СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
(повне найменування інституту, факультету) .

Кафедра інформаційних технологій та програмування
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
(бакалавр, магістр)

спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

спеціалізація Інженерія програмного забезпечення
(назва спеціалізації)

на тему «Гра платформер»

Виконав: студент групи ІПЗ-19бд _____ Я.О. Коломієць
(підпис) (ініціали і прізвище)

Керівник _____ В.О.Лифар
(підпис) (ініціали і прізвище)

Завідувач кафедри ІТП _____ В.О.Лифар
(підпис) (ініціали і прізвище)

Київ - 2023

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
(повне найменування інституту, факультету) .

Кафедра інформаційних технологій та програмування
(повна назва кафедри)

Освітній ступінь бакалавр
(бакалавр, магістр)

спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

спеціалізація «Інженерія програмного забезпечення»
(назва спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТП
__Лифар В.О._
“____” _____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Коломієць Ярослав Олексійович
(прізвище, ім’я, по батькові)

1. Тема роботи_Гра платформер

Керівник роботи Лифар Володимир Олексійович, д.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджений наказом університету від “26”__04____2023_року №_240/15.15-ОД____

2. Строк подання студентом роботи_до 19.06.2023

3.Вихідні дані до роботи Завдання створення ігрового продукту, середовище розробки Unity3D

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ, огляд ринку ігрової індустрії, порівняння найпопулярніших ігрових, порівняльна характеристика ігрових движків, аналіз пристроїв для розробки ігрових продуктів, розробка гри у середовищі Unity 3D.

5. Дата видачі завдання_24.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	30.03.23	
2	Укладання і погодження з керівником плану і етапів виконання роботи	05.04.23	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	11.04.23	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	22.04.23	
5	Укладання та тестування програмного продукту	06.05.23	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	03.05.23	
7	Здача готової пояснювальної записки на кафедру	13.05.23	
8	Укладання доповіді і презентації	30.05.23	

Студент _____ _____Я.О. Коломієць_____
(підпис) (ініціали і прізвище)

Керівник роботи _____ ___В.О. Лифар_____
(підпис) (ініціали і прізвище)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Гра платформер” складається із вступу, трьох розділів, загальних висновків, списку використаних джерел і містить 61 сторінки тексту, 26 рисунки. Список використаних джерел містить 16 найменувань.

Метою випускної роботи є дослідження ринку ігрової індустрії, опис основних програмних засобів, використовуваних для створення ігор, аналізування ключових етапів процесу розробки комп’ютерних ігор та створення комп’ютерної гри на підставі отриманих даних аналізу.

Предметом дослідження є комп'ютерна гра. Комп'ютерна гра представляє собою форму відеоігор, яку грають на особистому комп'ютері (ПК), а не на ігровій консолі або аркадному автоматі. Ця характеристика має свої підстави, які були досліджені у моїй дипломній роботі. Гра, подібно до будь-якого програмного продукту, пройшла через конкретні етапи, від задуму до готового продукту, і має свої унікальні особливості, що стосуються процесу створення та реалізації.

Об’єктом дослідження є аналіз ігрової індустрії та процесу впровадження нових програмних продуктів.

Зміст

ВСТУП

Світове суспільство пройшло еволюцію, яка привела до того, що тепер ми маємо величезний контроль над світом. Інтернет надав нам безліч можливостей і доступ до величезного обсягу інформації. Телефони та ноутбуки допомагають нам залишатися на зв'язку, а для багатьох з нас вони стали невід'ємною частиною життя. Ми зберігаємо фотографії в Інтернеті, контакти на хмарних сервісах, а програми в соціальних мережах завжди доступні. Наш сучасний світ недвозначно став цифровим. Нові технології, від соціальних медіа та систем GPS до штучного інтелекту, перетворюють нашу планету в абсолютно новий цифровий світ.

Ігрова індустрія різнобічна та відкрита для нових ідей, технологій та досягнень. Сьогодні ігри - це не просто спосіб цікаво провести час, вони надають яскраві емоції, незабутні враження і унікальний досвід. Вони також формують швидкозростаючі спільноти, а для деяких стають професією і навіть спортом. Розробка ігор - це справжнє мистецтво.

Люди грають в ігри не тільки самої гри ради, але й для того, щоб отримати досвід, який гра надає: захоплюючі адреналінові почуття, загадкові пригоди, психологічні виклики. Ігри дозволяють людям насолоджуватися миттєвими емоціями і переживаннями, подолати складні геймплейні виклики і забути про щоденні турботи. Інформаційні технології призвели до складнощів у житті сучасних маркетологів, вводячи нові терміни, поняття, методи і принципи роботи, але водночас вони надали компаніям нові можливості та розширили межі їхньої діяльності.

Тема дипломної роботи стає актуальною через необхідність створення нових програмних продуктів, що сприятимуть розвитку галузі ігор та розваг в цілому в контексті росту інформаційних технологій. Сьогодні у нас є багато технічних можливостей для створення багатоплатформених ігор, які можуть бути запущені на різних ігрових пристроях. Це дає змогу вийти на ринок, який найшвидше росте та розвивається в світі.

Об'єктом дослідження є технології розробки та впровадження ігрових програмних продуктів.

Предметом дослідження є комп'ютерна гра. Комп'ютерна гра відрізняється від ігрових консолей чи аркадних автоматів тим, що вона використовується на персональних комп'ютерах (ПК). Причини такого вибору були розглянуті в рамках дослідження, оскільки процес розробки та реалізації гри, як будь-якого програмного продукту, має свої етапи та особливості.

Метою дипломної роботи є проектування ігрового процесу та реалізація етапів розробки комп'ютерної гри.

Для досягнення мети дипломної роботи було визначено наступні завдання:

- провести дослідження ринку ігрової індустрії.
- надати опис основних програмних засобів, використовуваних для створення ігор.
- проаналізувати основні етапи процесу розробки комп'ютерних ігор.
- на основі отриманих даних аналізу розробити комп'ютерну гру.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз стану ігрової індустрії на сьогоднішній день

На сьогоднішній день ігрова індустрія є найбільшою галуззю розваг у світі з близько 3 мільярдами геймерів по всьому світу. Прогнозується, що у 2021 році ця індустрія заробить приблизно 175,8 млрд доларів США, що становитиме зменшення на -1,1% порівняно зі значним зростанням у 2020 році.

Обсяг світового ринку відеоігор в 2019 році оцінювався на рівні 151,06 млрд доларів США, і очікується, що він буде зростати з річною середньорічною темпом зростання (CAGR) 12,9% від 2020 до 2027 року. Очікується, що технологічний прогрес і інновації як у апаратному, так і у програмному забезпеченні будуть основними факторами росту. Розширення доступу до Інтернету та легкість доступу до ігор онлайн по всьому світу також сприятимуть оптимістичним перспективам ринку у майбутньому. Розробники ігор постійно вдосконалюються та премудро обходять технологічні обмеження, що стосуються рендерингу графіки в реальному часі, що, очікується, сприятиме подальшому зростанню галузі.

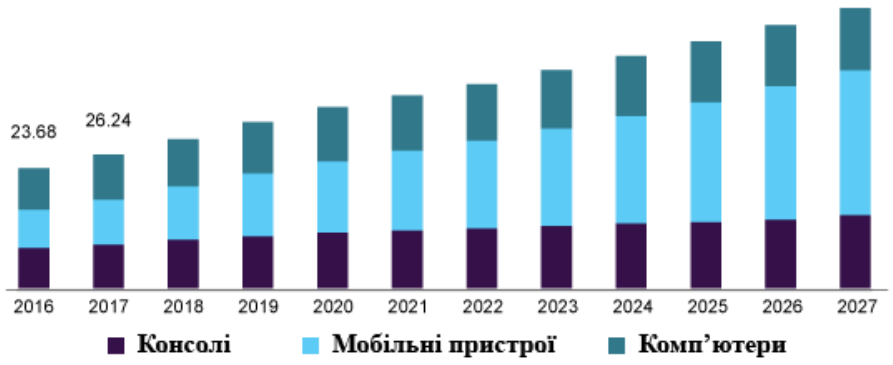


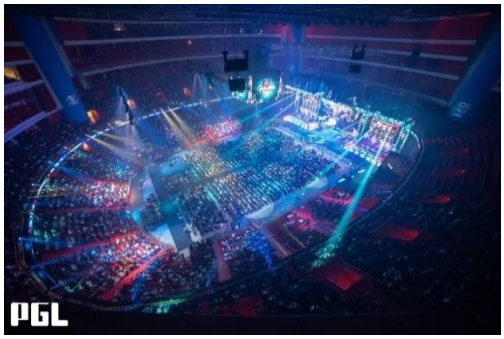
Рисунок 1.1. Динаміка розвитку ігрової індустрії

Зростаючий тренд переходу від фізичних ігор до онлайн-ігор змусив учасників галузі зосередитися на сумісності та ефективності апаратного забезпечення. Ігри, які пропонуються безкоштовно (Free2Play), масові багатокористувацькі онлайн ігри

(Massively Multiplayer Online, MMO) та ігри, спрямовані на велику кількість користувачів, поступово набувають популярності, і цей тренд очікується продовжитися протягом наступних восьми років. Зростаючий рівень доходів споживачів призводить до збільшення витрат на ігрові продукти. Крім того, зміни в уподобаннях споживачів сприяють широкому впровадженню більш вдосконалених ігрових консолей з такими складними функціями, як запис і спільне використання, а також багатоплатформені ігрові процеси [4].

Тенденція ігор у соціальних мережах матиме позитивний вплив на зростання ринку. Наприклад, значна частка населення світу використовує соціальні мережі, такі як Facebook і Reddit, для гри. Очікується, що доступність різних жанрів ігор, таких як екшн, рольові ігри, симулятори та стратегії, приверне більше клієнтів. Зростаюча популярність кіберспортивних турнірів і збільшення кількості професійних гравців призведуть до зростання продажів відеоігор, аксесуарів, ігрового обладнання та програмного забезпечення.

Протягом останніх років кіберспорт активно розвивається, як з точки зору призових коштів, так і з точки зору міжнародного визнання. Якщо ще десять років тому це була сфера діяльності обмеженої групи комп'ютерних фанатів, то тепер все більше підлітків з нетерпінням чекають можливості побудувати кар'єру в



кіберспорті.

Рисунок 1.2. Масштаб кіберспортивних змагань сьогодні

Кіберспорт уже був названий майбутнім у всьому спорті, особливо після спалаху COVID-19. Тепер стає цілком очевидним, що ця галузь є ідеальним вибором для підлітків, які захоплені грою і популярними дисциплінами кіберспорту.

Ринок відеоігор демонструє високий попит у різних сферах, таких як освітні заклади та корпоративні підприємства. Використання ігор як навчального інструменту надає можливість для більш глибокого та пізнавального навчання. Хоча поняття "ігри для навчання" існує вже давно, проте реальний потенціал гейміфікації в академічному середовищі тільки нещодавно отримав широке застосування.

Гейміфікація в процесі навчання передбачає використання ігрових елементів, таких як бали, конкуренція між однолітками, командна робота, оцінювальні таблиці, щоб стимулювати залучення, допомагати учням освоювати нову інформацію та перевіряти свої знання. Гейміфікація може застосовуватись як у шкільних предметах, так і в додатках та курсах самоосвіти, що свідчить про те, що переваги гейміфікації залишаються актуальними навіть для дорослих.

Технології пронизують значну частину нашого повсякденного життя, змінюючи спосіб, яким ми живемо, покупаємо, працюємо, граємо, харчуємося, знайомимося та спілкуємося з людьми. Політики починають розглядати потенційні переваги використання технологій для оптимізації робочого навантаження працівників.

Однак поширення фальсифікованої продукції за низькими цінами, особливо в країнах, таких як Китай та Індонезія, може призвести до обмеженого росту ринку. Проблеми, пов'язані з авторським правом та піратством, негативно впливають на досвід користувачів. Також зростає обурення користувачів щодо шахрайства під час транзакцій з грою, що також гальмує розвиток ринку. Різке зростання проблем зі здоров'ям, пов'язаних з відеоіграми, є ще одним фактором, який очікується, що стримуватиме зростання цієї індустрії. Прогнозується, що індустрія відеоігор продовжить зростати в наступні місяці, не зважаючи на вплив пандемії COVID-19 на світову економіку. Завдяки обмеженням, введеним урядами усього світу для запобігання поширенню вірусу, компанії спостерігають збільшення кількості користувачів та збільшення часу, який вони проводять, граючи онлайн-ігри. Крім того, деякі розробники вирішили випустити свої онлайн-ігри безкоштовно. Наприклад, у березні 2020 року компанія Activision Blizzard, Inc. безкоштовно

випустила гру "Call of Duty: Warzone", яка за один день зібрала близько 6 мільйонів завантажень [2].

Проте пандемія також призвела до затримок та перебоїв у випуску ігрових продуктів. Прес-конференції, на яких мали бути оголошені нові ігри та трейлери, також були відкладені. Багато з цих затримок пов'язані з закриттям виробничих потужностей в Китаї, де виробляється багато ігрових продуктів. Наприклад, у лютому 2020 року Nintendo Co., Ltd. оголосила про затримку випуску Nintendo Switch через закриття своїх виробничих потужностей через COVID-19.

Зростання популярності онлайн-ігор очікується і в майбутньому. Запит на багатокористувацькі ігри сприяє зростанню популярності онлайн-ігор, оскільки вони полегшують спілкування та покращують ігровий досвід. Соціальні медіа відіграють важливу роль у поширенні онлайн-відеоігор. Також очікується зростання інформованості про інтерактивні розважальні системи та збільшення кількості геймерів, які бачать ігри як форму розваги. Поширення смартфонів та хмарних ігор також стимулює зростання онлайн-сегмента. Розробники ігральних приставок акцентують на онлайн-можливостях. Наприклад, Xbox Live від Microsoft Corporation та PlayStation Network від Sony Corporation надають можливість грати онлайн. У 2019 році Азіатсько-Тихоокеанський регіон займав більшу половину ринку відеоігор, що було обумовлено розширенням ігрової індустрії в Китаї. Популярність смартфонів та попит на розваги в Китаї є основним фактором, що сприяє росту цього регіону. Tencent Holdings Limited, штаб-квартира якої знаходиться в Китаї, стала провідним гравцем на світовому ринку завдяки стратегії неорганічного зростання, такої як придбання Riot Games та Supercell Oy, розробників популярних ігор, наприклад, League of Legends та Clash of Clans. Росту китайських компаній приділяється велика увага, оскільки вони є важливим чинником загального росту індустрії в Китаї [7].

Збільшення кількості гравців в Інтернеті та зростання онлайн-ігрових турнірів в Азіатсько-Тихоокеанському регіоні спонукають постачальників запускати різноманітні платформи, які надають геймерам доступ до ігор високої якості. Наприклад, у грудні 2019 року Tencent у співпраці з NVIDIA запустила хмарний

ігровий сервіс START, який дозволяє геймерам отримувати доступ до AAA-ігор навіть на недостатньо потужних пристроях у будь-якому місці та в будь-який час. Південна Корея також очікується зробити вагомий внесок у регіональний розвиток завдяки зростанню кіберспорту та масовим багатокористувацьким онлайн-іграм [9].

Гравці ринку акцентують увагу на розробці цікавого контенту та наступного покоління ігрових консолей. Популярні консолі, такі як PlayStation 4 від Sony Corporation, Xbox One від Microsoft Corporation та Wii U від Nintendo Co., Ltd, розроблені цими компаніями. Розробники намагаються диференціювати свої продукти та вносити інновації, щоб збільшити свою частку на ринку.

Учасники ринку активно працюють над покращенням обслуговування клієнтів, пропонуючи продукти з різноманітними функціями, які дозволяють гравцям одночасно грати в ігри та переглядати веб-сторінки. Очікується, що такі багатофункціональні ігрові консолі сприятимуть загальному росту індустрії. Крім того, постачальники акцентуються на злиттях та поглинаннях малих та середніх компаній, щоб підтримувати свою конкурентоспроможність. Наприклад, в січні 2018 року Microsoft придбала PlayFab, постачальника ігрових послуг у режимі реального часу, для прискорення інновацій у розробці ігор на хмарній платформі.

На ринку відеоігор видатну роль відіграють компанії, такі як:

- Корпорація NVIDIA;
- Корпорація Sony.
- Rovio Entertainment Corporation;
- Корпорація Valve;
- ТОВ "PlayJam";
- Bluestack Systems, Inc.;

1.2 Фактори, які вплинуть на розробку ігор у майбутньому

З огляду на те, що більшість країн у світі впродовж великої частини 2020 та 2021 років застосовують соціальне дистанціювання, зацікавленість у цій масштабній

індустрії продовжує зростати, а люди, які раніше не виявляли інтересу до відеоігор, починають проявляти цікавість до них. Надзвичайний вільний час надав гравцям можливість розвивати нові навички та стратегії, які в майбутньому можуть бути показані на екранах телевізорів під час кіберспортивних змагань, що стають так само популярними, як і фінали Суперкубка або Чемпіонату світу.

Так само, як геймери на будь-якому рівні вдаються до пройдення будь-яких викликів у своїх іграх, вони також очікують від розробників використання нових методів та передових технологій, щоб просунути геймплей вперед, роблячи ігри ще більш реалістичними та складними, ніж будь-коли раніше. Ось кілька технологій, які відкривають шлях у майбутнє в галузі відеоігор.

1.2.1 Віртуальна та доповнена реальність

Віртуальна та доповнена реальність створюють захоплюючий іммерсивний ігровий досвід. Ці технології переносять ігри з фізично контрольованого світу в частково або повністю віртуальне середовище, що дозволяє гравцям ще глибше зануритися у гру. Зазвичай для ігор, що використовують віртуальну реальність, необхідні спеціальні гарнітури, які можуть бути не дуже зручними для користувачів. Однак, нові пристрої, такі як рукавички для зап'ястя, що незабаром з'являться на ринку, дозволяють розробникам ігор створювати тактильні елементи, з якими користувачі можуть взаємодіяти. Це відкриває абсолютно нові можливості для реалістичного ігрового процесу для широкої аудиторії. Таким чином, розробники зможуть привернути більше гравців, роблячи ігровий світ ще більш реалістичним.

Рисунок 1.3. Приклад застосування технологій доповненої реальності

UE5, новий ігровий двигун Unreal Engine 5, вважається ключовою силою в технологічному процесі створення ігор. AR/VR вже не є чимось особливим, але стає обов'язковим форматом [12].

Хоча багато звичайних людей ще не усвідомлюють цього, доповнена реальність вже давно використовується в масових розвагах. Перші застосування цієї технології з'явилися наприкінці 1990-х років, коли під час футбольних трансляцій було введено жовту лінію на полі, щоб глядачам було зручніше слідкувати за грою. Так само, ігри з доповненою реальністю поєднують елементи реального світу з віртуальними, як це можна бачити в мобільних іграх, наприклад, у Pokémon Go. Ігри з доповненою реальністю можуть привернути широку аудиторію, оскільки вони не вимагають дорогих консолей і можуть сподобатися навіть недосвідченим гравцям або початківцям. Ігрова індустрія доповненої реальності - це швидкозростаючий сектор в галузі відеоігор, вартість якого очікується перевищити 385 мільярдів доларів до 2023 року [5].

1.2.2 Хмарні технології

Постачальники хмарних послуг поступово перетворюються на нове покоління ігрових консолей. Широке поширення хмарних технологій та доступ до них змінили

підхід до створення, передачі та відтворення відео- та мобільних ігор. Час виходу на ринок значно прискорився. Тепер гравці можуть отримати доступ до нових ігор незалежно від свого місцезнаходження, якщо у них є доступ до Інтернету. Це суттєво скорочує час, необхідний для придбання ігор, пакетів розширень та додаткового контенту.

Сервіси, такі як Google Stadia, Xbox Game Pass і PlayStation Now, введені поняття хмарних ігор, яке було маловідомим ще десять років тому. Замість покупки консолі та фізичного диска, гру можна транслювати на будь-який пристрій, подібно до сервісу Netflix.

Хмарні ігри - це метод гри у відеоігри за допомогою віддалених серверів у центрах обробки даних. Немає потреби завантажувати та встановлювати ігри на своєму ПК або консолі. Замість цього, для потокових служб необхідне надійне підключення до Інтернету для передачі ігрової інформації до програми або браузера, встановленого на приймачеві пристрої. Гра відображається та відтворюється на віддаленому сервері, але користувач бачить та взаємодіє з усім локально на своєму пристрої. Єдине відмінність полягає в тому, що сервер, з якого надходить відеопотік, також може сприймати реакції користувача та реагувати на них. Це означає, що для хмарних ігор не потрібна потужна відеокарта RTX 30-ї серії або нова Xbox Series X або PlayStation 5. Все, що потрібно, це надійне підключення до Інтернету. Все це відбувається миттєво і виглядає так само, як ігра, завантажена на пристрій [8].

Наприклад, Microsoft переводить консолі Xbox на сервіси Xbox Cloud Gaming, де віртуальні консолі Xbox запускаються на їхніх серверних фермах, що забезпечує майже такий же досвід, як у домашньої консолі Xbox. Тепер Microsoft переходить на потужніше обладнання Xbox Series X, що поліпшує час завантаження, частоту кадрів та оптимізацію ігор, а також підтримує потокове відтворення на пристроях з більшим екраном.

У жовтні 2020 року Amazon також представив свій хмарний ігровий сервіс Luna, який надає необмежений доступ до ігор. Luna використовує спеціальний

локальний геймпад-контролер з окремим з'єднанням Wi-Fi, що допомагає зменшити затримку введення під час гри.

Це відкриває безліч можливостей. Тепер ви можете взяти свій телефон і насолоджуватися найновішими іграми AAA, або встановити хмарний ігровий додаток на Chromebook для гри в деякі портативні комп'ютерні ігри. Ось чому хмарні ігри настільки захоплюючі, але ця технологія все ще не розкрила свій повний потенціал.

Крім того, є інформація, що Netflix збільшує свою зацікавленість у геймінговому сегменті. Компанія розглядає можливість включення "пакетів" ігор у свою підписку. Цей сервіс схожий на той, який Apple запустив у вересні 2019 року під назвою Apple Arcade.

1.2.3 Новітні технології розробки ігор

Unreal Engine - це гральний рушій, що змінить гральну індустрію. Unreal Engine 5 з'явився у 2021 році і незабаром стане одним з основних інструментів, які дозволять всій гральній індустрії зробити крок уперед у якості графіки.

Як багатьом геймерам вже відомо, відеоігри складаються з тисяч полігонів, які відеокарта читає та інтерпретує. Unreal Engine 5 змінює спосіб взаємодії розробників з цими фігурами, запроваджуючи нову систему віртуалізованих полігонів.

Раніше розробникам доводилося жертвувати кількістю полігонів, а, отже, і якістю, щоб забезпечити геймерам плавну роботу. З Nanite від Unreal Engine розробники можуть створювати складну геометрію без втрати продуктивності.

Очікується, що остання версія Unreal Engine буде випущена на початку 2022 року. З випуском Unreal Engine 5 з'явиться безліч змін, які покращать будь-яку відеогру, створену за допомогою цього рушія. Ці зміни дозволять геймерам взаємодіяти з більш захоплюючими та просторими світами.

Процес анімації ігор повністю переглядається в Unreal Engine 5, що робить цей процес більш зручним для розробників [6].

Зокрема, Motion Wrapping є особливим інструментом, який входить до складу Unreal Engine. Завдяки Motion Wrapping розробникам не потрібно створювати окремі анімації для кожного рівня стрибка. Це дозволяє зробити анімацію персонажів більш плавною та захоплюючою під час взаємодії з оточенням.

Full-Body IK є ще однією функцією анімації, яка з'явиться у Unreal Engine 5. Вона дозволяє персонажам та об'єктам взаємодіяти з навколишнім середовищем в режимі реального часу природним чином. Персонажі будуть плавно коригувати своє положення, щоб врахувати, наприклад, рівень землі або спосіб взаємодії персонажів з відкритими дверима під час проходження.

Lumen - це нова система освітлення, яка спрощує створення реалістичного освітлення як для консолей, так і для ПК. Ця система освітлення взаємодіє з навколишнім середовищем, щоб створити більш реалістичний досвід. Світло, яке відображається у грі, тепер буде ідентичним до того, яке буде видно в остаточній версії, випущеній на консолі.

Тепер анімовані кат-сцени можна створювати безпосередньо в Unreal Engine з використанням Unreal Engine 5. Ця система означає менше попередньо відтворених відеороликів і більше рендерингу кат-сцен у грі. Це означає, що анімовані кат-сцени більше не потрібно передавати стороннім організаціям, і їх можна легко створити невеликою командою.

І, нарешті, Chaos Physics - це нова фізична система, яка була впроваджена в Unreal Engine і більш точно моделює фізику тканин та каміння. Це суттєва зміна, оскільки ця нова фізична система робить одяг більш реалістичним, ніж коли-небудь раніше.

Ці поліпшені анімації та фізика створять захоплюючий досвід та знищать межу між персонажем та навколишнім світом.

1.3 Сегментація аудиторії при розробці гри

Грамотна сегментація користувачів є одним з найважливіших факторів, який визначає успіх гри. Тому вміння вірно виділити цільову аудиторію на будь-яких платформах, таких як клієнтські, браузерні та мобільні, має велике значення.

Критерії, за якими геймдизайнери, аналітики та маркетологи виокремлюють аудиторію, включають:

- Демографічні дані, які включають вік, стать, місцезнаходження тощо.
- Вподобані жанри, які є характерними для ігор, кіно та розважальної галузі.
- Поведінкові особливості, включаючи психологічні типи гравців, які можуть бути класифіковані за допомогою певних моделей.
- Казуальність, яка відрізняє "хардкорних" гравців від "казуалів" і важко вимірюється числовими показниками.
- Схильність до інновацій і ставлення до нововведень, можливостей та типів геймплею.
- Платоспроможність, що має важливе значення для розробників.

Кожен сегмент цільової аудиторії можна описати за допомогою різних числових показників, таких як розмір аудиторії, конверсія, легкість переходу гравців до грального проекту, ступінь лояльності (retention) та здатність утримувати аудиторію (retention і sticky factor). В кожному сегменті існує безліч числових параметрів, які виділяються.

Ключовими характеристиками кожного сегмента є ROI (повернення інвестицій), LTV (життєвий цикл клієнта) і розмір цього сегмента. Вони частково включають такі важливі критерії, як середній дохід від кожного платника (ARPPU — середній дохід на одного платника), частка платних користувачів у кожному сегменті (PU) і retention (відданість аудиторії). На практиці існує набагато більше характеристик, які можна використовувати для виділення кожного сегменту. Наприклад, retention можна розраховувати за допомогою різних підходів. Класичний retention вимірює частку гравців, які зайшли у гру в конкретний день. "Retention

сьомого дня" означає, що зайшов гравець, який зареєструвався сім днів тому. Існує також rolling retention, коли гравець, що зареєструвався тиждень тому, зайшов у гру протягом наступних днів. Return retention враховує відвідування гравцем гри з другого до сьомого дня після реєстрації. Крім того, існує bracket dependent return retention і різні методи розрахунку цієї характеристики, які можуть варіюватися для кожного проекту.

Всі ці показники, як зазначено вище, можуть бути зведені до двох основних: lifetime value (довічний дохід з платника) і ROI (прибуток від інвестицій). ROI враховує не тільки дохід, отриманий від гравця, але й обсяг потенційних інвестицій у кожному сегменті аудиторії. Якщо розмір аудиторії малий, а LTV високий, то, ймовірно, інвестування в цей сегмент не має сенсу, оскільки повернення буде незначним через обмежений обсяг цільової аудиторії [2].

Крім числових характеристик, існують і такі, які не можуть бути виміряні числами. Наприклад, обсяг необхідного контенту в грі. Для деяких гравців у World of Warcraft можуть бути потрібні тисячі підземелень, великі поля бою, арени та багато іншого. У той же час, для любителів Candy Crush Saga можуть бути важливі тисячі простих рівнів, які схожі один на одного. Обсяг контенту, що виробляється, буде різним для цих двох гравців.

Графічний стиль також може варіюватися в різних іграх, залежно від жанру та цільової аудиторії. Це значно впливає на вартість розробки. Вартість створення графіки для League of Legends, World of Warcraft і Candy Crush Saga буде різною. Часто сеттинг та жанр впливають на графічний стиль, встановлюючи свої вимоги до графічного виконання. Існують різні види ігрових можливостей, які привертають різних гравців. Деякі гравці насолоджуються іграми, де можна створювати та виробляти різні речі. Іншим важлива монотонна механіка грінду та заточення. А деякі люди більше зацікавлені в убивстві, знищенні та домінуванні. Кожна з цих груп має свої вподобання щодо функцій, які вони шукають у іграх.

Розробка різних ігрових можливостей також має різну вартість. Коли ми визначаємо цільову аудиторію, ми повинні враховувати, які функції підходять для

кожної групи гравців. Демографічні показники, такі як стать, вік та регіон, грають важливу роль у визначенні вподобань гравців. Освіта також може впливати на вибір ігрових жанрів та готовність платити за гру [4].

Наприклад, різниця в освіті між людьми з середньою, середньою спеціальною та незакінченою вищою освітою не є суттєвою з точки зору прибутковості. Однак, люди з вищою освітою та школярі відрізняються як за розміром та лояльністю аудиторії, так і за прибутком, який вони можуть принести.

Регіональна приналежність також має значення при розробці ігор. Економічні особливості, умови ведення бізнесу та культурні аспекти регіону впливають на ринок ігор. Наприклад, в Африці та на Близькому Сході аудиторія гравців може бути меншою порівняно з іншими регіонами.

Останнім часом Китай перевершив США за обсягом ігрової індустрії. Середній дохід населення Китаю і Азії в цілому є вищим, але відсоток платників менший. Китайська ігрова індустрія раніше сильно підтримувалась браузерними іграми через обмеженість апаратного забезпечення та законодавчі заборони на консолі. Проте з появою доступних мобільних телефонів на базі Android, браузерні ігри швидко перейшли на мобільні платформи і стали великим сегментом китайського ринку ігор. Існує широкий спектр класифікацій ігор за жанрами, і деякі з них є традиційними термінами, що сформувалися в ігровій індустрії. Інші жанри використовують термінологію, яка була введена самими гравцями, наприклад, стрілялки, кульки, стратегії.

Різні жанри можуть мати різні чисельні характеристики, з яких дві найцікавіші - утримання (retention) та ARPU (середній дохід з гравця, розраховуваний зазвичай за місяць). Графік демонструє ці чисельні характеристики для сегмента мобільних ігор. Ліва частина відображає "семиденний retention", що вказує, скільки людей продовжує грати на сьомий день після реєстрації. Червона лінія відповідає правій шкалі і показує частку гравців, які платять.

Наприклад, у жанрі аркад ми спостерігаємо низькі рівні як утримання, так і платежів. Проте це не означає, що аркада є поганим або непотрібним жанром. Вони можуть мати більше активних гравців та генерувати достатній прибуток.

Отже, різні жанри ігор можуть мати різні чисельні характеристики, і важливо аналізувати їх для визначення успішності та прибутковості кожного жанру.

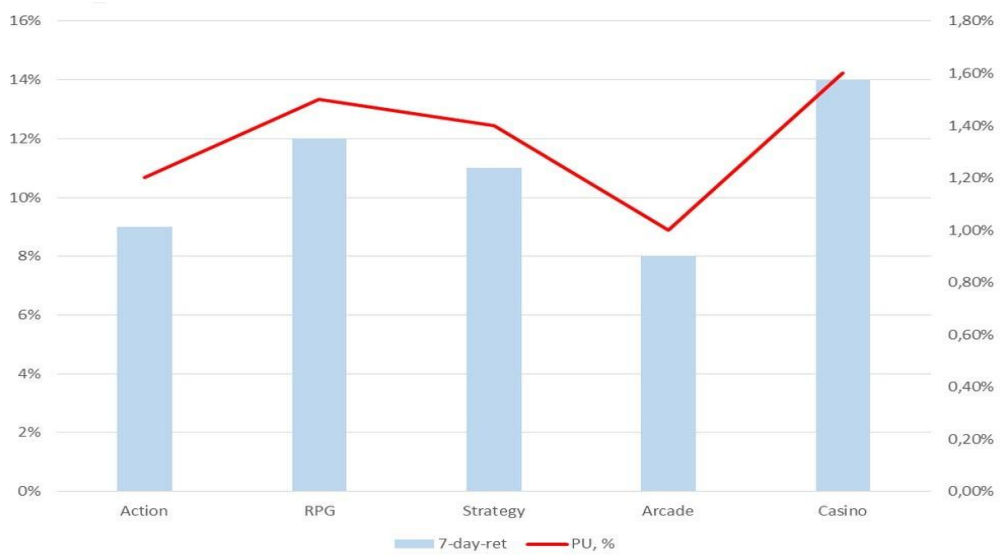


Рисунок 1.4. Чисельні характеристики жанрів мобільних ігор

Жанр казино є дуже прибутковим, особливо в соціальних мережах, де він набагато вигідніший, ніж у мобільному сегменті. На сьогоднішній день багато великих компаній успішно заробляють гроші на соціальних казино, незважаючи на затримку у розвитку соціальних мереж як ігрових платформ.

Стратегічні і рольові ігри (RPG) показують дуже стабільні результати. У цих жанрах високий відсоток платних користувачів і стабільне утримання (retention), але конкуренція в цій ніші досить серйозна.

Аудиторію можна поділити на п'ять основних груп залежно від ступеня прийняття інновацій, таких як нові ігрові можливості, жанри та особливості геймплею.

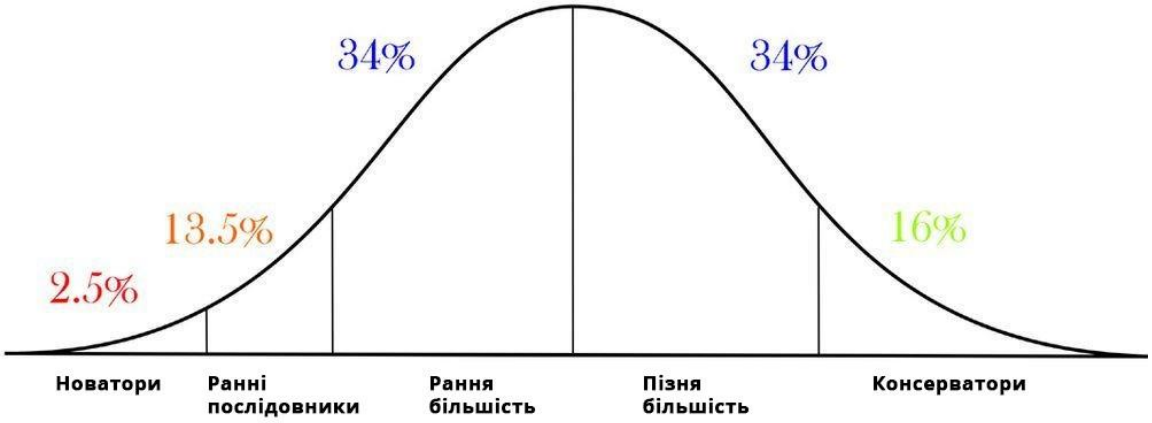


Рисунок 1.5. Групи гравців за ступенем прийняття інновацій

Найменша група, яку називають новаторами, складає приблизно 2,5% населення. Це люди, які творять щось нове, створюють ігри, гаджети та інші інноваційні речі. Вони постійно генерують нові ідеї, навіть якщо деякі з них можуть здатися безглуздими. Але серед їх ідей завжди є деякі цінні, які потім відбирають наступні послідовники, яких становить приблизно 13,5%. Ці люди створюють моду і допомагають поширювати ідеї.

Прикладом може бути Apple, коли вони просували свій смартфон. Вони орієнтувалися на групу ранніх послідовників, які були готові пропагувати цю ідею. Переміщення між цими групами або спрямування на кожну з них є вкрай складним, але завжди можна точно попасти в потрібний сегмент та мотивувати його поширювати ідеї. Цей підхід широко використовується в багатьох іграх.

Рання більшість, на відміну від послідовників, приймає продукт або ідею лише тоді, коли вони стають популярними. Наприклад, iPhone став модним, і представники ранньої більшості почали купувати його. Пізня більшість не приймає продукт, поки він не стане масовим і популярним серед всіх. А консерватори, остання група, починають приймати продукт, коли він вже втратив свою популярність. Наприклад, сьогодні консерватори - це ті, хто використовує старі телефони-розкладачі або Nokia і не приймають смартфони [3].

Багато сучасних ігор активно використовують цей принцип у своєму маркетингу. Два яскраві приклади цього - гра Candy Crush Saga та Clash of Clans. Розробники Candy Crush Saga добре розуміють, що людям важко перейти з однієї групи до іншої, і для цього потрібна сильна мотивація. Тому вони орієнтувалися на ранню більшість. Компанія King, перед тим як створити Candy Crush Saga, створила близько 200 ігор, зокрема соціальних, а не мобільних. Тому ринок був насичений іграми жанру "Match-3". Випустивши гру, вони назвали її масовим хітом, і це дало їм змогу залучити ранню більшість. І як тільки ця група користувачів зацікавилася грою Candy Crush Saga, до неї почала долучатися пізня більшість. Перехід між цими групами виявився простим, якщо повністю залучити ранню більшість.

Це можливо завдяки значним рекламним інвестиціям. Наприклад, компанія King у 2013 році заробила на грі Candy Crush Saga 2,2 млрд доларів. Щоб порівняти, обсяг української геймінгової індустрії на той час становив близько 1,6 млрд доларів. Зі зароблених 2,2 млрд доларів близько 1,7 млрд доларів було витрачено на маркетинг, зокрема на потужну рекламу по ТБ і в Інтернеті. Таким чином, компанія повністю охопила сектор ранньої більшості.

Інший приклад - компанія Supercell з грою Clash of Clans. Вони орієнтувалися на ранніх послідовників. Гра Clash of Clans створила практично новий жанр, якого раніше не існувало. Вона поєднала жанри "city builder" і "tower defense" у форматі асинхронних битв, що привело до появи нового жанру - Clash of Clans. Ранні послідовники зробили гру популярною, а завдяки великим зусиллям компанії до гри почала долучатися рання більшість. Пізня більшість поки не приймає Clash of Clans. За статистикою 2015 року, Clash of Clans є однією з найбільш прибуткових мобільних ігор у світі.

Таким чином, використання кривої прийняття інновацій дозволяє оптимізувати маркетинг і залучити різні групи користувачів до продукту. Одним з важливих факторів сегментації аудиторії є їх платоспроможність. Існує спосіб розділення аудиторії за обсягом здійснених платежів:

- Перша група складається з неплатників, яких становить більшість. У мобільному сегменті їх кількість перевищує 90%, за деякими даними навіть 97%.
- Друга група - це випадкові платники, яких становить близько 8%. У мобільному сегменті лише 1,5-2% є регулярними платниками. Ситуація подібна у соціальних іграх. У браузерних іграх близько 5% гравців роблять платежі, хоча існують винятки, наприклад, у деяких браузерних іграх, таких як "Легенда" та інші, показник платників може перевищувати 15%.
- Третя група - це регулярні платники, яких також можна розділити на три підгрупи. Перша підгрупа вносить невеликі суми, всього кілька доларів, але їх кількість становить основну частку платників і приносить компанії 15-30% виручки. Представники другої підгрупи вносять більше, але основну частку виручки формує третя підгрупа. Їх кількість невелика, але кожен з них приносить високі прибутки, які компенсують решту.

У Китаї, наприклад, монетизація ґрунтується виключно на представниках третьої підгрупи регулярних платників. Тому, хоча у грі лише 0,5% користувачів здійснюють платежі, кожен з них приносить приблизно 1000 доларів щомісяця.

1.4 Порівняльний аналіз найпопулярніших ігрових пристроїв

На сьогоднішній день, у 2021 році, геймери мають неймовірну удачу мати доступ до різноманітних платформ для своїх улюблених ігор. Три основні ігральні платформи - мобільні, персональні комп'ютери та консолі - є надзвичайно поширеними.

На кожній з цих трьох платформ геймери можуть насолоджуватись величезною бібліотекою ігор. Вони мають доступ до найпопулярніших відеоігор, а також до менш традиційних жанрів, наприклад, словесних ігор та навіть азартних додатків.

Кількість гравців у відеоігри значно збільшилась на 99% з 2019 по 2020 рік, і це частково пов'язано з COVID-19. Оскільки все більше людей залишаються вдома і проводять більше часу в колі відеоігор, це сприяє зростанню популярності геймінгу.

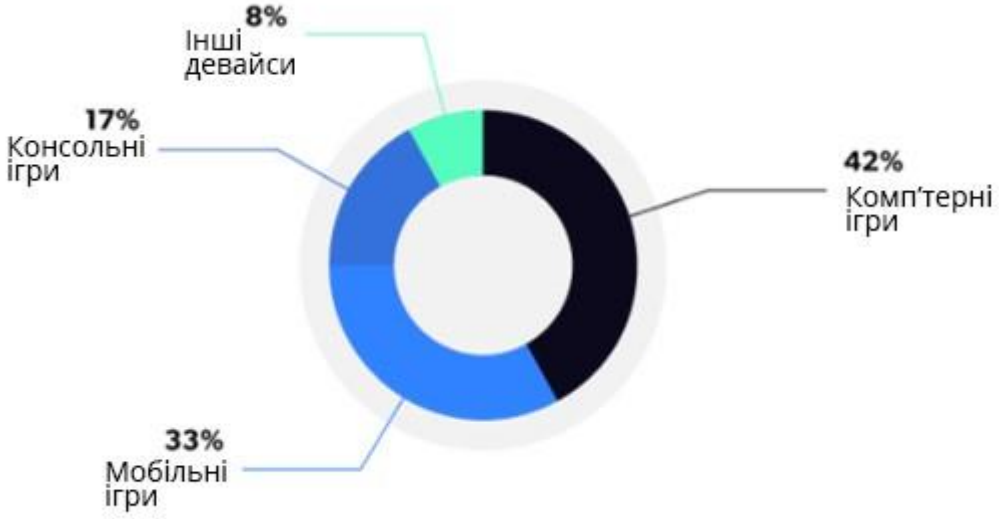


Рисунок 1.6. Статистика популярності ігрових пристроїв

1.4.1 Мобільні ігри

Популяція користувачів смартфонів по всьому світу продовжує зростати, і у 2020 році їх кількість перевищила 3,5 мільярда. Ці люди мають можливість мати світ ігор в своїх кишенях і грати в них практично скрізь і в будь-який час. Вже не йдеться лише про прості ігри, такі як Snake, хоча ця гра стала легендарною.

Завдяки потужності та технологіям сучасних смартфонів, ігри на мобільних пристроях стають потужнішими та захоплюючішими, ніж будь-коли раніше. Компанії, такі як Razer та Asus, навіть почали випускати спеціальні ігрові телефони з вражаючими характеристиками та спеціальними кнопками, призначеними саме для ігор [6].

Більшість мобільних ігор є коротшими та менш графічно насиченими, ніж їхні версії для ПК або консолей, але це не означає, що вони не можуть бути так само захоплюючими. Також існує величезне розмаїття ігор, які ідеально підходять для

мобільних пристроїв. Наприклад, гри у стилі Candy Crush є надзвичайно популярними і добре працюють на мобільних платформах.

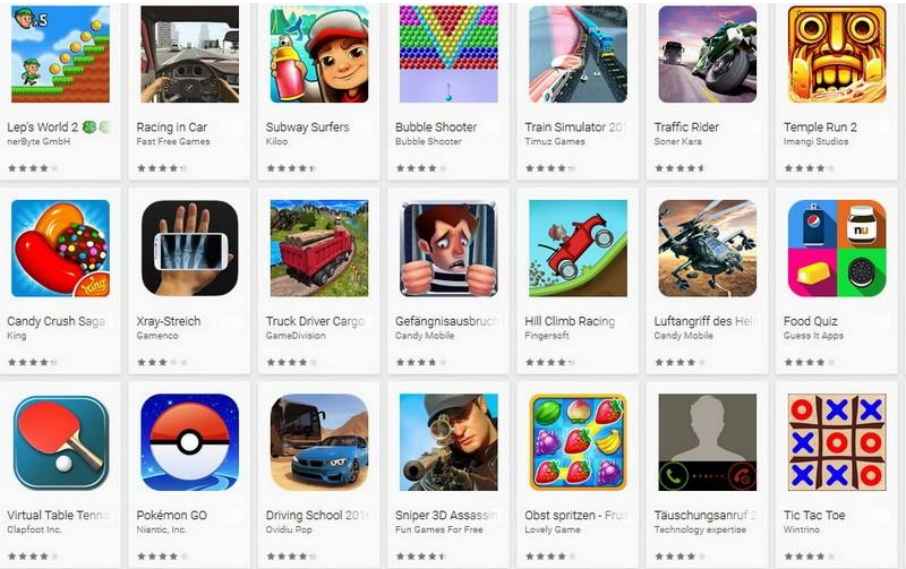


Рисунок 1.7. Найпопулярніші мобільні ігри

В загальному, мобільні ігри є надзвичайно переносним варіантом порівняно з будь-якою іншою ігровою платформою, які зазвичай є значно доступнішими в ціновому відношенні. Хоча вони можуть бути коротшими і мають меншу якість порівняно з іграми на ПК або консолях.

Основні переваги мобільних ігор включають:

- Портативність, що дозволяє гравцям грати в будь-який час і в будь-якому місці.
- Доступність, оскільки всі користувачі смартфонів можуть завантажувати ігри безпосередньо на свої пристрої.
- Невисока ціна ігор, що дозволяє отримати задоволення від ігор без значних витрат.
- Різноманітність типів ігор, що дає змогу знайти ігри, що відповідають інтересам різних гравців.

1.4.2 Комп’ютерні ігри

Багато геймерів мають тверду віру, що ПК є найкращою платформою для гри. Хоча це не є правдою, але ПК має достатньо переконливих аргументів, щоб бути вважаним однією з найкращих ігрових платформ взагалі.

У порівнянні з текстовими пригодницькими іграми 70-х років, сучасні комп'ютерні ігри пропонують неперевершені можливості, особливо щодо графіки та обчислювальної потужності.

Ігри на ПК забезпечують високий рівень налаштування, що є ключовим фактором їх популярності. Гравці мають можливість створити ПК, який відповідає різним бюджетам та потребам. Крім того, ПК можна легко модернізувати, що дозволяє залишатися на крок перед найновішими технологіями або покращувати свою систему з часом.

Геймери можуть підняти свій ігровий досвід на будь-який рівень, починаючи зі стандартного налаштування або глибоко зануритися у світ збірки, налаштування та модифікації.

Крім неймовірних ігрових можливостей, ПК також можна використовувати для різних завдань, включаючи роботу. Це робить його універсальним пристроєм, не обмеженим лише іграми. Звичайно, у порівнянні з консолями, геймерам на ПК потрібно мати більше технічних знань та досвіду для створення системи, яка перевершить консоль за той самий бюджет.

Останньою перевагою для геймерів на ПК є те, що ігри, як правило, коштують трохи менше. Однак, деякі ігри є ексклюзивними для консолей, тому вони недоступні для гри на ПК.

Основні переваги ігор на ПК включають:

- Найвища доступна продуктивність, що забезпечує високу якість графіки та швидкодію гри.
- Більш доступна ціна ігор порівняно з консольними.

- Можливість налаштування та оновлення системи відповідно до власних потреб та бюджету.
- Універсальність використання ПК для інших завдань поза іграми.

1.4.3 Консольні ігри

Консоль - це спеціалізоване графічне пристроєм, призначене для відеоігор. Дві найпопулярніші консолі - PlayStation від Sony та Xbox від Microsoft. Крім того, існує також консоль Wii від Nintendo, яка спеціалізується на імітації фізичної активності, такої як боулінг та теніс.

Ігрове програмне забезпечення для консолей доступне на компакт-дисках або DVD-дисках, хоча раніше використовувалися картриджі з чіпами пам'яті. Для гри на відеоігрових консолях потрібен телевізор або монітор.

Консолі використовують операційні системи та процесори, що відрізняються від настільних комп'ютерів. Вони керуються виробниками консолей, а програмне забезпечення адаптоване до можливостей самої консолі. Ігри для консолей несумісні з іграми для інших консолей або настільних комп'ютерів, хоча розробники можуть створювати ігри для кількох платформ [7].

Магнавокс Одіссей був першою консоллю, яка дозволила грати в ігри вдома на телевізорі ще у 1972 році. З тих пір консольні ігри швидко розвивалися як індустрія, випускаючи тисячі ігор щороку. Найбільша бібліотека відеоігор належить Sony PS2, яка містить понад 3800 назв. Це вражаюча кількість ігор, доступних у одній невеликій коробці.

Основна перевага консольних ігор полягає в їх простоті використання та зручному інтерфейсі. Придбавши ігрову консоль, ви отримуєте пристрій, спеціально розроблений для гри вдома. Вам лише потрібно розпакувати його, підключити, зробити кілька налаштувань і почати грати в ігри. Сучасні консолі навіть можуть виконувати роль повноцінних розважальних систем завдяки наявності програм для потокового відтворення відео та музики.

Хоча консолі не настільки потужні, як високопродуктивні геймінгові ПК, їх придбання є більш доступним. На жаль, нові консольні ігри не отримують таких знижок, які доступні на ринку ПК.

Коли розмова йде про самі ігри, вони завжди є ексклюзивними. Кожен виробник консолей резервує деякі ігри, які доступні тільки для його консолі. Наприклад, однією з найпопулярніших ігор останніх років є "God of War", що є ексклюзивом для PlayStation 4. Наявність ексклюзивних ігор для конкретної консолі є вирішальним фактором для багатьох геймерів під час вибору нової ігрової системи.

Переваги ігрових консолей:

- Висока доступність.
- Простота в експлуатації.
- Відносно низька вартість для початку гри.
- Доступ до ексклюзивних консольних ігор.

1.4.4 Портативні ігрові консолі

Мобільні відеоігри - це зменшені версії ігрових приставок з меншою складністю. Вони є повністю портативними самостійними пристроями з вбудованими батареями та компактними екранами.

Кишенькові ігрові пристрої були надзвичайно популярними наприкінці 80-х і протягом 90-х та 2000-х років, але вони зазнали застою, оскільки більшість гравців перейшли на домашні консолі або настільні комп'ютери. Однак завдяки успіху Nintendo Switch (а також портативної 3DS до нього), яка лідирує серед портативних консолей, а також зростанню мобільних ігор на смартфонах і розвитку сервісів потокової гри в хмарі, портативні ігри повертають свою популярність.

Портативні ігри існують вже десятиліттями, такі як Nintendo 3DS, DS, PlayStation Vita, PSP, Game Boy і Game & Watch. Але зараз спостерігається своєрідне оживлення. Багато з цих пристроїв є унікальними пристроями, які обіцяли вийти

протягом декількох років і, нарешті, будуть випущені після затримок у виробництві, пов'язаних з COVID.

Наприклад, PlayStation Portable, Nintendo Game Boy і ранні консолі Sega Game Gear та Atari Lynx є деякими прикладами таких пристроїв.

Рисунок 1.8. Портативна консоль Nintendo Switch

1.4.5 Порівняння пристроїв для VR-ігор

Ігри віртуальної реальності (VR) є останньою модною тенденцією на ринку. Як консолі, так і ПК мають свої позиції на цьому ринку. З консолей лише PlayStation 4 насправді пропонує VR-гарнітуру, тоді як у ПК є кілька доступних гарнітур, серед яких найпопулярнішими та найпотужнішими є HTC Vive та Oculus Rift.

З технічного погляду PS VR поступається пропозиціям ПК, як через свою застарілу технологію відстеження, так і через меншу потужність самого PlayStation 4. Крім того, кількість ігор для PS VR є меншою, що знижує його загальну привабливість. Для VR також рекомендується використовувати потужний ПК [4].

Неможливо визначити єдину ігрову платформу, яка буде абсолютно найкращою. Це дійсно залежить від очікувань та цілей кожного користувача. Якщо важливо мати доступ до ігор поза домом, то мобільні ігри є найкращим варіантом. Якщо мета - грати в ігри на максимальних налаштуваннях із високою продуктивністю, то краще використовувати ігровий ПК. Якщо потрібна

спеціалізована ігрова система, яка буде простою у використанні, має багато ігор і навіть ексклюзивні творіння, тоді ігрова консоль може бути відмінним вибором.

РОЗДІЛ 2. ТЕХНОЛОГІЇ РОЗРОБКИ ІГРОВИХ ПРОДУКТІВ

2.1. Визначення етапів розробки ігор

Для досягнення успішно реалізованого проекту важливо приділяти багато уваги його розробці. Проте під час розробки, розробникам можуть приходити нові ідеї, які можуть значно змінити попередньо визначену концепцію гри. Щоб запобігти цьому, важливо встановити чіткі етапи розробки фінального продукту. Тому процес розробки гри зазвичай описується за допомогою пайплайну - послідовності стадій, в яких розташовані завдання. Кожен елемент пайплайну є входом для наступного, що створює ефект конвеєра. Через пайплайн генеруються ідеї, приймаються рішення і реалізуються нові функції. Пайплайн представляє робочі процеси, дії та автоматизацію, необхідні для швидкого переходу від ідеї до випуску гри [3].

Для контролю над кожною стадією використовуються чотири основні метрики:

1. Час процесу - це час, необхідний для виконання однієї стадії (одного кроку).
2. Час виконання - це час, який потрібен від моменту виконання роботи на попередньому етапі до виконання на поточному. Іншими словами, це час затримки від останнього кроку, доданий до часу обробки поточного кроку.
3. Час очікування - це час, протягом якого робота знаходиться в очікуванні між стадіями.
4. Час виконання усього процесу - це загальний час, необхідний для завершення всього пайплайну.
5. Час виконання - це час, який пройшов від моменту виконання роботи на попередньому етапі до виконання на поточному. Іншими словами, це сума часу затримки з останнього етапу та часу обробки поточного етапу.
6. Час процесу - це час, необхідний для виконання однієї стадії (одного етапу).
7. Час процесу - це час, необхідний для виконання однієї стадії (одного етапу).

8. Час виконання - це час, який пройшов від моменту виконання роботи на попередньому етапі до виконання на поточному. Іншими словами, це сума часу затримки з останнього етапу та часу обробки поточного етапу.
9. Затримка - це період, коли робота не відбувається. Цей показник має бути якомога нижчим.
10. Відсоток виконаних робіт на кожному етапі без необхідності повторної обробки. Часті переробки на різних етапах призводять до збільшення часу реалізації та затримок.

Рисунок 2.1. Пайплайн створення ігрового продукту

2.1.1 Концепція

Створення гри завжди починається з ідеї, яка ляже в основу всього подальшого сюжету. Деталі цієї ідеї можуть змінюватись від графічних технологій до глобального контексту гри.

Процес створення гри є еволюційним. Протягом існування проекту можуть бути кілька змін основних моментів гри, включаючи жанр, а також дрібніші нюанси. Для стійкості гри в програмуванні потрібна міцна концепція. Створення концепції не може бути процесом, який відбувається поетапно. Спочатку слід створити концепцію, а потім переходити до її реалізації.

Перш за все потрібно визначитися з основною ідеєю. Головна ідея створює простір для майбутньої роботи. Можна провести аналогію між концепцією проекту та математичним поняттям базису - набором векторів, що однозначно задають простір. Процес створення концепції гри є складнішим, оскільки він не так математично визначений.

Для створення успішної гри необхідно постійно знаходити та генерувати ідеї, які привертають користувачів. Інспірацію можна знайти і в реальному житті, але є три основні напрями для пошуку ідей:

1. Аналіз популярних трендів на ринку.

2. Проведення арт-конкурсів серед геймерської спільноти.

3. Розробка унікальної концепції для привернення уваги.

Часто розробники ігор використовують арт-конкурси серед гравців. Гравці пропонують власні варіанти зброї чи персонажів, і потім сам

і голосують за найкращі, які потім додаються до гри. Так утворюється виграшна ситуація: розробники отримують нові ідеї (іноді навіть готові концепти), а гравці отримують призи, визнання, можливість соціалізації та додаткову захопленість.

2.1.2 Прототип

Цей етап розробки є критичним і відбувається безпосередньо перед фактичним виробництвом з метою перевірки основної концепції та ідеї. З наявністю доброго прототипу необхідно бути повністю усвідомленим усіх складностей, що можуть виникнути на шляху до успішного проекту, та бути готовим вирішувати їх. Це перший і вирішальний момент, на якому визначається подальша доля проекту. Якщо результати тестування першого ігрового прототипу задовільні, це означає, що можна перейти до повної розробки гри. Однак важливо прийняти рішення зупинити розробку та повернутися на попередній етап або переосмислити ідею, якщо цей етап виявив якісь недоліки чи проблеми.

2.1.3 Перший ігровий прототип

Перший ігровий прототип (FPP) представляє собою першу робочу версію гри. Хоча це ще не альфа-версія, вона все ж більш повна і виразна, ніж звичайний

прототип. Головна мета першого ігрового прототипу полягає в тому, щоб продемонструвати справжню сутність гри, а не просто концепцію. Хоча перший ігровий прототип не надає повного ігрового досвіду (часто відсутнє головне меню, рейтинг гравців або реальна історія), в ньому присутні основні елементи, які можна випробувати [1].

Одна з основних відмінностей між першим ігровим прототипом і звичайним прототипом - це арт-робота. Хоча використання шаблонів і неякісних ресурсів у прототипі є припустимим, вони повинні бути замінені на графічні об'єкти кращої якості, навіть якщо вони не є остаточними.

По завершенні цього етапу розробники мають готовий продукт, який можна показати потенційним інвесторам, не обтяжуючи їх серйозними помилками або тимчасовими недоліками. На цьому етапі вже достатньо контенту, щоб зрозуміти сюжет гри.

Демонстрація гри на цьому етапі використовується для перевірки дизайну, тестування ігрової механіки та внутрішнього тестування. Може бути впроваджене тестування залученими користувачами, які надають зворотний зв'язок, на основі якого вирішується, чи продовжувати розробку гри.

2.1.4 Alpha версія гри

Загалом, альфа-версія є одним з перших виявів програмного продукту і відповідає альфа-фазі його розробки. Зазвичай на цьому етапі внутрішні тестувальники компанії використовують метод "білого ящика", що передбачає огляд вихідного коду та оцінку функціональності.

Альфа-версія включає в себе повний контент гри та сюжет і є повноцінною грабельною версією. На цьому етапі можна починати працювати з постачальниками техніки для оптимізації та портативності на різні платформи. Також можна надати журналістам доступ до гри.

Основна мета альфа-версії полягає в створенні зручного, ефективного та привабливого ігрового досвіду. По завершенні альфа-версії гра повинна відповідати наступним критеріям виходу:

- Реліз є майже повним.

- Новий гравець з цільової аудиторії може грати у гру з мінімальним зовнішнім керівництвом.

- Ілюстрації та інтерфейс достатньо мотивують гравців для гри в основний сюжет.

- Продуктивність гри є мінімально прийнятною на основній цільовій платформі.

- Гравець з цільової аудиторії виражає бажання продовжувати грати самостійно.

- Реалізовано ключові звукові ефекти.

- Основний ігровий цикл чітко демонструє інтеграцію навчання у геймплейну механіку.

2.1.5 Beta версія гри

«Бета» - це стандартний термін, який використовується для позначення етапу випуску продукту, на якому функціональність гри включена та оптимізована (хоча можуть бути присутні помилки), ігровий контент завершений (але можуть бути проблеми з його реалізацією), і загалом вважається майже готовим. Бета представляє собою загальну картину того, якою буде гра, а зміни вмісту або функціональності після бета-тестування зазвичай вважаються зламом публічної угоди (так званого "управління змінами").

Інтернет-видавці часто публікують бета-версію програмного забезпечення як "відкриту бета-версію". Відкриті бета-версії запрошують клієнтів грати у гру та повідомляти про виявлені проблеми, брати участь у спільноті, що очікує гру, перед її

запуском. Відкриті бета-версії також є ефективним способом залучити гравців до маркетингової історії ще на ранньому етапі.

Бета-версія потрібна для дороблення ігрового контенту та виправлення помилок у коді. На цьому етапі перевіряється якість та правильність гри на різних платформах: поліпшуються налаштування, виправляються критичні помилки, проводиться інтенсивне тестування [10].

Отже, мета бета-версії полягає в створенні гарного, функціонального та повноцінного досвіду. Ігри, що пройшли бета-тестування, відповідатимуть наступним умовам:

- Всі функції повні та дороблені.
- Час проходження гри для цільової аудиторії повністю реалізований.
- Усі письмові матеріали професійно озвучені.
- Відсутні заповнювачі відсутніх елементів.
- На титульному екрані присутній справжній заголовок та логотип.
- Назва остаточно узгоджена.
- Усі художні матеріали відповідають основному стилю.
- Більшість нових гравців можуть грати без зовнішнього керівництва.
- Багато гравців з цільової аудиторії грають більшу частину основного ігрового контенту за власним бажанням.
- Усі звукові ефекти та музика вбудовані та відповідають основній темі.
- Усі взаємодії мають звукові ефекти.
- Гра працює з цільовою частотою кадрів у більшості тестових сценаріїв.
- Грабельність збережена в усіх протестованих сценаріях.
- Усі функції та контент завершені.
- Інтерфейс користувача завершений і відповідає основному стилю, за винятком заставки та титрів.

2.1.6 Фінальна збірка

Передостаннім етапом у процесі виробництва є фінальна збірка, де втілені всі особливості гри. Кожне рішення, що було прийнято, повинно бути реалізовано, щоб забезпечити повноцінний досвід для кожного гравця.

Фінальне складання має особливу важливість і потребує більш інтенсивного тестування, ніж попередні етапи розробки. Необхідно стежити за всіма основними проблемами, з якими зіткнулися ранні тестувальники, а також намагатися вирішити більшість вторинних проблем.

Ця версія гри, створена до дати релізу, включає виправлення всіх проблем, виявлених після бета-релізу [12].

Цю збірку також називають "Gold master", що означає, що вона готова для поширення. Гри на цьому рівні будуть готові до випуску та відповідатимуть наступним критеріям випуску:

- Цей реліз повністю налагоджено та перевірено.
- Усі інтерфейси завершено.
- Озвучення відповідає сценарію.
- Нові гравці можуть грати без зовнішнього керівництва.
- Відсутні дефекти різних рівнів.
- Всі звукові ефекти та озвучення вирівняні.
- Присутня достатня різноманітність звукових ефектів та музики, щоб уникнути помітного/відволікаючого повторення протягом цільового часу відтворення.
- Гра працює з цільовою частотою кадрів у всіх протестованих сценаріях.

2.1.7 Post Launch період

Цей період має велике значення з точки зору підтримки, оскільки будь-яка серйозна проблема може призвести до зниження продажів або негативних відгуків

користувачів. Також до цього етапу входить розробка, тестування та підтримка додаткового контенту, що випускається для гри.

Після випуску гри підтримка є необхідною для забезпечення постійної взаємодії з користувачами. Отримання відгуків від користувачів допомагає визначити, чи слід продовжувати розвивати свій напрям роботи або внести зміни. Для отримання максимально корисної інформації необхідно залучати користувачів до надання всієї доступної інформації та постійно аналізувати зворотний зв'язок. Залучення користувачів до залишення коментарів через соціальні мережі є чудовим способом отримання цінних та перевірених відгуків, необхідних для розвитку гри. Інший спосіб - підтримувати структуровані цикли зворотного зв'язку, наприклад, за допомогою листування з користувачами електронною поштою та проведення опитувань.

Аналітика є одним із найефективніших та найцінніших способів отримання інформації про перших користувачів, і вона не вимагає багато ресурсів для виконання. Зазвичай достатньо вставити код відстеження. Аналіз статистики дозволяє глибше вивчити користувачів та їхню поведінку і надає масу даних, які зазвичай не можна отримати безпосередньо через відгуки користувачів, такі як геолокація, реферальні веб-сайти, час відвідування, перегляди сторінок, дії. Також можна вимірювати ефективність програмного продукту на різних платформах та інші показники.

Тестування користувачів надає достовірні та чесні оцінки. На відміну від друзів та родичів, які можуть прикрашати свої коментарі, на цьому етапі збираються відгуки анонімних незнайомців, які легко повідомлять, що саме їм подобається і що не влаштовує у грі. Тестування користувача допомагає виявити приховані недоліки юзабіліті та точніше зрозуміти, що хоче аудиторія, щоб здійснювати розрахункові зміни та покращувати ігровий досвід.

Наявність спеціальної групи системних адміністраторів, які стежать, обслуговують та надають підтримку, забезпечує високий рівень довіри до ігрового

продукту та гарантує користувачам працюючу програму без критичних помилок та проблем з продуктивністю [11].

Обслуговування та підтримка після випуску схожі на страхування подорожей для програмного продукту. Єдина різниця полягає в тому, що є два з 10 шансів отримати травму у відпустці, або приблизно 10 з 10 шансів зіткнутися з технічними проблемами гри - це необхідна складова природи IT-індустрії.

Це також ідеальний час для представлення ігрового продукту інвесторам. З наявною працюючою програмою, відгуками користувачів і планом дій щодо подальшого розвитку, є можливість пропонувати свій продукт як бізнес, а не просто ідею.

2.2 Ігрові движки та їх порівняльна характеристика

Ігрові движки є інструментами, які дозволяють дизайнерам ігор швидко і легко програмувати та планувати гру, не починаючи з нуля. Вони надають інструменти для створення та розміщення елементів незалежно від того, чи це 2D-ігри, чи 3D-ігри.

Таким чином, ігрові движки є архітектурою, на яку розробники ігор можуть покладатися для запуску відеоігри. Вони містять різні компоненти, такі як введення, рендеринг, сценарії, виявлення зіткнень, штучний інтелект і т. д. Ці двигуни є багаторазовими елементами, які дозволяють розробникам створювати основу гри, зекономлюючи час і зусилля. Це дозволяє їм більше уваги приділяти унікальним аспектам, таким як моделі персонажів, текстури, взаємодія об'єктів тощо.

Без ігрових движків розробники повинні були б створювати ігри з нуля, що потребувало б більше часу та було б складніше. Таким чином, ігрові движки суттєво спрощують процес дизайну ігор. Розробники та менеджери ігор повинні ретельно обирати платформу ігрового двигуна для своїх потреб, забезпечуючи, що вона вирішує кілька проблем одночасно.

Один з ключових аспектів, який потрібно враховувати при виборі ігрового двигуна, це здатність спрощувати такі завдання, як фізика, введення та обробка

візуальних ресурсів. Ігровий двигун повинен забезпечувати ідеальний баланс між якістю моделювання фізики та обмеженнями обчислювальної потужності. Він також повинен добре працювати з введенням, особливо при кросплатформеній розробці. Крім того, він має полегшувати обробку візуальних ефектів, таких як освітлення, тіні, текстури і глибина чіткості.

Отже, обраний ігровий двигун повинен дозволяти виконувати ці завдання з меншими зусиллями при написанні коду. Це допомагає скоротити час розробки і дозволяє командам зосередитися на розробці своїх ігор, забезпечуючи унікальний і особливий інтерфейс користувача [9].

2.2.1. Unreal Engine

Один з найпопулярніших та широко використовуваних ігрових двигунів належить Epic Games і має назву Unreal Engine. Цей мультиплатформний двигун призначений для розробки ігор будь-якого розміру і дозволяє використовувати технології реального часу для перетворення ідей у захопливий візуальний контент.

Unreal Engine більше не обмежений студіями з великим бюджетом, але став доступним для геймдизайнерів будь-якого рівня підготовки, які тепер можуть реалізувати свої ідеї в епічних іграх. Випущений початково у 1998 році, цей висококласний ігровий двигун продовжує бути популярним і використовуватися в найбільших іграх. Його основна перевага полягає в гнучкості, яку він надає розробникам, дозволяючи створювати унікальні ігрові досвіди. Однак для цього необхідні кваліфіковані розробники з великим досвідом [10].

Unreal Engine написаний мовою C++ і підтримує різні операційні системи та платформи, включаючи Microsoft Windows, Linux, Mac OS, Xbox, PlayStation і Nintendo GameCube. Він також підтримує мобільні платформи, такі як iPod Touch, iPhone і Palm Pre. Для спрощення портування, двигун використовує модульну систему залежних компонентів, включаючи системи рендерингу, звуку, голосового відтворення тексту, мережеві модулі та пристрої вводу [9].

Unreal Engine також підтримує гру в мережі за допомогою технологій Windows Live, Xbox Live і GameSpy, що дозволяє підключати до 64 гравців одночасно. Хоча офіційно не підтримується велика кількість клієнтів на одному сервері, рушій успішно використовується для створення MMORPG-ігор, включаючи Lineage II.

Крім того, Epic Games придбала Quixel, що має вражаючу бібліотеку фотограметричної графіки, яку можна використовувати для створення анімації та відеоігор. Користувачі Unreal Engine мають безкоштовний доступ до інструментів Quixel (Bridge, Mixer) та ресурсів бібліотеки Quixel Megascans.

Рисунок 2.2. Набір зразків бібліотеки Quixel

Unreal Engine є одним із найпопулярніших ігрових двигунів. Незважаючи на те, що розробникам подобається UE4, ранній випуск Unreal Engine 5 (UE5) містить довгоочікувані поліпшення та захоплюючі нові функції. Unreal Engine 5 - це майбутня версія Unreal Engine, а доступ до нього був оголошений у травні 2021 року, повний випуск UE5 очікується на початку 2022 року.

Основною перевагою Unreal Engine є його універсальність та доступність, що дозволяє використовувати його як для досвідчених розробників, так і для новачків, які тільки починають свій шлях у створенні ігор. UE4 володіє двома мовами програмування за замовчуванням: текстовим C++, який вимагає написання коду, і візуальною мовою Blueprints, де ігрова логіка будується з використанням блоків, які

з'єднані між собою. Цей підхід допомагає зробити програмування більш наочним та зрозумілим для тих, хто не має великого досвіду.

Blueprints - це візуальна, нодова система програмування, що використовується в Unreal Engine. За допомогою збирання логічних блоків нодів можна створювати програми різної складності, починаючи з простих клікерів і закінчуючи повноцінними RPG-іграми. Оскільки Blueprints не використовує програмний код, кожен, хто розуміє основні принципи ООП, може створювати програми.

За допомогою Blueprints розробники можуть створювати різноманітні елементи, такі як:

- Ігрові режими: встановлювати правила гри та змінювати її поведінку в цілому;
- Гравці: призначати гравцям особливі характеристики та зовнішній вигляд;
- Камери: створювати різні ракурси та змінювати властивості камери в реальному часі;
- Управління: призначати кнопки для керування персонажем, автомобілем або рівнем гри;
- Речі: зброя, предмети, що підбираються та інше;
- Оточення: створювати випадково генероване оточення.

Узагалі, Unreal Engine та Blueprints надають потужний та доступний інструментарій для розробки ігор, що дозволяє розробникам реалізовувати свої ідеї незалежно від їхнього досвіду в програмуванні.

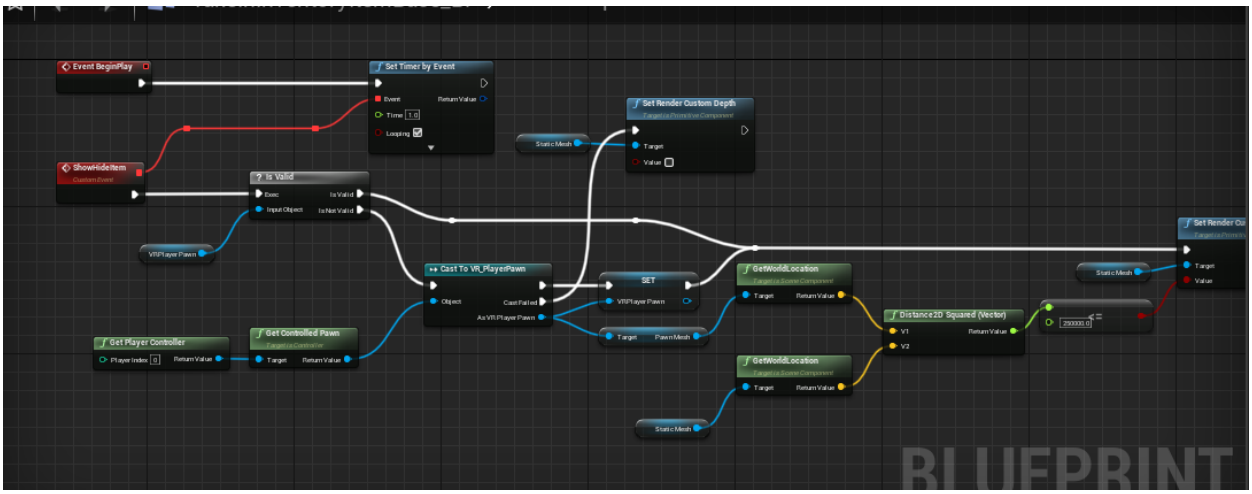


Рисунок 2.3. Приклад використання системи Blueprints

Unreal Engine є одним з основних продуктів для розробки ігор та віртуального виробництва. Завдяки UE5 розробники отримують можливість створювати великі проекти. Представники різних галузей зможуть співпрацювати в реальному часі, щоб надавати кінцевим користувачам захоплюючі візуальні ефекти та можливості.

Unreal Engine 5 стає ще більш деталізованим. Nanite - це віртуалізована геометрична система, яка дозволяє економити час при розробці великої кількості деталей. Це усуває вимогливі завдання щодо завантаження рівнів деталізації (LOD) і дає змогу імпортувати графіку кінематографічної якості.

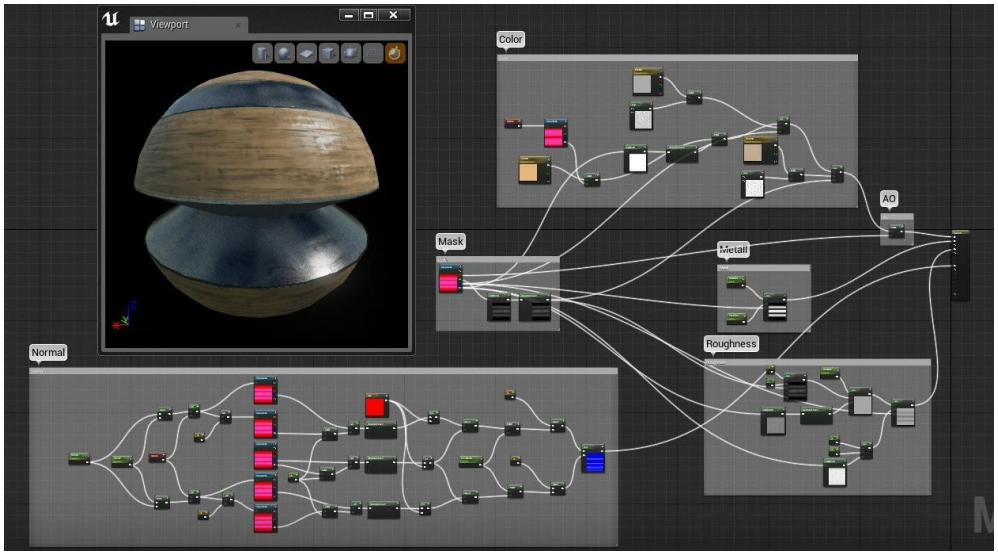


Рисунок 2.4. Розробка властивостей матеріалу

Створення реалістичних сцен часто зводиться до освітлення. Lumen надає можливість швидко вносити зміни, регулюючи освітлення залежно від часу доби, нових джерел світла, таких як ліхтарик, або раптового променя, що потрапляє в кадр і так далі. Він контролює освітлення відкритих динамічних сцен навіть до найдрібніших деталей [13].

UE4 вже відомий своєю здатністю створювати відкриті світи. Але Unreal Engine 5 піднімає його на новий рівень, що прискорює процес створення і спрощує роботу. Система World Partition використовує сітку для відображення підрівнів усього світу. Вона дозволяє керувати складними рівнями, які завантажуються і

розвантажуються по мірі переміщення гравця по ландшафту. Крім того, система One File Per Actor допомагає командам працювати паралельно.

Unreal Engine 5 розширює свій набір інструментів анімації, включаючи інструмент Control Rig. Тепер можна створювати та ділитися налаштуваннями між персонажами. Щоб отримати більш природні рухи, можна зберігати та застосовувати пози за допомогою розширення Full-Body IK.

MetaSounds надає повний контроль і покращену гнучкість у керуванні звуком. Він поліпшує робочий процес, щоб допомогти вам контролювати всі аспекти звуку.

Основні переваги Unreal Engine 5: масштабованість, багатофункціональність, широкі можливості налаштування, 2D та 3D.

2.2.2. Unity3D

Unity - це не просто движок, а середовище розробки комп'ютерних ігор, яке об'єднує різні програмні засоби, такі як текстовий редактор, компілятор, відладчик і т. д., використовувані під час створення програмного забезпечення. Unity робить процес розробки ігор максимально простим і комфортним завдяки його зручному інтерфейсу. Багатоплатформенні можливості движка дозволяють розробникам охопити якомога більше ігрових платформ та операційних систем [15].

Unity3D надає можливість розробляти ігри без спеціальних знань. Використовується компонентно-орієнтований підхід, де розробник створює об'єкти, такі як головний герой, і додає до них різні компоненти, наприклад, візуальне представлення персонажа та способи управління ним. Завдяки зручному інтерфейсу Drag & Drop і функціональному графічному редактору, движок дозволяє в реальному часі малювати карти і розміщувати об'єкти, а потім тестувати результати відразу [16].

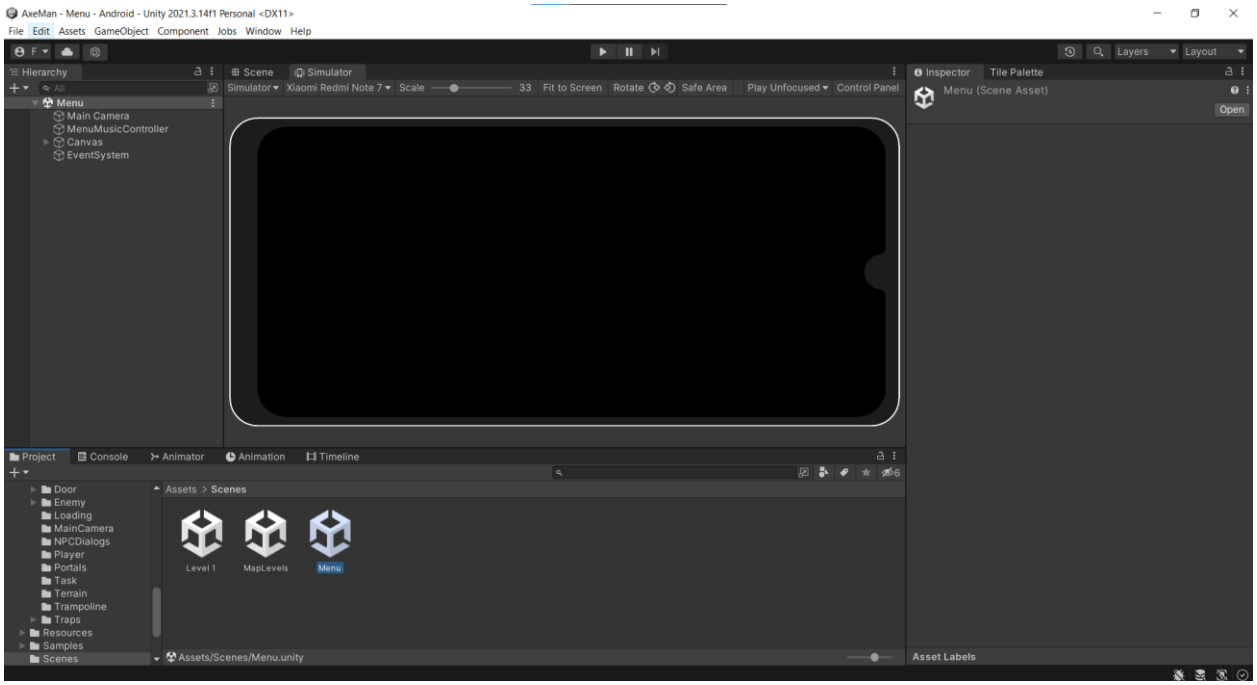


Рисунок 2.5. Інтерфейс середовища розробки Unity3D

Цей движок також відомий як середовище розробки інді та мобільних ігор. Згідно з даними з вересня 2019 року, 52% з 1000 найкращих мобільних ігор були створені на основі Unity, а також 60% всього контенту AR/VR було розроблено з його допомогою. Гравці в ігри, створені у Unity, розташовані в 195 країнах - це практично кожна країна на планеті. Ця платформа розробки в реальному часі дозволяє створювати 2D, 3D, VR і AR-контент та досягла 3 мільярдів пристроїв за останні 12 місяців.

Головна причина популярності Unity, особливо для невеликих проектів, - це безкоштовність. За особистою ліцензією движка розробники можуть створювати комерційні ігри безкоштовно, якщо їх прибуток або фінансування за останні 12 місяців не перевищує 100 000 доларів США.

Unity спрощує процес створення інтерактивного 3D-контенту. Багато великих організацій вибирають цей ігровий движок через його високу функціональність, якісний контент і універсальність для будь-якого типу гри. Він підтримує як 2D-, так і 3D-контент.

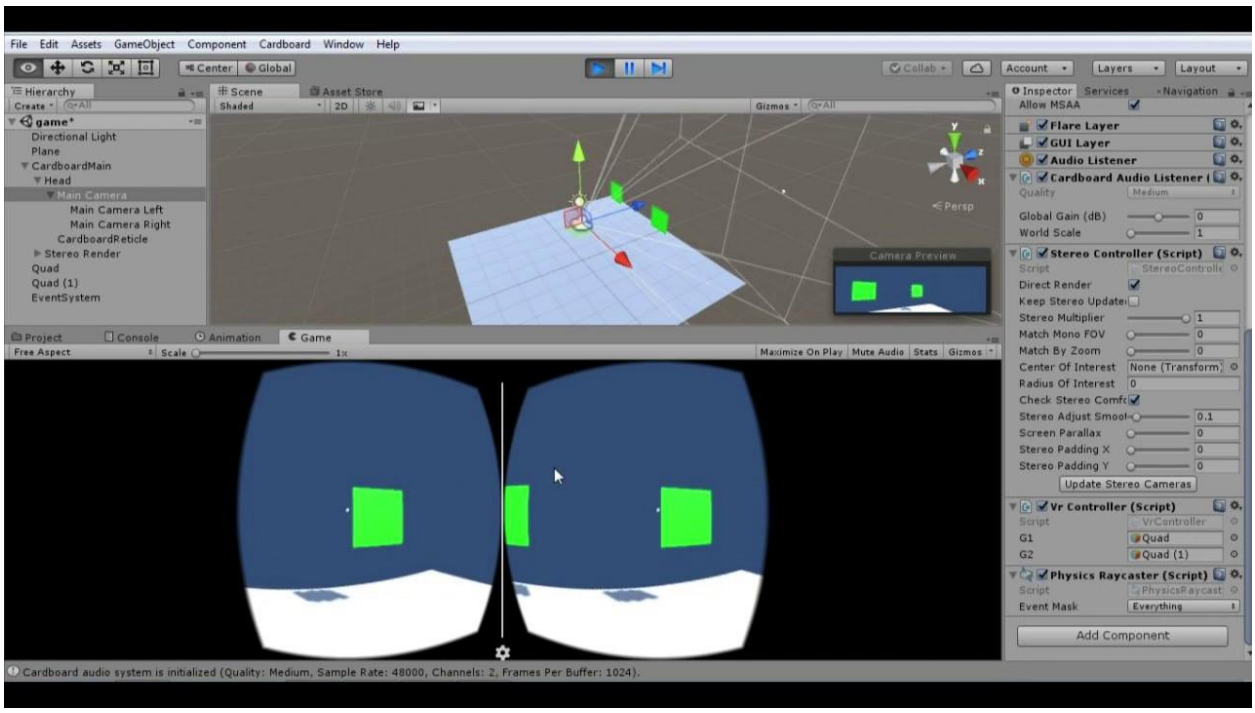


Рисунок 2.6. Створення гри з використанням VR технології

Завдяки універсальному редактору Unity, цей движок сумісний з різними платформами, такими як Windows, Mac, Linux, iOS, Android, Switch, Xbox, PS4, Tizen та інші. Його дружній інтерфейс спрощує розробку і зменшує потребу у навчанні. Unity Asset Store також містить велику колекцію інструментів та контенту, які постійно поповнюються [14].

Сьогодні Unity є одним з найпопулярніших ігрових движків у всьому світі. За офіційним сайтом Unity, понад 50% мобільних ігор розроблено з використанням Unity 3D. Крім того, 34% найкращих AAA-ігор були створені з використанням Unity. Ігри на движку Unity 3D встановлено на близько 28 мільярдів пристроїв, що втричі перевищує населення світу, свідчачи про неймовірний попит серед користувачів.

Основні переваги Unity включають:

- Єдиный редактор Unity, який поєднує редактор сцен, редактор ігрових об'єктів і редактори скриптів. Додатково надаються інструменти для створення дерев та теренів.
- Покращені можливості скриптінгу, з доступом до трьох мов: JavaScript, C# і Boo (варіант Python).

- Кросплатформенність, що охоплює Windows, macOS, Wii, iPhone, iPod, iPad, Android, PS3 і Xbox 360 (не всі доступні в безкоштовній ліцензії).
- Сучасний рівень графіки, який конкурує з іншими двигунами, включаючи можливості освітлення, постпроцесингу, прискореного опрацювання лайтмапів тощо.
- Розроблений фізичний двигун.
- Масштабованість та продуктивність, з відмінною обробкою більшості простих процесів.
- Запуск програм на Unity у веб-плагіні.
- Прийнятна ціна повної ліцензії для великих веб-розробників.

Недоліки Unity включають:

- Закритість коду і неможливість отримання вихідних кодів двигуна, навіть з ліцензією.
- Обмежена можливість розширення фізичного двигуна сторонніми рішеннями, такими як стороння фізика або SpeedTree.

2.3 Аналіз операційних систем для ігор

Windows не тільки є найкращою операційною системою для ігор завдяки своєму широкому вибору ігор, але й тому, що більшість ігор працюють краще на ній, ніж на Linux і macOS.

Різноманітність є однією з найсильніших переваг комп'ютерних ігор. Крім різних видів апаратних компонентів, користувачі також можуть вибирати операційну систему, яка найкраще відповідає їх потребам.

На сьогоднішній день існує три основних варіанти операційних систем: Windows, Linux і macOS. Кожна з них орієнтована на різні аудиторії і має свої переваги та недоліки.

Microsoft Windows є найпоширенішою та найпопулярнішою операційною системою. Започаткована Windows 3.0 в 1990 році, вона стала революційною, спрощуючи інтерфейс користувача та роблячи комп'ютери доступними широкому колу користувачів.

З того часу Windows отримала багато нових версій, таких як Windows 95, Windows 98, Windows NT, Windows 2000 та інші. Проте, справжній успіх у геймерському світі наступив з Windows XP. Після XP наступила Windows Vista, яка була швидко замінена вдосконаленою Windows 7. Потім були випущені Windows 8 і Windows 8.1, які отримали критику через свій орієнтований на планшет інтерфейс користувача.

Сьогоднішня найпросунутіша і найнадійніша версія Windows - це Windows 10. Замість регулярного випуску нових версій, Microsoft вирішила розвивати та вдосконалювати Windows 10 за допомогою безкоштовних оновлень, змінивши свій підхід до своєї флагманської операційної системи.

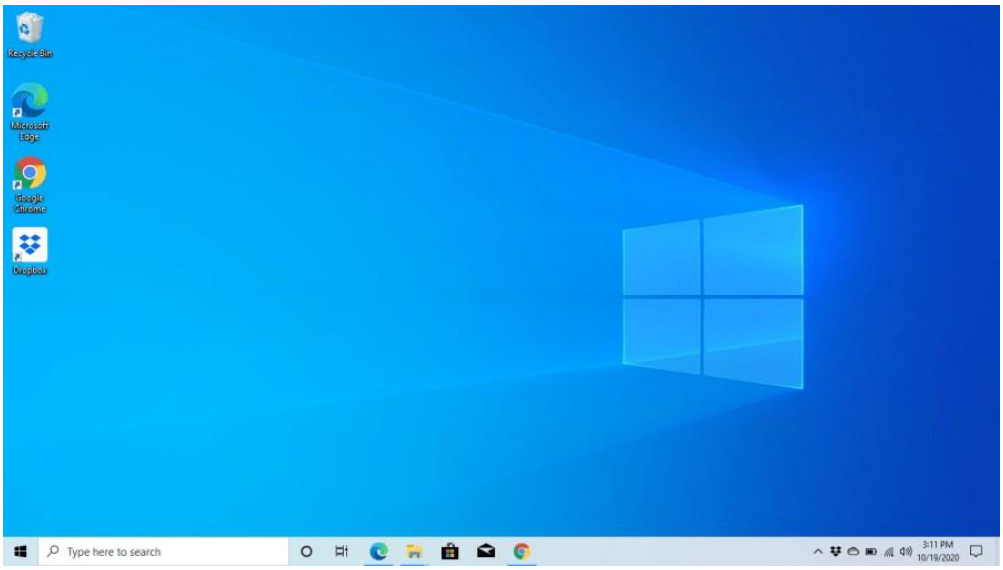


Рисунок 2.12. Інтерфейс Windows 10

Linux відзначається своєю незвичайністю, оскільки він представляє широкий спектр операційних систем, що базуються на відкритому вихідному коді ядра Linux. Ця система була створена Лінусом Торвальдсом у 1991 році з метою надати користувачам безкоштовну та гнучку операційну систему [8].

Однією з найвідоміших операційних систем на основі ядра Linux є AndroidOS від Google, яке використовується на смартфонах та планшетах. На персональних комп'ютерах найпопулярнішими дистрибутивами є Ubuntu, CentOS, Debian, OpenSUSE, Arch Linux, Fedora, SteveOS від Valve та інші, які повністю безкоштовні та розроблені різними командами розробників.

Linux, однак, більш спрямований на професіоналів та ентузіастів завдяки своїм потужним можливостям, гнучкості та низьким вимогам до апаратного забезпечення порівняно з Windows. Проте, його інтерфейс рідко можна назвати "дружнім до користувача", а кількість сумісного з Linux програмного забезпечення обмежена.

Хоча початки MacOS сягають 1984 року, сучасна версія, яку ми знаємо сьогодні, була випущена лише в 2001 році. Вона відстає в популярності від Windows, і доступна виключно на комп'ютерах Apple iMac і MacBook.

Apple зберігає свою технологію під контролем, що надає MacOS ряд переваг та недоліків. Воно оптимізоване для роботи з обладнанням Apple, що робить його ефективним. Крім того, воно створене з метою наближення до досконалості, результатом чого є високопродуктивна та зручна для користувача операційна система, яка популярна серед різних категорій користувачів.

Однак, основний недолік MacOS пов'язаний з його обмеженою сумісністю з апаратним забезпеченням. З точки зору потужності, більшість комп'ютерів Apple не відповідають вимогам програмного забезпечення для забезпечення високої продуктивності. Це призводить до скептицизму користувачів щодо високих цін на продукцію Apple.

Що стосується продуктивності в іграх, Windows і Linux зазвичай знаходяться на однаковому рівні, з невеликою перевагою однієї чи іншої в залежності від конкретної гри. Крім того, хоча кожна версія Windows підтримує однакову частоту оновлення кадрів, різні дистрибутиви Linux можуть мати різну продуктивність [7].

У плані вибору оптимального дистрибутива Linux важко знайти найефективніший варіант, проте на сьогоднішній день SteamOS і Ubuntu є

передовими. За своєю стабільністю, надійністю та продуктивністю Windows значно випереджає інші операційні системи.

MacOS, через апаратні обмеження, є менш вигідною платформою для геймінгу, а не через саму операційну систему. Apple розробляє комп'ютери, які мають обмежену можливість модифікацій та обмежений простір для потужних графічних процесорів. Для досягнення хорошої продуктивності на Mac потрібно вкласти значні зусилля та встановити зовнішню графічну карту.

Безсумнівно, більшість ігор для ПК випускаються для Windows, яка є найпопулярнішою геймінговою операційною системою. Вибір ігор для Linux та MacOS є обмеженим, хоча підтримка Linux постійно зростає. На даний момент близько 4000 ігор підтримуються Linux, тоді як для Windows ця кількість перевищує 20 000. У MacOS доступно приблизно 7000 ігор в Steam.

Windows є безспірним лідером на ринку ігор, надаючи підтримку для більшої кількості ігор порівняно з Linux та MacOS разом узятими. MacOS не є найбільш потужною платформою для високоякісних ігор, оскільки ноутбуки Mac мають обмежену можливість оновлення та налаштування, що ускладнює конкуренцію з настільними ПК з Windows за продуктивністю.

Linux не є оптимальною операційною системою для геймінгу, оскільки вона не пристосована до масових ігор.

Однак, Microsoft Windows пропонує найбільший вибір програм та ігор, зручну інтерфейс, а також ефективне виконання ігор з високими показниками FPS. MacOS має деякі обмеження щодо кількості ігор та потужності, а Linux є безкоштовною та гнучкою операційною системою, проте має проблеми з сумісністю програмного забезпечення та не так проста у використанні для необізнаних користувачів.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ГРИ

3.1. Концепція та сценарій гри

Обраний жанр гри - платформер. Платформер - це жанр комп'ютерних ігор, який також відноситься до аркадних ігор. У цьому жанрі основною особливістю геймплею є стрибки по платформах, лазіння по сходах, збирання предметів, а також завершення рівня та подолання перешкод. Ігри цього жанру, як правило, не є складними в реалізації, проте вони можуть захопити гравця на довгі години і зацікавити широку аудиторію.

У грі платформері "Unnamed" ви виступаєте у ролі таємничого героя, що прокинувся без пам'яті в небезпечному світі. Ваша мета - відновити втрачені спогади, розкривши таємницю свого минулого.

Подорожуючи через захоплюючі та розмаїті регіони, ви зіткнетеся з неймовірними випробуваннями та складними перешкодами. Ваші навички стрибків, акробатики та бойової майстерності будуть вирішальними в боротьбі з ворогами та подоланні складних пасток.

Кожен регіон має свій унікальний стиль та атмосферу, включаючи загадковий ліс з чарівними рослинами, вогненний кратер з виринаючою лавою, льодяний перевал зі сніжними бурями та храм вічності, де чекають найвеличніші випробування.

Крім того, ви збиратимете розкидані по рівнях артефакти, які допоможуть відкрити нові можливості та розкрити фрагменти вашої пам'яті. Пошук загадкових ключів, виконання завдань та зіткнення з потужними босами стануть частинами захоплюючого сюжету, що розкривається під час гри.

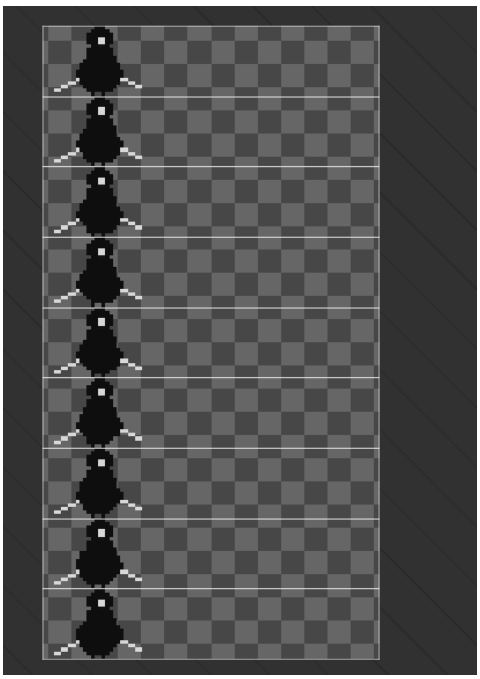


Рисунок 3.1. Головний персонаж

3.2. Підготовка графічних елементів

У грі використовуються зображення у форматі PNG (Portable Network Graphics), який забезпечує стиснення без втрат якості графічної інформації. Розміри зображень відповідають таким варіантам: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 пікселів.

У поточному етапі розробки створюються різноманітні графічні елементи для гри. Це включає фонові зображення, елементи платформ, зовнішній вигляд персонажів та перешкод, а також елементи інтерфейсу та логотип гри. З метою майбутньої анімації об'єктів, всі можливі стани об'єктів відображаються шляхом їх відповідного зображення [15].

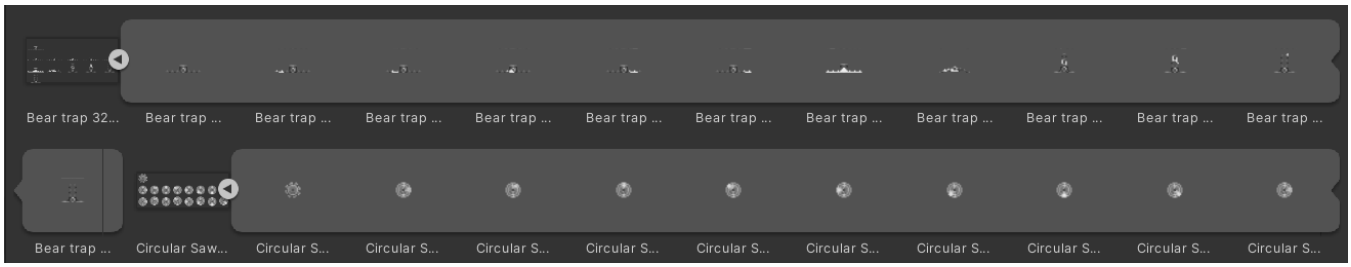


Рисунок 3.2. Стани графічних елементів гри

Початкове меню гри є першим способом взаємодії гравця з ігрою, тому вкрай важливо створити яскравий, зрозумілий та зручний інтерфейс.

Основне меню гри складається з різних елементів, включаючи логотип гри та основні кнопки з такими функціями:

- Кнопка зі стрілкою вправо - почати гру.
- Червона кнопка з дверима - вийти з гри.
- Кнопка зі зображенням шестерні - налаштування гри.

Керування в меню здійснюється за допомогою пальців, торкаючись екрану телефона.

Цих кнопок достатньо, щоб користувач міг спокійно та впевнено користуватися грою, інтуїтивно розуміючи їх функції. Під час гри, користувач бачить такі елементи інтерфейсу, як число набраних алмазів за кожен ігровий рівень, загальна кількість пройдених рівнів, набраний досвід та кількість доступних локацій.

Всі ці елементи інтерфейсу допомагають гравцю отримати повну інформацію та забезпечують комфортну геймплейну забаву, зрозумілу інтуїцію та контроль над грою.

Рисунок 3.3. Інтерфейс головного екрану гри

У випадку смерті персонажа, гравець має кілька варіантів: він може почати гру з початку з останнього чекпоїнту, повернутися на головний екран або вийти з гри.

Ігрова сцена представляє собою всі графічні елементи, які створюють ігровий простір, з яким взаємодіє головний герой. Іншими словами, це все, що гравець бачить на своєму екрані під час гри [16].

Давайте розглянемо структуру ігрової сцени більш детально. На початку гра, головний персонаж знаходиться на землі. Земля є безпечними ділянками, по яких герой може вільно рухатися та стрибати. На землі можуть бути різні статичні та динамічні елементи, які можуть завдавати шкоди персонажу, надавати опит або алмази, або ж бути просто елементами декору, які додають до загального вигляду гри.

Однією з нагород та стимулів для гравця є збір алмазів та набуття опиту. Алмази можуть розташовуватись на платформах, між ними або навіть над перешкодами. Також на платформах можуть бути розташовані коробки, які містять сфери опиту від 1 до 5 одиниць. Тому гравець повинен уважно проходити рівні, щоб не пропустити коробки з цінними предметами. Алмази та коробки є динамічними об'єктами, алмази можуть обертатись навколо своєї осі, а коробки можуть розбиватись, випускаючи опит або сердечка.

Таким чином, ігрова сцена пропонує різноманітні елементи, які гравець взаємодіє з ними, створюючи цікавий геймплей та надаючи можливості для збору нагород та досягнення успіхів.

У грі присутні різні динамічні графічні елементи, такі як рослини, труби та інші, які мають декоративну функцію, а також вороги, що взаємодіють з персонажем. При руху головного персонажа по землі, фон гри також рухається динамічно, що підсилює відчуття руху та швидкості.

Вороги можуть мати різні типи, такі як рухомі, ховаються або звичайні пастки. Пастки розміщуються на землі, і головний герой повинен перестрибнути їх, щоб продовжити гру. Якщо персонаж падає на пастку, він загибається, і гру можна почати з початку. Пастки утворюють перешкоду на землі, яку герой повинен перестрибнути.

Вороги можуть також розміщуватися на платформах, що ускладнює процес застрибування на них. Вони можуть рухатися та перекривати деякі шляхи, що призначені для руху ігрового персонажа, на короткий проміжок часу.

Над платформами можуть розташовуватися пастки, які спускаються різко, якщо персонаж наближається до них, і можуть вбити головного героя. Тому необхідно обережно та не поспішаючи проходити їх. Зіткнення з будь-якими перешкодами призводить до програшу, і гру можна буде почати з початку або з останнього чекпоїнта.

Рисунок 3.4. Перешкоди

3.3. Внутрішня ігрова логіка

Для даної гри було обрано ПК як ігрову платформу, оскільки вона є найзручнішою для одночасного створення та тестування гри. Крім того, комп'ютерні ігри все ще є найпопулярнішими серед всіх доступних платформ.

Майже кожну гру можна представити у вигляді основної функції, яка містить усю ігрову логіку і запускається, коли користувач виконує певні дії або після певного часу. Цей цикл запуску основної функції відбувається знову і знову і називається ігровим циклом [11].

У грі "Unnamed" основною функцією є проходження рівнів. Під час проходження рівнів активуються всі ключові події, вороги, NPC і т.д.

Земля та тунелі для переміщення персонажа є статичною картою, яка не повторюється кожного разу, коли гра починається з нового рівня. Гра має початкову точку, яка є першим рівнем, але всі наступні рівні створюються вручну під час проходження гри. У проекті Unity були створені фрагменти карти та різні внутрішні області, які обираються під час гри [13].

Рисунок 3.5. Фрагменти карти

Створення рівня за допомогою тайлсетів є поширеним підходом у розробці ігор в Unity2D. Тайлсети - це набори графічних елементів, відомих як тайли, які використовуються для побудови рівнів ігрового світу.

Ось кілька кроків, які можна виконати для створення рівня за допомогою тайлсетів в Unity2D:

1. Підготовка графіки: Спочатку потрібно підготувати або отримати графіку тайлсету. Тайли - це невеликі графічні зображення, які утворюють ваші рівні. Вони можуть представляти різні об'єкти, такі як плитки для підлоги, стіни, рослини тощо.
2. Створення нового рівня: У Unity2D можна створити новий рівень, створивши порожній об'єкт, який буде служити контейнером для рівня.
3. Додавання тайлсету: Далі потрібно додати компонент "Tilemap" до об'єкта рівня. Цей компонент дозволяє використовувати тайли в рівні. Вибераємо об'єкт рівня, додаємо компонент "Tilemap" за допомогою контекстного меню

"Add Component" або кнопки "Add Component" у верхньому правому куті інспектора.

4. Налаштування тайлсету: Після додавання компонента "Tilemap" можна прикріпити тайлсет до нього. У інспекторі компонента "Tilemap" виберємо ваш тайлсет або створюємо новий.
5. Побудова рівня: За допомогою інструментів тайлсету починаємо будувати свій рівень. Вибераємо потрібний тайл з тайлсету та нанесемо його на рівень, розташовуючи тайли у відповідних позиціях. Можемо створювати структуру рівня, додавати об'єкти, задавати їхні властивості та так далі.
6. Налаштування колізій: Якщо хочемо, щоб об'єкти взаємодіяли з рівнем, наприклад, персонаж не міг проходити через стіни, можемо налаштувати колізії для тайлів. Використовуємо компонент "Tilemap Collider 2D", щоб створити колізії для тайлів, забезпечуючи фізичну взаємодію.
7. Додаткові налаштування: Unity2D має багато додаткових інструментів та можливостей для роботи з тайлсетами. Задаємо розміри рівня, встановити масштаб тайлів, налаштувати режими затінення та багато іншого.

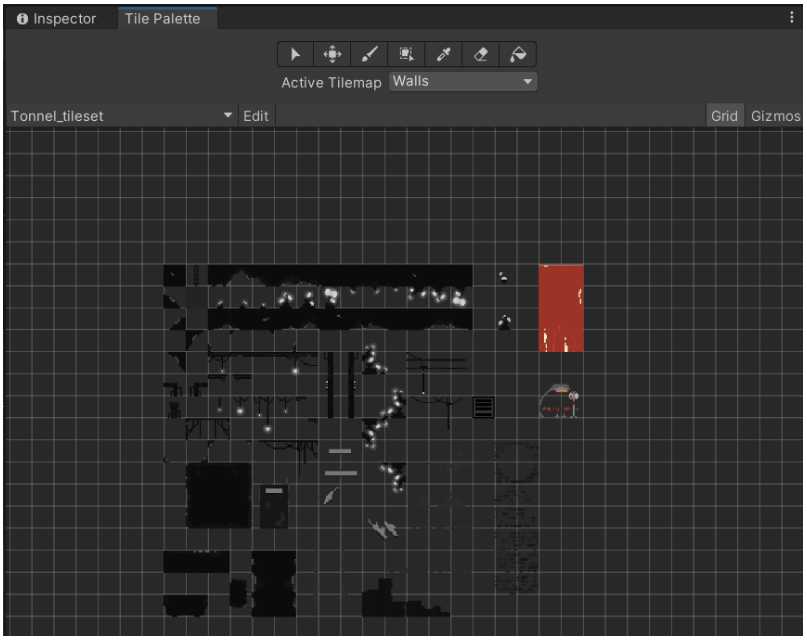


Рисунок 3.6. Палета для тайлів рівня

Для мого диплому я створював персонажа для платформера в Unity. Перший крок, який я зробив, це підготував графіку персонажа. Я збирався створити анімований спрайт набір, що складається з декількох зображень, які змінюються в залежності від рухів персонажа. Я створив графічні зображення для персонажа або використав наявні.

Потім я створив об'єкт персонажа в Unity, створивши порожній об'єкт, який буде представляти мого персонажа на сцені. Я додав компонент 'Sprite Renderer' до цього об'єкта, щоб відображати графіку персонажа.

Наступним кроком було налаштування анімацій персонажа. Я використовував анімаційні контролери Unity для налаштування рухів та анімацій мого персонажа. Створивши новий анімаційний контролер, я додавав різні стани, такі як 'Idle' (без руху), 'Run' (біг), 'Jump' (стрибок) та інші. Ці стани відповідали різним рухам та діям персонажа. Потім я прикріпив анімаційний контролер до компонента 'Animator' на об'єкті персонажа.

Після цього я налаштував фізику мого персонажа. Додав компонент 'Rigidbody2D' до об'єкта персонажа, щоб надати йому фізичні властивості. Налаштував параметри, такі як маса, демпфування та гравітація, щоб керувати рухами персонажа.

Останнім кроком було написання скрипта для керування персонажем. Цей скрипт відповідав за зчитування вхідних даних від гравця та зміну стану персонажа відповідно. Наприклад, у мене були функції для руху вліво/вправо, стрибка та інші дії. Я використовував клавіші або контролер для керування персонажем**.**

За всі дії персонажа відповідає скрипт:

```
Сообщение Unity | Ссылок: 0
private void Update()
{

    jumpCooldownTimer += Time.deltaTime;


    if (joystick.Horizontal >= .5f || joystick.Horizontal <= -.5f)
    {
        if (scriptStamina.enabledStamina == true)
        {

            scriptStamina.currentStamina -= 0.004f - (bonusStamina / 1000);

            dirX = joystick.Horizontal;
            rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
        }
        else if (scriptStamina.enabledStamina == false)
        {
            dirX = joystick.Horizontal;
            rb.velocity = new Vector2(dirX * moveSpeed / 2, rb.velocity.y);
        }
    }
    else if (!knockPlayer)
    {
        dirX = 0f;
        rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
    }

    if (joystick.Vertical >= .5f && IsGrounded() && jumpCooldownTimer >= jumpCooldown)
    {
        jumpCooldownTimer = 0;
        jumpSoundEffect.Play();
        rb.velocity = new Vector2(rb.velocity.x, jumpforce);
    }

    UpdateAnimationState();
}
```

Система Unity надає стандартизовані методи для реалізації управління і забезпечує розширену функціональність, яка виходить за межі базових рішень. Вона дозволяє прив'язати дії до логіки коду і легко налаштовувати різні пристрої та варіанти керування за допомогою візуального інтерфейсу вікна Input Action. Система введення також надає API, а пакет Input System доступний у менеджері пакетів.

Введення відіграє важливу роль у взаємодії гравця з грою, і якість ігрового досвіду, що отримують гравці, залежить від цього.

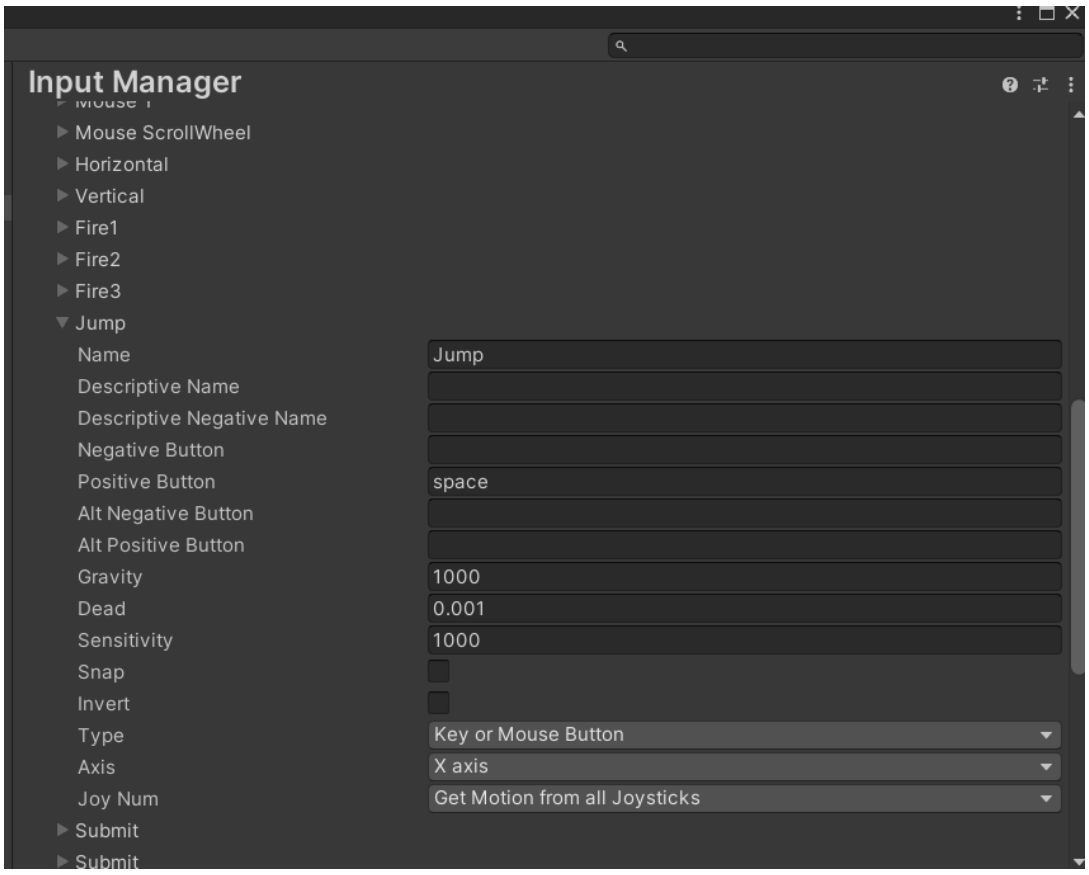


Рисунок 3.7. Панель налаштування введення для стрибка

Кожній вісі прив'язані дві кнопки на джойстику, миші або клавіатурі. Поле "Name" використовується для перевірки осі у скрипті. "Positive button" це кнопка, що зміщує значення осі у позитивному напрямку, а "Alt Positive Button" - альтернативна кнопка, що зміщує ось у протилежному напрямку. "Gravity" вказує на швидкість, з якою вісь повертається в нейтральне положення, коли кнопки не натиснуті. "Dead" вказує на розмір аналогової мертвої зони, де значення вважається нейтральними. "Sensitivity" визначає швидкість, з якою вісь рухатиметься до цільового значення. Ці параметри застосовуються тільки для цифрових пристроїв. "Type" визначає тип введення, який керуватиме віссю [12].

Після того, як персонаж готовий і введення налаштоване, необхідно налаштувати програвання анімації. "Animation Controller" є інструментом Unity, в якому описуються різні стани персонажа, наприклад, біг, стрибок або смерть. Кожному стану відповідає свій анімаційний кліп, і граф переходів між станами налаштовується, як показано на рисунку 3.8.

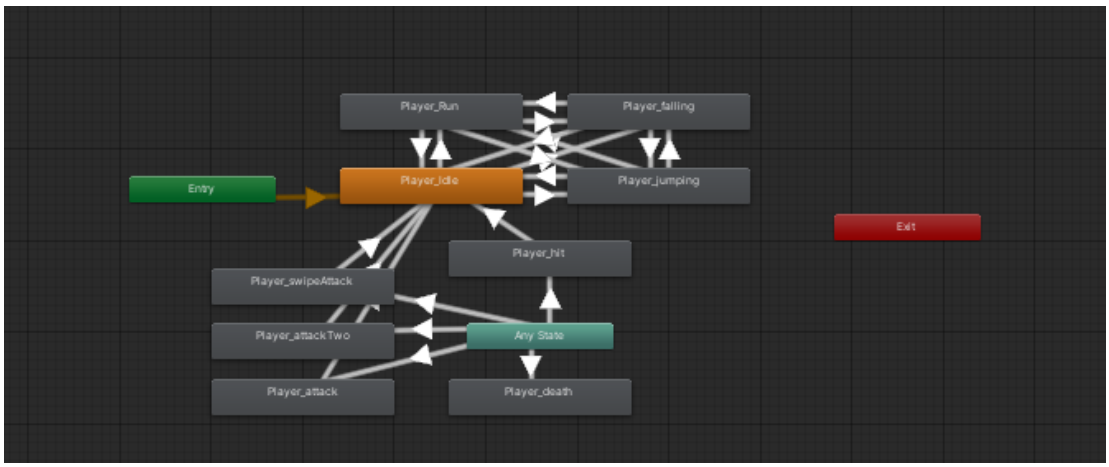


Рисунок 3.8. Блоки анімацій та графів переходів

Створення візуальних елементів гри - це лише одна частина роботи. У процесі розробки ігор часто потрібно додавати різноманітні звукові ефекти до різних подій. Додавання звуків до гри надає глибину та створює атмосферу гри. Навіть декілька базових звуків можуть значно розширити ігровий світ і збагатити його.

Unity пропонує простий у використанні візуальний редактор і підтримку широкого спектру аудіоформатів, що значно спрощує процес додавання звуків. У цьому уроці ви дізнаєтеся, як додавати звуки до гри "Barn Blaster" як за допомогою редактора Unity, так і за допомогою коду. Для відтворення звуків у Unity використовуються функції "AudioSource" та "AudioClip" [9].

"AudioSource" відповідає за відтворення звуку в 2D або 3D просторі. У тривимірному просторі гучність звуку може змінюватися в залежності від відстані між "AudioSource" та об'єктом, який повинен чути звук. У 2D-просторі "AudioSource" може відтворюватися з постійною гучністю незалежно від відстані до "AudioListener".

"AudioClip" представляє аудіофайл, який буде відтворюватися "AudioSource".

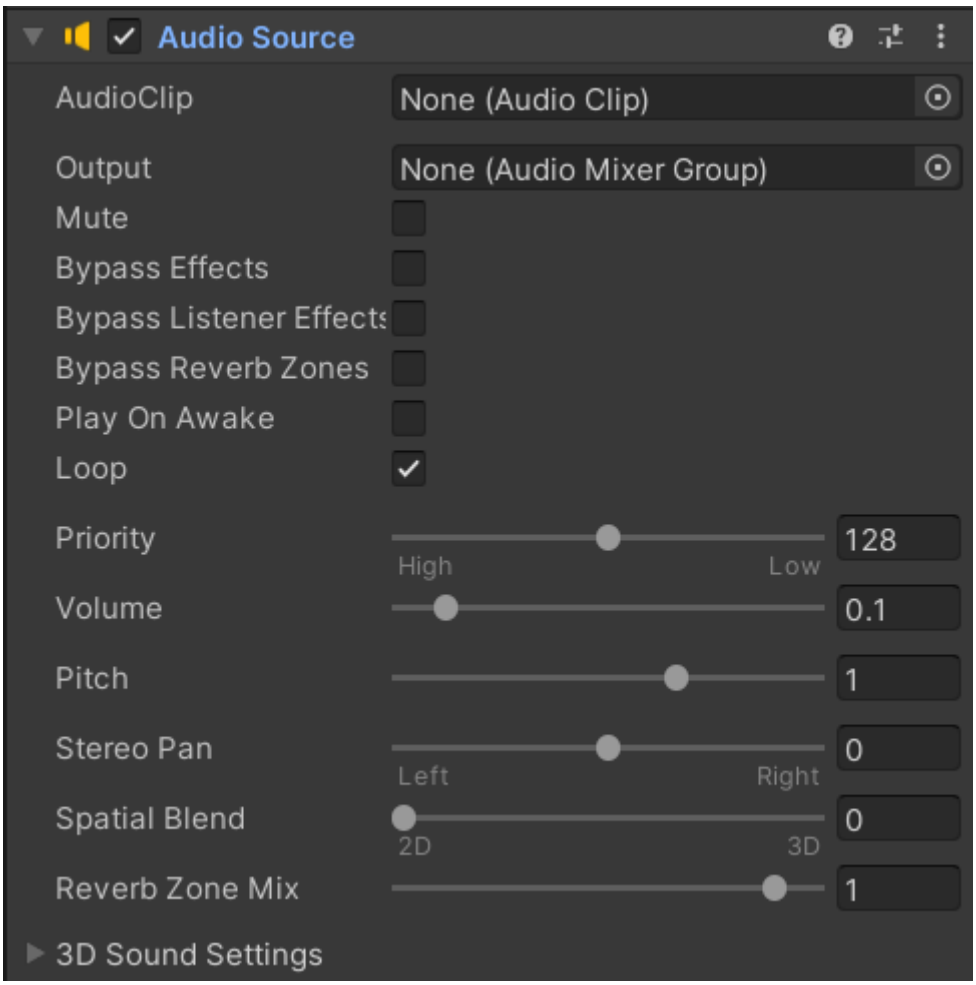


Рисунок 3.9. Налаштування джерела звуку

У цій грі кожна дія головного персонажу супроводжується звуком, таким як звук бігу, стрибка, падіння та приземлення. Крім того, звук присутній у нагородах, таких як монети та сундуки, а також у ворогів. Звукові ефекти супроводжують анімації графічних елементів гри. На наведеному нижче рисунку показані звуки, пов'язані з класом ворогів.

Рисунок 3.10. Звукові файли класу ворога

3.4. Аналіз отриманого результату

В результаті роботи була створена гра з різними рівнями, де є NPC, вороги, пастки та багато іншого. Під час гри головний персонаж повинен подолати різні перешкоди на своєму шляху і збирати монети.

Після завершення налаштування ігрових сцен, анімацій, звуків та внутрішньої логіки, створюється збірка гри, яку можна передавати та розповсюджувати. Збірка отримує назву "Build" і знаходиться на панелі "Build settings". В налаштуваннях гри обрані основна операційні система Android. Гра підтримує як 32-бітну, так і 64-бітну архітектуру для цієї операційної системи.

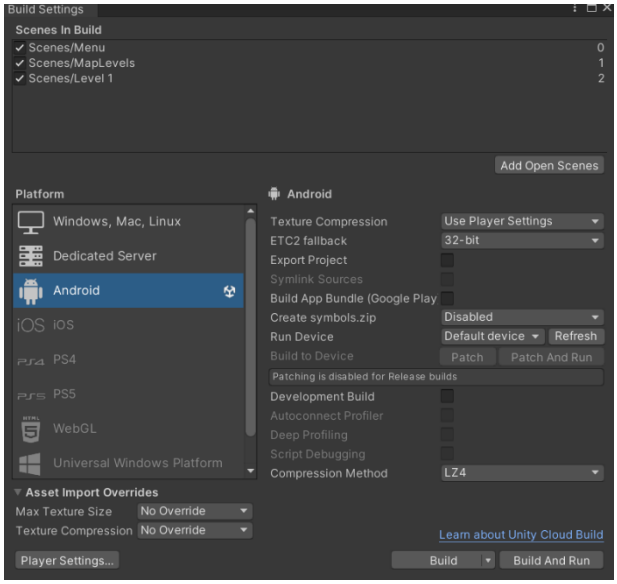


Рисунок 3.11. Створення збірки гри

У фінальному ігровому продукті було реалізовано велику кількість механік та інструментів Unity, що створюють унікальний ігровий досвід під час взаємодії користувача з грою.

Для подальшого розвитку та покращення гри визначено такі етапи:

- Допрацювання можливостей головного персонажа, щоб збільшити його функціональність та гнучкість.

- Створення нових ігрових рівнів, які додадуть більше викликів та розваги для гравців.
- Розробка кросплатформених версій гри, щоб забезпечити доступ до гри на різних пристроях та платформах.
- Розповсюдження гри на популярних платформах, таких як App Store, Play Store, для того, щоб гравці з усього світу могли насолоджуватися грою.
- Активна підтримка гри та вдосконалення на основі отриманих відгуків користувачів, щоб забезпечити якість гри та задоволення користувачів.

Ці етапи допоможуть продовжити розвиток гри, залучити нових гравців і забезпечити позитивний ігровий досвід для всіх користувачів.

ВИСНОВОК

Отже, після ретельного аналізу світового ринку ігрової індустрії можна зробити висновок, що ігри для різних платформ будуть набувати ще більшої актуальності. З покращенням інтернету та технологій все більше людей залучатимуться до різноманітних ігор, оскільки вони не просто дозволяють провести час, але й надають людям радість та широкий спектр емоцій. Цей сегмент ринку економіки постійно зростає та приносить збільшений прибуток.

У даній дипломній роботі було проведено глибокий аналіз індустрії та всіх її аспектів. На підставі отриманих даних можна стверджувати, що ігри будуть розвиватись нарівні з усіма інформаційними технологіями. Покращення ігрових рушіїв, що відбувається в даний момент, дозволить розробникам створювати нові захоплюючі ігрові світи з високоякісною графікою та фотореалістичними ефектами.

Під час написання дипломної роботи був розглянутий детальний процес створення гри та проаналізовані всі його етапи. Особлива увага була приділена розробці графіки та анімації графічних елементів. Unity 3D виявилося надійним та доступним середовищем для втілення проекту та всіх задуманих функцій. Усі процеси були зручно реалізовані завдяки внутрішнім можливостям цього рушія.

Підсумовуючи, можна зазначити, що середовище розробки Unity є високоякісним рушієм для створення професійних ігрових продуктів, які можуть конкурувати на ринку та користуватися великим попитом серед гравців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дацко М. А. Моделювання складних об'єктів. / М. А. Дацко. – М. : Максімас, 2015. – 111 с.
2. Крейтон, Р.Х. Основи розробки ігор у Unity / Р.Х. Крейтон. – Packt Publishing, 2010, – 83 с.
3. Любанова, Т.П. Бизнес-план: опыт, проблемы. Содержание бизнес-плана, пример разработки / Т.П. Любанова, Л.В. Мясоедова, Т.А. Грамотенко, и др..

– М.: Приор, 2012. – 204 с.

4. Хокінг Д. М. Unity в дії. Мультиплатформенна розробка на практиці. / Д. М. Хокінг, 2016. – 336 с.
5. Хорхе Паласиос Unity 5.x. Программирование искусственного интеллекта в играх. Руководство / Паласиос Хорхе. – М.: ДМК Пресс, 2017. – 427 с.
6. Apperley T. H. Genre and game studies: Toward a critical approach to video game genres / T. H. Apperley // Simulation & Gaming. – 2006. – Vol. 37. – No. 1. – P. 6 – 23.
7. Buckland Mat. Programming Game AI by Example – Texas, Wordware

Publishing. – 2004. – s. 25 – 43.

8. Clearwater D. What Defines Videogame Genre? Thinking about Genre Study after the Great Divide / D. Clearwater // The Journal of the Canadian Game Studies Association. – 2011. – No. 5. – s. 29 – 49.
9. Goldstone W. Unity Game Development Essentials. / W Goldstone. Birmingham: Packt Publishing Ltd. – 2009. – 316 s.
10. Gregory Jason. Game Engine Architecture. – New York, CRC Press. – 2009. – No. 5. – s. 15 – 24.
11. Gregory Jason. Game Engine Architecture: Second Edition. – New York, CRC Press. – 2014. – No. 32. – s. 23 – 30.
12. Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics:

Third Edition. – Boston, Course Technology. – 2012. – 215s.

13. McShaffry, Mike, Graham David. Game coding complete: Fourth Edition. – Boston, Cource Technology. – 2013. – 184s.
14. Microsoft documentation Справочник по языку C# [Електронний ресурс] / режим доступу: https://docs.microsoft.com/ru-ru/dotnet/csharp/languagereference/language-specification/introduction.
15. Unity Asset Store [Електронний ресурс] / режим доступу: https://assetstore.unity.com/
16. Unity Manual [Електронний ресурс] / режим доступу: https://docs.unity3d.com/Manual/index.html.

ДОДАТКИ

ПРОГРАМНИЙ КОД

```
public class PlayerMovement : MonoBehaviour
{
    public Rigidbody2D rb { get; private set; }
    public bool knockPlayer { get; set; }
    private BoxCollider2D coll;
    private Animator anim;
    private Joystick joystick;
[Header("-------------------------------------------------------------")]
[Header("Шар коробки, чекпоінта, бомби, ворога, банана")]
[SerializeField] private LayerMask boxLayer;
private BreackableBox box;
[SerializeField] private LayerMask checkpointLayer;
private CheckpointManager checkpoint;
[SerializeField] private LayerMask bombLayer;
private BombTrap bomb;
[SerializeField] private LayerMask enemyLayer;
private EnemyLife enemyHealth;
[SerializeField] private LayerMask bananaLayer;
private Banana bananaScript;

[Header("-------------------------------------------------------------")]
[Header("Удар по ворогу та зона поразки гравця")]
[SerializeField] private float colliderDistance;
[SerializeField] private int damage;
[SerializeField] private BoxCollider2D boxCollider;
```

```
//другий удар після першого
private int attackTwo = 0;

[Header("Кількість урону")]
[SerializeField] private float loss;

[Header("-------------------------------------------------------------")]
[Header("Шар землі для стрибка")]
[SerializeField] private LayerMask jumpableGround;

[Header("-------------------------------------------------------------")]
[Header("Пересування персонажа")]
private float dirX = 0f;
[SerializeField] public float moveSpeed = 7f;
[SerializeField] private float jumpforce = 14f;

[Header("Перезарядка стрибка")]
[SerializeField] private float jumpCooldown;
private float jumpCooldownTimer = Mathf.Infinity;

//масив анімацій для подальшого вибору
private enum MovementState { idle, run, jumping, falling }

[Header("-----------------------------------------------")]
[Header("Звукові ефекти")]
[SerializeField] private AudioSource jumpSoundEffect;
[SerializeField] private AudioSource runSoundEffect;
[SerializeField] private AudioSource hitSoundEffect;
```

```
[SerializeField] private AudioSource damagePlayerSoundEffect;
[Header("-------------------------------------------------")]
[Header("Перезарядка атаки гравця")]
[SerializeField] private float delay = 10f;
public bool attackBlocked;

//доступ до healthbar гравця
public HealthBar recalcHeart { get; private set; }
// доступ до панелі здоров'я гравця
public HealthBar recalcHeart { get; private set; }

// доступ до скрипту для управління витривалістю
private StaminaManager scriptStamina;
private float bonusStamina;
private int bonusForUpgradeDamage; // бонус до пошкодження ворога

private void Awake()
{
    // присвоєння змінній компонента з інспектора
    rb = GetComponent<Rigidbody2D>();
    coll = GetComponent<BoxCollider2D>();
    anim = GetComponent<Animator>();
    recalcHeart = FindObjectOfType<HealthBar>();
    joystick = FindObjectOfType<Joystick>();
    scriptStamina = FindObjectOfType<StaminaManager>();

    bonusForUpgradeDamage = PlayerPrefs.GetInt("Attack1") +
PlayerPrefs.GetInt("Attack2") + PlayerPrefs.GetInt("Attack3");
```

```
        loss += bonusForUpgradeDamage;

        bonusStamina = (float)(PlayerPrefs.GetInt("Stamina1") +
PlayerPrefs.GetInt("Stamina2") + PlayerPrefs.GetInt("Stamina3"));
    }

    private void Update()
    {
        // таймер перезарядки стрибка
        jumpCooldownTimer += Time.deltaTime;

        // зчитування руху вліво/вправо
        if (joystick.Horizontal >= .5f || joystick.Horizontal <= -.5f)
        {
            if (scriptStamina.enabledStamina == true)
            {
                // витривалість зменшується при русі
                scriptStamina.currentStamina -= 0.004f - (bonusStamina / 1000);

                dirX = joystick.Horizontal;
                rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
            }
            else if (scriptStamina.enabledStamina == false)
            {
                dirX = joystick.Horizontal;
                rb.velocity = new Vector2(dirX * moveSpeed / 2, rb.velocity.y);
            }
        }
```

```
        else if (!knockPlayer)
        {
            dirX = 0f;
            rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
        }

        // активація стрибка
        if (joystick.Vertical >= .5f && IsGrounded() && jumpCooldownTimer >= jumpCooldown)
        {
            jumpCooldownTimer = 0;
            jumpSoundEffect.Play();
            rb.velocity = new Vector2(rb.velocity.x, jumpforce);
        }

        // оновлення анімації
        UpdateAnimationState();
    }

    // функція для оновлення анімації
    private void UpdateAnimationState()
    {
        // змінна для вибору анімації
        MovementState state;

        // при русі вліво/вправо присвоюється відповідний номер анімації
        if (dirX > 0f)
        {
```

```
            state = MovementState.run;
            // зміна орієнтації тіла персонажа
            gameObject.transform.localScale = new Vector3(weightObjectPlayer, heightObjectPlayer, 1f);
        }
        else if (dirX < 0f)
        {
            state = MovementState.run;
            // зміна орієнтації тіла персонажа
            gameObject.transform.localScale = new Vector3(-weightObjectPlayer, heightObjectPlayer, 1f);
        }
        else
        {
            state = MovementState.idle;
        }

        if (rb.velocity.y > .1f)
        {
            state = MovementState.jumping;
        }
        else if (rb.velocity.y < -.1f)
        {
            state = MovementState.falling;
        }

        // підтвердження анімації і виведення її на екран
        anim.SetInteger("state", (int)state);
```

```
    }

    // прив'язка персонажа до платформ, по яким він стрибає
    private bool IsGrounded()
    {
        return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f, Vector2.down,
.1f, jumpableGround);
    }

    // смерть гравця
    public void DeadPlayer()
    {
        recalcHeart.recalc = true;
        damagePlayerSoundEffect.Play();
        rb.mass = 100f;
        attackBlocked = true;
        GetComponent<PlayerMovement>().enabled = false;
    }

    // функція для входу в двері
    public void EnterToDoor()
    {
        GetComponent<PlayerMovement>().enabled = false;
        rb.bodyType = RigidbodyType2D.Static;
    }

    // фіксація ворога в полі зору
    public void PlayerInSight(LayerMask target)
```

```
    {
        foreach (RaycastHit2D hit in Physics2D.BoxCastAll(boxCollider.bounds.center +
transform.right * damage * transform.localScale.x * colliderDistance,
            new Vector3(boxCollider.bounds.size.x * damage, 2f,
boxCollider.bounds.size.z),
            0, Vector2.left, 0, target))
        {
            // якщо поле атаки персонажа не порожнє
            if (hit.collider != null)
            {
                if (target == bombLayer) // якщо бомба в полі зору, активується вибух
                {
                    bomb = hit.transform.GetComponent<BombTrap>();
                    bomb.StartCoroutine(bomb.ActiveBombtrap()); ;
                    bomb.HitBombPlayer();
                }
                if (target == boxLayer)
                {
                    box = hit.transform.GetComponent<BreackableBox>();
                    box.Break();
                }
                if (target == checkpointLayer)
                {
                    checkpoint = hit.transform.GetComponent<CheckpointManager>();
                    checkpoint.Break();
                }
                if (target == enemyLayer)
                {
```

```
                enemyHealth = hit.transform.GetComponent<EnemyLife>();
                enemyHealth.TakeDamage(loss);
            }
            if (target == bananaLayer)
            {
                bananaScript = hit.transform.GetComponent<Banana>();
                bananaScript.Break();
            }
        }
    }

    // поле відображення для перевірки
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(boxCollider.bounds.center + transform.right * damage *
transform.localScale.x * colliderDistance,
            new Vector3(boxCollider.bounds.size.x * damage, 2f,
boxCollider.bounds.size.z));
    }

    // атака персонажа
    public void Attack()
    {
        if (attackBlocked)
            return;
```

```
        if (attackTwo == 0 && scriptStamina.currentStamina >= (0.4f - (bonusStamina / 10)) && scriptStamina.enabledStamina == true)
        {
            // радіус атаки
            damage = 2;
            anim.SetTrigger("attack");

            PlayerInSight(enemyLayer);
            PlayerInSight(boxLayer);
            PlayerInSight(checkpointLayer);
            PlayerInSight(bombLayer);
            PlayerInSight(bananaLayer);

            scriptStamina.currentStamina -= 0.4f - (bonusStamina / 10);

            // запуск куратини і присвоєння attackTwo нового значення для комбо
            StartCoroutine(attackTwoActivation());
        }
        else if (attackTwo == 1 && scriptStamina.currentStamina >= (0.5f - (bonusStamina / 10)) && scriptStamina.enabledStamina == true)
        {
            damage = 2;
            anim.SetTrigger("attack2");

            PlayerInSight(enemyLayer);
            PlayerInSight(boxLayer);
            PlayerInSight(checkpointLayer);
            PlayerInSight(bombLayer);
```

```
            PlayerInSight(bananaLayer);

            scriptStamina.currentStamina -= 0.5f - (bonusStamina / 10);

            StartCoroutine(attackSliceActivation());
        }
        else if (attackTwo == 2 && scriptStamina.currentStamina >= (0.6f - (bonusStamina / 10)) && scriptStamina.enabledStamina == true)
        {
            damage = 3;

            scriptStamina.currentStamina -= 0.6f - (bonusStamina / 10);

            StartCoroutine(Dash());

            anim.SetTrigger("swipe");

            PlayerInSight(enemyLayer);
            PlayerInSight(boxLayer);
            PlayerInSight(checkpointLayer);
            PlayerInSight(bombLayer);
            PlayerInSight(bananaLayer);

            attackTwo = 0;
        }
    }

    // час для другої атаки в комбо
```

```
    private IEnumerator attackTwoActivation()
    {
        attackTwo = 1;
        yield return new WaitForSeconds(0.6f);
        attackTwo = 0;
    }

    // час для третьої атаки в комбо
    private IEnumerator attackSliceActivation()
    {
        attackTwo = 2;
        yield return new WaitForSeconds(0.3f);
        attackTwo = 0;
    }

    // стрибок гравця під час третьої атаки в комбо
    private IEnumerator Dash()
    {
        // можливість відштовхування гравця вимкнена
        knockPlayer = true;

        // обчислення позиції, куди перемістити гравця під час атаки
        Vector2 direction = new Vector2(transform.localScale.x * 10f,
transform.localScale.y);
        // додавання ривка для гравця
        rb.velocity = direction;

        yield return new WaitForSeconds(0.2f);
```

```
    knockPlayer = false;
}

// активація звуку удару
private void HitPlayerSound()
{
    hitSoundEffect.Play();
}

// перезарядка атаки гравця
private IEnumerator DelayAttack()
{
    yield return new WaitForSeconds(0.8f);
    attackBlocked = false;
}

// таймер атаки гравця
public void TimerAttack()
{
    attackBlocked = true;
    StartCoroutine(DelayAttack());
}

// здоров'я гравця
public void TakeDamage(int damage)
{
    currentHealth -= damage;
```

```
    if (currentHealth <= 0)
    {
        DeadPlayer();
    }
    else
    {
        recalcHeart.recalc = true;
        hitPlayerSoundEffect.Play();
    }
}

// відновлення здоров'я гравця
public void HealPlayer(int heal)
{
    currentHealth += heal;

    if (currentHealth > maxHealth)
    {
        currentHealth = maxHealth;
    }

    recalcHeart.recalc = true;
    healPlayerSoundEffect.Play();
}
//Змінна, що підтверджує смерть
bool death = false;

//Поточне здоров'я
```

```
    public float currentHealth { get; private set; }

    //Скрипт, що відповідає за очки досвіду за вбивство ворогів
    private ScoreExperiense scriptScoreExperiense;

    private void Awake()
    {
        //Поточне здоров'я дорівнює початковому
        currentHealth = startingHealth;
        anim = GetComponent<Animator>();
        scriptScoreExperiense = FindObjectOfType<ScoreExperiense>();
    }

    //Функція, що виконує отримання ушкоджень
    public void TakeDamage(float _damage)
    {
        //Розрахунок поточного здоров'я виходячи з ушкоджень і повного здоров'я
        currentHealth = Mathf.Clamp(currentHealth - _damage, 0, startingHealth);

        //Якщо поточне здоров'я більше 0, то звичайне попадання
        if (currentHealth > 0)
        {
            anim.SetTrigger("hit");

            //Якщо поточна кількість очків гравця більше 0, то при попаданні до нього кількість досвіду зменшується
            if (scriptScoreExperiense.currentScoreExperiense >= 3)
            {
```

```
                scriptScoreExperiense.newMinusScoreExperiense += scriptScoreExperiense.currentScoreExperiense / 3;
                if (scriptScoreExperiense.currentScoreExperiense > 0 && scriptScoreExperiense.currentScoreExperiense <= 3)
                {
                    scriptScoreExperiense.newMinusScoreExperiense += scriptScoreExperiense.currentScoreExperiense - 1;
                }
            }
        }
        else if (currentHealth <= 0 && death == false)
        {
            anim.SetTrigger("death");
            death = true;
        }
    }

    //Функція, що додає здоров'я при збиранні його
    public void AddHealth(float _value)
    {
        //Розрахунок поточного здоров'я виходячи з ушкоджень і повного здоров'я
        currentHealth = Mathf.Clamp(currentHealth + _value, 0, startingHealth);
    }

    //Перезавантаження рівня
    private void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

```
}

//Респаун гравця на чекпоінті
public void Respawn()
{
    death = false;

    GetComponent<PlayerMovement>().enabled = true;
    AddHealth(startingHealth);
    anim.ResetTrigger("death");
    anim.Play("Player_Idle");

    //Скидання набраних очків
    scriptScoreExperiense.currentScoreExperiense = 0;
    scriptScoreExperiense.textCountExperiense.text = "0";

    GetComponent<PlayerMovement>().attackBlocked = false;
    GetComponent<PlayerMovement>().knockPlayer = false;
    GetComponent<PlayerMovement>().recalcHeart.recalc = true;
    GetComponent<PlayerMovement>().rb.mass = 1f;
}[SerializeField] private float damage;

[Header("радіус атаки")]
[SerializeField] private float colliderDistance;
[SerializeField] private BoxCollider2D boxCollider;

[Header("Кількість урону")]
[SerializeField] private float loss;
```

```
[Header("шар з гравцем")]
[SerializeField] private LayerMask playerLayer;

private Animator anim;
private Rigidbody2D rb;

//доступ до здоров'я гравця
private PlayerLife playerHealth;

//доступ до патрулювання
private EnemyPatrol enemyPatrol;
private EnemyForwardPlayer enemyForwardPlayer;

[Header("---------------------------------------------------------------")]
[Header("Звук отримання урону")]
[SerializeField] private AudioSource damageSoundEffect;
[Header("Звук руху")]
[SerializeField] private AudioSource moveSoundEffect;
[Header("Звук атаки")]
[SerializeField] private AudioSource attackSoundEffect;

//доступ до скрипту та розміщення об'єкту гравця
private PlayerMovement playerMove;

[Header("об'єкт досвіду")]
[SerializeField] private GameObject experiensePoint;
[SerializeField] private int experienseCount;
```

```
[Header("-----------------------------------------------------------------")]
// точка напрямку ворога
private Vector2 moveDirection;

[Header("Швидкість руху при ухиленні")]
[SerializeField] private float moveSpeed = 0.02f;
[Header("Інтервал часу між змінами напрямку руху")]
[SerializeField] private float directionChangeInterval = 1.5f; // інтервал часу між змінами напрямку руху

private float timeSinceLastDirectionChange; // час, що минув з останньої зміни напрямку руху
[SerializeField] private int targetRotation = 1; // цільовий кут напрямку руху
private bool enemyOutOfBounds; //змінна, що перевіряє, чи вийшов ворог за межі гравця

[Header("-----------------------------------------------------------------")]
[Header("Максимальна відстань гравця від ворога для прослуховування звуку ворога")]
[SerializeField] private float maxDistance = 10f;
[Header("Максимальна гучність ворога")]
[SerializeField] private float maxVolume = 0.15f;
[Header("Мінімальна гучність ворога")]
[SerializeField] private float minVolume = 0f;    void Start()
   {
      anim = GetComponent<Animator>();
      rb = GetComponent<Rigidbody2D>();
```

```
        playerMove = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMovement>();
        playerHealth = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerLife>();
        enemyPatrol = GetComponent<EnemyPatrol>();
        enemyForwardPlayer = GetComponent<EnemyForwardPlayer>();
        timeSinceLastDirectionChange = directionChangeInterval;
    }

    void Update()
    {
        // Перевірка, чи ворог знаходиться в межах зони звуку гравця
        float distanceToPlayer = Vector2.Distance(transform.position, playerMove.transform.position);
        float volume = Mathf.Lerp(minVolume, maxVolume, 1 - (distanceToPlayer / maxDistance));
        moveSoundEffect.volume = volume;

        // Перевірка, чи ворог не вийшов за межі гравця
        if (enemyOutOfBounds)
        {
            moveSoundEffect.Stop();
            return;
        }

        // Переміщення ворога
        if (enemyPatrol == null || !enemyPatrol.enabled)
        {
```

```
            MoveTowardsPlayer();
        }

        // Атака гравця, якщо ворог знаходиться в радіусі атаки
        if (cooldownTimer < attackCooldown)
        {
            cooldownTimer += Time.deltaTime;
        }
        else
        {
            AttackPlayer();
        }
    }

    void MoveTowardsPlayer()
    {
        // Обчислення напрямку руху до гравця
        moveDirection = (playerMove.transform.position - transform.position).normalized;

        // Рух в напрямку гравця
        rb.MovePosition(rb.position + moveDirection * moveSpeed * Time.fixedDeltaTime);

        // Зміна напрямку руху через певний інтервал часу
        timeSinceLastDirectionChange += Time.deltaTime;
        if (timeSinceLastDirectionChange >= directionChangeInterval)
        {
```

```
            targetRotation = Random.Range(-1, 2);
            timeSinceLastDirectionChange = 0;
        }

        // Обертання ворога
        Quaternion targetRotationQuaternion = Quaternion.Euler(0, 0, targetRotation * 90);
        transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRotationQuaternion, 90 * Time.deltaTime);
    }

    void AttackPlayer()
    {
        // Перевірка, чи гравець знаходиться в радіусі атаки
        Collider2D[] colliders = Physics2D.OverlapBoxAll(transform.position, new Vector2(colliderDistance, colliderDistance), 0, playerLayer);
        foreach (Collider2D collider in colliders)
        {
            if (collider.CompareTag("Player"))
            {
                // Атака гравця
                playerHealth.TakeDamage(damage);
                anim.SetTrigger("Attack");
                cooldownTimer = 0;

                // Відтворення звуку атаки
                attackSoundEffect.Play();
            }
```

```
    }
}

public void TakeDamage(float amount)
{
    // Відтворення звуку отримання урону
    damageSoundEffect.Play();

    // Зменшення здоров'я ворога
    health -= amount;

    if (health <= 0)
    {
        // Виклик методу смерті ворога
        Die();
    }
}

void Die()
{
    // Відтворення звуку смерті
    deathSoundEffect.Play();

    // Відключення компонентів та колайдера ворога
    enemyPatrol.enabled = false;
    enemyForwardPlayer.enabled = false;
    GetComponent<Collider2D>().enabled = false;
```

```
        // Анімація смерті
        anim.SetTrigger("Die");

        // Знищення ворога через певний інтервал часу
        Destroy(gameObject, destroyDelay);
    }
}public class EnemyPatrol : MonoBehaviour
{
    [Header("Точки переміщення")]
    [SerializeField] private Transform leftEdge;
    [SerializeField] private Transform rightEdge;

    [Header("Розташування ворога")]
    [SerializeField] private Transform enemy;

    [Header("Швидкість переміщення")]
    [SerializeField] private float speed;

    // Вектор для зміни напрямку вигляду ворога
    private Vector3 initScale;

    // Змінна, що відповідає за рух вліво
    private bool movingLeft = false;

    [Header("Час простою ворога біля точки")]
    [SerializeField] private float idleDuration;
    private float idleTimer = 0;
```

```
[Header("Анімація ворога")]
[SerializeField] private Animator anim;

private void Awake()
{
    initScale = enemy.localScale;
}

// При вимкненому скрипті рух ворога зупиняється
private void OnDisable()
{
    anim.SetBool("moving", false);
}

private void Update()
{
    // Якщо персонаж не досягнув точки
    if (movingLeft)
    {
        // Якщо персонаж досягнув точки
        if (enemy.position.x >= leftEdge.position.x)
            MoveInDirection(-1);
        else
            DirectionChange();
    }
    else
    {
        if (enemy.position.x <= rightEdge.position.x)
```

```
                MoveInDirection(1);
            else
                DirectionChange();
        }
    }

    // Зміна напрямку персонажа
    private void DirectionChange()
    {
        anim.SetBool("moving", false);
        idleTimer += Time.deltaTime;

        if (idleTimer > idleDuration)
        {
            movingLeft = !movingLeft;
            idleTimer = 0;
        }
    }

    // Напрямок до певної точки
    private void MoveInDirection(int _direction)
    {
        anim.SetBool("moving", true);
        // Напрямок ворога
        enemy.localScale = new Vector3(Mathf.Abs(initScale.x) * _direction,
initScale.y, initScale.z);
        // Позиція ворога
```

```
            enemy.position = new Vector3(enemy.position.x + Time.deltaTime * _direction
* speed, enemy.position.y, enemy.position.z);
        }
    }public class AchievementUnlock : MonoBehaviour
    {
        // Змінна для визначення того, чи знайдено на рівні секретний предмет
        public int achievementUnlock = 0;

        // Скрипт виходу з рівня
        private TriggerDoor scriptTriggerDoor;

        private void Start()
        {
            scriptTriggerDoor = FindObjectOfType<TriggerDoor>();

            // Знищення предмета, якщо він вже був знайдений раніше
            if (PlayerPrefs.GetInt("LvAchievement" + scriptTriggerDoor.levelIndex) == 1)
                Destroy(gameObject);
        }

        private void OnTriggerEnter2D(Collider2D collision)
        {
            if (collision.tag == "Player")
            {
                achievementUnlock = 1;
                Destroy(this.gameObject);
            }
        }
```

```
}public class ScoreExperience : MonoBehaviour
{
    [Header("Об'єкт для виводу кількості досвіду")]
    [SerializeField] public Text textCountExperience;

    [Header("Об'єкт для виводу очок, які будуть додаватись до поточного рахунку")]
    [SerializeField] public GameObject textPlusExperience;

    [Header("----------------------------------------------------------------")]
    [Header("Поточний рахунок, кількість досвіду для нарахування, кількість досвіду для віднімання")]
    // Рахунок, який відображається
    [SerializeField] public int currentScoreExperience;
    // Нові очки для додавання до рахунку
    [SerializeField] public int newScoreExperience;
    // Нові очки для віднімання від рахунку
    [SerializeField] public int newMinusScoreExperience;

    [Header("----------------------------------------------------------------")]
    [Header("Звук збирання досвіду")]
    [SerializeField] private AudioSource pickupExperience;

    private void Awake()
    {
        currentScoreExperience = 0;
        newScoreExperience = 0;
        newMinusScoreExperience = 0;
```

```
    }

    void Update()
    {
        AddNewScoreExperience();
        DelNewScoreExperience();
    }

    // Функція, що відповідає за додавання нового досвіду
    private void AddNewScoreExperience()
    {
        if (newScoreExperience > 0)
        {
            pickupExperience.Play();

            textPlusExperience.GetComponent<Text>().text = "+" + newScoreExperience.ToString();
            textPlusExperience.GetComponent<Animator>().SetBool("on", true);

            // Зменшення нового рахунку на одиницю
            newScoreExperience--;
            // Додавання нового рахунку до поточного на одиницю
            currentScoreExperience++;

            // Оновлення тексту рахунку
            textCountExperience.text = currentScoreExperience.ToString();
        }
        else if (newScoreExperience <= 1)
```

```
            {
                textPlusExperience.GetComponent<Animator>().SetBool("on", false);
            }
        }

        // Функція, що відповідає за віднімання досвіду за ушкодження
        private void DelNewScoreExperience()
        {
            if (newMinusScoreExperience > 0)
            {
                textPlusExperience.GetComponent<Text>().text = "-" +
newMinusScoreExperience.ToString();
                textPlusExperience.GetComponent<Animator>().SetBool("on", true);

                // Зменшення нового від'ємного рахунку на одиницю
                newMinusScoreExperience--;
                // Віднімання очків рахунку від поточного на одиницю
                currentScoreExperience--;

                // Оновлення тексту рахунку
                textCountExperience.text = currentScoreExperience.ToString();
            }
            else if (newScoreExperience <= 1)
            {
                textPlusExperience.GetComponent<Animator>().SetBool("on", false);
            }
        }
    }public class TriggerDoor : DetectionZone
```

```
{
    // Анімація дверей
    private Animator anim;

    // Змінна для відкриття дверей
    private bool enterPlayerDoor;

    // Доступ до скрипту гравця для активації функції входу гравця в двері
    private PlayerMovement playerMoveToDoor;

    // Доступ до скрипту кнопки з дверима
    private ButtonDoor scriptButtonDoor;

    // Скрипт, що відповідає за рахунок алмазів
    private Diamondsbar levelDiamondsNum;

    [Header("Індекс поточного рівня")]
    [SerializeField] public int levelIndex;

    [Header("Об'єкт меню після завершення рівня")]
    [SerializeField] private GameObject continueMenu;

    [Header("Об'єкт для виводу кількості алмазів")]
    [SerializeField] private Text completeDiamondCount;

    [Header("Об'єкт для виводу кількості зібраного досвіду")]
    [SerializeField] private Text completeExpCount;
```

```
[Header("Об'єкт для виводу інформації про досягнення")]
[SerializeField] private Text completeAchievement;

[Header("Об'єкт для виводу тексту повідомлення")]
[SerializeField] private Text notificationText;

[Header("Об'єкти, які треба вимкнути при відкритті рівня")]
[SerializeField] private GameObject notificationObject;

[Header("Об'єкти, які треба вимкнути при відкритті рівня")]
[SerializeField] private GameObject[] disableObjects;

// Доступ до скрипту для розблокування досягнень
private AchievementUnlock scriptAchievementUnlock;

// Скрипт, що відповідає за рахунок досвіду
private ScoreExperience levelExperienceNum;

private float expPercentSum; // Змінна, що відповідає за бонус до кількості зароблено досвіду
private int saveExp; // Змінна для збереження досвіду в системі
private int bonusExp; // Кількість досвіду, яке додається як бонус

[Header("Напис про кількість бонусів до досвіду")]
[SerializeField] private Text bonusExpText;

// Доступ до скрипту завантаження
private LoadingScene scriptLoadingScene;
```

```
private void Awake()
{
    scriptButtonDoor = FindObjectOfType<ButtonDoor>();
    playerMoveToDoor = FindObjectOfType<PlayerMovement>();
    anim = GetComponent<Animator>();
    levelDiamondsNum = FindObjectOfType<Diamondsbar>();
    scriptAchievementUnlock = FindObjectOfType<AchievementUnlock>();
    levelExperienceNum = FindObjectOfType<ScoreExperience>();
}

private void Update()
{
    // Визначення наявності гравця в масиві
    if (detectedObjs.Count > 0)
    {
        scriptButtonDoor.EnterButtonDoor();

        // Відкриття дверей при натисканні кнопки
        if (enterPlayerDoor)
        {
            anim.SetTrigger("DoorOpen");
            playerMoveToDoor.EnterToDoor();
            enterPlayerDoor = false;
            scriptButtonDoor.buttonDoor.gameObject.SetActive(false);
            Invoke("CompleteLevel", 0.8f);
        }
    }
```

```
            else
            {
                scriptButtonDoor.EnterButtonDoor();
            }
        }

        // Функція для кнопки входу в двері
        public void ButtonEnterToDoor()
        {
            enterPlayerDoor = true;
        }

        // Функція завершення рівня
        private void CompleteLevel()
        {
            // Якщо кількість зібраних алмазів більша, ніж попередня, то зберегти нову кількість для цього рівня
            if (levelDiamondsNum.score > PlayerPrefs.GetInt("Lv" + levelIndex))
            {
                PlayerPrefs.SetInt("Lv" + levelIndex, levelDiamondsNum.score);
            }

            // Якщо секретний предмет не був знайдений, то зберегти його
            if (PlayerPrefs.GetInt("LvAchievement" + levelIndex) == 0)
            {
                // Якщо досягнення розблоковано, то зберегти його для виводу в таблиці
```

```
            PlayerPrefs.SetInt("LvAchievement" + levelIndex,
scriptAchievementUnlock.achievementUnlock);
        }

        StartCoroutine(AddBonusExp());

        // Вивести кількість знайдених алмазів на рівні в меню після проходження
        completeDiamondCount.text = levelDiamondsNum.score.ToString() + "/" +
"5";

        // Якщо досягнення підібрано, то вивести це в меню
        if (PlayerPrefs.GetInt("LvAchievement" + levelIndex) == 1)
        {
            completeAchievement.text = "Секретний предмет " + "знайдено";
        }
        else
        {
            completeAchievement.text = "Секретний предмет " + "не знайдено";
        }

        // Вимкнути елементи екрану для меню
        foreach (GameObject button in disableObjects)
        {
            button.SetActive(false);
        }

        // Відкрити меню після проходження рівня
        continueMenu.SetActive(true);
```

```
    }

    // Функція додавання досвіду та бонусу до нього
    private IEnumerator AddBonusExp()
    {
        // Збереження досвіду в загальну статистику персонажа
        saveExp = levelExperienceNum.currentScoreExperience;

        // Змінна, що відповідає за бонусний досвід, який обчислюється від основного
        expPercentSum = PlayerPrefs.GetInt("Exp1") + PlayerPrefs.GetInt("Exp2") + PlayerPrefs.GetInt("Exp3");

        // Збереження досвіду в системі
        if (expPercentSum > 0)
        {
            // Розрахунок очків досвіду, які будуть додані
            bonusExp = (int)Mathf.Round(levelExperienceNum.currentScoreExperience * expPercentSum / 100);
            saveExp += bonusExp;

            // Додавання нової кількості досвіду до існуючого
            saveExp += PlayerPrefs.GetInt("Experience");
            PlayerPrefs.SetInt("Experience", saveExp);
        }
        else if (expPercentSum <= 0)
        {
            // Додавання нової кількості досвіду до існуючого
```

```
            saveExp += PlayerPrefs.GetInt("Experience");
            PlayerPrefs.SetInt("Experience", saveExp);
        }

        // Цикл набору очок досвіду в панелі після завершення місії
        for (int i = 0; i <= levelExperienceNum.currentScoreExperience; i++)
        {
            completeExperienceCount.text = i.ToString();
            yield return new WaitForSeconds(0.01f);
        }

        // Вивести повідомлення про бонус та додати бонусний досвід
        if (expPercentSum > 0)
        {
            bonusExpText.GetComponent<Animator>().SetTrigger("on");
            bonusExpText.text = expPercentSum.ToString() + "%\nбонус";
            yield return new WaitForSeconds(1.3f);

            for (int i = 0; i <= bonusExp; i++)
            {
                completeExperienceCount.text                                    =
(levelExperienceNum.currentScoreExperience + i).ToString();
                yield return new WaitForSeconds(0.01f);
            }
        }
    }

    private void ButtonBackMapInContinueMenu()
```

```
        {
            // Повернутися до карт з рівнями
            UIManager.instance.BackMapSelection();
        }

        public void ButtonNextLevel()
        {
            // Умова для кнопки, яка переходить на наступний рівень. Якщо рівень на локації дорівнює 5,
            // то необхідно повернутися до меню
            if (levelIndex % 5 == 0)
            {

notificationObject.GetComponent<Animator>().SetTrigger("openNotification");
                notificationText.text = "Усі рівні на даній локації відкриті. Поверніться назад до меню";
            }
            else if (PlayerPrefs.GetInt("Lv" + levelIndex) > 0 && levelIndex % 5 != 0)
            {
                scriptLoadingScene = FindObjectOfType<LoadingScene>();
                scriptLoadingScene.LoadScene("Level " + (levelIndex + 1).ToString());
            }
            else
            {

notificationObject.GetComponent<Animator>().SetTrigger("openNotification");
                notificationText.text = "Рівень " + (levelIndex + 1).ToString() + " ще не відкритий. Поверніться назад до меню";
```

```
        }
    }
}public class MainMenu : MonoBehaviour
{
    [SerializeField] private GameObject mainMenu;
    [SerializeField] private GameObject settingsMenu;

    private LoadingScene scriptLoadingScene;

    [SerializeField] private AudioSource buttonPressSound;

    private void Start()
    {
        StartCoroutine(TimerOpenMainMenu());
    }

    public void OpenMainMenu()
    {
        buttonPressSound.Play();
        StartCoroutine(TimerOpenMainMenu());
    }

    private IEnumerator TimerOpenMainMenu()
    {
        settingsMenu.GetComponent<Animator>().SetBool("openMenu", false);
        mainMenu.GetComponent<Animator>().SetBool("openMenu", true);
        yield return new WaitForSeconds(1f);
    }
```

```
    public void OpenSettingsMenu()
    {
        buttonPressSound.Play();
        StartCoroutine(TimerOpenSettingsMenu());
    }

    private IEnumerator TimerOpenSettingsMenu()
    {
        settingsMenu.GetComponent<Animator>().SetBool("openMenu", true);
        mainMenu.GetComponent<Animator>().SetBool("openMenu", false);
        yield return new WaitForSeconds(1f);
    }

    public void ExitApplications()
    {
        buttonPressSound.Play();

#if UNITY_EDITOR
        UnityEditor.EditorApplication.Exit(0);
#else
        Application.Quit();
#endif

    }

    public void DeleteAllData()
    {
```

```
        buttonPressSound.Play();
        PlayerPrefs.DeleteAll();
    }

    public void OpenMapLevels()
    {
        buttonPressSound.Play();
        scriptLoadingScene = FindObjectOfType<LoadingScene>();
        scriptLoadingScene.LoadScene("MapLevels");
    }
}public class DialogManager : MonoBehaviour
{
    [Header("об'єкт для виводу імені NPC")]
    [SerializeField] public Text nameText; // Текстове поле для виводу імені NPC
    [Header("об'єкт для виводу пропозицій NPC")]
    [SerializeField] public Text dialogText; // Текстове поле для виводу пропозицій NPC
    [Header("об'єкт діалогового вікна для анімації його відкриття")]
    [SerializeField] public Animator anim; // Аніматор для анімації відкриття діалогового вікна
    [Header("звук набору тексту")]
    [SerializeField] private AudioSource typingSound; // Звук набору тексту
    [Header("Іконка NPC")]
    [SerializeField] public Image iconSprite; // Зображення іконки NPC
    [Header("Спрацювання конкретного скрипту для NPC")]
    [SerializeField] public bool scriptTriggerToNPC; // Спрацювання конкретного скрипту для NPC
```

```
// Ігрові об'єкти, які з'являються після завершення діалогу
[SerializeField] private GameObject[] uiButtons; // Масив кнопок інтерфейсу

// Виділення пам'яті під чергу пропозицій NPC
private Queue<string> sentences; // Черга речень

public void Start()
{
    // Створення масиву пропозицій
    sentences = new Queue<string>();
}

// Функція початку діалогу при натисканні кнопки
public void StartDialog(Dialog dialog)
{
    // Відкриття діалогового вікна
    anim.SetBool("isOpen", true);

    nameText.text = dialog.name;

    // Підстановка зображення іконки персонажа
    iconSprite.GetComponent<Image>().sprite = dialog.icon;

    // Очищення масиву пропозицій
    sentences.Clear();

    // Цикл для додавання кожної пропозиції NPC до масиву
    foreach (string sentence in dialog.sentences)
```

```
        {
            sentences.Enqueue(sentence);
        }

        // Перехід до наступного речення
        DisplayNextSentence();
    }

    // Функція переходу до наступного речення
    public void DisplayNextSentence()
    {
        // Якщо кількість пропозицій дорівнює 0, то діалог закінчується
        if (sentences.Count == 0)
        {
            EndDialog();
            return;
        }

        // Видалення поточного речення з масиву і підстановка його в змінну для виводу
        string sentence = sentences.Dequeue();
        StopAllCoroutines();
        StartCoroutine(TypeSentence(sentence));
    }

    // Вивід посимвольно тексту в діалоговому вікні
    IEnumerator TypeSentence(string sentence)
    {
```

```
        dialogText.text = "";
        // Перебір усіх букв у реченні і їх вивід
        foreach (char letter in sentence.ToCharArray())
        {
            typingSound.Play();
            dialogText.text += letter;
            yield return null;
        }
    }

    // Функція завершення діалогу
    public void EndDialog()
    {
        scriptTriggerToNPC = true;
        anim.SetBool("isOpen", false);
        StartCoroutine(UiObjectVisible());
    }

    IEnumerator UiObjectVisible()
    {
        // Перебір об'єктів і включення їх відображення
        foreach (GameObject button in uiButtons)
        {
            button.SetActive(true);
        }
        yield return new WaitForSeconds(1.5f);
    }
}
```