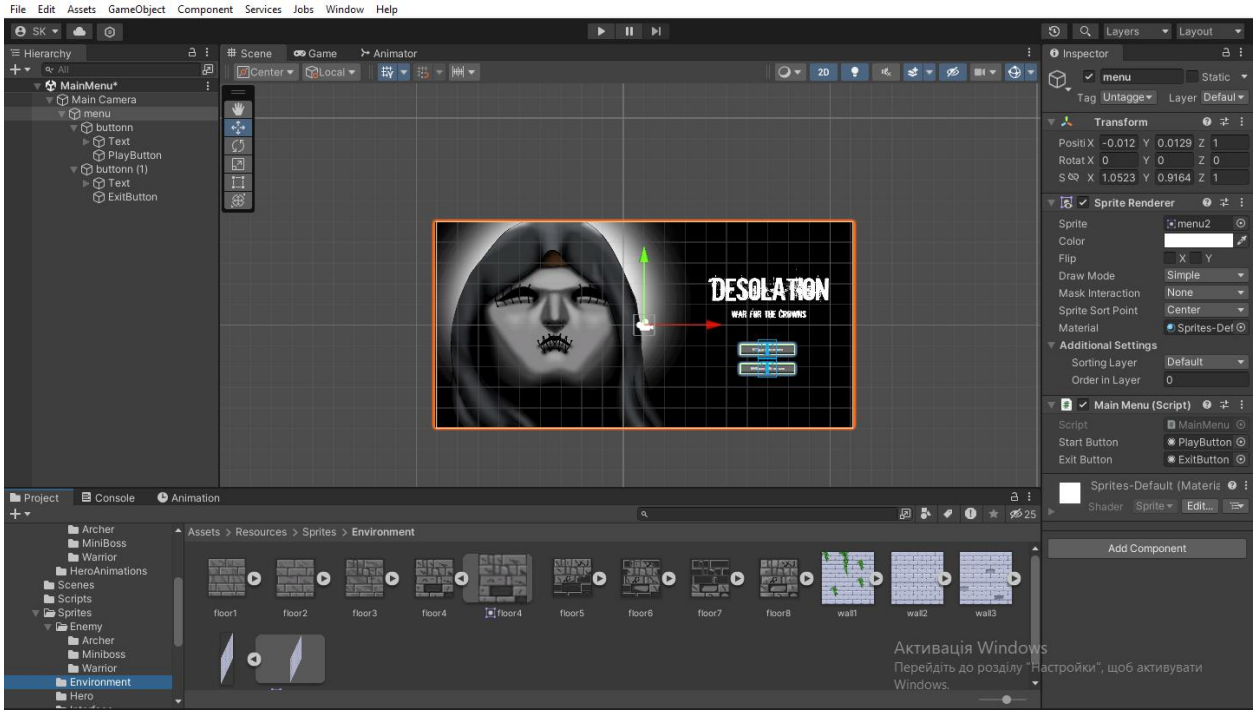

Рисунок 4.1 — Меню рушія Unity

Також на рисунку 4.1 можна було помітити головне меню моєї гри. Воно складається з тла та двох кнопок із текстом. При натисканні на одну з кнопок виконується скрипт, який визначає яка кнопка натиснута та або закриває гру, або завантажує наступну сцену, яка є основною сценою гри:

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

public class MainMenu : MonoBehaviour
{
    [SerializeField] Button startButton, exitButton;

    private void Start()
    {
```

```
        startButton.onClick.AddListener(StartButtonClick);
        exitButton.onClick.AddListener(ExitButtonClick);
    }

    void StartButtonClick()
    {
        SceneManager.LoadScene("MainScene");
        Debug.Log("Click");
    }
    void ExitButtonClick()
    {
        Application.Quit();
        Debug.Log("Click");
    }
}
```

Головна сцена представляє з себе арену (див. рис. 4.2) з 3 лучниками, 2 воїнами та 1 мінібосом.

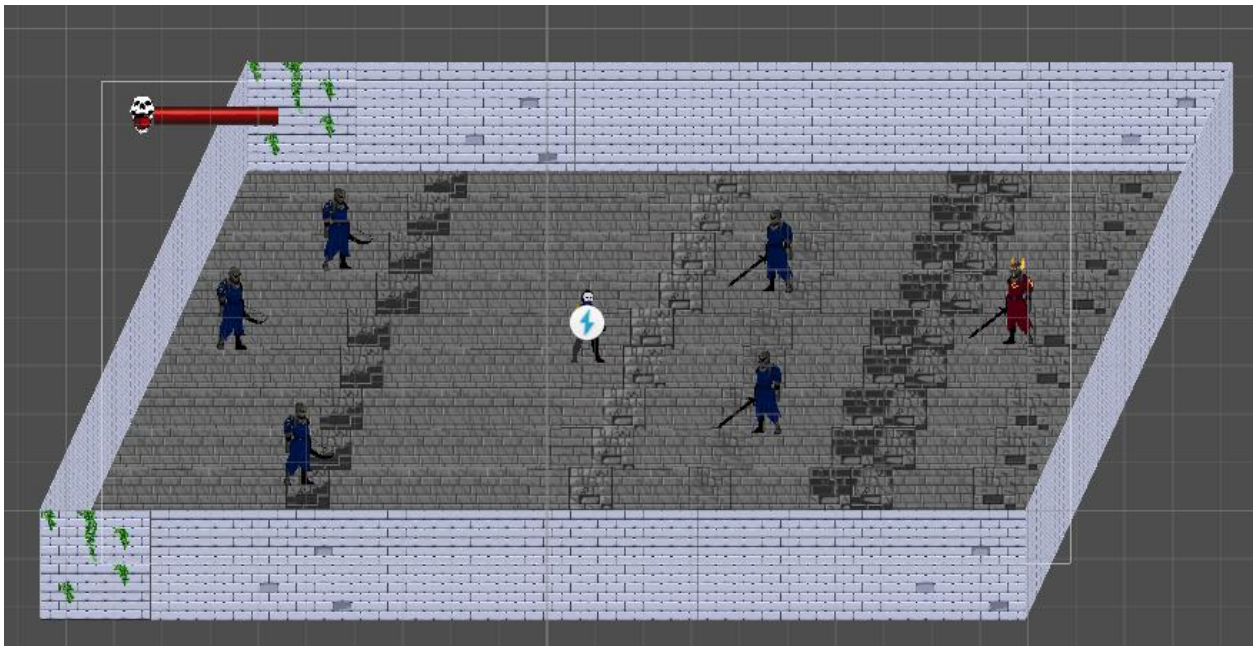

Рисунок 4.2 — Основна сцена гри

Ця сцена являє собою арену для тестування основних механік гри та балансування складності ворогів і гри загалом. У головного героя є здоров'я, якщо здоров'я опускається до нуля то персонаж помирає. Гравець має певний контроль над персонажем і може ходити, атакувати та робити ривок. Контроль персонажа реалізований за допомогою InputAction, який дозволяє встановлювати власні клавіші для керування персонажем. У цій частині коду

підключаються необхідні простори імен і створюється клас, який має певні поля та методи й унаслідується від MonoBehaviour:

```
using System.Collections;
using UnityEngine;
using UnityEngine.InputSystem;

public class HeroControls : MonoBehaviour
{
    [SerializeField] float moveSpeed;
    [SerializeField] Rigidbody2D rigidbody2D;
    public InputAction playerMovement;
    public InputAction playerDash;
    public InputAction playerAttack;
    [SerializeField] Collider2D playerAttackCollider;
    float dashForce = 20f;
    float dashDuration = 0.2f;
    float dashCooldown = 1f;
    bool canDash = true;
    bool isDashing = false;
    float attackDamage = 20f;
    float attackDuration = 0.5f;
    float attackCooldown = 0.2f;
    bool canAttack = true;
    bool isAttacking = false;

    Vector2 moveDirection = Vector2.zero;
    Vector3 originalScale;
```

Нижче представлений метод — метод Start, який є основним з методів Unity і спрацьовує один раз, коли починає працювати об'єкт, до якого прив'язаний клас. У даній реалізації метод Start отримує необхідні компоненти та задає початкові значення для деяких полів, зокрема для сили атаки й ривку, для їх тривалості, для їх перезарядки і тому подібне:

```
    void Start()
    {
        rigidbody2D = GetComponent<Rigidbody2D>();
        moveSpeed = 5f;
        originalScale = transform.localScale;
        playerAttackCollider.enabled = false;
        playerAttackCollider.tag = "PlayerAttack";
       attackDuration = 0.5f;
       attackCooldown = 0.2f;
       attackDamage = 20f;
       isDashing = false;
       canDash = true;
       dashCooldown = 1f;
```

```
        dashDuration = 0.2f;
        dashForce = 20f;
        isAttacking = false;
        canAttack = true;
    }
```

Метод OnEnable() є одним з методів життєвого циклу Unity, який викликається автоматично в момент, коли компонент (скрипт) стає активним. В методі OnEnable() зазвичай викликаються методи Enable() для об'єктів типу InputAction. Це дозволяє включити моніторинг вхідних подій (таких як натискання клавіш або кнопок на геймпаді), які пов'язані з конкретними діями гравця в грі. Після активації вхідних дій, до кожної з них приєднуються обробники подій (event handlers). Це зазвичай реалізується шляхом додавання методів до подій performed або інших подій, що визначені для відповідних InputAction. Обробники подій (event handlers) викликаються автоматично, коли відбувається відповідна вхідна подія (наприклад, натискання клавіші). Весь цей процес включення дозволяє компоненту (скрипту) почати відслідковувати та обробляти вхідні дії гравця відразу після того, як він став активним. Це важливо для коректної роботи інтерактивних елементів гри, таких як управління персонажем, обробка атаки, руху чи інших дій.

```
    private void OnEnable()
    {
        playerMovement.Enable();
        playerDash.Enable();
        playerDash.performed += OnDash;
        playerAttack.Enable();
        playerAttack.performed += OnAttack;
    }
```

Метод OnDisable() в компонентах Unity викликається при вимкненні самого компонента або об'єкта, до якого він прикріплений. Його основна мета полягає в очищенні або відключенні функціоналу, що був активований під час активації компонента (у методі OnEnable()). Вимкнення моніторингу вхідних подій: Наприклад, об'єкти типу InputAction можуть мати метод Disable(), який припиняє відслідковування вхідних подій (натискання кнопок чи клавіш).

Обробники подій, які були приєднані до різних подій або інтерфейсів, можуть бути відключені чи видалені, щоб уникнути непотрібних викликів функцій під час неактивності. У разі використання анімацій або інших процесів, які відтворюються в залежності від стану компонента, OnDisable() може зупиняти або призупиняти їхню роботу для економії ресурсів.

```
private void OnDisable()
{
    playerMovement.Disable();
    playerDash.Disable();
    playerDash.performed -= OnDash;
    playerAttack.Disable();
    playerAttack.performed -= OnAttack;
}
```

Метод Update() викликається кожен кадр в Unity і використовується для обробки вхідних даних та оновлення стану об'єкта. Він отримує значення вхідних подій для руху гравця за допомогою playerMovement.ReadValue<Vector2>(). На основі значень moveDirection.x (горизонтальне напрямок руху) відбувається зміна масштабу об'єкта (гравця), щоб відображати його напрямок. Цей метод використовується для обробки вхідних даних та обчислень, які потрібно оновлювати кожен кадр.

Метод FixedUpdate() також викликається кожен кадр, але з регулярністю, залежною від фізичного кроку симуляції (зазвичай 50 разів на секунду). Встановлює швидкість (velocity) об'єкта rigidbody2D в залежності від значень moveDirection та moveSpeed, що дозволяє об'єкту рухатися. Якщо змінна isDashing встановлена в true, метод викликає інший метод Dash(), що виконує спеціальний рух (ривок) з заданою швидкістю.

```
void Update()
{
     moveDirection = playerMovement.ReadValue<Vector2>();

    if (moveDirection.x > 0)
    {
        transform.localScale = new
Vector3(Mathf.Abs(originalScale.x), originalScale.y,
originalScale.z);
    }
```

```
        else if (moveDirection.x < 0)
        {
            transform.localScale = new Vector3(-
Mathf.Abs(originalScale.x), originalScale.y, originalScale.z);
        }
    }

    private void FixedUpdate()
    {
        rigidbody2D.velocity = new Vector2(moveDirection.x *
moveSpeed, moveDirection.y * moveSpeed);

        if (isDashing)
        {
            Dash();
        }
    }

    private void Dash()
    {
        rigidbody2D.velocity = new Vector2(moveDirection.x,
moveDirection.y).normalized * dashForce;

    }
```

Метод DashCoroutine() є ітератором (IEnumerator), що використовується для управління процесом ривка (dash) в грі. Встановлює прапорці canDash і isDashing відповідно на false і true, що забороняє нові ривки під час виконання поточного. Встановлює canAttack на false, щоб заборонити атаку під час ривка. Чекає протягом часу dashDuration, щоб тривати ривок. Після закінчення часу dashDuration, відновлює стани: встановлює isDashing на false, canAttack на true і обнулює швидкість (velocity) об'єкта. Потім чекає протягом часу dashCooldown, перш ніж дозволити новий ривок, встановлюючи canDash на true.

```
    private IEnumerator DashCoroutine()
    {
        canDash = false;
        isDashing = true;
        canAttack = false;
        yield return new WaitForSeconds(dashDuration);

        isDashing = false;
        canAttack = true;
        rigidbody2D.velocity = Vector2.zero;
```

```
        yield return new WaitForSeconds(dashCooldown);

        canDash = true;
    }
```

OnDash перевіряє чи може гравець зробити ривок, у разі наявності такої можливості, він викликає метод StartCotoutine, який запускає корутину DashCoroutine.

```
    private void OnDash(InputAction.CallbackContext context)
    {
        if (canDash)
        {
            StartCoroutine(DashCoroutine());
        }
    }
```

OnAttack перевіряє чи може гравець атакувати, у разі наявності такої можливості, він викликає метод StartCotoutine, який запускає корутину AttackCoroutine.

```
    private void OnAttack(InputAction.CallbackContext context)
    {
        if (canAttack)
        {
            StartCoroutine(AttackCoroutine());
        }
    }
```

Метод AttackCoroutine() є ітератором (IEnumerator), який керує процесом атаки в грі. Виводить повідомлення у консоль про початок атаки за допомогою Debug.Log("Attack"). Забороняє можливість атаки, встановлюючи canAttack на false, та встановлює isAttacking на true. Увімкнює коллайдер (playerAttackCollider), що відповідає за атаку, дозволяючи об'єкту взаємодіяти з іншими об'єктами під час атаки. Чекає протягом часу attackDuration, під час якого триває атака. Після закінчення атаки встановлює isAttacking на false та Вимикає коллайдер, встановлюючи playerAttackCollider.enabled на false. Чекає

протягом певного часу attackCooldown, перш ніж дозволити наступну атаку, встановлюючи canAttack на true.

```
    private IEnumerator AttackCoroutine()
    {
        Debug.Log("Attack");
        canAttack = false;
        isAttacking = true;
        playerAttackCollider.enabled = true;
        yield return new WaitForSeconds(attackDuration);

        isAttacking = false;
        playerAttackCollider.enabled = false;

        yield return new WaitForSeconds(attackCooldown);

        playerAttackCollider.enabled = false;
        canAttack = true;
    }
}
```

Протягом усього сеансу гравець буде спостерігати анімації персонажів. Завдяки зручному інтерфейсу Unity та меню Animator (див. рис. 4.3) вдалося досить швидко та зручно налаштувати зв'язки та перемикання між різними анімаціями.

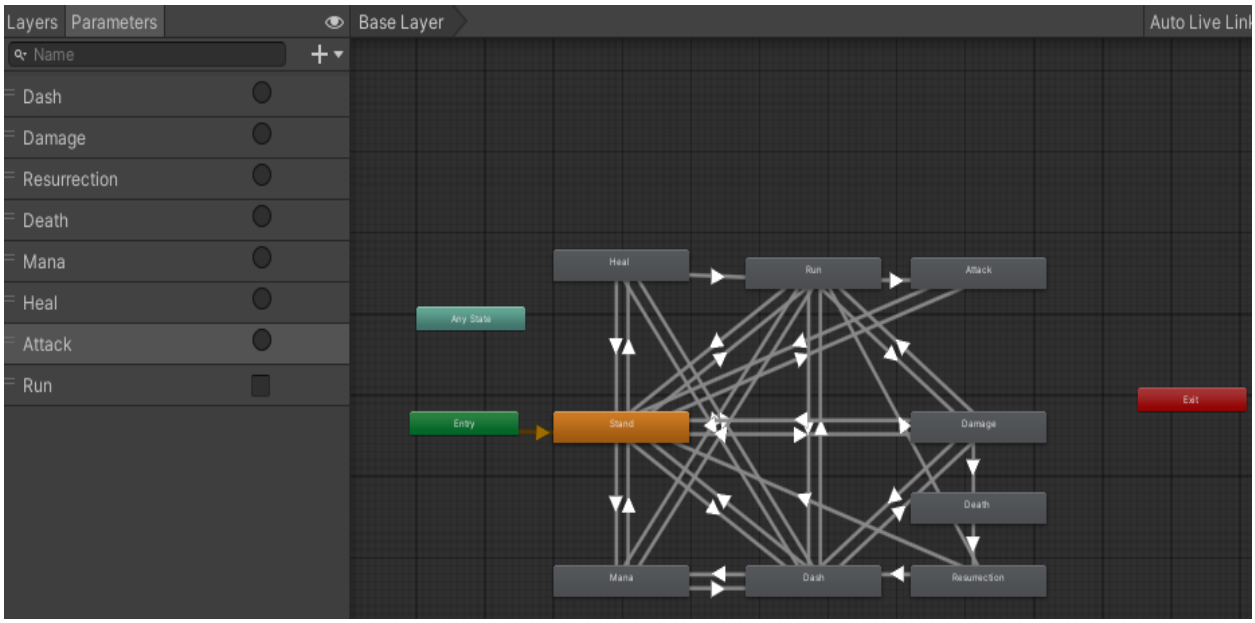

Рисунок 4.3 — Меню Animator

В Animator можна побачити прямокутник, які є різними станами персонажу, вони поєднанні стрілками, які слугують умовами переходу між станами. Також ліворуч можна побачити змінні, що при зміні яких і спрацьовують умови переходу між станами. Помаранчевим зображено поточний стан, зеленим початковий, бірюзовим будь-який, який здебільшого не використовується, як і червони — вихід, а сірим зображені користувацькі стани. Основний стан — стояння, від нього та до нього можна перейти з більшості станів, проте, наприклад, до стану смерті не можна перейти та з нього теж, стояння та смерть мають посередників між собою. Проте самі по собі стани не будуть переходити, для цього я розробив скрипт, який отримує необхідні компоненти та встановлює певні значення для умов переходу:

```
     class HeroAnimations : MonoBehaviour
{
    public Animator playerAnimator;
    HeroControls heroControls;
    private void Start()
    {
        playerAnimator = GetComponent<Animator>();
        heroControls = GetComponent<HeroControls>();
        heroControls.playerMovement.started +=
OnMovementStarted;
        heroControls.playerMovement.canceled +=
OnMovementCanceled;
    }
    private void OnMovementStarted(InputAction.CallbackContext
context)
    {
        playerAnimator.SetBool("Run", true);
    }

    private void OnMovementCanceled(InputAction.CallbackContext
context)
    {
        playerAnimator.SetBool("Run", false);
    }
    private void Update()
    {
        if (heroControls.playerAttack.triggered)
            playerAnimator.SetTrigger("Attack");
        if (heroControls.playerDash.triggered)
            playerAnimator.SetTrigger("Dash");
    }
}
```

4.2 Збір гри

Перший етап полягає в підготовці всіх необхідних ресурсів, таких як моделі персонажів, об'єкти у грі, текстури, звуки та інші асети. Важливо, щоб ці ресурси були готові для інтеграції в середовище Unity без проблем.

Другий етап включає налаштування самого середовища Unity для збирання гри. Це включає у себе встановлення параметрів компіляції, які можуть впливати на швидкодію та оптимізацію гри, такі як налаштування освітлення, шкали деталей об'єктів, управління пам'яттю та інші налаштування проекту.

Третій етап - це тестування гри після збірки для перевірки її правильної роботи. Важливо впевнитися, що всі функції гри працюють коректно, що немає помилок і що гра відповідає заданим вимогам.

Крім того, процес збору також може включати стадію оптимізації, де розробники працюють над покращенням продуктивності гри, зниженням обсягу пам'яті, необхідного для запуску гри, а також зменшенням часу завантаження та інших технічних аспектів, що впливають на загальний досвід гравця. На рисунку 4.4 зображено Скріншот екрану, де показано вже зібрану гру.

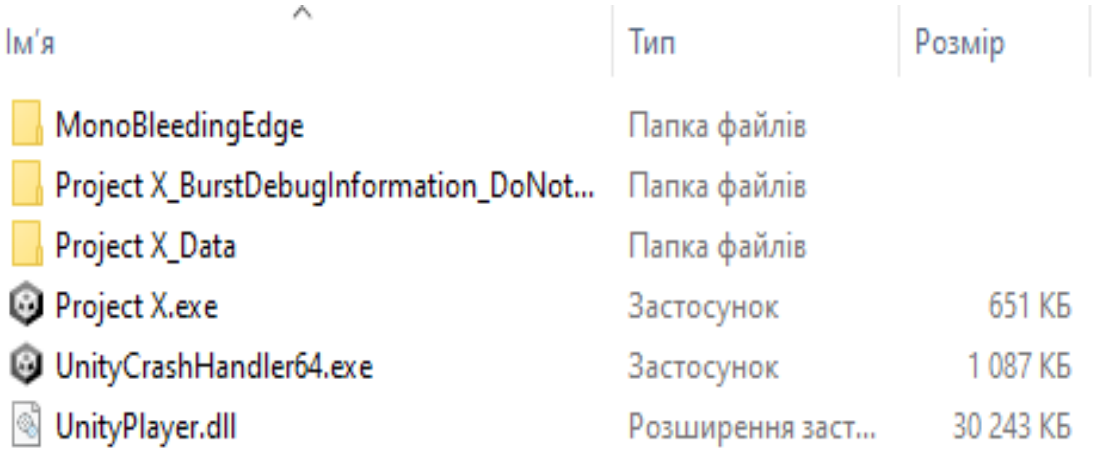

Рисунок 4.4 — Зібрана гра

4.3 Тестування гри

Під час розробки гри кожна нова функція тестується безпосередньо після її впровадження для швидкого виявлення та виправлення помилок у коді. Тести допомогли виявити найбільш ефективний метод патрулювання для різних типів ворогів: воїна, мінібоса і лучників.

Спочатку вороги були налаштовані на пряме переміщення до гравця, однак цей підхід виявився неефективним для лучників, які стріляють з відстані і намагаються тримати безпечну відстань. Після дослідження було вирішено використовувати різні підходи для різних типів ворогів:

Воїн і мінібос: ці вороги намагаються зближатися з гравцем для ближнього бою. Вони використовують алгоритм, що дозволяє їм наближатися до гравця, здійснюючи рух у напрямку гравця з певною швидкістю. Це підвищує напругу і викликає більше емоцій у гравця.

Лучники: ці вороги намагаються тримати безпечну відстань від гравця і стріляють здалеку. Вони використовують алгоритм, який дозволяє їм підтримувати оптимальну відстань, відступаючи від гравця, якщо він наближується, і залишаючи певну відстань, щоб зберегти свою безпеку.

Такий підхід дозволяє кожному типу ворогів використовувати відмінну стратегію патрулювання, що відповідає їхнім унікальним характеристикам і створює цікавий геймплей для гравців.

Додатково, в процесі розробки особлива увага приділяється інтерфейсу користувача, щоб забезпечити зручність взаємодії з грою, включаючи коректне відображення інформації та реакцію на дії гравця через різні пристрої введення.

Тестування цих рішень дозволило забезпечити якісний геймплей і задоволення від взаємодії з грою для всіх типів гравців.

ВИСНОВКИ

У рамках даного дипломного проекту була успішно реалізована гра на платформі Unity з використанням мови програмування C# та середовища розробки Visual Studio. Проект був ініційований з метою створення цікавого інтерактивного ігрового середовища, що сприяє покращенню користувацького досвіду та забезпеченню ефективного управління ґеймплеєм.

Гра використовує передові технології розробки ігор, зокрема Unity для створення ігрового середовища, реалізації графічних та фізичних ефектів, а також обробки взаємодії з користувачем через різні управляючі пристрої.

Проект включає ряд ключових функціональностей, таких як реалізація ігрового інтерфейсу, управління персонажем, ворогами та іншими об'єктами в грі, а також імплементація різноманітних ігрових механік для створення цікавого геймплею.

Гра пройшла комплексне тестування, включаючи функціональне, інтеграційне та ігрове тестування, що дозволило виявити та виправити потенційні проблеми та забезпечити високу якість геймплею.

В результаті дипломного проекту були досягнуті основні цілі, включаючи розробку ігрового функціоналу, оптимізацію продуктивності та створення задоволення від інтерактивної взаємодії з грою.

Можливості для подальшого розвитку проекту включають розширення функціоналу гри, додаткову оптимізацію та використання новітніх технологій для покращення ігрового досвіду користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Newzoo predicts global games market to reach $189 billion in 2024 | Multiplatform.com. [Електронний ресурс]. Режим доступу: https://multiplatform.com/news/newzoo-predicts-global-games-market-to-reach-189-billion-in-2024. (дата звернення 24.05.2024)
2. Global Games Market Expected To Hit $189 Billion In 2024 | GAM3S.GG. [Електронний ресурс]. Режим доступу: https://gam3s.gg/news/global-games-market-189-billion-2024. (дата звернення 24.05.2024)
3. Платформа Unity для розробки в реальному часі | Рушій для 3D, 2D, VR і AR. [Електронний ресурс]. Режим доступу: https://unity.com. (дата звернення 25.05.2024)
4. The most powerful real-time 3D creation tool - Unreal Engine. [Електронний ресурс]. Режим доступу: https://www.unrealengine.com/en-US. (дата звернення 25.05.2024)
5. Godot Engine - Free and open source 2D and 3D game engine. [Електронний ресурс]. Режим доступу: https://godotengine.org. (дата звернення 25.05.2024)
6. CRYENGINE | The complete solution for next generation game development by Crytek: [Електронний ресурс]. Режим доступу: https://www.cryengine.com. (дата звернення 25.05.2024)
7. Make Your Own Game with RPG Maker (rpgmakerweb.com). [Електронний ресурс]. Режим доступу: https://www.rpgmakerweb.com. (дата звернення 25.05.2024)
8. Best Programming Languages for Game Development (mooc.org). [Електронний ресурс]. Режим доступу: https://www.mooc.org/blog/best-programming-languages-for-game-development.
9. Unity - Manual: Scripting (unity3d.com). [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/ScriptingSection.html. (дата звернення 02.06.2024)

10. Unity Documentation. [Електронний ресурс]. Режим доступу: https://docs.unity.com. (дата звернення 02.06.2024)
11. Visual Studio. [Електронний ресурс]. Режим доступу: https://visualstudio.microsoft.com. (дата звернення 02.06.2024)
12. Visual Studio Code - Code Editing. Redefined. [Електронний ресурс]. Режим доступу: https://code.visualstudio.com. (дата звернення 02.06.2024)