МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики,
інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

"До захисту допущена"
Завідувач кафедри комп'ютерних
наук та прикладної математики
Турбал Юрій Васильович
____ _____ 20___ р.

КВАЛІФІКАЦІЙНА РОБОТА

«РОЗРОБКА HORROR-ГРИ З ВИКОРИСТАННЯМ МЕХАНІКИ ГОЛОСОВОГО КЕРУВАННЯ»

Виконав: Гомон Максим Петрович
_____
(прізвище, ім'я, по батькові) *(підпис)*
група ІПЗ-41

Керівник: Асистент, Белозерова О. Д.
_____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) *(підпис)*

Рівне - 2025

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 52 с., 24 малюнки, 23 джерела.

Метою кваліфікаційної роботи є аналіз ринку та спільноти відеоігор, розробка власного продукту з унікальним методом взаємодії гравця з комп'ютером.

Об'єкт дослідження - процеси взаємодії відеоігор між користувачем та продуктом.
Предметом дослідження - ігрова механіка голосового керування.

Методи дослідження - аналіз джерел інформації, порівняння подібних продуктів, перегляд відгуків та рецензій спільноти.

Оскільки відеоігри є одним з найпопулярніших методів розваг на сьогодні, то реалізований мною проект, в рамках цієї кваліфікаційної роботи, дає змогу користувачам отримати те, заради чого вони й зацікавлені в ігровій індустрії - емоції - саме жанр хоррор дає найбільший їх спектр.

Ключові слова: UNITY, ХОРРОР, ВІДЕОІГРИ, КОМПОНЕНТ, C#, МЕХАНІКИ, КОНФІГ, ІНСПЕКТОР.

ВСТУП

Все своє існування людство намагалося знайти собі якесь заняття. Колись, у давнину, це було добування їжі, виробництво нових інструментів для збору ресурсів чи навіть розпалювання вогня. І завжди нам, людям, треба було розслаблятися і зайняти себе чимось крім роботи - так і з'явилися розваги. Ми малювали наскельні малюнки, робили музичні інструменти, танцювали чи поклонялися вищим силам через різноманітні обряди.

У нашому сучасному світі відпочинок набув нових і цікавих форм. Генерування кумедних картинок через штучний інтелект, будування робототехніки, запуск феєрверків у нічне небо. Одним з таких атракцій є відеоігри. Цей вид відпочинку з'явився за допомогою розробки відеоігор. Вони є не тільки одним з найпопулярніших засобів розваги, а ще й ефективним інструментом для дослідження абсолютно нових форм взаємодії між комп'ютером та людиною. Розвиток технологій дає змогу добавляти нові методи спілкування між двома світами. Одним з таких став новий спосіб: голосове керування. Цей спосіб взаємодії дуже стрімко підвищує рівень занурення та створює максимально унікальний досвід для будь-якого користувача.

Особливо виразно весь потенціал голосового керування можна застосувати в жанрі хоррор. При грі в такі ігри, завжди є психологічна і фізична напруга і ці почуття ще й можна загострити таким новим інтерактивним методом, який ще більше занурює гравця в атмосферу.

Дивлячись на стрімкий розвиток індустрії відеоігор та нові форми зв'язку між користувачем та комп'ютером, окремої уваги заслуговує дослідження цікавих ігрових механік. Через це у межах моєї кваліфікаційної роботи я максимально дослідив ринки відеоігор та їх цікаві і новітні механіки.

Мета: У моїй кваліфікаційній роботі основною метою було поставлено дослідити цікаві ігрові механіки, однією з яких є голосове керування, та розробити свій проект-гру з таким методом для взаємодії гравця та гри на абсолютно новому рівні.

Завдання: Як основні завдання мною було обрано суміш з дослідження вже готових хоррор проектів з механікою керування голосом, та розробка свого власного продукту з урахувань помилок та успіхів інших ігор.

Основні завдання:

- Дослідити ігровий ринок;
- Дослідити цікаві та новітні ігрові механіки;
- Аналіз хоррор-ігор контурентів;
- Розробка свого методу взаємодії гравця з ігровим світом за допомогою голосового керування;
- Розробка свого прототипу хоррор-гри в ретро стилістиці з раніше розробленим методом голосового керування;
- Оцінка ефективності голосового керування як інструменту, який надає користувачу новий досвід.

Актуальність теми: Сьогодні відеоігри - це невід'ємна частина життя сучасної людини. Це один з найкращих способів відпочинку, який не вимагає великих затрат і сил. З кожною хвилиною у світі з'являється новий та цікавий проект з потужним потенціалом, який може завоювати ринок. Люди купують, грають та отримують море задоволення від процесу.

Гра - це річ, яка буде актуальною завжди через свою специфікацію та початкового рівня входу. Але, на жаль, чим далі йде розвиток ігрової індустрії та технологій, тим більш вимогливими стають гравці по всьому світу. Через це розробники завжди стараються придумати нові методи для здивування гравців і дарування їм нового спектру емоцій та досвіду. Через це й з'явились ігри, які задали нові механіки в індустрію, і ці методи настільки могли подобатися людям, що вони переростали в нові окремі жанри відеоігор. Одним з таких прикладів є гра PUBG, яка створила новий жанр, а саме Королівська Битва.

Дивлячись на вищеперечислене, я вирішив дослідити більше інформації про розвиток цікавих методів у відеоіграх і розробити власний проект з унікальною в своєму роді механікою. Так мною було придумано гру з кодовою

назвою “Below the Void”, в якій головною унікальною особливістю було б голосове керування. Ідея проекту заключається в тому, щоб гравець своїм голосом міг отримувати видимість в ігровому середовищі, і в залежності від гучності змінювався б радіус зору. Детальніше в пунктах “2.4. Голосове керування” та “3.2 Голосове керування”.

Об’єкт дослідження: Об’єктом дослідження у моїй кваліфікаційній роботі є процеси взаємодії відеоігор між гравцем та грою. Самих методів такого зв’язку існує просто безліч, вони поділяються на 2 види: система управління та ігрові механіки.

Системне управління:

- Клавіатура;
- Мишка;
- Геймпади та контролери;
- Жестове керування;
- VR/AR;
- Голосове керування;
- Сенсорне керування.

Ігрові механіки:

- Діалоги;
- Крафтинг;
- Рольова система;
- Інші механіки.

Методи зв’язку системного управління можуть змінювати свій вид в залежності від контексту в якому вони використовуються в грі. Прикладом цього є голосове керування. Якщо у відеогрі цей зв’язок використовується як керування інтерфейсом - як приклад, голосова команда: “Запусти меню” - це рахується за системне управління, якщо він використовується як тригер - як приклад, гравець закричав = смерть персонажа - це рахується як ігрова механіка. Це стосується не тільки голосового керування, а й інших системних управлінь.

Предмет дослідження: Предметом дослідження у моїй кваліфікаційній роботі є голосове керування, в даному контексті мого проекту - це є повноцінною ігровою механікою. На відміну від інших схожих хоррор-ігор, цей метод занурення підсилює різні відчуття гравців під час ігрового процесу за рахунок того, що ця система впливає на шанси користувача вижити в ігровому середовищі. Цей підхід надає можливість користувачу обережно використовувати свій головний ресурс - голос, адже кожна його фраза чи звук може стати для нього останнім. Це значно посилює атмосферу гри для гравця, що робить ігровий досвід для гравця більш цікавим і заставляє його мозок виділити місце в своїй пам'яті для цього продукту.

Отже, можна зробити висновок, що для гравця голос стає не лише якимось новим методом керування, а й повноцінною частиною ігроладу, яка впливає на все, що буде відбуватися в грі.

Методи дослідження: Як методи дослідження в моїй кваліфікаційній роботі для максимально точного результату, було обрано певний список, щоб якнайкраще провести дослідження.

Використані методи дослідження:

- Аналіз джерел інформації;
    1. Перегляд рецензій користувачів;
    2. Аналіз постів в соціальних мережах та ігрових форумах.
- Порівняння хоррор-ігор;
- Спостереження за відео-проходженнями різних хоррор проектів;
    1. Оцінка емоційного стану гравців під час ігроладу;
    2. Оцінка рівня занурення в ігри;
- Анкетування розробленого продукту;
    1. Аналіз відгуків користувачів;
    2. Оцінка розробленого продукту від тестувальників.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ІГРОВОГО РИНКУ

1.1. Зворотній зв'язок від спільноти

Під час дослідження перевірених часом джерел інформації, було переглянуто кілька позитивних і негативних рецензій користувачів на рахунок конкретних відеоігор. Ігри вибирались за різним ігроладом. Переглядалися лише виключно відгуки, в яких користувачі коментували ігрові механіки.

Список ігор, які брали участь в аналізі:

- **Dead By Daylight [1]** - це багатокористувацький сурвайвл-хоррор (виживання в жанрі жахів), де 4 гравці (вижившi) намагаються втекти від іншого гравця (маньяк);
- **Outlast [2]** - це сурвайвл-хоррор (виживання в жанрі жахів), в якому гравцю потрібно пройти сюжетку кампанію граючи за кореспондента, який пішов досліджувати закинуту психлікарню;
- **Five Nights at Freddy`s [3]** - це сурвайвл-хоррор (виживання в жанрі жахів), в якому гравець, який грає за охоронця піцерії, повинен пережити 5 ночей в своєму офісі проти аніматроніків/роботів.

1.1.1. Перегляд рецензій користувачів

Відгуки Dead By Daylight [4]

Список **позитивних** відгуків від гравців:

- Користувач Kvïtyk:

“Що найбільше мені подобається в цій грі, так це варіативність - купа різних персонажів, з дуже різноманітними скілами - є з чого вибрати! Кожен по-своєму крутий (а я ще навіть не всіх спробував)! І найкраще в цій грі - грати з кимось ;-)” - Гравець описує, що йому найбільше подобається різноманітність здібностей в персонажів, які є однією з основних механік гри;

- Користувач Dmytro Joestarovych:

“Загалом концепція гри мені дуже подобалась: 4 втікачі координуються один з одним для спільної мети - завести 5 з 7 генераторів та втекти. Однак окрім них є ще п'ятий гравець в якого абсолютно протилежна мета - принести в жертву якомога більше втікачів. Я не буду описувати усі механіки гри, бо банально не вистачить символів. Однак маю сказати, що розробники постійно працюють над новим контентом в грі, постійно балансують предмети та перки тому те, що працювало минулого року може бути не таким ефективним зараз.“ - Гравець описує, що йому в загальному подобається концепція гри і її головна механіка з ремонтуванням генераторів, задля перемоги команди;

Список **негативних** відгуків від гравців:

- Користувач Yolo:

“Якщо у вас слабка психіка - не купуйте це, в цій грі ви будете втікати від типа, в якого зброя,а у вас - нічого. Що тут цікавого? Як на мене, абсолютно нічого.“ - Гравець описує, що йому не подобається механіка гри, що команда не може дати відпір протилежному гравцю;

- Користувач xal:

“ГРА ДЛЯ КЕМПЕРІВ І ТУНЕЛІВ, МЕХАНІКА ГРИ ПОЛОМАНА В НОЛЬ, ТОРГОВКА ЧЕРЕПІВ ЙДЕ ♥♥♥♥♥(до біса) ВОБЩЕ ЗІ СВОЇМ ТРИКУТНИКОМ, НЕ КУПУЙТЕ ЦЮ ГРУ, НАРКОТИКИ І ТО КРАЩЕ НІЖ ЦЯ ♥♥♥♥♥♥(дурня).“ - Гравець агресивно і з ненормативною лексикою описує, що йому не подобається боротьба розробників з так званими “переслідуванням одного гравця” та “контролюванням одного гравця”.

Відгуки Outlast [5]

Список **позитивних** відгуків від гравців:

- Користувач Nachtigall:

“Тобі доводиться ховатись у шафах, під ліжками, затамовувати подих у темряві, сподіваючись, що той виродок із ножем не загляне всередину.” -

Гравець описує, що йому подобається одна з основних механік, а саме хования в різних місцях від ворогів;

- Користувач пінка:

“Гравець ні на хвилину не залишається без активності. Безліч скрімерів, (часом дуже несподіваних) поворотів сюжету, нових завдань і перешкод, що постають на шляху гг.“ - Гравець описує, що йому до вподоби скримери (різкий звук/відеоряд) та система перешкод;

Список **негативних** відгуків від гравців:

- Користувач Max Gerald:

“... батарейки сідають ой як стрімко, надто в нічному режимі. Та подібна родзинка stealth horror'ів – це водночас і слабкість, адже в сумі розробники не подають нічого, крім нескінченних хованок і примітивних забігів на киталт "покрути два вентилі".“ - Гравець описує, що йому не до вподоби цей старий спосіб залякування користувачів;

- Користувач Punk:

“Сам гг втомлює своїм небажанням захищатися. На початку гри ще зрозуміло, але коли вже кілька годин ти ходиш по такому небезпечному місцю, де за кожним поворотом тебе можуть кокнути, ну невже не можна взяти хоч якусь палицю в руки і відбиватися?“ - Гравець описує, що йому не подобається те, що головний герой не може відбиватися від ворогів.

Відгуки Five Nights At Freddy`s [6]

Список **позитивних** відгуків від гравців:

- Користувач F.i.x.E:

“Логіка проста: не спати, не моргати і не сідати в туалет. Щоб вижити, треба стежити за камерами спостереження, тримати двері закритими і не забути при цьому не подихати від страху.” - Гравець описує, що йому подобається простота ігроладу та механік гри;

- Користувач Korivka:

“Все що ви можете - спробувати закритись за дверми, які ці неживі істоти зламають, якщо гравець буде робити все не швидко.“ - Гравець описує, що це хороший хоррор через те, що головний герой повинен все робити швидко, адже якщо робити це повільно - ти програєш;

Список **негативних** відгуків від гравців:

- Користувач risuyunatvoixgubax:

“ТУТ НЕ МОЖНА ХОДИТИ“ - Гравець описує, що йому не подобається, що головного героя прикріплюють до одного місця і не дають змоги подорожувати по локації;

- Користувач Zavzdy Eshkere:

“Мені не подобається, що гравець не бачить на камерах рух аніматроніків“ - Гравець описує, що йому абсолютно не до вподоби, що користувач не бачить рух самих роботів по локації на камерах.

З проаналізованих рецензій користувачів платформи Steam, можна зробити кілька висновків. А саме те, що гравці потребують: не занадто важкі, але й не занадто прості механіки гри; волю до дій, як підбирання предметів чи не обмежений рух по локаціям; активної гри.

1.1.2. Аналіз постів в соціальних мережах та ігрових форумах

Для дослідження використовувались внутрішній ігровий форум в Steam та соціальна мережа Instagram. Було проаналізовано, який контент користувачам подобається найбільше, це було зроблено для того, щоб зробити висновок по тому, як зробити якісний продукт для потенційного користувача.

Dead By Daylight - користувачі максимально обожнюють творчий та розважальний контент. Люди люблять створювати різноманітні цифрові картини та робити смішні скріншоти в грі.

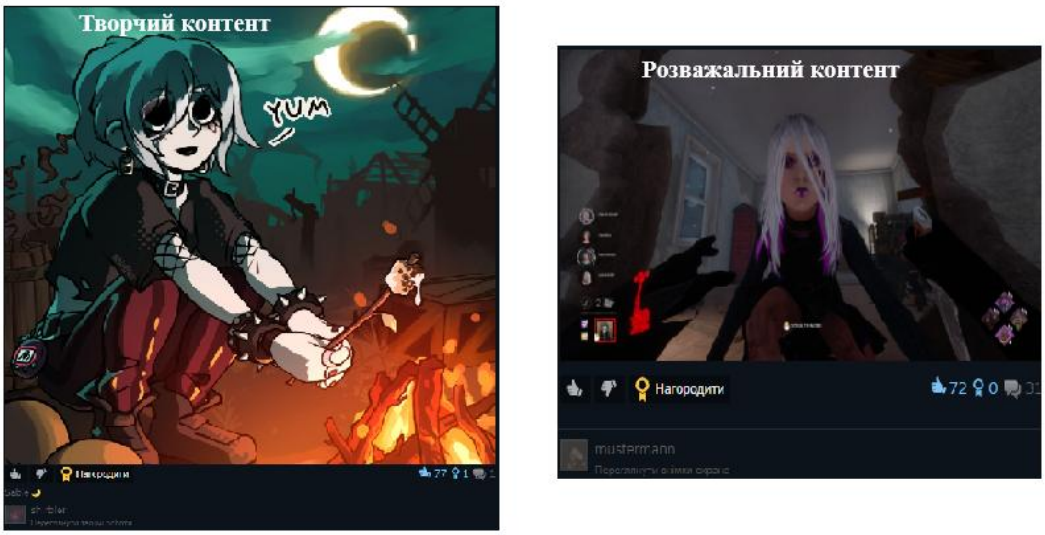

Рис. 1.1. Вид контенту по Dead By Daylight.

Outlast - користувачі зачасту поширюють свої творчі роботи та атмосферні знімки екрану. По публікаціям видно, що кожен в цій спільноті цінує енергетику цієї гри.

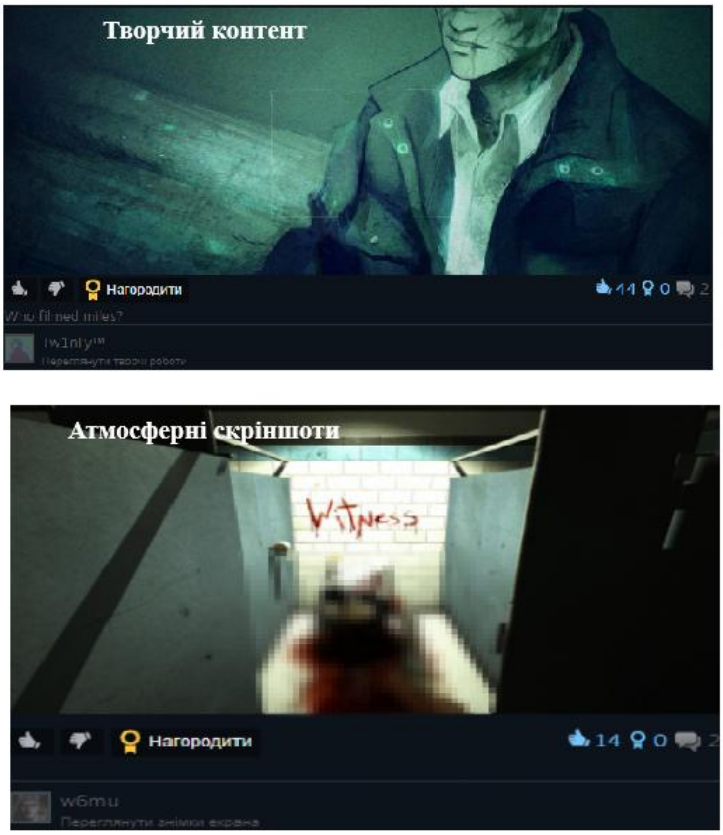

Рис. 1.1. Вид контенту по Outlast.

Five Nights At Freddy`s - користувачам найбільше подобається сюжетний та розважальний контент. Людям подобається будувати теорії та розв'язувати загадки.

Рис. 1.1. Вид контенту по Five Nights At Freddy`s.

Якщо узагальнити всю отриману інформацію, то можна прийти до висновків, що найбільш гравці цінують такий контент як:

- Розважальний;
- Творчий;
- Сюжетний;
- Різноманітні знімки екрану.

Тобто, один з методів досягнення поваги спільноти гравців - надати їм можливість створювати такий контент. Коротко про можливі рішення, які можна задіяти під час розробки задля розвитку спільноти:

1. *Розважальний* - в деяких етапах гри добавити абсурдні діалоги або ситуації;
2. *Творчий* - створити естетичні та продумані ігрові моделі ворогів чи персонажів, щоб з їх натури могли робити цифрові картини;
3. *Сюжетний* - написати сильний та цікавий сюжет, який не розкривати до кінця, щоб спільнота могла це обговорювати на різноманітних форумах чи соціальних мережах;

4. *Різноманітні знімки екрану* - максимально наповнити локації атмосферними та естетичними предметами. Зробити ігрові локації гарними та такими, щоб гравець на кожному кроці хотів зробити знімок;

1.1.3. Дизайнерська стилістика

Головною стилістикою гри було обрано ретро-стиль. Він ідеально підходить для хоррор жанру через психологію людини. Нечітка графіки, візуальний шум, розпливчасте зображення - це викликає не лише ефект ностальгії в гравця, а й створює тривожну атмосферу гри. Під час ігроладу мозок починає будувати те, чого може навіть і не існувати.

Основні причини чому ретро стиль лякає:

- "Заборонена гра" - це ефект при якому людина відчуває, ніби вона знайшла щось "заборонене" і старе, ніби це не мало існувати;
- Недосконалість комп'ютерної графіки - коли людина не бачить картинку на моніторі детально, а мозок сам починає домальовувати страшні деталі. Все це через тусклі кольори, графіку з елементами піксельності чи не ідеальні анімації;
- "Моторошна долина" - зазвичай це відносять до зображень людей, які створені штучним інтелектом, але в даному випадку графіка гри також викликає цей ефект відразу через асоціації з реальністю;
- Звуки - зазвичай вони мають обмеження в 8-біт або 16-біт та звучать максимально неприродньо. Цей ефект нечіткості та неякісності створює відчуття напруги та тривоги.

Під час аналізу ринку я знайшов кілька хоррор відеоігор в ретро стилістиці. Цими проектами є: Elevated Dread та Rewind or Die. Ці проекти є прямим підженням всім вищенаписаним причинам страху в людей від такого стилю.

Рис. 1.1. Зображення з гри Elevated Dread **[7]**.

Рис. 1.1. Зображення з гри Rewind or Die **[8]**.

Дивлячись на зображення з ігор виникає відчуття тривожності, чогось забутого та тривоги. Ці сцени викликають погані асоціації і цим викликають психологічний ефект.

1.1.4. Спостереження за відео-проходженнями хоррор проектів

Для дослідження емоційного стану та занурення гравців у хоррор-ігри, я використовував відеохостинг Youtube. Вибір впав на цю платформу через дуже велику кількість різноманітного контенту з проходженням та обговоренням ігор.

Оцінки емоційного стану та рівня занурення в гру:

1. **Dead By Daylight:**

- **Відео:** Огляд гри Dead By Daylight Українською! **[9]**
- **Коментарі автора:** "*Гра справді передає тобі відчуття наче на тебе полюють. І це дуже круто, коли ти щиро лякаєшся, коли на тебе вилітає ворог. Кастомізація персонажу впливає на вижевабильність персонажу, а для особливо хардкорних чуваків, тут завезли комплекти яскравого одягу.*"

2. **Outlast:**

- **Відео:** Outlast#1. Я не твоє Поросятко! Хрю. (УкрСуб) **[10]**
- **Емоції та занурення автора:** Гравець не один раз під час гри кричав від різних різких звуків чи лякливого відеоряду. Зазначав, що гра має атмосферні звуки та музику, які доповнюють ще більше посилюють страх. Користувач постійно напружений і боїться кожного шороху та повороту. Йому подобається наповнення локацій та дизайн рівнів.

3. **Five Nights At Freddy`s:**

- **Відео:** ВПЕРШЕ граю у FIVE NIGHTS AT FREDDY`S 1 **[11]**
- **Емоції та занурення автора:** Геймерка переживає майже кожну секунду, у неї постійне напруження через страх, який викладає гра. Вона в будь-якій страшній ситуації прикриває лице руками через захисну реакцію організму.

Проаналізувавши всю інформацію з відео, можна заключити певні висновки. Кожен з авторів отримував дозу адреналіну під час гри та естетичне задоволення від антуражу гри. Тобто, ідеальна хоррор-гра повинна мати в собі кілька характеристик: момент несподіванки, гучні звуки, атмосферні локації, свою унікальну стилістику, та надавати гравцеві нові емоції.

Рис. 1.1. Емоції під час гри авторки відео “ВПЕРШЕ граю у FIVE NIGHTS AT FREDDY`S 1”.

1.2. Порівняння хоррор-ігор

Під час аналізу рецензій та контенту по відеоіграм: Dead By Daylight, Outlast та Five Nights At Freddy`s - було зроблено певні висновки. Дослідження дало нам інформацію про контент, який до вподоби гравцям та які цікаві механіки в іграх їм подобаються та не подобаються.

Основна мета зробити максимально цікавий ігролад, який унікальна ігрова механіка буде доповнювати. Цією механікою якраз і буде голосове керування, яке доповнить гру. Також потрібно надати спільноті можливість робити свій власний контент і популіризувати гру серед інших гравців.

Ці ігри не зовсім можна порівнювати, адже це все унікальні та водночас схожі проекти. В них різна ідея, яка і робить їх особливими й такими цікавими для різноманітних користувачів.

Dead By Daylight - це про співпрацю команди заради однієї цілі;

Outlast - це про страз зайти за кожен куток;

Five Nights At Freddy`s - це про уважність та знання поведінки ворогів.

РОЗДІЛ 2 РОЗРОБКА ХОРРОР-ГРИ В РЕТРО СТИЛІСТИЦІ З ВИКОРИСТАННЯМ ГОЛОСОВОГО КЕРУВАННЯ

2.1. Вибір інструментів та технологій

2.1.1. Ігровий рушій та мова програмування

Під час будування плану по розробці, одним з найважливішого було вибрати ігровий рушій та мову програмування, адже від цього залежала якість, гнучкість та легкість написання проекту. Мій вибір зупинився на Unity, адже в ньому я маю найбільше досвіду серед всіх ігрових рушіїв.

Unity [12] - це рушій, який дає змогу розробникам створювати 2D та 3D відеоігри, та застосунки різної складності за допомогою мови C#. Зручний інтерфейс, підтримка різних видів фізики, анімації, системи частинок та шейдери - *це про Unity*.

Переваги Unity:

- Багатоплатформний - дозволяє експортувати свої ігри на ПК, консолі PlayStation та XBOX, смартфони Android та IOS, VR/AR;
- Мова C# - одна з найгнучкіший мов програмування, яка відкриває великий спектр взаємодії між компонентами та об’єктами;
- Зручний інтерфейс - Unity має великий список різноманітних функцій, які можна легко знайти та використати;
- Asset Store - великий магазин інтернет товарів для Unity, він включає в себе: моделі, спрайти, аудіо, інструменти, плагіни;
- Безкоштовний - для приватного користування, можна використовувати безоплатно.

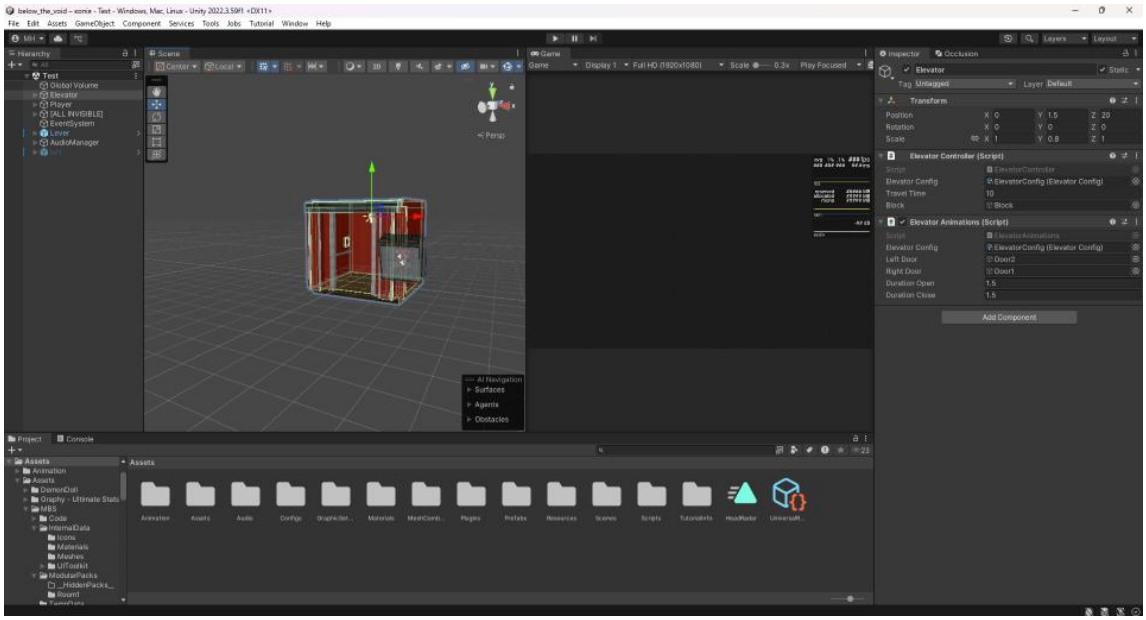

Рис. 2.1. Інтерфейс Unity.

C# [13] - це безпечна і зручна об’єктно-орієнтована мова програмування, яка була розроблена як покращена версія C++. Інтеграція з платформою .NET відкриває доступ до величезної кількості бібліотек.

Переваги C#:

- Синтаксис - максимально читабельний та простий код, який схожий на інші популярні мови;
- Garbage Collector - це вбудований механізм .NET, який автоматично аналізує виконання програми і дає змогу виділяти та видаляти об’єкти з пам’яті;
- .NET - це екосистема, яка надає C# доступ до безмежної кількості бібліотек, інструментів та фреймворків;
- Об’єктно-орієнтоване програмування - це парадигма програмування, яка дозволяє об’єктам взаємодіяти між собою, що дає змогу використовувати поліморфізм, інкапсуляцію та наслідування.

2.1.2. Додаткове програмне забезпечення

Під час розробки свого проекту для кваліфікаційної роботи, я використовував кілька додаткових програм для полегшення процесу. Ними

стали відомі GitHub та Blender. Вони були використані для більш зручного створення проекта.

GitHub [14] - це одна з найпопулярніших веб-платформ для спільної розробки проектів від Microsoft, яка працює на системі керування версіями Git. Цей інструмент дає змогу керувати змінами в репозиторіях.

Для розробки я використовував *GitHub Desktop* - це програмне забезпечення від розробників GitHub, яке надає користувачам максимально зручний і гнучкий графічний інтерфейс для керування всіма своїми репозиторіями.

Причини використання для мене є простими, як і для будь-якого розробника, а саме:

- Резервна копія - під час виникнення критичних помилок, завжди є можливість завантажити старішу версію проекта;
- Історія - можливість переглядати, коли і який коміт був завантажений.

Blender [15] - це програма для створення тривимірної комп'ютерної графіки. Має велику кількість інструментів для роботи з моделями та середовищем. Цінується спільнотою через малу вагу, швидкодію та наявність підтримки багатьох версій операційних систем.

Причини використання є такими:

- Інтерфейс - зручний і максимально зрозумілий, навіть для початківця;
- Простота - для ретро-стилістики дуже підходить, адже не потрібно вивчати всі аспекти програми;
- Безкоштовно - програму можна завантажити безоплатно в Steam.

Trello [16] - це популярний web-сайт для зручної побудови плану. Сервіс дає змогу відмічати по категоріям стадії реалізації завдання, такі як "Потрібно зробити", "В роботі", "Зроблено" і будь-які інші власно придумані.

Причини використання є такими:

- Планування - зручний доступ до перегляду загальної картини

проекту;

- Контроль - швидке відстеження виконаних етапів у проекті.

2.1.3. Використані компоненти та інструменти в Unity

Для реалізації проекту були використані як і вбудовані в Unity компоненти та інструменти, так і завантажені з офіційного магазину *Asset Store* **[17]**. З вбудованих компонентів було використано: Rigidbody, Transform, всі види Collider-ів, Light, Character Controller, Audio Source, Animator **[12]**. У список компонентів з Asset Store попали: Mesh Combiner, Quick Outline, MBS - Modular Building System, Mesh Exploder, DOTween. Також всі компоненти в Unity являються класами.

2.1.3.1. Вбудовані компоненти та інструменти

Rigidbody - це компонент, який надає об'єкту фізичну поведінку. Після застосування і налаштування, об'єкт може взаємодіяти з навколишнім фізичним світом. Загалом потрібен для фізичної симуляції;

Transform - це компонент, який відповідає за позицію, рівень оберту, масштаб об'єкту, дочірні та батьківські зв'язки. Являється одним з найголовніших компонентів в Unity;

Collider (Колайдер) - це компонент, який надає об'єкту фізичну форму і використовується для визначень зіткнень на сцені. Має такі форми як *Sphere(Сфера), Capsule(Капсула), Box(Куб), Mesh(Сітка)*;

Light - це компонент, який робить з об'єкта джерело світла і дозволяє освітлювати територію в налаштованому радіусі;

Character Controller - це компонент, який використовується для керування від першого або третього лиця. Не працює з фізикою Rigidbody, адже вони мають дві різні системи фізики. Також має вбудований Capsule Collider;

Audio Source - це компонент, який повинен програвати аудіо(звуки,музику) на сцені. Працює в парі з Audio Clip. Також на сцені повинен бути компонент Audio Listener, щоб звук передавався до динаміків;

Animator - це компонент, який надає гнучке керування анімаціями. Можна задавати безліч умов, переходів і станів анімацій.

2.1.3.2. Сторонні компоненти та інструменти

Mesh Combiner [18] - це компонент, який дозволяє об'єднати кілька моделей в одну. Сітка, матеріал, колайдер - це все, що об'єднує в один об'єкт інструмент;

Quick Outline [19] - це компонент, який надає можливість робити різноманітну підсвітку/обведення навколо об'єкту. Має кілька гнучких налаштувань, яких вдосталь для будь якої потреби;

MBS - Modular Building System [20] - це інструмент, який полекшує будування рівня. Він дає змогу зручно розставляти різноманітні стіни, підлогу, предмети;

Mesh Exploder [21] - це компонент, який дозволяє зробити ефект вибуху/руйнування об'єкту на сцені. Дуже гнучкі налаштування, які допомагають досягти потрібного результату;

DOTween [22] - це бібліотека, яка дає змогу легко анімувати/переміщати об'єкти через код за допомогою різноманітних кривих.

2.2. Дизайн ігрового процесу

2.2.1. Основні елементи хоррору

Як завжди було відомо, першочергове завдання будь якої гри в жанрі жахів - це викликати певний спектр емоцій в гравців. Найбільш ключевими емоціями є страх, переживання, напруженість, нерозуміння, дизорієнтація та тривога. Вони сприяють, щоб гравець проникся процес та у всю атмосферу відеогри.

Для досягнення таких ефектів розробники та інші творці протягом кількох десятиліть придумували різноманітні способи, щоб викликати придумали певний список базових принципів для хоррор індустрії, які були мною використані в розробці відеогри, а саме:

- Елементи сурвайвл;
- Графіка;
- Сетинг гри та дизайн рівнів;

2.2.2. Опис та реалізація елементів хоррору в проекті

2.2.2.1. Елементи сурвайвл

Основне:

Головними принципами сурвайвл хоррорів є те, що гравець є майже беззахисним, він ховається та тікає від ворогів, обмежений кількістю ресурсів та має почуття постійної загрози.

Реалізація:

Гравець може підбирати лише необхідні для проходження гри предмети, якими він ніяк не може відбиватися від ворогів. Також всі рівні є максимально темними, користувач не бачить нічого, поки не скористається мікрофоном для освітлення території, при цьому світло може спровокувати ворогів на мапі - цією темрявою викликається почуття постійної загрози та страх кожного повороту, а використання головної механіки гри стає дуже дорогоцінним ресурсом, який

потрібно використовувати лише за необхідності. Про реалізацію головної механіки буде детальніше розказано в пункті “3.2. Голосове керування”.

2.2.2.2. Графіка

Основне:

Як було розписано в пункті “1.1.3. Дизайнерська стилістика”, мною було обрано ретро стиль. Цей вид графіки старається повторити візуал PS1 та PSP консолей. Головними критеріями якої є: низьке розширення, низька деталізація текстур, низько полігональні моделі.

Реалізація:

Розширення - для досягнення низької якості картинки, я використав Universal Render Pipeline(надалі URP) **[23]**. Він має широкий список налаштувань, яким можна зробити такий ефект зображення. Я знизив рівень Render Scale до 0.31 в пункті Quality, який знаходиться в автоматично створеному файлі URP-High Fidelity, також у ньому було знижено і якість освітлення та тіней до мінімального.

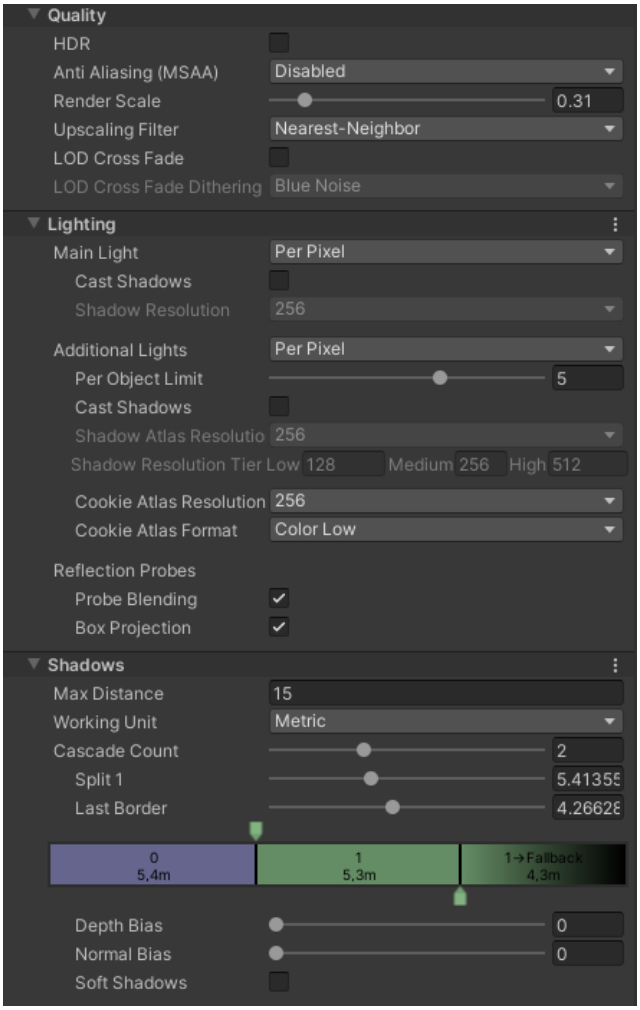

Рис. 2.2. Налаштування в файлі URP-High Fidelity.

У головному файлі пост-обробки, який я створив, було добавлено і налаштовано ефекти:

- Bloom (Ефект сяйва) - розсіяння меж світла та збільшення його м'якості;
- Film Grain (Фільмовий шум) - імітація потертості плівки;
- Shadows Midtones Highlights (Тіні, середні тони, світло) - контроль яскравості в різних діапазонах;
- Vignette (Віньєтка) - затемнення зображення по краях;
- Color Curves (Криві кольорів) - зміна властивостей кольорів через криву;
- Lift Gamma Gain (Підйом, гамма, посилення) - контроль над темними, середніми, світлими тонами;

- Panini Projection (Проекція Паніні) - перспективна трансформація, яка змінює/викривлює кут зображення.

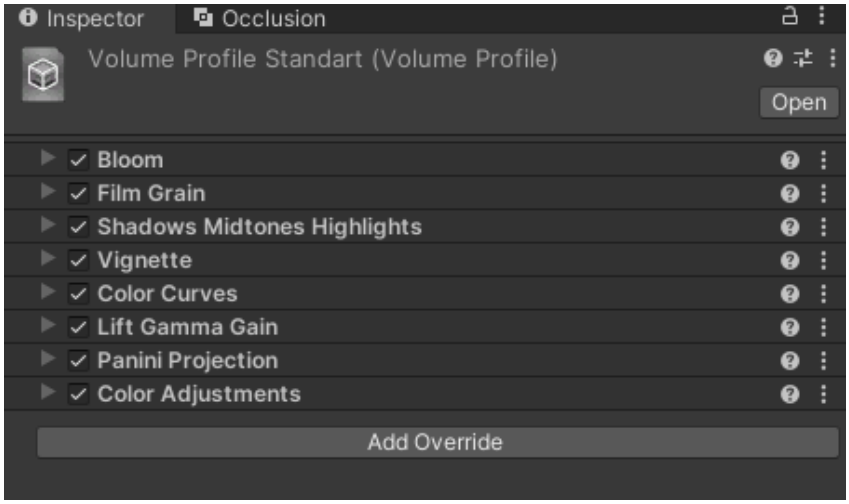

Рис. 2.2. Список ефектів в головному файлі пост-обробки.

Текстури - задля досягнення низької якості текстур, було використано зображення з розширенням від 32x32 до 512x512 пікселів. До цього всього було використано метод компресії “Normal Quality” та знижено якість відображення текстури до 50% через Import Setting, яке є вбудованим в Unity.

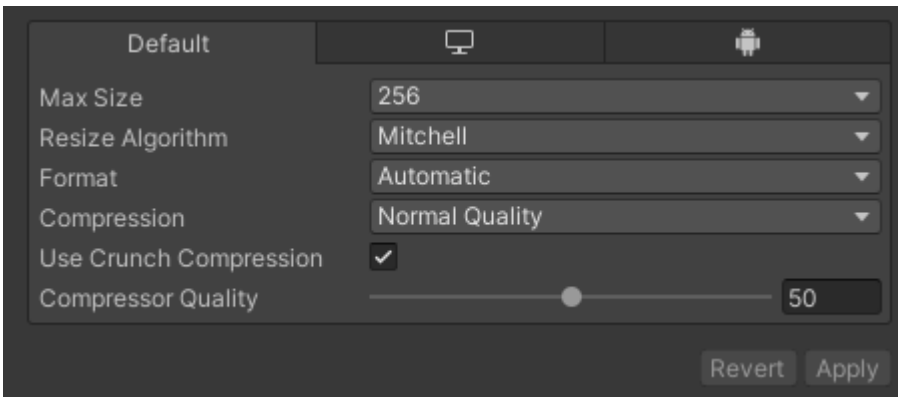

Рис. 2.2. Налаштування однієї з текстур.

Моделі - використовувалась програма Blender, в якій була використана функція обмеження максимальної кількості полігонів. Також при необхідності створення відносно простих моделей, я використовував примітивні об’єкти в Unity, а саме куби, сфери, платформи, капсули, циліндри.

2.2.2.3. Сетинг гри та дизайн рівнів

Основне:

Сетинг - це сукупність всіх елементів гри, таких як: час, сюжет та стилістична характеристика - які в результаті ідентифікують весь світ, в якому будуть відбуватися події гри. З сетингу завжди випливає дизайн рівнів. У моєму проекті я вирішив обрати пост-апокаліпсис обстановку змішаною з пост-панк антуражем, які поєднуються в блідий Всесвіт.

Реалізація:

Час - всі події відбуваються в недалекому майбутньому, а саме 2029 році, у світі стався масштабний апокаліпсис, після якого людство зникло.

Сюжет - людство знищено через помилку американської команди вчених, весь звичний світ зник через детонування абсолютно всіх труб і джерел з природним газом. Єдиним живим куточком на планеті залишився дуже глибокий бункер, в якому розроблялися об'єкти за допомогою генної інженерії. Вони лежали в штучній комі і були готові до вилазок на випадок ядерної війни. Головний герой - це один з цих об'єктів, але він був невдалим. Його головна місія була освітлювати прокопані дороги для збройних сил, але через відпір імунної системи - він міг це робити лише при розмові. Після катастрофи він прокинувся, але нажаль не сам.

Стилістична характеристика - основний акцент зроблено на приглушених кольорах та на візуальному стилі рівнів. Один рівень - це один поверх, адже події відбуваються в бункері. Ним може бути як і зона відпочинку, так і місце збереження чи розробки об'єктів. Поверхи майже пусті; по всіх куточках купки бетону, які обвалилися після вибуху; валяються ключ карти; на стінах шпалери або вапно, підлога дерев'яна або бетонна. Вороги можуть мати вигляд як і жива істота, так і як купка різноманітних механізмів.

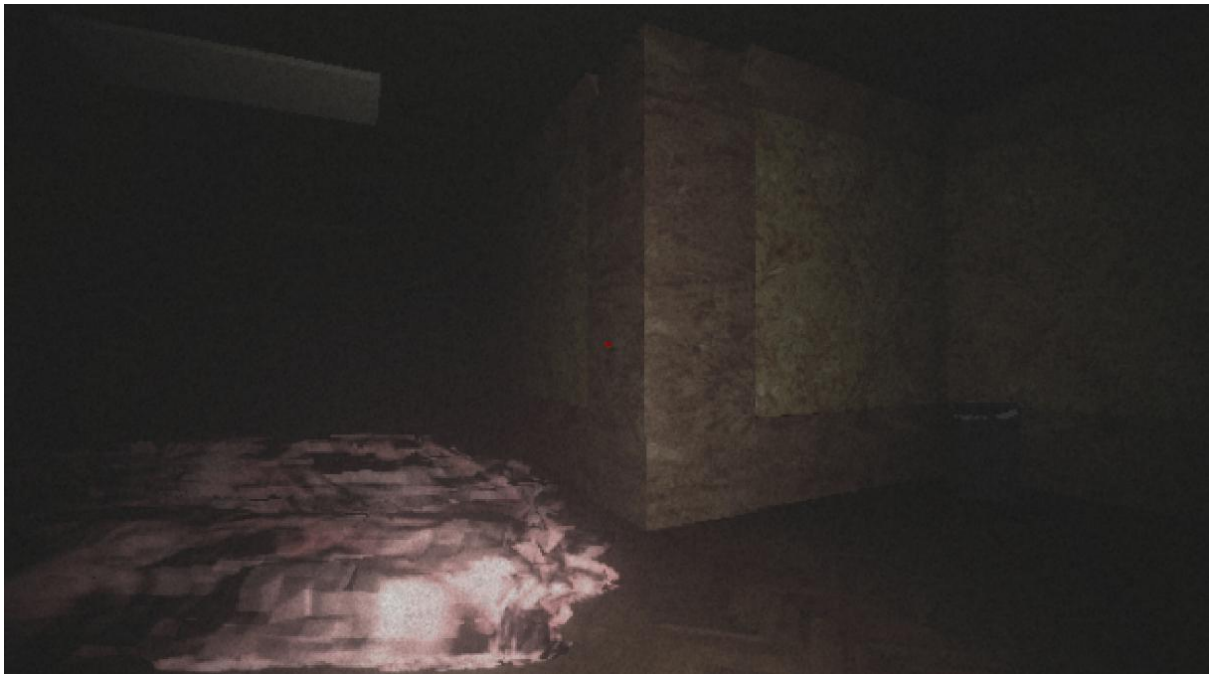

Рис. 2.2. Загальний вигляд стилізованої графіки.

2.3. Основні механіки

Було розроблено чіткий план по реалізації всього проекту і в тому числі основних механік. Всі ці завдання були додані до віртуальної дошки за допомогою web-сайту Trello. Всі уже реалізовані завдання переміщалися в категорію “Готово”.

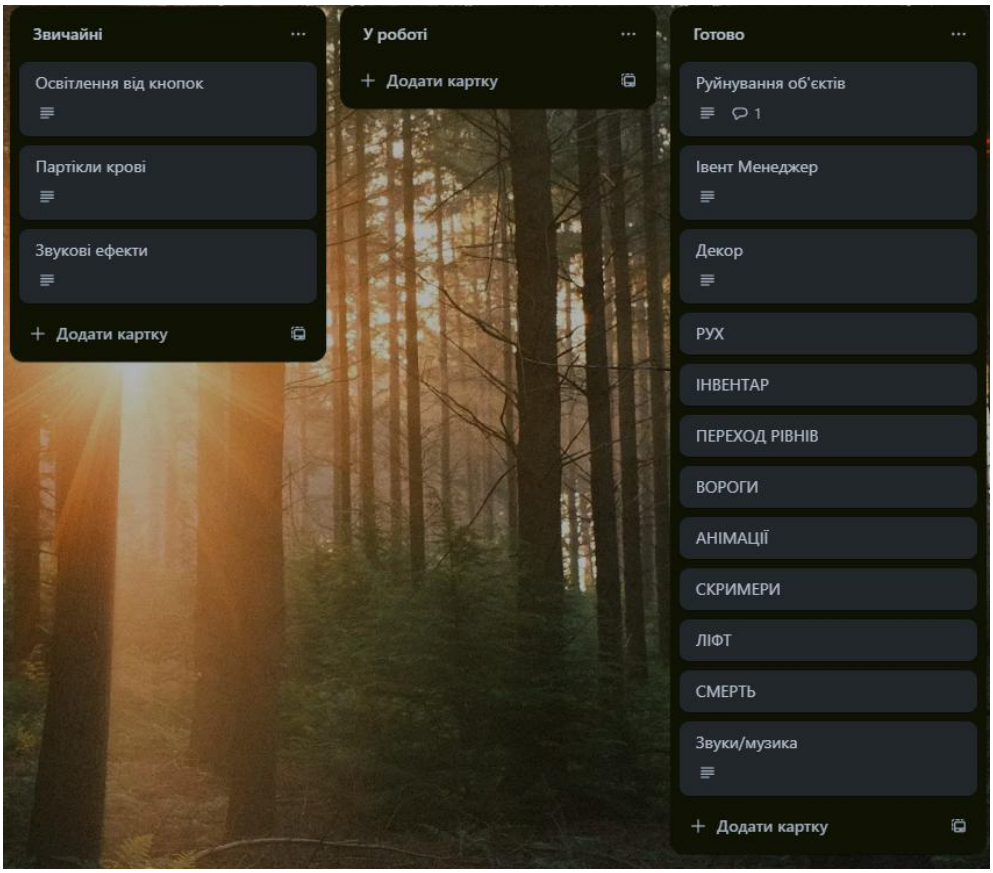

Рис. 2.3. Віртуальна дошка проекту на Trello.

Основними механіками в контексті цього проекту, можна назвати самі базові речі для будь-якої гри, такі як:

- Рух;
- Інвентар;
- Перехід між рівнями;
- Руйнування об’єктів;
- Збереження даних;
- Звуки.

Також до них можна віднести й основні механіки будь-якої хоррор гри, такі як:

- Вороги;
- Скримери;

Усі ці механіки стали основою для побудови повноцінного ігроладу, який повинен відповідати будь-яким відеоіграм і в тому числі й в жанрі хоррор. Кожна з них розроблялась, тестувалась і покращувалась поетапно.

2.4. Голосове керування

У грі як унікальна особливість була використана цікава і доволі нечаста механіка, а саме голосове керування. За задумкою, користувач повинен кричати/говорити в мікрофон, щоб отримувати хоч якусь видимість на рівні. Від гучної вхідного звуку залежить інтенсивність і дальність світла, яке персонаж починає випромінювати. Також на світло, яке генерує гравець, можуть звертати увагу вороги, які існують на рівні. Чи зверне увагу ворожий персонаж - залежить від інтенсивності на радіусу світла, якщо світло слабке - ворог не помітить нічого, якщо сильне - помінить.

Ця механіка ставить гравця перед вибором між безпекою на необхідністю бачити хоч щось на локації. Це не тільки робить ігровий процес важчим для користувача, а й сприяє емоційному “вибуху” та зануренню у світ відеогри.

РОЗДІЛ 3 РЕЗУЛЬТАТ РОБОТИ

В проекті було реалізовано всі основні принципи, які були зазначені в пункті “2.3. Основні механіки”. Також було розроблено унікальну механіку взаємодії, про яку було розказано в пункті “2.4. Голосове керування”.

3.1. Реалізація основних механік

1. Рух.

Основне: Найчастіше, що робить гравець - це переміщення, і за мету було поставлено, зробити його максимально зручним і без лишнього, непотрібного функціоналу. Стрибки були прибрані після етапу тестування, адже користувачі рідко його використовували, бо вони ніде не були потрібний.

Реалізація: Скрипт PlayerController - З конфігу дістаються значення: звичайна швидкість, швидкість бігу, швидкість повороту мишки. В методі Move відбувається перевірка чи біжить гравець, після чого за допомогою (Input.GetAxis("Horizontal") та Input.GetAxis("Vertical") визначається в яку сторону дивиться гравець і це множиться на швидкість. Після цього в Character Controller задаю рух через .Move().

Для крутіння камерою дізнаємося (Input.GetAxis("Horizontal") та Input.GetAxis("Vertical") і множимо це на швидкість повороту мишки. Після чого по простій формулі визначаємо градус, на який повинна бути повернута камера.

2. Інвентар.

Основне: Для того, щоб добавити в гру невелику систему квесту, було прийнято рішення добавити предмети, один з яких це ключ карта, яка дає доступ до важеля, який подає енергію на ліфт. Для цього було потрібно зробити невелику систему по підбиранню та зберіганню предметів.

Реалізація: Скрипт *Item* зберігає унікальний конфіг для кожного предмету з своїми вказаними даними, такими як: індекс, сам предмет та звук підбирання/викидання.

В скрипті *PlayerController* є метод Interaction(), який перевіряє натиск лівої кнопки мишки, якщо це сталося, то з центру камери гравця випускається невидимий промінь за допомогою RayCast. Якщо він на своєму шляху зіткнувся з предметом на шарі “Item”, то система в цього предмета викликає метод з інтерфейсу Take().

Метод в кожного предмету дістає його власний конфіг та викликає в скрипта *Inventory* метод Take(), в якому відбувається перевірка “чи є якийсь предмет в руках”, якщо немає предмету в руках - підбираємо предмет, якщо є предмет в руках - викидаємо цей предмет і підбираємо новий. Алгоритм інвентарю простий - є об’єкт рука, який лежить у гравці, в ньому знаходяться всі предмети, але вони мають статус “неактивні”, коли користувач підбирає предмет, то система активує його по індексу, який записаний в конфігу.
Коли користувач натискає праву кнопку мишки, то запускається метод Drop(), в якому відбувається перевірка чи є предмет в руках, якщо є - система створює копію предмету, переміщає його в пустий об’єкт на сцені “Items” і дає йому силу поштовху, щоб він полетів в перед, ніби його кинули. Після чого об’єкт в руці просто вимикається.

Коли користувач хоче використати предмет, то в тому ж методі Interaction() робиться перевірка, чи є предмет в руках, якщо так - викликається в інтерфейса метод Using() в якому прописана абсолютна вся логіка. Також є скрипт AnyItem, якщо його додати на будь-який об’єкт, то його можна буде підбирати та викидати, але не використовувати - це було зроблено для більшого погруження і збільшення можливостей/свободи гравця.

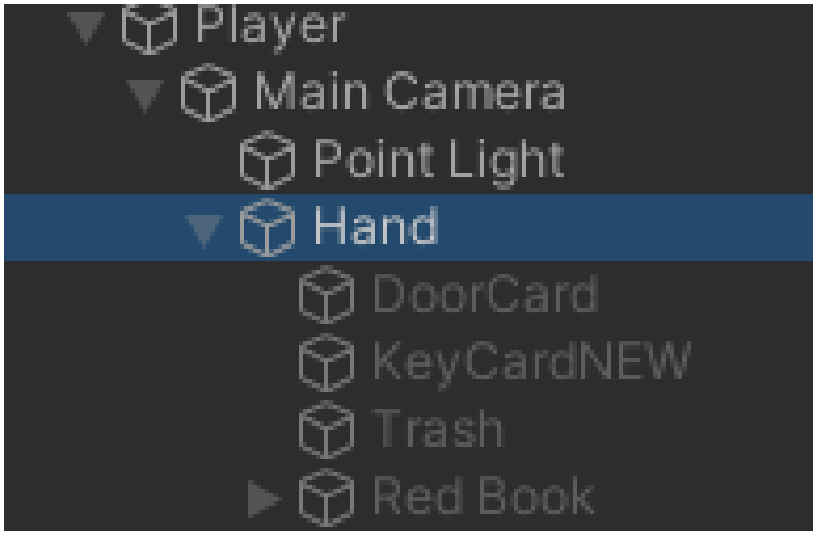

Рис. 3.1. Об’єкт “Рука” в Інспекторі.

3. Перехід між рівнями.

Основне: Всі події в грі відбуваються на одній сцені. Головна задача була зробити непомітний для гравця перехід рівнів. Також ідею хотілося підв’язати під сюжет та стиль гри. Так як події проходять в дуже глибокому бункері логічно було реалізувати ліфт, в якому якраз таки можна було б зробити непомітний перехід на нову локацію.

Принцип дій для проходження рівня:

1. Знайти ключ карту;
2. Застосувати ключ карту на замок важеля;
3. Активувати важіль;
4. Зачекати кінця анімації важеля;
5. Зайти в ліфт;
6. Натиснути кнопку активації ліфту;
7. Зачекати кінця поїздки;

Реалізація: Скрипт *GlobalEventManager* - в ньому знаходяться всі підписки, які тільки існують, клас є статичним, що дає змогу з будь-якої точки коду підписатися чи дати сигнал підписникам про якусь подію. Коли користувач знаходить ключ карту він може підійти до замка на важелі і активувати її - в скрипті KeyCard запуститься метод інтерфейса Using(), в якому даємо знати

підписникам, що замок було знято, також активується анімація його ламання. Важіль стає доступний для використання. Якщо гравець натискає важіль - запускається метод інтерфейсу Use(), який надсилає сигнал SendLeverActivated() всім підписникам OnLeverActivated() - ця подія активує анімацію та звук важеля. Після цих дій, гравцю стає доступна кнопка в ліфті, якщо її активувати, то подається сигнал всім підписникам OnButtonElevatorActivated().

В скрипті *ElevatorController*, запускається курутина під назвою StartElevatorCoroutine, в яку передається рівень, який має бути наступним. Всі дані, які використовуються в курутині, такі як: час подорожі, індекс рівня та об'єкт "рівень" - дістаються з конфігу ліфта. Спочатку переходу рівня деактивуються важіль, кнопка та замок - вони вертаються в свій початковий стан. Далі активується блок, який не дає гравцю вийти з ліфта, поки він їде вгору, після чого перевіряється "чи існує наступний рівень", якщо так до дістаються дані з конфігу. Після цього знищується минулий та створюється новий рівень, блок виключається і дає змогу користувачу потрапити на локацію.

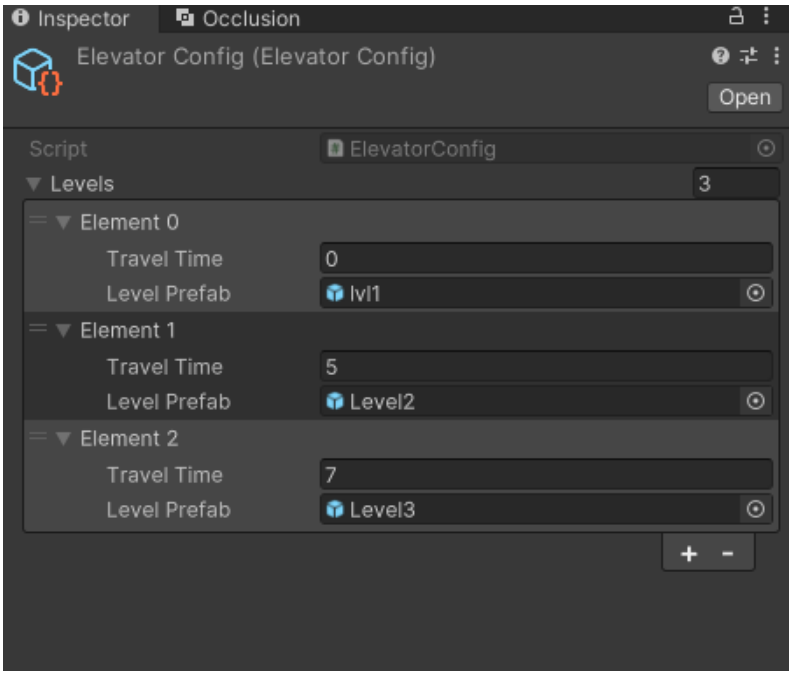

Рис. 3.1. Конфіг з налаштуваннями ліфта.

4. Руйнування об'єктів.

Основне: Для додаткового погруження в ігровий процес потрібно було добавити інтерактивності в гру. Для цього було прийнято рішення добавити руйнування об'єктів - користувач може кілька разів вдарити по предмету і він зламається, після чого з нього можуть випасти різноманітні ресурси.

Реалізація: Для реалізації було використано сторонній компонент *Mesh Exploder*. Як було розписано в *"2. Інвентар"*, існує скрипт *PlayerController,* в якому є метод Interaction(), який відповідає за взаємодію між гравцем та об'єктами на локаціях. Після перевірки чи зіткнувся промінь з об'єктом на шарі "Item", йде перевірка на те, чи був на його шляху об'єкт на шарі "Object" - якщо так, то в нього активується метод DestroyItemObject(), в нього передається напрямок погляду гравця, з інтерфейсу *IDestroyble*. Це все можна дістати через те, що при зіткнені в var item записується об'єкт, з яким відбувся контакт, після чого за допомогою item.GetComponent<назва_скрипта>() можна отримати доступ до всіх public даних.

Метод DestroyItemObject() реалізовано в скрипті *DestroyObject*, який повинен бути доданий на об'єкт, який повинен руйнуватися. При кожному нажатті на об'єкт запускається курутина Kick(), в яку по ланцюжку передається напрямок погляду гравця. В середині неї в предмета віднімається здоров'я і також відбувається фізичний поштовх за допомогою Rigidbody.AddForce. Після виходу з курутини йде перевірка чи здоров'я не стало менше нуля - якщо так, то за допомогою методу EXPLODE() з Mesh Exploder, об'єкт ломається на багато дрібних частин, які мають фізику і через деякий час зникають. Коли його зруйновано, то перевіряється чи випадають з нього рандомні предмети, якщо так - запускається метод SpawnRandomItem(), в якому вибирається і створюється випадкова річ з списку _itemsToSpawn; якщо ні - то запускається метод SpawnAllItems(), в ньому на сцену створюються всі предмети з списку _itemsToSpawn.

Самому об'єкту в Інспекторі можна налаштувати різноманітні параметри:

- itemsToSpawn - список предметів, які можуть випасти після руйнування;
- force - сила, з якою гравець штовхає об’єкт при ударі;
- explosionTime - час, за який уламки пропадуть;
- hp - рівень здоров’я об’єкта;
- canDestroy - це bool змінна, яка відповідає за можливість ламання предмета;
- dropRandomItem - це bool змінна, яка відповідає за те чи випаде випадковий предмет чи всі, які є в списку.

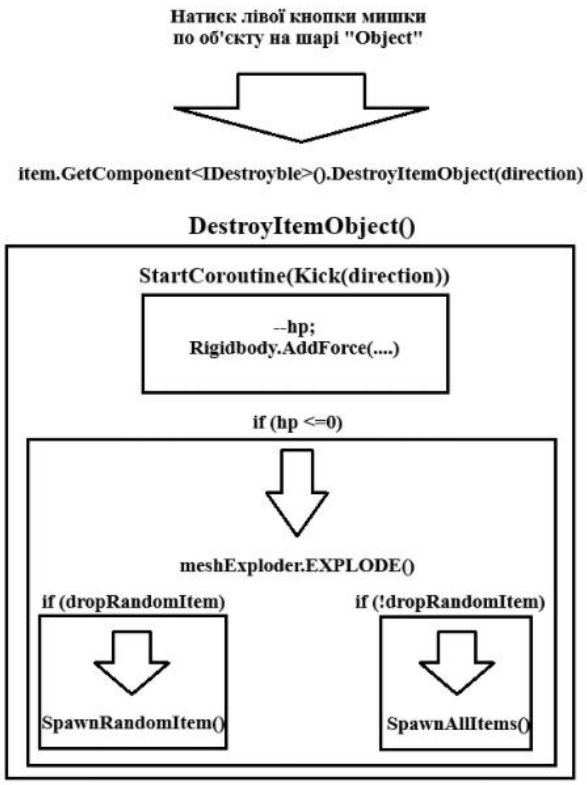

Рис. 3.1. Алгоритм роботи руйнування об’єктів.

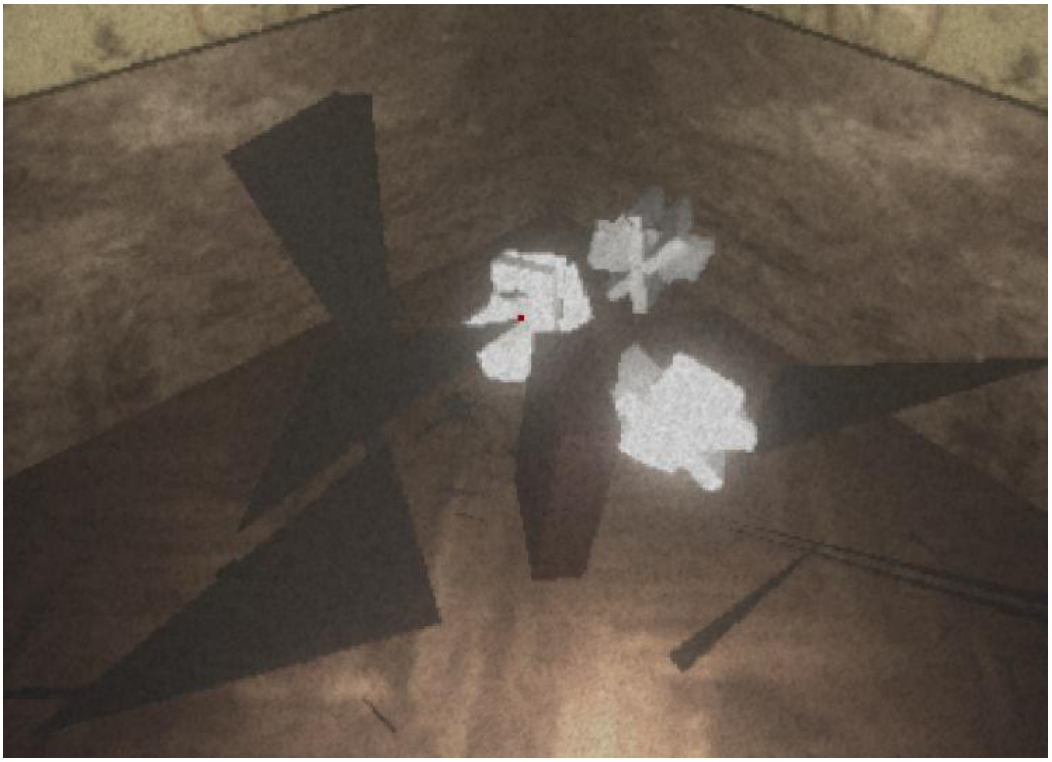

Рис. 3.1. Руйнування об’єктів у грі.

5. Збереження даних.

Основне: Коли гравець проходить якусь частину гри, вона має зберігатися, щоб йому не прийшлося заново це все проходити. Для цього було розроблено доволі просту систему, яка відбувається кожен раз при проходженні одного поверху. Все працює за допомогою вбудованого механізму збереження даних PlayerPrefs.

Реалізація: В скрипті *ElevatorController* система перевіряє нажаття кнопки в ліфті, якщо вона була активована, то через підписку OnButtonElevatorActivated запускається курутина StartElevatorCoroutine(), в яку передається індекс поточного поверху. В її кінці відбувається запуск методу SaveAllData з скрипту DataSave.

Під час розробки, мною було створено скрипт *DataSave*, який має в собі список методів для зберігання та роботи з вже збереженими даними. В ньому знаходяться такі методи:

- SaveLevel(int level) - це метод, який зберігає номер рівня. Для коректної роботи в нього передається поточний індекс рівня, після чого відбувається

перевірка чи існує ключ “Level” в PlayerPrefs (вигляд в коді: if (PlayerPrefs.HasKey("Level"))).

- ІСНУЄ - в статичному скрипті DataLevel, яка зберігає поточний рівень, змінна level починає дорівнювати PlayerPrefs.GetInt(“Level”), тобто в неї записується значення збереженого рівня. Далі відбувається перевірка, чи DataLevel.level менша ніж теперішній рівень. Якщо так - записуємо поточний рівень в PlayerPrefs та DataLevel.level.
- НЕ ІСНУЄ - записуємо 0 в PlayerPrefs та DataLevel.level - це робиться, щоб якщо користувач ще не грав, то запускався 0 рівень.
- SaveInventory() - це метод, який зберігає предмет в руках. Спочатку йде перевірка, яка перевіряє чи існує ключ “Inventory” в PlayerPrefs.
- ІСНУЄ - відбувається перевірка чи в руці є предмет, якщо вона пуста, то видаляємо ключ “Inventory”, якщо ні - зберігаємо в PlayerPrefs в ключ “Inventory” індекс предмету, який зараз активний (Inventory.myItem.Index) і включаємо предмет в руках за допомогою .SetActive(true).
- НЕ ІСНУЄ - просто перевіряє чи є предмет в руці, якщо є, то зберігаємо його в PlayerPrefs в ключ “Inventory”.
- LoadAllData() - це метод, який дістає всі збережені дані з PlayerPrefs. Його алгоритм роботи дуже прости, а саме - якщо існує “Level”, то в DataLevel.level записується значення з цього ключа; якщо не існує, то записується і зберігається 0. Дані про інвентар дістаються після перевірки чи існує ключ “Inventory”, якщо так - то з рук дістається по збереженому індексу в поточному ключі конфіг предмету і записується записується (Inventory.myItem=Inventory.GetItemsInHand()[PlayerPrefs.GetInt("Inventory")].GetComponent<Item>().itemConfig;). Після цього предмет в руках стає активним за допомогою .SetActive(true).
- ClearData() - це метод, який при визові видаляє всі збережені дані і додатково обнуляє DataLevel.level та Inventory.myItem.

- SaveAllData() - це метод, який зберігає зразу всю інформацію, він запускає два інших метода, а саме SaveLevel(int level) та SaveInventory().

6. Звуки.

Основне: Звукова доріжка - це одне з найголовнішого для якнайкращого погруження гравця у атмосферу та світ гри. Для хорошої структура проекта було прийнято рішення - зробити AudioManager, який буде керувати всіма звуками в грі.

Реалізація: В скрипті *AudioManager* є великий список різноманітних аудіо кліпів, такі як: музика, ліфт, удар, поточний предмет, важіль, ходьба - всі вони вказуються в Інспекторі. Під час ініціалізації всі методи в скрипті підписуються на події з GlobalEventManager. Вони всі мають спільні риси - гучність звуку (він працює по формулі: гучність поточного звуку * глобальне налаштування звуку) та висота звуку(робить звук нижчим/повільнішим або високим/швидким).

Приклад коду:

```
private void Play_Звук()
{
   _sfxSource.volume = 0.85f * globalVolumeSFX;
   _sfxSource.pitch = 1f;
   _sfxSource.PlayOneShot(назва_змінної_звуку);
}
```

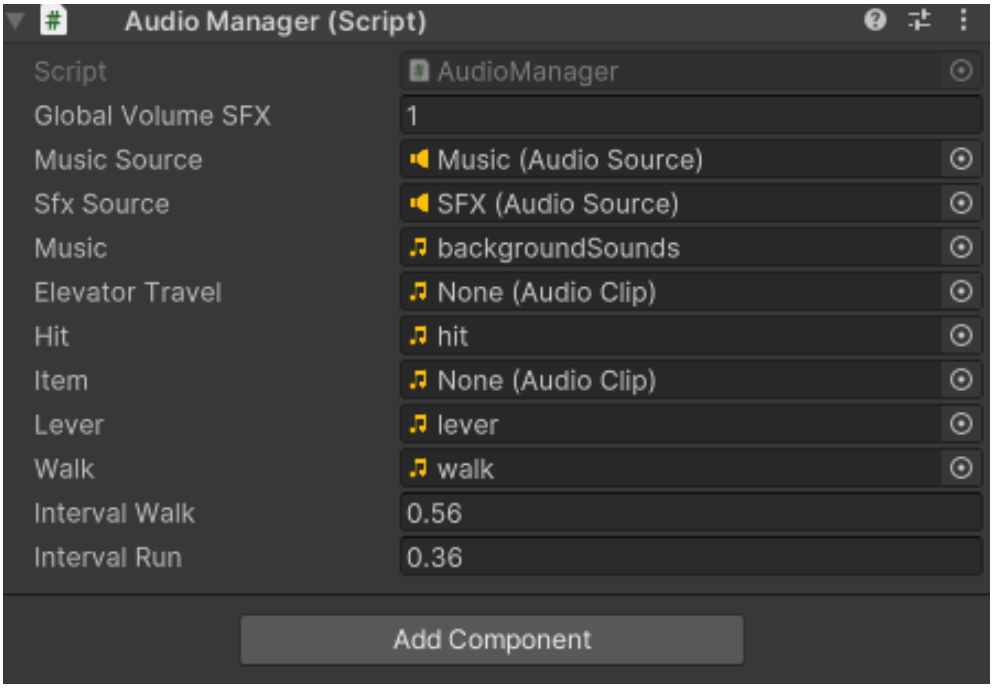

Рис. 3.1. AudioManager в Інспекторі

7. Вороги.

Основне: Для більшої атмосферності та добавлення елементу виживання, несподіванки та тривоги - було прийнято рішення добавити ворогів, які нападають на гравця, якщо його помітили перед собою, або якщо він був занадто шумний. Під час розробки було два варіанти їх реалізації: NavMesh і власний штучний інтелект - було обрано другий.

Реалізація: Кожен ворог має свій власний конфіг з своїми параметрами. Характеристики, які є у ворога:

- Швидкість ходьби/стандартна;
- Швидкість бігу/пришвидшення;
- Швидкість повороту;
- Дальність огляду;
- Модель для скримеру;
- Звук скримера.

В скрипті *EnemyControlling,* в стандартному методі Start() від Unity, ініціалізуються всі параметри і також записується в _target гравець - він

шукається на сцені за допомогою FindGameObjectWithTag(). Існує курутина Checker(), яка створена для виявлення користувача при порушенні низки умов. В самому методі спочатку йде перевірка, **якщо невидимий промінь, який випускає ворог, попадає на гравця, чи дистанція від ворога до героя менша ніж відстань, яку випромінює користувач, то bool змінна _isTarget стає true, і тоді запускається метод AgroMove()**. В ньому визначається напрямок, в яку йде ворог за допомогою формули (_target.position - transform.position).normalized. Після цього через кватерніон визначається куди повинен дивитися ворог, а далі завдяки RigidBody.MoveRotation() він повертається. Визначається також вектор руху, який визначається по простій формулі transform.forward * _runSpeed * Time.fixedDeltaTime і задається сам рух через RigidBody.MovePosition().

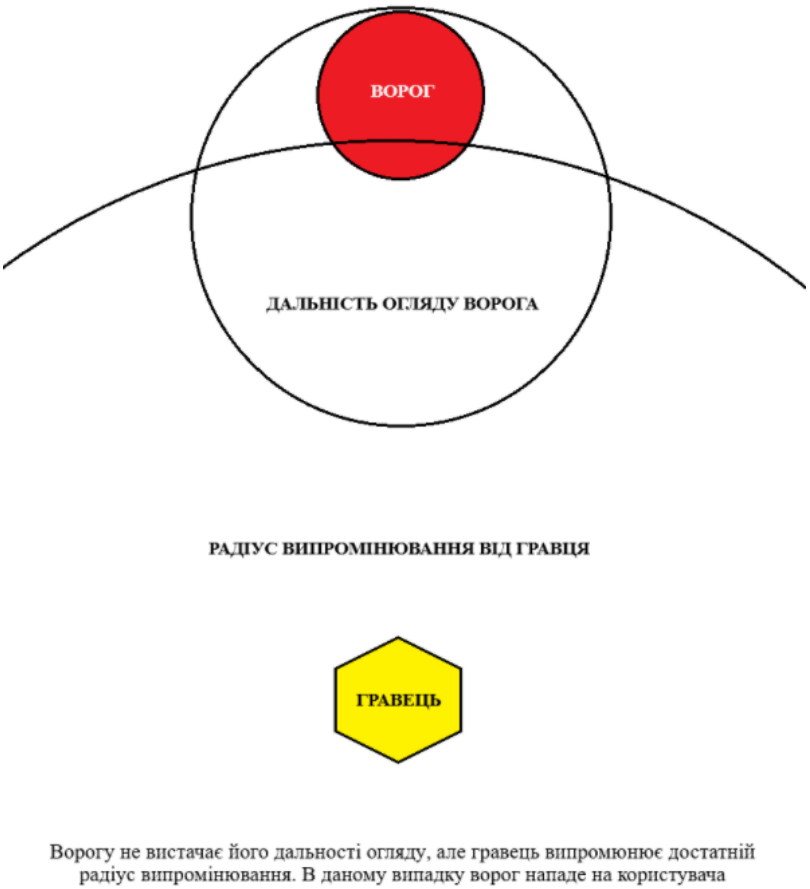

Рис. 3.1. Алгоритм роботи системи ворога.

Модель ворога була повністю зроблена через програму для моделювання Blender. Наразі в грі доступний лише один ворог, який виглядає як хробак з купи

сміття, а як голова використовується годинник. Також, до ворога прив’язане аудіо, при русі - це тяжкий звук металу, а при скримері - це крик, який поєднано з звуком старого настільного будильнику.

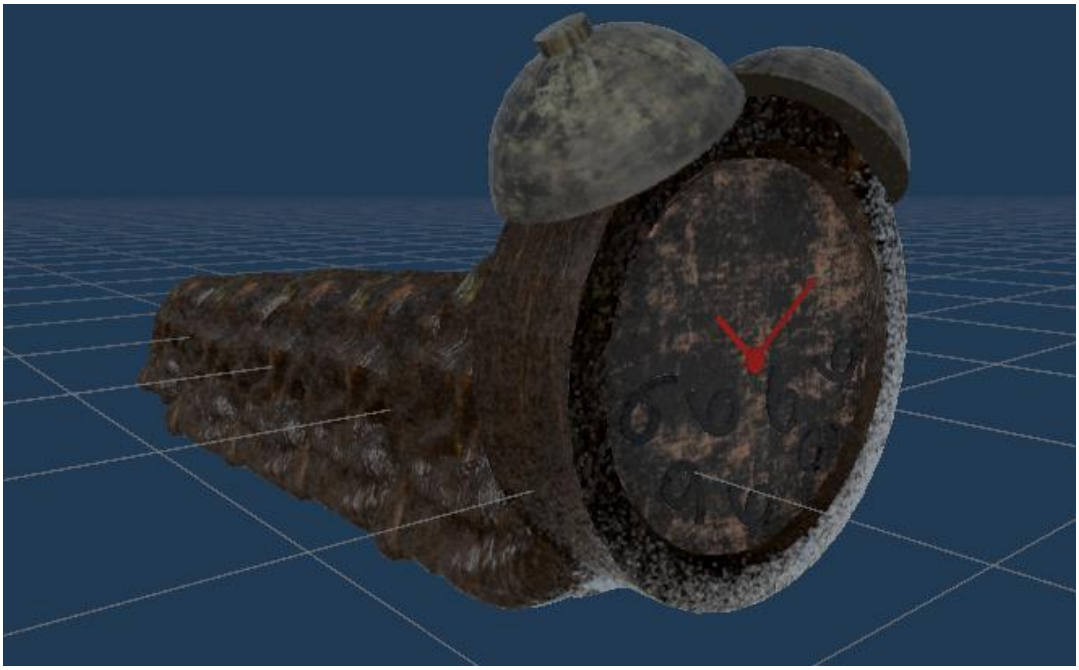

Рис. 3.1. Модель ворога “Object 28”.

8. Скримери

Основне:

Скримери - це різкий візуальний або гучний аудіо ефекти, який виникає перед гравцем. Він використовується як інструмент для досягнення несподіваного страху в користувачів.

У проекті я реалізував скримери як обличчя ворога, яке з’являється на весь екран гравця, має червоні тона і супроводжується гучним і неприємним звуком.

Реалізація:

Скример з'являється при умові, якщо ворог досягнув гравця. Був вибір між двома варіантами реалізації цього, а саме метод OnCollisionEnter(Collision) і вимірювання дистанції між ворогом та користувачем за допомогою RayCast(невидимий промінь). Було обрано другий варіант, адже перший дуже ресурсозатратний через необхідність обробки зіткнень фізики в реальному часі. Натомість перевірка відстані є набагато простішою по нагрузці системи, що

збільшує її продуктивність. В скрипті EnemyControlling є метод Screamer(), який запускається лише при умові, що ворог бачить гравця. В ньому прописана проста логіка, а саме перевірка, чи дистанція між двома менша за 1f. Якщо вона менша, то ворог видаляється та з героя дістається метод Death(), в який передається модель скримеру та його тривалість в секундах. В ньому активується об'єкт _screamerObject через .SetActive(true), в ньому створюється об'єкт скримера і включається його звук завдяки компоненту AudioSource. Також запускається курутина RestartLevel(seconds), яка через передану тривалість визначає, через який час рівень перезапуститься.

Скример супроводжується анімацією трясіння голови ворога, яка була зроблена за допомогою компонента Animation та Animator. Запуск відбувається при його включені за допомогою Start().

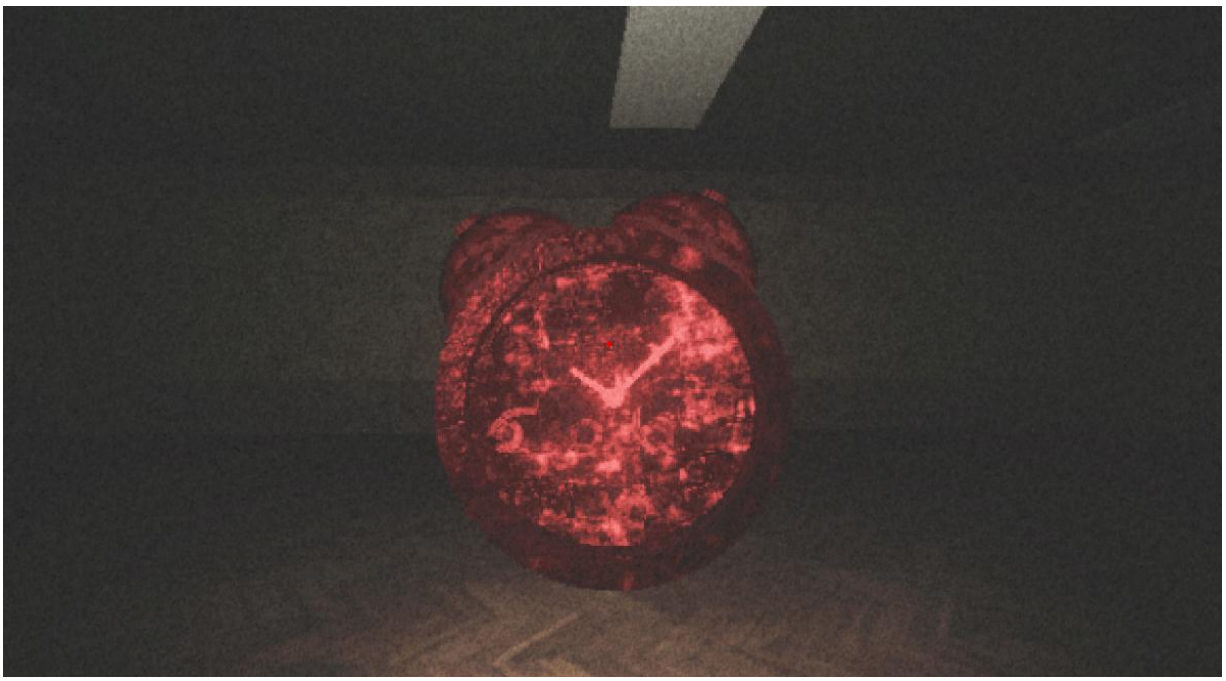

Рис. 3.1. Вигляд скримеру.

3.2. Голосове керування

У грі реалізовано ефективна механіка голосового керування, яка являє собою аналіз звуку з мікрофону в гравця. Він вибирається стандартним, який стоїть в налаштуваннях Windows. Головний компонент, який використовує система - це Light на пустому об'єкті. Змінюється дальність та інтенсивність в *реальному часі залежно від гучності вхідного сигналу у пристрій* - наприклад гучний звук, крик або навіть шепіт починають змінювати яскравість джерела світла. Для роботи з цим механізмом було створено окремий скрипт *LightControll*, який відповідає за зчитування звуку з мікрофона та перетворення звукової доріжки в числове значення.

В скрипті *LightControll* спочатку відбувається ініціалізація, в якій відбувається перевірка "чи існують мікрофони в системі" в змінну записується стандартний мікрофон користувача і починається запис звуку: _micClip = Microphone.Start(_selectedDevice, true, 1, AudioSettings.outputSampleRate). Після цього в стандартному методі Update() відбувається весь основний алгоритм механіки. Спочатку система перевіряє чи ініціалізувався мікрофон і почався запис, якщо так, то система запускається ще одна перевірка, яка перевіряє де знаходиться "курсор" запису, якщо позиція більша за розмір вибірки в 128, то скрипт продовжує роботу і переходить до зчитування аудіосемплів за допомогою рядку _micClip.GetData(_samples, micPos - _sampleSize). Далі відбувається розрахунок середньоквадратичного значення(Root Mean Square) - це метод оцінки середньої амплітуди/гучності сигналу. RMS визначається по формулі:

$$rms = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}, \qquad (3.2.)$$

де

n - кількість семплів(_sampleSize);

x_i - значення амплітуди(_samples[i]).

Інтенсивність випромінюваного світла від гравця визначається по формулі значення RMS множиться на коефіцієнт інтенсивності і обмежується встановленою межею:

$$targetIntensivity = rms * intensivityMultiplier \quad (3.2.)$$

де

rms - це середня гучність;

intensivityMultiplier - це кофіцієнт інтенсивності.

Також є рядок коду, який робить перехід зміни світла плавним за допомогою Mathf.Lerp, яка приймає в собі поточну гучність, інтенсивність світла і швидкість зглажування світла помножена на Time.deltaTime. Після цього світло приймає всі параметри - _light.intensity та _light.range.

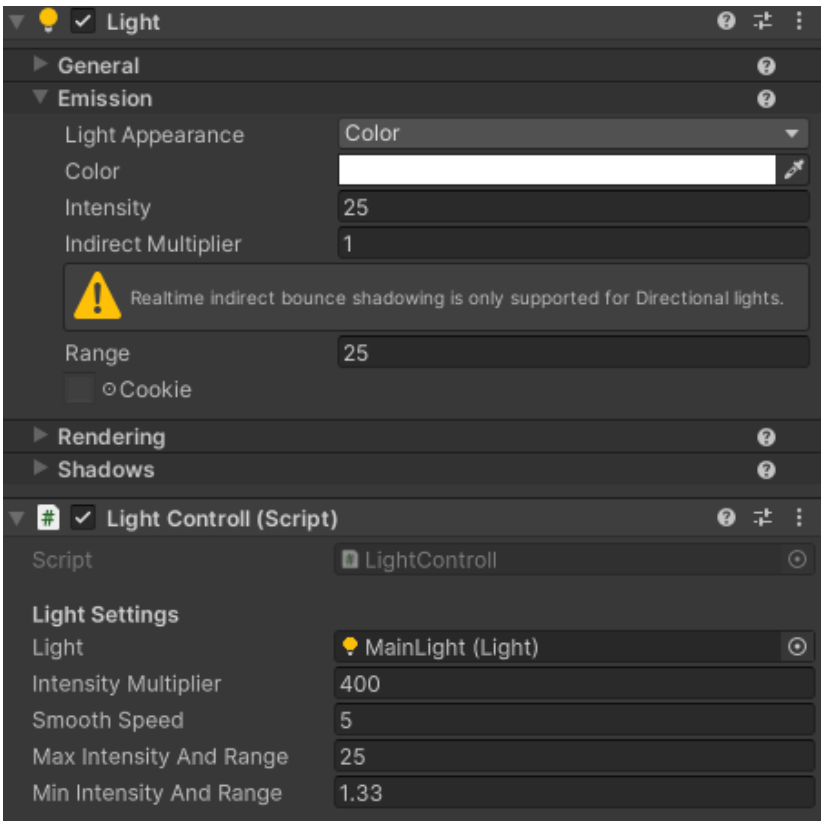

Рис. 3.2. Налаштування світла і LightControll.

3.3. Оптимізація проекту

Під час розробки гри велику увагу було приділено оптимізації, адже це один з ключових пунктів для успіху продукту. Чим більше кадрів в секунду буде

видавати проект, тим більша кількість користувачів можуть в неї пограти, купити та популіризувати. В процесі реалізації було застосовано як і загальні підходи до оптимізації, так і більш індивідуальні рішення.

Як і в будь-якому проекті, в першу чергу було налаштовано static і dynamic batching. Вони значно збільшують глобальну оптимізацію гри.

- Статичний батчінг - об'єднання нерухомий об'єктів на сцені. Такі об'єкти позначаються в Інспекторі як static і Unity виконує сам всі внутрішні операції, які й збільшують кадри в секунду.
- Динамічний батчінг - тимчасово об'єднує рухомі об'єкти з одним тим самим матеріалом. Весь рендер відбувається в одному кадрі, а не по одному, що й збільшує FPS.

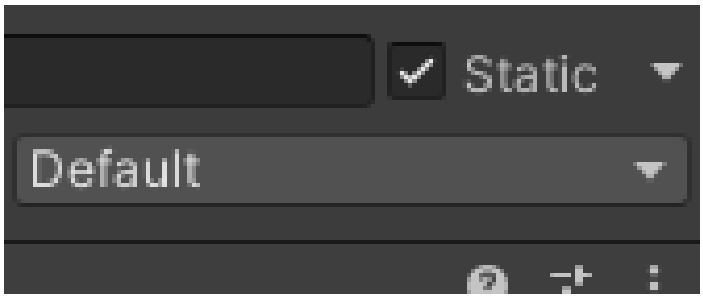

Рис. 3.3. Вмикання статичного батчінгу в Unity.

Обидва ці методи зменшують кількість draw call(кількість спроб рендеру), що зменшує навантаження на відеокарту, що є доволі важливо в цьому проекті, адже всі події відбуваються на одній сцені.

З глобальної сторони також було обмежено кількість викликів методу Update() - викликається кожен кадр, і FixedUpdate() - викликається за фіксовані проміжки часу - вони дуже нагружають систему. Весь код написано по правильній структурі і без використання важких методів, таких як: Find(), FindObjectOfType(), GetComponentInChildren(), Destroy() - але інколи вони все ж таки використовувались. Ще обмежена кількість викликів Physics.Raycast, Rigidbody та різних методів для роботи з колайдерами.

Було використано сторонній компонент з Asset Store, а саме Mesh Combiner, який об’єднує кілька об’єктів в одну сітку. Це було використано в основному на предмети, які гравець може підбирати та ворогів.

Стилістика та задумка гри також дала свої результати, так як гравець не бачить далеко через штучні обмеження від скрипта *LightControll,* максимальна його видимість є 25 метрів. Отож було прийнято рішення, виставити дальність огляду камери до 50 метрів, а кут огляду є 70 градусів, що також дало суттєвий приріст.

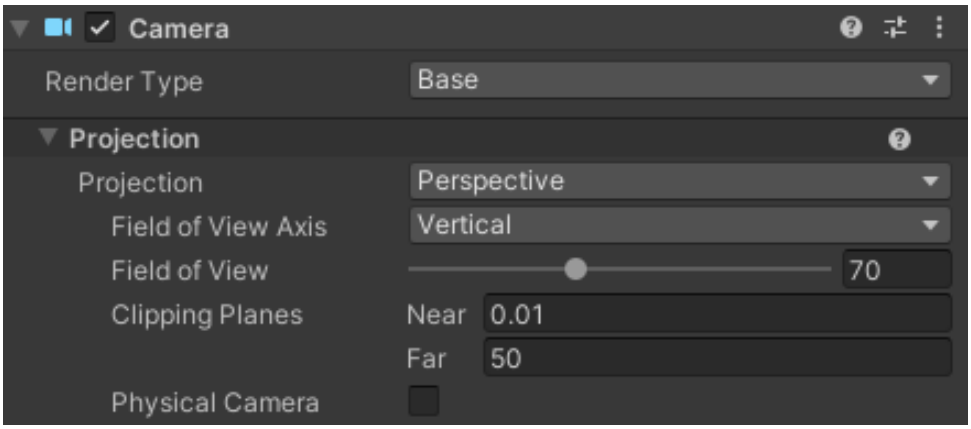

Рис. 3.3. Налаштування камери гравця.

За рахунок стилістичної графіки гри - ретро-графіка - вдалося також збільшити кількість кадрів в секунду. Адже розширення екрану гри є всього 31% і графічному процесору не потрібно витрачати лишні ресурси на обробку деталізації моделей. До цього всього була пророблена робота з освітленням та тінями.

- Освітлення - було виставлено обмежену кількість джерел світла, отже один об’єкт може приймати обмежену кількість променів - всього 5.
- Тіні - максимальна дальність їх промалювання виставлена 15 метрів, а якість розширення залежить від дальності, а саме 512 - якщо відстань не більша за 4.3 метра, 256 - якщо менша ніж 5.3 метра і 128 - якщо більша ніж 5.4 метра. Також тіні є не Soft(мягкими), що також добавляє кадри.

Всі текстури в грі мають маленьке розширення, в середньому це значення 256. Звісно, деякі менші об’єкти мають шкіри по 32, а більші по 512 - і це є максимальне число.

Більшість анімацій було зроблено за допомогою сторонньої бібліотеки DOTween, яка дає змогу робити їх в скриптах. Таким чином система уникає лишнього звернення до доволі важкого компонента Animator.

Аудіо також попали під руку оптимізації. Була зменшена якість самого звуку приблизно до 90%, а самі файли були скопресовані, що зменшує вихідних розмір проекту.

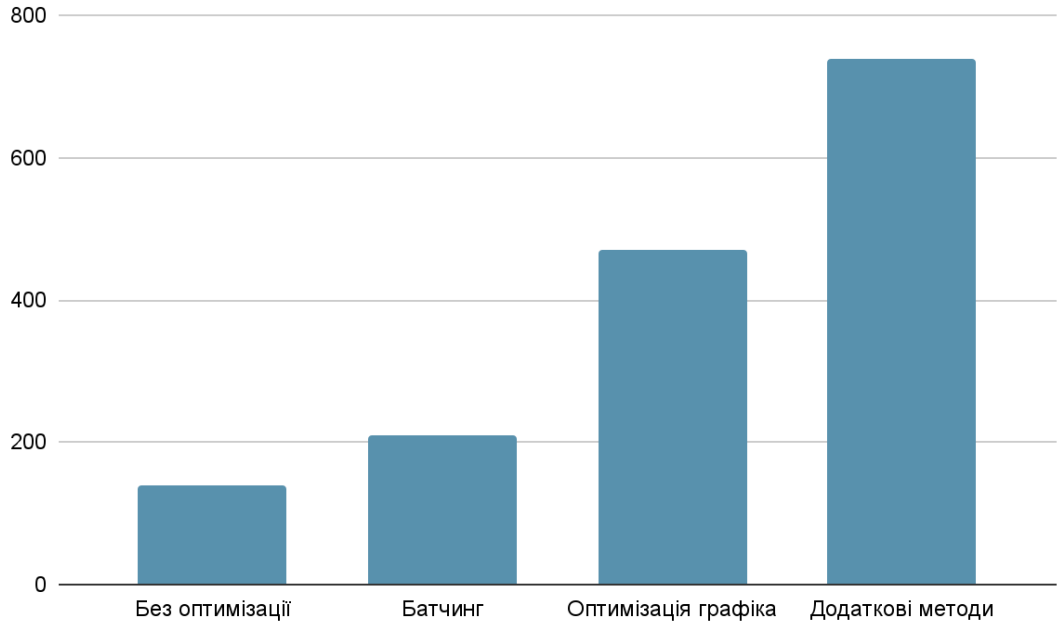

Рис. 3.3. Діаграма етапного зростання FPS(кількість кадрів в секунду) в результаті послідовного застосування методів оптимізації.

ВИСНОВКИ

В даній кваліфікаційній роботі було розроблено хоррор-гру з ретро стилістикою та голосовим керуванням на Unity.

В розробленому продукті було реалізовано рух, систему предметів та інвентарю, руйнування об’єктів, ворогів, скримери, смерть гравця, інтерактивність з навколишнім середовищем, ретро графіку, механіку голосового керування, аудіосистему, систему збереження даних.

Гра являється актуальним методом розваг серед людей. Даний проект має потужний потенціал зацікавити досить широку аудиторію завдяки унікальності ігроладу і головної механіки - голосового керування. Ретро графіка, яка реалізована в грі, викликає в користувача такі ефекти як: тривога, страх, дискомфорт, ефект невідомого. Також вона не потребує велику кількість ресурсів від апаратного забезпечення.

Даний проект являється не тільки як метод розваги, а й слугує прикладом інтеграції нових і унікальних методів взаємодії користувача з грою, що може відкрити нові методи для дослідження у сфері ігрової індустрії.

Отож, розроблена гра демонструє, що у межах інді-розробки можливо реалізувати нестандартні рішення, які можуть надати конкуренції на ринку, та й слугувати прикладом для великих ігрових компаній.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Гра Dead By Daylight в магазині електронних продуктів Steam. *Behaviour Interactive Inc.*. URL: https://store.steampowered.com/app/381210/Dead_by_Daylight/ (дата звернення: 22.02.2025).
2. Гра Outlast в магазині електронних продуктів Steam. *Red Barrels.* URL: https://store.steampowered.com/app/238320/Outlast/ (дата звернення: 22.02.2025).
3. Гра Five Nights at Freddy`s в магазині електронних продуктів Steam. *Scott Cawthon.* URL: https://store.steampowered.com/app/319510/Five_Nights_at_Freddys/ (дата звернення: 22.02.2025).
4. Kvïtyk, Dmytro Joestarovych, Yolo, xal. Рецензії до гри Dead By Daylight. *Steam.* URL: https://steamcommunity.com/app/381210/reviews/?browsefilter=toprated&snr=1_5_100010_ (дата звернення: 23.02.2025).
5. Nachtigall, пінка, Max Gerald, Punk. Рецензії до гри Outlast. *Steam.* URL: https://steamcommunity.com/app/238320/reviews/?browsefilter=toprated&snr=1_5_100010_ (дата звернення: 23.02.2025).
6. F.i.x.E, Korivka, risuyunatvoixgubax, Zavzdy Eshkere. Рецензії до гри Five Nights at Freddy`s. *Steam.* URL: https://steamcommunity.com/app/319510/reviews/?browsefilter=toprated&snr=1_5_100010_ (дата звернення: 23.02.2025).
7. Skinned Alive. Взято зображення. URL: https://rawg.io/games/skinned-alive (дата звернення 24.02.2025).
8. Steve Summers. Review - Rewind or Die. Взято зображення. URL: https://www.relyonhorror.com/reviews/rewind-or-die-review/ (дата звернення 24.02.2025).
9. My Arcanum. Відео “Огляд гри Dead By Daylight Українською!”. URL: https://www.youtube.com/watch?v=EGMroJHyZj4 (дата звернення: 24.02.2025).

10. Dimon Rampage. Відео “Outlast#1. Я не твоє Поросятко! Хрю. (УкрСуб) #проходженняукраїнською”. URL: https://www.youtube.com/watch?v=ytvETlgcYWw&list=PLtn8Q82-3mRpLRYwBBRbO0D-PtE3VFUmp (дата звернення: 24.02.2025).

11. Nika Karuso. Відео “ВПЕРШЕ граю у FIVE NIGHTS AT FREDDY'S 1”. URL: https://www.youtube.com/watch?v=PdB9IR_S6WA (дата звернення: 24.02.2025).

12. Joseph Hocking. UNITY IN ACTION. Manchester: Manning Publications, 2015. 402 pages.

13. Про середовище програмування C#. *FoxmindEd.* URL: https://foxminded.ua/seredovyshche-prohramuvannia-si-sharp/ (дата звернення: 01.03.2025).

14. Що таке GitHub і навіщо він потрібен розробнику. *FoxmindEd.* URL: https://foxminded.ua/shho-take-github-i-navishho-vin-potriben-rozrobniku/ (дата звернення: 01.03.2025).

15. Макс Плєханов. Не соромно запитати: Як працює Blender. 3D-художник Макс Плєханов — про сильні сторони Blender, воркфлоу, гарячі клавіші й про те, з чого почати в моделюванні. *Skvot Mag.* URL: https://skvot.io/uk/blog/how-blender-works (дата звернення: 01.03.2025).

16. Eugenia. Trello: що це таке та як ним користуватися. *HOSTiQ.* URL: https://hostiq.ua/blog/ukr/what-is-trello-2/ (дата звернення: 01.03.2025).

17. Unity Asset Store. *The Best Assets for Game Making | Unity Asset Store*. URL: https://assetstore.unity.com (date of access: 01.03.2025).

18. Mesh Combiner | Modeling | Unity Asset Store. *The Best Assets for Game Making | Unity Asset Store*. URL: https://assetstore.unity.com/packages/tools/modeling/mesh-combiner-157192 (date of access: 01.03.2025).

19. Quick Outline | Particles/Effects | Unity Asset Store. *The Best Assets for Game Making | Unity Asset Store*. URL:

https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488 (date of access: 01.03.2025).

20. MBS - Modular Building System | Level Design | Unity Asset Store. *The Best Assets for Game Making | Unity Asset Store*. URL: https://assetstore.unity.com/packages/tools/level-design/mbs-modular-building-system-208505 (date of access: 01.03.2025).

21. Mesh Exploder | Particles/Effects | Unity Asset Store. *The Best Assets for Game Making | Unity Asset Store*. URL: https://assetstore.unity.com/packages/tools/particles-effects/mesh-exploder-307984 (date of access: 01.03.2025).

22. DOTWeen | Libraries | Unity Asset Store. *The Best Assets for Game Making | Unity Asset Store*. URL: https://assetstore.unity.com/packages/p/dotween-hotween-v2-27676 (date of access: 01.03.2025).

23. Universal Render Pipeline (URP) | Unity. *Unity*. URL: https://unity.com/features/srp/universal-render-pipeline (date of access: 15.04.2025).