МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «УНІВЕРСИТЕТ «КРОК»

Фаховий коледж Університету «КРОК»

ДИПЛОМНА РОБОТА

за темою

«Розробка гри Ранер»

Студент 4 курсу групи КН-20К

Марцьоха Владислав Андрійович

(прізвище, ім'я та по-батькові студента)

(підпис студента)

10.06.24

(дата)

Керівник дипломної роботи

Кафедра комп'ютерних наук,доцент

(посада керівника)

Чернозубкін Ігор Олександрович

(прізвище, ім'я та по-батькові керівника)

До захисту

(резолюція «До захисту»)

(підпис викладача)

Київ, 2024 рік

Скорочення

NPC (Non-Player Character, персонаж, яким не керує гравець) — це персонаж у відеоіграх або рольових іграх, який контролюється програмою чи ведучим гри (майстром).

2D (двовимірний) в контексті ігор означає ігри, в яких об'єкти та персонажі представлені на плоскій площині, маючи лише дві виміри: довжину та ширину. У таких іграх відсутня глибина, і геймплей зазвичай розгортається в площині, обмеженій осями X і Y.

3D (тривимірний) в контексті ігор означає ігри, в яких об'єкти та персонажі мають три виміри: довжину, ширину та глибину. Це створює відчуття простору і об'ємності, дозволяючи гравцям пересуватися і взаємодіяти в більш реалістичному середовищі, де геймплей охоплює осі X, Y і Z.

AR або доповнена реальність - це технологія, яка дозволяє додавати віртуальні об'єкти і інформацію до реального світу, зберігаючи одночасну інтеракцію з ним. Вона створює враження того, що віртуальні об'єкти існують в реальному просторі користувача, що відкриває широкий спектр можливостей для ігор, навчання, дизайну і бізнесу.

Зміст

Вступ

Ігрова індустрія стала однією з найбільш динамічних та швидкозростаючих галузей сучасного технологічного світу. Розробка відеоігор, зокрема платформерів, є цікавим і творчим процесом, що вимагає знань у програмуванні, дизайні та менеджменті проєктів. Unity є однією з найпопулярніших платформ для створення ігор завдяки своїй доступності та потужним інструментам.

Ігри жанру платформер завжди були популярні серед гравців різного віку завдяки їх динамічному геймплею та простій концепції. Основною ідеєю платформерів є керування персонажем, який пересувається по рівню, долаючи перешкоди та ворогів, збираючи предмети та досягаючи кінцевої точки. Платформери мають велику історію і включають в себе класичні ігри, такі як Super Mario Bros., Sonic the Hedgehog, Castlevania, які залишаються актуальними й до сьогодні.

Інді-ігри є видом відеоігор, створених незалежними розробниками або маленькими студіями, зазвичай без фінансової підтримки великих видавців. Вони часто відрізняються оригінальним підходом до геймплею, унікальним художнім стилем і часто мають глибокий сюжет або метафоричний підтекст. Ось деякі особливості інді-ігор:

Творча свобода: Однією з головних особливостей інді-ігор є творча свобода розробників. Вони часто мають можливість втілити в життя свої найсміливіші ідеї без обмежень, які можуть бути присутніми при роботі з великими видавцями.

Різноманітність стилів та жанрів: Інді-ігри охоплюють широкий спектр жанрів, від пригодницьких ігор та платформерів до головоломок та симуляторів. Це дозволяє гравцям знаходити ігри, які відповідають їх смакам та уподобанням.

Експерименти та інновації: Інді-розробники часто експериментують з ігровими механіками, створюючи унікальні та нестандартні ігрові досліди.

Вони можуть впроваджувати новаторські механіки, ігрові механізми або використати незвичайні історії та візуальні стилі.

Глибокі сюжети і метафори: Багато інди-ігри мають глибокий сюжет і метафоричний підтекст, торкаючись тем, таких як дружба, любов, втрата, толерантність, соціальна несправедливість і навіть психологічні проблеми.

Спільнота та підтримка: Інді-ігри часто мають активну спільноту гравців та розробників, які обмінюються ідеями, обговорюють ігровий процес і навіть пропонують зворотний зв'язок для покращення ігор. Це створює ближчі зв'язки між творцями та споживачами.

Розробка 2D-платформера в Unity дозволяє навчитися основам програмування ігрових механік, створення анімацій, роботи з фізичними об'єктами та взаємодії з різними компонентами ігрового рушія. Unity надає широкі можливості для реалізації ідей розробників, забезпечуючи інструменти для роботи зі спрайтами, звуком та анімацією. Крім того, Unity пропонує великий вибір готових ресурсів і модулів, які значно спрощують процес розробки і зменшують час на створення прототипів.

Мета даної дипломної роботи полягає у створенні шаблону 2D-платформера, який може бути використаний як базис для подальшої розробки ігор цього жанру. Завданнями роботи є налаштування проєкту в Unity, розробка основних ігрових механік, тестування та оптимізація продуктивності гри. Цей шаблон стане корисним інструментом для розробників-початківців, які хочуть ознайомитися з основними принципами створення ігор на платформі Unity.

У даній роботі буде детально розглянуто процес створення 2D-платформера, починаючи з вибору концепції та налаштування проєкту, закінчуючи тестуванням та налагодженням фінальної версії гри. Розглядаються також ключові аспекти ігрового дизайну, такі як створення персонажів, рівнів, анімацій та звукового супроводу. Особлива увага приділяється оптимізації продуктивності та забезпеченню плавного геймплею, що є важливим для успіху будь-якої гри.

Розробка ігрових проєктів у Unity є чудовим навчальним інструментом для студентів, оскільки дозволяє інтегрувати теоретичні знання з практичними навичками. Проєктування та програмування ігор стимулює розвиток логічного мислення, креативності та навичок розв'язання проблем. Крім того, створення власної гри може стати цінним додатком до портфоліо, що підвищує шанси на успішне працевлаштування в ігровій індустрії.

Таким чином, виконання даного проєкту не лише дозволить оволодіти базовими навичками розробки ігор, але й забезпечить створення якісного продукту, який може бути використаний для подальшої роботи або комерційних цілей. Успішне завершення проєкту підтвердить здатність студента застосовувати отримані знання на практиці та досягати поставлених цілей у сфері розробки відеоігор.

Розділ 1.Огляд платформи Unity

Unity — це потужний рушій для розробки ігор, який складається з багатьох компонентів, кожен з яких виконує важливу роль у процесі створення інтерактивних додатків та ігор. Давайте розглянемо основні компоненти Unity детальніше:

1. Unity Editor

Unity Editor - це інтегроване середовище розробки (IDE), яке забезпечує всі необхідні інструменти для створення, тестування та налаштування ігор. Основні частини Unity Editor включають:

- Scene View: дозволяє розробникам візуально розміщувати та налаштовувати об'єкти в грі. Це вікно надає можливість маніпулювати 3D та 2D об'єктами, встановлювати позицію, масштаб і ротацію.
- Game View: використовується для попереднього перегляду гри під час розробки. Тут можна бачити, як гра виглядатиме на різних платформах.
- Hierarchy: список всіх об'єктів у поточній сцені. Ці об'єкти можуть бути організовані в ієрархічну структуру, що полегшує їх управління.
- Project Window: показує всі файли та ресурси, які доступні в проекті. Це місце, де розташовуються скрипти, текстури, моделі, звукові файли та інші ресурси.
- Inspector: дозволяє переглядати та редагувати властивості вибраного об'єкта. Тут можна змінювати компоненти, додавати нові і налаштовувати існуючі.

2. GameObjects і Components

У Unity кожен об'єкт у грі представляється як GameObject. GameObjects є основними будівельними блоками в Unity і можуть бути використані для

створення будь-яких ігрових елементів - від персонажів і середовищ до камер і світлових джерел. Основні характеристики GameObjects включають:

3. Системи анімації та фізики

Unity має вбудовані системи анімації та фізики, які спрощують створення реалістичної поведінки об'єктів у грі:

- Animation System: дозволяє створювати і керувати анімаціями для персонажів та об'єктів. Animator Controller використовується для керування різними анімаційними станами і переходами між ними. Unity підтримує як 2D, так і 3D анімації, а також імпорт анімацій з інших програм.
- Physics System: забезпечує реалістичну симуляцію фізичних явищ, таких як гравітація, зіткнення та рух. Основними компонентами системи фізики є Rigidbody, Colliders і Joints. Rigidbody додає об'єкту фізичні властивості, Colliders визначають форму об'єкта для зіткнень, а Joints забезпечують зв'язок між об'єктами.

4. UI (User Interface) та UI Components

Unity також має потужні інструменти для створення користувацького інтерфейсу (UI):

- Canvas: головний компонент для створення UI. Всі елементи інтерфейсу, такі як кнопки, текстові поля і панелі, розміщуються на Canvas.
- UI Elements: Unity надає широкий набір готових елементів UI, таких як Button, Text, Image, Slider тощо. Ці елементи можна налаштовувати і інтерактивно з ними взаємодіяти через скрипти.
- Event System: відповідає за обробку вводу від користувача, такого як натискання миші або торкання на екрані. Це дозволяє створювати інтерактивні елементи інтерфейсу.

5. Scene Management і Asset Management

- Scene Management: Unity дозволяє розробникам організовувати гру в кілька сцен, які можна завантажувати і розвантажувати під час гри. Це зручно для розподілу гри на рівні або розділи.
- Asset Management: Unity підтримує широкий спектр форматів файлів і забезпечує зручне управління ресурсами. Asset Import Pipeline дозволяє автоматично імпортувати і обробляти ресурси, такі як моделі, текстури і аудіо.

6. Системи рендерингу та освітлення

Unity має потужні системи для рендерингу та освітлення, які дозволяють створювати візуально привабливі ігри:

- Rendering Pipeline: Unity підтримує різні рендеринг-пайплайни, включаючи Built-in, Universal Render Pipeline (URP) і High Definition Render Pipeline (HDRP). Кожен з них має свої особливості і підходить для різних типів проектів.
- Lighting: Unity підтримує різні типи освітлення, включаючи реальне часове освітлення, світлові мапи і глобальне освітлення. Це дозволяє створювати реалістичні і атмосферні сцени.

7. Навколишнє середовище та інструменти

Unity пропонує різні інструменти для створення навколишнього середовища та ефектів:

- Terrain Editor: дозволяє створювати і редагувати ландшафти. Інструменти для створення ландшафтів включають малювання висот, текстурування і розміщення дерев та рослинності.
- Particle System: забезпечує створення ефектів, таких як вогонь, дим, сніг і вибухи. Ця система дозволяє детально налаштовувати поведінку частинок і їх взаємодію з іншими об'єктами.

8. Інтеграція та розгортання

- Підтримка платформ: Unity підтримує розгортання на різні платформи, включаючи ПК, мобільні пристрої, консолі та веб-браузери. Це забезпечує максимальну гнучкість для розробників.
- Інтеграція з іншими сервісами: Unity дозволяє інтегрувати різні сервіси, такі як аналітика, монетизація, реклама та хмарні сервіси, що спрощує керування проектами і покращує взаємодію з гравцями.

1.1 Переваги використання Unity

Серед ключових переваг Unity можна виділити наступні аспекти.

По-перше, Unity забезпечує кросплатформену підтримку, що дозволяє розробникам створювати проекти для різних платформ, таких як ПК, мобільні пристрої, консолі, веб-браузери та навіть VR/AR пристрої. Це значно розширює аудиторію потенційних користувачів і спрощує процес розгортання проектів на різних платформах. Єдина кодова база може бути використана для створення гри, яка буде працювати на різних пристроях, що економить час і ресурси розробників.

По-друге, Unity має потужний і інтуїтивно зрозумілий Unity Editor, який є інтегрованим середовищем розробки (IDE). Unity Editor забезпечує розробникам доступ до різноманітних інструментів для створення, редагування і тестування ігор. Вікна, такі як Scene View, Game View, Hierarchy, Project Window та Inspector, роблять процес розробки зручним і ефективним. Scene View дозволяє візуально розміщувати об'єкти у сцені, Game View надає можливість попереднього перегляду гри, а Hierarchy і Project Window спрощують організацію об'єктів і ресурсів у проекті.

Ще однією важливою перевагою Unity є використання GameObjects і Components. Кожен об'єкт у грі представлений як GameObject, який може містити різні компоненти для додавання функціональності. Компоненти, такі як Transform, MeshRenderer, Rigidbody та інші, дозволяють налаштовувати об'єкти для відображення, фізичної взаємодії, звуку тощо. Це забезпечує гнучкість і модульність у розробці ігор, дозволяючи легко додавати, видаляти і налаштовувати функціональність об'єктів.

Unity також пропонує потужні системи анімації та фізики. Система анімації дозволяє створювати і керувати анімаціями для персонажів та об'єктів, використовуючи Animator Controller для керування анімаційними станами і переходами. Це дозволяє створювати реалістичні і плавні анімації. Система фізики Unity забезпечує реалістичну симуляцію фізичних явищ, таких як гравітація, зіткнення та рух. Компоненти, такі як Rigidbody, Colliders і Joints, дозволяють створювати фізично коректну поведінку об'єктів у грі.

Важливою перевагою Unity є також потужні інструменти для створення користувацького інтерфейсу (UI). Unity надає широкі можливості для створення і налаштування елементів UI, таких як кнопки, текстові поля, зображення та інші елементи. Компонент Canvas дозволяє організовувати UI елементи, а Event System забезпечує обробку вводу від користувачів. Це дозволяє створювати інтерактивний і зручний інтерфейс для користувачів гри.

Unity підтримує організацію гри в кілька сцен, що дозволяє розробникам розподіляти проект на окремі рівні або розділи. Система управління сценами спрощує завантаження і розвантаження сцен під час гри, що робить процес розробки більш структурованим і керованим. Крім того, Unity підтримує широкий спектр форматів файлів і забезпечує зручне управління ресурсами через Asset Management. Asset Import Pipeline дозволяє автоматично імпортувати і обробляти різні ресурси, такі як моделі, текстури, аудіо та інші.

Однією з ключових переваг Unity є її потужні системи рендерингу та освітлення. Unity підтримує різні рендеринг-пайплайни, включаючи Built-in, Universal Render Pipeline (URP) і High Definition Render Pipeline (HDRP). Це дозволяє розробникам вибирати оптимальний пайплайн залежно від вимог проекту. Система освітлення Unity підтримує різні типи освітлення, такі як реальне часове освітлення, світлові мапи і глобальне освітлення, що дозволяє створювати реалістичні і атмосферні сцени.

Unity також пропонує різноманітні інструменти для створення навколишнього середовища та ефектів. Terrain Editor дозволяє створювати і редагувати ландшафти, використовуючи інструменти для малювання висот,

текстурування і розміщення рослинності. Particle System забезпечує створення ефектів, таких як вогонь, дим, сніг і вибухи, дозволяючи детально налаштовувати поведінку частинок.

Крім того, Unity дозволяє інтегрувати різні сервіси, такі як аналітика, монетизація, реклама та хмарні сервіси. Це спрощує керування проектами і покращує взаємодію з гравцями. Unity надає розробникам доступ до Unity Asset Store, де можна знайти і придбати різноманітні ресурси, такі як моделі, текстури, звукові ефекти і готові скрипти, що прискорює процес розробки ігор.

Загалом, Unity - це універсальна і потужна платформа для розробки ігор, яка пропонує широкий спектр інструментів і можливостей. Кросплатформена підтримка, зручний Unity Editor, використання GameObjects і Components, потужні системи анімації і фізики, інструменти для створення UI, сцени та управління ресурсами, системи рендерингу та освітлення, інструменти для створення навколишнього середовища та інтеграція з різними сервісами - все це робить Unity незамінним інструментом для розробників ігор. Завдяки своїй гнучкості, масштабованості та потужності, Unity є популярним вибором серед розробників по всьому світу, дозволяючи створювати високоякісні ігри та інтерактивні додатки для різних платформ.

Розділ 2.Створення 2D-платформера

Відеоігри стали невід'ємною частиною сучасної культури, пропонуючи розваги та занурення в різні світи та історії. Серед багатьох жанрів ігор 2D-платформери виділяються своєю простотою та захоплюючим геймплеєм. Вони привертають гравців усіх вікових категорій завдяки зрозумілим механікам і яскравій графіці. Розробка 2D-платформерів стала ще доступнішою завдяки потужному ігровому рушію Unity, який надає безліч інструментів для створення як простих, так і складних ігор.

Рис.1 Приклад 2D гри.

У цій роботі розглядається процес створення

2D-платформера за допомогою шаблону 2D Platformer Template від Unity. Будуть охоплені всі етапи розробки: від налаштування середовища до тестування та налагодження гри. Це керівництво допоможе розробникам-початківцям зрозуміти основні принципи створення ігор та застосувати їх на практиці

2.1 Налаштування середовища розробки

Першим кроком у розробці будь-якої гри є налаштування середовища розробки. Для створення 2D-платформера на Unity необхідно виконати наступні дії:

1. Встановлення Unity:
   - Завантажити Unity Hub з офіційного сайту Unity.
   - Встановити останню версію Unity Editor через Unity Hub.
2. Створення нового проекту:
   - Відкрити Unity Hub і натиснути кнопку "New Project".

- Вибрати шаблон "2D" і задати назву проекту.
- Натиснути "Create" для створення нового проекту.

3. Імпорт шаблону 2D Platformer Template:
   - Завантажити шаблон з Unity Asset Store.
   - Імпортувати шаблон у новостворений проект через Unity Editor.

2.2 **Використання шаблону 2D Platformer Template**

Шаблон 2D Platformer Template надає базові компоненти та механіки для створення 2D-платформерів. Він містить префабрикати персонажів, ворогів, платформ та інших об'єктів, а також готові скрипти для керування ними.

1. Ознайомлення з компонентами шаблону:
   - Сцени: Шаблон містить кілька сцен, які демонструють різні аспекти гри.
   - Префабрикати: Готові об'єкти, які можна використовувати на сценах.
   - Скрипти: Коди, що забезпечують функціональність персонажів та об'єктів.
   - Анімації: Анімаційні кліпи для рухів персонажів та ворогів.

2.3 **Редагування сцени**:
- Відкрити сцену з шаблону та розмістити об'єкти на ній.
- Додати платформи, ворогів та збиральні предмети.

2.4 **Створення рівнів**

Процес створення рівнів включає в себе розміщення об'єктів та налаштування їх властивостей. Основні етапи цього процесу:

1. Розміщення платформ:
   - Використовувати різні типи платформ для створення цікавих рівнів.
   - Налаштувати властивості платформ, такі як рух або руйнування.
2. Додавання ворогів:
   - Розмістити ворогів на рівнях.
   - Налаштувати їх поведінку та взаємодію з гравцем.
3. Розміщення збиральних предметів:
   - Додати об'єкти, які гравець може збирати для отримання очок або бонусів.
   - Встановити правила їх появи та зникнення.

2.5 Розширення функціональності

Для того, щоб зробити гру більш цікавою та унікальною, необхідно додати нові механіки та функціональність. Це може включати:

1. Нові здібності персонажа:
   - Додати можливість подвійного стрибка, стрільби або використання спеціальних здібностей.
   - Створити нові анімації для цих дій.
2. Складніші вороги:
   - Розробити ворогів з більш складною поведінкою та атаками.
   - Додати босів з унікальними механіками.
3. Інтерактивні об'єкти:
   - Додати об'єкти, з якими гравець може взаємодіяти, такі як важелі, двері або пастки.
   - Налаштувати їх взаємодію з іншими об'єктами на сцені.
4. Тестування та налагодження

Регулярне тестування та налагодження є важливими етапами розробки гри. Це допомагає виявити помилки та покращити ігровий процес.

1. Тестування гри:
   - Проводити тестування на різних етапах розробки.
   - Виявляти та виправляти помилки в коді та анімаціях.
2. Оптимізація продуктивності:
   - Перевірити продуктивність гри на різних пристроях.
   - Оптимізувати графіку та коди для забезпечення плавного геймплею.3. Збір відгуків:
   - Попросити друзів або колег протестувати гру та надати зворотній зв'язок.
   - Внести необхідні зміни на основі отриманих відгуків.

Створення 2D-платформера за допомогою шаблону 2D Platformer Template від Unity є захоплюючим та навчальним процесом. Використання готових компонентів та механік дозволяє зосередитися на творчій складовій та реалізації унікальних ідей. Завдяки Unity навіть початківці можуть створювати якісні та цікаві ігри, що привертають увагу гравців. Цей проект стане корисним

керівництвом для всіх, хто прагне розпочати свою подорож у світі розробки відеоігор.

Розділ 3.Розробка ігрових механік

Розробка ігрових механік є ключовим етапом у створенні відеоігор. Вона визначає, як гравці будуть взаємодіяти з грою, які виклики вони зустрічатимуть та як вони будуть досягати своїх цілей. Ігрові механіки включають різні аспекти, від руху персонажа до системи бою, вирішення головоломок та соціальних взаємодій. Вони формують основу ігрового процесу та визначають унікальність кожної гри. У цьому розділі ми розглянемо процес розробки ігрових механік, їх види та етапи реалізації.

3.1. Основи розробки ігрових механік

Ігрові механіки складаються з набору правил та систем, що визначають, як гра функціонує. Основні кроки розробки включають:

1. Ідея та концепція:
   - Визначення основної концепції гри.
   - Розробка головної механіки, яка буде визначати геймплей.
2. Дизайн документація:
   - Створення докладного опису ігрових механік.
   - Включення опису персонажів, ворогів, об'єктів, а також їх взаємодії.
3. Прототипування:
   - Створення базових прототипів для тестування ідей.
   - Перевірка основних механік на практиці та внесення коригувань.

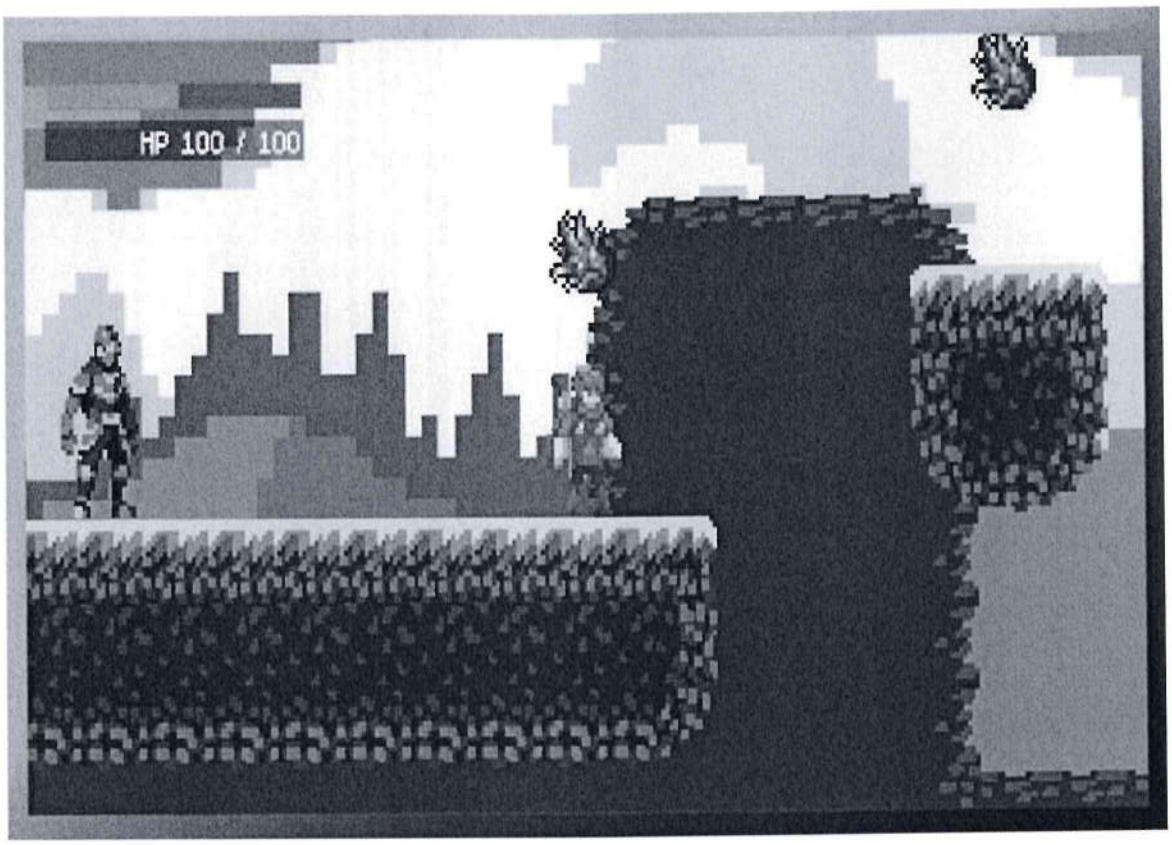

Рис.2. Другий приклад 2D гри.

3.2.Види ігрових механік

Існує кілька основних видів ігрових механік, які можуть бути використані в іграх. Серед них:

1. Рух та управління:
   - Основні механіки руху персонажа, такі як ходьба, біг, стрибки.
   - Механіки управління, включаючи перемикання між персонажами або об'єктами.
2. Бойова система:
   - Механіки атаки, захисту та уникнення ударів. - Різні типи зброї та їх використання.
3. Взаємодія з об'єктами:
   - Взаємодія з предметами на рівнях, наприклад, підняття, переміщення, активація.
   - Механіки вирішення головоломок.
4. Система прогресії:
   - Система накопичення очок досвіду, рівнів та навичок. - Відкриття нових здібностей та предметів.

5. Соціальні взаємодії:
   - Взаємодія з іншими гравцями або NPC (неігровими персонажами).
   - Система діалогів та прийняття рішень.

3.3. Процес розробки ігрових механік

Розробка ігрових механік включає кілька етапів, від початкового задуму до кінцевої реалізації:

1. Концептуальний дизайн:
   - Визначення ключових механік гри.
   - Створення концептуальних схем та документів.
2. Технічне проектування:
   - Розробка технічних специфікацій та алгоритмів
   - Вибір інструментів та технологій для реалізації механік.
3. Прототипування та тестування:
   - Створення прототипів для перевірки механік.
   - Тестування на цільовій аудиторії та збір відгуків.
4. Реалізація та інтеграція:
   - Реалізація механік у коді гри.
   - Інтеграція різних механік між собою.
5. Тестування та налагодження:
   - Детальне тестування механік у різних ситуаціях.
   - Виправлення помилок та оптимізація продуктивності.

3.4.Приклади ігрових механік

Розглянемо кілька прикладів ігрових механік, реалізованих у відомих іграх:

1. Система паркуру в Assassin's Creed:
   - Механіка дозволяє гравцю легко переміщуватися по будівлях та ландшафту.
   - Включає автоматичне визначення точок захоплення та плавний перехід між ними.
2. Бойова система в Dark Souls:
   - Висока складність та тактичний підхід до бою.
   - Включає різні типи зброї, магії та маневрування.

3. Портальна механіка в Portal:

- Використання портативного пристрою для створення порталів.
- Гравець вирішує головоломки, переміщуючись через портали.

3.5.Виклики у розробці ігрових механік

Розробка ігрових механік може зустрічатися з кількома викликами:

1. Балансування:

- Забезпечення рівноваги між складністю та доступністю.
- Балансування різних елементів гри, таких як зброя, вороги, завдання.

2. Унікальність:

- Створення унікальних механік, які відрізняють гру від інших.
- Впровадження інноваційних ідей.

3. Продуктивність:

- Оптимізація механік для забезпечення плавного геймплею.
- Зниження навантаження на апаратні ресурси.

4. Взаємодія з гравцями:

- Збір відгуків від гравців та внесення необхідних змін. - Врахування різних стилів гри та вподобань аудиторії.

Розробка ігрових механік є складним та багатогранним процесом, що вимагає креативності, технічних знань та здатності до аналізу. Від якості ігрових механік залежить успіх гри та задоволення гравців. Важливо не тільки розробити цікаві та унікальні механіки, але й забезпечити їхню збалансованість та оптимізацію. Розробка ігрових механік – це мистецтво, яке поєднує технічні навички та креативний підхід, що дозволяє створювати захоплюючі та незабутні ігрові світи.

Розділ 4.Тестування та налагодження гри

Тестування та налагодження є важливими етапами розробки відеоігор, які забезпечують якість, стабільність та загальну задоволеність гравців. Ці процеси включають виявлення і виправлення помилок, перевірку

функціональності та забезпечення відповідності гри технічним вимогам. Тестування допомагає знайти проблеми, які можуть вплинути на геймплей, продуктивність та користувацький досвід, тоді як налагодження забезпечує їх усунення. У цьому розділі ми розглянемо основні аспекти тестування та налагодження гри, методи тестування, інструменти та процеси, які допомагають створити якісний продукт.

4.1.Основи тестування та налагодження гри

Тестування та налагодження гри складається з кількох ключових кроків:

1. Планування тестування:
   - Визначення цілей та завдань тестування.
   - Розробка стратегії тестування, включаючи вибір методів та інструментів.
2. Розробка тестових сценаріїв:
   - Створення сценаріїв тестування для перевірки різних аспектів гри.
   - Включення як функціональних, так і нефункціональних тестів.
3. Автоматизоване та ручне тестування:
   - Використання автоматизованих тестів для перевірки повторюваних завдань.
   - Проведення ручного тестування для виявлення специфічних проблем.
4. Збір та аналіз даних:
   - Реєстрація результатів тестування.
   - Аналіз даних для виявлення тенденцій та основних проблем.
5. Налагодження:
   - Виправлення виявлених помилок.
   - Оптимізація продуктивності та стабільності гри.

4.2.Види тестування

Існує кілька основних видів тестування, які використовуються в процесі розробки гри:

1. Функціональне тестування:

- Перевірка основних функцій гри, таких як рух персонажа, бойова система, інтерфейс користувача.

- Тестування відповідності гри технічним вимогам та дизайну.

2. Нефункціональне тестування:

- Перевірка продуктивності, стабільності та сумісності гри. - Включає тестування на різних платформах та пристроях.

3. Регресійне тестування:

- Повторне тестування після внесення змін до гри для переконання, що нові зміни не викликали нових помилок.

- Включає автоматизовані тести для перевірки ключових функцій.

4. Бета-тестування:

- Проведення тестування з участю реальних користувачів для отримання відгуків.

- Виявлення проблем, які можуть бути пропущені внутрішніми тестерами.

5. Тестування на відповідність стандартам:

- Перевірка гри на відповідність стандартам та вимогам платформ (наприклад, PlayStation, Xbox).

- Включає тестування на відповідність політикам магазинів додатків.

4.3.Процес налагодження

Процес налагодження включає кілька етапів:

1. Виявлення помилок:

- Використання тестових сценаріїв для виявлення проблем. - Аналіз зворотного зв'язку від тестерів та користувачів.

2. Виправлення помилок:

- Виправлення знайдених проблем у коді гри.

- Проведення регресійного тестування після виправлення.

3. Оптимізація продуктивності:

- Оптимізація коду для покращення продуктивності гри. - Зниження навантаження на ресурси пристроїв.

4. Верифікація виправлень:

- Повторне тестування після внесення виправлень для переконання у їх ефективності.

- Збір нових даних для подальшого аналізу.

5. Випуск оновлень:

- Випуск патчів та оновлень для гри.

- Постійне підтримання та оновлення гри після її виходу.

4.5 Інструменти для тестування та налагодження

Існує багато інструментів, які допомагають в процесі тестування та налагодження гри:

1. Автоматизовані тестові фреймворки:

- Використання фреймворків для автоматизованого тестування, таких як Selenium, Appium, TestComplete. - Спрощення процесу регресійного тестування.

2. Інструменти для профілювання:

- Використання інструментів для аналізу продуктивності, таких як Unity Profiler, Unreal Engine Profiler.

- Виявлення вузьких місць у продуктивності гри.

3. Баг-трекінгові системи:

- Використання систем для управління помилками, таких як JIRA, Bugzilla.

Організація процесу виявлення та виправлення помилок.

Розробка 2D платформера — це складний і багатогранний процес, який вимагає поєднання технічних навичок, творчого підходу і систематичного

підходу до вирішення проблем. Кожен етап розробки гри, від концепції до випуску, є важливим і вимагає уваги до деталей. У цьому розділі ми підведемо підсумки роботи, оцінюючи результати кожного етапу та визначаючи ключові фактори успіху.

4.4.Розробка концепції та дизайну гри

На початковому етапі розробки 2D платформера важливо було створити концепцію гри, яка включала б основну ідею, сюжет, персонажів і механіки геймплею. Ми визначили цільову аудиторію і врахували їхні очікування від гри.

Ключові досягнення:

- Створення унікальної концепції: Розробка унікального сюжету та персонажів, які зацікавлять гравців.
- Дизайн рівнів: Створення захоплюючих і викликових рівнів, які мотивують гравців до подальшої гри.
- Візуальний стиль: Вибір візуального стилю, який відповідає загальній атмосфері гри та привертає увагу гравців.

4.5.Технічна реалізація

Технічна реалізація включала розробку основних компонентів гри, таких як рух персонажа, фізика, анімація та інтерфейс користувача. Ми використовували Unity як основний інструмент для розробки, що дозволило швидко створювати і тестувати різні елементи гри.

Ключові досягнення:

- Програмування основних механік: Реалізація руху персонажа, стрибків, атак та інших механік геймплею. - Фізика гри: Інтеграція фізики для створення реалістичних взаємодій у світі гри.
- Анімації: Створення плавних і реалістичних анімацій для персонажів та об'єктів.

5.6.Розробка ігрових механік

Ігрові механіки визначають, як гравці взаємодіють з грою і як вони досягають поставлених цілей. Ми зосередилися на створенні захоплюючих та інтуїтивно зрозумілих механік, які роблять гру цікавою та викликовою.

Ключові досягнення:

-Геймплейні механіки: Розробка механік, які сприяють глибокому зануренню у гру, таких як збирання бонусів, використання різних типів атак і перешкод.

- Баланс складності: Забезпечення рівня складності, який є достатньо викликовим для досвідчених гравців, але не надто важким для новачків.

- Інтерактивність: Додавання інтерактивних елементів, які роблять світ гри більш динамічним і цікавим.

5.5.Тестування та налагодження

Тестування та налагодження були критичними етапами, які забезпечили високу якість та стабільність гри. Ми провели кілька раундів тестування, щоб виявити і виправити помилки, оптимізувати продуктивність та покращити користувацький досвід.

Ключові досягнення:

- Виявлення та виправлення помилок: Виявлення багів та їх оперативне виправлення для забезпечення безперебійної роботи гри.

- Оптимізація продуктивності: Підвищення продуктивності гри на різних платформах шляхом оптимізації коду та ресурсів.

- Тестування користувачами: Проведення бета-тестування з участю реальних користувачів для отримання зворотного зв'язку та подальшого покращення гри.

5.6.Випуск гри та маркетинг

Остаточним етапом був випуск гри та її просування на ринку. Ми розробили маркетингову стратегію, спрямовану на залучення максимальної кількості гравців та підвищення впізнаваності гри.

Ключові досягнення:

- Стратегія випуску: Підготовка до випуску гри на різних платформах, включаючи Steam, App Store та Google Play. - Просування гри: Використання соціальних медіа, трейлерів, демонстрацій та прес-релізів для залучення уваги до гри. - Підтримка гравців: Створення системи підтримки гравців та регулярне оновлення гри для підтримання інтересу та задоволення гравців. до гри. - Підтримка гравців: Створення системи підтримки гравців та регулярне оновлення гри для підтримання інтересу та задоволення гравців.

Висновок

У даній дипломній роботі була розглянута розробка гри у жанрі ранер з використанням сучасних інструментів і технологій, зокрема Unity. Метою роботи було створення повноцінного ігрового продукту, який би відповідав сучасним вимогам до якості та геймплейних механік. Протягом роботи було охоплено всі етапи розробки: від концептуального дизайну та підготовки середовища до реалізації ігрових механік, тестування та фінальної оптимізації.

Основні етапи розробки

На початковому етапі роботи було проведено аналіз жанру ранер, його особливостей та ключових елементів, які визначають успіх подібних ігор. Було визначено, що ранер відрізняється динамічним геймплеєм, простотою управління та високою реіграбельністю. Ці аспекти стали основними орієнтирами при розробці концепції гри.

Наступним кроком було створення детального плану розробки, включаючи визначення функціональних вимог до гри, дизайн рівнів, вибір графічного стилю та звукового супроводу. Важливим аспектом було визначення цільової аудиторії та врахування її потреб та очікувань.

Для реалізації проекту було обрано платформу Unity, яка забезпечує широкий спектр можливостей для розробки 2D- та 3D-ігор. Серед її переваг можна виділити зручний інтерфейс, багатий набір інструментів, а також потужні можливості для оптимізації та кросплатформеності.

Розробка гри

Основна частина роботи полягала у безпосередній розробці гри. Було створено декілька рівнів, кожен з яких мав унікальний дизайн та різноманітні перешкоди. Використовуючи Unity, було реалізовано основні механіки гри, включаючи управління персонажем, систему очок, генерацію перешкод та бонусів, а також інтеграцію звукових ефектів і музичного супроводу.

Особлива увага приділялась оптимізації ігрового процесу та забезпеченню плавності анімацій. Використання компонентного підходу

дозволило створити гнучку та розширювану архітектуру гри, що полегшує її подальший розвиток та оновлення.

Тестування та налагодження

Завершальним етапом розробки стало тестування гри. Було проведено внутрішнє та зовнішнє тестування з метою виявлення та усунення багів, покращення продуктивності та забезпечення стабільної роботи гри на різних пристроях. Зворотний зв'язок від тестувальників дозволив внести необхідні корективи та вдосконалити геймплей.

Досягнуті результати

У результаті проведеної роботи було створено повноцінну гру у жанрі ранер, яка відповідає сучасним вимогам до якості та геймплейних механік. Гра має привабливий візуальний стиль, динамічний та захоплюючий геймплей, а також високу реіграбельність завдяки різноманітності рівнів та перешкод.

Надалі можливо розширення функціональності гри, додавання нових рівнів, персонажів, бонусів та інших елементів, що зроблять гру ще цікавішою та привабливішою для користувачів. Також є потенціал для монетизації проекту через рекламу та внутрішньоігрові покупки, що дозволить підтримувати та розвивати гру у майбутньому.

Література

1. "Unity Documentation." Unity Technologies. https://docs.unity3d.com/Manual/index.html
2. Johnson, Ben. "Developing 2D Games with Unity: Independent Game Programming with C#." CRC Press, 2019.
3. Hocking, Jon. "Unity in Action: Multiplatform Game Development in C# with Unity 5." Manning Publications, 2015. 4. Gambhir, Sanjay. "Mastering Unity 2D Game Development." Packt Publishing, 2014.
5. Goldstone, Will. "Unity 2017 Game Development Essentials." Packt Publishing, 2017.
6. "Best Practices for Performance Optimization in Unity." Unity Technologies. [https://unity.com/how-to/best-practices-optimizing-performanceunity](https://unity.com/how-to/best-practices-optimizing-performance-unity)
7. "Unity Learn: Create with Code." Unity Technologies. https://learn.unity.com/course/create-with-code
8. "Introduction to Game Design, Prototyping, and Development." Pearson, 2017.
9. "2D Game Kit." Unity Technologies. https://learn.unity.com/project/2d-game-kit
10. "Understanding 2D Animation Rigging and Animation in Unity." Unity Technologies. https://docs.unity3d.com/Manual/2DAnimation.html
11. Murray, Charles. "Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for Your Game." CRC Press, 2018.
12. "Physics in Unity: Rigidbodies, Colliders, and Joints." Unity Technologies.

https://docs.unity3d.com/Manual/PhysicsSection.html

13. "Debugging and Troubleshooting Unity Games." Unity Technologies. https://docs.unity3d.com/Manual/Debugging.html

14. "Game Testing: All in One." CRC Press, 2017.

15. "Building a Successful 2D Game in Unity: From Concept to Publishing." Unity Technologies. https://learn.unity.com/tutorial/building-a-successful-2d-game

16. Preez,S. "Efficient 2D Game Development with Unity and C#." Packt Publishing, 2018.

17. "Unity User Manual: Graphics Overview." Unity Technologies. https://docs.unity3d.com/Manual/Graphics.html

18. "Unity Scripting API." Unity Technologies. https://docs.unity3d.com/ScriptReference/

19. "Using Unity's Built-in 2D Game Kit." Unity Technologies. https://learn.unity.com/project/2d-game-kit

20. "Monetizing Your Mobile Game with Unity Ads." Unity Technologies. [https://unity.com/solutions/unity-ads](https://unity.com/solutions/ unity-ads)

Додаток А

Код:

```
using System.Collections;
using System.Collections.Generic; using
Platformer.Mechanics;

using UnityEditor;
using UnityEngine;
namespace Platformer
{
    [CustomEditor(typeof(PatrolPath))]

    public class PatrolPathGizmo : Editor
    {
        public void OnSceneGUI()

        {
            var path = target as PatrolPath;
            using (var cc = new
EditorGUI.ChangeCheckScope())
            {
                var sp =
path.transform.InverseTransformPoint(Handles.Posit
ionHandle(path.transform.TransformPoint(path.start
```

```
Position), path.transform.rotation));
                var ep =
path.transform.InverseTransformPoint(Handles.Posit


ionHandle(path.transform.TransformPoint(path.endPo


                    if (cc.changed)
                            {
                        sp.y = 0;
                        ep.y = 0;
                    path.startPosition = sp;
```

```
                    path.endPosition = ep;
                }
            }
                Handles.Label(path.transform.position,
(path.startPosition -
path.endPosition).magnitude.ToString());
        }


        [DrawGizmo(GizmoType.Selected |
```

```
GizmoType.NonSelected)]

        static void OnDrawGizmo(PatrolPath path,
GizmoType gizmoType)
        {
            var start =
path.transform.TransformPoint(path.startPosition);

            var end =
path.transform.TransformPoint(path.endPosition);

            Handles.color = Color.yellow;
              Handles.DrawDottedLine(start, end, 5);

             Handles.DrawSolidDisc(start,
path.transform.forward, 0.1f);
             Handles.DrawSolidDisc(end,
path.transform.forward, 0.1f);

        }
    }
```

Додаток В

```
using System.Linq;
using Unity.Play.Publisher.Editor;


using UnityEngine;

namespace Unity.Tutorials
{



    /// <summary>
    /// Contaisn all the callbacks needed for the
Build And Publish tutorial
    /// </summary>
     [CreateAssetMenu(fileName = "PublishCriteria",


menuName = "Tutorials/Microgame/PublishCriteria")]
    class PublishCriteria : ScriptableObject


    {
        static PublisherWindow publisherWindow;
        public bool
IsNotDisplayingFirstTimeInstructions()
```

```
        {
            if (!IsWebGLPublisherOpen()) { return
return
(!string.IsNullOrEmpty(publisherWindow.CurrentTab)
&& publisherWindow.CurrentTab !=

PublisherWindow.TabIntroduction);
}

public bool IsUserLoggedIn()

{
if (!IsWebGLPublisherOpen()) { return
false; }

            return (publisherWindow.CurrentTab !=
PublisherWindow.TabNotLoggedIn);
                                                }

                public bool IsBuildBeingUploaded()
                                                {
            if (!IsWebGLPublisherOpen()) { return
                                        false; }
                                          switch
(PublisherUtils.GetCurrentPublisherState(publisher
                                        Window))
```

```
                                                    {
                        case PublisherState.Upload:
                    case PublisherState.Process:
                                        return true;
                                    default: break;
                                            } return
!string.IsNullOrEmpty(PublisherUtils.GetUrlOfLastP
                    ublishedBuild(publisherWindow));
                                                    }
```

```
        public bool IsBuildPublished()
        {

                if (!IsWebGLPublisherOpen()) { return

false; }
            return

!string.IsNullOrEmpty(PublisherUtils.GetUrlOfLastP
ublishedBuild(publisherWindow));

        }

            public bool AtLeastOneBuildIsRegistered()
```

```
        {


                if (!IsWebGLPublisherOpen()) { return
false; }
            switch
(PublisherUtils.GetCurrentPublisherState(publisher
Window))


            {
                  case PublisherState.Zip:



                case PublisherState.Upload:

                return true;

                default: break;

                }

                int availableBuilds =

                PublisherUtils.GetAllBuildsDirectorie
                s()
```

```
            .Where(p => (p != string.Empty)



            &&
            PublisherUtils.BuildIsValid(p)).Count
            ();
            return availableBuilds > 0;

            }



            bool IsWebGLPublisherOpen()

            {

            if (publisherWindow) { return true; }

            publisherWindow =

            PublisherWindow.FindInstance();

            return false;

    }
            }
```

Додаток С

```
using UnityEngine;
using Unity.Tutorials.Core.Editor;


using UnityEditor;

namespace Unity.Tutorials
{


    /// <summary>
      /// Implement your Tutorial callbacks here.
    /// </summary>
    public class TutorialCallbacks :


ScriptableObject
    {



        public GameObject TokenToSelect;

        /// Selects a GameObject in the scene,
        marking it as the active object for selection
        /// </summary>
```

```
        /// <param
        name="futureObjectReference"></param>
        public void
        SelectSpawnedGameObject(FutureObjectReference
        futureObjectReference)
        { if
        (futureObjectReference.SceneObjectReference ==
        null) { return; }

        Reference.ReferencedObjectAsGameObject);
        }

        public void SelectGameObject(GameObject
        gameObject)
        {
        if (!gameObject) { return; }
        Selection.activeObject = gameObject;
        }

        public void SelectToken()
        {
        if (!TokenToSelect)
TokenToSelect =
```

```
GameObject.FindGameObjectWithTag("TutorialRequirem

ent");

if (!TokenToSelect)

{
```

```
Debug.LogErrorFormat("A


TokenInstance with the tag '{0}' must be in the


scene in order to make this tutorial work


properly. Please add the tag {0} to one of your tokens
in the scene", "TutorialRequirement");
```

```
return;
}
SelectGameObject(TokenToSelect);
}

public void SelectMoveTool()
{

Tools.current = Tool.Move;
}


public void SelectRotateTool()
{

Tools.current = Tool.Rotate;
}
public void StartTutorial(Tutorial
tutorial)
```

```
TutorialWindowUtils.StartTutorial(tutorial);
}
}
}
```