**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня бакалавр**

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційне і програмне забезпечення ігрової системи піджанру рольовий бойовик»

здобувача групи ІН - 03 Годонюка Дмитра Сергійовича

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

_____ Дмитро ГОДОНЮК

(підпис)

Керівник, кандидат технічних наук
доцент кафедри комп'ютерних наук Ігор ШЕЛЕХОВ _____

(підпис)

**Суми – 2024**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня бакалавра**

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-03 Годонюка Дмитра Сергійовича

1. Тема роботи: «Інформаційне і програмне забезпечення ігрової системи піджанру рольовий бойовик»
затверджую наказом по СумДУ від *«» червня 2024 р.*
2. Термін здачі здобувачем кваліфікаційної роботи *до червня 2024 р.*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1)Аналіз предметної області, постановка й формування завдань дослідження. 2)Огляд технологій, що використовуються для розробки мобільного додатку. 3)Розробка та тестування ігрового рішення. 4)Аналіз отриманих результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «___» _____ ____ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| *1* | *Аналіз предметної області, постановка й формування завдань дослідження* | 06-08.05.2024 | |
| *2* | *Огляд технологій, що використовуються для розробки мобільного додатку* | 09-11.05.2024 | |
| *3* | *Розробка та тестування ігрового рішення* | 12-14.05.2024 | |
| *4* | *Аналіз отриманих результатів* | 15-17.05.2024 | |
| *5* | *Оформлення пояснювальної записки до кваліфікаційної роботи* | 18-20.05.2024 | |

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

# АНОТАЦІЯ

**Записка**: 75 сторінок, 29 рисунків, 3 додатки, 2 таблиці, 33 використаних джерел.

**Обґрунтування актуальності теми роботи** – тема кваліфікаційної роботи є актуальною, оскільки мобільна ігрова індустрія стрімко зростає, залучаючи все більше гравців. Цей ріст обумовлений підвищеною доступністю мобільних пристроїв, розширенням мобільних ігрових платформ і підвищенням популярності ігор для мобільних пристроїв. Завдяки цьому виникають значні перспективи для розробників ігор, які можуть досягти успіху та отримати прибуток. Постійний прогрес у технологіях відкриває нові можливості для створення більш складних, реалістичних і захоплюючих мобільних ігор. Отже, розробка мобільних ігор має великий потенціал і приваблює багато гравців.

**Об’єкт дослідження** — процес розробки ігрової системи піджанру рольовий бойовик.

**Предмет дослідження** — новітні інструменти для реалізації ігрового рішення.

**Мета роботи** — розробка інформаційного і програмного забезпечення ігрової системи піджанру рольовий бойовик, що включає створення інтуїтивно зрозумілого інтерфейсу, розробку алгоритмів штучного інтелекту для персонажів та забезпечення високої продуктивності й стабільності роботи.

**Методи дослідження** — аналіз ігрових рушіїв, пошук методів прискорення розробки, уніфікованих алгоритмів та прототипів.

**Результати** — розроблено повнофункціональну ігрову систему піджанру рольовий бойовик, яка має зрозумілий інтерфейс, різновид ворогів та анімацій, система інвентаря дозволяє зберігати предмети, система навичок дає можливість розвивати персонажа та збереження прогресу, що дозволяє гравцям зберігати свої досягнення, забезпечуючи комфортне та безперервне ігрове задоволення.

ІГРОВА СИСТЕМА, ІГРОВИЙ РУШІЙ, C#, UNITY, BLENDER

**ЗМІСТ**

# ВСТУП

**Актуальність.** Тема кваліфікаційної роботи є актуальною, оскільки мобільна ігрова індустрія стрімко зростає, залучаючи все більше гравців. Цей ріст обумовлений підвищеною доступністю мобільних пристроїв, розширенням мобільних ігрових платформ і підвищенням популярності ігор для мобільних пристроїв. Завдяки цьому виникають значні перспективи для розробників ігор, які можуть досягти успіху та отримати прибуток. Постійний прогрес у технологіях відкриває нові можливості для створення більш складних, реалістичних і захоплюючих мобільних ігор. Отже, розробка мобільних ігор має великий потенціал і приваблює багато гравців.

**Об’єкт дослідження.** Інформаційне і програмне забезпечення ігрової системи піджанру рольовий бойовик.

**Предмет дослідження.** Новітні інструменти для реалізації ігрового рішення.

**Гіпотеза.** Використання сучасного інформаційного та програмного забезпечення в ігрових системах піджанру рольовий бойовик значно покращує взаємодію користувача з грою, забезпечуючи більш зрозумілий інтерфейс, реалістичні анімації, різноманітність ворогів, ефективну систему інвентаря та навичок, а також надійне збереження прогресу, що в результаті підвищує задоволеність та залученість гравців.

**Новизна.** Використання C# для розробки складних алгоритмів поведінки ворогів, забезпечуючи більш реалістичний та динамічний ігровий процес. Використання Unity для створення надійної та ефективної системи збереження ігрового прогресу, яка дозволяє зберігати різні стани гри. Застосування технологій Unity для динамічного створення ігрових рівнів та сценаріїв. Використання мовних та інструментальних можливостей C# для оптимізації коду, що дозволяє досягти високої продуктивності та стабільності гри.

**Структура.** Дана робота складається зі вступу, інформаційного огляду, постановки задачі, вибору методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

# 1 ІНФОРМАЦІЙНИЙ ОГЛЯД

## 1.1 Мобільні ігри та їх класифікація

Протягом останніх кількох років ігрова індустрія спостерігає надзвичайний ріст, який в першу чергу можна пояснити збільшенням популярності мобільних ігор. Сучасні мобільні пристрої, такі як смартфони і планшети, надають користувачам захоплюючий і надзвичайно підходящий для гри досвід.

Мобільні ігри стали однією з перших форм цифрового розвагового контенту, які можна було брати з собою в будь-яке місце, і сьогодні вони залишаються однією з ключових причин популярності мобільного геймінгу. Можливість мати доступ до розваг у будь-якому місці і часі важлива для багатьох. Це ідеальний спосіб розважитися під час очікування в черзі або під час подорожі автобусом або поїздом. Крім того, мобільні ігри зазвичай більш доступні з фінансової точки зору порівняно з іграми для ігрових консолей, що робить їх прекрасним вибором для тих, хто має обмежений бюджет.[3]

Мобільний геймінг відзначається кількома основними характеристиками, які виділяють його у власну категорію. Найголовнішою серед них є безперервна доступність улюблених ігор, які, фактично, завжди наявні у вашій кишені. Це призводить до ключової особливості мобільного геймінгу - більш часті, але коротші ігрові сесії. Багато популярних мобільних ігор спеціально розроблені так, щоб гравець міг повертатися до гри протягом дня з певною регулярністю.[1]

Мобільні ігри – це відеоігри, призначені для гри на різних мобільних пристроях, таких як смартфони, планшети та інші. Ці ігри розробляються та оптимізуються спеціально для таких пристроїв, з урахуванням їх обмежених можливостей і особливостей взаємодії з користувачем, які здійснюються через сенсорний екран та акселерометр. Їх можна завантажити з магазинів додатків, таких як Apple App Store або Google Play, або завантажити їх безпосередньо на пристрій. Крім того, деякі ігри можна грати з використанням хмарних сервісів.

В минулому, можливості мобільних ігор були значно обмежені. Перші ігри були досить простими, обмеженими в можливостях та менш продуманими. Вони мали низьку якість графіки, мінімальні або навіть жодні спеціальні ефекти, прості графічні символи, обмежену функціональність та були вразливі до затримок в роботі та швидким рухам.

Коли розвиток мобільного апаратного забезпечення був на початковому етапі, індустрія розробки мобільних ігор стикалася з численними викликами, такими як короткий час автономної роботи, обмежена доступна пам'ять і обмежені можливості графічного обладнання.

Однак революційним моментом для індустрії стала поява смартфонів. Із запровадженням цих пристроїв стали можливими значні досягнення завдяки дослідженням, розробкам та аналізу ринку, що призвели до послідовних версій високопродуктивних мобільних пристроїв, які із кожним новим поколінням отримували значно краще програмне і апаратне забезпечення.

Мобільні платформи не мають обмежень, характерних для розробки ігор для консолей. Вони більш доступні і дешеві у розробці, та не потребують великих видавців. З'явлення Android і iPhone також позначилося на покращенні якості розроблених мобільних ігор та зменшенні розриву з консольними проектами.[5]

Сюжетний світ гри зазвичай обмежений жанром гри, який визначає форму, тематику та техніку в іграх, а також в інших видових творах, таких як література, кіно та музика. Існують дві основні категорії ігрових жанрів, які базуються на якостях оповіді та ігрового процесу. Жанр оповідання використовується для класифікації ігор за широкими тематичними категоріями, такими як фентезі, історія, наукова фантастика, війна та інші. Цей аспект ігрового жанру описує сюжет і тематику гри, незалежно від її механіки та стилю візуального виконання. Також ігри можна класифікувати за типом геймплею, який вони пропонують. Це визначає сам процес гри і як гравці взаємодіють з ігровим світом. Важливо розрізняти між геймплеєм та

сюжетним жанром, оскільки ці аспекти можуть відрізнятися один від одного, хоча в грі можуть бути обидва.

У деяких відеоіграх основний акцент робиться на розповіді складних історій з деталізованими персонажами і захопливими сюжетами, а також на створенні захоплюючого ігрового середовища. Такі ігри можна віднести до категорії сюжетних. Інші ж ігри більше спрямовані на ігрову механіку та виклики, з меншим наголосом на розповідь. Ці ігри можна віднести до категорії ігрових. Також є відеоігри, які призначені для однієї особи, де гравець контролює всіх персонажів та взаємодіє з ігровим світом. Інші ж ігри розроблені для гри з кількома гравцями, будь то локально чи онлайн. Ці ігри відомі як ігри для багатьох гравців.

Чимало відеоігор не обов'язково належать до однієї з цих категорій. Наприклад, гра може поєднувати як сюжетні, так і багатокористувацькі елементи, або вона може бути одночасно ігровою та призначеною для одного гравця.

Жанри використовуються для систематизації та класифікації відеоігор згідно з їхніми ігровими механіками, налаштуваннями, тематикою та іншими характеристиками. Деякі зі стандартних жанрів відеоігор включають бойовики, пригоди, рольові ігри, симулятори, стратегії та спортивні ігри.

Жанри відеоігор можуть бути або загальними і універсальними, охоплюючи різноманітні типи ігор, або вузькоспеціалізованими. Наприклад, жанр "екшн" є широкою категорією, що включає різноманітні піджанри, такі як шутери від першої особи, ігри про хакерів та платформери. У той час як жанр "рольової гри" є більш конкретним і, як правило, стосується ігор, які включають в себе створення персонажа, його розвиток та прийняття рішень в фентезійному або науково-фантастичному світі.

Поняття жанру є корисним як для геймерів, так і для розробників, оскільки воно сприяє кращому розумінню важливих особливостей та очікувань від гри, а також допомагає знайти інші ігри, що схожі за стилем або змістом. Проте важливо враховувати, що межі між жанрами можуть бути

нечіткими, і багато ігор поєднують елементи з кількох жанрів, ускладнюючи їх класифікацію.[6]

Рольовий бойовик (англ. Action role-playing game, Action RPG, ARPG) є одним із ключових піджанрів у жанрі рольових відеоігор. Ця категорія включає в себе RPG-ігри, в яких бойова система поєднує механіку рольових ігор з динамічними бойовими системами, спрямованими на рефлекси, схожими на ті, що зазвичай зустрічаються у відеоіграх жанру екшн. Відмінністю є відсутність більш абстрактних бойових систем, характерних для типових RPG (таких як покроковий бій, умовно-покроковий бій або тактичний бій у реальному часі з можливістю паузи). У цьому піджанрі гравці мають прямий контроль над рухами та діями свого персонажа під час бою, вони повинні натискати кнопки атаки (або комбінації кнопок) для атаки ворогів у реальному часі. Рольовий бойовик поєднує в собі вишукану RPG-механіку з динамічною аркадною бойовою системою, яка вимагає від гравця відмінних навичок та реакцій.[7]

Ігровий процес у кожній відеогрі є унікальним, і його складові елементи можуть варіюватися. Проте, існують загальні аспекти, які спільні багатьом іграм. Багато ігор мають рівні, на яких гравцеві потрібно контролювати свого головного героя, заробляючи очки, збираючи бонуси для покращення його характеристик і використовуючи спеціальні атаки, щоб перемогти ворогів або уникнути їх. Пошкодження головного героя призводить до зменшення його здоров'я, і якщо це здоров'я досягне нуля або головний герой потрапить у безвихідну ситуацію, гра завершується. Багато рівнів ігор, а також їхні фінальні етапи, завершуються боєм з босом, якого гравець повинен перемогти, щоб продовжити або завершити гру. Деякі ігри також включають можливість збереження гри на проміжних точках, де гравець може зберегти свій прогрес на носії даних. Ця можливість дозволяє гравцю завантажити свій прогрес у випадку поразки або потреби в припиненні гри, щоб потім повернутися до неї. Це лише декілька прикладів загальних аспектів ігрового процесу, які можна обговорювати годинами.

## 1.2 Аналіз продуктів аналогічного призначення

Розглянемо декілька рольових бойовиків для мобільних пристроїв та проведемо їх порівняльний аналіз.

Almora Darkosen RPG — це рольова гра, створена польським розробником Гжегожом Борковським. Дія гри розгортається у фентезійному світі під назвою Альмора, де ви маєте можливість досліджувати різноманітні локації, взаємодіяти з неігровими персонажами, виконувати різноманітні квести, збирати та створювати ігрові предмети, торгувати з купцями, боротися з ворогами та розвивати свого персонажа.

Гра в стилі піксельного мистецтва, що викликає відчуття ностальгії. Візуально графіка яскрава та деталізована, а анімації в ній відзначаються плавністю та реалістичністю. Гра також супроводжується чарівним саундтреком, який ідеально вписується в атмосферу кожної сцени. Звукові ефекти також виконані на високому рівні і сприяють загальній атмосфері гри.

Гра пропонує обширний та різноманітний відкритий світ для вільного дослідження. У цьому світі ви маєте можливість відвідувати різноманітні регіони, такі як пустелі, гори, ліси, болота, печери, підземелля і міста. Кожен з них має власний унікальний ландшафт, рослинний та тваринний світ, а також погоду і приховані загадки. Гра також пропонує різноманітні та захопливі квести, починаючи від простих завдань на доставку та закінчуючи складними розслідуваннями і битвами. Є можливість розвивати різні навички вашого персонажа завдяки системі навичок в грі. У вас є можливість обрати одну з трьох основних категорій навичок: бій, магія та виживання. Кожна з цих категорій має в собі кілька підкатегорій, які пропонують широкий спектр переваг та ефектів для вашого персонажа. Система здобичі в грі дозволяє гравцям знаходити та збирати різноманітні предмети з ворогів, ящиків, бочок, скриньок та інших подібних джерел. Крім того, у гравця є можливість користуватися системою крафту для створення різноманітних видів предметів та системою торгівлі для продажу та купівлі різних предметів.[8]

Скріншот гри «Almora Darkosen RPG» зображено на рисунку 1.1.


Рисунок 1.1 – Скріншот гри «Almora Darkosen RPG»

Anima ARPG — це рольовий бойовик, розроблений італійською компанією Redeev, який вражає безперебійним та захоплюючим боєм, майже бездоганним управлінням і простою системою крафту. Гра відзначається художньою графікою у стилі зверху вниз і моторошним музичним супроводженням, що створює особливу атмосферу.

У грі присутні різноманітні класи персонажів у стилі класичних рольових ігор, які дають вам можливість обрати роль воїна, мага або стрільця. Кожен клас зазвичай володіє своїми унікальними сильними сторонами та здібностями. Наприклад, маг може володіти здатністю телепортації, а воїн вражаючою силою у ближньому бою, тоді як стрілець здатний атакувати ворогів на віддаленій дистанції.

Anima ARPG реалізовує справжню суть хардкорного підземелля. Хоча босів у грі не так вже й багато, кожен з них становить справжній виклик. Зазвичай, коли ви зіткнетесь з босом, ви відчуєте себе абсолютно непідготовленими. Для того щоб перемогти босів, обов'язково потрібна стратегія та терпіння. Ця гра може стати справжнім випробуванням для деяких гравців. Вивчення локацій є необхідним, щоб досягти високого рівня, який дає шанс на виграш в битві з будь-яким босом.

Керування в грі інтуїтивно зрозуміле, а рух персонажа відбувається плавно. Немає незручних екранних кнопок, як у деяких мобільних іграх, які заважають геймплею.

В грі зустрічаються різноманітні вороги, починаючи звичайними зомбі і закінчуючи наелектризованими чудовиськами, які відомі як Обпалені. Деякі з них оточені аурами, які можуть зменшити ваші навички або підсилити їхні. Однак найбільше обурює проклята аура, що робить персонажа вразливішим, зменшуючи його здоров'я вдвічі, поки він перебуває в її зоні впливу.

Сюжет у грі, на жаль, є рідкісний і практично порожній. Проте захопливий геймплей та інтенсивна дія гри компенсують відсутність виразного сюжету.[9]

Скріншот гри «Anima ARPG» зображено на рисунку 1.2.


Рисунок 1.2 – Скріншот гри «Anima ARPG»

Eternium - це мобільна рольова екшн-гра, створена компанією Making Fun, Inc., яка розгортається в фантастичному світі, де гравець виступає у ролі героя, що веде боротьбу проти сил темряви. Сюжет гри захоплюючий і майстерно прописаний, з різноманітними квестами, підземеллями і босами, з якими гравцю доведеться зіткнутися.

Зручний інтерфейс - одна з основних особливостей Eternium. Управління в грі інтуїтивно зрозуміле, що робить навігацію більш комфортною для гравців.

Геймплей Eternium вражає своєю швидкістю і насиченістю подіями. Бойова система детально розроблена і включає в себе різноманітні навички та здібності, які гравцю необхідно освоїти. Елементи управління в грі чутливі, що дозволяє гравцям виконувати дії без зайвих затримок.

Eternium пропонує три основні класи персонажів: воїн, маг і мисливець за головами. Кожен клас володіє унікальними навичками та здібностями, і гравці можуть змінювати класи в будь-який момент. Система розвитку персонажів також добре продумана, дозволяючи гравцям поліпшувати навички і здібності своїх персонажів по мірі прогресу проходження гри.

Маг виконує роль заклинача, що контролює різні стихійні чари. Воїн є передовим бійцем ближнього бою з потужними атаками мечем і щитом. Мисливець за головами майстер знищення ворогів на відстані, використовуючи різноманітні гармати та пастки.

Eternium має вражаючу графіку з високоякісними текстурами та добре деталізовані моделі персонажів. Звукова сторона гри також на високому рівні, з різноманітними звуковими ефектами під час бою, заклинань і використання здібностей. Музика в грі додає особливого настрою та гармонійно вписується в загальну атмосферу.[10]

Скріншот гри «Eternium» зображено на рисунку 1.3.


Рисунок 1.3 – Скріншот гри «Eternium»

Розглянувши декілька рольових бойовиків для мобільних пристроїв можемо підвести підсумки:

Almora Darkosen RPG - це рольова екшн-гра в стилі ретро з захопливим сюжетом. Дія гри розгортається на острові Альмора, який приховує безліч

різноманітних та захопливих локацій. Графіка вражає простотою, але в той же час є яскравою та деталізованою, а анімації відзначаються плавністю та реалістичністю. Бойова система детально розроблена і включає в себе різноманітні навички та здібності. У грі вас чекає широкий та різноманітний відкритий світ, населений монстрами та загадками, який варто вивчати і досліджувати.[8]

Anima ARPG - це захоплюючий мобільний рольовий бойовик з чудовим та надійним геймплеєм. Гра відзначається художньою графікою у стилі зверху вниз і моторошним музичним супроводженням, що створює особливу атмосферу. В грі зустрічаються різноманітні вороги, починаючи звичайними зомбі і закінчуючи босами, де зустріч з кожним з них становить справжній виклик. Незважаючи на відсутність глибокого сюжету, гра пропонує веселий ігровий процес, а оточуючий світ завжди інтригує, утримуючи ваш інтерес протягом проходження усієї компанії.[9]

Eternium - це добре продумана мобільна гра в жанрі рольовий екшн, яка захоплює зручним інтерфейсом, захоплюючим геймплеєм, вражаючою графікою з добре деталізованими моделями персонажів, майстерно прописаним сюжетом, що включає в себе велику кількість різноманітних квестів, підземеллями і босами, з якими гравцю доведеться зіткнутися, а також має музичну і звукову сторону на високому рівні. Система розвитку персонажів також добре продумана, дозволяючи гравцям поліпшувати навички і здібності своїх персонажів по мірі прогресу проходження гри. Гра підходить для будь-якого любителя мобільних ігор і є обов'язковою для проходження.[10]

Порівняльна характеристика продуктів аналогічного призначення представлена в таблиці 1.1.

Таблиця 1.1 - Порівняльна характеристика продуктів аналогічного призначення

| Категорії | Almora Darkosen RPG | Anima ARPG | Eternium |
|---|---|---|---|
| Сюжет | + | - | + |
| Графіка | + | + | + |
| Анімації | + | + | + |
| Інтерфейс | + | + | + |
| Аудіосупроводження | + | + | + |
| Динамічність | + | + | + |
| Оптимізація | + | + | + |

Порівнюючи ігри, можна встановити, що всі показники знаходяться майже на одному рівні. Однак Anima ARPG поступається своїм конкурентам у якості гарного сюжету.

### 1.3 Постановка задачі

За результатами проведеного аналітичного огляду сучасних мобільних ігор, їх різновидностей, класифікації було визначено актуальність розробки інформаційного і програмного забезпечення ігрової системи піджанру рольовий бойовик. Для того щоб досягти поставленої мети, необхідно виконати ряд завдань:

1) розробку ігрової концепції;
2) прототипування ігрового середовища;
3) розробку дизайну гри;
4) вибір середовища розробки та мови програмування;
5) програмну реалізацію ігрової системи;
6) тестування програмного забезпечення.

# 2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

## 2.1 Розробка ігрової концепції

Для досягнення цілі в грі нам необхідно використовувати різноманітні стратегії. Наприклад, використання внутрішнього поля, втручання в гру і вплив на інших гравців можна вважати ефективними способами досягнення мети, проте їх не можна розглядати як окремі методи або гарантії перемоги.

Засоби, що можна використовувати у грі для досягнення перемоги, це той підхід, що називається ігровим методом. Іншими словами, це спосіб, який можна використовувати для досягнення цілі у грі.

Правила гри встановлені для заборони певних ефективних стратегій, що допомагають досягти результату перед початком гри. Наприклад, постійне сповільнення у грі "камінь-ножиці-папір" є ефективним, але забороненим. Правила разом з чітко визначеними цілями перед грою, створюють умови, необхідні для того, щоб грати в гру за правилами, які ми вважаємо загальноприйнятими.

Всі методи, які є найбільш простими, ефективними, легкими та зрозумілими, завжди виключаються з правил побудови, вдаючись до більш складних та складних альтернатив.

Отже, правила гри визначають оптимальний спосіб досягнення мети гри і спонукають нас використовувати менш ефективні методи. Іншими словами, гра має свої обмеження, і гравець повинен уміло користуватися ними для досягнення своїх цілей.

Разом із основними правилами побудови існують розширені правила, такі як каральні, що можуть передбачати, наприклад, тримінутний захист на баскетбольному майданчику. Ці правила не роблять гру неможливою при порушенні, але передбачають покарання, тому вони лише доповнюють основні правила побудови.

Складно уявити, як гравець добровільно відмовляється від швидких засобів досягнення мети і згоджується на обмеження участі в грі за "неефективним" методом. Саме такий підхід робить гру більш гладкою.

Концепція гри включає в себе встановлення її мети, а правила, що стосуються її структури, формують основу логіки гри. Різні правила можуть встановлювати різні обмеження та наслідки, що впливають на способи досягнення ігрових цілей. Для досягнення оптимального балансу між цілями, методами та правилами необхідна систематична ігрова механіка, яка регулює взаємодію цих елементів.

Важливо зазначити, що концепцію механіки та правил можна легко сплутати. Фактично, механіка функціонує як додатковий шар до правил, що розширює їх, з метою поліпшення досвіду гравців під час гри. Наприклад, механіка нагород та покарань доповнює правила у даному контексті, з метою поліпшення гри в цілому.

Всі гравці мають доступ до відкритих та згоджених правил, тоді як ігрові механізми можуть залишатися прихованими. Наприклад, у випадку, коли гравець потрапляє в складну ситуацію, гра може автоматично знизити її складність, щоб забезпечити більшу участь гравця. Це відображає роль механік у координації та збалансуванні процесу, а головне - їхню здатність поліпшувати досвід гравця.

Механіка - це концепція функціонування, що об'єднує всі складові об'єкта та спонукає їх працювати у взаємодії. У гральній сфері ігрова механіка не повинна бути лише абстрактною і внутрішньо логічною концепцією, відокремленою від цілей, методів та правил гри. Замість цього, вона є збалансованою системою, що ґрунтується на існуючих принципах.

Ігрова механіка - це метод, що активується гравцем для полегшення взаємодії з грою. Найкращим способом усвідомити ігрову механіку як методи - це уявити її у формі дієслів, таких як повзти, їздити (кінь), колоти, стрибати, стріляти (стріла), хапати, бігати і так далі. Усі ці дієслова відображають дії, які гравець може використовувати для взаємодії з грою з метою досягнення поставлених цілей в межах правил гри.

Але ігрова механіка — це не лише метод, а й ігрові елементи, режими, навички, стан та інше. Вона є результатом явних та прихованих систем у

процесі досягнення та узгодження цілей гри, методів та правил, та на них впливають усі ці аспекти одночасно.

Основні ігрові механіки:

— Механіка часу: день, ніч, години, хвилини, секунди.
— Механіка протистояння: раунд, напівраунд, реальний час, обмежений час.
— Механіка пошуку партнерів: реальні люди (боротьба, об’єднання), боти (товариші по команді, суперники).
— Механіка місії: умови (тригер, завершення), цілі (бігти, збирати, бити, діалог).
— Механіка ресурсів: норми виробництва, норми споживання, регіональні обмеження.
— Сценарна механіка: карти, рівні, механіка вкладення, координати місць.
— Просторова механіка: відкрита, закрита.
— Механіка дій: стояти, сидіти, присідати, ходити, бігати, стрибати, літати, плавати.
— Механіка поведінки: стріляй, хапай, рубай, веди, керуй, переключайся.
— Механіка персонажа: виживання (життя, витривалість), атаки (проникнення (ігнорування), розбивання броні (зменшення), швидкість атаки), захист (броня, опір, парирування (без пошкоджень), блок (зменшена шкода)), спритність (швидкість руху, ухилення), тип (фізичний, магічний, стихійний (лід, вода, вогонь, вітер, блискавка, вода, земля, світло, темрява)), статус (нормальний, стан (тригер, кінець), зміна (посилення, ослаблення)).
— Механіка навичок: період (перед ударом, після удару, післяударний період, відновлення), інтервали (обов'язкові, необов'язкові).
— Комбо-механіка: комбінації типів, комбінації навичок, комбінації статусів, ефекти (посилення, ослаблення).
— Механіка предметів: обладнання (числові ефекти, обмеження), реквізит (споживання, тривалість, числові ефекти).

— Механіка росту: досвід (персонажі, обладнання), рівні (персонажі, обладнання, реквізит), виготовлення (обладнання, предмети), умови (обладнання, предмети, ресурси, ймовірність), додаткові атрибути (обладнання, предмети).
— Соціальна механіка: чат, друг, шлюб, майстер і учень.
— Механіка табору: команди, гільдії, банди, професії, країни, раси.
— Чисельна механіка: фіксовані величини, відсотки.
— Стохастична механіка: на основі правил, на основі ймовірностей.
— Змагальна механіка: бойова сила, коефіцієнт виграшу, рейтинг.
— Механіка винагороди та покарання: винагорода (умови, результати), покарання (смерть, невдача).
— Економічна механіка: торгова система (токени, реальні гроші).
— Механіка балансування.

Гра має різноманітні ігрові механіки, які забезпечують гравцям широкий спектр досвіду та можливостей. Хоча вона складається з різних механік, основною є ключова ігрова механіка.

З системного погляду, основна механіка визначається як геймплей, спрямований на досягнення кінцевого стану гри. Наприклад, у шутерах від першої особи стрілянина є основною механікою, оскільки саме вона переважно сприяє досягненню завершального етапу гри.

У певних іграх відсутній чіткий кінцевий пункт, проте гравець має фокусуватися на досягненні конкретного стану, наприклад, підтримки міського бюджету у найкращому стані. Ці ігри використовують певний набір ігрових механік, спрямованих на ці стани, що дозволяє говорити про базові механіки, навіть якщо відсутній визначений кінцевий стан в системі. Таким чином, в деяких іграх головна механіка полягає у досягненні стану рівноваги.

Ігровий процес виникає внаслідок взаємодії гравця з грою через певні ігрові механіки. Так звані типи ігор складають лише частину гри, вони визначаються узгодженим тематичним контекстом, який спрощує спілкування між гравцями. Наприклад, коли ми згадуємо екшн-ігри, маємо на увазі ігри з

вираженими елементами екшну; музичні ігри, у свою чергу, містять музичну тематику і так далі.

Елементи гри - це основні складові її світу, що взаємодіють з гравцем, тоді як тематичні елементи мають конкретну спрямованість. Наприклад, музика може стати тематичним елементом, якщо вона визначає атмосферу гри.

Елемент дії, що розглядається, є об'єктивним визначенням, спрямованим на функціональність, тоді як механіка дії включає в себе конкретну функцію для узгодження та забезпечення балансу мети гри, методів і правил. Це означає, що певні елементи гри можна перетворити на ігрову механіку, якщо надати їм відповідні функції, які дозволять їм взаємодіяти та координуватися з метою, методами та правилами гри.

Наприклад, обладнання в грі є елементом гри, але розробник гри може надати йому певну функцію, щоб збалансувати цілі, методи та правила гри, і елемент стане механікою предмета, таким чином генеруючи ігровий процес у взаємодія з гравцем.

Слід зазначити, що "фреймворк ігрової механіки" не створений для перейменування чи переформулювання різних типів ігор, але для більш раціонального зрозуміння типів ігор, механік гри, геймплею та тематичних аспектів. Його мета - об'єднання та створення нового контенту, базуючись на основній логіці гри, яка служить джерелом та рушійною силою інновацій у самій грі.

Класифікація ігрових категорій зазвичай ґрунтується на тематичних елементах, які акцентуються в грі. Це часто призводить до змішання загального опису ігрових категорій із геймплеєм та ігровою механікою. Наприклад, іноді в описах можна зустріти твердження, що "гра у пісочниці" є типом ігрового процесу, хоча насправді це є неузгодженістю визначень.

Механіка ігор у стилі "гра-пісочниця" складається з елементів «битви», «простору», «сценарію» та «рольової механіки», і часто акцентує увагу на "реальному часі" через механіку бою. Це стає очевидним у багатьох таких іграх, де простір представлений як вкладений у себе. Запровадження RPG або

механіки екшн-ігор додає такі елементи, як «екшн», «предмет», «комбо» та інші. Таким чином, комбінуючи геймплейні механіки, "гра-пісочниця" може стати "рольовою грою-пісочницею" або "екшн-грою-пісочницею".

Елементи в грі можуть мати різноманітні форми, а їх опис може охоплювати широкий спектр: будь-який об'єкт, який ми сприймаємо інтуїтивно, аналізуємо логічно та взаємодіємо з ним, може стати складовою частиною гри. Наприклад, основними механіками так званого "шутера від першої особи" є "сценарна, командна, предметна, дієва, поведінкова, бойова та відповідна механіка", а основним ігровим процесом є "битва", при цьому тематичними елементами є "від першої особи" та "стрільба".

Термін "елемент" більше відноситься до "тематичної складової". Ця "тематична складова" гри може впливати на "геймплей", одночасно беручи участь у поєднанні ігрової механіки та процесу гри.

Наприклад, "стрільба" може використовуватися як один із способів взаємодії гравця з грою і стати складовою частиною різноманітних геймплейних механік, таких як "рольова система" і "система навичок". При їх використанні гра може перетворитися на "рольову гру" з акцентом на бойовий аспект, де "стрільба" стає ключовим елементом геймплею, використовуючи механіку цього виду взаємодії.

Впровадження будь-якої нової ігрової механіки збагатить гравця новими враженнями, а комбінування різних ігрових елементів перетворить гру на зовсім новий досвід, а також відкриє нові можливості для гри.

Посилення можливостей елементів через використання відповідної та точної точки входу може сприяти створенню захоплюючої геймплейної механіки. У звичайних іграх з персонажами NPC гравець взаємодіє з ними через діалоги, кліки та вибір функцій, в той час як роль NPC в грі зазвичай полягає в допомозі гравцеві та розвитку сюжету гри.

Крім того, в іграх, які акцентуються на "механіці протистояння", NPC будуть використовуватися для підсилення викликів, що ставляться перед гравцем, тоді як у тих, що спрямовані на сюжет, оповідь та емоції, NPC будуть

мати вирішальне значення. Вони відіграють ключову роль як у спрощенні "механіки квесту", яка допомагає гравцям зануритися в сюжет, так і у "механіці персонажа", яка дозволяє гравцям спілкуватися з емоційними аспектами персонажів у грі.

Ці NPC були внесені до гри з власним сценарієм або наративом, але вони не перетворилися на частину ігрової механіки. За обмежень сучасних технологій NPC не можуть досягати основних цілей з такою самою великою вільністю, як інші персонажі гри. Однак, якщо ми дозволимо людям різних характерів та походження вільно спілкуватися між собою, як у реальному житті, і розвивати історію та встановлювати емоційні зв'язки, наскільки бажає гравець, то "елемент персонажа" для NPC може стати частиною ігрової механіки.

Деякі сучасні ігри внесли аспекти особистості до NPC, що може вплинути на гравця, але це не гармонізується з цілями, методами і правилами гри. Такий підхід не змінює основних механік гри. Розширення можливостей персонажів NPC, щоб вони могли впливати на цілі, правила та методи гри, є інновацією в ігровій механіці, що розробляється з нуля.

Розширенням особистості NPC є можливість їх реагувати на гравця за його слова або дії, наприклад, напад на нього за образливі слова або надання допомоги через загрози. Це вплине на правила та стиль гри, роблячи її більш відкритою та насиченою. Водночас цей новий аспект персонажів також може стати основою для нового типу геймплею, який можна назвати "рольовою грою з глибоким сюжетом".

Впровадження цієї системи буде складним та тривалим процесом. Необхідно не лише надати кожному персонажу новий рівень реалізму, щоб вони могли приймати автономні рішення відповідно до своїх характерів, але й забезпечити баланс з іншими ігровими механіками. Це означає, що цілі, методи та правила гри мають взаємодіяти краще, щоб підсилити враження від гри та перевершити очікування гравців.[20]

Багато ігрових розробників віддають перевагу певній свободі в плані внутрішньої логіки. Часом вони порушують правила фізики, біології або навіть людської поведінки, щоб гравці могли отримати максимум задоволення від гри. Ці порушення правил часто дозволяють розробникам керувати сюжетом або створювати виклики для гравців, не обмежуючись рамками загальної структури гри.

Незалежно від причини, можна зазначити одне: багато ігор мають незвичну логіку, яка може бути цікавою та розважальною або заплутаною та комічною.

Часом розробники ігор вносять зміни до правил, спрямовуючись на поліпшення зручності для гравців. Наприклад, у шутерах від першої особи гравцям можуть дозволяти отримати додаткові життя без смертельних наслідків. Після отримання поранення гравцям дають можливість швидко сховатися від ворогів та відновитися від травм за декілька секунд.

Також приємно, коли розробники дозволяють гравцям виконувати захоплюючі, вражаючі дії, щоб гравець міг швидко пересуватися по карті або відчувати короткий притік адреналіну.

Існують різні способи, якими розробники використовують корисні перерви в логіці гри, окрім можливості швидкого відновлення після травми або стрибків з висоти без ризику смерті. Деякі ігри пропонують зручні функції для гравців, такі як подвійні стрибки, які неможливі в реальному житті, але дозволяють легко пересуватися по карті. Є блоки, що парять у повітрі, ігноруючи гравітацію яка могла б притягнути їх вниз. Також є рюкзак для зберігання предметів, які гравець не потребує наразі. Вони можуть бути досить великими, але не відповідати об'єму внутрішніх розмірів.

Зрештою, відеоігри - це лише розвага і не завжди повинні бути логічними. Проте гравці відзначають непослідовність у порушенні логіки, навіть якщо вона виявиться корисною. Незважаючи на те, що природа цієї логіки в іграх може бути незрозумілою, це практика, яка буде тривати, доки існуватимуть відеоігри.[21]

## 2.2 Прототипування ігрового середовища

Алгоритми - це набори інструкцій, які допомагають у вирішенні завдань або проблем. У розробці ігор вони використовуються для створення складних візуальних елементів та механіки, які інакше були б недосяжними. Вони можуть бути застосовані до різноманітних завдань - від виявлення зіткнень та пошуку шляху до 3D-рендерінгу та штучного інтелекту. За допомогою потужності алгоритмів, розробники можуть створювати ігри з складною поведінкою та вражаючими візуальними ефектами. Сутність алгоритмів полягає в тому, що вони лише набір команд, які вказують комп'ютеру, що робити.

Один з популярних видів алгоритмів у галузі розробки ігор - це фізичні моделі, які наслідують реалістичну поведінку об'єктів, таку як гравітація, тертя та інерція. Ці механізми забезпечують автентичний рух та взаємодію між об'єктами у віртуальному світі гри. Крім того, алгоритми штучного інтелекту програмують неігрових персонажів (NPC) на виконання певних дій. Таким чином, штучний інтелект можна застосовувати у всіх сферах: від розпізнавання облич та обробки природної мови до симуляції автономного керування та розробки військових стратегій противника.

Розробка ігор також потребує використання алгоритмів для пошуку шляху, які допомагають NPC або керованим гравцем персонажам орієнтуватися в складних оточеннях. Ці алгоритми враховують перешкоди, такі як стіни або річки, коли обчислюють найкоротший шлях від пункту А до пункту Б.

Алгоритми 3D-рендерінгу є ключовими для генерації живих візуальних образів, які оживляють ігрові всесвіти. Ці алгоритми використовують математичні моделі, такі як променеве трасування та обчислення освітлення, для створення реалістичних 3D-зображень з тінями та відчуттям глибини.

Удача може відігравати важливу роль у створенні геймплею та дизайні гри. Багато сучасних ігор інтегрують елементи удачі в їхню механіку, це дає гравцям більшу можливість контролювати результати. У деяких іграх є прямі

елементи удачі, такі як кидки кубиків або розіграші карт, тоді як інші дають гравцям змогу стратегічно управляти випадковими подіями.

Елементи, що ґрунтуються на удачу, можуть спричиняти додатковий адреналін та виклик для гравців, проте надмірна удача може призвести до розчарування через непередбачувані результати. Іноді удача виступає як засіб балансування, вирівнюючи шанси між досвідченими та неопітними гравцями. Наприклад, онлайн-ігри з джекпотом та слоти значною мірою залежать від випадковості, оскільки їхні результати визначаються генераторами випадкових чисел (ГВЧ). У більшості країн ігри з реальними грошима є законними, лише у випадку, якщо їхні результати залежать від вміння гравця, а не від випадковості. Розробникам слід проконсультуватися з провідним юристом у справах ігор на реальні гроші, якщо їхні ігри приносять прибуток або цінні предмети.

Ці ігри мають високий рейтинг завдяки можливості отримати швидкий досвід та виграти величезні суми та джекпоти. Навіть якщо стратегія може підвищити шанси на перемогу, в кінці кінців результат залежить від удачі, що забезпечує рівні умови для всіх гравців без упередженості.

Розробники можуть покращити реалізм і достовірність персонажів у грі, впроваджуючи в них людські риси. Це сприяє зануренню гравців у історію і дозволяє краще розуміти мотивації персонажів, що забезпечує більш насичений та задовільний геймплей.

Впровадження гуманізованих елементів у гру може зменшити відстань між реальним світом та віртуальною реальністю. Завдяки реалістичним реакціям NPC на дії гравців, таким як діалог, що адаптується до їхніх вчинків або спілкування, ці персонажі виглядають інтелектуальними та відчутними, сприяючи співпереживанню з боку гравців. Це особливо важливо в іграх, які ставлять на передній план сюжет, оскільки вони можуть потребувати від гравців прийняття рішень, що мають довгостроковий вплив на життя інших персонажів або навіть на весь вигаданий світ.

Використання рандомізації в ігровій розробці - ключова концепція, що не лише привертає увагу гравців, але й додає елемент несподіваності. Цей метод може зробити гру більш захопливою, адже гравці ніколи не знають, що очікувати. Такий підхід уникає звикання гравців до певних схем чи стратегій, здобутих у попередніх іграх. Крім того, рандомізація дозволяє збагатити ігрове оточення, створюючи нові виклики, які залишаються актуальними й після тривалих ігрових сесій.

Занадто значна ступінь випадковості може негативно вплинути на структуру гри та розчарувати гравців. Якщо випадковість реалізована недосконало, це може спричинити несправедливі результати, які позбавлять гравців задоволення від гри.

Використання алгоритмів у розробці відеоігор є все більш популярною тенденцією, яка продовжить перетворювати галузь у майбутньому. Завдяки технологічному прогресу можна використовувати більш складні алгоритми для створення захоплюючих і вражаючих ігрових вражень. За відкритішого доступу до віртуальної реальності та технологій доповненої реальності розробники зможуть впроваджувати в свої ігри більш складні 3D-візуальні ефекти.

Алгоритми штучного інтелекту будуть ставати все потужнішими, що дозволить неігровим персонажам бути більш емоційно зв'язаними з гравцями. Крім того, прогрес у машинному навчанні дозволяє розробникам ігор створювати ігри, які адаптуються до індивідуального стилю гри кожного гравця.

Алгоритми виявляють закономірності у діях гравців і налаштовують гру відповідно до цього, створюючи унікальні виклики та перешкоди для кожного користувача. Цей індивідуалізований підхід робить гру більш захопливою, оскільки гравці постійно знаходяться в напруженій ситуації і не можуть передбачити, що відбудеться далі.[22]

Використання штучного інтелекту для процедурної генерації в іграх кардинально змінює спосіб розробки гри. Це означає, що алгоритми та

технології автоматично створюють ігровий вміст, такий як рівні, персонажі та квести, без необхідності в ручному проектуванні. Такий підхід відкриває нескінченні можливості та забезпечує унікальність кожного проходження гри.

Перевагою процедурної генерації, що базується на штучному інтелекті, є можливість ефективно створювати великі та детальні ігрові світи. Використовуючи алгоритми штучного інтелекту, розробники можуть автоматизувати процес створення ландшафтів, оточень та навіть цілих міст з реалістичними текстурами та складними деталями. Це не лише заощаджує час, але й підвищує занурення для гравців, даруючи їм просторові віртуальні світи, що здаються живими.

Процедурне створення на базі штучного інтелекту забезпечує гнучкий ігровий процес, що адаптується до індивідуальних вподобань гравців. Завдяки методам машинного навчання ігри можуть аналізувати поведінку гравців і відповідно налаштовувати такі параметри, як рівень складності чи характеристики ворогів. Ця динамічна настройка гарантує, що гравці постійно знаходяться в напруженій ситуації, не відчуваючи нудьги чи втоми.

Загалом, застосування процедурної генерації на базі штучного інтелекту у відеоіграх відкриває двері перед розробниками, дозволяючи створювати глибші імерсивні світи та забезпечувати індивідуалізований геймплей для гравців. З розвитком та вдосконаленням цієї технології можна очікувати ще більше інновацій у майбутніх іграх.

Штучний інтелект також змінює спосіб функціонування багатокористувацьких ігор, впроваджуючи інтелектуальні системи підбору партнерів. Замість того, щоб використовувати лише рейтинг гравців або випадкові критерії для утворення матчів, алгоритми штучного інтелекту аналізують різні аспекти, такі як стиль гри, історія навичок і навіть моделі поведінки, щоб забезпечити справедливі та збалансовані поєдинки. Це не лише покращує загальний геймплей, але й спонукає до постійного вдосконалення та створює здорову конкуренцію між гравцями.[23]

Створення захопливих та реалістичних віртуальних світів у галузі розробки ігор і симуляцій базується на поєднанні різноманітних алгоритмів та математичних концепцій. Ось декілька ключових алгоритмів та математичних принципів, які застосовуються в цій сфері:

— Ньютонівська фізика: для моделювання основних фізичних взаємодій, таких як гравітація, зіткнення та динаміка твердого тіла.
— Інтеграція Верлета: чисельний метод, який використовується для моделювання динаміки м’яких тіл, тканини та інших деформованих об’єктів.
— Трасування променів: техніка для моделювання зіткнень і взаємодії між променями (лініями) та об’єктами у віртуальному середовищі.
— Системи частинок: алгоритми для моделювання великої кількості частинок, які використовуються для таких ефектів, як вогонь, дим, вода та вибухи.
— Матриці трансформації: матриці, які використовуються для представлення поворотів, переміщень і масштабування 3D-об’єктів у світі гри.
— Перспективна проекція: математичний метод, який використовується для перетворення 3D-точок у 2D-екранні координати для візуалізації.
— Моделі освітлення: алгоритми для імітації ефектів освітлення та затінення 3D-об’єктів, включаючи навколишнє, розсіяне та дзеркальне освітлення.
— Відображення текстур: методи застосування текстур до 3D-моделей для підвищення реалістичності.
— Генерація та оптимізація сітки: алгоритми для створення та оптимізації геометрії 3D-сітки для ефективного рендерингу.
— Ієрархії обмежувальних об'ємів (BVH): техніка для прискорення виявлення зіткнень шляхом організації об’єктів в ієрархічну структуру даних.
— Знайти та обрізати (SAP): широкофазний алгоритм виявлення зіткнень, який використовується у фізичних механізмах.
— Алгоритм A* (A-зірка): широко використовуваний алгоритм пошуку найкоротшого шляху між двома точками в середовищі на основі сітки.

— Алгоритм Дейкстри: ще один алгоритм пошуку найкоротшого шляху в середовищі на основі графів.
— Скінченні автомати (FSM): модель, яка використовується для представлення поведінки ШІ з набором станів і переходів між ними.
— Дерева поведінки: ієрархічні структури, які використовуються для прийняття складних рішень ШІ та виконання завдань.
— Поведінка керування: алгоритми, які імітують природний рух і поведінку для автономних агентів (наприклад, зграя, слідування шляху).
— Шум Перлина: широко використовувана шумова функція для створення природного рельєфу та текстур.
— Клітинні автомати: алгоритми, що використовуються для генерації складних шаблонів і структур, часто використовуються в дизайні рівнів і генерації рельєфу.
— Теорія корисності: математичний підхід, який використовується для моделювання прийняття рішень агентами штучного інтелекту на основі різних факторів і переваг.
— Марковські процеси вирішування (MDP): структура для моделювання послідовних процесів прийняття рішень у невизначених середовищах.
— Архітектура клієнт-сервер: організація мережевих ігор і симуляцій із клієнтами, які взаємодіють із центральним сервером.
— Компенсація затримки: методи врахування затримки мережі в багатокористувацьких іграх для покращення синхронізації.

Це лише кілька прикладів алгоритмів та математичних методів, які використовуються у розробці ігор та симуляцій. Розробка ігор часто включає комбінування цих методів для створення захоплюючих та інтерактивних віртуальних вражень. Оскільки галузь розробки ігор продовжує розвиватися, нові алгоритми та математичні концепції неперервно досліджуються та впроваджуються для поліпшення реалістичності та інтерактивності у віртуальних світах.[24]

## 2.3 Дизайн гри

У сучасному світі мобільних ігор важлива роль належить візуальному мистецтву, яке приваблює аудиторію. Естетичний вигляд гри може визначати її успіх. Досвід розробників мобільних ігор та мистецьких студій, з якими вони співпрацюють, має величезне значення. Вони поєднують креативність і технічну майстерність, формуючи унікальну візуальну ідентичність мобільних ігор.

Розвиток мистецтва мобільних ігор переживав тривалий етап з самого свого витоку. Спочатку графіка в них була простішою, адже обмежені можливості телефонів не дозволяли запускати складніші візуальні ефекти. Проте ситуація дуже змінилася. Сучасні мобільні ігри мають неймовірно якісну графіку, завдяки вдосконаленню технологій у смартфонах. Деякі з них навіть можуть похвалитися візуальним виконанням, що перевищує рівень графіки у іграх для великих ігрових консолей. Це значне покращення, насамперед, завдяки двом чинникам: прогресу в технологіях мобільних пристроїв та креативному підходу розробників мобільних ігор до створення ігрових візуальних ефектів.

Ці компанії розширюють можливості графіки в мобільних іграх, не лише створюючи вигляд якісних проектів, але й забезпечуючи їх безперебійну роботу на широкому спектрі мобільних пристроїв. Це означає, що більше користувачів зможуть насолоджуватися грою, незалежно від моделі телефону. Дивовижно, наскільки далеко просунулося мистецтво мобільних ігор, перетворившись зі скромних дизайнів на захоплюючі, деталізовані візуальні враження, що роблять гру на вашому телефоні справді захоплюючим досвідом.

Тенденції у візуальному оформленні мобільних ігор постійно змінюються, віддзеркалюючи як технологічні досягнення, так і зміну вподобань гравців. В цьому контексті варто відзначити деякі ключові напрямки, які визначають зовнішній вигляд сучасних мобільних ігор:

— Графіка високої чіткості: Завдяки зростанню потужності смартфонів мобільні ігри тепер можуть запропонувати високоякісну графіку, що забезпечує насичений візуальний досвід. Рівень чіткості та деталізації у таких іграх сягає показників, подібних до ігор для ПК та ігрових консолей, що робить їх привабливими для різних гравців.

— Реалістична 3D графіка: Набуває популярності тривимірна графіка, що відтворює реальний світ, щоб забезпечити захопливий ігровий досвід. Це охоплює докладні моделі персонажів, живе освітлення і реалістичне оточення.

— Мінімалістичний і плоский дизайн: Багато ігор використовують мінімалістичний підхід, віддавши перевагу плоскому дизайну та простим кольоровим палітрам. Це дозволяє їм виглядати сучасно та стильно, а також гарантує, що вони працюватимуть плавно на більшій кількості пристроїв, зменшуючи споживання ресурсів.

— Ретро та піксельне мистецтво: У мобільних іграх велике значення приділяється почуттю ностальгії. Чимало розробників обирають створювати гри з ретро-графікою та піксельним мистецтвом, що призводить до згадок про перші геймінгові дні. Цей стиль має свій унікальний шарм, який приваблює як досвідчених гравців, так і новачків.

— Яскраві та сміливі кольорові схеми: Часто в мобільних іграх використовуються яскраві та насичені кольори для того, щоб привернути увагу гравців та створити візуально привабливий досвід. Такі кольорові схеми особливо ефективні в казуальних і гіпер-казуальних іграх, вирізняючи їх серед інших ігор.

— Графіка доповненої реальності (AR): З останнім зростанням доступності технології доповненої реальності, мобільні ігри використовують AR для злиття віртуальних об'єктів з реальністю, що створює унікальний інтерактивний ігровий враження.

— Стилі мальованого мистецтва: Деякі ігри відмовляються від цифрової графіки на користь ручної художньої стилізації. Це дає їм унікальний та художній шар, який робить їх відзначними серед заповненого ринку.
— Динамічне та інтерактивне середовище: Сучасні ігри включають різноманітні середовища, з якими гравці можуть взаємодіяти за допомогою різних способів. Це означає наявність руйнівних елементів, змінні погодні умови та зміни циклів дня і ночі, що забезпечує грі реалістичність та захопливість. Ці тенденції у візуальному представленні мобільних ігор не лише покращують враження гравця, але й відображають творчі можливості в розробці таких ігор для мобільних пристроїв. З огляду на постійний розвиток технологій, ми можемо очікувати ще більше інноваційних та вражаючих ігор на ринку мобільних додатків.

Розвиток мобільних ігор зробив значний крок вперед, застосовуючи різноманітні захоплюючі методи, щоб забезпечити привабливий вигляд гри. Ось деякі ключові підходи, які вони використовують:

— Розширена анімація: Ігри наразі використовують вдосконалену анімацію, щоб персонажі та ігровий світ мали більш природний вигляд, включаючи плавність та реалістичність рухів.
— Розумне використання текстур: Так як мобільні телефони не в змозі обробляти велику кількість даних, розробники використовують менш складні графічні ефекти у певних аспектах гри. Це сприяє покращенню її продуктивності без великого навантаження на пам'ять.
— Системи частинок: Вони застосовуються для створення вражаючих ефектів, таких як полум'я, дим або блиск, що додає грі вражаючого вигляду без збільшення часу обробки.
— Шейдери та освітлення: Це візуальні ефекти, призначені для покращення освітлення та текстур у грі, які роблять предмети більш реалістичними, наприклад, відтворюють реалістично воду чи метал.[15]

У мобільних іграх, так само, як і в іграх для ПК та консолей, приваблива графіка та анімація є важливими факторами для привернення уваги гравців.

Від початкового проектування до фінальної реалізації та досягнення високої візуальної якості, якість графіки та анімації в мобільних іграх часто визначає, чи приверне вашу аудиторію ваша гра чи ні.

Створення високоякісної графіки та анімації для мобільних ігор має свої відмінності від тих, що призначені для консолей і комп’ютерів, проте загальні цілі та підходи залишаються суттєво незмінними. У процесі розробки естетично привабливих мобільних ігор важливо пам’ятати про ключові аспекти, пов'язані з мобільними пристроями.

Перш ніж глибоко вдаватися у розробку візуальних ефектів та анімації для мобільної гри, потрібно зробити важливі вирішальні кроки, що визначатимуть ключові аспекти. По-перше, необхідно визначити, який саме вид гри ви маєте на меті створити. Залежно від жанру та тематики, візуальний стиль може значно відрізнятися. Наприклад, якщо ви плануєте створити військову гру, яка демонструє жахи війни, може бути логічним обрати гіперреалістичні візуальні ефекти. Натомість, у випадку легкої гри, призначеної для маленьких дітей, ймовірно, краще підійдуть стилізовані візуальні елементи.

Цей процес прийняття рішень також впливає на анімацію вашої мобільної гри. Якщо гра має реалістичні візуальні ефекти, то вона повинна мати відповідну анімацію, щоб зберегти цілісність і зберегти почуття занурення гравців. Мобільні ігри зі стилізованими візуальними елементами можуть користуватися різноманітністю стилів анімації. Залежно від загального візуального напрямку, стилізовані мобільні ігри можуть підвищити комедійну цінність шляхом використання перебільшених або непередбачуваних анімацій.

Мобільні пристрої не так потужні, як ПК та ігрові консолі. Навіть з урахуванням значного зростання їхньої продуктивності протягом останніх років, порівняно з іншими платформами вони залишаються обмеженими. Отже, важливо оптимізувати мобільні ігри з урахуванням цього факту та різноманітних характеристик мобільних пристроїв, що присутні на ринку.

Навіть якщо мобільні пристрої мають меншу потужність, ніж комп’ютери та консолі, їх якість зображення варіюється від моделі до моделі. Це означає, що для досягнення оптимальної якості графіки та анімації у мобільній грі потрібні індивідуальні тести та оптимізація для кожного пристрою, щоб забезпечити плавну роботу без втрати якості.

Більшість мобільних пристроїв мають різні розміри екранів та роздільність залежно від типу. Планшети, як правило, мають більші екрани, ніж смартфони, і роздільність може відрізнятися залежно від їхньої ціни та новизни. Для гарантії якісної роботи та уникнення візуальних дефектів, перед запуском мобільної гри важливо уважно перевірити розміри та роздільність екрана.

Для того щоб залучити увагу аудиторії та забезпечити високий рівень занурення, важливо, щоб візуальний дизайн став основним пріоритетом для ігор незалежно від платформи, особливо це стосується мобільних ігор. Навіть якщо мобільні пристрої не мають потужності для відтворення графіки на рівні консолей або ПК, цей недолік можна компенсувати якісним та незабутнім візуальним дизайном.

Ця концепція також може бути застосована до анімації мобільних ігор. Малоймовірно, що розробник мобільних ігор зможе досягти гіперреалістичної анімації, яку можна бачити в консольних і комп’ютерних іграх AAA. Таким чином, приділяючи значну увагу дизайну, ваші анімації можуть бути привабливими, навіть не досягаючи високого рівня візуальної якості. Експериментування з основними принципами анімації може призвести до переконливих результатів. Збільшення інтервалу між ключовими кадрами, використання передбачень та акцентування розташування персонажів і поз можуть відмінно оживити анімацію вашої мобільної гри.

Значні вкладення в дизайн також є важливими для мобільних ігор. Оскільки інтерфейс у таких іграх оперує на їхніх власних сенсорних екранах, потрібно спеціальне проектування. На відміну від консолей та ПК, у мобільних іграх значно менше простору на екрані, тому ефективне використання цього

простору має велике значення. Також, взаємодія з елементами інтерфейсу відбувається безпосередньо, без використання контролерів чи миші та клавіатури. Елементи інтерфейсу мають бути простими й зрозумілими, аби максимально ефективно використовувати простір на екрані.[16]

Створення графіки відеоігор розпочинається з операції затінення вершин, яка визначає тривимірне розташування об'єктів, що потім відображаються на екрані. Це ключовий момент, бо від нього залежить видимість різних частин ігрового світу для гравця та необхідність їх візуалізації.

Після затінення вершин настає етап растеризації, коли тривимірні моделі перетворюються на двовимірні трикутники, що відповідають розташуванню пікселів екрана. Ці пікселі потім заповнюються кольором і текстурою, утворюючи зображення, яке бачить гравець під час гри.

Покращення візуальних ефектів досягається завдяки операції затінення фрагментів. Цей етап налаштовує кольори пікселів, щоб імітувати, як світло впливає на реальні поверхні, додаючи глибину та реалістичність. Він показує, як світло взаємодіє з поверхнями, створюючи тіні та відблиски, що роблять ігрове середовище більш живим.

Проте ці ретельні розрахунки потребують значної обчислювальної потужності, і саме тут на допомогу приходять графічні процесори (GPU). Графічні процесори - це апаратні компоненти, спеціально розроблені для виконання складних завдань рендерингу, що забезпечують плавну та вражаючу візуально гру.

Навіть при наявності потужних графічних процесорів можуть виникати проблеми, такі як накладання об'єктів, що ускладнює визначення порядку їх розташування. Технологія глибинної буферизації вирішує цю проблему, відстежуючи глибину об'єктів і гарантуючи, що на кожному пікселі відображається тільки найближчий об'єкт. Для поліпшення якості зображення існує техніка, відома як згладжування. Цей метод прибирає нерівності по

краях, що можуть виникати при наявності діагональних або кривих ліній у грі, забезпечуючи більш чистий і реалістичний вигляд зображень.

У відеоіграх часто відбуваються динамічні зміни сцен, що вимагають швидкого перерахунку рендерингу. Це необхідно, щоб ігровий світ миттєво реагував на дії гравця. Правильне моделювання світла й тіні відіграє ключову роль у створенні реалістичного вигляду об'єктів. Освітлення та затінення в грі допомагають створити середовище, що максимально наближене до реального світу.[17]

Можна з упевненістю стверджувати, що якість графіки на мобільних пристроях є ключовим фактором успіху будь-якої ігрової програми. Нині вона відіграє важливу роль не лише як привабливий аспект, а й у залученні та утриманні користувачів. Однак, важливо пам'ятати, що потужна графіка може стати проблемою для різних пристроїв. Вона може сповільнити гру і знизити задоволення від неї для користувачів. Тут важливо вміло застосовувати методи оптимізації.

Параметри візуалізації безпосередньо впливають на продуктивність мобільних пристроїв, що забезпечує безперервну та ефективну роботу вашої гри на різних пристроях. Нижче наведено способи розуміння впливу параметрів візуалізації на продуктивність мобільних пристроїв:

— Вибір правильних графічних ресурсів: Розумно обирайте графічні ресурси. Використання текстур високої якості та складних 3D-моделей може збільшити навантаження на мобільний пристрій. Оптімізуйте вибір активів, намагаючись обирати простіші варіанти, коли це можливо.

— Монітор частоти кадрів: Скористайтеся засобами моніторингу частоти кадрів, щоб спостерігати за продуктивністю гри на різних пристроях. Намагайтеся досягти стабільної частоти кадрів, зазвичай 30 або 60 кадрів на секунду (FPS), в залежності від вимог вашої гри.

— Тест на реальних пристроях: Емулятори корисні, але справжні пристрої забезпечують більш точну інформацію про продуктивність. Проведіть тестування вашої гри на різних пристроях, щоб виявити потенційні

проблеми з продуктивністю, які можуть бути характерні для кожного з них.

Обирайте графічні ресурси для вашої мобільної гри розумно, забезпечуючи оптимальний баланс між гарним виглядом та продуктивністю, щоб забезпечити максимальне задоволення гравців на будь-яких пристроях.

Нижче наведено способи вибрати правильні графічні ресурси для своєї мобільної гри:

— Пріоритет ефективності: Оберіть графічний стиль, що забезпечить гармонію між візуальними характеристиками та ефективністю. Розгляньте можливість використання текстур з меншою роздільною здатністю та простіших моделей для зменшення навантаження на ресурси.

— Стиснення текстури: Використовуйте методи стиснення текстур, такі як ETC2 або ASTC, для зменшення обсягу вашого текстурного матеріалу, при цьому зберігаючи задовільну візуальну якість.

— Оптимальні формати файлів: Оберіть файли у форматах PNG або JPEG для 2D-ресурсів і моделей у форматах FBX або OBJ, які будуть сумісні з вашим ігровим двигуном та мобільними платформами.

— Управління Mipmap: Використовуйте рівні mipmap для текстур, щоб оптимізувати відображення на різних відстанях. Це може суттєво підвищити продуктивність, особливо в 3D-іграх.

— Аркуші спрайтів для 2D-ігор: Якщо ви створюєте 2D-гру, рекомендується використовувати таблиці спрайтів для об'єднання кількох спрайтів у єдине зображення. Це допомагає зменшити навантаження на процесор під час малювання та підвищує ефективність візуалізації.[18]

Графічні двовимірні ігри використовують плоску графіку, таку як спрайти, плитки або векторні зображення, для створення своїх ігрових елементів. Вони обмежені плоским простором без глибини та перспективи. На відміну від цього, тривимірні ігри використовують тривимірну графіку, таку як моделі, багатокутники та текстури, щоб створити персонажів, оточення та ресурси, що передають тривимірний простір реального світу. 3D-моделі мають

глибину та перспективу, що створює захоплюючий та інтерактивний ігровий досвід.

Як 2D, так і 3D використовуються для створення ігор різних жанрів і мають різні стилі мистецтва. У 2D це можуть бути векторне мистецтво, піксельне мистецтво або мистецтво з вирізами. У 3D популярні стилі включають реалістичне мистецтво, мистецтво фентезі-реалізму та низькополігональне мистецтво, які використовуються у розробці відеоігор.

У випадку з 2D-іграми, рухові можливості обмежені. Гравці можуть рухатися вперед, назад, вгору, вниз, ліворуч і праворуч. Однак у тривимірних іграх сценарії більш динамічні, дозволяючи виконувати складніші рухи, такі як обертання, що неможливо втілити у 2D-графіці. Створення та синхронізація таких рухів у тривимірних іграх вимагає більше ресурсів і займає більше часу.

У зв'язку з плоскою структурою та обмеженими рухами, у 2D-іграх простіше геймплейна механіка. Тоді як у 3D-іграх геймплей стає складнішим через більш динамічні рухи та графічні можливості. Отже, керування рухами у 2D-іграх вважається легшим, ніж у 3D-іграх.

Проте вибір між ними залежить від ваших потреб і типу гри, яку ви прагнете створити. Якщо ви маєте на меті створити більш інтерактивний ігровий досвід, тоді 3D — кращий вибір. Але якщо ваші цілі включають у себе економію, легкість та меншу складність, то 2D може бути кращим варіантом.

У 2D-іграх застосовується ортографічна камера, яка зберігає постійний розмір зображення незалежно від відстані до камери. Це означає, що вони можуть мати різні види огляду, такі як боковий прокрут, вид зверху або ізометричний, який створює відчуття глибини. У 2D іграх гравцям не потрібно керувати камерою, що робить гру більш доступною. Платформери, головоломки та бойові ігри є деякими з найпопулярніших жанрів у 2D.

У 3D-іграх, натомість, використовуються плечова та перспективна камери для стеження за рухом гравця. Перспективна камера може мати кілька видів огляду, включаючи фіксовану, від першої та від третьої особи. В

залежності від контексту, наприклад, стрільби чи ближнього бою, використовують різні типи камер для кращого досвіду гравця.

Як і 2D, так і 3D-ігри користуються різними засобами для створення ігрових активів. Adobe Photoshop, Adobe Illustrator, Clip Studio Paint, GIMP, Aseprite та інші програми використовуються для розробки 2D-графіки, тоді як для створення 3D-ігрових активів застосовуються різноманітні інструменти, такі як Autodesk Maya, Blender, 3ds Max, Substance Painter і інші.

Вибір ігрового движка також важливий, оскільки він визначає доступні можливості для розробки. Художники обирають той движок, який надає найбільше функцій для створення захоплюючого мистецтва як у форматі 2D, так і у форматі 3D.

У процесі тестування використовуються різноманітні підходи для 2D та 3D ігор. Для 2D гри проводиться перевірка спрайтів, роздільної здатності, зіткнень, інтерфейсу користувача та ефекту паралаксу. У той час як для 3D гри проводяться тести моделей та сіток, анімації, освітлення та тіней, фізики, зіткнень і камери.

Деякі аспекти тестування, такі як узгодженість художнього стилю, локалізація, інтерфейс користувача, та доступність, є важливими як для 2D, так і для 3D ігор.

Загалом, ігри у форматі 2D та 3D мають свої відмінності у таких аспектах, як рух, кут огляду, якість графіки, інструменти розробки та типи тестування. Вибір між ними залежить від ваших власних потреб. Якщо ваша мета - створити просту, економічну та менш складну гру, 2D може бути оптимальним варіантом. Проте, якщо вам необхідна висока якість графіки та інтерактивний геймплей, 3D буде кращим вибором.[19]

Розвиток мобільних ігор - свідчення творчості та високого технічного рівня у мобільній галузі розробки. Співпраця між компаніями, що створюють мобільні ігри, та аутсорсинговими студіями відображається в постійному розширенні візуальних можливостей і технік гри, що перетворює мобільні ігри на мистецтво, яке встановлює нові стандарти візуальної якості.

# 3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

## 3.1 Вибір середовища розробки та мови програмування

Ігровий рушій (англ. Game engine) - це середовище розробки програмного забезпечення, спеціально призначене для створення відеоігор. Головна мета ігрового рушія полягає в забезпеченні розробників готовим набором інструментів і функцій, які спрощують процес створення гри. Це дозволяє розробникам зосередитися на творчому процесі, не вдаючись у деталі складної реалізації. Ігровий рушій надає високий рівень абстракції, що робить багато завдань легкими для виконання.

Ігровий рушій створено для розробки ігор, подібно до будь-якої іншої інтегрованої середи розробки (IDE), призначеної для конкретної мови програмування. Всі компоненти ігрового рушія розроблені та інтегровані з огляду на вимоги ігрової розробки, і включають в себе наступні аспекти:

— Вхідні дані: Ігровий рушій підтримує різноманітні пристрої введення і використовує для цього два основних методи обробки: events та polling. Метод events передбачає реєстрацію подій введення, таких як клацання правою кнопкою миші або натискання клавіші зі стрілкою вгору, і виконання спеціального коду на їхній основі. Метод polling використовується для отримання значень конкретних параметрів, таких як координати позиції вказівника миші або кут нахилу смартфона.

— Графіка: Для створення 3D-графіки в іграх, спершу створюються 3D-активи за допомогою зовнішніх спеціалізованих програм для 3D-рендерінгу, таких як Maya, Blender та інші. Потім ці активи імпортуються в ігровий рушій. Отже, ігровий рушій повинен підтримувати різні формати імпорту 3D-ресурсів. Ігровий рушій також надає різноманітні функції, такі як світлові ефекти, генерація тіней, картування рельєфу, змішування анімації та інші, щоб зробити імпортовані ресурси максимально реалістичними та відповідними задачам графічного оформлення гри.

— Звук: Аудіо в ігровому середовищі є важливою частиною механізму і використовується для управління звуковими ефектами. Для цього

використовуються програмні інтерфейси (API), такі як OpenAL, SDL Audio, XAudio 2 та Web Audio, які надають абстракцію для взаємодії з аудіопроцесором.

— Фізика: Фізичний рушій - це програмне забезпечення, призначене для точного моделювання різних фізичних систем, які існують у реальному світі. Вони є складними компонентами, які інтегруються в сучасні ігрові рушії, що в основному використовуються у відеоіграх. Фізичний рушій відповідає за обробку різних аспектів, таких як гравітація, виявлення зіткнень, переміщення об'єктів та інші подібні параметри в грі.

— Штучний інтелект: В сучасній розробці відеоігор штучний інтелект грає важливу роль. Завдяки спеціальному вбудованому програмному забезпеченню можна визначити, яку зброю гравець буде використовувати, враховуючи конкретну ситуацію та запис його поведінки та дій. Зазвичай інтеграція штучного інтелекту в іграх здійснюється за допомогою готових сценаріїв.[11]

На сьогоднішній день існує достатньо багато готових програмних рішень, і навіть великі ігрові компанії використовують їх, замість створення власного ігрового рушія.

Unity — це ігровий рушій для розробки ігор у 2D і 3D, який функціонує з 2005 року. Цей рушій розроблений компанією Unity Technologies з метою забезпечити розробників доступом до інструментів для створення ігор, що було інноваційним у свій час. Протягом свого тривалого існування рушій постійно розширювався та вдосконалювався, щоб відповідати останнім тенденціям і технологічним досягненням.

Сьогодні головна мета ігрового рушія - надати надійний набір інструментів для індустрії розробки ігор та максимально спростити його використання. Крім того, розробники ігрового рушія розширили свою діяльність в інших сферах, зосереджуючись на розробці 3D-контенту в реальному часі, що зробило його одним з найпотужніших ігрових рушіїв на ринку.

Unity підтримує як 3D, так і 2D графіку, що дає можливість вибирати художній стиль для вашого проекту на свій розсуд. Кожен тип графіки має свій унікальний набір спеціалізованих інструментів і навіть володіє власними API-інтерфейсами сценаріїв, які можна використовувати для налаштування різних фізичних параметрів, що ідеально підходять для кожного стилю.

Тризначна графіка в Unity пропонує надзвичайно надійний інструментарій. Ви можете створювати власні матеріали, легко налаштовувати освітлення, застосовувати ефекти постобробки та розробляти шейдери з використанням Shader Graph. Без винятку, незалежно від того, чи це генерація 3D-рельєфу чи створення 2D-мозаїчних карт, Unity надає вам повний набір інструментів для роботи з будь-яким типом графіки та досягнення бажаного результату.

Unity надає чіткий підхід до створення архітектури гри. У проекті гри кожен «рівень» розділяється на сцени, і кожна сцена включає в себе всі ігрові об'єкти, необхідні гравцеві для взаємодії з рівнем - це може бути фон, головний персонаж, вороги, кулі або інші елементи гри.

Unity постачається з потужним API сценаріїв на C#, який надає швидкий доступ до найбільш важливих функцій. Серед них не лише загальні функції гри, але і конкретні виклики API, що дозволяють отримувати доступ до конкретних функцій та особливостей рушія.

Ігри створені в Unity можуть бути легко адаптовані на різноманітні платформи. Після завантаження відповідного набору інструментів розробник може експортувати свою гру для таких платформ як Android, iOS, Windows, MacOS, Linux, PS4, Xbox One та інші.

Unity Asset Store є одним з головних плюсів Unity. Це інтернет-магазин, де ви можете знайти різноманітні активи, як платні, так і безкоштовні, які можна використовувати у будь-якому проекті. Хоча деякі з цих активів розроблені самою Unity, багато із них створено спільнотою користувачів, що дає вам широкий вибір можливостей для вашого проекту.

Unity безкоштовно надає багато власних пакетів і ресурсів, які розширюють можливості рушія. Наприклад, ресурс Bolt дозволяє вам впроваджувати візуальні сценарії в середовище Unity.

Unity можна використовувати абсолютно безкоштовно. Навіть версія Personal дозволяє вам продавати ігри, створені з її використанням, без витрат. Лише при доході від гри на рівні 100 000$ потрібно буде розглядати дорожчі варіанти ліцензій, такі як Plus, Pro і Enterprise, які коштують від €1,877 до €4,554 на рік.[12]

Unreal Engine 4 - це відомий та широко застосовуваний ігровий рушій, розроблений компанією Epic Games. Перша версія Unreal Engine була випущена в 1998 році, за нею слідував Unreal Engine 2 у 2002 році, Unreal Engine 3 у 2006 році, і наразі актуальною є версія Unreal Engine 4, представлена в 2014 році. Цей рушій надає можливість розробки ігор для різних платформ , включаючи ПК і консолі, такі як PS4, Xbox One і Nintendo Switch. Ця гнучкість в роботі з різними платформами частково пояснює широке поширення Unreal Engine 4.

Більш досвідчені розробники мають можливість використовувати мову програмування C++ для розробки власних сценаріїв, які виконуються у межах ігрового рушія. У той час як для менш досвідчених розробників доступні потужні Blueprints, що суттєво спрощують процес, бо фактично є готовими блоками коду, які можна легко додавати до своїх об'єктів для взаємодії. Утім, Unreal менш підходить для створення невеликих або мобільних проектів, хоча він підтримує мобільні ігрові платформи. Цей рушій в основному спрямований на проекти з великою командою спеціалізованих розробників і не є оптимальним вибором для початківців.

Unreal Engine 4 використовує широко визнаний робочий процес PBR для створення матеріалів та візуалізації. У поєднанні з динамічними або затіненими тінями та освітленням, цей підхід дозволяє створювати фотореалістичний контент, який все ще здатний працювати в режимі реального часу.

Unreal Engine 4 включає в себе відмінний вбудований фізичний рушій, який підтримує використання фізики м'яких тіл, ефектів частинок і навіть таких простих речей як гравітація.

Unreal Engine 4 також надає інструменти для швидкого створення власної місцевості. Ви зможете легко створювати повні сцени на відкритому повітрі за декілька хвилин, і ці сцени будуть вже оптимізовані для роботи у вашій грі, якщо ви використовуєте вбудовані рослинні сітки, які надаються з Unreal Engine.

Unreal Engine 4 доступний для користування абсолютно безкоштовно. Незалежно від вашого статусу, чи ви розробник-початківець, чи велика AAA студія, вам не потрібно сплачувати жодного вступного внеску за користування Unreal. Замість цього, Epic Games встановлює угоду про роялті, відраховуючи лише 5% від всього доходу, який ви заробите від гри, якщо цей дохід перевищує 3000$ за квартал. Це єдиний спосіб для Epic Games отримувати прибуток від вашого проекту.[13]

Серед різноманітних мов програмування особливе визнання отримують мови високого рівня, такі як C++ і C#. Ці мови є основними компонентами в найпопулярніших ігрових рушіях завдяки своїй високій оптимізації та багатому функціоналу.

Якщо розглядати мови програмування C# і C++ на дуже базовому рівні, то вони мають подібний код. Проте C# є значно новішою мовою в галузі ігрової розробки. Вона була представлена компанією Microsoft у 2000 році як конкурент Java та входить до екосистеми ASP.NET. У свою чергу, C++ є базовою мовою для багатьох інших мов програмування і була представлена Б'ярном Страуструпом ще в 1980-х роках як «C з класами».

Як C++, так і C# є об'єктно-орієнтованими мовами програмування, проте C++ вважається більш складною мовою для роботи. Обидві мови можуть бути використані для розробки веб-додатків та настільних додатків, але на сьогоднішній день C# є більш популярним вибором для обох видів додатків.

Головна відмінність полягає у додатковому етапі компіляції, який необхідний C# перед перетворенням у машинний код. C++ сприймається рідною мовою, оскільки він компілюється безпосередньо в машинний код. У випадку C# спочатку код компілюється в Microsoft Intermediate Language (MSIL), а потім Just-In-Time (JIT) компілятор перетворює його в машинний код. Це призводить до того, що C++ зазвичай працює швидше, ніж C#.

Багато чинників впливають на продуктивність програми, не обмежуючись лише вродженою продуктивністю основної мови програмування. Чинники такі, як якість написаного коду, використання конкретного фреймворку та функціональність програми, мають значно більше значення, ніж вроджена швидкість C++ у питаннях продуктивності.

Загальне практичне правило вказує на те, що для розробки мобільних додатків частіше використовують мови вищого рівня, такі як C#. З іншого боку, C++ є більш універсальною мовою з точки зору платформ і цільових додатків, але має обмежений попит серед розробників мобільних додатків.[14]

Після аналізу мов програмування та ігрових рушіїв для розробки мобільних ігор, було прийнято рішення віддати перевагу мові програмування C# та ігровому рушію Unity, відмовившись від варіанта C++ та Unreal Engine 4, а також інших альтернатив. Ось декілька ключових причин такого вибору:

— Рушій: Unity - це популярний та потужний ігровий рушій, який відкриває широкі можливості для створення графіки, фізики, анімацій та звукового супроводу. Також він має велику кількість доступних плагінів та активів.
— Мова програмування: C# є мовою програмування вищого рівня з простим синтаксисом і багатою базою ресурсів для навчання. Це сприяє швидкому старту розробки гри та зменшує час, витрачений на програмування.
— Багатоплатформність: Якщо вам потрібно створити мобільну гру для різних платформ, то вибір C# та Unity дозволить легко перенести гру на різні мобільні платформи, такі як Android та iOS.

З урахуванням цих факторів, вибір мови програмування C# та ігрового рушія Unity є доцільним для розробки мобільної гри.

## 3.2 Короткий опис програмної реалізації

MonoBehaviour є основним класом, від якого успадковуються багато сценаріїв у Unity. MonoBehaviours надає функції життєвого циклу, що спрощують розробку в Unity.

MonoBehaviours завжди існують як компоненти GameObject і можуть бути створені за допомогою GameObject.AddComponent. Якщо об’єкт має існувати незалежно від GameObject, його слід успадковувати від ScriptableObject.

MonoBehaviour можна видалити за допомогою Object.Destroy або Object.DestroyImmediate. Коли GameObject знищується, усі його компоненти, включаючи MonoBehaviours, також видаляються.

Після видалення базового компонента об'єкт C# для MonoBehaviour залишається в пам'яті до моменту збору сміття. У такому стані MonoBehaviour поводиться так, ніби він нульовий, наприклад, повертає true для перевірки "obj == null". Проте цей клас не підтримує нульовий умовний оператор ( ?. ) і нульовий оператор об’єднання ( ?? ).

При серіалізації MonoBehaviour значення полів C# включаються відповідно до правил серіалізації Unity. Серіалізовані дані також містять внутрішні властивості, як-от посилання на MonoScript, що відстежує клас реалізації для об'єкта.[27]

Функція Start викликається у кадрі, коли сценарій активований безпосередньо перед першим викликом будь-якого методу оновлення.

Подібно до функції Awake, Start викликається лише один раз протягом життєвого циклу сценарію. Однак Awake викликається під час ініціалізації об’єкта сценарію, незалежно від того, активований сценарій чи ні. Start не може бути викликаний у тому ж кадрі, що й Awake, якщо сценарій не активований під час ініціалізації.

Функція Awake викликається для всіх об’єктів у Scene перед викликом функції Start будь-якого об’єкта. Це корисно у випадках, коли ініціалізаційний

код об’єкта А залежить від об’єкта В, який уже проініціалізований; ініціалізацію В слід виконувати в Awake, а А – у Start.

Якщо екземпляри об’єктів створюються під час гри, їх функція Awake викликається після завершення функцій Start об’єктів Scene.

Функцію Start можна визначити як Coroutine, що дозволяє призупиняти її виконання.[28]

Update викликається для кожного кадру, якщо MonoBehaviour увімкнено. Update є найбільш поширеною функцією для реалізації будь-якого ігрового сценарію. Однак не кожному MonoBehaviour сценарію потрібен Update.[29]

Незалежне від частоти кадрів повідомлення MonoBehaviour.FixedUpdate використовується для фізичних розрахунків.

MonoBehaviour.FixedUpdate викликається з фіксованою частотою фізичної системи, тобто кожен кадр із постійним інтервалом часу. Обчислення системи Physics виконуються після кожного виклику FixedUpdate. За замовчуванням цей інтервал становить 0,02 секунди (50 викликів на секунду). Щоб отримати це значення, скористайтеся Time.fixedDeltaTime. Ви можете змінити його, встановивши бажане значення в сценарії, або перейшовши до Edit > Settings > Time > Fixed Timestep.

Частота викликів FixedUpdate може відрізнятися від частоти викликів Update. Наприклад, якщо програма працює зі швидкістю 25 кадрів за секунду (fps), Unity викликає FixedUpdate приблизно двічі за кадр. При 100 кадрах за секунду FixedUpdate викликається приблизно на кожні два кадри. Контролюйте необхідну частоту кадрів та частоту Fixed Timestep за допомогою налаштувань Time. Для встановлення частоти кадрів використовуйте Application.targetFrameRate.

Застосовуйте FixedUpdate при роботі з Rigidbody. Якщо ви встановлюєте силу для Rigidbody, вона застосовується на кожен фіксований кадр. FixedUpdate відбувається у визначений інтервал часу, який зазвичай не збігається з MonoBehaviour.Update.[30]

LateUpdate викликається кожного кадру, якщо Behaviour увімкнено.

LateUpdate запускається після всіх інших функцій Update. Це зручно для впорядкування виконання сценарію. Наприклад, камера стеження завжди повинна реалізовуватися в LateUpdate, оскільки вона відслідковує об’єкти, які могли переміститися під час Update.[31]

ScriptableObject — це контейнер для зберігання даних, який можна використовувати для збереження великих обсягів інформації незалежно від екземплярів класу. Одним з основних його переваг є зменшення використання пам’яті в проекті шляхом уникнення дублювання значень. Це особливо корисно, якщо у вашому проекті є префаби, які зберігають незмінні дані в прикріплених скриптах MonoBehaviour.

Зазвичай, створюючи префаб, він отримує власну копію даних. Щоб уникнути цього, можна використовувати ScriptableObject для зберігання даних і надавати до них доступ усім префабам через посилання. Це означає, що в пам’яті буде лише одна копія даних.

Подібно до MonoBehaviour, ScriptableObjects походять від базового об’єкта Unity, але на відміну від MonoBehaviour, їх не можна прикріплювати до GameObject. Натомість їх зберігають як ресурси у проекті.

У редакторі дані можуть зберігатися в ScriptableObjects як під час редагування, так і під час виконання, оскільки ScriptableObjects використовують простір імен та скрипти редактора. Проте, у розгорнутій збірці не можна використовувати ScriptableObjects для збереження даних, але можна використовувати збережені під час розробки дані з ресурсів ScriptableObject.

Дані, збережені в ScriptableObjects за допомогою інструментів редактора як ресурс, записуються на диск і залишаються постійними між сеансами.[32]

Використовуйте клас JsonUtility для перетворення об'єктів Unity у формат JSON і навпаки. Наприклад, серіалізація JSON може бути корисною для взаємодії з веб-службами або для зручного пакування та розпакування даних у текстовий формат.

Серіалізація JSON базується на концепції «структурованого» JSON: ви створюєте клас або структуру для визначення змінних, які хочете зберігати у своєму JSON.

Ось приклад звичайного класу C#, що містить три змінні (level, timeElapsed і playerName), і позначеного атрибутом Serializable для роботи з JSON-серіалізатором.

Щоб перетворити об'єкт цього класу у формат JSON, використовуйте метод JsonUtility.ToJson. Для перетворення JSON назад в об'єкт використовуйте метод JsonUtility.FromJson.

Це створює новий екземпляр MyClass і встановлює для нього значення з JSON-даних. Якщо JSON-дані містять значення, що не відповідають полям у MyClass, серіалізатор ігнорує ці значення. Якщо в JSON-даних відсутні значення для полів у MyClass, серіалізатор залишає початкові значення цих полів у створеному об'єкті.

Якщо у даних JSON відсутнє значення для певного поля, серіалізатор не змінює значення цього поля. Цей підхід дозволяє мінімізувати розподіл об’єктів шляхом повторного використання вже створених об’єктів. Він також дозволяє "виправляти" об’єкти, навмисно перезаписуючи їх JSON-даними, які містять лише невелику підмножину полів.

JSON Serializer API підтримує підкласи MonoBehaviour і ScriptableObject, а також звичайні структури та класи. Однак під час десеріалізації JSON у підкласи MonoBehaviour або ScriptableObject необхідно використовувати метод FromJsonOverwrite. Використання методу FromJson у цьому випадку призведе до винятку в Unity, оскільки така поведінка не підтримується.

JsonUtility та EditorJsonUtility — це утиліти для серіалізації об'єктів у рядковий формат JSON і навпаки, використовуючи правила серіалізації Unity. Якщо потрібно обробляти JSON дані через код або серіалізувати структури даних, які не підтримуються Unity, можна додатково використовувати загальну .NET бібліотеку JSON.

Порівняльні тести показали, що JsonUtility значно швидший за популярні .NET бібліотеки JSON, хоча й може пропонувати менше функцій у деяких випадках.[33]

Нижче наведено декілька реалізованих алгоритмів, які були розроблені для гри:

```
Ссылок: 1
private void StaminaRecovery()
{
    Debug.Log("Current stamina"+ CurrentStm);
    if (CurrentStm < stm)
    {
        timeSinceLastRecovery += Time.deltaTime;
        if (timeSinceLastRecovery >= 1f / RecoveryRate)
        {
            CurrentStm += 5;
            timeSinceLastRecovery = 0f;
        }
    }
}
```

Рисунок 3.1 – Відновлення витривалості

```
Ссылок: 1
private void HealthRecovery()
{
    Debug.Log("Current health" + cHP);
    if (cHP < mHP)
    {
        timeSinceLastRecovery += Time.deltaTime;
        if (timeSinceLastRecovery >= 1f / RecoveryRate)
        {
            cHP++;
            timeSinceLastRecovery = 0f;
        }
    }
}
```

Рисунок 3.2 – Відновлення здоров’я

```
Ссылок: 0
public void OnClickStaminaAttack()
{
    if (CurrentStm >= staminaCostPerAttack)
    {
        CurrentStm -= staminaCostPerAttack;
        Combat.Instance.Attack();
    }
    else
    {
        Debug.Log("Need more stamina!");
    }
}
```

Рисунок 3.3 – Витрата витривалості при атаках

```
Ссылок: 2
public void Equip(Equipments newItem)
{
    int slotIndex = (int)newItem.equipSlot;

    Equipments oldItem = Unequip(slotIndex);
    if (onEquipmentChanged != null)
    {
        onEquipmentChanged.Invoke(newItem, oldItem);
    }
    currentEquipment[slotIndex] = newItem;
    AttachToMesh(newItem, slotIndex);
}
```

Рисунок 3.4 – Надягання спорядження

```
Ссылок: 5
public Equipments Unequip(int slotIndex)
{
    Equipments oldItem = null;

    if (currentEquipment[slotIndex] != null)
    {
        oldItem = currentEquipment[slotIndex];
        inventory.Add(oldItem);

        SetBlendShapeWeight(oldItem, 0);
        if (currentMeshes[slotIndex] != null)
        {
            Destroy(currentMeshes[slotIndex].gameObject);
        }
        currentEquipment[slotIndex] = null;
        if (onEquipmentChanged != null)
        {
            onEquipmentChanged.Invoke(null, oldItem);
        }
    }
    return oldItem;
}
```

Рисунок 3.5 – Знімання спорядження

```
Ссылок: 0
public void SavePortalsState()
{
    PortalData portalData = new PortalData();
    portalData.portalStates = new bool[portals.Length];

    for (int i = 0; i < portals.Length; i++)
    {
        portalData.portalStates[i] = portals[i].activeSelf;
    }

    string jsonData = JsonConvert.SerializeObject(portalData);
    File.WriteAllText(saveFilePath, jsonData);
}
```

Рисунок 3.6 – Зберігання інформації про стан порталів

```
Сообщение Unity | Ссылок: 0
private void OnTriggerEnter(Collider other)
{
    if (Time.time - lastDamageTime >= damageCooldown)
    {
        if (other.CompareTag("Player"))
        {
            Stats playerStats = other.GetComponentInParent<Stats>();

            if (playerStats != null)
            {
                playerStats.DamageTaken(enemyStats.EnemyDamageCount);

                lastDamageTime = Time.time;
            }
        }
    }
}
```

Рисунок 3.7 – Отримання шкоди

```
Скрипт Unity | Ссылок: 0
public class ShaseBehavior : StateMachineBehaviour
{
    NavMeshAgent agent;
    Transform player;
    float attakRange = 1.5f;
    Сообщение Unity | Ссылок: 0
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        agent = animator.GetComponent<NavMeshAgent>();
        player = GameObject.FindGameObjectWithTag("Player").transform;
    }


    Сообщение Unity | Ссылок: 0
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.transform.LookAt(player.transform.position);
        agent.SetDestination(player.position);
        float distance = Vector3.Distance(animator.transform.position, player.position);
        if (distance < attakRange)
        {
            animator.SetBool("isAttack", true);

        }


    }


    Сообщение Unity | Ссылок: 0
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
       agent.SetDestination(agent.transform.position);
        agent.speed = 4;
    }

}
```

Рисунок 3.8 – Переслідування головного героя ворогом

```
Ссылок: 3
public void OnDrag(PointerEventData eventData)
{
    cam = null;
    if (canvas.renderMode == RenderMode.ScreenSpaceCamera)
        cam = canvas.worldCamera;

    Vector2 position = RectTransformUtility.WorldToScreenPoint(cam, background.position);
    Vector2 radius = background.sizeDelta / 2;
    input = (eventData.position - position) / (radius * canvas.scaleFactor);
    FormatInput();
    HandleInput(input.magnitude, input.normalized, radius, cam);
    handle.anchoredPosition = input * radius * handleRange;
}
```

Рисунок 3.9 – Логіка роботи джойстика

```
Ссылок: 1
private void CharecterMove()
{
    moveVector = Vector3.zero;
    moveVector.y = 0.00001f;
    moveVector.x = joystick.Direction.x * speedMove * Time.deltaTime;
    moveVector.z = joystick.Direction.y * speedMove * Time.deltaTime;


    if (moveVector.x != 0 || moveVector.z != 0) ch_animator.SetBool("Move", true);
    else ch_animator.SetBool("Move", false);

    if (joystick.Horizontal != 0 || joystick.Vertical != 0)
    {
        transform.rotation = Quaternion.LookRotation(moveVector);
    }
    moveVector.y = gravityForce;
    ch_controller.Move(moveVector * Time.deltaTime);


}
Ссылок: 1
private void GamingGraviry()
{
    if (!ch_controller.isGrounded) gravityForce -= 20f * Time.deltaTime;
    else gravityForce = -1f;
}
```

Рисунок 3.10 – Рух головного героя

## 3.3 Тестування програмного забезпечення

Ручне тестування – це метод перевірки програмного забезпечення, при якому тестувальник виконує тестові сценарії вручну, без застосування автоматизованих засобів. Основна мета ручного тестування полягає у виявленні помилок, недоліків та дефектів у програмному продукті. Цей вид тестування є найпростішою технікою серед усіх типів тестування, що дозволяє знаходити критичні помилки в програмному додатку.

Переваги ручного тестування:

— Ручне тестування дозволяє виявити майже всі помилки та проблеми програмного забезпечення.
— Завдяки ручному тестуванню тестувальники можуть перевіряти візуальні компоненти, такі як макет, текст та інші елементи. Вони також можуть виявляти проблеми з UI та UX.
— Цей метод підходить для внесення незапланованих змін до програм, оскільки їх легко прийняти.
— Через відсутність необхідності у використанні висококваліфікованих навичок чи спеціальних інструментів, вартість операцій буде низькою.

Недоліки ручного тестування:

— Головний мінус ручного тестування полягає в тому, що воно потребує багато часу.
— Виявлення комбінацій кольорів і відмінностей у розмірах об'єктів GUI за допомогою ручного тестування є складним завданням.
— Тестування продуктивності та навантаження неможливо здійснити вручну.
— Проведення регресійних тестів у ручному тестуванні займає багато часу.

Види ручного тестування:

— Тестування білого ящика: Також відоме як тестування прозорої коробки, структурне або скляної коробки. У цьому методі тестувальники використовують внутрішнє розуміння системи та навички програмування для створення тестів. Зазвичай таке тестування проводиться на рівні підрозділу.

— Тестування чорного ящика: При тестуванні «чорної скриньки» тестувальники оцінюють функціональність програми без знання її внутрішньої структури. Тестування «чорного ящика» може проводитися на всіх рівнях тестування, таких як інтеграційне, модульне, приймальне та системне тестування.
— Приймальні випробування: Відомі також як передсерійні випробування. Крім тестувальників, у цьому тестуванні беруть участь кінцеві користувачі для перевірки функціональності програми. Після успішного завершення приймального тестування проводиться формальна оцінка, щоб вирішити, чи відповідає програма вимогам.
— Модульне тестування: Також називається тестуванням компонентів або модулів. Проводиться для перевірки правильності роботи окремого модуля або одиниці коду. Виконується розробниками у їхньому середовищі.
— Тестування системи: Це процес тестування повністю інтегрованої програми для оцінки її відповідності визначеним вимогам. Відомий також як наскрізне тестування, він перевіряє всю систему, щоб упевнитися, що програма працює відповідно до плану.
— Інтеграційне тестування: Це метод тестування взаємодії між двома програмними модулями. Інтеграційне тестування виконується за допомогою різних підходів, таких як «зверху вниз», «знизу вгору» та «великого вибуху».

Помилка — це недолік або дефект програми, який порушує її нормальну роботу, викликаючи відмінність між очікуваною та фактичною поведінкою програми. Дефект з'являється, коли розробник допускає помилку під час створення програми. Якщо тестувальник виявляє цю ваду, це називається дефектом або помилкою.

Життєвий цикл помилки, також відомий як життєвий цикл дефекту, охоплює всі етапи, через які проходить помилка від моменту її виявлення до виправлення. Цей процес починається з того, що тестувальник знаходить нову помилку, і завершується, коли тестувальник підтверджує, що вона виправлена.

Стани життєвого циклу помилки:

— Новий: Це початковий етап життєвого циклу помилки. Коли тестувальник виявляє нову помилку, вона одразу отримує статус "Новий", після чого відбувається її тестування та перевірка.
— Призначено: На цьому етапі керівник проекту або тестувальної групи призначає виявлену помилку команді розробників для подальшої роботи.
— Відкрито: На цьому етапі розробник починає аналізувати та виправляти помилку. Якщо помилка визнана недоречною, її статус може бути змінений на один з наступних: "Відкладено", "Дублікат", "Не є помилкою" або "Відхилено" в залежності від причини.
— Виправлено: Після внесення змін для виправлення помилки розробник змінює її статус на "Виправлено".
— Повторне тестування: На цьому етапі тестувальник повторно перевіряє помилку, щоб упевнитися, що розробник виправив її відповідно до вимог.
— Повторне відкриття: Якщо помилка залишається, команда тестувальників знову призначає її розробнику для перевірки, змінюючи статус на "Повторно відкрити".
— Перевірено: Якщо після повторного тестування тестувальник не виявляє жодних проблем і вважає, що помилку виправлено правильно, він змінює її статус на "Перевірено".
— Закрито: Коли помилка більше не існує, тестувальник змінює її статус на "Закрито".
— Дублікат: Якщо розробник виявляє, що помилка аналогічна вже існуючій або має таку ж концепцію, він змінює її статус на "Дублікат".
— Відхилено: Якщо розробник вважає, що помилка не є дійсною, він позначає її як "Відхилено".
— Не помилка: Якщо помилка не впливає на функціональність програми, її статус змінюється на "Не помилка".

— Відкладено: Якщо розробник вважає, що помилка має низький пріоритет і може бути виправлена в майбутніх випусках, він змінює її статус на "Відкладено".

Тестовий приклад — це набір умов, за яких тестувальник визначає, чи відповідає програма очікуваним результатам. Розробка тестових прикладів включає в себе назви випадків, початкові умови, очікувані результати та вхідні дані. Тестовий приклад є першим кроком, який випливає з тестових сценаріїв. Це докладний документ, який містить інформацію про процес тестування, стратегію тестування, очікуваний результат і початкові умови.[25]

Основна мета тестування ігор - виявлення помилок, щоб зменшити їх кількість перед випуском продукту. Баги - це результат неправильно написаного коду. Вони можуть варіюватися від серйозних дефектів, які заважають гравцям проходити гру, до менш значних, таких як неправильне розміщення тексту. Для полегшення роботи розробників баги класифікуються за ступенем серйозності. Це допомагає визначити, які помилки потрібно виправити в першу чергу.

Класифікація багів за важливістю:

— S1 Блокуюча: Ця помилка призводить до припинення нормальної роботи гри, що унеможливлює подальше тестування її функціоналу. Вирішення цієї проблеми є необхідним для продовження функціонування гри.

— S2 Критична: Критичний дефект впливає на основні аспекти, і ключова логіка гри не функціонує належним чином. Наприклад, існує потенційна безпекова уразливість, що може спричинити тимчасове відключення сервера або недоступність певної функціональності без можливості усунення проблеми через альтернативні шляхи. Вирішення цих проблем є обов'язковим для продовження роботи з основними аспектами перевіреної гри.

— S3 Значна: Основний механізм не працює належним чином через недоліки в логіці. Хоча помилка не критична або можливі способи вирішення її за допомогою альтернативних входів.

— S4 Незначна: Маленька помилка, що не порушує логіку перевіреної гри, зазвичай виникає через проблеми з інтерфейсом користувача.

— S5 Тривіальна: Помилка не стосується логіки гри. Замість цього, це невдало відтворений дефект, незначна проблема через інтерфейс користувача, питання зі сторонніми службами або помилка, що не має вирішального значення для загальної якості продукту.

Класифікація багів за категоріями:

— Візуальні аспекти: Розрив та обрізання областей зображення, відсутність текстур.

— Аудіо: Відсутність або спотворення звуку, надмірно висока або низька гучність.

— Дизайн рівнів: Невидимі стіни, відсутність колізії.

— Штучний інтелект: NPC не може правильно рухатися під час гри або не може відкрити двері.

— Фізика: Об’єкти літають у повітрі, їх неможливо зламати або зупинити після штовхання.

— Стабільність: Зависання, вильоти, примусовий вихід з гри.

— Продуктивність: Рівні завантажуються надто довго, немає можливості запустити гру з мінімальними вимогами, робота гри занадто часто зупиняється через завантаження даних.

— Мережа: Проблеми з підключенням, невидимі гравці, неможливо приєднатися по запрошенню.[26]

Нижче наведено декілька скріншотів, які були створені під час тестування гри:

Рисунок 3.11 – Підсвічування активного порталу

Рисунок 3.12 – Оглушення під час атаки ворога


Рисунок 3.13 – Скидання використаних балів навичок

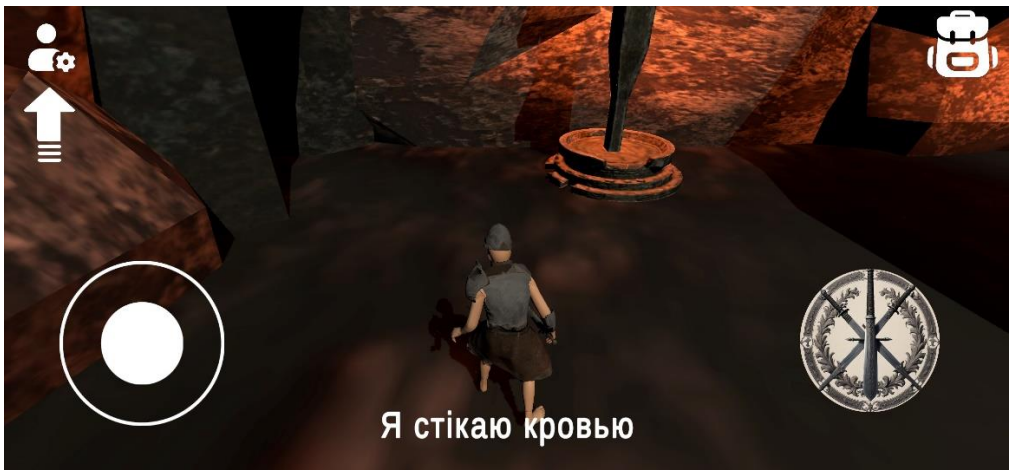


Рисунок 3.14 – Повідомлення при зниженні здоров'я до 50 відсотків


Рисунок 3.15 – Збереження речей, балів навичок та прогресу проходження

Рисунок 3.16 – Зміна спорядження через інвентар

Рисунок 3.17 – Портал не активний, доки головний ворог залишається живим


Рисунок 3.18 – Повідомлення при зниженні здоров'я до 30 відсотків

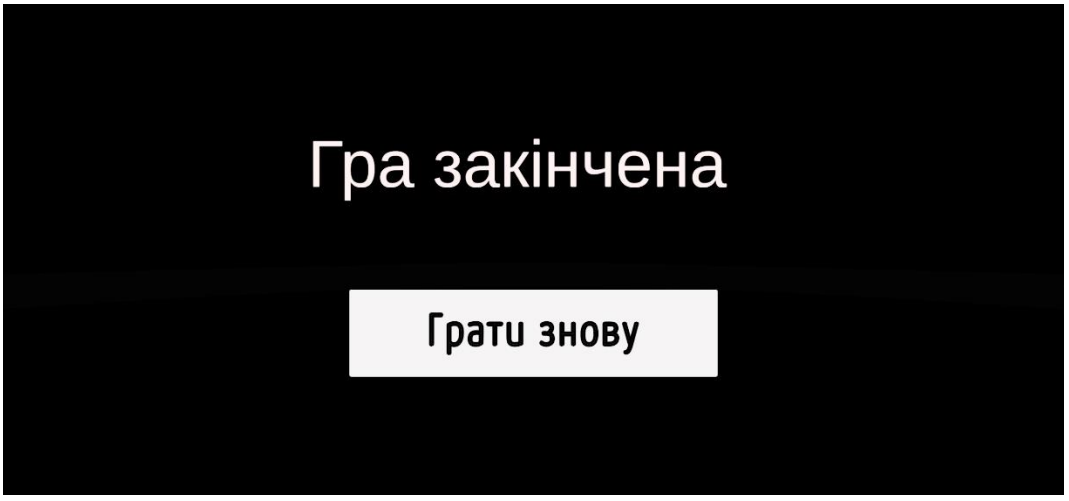


Рисунок 3.19 – Після смерті головного героя на екрані з'являється повідомлення

# ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було успішно розроблено та протестовано інформаційне та програмне забезпечення ігрової системи піджанру рольовий бойовик. В процесі дослідження було детально розглянуто різноманітні інструменти, проведено аналіз їх переваг та недоліків, в результаті якого був обраний найбільш ефективний для досягнення поставленої задачі.

У ході виконання кваліфікаційної роботи бакалавра було виконано наступні завдання:

- Розглянуто що таке мобільні ігри та різновидність їх класифікацій.
- Розглянуто та проаналізовано продукти аналогічного призначення.
- Розглянуто та проаналізовано розробку ігрової концепції.
- Розглянуто та проаналізовано прототипування ігрового середовища.
- Розглянуто та проаналізовано дизайн гри.
- Розглянуто та проаналізовано вибір середовища розробки та мови програмування.
- Зроблено короткий опис програмної реалізації.
- Проведено тестування програмного забезпечення.

Результатом цієї роботи стало успішне створення та тестування програмного продукту, який відповідає вимогам і потребам сучасного гравця рольових ігор.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Тенденції мобільного геймінгу у 2023 році. *Новини України*. URL: https://ua-news.in.ua/tendencii-mobilnogo-gejmingu-u-2023-roci/ (дата звернення: 28.09.2023).
2. Найперспективніші жанри для розробки мобільних ігор в 2022 ℗. *Новинарня - новини України, що воює*. URL: https://novynarnia.com/2022/05/04/najperspektyvnishi-zhanry-dlya-rozrobky-mobilnyh-igor-v-2022/ (дата звернення: 28.09.2023).
3. Rise of Mobile Games Over the Years: 9 Shocking Reasons. *Juego Studio*. URL: https://www.juegostudio.com/blog/why-mobile-games-are-so-popular-major-reasons-behind-the-trend (дата звернення: 28.09.2023).
4. Gitnux. A Closer Look at the Mobile Gaming Industry 2023: Statistics and Trends • Gitnux. *GITNUX*. URL: https://blog.gitnux.com/mobile-gaming-industry-statistics/ (дата звернення: 28.09.2023).
5. Redbytes Software. Mobile Game Development – It's Past, Present & Future. *Medium*. URL: https://medium.com/@RedbytesSoftwar/mobile-game-development-its-past-present-future-a8ce77b5eabd (дата звернення: 28.09.2023).
6. Myk Eff. Game Genres & Tropes. *Medium*. URL: https://soundand.design/game-genres-tropes-7cc4134259f7 (дата звернення: 28.09.2023).
7. Action RPG (Concept) - Giant Bomb. *Giant Bomb*. URL: https://www.giantbomb.com/action-rpg/3015-8592/#:~:text=In%20action%20RPGs,%20the%20player,attack%20enemies%20in%20real-time (дата звернення: 28.09.2023).
8. Almora Darkosen RPG: A Stunning Mobile Game with Mod APK Download. URL: https://www.pixnet.net/pcard/8656564c710e02fedb/article/a2c3f680-31e3-11ee-8790-5f9b399e65bc (дата звернення: 28.09.2023).

9. Anima ARPG (2019) Review - Hardcore Droid % Hardcore Droid. *Hardcore Droid*. URL: https://www.hardcoredroid.com/anima-arpg-2019-review/ (дата звернення: 28.09.2023).
10. Eternium - Everything You Need to Know - Game Review. *GamesVega.com*. URL: https://gamesvega.com/eternium-game-review/ (дата звернення: 28.09.2023).
11. What is a Game Engine? | Studytonight. *Studytonight - Best place to Learn Coding Online*. URL: https://www.studytonight.com/3d-game-engineering-with-unity/game-engine (дата звернення: 28.09.2023).
12. What Is Unity? - A Top Game Engine Solution For Video Games - GameDev Academy. *GameDev Academy*. URL: https://gamedevacademy.org/what-is-unity/ (дата звернення: 28.09.2023).
13. What is Unreal Engine?. *Concept Art Empire*. URL: https://conceptartempire.com/what-is-unreal-engine/ (дата звернення: 28.09.2023).
14. Yoshitaka Shiotsu. C# vs. C++: Which Language is Best for Your Project?. *Upwork*. URL: https://www.upwork.com/resources/c-sharp-vs-c-plus-plus#:~:text=C++%20programs%20include%20server-side,web,%20mobile%20and%20desktop%20applications (дата звернення: 28.09.2023).
15. Prakhar juego. The Art of Mobile Games: Exploring Visual Trends and Techniques. *Medium*. URL: https://medium.com/@prakhar.l/the-art-of-mobile-games-exploring-visual-trends-and-techniques-ead7d59ab96a (дата звернення: 03.05.2024).
16. Magic Media Mobile Studio. Creating Stunning Visuals: Graphics and Animation in Mobile Games. *LinkedIn: Log In or Sign Up*. URL: https://www.linkedin.com/pulse/creating-stunning-visuals-graphics-animation-mobile?trk=article-ssr-frontend-pulse_more-articles_related-content-card (дата звернення: 03.05.2024).

17. How do video game graphics work?. *Geeky Gadgets*. URL: https://www.geeky-gadgets.com/how-do-video-game-graphics-work/ (дата звернення: 03.05.2024).
18. Silvaberto. How to Optimize Mobile Game Graphics. *Medium*. URL: https://medium.com/@silvaberto475/how-to-optimize-mobile-game-graphics-cb44d6943422 (дата звернення: 03.05.2024).
19. 2D vs 3D Game Development: Know the Major Differences. *Juego Studio*. URL: https://www.juegostudio.com/blog/2d-vs-3d-game-development (дата звернення 03.05.2024).
20. rct AI. Rules, Mechanics, Gameplays and Logic of Game Development. *Medium*. URL: https://rctai.medium.com/rules-mechanics-gameplays-and-logic-of-game-development-3573b9d730aa (дата звернення: 03.05.2024).
21. The Strange World of Video Game Logic. *Game Rant*. URL: https://gamerant.com/worst-video-game-logic/ (дата звернення: 03.05.2024).
22. Video Game Development and Algorithms. *TheGWW.com*. URL: https://thegww.com/video-game-development-and-algorithms/ (дата звернення: 03.05.2024).
23. How AI Is Changing Video Games. *Medium*. URL: https://medium.com/@vakingbizz12/how-ai-is-changing-video-games-e83300d7b2c2 (дата звернення: 03.05.2024).
24. Algorithms for Game and Simulation Development. *Skills Over Paper*. URL: https://www.skillsoverpaper.com/blog/algorithms-for%20game%20and%20simulation%20development (дата звернення: 03.05.2024).
25. Sharma M. Manual Testing Tutorial: What is, Types, Concepts. *LinkedIn: Log In or Sign Up*. URL: https://www.linkedin.com/pulse/manual-testing-tutorial-what-types-concepts-mamta-sharma (дата звернення: 03.05.2024).

26. The most common categories of game bugs | Online courses from QATestLab. *Online courses from QATestLab | Main page*. URL: https://en.training.qatestlab.com/blog/technical-articles/category-bug-game/ (дата звернення: 03.05.2024).
27. Unity - Scripting API: MonoBehaviour. *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (дата звернення: 03.05.2024).
28. Unity - Scripting API: MonoBehaviour.Start(). *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html ( дата звернення: 03.05.2024).
29. Unity - Scripting API: MonoBehaviour.Update(). *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html (дата звернення: 03.05.2024).
30. Unity - Scripting API: MonoBehaviour.FixedUpdate(). *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html (дата звернення: 03.05.2024).
31. Unity - Scripting API: MonoBehaviour.LateUpdate(). *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.LateUpdate.html (дата звернення: 03.05.2024).
32. Unity - Manual: ScriptableObject. *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (дата звернення: 03.05.2024).
33. Unity - Manual: JSON. *Unity - Manual: Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/Manual/JSONSerialization.html (дата звернення: 03.05.2024).

# ДОДАТОК А. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ПРОДУКТІВ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ

| Категорії | Almora Darkosen RPG | Anima ARPG | Eternium |
|---|---|---|---|
| Сюжет | + | - | + |
| Графіка | + | + | + |
| Анімації | + | + | + |
| Інтерфейс | + | + | + |
| Аудіосупроводження | + | + | + |
| Динамічність | + | + | + |
| Оптимізація | + | + | + |

# ДОДАТОК Б. ЗНІМКИ ЕКРАНУ РОЗРОБЛЕНОГО ДОДАТКУ

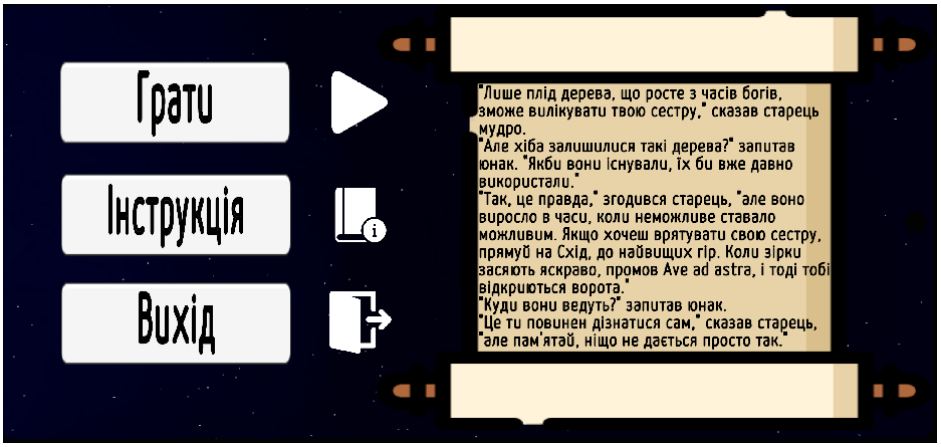


Рисунок Б.1 – Меню гри при запуску


Рисунок Б.2 – Бали навичок на інвентар персонажа

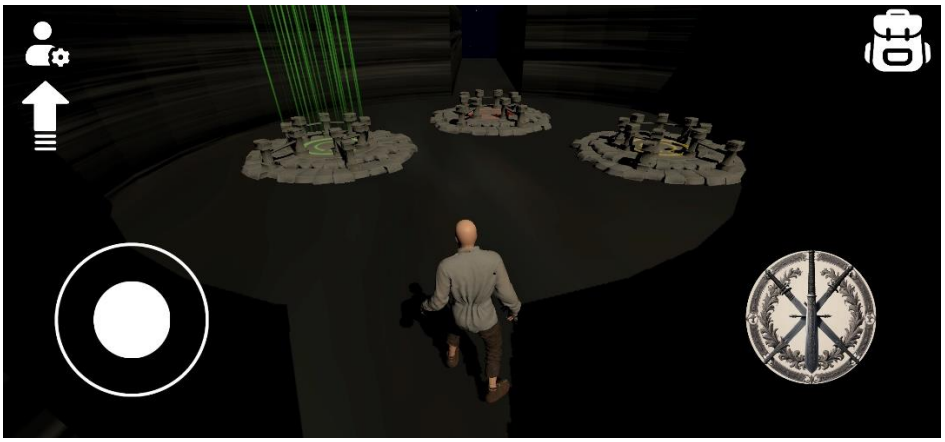

Рисунок Б.3 – Головна ігрова локація

Рисунок Б.4 – Перший ігровий рівень

Рисунок Б.5 – Другий ігровий рівень

Рисунок Б.6 – Третій ігровий рівень

# ДОДАТОК С. ВИХІДНІ КОДИ РОЗРОБЛЕНОГО ДОДАТКУ

Оскільки обсяг всіх скриптів досить великий, тут буде представлена лише їх частина. Решту коду можна знайти в репозиторії разом з іншими компонентами програми. Посилання на репозиторій наведено нижче.

https://github.com/HodoniukDmytro/Infoware-and-Software-of-Action-Role-Playing-Game-System

Скрипт під назвою «Stats».

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

public class Stats : MonoBehaviour
{
    #region
    public static Stats instanse;
    private void Awake()
    {
        instanse = this;
    }

    #endregion
    public StatsEquipment statDamage;
    public StatsEquipment statArmor;
    public StatsEquipment statCriticalResistance;
    public int cHP;
    public int mHP = 100;
    public double spd;
    public int stm = 100;
    public float exp = 0;
    public int damage = 10;
    public int Lvl = 1;
    public int lvlPoint;
    public float RecoveryRate = 1f;
    public int staminaCostPerAttack = 20;
    private float timeSinceLastRecovery = 0f;
    public int CurrentStm;
    public float RequiredExperience = 10;
    private System.Random random;
    public float CriticalChance = 0.3f;
    private string savePath;
    Animator ch_animator;

    void Start()
    {
        random = new System.Random();
        cHP = mHP;
        CurrentStm = stm;
        Debug.Log(+CurrentStm);
        Debug.Log(+mHP);
```

```
        ch_animator = GetComponent<Animator>();
        EquipmentManager.instance.onEquipmentChanged += OnEquipmentChanget;
        savePath = Application.persistentDataPath + "/stats.json";
        Debug.Log(savePath);
    }


    void FixedUpdate()
    {
        StaminaRecovery();
        HealthRecovery();
    }

    public void SaveStats()
    {
        StatsData data = new StatsData();
        data.Lvl = Lvl;
        data.LvlPoint = lvlPoint;
        data.mHP = mHP;
        data.exp = exp;
        data.damage = damage;

        string jsonData = JsonUtility.ToJson(data);
        File.WriteAllText(savePath, jsonData);
    }

    public void LoadStats()
    {
        if (File.Exists(savePath))
        {
            string jsonData = File.ReadAllText(savePath);
            StatsData data = JsonUtility.FromJson<StatsData>(jsonData);

            Lvl = data.Lvl;
            lvlPoint = data.LvlPoint;
            mHP = data.mHP;
            exp = data.exp;
            damage = data.damage;
        }
    }

    public bool IsCritical()
    {
        float CR = statCriticalResistance.GetValue() * 0.01f;
        float rand = (float)random.NextDouble();
        if (rand <= CriticalChance - CR)
        {
            return true;
        }
        return false;
    }

    public void DamageTaken(int dT)
    {
        int fd = statArmor.GetValue();
        if(dT > fd)
        {
            cHP -= dT - statArmor.GetValue();
            if (cHP <= 0)
```

```
            {
                InterfaceUI.instanse.InterfaceHide();
                ch_animator.SetTrigger("Death");
                StartCoroutine(DelayedSceneLoad(4, 2));

            }
            if (IsCritical())
            {
                ch_animator.SetTrigger("isStun");
            }
            Debug.Log("Current HP" + cHP);
        }
        if (cHP <= cHP/100 * 30)
        {

        }
        if (cHP >= cHP / 100 * 30)
        {

        }

    }

    private IEnumerator DelayedSceneLoad(float delay, int sceneIndex)
    {
        yield return new WaitForSeconds(delay);
        SceneManager.LoadScene(sceneIndex);
    }

    public void LevelUp(int experiencePoints)
    {
        exp += experiencePoints;
        while (exp >= RequiredExperience*Lvl)
        {
            Lvl++;
            exp -= RequiredExperience;
            RequiredExperience += RequiredExperience * 1.5f;
            lvlPoint++;
        }

    }


    private void StaminaRecovery()
    {
        Debug.Log("Current stamina"+ CurrentStm);
        if (CurrentStm < stm)
        {
            timeSinceLastRecovery += Time.deltaTime;
            if (timeSinceLastRecovery >= 1f / RecoveryRate)
            {
                CurrentStm += 5;
                timeSinceLastRecovery = 0f;
            }
        }
    }

    private void HealthRecovery()
    {
```

```
    Debug.Log("Current health" + cHP);
    if (cHP < mHP)
    {
        timeSinceLastRecovery += Time.deltaTime;
        if (timeSinceLastRecovery >= 1f / RecoveryRate)
        {
            cHP += 2;
            timeSinceLastRecovery = 0f;
        }
    }
}

public void OnClickStaminaAttack()
{
    if (CurrentStm >= staminaCostPerAttack)
    {
        CurrentStm -= staminaCostPerAttack;
        Combat.Instance.Attack();
    }
    else
    {
        Debug.Log("Need more stamina!");
    }
}

public void OnClickStaminaUp()
{
    if (lvlPoint >= 1)
    {
        lvlPoint--;
        stm += 10;
    }
}

public void OnClickHealthUp()
{
    if (lvlPoint >= 1)
    {
        lvlPoint--;
        mHP += 5;
    }
}

public void OnClickDamageUp()
{
    if (lvlPoint >= 1)
    {
        lvlPoint--;
        damage += 5;
    }
}

public void ResetStats()
{

    lvlPoint += (damage - 10) / 5 + (mHP - 100) / 5 + (stm - 100) / 10;
    damage = 10;
    mHP = 100;
    stm = 100;
```

```
    }

    void OnEquipmentChanget(Equipments newItem, Equipments oldItem)
    {
        if (newItem != null)
        {
            statArmor.AddModifier(newItem.armorModifier);
            statDamage.AddModifier(newItem.damageModifier);
            statCriticalResistance.AddModifier(newItem.criticalModifer);
        }

        if (oldItem != null)
        {
            statArmor.RemoveModifier(oldItem.armorModifier);
            statDamage.RemoveModifier(oldItem.damageModifier);
            statCriticalResistance.RemoveModifier(oldItem.criticalModifer);
        }
    }
}
```

Скрипт під назвою «CharacterMovement».

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class CharacterMovement : MonoBehaviour
{
    public float speedMove;
    private Vector3 moveVector;
    private float gravityForce;
    private CharacterController ch_controller;
    private Animator ch_animator;
    public FixedJoystick joystick;
    public GameObject targetEnemy;

    public int sceneIndex;
    Transform enemy;

    void Start()
    {
        ch_controller = GetComponent<CharacterController>();
        ch_animator = GetComponent<Animator>();
    }

    void Update()
    {
        GamingGraviry();
        CharecterMove();
    }


    private void CharecterMove()
    {
        moveVector = Vector3.zero;
        moveVector.y = 0.00001f;
        moveVector.x = joystick.Direction.x * speedMove * Time.deltaTime;
        moveVector.z = joystick.Direction.y * speedMove * Time.deltaTime;
```

```
        if (moveVector.x != 0 || moveVector.z != 0) ch_animator.SetBool("Move",
true);
        else ch_animator.SetBool("Move", false);

        if (joystick.Horizontal != 0 || joystick.Vertical != 0)
        {
            transform.rotation = Quaternion.LookRotation(moveVector);
        }
        moveVector.y = gravityForce;
        ch_controller.Move(moveVector * Time.deltaTime);

    }

    private void GamingGraviry()
    {
        if (!ch_controller.isGrounded) gravityForce -= 20f * Time.deltaTime;
        else gravityForce = -1f;
    }
}
```

Скрипт під назвою «CharacterDamage».

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

public class CharacterDamage : MonoBehaviour
{
    public int CurrentDamage;
    public float damageCooldown = 1f;

    private float lastDamageTime;

    public void FixedUpdate()
    {
        CurrentDamage = Stats.instanse.damage +
Stats.instanse.statDamage.GetValue();
    }

    private void OnTriggerEnter(Collider other)
    {
        Debug.Log("damage enemy");
        if (Time.time - lastDamageTime >= damageCooldown)
        {
            if (other.CompareTag("Enemy"))
            {
                EnemyStats enemyStats = other.GetComponentInParent<EnemyStats>();

                if (enemyStats != null)
                {
                    enemyStats.TakeDamage(CurrentDamage);

                    lastDamageTime = Time.time;
                }
            }
        }
    }
}
```