МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки

«До захисту допущена»
Зав. кафедри комп'ютерних наук
та прикладної математики
д.т.н., професор Турбал Ю.В.
«____» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

Проєктування та розробка гри "Створення гри на рушії Unity з використанням процедурної генерації ландшафту"

Виконав: Уксусов Іван Сергійович

студент навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки

група КН-41

_____

Керівник: к.ф.-м.н., доцент кафедри комп'ютерних наук та прикладної математики, Степанченко О.М.

_____

Рівне-2023

Зміст

Національний університет водного господарства та природокористування

НН інститут **автоматики, кібернетики та обчислювальної техніки**

Кафедра **комп'ютерних наук та прикладної математики**

Освітньо-кваліфікаційний рівень **бакалавр**

Галузь знань 12 інформаційні технології

Спеціальність **122 Комп'ютерні науки**

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

та прикладної математики

д.т.н., професор Турбал Ю.В.

«____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Уксусову Івану Сергійовичу

1. Тема роботи: Створення гри на рушії Unity з використанням процедурної генерації ландшафту

керівник роботи: Степанченко Ольга Миколаївна, к.ф.-м.н., доцент кафедри комп'ютерних наук та прикладної математики,

затверджені наказом вищого навчального закладу від "__"_____2023 року № __

2. Термін подання роботи студентом: 5 червня 2023 р.

3. Вихідні дані до роботи: ринок ігор

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): розділ 1 - дослідження проблеми, розділ 2 - розробка програмного рішення, розділ - 3 Оптимізація програмного продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): візуалізація основних етапів створення програмного продукту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	Степанченко О. М.		
Розділ 2	Степанченко О. М.		
Розділ 3	Степанченко О. М.		

7. Дата видачі завдання 30 жовтня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
	Дослідження ринку	30.02.2023 - 13.03.2023	
	Дослідження інструментів для розробки	13.03.2023 - 13.04.2023	Обрати рушій та SDK для роботи з VR
	Розробка прототипу	13.04.2023 -	

	додатку	10.05.2023	
	Тестування та доопрацювання	10.05.2023 - 20.05.2023	

Студент

_____ _____

підпис

прізвище та ініціали

Керівник роботи

_____ _____

підпис

прізвище та ініціали

\

РЕФЕРАТ

Кваліфікаційна робота: 50 с., 9 рисунків, 6 джерел, 1 додаток.

Мета роботи:

- провести дослідження ринку ігрової індустрії;
- розробити гіперказуальну гру на платформу Android;

Проведено аналіз ринку та тенденцій в ігровій індустрії. Обрано ідею, рушій та спосіб реалізації проєкту. Розроблено гру яка готова до публікації на різноманітних платформах

Ключові слова: ІГРИ, РОЗРОБКА, HYPERCASUAL, 3D, UNITY, ВІДЕОІГРИ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SDK - software development kit (набір із засобів розробки)

ВСТУП

У сучасній індустрії відеоігор процедурна генерація стала надзвичайно популярною та потужною технікою, яка дозволяє створювати нескінченні, унікальні та захоплюючі рівні гри. Цей проєкт-гра, розроблений на платформі Unity, поєднує два види процедурної генерації: сіткову генерацію рівня та генерацію ландшафту для кожної клітинки сітки.

Основна мета цього проєкту-гри полягає в тому, щоб досліджувати та розробляти ефективні методи процедурної генерації, які дозволяють створювати різноманітні рівні для забезпечення цікавого ігрового досвіду з використанням мінімум людського ресурсу. Сіткова генерація карт забезпечує створення базової структури рівня, а генерація ландшафту дозволяє деталізувати ігровий ландшафт.

У цьому проєкті будуть використані різноманітні алгоритми та техніки процедурної генерації, такі як використання випадкових чисел та кривих безьє Крім того, проєкт буде реалізований на платформі Unity, що є однією з найпопулярніших інструментальних систем для розробки ігор. Unity надає широкі можливості для реалізації процедурної генерації та візуалізації створених рівнів. Він дає можливість працювати зі зручним інтерфейсом, потужними засобами програмування та візуалізації для реалізації різних цікавих ідей та дослідження можливості процедурної генерації в ігровому середовищі.

Розділ 1 Алгоритми моделювання та генерації у сфері відеоігор

1.1 Поняття генерації та моделювання у відеоіграх

Відеогра (відеогра) - це електронна гра, яка відтворюється на комп'ютері або іншій гральній платформі, такій як консоль або мобільний пристрій. Вона залучає гравця до віртуального світу та надає йому можливість взаємодіяти з ним за допомогою контролера, клавіатури, миші або сенсорного екрану.

Відеогри можуть мати різні жанри та стилі геймплею, включаючи пригоди, екшн, стратегії, рольові ігри, спортивні симулятори, головоломки та багато іншого. Вони можуть бути одиночними, де гравець виконує завдання або просувається по сюжету в одиночку, або мультиплеєрними, де кілька гравців змагаються або співпрацюють в онлайн - або офлайн режимі.

Відеогри можуть мати великий рівень складності та деталізації. Вони часто включають в себе графічні ефекти, звукові ефекти, музику та діалоги, щоб створити більш імерсивний досвід для гравця. Багато відеоігор мають також прогресію, системи нагород та досягнень, які стимулюють гравців до досягнення нових цілей та покращення своїх навичок.

Як і в будь-якій іншій програмі, в іграх можуть використовуватись безліч алгоритмів, які стають інструментом розробки для реалізації генеративних світів, поведінки віртуальних істот та створення різного роду анімацій. Одними з основних алгоритмів у відеоіграх є алгоритми для генерації та моделювання поведінки ігрових об’єктів.

Генерація (процедурна генерація) відноситься до створення віртуальних елементів, таких як місцевості, умовні пейзажі, рівні або текстури, за допомогою алгоритмів та правил. Замість того, щоб створювати кожен елемент вручну, розробники використовують програмні алгоритми, щоб автоматично генерувати

ці елементи. Це дає можливість створювати безкінечні або рандомізовані світи, які кожного разу, коли гра запускається, виглядають по-різному.

Своєю чергою моделювання поведінки належать до процесу створення правил, алгоритмів та логіки, які визначають поведінку об'єктів, персонажів або інших елементів у відеоіграх. Цей процес дозволяє розробникам контролювати та керувати тим, як об'єкти взаємодіють один з одним та з оточенням гри.

1.2 Види та приклади алгоритмів генерації

У відеоіграх використовується різноманітні генераційні методи та техніки для створення різних елементів гри. Ось деякі види та приклади генерації, що застосовуються в ігровій індустрії:

1. Процедурна генерація місцевості: Цей тип генерації використовується для створення випадкових або безкінечних світів гри(рис 1.1). Великі ландшафти, пейзажі, гори, ліси, пустелі та інші місцевості можуть бути створені автоматично з використанням алгоритмів та правил. Наприклад, гра Minecraft використовує процедурну генерацію для створення світу з блоків, де кожен світ генерується випадковим чином при кожному новому запуску гри.

Рис 1.1 Приклад гри з процедурною генерацією

2. Процедурна генерація рівнів: Цей тип генерації використовується для створення різноманітних рівнів або різновидів рівнів у грі. Рівні можуть бути генеровані автоматично з використанням алгоритмів або базуються на параметрах, які встановлюють розробники. Наприклад, у грі Spelunky кожен рівень генерується випадковим чином, створюючи унікальний досвід гри кожного разу.

3. Генерація персонажів: Цей тип генерації використовується для створення випадкових або унікальних персонажів у грі. Атрибути персонажів, такі як вигляд, характеристики, навички та інші властивості, можуть бути генеровані з використанням алгоритмів або систем, які враховують різні фактори. Наприклад, у грі The Elder Scrolls V: Skyrim система генерації персонажів дозволяє гравцю створювати свого унікального героя шляхом вибору різних атрибутів та характеристик.

4. Генерація ворогів та противників: У багатьох відеоіграх вороги та противники можуть бути генеровані автоматично або з використанням заранее визначених правил. Це дозволяє створювати різноманітних противників з різними характеристиками та властивостями. Наприклад, у грі Diablo IV вороги генеруються автоматично з використанням алгоритмів, що дозволяє створювати велику кількість різних монстрів для гравця.

Рис. 1.2 Приклад генерації ворогів

5. Генерація завдань та контенту: Деякі ігри використовують генерацію для створення різних завдань, місій або контенту для гравця. Це дозволяє створювати безкінечний потік викликів та варіацій у грі. Наприклад, у грі No Man's Sky галактика, планети, створення та завдання генеруються автоматично, що дає гравцеві безкінечний космічний світ для дослідження.

Це лише кілька прикладів, як генерація використовується у відеоіграх. В ігровій індустрії існує безліч технік і методів генерації, що допомагають розширити ігровий досвід та забезпечити різноманітність та повторюваність гри.

1.3 Створення алгоритмів генерації для відеоігор

Математичні алгоритми відіграють важливу роль у генерації різних елементів у відеоіграх. Деякі поняття та алгоритми, що часто використовуються для генерації у відеоіграх включають:

1. Випадковість: Випадковість є ключовим поняттям у генерації у відеоіграх. Генератори випадкових чисел (ГВЧ) використовуються для створення випадкових значень, які впливають на різні аспекти гри, включаючи

місцевість, розміщення об'єктів, поведінку персонажів та інше. Алгоритми ГВЧ гарантують, що генеровані значення є статистично розподіленими і випадковими.

2. Шум Перліна (Perlin Noise): Шум Перліна є алгоритмом генерації псевдовипадкових чисел, який використовується для створення нерегулярних та органічних текстур, місцевостей та інших елементів. Він широко використовується для створення рельєфу, хвильових ефектів, хмар, ландшафтів та інших природних форм.

3. Карти шуму (Noise Maps): Карти шуму є двовимірними або тривимірними структурами даних, що містять генеровані випадкові значення. Вони використовуються для створення різних властивостей у відеоіграх, таких як висоти, текстури, густота розміщення об'єктів і т. д. Карти шуму можуть бути створені за допомогою алгоритмів шуму Перліна, Фрактального шуму або інших методів.

4. Деталізованість (Level of Detail, LOD): LOD використовується для оптимізації процесу генерації у відеоіграх. Він визначає рівень деталізації об'єктів або місцевості, який залежить від відстані гравця до них. Це дозволяє зменшити обчислювальну потужність, зберігаючи при цьому якість гри. Алгоритми LOD включають зменшення кількості полігонів, заміну деталей на текстури, використання простіших моделей та інші методи.

5. Алгоритми пошуку шляху (Pathfinding): Алгоритми пошуку шляху використовуються для визначення оптимального шляху персонажа у грі.

Вони враховують перешкоди, терен та інші фактори для знаходження найкоротшого шляху від пункту А до пункту Б.

1.4 Алгоритми на основі кривих Безьє

Алгоритм кривих Безьє є одним з найпоширеніших методів для представлення та рендерингу гладких кривих у комп'ютерній графіці. Цей алгоритм був розроблений французьким інженером П'єром Безьє у 1962 році і згодом став важливою складовою частиною багатьох графічних програм та пакетів, включаючи відеоігри.

Основна ідея кривих Безьє полягає в тому, що крива представляється як комбінація контрольних точок, відомих як точки Безьє. Ці точки визначають форму та характеристики кривої. Зазвичай використовуються криві Безьє другого або третього порядку, які визначаються чотирма або п'ятьма точками Безьє відповідно.

Основний алгоритм кривих Безьє для рендерингу полягає в інтерполяції точок по кривій між контрольними точками. Для цього використовується параметр t, який змінюється від 0 до 1. Значення t вказує на прогрес по кривій, де t = 0 відповідає початковій точці Безье, а t = 1 відповідає кінцевій точці Безье.

Процес рендерингу кривої Безье включає обчислення лінійних комбінацій координат контрольних точок з використанням поліноміальних функцій Бернштейна. Ці поліноміальні функції визначаються порядком кривої та значенням параметра t. Після обчислення координат точок по кривій, їх можна використовувати для рендерингу на екрані або подальшої обробки.

Одна з переваг кривих Безье полягає у їх гладкості та контролі. Зміна положень контрольних точок дозволяє варіювати форму кривої, що робить їх дуже корисними для створення різних ефектів, таких як анімація руху, криві об'єму, форми об'єктів та багатьох інших.

1.5 Використання кривих Безьє у відеоіграх

Криві Безьє мають широке застосування у відеоіграх і використовуються для різних цілей. Кілька основних використань кривих Безьє у відеоіграх це:

1. Анімація руху: Криві Безьє дозволяють плавно анімувати рух об'єктів та персонажів у відеоіграх. Вони можуть використовуватись для визначення траєкторій руху, зміни швидкості та позиції об'єктів у просторі. Це дозволяє створювати реалістичні та плавні анімації, які підкреслюють гнучкість та органічність руху персонажів.

2. Контроль камери: Криві Безьє можуть бути використані для керування рухом камери відеоігри. Вони дозволяють плавно переходити між різними ракурсами, створюючи кінематографічний ефект. Камера може рухатися по певній траєкторії, змінювати свою позицію та орієнтацію, що створює цікаві та динамічні кадри для гравця.

3. Фізичні ефекти: Криві Безьє можуть бути використані для моделювання різних фізичних ефектів у відеоіграх. Наприклад, їх можна застосовувати для створення ефектів анімації волосся, пламені, води, диму тощо. Криві Безьє дозволяють точно контролювати форму та поведінку таких ефектів, що приносить більшу реалістичність та деталізацію графіки.

4. Інтерполяція параметрів: Криві Безьє можуть бути використані для інтерполяції параметрів об'єктів у відеоіграх. Наприклад, вони можуть контролювати зміну розміру, колірів, прозорості або будь-яких інших властивостей об'єктів у залежності від часу або інших факторів. Це дозволяє створювати

плавні та динамічні переходи між різними станами об'єктів, такі як зміна розміру або анімація зміни кольору.

Криві Безьє є потужним інструментом для контролю та анімації різних аспектів відеоігор. Вони дозволяють створювати реалістичні та ефективні графічні ефекти, покращуючи враження гравця та створюючи динамічний світ відеоігри.

Розділ 2 Аналіз інструментів на способів реалізації ігрового проєкту

2.1 Аналіз ринку мобільних відеоігор

Мобільні ігри є одним з найбільших сегментів галузі відеоігор, що розвиваються. З появою смартфонів та планшетів, доступних для широкої аудиторії, мобільні ігри стали надзвичайно популярними та економічно вигідними продуктами. Аналіз ринку мобільних ігор допоможе нам зрозуміти тенденції, перспективи та конкурентну ситуацію в цій галузі.

Розмір ринку: Мобільні ігри представляють великий та постійно зростаючий ринок. За останні роки спостерігається значний приріст кількості користувачів мобільних пристроїв та їх активного залучення до ігор. За даними дослідницьких компаній, обсяг ринку мобільних ігор перевищує кілька мільярдів доларів і очікується подальше зростання. А велику частку ринку мобільних ігор займають саме аркадні мобільні ігри. На цей жанр націлений проєкт, що реалізується у роботі. Ігри такого жанру відрізняються відкритим геймплеєм, простотою управління та швидкими, динамічними сесіями. Деякі характеристики та особливості аркадних мобільних ігор:

- Простота управління: Аркадні ігри намагаються зробити управління максимально простим та зрозумілим для гравців. Це дозволяє новачкам швидко освоїти гру і насолоджуватися геймплеєм без складних навичок або інструкцій.
- Швидкість та рефлекси: У аркадних іграх часто присутні швидкість та вимога до гравців мати швидкі рефлекси. Це може включати

уникання перешкод, збирання об'єктів або боротьбу зі злочинцями. Гравці повинні реагувати швидко і точно, щоб вижити і отримувати високі бали.

- Різноманітність геймплею: Аркадні ігри можуть мати різноманітні механіки гри, включаючи платформери, стрілялки, летючиі ігри, перегони, лабіринти та багато інших. Це дозволяє гравцям знайти свій улюблений стиль гри і насолоджуватися різноманітними викликами та пригодами.

- Високий рівень повторності: Багато аркадних ігор базуються на принципі "грати ще раз" (play-again). Це означає, що гравці постійно повертаються до гри з метою покращення свого рекорду або досягнення нових викликів. Ця висока повторюваність стимулює конкуренцію та мотивує гравців досягти кращих результатів.

- Зручність для мобільних пристроїв: Аркадні ігри часто розробляються з урахуванням мобільних пристроїв, що дозволяє їм бути доступними іграми "на ходу". Ігри можуть мати короткі сесії, що добре підходить для коротких перерв або моментів вільного часу.

Ці особливості роблять аркадні мобільні ігри популярними серед широкої аудиторії, включаючи як новачків, так і досвідчених гравців.

Тенденції ринку: Ринок мобільних ігор постійно змінюється та вдосконалюється. Останні тенденції включають розширену реальність (AR) та віртуальну реальність (VR), мультиплеєрність, процедурну

генерацію, гібридні моделі монетизації, ігри зі стрімінгом та хмарним геймінгом, а також ігри зі спільнотами та соціальними функціями.

Монетизація мобільних ігор є важливою складовою успіху в цій галузі. Існує кілька моделей монетизації, які розробники ігор використовують для отримання доходів. Ось деякі з них:

- Безкоштовні ігри з внутрішніми покупками (Freemium): Ця модель передбачає безкоштовне скачування та грати у гру, але вона має внутрішні елементи, які можна придбати за допомогою реальних грошей. Це можуть бути різні предмети, валюта в грі, покращення персонажів або прискорення прогресу. Ця модель дозволяє гравцям насолоджуватися грою без обов'язкових витрат, але в той же час надає можливість отримувати доходи від гравців, які готові інвестувати у гру.

- Реклама: Багато мобільних ігор використовують рекламу як спосіб заробітку. Це можуть бути відеоролики, банери або виклики до дії, які відображаються під час гри. Реклама може бути показана до гри, під час перерви або як нагорода за досягнення певного рівня. Розробники отримують дохід за кожен показ або кожне натискання на рекламний матеріал.

- Підписки: Ця модель монетизації передбачає, що користувачі платять певну суму грошей на певний період, щоб мати доступ до привілеїв або додаткового контенту. Це можуть бути щотижневі, щомісячні або річні підписки, які надають користувачеві особливі переваги,

наприклад, ексклюзивний контент, безкоштовні оновлення або спеціальні пропозиції.

Цільова аудиторія: Розуміння цільової аудиторії є критичним для успішної мобільної гри. З основних аспектів потенційної аудиторії можна визначити декілька, а саме вікова група, що може складати від 10 до 25 років, інтереси до жанрів, а саме жанр аркади, та звички потенційно групи, що у випадку з обраною темою проєкту може бути бажання пограти у якось просту аркаду, коли немає чимось зайнятись.

В результаті аналізу ринку мобільних ігор було отримано важливі висновки та рекомендації, які допомогли у розробці та позиціонуванні проєктної гри. Враховуючи тенденції, конкуренцію та потреби цільової аудиторії, можна виділити наступні аспекти проєкти для реалізації:

- Простий та зрозумілий ігролад
- Високий рівень повторюваності шляхом процедурної генерації
- Зручність для мобільних пристроїв
- Безплатність кінцевого продукту

2.2 Ідея розробки та постановка вимог

Головною ідеєю цієї аркадної гри є створення водного шляху шляхом малювання лінії на квадратній мапі. Гра використовує квадратні сектори, щоб подати гравцеві платформу для творчості та вибору шляху.

Гравець повинен створити шлях, який буде прокладатися через ці сектори, а його головна мета - створити функціональний канал для переміщення води.

Гравець починає гру з порожньої квадратної мапи, яка складається з квадратних секторів. Він може вибрати будь-який сектор і розпочати малювати шлях на мапі. Гравець може змінювати напрямок шляху, обираючи сектори в мапі, і малювати лінію, яка прокладається через них.

Мета гравця - прокласти функціональний канал для води, який буде відповідати певним вимогам. Ці вимоги можуть включати, наприклад, мінімальну довжину каналу, використання певних типів секторів, забезпечення оптимального потоку води або запобігання перешкодам на шляху.

Для успішного розроблення і реалізації гри "Намалюй шлях для води" необхідно врахувати наступні функціональні вимоги:

Гравець повинен мати можливість обирати сектори на мапі та малювати лінію для прокладання шляху.

Сектори повинні відображатися у візуальному інтерфейсі гри з можливістю вибору.

Гра повинна встановлювати певні вимоги до шляху, який повинен відповідати, наприклад, довжині, використанню певних типів секторів тощо.

Канал повинен бути функціональним і забезпечувати переміщення води від початкової до кінцевої точки шляху.

Гра повинна мати візуальні ефекти, що ілюструють рух води через канал.

Фізика руху води повинна бути реалістичною і забезпечувати відтворення потоку та взаємодії з перешкодами на шляху.

Гра повинна містити різні рівні складності, які викликають зростаючі вимоги до прокладання шляху та каналу.

Гравці повинні мати можливість отримувати досягнення або нагороди за успішне проходження рівнів.

Крім функціональних вимог, необхідно врахувати такі нефункціональні вимоги:

Гра повинна бути розроблена для мобільних телефонів і підтримувати різні операційні системи, такі як Android та iOS.

Інтерфейс користувача повинен бути зручним для використання на екранах сенсорних пристроїв.

Гра повинна мати привабливу графіку, яка відображає мапу, сектори, шлях та воду.

Звукові ефекти повинні доповнювати геймплей та створювати атмосферу.

Інтерфейс гри повинен бути інтуїтивно зрозумілим та забезпечувати зручне малювання шляху на мапі.

Гра повинна мати меню, налаштування та можливість збереження прогресу гравця.

Гра повинна бути стабільною та максимізувати оптимальну продуктивність на різних пристроях.

Розмір гри повинен бути обмежений, щоб забезпечити швидке завантаження та ефективне використання пам'яті пристрою.

2.3 Огляд технологій реалізації

Ігровий рушій (game engine) - це програмне забезпечення, що використовується для створення, розробки та запуску відеоігор. Він надає засоби для реалізації геймплею, графіки, фізики, звуку, штучного інтелекту та інших компонентів відеоігрового досвіду. Ігровий рушій забезпечує середовище, у якому розробники можуть створювати та відтворювати свої ігрові проєкти.

Основні компоненти ігрового рушія включають:

- Графічний рушій: Він відповідає за обробку та відтворення графіки в ігрі. Це включає роботу з тривимірною графікою (3D), текстурами, освітленням, тіньовими ефектами, анімацією персонажів та об'єктів, частинками та іншими візуальними елементами.
- Фізичний рушій: Він відповідає за симуляцію фізичних взаємодій у грі, таких як гравітація, колізії, рух об'єктів, силові ефекти та інші фізичні властивості. Фізичний рушій дозволяє об'єктам у грі взаємодіяти між собою та з навколишнім середовищем, що створює більш реалістичний світ.
- Аудіо рушій: Він відповідає за обробку та відтворення звукових ефектів у грі. Це включає музику, звуки персонажів, звукові ефекти оточуючого середовища та інші аудіо компоненти. Аудіо рушій забезпечує цікавий звуковий досвід, який доповнює геймплей та графіку.
- Інструменти розробки: Ігровий рушій зазвичай надає набір інструментів, що спрощують процес розробки гри. Це можуть бути редактори рівнів, інтерфейсу користувача, скриптові мови, системи анімації та інші

інструменти, які допомагають розробникам створювати, тестувати та відлажувати ігрові проекти.

Існує багато різних ігрових рушіїв, таких як Unity, Unreal Engine, CryEngine, Godot та багато інших. Кожен з них має свої особливості та можливості, але загальна мета - надати розробникам інструменти для створення вражаючих та захоплюючих відеоігор.

Як інструмент для реалізації відеогри було обрано ігровий рушій Unity, який має безліч переваг, таких як:

- Кросплатформеність: Unity дозволяє розробляти ігри для різних платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність (VR) і доповнену реальність (AR). Це означає, що розробники можуть створювати свої ігри один раз і потім експортувати їх на різні платформи без необхідності переписування коду.
- Легкість використання: Unity надає інтуїтивний і простий у використанні інтерфейс, що дозволяє навчення та розробку швидко. Він має широкий набір інструментів та редакторів, які спрощують процес створення ігрових об'єктів, налаштування параметрів, роботу зі штучним інтелектом та анімацією.
- Велика спільнота розробників: Unity має велику та активну спільноту розробників, де можна знайти безліч ресурсів, уроків, пакетів активів та підтримки. Це дозволяє розробникам обмінюватися знаннями, отримувати допомогу та швидко розв'язувати проблеми під час розробки.
- Мультимедійні можливості: Unity має потужні можливості управління графікою, звуком та відео. Він підтримує високоякісну

тривимірну графіку, різноманітні спеціальні ефекти, системи частинок, фізичну симуляцію та багато інших функцій.

2.4 Сітковий алгоритм генерації

Для реалізації рандомної генерації сіткової карти 8 на 8 у грі на платформі Unity можна використати наступні кроки:

Спочатку створюємо пусту сітку розміром 8 на 8, яка буде представляти карту гри. Можна використати двовимірний масив або інші структури даних для представлення сітки.

Для генерації рандомних об'єктів на карті, таких як дерева і квадрати з будинками, можна застосувати наступний алгоритм:

Проходимося по кожній клітинці сітки.

Для кожної клітинки випадковим чином генеруємо число, яке визначає тип об'єкта, що буде розміщений в цій клітинці.

Якщо згенероване число відповідає типу дерева, то розміщуємо дерево в цій клітинці.

Якщо згенероване число відповідає типу квадрата з будинком, то розміщуємо квадрат з будинком в цій клітинці.

Якщо згенероване число відповідає типу пустого квадрата, то залишаємо клітинку порожньою.

У середовищі розробки Unity можна використати наступні елементи для реалізації генерації сіткової карти:

Створюємо сцену у Unity, де буде розміщена карта гри. Додаємо необхідні об'єкти, такі як пустий квадрат, квадрат з будинком та дерево, в якості префабів.

Створюємо скрипт мовою програмування C# для реалізації генерації сіткової карти. В цьому скрипті ми виконуємо наш алгоритм рандомної генерації об'єктів на карті:

Створюємо двовимірний масив або використовуємо іншу структуру даних для представлення сітки.

Проходимося по кожній клітинці сітки і випадковим чином генеруємо число відповідно до типу об'єкта.

За допомогою функцій Unity, таких як Instantiate, розміщуємо об'єкти на карті, враховуючи їхні координати в сітці.

Розділ 3 Програмна реалізація ігрового продукту

3.1 Сіткова генерація рівня

Одним з ключових елементів гри "Намалюй шлях для води" є генерація сіткової карти розміром 8 на 8, де кожен елемент сітки може мати різні типи об'єктів, такі як дерево, будинок або пустий квадрат.

Генерація сіткової карти може бути виконана за допомогою алгоритму, що базується на випадковості. Основна ідея полягає в тому, щоб проходити крізь кожен елемент сітки і випадковим чином призначати йому певний тип об'єкта з певною ймовірністю.

У розглянутому коді при генерації сітки використовується подвійний цикл, який проходить через кожен рядок і кожен стовпчик сітки. На кожному кроці коду перевіряється випадкове число за допомогою функції Random.Range(0f, 1f). Це число знаходиться в діапазоні від 0 до 1 і використовується для порівняння з ймовірністю появи об'єкта. Якщо випадкове число менше ймовірності, то об'єкт призначається елементу сітки.

У цьому прикладі використовуються дві ймовірності: ймовірність появи дерева (treeProbability) та ймовірність появи будинку (buildingProbability). Ці значення можна змінювати, щоб контролювати частоту появи різних типів об'єктів на сітці.

Отриману сітку можна використовувати у вашому коді гри для подальшого розміщення об'єктів на сцені Unity. Наприклад, ви можете перебрати всі елементи сітки і в залежності від їх типу створити візуальні об'єкти, які відповідають деревам, будинкам або пустим квадратам.

Генерація рандомної сіткової карти додає в гру варіативність та цікавість, оскільки кожна нова гра може мати іншу конфігурацію об'єктів на карті.

Це дозволяє гравцям досліджувати різні стратегії та шляхи для прокладання каналу, що підвищує повторність і геймплейну цінність гри.

Для реалізації рандомної генерації сіткової карти розміром 8 на 8 в грі "Намалюй шлях для води", розробленої на Unity, можна використати наступний підхід:

- Створення класу для представлення кожного елемента сітки:
- Створення класу для генерації сіткової карти

```
public class GridGenerator
{
    public GridElement[,] GenerateGrid(int rows, int columns)
    {
        GridElement[,] grid = new GridElement[rows, columns];

        // Генерація випадкових об'єктів на сітці
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < columns; j++)
            {
                grid[i, j] = new GridElement();

                // Випадкове розміщення дерева
                if (Random.Range(0f, 1f) < treeProbability)
                {
                    grid[i, j].hasTree = true;
                }

                // Випадкове розміщення будинку
                if (Random.Range(0f, 1f) < buildingProbability)
                {
                    grid[i, j].hasBuilding = true;
                }
            }
        }

        return grid;
    }
}
```

Рис. 3.1 код генератора сіткової карти

Зверніть увагу, що treeProbability та buildingProbability - це ймовірності появи дерева та будинку на кожному елементі сітки. Ви можете налаштувати ці значення відповідно до вашого бажання.

- Виклик генератора сітки та використання отриманого результату у вашому коді гри:

```
public class Game : MonoBehaviour
{
    private GridElement[,] grid;

    private void Start()
    {
        GridGenerator gridGenerator = new GridGenerator();
        grid = gridGenerator.GenerateGrid(8, 8);

        // Використання отриманої сітки у вашому коді гри
        // Наприклад, перебір усіх елементів сітки та розміщення об'єктів на сце
    }
}
```

Рис 3.2 Розміщення елементів

За допомогою цього підходу ви можете згенерувати сіткову карту розміром 8 на 8 з випадково розташованими деревами та будинками. Ви можете використовувати цю карту для подальшого розміщення каналів та розробки іншої логіки вашої гри.

Зверніть увагу, що в коді гри ви повинні використовувати отриману сітку для заборони прокладання каналів через елементи з деревами або будинками.

3.2 Полігональна генерація ландшафту

У даному розділі описано процес створення динамічної полігональної зміни ландшафту під час того, як некеровані персонажі (NPC) копають канал по заданому маршруту в грі "Намалюй шлях для води".

Перед початком реалізації динамічної полігональної зміни ландшафту, використайте алгоритм шляхового планування, наприклад, алгоритм A*, для визначення маршруту, по якому NPC будуть копати канал. Збережіть отриманий маршрут у відповідній структурі даних, наприклад, у вигляді списку точок чи координат.

Створіть полігональну сітку, яка буде представляти мапу гри. У Unity це можна зробити, використовуючи меш (Mesh) та мешовий фільтр (Mesh Filter). Задайте необхідну кількість вершин, ребер та граней для створення бажаної форми ландшафту. Розмістіть мешовий фільтр на відповідному об'єкті в сцені гри.

Для створення анімації копання каналу NPC можна використовувати скрипти Unity. Створіть скрипт, який буде керувати рухом NPC по маршруту та змінювати положення вершин сітки для ефекту копання.

```
public class DiggingAnimation : MonoBehaviour {
    public Transform target; // Цільова позиція NPC для копання каналу
    public float diggingSpeed = 2f; // Швидкість копання

    private Vector3 startPosition;
    private bool isDigging;

    private void Start() {
        startPosition = transform.position;
        isDigging = false;
    }

    private void Update() {
        if (isDigging) {
            float step = diggingSpeed * Time.deltaTime;
            transform.position = Vector3.MoveTowards(transform.position, t
        }
    }

    public void StartDigging() {
        isDigging = true;
    }

    public void StopDigging() {
        isDigging = false;
        transform.position = startPosition;
    }
}
```

Рис 3.3 Генерація ландшафту

Отримайте доступ до компонентів мешового фільтра, щоб отримати доступ до вершин сітки.

Використовуйте цикл анімації, наприклад, FixedUpdate, для зміни положення вершин сітки на кожному кроці анімації.

Залежно від проходження NPC по маршруту, змінюйте позиції вершин сітки, щоб створити враження копання каналу.

Контролюйте швидкість та плавність руху NPC та анімацію зміни ландшафту, використовуючи плавні переходи між позиціями вершин.

Щоб впливати на прокладання каналу для води під час копання NPC, необхідно перевіряти можливість прокладання каналу під час проходження NPC по маршруту.

Перевіряйте наявність перешкод, таких як дерева чи будинки, на шляху NPC.

Якщо NPC зіткнуться з перешкодою, зупиніть копання каналу в цій області та забороніть прокладання каналу через неї.

Забезпечте відображення цих обмежень на ландшафті, можливо, змінивши колір чи текстуру певних областей.

Під час реалізації динамічної полігональної зміни ландшафту важливо забезпечити її ефективність та якість. Деякі кроки для оптимізації та покращення можуть включати:

- Використання алгоритмів шляхового планування з оптимальними швидкостями та ефективністю для розрахунку маршрутів NPC.
- Застосування масивів або буферів для збереження та маніпуляції даними ландшафту, замість безпосереднього звернення до вершин сітки.
- Використання LOD (рівні деталізації) для оптимізації відображення ландшафту, зменшуючи кількість вершин для віддалених областей.

У цьому розділі було описано процес створення динамічної полігональної зміни ландшафту під час копання каналу NPC по заданому маршруту. Визначення маршруту NPC, створення полігональної сітки, анімація копання каналу та вплив на прокладання каналу були описані як частини цього процесу. Оптимізація та покращення динамічної полігональної зміни є важливими аспектами для забезпечення якісної та ефективної гри.

У наступних розділах будуть розглянуті інші аспекти розробки гри "Намалюй шлях для води", включаючи інтеграцію звуків, розробку інтерфейсу та створення керування

3.3 Керування

Перш за все, необхідно визначити метод керування для гри. Залежно від платформи, на якій буде запускатися гра, можна обрати різні способи керування, такі як тачскрін, клавіатура або геймпад. У цьому випадку, для гри на телефоні, оптимальним варіантом буде використання тачскріна.

Для розробки керування грою з використанням тачскріна на телефоні можна виконати наступні кроки:

- Створити візуальні елементи управління на екрані, такі як кнопки або панель інструментів для малювання шляху.
- Використати функції Unity для визначення дотику користувача до екрану та отримання координат дотику.
- За допомогою скриптів Unity визначити логіку реагування гри на дотик користувача.
- Реалізувати логіку малювання шляху на мапі відповідно до руху пальця користувача.

Під час розробки керування важливо забезпечити зручність та точність управління для гравця. Для досягнення цього можна виконати наступні дії:

- Забезпечити візуальний зворотний зв'язок під час малювання шляху, наприклад, за допомогою зміни кольору чи відображення ефектів.

- Налаштувати чутливість керування таким чином, щоб реагування гри на рухи пальця було точним і плавним.
- Виконати тестування гри на різних пристроях та забезпечити сумісність з різними роздільними здатностями екранів.

У цьому розділі було описано процес створення керування для гри "Намалюй шлях для води". Вибір методу керування залежить від платформи, на якій запускається гра, і в даному випадку оптимальним варіантом є використання тачскріна. Розробка керування включає створення візуальних елементів управління та використання функцій Unity для обробки дотику користувача. Оптимізація керування включає забезпечення зручності та точності управління для гравця.

3.4 Моделювання ігрових об’єктів

У рамках даного проекту було виконано створення лов полі 3D-моделей за допомогою програми Blender. Процес роботи включав кілька етапів, які будуть описані нижче.

- Планування: перед початком моделювання було проведено планування проєкту. Було визначено, що метою створення 3D-моделей є створення низькополігональних об'єктів для використання в грі. Були розроблені концепції та визначені наміри досягнення.
- Початок роботи з Blender: після запуску програми Blender було створено новий проєкт. Була використана платформа моделювання, де були додані базові форми, такі як куби, сфери або циліндри. Ці базові форми стали основою для подальшого моделювання.

- Моделювання форм: за допомогою різних інструментів та технік моделювання в Blender було створено бажані форми об'єктів. Інструменти, такі як "Extrude", "Bevel", "Subdivide" та "Sculpt", були використані для деталізації та формування об'єктів згідно з концепцією.

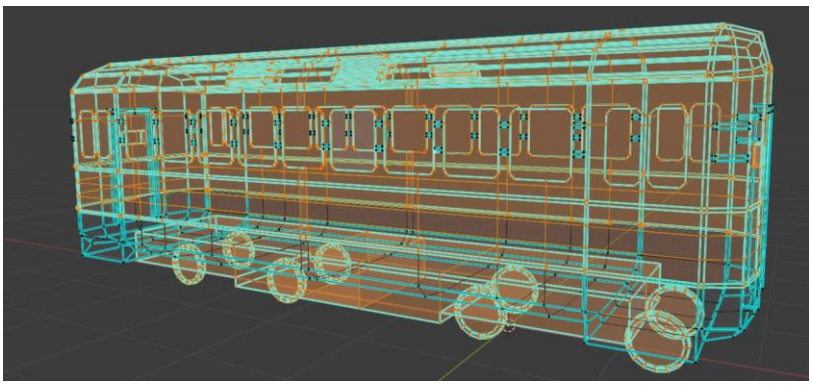

Рис 3.4 Деталізація моделі

- Топологія: велику увагу було приділено створенню правильної топології об'єктів. Було організовано вершини, грани та ребра таким чином, щоб забезпечити збереження форми об'єкта та додаткову деталізацію.

Процес текстурування також є важливою частиною створення 3D-моделей. В ньому використовуються різні інструменти та техніки для надання об'єктам більш реалістичного вигляду. Процесу текстурування в програмі Blender складається з наступних етапів:

- Збір матеріалів: перш за все, для текстурування об'єкта потрібно мати набір матеріалів. Матеріал - це комбінація текстур, кольорів, властивостей поверхні та інших параметрів, які визначають

зовнішній вигляд об'єкта. Матеріал може містити кольорові текстури, мапи нормалей, відбиття, прозорість і багато іншого.

- Налаштування матеріалів: у Blender є можливість створення та редагування матеріалів за допомогою редактора матеріалів. Цей редактор дозволяє вам керувати всіма аспектами матеріалу, включаючи кольори, текстури, блиск і тіні. Ви можете вибрати текстуру, яку хочете застосувати до об'єкта, і задати її параметри, такі як масштаб, поворот і зміщення.

- Додавання текстур: у Blender можна додати текстури до об'єктів за допомогою вкладки "Текстури" в редакторі матеріалів. У цій вкладці можна вибрати тип текстури (наприклад, кольорова, нормальна, бамп-карта тощо) та завантажити власні текстури або використовувати вбудовані. Після цього є можливість встановити параметри текстури, такі як масштабування, поворот і зміщення, щоб налаштувати її вигляд на об'єкті.

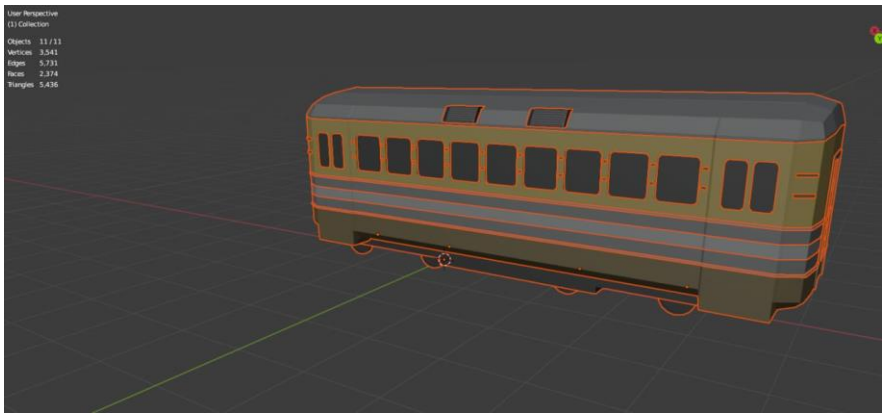

Рис 3.5 Вигляд моделі

3.5 Збереження даних користувача

Для збереження даних користувача (кількості зібраних ігрових монет) можна використати PlayerPrefs. PlayerPrefs - це механізм збереження даних в Unity, який дозволяє зберігати прості значення, такі як рядки, числа та булеві значення, між сеансами гри. Вони записуються на диск у вигляді ключ-значення і доступні для зчитування та запису в будь-який час.

Для цього можна використати наступні API:

1. Запис значення: За допомогою методу SetInt можна записати значення кількості монет за ключем (наприклад, “coins”):
PlayerPrefs.SetInt("coins", Coins);

2. Зчитування значення: Щоб отримати збережене значення за його ключем (“coins”), можна використати метод GetInt:
int savedCoins = PlayerPrefs.GetInt("coins");

3. Оновлення значення: Для оновлення збереженого значення можна просто встановити нове значення для відповідного ключа методом SetInt:
PlayerPrefs.SetInt("coins", Coins);

3.6 Реалізація звуків та інтерфейсу

Інтеграція звуків в гру додає атмосферу та покращує враження від геймплею. Для реалізації звукових ефектів у грі "Намалюй шлях для води" можна виконати наступні кроки:

- Створити аудіофайли для різних подій у грі, таких як вибір клітинки, прокладання каналу, зіткнення з перешкодою або досягнення певного рівня.
- Додати аудіо джерела до об'єктів у грі, де потрібні звукові ефекти.
- Використати скрипти Unity для управління відтворенням звуків під час відповідних подій у грі.

Розробка інтерфейсу гри "Намалюй шлях для води" включає створення зручного та привабливого користувацького інтерфейсу (UI), що допомагає гравцеві взаємодіяти з грою. Основні елементи розробки інтерфейсу можуть включати:

- Створення головного меню, де гравець може почати нову гру, переглянути налаштування або перейти до інших розділів гри.
- Відображення інформації про поточний рівень, досягнення гравця та інші статуси гри.
- Відображення панелі інструментів або іншого елементу управління для малювання шляху на мапі.
- Розробка візуальних ефектів, які підсилюють взаємодію гравця з об'єктами на карті, наприклад, зміна кольору або анімація при виборі клітинки.

Інтерфейс гри "Намалюй шлях для води" повинен бути зрозумілим та забезпечувати зручне малювання шляху на мапі. Гра повинна мати привабливий дизайн, який відображає мапу, сектори, шлях та воду. Додавання візуальних ефектів та анімації покращує враження від гри та залучає гравця до процесу.

3.7 Підключення аналітики

Підключення Facebook та GameAnalytics аналітики до проекту Unity буде корисним для отримання аналітичних даних про використання гри,

поведінку гравців та іншої цінної інформації. Нижче описані процеси підключення кожної з аналітичних платформ.

Підключення Facebook та GameAnalytics аналітики до проєкту Unity має наступні кроки:

1. Створення акаунту розробника Facebook та GameAnalytics та створення нового застосунку на платформі Facebook Developers та Google Analytics.
2. Завантаження та інтегрування Facebook SDK і GameAnalytics SDK для Unity у свій проєкт.
3. Встановлення ідентифікатора застосунку Facebook (App ID) і ідентифікатора потоку даних GameAnalytics (Tracking ID) у налаштуваннях проєкту Unity.
4. Додавання коду, який ініціалізує Facebook SDK і GameAnalytics SDK та виконання налаштування аналітики. Цей код може включати відстеження подій, відправку інформації про гравців, створення власних подій та багато іншого.
5. Тестування застосунку та перевірка, чи отримують сервіси аналітичні дані у акаунті розробника Facebook і GameAnalytics.

Код запуску аналітики має наступний вигляд:

```
1 reference
private static void InitAPI()
{
    try
    {
        _settings = (GameAnalyticsSDK.Setup.Settings)Resources.Load("GameAnalytics/Settings", typeof(GameAnalyticsSDK.Setup.Settings));
        GameAnalyticsSDK.State.GAState.Init();

#if UNITY_EDITOR
        if(_settings == null)...
#endif
    }
    catch(System.Exception e)
    {
        Debug.Log("Error getting Settings in InitAPI: " + e.Message);
    }
}
```

Рис. 3.6 Код запуску аналітики

3.8 Фінальний білд проєкту

Для налаштування проєкту Unity для білда на Android знадобиться виконати декілька кроків. Загальний опис процесу виглядає наступним чином:

1. Перевірка налаштування платформи: Для цього можна перейти у "File" (Файл) -> "Build Settings" (Налаштування збірки) у головному меню Unity. У вікні Build Settings (Налаштування збірки) потрібно переконатись, що платформа Android вибрана як активна платформа.

2. Налаштування Player Settings (Налаштування гравця): У вікні Build Settings (Налаштування збірки) необхідно перейти в "Player Settings" (Налаштування гравця). З'явиться вікно Inspector з налаштуваннями гравця. У цьому вікні є безліч налаштувань, які потрібно здійснити:

Bundle Identifier (Ідентифікатор пакета): Необхідно встановити унікальний ідентифікатор пакета застосунку.
Minimum API Level (Мінімальний рівень API): Необхідно обрати мінімальний рівень API, який повинен підтримуватись застосунком. Для роботи з Google Play Market необхідно встановити параметр як API level 33 (станом на травень 2023

року).

Graphics API (Графічний API): Необхідно обрати потрібні графічні API, такі як OpenGL ES 2.0 або Vulkan. У вас з обраним проєкт можна залишити OpenGL.

Other Settings (Інші налаштування): У вкладках "Identification" (Ідентифікація) та "Configuration" (Конфігурація) потрібно знайти і налаштувати інші параметри, такі як ім'я додатка, версію, підписи, налаштування орієнтації екрана відповідно до проєкту.

Scripting Backend необхідно обрати IL2CPP для роботи з Google Play Market. Також для Play Market необхідно встановити параметри ARMv7 та ARM64 для змоги застосунку правильно працювати з 64-бітною архітектурою.

Після здійснення налаштувань можна здійснювати білд програми у форматі APK (для тестування на телефоні) і ABB (для Google Play Market). Для цього потрібно натиснути кнопку "Build" (Збудувати), після чого Unity почне процес збудови застосунку. Заздалегідь зазначений шлях буде містити створений файл.

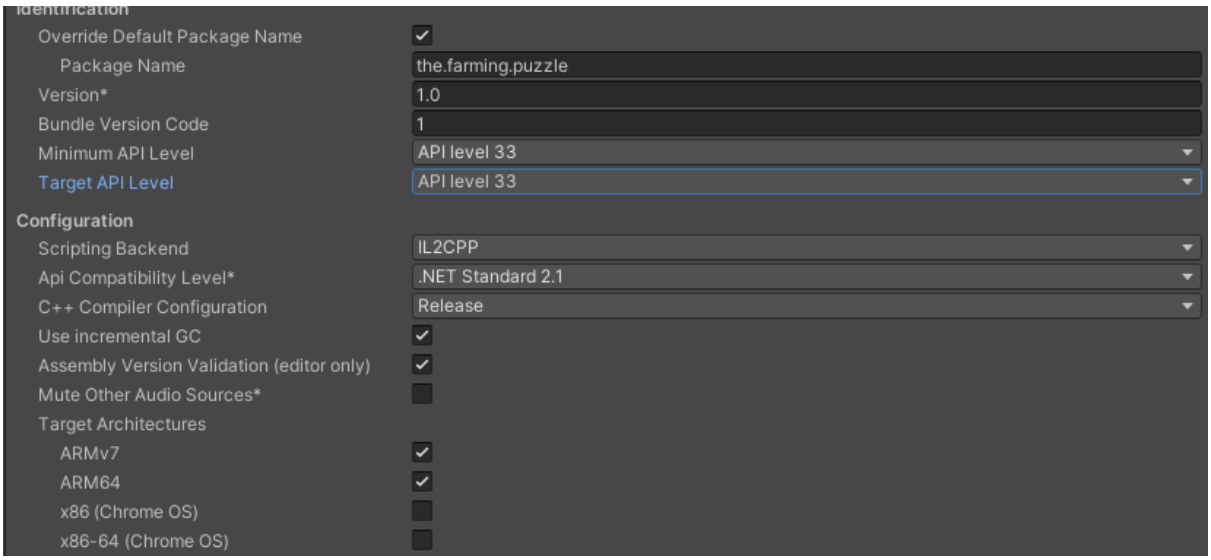

Рис 3.6 Налаштування білда

ВИСНОВКИ

У цій роботі було:

- Розглянуто види жанрів ігор
- Проаналізовано ринок ігор
- Досліджено наявні методи розробки
- Поєднано у інтерактивну гру для більшого зацікавлення потенційного користувача

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Офіційна документація Unity URL: https://docs.unity3d.com/Manual/index.html
2. UNIVERSAL RENDER PIPELINE (URP). Unity. URL: https://unity.com/srp/universal-render-pipeline
3. Unity VR URL: https://docs.unity3d.com/Manual/VROverview.html
4. Unity video player: https://docs.unity3d.com/ScriptReference/Video.VideoPlayer.html
5. Wikipedia URL: https://uk.wikipedia.org/wiki/
6. Unity Asset Store URL: https://assetstore.unity.com/

ДОДАТОК

Додаток А

Фрагмент скирпту кривих Безьє

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


public class BezierCurves : MonoBehaviour
{
    public static Vector3 GetPointByLerp(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3, float t)
    {
        Vector3 p01 = Vector3.Lerp(p0, p1, t);
        Vector3 p12 = Vector3.Lerp(p1, p2, t);
        Vector3 p23 = Vector3.Lerp(p2, p3, t);

        Vector3 p012 = Vector3.Lerp(p01, p12, t);
        Vector3 p112 = Vector3.Lerp(p1, p23, t);

        Vector3 p0123 = Vector3.Lerp(p012, p112, t);
```

```
        return p0123;
    }

    public static Vector3 GetPoint(Vector3 p0, Vector3 p1, Vector3 p2, Vector3
p3, float t)
    {
        t = Mathf.Clamp01(t);
        float oneMinusT = 1f - t;
        return
            Mathf.Pow(oneMinusT, 3) * p0 +
            3f * Mathf.Pow(oneMinusT, 2) * t * p1 +
            3f * oneMinusT * Mathf.Pow(t, 2) * p2 +
            Mathf.Pow(t, 3) * p3;
    }

    public static Vector3 GetFirstDerivate(Vector3 p0, Vector3 p1, Vector3 p2,
Vector3 p3, float t)
    {
        t = Mathf.Clamp01(t);
        float oneMinusT = 1f - t;
```

```
        return
            3f * oneMinusT * oneMinusT * (p1 - p0) +
            6f * oneMinusT * t * (p2 - p1) +
            3f * t * t * (p3 - p2);
    }
}
```

Додаток Б

Фрагмент скирпту управління кривими Безьє

```
using UnityEngine;


public static class Bezier
{
    public static Vector3 GetPoint(Vector3 pointA, Vector3 pointB, Vector3
pointC, Vector3 pointD, float t)
    {
        Vector3 pointAB = Vector3.Lerp(pointA, pointB, t);
        Vector3 pointBC = Vector3.Lerp(pointB, pointC, t);
        Vector3 pointCD = Vector3.Lerp(pointC, pointD, t);


        Vector3 pointABBC = Vector3.Lerp(pointAB, pointBC, t);
        Vector3 pointBCCD = Vector3.Lerp(pointBC, pointCD, t);
```

```
        Vector3 point = Vector3.Lerp(pointABBC, pointBCCD, t);


        return point;
    }
}
```

Додаток Г

Фрагмент скирпту керування аналітикою

```
using System;
using System.Collections;
using System.Collections.Generic;
using Facebook.Unity;
using GameAnalyticsSDK;
using UnityEngine;


public class AnalyticsController : MonoBehaviour
{
    public static AnalyticsController analyticsController;


    Dictionary<string, object> event_parameters = new Dictionary<string,
object>();
```

```
//Dictionary<string, object> event_parameters;
[HideInInspector] public int level_number = 1;
[HideInInspector] public int level_count;
[HideInInspector] public float time;



private void Awake()
{
    if (!FB.IsInitialized)
    {
        // Initialize the Facebook SDK
        FB.Init();
    }
    else
    {
        // Already initialized, signal an app activation App Event
        FB.ActivateApp();
    }
```

```
    if (analyticsController == null)
    {
        //level_count = PlayerPrefs.GetInt("level_count", 0);
        analyticsController = this;
        DontDestroyOnLoad(this.gameObject);
    }
    else
    {
        Destroy(this.gameObject);
    }

    level_count++;
    start_analitics();
}

private void Start()
{
    GameAnalytics.Initialize();
}
```

```
    private void Common_parameters()
    {
        event_parameters.Add("level_count", level_count);
        event_parameters.Add("level_number", level_number);
        event_parameters.Add("level_random", true);
    }

    private void start_analitics()
    {
        event_parameters.Clear();
        Common_parameters();
        Event_analitics("level_start");

        //Debug.Log(level_count + " / " + PlayerPrefs.GetInt("Level_analitics") +
" / " + level_loop);
    }

    private void finish_analitics()
    {
        event_parameters.Clear();
        Common_parameters();
```

```
        event_parameters.Add("time", (int) time);
        Event_analitics("level_finish");

        //Debug.Log(level_count + " / " + PlayerPrefs.GetInt("Level_analitics") +
" / " + level_loop + " / " + result + " / " + time + " / " + progress);
    }

    private void Event_analitics(string start_finish)
    {
    }

    public void ResetVar()
    {
        finish_analitics();
        time = 0;
        level_number++;
        level_count++;
        PlayerPrefs.SetInt("level_count", level_count);
        start_analitics();
    }
```

```
    void FixedUpdate()
    {
        time += Time.deltaTime;
    }
}
```