

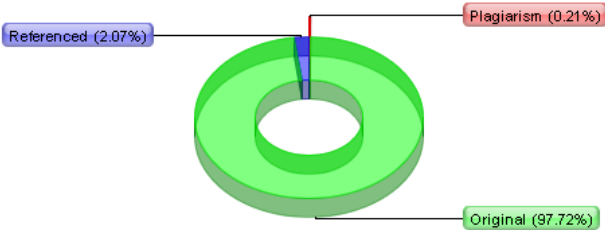
Детектор Плагиата v. 2215 - Отчёт оригинальности: 16.06.2024 18:35:55

Проанализированный документ: ПЗ_Вередін.docx Лицензия: ВОЛОДИМИР МАТІЄВСЬКИЙ_License2

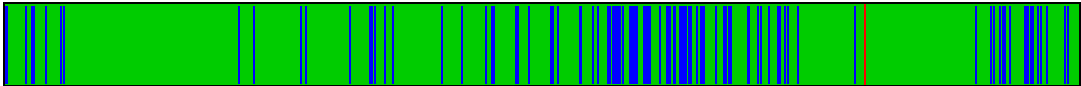
- ? Тип поиска: Поиск переписанного ? Язык: Uk
- ? Тип проверки: Интернет
- ТЕЕ и кодировка: DocX n/a

Детальный анализ тела документа:

? Диаграмма соотношения частей:



? Граф распределения зон:



? Источники плагиата: 7

→ 0,2% ABC 30	1. https://uk.wikipedia.org/wiki/Освіта
→ 0,2% ABC 31	2. https://itedu.center.ua/blog/career/10-most-in-demand-it-tech-jobs/
→ 0,2% ABC 22	3. https://2020.moodlemoot.in.ua/course/view.php?id=28

? Детали обработанных ресурсов: 221 - ОК / 3 - Ошибка

? Важные замечания:

Википедия:	Google Книги:	Сервисы платных работ:	Античит:
Обнаружена Wiki!	[не обнаружено]	[не обнаружено]	Обнаружено сокрытие!

? Античит-отчет UACE:

1. Статус: Анализатор Включен Нормализатор Включен сходство символов установлено на 100%
2. Обнаруженный процент загрязнения UniCode: 15,2% с лимитом: 4%
3. Процент нераспознанных символов после нормализации: 8,9%
4. Все подозрительные символы будут отмечены фиолетовым цветом: Abcd...
5. Найдены невидимые символы: 0
Рекомендации по оценке: Особое внимание следует уделить анализу этого отчета! Предполагается, что этот документ содержит значительное количество символов, чуждых языку документа. Это прямое указание на то, что автор документа использовал специальное программное обеспечение\онлайн-веб-сервис, чтобы эффективно скрыть текст в попытке избежать обнаружения потенциального плагиата. Настоятельно рекомендуется передать это дело на более высокий уровень! В случае сомнений обращайтесь: в службу поддержки Детектора плагиата!
Алфавитная статистика и анализ символов:

? Активные ссылки (URL-адреса, извлеченные из документа):

URL не найдены

? Исключённые ресурсы:

URL не найдены

? Включённые ресурсы:

URL не найдены

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДЕРЖАВНИЙ ЗАКЛАД	
Цитування: 0,04%	id: 1
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА”	
Навчально-науковий інститут математики та інформаційних технологій Кафедра інформаційних технологій та систем Пояснювальна записка до кваліфікаційної роботи за першим (бакалаврським) рівнем освіти на тему: Розробка ігрового додатку віртуальної реальності для VR-шолому Meta Quest Виконав: здобувач вищої освіти 4 курсу спеціальності 121	
Цитування: 0,02%	id: 2
«Інженерія програмного забезпечення»	
(шифр і назва напрямку підготовки, спеціальності) Вередін М. О. (прізвище та ініціали) Керівник ____ Переяславська С.О. (прізвище та ініціали) Рецензент ____ Козуб Ю.Г. (прізвище та ініціали) Полтава – 2024 Зміст ВСТУП4 РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІГРОВОГО ДОДАТКУ ДЛЯ VR-ШОЛОМУ META QUEST6 Віртуальна реальність та її основні принципи6 Історія розвитку віртуальної реальності6 Технологічні складові віртуальної реальності9 Принципи роботи VR-систем11 Сфери застосування віртуальної реальності12 Основні переваги та недоліки віртуальної реальності14 Етичні та соціальні аспекти віртуальної реальності15 Технічні характеристики та можливості VR-шолому Meta Quest17 Особливості розробки ігрових додатків для віртуальної реальності18 Загальні принципи розробки VR-ігор18 Геймдизайн для VR20 Мінімізація ризику нудоти та запаморочення21 Розробка інтерактивного контенту22 Методи взаємодії користувача з ігровим середовищем у VR24 Висновки до розділу 126 Розділ 2. ПІДГОТОВЧІ ЕТАПИ ДО РОЗРОБКИ ДОДАТКУ27 Концепція та ідея ігрового додатку27 Вибір технологій та інструментів для розробки28 Загальний спектр необхідних технологій та інструментів 29 Вибір рушія для розробки гри30 Висновок вибору рушія33 Вибір SDK для рушія33 Створення проекту в Unity та підключення SDK34 Архітектура ігрового додатку та взаємодія з VR-шоломом35 Вибір архітектурного підходу36 Взаємодія з VR-шоломом Meta Quest37 Візуалізація в VR38 Управління користувачем40 Висновки до розділу 242 РОЗДІЛ 3. РОЗРОБКА ІГРОВОГО ДОДАТКУ ДЛЯ VR ШОЛОМУ META QUEST43 Реалізація графічного контенту43 Ключові цілі та завдання43 Моделювання та текстурування об'єктів44 Створення частин рівня45 Програмування випадкової генерації рівня47 Дизайн та ідея ігрового меню50 Інтерфейс51 Вороги53 Реалізація звукового контенту57 Фонова музика57 Звуки пострілів58 Звуки руху60 Взаємодія з користувачем та управління грою у віртуальному середовищі61 Інтеграція та використання SDK	
Цитування: 0,02%	id: 3
«Unity XR Interaction Toolkit»	
61 Керування персонажем63 Взаємодія з інтерфейсом65 Висновки до розділу 367 ВИСНОВКИ68 Список використаних джерел69 ДОДАТОК А73 ДОДАТОК В82 ВСТУП Віртуальна реальність (VR) стала однією з найперспективніших технологій сучасності, яка знаходить застосування в різних сферах — від розваг і освіти до медицини та промисловості. Завдяки можливості занурити користувача в інший світ, VR забезпечує новий рівень інтерактивності та вражень, що відкриває безліч можливостей для розробників ігрового контенту. У дипломній роботі ми зосередимося на розробці ігрового додатку для VR-шолому Meta Quest, який є одним з найбільш популярних та доступних пристроїв на ринку. Розробка ігрових додатків для VR вимагає врахування специфічних аспектів, таких як мінімізація нудоти та запаморочення, оптимізація графіки для реалістичності, та забезпечення природної взаємодії користувача з віртуальним середовищем. Мета роботи – розробка ігрового додатку	
Цитування: 0,02%	id: 4
“ Fitness runner VR ”	
для шолому віртуальної реальності Meta Quest. Об'єкт дослідження – ігровий VR додаток	
Цитування: 0,02%	id: 5
"Fitness runner VR"	
Предмет дослідження – розробка ігрових додатків для платформи Meta Quest із застосовуванням технологій для підвищення активності та здоров'я. Відповідно до предмета дослідження і мети, були виділені основні завдання дослідження: - узагальнити поняття, основні принципи віртуальної реальності; - дослідити особливості розробки ігрових додатків для віртуальної реальності, методи взаємодії користувача з ігровим середовищем у VR; - розглянути технічні характеристики та можливості VR-шолому Meta Quest; - розробити ігровий додаток віртуальної реальності для VR-шолому Meta Quest за допомогою середовища Unity. Для вирішення завдань дослідження використано такі методи дослідження: теоретичні: аналіз літератури з питань розробки віртуальної реальності (VR), вивчення особливостей роботи для VR-шоломів та їх можливостей; порівняльний аналіз різних методів та підходів до розробки ігрових додатків для VR; емпіричні: дослідження різних рушіїв та SDK для розробки ігор у віртуальній реальності; експериментальні: тестування розробленого додатка. До складу роботи входять три розділи, які висвітлюють теоретичні й практичні питання розробки ігрового додатку	

віртуальної реальності для [VR](#)-шолому [Meta Quest](#). Практичне значення розробки – розроблений ігровий додаток

Цитування: **0,02%**

id: **6**

“Fitness runner VR”

для платформи [Meta Quest](#), який сприяє підвищенню активності та здоров'я користувачів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІГРОВОГО ДОДАТКУ ДЛЯ [VR](#)-ШОЛОМУ [META QUEST](#) Віртуальна реальність та її основні принципи Віртуальна реальність ([VR](#)) - це комп'ютерно створене середовище, яке сприймається користувачем як реальне, де він може взаємодіяти з об'єктами та виконувати дії, використовуючи спеціальне обладнання, як [VR](#)-шоломи, контролери тощо. Основною метою [VR](#) є створення імерсійного враження, що дозволяє користувачеві відчувати себе частиною віртуального світу. Віртуальна реальність відрізняється від інших схожих концепцій, таких як доповнена реальність ([AR](#)) та змішана реальність ([MR](#)). В [AR](#) реальні об'єкти доповнюються віртуальними елементами, які накладаються на навколишній світ за допомогою пристроїв, таких як смартфони або спеціальні окуляри. [MR](#), у свою чергу, поєднує елементи [AR](#) та [VR](#), дозволяючи користувачеві взаємодіяти з віртуальними об'єктами, які інтегровані в реальний світ. Більш детально можна прочитати у [1]. Основна відмінність [VR](#) від цих технологій полягає в повному зануренні користувача в створене віртуальне середовище, у якому він може рухатися, досліджувати та взаємодіяти з об'єктами так, ніби це реальність. Це створює унікальний досвід, який може використовуватися в різних сферах, від ігор та розваг до освіти та медицини. Історія розвитку віртуальної реальності Історія віртуальної реальності простягається на кілька десятиліть і починається ще у середині 20-го століття. Розглянемо кілька ключових моментів в історії розвитку технології [VR](#): 1960-ті роки: Одним з перших піонерів у галузі віртуальної реальності був Кінематографіст Мортон Хейліг, який створив у 1960 році перший прототип пристрою під назвою [Sensorama](#). Це був багатосенсорний симулятор, який він назвав

Цитування: **0,01%**

id: **7**

«Театральний Досвід»

(рис.1.1.) що був здатний повністю охопити та поринути глядача у гру фільму за допомогою стереозвуку, вібрації та стереоскопічних зображень, що створювали іммерсивний досвід. Детальніше можна дізнатись у [2]. Рис.1.1. Прототип пристрою під назвою

Цитування: **0,01%**

id: **8**

«[Sensorama](#)»

1980-ті роки: Перші практичні системи віртуальної реальності були створені в 80-х роках минулого століття. В цей час [NASA](#) почала використовувати [VR](#)-технології для тренування астронавтів (рис.1.2.). Детальніше описано у [3]. В цей же період розвивались перші прототипи шоломів, рукавичок та датчиків руху. Рис.1.2. Інтерактивна рукавичка [Power Glove](#) та шолом віртуальної реальності [NASA](#) 1990-ті роки: У цей період компанії почали активно інвестувати в розробку комерційних [VR](#)-систем, але висока вартість технології та обмежена обчислювальна потужність призвели до їх невеликого поширення. 2000-ні роки: Завдяки розвитку графічних процесорів та комп'ютерної технології, віртуальна реальність стала більш доступною. З'явилися нові шоломи та контролери, які були спрямовані на ігровий ринок. 2010-ті роки: З цього періоду [VR](#) почала стрімко розвиватися, завдяки появі сучасних [VR](#)-шоломів, таких як [Oculus Rift](#), [HTC Vive](#) та [PlayStation VR](#). Також з'явилися інші інновації, включаючи мобільні [VR](#)-рішення (Рис 1.3.), такі як [Google Cardboard](#) та [Samsung Gear VR](#). Рис.1.3. Приклад мобільних [VR](#) окулярів Сьогодні віртуальна реальність є однією з найперспективніших технологій, яка знаходить застосування у різних сферах: від розваг до медицини та освіти. Технологія продовжує вдосконалюватися, стаючи все більш доступною та реалістичною для користувачів. 1.1.2. Технологічні складові віртуальної реальності Технології, які забезпечують функціонування віртуальної реальності ([VR](#)), є надзвичайно складними і складаються з різних компонентів, які працюють разом для створення інтерактивного та захоплюючого віртуального середовища для користувача. Більш детально можна прочитати у [4]. Основні складові [VR](#)-систем включають: Дисплеї [VR](#)-шоломів: Основний елемент системи [VR](#) — це дисплей, вбудований у шолом, який забезпечує стереоскопічне зображення та дозволяє гравцю бачити тривимірний віртуальний світ. Шоломи зазвичай використовують 2 високоякісні дисплеї з високою роздільною здатністю та швидкою частотою оновлення, щоб забезпечити реалістичний та плавний візуальний досвід. Відстеження руху: Системи відстеження руху контролюють рух голови та позицію користувача, а також рухи контролерів чи інших пристроїв взаємодії. Це дозволяє синхронізувати дії користувача з віртуальним середовищем, створюючи ефект занурення та забезпечуючи природний досвід взаємодії. Контролери: Контролери служать засобами взаємодії користувача з віртуальним світом. Вони можуть бути різних форм — від ручних трекерів до спеціалізованих пристроїв у формі рукавичок чи костюмів. Контролери дозволяють користувачеві здійснювати дії, такі як захоплення, рух та маніпуляція об'єктами у [VR](#). Обробка графіки: Процесори для обробки графіки ([GPU](#)) грають вирішальну роль у [VR](#)-системах, оскільки вони відповідають за генерацію та рендеринг тривимірного віртуального середовища в режимі реального часу. Потужні графічні процесори необхідні для забезпечення високої якості зображень, плавності та мінімізації затримок у [VR](#)-досвіді. Аудіо та звуковий простір: Реалістичне аудіо є важливим компонентом віртуальної

реальності, оскільки звуковий простір підвищує ступінь занурення. Використання просторового аудіо дозволяє користувачам чути звуки у різних напрямках, відповідно до їхнього положення у віртуальному середовищі. Ці технологічні складові в сукупності створюють досвід повного занурення, дозволяючи користувачам відчувати себе частиною віртуального світу та взаємодіяти з ним. Постійний розвиток VR-технологій призводить до вдосконалення цих компонентів, підвищуючи якість та реалістичність віртуального досвіду. Принципи роботи VR-систем

Принципи роботи VR-систем базуються на тому, щоб створити реалістичний та захоплюючий досвід користувача у віртуальному середовищі. Для досягнення цього використовуються декілька основних технологій та методів: Стереоскопічне зображення: Одним з ключових принципів роботи VR-систем є створення стереоскопічного зображення, яке забезпечує користувачеві відчуття глибини. Це досягається завдяки подачі різних зображень до лівого та правого ока користувача через дисплей у шоломі. Завдяки цьому виникає ілюзія тривимірного простору, що дозволяє користувачеві відчувати себе частиною віртуального середовища. Трекінг голови та рухів: VR-системи використовують спеціальні сенсори для відстеження положення та руху голови користувача. Це дозволяє синхронізувати зміни віртуального середовища з рухами голови, створюючи ефект природної інтеракції. Аналогічні сенсори можуть бути використані для відстеження рухів рук та інших частин тіла, що дозволяє користувачеві взаємодіяти з віртуальним світом [5]. Відстеження погляду: Деякі сучасні VR-системи мають можливість відстеження погляду користувача, що дозволяє контролювати, куди дивиться гравець, та адаптувати віртуальне середовище відповідно до його дій. Більш детально описано у [6]. Методи створення відчуття присутності: Створення відчуття присутності є одним з основних завдань VR-систем. Це досягається шляхом поєднання реалістичних зображень, звукового простору та інших відчуттів. Використання просторового аудіо, вібраційних пристроїв та інших сенсорних технологій допомагає підвищити рівень занурення та взаємодії у віртуальному середовищі. Детальніше про відчуття присутності у VR можна прочитати у [7]. Контролери та інші засоби взаємодії: Контролери та інші пристрої взаємодії, такі як спеціальні рукавички або костюми, дозволяють користувачеві здійснювати різноманітні дії у віртуальному світі. Ці пристрої забезпечують високий рівень точності та природності у взаємодії з об'єктами та середовищем у грі. Усі ці принципи працюють разом, щоб створити цілісний досвід віртуальної реальності, який наближає користувача до повного занурення у цифровий світ. Завдяки цим технологіям VR стає все більш реалістичним та доступним для різних сфер застосування.

1.1.4. Сфери застосування віртуальної реальності

Віртуальна реальність (VR) надає безліч можливостей у різних галузях завдяки своїй здатності створювати інтерактивні та іммерсивні середовища. Ось кілька ключових сфер застосування VR: Ігри та розваги: Ігрова індустрія є одним з найбільших споживачів VR-технологій. VR-ігри дозволяють гравцям повністю зануритися в ігровий процес, забезпечуючи унікальний та захоплюючий досвід. Ігри можуть охоплювати різноманітні жанри, включаючи пригодницькі, шутери, симулятори та багато інших. Освіта та навчання: VR може бути ефективним інструментом навчання, дозволяючи студентам зануритися в реалістичні середовища та ситуації. Це включає в себе моделювання складних наукових та історичних подій, навчання лікарів хірургії, авіаційні тренажери та багато інших прикладів. Медицина та охорона здоров'я: У медицині VR використовується для тренування лікарів і медсестер, проведення віртуальних операцій, а також терапії пацієнтів з різними захворюваннями, такими як фобії або посттравматичний стресовий розлад. Архітектура та дизайн: VR допомагає архітекторам та дизайнерам створювати тривимірні моделі будівель та інтер'єрів, що дозволяє клієнтам віртуально прогулятися по проєктованих об'єктах перед початком будівництва. Інженерія та виробництво: Інженери можуть використовувати VR для моделювання та тестування складних машин та механізмів у віртуальному просторі, перш ніж розпочати реальне виробництво. Це сприяє зменшенню витрат та прискоренню інновацій. Тренування та симуляція: VR використовується для тренування персоналу в різних галузях, включаючи військову, авіаційну, пожежну та поліцейську служби. Віртуальні симулятори дозволяють тренувати навички в безпечному середовищі, відтворюючи реальні ситуації. Туризм та подорожі: Віртуальна реальність може допомогти людям досліджувати нові місця та планувати свої подорожі, надаючи можливість здійснювати віртуальні тури по всьому світу. Розробка прототипів та продукції: VR використовується для створення та тестування прототипів продукції, що дозволяє виявляти можливі недоліки та оптимізувати дизайн ще до виготовлення реальних зразків. Ці сфери застосування демонструють багатогранність технології VR і її потенціал для розширення можливостей у різних галузях. Віртуальна реальність продовжує розвиватися та знаходить все новіші сфери застосування. Більш детально про сфери застосування описано у [8].

1.1.5. Основні переваги та недоліки віртуальної реальності

Віртуальна реальність надає унікальні можливості для створення захоплюючих ігрових середовищ, освітніх програм та симуляцій, а також відкриває нові горизонти у багатьох галузях, що ми розглянули вище. Однак, поряд з безперечними перевагами, технологія VR також має певні обмеження та виклики, які необхідно враховувати при розробці та використанні VR-додатків. У цьому підпункті ми детально розглянемо ці аспекти, щоб надати всебічний аналіз VR з точки зору користувача та розробника. Переваги VR-технологій: Іммерсивний досвід: VR дозволяє користувачам повністю зануритися у віртуальний світ, забезпечуючи реалістичний досвід, який неможливо відтворити за допомогою традиційних засобів. Покращена візуалізація: Завдяки можливостям тривимірної графіки та інтерактивних середовищ, VR надає унікальні можливості для візуалізації складних об'єктів, проєктів та

процесів. Навчання та тренування: [VR](#) дозволяє моделювати реальні ситуації, що робить навчання більш ефективним та цікавим. Це особливо корисно у таких сферах, як медицина, авіація, інженерія та військова справа. Розширення творчих можливостей: Для митців та дизайнерів [VR](#) відкриває нові можливості для створення інноваційних проєктів, використовуючи тривимірне середовище для творчого вираження. Інтерактивність та взаємодія: [VR](#) надає користувачам можливість взаємодіяти з віртуальними об'єктами та середовищем за допомогою контролерів та інших пристроїв, що створює інтерактивний та захоплюючий досвід. Недоліки [VR](#)-технологій: Кібербезпека: Як і будь-яка інша технологія, [VR](#)-системи можуть бути вразливими до кібератак. Зловмисники можуть отримати доступ до пристроїв користувачів та їхніх даних, що створює ризик для конфіденційності та безпеки. Проблеми з конфіденційністю: [VR](#)-технології можуть збирати значну кількість даних про користувача, включаючи його рухи та поведінку. Це піднімає питання щодо захисту особистої інформації та її використання. Соціальні аспекти: Надмірне використання [VR](#) може призвести до ізоляції користувачів від реального світу, а також впливати на їхнє психічне та фізичне здоров'я. Вартість та доступність: Високі витрати на обладнання та обмежена доступність [VR](#)-пристроїв можуть стримувати поширення технології серед широкого кола користувачів. Технологічні обмеження: Попри значний прогрес, [VR](#)-технології все ще стикаються з низкою обмежень, таких як затримка зображення, відсутність реалістичного зворотного зв'язку та обмежена роздільна здатність дисплеїв. Отже ці переваги та виклики показують, що [VR](#) є надзвичайно перспективною технологією з великим потенціалом, однак її розвиток супроводжується низкою труднощів, які необхідно подолати для повної реалізації її можливостей. Більше про переваги та недоліки [VR](#) можна прочитати у [9].

1.1.6. Етичні та соціальні аспекти віртуальної реальності

Віртуальна реальність ([VR](#)) як новітня технологія відкриває безліч можливостей для користувачів, проте, виходячи з аналізу її переваг та недоліків – вона також ставить перед нами й низку етичних та соціальних питань, які необхідно враховувати для забезпечення безпечного та етичного використання [VR](#). Розглянемо ці аспекти окремо: Конфіденційність даних: Одним із найважливіших питань у сфері [VR](#) є захист конфіденційності користувачів. [VR](#)-системи збирають багато даних про користувачів, включаючи їхні рухи, поведінку та навіть психологічні реакції. Забезпечення захисту цих даних та недопущення їх зловживання є пріоритетним завданням. Вплив на психологічний стан: Занурення у віртуальні світи може впливати на психологічний стан користувачів. Деякі користувачі можуть відчувати дезорієнтацію або тривожність під час тривалого перебування у [VR](#)-середовищі. Дослідження також вказують на можливий зв'язок між надмірним використанням [VR](#) та проблемами зі здоров'ям, такими як вестибулярні порушення. Маніпуляція свідомістю: [VR](#) відкриває можливості для створення високо реалістичних та переконливих віртуальних середовищ, які можуть впливати на сприйняття реальності користувачами. Це ставить питання щодо можливості маніпуляції свідомістю, особливо у випадках, коли [VR](#) використовується для розважальних, рекламних або пропагандистських цілей. Соціальна ізоляція: Надмірне використання [VR](#) може призводити до соціальної ізоляції, оскільки користувачі проводять більше часу у віртуальному середовищі, ніж у взаємодії з іншими людьми у реальному світі. Це може впливати на соціальні навички та емоційне здоров'я. Доступність та інклюзивність: Розробники [VR](#)-систем мають враховувати інклюзивність та доступність для всіх користувачів, включаючи осіб з обмеженими можливостями. Це передбачає створення [VR](#)-досвіду, який враховує різні потреби та обмеження користувачів. Розуміння та вирішення цих етичних та соціальних аспектів є важливим кроком до відповідального та безпечного використання [VR](#)-технологій. Як розробники, так і користувачі повинні мати чіткі орієнтири щодо етичного використання [VR](#), щоб максимізувати її переваги та мінімізувати можливі ризики. Детальніше про це можна прочитати у [10, 11].

Технічні характеристики та можливості [VR](#)-шолому

[Meta Quest VR](#)-шоломи [Meta Quest](#) є важливим представником сучасної технології віртуальної реальності. Ці пристрої відповідають високим стандартам та володіють рядом характеристик, що роблять їх привабливими для користувачів та розробників ігрових додатків. Розглянемо технічні характеристики [VR](#)-шоломів на прикладі шолому [Meta Quest 2](#): Дисплеї з високою роздільною здатністю: [Meta Quest 2](#) оснащений РК-дисплеєм з роздільною здатністю 1832x1920 пікселів на кожне око, що забезпечує якісне та реалістичне зображення у віртуальному середовищі. Змінна частота оновлення 60/120 Гц: Висока частота оновлення дозволяє плавно відтворювати зображення, що є важливим для забезпечення комфортного користування та запобігання відчуттю головного обертання. Вбудована система відслідковування рухів: [Meta Quest 2](#) використовує систему [Oculus Insight](#) для відслідковування рухів користувача. Ця система включає в себе вбудовані камери та датчики, що дозволяють точно визначати позицію та рухи гравця у просторі без необхідності використання зовнішніх сенсорів чи базисних станцій. Контролери [Oculus Touch](#): [VR](#)-шолом поставляється з двома контролерами [Oculus Touch](#), які відображаються у віртуальному просторі та дозволяють користувачам взаємодіяти з об'єктами та середовищем у грі. Звукова система з тривимірним звуком: Вбудована звукова система [Meta Quest 2](#) забезпечує тривимірний звук, що дозволяє користувачам відчувати просторовий звуковий ефект у віртуальному середовищі. Процесор [Qualcomm Snapdragon XR2](#): Завдяки вбудованому процесору [Snapdragon XR2](#), [Meta Quest 2](#) має достатню обчислювальну потужність для обробки графіки та даних в реальному часі, що дозволяє запускати ігри та додатки високої якості без затримок. Ці технічні характеристики роблять [VR](#)-шолом [Meta Quest 2](#) привабливим вибором для користувачів, які шукають якісний іммерсійний досвід у віртуальній реальності, а також для розробників ігрових додатків, які

хочуть створювати контент для цієї платформи. Детальні характеристики описані на сайті виробника [12]. Особливості розробки ігрових додатків для віртуальної реальності Загальні принципи розробки **VR**-ігор Розробка **VR**-ігор передбачає створення ігрового середовища, яке дозволяє користувачам зануритися у віртуальний світ та взаємодіяти з ним максимально природно та інтуїтивно. Існують кілька принципів, яких необхідно дотримуватися для забезпечення якісного **VR**-досвіду: Реалістичне середовище: Важливо створити віртуальні світи, які виглядають і поведуться реалістично. Це стосується не тільки графіки, а й фізичних законів, освітлення, текстур та звукового супроводу. Чим більш реалістичним буде середовище, тим більш переконливим і занурюючим стане досвід користувачів. Природний взаємозв'язок: Взаємодія користувача з віртуальним середовищем має бути максимально природною та інтуїтивною. Це досягається шляхом використання контролерів та інших пристроїв введення, які відтворюють рухи та дії користувача у реальному часі. Інтерактивність: **VR**-ігри повинні пропонувати інтерактивний досвід, що дозволяє користувачам взаємодіяти з об'єктами, персонажами та середовищем. Інтерактивність може включати можливість маніпулювати об'єктами, спілкуватися з **NPC**, виконувати завдання та розгадувати головоломки. Занурення у гру: Важливим аспектом **VR**-ігор є забезпечення повного занурення користувача у віртуальний світ. Це включає не тільки візуальні та звукові ефекти, але й відчуття присутності, коли користувач повністю відволікається від реального світу та зосереджується на віртуальному. Збалансований контент: Розробники повинні забезпечити збалансований контент, щоб не переважувати користувачів сенсорними та когнітивними стимулами. Збалансований контент забезпечує комфортний та безпечний досвід користувачам. Плавність та продуктивність: Плавність графіки та звуку, а також висока продуктивність гри є важливими для забезпечення приємного **VR**-досвіду. Будь-які затримки або збої в роботі системи можуть негативно вплинути на занурення користувачів. Дотримання цих принципів є важливим для успішної розробки **VR**-ігор. Врахування особливостей **VR**-технологій дозволить створювати більш привабливі та інтерактивні ігрові досвіди, що залучатимуть користувачів та відкриватимуть нові можливості для розвитку ігрової індустрії. Детальніше про принципи розробки **VR**-ігор описано у [13]. Геймдизайн для **VR** Геймдизайн у **VR**-іграх значно відрізняється від традиційних ігрових підходів, оскільки він повинен забезпечувати максимальну зануреність і взаємодію користувачів із віртуальним світом. Кілька ключових аспектів геймдизайну для **VR** включають: Логічний та інтуїтивний інтерфейс: Розробка інтерфейсів для **VR** має відбуватися з урахуванням особливостей віртуального середовища та його впливу на користувачів. Інтерфейс повинен бути мінімалістичним, зрозумілим і логічним, уникаючи переваження користувачів зайвими елементами. Висока якість візуалізації: Візуальні елементи відіграють важливу роль у забезпеченні переконливого та захоплюючого **VR**-досвіду. Розробники мають приділяти особливу увагу деталізації та реалістичності візуалізації, використовуючи сучасні технології рендерінгу, текстуровання та освітлення. Інтерактивні механіки: Геймдизайн у **VR** має враховувати інтерактивність користувачів із віртуальним світом. Розробники повинні розробляти механіки гри, які дозволяють користувачам активно взаємодіяти з об'єктами та персонажами віртуального світу, наприклад, за допомогою маніпуляції об'єктами, дій, жестів та рухів. Сценарії та наратив: Розробка сценаріїв та наративу в **VR**-іграх має бути спрямована на забезпечення захоплюючого та взаємопов'язаного досвіду. Це включає створення цікавих історій, сюжетних ліній та персонажів, які залучають користувачів та спонукають їх досліджувати віртуальний світ. Поступове навчання та підказки: У **VR**-іграх користувачі можуть відчувати складнощі з орієнтуванням у віртуальному середовищі. Тому важливо забезпечити поступове навчання та надання підказок, щоб користувачі могли швидко адаптуватися до нового середовища та механік гри. Зворотний зв'язок та інформативність: Реакції гри на дії користувача, зворотний зв'язок та інформативність відіграють важливу роль у **VR**-іграх. Розробники повинні забезпечити чітке та своєчасне надання інформації користувачам про стан гри, їх дії та прогрес. Повага до фізіологічних обмежень: Важливо враховувати фізіологічні обмеження користувачів, наприклад, уникати рухів та ефектів, які можуть викликати нудоту або запаморочення. Розробники повинні дбати про комфорт користувачів. Геймдизайн для **VR**-ігор є важливим компонентом розробки, оскільки забезпечує користувачам максимально захоплюючий і природний досвід у віртуальному середовищі. Підхід до геймдизайну повинен бути ретельно продуманим, враховуючи всі особливості **VR**-технології, щоб створити успішну та привабливу гру. Мінімізація ризику нудоти та запаморочення Розробники **VR**-ігор мають враховувати безпеку та комфорт користувачів під час створення ігрових продуктів для віртуальної реальності. Важливо пам'ятати, що **VR**-середовище може призвести до різних фізіологічних реакцій, включаючи нудоту, запаморочення, головний біль або так званий

Цитування: 0,01%

id: 9

"**VR**-синдром".

Для зменшення неприємних відчуттів під час використання **VR**, слід уникати різких змін перспективи, руху камери або тривалих періодів без фіксованої точки зору. Варто підтримувати плавний рух камери і забезпечити стабільний зоровий контакт. Тому під час розробки **VR**-ігор необхідно враховувати такі аспекти: Оптимізація інтерфейсу та управління: Важливо розробити інтуїтивно зрозумілий інтерфейс, який не буде відволікати користувачів або ускладнювати їх взаємодію з грою. Управління має бути простим та зрозумілим, щоб користувачі не мали труднощів з виконанням необхідних дій.

Комфортність та налаштування гри: Розробники повинні надати можливість користувачам налаштувати ігровий досвід відповідно до своїх потреб та можливостей. Це може включати налаштування яскравості, контрастності, звуку та інших параметрів для підвищення комфорту користувачів. Врахування індивідуальних фізіологічних особливостей: Користувачі можуть мати різні фізіологічні особливості, тому важливо забезпечити індивідуальні налаштування гри для підвищення їхнього комфорту. Наприклад, регулювання швидкості гри або зміна перспективи. Рекомендації щодо часу гри: Розробники можуть надати рекомендації щодо оптимальної тривалості сеансу гри, щоб уникнути перевантаження користувачів та зменшити ризик появи негативних ефектів від тривалого використання [VR](#). Забезпечення безпеки та комфортності користувачів у [VR](#)-іграх є важливим елементом розробки, який сприяє позитивному досвіду користувачів та підвищує якість гри. Врахування цих аспектів допоможе створити [VR](#)-гру, яка буде зручна та безпечна для різних категорій користувачів. Про те як уникати користувачам заколисування, та що робити розробникам, для недопущення

Цитування: 0,03%

id: 10

«синдрому-[VR](#)», - описано у [14]

Розробка інтерактивного контенту Розробка інтерактивного контенту у [VR](#)-іграх є ключовим елементом, оскільки дозволяє користувачам взаємодіяти з віртуальним середовищем та його елементами, що забезпечує більш захоплюючий та реалістичний досвід. Це може включати: Інтерактивні об'єкти: Об'єкти у грі повинні бути взаємодійними, дозволяти користувачам підбирати, переміщувати або маніпулювати ними. Це може бути реалізовано через різноманітні механізми введення, такі як контролери або інші пристрої, що відстежують рухи користувача. Віртуальні персонажі ([NPC](#)): Розробка [NPC](#), з якими користувачі можуть взаємодіяти, додає рівня складності та занурення в гру. [NPC](#) можуть виконувати різні ролі, наприклад, супроводжувати гравця, нападати або надавати інформацію за завдання. Сценарії та події: Інтерактивні сценарії та події допомагають підтримувати інтерес користувачів до гри та дозволяють їм впливати на розвиток сюжету. Це може бути досягнуто шляхом надання користувачам можливості приймати рішення та реагувати на різні ситуації. Динамічне середовище: Створення динамічного середовища, яке реагує на дії користувача, підвищує рівень взаємодії та занурення у гру. Наприклад, зміна погоди, часу доби або взаємодія з оточенням можуть надати грі більш реалістичний вигляд. Головоломки та завдання: Додавання інтерактивних головоломок та завдань може стимулювати користувачів до дослідження світу гри та взаємодії з різними об'єктами та механіками. Інтерактивний контент дозволяє користувачам занурюватися у гру, взаємодіяти з об'єктами, [NPC](#) та сценаріями, що забезпечує захоплюючий і реалістичний досвід. Врахування цих особливостей під час розробки [VR](#)-ігор є ключем до створення успішних проектів, які будуть залучати та утримувати гравців. Про це, та багато інших важливих моментів для створення інтерактивного [VR](#)-контенту можна прочитати у [15]. Методи взаємодії користувача з ігровим середовищем у [VR](#) Віртуальна реальність ([VR](#)) відкриває широкі можливості для інтерактивної взаємодії користувача з ігровим середовищем. В розробці ігрових додатків для [VR](#) використовуються різноманітні методи взаємодії, що роблять гру більш імерсійною та захоплюючою. Роки досліджень створили безліч методів взаємодії для [VR](#), і ці методи підтримують такі типи дій як: пересування, вибір та маніпуляція. Детальніше у [16]. Давайте розглянемо деякі з найпоширеніших методів взаємодії у [VR](#): Контролери руху: Контролери руху, які супроводжуються [VR](#)-шоломами, дозволяють користувачам взаємодіяти з об'єктами у віртуальному середовищі за допомогою реальних рухів. Вони можуть використовуватися для взаємодії з об'єктами, переміщення гравця та виконання різноманітних дій. Жести та міміка: Деякі ігри для [VR](#) використовують жести та міміку обличчя користувача для управління грою або взаємодії з персонажами. Це додає реалістичності та глибини до взаємодії у віртуальному світі. Голосові команди: Окремі [VR](#)-ігри підтримують введення команд за допомогою голосу. Користувач може використовувати голосові команди для керування персонажем, активації спеціальних вмінь або взаємодії з навколишнім середовищем. Технологія відслідковування рухів: Системи відслідковування рухів, такі як [Oculus Insight](#), використовують камери та сенсори для відслідковування рухів користувача у просторі. Це дозволяє реалізувати взаємодію користувача з об'єктами у віртуальному середовищі за допомогою рухів тіла. Тактильні відчуття: Деякі [VR](#)-ігри використовують тактильні відчуття, такі як вібрація контролерів або вібраційні ободи, що надають гравцеві фізичні відчуття взаємодії з об'єктами у віртуальному світі. Разом з перевагами кожного методу, існують і певні виклики та обмеження. Наприклад, точність відслідковування рухів може бути обмеженою в деяких сценаріях, а використання голосових команд може бути неефективним у шумних оточеннях. Також важливо враховувати індивідуальні можливості користувачів та їх зручність у використанні певного методу взаємодії. Отже, вибір методу взаємодії залежить від конкретних потреб гри та користувальницького досвіду. Висновки до розділу 1 Розглянувши теоретичні основи розробки ігрових додатків для віртуальної реальності ([VR](#)) та детально проаналізувавши технічні характеристики та особливості [VR](#)-шолому [Meta Quest](#), ми дійшли наступних висновків: По-перше, розуміння принципів функціонування віртуальної реальності та її основних концепцій є ключовим для успішної розробки ігрових додатків у цій області. Імерсійний досвід, який [VR](#) може забезпечити, вимагає ретельного вивчення та врахування особливостей взаємодії користувача з віртуальним середовищем. По-друге, технічні характеристики [VR](#)-шолому [Meta Quest](#) роблять його привабливим вибором для розробки ігор та додатків. Його висока роздільна здатність, система

відслідковування рухів та вбудований процесор забезпечують стабільний та реалістичний ігровий досвід. По-третє, розробка ігрових додатків для віртуальної реальності потребує уважної уваги до особливостей взаємодії користувача з ігровим середовищем. Методи взаємодії, такі як контролери руху, жести, голосові команди та інші, відіграють важливу роль у створенні емоційно зворушливого та захоплюючого геймплею. Отже, розуміння та врахування цих аспектів допоможе забезпечити успішну розробку ігрового додатку для [VR-шолому Meta Quest](#), який буде надійно відповідати вимогам і очікуванням користувачів.

РОЗДІЛ 2. ПІДГОТОВЧІ ЕТАПИ ДО РОЗРОБКИ ДОДАТКУ Концепція та ідея ігрового додатку Під час виконання дипломної роботи була розроблена концепція проекту. А саме: Назва гри:

Цитування: 0,02%

id: 11

"Fitness Runner VR"

Події та жанр гри:

Цитування: 0,02%

id: 12

"Fitness Runner VR"

- це ігровий додаток, що поєднує в собі елементи нескінченного раннера та фітнес ігор. Гравець виступає в ролі людини, яка опиняється у всесвіті своїх снів, де йому належить пробігти якомога більшу дистанцію у боротьбі з великим ведмедем, що його переслідує. Події відбуваються у місті, яке зазнало страшної снігової бурі, де гравець повинен уникати перешкод та вбивати ворогів, щоб вижити. Тематика гри викликає адреналін та спонукає користувача активно рухатися, що відповідає концепції фітнес-ігор. Головна ідея: Головною ідеєю

Цитування: 0,02%

id: 13

"Fitness Runner VR"

є створення іммерсивного фізичного досвіду, де гравець взаємодіє з ігровим середовищем, виконуючи рухи, які співвідносяться з бігом у реальному житті. Швидкість бігу у грі контролюється інтенсивністю рухів контролерами вгору-вниз, що надає реалістичний фітнес-досвід та сприяє покращенню фізичної активності гравця. Геймплей: Гравець повинен активно керувати бігом свого персонажу, виконуючи інтенсивні рухи контролерами вгору-вниз для прискорення. Під час бігу гравець повинен уникати перешкод та ворогів, що зустрічає на своєму шляху, а також збирати монети, які у майбутньому зможе витратити у ігровому магазині. Великий акцент робиться на реалістичності фізичної взаємодії та фітнес-елементів гри. Цільова аудиторія: Гра призначена для широкої аудиторії, включаючи фанатів нескінченних раннерів, любителів фітнес-ігор та тих, хто просто шукає іммерсивний ігровий досвід у віртуальній реальності. Вона особливо приваблива для тих, хто хоче поєднати розвагу з фізичною активністю. Технічні обмеження та можливості: Віртуальний простір: Мета [Quest 2](#) має обмежений обсяг віртуального простору, який варто використовувати раціонально. Важливо розробити геймплей та архітектуру рівню так, щоб він був оптимізований для використання обмеженого простору та не заважав користувачеві. Відслідковування рухів: Оскільки гра вимагає інтенсивних рухів контролерами, важливо врахувати точність та надійність системи відслідковування рухів. Недоліки в цій системі можуть вплинути на реалізм та задоволення від гри. Енергоефективність: Розробка гри повинна враховувати енергоефективність, оскільки це важливий аспект для мобільних пристроїв, таких як [Meta Quest](#). Мінімізація використання ресурсів та оптимізація програмного коду допоможе зберегти заряд батареї пристрою. Врахування цих технічних обмежень та можливостей допоможе забезпечити ефективну розробку та оптимізацію ігрового додатку, забезпечуючи оптимальний досвід гравця. Вибір технологій та інструментів для розробки Під час розробки ігрового додатку для [VR](#), важливо правильно підібрати необхідні технології та інструменти для ефективного та продуктивного процесу створення продукту. У цьому розділі ми розглянемо загальний спектр технологій та інструментів, необхідних для розробки гри для віртуальної реальності, а також проведемо вибір тих, що найкраще відповідають потребам нашого проекту. Загальний спектр необхідних технологій та інструментів Розробка ігор для віртуальної реальності вимагає комплексного підходу та вибору оптимальних технологій та інструментів. Першим кроком розглянемо загальний спектр технологій та інструментів, які знадобляться для успішного втілення проекту: Інтегроване середовище розробки ([IDE](#)): Важливо мати зручне та функціональне середовище розробки, яке дозволяє ефективно працювати з кодом, графікою та аудіо складовими проекту. Мова програмування: Вибір мови програмування визначає зручність та продуктивність розробки. Різні рушії підтримують різні мови програмування, такі як [C#](#), [C++](#), [JavaScript](#) тощо. Графічний редактор: Важливо мати доступ до потужного графічного редактора для створення моделей, текстур та анімацій. [VR](#)-платформи та [SDK](#): Оскільки наша гра буде розроблена для [VR-шолому Meta Quest 2](#), необхідно вибрати рушій та [SDK](#), які підтримують цю платформу та дозволяють ефективно взаємодіяти з її функціоналом. Розглянувши ці аспекти, ми зможемо об'єктивно оцінити можливості різних рушіїв для розробки [VR-ігор](#), та вибрати найкращий варіант для нашого проекту. Вибір рушія для розробки гри У світі віртуальної реальності, вибір правильного рушія для розробки ігор може кардинально вплинути на успіх проекту. Три з найбільш відомих та потужних рушіїв у цій області - [Unity](#), [Unreal Engine](#) та [CryEngine](#) - надають розробникам різні інструменти та можливості для створення іммерсивних [VR-ігор](#). [Unity](#): [Unity](#) - один із найпопулярніших

рушіїв у світі геймдеву, відомий своєю легкістю в освоєнні, широкою підтримкою платформ та активною спільнотою розробників. [Unity](#) надає зручний інтерфейс та різноманітні інструменти для створення ігор у віртуальній реальності, що робить його відмінним вибором для початківців та досвідчених розробників одночасно. Для чого використовують [Unity](#) більш детально описано у [1]. [Unreal Engine](#): [Unreal Engine](#) - ще один потужний рушій, відомий своєю неймовірною графікою та високоякісними візуальними ефектами. Завдяки широкому функціоналу та гнучкості налаштувань, [Unreal Engine](#) дозволяє створювати складні та реалістичні ігри віртуальної реальності, привертаючи до себе увагу професіоналів геймдеву. Як працює [Unreal Engine](#) дізнатись можна в [2]. [CryEngine](#): [CryEngine](#) - цей рушій відомий своєю вражаючою графікою та реалістичними світами, що робить його привабливим вибором для розробки ігор у віртуальній реальності. Завдяки потужним інструментам та гнучкому відкритому коду, [CryEngine](#) надає розробникам можливість створювати неймовірні ігрові враження, які занурюють гравців у світ віртуальної реальності. Всю необхідну інформацію про цей рушій можна знайти у [3]. Далі ми розглянемо кожен з цих рушіїв детальніше, проаналізуємо їхні переваги та недоліки, щоб визначити, який з них найкраще підходить для нашого проекту

Цитування: 0,02%

id: 14

"[Fitness Runner VR](#)":

[Unity](#): Плюси: Легкість в освоєнні: [Unity](#) має дружній інтерфейс, широкий набір документації та онлайн-ресурсів, що дозволяє швидко освоювати платформу навіть для початківців. Широкий вибір платформ: [Unity](#) підтримує багато різних [VR](#)-платформ, включаючи [Oculus Rift](#), [HTC Vive](#), [PlayStation VR](#) та [Meta Quest](#). Активна спільнота: [Unity](#) має велику та активну спільноту розробників, яка надає підтримку та спільну діяльність у вирішенні проблем та пошуку рішень. Мінуси: Менша якість графіки: У порівнянні з [Unreal Engine](#), [Unity](#) може мати менш вражаючу графіку та меншу продуктивність при роботі з великими обсягами даних. Додаткові витрати: Деякі розширення та функціонал можуть вимагати додаткових витрат, що може збільшити вартість розробки. [Unreal Engine](#): Плюси: Вражаюча графіка: [Unreal Engine](#) відомий своєю вражаючою графікою та реалістичними ефектами, що робить його ідеальним для створення відмінних [VR](#)-ігор з високою якістю візуальних ефектів. Великий функціонал: [Unreal Engine](#) має широкий функціонал та гнучкість налаштувань, що дозволяє розробникам створювати складні та інноваційні проекти. Безкоштовний доступ до джерела: [Unreal Engine](#) має безкоштовний доступ до вихідних кодів, що дозволяє власникам проектів вносити власні зміни та адаптувати движок під свої потреби. Мінуси: Складність в освоєнні: У порівнянні з [Unity](#), [Unreal Engine](#) може бути складнішим у освоєнні для новачків та вимагати більше часу для вивчення. Великі вимоги до обладнання: [Unreal Engine](#) може вимагати потужного обладнання для роботи, що може призвести до додаткових витрат. [CryEngine](#): Плюси: Графіка високої якості: [CryEngine](#) відомий своєю вражаючою графікою та високою якістю візуальних ефектів. Потужність: [CryEngine](#) має потужні інструменти для реалізації реалістичних світів та ефектів, що робить його привабливим для амбіційних проектів. Гнучкість відкритого коду: Доступ до вихідних кодів дозволяє розробникам змінювати та адаптувати движок під свої потреби. Мінуси: Складність в освоєнні: Подібно до [Unreal Engine](#), [CryEngine](#) може бути складним у освоєнні та вимагати більше часу для вивчення. Обмежений вибір платформ: У порівнянні з [Unity](#) та [Unreal Engine](#), [CryEngine](#) має меншу кількість підтримуваних платформ, що може обмежити його застосування в певних проектах. Цей аналіз допоможе нам краще зрозуміти переваги та недоліки кожного рушія та обрати той, який найкраще підходить для нашого проекту

Цитування: 0,02%

id: 15

"[Fitness Runner VR](#)".

Висновок вибору рушія Після детального аналізу рушіїв для розробки віртуальної реальності, [Unity](#) виявився оптимальним вибором для нашого проекту

Цитування: 0,02%

id: 16

"[Fitness Runner VR](#)".

Незважаючи на те, що [Unreal Engine](#) та [CryEngine](#) мають свої переваги, [Unity](#) пропонує ряд важливих переваг, які роблять його ідеальним вибором для нашого проекту: Легкість в освоєнні. Широкий вибір платформ. Активна спільнота розробників. Ефективне використання ресурсів. Отже, на основі проведеного аналізу ми приймаємо рішення використовувати [Unity](#) для розробки нашого проекту

Цитування: 0,02%

id: 17

"[Fitness Runner VR](#)",

вважаючи його ідеальним вибором для створення іммерсивних [VR](#)-ігор без великої команди. Вибір [SDK](#) для рушія При виборі [SDK](#) для розробки гри для [VR](#)-шолому [Meta Quest](#), ми розглядали кілька варіантів і проаналізували їхні переваги та недоліки: [XR Interaction Toolkit](#): Переваги: Спеціально розроблений для створення віртуальної реальності у середовищі [Unity](#). Легкий в освоєнні та використанні, з добре документованою [API](#) та активною спільнотою розробників. Підтримка [Meta Quest](#) та інших популярних [VR](#)-платформ. Широкий набір готових компонентів для реалізації різних видів взаємодії та ефектів у віртуальній реальності. Недоліки: Деякі функції можуть бути обмежені в порівнянні з більш розширеними [SDK](#). [VIVE Input Utility](#): Переваги: Розроблений [HTC](#) для

роботи зі [SteamVR](#) та [VIVE](#) в середовищі [Unity](#). Має додаткові можливості для роботи з руками та контролерами у віртуальній реальності. Недоліки: Обмежена підтримка інших платформ, окрім [HTC Vive](#). [Oculus Integration](#): Переваги: Спеціально розроблений для роботи з [Oculus Rift](#) та [Meta Quest](#) в [Unity](#). Глибока інтеграція з функціями та можливостями [Oculus SDK](#). Недоліки: Може бути менш гнучким у порівнянні з більш універсальними [SDK](#), як

Цитування: **0,02%**

id: **18**

«[XR Interaction Toolkit](#)».

Після ретельного аналізу ми вирішили використовувати [XR Interaction Toolkit](#) через його широкий функціонал, легкість використання та підтримку різних [VR](#)-платформ, включаючи [Meta Quest](#). Цей [SDK](#) найбільш відповідає нашим потребам у розробці гри для даної платформи, надаючи нам всі необхідні інструменти та підтримку для створення високоякісного і іммерсивного віртуального досвіду не тільки для шоломів [Oculus](#), а й в подальшому інших пристроїв віртуальної реальності. Завантажити та ознайомитись детальніше з цим [SDK](#) можна у [5]. Створення проекту в [Unity](#) та підключення [SDK](#) Оскільки ми зупинилися на рушії [Unity](#), першим кроком було завантаження [Unity Hub](#) та встановлення необхідної версії рушія [Unity](#). Важливо було додати підтримку [Android](#) платформи, оскільки шолом [Meta Quest](#) побудований на платформі [Android](#). Тому ми також встановили компоненти [OpenJDK](#), та [Android SDK & NDK Tools](#). При створенні проекту ми обрали його тип як

Цитування: **0,01%**

id: **19**

"[VR Core](#)".

Після створення були проведені необхідні базові налаштування, що включали: Встановлення параметрів графіки та продуктивності для оптимальної роботи в [VR](#)-середовищі. Підключення та конфігурацію [XR Interaction Toolkit](#) для забезпечення взаємодії з [VR](#)-шоломом [Meta Quest](#). Налаштування контролерів та інших елементів управління для коректної роботи у [VR](#). Ці кроки забезпечили базову готовність проекту до подальшого розвитку та тестування. Архітектура ігрового додатку та взаємодія з [VR](#)-шоломом Під час розробки гри важливим етапом буде створення ефективної архітектури додатку, яка забезпечить оптимальну взаємодію з обладнанням та створення іммерсивного віртуального досвіду для користувачів. Розглянемо загальну архітектуру нашого ігрового додатку та технічні аспекти його взаємодії з [VR](#)-шоломом [Meta Quest](#): Архітектурний підхід: Ми виберемо архітектурний підхід, який дозволить нам створити модульну та розширювану систему. Гра буде складатися з різних модулів, таких як модуль управління персонажем, модуль обробки взаємодії з об'єктами та модуль управління інтерфейсом. Взаємодія з [VR](#)-шоломом [Meta Quest](#): Для забезпечення взаємодії гравця з грою ми будемо використовувати [API](#) та функціонал [SDK](#) для [Meta Quest](#), включаючи [XR Interaction Toolkit](#). Це дозволить нам отримати доступ до даних з контролерів, відстежування руху гравця та інших функцій, необхідних для створення інтерактивного досвіду. Візуалізація в [VR](#): Для створення іммерсивної графіки та візуалізації в [VR](#) ми будемо використовувати різноманітні техніки, такі як стереоскопічний рендеринг, обробка освітлення та тіней, а також оптимізація для плавного відтворення на [VR](#)-шоломі. Управління користувачем: Важливим аспектом гри в [VR](#) є інтуїтивне управління для користувача. Ми будемо використовувати рухи контролерів [Meta Quest](#) для керування рухами персонажа та взаємодії з об'єктами у віртуальному середовищі. Отже, в цьому підрозділі ми розглянемо архітектурні рішення та технічні деталі, що стосуються взаємодії нашого ігрового додатку з [VR](#)-шоломом [Meta Quest](#). Вибір архітектурного підходу В розробці ігор для віртуальної реальності використовуються різні архітектурні підходи. Розглянемо декілька з найпоширеніших архітектурних підходів у розробці ігор для віртуальної реальності: [MVC](#) ([Model-View-Controller](#)): Цей підхід відділяє дані (модель), відображення (вид) та логіку (контролер). Модель відповідає за управління даними та бізнес-логікою, вид відповідає за представлення даних користувачеві, а контролер відповідає за взаємодією між моделлю та видом. Детальніше ознайомитись з [MVC](#) можна у [6]. [Entity-Component-System](#) ([ECS](#)): Цей підхід розглядає всі об'єкти в грі як сукупність сутностей ([entities](#)), компонентів ([components](#)) та систем ([systems](#)). Компоненти містять дані, системи виконують операції над цими даними, а сутності є просто контейнерами, що об'єднують компоненти разом. Всю необхідну інформацію про [ECS](#) можна знайти у [7]. [Behavior Tree](#): Цей підхід використовує деревоподібну структуру для управління поведінкою об'єктів у грі. Кожен вузол в дереві представляє певну дію або умову, і граф може виконуватися відповідно до результату обходу цього дерева. Дізнатись більше про дерево поведінки можна ознайомившись з [8]. [Data-Oriented Design](#) ([DOD](#)): Цей підхід ставить у центр уваги оптимізацію роботи з даними. Він орієнтований на ефективне використання пам'яті та оптимізацію шляхом групування та обробки даних пакетами. Детально про цей підхід описано у [9]. [Component-Based Architecture](#): Цей підхід полягає в побудові програми з невеликих, повторно використовуваних компонентів, які можуть бути динамічно додаваними та змінюваними. Компоненти можуть включати функціональність, візуальні ефекти, фізичні властивості та інше. Кожен з цих підходів має свої переваги та недоліки, і вибір конкретного підходу залежить від вимог проекту, розміру команди, досвіду розробника та інших факторів. У нашому випадку ми оберемо підхід з використанням [Component-Based Architecture](#), оскільки він надає гнучкість та можливість легко розширювати нашу систему, що є важливим для розробки ігор у віртуальній реальності. Корисна інформація для розуміння цього підходу надана у [10]. Взаємодія з [VR](#)-шоломом [Meta Quest](#) В нашому проекті взаємодія з [VR](#)-

шоломом [Meta Quest](#) відіграє ключову роль у створенні іммерсивного ігрового досвіду для користувачів. Нижче представлено основні методи взаємодії з цим шоломом, що використані в нашому проекті: Контролери руху: Використання контролерів руху дозволяє гравцеві взаємодіяти з ігровим світом. У нашому проекті контролери руху використовуються для керування рухом персонажа, взаємодії з об'єктами та виконання майже всіх ігрових дій. 6DoF відстеження: [VR](#)-шолом [Meta Quest](#) підтримує 6DoF відстеження, що дозволяє точно відслідковувати рухи користувача у просторі. Це додає до ігрового досвіду реалістичність та свободу рухів. У нашій грі можна вільно переміщуватись в будь-якому напрямку, за потреби. Відтворення звуку: Інтеграція аудіо файлів та звукових ефектів забезпечує аудіо імєрсію гравця. Ми використовуємо вбудовані можливості [Meta Quest](#) для відтворення звуків та музики, відповідно до сценарію гри. Графічний контент: Оптимізація графічного контенту забезпечує плавну роботу ігрового середовища на шоломі [Meta Quest](#). Ми використовуємо технології, які дозволяють досягти оптимального балансу між якістю графіки та продуктивністю. Оптимізація продуктивності: Для забезпечення оптимальної продуктивності гри на [VR](#)-шоломі [Meta Quest](#) ми використовуємо низько-рівневі оптимізації, такі як зменшення полігонів моделей та оптимізація текстур. Наш підхід до взаємодії з [VR](#)-шоломом [Meta Quest](#) дозволяє створювати ігровий досвід, який є одночасно захоплюючим, реалістичним та оптимізованим для платформи віртуальної реальності. Візуалізація в [VR](#) Для нашого проекту

Цитування: 0,02%

id: 20

"Fitness Runner VR"

будуть використані наступні технічні аспекти, що впливають на створення іммерсивного досвіду: Стереоскопічний рендеринг: Один із основних аспектів візуалізації в [VR](#) - це стереоскопічний рендеринг. Він дозволяє створити ілюзію просторовості та глибини, використовуючи два окремі зображення для кожного ока користувача. Ми будемо використовувати цей підхід для створення реалістичної глибини і відчуття присутності у віртуальному світі. Багато корисної інформації обговорено у [11]. Оптимізація графічного контенту: Оскільки [VR](#) вимагає великої кількості обчислень для рендерингу зображення в реальному часі, важливо провести оптимізацію графічного контенту для забезпечення плавного та стабільного відтворення на [VR](#)-шоломі. Ми будемо використовувати техніки оптимізації, такі як відстань до появи та зникання об'єктів, різні рівні деталізації, оптимізація освітлення та використання техніки масштабування для забезпечення високої продуктивності. Обробка освітлення та ефекти: Ефективне використання освітлення та візуальних ефектів грає важливу роль у створенні іммерсивного віртуального світу. Ми будемо використовувати різноманітні техніки обробки освітлення, такі як високо динамічний діапазон освітлення ([HDR](#)), тінь, реалістичне відображення матеріалів та спеціальні ефекти, щоб зробити гру більш виразною та привабливою. Інтерфейс користувача в [VR](#): Розробка інтерфейсу користувача в [VR](#) відбувається з урахуванням особливостей віртуального середовища. Ми будемо створювати інтуїтивний та ергономічний інтерфейс, що відповідає специфіці [VR](#), зокрема розміщення елементів у просторі та методи взаємодії з ними за допомогою контролерів [Meta Quest](#). У цьому підрозділі детально розглянуто технічні аспекти візуалізації в [VR](#) для нашого проекту, включаючи техніки рендерингу, оптимізації графічного контенту, обробки освітлення, ефектів, та розробку інтерфейсу користувача. Управління користувачем Управління користувачем в ігровому додатку для віртуальної реальності ([VR](#)) відіграє критичну роль у створенні іммерсивного та захоплюючого досвіду для гравців. Розглянемо різноманітні аспекти управління користувачем в нашому проекті

Цитування: 0,02%

id: 21

"Fitness Runner VR"

та розкриємо технічні деталі, які впливають на спосіб взаємодії гравця з віртуальним середовищем: Взаємодія з контролерами: Оскільки [VR](#)-шолом [Meta Quest](#) включає в себе контролери, які розпізнають рухи та жести, ми реалізуємо взаємодію гравця з віртуальним середовищем за допомогою цих контролерів. Гравець зможе керувати персонажем та взаємодіяти з об'єктами у грі, використовуючи рухи рук, натискання клавш і жести. Рух та переміщення: Управління переміщенням гравця у віртуальному середовищі відбуватиметься шляхом рухів контролерами. Ми реалізуємо це, використовуючи вбудовані функції [VR](#)-шолому [Meta Quest](#) та інструменти розробки, які забезпечують стабільність та плавність рухів. Взаємодія з об'єктами: Гравець зможе взаємодіяти з різними об'єктами у віртуальному світі, такими як перешкоди, збирання предметів або зброя, за допомогою контролерів. Ми реалізуємо механіку, яка передає реалістичність та відчуття взаємодії з об'єктами. Управління меню та інтерфейсом: Для навігації по меню та виконання різних дій (наприклад, завантаження рівня, перезапуск або вихід з гри) ми розробимо інтерфейс, який відповідає специфіці [VR](#)-середовища. Це включає в себе використання жестів для наведення на потрібну кнопку, та натискання тригерів. У цьому підрозділі розглянуто різноманітні аспекти управління користувачем в нашому ігровому додатку для [VR](#)-шолому [Meta Quest](#), включаючи взаємодію з контролерами, управління переміщенням та рухом, взаємодію з об'єктами, керування інтерфейсом та навігаційні елементи, спрямовані на створення реалістичного та іммерсивного досвіду для гравців. Висновки до розділу 2 У другому розділі було детально розглянуто підготовчі етапи до розробки ігрового додатку для [VR](#)-шолому [Meta Quest](#) 2. Спершу ми дослідили концепцію та ідею гри, визначивши її жанр, головну ідею, геймплей, цільову аудиторію, а також технічні обмеження та

можливості проекту. Далі було здійснено вибір технологій та інструментів для розробки, включаючи вибір рушія [Unity](#) та відповідного [SDK](#). Ми описали процес завантаження та налаштування проекту в [Unity](#), а також розглянули архітектурний підхід і взаємодію з [VR](#)-шоломом [Meta Quest](#). Завдяки систематичному підходу до вибору інструментів та визначенню архітектури, ми заклали міцну основу для подальшої розробки та реалізації ігрового додатку, що забезпечить високу якість та інтерактивність користувацького досвіду у [VR](#). РОЗДІЛ 3. РОЗРОБКА ІГРОВОГО ДОДАТКУ ДЛЯ [VR](#) ШОЛОМУ [META QUEST](#)

Реалізація графічного контенту Перша частина розробки нашого проекту починалась з моделювання графічних компонентів. У цьому розділі ми познайомимось з основними аспектами, а також зазначимо ключові мети та завдання, які стоять перед нами у процесі реалізації графічного контенту для нашого проекту. Ключові цілі та завдання Було виділено наступні ключові мети та завдання для розробки проекту

Цитування: 0,02%

id: 22

«[Fitness runner VR](#)»:

Створення реалістичного та іммерсивного віртуального світу: Головною метою є створення вражаючого та живого ігрового середовища, яке повністю поглине гравця та дасть йому відчуття присутності у грі. Оптимізація графічного контенту: Ми ставимо перед собою завдання забезпечити високу якість графічного контенту, одночасно оптимізуючи його для ефективної роботи на пристроях та забезпечення плавного геймплею. Створення ефективної системи взаємодії з гравцем: Ми прагнемо до створення інтуїтивної та зручної системи управління та взаємодії з гравцем, яка дозволить забезпечити комфортний та натуральний геймплей у віртуальному середовищі. Забезпечення відповідності до тематики та атмосфери гри: Наша мета - створити графічний, який ідеально відображатиме тематику та атмосферу гри

Цитування: 0,02%

id: 23

"[Fitness Runner VR](#)",

доповнюючи та підсилюючи ігровий досвід гравця. Моделювання та текстурування об'єктів Для створення графічного контенту у нашій грі

Цитування: 0,02%

id: 24

"[Fitness Runner VR](#)"

ми використовували програму [Blender](#). [Blender](#) - це потужний та безкоштовний інструмент для 3D-моделювання, який надає широкі можливості у створенні різноманітних об'єктів, персонажів та архітектурних елементів. Детальніше з цим тривимірним графічним редактором можна ознайомитись на офіційному сайті [12]. Процес створення 3D моделей складався з двох етапів, а саме: Створення 3D моделей: У процесі моделювання, ми почали розробку моделі з базової форми - куба. За допомогою різноманітних інструментів [Blender](#), таких як [Extrude](#) (Видавити), [Scale](#) (Масштабувати), [Rotate](#) (Розвернути) та [Knife](#) (Ніж), ми перетворювали цей куб на складніші форми та ландшафти, в результаті отримавши моделі потрібної нам форми (рис.3.1.) Рис. 3.1. Модель земної поверхні з провалом ґрунту, без текстури Текстурування моделей: Для текстурування моделей, заздалегідь була завантажена палітра кольорів, з відкритих джерел інтернету. Для оптимізації гри - ця палітра використовується на багатьох об'єктах одночасно. Для розмалювання моделей - ми використовували інструмент [Shading](#) у [Blender](#). Визначали зображення з палітрою кольорів, як [Base Color](#) для наших моделей, після чого, в інструменті [UV Editing](#), обирали полігони моделі, та переміщували їх на потрібний нам колір. В результаті отримували завершені 3D моделі, які в подальшому стануть складовою частиною нашого ігрового світу у віртуальній реальності. Створення частин рівня У нашій грі, була реалізована концепція нескінченного рівня. Це було досягнуто завдяки системі генерації рівня з різноманітних тайлів (заготовлені частини, зібрані з багатьох об'єктів в один), які послідовно розміщуються на сцені під час гри. Кожен тайл включає в себе різні елементи, такі як ділянки дороги, декоративні об'єкти, зони для спавну будівель, перешкод та монет. Ці елементи були зібрані з різних моделей, які були створені в самому рушії [Unity](#), або імпортовані з програми моделювання [Blender](#). Для створення тайлів, ми використовували різноманітні об'єкти, такі як заготовлені префаби землі з дорогами, будівлі, та різноманітні перешкоди, які були оброблені та деталізовані під єдиний стиль, щоб створити імітацію нескінченної дороги та оточуючого середовища, схожого на вулицю в місті. Після підготовки всіх складових - кожен тайл був зібраний з цих елементів у відповідному порядку, щоб створити реалістичний та різноманітний ігровий світ (рис. 3.2.). На малюнку (рис. 3.2.) ми бачимо один із трьох видів зібраних нами, тайлів для гри. Ми можемо помітити об'єкти, з яких складається дорога, кілька автомобілів вдалині та підйоми в гору, по обидва боки дороги. У ієрархії проекту є три групи (рис. 3.3.):

Цитування: 0,01%

id: 25

"[Obstacle_Spawn_Points](#)"

(група точок для появи перешкод),

Цитування: 0,01%

id: 26

"[Bonuses_Spawn_Points](#)"

(група точок для появи монет) та

Цитування: 0,01%

id: 27

"Buildings_Spawn_Points"

(група точок для появи будівель). Рис. 3.2. Приклад одного тайлу для гри Рис. 3.3. Групи точок для появи відповідних об'єктів на їх координатах Кожна з цих груп містить пусті, прозорі моделі, які служать координатами появи відповідних об'єктів. Такий підхід був зроблений для оптимізації гри та створення ефекту нескінченності. Завдяки тому що ми завчасно знаємо координати розташування цих точок на тайлах, ми можемо за допомогою коду забезпечити появу потрібних об'єктів у перевірених місцях, і лише в області нашого поля зору. Це дозволяє зменшити навантаження на пристрій, вивантажуючи одночасно меншу кількість моделей на сцену. Крім того, ці точки використовуються щоб об'єкти, що з'являються на них, - обирались завжди у абсолютно випадковому порядку, зі списку доступних. Таким чином, гравець отримує можливість досліджувати нескінченний рівень, що складається всього з трьох варіантів тайлів, при цьому зовсім не помічаючи між ними схожості та переходу, бо кожен відрізок завжди є унікальним, не залежно від порядку появи на сцені, оскільки майже всі об'єкти на ньому з'являються у випадковому порядку. Програмування випадкової генерації рівня Щоб імплементувати одну з ключових механік гри, а саме випадкову генерацію нескінченного рівня, ми звернулися до написання скриптів для графічних об'єктів, сцени та тайлів. Основний скрипт

Цитування: 0,01%

id: 28

"Tile_Generator"

- відповідає за генерацію тайлів. Ми зробили на сцені порожній об'єкт і додали до нього скрипт

Цитування: 0,01%

id: 29

"TileGenerator".

Розберемо ключові методи генерації тайлів, що були написані в скрипті: Ініціалізація: `void Start()` { // Згенерувати початкову кількість тайлів `for (int i = 0; i < startTiles; i++)` { // Якщо це перший тайл, згенерувати тайл-початок `if (i == 0) SpawnTile(3);` // Згенерувати випадковий тайл зі списку префабів `SpawnTile(Random.Range(0, tilePrefabs.Length));` } } У методі `Start()` з циклу `for` згенеровано початкову кількість тайлів (`startTiles`). Перший тайл (з індексом 3) є тайлом-початком, а решта тайлів випадковим чином обираються зі списку префабів. Початковий тайл є найбільш розвантаженим, і розробленим спеціально під коректну появу на ньому персонажа на початку гри. Оновлення: `void Update()` { // Якщо позиція гравця перевищує позицію спавну наступного тайла `if (player.position.z - 60 > spawnPos + (startTiles * tileLength))` { // Згенерувати новий тайл `SpawnTile(Random.Range(0, tilePrefabs.Length));` // Видалити найстаріший тайл `DeleteTile();` } } Метод `Update()` перевіряє, чи поточна позиція гравця перевищує позицію, де потрібно згенерувати наступний тайл. Якщо так, то генерується новий тайл і видаляється найстаріший, щоб уникнути перевищення кількості активних тайлів, для оптимізації проекту. Генерація: // Метод для генерації тайла з вказаним індексом `private void SpawnTile(int tileIndex)` { // Створити новий тайл на позиції спавну `GameObject nextTile = Instantiate(tilePrefabs[tileIndex], transform.forward * spawnPos, transform.rotation);` // Додати тайл до списку активних тайлів `activeTiles.Add(nextTile);` // Збільшити позицію спавну для наступного тайла `spawnPos += tileLength;` } Метод `SpawnTile(int tileIndex)` відповідає за створення нового тайлу за допомогою функції `Instantiate`. Він отримує індекс префаба тайла з масиву `tilePrefabs`, встановлює позицію спавну за допомогою `transform.forward * spawnPos` і додає створений тайл до списку активних тайлів. Повний код можна побачити у додатку В. Після розробки скрипту – він був налаштований на сцені, в інспекторі об'єкта до якого він прикріплений. Було додано префаби тайлів, та камеру, яка буде відповідати за координати гравця, для коректної роботи коду. Тепер, після впровадження цього скрипту, при старті сцени, гра пропонує нескінченний рівень. Для генерації об'єктів на тайлах ми розробили ще три додаткових скрипти, кожен з яких відповідає за генерацію своєї категорії об'єктів, з випадковим вибором та розміщенням їх у визначених можливих позиціях. Вони мають подібну між собою структуру, тому ми розберемо лише основні методи, на прикладі скрипту

Цитування: 0,01%

id: 30

<TileWithObstacles>

(Перешкоди на тайлі) (див. додаток В): Ініціалізація: У методі `Start()`, якщо `mainCamera` не було задано, вона шукається автоматично за назвою

Цитування: 0,01%

id: 31

<Camera.main>.

Також, ініціалізується масив об'єктів (перешкод) `obstacleSpawnedOnCollider`. Оновлення: У методі `Update()`, перевіряється відстань між камерою та кожним колайдером. Якщо відстань менша за `spawnDistanceThreshold` і перешкода на цьому колайдері ще не була створена, генерується нова перешкода. Генерація перешкоди: Метод `GenerateObstacle(int colliderIndex)` обирає випадкову перешкоду з `obstaclePrefabs` та створює її на місці колайдера. Позначення перешкоди: Метод `MarkObstacleSpawned(Collider spawnPoint, GameObject obstacle)` додає тег

Цитування: 0,01%

id: 32

"Obstacle"

для позначення перешкоди. Видалення перешкоди: Коли об'єкт стає невидимим для

камери (метод [OnBecameInvisible\(\)](#)), він видаляється з сцени. Далі цей, та подібні скрипти для генерації будівель та монет на рівні - прикріплюються на кожен з тайлів окремо, та налаштовується в інспекторі. До скриптів заносимо окремо кожен з підготовлених префабів моделей перешкод, визначаємо всі точки, де ці перешкоди можуть з'являтися, та встановлюємо відстань від гравця до найближчої точки спавну, при якій має з'явитися перешкода. Також встановлюємо час, протягом якого ця перешкода існуватиме на сцені. Оскільки цей тайл не є об'єктом на сцені, а з'являється під час гри, - камеру додати не можна. Тому, як вказано у попередньому скрипті, камера буде знаходитися за її назвою на сцені під час гри. Таким чином, із впровадженням цих скриптів у гру був доданий нескінченний рівень, який майже повністю генерується випадковим чином. Водночас, ми впровадили методи для оптимізації проекту, такі як: видалення старих тайлів, видалення об'єктів через певний час, та після певної пройденої дистанції. Цей підхід дозволив нам створити гнучку систему генерації рівня, яка забезпечує унікальність та різноманіття кожного ігрового моменту, забезпечуючи при цьому неперервну геймплейну динаміку та захоплення для гравців. Дизайн та ідея ігрового меню У нашому проекті ми пропонуємо гравцеві непересічний досвід, поєднуючи елементи ігрового меню з історією гри. Після входу у гру, гравець опиняється у віртуальному будинку головного героя, біля дивану. Цей будинок він може досліджувати у віртуальній реальності. На головній стіні будинку розташована статистика, що відображає кількість вбивств, рахунок монет та максимальну пройденої дистанцію за один раз у грі. За сюжетом гри, все, що відбувається на ігровій сцені, є сном головного героя, який він бачить засинаючи на цьому дивані. Таким чином, будинок у меню рахується за реальність, де гравець записує результати після кожного сну на стіні, намагаючись при цьому бити там власні рекорди. На сцені меню, після додавання будинку головного героя, ми реалізували ефект запікання світла, використовуючи вбудовані можливості [Unity](#). Запікання світла - це процес обчислення освітлення в ігровій сцені перед часом її запуску. Воно дозволяє зменшити обчислювальне навантаження під час гри, замість того, щоб обчислювати освітлення в реальному часі. Дізнатися більше про те як відбувається запікання світла в [Unity](#) можна у офіційній документації [13]. У нашій роботі, для запікання світла ми додали та налаштували об'єкт

Цитування: 0,01%

id: 33

«[Spot Light](#)»

(Точкове світло) для досягнення ефекту освітлення від люстри в будинку. Для загального освітлення сцени ми налаштували стандартне освітлення

Цитування: 0,01%

id: 34

«[Directional Light](#)»

(Спрямоване світло), яке рівномірно освітлює всю сцену загалом. Після того, як було проведено налаштування сцени - ми використали функціонал [Unity](#) для створення запеченого світла. [Unity](#) використовує запечені дані для швидкого та ефективного відображення освітлення в сцені. Це дозволяє досягти реалістичного ефекту освітлення без значного впливу на продуктивність гри. В результаті розробки ідеї та дизайну ігрового меню нам вдалося створити зручний інтерфейс для користувачів, та додати цікавий елемент, який поглиблює атмосферу гри. Замість звичного меню ми забезпечили гравця можливістю зануритися в атмосферу самого головного героя, де він може оглянути свої досягнення та спостерігати за їх зміною в реальному часі. Крім того, застосування світлових ефектів, зокрема запікання світла, додало реалістичності та зробило обстановку оптимізованою, та ще більш привабливою для сприйняття. Інтерфейс Ми приділили великий акцент зручності інтерфейсу, щоб забезпечити гравцям приємний та безперебійний досвід від геймплею. Замість стандартного прикріплення до персонажа, інтерфейс розташований попереду, вгорі, забезпечуючи гравцеві зручний доступ до інформації під час бігу по нескінченному рівню. Інтерфейс включає в себе прозорий [Canvas](#) із наступними лічильниками: Максимальний рекорд відстані Поточна пройдена відстань Кількість монет Кількість вбивств за раунд Крім того, ми реалізували

Цитування: 0,01%

id: 35

"[Lose canvas](#)"

(рис. 3.4.), це палень, яка з'являється при програві. Вона містить аналогічні лічильники (статистика), а також кнопки для рестарту чи виходу в меню. Рис.3.4. Меню програшу у грі Для натискання кнопки гравець може навести контролер на панель програшу, і тоді з контролера вийде луч, яким можна більш чітко відслідковувати наведення контролера (рис. 3.4.). Після того як луч контролера попаде на потрібну гравцю кнопку, - при натисканні тригера відбудеться активація необхідного методу. Цей підхід забезпечує точне управління та зручний доступ до функцій гри без зайвих ускладнень. Інтерфейс у меню має схожу структуру з

Цитування: 0,01%

id: 36

"[Lose canvas](#)"

у грі, але забезпечує інші функції, відповідно до контексту меню. Також він надає доступ до основної інформації. А саме: Максимальний рекорд пройденої відстані за весь час Кількість зібраних монет Кількість вбивств за весь час загалом Вороги Вороги виконують ключову роль у підтримці напруження та захоплення гравця. Розглянемо два основних типи ворогів у нашому проекті: Випадкові вороги: Перші вороги з'являються попереду

гравця у випадковий для нього момент та у випадковому порядку. Вони завжди направлені на головного героя з метою перешкоди у досягненні цілей. Гравець має можливість вбити або намагатися обійти їх. Цей елемент додає гри додаткової динаміки та вибору, створюючи непередбачуваність та викликаючи реакцію гравця. Постійний ворог: Головним ворогом у гри є ведмідь, який неперервно переслідує головного героя. Цей ворог не дає можливості зупинитись та розслабитись, стимулюючи гравця бігти все далі і далі. Зіткнення з будь-яким видом ворогів призводить до програшу та викликає

Цитування: 0,01%

id: 37

"Lose canvas",

спонукаючи гравця до стратегічних рішень та швидких дій. Розберемо більш детально, як був реалізований перший тип ворогів: Випадкові вороги: Першим кроком ми створили об'єкт

Цитування: 0,01%

id: 38

«Spawner animal»,

для встановлення відстані, на якій будуть з'являтися вороги, відносно головного героя. Для коректної роботи, він повинен завжди бути на відстані від персонажа. Для цього ми додали на сцену об'єкт, під назвою

Цитування: 0,01%

id: 39

«Stop Plane»,

що буде батьком для всіх інших, які повинні бути на відстані від гравця. Сюди входить багато об'єктів, в тому числі й інтерфейс, що ми розглянули раніше. Для цього об'єкта був розроблений скрипт

Цитування: 0,01%

id: 40

«StopPlane»

(повний код див. додаток В). Розглянемо його ключовий метод: Метод для переміщення об'єктів: `private void Update() { if (player != null && stopPlane != null) { // Отримуємо відстань по осі Z між гравцем і`

Цитування: 0,01%

id: 41

"StopPlane"

`float distanceZ = Mathf.Abs(player.position.z - stopPlane.position.z); // Якщо відстань більше за максимальну допустиму відстань if (distanceZ > maxDistance) { // Переміщуємо`

Цитування: 0,01%

id: 42

"StopPlane"

`ближче до гравця Vector3 newPosition = stopPlane.position; newPosition.z = player.position.z + maxDistance - 4; stopPlane.position = newPosition; } } } Цей метод перевіряє, чи потрібно перемістити площину`

Цитування: 0,01%

id: 43

"StopPlane".

Ми перевіряємо, чи існують посилання на гравця та площину

Цитування: 0,01%

id: 44

"StopPlane".

Розраховуємо відстань по осі Z між ними. Та перевіряємо, чи відстань більше за максимально допустиму. Якщо умова виконана - створюємо нову позицію для

Цитування: 0,01%

id: 45

"StopPlane"

на основі поточної, та переміщуємо до неї об'єкт. Таким чином тепер ми маємо можливість закріпити об'єкти на сцені на визначеній відстані. Окрім змоги переміщувати об'єкти вперед - позаду персонажа, та з кожного боку було встановлено невидимі стіни, що будуть слугувати границями, для унеможливлення виходу за межі ігрового рівня. Після розробки цих кроків - вже можна приступити до розробки скрипту для появи ворогів на сцені.

Розглянемо основний метод скрипту

Цитування: 0,01%

id: 46

«Animal Spawner»

під назвою

Цитування: 0,01%

id: 47

«SpawnAnimalsRandomly ()»

(див додаток В): Цей метод використовується для створення ворога у випадкові моменти часу. Цикл `while` безкінечно повторюється, щоб постійно створювати тварини. За допомогою `Random.Range` - генерується випадковий час затримки перед наступним спавном ворога, який знаходиться між мінімальним (`minSpawnDelay`) та максимальним (`maxSpawnDelay`) значеннями. Далі обирається випадковий префаб ворога з масиву `animalPrefabs`, та генерується випадкова позиція по осі X в межах від -1 до 1. В поточній позиції вже створюється ворог, і якщо за 10 секунд його не було вбито - він зникає зі

сцени, для оптимізації гри. Цей код кріпимо на об'єкт

Цитирования: 0,01% id: 48

«[Spawner animal](#)»,

та додаємо в інспекторі префаби ворогів, що будуть створюватись на сцені. Нижче вказуємо мінімальний час між появою ворогів ([Min Spawn Delay](#)), та максимальний час ([Max Spawn Delay](#)). Для того щоб вороги рухались в бік гравця, з метою перешкодити йому рух – був розроблений універсальний код

Цитирования: 0,01% id: 49

«[AnimalMovement](#)»

(див. додаток В), який кріпиться на кожного ворога окремо. Він включає в себе наступне: У методі

Цитирования: 0,01% id: 50

«[Start\(\)](#)»

відбувається пошук об'єкта гравця за тегом

Цитирования: 0,01% id: 51

"[MainCamera](#)".

Якщо посилання на об'єкт гравця не знайдено, виводиться відповідна помилка в лог. У методі

Цитирования: 0,01% id: 52

«[Update\(\)](#)»

здійснюється перевірка наявності посилання на об'єкт гравця. Якщо посилання є, ворог обертається у напрямку до гравця, а потім рухається вперед із заданою швидкістю. Метод

Цитирования: 0,01% id: 53

«[OnCollisionEnter](#)»

обробляє зіткнення тварини з іншими об'єктами. Якщо тварина стикається з об'єктом, що має тег

Цитирования: 0,01% id: 54

"[kulya](#)",

вона знищується, а на її місці спавниться об'єкт

Цитирования: 0,01% id: 55

«[FirePrefab](#)».

Таким чином, цей скрипт забезпечує базову поведінку руху тварини, орієнтацію на гравця та реакцію на зіткнення з іншими об'єктами у ігровому середовищі [Unity](#). Постійний ворог: Що стосується другого, постійного виду ворогів, а конкретно ведмедя позаду – тут все зроблено інакше. Цей ворог має іншу мету. Він розроблений для того, щоб гравцю ніколи було відпочивати. Через те що він постійно біжить позаду – гравцю треба бігти вперед, що додає як стимул, так і покращує елементи фітнесу у грі (про це трохи згодом). Спочатку ми завантажили цю тварину з офіційного магазину [Unity](#) [14]. Після чого розробили на сцені об'єкт

Цитирования: 0,01% id: 56

«[LosePlane](#)»,

що слугуватиме невидимою стіною, яка буде рухатись разом з ведмедем, та відповідатиме за меню програшу у разі зіткнення. Такий метод обраний як для оптимізації, так і для того, щоб неможливо було зламати логіку гри, та обійти ведмедя, при цьому не програвши. Для об'єкта

Цитирования: 0,01% id: 57

«[LosePlane](#)»

був розроблений окремий код, з однойменною назвою (див. додаток В). Розглянемо його основні моменти: У методі [OnTriggerEnter\(\)](#) обробляються зіткнення

Цитирования: 0,01% id: 58

"[LosePlane](#)"

з іншими об'єктами. Якщо зіткнення відбувається з гравцем, - зупиняється рух

Цитирования: 0,01% id: 59

"[LosePlane](#)",

активується темне світло, вимикається звичайне світло, вмикається канвас програшу та об'єкти взаємодії. Крім того, всі об'єкти з тегом

Цитирования: 0,01% id: 60

"[enemy](#)"

видаляються з сцени. Якщо зіткнення відбувається з перешкодою або монетою, - об'єкт цієї перешкоди або монети видаляється з сцени, а також з'являється ефект знищення на місці знищеного об'єкта. Для самого ведмедя також був розроблений код, для того щоб

керувати його анімаціями. Вони змінюються в залежності від сценарію гри. За стандартом ведмідь біжить вперед, але анімація змінюється на анімацію атаки, коли він наближується до гравця, та зараховує програш. Таким чином додаючи реалістичності та емоційності від ігрового процесу. Отже в цьому підрозділі ми оглянули всі можливі випадки для програшу у грі, розібравши як графічну, так і основну кодову складову цих надважливих елементів гри. Реалізація звукового контенту Оглянувши ключові моменти графічного контенту – слід розповісти і про звуковий. Наш арсенал звуків охоплює постріли, звуки руху та захоплюючу фонову музику, яка створює атмосферу імерсії та вносить різноманіття в геймплей. Для того, щоб забезпечити безперебійну роботу звукового контенту, ми завантажили необхідні аудіофайли з відкритих джерел [15,16] та імпортували їх у наш проект. Щоб уникнути будь-яких непорозумінь та забезпечити організованість, ми створили спеціальну папку

Цитирования: 0,01% id: 61

"Sounds",

в якій кожен звук розміщено у відповідній папці, згідно з його призначенням. Таким чином, ми додали три основні типи звуків: фонову музику, яка грає постійно протягом рівня, звуки руху, які активуються під час пересування гравця, з врахуванням інтенсивності та швидкості руху, а також звуки пострілів, які з'являються лише при використанні зброї. Фонова музика Для покращення атмосфери, та поглиблення у гру – вона повинна містити фонову музику. Музика на фоні здатна завдавати будь-якої атмосфери, в залежності від її типу. У нашому випадку була обрана динамічна, футуристична композиція, що надає атмосферу, з якою хочеться бігти вперед і встановлювати все більші і більші рекорди, минаючи перешкоди та вбиваючи ворогів перед собою. Щоб реалізувати фонову музику, на об'єкт камери ми додали компоненти

Цитирования: 0,01% id: 62

«Audio Listen»

та

Цитирования: 0,01% id: 63

«Audio Source».

Ці об'єкти використовуються для роботи зі звуком в [Unity](#).

Цитирования: 0,01% id: 64

«Audio Listen»

(Аудіо слухач) потрібен нам у грі, щоб ми могли слухати всі можливі звуки. На сцені він може бути тільки один, і оскільки об'єкт

Цитирования: 0,01% id: 65

«камера»

це наші очі – буде правильним зробити її і нашими вухами, додаючи на неї компонент

Цитирования: 0,01% id: 66

«Audio Listen».

Audio Source (Аудіо джерело) – це компонент, який дозволяє відтворювати звук. Цих об'єктів може бути декілька на сцені, оскільки кожен такий об'єкт може відтворювати окремий аудіо файл. Для фонові музики найкращим рішенням буде додати його також на камеру. Таким чином вона буде в прямому сенсі завжди грати у нас в голові. Цей метод використовують всі ігрові студії, для створення фонового супровіду. В компонент

Цитирования: 0,01% id: 67

«Audio Source»

у поле

Цитирования: 0,01% id: 68

«AudioClip»

ми перенесли наш музичний файл з папки у проєкті. В параметрах компонента включили параметри

Цитирования: 0,02% id: 69

«Play On Awake»

та

Цитирования: 0,01% id: 70

«Loop»,

що означає що музика буде грати з самого початку, при старті рівня, та після закінчення буде автоматично повторюватись. Такі параметри, як гучність, пріоритет, ефект стерео та інші – були налаштовані оптимально під сценарій гри. Звуки пострілів Оскільки в нашій грі задумана можливість стріляти по ворогам – ми повинні забезпечити кожен постріл характерним звуком, для більш реалістичного відчуття гри. Для початку розберемо роботу пістолета у грі. Рис. 3.5. Пістолет та точка появи куль на сцені Ми додали на сцену модель пістолета, та прикріпили її до правого контролера, щоб мати можливість керувати ним за допомогою рук, максимально схоже до реального (рис. 3.5.). Перед пістолетом був доданий

порожній об'єкт, яким ми назвали

Цитування: 0,01%

id: 71

«spawn_point».

Цей об'єкт буде слугувати точкою, на якій буде з'являтися куля на сцені. Наступним кроком ми приступили до програмування. Розберемо головні методи з коду

Цитування: 0,01%

id: 72

«GunController»

(див. додаток В), щоб дізнатись детальніше як працює керування пістолетом: Цей скрипт керує стрільниною зі зброї у VR-грі. Основні функції включають: Ініціалізація тригера: Зчитування натискання тригера на правому контролері. Рух кулі: Створення об'єкта

Цитування: 0,01%

id: 73

"kulya"

при утриманні тригера з інтервалом у 0.33 секунди. Визначення швидкості кулі: Куля рухається зі швидкістю 3 одиниці в напрямку зброї. Відтворення звуку: При пострілі відтворюється звук

Цитування: 0,01%

id: 74

"Shot".

Цей скрипт забезпечує функціональність стрільнини та анімацію кулі в грі. Після написання скрипту – його було додано до об'єкта пістолета на сцену, та налаштовано наступним чином: В інспекторі було додано об'єкт

Цитування: 0,01%

id: 75

«spawn_point»,

який слугує невидимою точкою на кінці пістолета, звідки вилітає куля. Далі ми додали префаб самої кулі, та вказали інтервал, що слугує затримкою між пострілами, для реалістичності, та швидкість кулі. У нижнє поле додали сам звук пострілу. Також, його було додано у компонент

Цитування: 0,01%

id: 76

«Audio Source»,

для того щоб цей звук лунав саме з пістолета, а не з камери, як фонові музика. Таким чином ми отримали працюючий пістолет, при пострілах з якого гравець буде чути характерні звуки. Особливість цього методу полягає в тому, що звук пострілу буде 100% лунати саме від зброї, а не на фоні, що допоможе підтримувати іммерсивну атмосферу та занурення у гру. Звуки руху Оскільки головна мета гри – пробігти якомога далі, встановлюючи рекорд подоланої відстані – ми не можемо залишити наш біг без характерних звуків. Для реалізації звуку переміщення ми записали звук одного шагу та додали його у гру. Вважається, що цього буде достатньо. Оскільки у грі ми маємо можливість плавно змінювати швидкість руху – логічним рішенням буде зробити і коректний звуковий супровід, для того щоб краще передати відчуття швидкості. Для досягнення цієї мети був модифікований код руху персонажа

Цитування: 0,01%

id: 77

«DynamicMoveProvider»

(див додаток В), який буде детально розглянуто пізніше. Розберемо ключові моменти в коді, що відповідають саме за звуки шагів: В скрипті ми отримуємо посилання на компонент

Цитування: 0,01%

id: 78

«AudioSource»,

зі звуком шагу, після чого робимо умови для його відтворення та припинення. У частині коду, де ми змінюємо швидкість руху – додаємо умови, при яких змінюється параметр

Цитування: 0,01%

id: 79

«pitch»

(швидкість) звуку шагів в залежності від значення moveSpeed. Після модифікації скрипту ми додали компонент

Цитування: 0,01%

id: 80

«AudioSource»

в інспекторі об'єкта

Цитування: 0,01%

id: 81

«move»,

як і у минулих випадках, вказуючи там звук шагу. Таким чином, ми отримали звуковий ефект, де шаги лунають лише під час руху гравця та змінюють свою інтенсивність залежно від швидкості руху. Отже тепер ми маємо уявлення як працюють три види звуків у нашому проєкті, а саме: Постійні звуки (фонові музика): Це звуки, які відтворюються безперервно протягом усієї гри, такі як фонові музика, яка створює атмосферу гри. Звуки, що викликаються натисканням (звуки пострілу): Ці звуки відтворюються у відповідь на

конкретні дії гравця, наприклад, постріли зі зброї. Звуки, що змінюються в залежності від дій гравця (звуки руху персонажа): Це звуки, які змінюються в залежності від активності гравця, наприклад, звуки кроків, які змінюють свою інтенсивність в залежності від швидкості руху персонажа. Взаємодія з користувачем та управління грою у віртуальному середовищі Взаємодія з користувачем та управління грою у віртуальному середовищі є одним із найважливіших аспектів розробки віртуальних ігор, який сильно відрізняється від управління на інших платформах. У цьому розділі ми розглянемо методи та інструменти, які були використані для створення ефективного та зручного управління грою у віртуальному середовищі. Ми дослідимо взаємодію з контролерами, реалізацію руху та навігації, обробку жестів та введення даних, а також інші аспекти, що сприяють зручності та іммерсивності ігрового досвіду віртуальної реальності. Інтеграція та використання [SDK](#)

Цитування: 0,02%

id: 82

«[Unity XR Interaction Toolkit](#)»

У нашій грі керування персонажем реалізовано за допомогою відштовхування стіку лівого контролера, що дозволяє переміщувати гравця у віртуальному просторі на всі 360 градусів. Окрім цього, для зміни швидкості руху гравця використовуються рухи контролерів у протилежних напрямках. Наприклад, якщо рухати контролери у протилежні сторони вгору та вниз, як у звичайному бігу, то персонаж рухатиметься швидше. Інтенсивність руху контролерів відображається на швидкості бігу гравця. Цей механізм додає у гру елементи фітнесу, оскільки гравець постійно активно рухається, що сприяє тренуванню м'язів рук і спалюванню зайвих калорій. Першим кроком, для того щоб керувати персонажем – треба створити самого персонажа. Оскільки віртуальна реальність дає нам змогу керувати головою, за допомогою шолому, та руками, за допомогою контролерів – треба додати ці елементи на нашу сцену. Для цього ми скористались завантаженням раніше, спеціальним [SDK](#) для розробки під шолом [Meta Quest 2](#), а саме -

Цитування: 0,02%

id: 83

«[Unity XR Interaction Toolkit](#)»

[XR Interaction Toolkit](#) має великий набір налаштованих, базових компонентів, які дозволяють об'єктам взаємодіяти з іншими об'єктами в середовищі [VR](#) або [AR](#), такими як захоплення, трансформація, переміщення тощо. Тому ми будемо використовувати деякі скрипти і напрацювання, які були розроблені в

Цитування: 0,02%

id: 84

«[XR Interaction Toolkit](#)»

для розробників ігор. Даний [SDK](#) має префаб (заготовку) базового компоненту з налаштуваннями для керування камерою та контролерами у [VR](#), під назвою

Цитування: 0,03%

id: 85

«[Complete XR Origin Set Up](#)».

Для початку було прийнято рішення імпортувати цей компонент на нашу сцену: Рис. 3.6. Компоненти керування персонажем на ігровій сцені Префаб має багато компонентів та програмного коду (рис. 3.6.). Розглянемо детально основні складові

Цитування: 0,03%

id: 86

«[Complete XR Origin Set Up](#)»:

[Input Action Manager](#): Це компонент, який управляє введенням у грі. Він дозволяє налаштовувати та обробляти дії гравця, такі як натискання кнопок, рухи контролерів та інші події. [XR Interaction Manager](#): Цей компонент відповідає за взаємодію з об'єктами-інтерактивами, їх захоплення та переміщення. [EventSystem](#): Це стандартний компонент [Unity](#), який відповідає за обробку подій вводу. В [VR](#) він використовується для обробки взаємодії з об'єктами, натискання кнопок та інших подій вводу. [XR Origin \(XR Rig\)](#): Це компонент, який представляє базовий об'єкт для всього віртуального персонажа гравця. Він визначає положення та орієнтацію всього ігрового простору у віртуальному середовищі. [Camera Offset](#): Це компонент, який використовується для зсуву камери у віртуальному середовищі. Він дозволяє налаштовувати положення камери відносно голови гравця для кращого комфорту та іммерсії. [Main Camera](#): Це стандартна камера [Unity](#), яка відповідає за відображення гри на екрані. У віртуальній реальності це відповідає за відображення гри у віртуальному просторі через окуляри. [Gaze Interactor](#): Цей компонент дозволяє взаємодіяти з об'єктами шляхом спрямування погляду на них. [Gaze Stabilized](#): Стабілізує взаємодію з об'єктами за допомогою погляду гравця. Він забезпечує стабільність позиції взаємодії навіть при рухах голови. [Left Controller](#) та [Right Controller](#): Це представлення лівого та правого контролера віртуальної реальності. Ці компоненти відповідають за взаємодію з об'єктами у грі за допомогою контролерів. [Left Controller Stabilized](#) та [Right Controller Stabilized](#): Забезпечують стабільність позиції контролерів навіть при рухах гравця. [Locomotion System](#): Це система переміщення віртуального персонажа в грі. Вона відповідає за рух гравця у віртуальному середовищі, використовуючи контролери чи інші методи переміщення. Керування персонажем Після детального вивчення компонентів, наданих нам виробником у

Цитування: 0,02%

id: 87

«[Unity XR Interaction Toolkit](#)»

– ми перейшли до вдосконалення скриптів, для роботи з контролерами, оскільки для нашої гри не підходять стандартні налаштування з даного [SDK](#). Для керування персонажем ми перейшли до змін в основному скрипті в об'єкті

Цитування: 0,01% id: 88

«[DynamicMoveProvider](#)»

(див додаток В) компонента

Цитування: 0,01% id: 89

«[Locomotion System](#)».

В даному коді була розроблена одна з найголовніших механік нашої фітнес гри, а саме – біг за допомогою рухів руками (контролерами в руках) у різних напрямках, для зміни швидкості руху. Розглянемо детальніше нововведення: У цьому скрипті обчислюється поточне положення контролерів, та порівнюються зміни в положенні контролерів з попередніми значеннями для визначення того, чи рухаються контролери в протилежних напрямках. В залежності від результату перевірки, швидкість руху ([moveSpeed](#)) лінійно змінюється за допомогою методу [Mathf.Lerp](#) між значеннями [originalMoveSpeed](#) (оригінальна швидкість) та [modifiedMoveSpeed](#) (модифікована швидкість). Таким чином нами був реалізований ключовий фітнес елемент гри. Однак є ще деякі моменти, які треба додати для керування персонажем. Оскільки в нашій грі передбачені перешкоди – логічним рішенням буде зробити можливість не просто оминати їх, а й перестрибувати. Тому ми вирішили реалізувати цю функцію окремим скриптом

Цитування: 0,01% id: 90

«[Jump](#)»

(повний код є в додатку В), задіявши при цьому стік правого контролера. Розглянемо

детально ключові частини коду: Метод стрибку: [private void Jumping\(InputAction.CallbackContext obj\)](#) { [if \(!_characterController.isGrounded\)](#) [return](#); [_playerVelocity.y += Mathf.Sqrt\(jumpHeight * -3.0f * gravityValue\);](#) } Метод

Цитування: 0,01% id: 91

«[Jumping](#)»

викликається при відхиленні вгору кнопки стрибка. Він перевіряє, чи персонаж на землі, і якщо так, то додає швидкість у вигляді вектора вгору для стрибка. Метод оновлення:

[private void Update\(\)](#) { [if \(_characterController.isGrounded && _playerVelocity.y > 0\)](#) { [_playerVelocity.y = 0f;](#) } [_playerVelocity.y += gravityValue * Time.deltaTime;](#) [_characterController.Move\(_playerVelocity * Time.deltaTime\);](#) } У цьому методі кожного кадру перевіряється, чи персонаж на землі і якщо ні, то перевіряється його швидкість у вертикальному напрямку. Якщо швидкість йде вниз, то вона анулюється, а якщо ні, то до швидкості додається сила гравітації. Потім ця швидкість застосовується до персонажа за допомогою методу [Move\(\)](#) компонента [CharacterController](#). Таким чином тепер в нас є можливість перестрибнути через перешкоду, що додає динамічності та альтернативності у проходженні гри. Взаємодія з інтерфейсом У нашій грі як в меню, так і на ігровій сцені – часто доводиться взаємодіяти з інтерфейсом, для натискання кнопок. Стандартні компоненти з [SDK](#)

Цитування: 0,02% id: 92

«[XB Interaction Toolkit](#)»

не зовсім підходять під наш сценарій гри. Тому нами було прийняте рішення розробити допоміжний скрипт, для керування деякими функціями взаємодії з інтерфейсом. Розберемо як працює взаємодія з інтерфейсом на ігровій сцені: Під час самої гри у нас немає можливості натискати на елементи інтерфейсу, і навіть поставити гру на паузу. Ми розробили такий задум, спираючись на те, що це повинна бути не просто розважальна гра, а повноцінний фітнес-інструмент. Завдяки тому що під час гри немає можливості увімкнути паузу – гравець не може зупинити тренування для передиху, та продовжити далі, не онуляючи при цьому результат. Таким чином ми забезпечили чесність між гравцями, адже коли людина ставить новий рекорд пройденої дистанції – ми можемо гарантувати, що він був поставлений без махінацій, за один безперервний забіг. Тому наше спливаюче меню інтерфейсу повинно з'являтися вже після програшу у грі. Для реалізації наших планів, спочатку ми вимкнули на сцені такі компоненти як

Цитування: 0,01% id: 93

«[Ray Interactor](#)»

лівого та правого контролера, та

Цитування: 0,01% id: 94

«[Lose Canvas](#)».

Наступним кроком додали увімкнення цих об'єктів у допоміжному скрипті

Цитування: 0,01% id: 95

«[Lose Plane](#)»,

що розглядали раніше, в умову методу [OnTriggerEnter](#), коли

Цитування: 0,01% id: 96

«Lose Plane»

стикається з гравцем. Таким чином, при спрацюванні методу для програшу у нас вмикається інтерфейс програшу, та активуються об'єкти інтерактивності лівого та правого контролерів, за допомогою яких ми можемо навести промені з рук на потрібні нам кнопки, та натиснути їх клавішею тригером контролера. Аналогічна умова програшу була додана до скрипту інших ворогів. Реалізація методів натискання на кнопки у інтерфейсі особливо не відрізняється від інших платформ. У кожній кнопці в [Unity](#) за стандартом є подія натискання, яку можна налаштувати за допомогою скриптів. Ми використали універсальний скрипт

Цитування: 0,01%

id: 97

«Scenes»

для кнопок

Цитування: 0,01%

id: 98

«Restart»

та

Цитування: 0,01%

id: 99

«Menu»,

в якому додали метод обирання сцени в інспекторі кнопки: Метод зміни сцени: [public void NextLevel\(int _sceneNumber\)](#) { [SceneManager.LoadScene\(_sceneNumber\);](#) } Цей скрипт додається на пустий об'єкт

Цитування: 0,01%

id: 100

«SceneManager»

на сцені, для зручності його використання. Наступним кроком ми додали цей об'єкт в обробник подій при натисканні кнопки ([On Click](#)), та вказали номер сцени, на яку треба перейти, після натискання кнопки. В цьому розділі ми розглянули при яких обставинах викликається інтерфейс на ігровій сцені, та яким чином відбувається взаємодія між інтерфейсом та контролерами гравця. Висновки до розділу 3 У третьому розділі було детально розглянуто процес розробки ігрового додатку для [VR](#)-шолому [Meta Quest](#). Основна увага приділялася реалізації графічного та звукового контенту, а також забезпеченню взаємодії користувача з грою у віртуальному середовищі. Реалізація графічного контенту включала визначення ключових цілей і завдань, моделювання та текстурування об'єктів, створення частин рівня, програмування випадкової генерації рівня, розробку дизайну та ідеї ігрового меню, створення інтерфейсу та ворогів. Звуковий контент, включаючи фонову музику, звуки пострілів та звуки руху, був інтегрований для покращення імерсії гравців у віртуальну реальність. Взаємодія з користувачем та управління грою у віртуальному середовищі були реалізовані за допомогою інтеграції та використання [SDK](#)

Цитування: 0,02%

id: 101

«Unity XR Interaction Toolkit».

Особлива увага приділялася розробці механік керування персонажем та взаємодії з інтерфейсом, що забезпечило зручність та інтуїтивність гри. Загалом, у розділі 3 було успішно реалізовано всі необхідні етапи розробки ігрового додатку для [VR](#)-шолому [Meta Quest](#), що дозволило створити повноцінний та функціональний ігровий продукт. ВИСНОВКИ На основі проведених досліджень було детально проаналізовано історію розвитку та технічні складові [VR](#)-технологій, а також принципи роботи [VR](#)-систем. Особливу увагу було приділено технічним характеристикам та можливостям [VR](#)-шолому [Meta Quest](#), що визначали специфіку розробки ігрового додатку. Результати дослідження і розробки ігрового додатку для [VR](#)-шолому [Meta Quest](#) підтвердили ефективність обраних технологій та підходів. У процесі роботи було створено концепцію інноваційного [VR](#)-досвіду, що забезпечує високий рівень занурення користувача у віртуальне середовище. Особлива увага приділялася оптимізації продуктивності та реалістичності віртуального світу, що було досягнуто завдяки використанню [Unity](#) та [Unity XR Interaction Toolkit](#). Тести підтвердили стабільну роботу додатку та відповідність його функціональним вимогам. Під час розробки ігрового додатку було успішно реалізовано графічний та звуковий контент, забезпечено інтерактивність через інтеграцію відповідних [SDK](#), а також розроблено механіки керування персонажем та інтерфейсом. Завдяки цьому проект став значним кроком у розумінні та застосуванні сучасних [VR](#)-технологій, надавши змогу створити унікальний ігровий досвід та продемонструвавши потенціал для подальших досліджень і вдосконалення у цій сфері. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ [Imena.ua AR, VR, MR, XR](#): що це за реальності та чим вони відрізняються? [URL](#):

<https://www.imena.ua/blog/what-is-ar-vr-mr-xr/> (дата звернення: 10.03.2024). Г'вара Медіа.

Історія віртуальної реальності. [URL](#): <https://gwaramedia.com/istoriia-virtualnoi-realnosti/> (дата звернення: 13.03.2024). [Virtual Speech](#). Історія [VR](#) – Хронологія подій і розвитку технологій.

[URL](#): <https://virtualspeech.com/blog/history-of-vr> (дата звернення: 13.03.2024). [Educative](#). [What are the basic components of virtual reality?](#) [URL](#):

<https://www.educative.io/answers/what-are-the-basic-components-of-virtual-reality> (дата звернення: 15.03.2024). [Tobii](#). [Eye tracking in VR – A vital component](#). [URL](#):

<https://unboundxr.eu/hoer-vr-tracking-werkt> (дата звернення: 18.03.2024). [Unboundxr](#). [Virtual](#)

Reatily (VR) [tracking explained](#). URL: <https://www.tobii.com/blog/eye-tracking-in-vr-a-vital-component> (дата звернення: 26.03.2024).
[Interaction Design](#). [Presence in Virtual Reality \(VR\)](#). URL: <https://www.interaction-design.org/literature/topics/presence> (дата звернення: 28.03.2024).
[Virtual Speech](#). [VR Applications: Key Industries already using Virtual Reality](#). URL: <https://virtuallspeech.com/blog/vr-applications> (дата звернення: 02.04.2024). [Simbott](#). 11 [Virtual Reality Advantages And Disadvantages](#) (2024). URL: <https://simbott.com/virtual-reality-advantages-and-disadvantages/> (дата звернення: 03.04.2024). [Digital Reality](#). [Ethics in Virtual Reality](#). URL: <https://digitalreality.ieee.org/publications/ethics-in-vr> (дата звернення: 05.04.2024). [Maloney, E.](#) (2021). [Social VIRTUAL REALITY](#). [PDF]. URL: <https://www.andrewrobb.io/publication/maloney-2021-social/maloney-2021-social.pdf> (дата звернення: 05.04.2024). [Meta](#). [Get started with Meta Quest 2](#). URL: <https://www.meta.com/quest/products/quest-2/tech-specs/> (дата звернення: 07.04.2024). [Fxmedia](#). [VR Game Design Principles for Immersive and Enjoyable Experiences](#). URL: <https://www.fxmweb.com/insights/vr-game-design-principles-for-immersive-and-enjoyable-experiences.html> (дата звернення: 10.04.2024). [Techtarget](#). [Virtual reality sickness \(VR motion sickness\)](#). URL: <https://www.techtartget.com/iotagenda/definition/virtual-reality-sickness-VR-motion-sickness> (дата звернення: 10.04.2024). [Fastercapital](#). [Creating Interactive VR Content](#). URL: <https://fastercapital.com/topics/creating-interactive-vr-content.html> (дата звернення: 11.04.2024). [Futurelearn](#). [Interaction Techniques in VR](#). URL: <https://www.futurelearn.com/info/courses/construct-a-virtual-reality-experience/0/steps/96390> (дата звернення: 12.04.2024). [Lemon School](#). [Що таке Unity?](#) URL: <https://lemon.school/blog/shho-take-unity> (дата звернення: 13.04.2024). [SKVOT Mag](#). Не соромно запитати: як працює [unreal engine](#). URL: <https://skvot.io/uk/blog/ne-soromno-zapitati-yak-pracyuye-unreal-engine> (дата звернення: 13.04.2024). [Crytek](#). [The power to achieve your vision](#). URL: <https://www.crytek.com/cryengine> (дата звернення: 13.04.2024). [Unity](#). [UNITY HUB](#). URL: <https://unity.com/unity-hub> (дата звернення: 13.04.2024). [Unity](#). [XR Interaction Toolkit](#). URL: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html> (дата звернення: 13.04.2024). [Hillel blog](#). Побудова динамічних вебзастосунків за допомогою [MVC](#). URL: <https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc> (дата звернення: 13.04.2024). [Genesis](#). Як декомпонувати та розробити гру помодульно з допомогою [FCS](#). URL: <https://www.gen.tech/post/yak-rozrobiti-gru-pomodulno-z-ecs> (дата звернення: 14.04.2024). [Unrealengine](#). [Behavior Tree Overview](#). URL: <https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/BehaviorTrees/BehaviorTree> (дата звернення: 14.04.2024). [Medium](#). [Intro to Data Oriented Design \(DOD\) with Unity](#). URL: <https://medium.com/@nitzanwilnai/intro-to-data-oriented-design-dod-with-unity-991b0239402> (дата звернення: 15.04.2024). [Medium](#). [Component Based Architecture](#). URL: <https://medium.com/clean-code-channel/component-based-architecture-2cb38ea1b247> (дата звернення: 15.04.2024). [Reddit](#). [Stereoscopic Rendering for VR from Scratch?](#) URL: https://www.reddit.com/r/GraphicsProgramming/comments/rur91o/stereoscopic_rendering_for_vr_from_scratch/ (дата звернення: 15.04.2024). [Blender](#). [Вступ-Introduction](#). URL: https://docs.blender.org/manual/uk/2.82/getting_started/about/introduction.html (дата звернення: 16.04.2024). [Unity3d](#). [Baked lighting](#). URL: <https://docs.unity3d.com/2019.1/Documentation/Manual/LightMode-Baked.html> (дата звернення: 16.04.2024). [Unity Assetstore](#). [FREE Stylized Bear - RPG Forest Animal](#). URL: <https://assetstore.unity.com/packages/3d/characters/animals/free-stylized-bear-rpg-forest-animal-228910> (дата звернення: 16.04.2024). [Pixabay](#). [No Copyright Music no copyright music](#). URL: <https://pixabay.com/music/search/no%20copyright%20music/> (дата звернення: 17.04.2024). [Fesliyanstudios](#). [Download Free Gun Shooting Sound Effect](#). URL: <https://www.fesliyanstudios.com/royalty-free-sound-effects-download/gun-shooting-300> (дата звернення: 17.04.2024). Додаток А Міністерство освіти і науки України Державний заклад

Цитування: 0,04%

id: 102

«Луганський національний університет імені Тараса Шевченка»

Факультет (інститут) Кафедра Навчально-науковий інститут математики та інформаційних технологій (повна назва) Інформаційних технологій та систем (повна назва) Програма та методика тестування на виконання програмної розробки (ПР): Розробка ігрового додатку віртуальної реальності для [VR](#)-шолому [Meta Quest](#) Полтава – 2024 ЗМІСТ ТЕСТ ПЛАН65 Вступ66 Мета66 Цілі тестування66 Об'єкти тестування66 Тестові сценарії66 Методи тестування66 План тестування та час тестування66 Тестувальне середовище66 Тест-кейси67 Запуск гри на шоломі [Meta Quest](#) 267 Перевірка управління гравцем67 Тестування функціоналу гри67 Тестування аудіо та візуальних ефектів67 Перевірка сумісності та продуктивності67 Тестування відновлення гри після втрати зв'язку67 Очікувані результати67 Виконання тестування68 Підготовка тестового середовища68 Запуск тестових сценаріїв68 Реєстрація результатів68 Виявлення помилок68 Виправлення помилок71 Повторне тестування71 Висновки тестування72 ТЕСТ ПЛАН Вступ Мета Тест-план для проекту визначає стратегії та методи тестування, необхідні для забезпечення якості та стабільності гри. Основна мета цього тест-плану - деталізувати процеси

Обнаружен Плагіат: 0,19% <https://itedu.center/ua/blog/career/10...>

id: 103

виявлення та усунення помилок, а також оптимізації продуктивності. Документ дозволяє отримати інформацію про план тестових робіт. Цілі тестування Перевірка функціональності гри на відповідність вимогам специфікації. Виявлення та усунення помилок та дефектів

в роботі гри. Оцінка ефективності та продуктивності гри на пристроях. Перевірка коректності та стабільності гри під час різних сценаріїв гри. Об'єкти тестування Головне меню та ігровий інтерфейс. Керування персонажем та взаємодія з оточенням. Геймплейні механіки (стріляння, рух, взаємодія з об'єктами). Графіка та анімація. Звуковий ефект та музикальний супровід. Тестові сценарії Запуск гри та перевірка роботи головного меню. Тестування головного режиму гри. Перевірка роботи гри в різних умовах. Взаємодія з різними об'єктами у грі (зброя, предмети, перешкоди). Перевірка реакції гри на дії гравця у різних умовах (рух, стріляння, обминання ворога). Методи тестування Ручне тестування: вручну перевіряти різні аспекти гри та фіксувати помилки. Автоматизоване тестування: використовувати автоматизовані скрипти для тестування функціональності гри. Тестування на реальних пристроях: перевірка гри на різних пристроях та платформах для забезпечення сумісності та оптимізації. План тестування на час тестування Визначення термінів та графіку проведення тестування. Фіксація та аналіз результатів тестування. Внесення коректив у розробку гри на основі виявлених дефектів та пропозицій. Тестувальне середовище Для проведення тестування нашої гри було використано шолом віртуальної реальності [Mets Quest 2](#), оскільки наша гра розроблена виключно під цю модель та версію пристрою. Цей шолом був обраним як основне тестувальне середовище через його популярність серед користувачів та високу сумісність з різними ігровими додатками, що розроблені під цю платформу. Крім шолому [Mets Quest 2](#), для тестування також використовувалися наступне обладнання та програмне забезпечення: ПК для розробки та тестування: Комп'ютер з високопродуктивними характеристиками, який використовувався для розробки та запуску не зібраного проекту на шоломі [Mets Quest 2](#). Ігрові контролери [Mets Quest 2](#): Для керування гравцем у віртуальному середовищі та взаємодії з ігровими елементами. Програмне забезпечення для тестування: Додаткові програмні засоби, такі як системи логування помилок, а також програми для аналізу продуктивності гри на різних конфігураціях комп'ютера. Це тестувальне середовище забезпечило нас можливістю відтворювати реальні умови гри та переконатися в якості та стабільності гри на конкретній платформі. Тест-кейси Запуск гри на шоломі [Meta Quest 2](#): Переконатися, що гра запускається на шоломі [Meta Quest 2](#) без будь-яких помилок або збоїв. Перевірити, що гра запускається зі сцени меню. Перевірка управління гравцем: Переконатися, що керування персонажем відбувається відповідно до відштовхування стіку лівого контролера. Перевірити, чи змінюється швидкість руху гравця відповідно до руху контролерів у протилежних напрямках. Тестування функціоналу гри: Перевірити роботу основних ігрових механік, таких як стрибки, збір монет та взаємодія з ворогами та оточуючим середовищем. Переконатися, що гра працює стабільно та безперервно протягом тривалості геймплею. Тестування аудіо та візуальних ефектів: Перевірити відтворення звуків шагів та інших аудіо ефектів. Переконатися, що візуальні ефекти, такі як анімації вбивства або руйнування перешкод, відтворюються коректно. Перевірка сумісності та продуктивності: Перевірити сумісність гри з шоломом [Mets Quest 2](#) та його основними характеристиками. Переконатися в стабільній роботі гри. Тестування відновлення гри після втрати зв'язку: Симулювати втрату зв'язку з контролерами або інші непередбачені обставини. Переконатися, що гра коректно відновлюється після відновлення зв'язку з додатковими пристроями, та не втрачає поточний прогрес. Очікувані результати: Усі тест-кейси пройдені успішно без помилок або неполадок. Гра працює стабільно та без перебоїв на шоломі [Meta Quest 2](#). Керування персонажем реагує на всі вхідні дані коректно і без затримок. Аудіо та візуальні ефекти відтворюються без перерв та відповідають очікуванням. Гра зберігає прогрес користувача та відновлює його після втрати зв'язку або інших подібних ситуацій. Загальне враження від геймплею є позитивним і задовільним для користувача. Виконання тестування. Процес тестування включає в себе кілька кроків, спрямованих на забезпечення якості та надійності гри на шоломі [Mets Quest 2](#). Ось докладний опис виконання кожного етапу: Підготовка тестового середовища: Першим кроком у процесі тестування є підготовка тестового середовища. Для цього ми маємо фізичний доступ до необхідного обладнання, включаючи шолом віртуальної реальності [Meta Quest 2](#) та контролери, які використовуються для розробки та тестування гри. Перевірка обладнання: Переконуємося, що шолом має налаштований звук, відео та контролери. Перевіряємо стан батареї та наявність достатнього простору та освітлення для роботи гарнітури та компонентів віртуальної реальності. Запуск програмного забезпечення: Запускаємо програмне забезпечення автономно на шоломі [Meta Quest 2](#) та переконуємося, що воно працює належним чином, без помилок. Перевірка готовності гри: Перевіряємо, що усі необхідні ресурси та компоненти ігрового середовища завантажились належним чином, та налаштовані і готові до використання. Перевірка інтеграції: Переконуємося, що гра належним чином інтегрована в шолом [Meta Quest 2](#) та може взаємодіяти з усіма його функціями та можливостями. Цей крок допомагає забезпечити, що тестування гри буде проводитися належним чином та усі аспекти гри будуть перевірені відповідно до вимог та очікувань. Запуск тестових сценаріїв: Після підготовки тестового середовища розпочинається запуск тестових сценаріїв, які покликані перевірити різні аспекти гри та функціональність ігрового додатку: Збір тестових сценаріїв: Спочатку перевіряємо наявність усіх запланованих тестових сценаріїв і переконуємося, що вони

відображають різноманітні аспекти гри, такі як керування, геймплей, взаємодія з об'єктами тощо. Виконання тестових сценаріїв: Запускаємо гру на шоломі [Mets Quest 2](#) і послідовно виконуємо кожен тестовий сценарій зі списку. Під час тестування активно взаємодіємо з грою, спробуючи відтворити різні ситуації, які можуть виникнути під час гри. Фіксація результатів: Під час виконання кожного тестового сценарію записуємо всі виявлені проблеми, помилки та недоліки у спеціальній таблиці. Описуємо кожну проблему докладно, вказуючи на її характер, відтворення та вплив на геймплей. Документування помилок: Після завершення тестового сеансу створюємо детальні звіти про кожен тестовий сценарій, де документуємо всі виявлені помилки та недоліки. Кожен звіт включає опис проблеми, кроки для відтворення, пріоритетність та рекомендовані кроки для виправлення. Реєстрація результатів: Після завершення кожного тестового сценарію необхідно зареєструвати отримані результати для подальшого аналізу та виправлення виявлених проблем. Фіксація помилок: Записуємо будь-які виявлені під час тестування помилки або недоліки у спеціальний журнал помилок. Кожній помилці надаємо опис, дату виявлення, час та шлях до відповідного фрагмента коду або сцени. Документування виявлених проблем: Крім самої помилки, фіксуємо опис можливих причин виникнення проблеми та можливі способи її виправлення. Призначення пріоритету: Кожній зареєстрованій помилці або проблемі призначаємо пріоритет відповідно до її важливості та впливу на користувачів. Це допомагає визначити порядок виправлення проблем та розподілити ресурси ефективно. Підготовка звіту: На основі зібраних даних складаємо звіт про результати тестування, який включає опис виявлених проблем, їхній пріоритет, можливі причини та рекомендації щодо виправлення. Звіт допомагає краще зрозуміти стан проекту та прийняти необхідні рішення. Реєстрація результатів тестування є важливим етапом у процесі розробки гри, що дозволяє виявити та виправити всі можливі проблеми та недоліки перед випуском готового продукту. Виявлення помилок: Під час тестування було виявлено, що при старті ігрової сцени іноді в області, де з'являється гравець, - з'являється одна з перешкод, що унеможливорює рух, через конфлікт об'єктів. Ця проблема може призвести до незручностей для гравців та порушити геймплей. Необхідно дослідити цю ситуацію та виправити проблему, щоб забезпечити плавний та комфортний геймплей для користувачів. Виправлення помилок Після виявлення та аналізу помилок ми вживаємо наступні кроки для їх виправлення: Визначення причини помилки: Першим кроком є визначення кореневої причини помилки. Ми аналізуємо код, конфігурації та взаємодію різних компонентів гри, щоб з'ясувати, що саме призвело до виникнення проблеми з появою в перешкоді та можливих причин інших помилок. Виправлення спавну в перешкоді: Ми внесли зміни в локації спавну перешкоди на початку гри, перемістивши її на безпечніше місце, де вона не заважатиме появі нашого персонажа. Це виправило проблему зі спавном в перешкоді та гарантує плавний початок гри для користувачів. Оптимізація гри: Для покращення продуктивності та оптимізації загального досвіду гри ми скоротили можливу максимальну кількість перешкод на тайлі. Це допомагає уникнути перевантаження гри та забезпечити плавну геймплейну динаміку. Повторне тестування Після внесення виправлень ми провели повторне тестування сценаріїв гри, для перевірки ефективності змін та їх впливу на загальний досвід від гри. Під час цього тестування не було виявлено жодних нових помилок або неполадок. Крім того, відзначили помітне поліпшення продуктивності гри: вона стала працювати плавніше, з більшою кількістю кадрів на секунду. Це досягнуто завдяки додатковій оптимізації, яка сприяла покращенню швидкості та ефективності роботи гри на шоломі [Meta Quest 2](#). Висновки тестування Після проведення всебічного тестування ігрового додатку на шоломі [Meta Quest 2](#) ми отримали важливі відомості про його функціональність, продуктивність та коректну роботу. Виявлені недоліки були успішно виправлені, а гра піддана ряду оптимізаційних заходів, що позитивно позначилося на її швидкості та якості відтворення. Загалом, результати тестування свідчать про високу якість та готовність ігрового додатку до подальшого розвитку та випуску. Важливо враховувати отримані відомості під час подальшої розробки, з метою забезпечення максимальної задоволеності користувачів та успішного впровадження гри на ринок. ДОДАТОК В Вихідний код скриптів Вихідний код скриптів, які організовують роботу програми - [TileGenerator](#), [TileWithObstacles](#), [StopPlane](#), [AnimalSpawner](#), [AnimalMovement](#), [LosePlane](#), [GunController](#), [DynamicMoveProvider](#). `using System.Collections; using System.Collections.Generic; using UnityEngine; public class TileGenerator : MonoBehaviour { public GameObject[] tilePrefabs; private List<GameObject> activeTiles = new List<GameObject>(); private float spawnPos = 0; private float tileLength = 80f; [SerializeField] private Transform player; private int startTiles = 2; void Start() { for (int i = 0; i < startTiles; i++) { if (i == 0) SpawnTile(3); SpawnTile(Random.Range(0, tilePrefabs.Length)); } } void Update() { if (player.position.z - 60 > spawnPos - (startTiles * tileLength)) { SpawnTile(Random.Range(0, tilePrefabs.Length)); DeleteTile(); } } private void SpawnTile(int tileIndex) { GameObject nextTile = Instantiate(tilePrefabs[tileIndex], transform.forward * spawnPos, transform.rotation); activeTiles.Add(nextTile); spawnPos += tileLength; } private void DeleteTile() { Destroy(activeTiles[0]); activeTiles.RemoveAt(0); } } using UnityEngine; public class TileWithObstacles : MonoBehaviour { public GameObject[] obstaclePrefabs; public Collider[] obstacleSpawnPoints; public float spawnDistanceThreshold = 40f; public Camera mainCamera; public float obstacleLifetime = 5f; private bool obstaclesSpawned = false; private bool[] obstacleSpawnedOnCollider; void Start() { if (mainCamera == null) { mainCamera = Camera.main; } obstacleSpawnedOnCollider = new bool[obstacleSpawnPoints.Length]; } void Update() { float cameraZ = mainCamera.transform.position.z; for (int i = 0; i < obstacleSpawnPoints.Length; i++) { float spawnPointZ =`


```

obstacleSpawnPoints[i].transform.position.z; float distance = Mathf.Abs(cameraZ - spawnPointZ);
if (!obstaclesSpawned && distance < spawnDistanceThreshold && !obstacleSpawnedOnCollider[i]) {
GenerateObstacle(i); } } void GenerateObstacle(int colliderIndex) { GameObject obstaclePrefab =
obstaclePrefabs[Random.Range(0, obstaclePrefabs.Length)]; Quaternion spawnRotation =
obstacleSpawnPoints[colliderIndex].transform.rotation; GameObject obstacle =
Instantiate(obstaclePrefab, obstacleSpawnPoints[colliderIndex].transform.position,
spawnRotation); obstacleSpawnedOnCollider[colliderIndex] = true;
MarkObstacleSpawned(obstacleSpawnPoints[colliderIndex], obstacle); Destroy(obstacle,
obstacleLifetime); } void MarkObstacleSpawned(Collider spawnPoint, GameObject obstacle) {
obstacle.tag =

```

Цитирования: 0,01%

id: 104

"Obstacle"

```

; } void OnBecameInvisible() { Destroy(gameObject); } } using UnityEngine; public class
StopPlane : MonoBehaviour { public Transform player; public Transform stopPlane; public float
maxDistance = 2.0f; private void Update() { if (player != null && stopPlane != null) { float
distanceZ = Mathf.Abs(player.position.z - stopPlane.position.z); if (distanceZ < maxDistance) {
Vector3 newPosition = stopPlane.position; newPosition.z = player.position.z + maxDistance - 4;
stopPlane.position = newPosition; } } } using UnityEngine; using System.Collections; public
class AnimalSpawner : MonoBehaviour { public GameObject[] animalPrefabs; public float
minSpawnDelay = 2f; public float maxSpawnDelay = 10f; private void Start() {
StartCoroutine(SpawnAnimalsRandomly()); } private IEnumerator SpawnAnimalsRandomly() {
while (true) { float delay = Random.Range(minSpawnDelay, maxSpawnDelay); yield return new
WaitForSeconds(delay); GameObject animalPrefab = animalPrefabs[Random.Range(0,
animalPrefabs.Length)]; float randomX = Random.Range(-1f, 1f); GameObject spawnedAnimal =
Instantiate(animalPrefab, new Vector3(randomX, transform.position.y, transform.position.z),
Quaternion.identity); yield return new WaitForSeconds(10f); Destroy(spawnedAnimal); } } using
UnityEngine; using System.Collections; public class AnimalMovement : MonoBehaviour { public
float moveSpeed = 5f; public GameObject player; public GameObject Animals; public
GameObject FirePrefab; private BoxCollider boxCollider; private void Start() { if (player == null) {
player = GameObject.FindWithTag(

```

Цитирования: 0,01%

id: 105

"MainCamera"

```

); } if (player == null) { Debug.LogError(

```

Цитирования: 0,02%

id: 106

"Player object not found."

```

); } } private void Update() { if (player == null) { return; } Vector3 targetDirection =
player.transform.position - transform.position; targetDirection.y = 0f; Quaternion newRotation =
Quaternion.LookRotation(targetDirection); transform.rotation = Quaternion.Euler(0f,
newRotation.eulerAngles.y, 0f); transform.Translate(Vector3.forward * moveSpeed *
Time.deltaTime); } private void OnCollisionEnter(Collision collision) { if
(collision.gameObject.CompareTag(

```

Цитирования: 0,01%

id: 107

"kulya"

```

)) { Destroy(gameObject); Instantiate(FirePrefab, transform.position, Quaternion.identity); } }
using System.Collections; using System.Collections.Generic; using UnityEngine; using
UnityEngine.SceneManagement; public class LosePlane : MonoBehaviour { public float
moveSpeed = 5f; public GameObject darkLight; public GameObject normalLight; public
GameObject CanvasLose; public GameObject RayInteractorL; public GameObject RayInteractorR;
public GameObject spawnerAnimal; public GameObject destructionEffectPrefab; private void
Update() { transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime); } private void
OnTriggerEnter(Collider other) { if (other.CompareTag(

```

Цитирования: 0,01%

id: 108

"Player"

```

)) { moveSpeed = 0f; darkLight.SetActive(true); normalLight.SetActive(false);
CanvasLose.SetActive(true); RayInteractorL.SetActive(true); RayInteractorR.SetActive(true);
spawnerAnimal.SetActive(false); GameObject[] enemies =
GameObject.FindGameObjectsWithTag(

```

Цитирования: 0,01%

id: 109

"enemy"

```

); foreach (GameObject enemy in enemies) { Destroy(enemy); } } if (other.CompareTag(

```

Цитирования: 0,01%

id: 110

"Obstacle"

```

) || other.CompareTag(

```

Цитирования: 0,01%

id: 111

"CoinsUI"

```

)) { Destroy(other.gameObject); if (destructionEffectPrefab != null) {

```

```
Instantiate(destructionEffectPrefab, other.transform.position, Quaternion.identity); } } } using
UnityEngine; using UnityEngine.InputSystem; public class GunController : MonoBehaviour { public
GameObject gun; public GameObject kulyaPrefab; public float spawnInterval = 0.33f; public float
kulyaSpeed = 3f; public AudioClip shotSound; private InputAction triggerAction; private bool
triggerPressed = false; private float lastSpawnTime; void OnEnable() { triggerAction.Enable(); }
void OnDisable() { triggerAction.Disable(); } void Start() { triggerAction = new
InputAction(binding:
```

Цитирования: **0,01%** id: **112**

"XRController {RightHand}/trigger"

```
); triggerAction.started += OnTriggerPressed; triggerAction.canceled += OnTriggerReleased;
triggerAction.Enable(); } void Update() { if (triggerPressed && Time.time - lastSpawnTime
spawnInterval) { SpawnKulya(); lastSpawnTime = Time.time; } } void
OnTriggerPressed(InputAction.CallbackContext context) { triggerPressed = true; } void
OnTriggerReleased(InputAction.CallbackContext context) { triggerPressed = false; } void
SpawnKulya() { if (gun != null) { GameObject kulya = Instantiate(kulyaPrefab,
gun.transform.position, gun.transform.rotation); Rigidbody kulyaRigidbody =
kulya.GetComponent<Rigidbody>(); if (kulyaRigidbody != null) { kulyaRigidbody.velocity =
gun.transform.forward * kulyaSpeed; } if (shotSound != null) {
AudioSource.PlayClipAtPoint(shotSound, transform.position); } } } using Unity.XR.CoreUtils;
using UnityEngine.Assertions; using UnityEngine.XR.Interaction.Toolkit; namespace
UnityEngine.XR.Interaction.Toolkit.Samples.StarterAssets { public class DynamicMoveProvider :
ActionBasedContinuousMoveProvider { private CapsuleCollider col; private float
originalMoveSpeed = 2.0f; private float modifiedMoveSpeed = 35.0f; private bool
areControllersMoving = false; private Vector3 lastLeftControllerPosition; private Vector3
lastRightControllerPosition; //private ActionBasedContinuousMoveProvider moveProvider; //
Посилання на компонент ActionBasedContinuousMoveProvider для отримання значення
moveSpeed public AudioSource footstepAudioSource; // Посилання на компонент AudioSource
для відтворення звуку шагів public enum MovementDirection { HeadRelative, HandRelative, }
[Space, Header(
```

Цитирования: **0,02%** id: **113**

"MovementDirection")][SerializeField]

[Tooltip(

Цитирования: **0,14%** id: **114**

"Directs the XR Origin's movement when using the head-relative mode. If not set, will automatically find and use the XR Origin Camera."

```
)] Transform m_HeadTransform; public Transform headTransform { get = m_HeadTransform; set
= m_HeadTransform = value; } [SerializeField] [Tooltip(
```

Цитирования: **0,09%** id: **115**

"Directs the XR Origin's movement when using the hand-relative mode with the left hand."

```
)] Transform m_LeftControllerTransform; public Transform leftControllerTransform { get =
m_LeftControllerTransform; set = m_LeftControllerTransform = value; } [SerializeField] [Tooltip(
```

Цитирования: **0,09%** id: **116**

"Directs the XR Origin's movement when using the hand-relative mode with the right hand."

```
)] Transform m_RightControllerTransform; public Transform rightControllerTransform { get =
m_RightControllerTransform; set = m_RightControllerTransform = value; } [SerializeField]
```

[Tooltip(

Цитирования: **0,13%** id: **117**

"Whether to use the specified head transform or left controller transform to direct the XR Origin's movement for the left hand."

```
)] MovementDirection m_LeftHandMovementDirection; public MovementDirection
leftHandMovementDirection { get = m_LeftHandMovementDirection; set =
m_LeftHandMovementDirection = value; } [SerializeField] [Tooltip(
```

Цитирования: **0,13%** id: **118**

"Whether to use the specified head transform or right controller transform to direct the XR Origin's movement for the right hand."

```
)] MovementDirection m_RightHandMovementDirection; public MovementDirection
rightHandMovementDirection { get = m_RightHandMovementDirection; set =
m_RightHandMovementDirection = value; } [SerializeField] [Tooltip(
```

Цитирования: **0,05%** id: **119**

"Deadzone value for Y-axis movement of the controllers."

```
)] float m_Deadzone = 0.05f; public float deadzone { get = m_Deadzone; set = m_Deadzone =
value; } Transform m_CombinedTransform; Pose m_LeftMovementPose = Pose.identity; Pose
m_RightMovementPose = Pose.identity; protected override void Awake() { base.Awake(); //
Отримати посилання на компонент AudioSource footstepAudioSource = GetComponent
AudioSource (); originalMoveSpeed = moveSpeed; m_CombinedTransform = new GameObject(
```

Цитирования: 0,04%	id: 120
"[Dynamic Move Provider] Combined Forward Source"	
<pre>).transform; m_CombinedTransform.SetParent(transform, false); m_CombinedTransform.localPosition = Vector3.zero; m_CombinedTransform.localRotation = Quaternion.identity; forwardSource = m_CombinedTransform; lastLeftControllerPosition = leftControllerTransform.position; lastRightControllerPosition = rightControllerTransform.position; } private bool isObstacleDetected = false; // Додали змінну для відстеження зіткнення з перешкодою private void OnTriggerEnter(Collider other) { if (other.CompareTag(</pre>	
Цитирования: 0,01%	id: 121
"obstacle"	
<pre>)) { isObstacleDetected = true; // Встановлюємо прапорець, що була знайдена перешкода // Зупиняємо рух, встановлюючи швидкості руху на 0 moveSpeed = 0.0f; } } protected override Vector3 ComputeDesiredMove(Vector2 input) { if (input == Vector2.zero) { areControllersMoving = false; return Vector3.zero; } // Відтворення звуку шагів if (input != Vector2.zero && footstepAudioSource != null && !footstepAudioSource.isPlaying) { footstepAudioSource.Play(); } // Перевірка, чи не рухається стік лівого контролера, і зупинка відтворення звуку шагів if (input == Vector2.zero && footstepAudioSource != null && footstepAudioSource.isPlaying) { footstepAudioSource.Stop(); } var currentLeftControllerPosition = leftControllerTransform.position; var currentRightControllerPosition = rightControllerTransform.position; if (!areControllersMoving && Mathf.Abs(currentLeftControllerPosition.y - lastLeftControllerPosition.y) = m_Deadzone && Mathf.Abs(currentRightControllerPosition.y - lastRightControllerPosition.y) = m_Deadzone && Mathf.Sign(currentLeftControllerPosition.y - lastLeftControllerPosition.y) != Mathf.Sign(currentRightControllerPosition.y - lastRightControllerPosition.y)) { areControllersMoving = true; } else if (areControllersMoving) { areControllersMoving = false; } lastLeftControllerPosition = currentLeftControllerPosition; lastRightControllerPosition = currentRightControllerPosition; if (!areControllersMoving) { moveSpeed = Mathf.Lerp(moveSpeed, originalMoveSpeed, Time.deltaTime); // Плавний перехід до оригінальної швидкості } else { moveSpeed = Mathf.Lerp(moveSpeed, modifiedMoveSpeed, Time.deltaTime); // Плавний перехід до модифікованої швидкості } if (m_HeadTransform == null) { var xrOrigin = system.xrOrigin; if (xrOrigin != null) { var xrCamera = xrOrigin.Camera; if (xrCamera != null) m_HeadTransform = xrCamera.transform; } } switch (m_LeftHandMovementDirection) { case MovementDirection.HeadRelative: if (m_HeadTransform != null) m_LeftMovementPose = m_HeadTransform.GetWorldPose(); break; case MovementDirection.HandRelative: if (m_LeftControllerTransform != null) m_LeftMovementPose = m_LeftControllerTransform.GetWorldPose(); break; default: Assert.IsTrue(false, \$ </pre>	
Цитирования: 0,02%	id: 122
"Unhandled {nameof(MovementDirection)}={m_LeftHandMovementDirection}"	
<pre>); break; } switch (m_RightHandMovementDirection) { case MovementDirection.HeadRelative: if (m_HeadTransform != null) m_RightMovementPose = m_HeadTransform.GetWorldPose(); break; case MovementDirection.HandRelative: if (m_RightControllerTransform != null) m_RightMovementPose = m_RightControllerTransform.GetWorldPose(); break; default: Assert.IsTrue(false, \$ </pre>	
Цитирования: 0,02%	id: 123
"Unhandled {nameof(MovementDirection)}={m_RightHandMovementDirection}"	
<pre>); break; } var move = base.ComputeDesiredMove(input); // Оновлюємо параметр pitch звуку шагів в залежності від значення moveSpeed if (footstepAudioSource != null) { footstepAudioSource.pitch = Mathf.Lerp(1f, 15f, moveSpeed / modifiedMoveSpeed); // Змінюємо pitch від 0.5 до 1.5 залежно від значення moveSpeed } //return move; var leftHandValue = leftHandMoveAction.action?.ReadValue Vector2 () ?? Vector2.zero; var rightHandValue = rightHandMoveAction.action?.ReadValue Vector2 () ?? Vector2.zero; var totalSqrMagnitude = leftHandValue.sqrMagnitude + rightHandValue.sqrMagnitude; var leftHandBlend = 0.5f; if (totalSqrMagnitude > Mathf.Epsilon) leftHandBlend = leftHandValue.sqrMagnitude / totalSqrMagnitude; var combinedPosition = Vector3.Lerp(m_RightMovementPose.position, m_LeftMovementPose.position, leftHandBlend); var combinedRotation = Quaternion.Slerp(m_RightMovementPose.rotation, m_LeftMovementPose.rotation, leftHandBlend); m_CombinedTransform.SetPositionAndRotation(combinedPosition, combinedRotation); return base.ComputeDesiredMove(input); } } } </pre>	

Заявление об ограничении ответственности:

Этот отчет должен быть правильно истолкован и проанализирован квалифицированным специалистом, который несет ответственность за оценку!

Любая информация, представленная в этом отчете, не является окончательной и подлежит ручному просмотру и анализу. Пожалуйста, следуйте инструкциям: [Рекомендации по оценке](#)

Детектор Плагиата - Ваше право на оригинальность! © SkyLine LLC