Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Ігровий програмний застосунок у жанрі космічного симулятора. VFX та їх оптимізація, механіки ворогів, UI.
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-6

Черепко Є.Ю.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. Новіков Ю.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Черепко Євгену Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий програмний застосунок у жанрі космічного симулятора. VFX та їх оптимізація, механіки ворогів, UI.
Затверджена наказом по університету від 20.05. 2024р. № 471 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 07.06.2024
3. Вихідні дані до роботи Розробити ігровий застосунок в жанрі космічного симулятора, а саме такі елементи гри: візуальні ефекти, механіки ворогів, інтерфейс користувача. Візуальні ефекти передбачають їх оптимізацію.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, впровадження, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	11.04.2024	*виконано*
2	Створення специфікації ПЗ	15.04.2024	*виконано*
3	Проектування ПЗ	20.04.2024	*виконано*
4	Розробка ПЗ	16.05.2024	*виконано*
5	Тестування ПЗ	23.05.2024	*виконано*
6	Оформлення пояснювальної записки	28.05.2024	*виконано*
7	Підготовка презентації та доповіді	31.05.2024	*виконано*
8	Нормоконтроль, рецензування	02.06.2024	*виконано*
9	Здача роботи у електронний архів	02.06.2024	*виконано*
10	Попередній захист	04.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	07.06.2024	*виконано*

Дата видачі завдання 8 травня 2024р.

Студент (ка)_____ Черепко Є.Ю.
(підпис)

Керівник роботи _____ ст.викл. Новіков Ю.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, стор. 133, рис. 99, табл. 2, джерел 21.

БАГАТОКОРИСТУВАЦЬКИЙ, ВОРОГИ, ГРА, КООПЕРАТИВ ОПТИМІЗАЦІЯ, AI, VFX, UI

Об’єкт розробки – ігровий програмний застосунок в жанрі космічного симулятора.

Мета розробки – створення ігрового програмного застосунку від першої особи, який дозволяє будувати тактику гравцям під час вирішення проблем на космічному кораблі з можливості кооперативній взаємодії гравців.

Метод рішення – середовище розробки Unreal Engine 5, Visual Studio 2022, Photoshop та Blender, мови програмування Blueprint та C++.

У результаті розробки створено ігровий програмний застосунок, котрий має три режими – однокористувацький, розділений екран та онлайн кооператив. Гра дозволяє гравцям будувати стратегію для досягнення найкращого результату, розподіляючи ресурси, будуючи маршрути та цілі, під час вирішення проблем виникаючих на космічному кораблі

MULTIPLAYER, ENEMIES, GAME, COOPERATIVE, OPTIMIZATION, AI, VFX, UI

The object of development is a game software application in the genre of space simulator.

The purpose of the development is to create a first-person gaming software application that allows players to build tactics while solving problems on a spaceship with the possibility of cooperative interaction between players.

The method of solution is the Unreal Engine 5 development environment, Visual Studio 2022, Photoshop and Blender, Blueprint and C++ programming languages.

As a result of the development, a gaming software application was created that has three modes: single-player, split-screen, and online cooperative. The game allows players to build a strategy to achieve the best result by allocating resources, building routes and goals, while solving problems that arise on the spacecraft.

Я, Черепко Євген Юрійович, студент гр. ПЗПІ-20-6, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Ігровий програмний застосунок у жанрі космічного симулятора. VFX та їх оптимізація, механіки ворогів, UI», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 9
1 Аналіз предметної галузі 11
1.1 Аналіз предметної галузі 11
1.2 Виявлення та вирішення проблем 17
1.3 Постановка задачі 17
2 Формування вимог до програмної системи 20
2.1 Основні завдання та функціональні вимоги 20
2.2 Інструменти розробки та сторонні ресурси 23
2.3 Рекомендоване технічнічне обладнання та програмні характеристики 24
3 Архітектура та проектування 25
3.1 Загальні положення 25
3.2 Архітектура ворогів та проектування AI 25
3.3 Проектування VFX 29
3.4 Проектування UI / UX 33
4 Опис прийнятих програмних рішень 41
4.1 Створення VFX 41
4.2 Створення ворогів 51
4.3 Створення UI 61
5 Тестування програмного забезпечення 68
5.1 Розробка мапи думок тестування 68
5.2 Розробка тестових випадків 69
6 Впровадження програмного забезпечення 74
Висновки 75
Перелік джерел посилання 76
Додаток А Звіт результатів перевірки на унікальність тексту в базі хнуре 79
Додаток Б Слайди презентації 80
Додаток В Геймдизайн-документ 89
Додаток Г Тест-план 106

Додаток Д Приклад програмного коду Header BPC_HitTrace..... 126
Додаток Е Тези доповіді для науково-практичної інтернет-конференції..... 129

СКОРОЧЕННЯ

P2P – Peer to Peer

NPC – Non Player Character

ВСТУП

Останніми роками ринок розваг безупинно збільшується, що створює попит на різні типи медіа, серед яких вагоме місце займає відеоігрова індустрія. Її кореням налічує майже сотня років, починаючи з 1950 років коли були створені перші найпростіші ігри типажу «Хрестики-нулики». Надалі ігри почали тісно розвиватися разом з технологіями – з'явилися ігрові автомати, які стали підґрунтям золотого віку відеоігор, та консолі, які відкрили простір для домашнього геймінгу [1]. Зрештою індустрія зсунулася у бік консолей, ПК та мобільних пристроїв та налічує прибутки у декілька сотень мільярдів доларів щорічно [2].

Особливу нішу займають багатокористувацькі ігри, які приваблюють значно більшу кількість гравців порівняно з іншими типами ігор, що в свою чергу збільшує і дохід. Взаємодія з користувачами, змагання з іншими гравцями та/або спільне проведення часу – основні фактори які привертають увагу до багатокористувацьких ігор [3]. Також, якщо грає передбачає командну взаємодію це теж сприяє її просуванню. Наприклад, якщо гра має можливість кооперації в команді на 4 людини, то вірогідно що людина спробує залучити своїх знайомих до такої гри. І звісно такі ігри створюють спільноту навколо себе, адже вони зазвичай довше тримають свою аудиторію (ігри які давно не оновлюються, але мають увімкнуті сервера тримають більшу щоденну кількість користувачів) та часто будують цілі історії, які породжуються аудиторією [4].

Рух індустрії розваг, актуальність нового багатокористувацького досвіду та залученість гравців у нього були двигуном для створення унікального ігрового проекту. Темою комплексної кваліфікаційної роботи є ігровий програмний застосунок в жанрі космічного симулятора. Проект передбачає гру від першої особи, розділену на декілька рівнів з короткими перервами, де гравець власноруч, або в команді намагається вижити на космічному кораблі вирішуючи незгоди, які трапляються. Метою є створення захоплюючого досвіду, де гравець повинен побудувати власну стратегію та взаємодію з іншими гравцями задля досягнення найкращого результату.

Завдання передбачає створення демо версії ігрового програмного забезпечення. Індивідуальна частина комплексної роботи базується на створенні візуальних ефектів для гри і їх оптимізації, реалізації механік ворогів та їх поведінки (AI), а також проектуванні та створенні інтерфейсу користувача. Крім того робота передбачає створення та взаємодію з товариством навколо розроблюваної гри, як спосіб пошуку нових ідей та покращень під час розробки, а також як прецедент просування гри.

Набутий досвід під час повноцінної розробки гри та роботи з різними її аспектами може бути застосований для створення інших ігор у різних жанрах. Знання використаного ігрового рушія, отримані під час роботи над проектом, будуть корисними для розробки майбутніх ігор на цьому двигуні. Крім того, результати роботи можуть бути використані для створення демо-версії гри, яка може бути представлена потенційним інвесторам або видавцям. Додаткова доробка гри дозволить розмістити продукт на будь-якій платформі розповсюдження цифрових ігор, таких як Steam або EGS.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

У мерехтливому світлі вогню печер наших предків співпраця була не вибором, а необхідністю. Полювання на мамонтів вимагало командної роботи, виховання дітей потребувало селища, а виживання в суворих реаліях природи виховувало глибоке почуття спільноти. Цей первісний потяг до співпраці вплетений у саму тканину людства.

Тим не менш цей потяг знайшов сучасний вихід у сплеску кооперативних ігор. Тут перемога – це не самотній тріумф, а спільне святкування. Ми даємо п'ять товаришам по команді після знешкодження віртуальної бомби або радіємо в унісон, коли колективно будуємо чудове цифрове місто. Але чому так зростає попит на ці спільні досвіди?

Можливо, це контрапункт нашому дедалі більш роздробленому суспільству. Соціальні мережі пов'язують нас, але часто поверхнево. Ми прагнемо глибокого задоволення від роботи над досягненням спільної мети, від відчуття себе важливим гвинтиком у добре змащеній машині. Кооперативні ігри пропонують безпечний простір для розробки стратегій, вирішення проблем і спільних святкувань. Вони сприяють комунікації, емпатії та радості від спільних досягнень.

Дослідження показують, що кооперативні ігри можуть покращити навички командної роботи і навіть стимулювати креативність. У світі, який вимагає дедалі більшої співпраці, це цінні висновки. Крім того, у святкові дні дедалі більше людей передають перевагу кооперативним іграм, порівняно з настільними іграми та відпочинком на природі [5].

Для ігрових студій та ентузіастів це відкриває простір для дослідження можливостей кооперативних ігор, адже індустрія та технологія постійно модернізуються, а гравці шукають нового досвіду.

Метою цього дослідження є розробка гри у жанрі симулятора, де гравці виступають членом екіпажу зоряного судна. Це дозволить створити унікальний досвід гарного та спільного проведення часу, та зможе розвити навички гравців при

побудові стратегії та прийнятті рішень в кооперативному чи однокористувацьому середовищі.

1.1.1 Вибір моделі багатокористувацького кооперативу

Важливою частиною розробки будь-якого онлайн кооперативу є вибір моделі, або типу поширення та взаємодії гравців в багатокористувацькому середовищі. Це є монументальним рішенням, адже від нього буде залежати, як функціональні вимоги та архітектура, так і досвід кінцевий досвід користувача.

Умовно можна виокремити чотири основні типи кооперативних багатокористувацьких ігор: ігри сервіси, кооператив розміщений гравцями, розділений екран та офлайн мультиплеєр.

Ігри сервіси, або ігри як сервіси – представляють собою ігри з моделлю постійного доходу, подібно до моделі поширення програм SaaS. Ігри як сервіс це спосіб монетизації ігор після їхнього релізу, або підтримки безкоштовної моделі розповсюдження. Зазвичай вони мають довгостроковий потік монетизованого контенту з часом [6]. Відмінними рисами кооперативних ігор є окремий хаб де формуються команди, обирається режими гри та спорядження, витрачаються ігрові валюти. Відповідно, такі ігри мають окремі виділені сервера де перевіряється і виконується ігрова логіка та зберігаються дані користувачів, що викликає потребу в постійного інтернет підключення. Основними типами монетизації є підписка, продаж ігровий контенту та hard валюти [7], платні доповнення та/або платна базова гра. Це робить їх більш прибутковими, але і вимагає високого досвіду при розробці та оптимізації Net коду, а також обслуговування серверів, або оплати хмарних рішень під час підтримки продукту.

Серед останніх прикладів кооперативних ігор цієї моделі розповсюдження можна навести Helldivers 2. Тут гравці об’єднуються в команди до 4 осіб щоб звільнити планети навколишньої галактики від рас, які вторгаються. Кожна вилазка змушує вдумливо обирати спорядження відповідним членам команди, розробляти стратегії підходу для кожної місії та досягати разом зазначені цілі. Її модель

монетизації включає придбання базової гри, та наявність hard валюти, яку також можна отримати в помірних значенням шляхом грінду.

Кооператив розміщений гравцями являє собою ігри де один, або декілька з гравців виступають хостом для запущеної сесії. Ця характеристика притаманна для P2P та слухаючого типу сервера. З цього виходить що цей підхід більш економічний, адже не потребує додаткового виділення обладнання з боку розробників. Часто цей тип кооперативу йде в доповнення звичному однокористувацькому офлайн режиму. Крім того ці ігри зазвичай передбачають платну базову копію гри, яку повинен мати кожен гравець.

Прикладом такого підходу є хіт минулого року Baldurs Date 3. Це гра жанру CRPG де кожен кравець створює свого персонажа унікального класу, раси та інших характеристик об’єднуючись у партії. Baldurs Date 3 заснована на правилах п'ятої редакції популярної настільної гри Dungen & Dragons, комплексна складова якої дозволяє різними способами комбінувати риси та здібностей персонажів щоб одолати ворогів, проникнути у якесь місце чи заговорити NPC. Гра базується на такому типі монетизації, коли купив один раз, а граєш досхочу.

Ігри з розділений екраном передбачають розділення екрану на частини, які зображується ігрове середовище відповідні кожному гравцю. Це локальні ігри які не потребують додаткових копій гри, але вимагаються декілька пристроїв вводу, таких як геймпад чи клавіатура, хоча інколи і дають змогу використовувати один пристрій для пари людей. Тому цей підхід кооперативу гарно підходить для консолей, які популярні для «гри на одному дивані». Важливо розуміти, що цей підхід не вимагає з'єднання з віддаленими клієнтами, тому затримок в ігровому процесі просто немає. Вагомим недоліком ігор з розділеним екраном є їхня значне навантаження на обладнання, адже потребує рендеру середовища для кожного гравця окремо.

Baldur’s Gate 3 також є прикладом гри з розділеним екраном на дві людини. Більш того вона дозволяє поєднувати розділений екран з онлайн кооперативом. Один екран дозволяє спостерігати за проходженням гравців одночасно, що з однієї сторони дозволяє їм не прогавати цікаві ігрові частини, адже гравці не прив'язані

один до одного, але з іншого боку немає можливості сховати якусь інформацію, щоб перехетрити іншого гравця чи персонально провести час з улюбленим компаньйоном. Baldur’s Gate 3 мала проблеми з оптимізацією кооперативу під Xbox Series X через що був затриманий реліз на декілька місяців, у якому з рештою вирізали цей режим на консолі.

Асинхронний мультиплеєр не потребує користувачам грати в один і той же час [8]. Цей підхід охоплює різні жанри ігор включаючи покрокові, де гравець чекає на відстрочений хід іншого, ігри де гравець може провокувати деякі події, які не потребують залученості іншого гравця, наприклад напад на базу, а також ігри де гравець бачить події які сталися у світі інших гравців і можливо може з ними взаємодіяти (з подіями). Цей підхід широко застосовується в мобільних іграх, але інколи зустрічається і в сюжетних іграх на ПК та консолях, адже при належному використанні може додати глибину механік та історії світу за рахунок непрямої взаємодії гравців. Асинхронний мультиплеєр потребують окремі сервера, які оброблять події сгенеровані користувачами.

Незважаючи на те, що цей тип мильтиплеєрних ігор не дозволяє координувати стратегії гравців в реальному часі, він може мати своє застосування. Наприклад, в Death Stranding, грі японського геймдизайнера, головний герой Сем кур'єр, який намагається об’єднати частини розгалуженої Америки долаючи складні рельєфи природи на своїх двоїх. Гаджети які будують та розташовують гравці у своєму світі з'являються і в світах інших гравців, включаючи друзів, з якими ви з'єднатися. Це створює особливу атмосферу єднання, адже ти допомагаєш не тільки собі, а й іншим. Крім того, гаджети можуть отримувати вподобайки, які дозволяють отримати зворотній зв'язок та збільшити шанси на появи гаджету в інших гравців.

Також можна було виділити і п’ятий тип реалізації кооперативних ігор – спільна гра на одному пристрої на одній мапі. Але такий підхід вимагає маленькі розміри мап ігрового світу, або вид камери згори, який би суперечив першочерговій ідеї гри від першої особи на космічному кораблі.

Більшість з перелічених типів кооперативних ігор можна вільно комбінувати, створюючи унікальний досвід для гравців. Єдиним винятком є інтеграція розділеного екрану в сервісні ігри, адже вони передбачають чітке пов’язання акаунту гравця з придбаною копією гри, тому взаємодія двох чи більше гравців з одного пристрою просто буде не доцільною.

При оборі підходу до мультиплеєрної складової гри потрібно зважити основні фактори при розробці, такі як ціна підтримки та складність, її ігрові можливості у сфері обраної теми та попит на ринку кооперативних ігор. В таблиці 1.1 наведено порівняння різних підходів створення багатокористувацького кооперативу.

Таблиця 1.1 – Порівняння типів кооперативу

Тип кооперативу	Вартість підтримки	Складність реалізації	Ігрові можливості та доречність темі	Попит
Ігри сервіси	Висока	Дуже складний	Високий	Високий
Кооператив розміщений гравцями	Немає	Середній	Високий	Високий
Розділений екран	Немає	Простий	Високий	Низький
Асинхронний мультиплеєр	Середня	Складний	Низький	Середній

Найкращим рішення можна цілком вважати локальний та онлайн кооператив розміщений гравцями, адже він приваблює високу кількість гравців за помірну складність реалізації, при цьому маючи широкий простір можливостей для взаємодії гравців в командному середовищі. Ігри сервіси хоч і доволі популярні у нас час, але вимагають висококваліфікованих розробників та відповідний бюджет

проекту. Розділений екран є непоганим рішенням враховуючи його простоту та відсутність додаткових витрат, тому його можна інтегрувати в програмний продукт. Асинхронний мультиплеєр є теж непоганим варіантом, але через наявність витрат на сервери його було вирішено його оминути, хоча можливість його подальшого впровадження після створення демо цілком можлива.

Таким чином провівши аналіз можливих варіантів створення багатокористувацьких рішень було обрано кооператив розміщений гравцями та розділений екран, як два популярні способи реалізації кооперативних ігор від першої особи для інді команди.

1.1.2 Обґрунтування вибору ігрового рушія

У ролі ігрового рушія виступить Unreal Engine 5.3. Цей двигун є одним з найпотужніших готових рішень які є на даний момент. Використання мови C++ дозволяє оптимально працювати з 3D графікою, яка була обрана для цього проекту. Наявність візуальної мови програмування Blueprint, дозволяє швидко створювати прототипи та мати доступ повний доступ до інструментів двигуна, що дозволяє у зручній формі інтегрувати її з візуальними ефектами чи анімаціями. Також Unreal Engine з самого початку був основою для створення онлайн шутерів, тому на поточний час він має повноцінну систему для реплікації даних багатокористувацьких застосунків прикладом якого також є розроблювана гра. Широка спільнота та багата документація поліпшить вивчення архітектури та роботи рушія, а також дозволить використати готові розширення

Unreal Engine є 5.3 є останньої стабільною версією двигуна, зокрема вибір саме п'ятої версії є важливим через наявність широкого спектру інструментарію, для роботи з анімаціями, звуком, VFX та іншого. Зокрема п'ята версія відома системи глобального освітлення Lumen, яка дозволить створити реалістичне освітлення на закритому просторі космічного корабля, та віртуальної візуалізації Nanite, яка здатні оброблювати мільйони трикутників ігрового середовища [9], що дозволить використати широкий спектр 3D моделей для побудови середовища гри.

1.2 Виявлення та вирішення проблем

Головною з проблем які вирішуватиме застосунок є задоволення попиту на кооперативні проекти та розваги в цілому. З аналізу предметної області було виявлено, що зараз люди прагнуть все більше проводити відпочинок разом з сім'єю в кооперативному середовищі. Гра дозволить весело провести час з друзями, спільно вирішуючи задачі та об'єднуючи сили в розважальній формі.

Незважаючи на розважальний характер більшості ігор в цілому, вони дозволяють покращити навички людей, шляхом виявлення цілей та побудови тактики та підходів під різноманітні обставини. Розвиток критичного мислення на швидкого прийняття рішень є підставою для створення ігор зі швидким темпом гри. Розроблювана гра дозволить об'єнати ці два фактори, за рахунок геймплею від першої особи де розумне управління ресурсами та побудова стратегії буде сприяти кращому результату гравця у кінці.

Таким чином ця гра має широкий спектр застосування як форма розваги, чи спосіб підняття здібностей гравця. Це дозволяє охопити ще більшу аудиторію, враховуючи її орієнтованість на декілька гравців загалом.

1.3 Постановка задачі

Задачею кваліфікаційної роботи є створення ігрового застосунку, який привабить широку аудиторію різного віку та національності, шляхом інтеграції багатокористувацької кооперативної складової та швидкого геймплею де важливо аналізувати середовище, щоб приймати критичні рішення.

Гра розрахована на камеру від першої особи, для більшого занурення в ігровий досвід. Ігрове середовище буде розташоване на космічному кораблі, який складається з ряду важливих модулів, кожен з яких відповідає за певну його функціональність, будь-то напруга, камери чи кисень.

Під час партії розділеної на рівні, гравці будуть стикатися з різними загрозами, такими як обстріл корабля чи вторгнення прибульців. Це в свою чергу

буде породжувати різноманітні події, як пробоїни що позбавляють кисню, вогонь чи зупинка корабля через несправність двигуна.

Щоб завершити партію гравці можуть об'єднуватися в групи до 4 осіб, щоб нівелювати загрози, шляхом вдумливого використання ресурсів, таких як зброя, інструменти починки та розхідні предмети. Гравці можуть будувати маршрути через комплексну структуру корабля для ефективного вирішення задач та створювати переваги під час захисту тих чи інших життєво важливих систем корабля.

За швидкість завершення партії, кількості витрачених ресурсів та цілісності корабля буде нараховувати оцінка, яка буде показником успішної взаємодії команди та якості прийнятих рішень.

Повністю ознайомитися з концепцією гри можна за допомоги дизайн документу наведеного у додатку В.

1.3.1 Цільова аудиторія

Основна цільова аудиторія це люди віком від 12 до 44 років. Як показує статистика, відсоток людей серед британців, які полюблять кооперативні ігри на мобільних пристроях та консолях, тримається на рівні 50% до віку 44 років [10]. Відсутність же насильницького вмісту у грі тільки сприятиме залученню молодих гравців 13-15 років, які є найбільшим зацікавленим сегментом цього жанру ігор.

Ігровий програмний продукт в першу чергу привабить командних гравців, які полюбляють проводити час з рідними та друзями. Люди налаштовані на швидких геймплей з коротким часом на прийняття рішень можуть зацікавитися продуктом у тому числі.

1.3.2 Час ігрової сесії

Гра розрахована на одну сесію, яка буде розділена на декілька проміжків по 5-10 хв. З рівнями буде збільшуватися складність, тому час буде збільшуватися поступово, але в тому числі залежати і від тактики гравця та його навичок. Між рівнями буде короткий час біля хвилини на перепочинок та оновлення

спорядження. Враховуючи довжину гри у 3 рівня, час повної сесії буде біля 25 хвилин.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Основні завдання та функціональні вимоги

2.1.1 Завдання та вимоги до VFX

Ігровий застосунок повинна включати наступні VFX:

– ефект вогню при пожежі;
– ефект малого пошкодження корабельного модуля;
– ефект сильного пошкодження корабельного модуля;
– ефект торгівця;
– ефект карбону для вогнегасника;
– ефект починки та використання зварювального апарату;
– ефект починки та використання паяльника;
– ефект отруйної хмари ворога.

VFX повинні бути оптимізовані, тобто споживати оптимальну кількість ресурсів заліза. Для цього повинні бути використані техніки поліпшення шейдерів та систем частинок.

2.1.2 Функціонал ворогів

Передбачено три типи ворогів, по кожному на етап гри. Відповідно на більш пізніх етапах вороги повинні представляти більшу загрозу.

Визначення особливостей ворогів залежить від підібраних асетів. Так було знайдено Кракена, Прибульця та Раптора (див. рис. 2.1). Підбір атак виконувався на базі доступних анімацій.

Кракен це швидке та маленьке сторіння, воно різко атакує та ухиляється від пострілів персонажа. Має пару одиночних та пару комбінованих атак.

Прибулець представляє усередненого ворога з точки зору міці, швидкості та арсеналу. Крім звичайної атаки він може проводити атаку здалеку та масову відштовхуючу атаку поблизу.

Раптор це найсильний ворог, який з’являється з 3 етапу і має наступні атаки:

– підвійний укус поблизу;

– подвійний удар двома крилами поблизу;
– масова крогова атака хвостом;
– отруйний подих, який створює хмару на деякий час, наносячи урон усім поблизу;
– смертельна комбінована атака поодиноких та одночайних атак крилами.

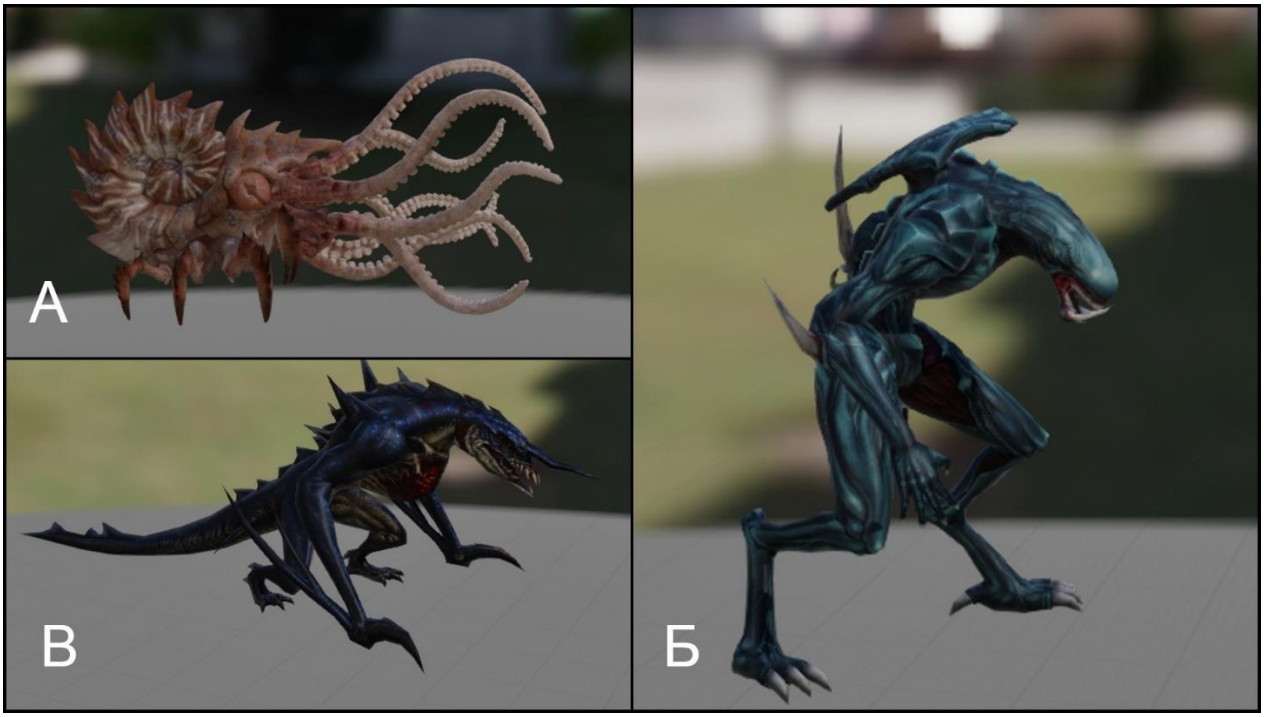


Рисунок 2.1 – Вороги: А – Кракен, Б – Прибулець, В – Раптор

Основний функціонал ворогів включає:

– дослідження навколишнього середовища;
– реакції на такі системи чуття як зір, дотик та слух;
– реакція на персонажів та модулі;
– пошук гравців з аналізом навколишнього середовища;
– переслідування гравців;
– атака та її підбір залежно від таких факторів, як кількість гравців та відстань до них, потужності атаки;
– синхронізація атак великої купи ворогів;

Залежно від розміру ворога він може проходити різні типи дверей, так Раптор може переміщуватися тільки по поверхам між докером, основним відсіком та складом.

Крім того, вороги повинні змішувати свої анімації, наприклад, під час одночасного пересування та атаки, а також мати можливість переривання атак, коли ворог втрачає ціль чи змінює поведінку.

Основною задачею при створенні AI буде налаштування їх реактивності. Це дозволить гравцям відчувати їх більш справжніми, адже ворог буде поводитися по різному залежно від обставин.

2.1.3 Функціональні компоненти UI та вимоги до них

Основні екрани гри це головне меню, його підменю для різних режимів та налаштування графіки, меню паузи та меню магазину. Усі екрани передбачають основну реалізацію взаємодії з ними та їх функціонал, окрім онлайн функцій.

Головне меню включає наступні функціональні компоненти:

- початок однокористувацього режиму з головного меню;
- підменю режиму розділеного екрану з вибором ролей на кожного гравця та початком відповідного режиму;
- підменю для навігації в онлайн режимі, а саме для авторизації, переходу до списку сесій, виходу з акаунту чи створення сесії;
- підменю створення сесії по назві;
- підменю зі списком сесій, до яких можна підключитися;
- підменю вибору ролей кожного гравця в сесії, через яке починається онлайн режим кооперативу;
- підменю налаштувань, яке включає налаштування графіки, роздільної здатності екрану та режиму екрану;
- вихід з головного меню.

Меню паузи виступає додатковим виходу із застосунку під час партії, як до головного меню, так і до робочого екрану. Воно не передбачає активної паузи, а просто слугує для навігації.

Меню закупівлі включає список доступних товарів, які можуть оновлювати між рівнями. Через це меню можна придбати товари, оглянути наявну кількість ігрової валюти та оживити членів команди у разі їхньої гибелі. Повний список товарів включає наступні:

– легкі патрони для пістолетів;
– середньокаліберні патрони;
– патрони великого калібру;
– шприц для відновлення здоров'я;
– балон кисню;
– напівавтоматичний автомат;
– автоматична гвинтівка;
– оживлення одного з сопартійців залежно від ролі, а саме капітан, інженер, механік та офіцер.

До основних вимог належить створення гарного UX та адаптивність то розділеного екрану. Враховуючи що режим роздільного екрану може включати до чотирьох осіб, усі ігрові меню які доступні під час самої партії у режимі розділеного екрану повинні буде налаштовані на чотири варінати: повний екран, два екрани розділених горизонтально, три екрани де один займає нижню половину екрану, а два інших порівну ділять верхню вертикально, чотири екрани розділених пропорційно.

2.2 Інструменти розробки та сторонні ресурси

Основним інструментом розробки виступить рушій Unreal Engine 5.3. Він буде використаний для створення основної ігрової логіки, створення VFX та налаштування анімації. Зокрема для створення частинок буде використана система Niagara.

Visual Studio 2022 виступить редактором коду мови C++ на якій буде реалізована частина логіки.

Для створення 3D моделей та налагоджень сторонніх моделей та текстур задіяний Blender.

Для створення елементів UI буде використовуватися Photoshop.

Більша частина 3D моделей буде використана з офіційного магазину Unreal Engine Marketplace та платформи Sketchfab.

Додаткові моделі та текстури можуть бути узяті з інтернету з урахуванням авторських прав.

Системою контролю версій виступить GitLab.

2.3 Рекомендоване технічнічне обладнання та програмні характеристики

Ігровий програмний застосунок розрахований на операційну систему Windows 10 та 11 64-бітної розрядності.

Мінімальні системні вимоги до заліза:

– процесор: чотири ядра з частотою 2.4 GHz;
– ОЗУ: 8 GB;
– GPU: GeForce GTX 1050;
– пам’ять: 10 GB вільного місця.

Для однокористувацької гри та онлайн кооперативу потребує клавіатуру або мишу.

Для локального кооперативу потребує геймпад на кожного гравця.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ

3.1 Загальні положення

При розробці будь-якого програмного забезпечення важливо робити програму з мінімальною кількістю залежностей та з модульними складовими. Це дозволяє зробити програму більш зрозумілою, зменшити її складностість та облегшити подальшу підтримку. Саме тому задача реалізації будь якої частини роботи полягали в абстрагуванні залежностей та створення модулярності.

Unreal Engine 5 теж дотримується цього принципу та дозволяючи використовувати різні засоби для досягнення кращої підтримки коду. Зокрема актори, які виступають головними об'єктами на мапі, можуть вміщати так звані компоненти. Компоненти – це особливий тип об'єктів, які актори можуть приєднувати до себе як підоб'єкти. Компоненти корисні для спільного використання дій, таких як можливість відображати візуальне представлення, відтворювати звуки [11]. Це дозволяє відокремити такі частини програми як обробка здоров'я, характеристики, система токенів від конкретних акторів, дозволяючи перевикористовувати механіки як у ворогах, так в персонажах і корабельних модулях.

3.2 Архітектура ворогів та проектування AI

3.2.1 Архітектура ворогів

Важливим етапом проектування програмного забезпечення є визначення його функцій та зв'язків. Тому, для ілюстрації взаємодії користувача з ворогами було створено діаграму послідовності (див. рис. 3.1), яка зображує взаємодію ворога, як об'єкта з навколишнім середовищем та гравцем. Актором виступає цільовий гравець, який взаємодії з такими об'єктами як ворог, навколишнє середовище та корабельні об'єкти (модулі, двері, дірки тощо). Взаємодія проходить у певній послідовності, ініціюючись на гравці, підкреслюючи логічні етапи взаємодії та реакції між ними.

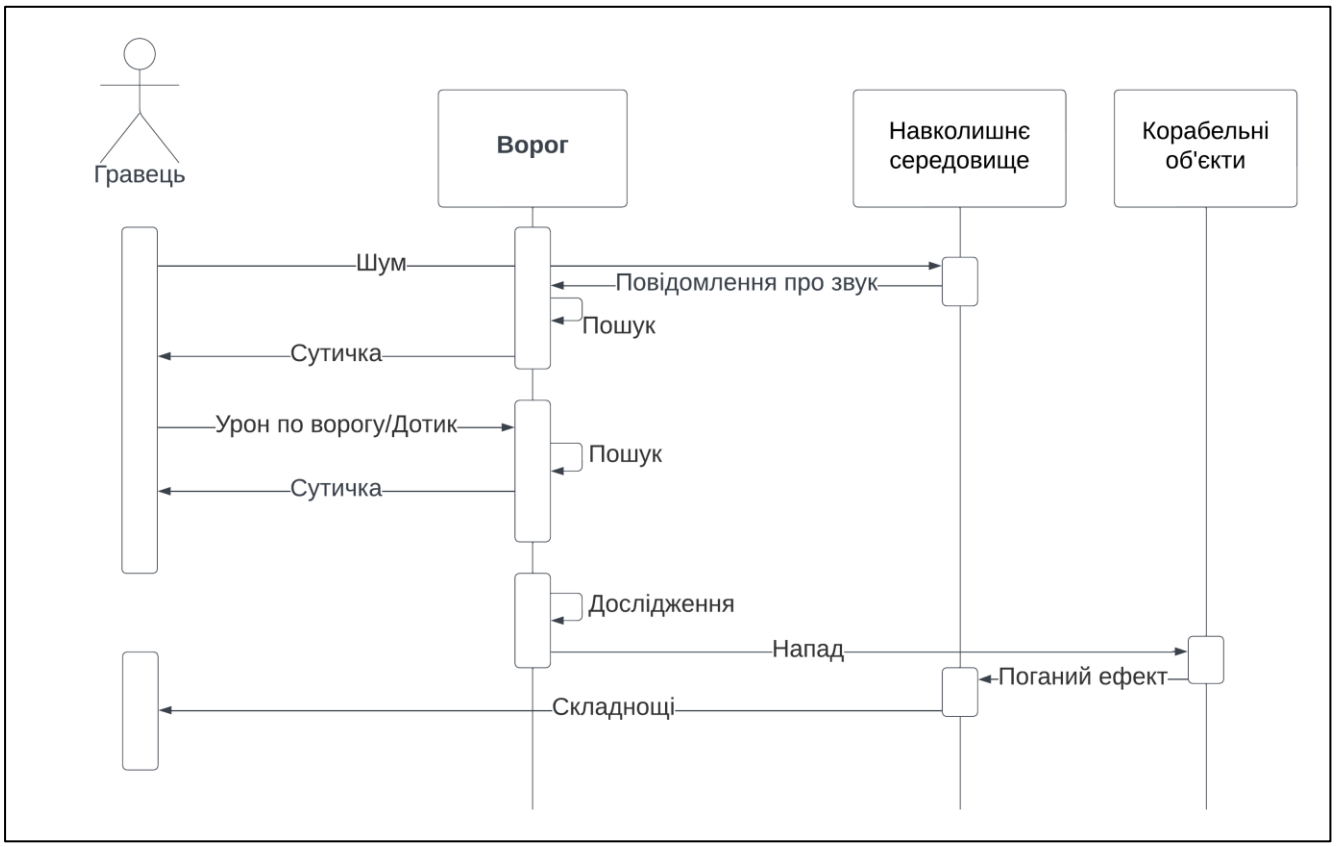


Рисунок 3.1 – Діаграма послідовності взаємодії ворога

Вороги це комплексний об'єкт адже він повинен реагувати, як на персонажів, так і на навколишнє середовище. Для цього вони наділені різними сенсорами, такими як сенсор шуму, пошкодження, дотику. Залежно від цього вони підбирають відповідний стан від якого буде залежати їх поведінка. У випадку сприйняття шуму в певному радіусі ворог спробує потрапити до його джерела. При цьому, враховуючи тип інопланетного ворога, вони можуть сприймати шум по різному. Так, найпростіший ворог Кракен не буде розрізняти типи шумів, у свою чергу більш розвинені форми життя, як Раптор, можуть додатково почати пошук цілі на території біля джерела у разі звуку постріла зі зброї гравців. Таким чином класи певних ворогів, зможуть доповнити чи замінити поведінку відповідним функцій для сприйняття сенсорів.

Доцільно продемонструвати архітектуру через діаграму класів (див. рис. 3.2), адже логіка ворогів розбита на окремі компоненти, відповідаючий за певний окремий функціонал згідно з принципу єдиної відповідальності SOLID.

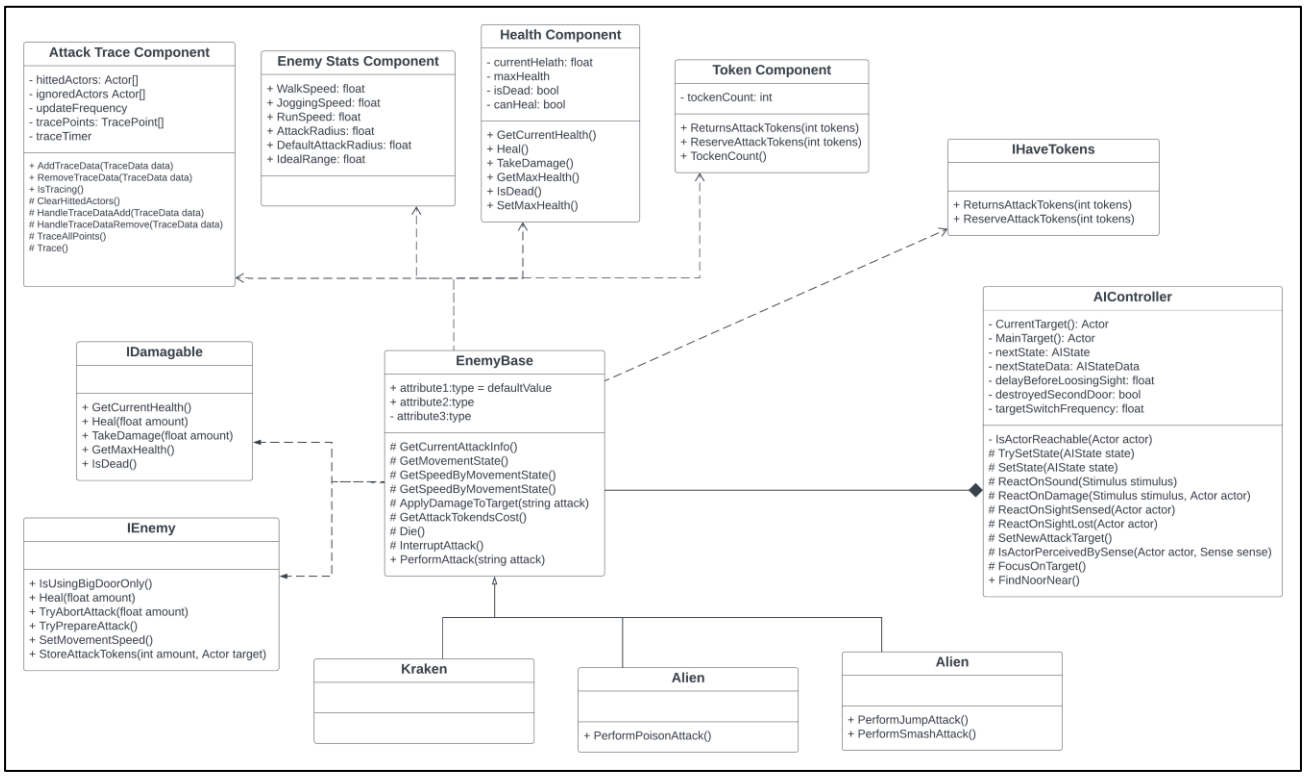


Рисунок 3.2 – Діаграма класів ворога

Можна побачити, що в нас є два основних об'єкти – це ворог та контролер ворога.

AIController це об'єкт, який зосереджений на реагуванні на вхідні дані з навколишнього середовища та ігрового світу. Робота AIController полягає у спостереженні за навколишнім світом, прийнятті рішень і відповідній реакції без явного втручання людини-гравця [12]. Наша розширена версія базового контролеру вміщає усю логіку зміну станів AI, реакції на різні відчуття, які отримує ворог з навколишнього середовища, методи фокусу на цілі при її переслідуванні, пошуку перешкод, таких як двері. Контролер тільки змінює основний стан та поведінку ворога. І він є мостом між ним та деревами поведінки про які згадано далі.

На плечі ворога накладений функціонал атак, смерті, нанесення та отримання урону, а також взаємодія з компонентами. Важливо помітити що ворог залежить від контролера, і не може без нього існувати. Контролер спілкується з ворогом через його загальний інтерфейс IEnemy, щоб створити додатковий рівень абстракції.

Крім інтерфейсу IEnemy, ворог реалізує інтерфейс IDamagable, який дозволяє отримувати йому урон від гравців. Важливо помітити, що урон повинен накладається через актора, а не відповідну компоненту, адже ворог може додатково реагувати на урон зі своєї сторони, наприклад, відігравати анімацію чи створювати ефект пошкодження.

Також з диграми видно, що актор залежить від 4 компонент:

– компонента здоров'я відповідальна за визначення поточного та максимального здоров'я, обробку смерті та визначення чи може отримати ціль урон чи ні;
– компонента токенів забезпечує управління токенами, які використовують під час атак ворогів. Про токени пояснено далі в цьому розділі;
– компонента характеристик вміщає абсолютні показники особливостей ворога. Це дозволить отримати відповідні значення з різних підсистем, без додаткових перетворень при роботі з базовою пішкою, від якої унаслідується основний клас ворога.
– компонента трасування атаки відповідає за фіксування потраплянь по цілям під час атак ворога. Вона орієнтована на обробку декількох атак одночасно, наприклад при замахуванні декількома кінцівками.

Нарешті, від ворога успадковуються інші підкласи Kraken, Raptor та Alien. Кожен з них є має окремі характеристики, типу швидкості атаки та її радіусу, відстані зору та слуху.

На діаграмі можна було помітити ще інтефейс IHaveTokens. Він необхідний для перевірки наявності токенів у кілі перед атакої на неї.

3.2.2 Система токенів

Система токенів це один з підходів вирішення проблеми синхронізації атак великої кількості ворогів. Гравець може уникати їх, тому інколи виникають ситуації коли персонажа оточує дуже велика велика кількість ворогів. Це може моментально привести до смерті, або викликати проблеми, коли гравець оточений, і не може вибратися з натовпу.

Так звана «Система токенів атака» була вперше використана у Doom 2016. Вона полягає в тому, що будь-яка атака ворога має певну ціну в токенах [13]. У свою чергу цілі ворогів, мають певний запас токенів. Коли ворог намагається атакувати, він перевіряє чи має ціль достатню кількість токенів для атаки. Якщо так, то він їх бронює на час атаки, а потім повертає. Якщо ні, но шукає іншу вільну ціль. Таким чином можна гнучко балансувати кількість атакуючих ворогів та складність сутички, виставляючи потрібну ціну атак від їх смертоносності чи розміру ворога.

3.2.3 Проектування AI

Для реалізації AI будуть використані дерева поведінки. Дерево поведінки – це дерево ієрархічних вузлів, які контролюють потік прийняття рішень об'єктом ШІ. Листя дерева – це власне команди, які керують об'єктом AI, а гілки – це різні типи утилітарних вузлів, які керують рухом AI по деревах, щоб досягти послідовності команд, які найкраще підходять для даної ситуації [14].

Дерева поведінки дозволяють створювати складні та розгалужені поведінки, які можна адаптувати до різних ситуацій. Це робить їх ідеальними для моделювання поведінки AI, який має реагувати на дії гравця та своє оточення. Дерева поведінки, як правило, є кращим вибором для реалізації AI порівняно з машинами станів. Адже переходи між станами є більш гнучкими, порівняно з фіксованими у машинах станів. Більш того, усі команди які використовуються є модульними, тому спільні завдання між ворогами буде легко розділити.

Unreal Engine має вбудовано систему дерев, що дозволить візульно налаштовувати AI всередині двигуна.

3.3 Проектування VFX

Основним інтрументом створення VFX виступить системи частинок Niagara. Niagara – це система візуалізації наступного покоління Unreal Engine. Кожен ефект упаковується в окрему систему, яка є певним контейнером. Усередині цієї системи у можуть бути різні будівельні блоки, які складаються, щоб допомогти створити

загальний ефект [15]. Цими будівельними блоками виступають так звані емітери. Емітери виступають певною частиною загального ефекту, і керують повним життєвим циклом частинок цього підефекту. Поділ систем на емітери сприяють модульності, адже емітери можна перевикористати в іншим системах частинок. Крім того, емітери складають з модулів, які теж є теж модульними. Ці модулі відповідають за окрему функцію оновлення частинки чи емітеру загалом, наприкад, для симуляції випадкового руху частинок є окремий модуль для накладання фізичної сили шуму на них.

Таким чином, загальним принципом при роботі з VFX буде створення окремих емітерів та модулів, які зможуть бути знайти своє використання в різних місцях середовища та подіях на космічному кораблі. Зокрема ефекти диму та іскор є основними кандидатами на модульність.

Для оптимізації ефектів потрібно розуміти основні вразливі системи ігрового двигуна та особливості, як ефекти впливають на них. В Unreal Engine 5 є три такі системи: ігровий процес, процес рендеру та GPU. Перші два виконується на CPU, другий відповідно назві на GPU. Від навантаження на ці системи залежить час обробки кожного кадру гри. При цьому час буде не менше ніж найдовший час обробки кадру в кожній системі. Це працює так, що дані з ігрового процесу передаються на процес рендеру, а далі на GPU, тобто системи працюють послідовно (див. рис. 3.3). Варто зупинитися на кожному процесі окремо, для більш глубокого розуміння проблем, які викликають ефекти на інші вузькі горлечка при розробці на Unreal.

Ігровий процес відповідає за обробку ігрової логики. Це включає оновлення розташування об'єктів, зміну внутрішніх станів через події та щакадрові обрахунки, фізику, AI тощо. Тобто тут виконуються ті обрахунки, які не пов'язані напряму з візуалізацією.

Процес рендеру у свою чергу відповідни за визначення тих об'єктів, які в подальшому повинні бути відмальовані. Це виконуються за рахунок відсіювання об'єктів, які не потрапили в зону екрану чи розташовані за перепоною, через які їх

не видно. Крім того тут формуються так звані «виклики малювання», які пакуються та передаються на GPU.

Нарешті процес GPU відповідальний за відмалювання усіх пікселів на екрані. На ньому оброблюється геометрія та накладаються шейдери.

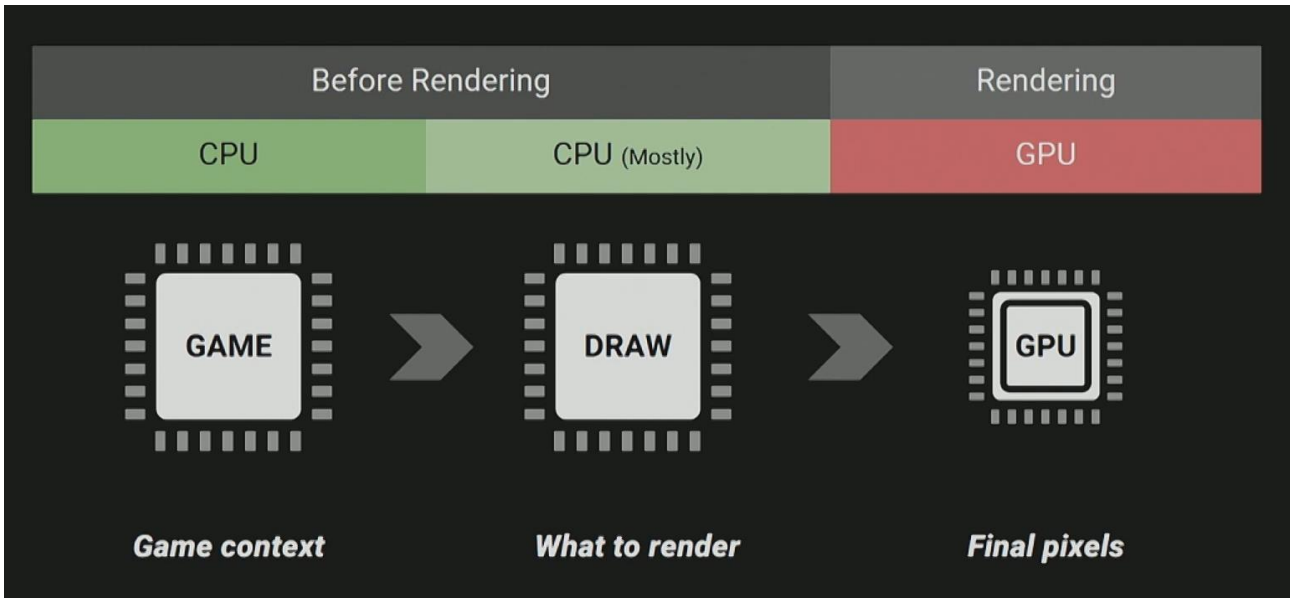


Рисунок 3.3 – Основні системи обробки кадру в Unreal Engine [16]

Кількість частинок напряму впливає на всі зазначені системи. Чим більше частинок на сцені та чим довше вони живуть, тим більше витрат припадає на їх обробку та відмалювання [17]. Тому потрібно в розумних рамках збалансувати візуальну привабливість та продуктивність ефектів.

Теж стосується емітерів та систем, адже на підтримку кожної потрібні ресурси. Замість використання декількох схожих емітерів можна присвоювати групу породження кожній частинці і залежно від неї налаштовувати параметри частинок. У свою чергу для групування систем можна використати так звані канали даних Niagara, які комбінують усі системи Niagara в одну, що набагато продуктивніше [18].

Однією вагомою проблемою при розробці частинок є перемалювання, або ціна шейдеру пікселя. Воно спостерігається на GPU та залежить від складності шейдерів та кількості прозорих шарів [17]. Перемалювання можна представити наступною формулою:

$$\text{Перемалювання} = LayersNum * \overline{AffectedPixelNum} * \overline{LayerInstrNum} \quad (3.1)$$

де *LayersNum* – кількість шарів;

AffectedPixelNum – кількість пікселів зачеплених шаром;

LayerInstrNum – кількість інструкцій за шар.

Оскільки більшість частинок це прозорі спрайти, важливо звертати увагу на фінальну складність шейдерів сцени. Для частинок де спрайт більший на частинку яку видно, треба використати обрізання частинок, щоб розмір спрайту був відповідний частинці. Це актуально для атласів анімацій диму та вогню, які будуть використані для відповідних ефектів, адже в них часто площа тайлу варіюється від поточного тайлу.

Для спрощення шейдерів будуть використані запечені текстури шумів, замість процедурних. Крім того для різних налаштувань графіки деякі складні обрахунки чи функції у матеріалах потрібно підлаштувати для них.

Для охолодження великої кількості систем буде використана система масштабування, яка є потужним інструментом для оптимізації ефектів залежно від поточних ресурсів пристрою, платформи, або відстані до об’єкта. Це сприяє зменшенню динамічних витрат основних систем Unreal Engine, без вагомих збитків візуальної складової продукту при належному налаштуванні інструменту [19]. Загалом ця система актуальна для вогню, який потенційно може виникнути на усьому кораблі одночасно, а також ефектів поламок, які прив'язані до окремих модулів.

Для всіх ефектів, які породжуються в великій кількості, як іскри, дим та вогонь, по можливості будуть використаний GPU режим симуляції, адже він розрахований на обробку великих кількостей частинок, на відміну від CPU. Відповідно до цього іскри повинні використати методи колізій доступні на GPU – буфер глубини чи поля дистанцій.

Оскільки GPU симуляції не підтримують динамічного визначення границь ефекту, вони будуть підібрані вручну, щоб прибирати з рендеру непотрібні VFX.

Таким чином зазначені техніки будуть використані при розробці візуальних ефектів, щоб досягти оптимального споживання ресурсів на їх обробку.

3.4 Проектування UI / UX

Важливим етап створення UI є створення так званих мокапів – спрощених копій інтерфейсу, який відображають положення елементів, їх групування, розмір та потік з іншими мокапами. Для цього потрібно по-перше виокремити окремі частини інтерфейсу, які відображають певні екрани та/або його підменю. По-друге усі екрани розбиваються на елементи, щоб виявити, інтерактивні частини, відповідаючі за дії чи навігацію. Для цього варто відповісти на такі питання, як: «Що я хочу побачити в цьому меню?», «Як я можу взаємодіяти з цим меню?» та «Як я можу перейти до інших меню та підменю?». Це дозволить зрозуміти очікування користувача, виокремити дані потрібні для відображення та зрозуміти потік між екранами. Крім того розділення на елементи дозволяє визначити модульні частини, які можна перевикориставти.

Зокрема було виділено наступні екрани та підекрани: головне меню, меню налаштувань, меню створення кооперативної партії, підменю авторизації в онлайн сесію, підменю пошуку сесії, меню лобі сесії, підменю паузи, меню магазину.

Крім того важливо обрати стилістику інтерфейсу для усіх меню. Це має значення адже інтерфейс повинен бути плавним продовженням самої гри та її естетики, що сприятиме зануренню в ігровий процес. Оскільки гра в sc-fi тематиці було вирішено спробувати білі тона. Білий колір створює сильний контраст з темним простором космосу, що робить його ідеальним для його виділення серед затемнілих просторів космічного корабля.

На рисунку 3.4 зображено головне меню гри. З нього можна перейти в кооперативні режимі, почати однокористувацьку гру чи вийти з гри.

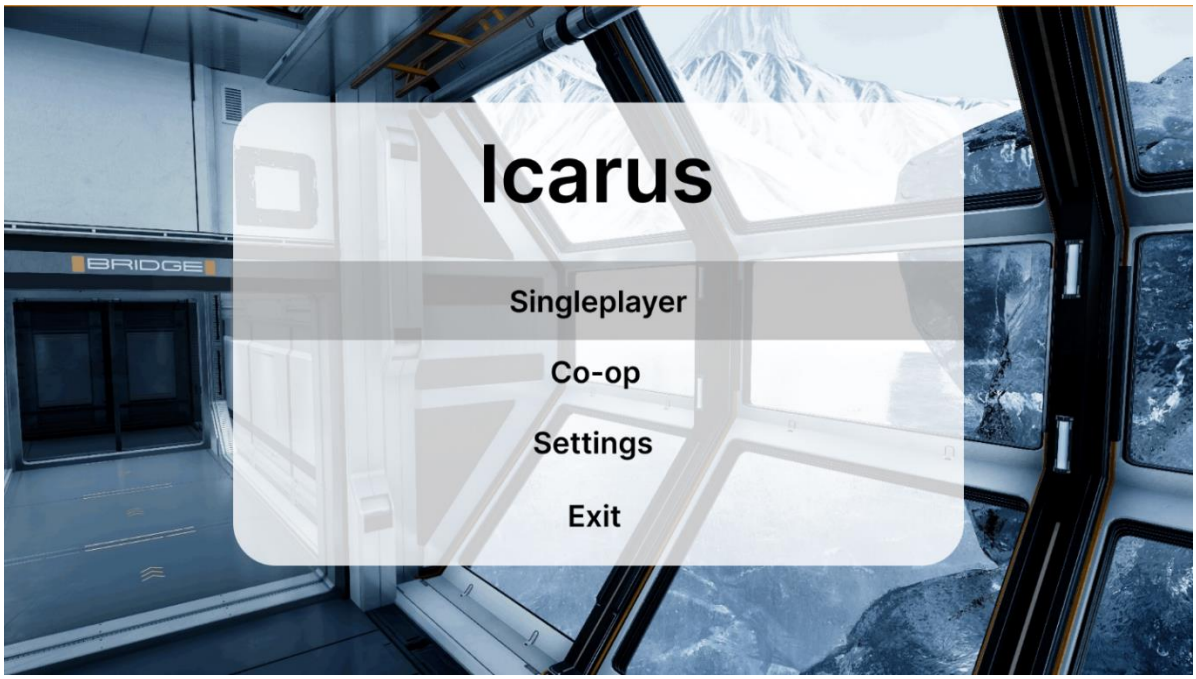


Рисунок 3.4 – Мокап головного меню гри

З початку проектування UI також важливо думати про можливості вводу в грі. Т.я. застосунок підтримує як клавіатуру, так і миш, використання картинок клавіш та кнопок є важливим елементом інтерактивних частин інтерфейсу.

З точки зору UX доцільно запитувати користувача при важливих діях, таких як вихід з гри, чи купівля товару. Тому було створено мокап вікна підтвердження дії (див. рис. 3.5).

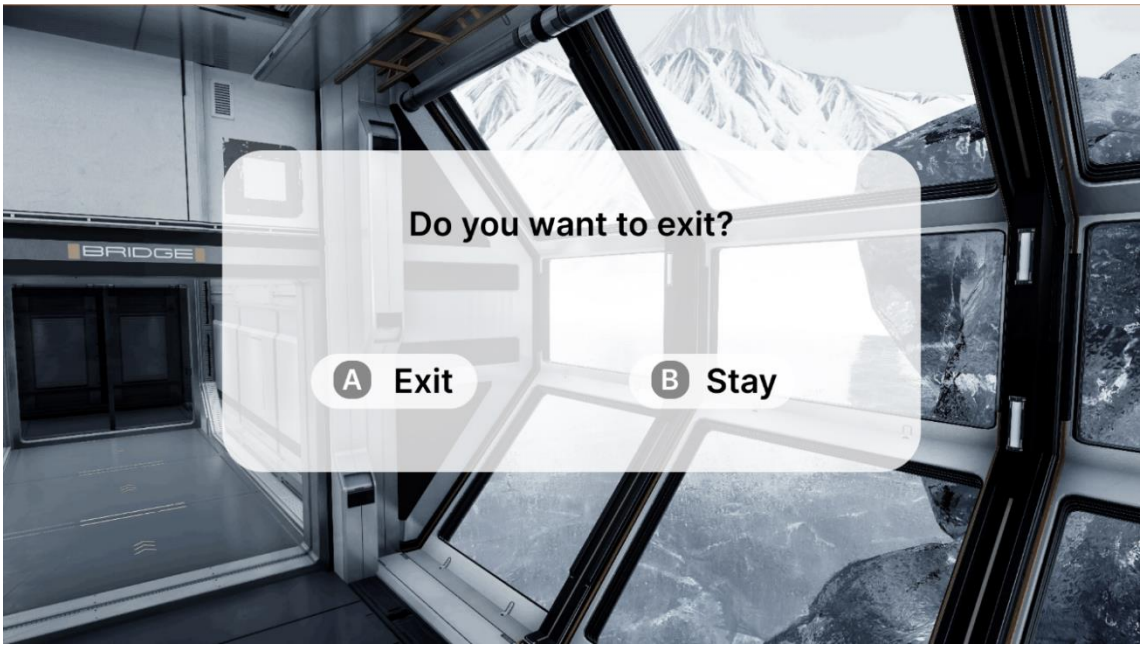


Рисунок 3.5 – Мокап вікна підтвердження

Наступним екраном йде меню кооперативної сесії, де кожен гравець обирає роль зі свого геймпаду окремо (див. рис. 3.6).

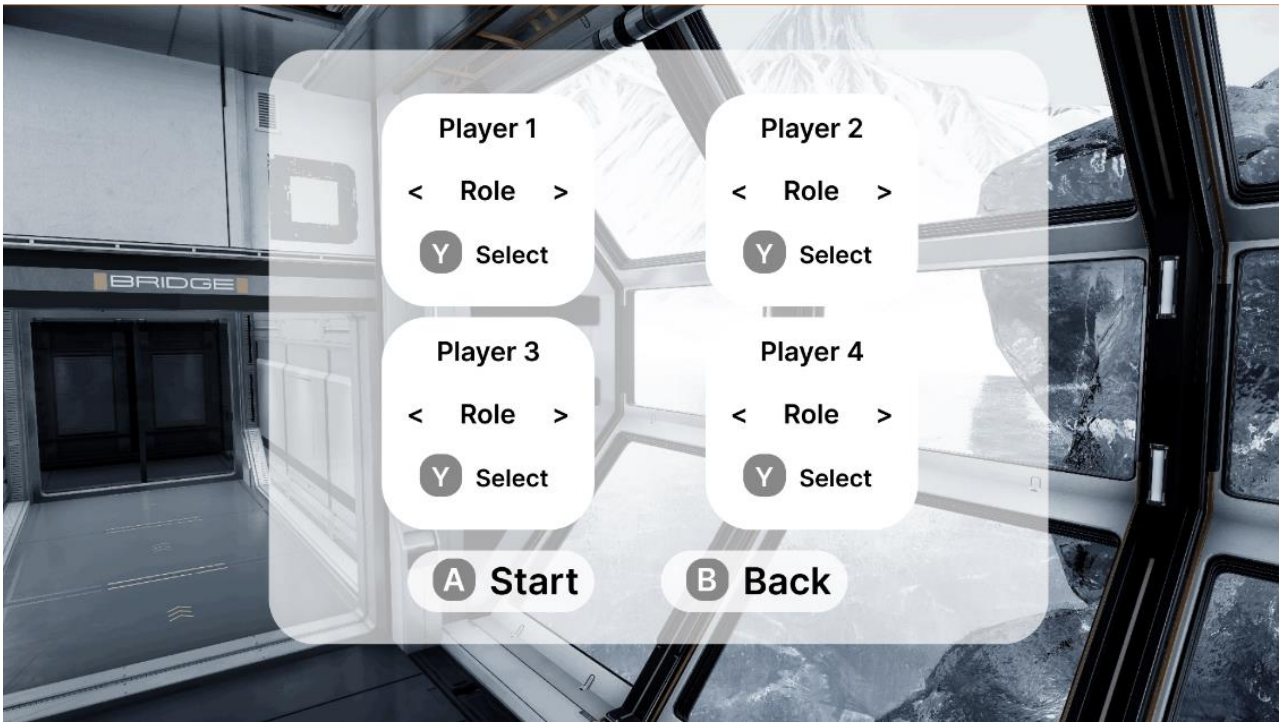


Рисунок 3.6 – Мокап вікна режиму роздільного екрану

Вигляд макету вікна налаштувань зображено на рисунку 3.7.

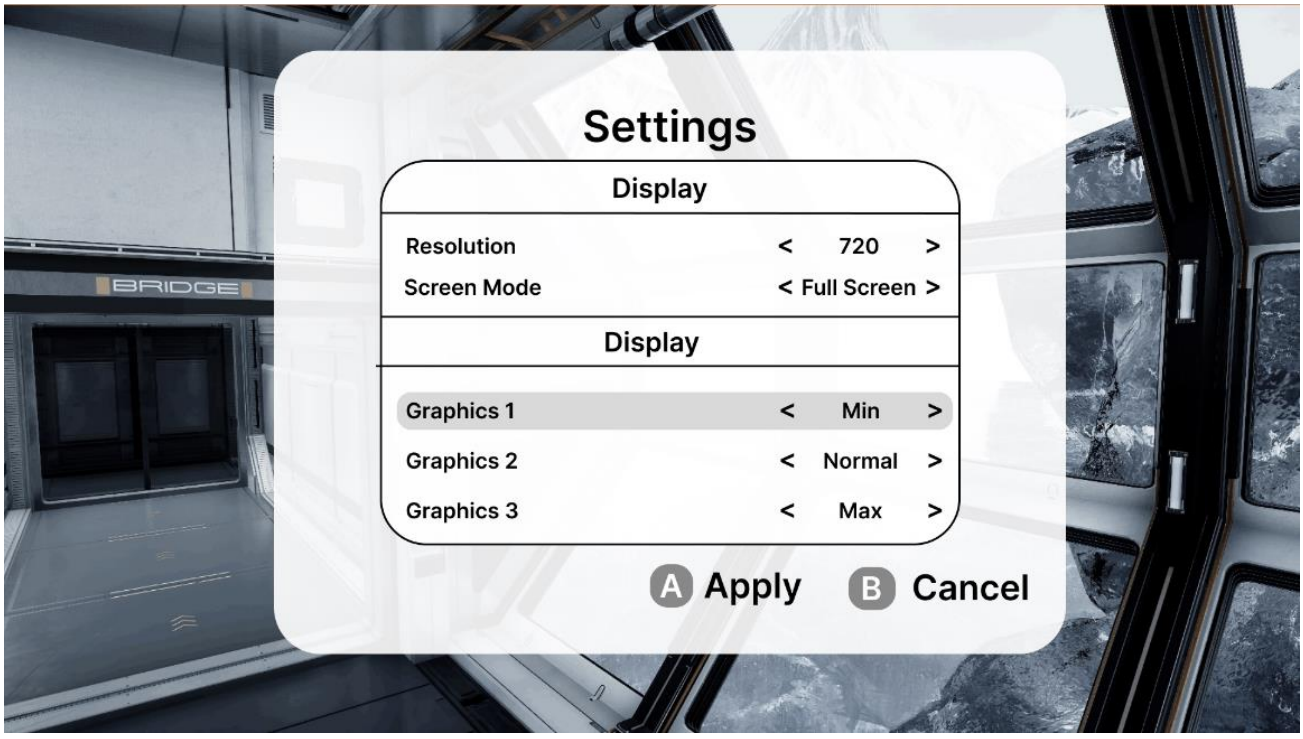


Рисунок 3.7 – Мокап вікна налаштувань

На рисунку 3.8 наведено мокап підменю для створення сесії. Опція «Create Session» відкриває меню зліва, «Exit» та «Log out» відправляють до головного меню, з попереднім виходом з акаунту у другому випадку. Опція «Connect» відправляє до списку сесій зображного на рисунку 3.8. Через це меню можна підключитися до онлайн сесії з меню вибором ролі, яке було наведено раніше.

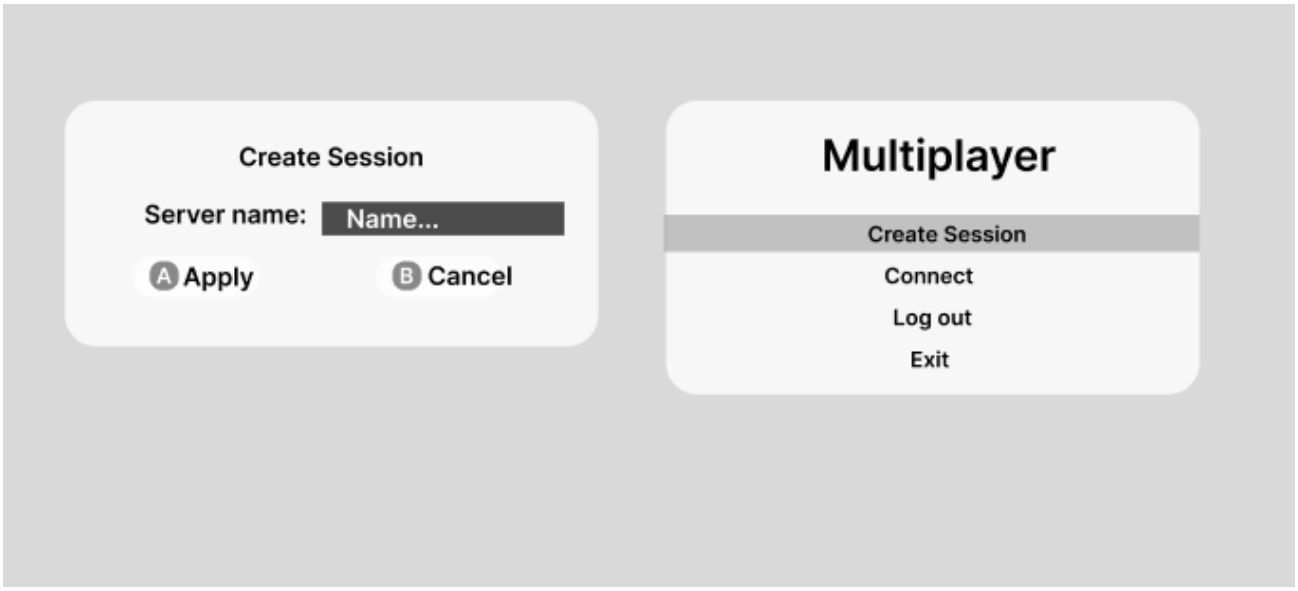


Рисунок 3.8 – Мокап створення сесії та меню

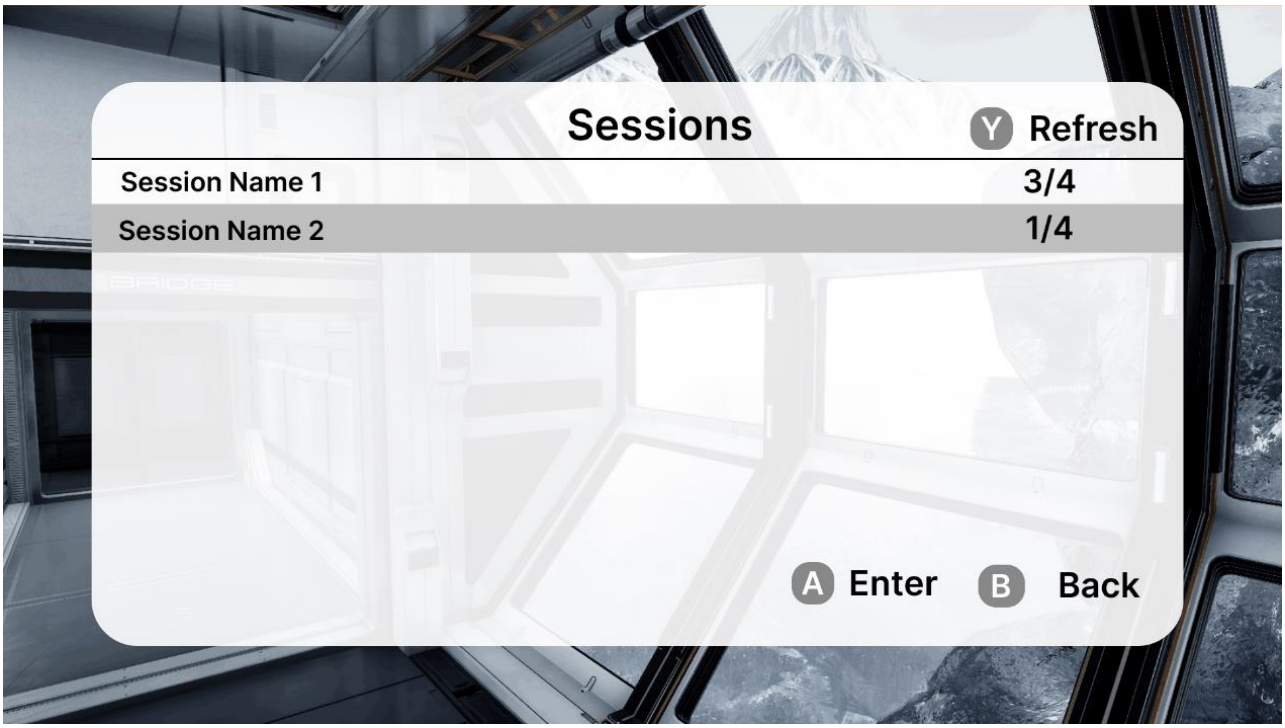


Рисунок 3.9 –Мокап екрану списку серсерів

В меню сесій можна побачити назву сесії, по якій можна знайти потрібну партію та кількість гравців в лобі, щоб знайти повну команду.

Меню завантаження включає віджет завантаження на підказки для гравця (див. рис. 3.10).

Рисунок 3.10 – Мокап екрану завантаження

Рисунок 3.11 відображає меню паузи під час ігрової партії.

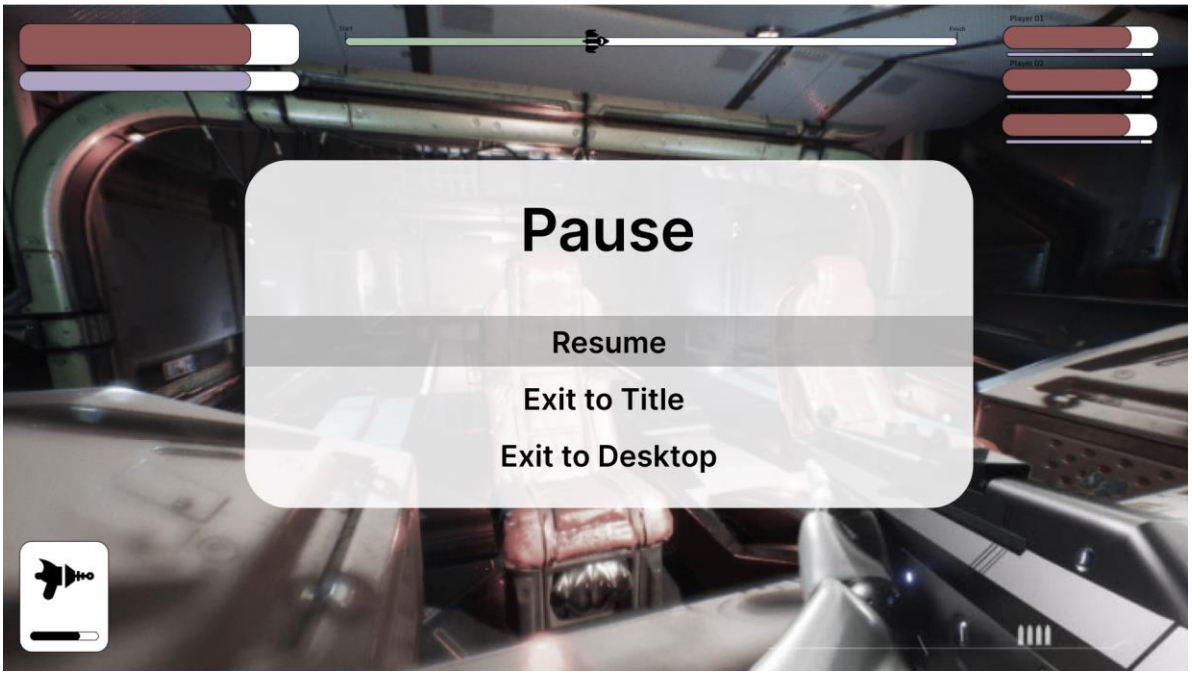


Рисунок 3.11 – Мокап меню паузи

Останнім екраном є меню магазину (див. рис. 3.12). Тут перед гравцем відображено список усіх товарів. Важливим момент є визначення максимальної висоти вікна, адже гравець повинен бачити свої показники при вирішенні, які предмети купити.

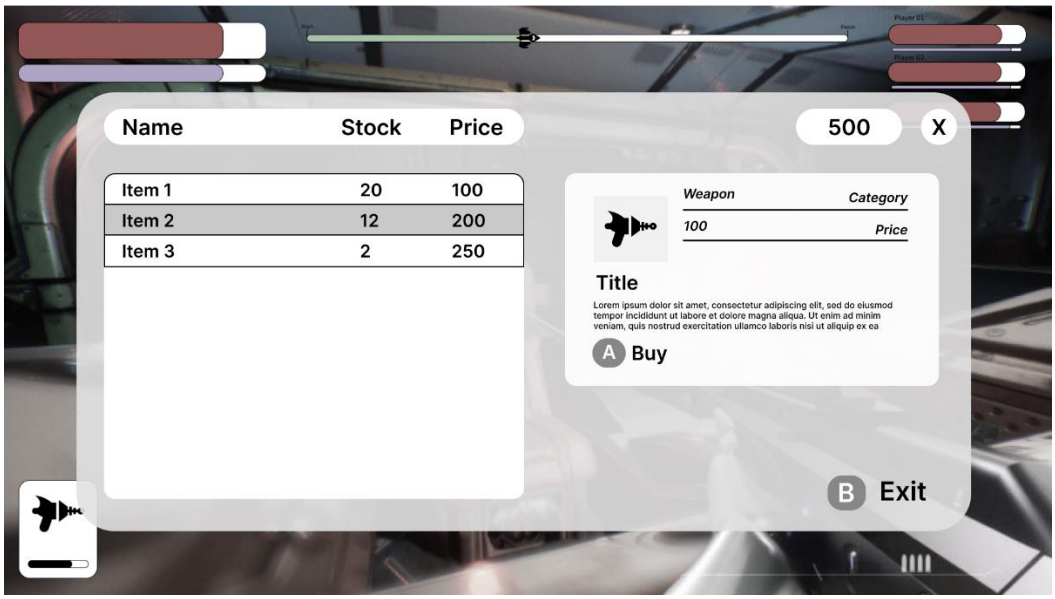


Рисунок 3.12 – Мокап екрану магазину

Крім того було спроектовано екрани для розділеного екрану на двох, трьох та чотирьох гравців (див. рис. 3.13-3.15).

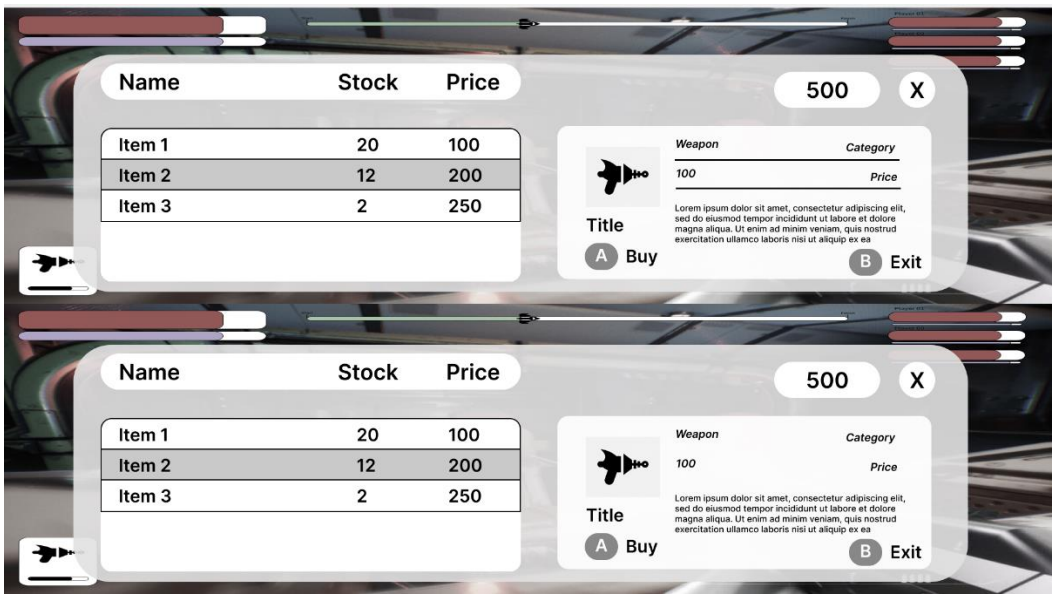


Рисунок 3.13 – Мокап екрану магазину на двох гравців

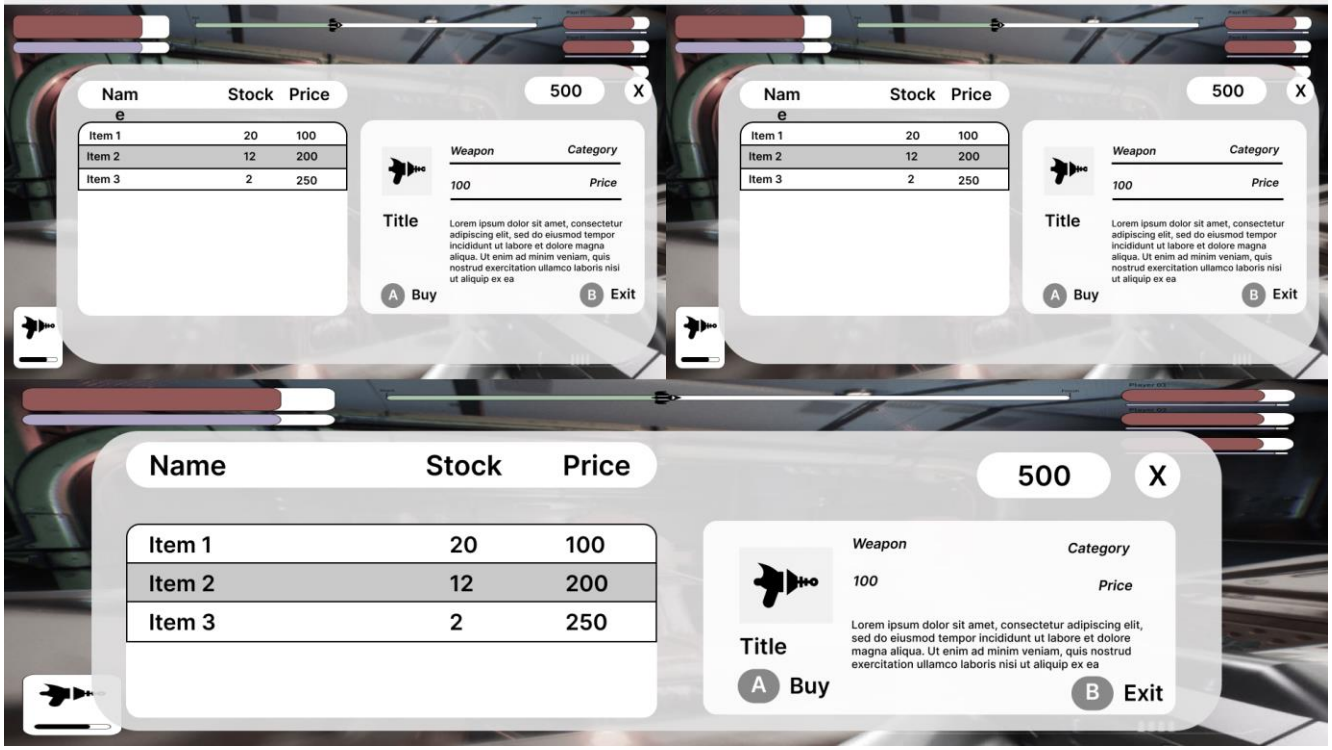


Рисунок 3.14 – Мокап екрану магазину на двох гравців

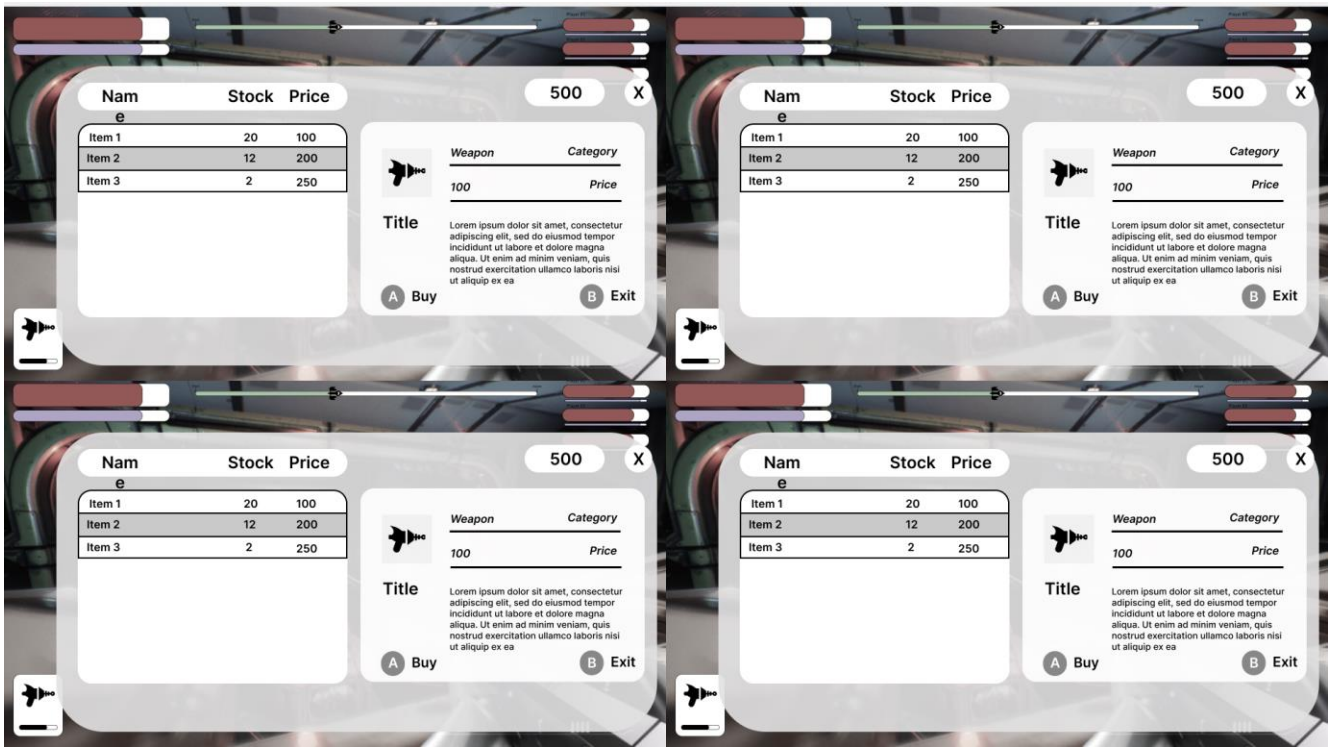


Рисунок 3.15 – Мокап екрану магазину на двох гравців

Представлені мокапи не є остаточною версією реалізації, адже усі моменти не можливо продумати наперед, а нові ідеї які можуть покращити елементи

дизайну часто виникають під час самої розробки. Тим не менш ці макети дозволять краще зрозуміти особливості взаємодії з кожним екраном, виокремивши інтерактивні елементи, якими гравець управляє з пристрою вводу.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Створення VFX

Першочергово для ефекті були підготовлені об'єкти та текстури. Більшість з них була взята з наборів асетів, але деякі були згенеровані всередині двигуна. Зокрема було створено їдкий шум та шум мармуру. Запечені текстури шумів, дозволяють спростити шейдери, адже динамічний обрахунок заміняється звичайною алокацією пам'яті, яку легше контролювати.

Першими найпростішими ефектами були пошкодження модулів, які представляли дим (див. рис. 4.1). Вони створювалися за допомоги атласів анімації.

Рисунок 4.1 – Ефект диму критичної (зліва) та некритичної (справа) поламки модуля

За рахунок обрізання текстур та зменшення спрайтів отримуємо оптимальне складність шейдерів (див. рис. 4.2).

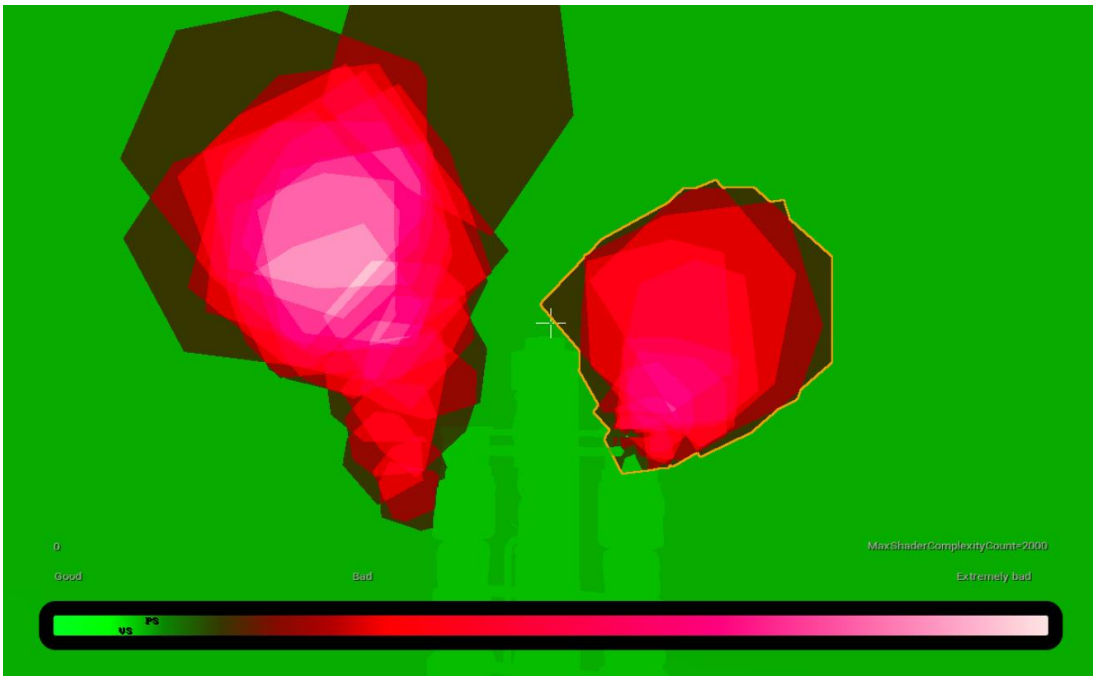


Рисунок 4.2 – Складність шейдерів диму критичної (зліва) та некритичної (справа) поламки модуля

Для створення ефекту стиснутого газу (див. рис. 4.3) для вогнегасника також був використаний атлас, тільки більш пухкий.

Рисунок 4.3 – Ефект стиснутого газу з вогнегасника

Для відтворення колізій був використаний метод полів дистанції. Це дозволяє більш надійно проводити розрахунки колізій, порівняно в буфером глибини.

Оскільки ефект прикріплений до вогнегасника яким керую гравець, частинки можуть бути не в полі зору камери. Поля дистанції дозволяють проводити перевірку фізики без обмежень камери.

Для створення зварювального променя для зварки був використаний стрічковий випромінювач з формою трубки (див. рис. 4.4).

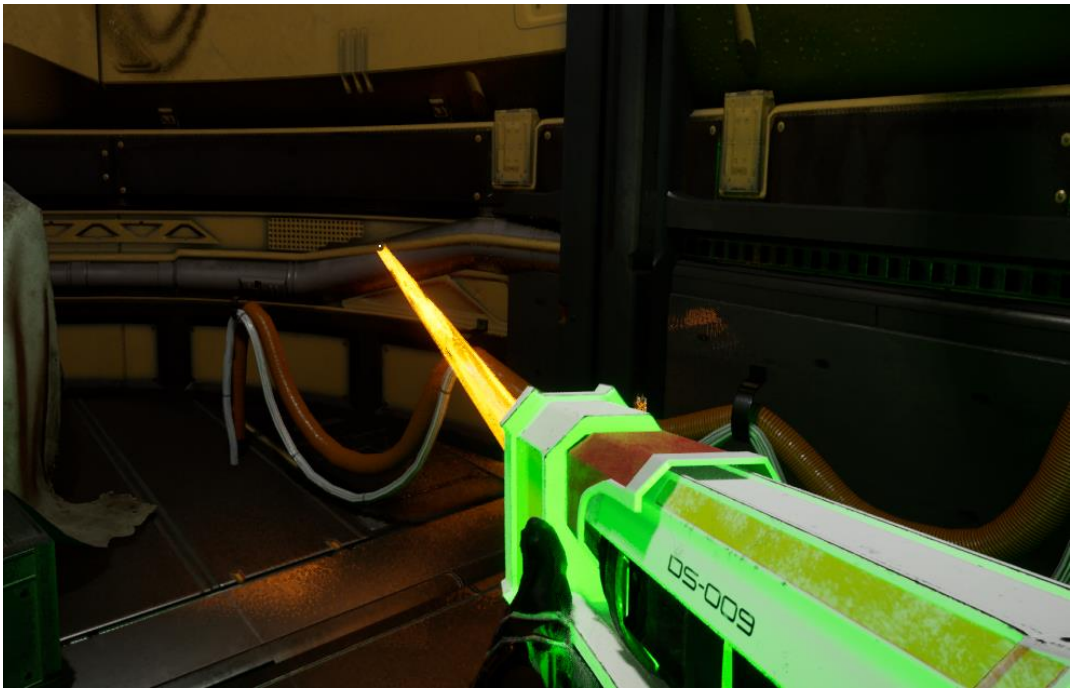


Рисунок 4.4 – Ефект променя зварювального апарату

Щоб надати унікальних рис променю був створений матеріал зі спотворенням з використанням шуму (див. рис. 4.5).

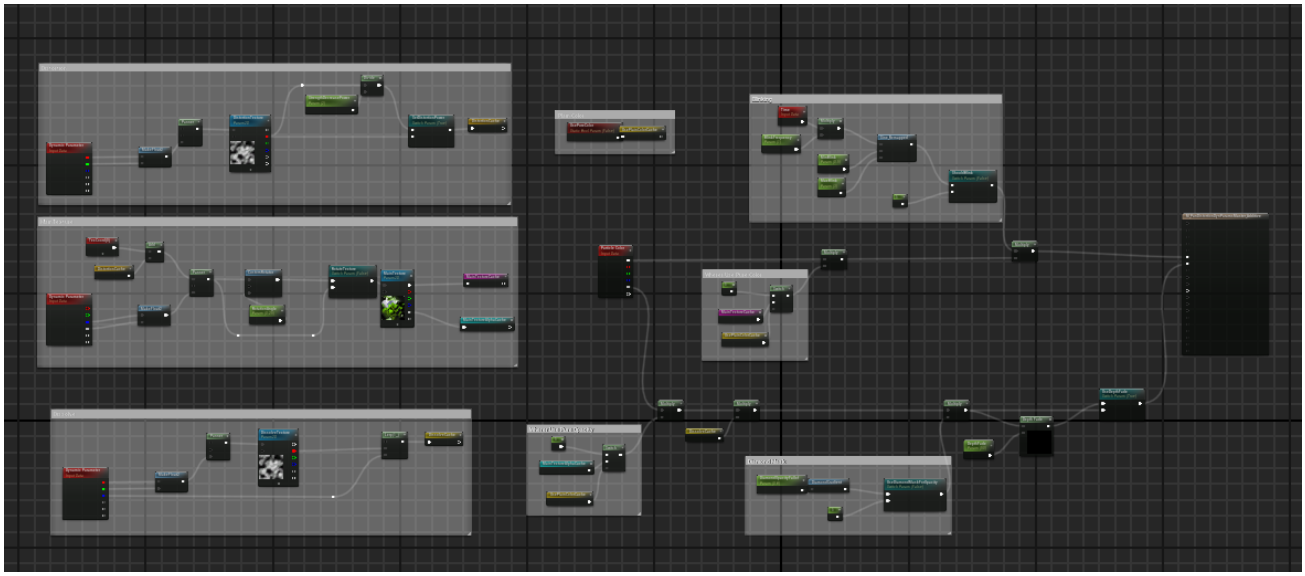


Рисунок 4.5 – Матеріал з рухомим шумом

Матеріал приймає основну текстуру, текстуру спотворення, текстуру розмиття й ряд параметрів для їх налаштування. Текстура спотворення переміщую UV координати основної, а текстура розмиття дозволяє приховати деякі деталі. Взявши для цього попередньо запечені шуми та додавши рух до кожної текстури, отримуємо неоднорідний паттерн. При цьому швидкість кожної текстури, а також сила спотворення та розмиття передаються через динамічні параметри, які застосовуються в окремих емітерах частинок, які їх використовують. Також матеріал включає параметри для регулювання яскравості, зокрема дозволяючи змінювати її з часом. Це додає окремих деталей ефекту, адже він випромінює світло на темні частини корабля.

Для створення ефекту «розсіювання» після використання лазера, на місці колізії променя з об'єктами корабля створюються так звані наклейки. Вони дозволяють зробити проекцію матеріалу на поверхню, у випадку з променем це матеріал перегрітої поверхні (див. рис. 4.6).

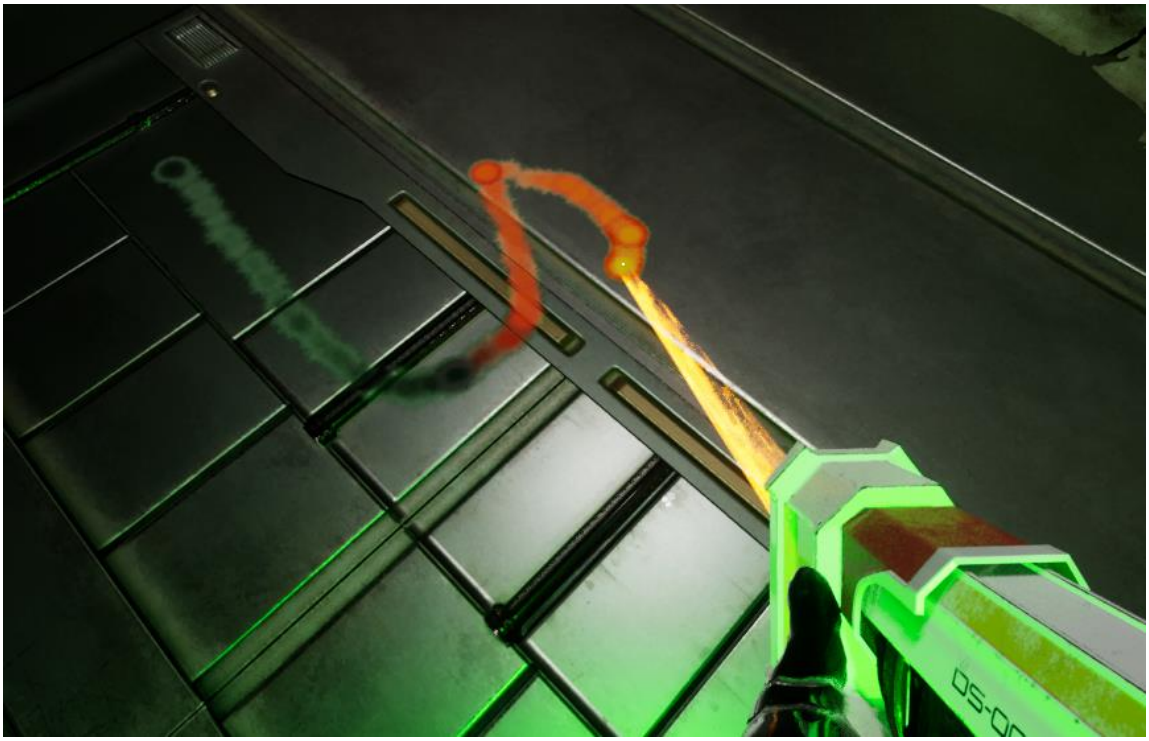


Рисунок 4.6 – Розсіювання у місцях колізії променю

Взагалі ефекти розсіювання є важливими складовими VFX. Вони дозволяють гравцю відчути результат якоїсь події. У цьому випадку взаємодії з інструментом

зварювання. Крім візуальної складової, ці наклейки також відкривають додатковий простір для комунікації між гравцями. Наприклад, вони можуть залишати позначки на підлозі, коли на кораблі пошкоджене світло, адже наклейка випромінює легке світло протягом декількох секунд перед згасанням і зникненням.

При взаємодії променя з модулями, які можна ремонтувати виникає ефект самої зварки (див. рис. 4.7). Він одразу поєднує яскраву електричну дугу, декілька видів іскор та легкий димок. Іскри, як і стиснутий газ, покладаються на полі дистанцій для створення ефекту відскоку від поверхні, зберігаючи свій імпульс.

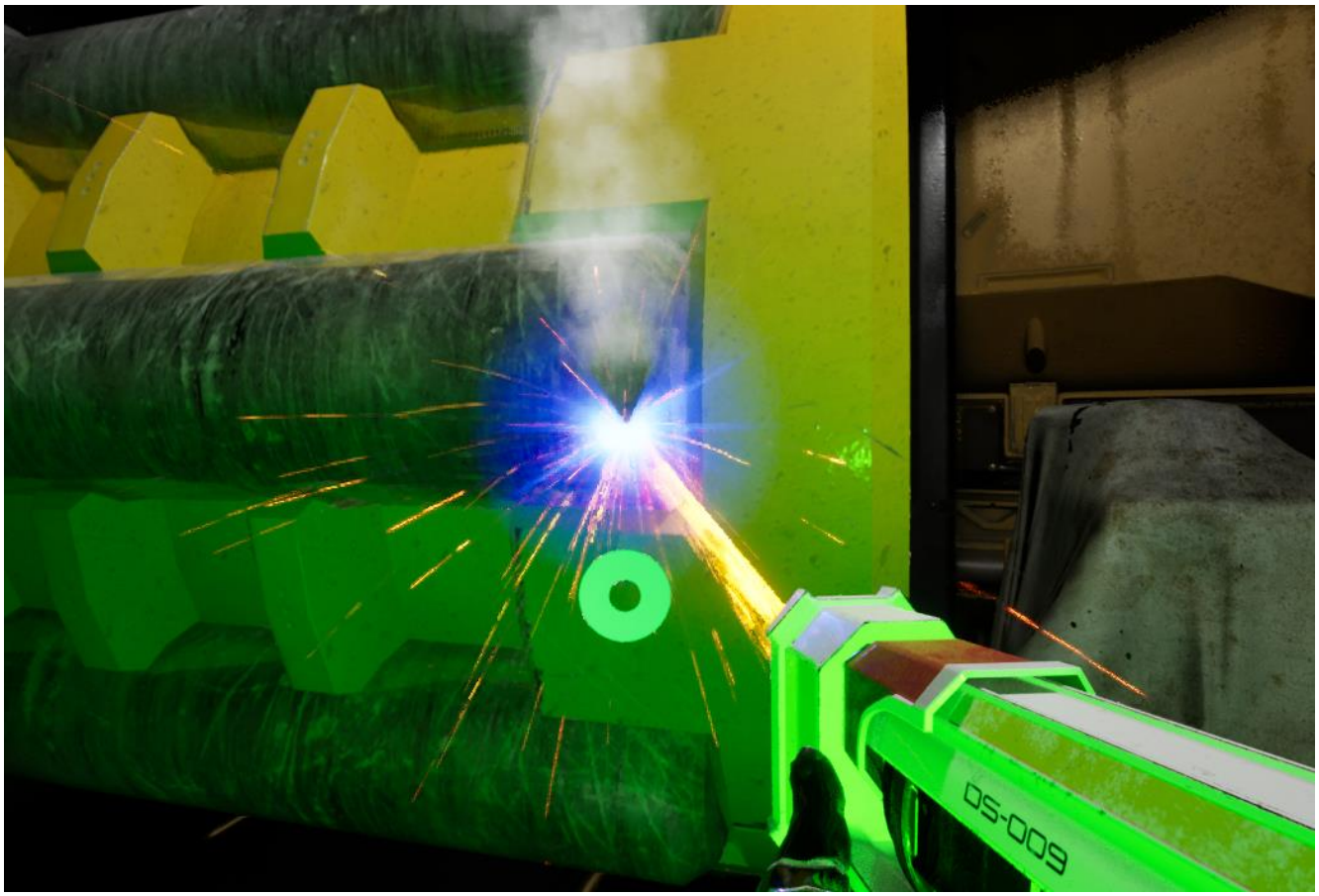


Рисунок 4.7 – Розсіювання у місцях колізії променю

Оскільки гра футуристичної естетики ефект починки паяльнику був виконаний у вигляді блискавки (див. рис. 4.8). Незважаючи на те що ефект паяльнику дещо схожий по взаємодії на ефект зварювання, адже вони представляють певний промінь який виходить з інструмента гравця, паяльник був реалізований інакше. Замість єдиного стрічкового випромінювача, тут звичайний

випромінювач спрайтів. Він вирівнюється уздовж нормалі від дула інструмента до точки колізії. При цьому в матеріалі використані довгі текстури блискавок, які додатково розтягуються. Спрайт було вирішено використати, задля надання нерівномірності ефекту блискавки, яка з'являється на визначеній площині кола уздовж нормалі.

Рисунок 4.8 – Ефект використання паяльного інструменту

Під час починки блискавка розсіюється та з'являються невеличкі ефекти струму, які кружляють.

Цікавим під час реалізації був ефект торгівця. Було вирішено створити деякий силует з частинок з жіночою формою (див. рис. 4.9). Для цього використовується сітка скелету, яка дискретизується на GPU. Оновлення виконується кожен кадр, що дозволяє відігравати анімації на скелеті. Зокрема при вході у кімнату силует махає гравцеві рукою. Щоб надати частинкам цифровою уніфікованості, вони розташовуються уздовж X та Y координат систем із зазначеним кроком. Кожна частинка отримує різні значення максимальної

яскравості та розміру, які застосовуються з перемінною частотою, що створює певний ембієнт ефект, коли гравець стоїть поряд чи взаємодії напряму з торгівцем.

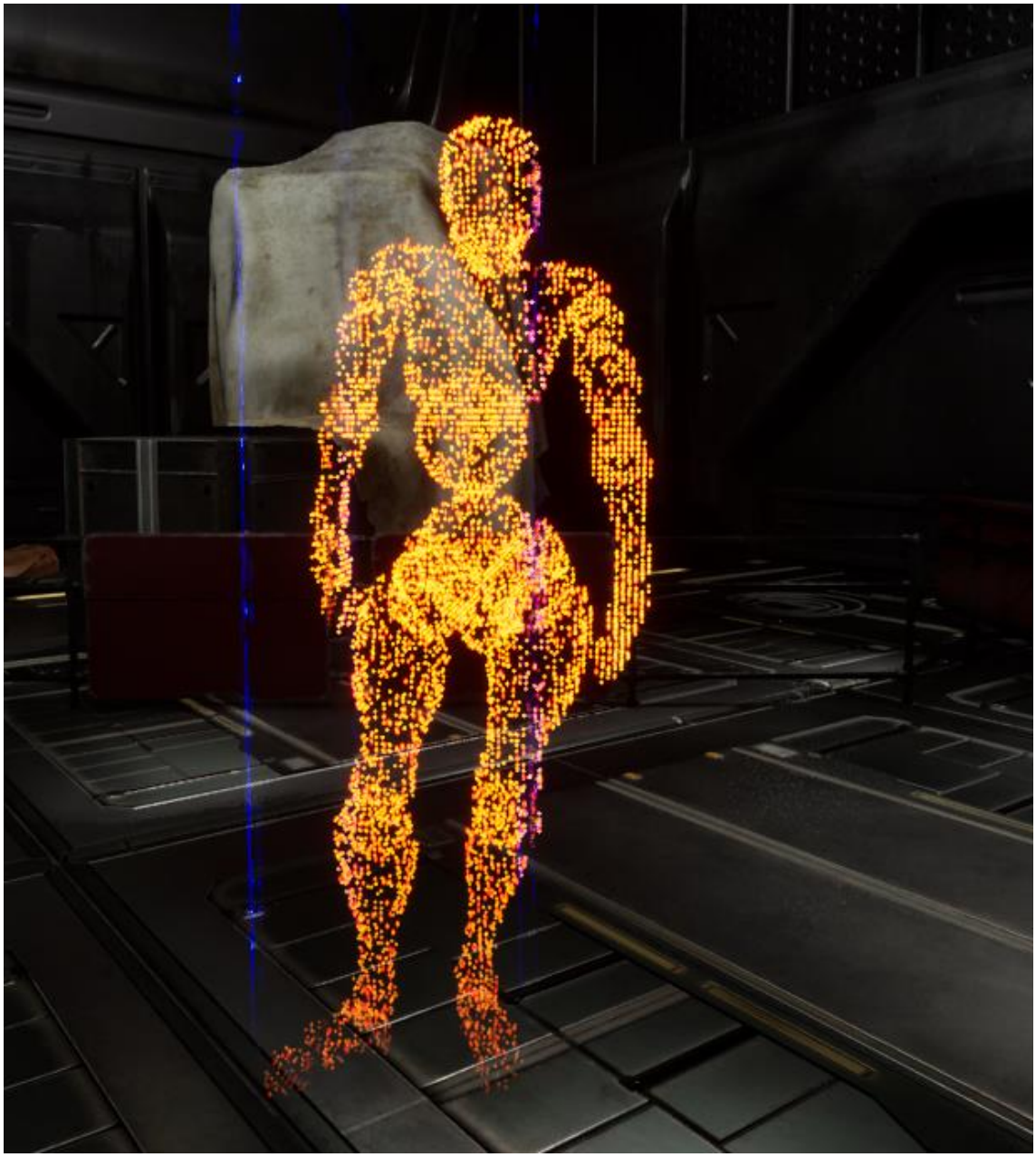


Рисунок 4.9 – Ефект торгівця

Для додання більшої кількості деталей були створені свого роду збої, які також можна побачити на рисунку. Вони частково надихалися ефектом, який можна було побачити в Cyperpunk 2077. Збої виникають у певному діапазоні та рухаються уздовж бокової осі скелету. При цьому вони змінюють розмір та приховують відповідно нього частинки навколо скелету. Для цього в основному емітері використовується читач атрибутів емітеру зброїв. Далі для групи оновлення

частинок був створений спеціальний модуль, який зчитує положення та розмір збоїв, ітерує через усі частинки і визначає їх альфа значення додатково роблячи інтерполяцію на базі дистанції до збою, щоб забезпечити плавне зникнення частинок. Відповідні два емітери можна побачити на рисунку 4.10, де зображені усі використані модулі та власні змінні.

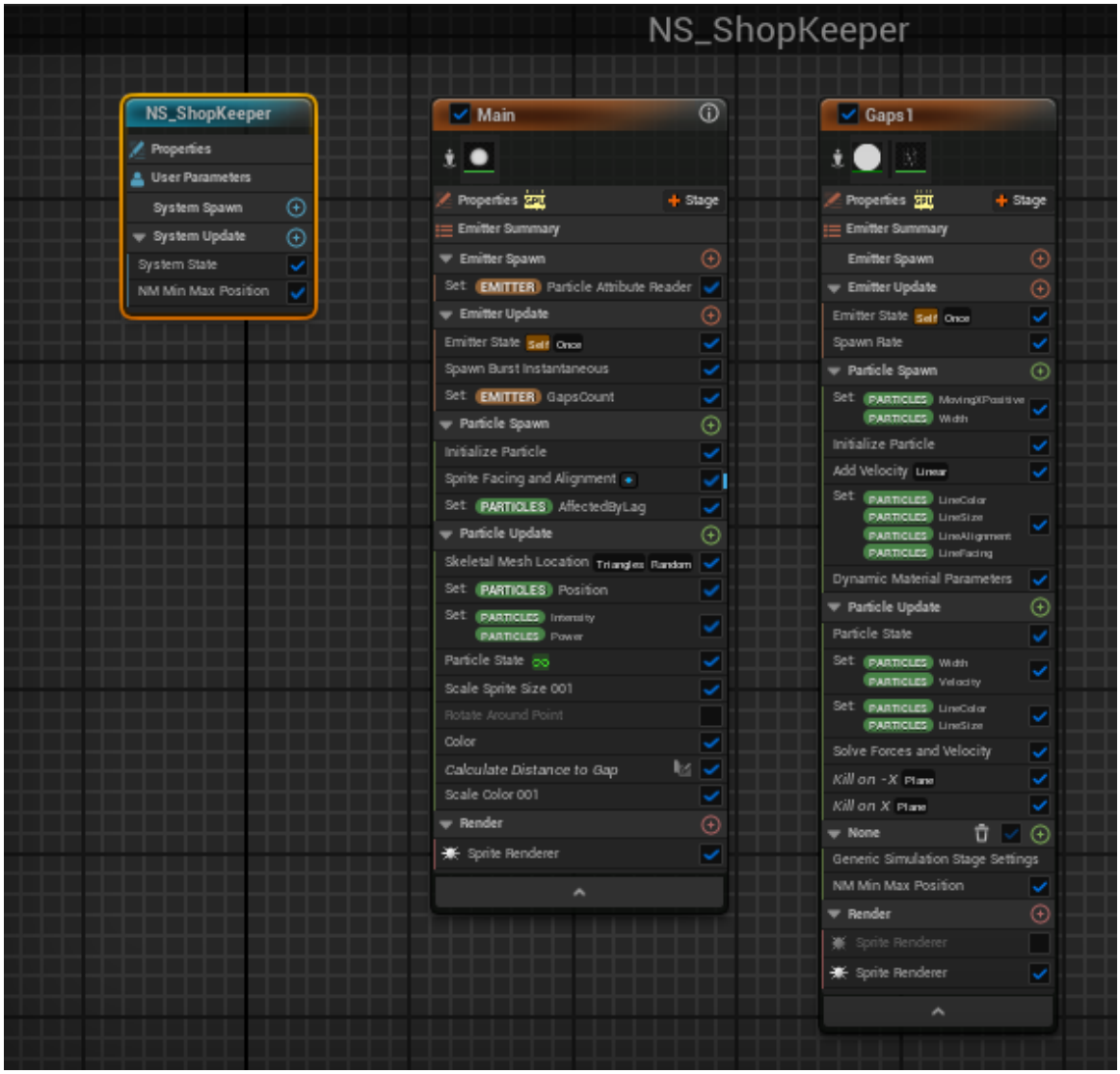


Рисунок 4.10 – Скрипт системи Niagara для торгівця

Матеріал збоїв використовую знайдену текстуру цифрового шуму, яка розмивається іншою схожою текстурою та на які накладається рух та тайлінг на координати UV. Потрібні для цього параметри передаються через відповідний емітер, що дозволяє рандомізувати початкові значення для кожного збою при його породженні.

Для створення ефекту отруйної хмари було використано дві окремих системи. Перша кріпиться до сокету на скелеті Раптора біля пащі та формує стадію «очікування» атаки. Далі перед ворогом створюється друга система, яка прикріплена до актора. Вона створює хмару яка розростається до заданого радіусу за заданий час. У той же час, актор хмари розширює циліндричну фігуру, яка фіксує гравців у зоні ураження (див. рис. 4.11).

Рисунок 4.11 – Ефект отруйної хмари з циліндром фіксування гравців

Також для цього ефекту був створений окремий модуль для створення елементу загасання хмари. Оскільки кожна частка би мала різний час життя залежно від часу її породження, зробити звичайну інтерполяцію часу її життя для зміни прозорості не вдалося би. Тому модуль загасання перевіряє стан виконання системи на деактивованість, у разі якої починається відлік згасання, який обраховую нормалізований час згасання та визначає чи повинна ще жити частинка.

Останнім ефектом для створення був ефект вогню (див. рис. 4.12). Він складається з кількох емітерів, які породжують основу полом'я, його верхню частину яка постійно піднімається, два види іскор та дим. За рахунок обмеження

кількості частинок вогню та диму вдалося отримати оптимальну складність шейдерів (див. рис. 4.13).

Рисунок 4.12 – Ефект полум'я при пожежі

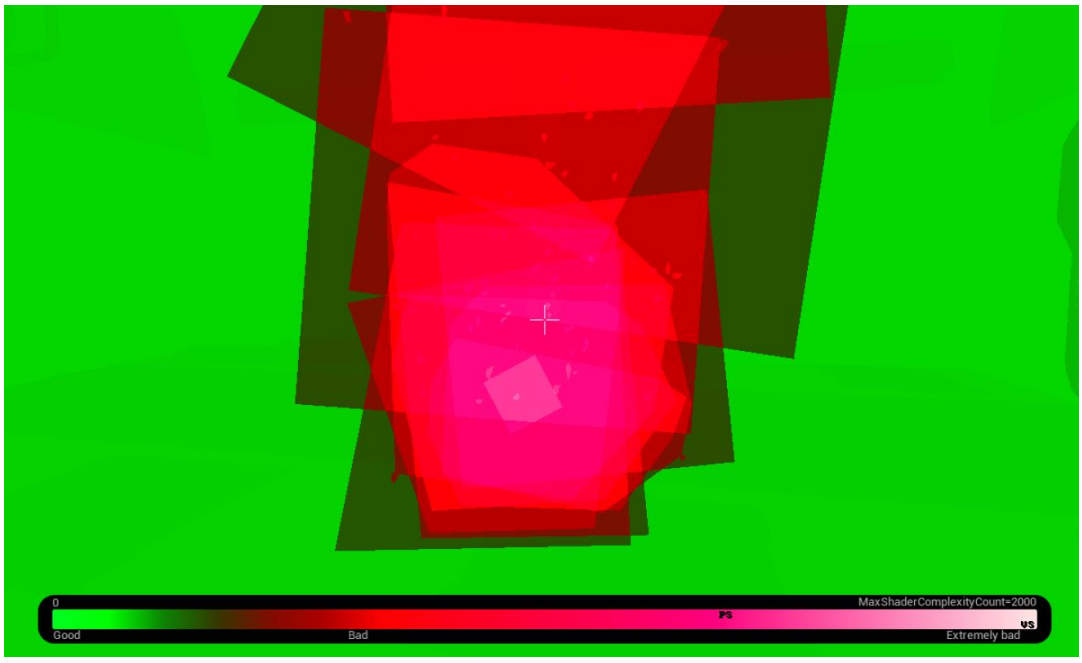


Рисунок 4.13 – Складність шейдерів ефекту вогню

Основною проблемою оптимізації вогню була їх кількість. Навіть враховуючи, що за рахунок фіксованих границь, вогонь який знаходиться не в полі

зору гравця не рендериться на GPU, час на обробку частинок процесом рендеру на CPU залишається майже без змін. Для вирішення цього моменту був застосований профіль масштабування, який прикріплений до цілої системи. Він заморожує оновлення систем на базі часу останнього виклику рендера, при цьому перевіряючи потрібність в розморожуванні кожен кадр. Таким чином фактор замороження системи звівся до перевірки потрібності в рендері на базі фіксованих границь, що не створює додаткових навантажень кожен фрейм.

З іншого боку, через передчасну заморозку всього вогню, він не встигає початися, тому коли гравець вперше повертається до такого вогню, він бачить поступове збільшення вогню, незалежно від того, як довго він був на сцені. Ця проблема була вирішена шляхом програмного прискорення систем, тобто при розморозці системи обробляється декількома ітераціями, якби її стан оновлювався і до цього до повного збільшення вогню.

Таким чином, було створено та оптимізовано різноманітні візуальні ефекти, при появі яких час обробки кадру може підвищуватися всього на кілька мілісекунд.

4.2 Створення ворогів

4.2.1 Налаштування анімацій

Поршочергово при імпорті асетів ворогів та їх анімацій була виявлена негаразда, яка полягала в тому, що анімації Прибульця та Раптора не використовували «рух коренів», при цьому анімація мала накладене переміщення на скелет. Це викликало проблему, що скелет рухається відповідно анімації, але капсула, яка прикріплена до ворога та реєструє колізії, не рухалася разом з моделлю скелета. Це пояснюється тим, що капсула не знає місця основи, або кореня скелета, від якої виконувати переміщення та який би був розташований відносно основи самої капсули.

Для вирішення проблеми було два варіанти, вилучити з анімації переміщення та накладати його через код, або виправити скелет, додавши кореневу кісту та перемістивши дані переміщення на неї, щоб дозволило використати рух коренів. Був обраний другий варіант, адже він більш гнучкий під час розробки, бо дозволяє

вимикати та вимикати рух з кореня динамічно. Тому для анімацій пересування, рух буде вилучений та накладений вручну, що дозволить легко керувати швидкістю ворога через змінні швидкості. У той же час для анімацій атак, які не передбачають одночасного переслідування персонажа, буде використаний рух кореня, адже такі анімації зазвичай мають певні ривки при замаху, для яких складно визначати зсув ворога через код.

Не відходячи від теми анімацій, був створений інтерфейс з функцією виключення руху кореня, яка буде викликатися на графі анімацій, щоб виконати відповідну дію та перемкнутися між режимом змішування анімацій. Загальний вид графу анімацій для всіх ворогів можна побачити на рисунку 4.14.

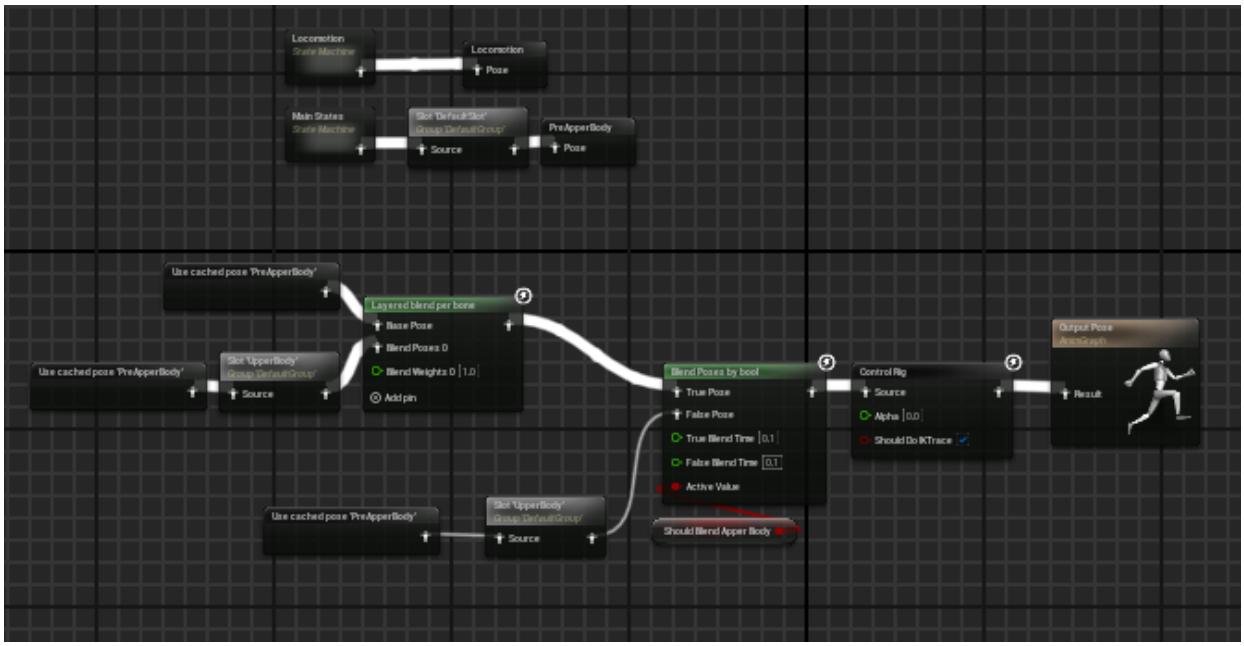


Рисунок 4.14 – Граф анімацій ворогів

Спочатку проводиться кешування основної пози ворога, яка залежить від його поточної швидкості на компоненті переміщення. Далі вона змішується на базі маски змішування з анімацією, яка виконується на гнізді верхнього тіла. Тобто це ті анімації, з яких ми хочемо вилучити трансформацію кісток тільки з верхніх кінцівок, наприклад при атаці рукою. Маска змішування при цьому визначає який відсоток трансформації застосовується на кожну окрему кість з анімації верхнього тіла та поточної основної пози. Наприкінці на базі булевого значення обирається

або попередня поза, або рух анімації на верхніх кінцівках, залежно від того, чи хочемо ми використати рух кореня для поточної анімації.

Виклик зміни стану використання руху кореня ініціюється на рівні анімацій. Це дозволяє гнучно визначати моменти, коли потрібно перемкнутися. Наприклад, для легкої анімації атаки Прибульця створена прогалина, де він може вільно рухатися поки виконує замах перші 6 кадрів (див. рис. 4.15). Це було досягнено за допомоги сповіщень анімацій, які були використані і для інших систем, такі як трасування атак та переривання їх анімації.

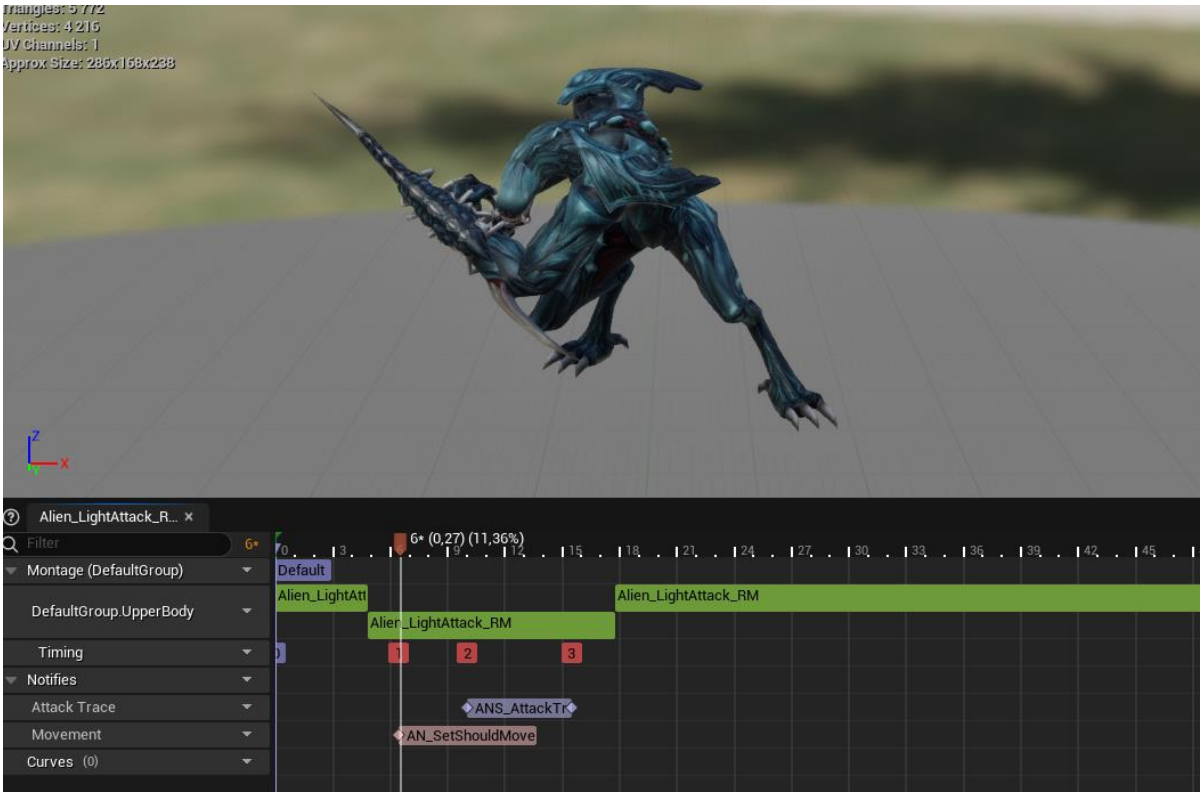


Рисунок 4.15 – Монтаж анімації легкої атаки Прибульця

4.2.2 Компонента трасування атак

Для трасування атак була створена окрема компонента. При додаванні нової атаки для трасування в неї передається дані щодо двох точок між якими проводиться трасування та розмір куба, який створюється між цих точками. При цьому данні зберігаються в окремих статичних таблицях для кожного ворога (див.

рис. 4.16). Варто помітити, що точки трасування визначені за допомоги сокетів на самому скелеті і таблиця включає тільки їх назви.

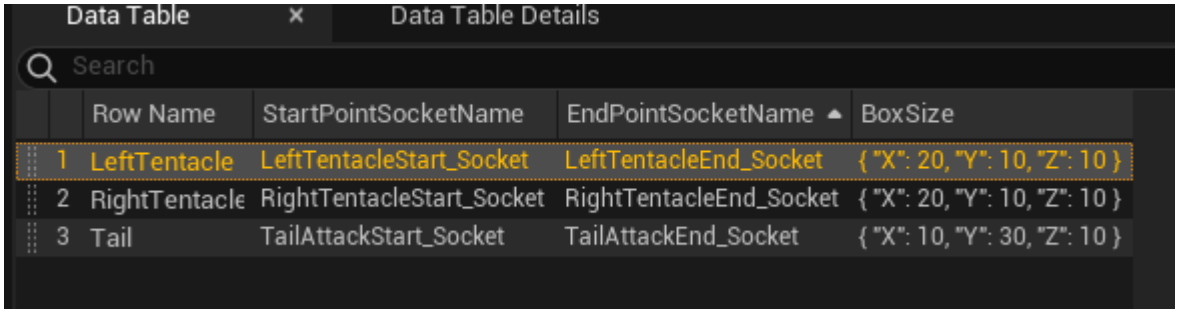


Рисунок 4.16 – Таблиця для трасувань атак Раптора

Компонента працює наступним чином:

– початок повідомлення стану анімації передає дані з рядку таблиці по якому треба виконати трасування атаки, а також індекс удару;
– компонента додає до масиву унікальні значення трасування;
– якщо це перші додані дані, починається таймер трасування;
– при спрацюванні таймера, виконується прохід по всім точкам та виконується трасування кубом відповідних розмірів між точками, які були отримані по назві сокету з компоненти сцени володаря (у випадку ворога це компонента скелету), яка отримується через інтерфейс;
– усі нові виявлені актори по заданому каналі колізій додаються у кінцевий масив відповідно індексу удару та викликаються події з диспатчеру подій передаючи кожного актора та індекс удару;
– кінець повідомлення стану анімації посилає подію закінчення трасування на відповідний індекс удару;
– відповідний удар видаляється з масиву ударів, очищується масив акторів по яким потрапив удар, та якщо більше атак немає, таймер зупиняється.

Детальний код компоненти наведено в додатку Д.

Таким чином, володар компоненти може підписатися на подію пошкодження актора при трасуванні, щоб надалі накласти на нього пошкодження відповідно

поточній атаці. Для цього використовується назва атаки, щоб отримати з таблиці атак її характеристики, та індекс удару для визначення сили пошкодження.

4.2.3 Стани ворогів

Основна логіка станів від якої залежить поведінка ворога знаходиться в контролері AI. По перше, тут додана компонента чуттів. Вона дозволяє налаштувати конфігурацію для кожного чуття – зору, слуху, пошкодження, та дотику (див. рис. 4.17). Відповідні події генеруються гравцем, зброєю, навколишнім середовищем і т.д.

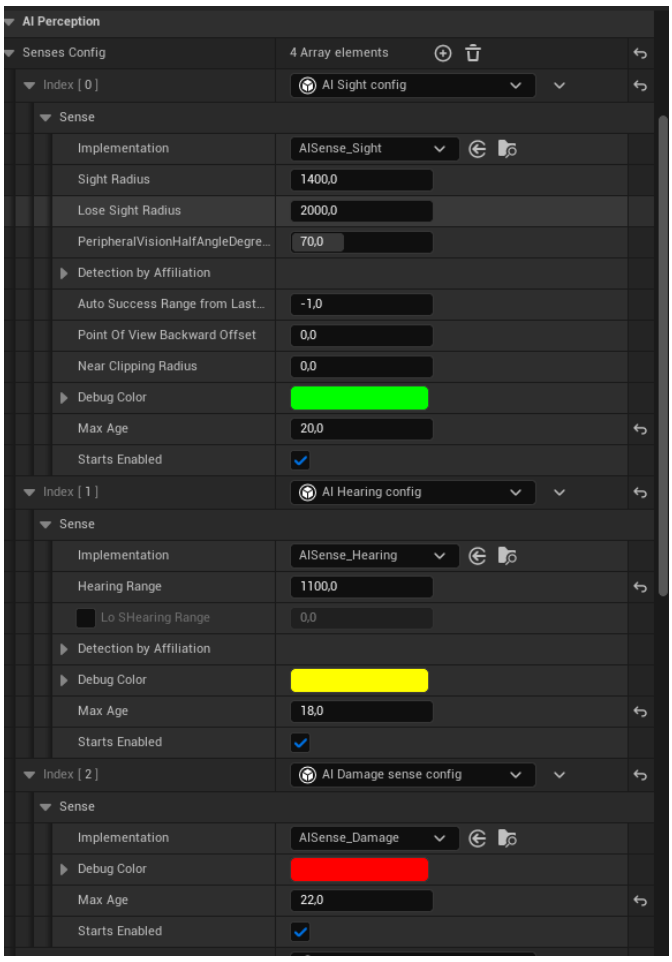


Рисунок 4.17 – Налаштування компоненти чуттів

Коли виникає чи забувається чуття, викликається подія обробки, і відповідно до чуття та поточного стану ворог реагує по різному. Всього є 7 станів: сутичка з гравцем, пасивний, пошук гравця, розслідування, смерть, атака модуля та знищення двері. Залежно від обраного стану застосовується відповідна вітка дерева поведінки кожного ворога (див. рис. 4.18). Стани які включають виконання атак

напряму розгортаються в основному дереві, т.я. атаки специфічні до кожного ворога. У той де час інші події мають свої піддерева, які спільні для всіх ворогів.

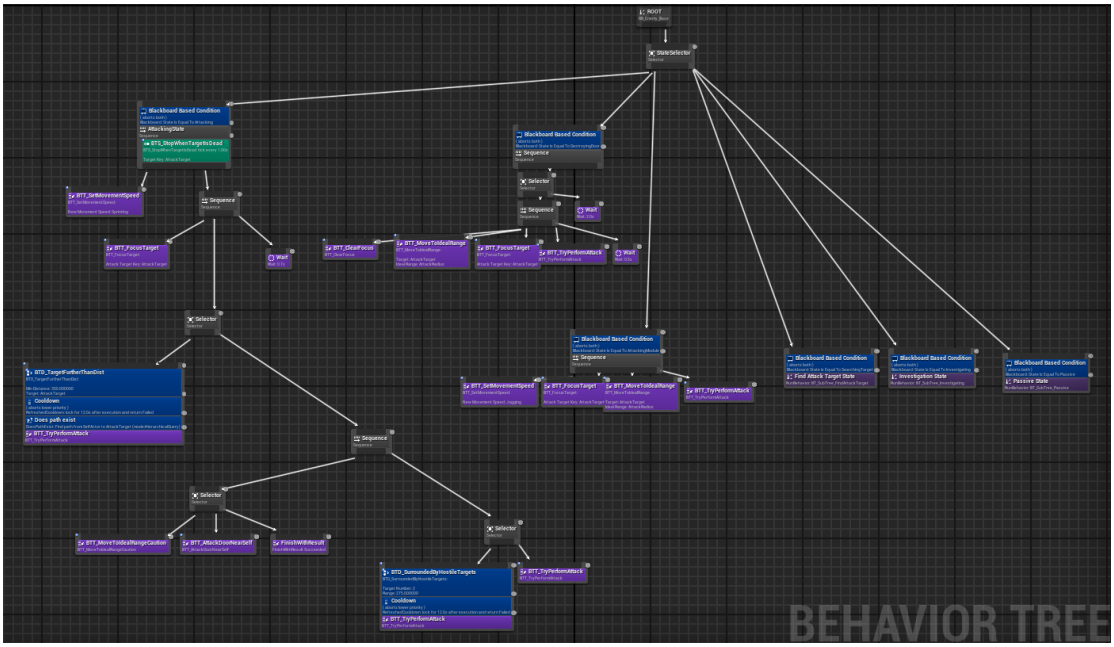


Рисунок 4.18 – Дерево поведінки Прибульця

Основний стан це сутичка з гравцем. Поки ворог у цьому стані, він не покине його, поки не загубить саму ціль чи не помре. При цьому через деякі інтервали часу проводиться перевірка на найближчого персонажа у полі зору, щоб мати можливість на більш доцільну атаку. Ця ж методика проводиться коли гравець вперше потрапляє в поле зору.

Коли ворог доходить до цілі, він обирає найбільш доцільну атаку та намагається її виконати попередньо забронювавши токени. Вибір атаки залежить від арсеналу ворога, кількості цілей, які його оточують, дистанції. Так, Прибулець з деякими інтервалами стрибає на гравця, якщо він в радіусі 350 метрів та до нього існує шлях.

Бронювання токенів для початку обраної атаки виконується шляхом запиту через інтерфейс на бронювання. Якщо бронювання успішне – ворог починає атаку (див. рис. 4.19). Сама атака реплікується на клієнти та включає програвання анімації, переслідування та повернення токенів у разі її переривання чи кінця.

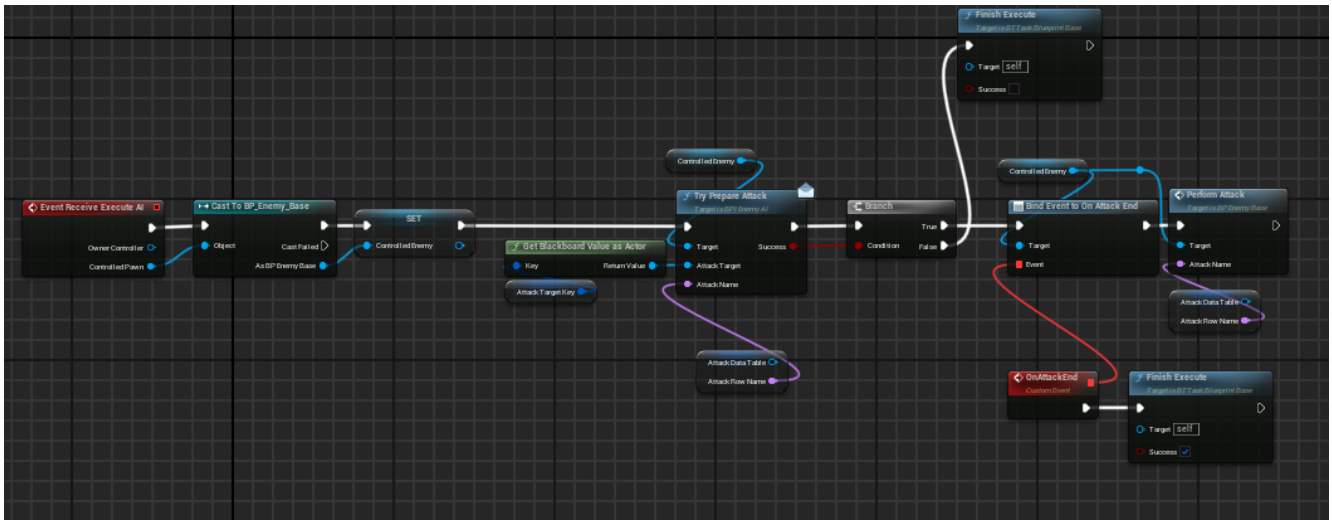


Рисунок 4.19 – Вузел завдання дерева на спробу виконати атаку.

Деякі специфічні атаки окрім перевірки колізій потребують особливої обробки, наприклад, отруйна хмара Раптора, чи масовий удар Прибульця. Для цього у відповідному класі ворога йде підписка на подію отримання повідомлення анімації. Так, при масовому ударі створюється окремий актор для реєстрації масових пошкоджень, який має подію перетину актора заданого типу об'єктів, на яку підписується ворог, щоб надалі відкинути цілі (див. рис. 4.20).

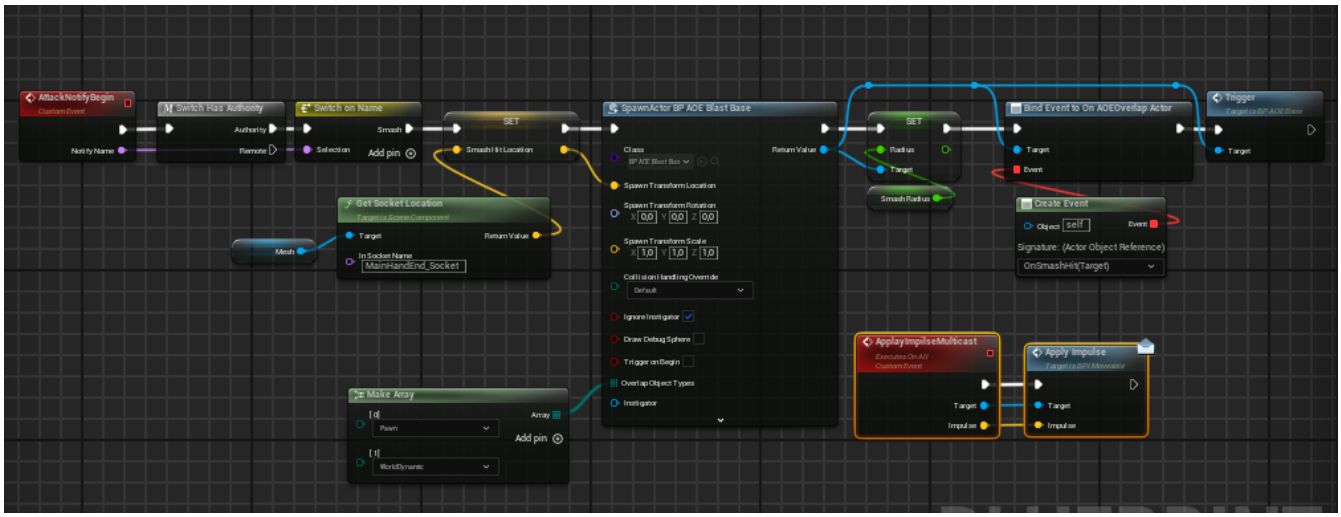


Рисунок 4.20 – Подія створення масового удару Прибульця

Для стану атаки Кракену був створений окремий вузел для переміщення (див. рис. 4.21). Він базується на тому що ворог намагається оминути гравця, обираючи точку переміщення через запити середовища. У той же час, він прослуховує події

звуку з контролера, щоб у разі стрільби початки ухиляння. Оскільки Кракен мобільний, він швидко зміщується у випадкові сторони поряд, поки стрілянина продовжується.

Рисунок 4.21 – Вузел завдання дерева на переміщення за ухиляннням

В пасивному стані ворог досліджує поточну локацію, блукаючи та повертаючись вільні місця.

Пошук гравця виникає, коли той сприймається одним з чуттів, при цьому ворог не у стані сутички. Так, якщо ворог побачив гравця, то він перейде до стану сутички. Якщо він сприйняв гравця іншим чуттям, наприклад, через отримане пошкодження, він перейде у стан розслідування, проводячи пошук місця зі сторони якого був надісланий удар. У той же час, якщо ворог буде у стані знищення двері за якою знаходиться інший гравець, він не буде цікавитися звуками здалеку. Таким чином, ворог адаптується до ситуації в нього є пріоритети.

Стан пошуку базується на дослідженні місцевості, поряд з останньою локацію, де був побачений гравець. Воно продемонстровано деревом на рисунку 4.22.

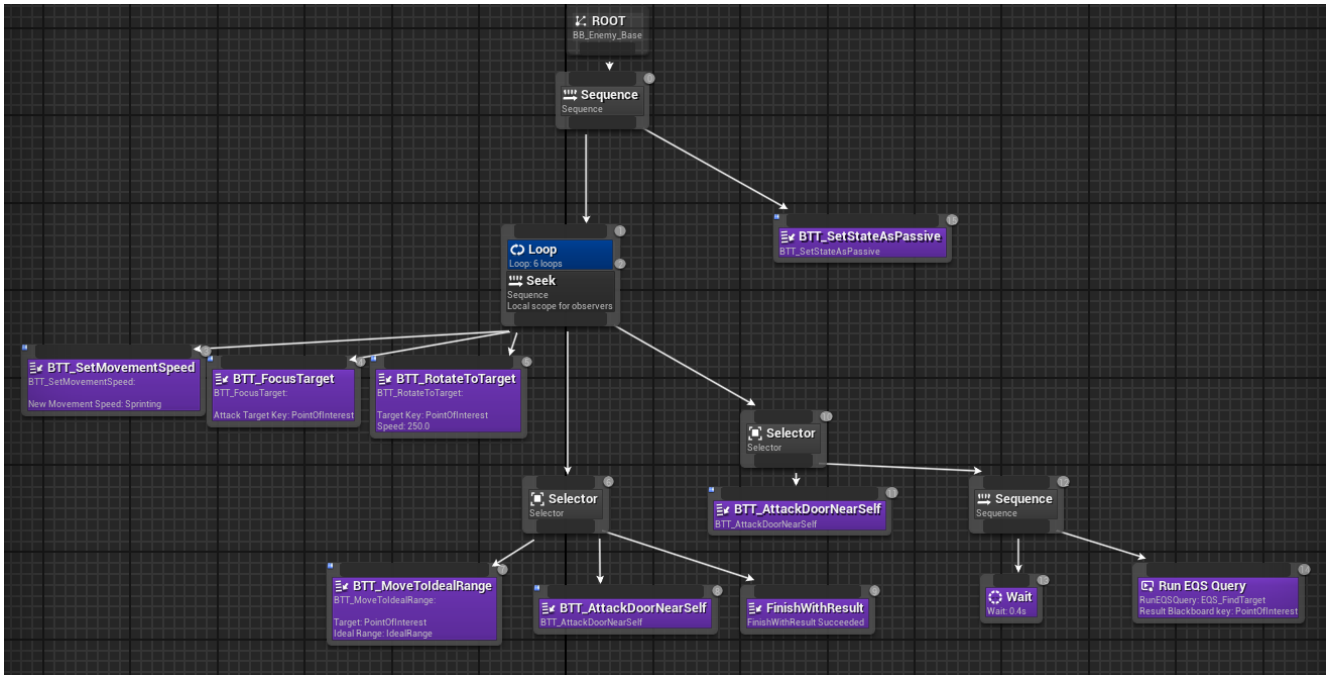


Рисунок 4.22 – Дерево пошуку гравця

Спочатку ворогу виставляється стан бігу для пересування, він фокусується на точці інтересу та починає повертатися до неї. Далі він намагається дійти до цієї точки, у разі невдачі, намагається знайти двері біля себе, щоб перейти у стан їх знищення. Якщо ж ворог доходить до точки і біля неї нема двері, через мить очікування він намагається знайти нову точку інтересу. Пошук цієї точки виконується за допомоги запитів навколишнього середовища. Був створений запит, який намагається обрати нову точку пошуку, на базі напрямку ворога та перешкод (див. рис. 4.23). Т.я. ворог не може бачити через стіни, місця які загороджені місцевістю, мають більшу перевагу при виборі.

Так проходить декілька ітерацій, у разі невдачі яких, ворог переходить по пасивного стану.

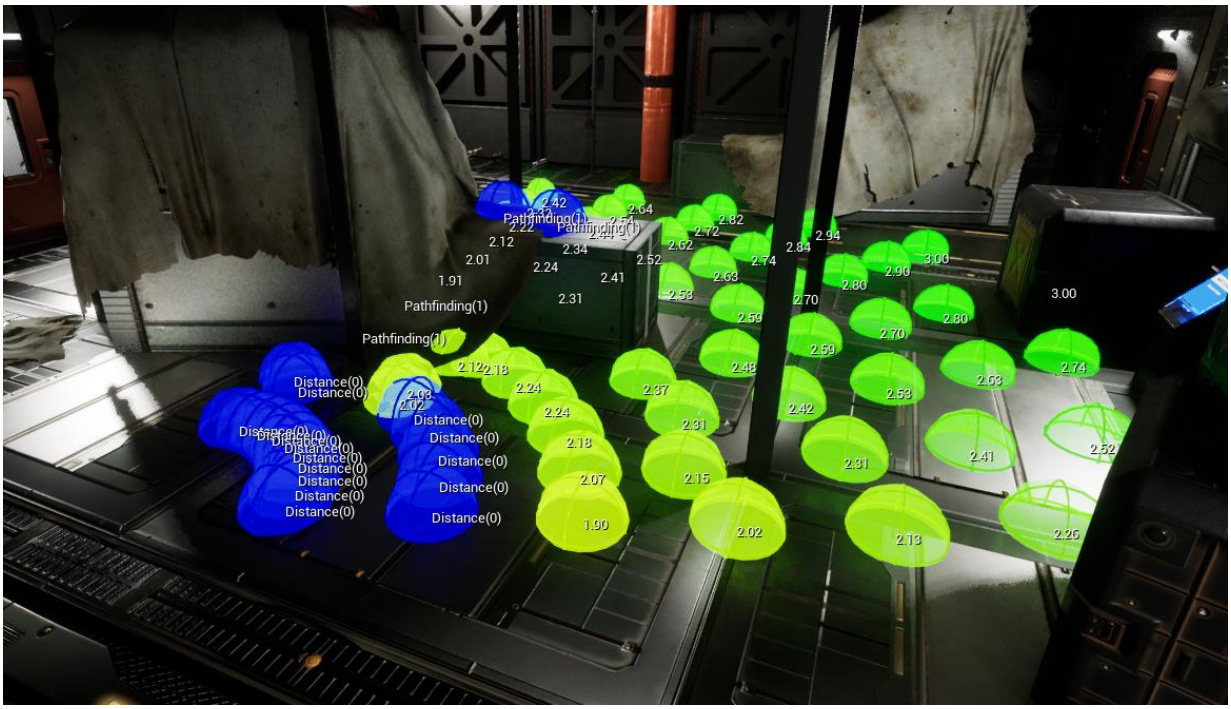


Рисунок 4.23 – Запит середовища на нову точку пошуку

Стан дослідження виникає при прийнятті звуків, пошкодження чи дотику коли ворог пасивний. Він дещо схожий на пошук, але ворог вже не так цікавиться дверима та шукає більш далекі та відкриті місцевості для дослідження в ширшому полі зору (див. рис. 4.24).

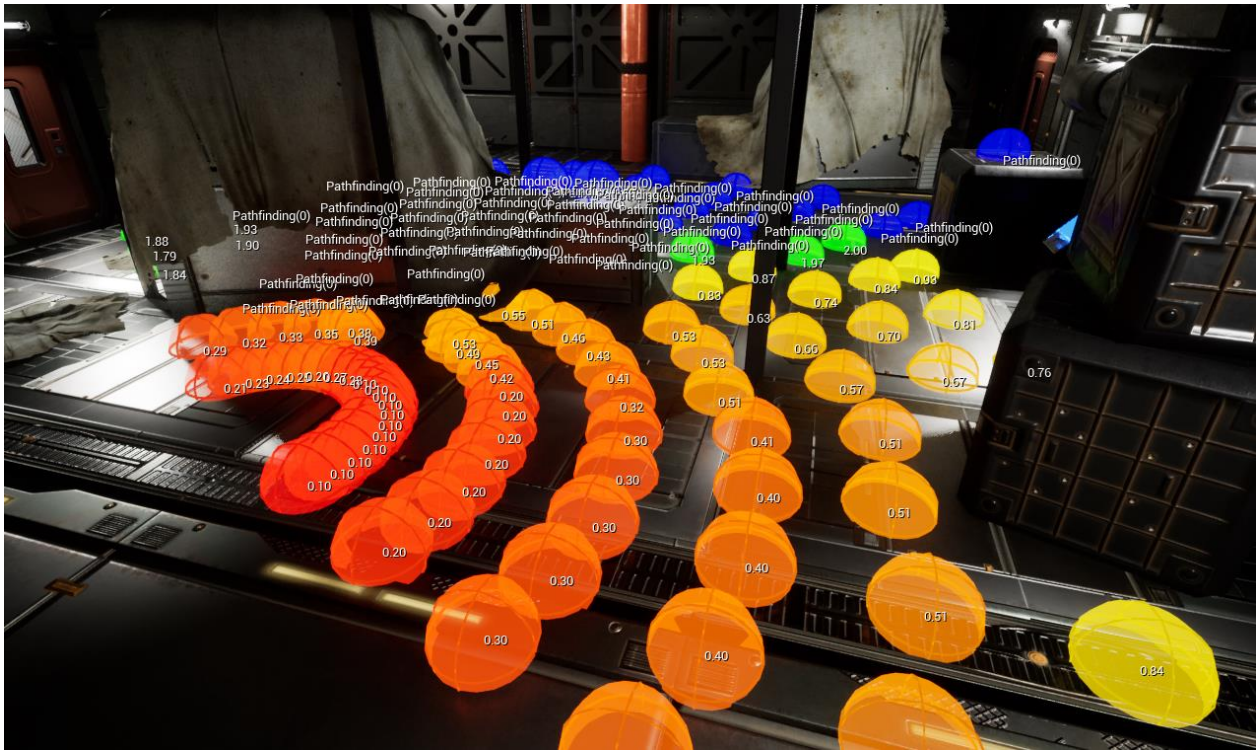


Рисунок 4.24 – Запит середовища на нову точку дослідження

При смерті ворог перериває поточну анімацію, грає анімацію загибелі, повертає усі токени гравцям та модулям і знищується.

Стан атаки модуля слугує для підбору більш доцільних атак до статичної цілі та для створення пріоритету. Тобто ворог вже не реагує на звуки в цьому стані, але при отриманні інших відчуттів, переключається на гравця.

Стан знищення дверей слугує для поламки дверей на мостиках корабля. Він обумовлений тим фактом, що мостики включають завжди дві двері, тому ворог завжди повинен переходити до другої, у разі пошуку гравця.

Таким чином, шляхом керування станом ворога через сприйняття чуттів та вибір дій через дерева пошуку, вдалося створити гнучку систему, яку можна підлаштовувати під певні типи ворогів. Впровадження сповіщень через анімації, дозволило зручно керувати станом атак і налаштовувати їх на конкретні проміжки часу анімації.

4.3 Створення UI

Перед побудовою ігрових меню було розроблено окрему компоненту для моніторингу пристрою вводу (див. рис. 4.25). Система слідкує за останнім використаним пристроєм, та повідомляє через диспетчер подій про їх зміну, щоб усі віджети адаптувалися під пристрій. Зокрема, компонента надає картинки та текст відповідно натиснутій кнопці пристрою.

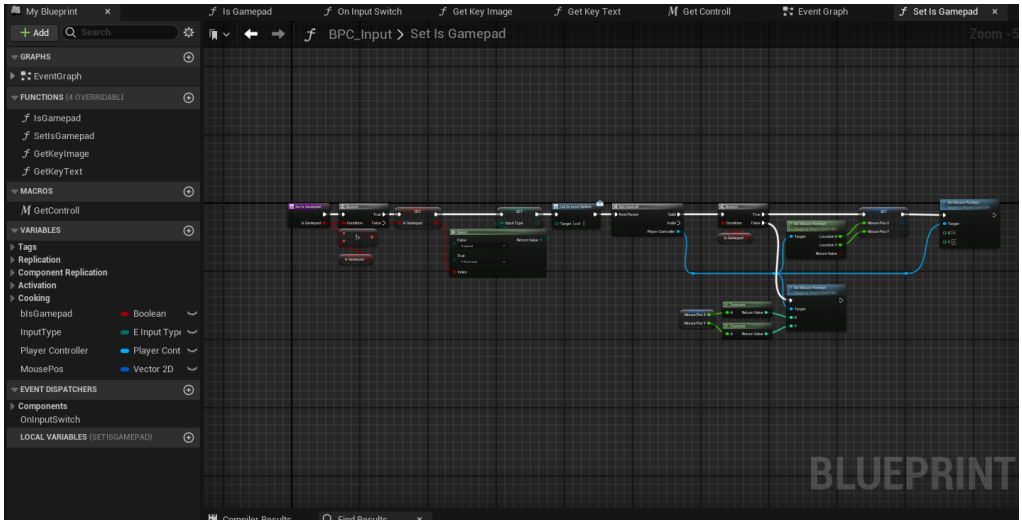


Рисунок 4.25 – Метод реакції на ввід корстувача

Для головного меню була створена система станів, між якими виконується переключення. Це дозволило розташувати усі підменю в головному меню (див. рис. 4.26). Коли гравець взаємодіє з кнопкою підтвердження, залежно від поточного стану ховається попереднє меню, та ініціалізується нове.

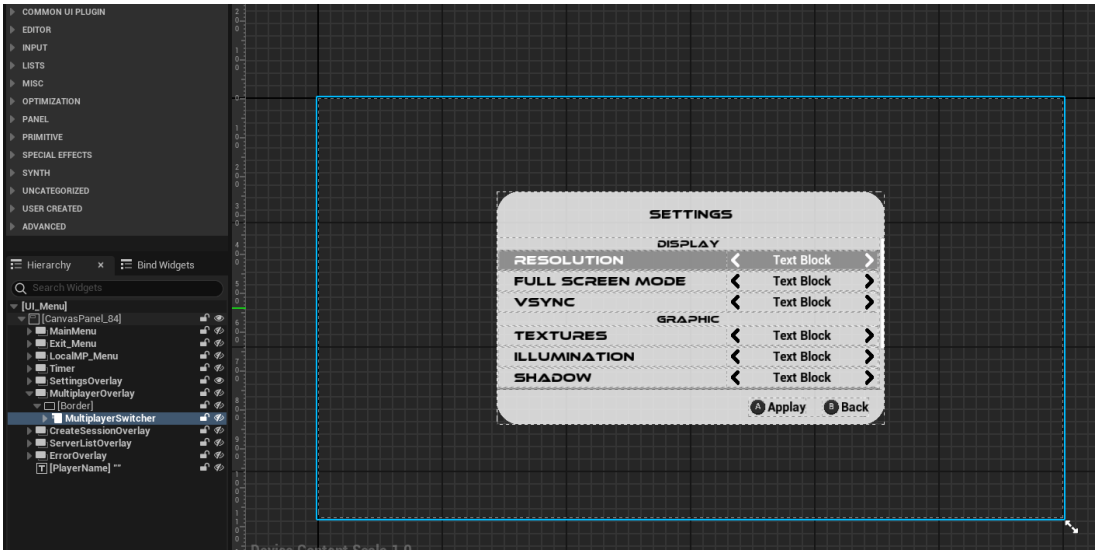


Рисунок 4.26 – Віджет головного меню з його ієрархією

Усі події вводу обробляються в контролері гравця відповідного за головне меню. Події вводу представлені окремими асетами, які групуються залежно від меню та перевикористовуються. До них належать:

– IA_Up – відповідає за прогортання поточного елементу меню вгору;
– IA_Down – відповідає за прогортання поточного елементу меню вниз;
– IA_Left – відповідає за прогортання поточного елементу меню вліво;
– IA_Right – відповідає за прогортання поточного елементу меню вправо;
– IA_Confirm – відповідає за підтвердження дії чи взаємодії з поточним елементом меню;
– IA_Cancel – відповідає за відміну дії чи повернення до попереднього меню;
– IA_Exit – відповідає за вихід з меню та початок кооперативної партії;
– IA_Refresh – відповідає за оновлення асинхронних списків.

Схеми управління представлені на рисунках 4.27-4.28. При їх створенні була задача створити доступне керування, яке виправдовує очікування та наміри гравця.

Адже управління повинно бути простим у вивченні та використанні, створюючи для гравця повний контроль. Саме тому схема взаємодії з клавіатури має різні способи контролю. Наприклад, гравець може використати клавіші WASD чи стрілки у головному меню, адже біля них знаходиться ліва рука. У той же час, коли проходить партія, для навігації в магазині можна використати мишу, яка заблокована для повороту камери персонажа під час закупівлі.

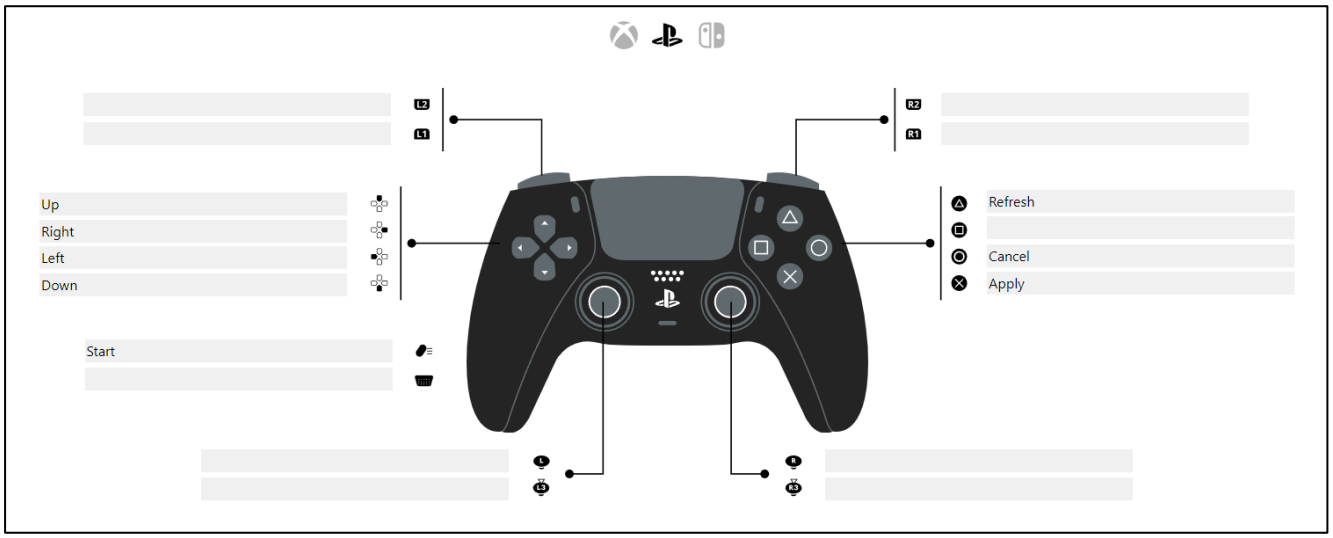


Рисунок 4.27 – Схема управління меню з геймпаду

Рисунок 4.28 – Схема управління меню з клавіатури та миші

Порівняно з макетом головним меню було змінено, щоб розділити локальний мультиплеєр з роздільним екраном та онлайн мультиплеєр (див. рис. 4.27). Також була додана кнопка для титрів, щоб загадати розробників при публікації гри в інтернеті, а також згадати авторів усіх асетів з вільними ліцензіями, які були використані.

Також замість статичних картинок меню було створено окремі маленькі сцени космічного корабля, рендер яких проводиться на задньому фоні. У свою чергу шрифт меню був змінений, для єднання користувацького інтерфейсу з естетикою наукової фантастики.

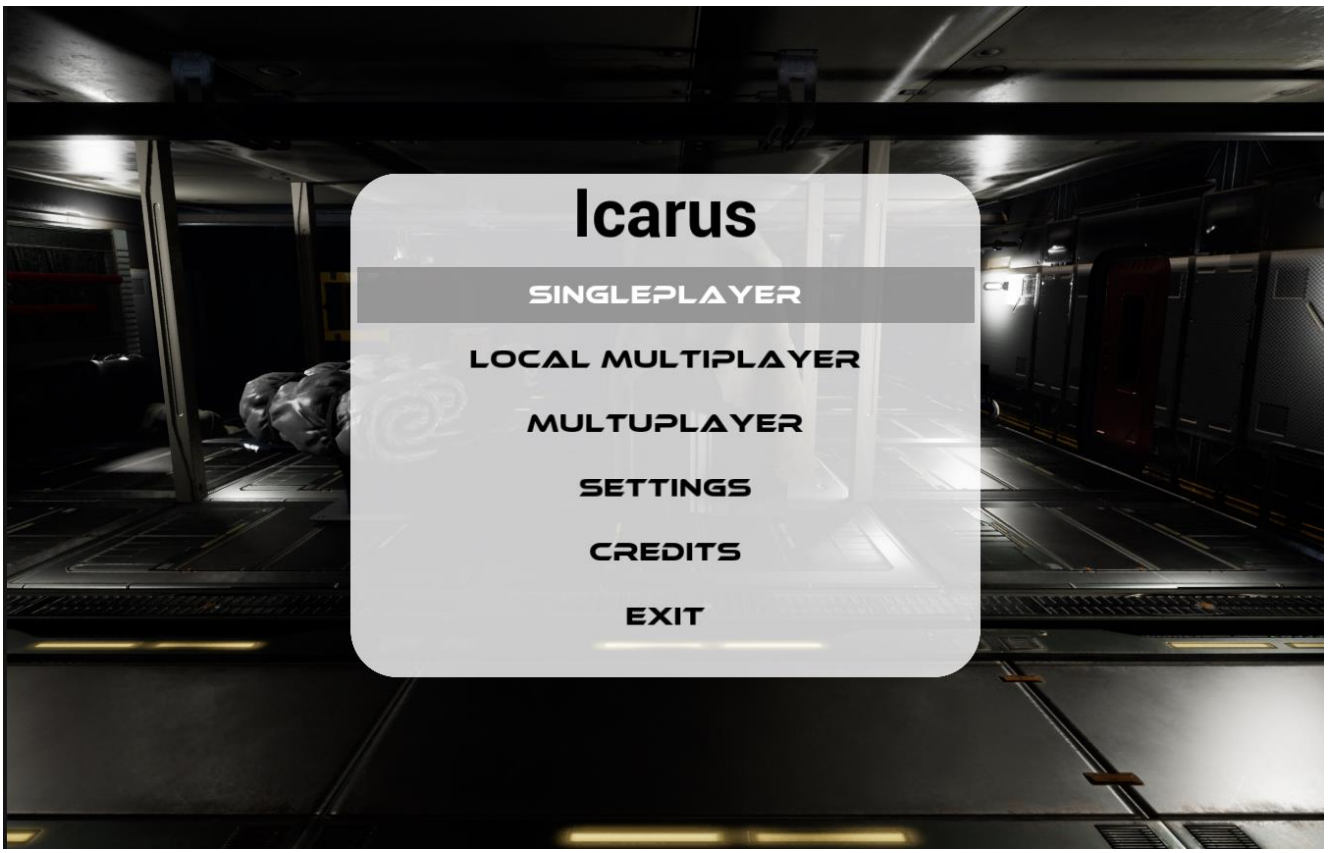


Рисунок 4.27 – Оновлене головне меню

До меню налаштування була додана опція вертикальної синхронізації, щоб уникнути розірвань картинки при швидкому переміщенні камери (див.рис. 4.28).

Рисунок 4.27 – Оновлене головне меню

Меню паузи також зазнало змін. Щоб звільнити видимий простір для основного ігрового процесу, воно було зменшене та пересунуте ліворуч (див. рис. 4.28).

Рисунок 4.28 – Оновлене меню паузи

Для товарів магазину було створено обкладинки у Photoshop, які демонструють предмет, без потреби читати назву. Загальний вигляд магазину представлено на рисунку 4.29.

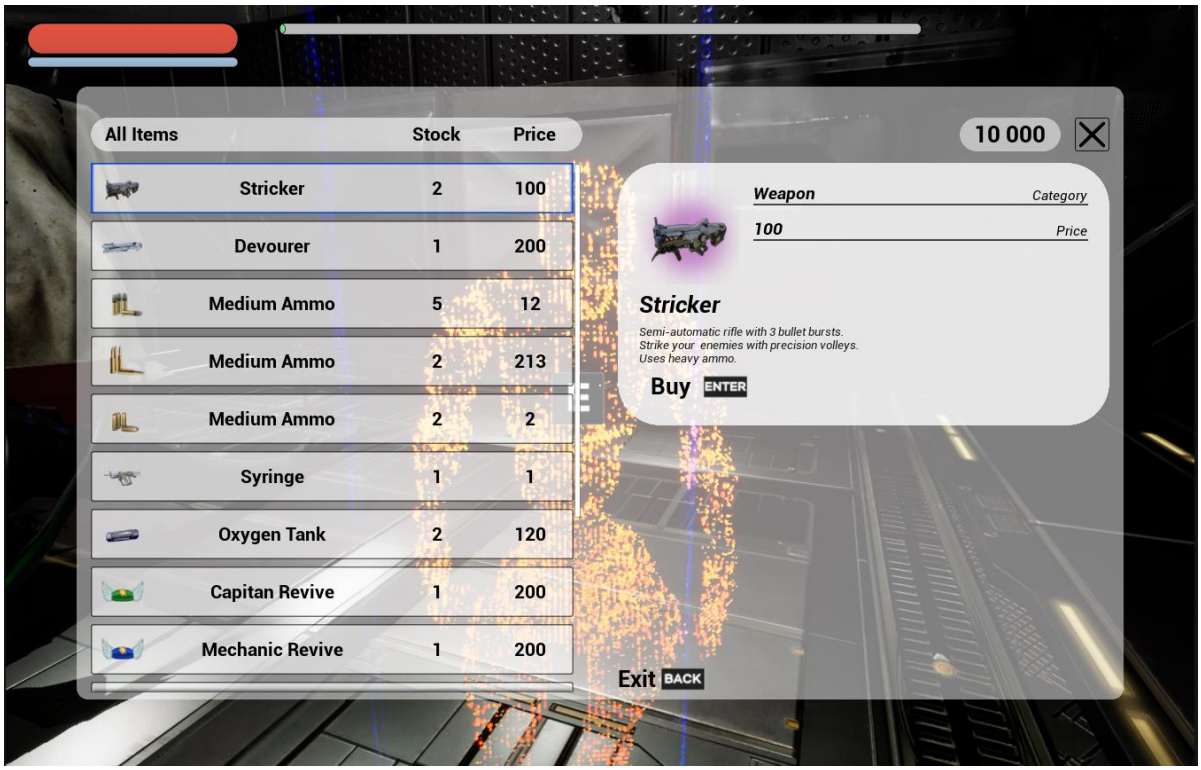


Рисунок 4.28 – Вікно магазину

Дані про ціну, опис, категорію і т.д. підтягуються з окремої таблиці, що дозволяє імпортувати її одразу з excel після балансування.

Магазин оновлює свій запас між рівнями через актора ігрового режиму. Там же виконується і додавання товарів по відродженню гравців. Оскільки ігровий режим знаходиться на сервері, це дозволяє легко оновити магазин, та сповідомити клієнти про новий асортимент з початку рівня.

Unreal Engine не має можливості змінювати роздільну здатність віджетів в режимі роздільного екрану, тому для всіх елементів магазину, які потребували масштабування, були застосовані «розмірні коробки». Коли кількість гравців змінюється, контролер сповіщує усі віджеті про режим роботи, який базується на кількості гравців. Далі в кожному віджеті можна окремо налаштовувати розмірність елементів відповідно до розміру частини екрану гравця. Так, деякі коробки змінюють тип розтягування масштабуючи елементи під свій розмір, а проміжки між блоками елементів зменшуються.

Рисунок 4.28 – Вікно магазину в режимі гри роздільного екрану на трьох

Таким чином, на базі спроектованих макетів були побудовані усі ігрові меню, які гарно працюють з різними пристроями вводу та адаптивні до режиму роздільного екрану, що в цілому надає доброзичливий досвід користувачеві.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розробка мапи думок тестування

Перед створенням та виконанням тестів була розроблена мапа думок ігрового застосунку для структурування інформації (див. рис. 5.1). Мапа дозволяє чітко окреслити цілі тестування, гарантуючи, що всі тести спрямовані на досягнення цих цілей. Крім того вона дозволяє організувати виявлені помилки за типом, серйозністю або іншими критеріями, що полегшує їх аналіз та пріоритетизацію.

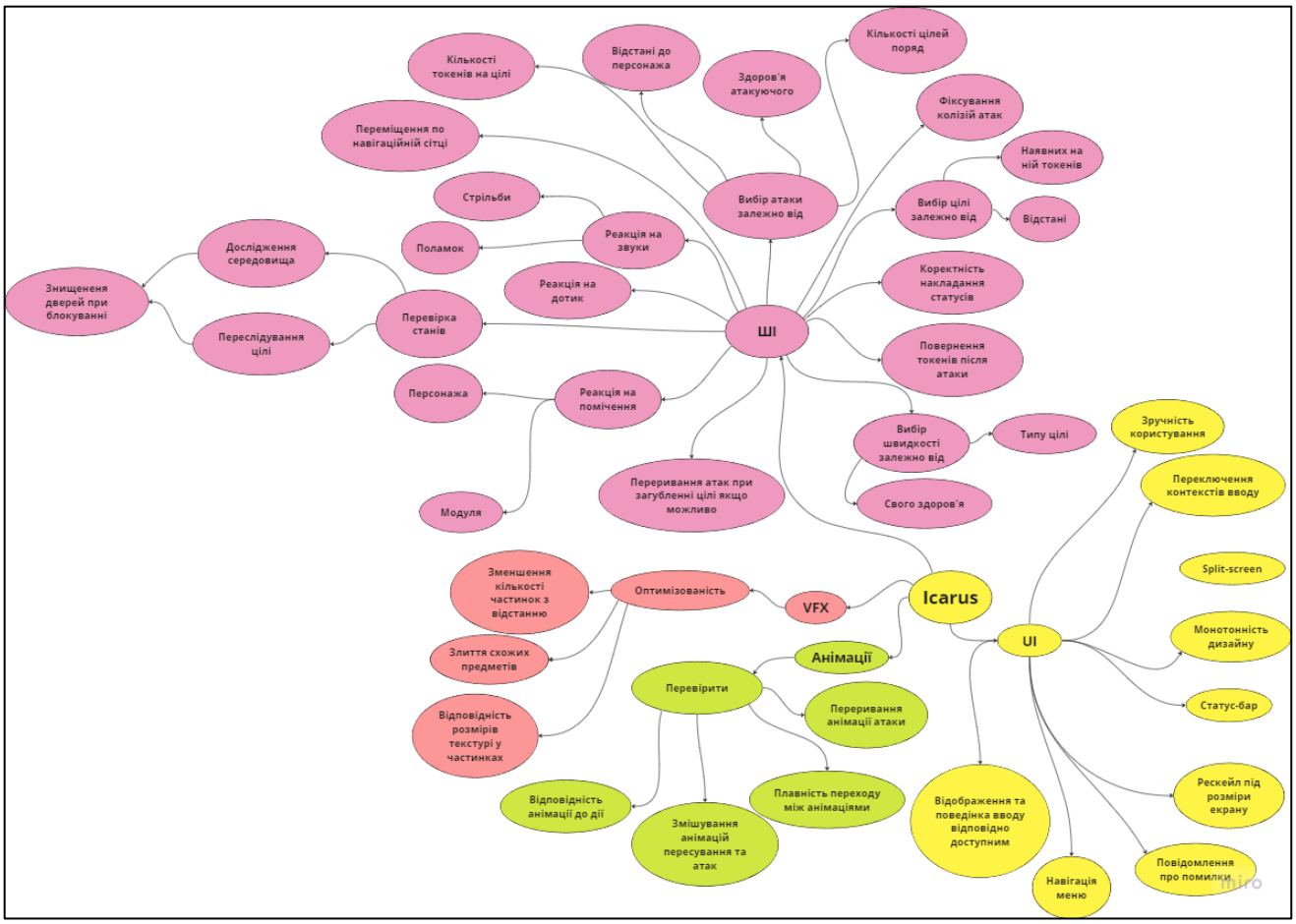


Рисунок 5.1 – Мапа думок проекту

Мапа виділяє 4 окремі структурних елемента – штучний інтелект, візуальні ефекти, анімації та інтерфейс користувача.

Основним елементом є штучний інтелект, який включає реакції на різноманітні звуки, дотик, зір; перевірка станів ворога та відповідної реакції на цілі та середовище, підбір атаки та цілі залежно від факторів, як відстань чи наявність

токенів. Метод «білої скрині» був використаний під час тестування воргів, для перевірка правильності роботи програми шляхом аналізу її внутрішньої логіки та структури коду.

З іншої сторони перевірка інших структурних елементів, хоч і невелика, але зачіпає непов’язані між собою системи, які здебільшого мають візуальне представлення у грі. Інтерфейс користувача включає тестування UX, різних способів вводу, коректність відображення та адаптація до роздільного екрану, перевірку навігації. Анімації потребують розуміння поточного стану ворога та його співвідношення з анімацією. Також сюди входить плавність анімації при їх змішуванні чи перериваннях. VFX потребує перевірки розмірів частинок та їх шарування відповідно розмірів екрану.

Для цих типів елементів тестування використано метод мануального тестування. Для UI є візуальна перевірка елементів інтерфейсу, таких як кнопки, поля введення, меню, звертаючи увагу на їх правильність розташування та відповідність дизайну. Також до нього відноситься під'єдання різних пристроїв вводу для перевірки зміни усіх меню. Крім того потрібно перевірити зручність та оцінити інтуїтивність інтерфейсу. VFX включає мануальне перевірку усіх ефектів через меню складності шейдерів та інструментів профілювання.

5.2 Розробка тестових випадків

За допомоги мапи думок було розроблено тест випадки, які були спроектовані для різних частин систем застосунку. Кожному випадку присвоєно назву, опис, компоненту, якої він стосується, серйозність, пріоритет, кроки відтворення, фактичний та очікуваний результати. Серйозність ділиться на п'ять категорій починаючи з S5 – прості помилки, в не дуже важливих системах та закінчуючи S1 – помилки, які зачіпають основний процес гри та мають найбільший шанс появи у гравця. Окрім серйозності є градація за пріоритетом, від P3 – наймеш пріритетні, які спливають тільки на візуальну складову, до P1 – першочергові для вирішення помилки, які можуть блокувати ігровий процес. Тест випадки наведено в таблиці 5.1.

Таблиця 5.1 – Список тестових випадків

Тест № 1	
Назва тесту:	Атакуючий ворог після смерті гравця
Опис тесту:	Інколи, якщо персонаж помирає, ворог продовжує наносити по ньому атаки, допоки не побачить іншого персонажа
Компонент системи:	Enemy
Пріоритет:	S2
Критичність:	P2
Кроки відтворення:	Дати вмерти персонажу від ворога і за допомоги від'єднання камери відслідкувати його дії
Очікуваний результат:	Ворог повинен розфокусуватися від гравця і почати досліджувати місцевість
Фактичний результат:	Ворог концетрується на померлому гравці і ходить довколо нього
Результат:	Знайдено баг, виправлено 25.04.2024
Тест № 2	
Назва тесту:	Лагодження дверей в відстикований відсік
Опис тесту:	Іноді Raptor телепортується при спуску з другого поверх на перший після знищенні першої двері і переміщенні до другої. При цьому він переходить у стан спокою

Продовження таблиці 5.1

Компонент системи:	Enemy- Raptor
Пріоритет:	S1
Критичність:	P1
Кроки відтворення:	Запустити SpaceShip рівень і слідкувати за Raptor на другому поверсі
Очікуваний результат:	Ворог повинен нормально дійти до двері і почати її ламати
Фактичний результат:	Ворог телепортується перериваючи свій стан знищення двері
Результат:	Знайдено баг, виправлено 25.04.2024
Тест № 3	
Назва тесту:	Налаштування Resolution не скидається
Опис тесту:	Якщо через головне меню спробувати вибрати Resolution, а потім вийти з меню і знову увійти буде встановлене останне значення
Компонент системи:	MainMenu
Пріоритет:	S4
Критичність:	P3
Кроки відтворення:	Через головне меню зайти у налаштування, вибрати нове значення Resolution, вийти з меню налаштувань, зайти знову.

Продовження таблиці 5.1

Очікуваний результат:	Налаштування Resolution повинно залишатися на опції останнього застосованого
Фактичний результат:	Налаштування Resolution залишається на опції останнього обраного
Результат:	Знайдено баг, виправлено 26.04.2024
Тест № 4	
Назва тесту:	Зфокусований ворог після смерті
Опис тесту:	Інколи після смерті ворог фокусується на персонажі (беззупинно повертається до нього) коли програється його анімація смерті
Компонент системи:	Enemy
Пріоритет:	S4
Критичність:	P3
Кроки відтворення:	Вбити ворога та почати рухатися навколо нього
Очікуваний результат:	Ворог повинен залишити ротацію, яка була в нього при смертельному ударі
Фактичний результат:	Ворог починає програвати анімацію смерті, але при цьому завжди повератється до сторони гравця
Результат:	Знайдено баг, виправлено 25.04.2024
Тест № 5	
Назва тесту:	Raptor перестає преслідувати гравця

Кінець таблиці 5.1

Опис тесту:	Інколи коли Raptor переслідує гравця з основного відсіку у склад на мостику, коли гравец переходе на склад через двері, Raptor зупиняє переслідування
Компонент системи:	Enemy- Raptor
Пріоритет:	P1
Критичність:	S1
Кроки відтворення:	Спровокувати Raptor в основному відсіку, сховатися на мостику до складу за першими дверима, дочекатися поламки двері, перебігти у склад
Очікуваний результат:	Ворог повинен переслідувати персонажа
Фактичний результат:	Ворог втрачає інтерес до персонажа
Результат:	Знайдено баг, виправлено 26.04.2024

Інші знайдені помилки, були виправлені на місці. Загалом тестування показало малий показник візуальних дефектів, адже їх були легко виявити під час самої розробки і перевірки відповідних механік чи ефектів. Основні помилки були зосереджені на системі ворогів, через її розміри, пов'язаними з різноманітними станами ворогів та можливістю переривання дій.

Повний тест-план гри наведено у додатку Г.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У межах тематики кваліфікаційної роботи була підготовлена наукова стаття у формі тез для виставки технічної молоді – XXVIII Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті»[20], який проходив з 16 по 18 квітня 2024 року в онлайн-форматі на базі Харківського національного університету радіоелектроніки (ХНУРЕ). Представлена робота була опублікована у шостому випуску збірника матеріалів конференції під заголовком «Масштабування VFX як спосіб оптимізації в Unreal Engine»[21].

Доповідь чітко окреслила проблему оптимізації VFX, яка є ключовою для забезпечення високої якості та продуктивності візуальних ефектів у медіа. Було представлено потужний інструмент - масштабування, який дозволяє гнучко оптимізувати VFX залежно від відстані до них або навантаження на систему. Автор продемонстрував, як використання профілів масштабування у системах частинок Niagara може значно покращити продуктивність без шкоди для візуальної складової.

Наукова публікація окреслює основні системи Unreal Engine від яких залежить продуктивність розробок на двигуні. Зокрема, було оглянуто часті проблеми, які виникають при використанні систем частинок та способи їх вирішення. Робота проводить аналіз метода ефективності використанні профілів масштабування на прикладі реалізації техніки LOD для систем частинок. Тези наведені у додатку Е.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено унікальний кооперативний ігровий застосунок у жанрі космічного симулятора. Розробка базувалася на проведеному аналізі предметної області щодо актуальності теми створення кооперативних ігор. Було проведено порівняння різних моделей реалізації багатокористувацьких кооперативних ігор серед який було знайдені найкращі для реалізації у кваліфікаційний роботі. Також була визначені основні проблеми, які можуть вирішити ігри кооперативного жанру. Для гри була підібрана цільова аудиторія та час ігрової сесії ігрового застосунку.

В процесі роботи було сформовано та висвітлено ідею гри, її основні механіки та особливості. На базі сформованого функціоналу було визначено вимоги до майбутнього продукту. Крім того було обрано та обґрунтовано використання ігрового рушія та додаткового інструментарію, а також визначені основні вимоги до заліза, для запуску ігрового продукту.

В результаті було побудовано модульну на абстраговану архітектуру для системи ворогів, продемонстровану на UML діаграмах. Було визначено підхід для синхронізації дій ворогів та реалізації їхньої штучної поведінки. Також були сформовані вимоги до візуальних ефектів та спроектовані основні екрани інтерфейсу користувача.

Розробка включала створення різноманітних ефектів за допомоги систем частинок, а також використання технік оптимізації, для адаптації гри під більшу кількість пристроїв. Комплексна система ворогів, дозволила надати життя створінням яким протистоять гравці, створюючи для них широких простір для навчання та побудови стратегій. Також був створений футуристичний інтерфейс для ігрового застосунку, який дозволяє зручно взаємодіяти з ігровим процесом та підтримує різні пристрої вводу. Із закінчення розробки було проведено тестування програмного продуту згідно мапі думок сформованої напередодні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. History of video games. URL: https://en.wikipedia.org/wiki/History_of_video_games#:~:text=The%20history%20of%20video%20games,games%20on%20a%20video%20display (Дата звернення: 11.05.2024).

2. Video Games – Worldwide. URL: https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide#key-market-indicators (Дата звернення: 11.05.2024).

3. Playing Multiplayer Online Games Attractive Factors / García Hernáez, Samaniego. 2006.

4. Real stories too hard to believe. MMORPG-s. URL: https://medium.com/destream/real-stories-too-hard-to-believe-mmorpg-s-52272fcf36df (Дата звернення: 11.05.2024).

5. The co-op opportunity. URL: https://www.midiaresearch.com/blog/the-co-op-opportunity (Дата звернення: 13.05.2024).

6. Games as a service. URL: https://en.wikipedia.org/wiki/Games_as_a_service (Дата звернення: 11.05.2024).

7. 11 in-game currencies you need to know about. URL: https://medium.com/ironsource-levelup/11-in-game-currencies-you-need-to-know-about-8775c6724bcb (Дата звернення: 11.05.2024).

8. Multiplayer video game. URL: https://en.wikipedia.org/wiki/Multiplayer_video_game#cite_note-15 (Дата звернення: 11.05.2024).

9. Unreal Engine for AAA games. URL: https://www.unrealengine.com/en-US/uses/games (Дата звернення: 11.05.2024).

10. Online gaming statistics 2022. URL: https://www.uswitch.com/broadband/studies/online-gaming-statistics/ (Дата звернення: 11.05.2024).

11. Components. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/components-in-unreal-engine?application_version=5.3 (Дата звернення: 13.05.2024).

12. AI Controllers. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/ai-controllers-in-unreal-engine?application_version=5.3 (Дата звернення: 13.05.2024).

13. Cyber Demons | The AI of DOOM (2016). URL: https://www.gamedeveloper.com/design/cyber-demons-the-ai-of-doom-2016- (Дата звернення: 13.05.2024).

14. Behavior trees for AI: How they work. URL: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work (Дата звернення: 13.05.2024).

15. Niagara Overview. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-niagara-effects-for-unreal-engine?application_version=5.3 (Дата звернення: 24.05.2024).

16. Profiling and Optimization in UE4 | Unreal Indie Dev Days 2019 | Unreal Engine. URL: https://www.youtube.com/watch?v=EbXakIuZPFo&t=1747s (Дата звернення: 24.05.2024).

17. VFX Optimization Guide | Getting Results. URL: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/ParticleSystems/Optimization/Results/ (Дата звернення: 24.05.2024).

18. Niagara Data Channels Intro. URL: https://dev.epicgames.com/community/learning/tutorials/RJbm/unreal-engine-niagara-data-channels-intro (Дата звернення: 24.05.2024).

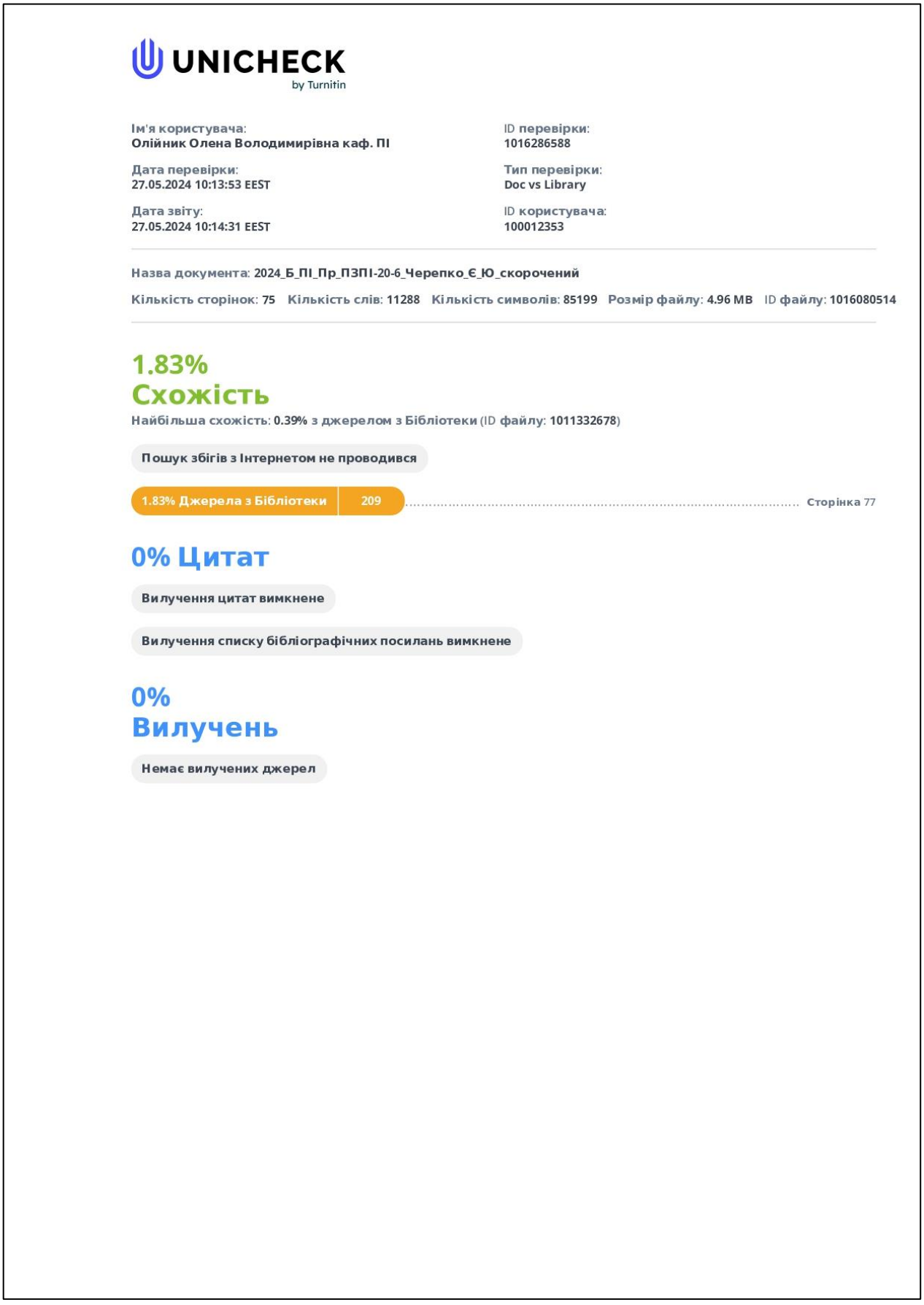
19. Черепко Є.Ю., Масштабування VFX як спосіб оптимізації в Unreal Engine 28-й // Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6., – Харків: ХНУРЕ. 2024. – 958 с.

20. Каталог виставки технічної творчості молоді. [Електронний ресурс] – URL: https://openarchive.nure.ua/handle/document/26355 (дата звернення: 25.05.2024).

21. Т. 6. Конференція "Інформаційні інтелектуальні системи". [Електронний ресурс] – URL: https://openarchive.nure.ua/handle/document/26355 (дата звернення: 25.05.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі хнуре



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016286588

Дата перевірки:
27.05.2024 10:13:53 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
27.05.2024 10:14:31 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_Пр_ПЗПІ-20-6_Черепко_Є_Ю_скорочений

Кількість сторінок: 75 Кількість слів: 11288 Кількість символів: 85199 Розмір файлу: 4.96 MB ID файлу: 1016080514

1.83%
Схожість

Найбільша схожість: 0.39% з джерелом з Бібліотеки (ID файлу: 1011332678)

Пошук збігів з Інтернетом не проводився

1.83% Джерела з Бібліотеки 209 ..... Сторінка 77

0% Цитат

Вилучення цитат вимкнене

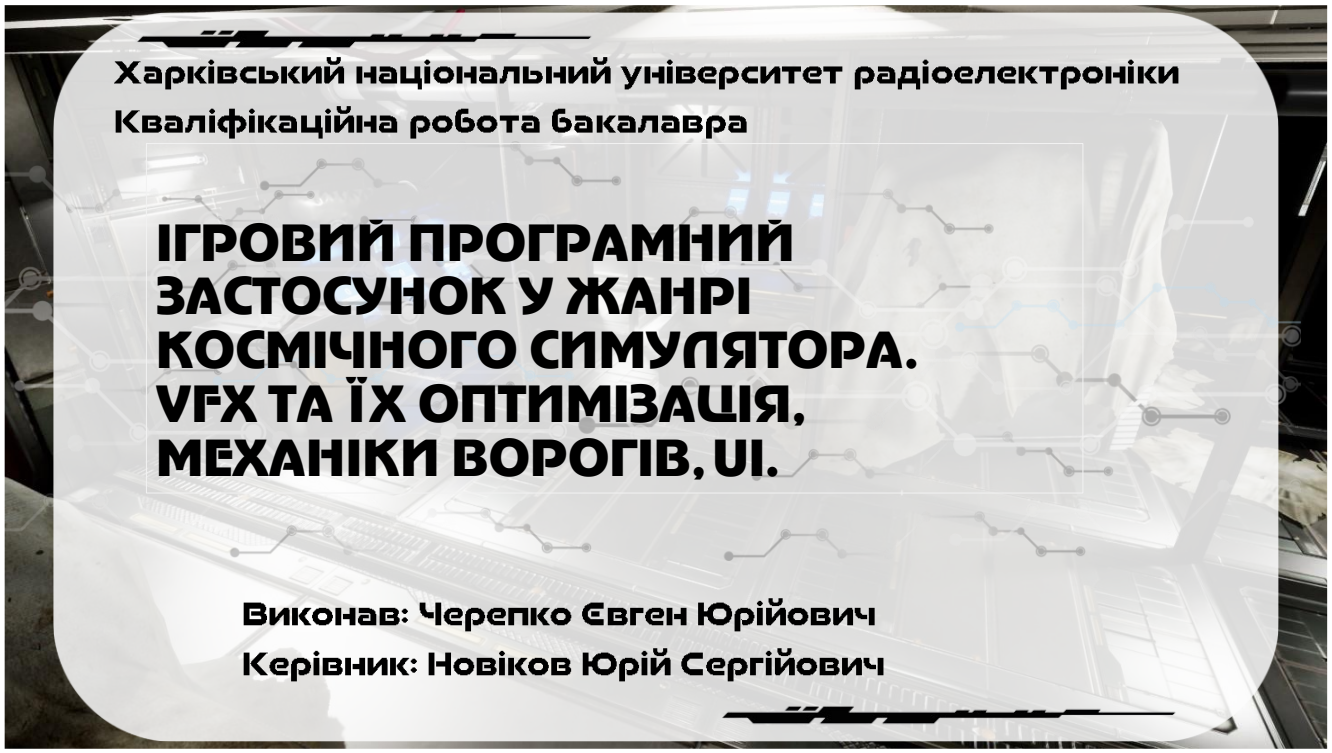
Вилучення списку бібліографічних посилань вимкнене

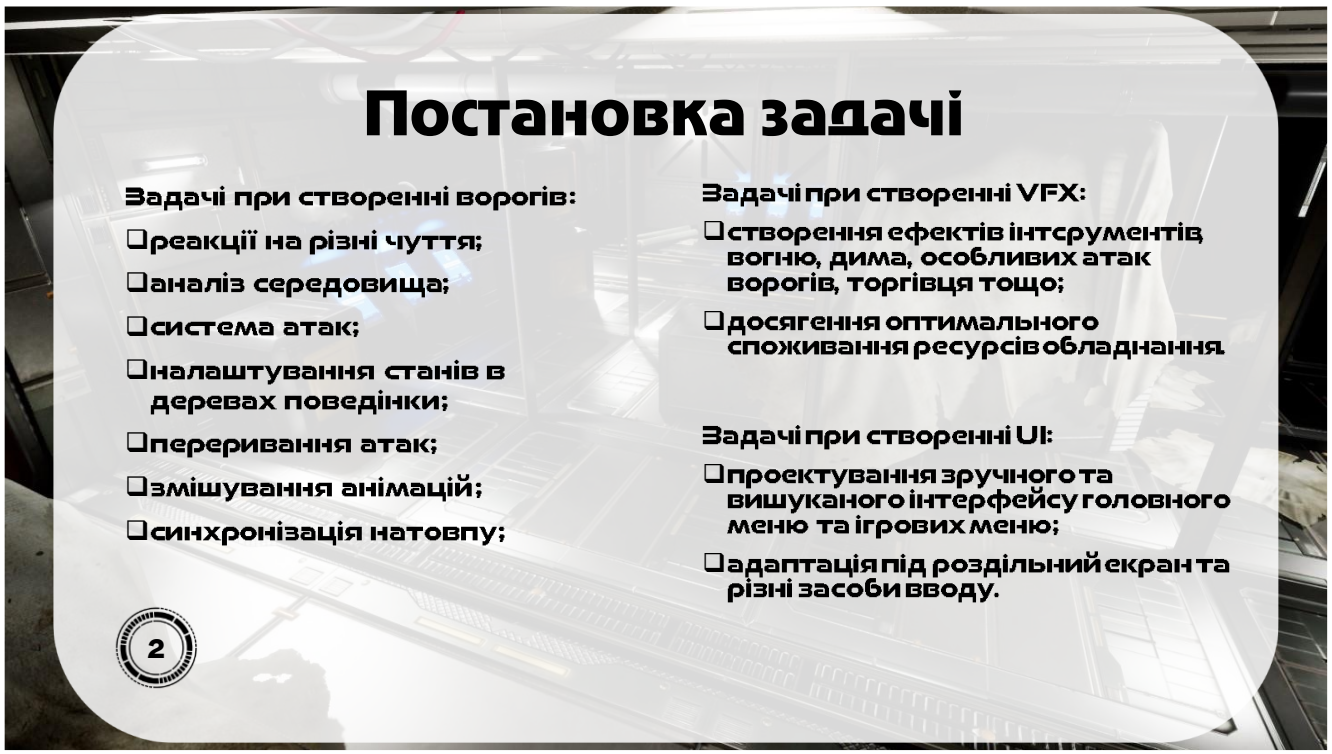
0%
Вилучень

Немає вилучених джерел

ДОДАТОК Б

Слайди презентації





Інструменти

GitLab

UNREAL ENGINE

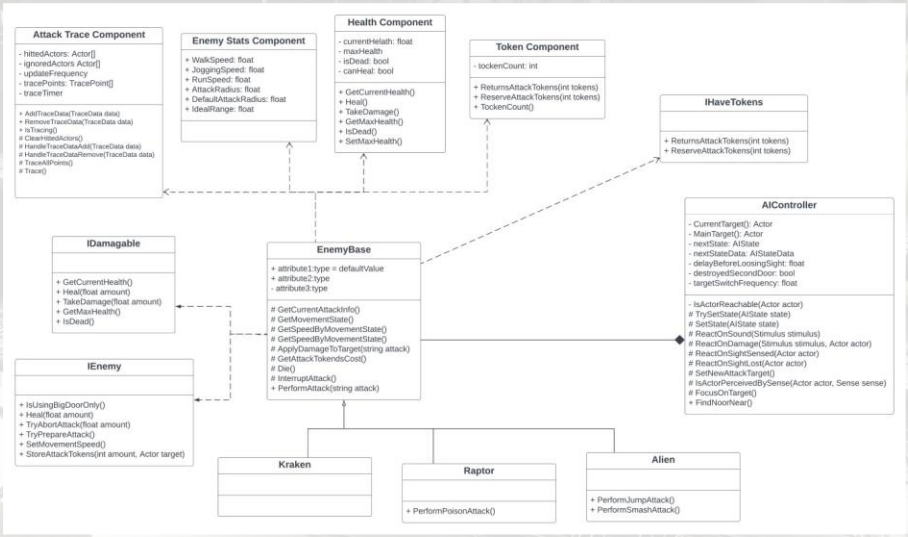
Figma

Ps

blender

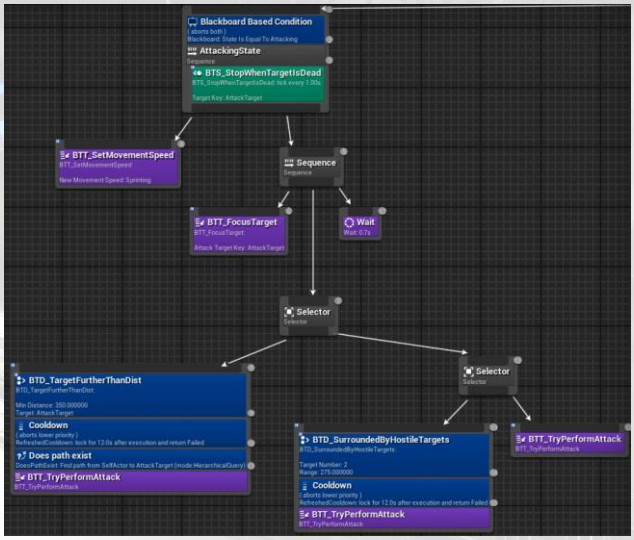
3

Архітектура ворогів



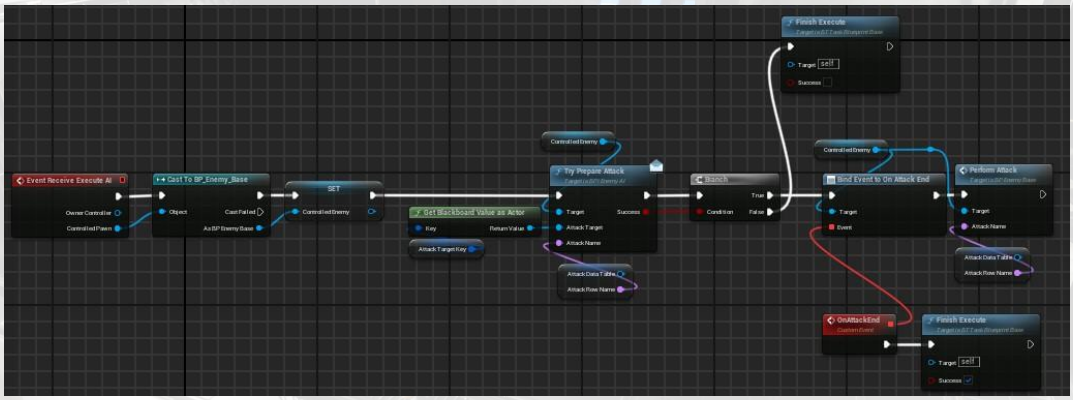
4

Дерева поведінки



5

Підготовка атаки

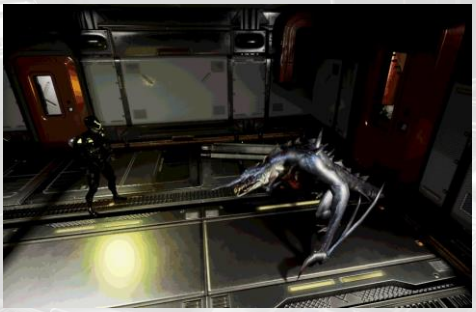


6

Підготовка атаки

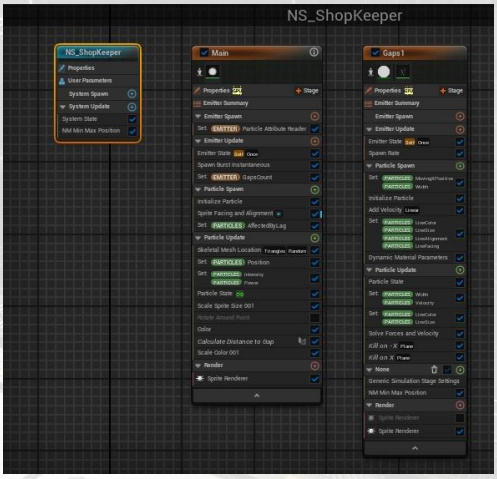
7

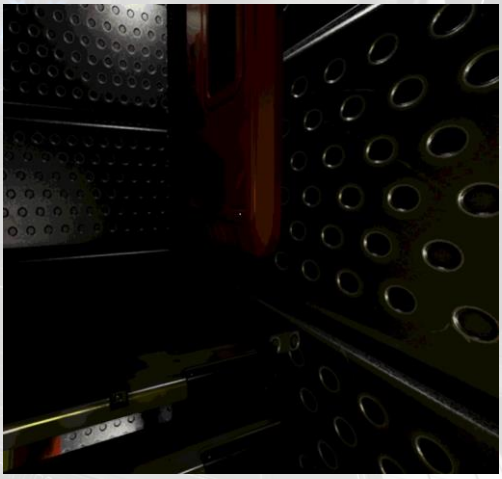
Анімовані ефекти



8

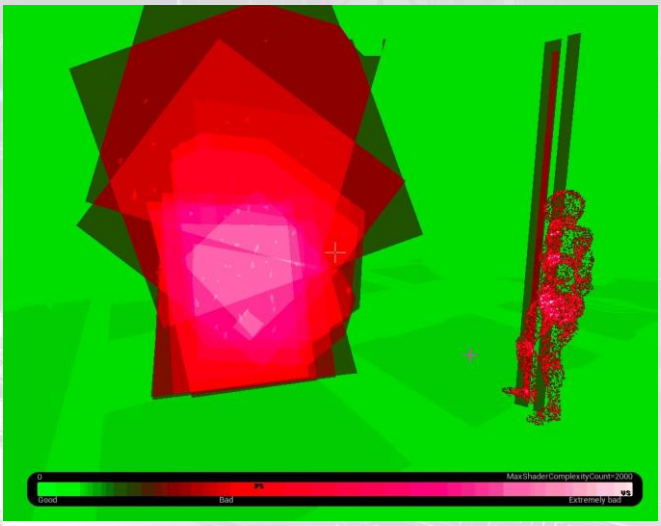
Ефект торгівця





9

Оптимізація

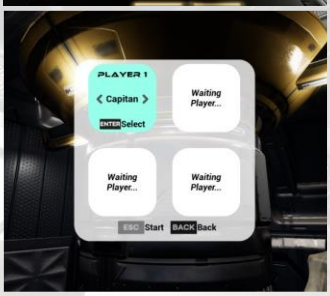


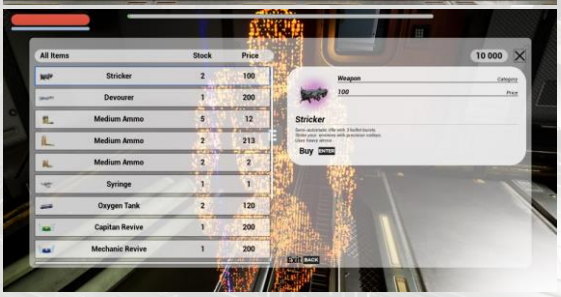
10

UI


Icarus
SINGLEPLAYER
LOCAL MULTIPLAYER
MULTUPLAYER
SETTINGS
CREDITS
EXIT


SETTINGS
DISPLAY
RESOLUTION 1024x768
FULL SCREEN MODE Windows
VSYNC Disable
GRAPHIC
TEXTURES Low
ILLUMINATION Medium
SHADOW Low
ENTER Applay
BACK Back


PLAYER 1
Capitan
ENTER Select
Waiting Player...
Waiting Player...
Waiting Player...
ESC Start
BACK Back


All Items
Stock
Price
10 000
Stricker 2 100
Devourer 1 200
Medium Ammo 5 12
Medium Ammo 2 213
Medium Ammo 2 2
Syringe 1 1
Oxygen Tank 2 120
Capitan Revive 1 200
Mechanic Revive 1 200
Weapon
100
Stricker
Buy
Exit BACK

11

UX

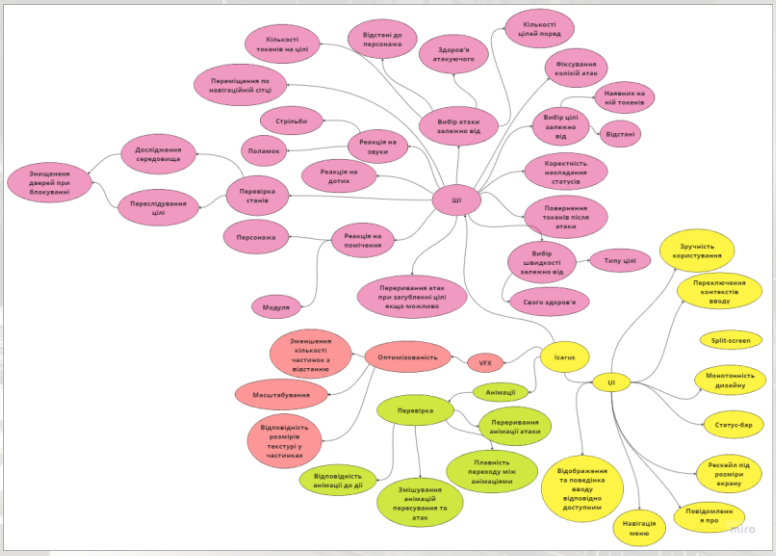

Icarus
SINGLEPLAYER
LOCAL MULTIPLAYER
MULTUPLAYER
SETTINGS
CREDITS
EXIT

12

Адаптація роздільного екрану

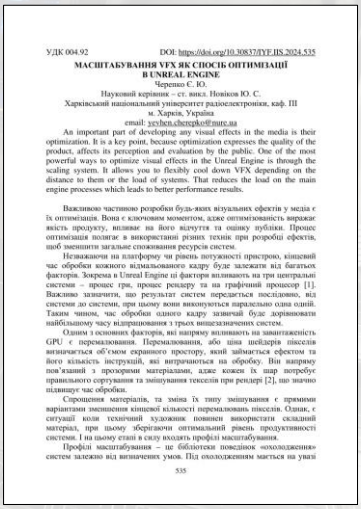
13

Тестування



14

Впровадження



УДК 004.92 DOI: https://doi.org/10.30837/ITSE.IIS.2024.535

МАСШТАБУВАННЯ VFX ЯК СПОСІБ ОПТИМІЗАЦІЇ В UNREAL ENGINE

Черепко Є. Ю.
Науковий керівник – ст. викл. Новіков Ю. С.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна
email: yevhen.cherepko@nure.ua

An important part of developing any visual effects in the media is their optimization. It is a key point, because optimization expresses the quality of the product, affects its perception and evaluation by the public. One of the most powerful ways to optimize visual effects in the Unreal Engine is through the scaling system. It allows you to flexibly cool down VFX depending on the distance to them or the load of systems. That reduces the load on the main engine processes which leads to better performance results.

Важливою частиною розробки будь-яких візуальних ефектів у медіа є їх оптимізація. Вона є ключовим моментом, адже оптимізованість виражає якість продукту, впливає на його відчуття та оцінку публіки. Процес оптимізації полягає в використанні різних технік при розробці ефектів, щоб зменшити загальне споживання ресурсів систем.

Незважаючи на платформу чи рівень потужності пристрою, кінцевий час обробки кожного відмальованого кадру буде залежати від багатьох факторів. Зокрема в Unreal Engine ці фактори впливають на три центральні системи – процес гри, процес рендеру та на графічний процесор [1]. Важливо зазначити, що результат систем передається послідовно, від системи до системи, при цьому вони виконуються паралельно одна одній. Таким чином, час обробки одного кадру зазвичай буде дорівнювати найбільшому часу відпрацювання з трьох вищезазначених систем.

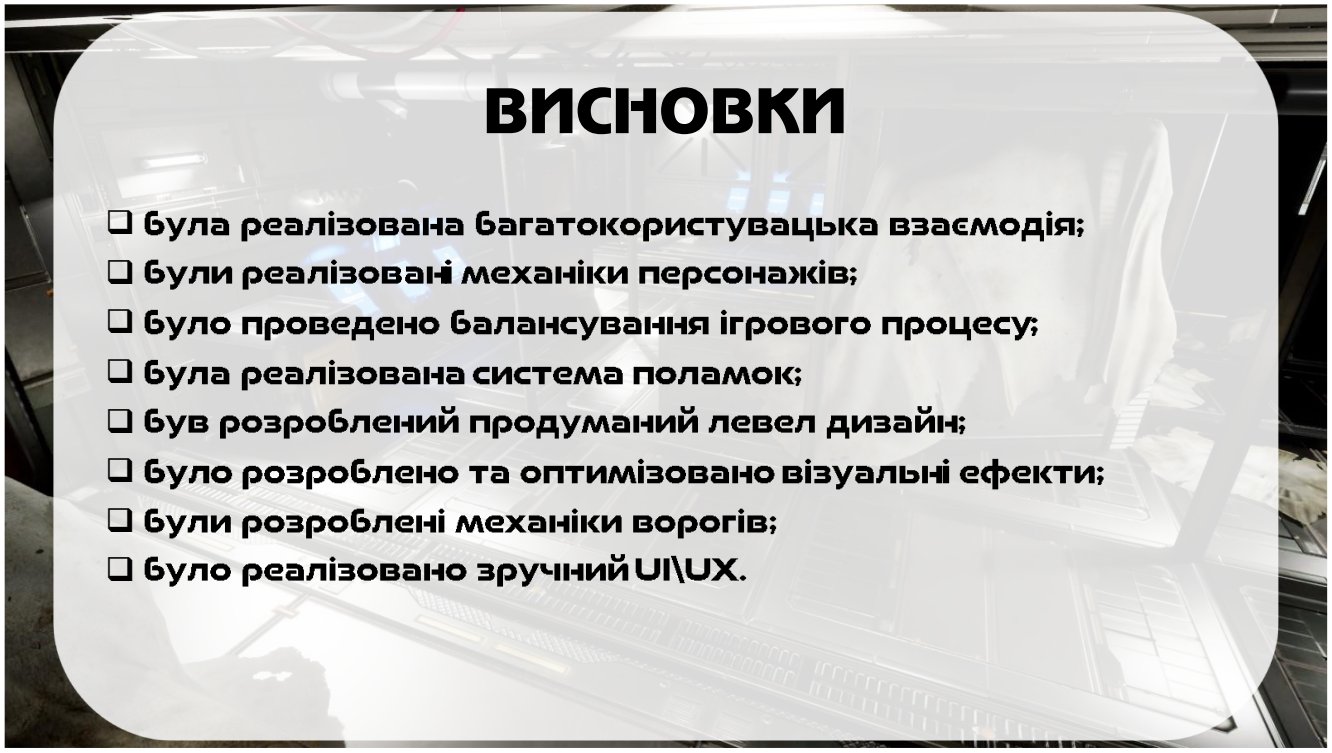
Одним з основних факторів, які напряму впливають на завантаженість GPU є перемальовання. Перемальовання, або ціна шейдерів пікселів визначається об'ємом екранного простору, який займається ефектом та його кількість інструкцій, які витрачаються на обробку. Він напряму пов'язаний з прозорими матеріалами, адже кожен їх шар потребує правильного сортування та змішування текселів при рендері [2], що значно підвищує час обробки.

Спрощення матеріалів, та зміна їх типу змішування є прямими варіантами зменшення кінцевої кількості перемальовань пікселів. Однак, є ситуації коли технічний художник повинен використати складний матеріал, при цьому зберігаючи оптимальний рівень продуктивності системи. І на цьому етапі в силу входять профілі масштабування.

Профілі масштабування – це бібліотеки поведінок «охолодження» систем залежно від визначених умов. Під охолодженням мається на увазі

Висновки

В ході кваліфікаційної роботи було розроблено комплексну систему ворогів, які адаптуються під дії гравців та навколишнє середовище. Було налаштовано анімації, які динамічно змішуються відповідно станів ворогів. Створено ряд візуальних ефектів, які оптимізовано для зручної гри. Було спроектован ігрові та неігрові меню, які адаптовано під різні режими роздільного екрану та способи вводу.


ВИСНОВКИ
❑ Була реалізована багатокористувацька взаємодія;
❑ Були реалізовані механіки персонажів;
❑ Було проведено балансування ігрового процесу;
❑ Була реалізована система поламок;
❑ Був розроблений продуманий левел дизайн;
❑ Було розроблено та оптимізовано візуальні ефекти;
❑ Були розроблені механіки ворогів;
❑ Було реалізовано зручний UI\UX.


Let's play!

ДОДАТОК В

Геймдизайн-документ

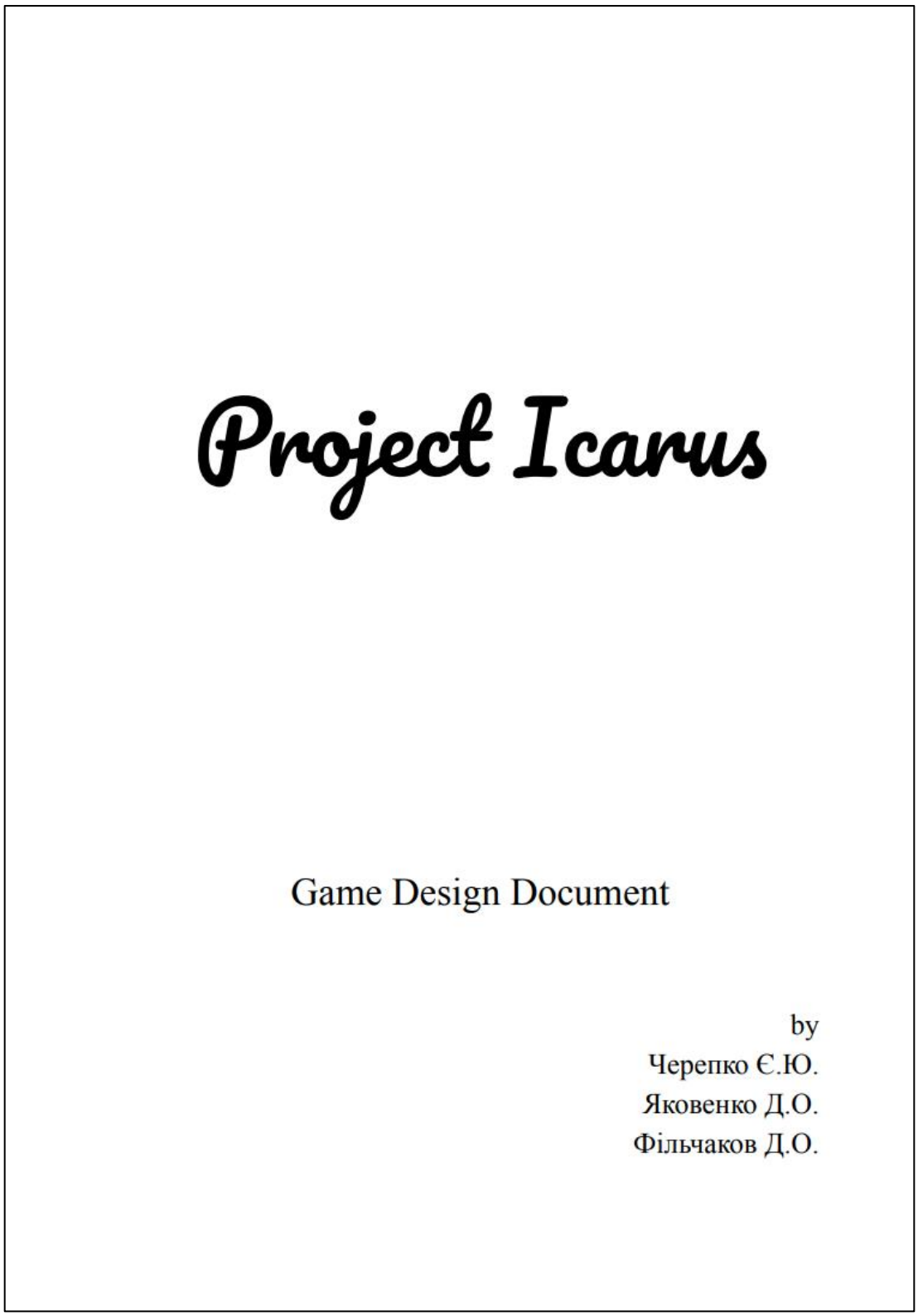


Рисунок В.1 – Сторінка 1 дизайн-документу

1. Загальні положення 3
1.1 Тетра 3
1.2 Введення 3
1.3 Цільова аудиторія 3
1.4 Unique selling proposition 3
1.5 Час ігрової сесії 4
1.6 Технічні характеристики 4
2. Ігровий процес 5
2.1 Характеристики персонажів 5
2.2 Класи персонажів 6
2.3 Механіки 7
2.3.1 Поломки 7
2.3.2 Лагодження систем корабля 8
2.3.3 Розхідники 8
2.3.4 Гроші 9
2.3.5 Магазин 9
2.4 Корабель 10
2.4.1 Палуба 11
2.4.2 Секція реактора 12
2.4.3 Секція двигуна 12
2.4.4 Секція охолодження 12
2.4.5 Склад 13
2.4.6 Секція життєзабезпечення 13
2.4.7 Генераторна 13
2.4.8 Хаб 13
2.4.9 Докер 13
2.4.10 Посадковий відсік 14
2.5 Зброя 14
3. Референси 16
3.1 Гравці 16
4. Інтерфейс 17
4.1 Основний геймплей 17

Рисунок В.2 – Сторінка 2 дизайн-документу

1. Загальні положення

1.1 Тетра

1) технологія – ПК;
2) механіка – симулятор виживання на космічному кораблі;
3) історія – злочинець;
4) естетика – Sci-fi.

1.2 Введення

Космічні пірати потрапляють у чергову пригоду, під час якої їх корабль зазнає пошкоджень і з кожною секундою починає розвалюватись. Їм треба досягти фінальної точки до того моменту, як від космічного корабля залишиться сама стеля, об'єднавшись з командою, або взявши все у свої руки. Слідкуйте за станом корабля, лагодіть що можливо полагодити, відбивайтеся від загарбників.

1.3 Цільова аудиторія

Основна аудиторія цієї гри це люди віком від 12 до 27 років. В першу чергу гра зацікавить командних гравців, або людей які полюбляють грати з рідними чи друзями. Також гра може залучити людей яким подобається динамічний геймплей зі швидким прийняттям рішень.

1.4 Unique selling proposition

- Проривайтеся крізь космічні простори у спробі втекти від космічної федерації на кораблі, що кожної миті розвалюється на шматочки!
- Зберіть команду спеціалістів у кооперативі на розділеному екрані та організуйте свою роботу, або спробуйте вижити на кораблі наодинці;

Рисунок В.3 – Сторінка 3 дизайн-документу

- Вчасно реагуйте на збої свого космічного судна та відбивайте атаки космічних загарбників.

1.5 Час ігрової сесії

Гра буде поділена на рівні, кожен з яких буде тривати по 10-15 хв. З кожним рівнем складність буде зростати, а в перервах можна буде купити потрібне обладнання та озброєння. Всього буде 3 рівні.

1.6 Технічні характеристики

Мінімальні системні вимоги для ПК:

Windows 10 64-bit, Quad Core 2.4 GHz, 8 GB RAM, GeForce GTX 1050;

Управління: клавіатура та миш, геймпад.

Рисунок В.4 – Сторінка 4 дизайн-документу

2. Ігровий процес

Гра починається на космічному кораблі, який рухається до наступної контрольной точки. Всього контрольних точок 3, включаючи фінальну. Основна задачи гри вижити дійшовши до фінальної точки. Виживанню персонажів буде сприяти стан корабля, а також працюючі модулі на ньому. Під час мандрівки, корабель буде обстрілюватися, що може наносити шкоди корпусу та модулям корабля, екіпажу, а також викликати пожежі (більш детально про загрози на кораблі описано у пункті 2.3.1). Крім того на корабель можуть проникати загарбники, які також несуть загрозу як екіпажу так і кораблю. При досягненні фіналу гри, гравці отримують результат в очках, який буде залежати від кількості грошей, які команда змогла зберегти під час мандрівки, та від цілісності корабля.

2.1 Характеристики персонажів

Ігрові персонажі будуть мати наступні характеристики:

- HP – здоров'я персонажа. При зниженні до 0, персонаж помирає, тим самим вибуває з гри повністю, або якщо гра виконувалась у со-ор інший гравець може викупити нового персонажа в магазині для вільного гравця;
- Кисень. Під час перебування у секції з пожежею або пробоїною він буде знижуватися (кожны 2 сек на 1 од). Після вичерпання запасів кисню, почне знижуватись HP (кожні 1 сек на 1 од), що може призвести до смерті. Якщо персонаж опускає забрало шолома скафандру, то кисень спочатку буде витрачатися з нього.
- Швидкість переміщення. Вона залежить від предмета, який тримає гравець.
- Рівень бойових навичок – впливає на якість персонажа у бою - додатковий урон від зброї, розкид стрільби (хитання ствола), відстань відкидання.

Рисунок В.5 – Сторінка 5 дизайн-документу

- Рівень навичок Інженера – впливає на здатність персонажу лагодити складні електричні системи, такі як реактор, двигуни, систему охолодження, систему подачи кисню, систему автопілота, генератори, камери та систему спостереження.
- Рівень навичок Механіка – впливає на здатність персонажу лагодити механічні пошкодження корабля, такі як пробоїни, дверей та пожеж.

2.2 Класи персонажів

В грі є чотири класи персонажів: Офіцер, Механік, Інженер, Капітан. У кожного рівень навичок розподілений відповідно свого класу:

В одиночній грі

	Рівень навичок Механіка	Рівень навичок Інженера	Рівень бойових навичок	Здоров'я	Кисень
Капітан	3	3	3	200	200

В коопі

	Рівень навичок Механіка	Рівень навичок Інженера	Рівень бойових навичок	Здоров'я	Кисень
Офіцер	1	1	3	200	100
Капітан	2	2	2	100	100
Механік	3	1	1	100	200
Інженер	2	3	1	100	100

Рисунок В.6 – Сторінка 6 дизайн-документу

2.3 Механіки

2.3.1 Поломки

Час від часу секції корабля будуть отримувати поломки. Поломками може бути пошкодження корпусу, викликана пожежа, поломка модуля, або втрата цілого відсіку.

2.3.1.1 Пошкодження корпусу та дверей

При пошкодженні корпусу відповідний відсік втрачає кисень. При трьох і більше пошкоджень в одному відсіку, швидкість переміщення в ньому зменшується.

При пошкодженні дверей їх буде неможливо зачинити, що буде призводити до розповсюдження безкисневості у відкриті відсіки, якщо суміжні з ними відсіки мають пробоїни.

2.3.1.2 Пожежа

При тривалому не ремонтуванні критичних поломок, почнеться пожежа. Коли у модулі почалась пожежа, у ньому закінчиться кисень, і потрібно буде використовувати його запаси зі скафандру. При тривалій пожежі, вона може перекинутись на інші відсіки.

2.3.1.3 Поломка модуля

Модулі корабля напряму впливають на ті чи інші його системи, тому їх пошкодження буде впливати на якість та працездатність цих систем. Кожен модуль має два рівні поломок – звичайний та критичний (про особливості поломок дивись відсіки корабля в яких він розташований у пунктах 2.5.1 - 2.5.9).

Рисунок В.7 – Сторінка 7 дизайн-документу

2.3.1.4 Втрата відсіку

При втраті відсіку, його вже ніяк не можна буде повернути, при цьому втрачаються всі модулі відсіку. Деякі відсіки неможливо втратити, деякі викликають миттєву поразку. Про наслідки для кожного відсіку див. п. 2.5.1 - 2.5.9.

2.3.2 Лагодження систем корабля

Для лагодження пошкоджень потрібні відповідні для цього предмети:

- пошкодження корпусу та дверей лагодяться зварювальним апаратом (3 сек);
- пожежі гасяться вогнегасником (секунда за одиницю вогню);
- пошкодження модулів лагодяться паяльником (звичайне - 4 сек, критичне - 8 сек).

Щоб полагодити пошкодження, треба знайти місце поломку, яка буде виділена vfx ефектами і направити на неї зварювальним апаратом, якщо це пробоїна корпусу, або паяльником, якщо це пошкодження модуля. При цьому знижується заряд відповідного інструмента.

Якщо у гравця в руках немає потрібного обладнання, або він не хоче витрачати заряд відповідного обладнання, можна використати універсальний ресурс (синю ізоленту), щоб відремонтувати будь-яку поломку, але на дуже короткий термін (10 секунд).

Щоб загасити пожежу, треба взяти вогнегасник та почати розпилювати порошок на вогонь.

2.3.3 Розхідники

Всі розхідники у грі можна розділити на 3 категорії:

1) заряди для інструментів – заряди для паяльника та заряди для зварювального апарату.
2) розхідники персонажів – аптечки та балони з киснем;

Рисунок В.8 – Сторінка 8 дизайн-документу

3) бойові розхідники – амуніція для зброї.

2.3.4 Гроші

На початку гри гравці мають фіксовану суму грошей. Гроші витрачаються у магазинах, а також від неї залежить фінальний результат гри.

2.3.5 Магазин

Магазин відкривається на контрольних точках і доступний біля докера. Товари магазину включають наступні:

Товар	**Ціна**
заряди для паяльника	15
заряди для вогнегасника	
заряди для зварювального апарату	10
аптечка	40
найняти нового члена екіпажу(воскресити померлого)	100
балони кисню	25
патрони для пістолету	10
напівавтоматичний автомат	100
патроны напівавтоматичного для автомату	15
гвинтівка	200

Рисунок В.9 – Сторінка 9 дизайн-документу

патроны для гвинтівки	20

заряди для паяльника;

заряди для зварювального апарату;

аптечка;

найняти нового члена екіпажу(воскресити померлого);

балони кисню;

патрони для пістолету;

напівавтоматичний автомат;

патроны напівавтоматичного для автомату;

гвинтівка;

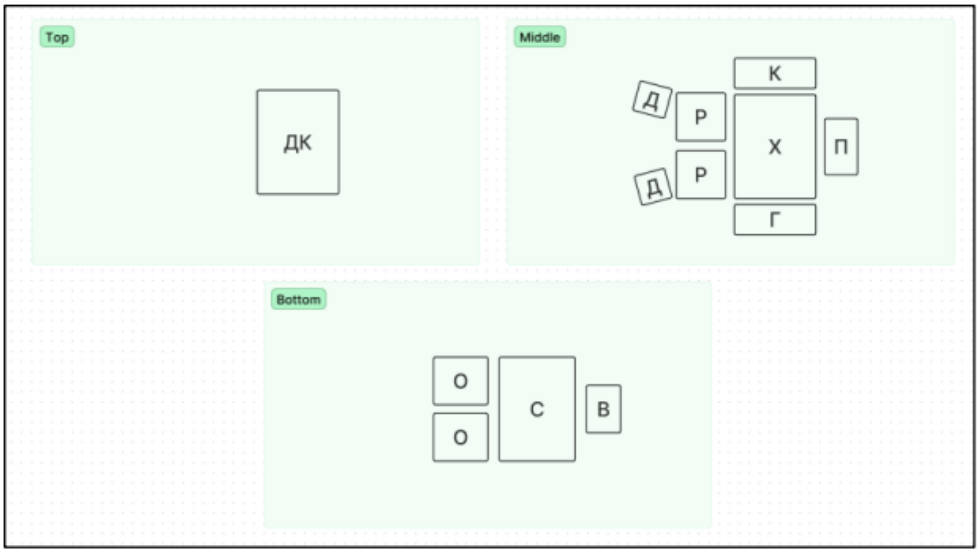
патроны для гвинтівки.

2.4 Корабель

Корабель розділений на секції, які з'єднані між собою шлюзом, або дверима. До основних відсіки корабля належать: палуба, секція реактора, секція двигуна, секція охолодження, склад, секція життєзабезпечення, генераторна, хаб, докер. Більшість з цих секції мають модулі, які можуть бути пошкоджені.

Загальна схема корабля має наступний вигляд:

Рисунок В.10 – Сторінка 10 дизайн-документу



де П - палуба

Р - реактор

Д - секція двигуна

О - секція охолодження

С - склад

К - секція життєзабезпечення

Г - генераторна

Х - хаб

ДК - докер

В - посадковий відсік

2.4.1 Палуба

Секція, на якій будуте розташована система спостереження для аналізу стану корабля та моніторингу вторгнень, інструменти розгерметизування частин корабля та система автопілота.

Система спостереження представлена у вигляді зображень з камер кожного відсіку. Коли виникає поломку у відсіку, індикатор загоряється біля

Рисунок В.11 – Сторінка 11 дизайн-документу

відповідної камери. Дізнатися деталі поломки вже можна через камеру. При поломці, камери перестають працювати, але індикатори працюють. При критичній поломці не працюють як камери, так і індикатори. Також цей відсік неможливо втратити.

Інструмент розгерметизації дозволяє відстикувати модуль корабля, щоб наслідник його поломки цього не перейшли на інші модулі. При його поломці, відстикування модулів корабля буде проводитись повільніше. При критичній поломці, капітан не зможе відстикувати модулі корабля

При поломці автопілота, корабель почне частіше отримувати пошкодження корпусу, тобто пробоїни. В залежності від тривалості поломки, частота появи пробоїн почне збільшуватись.

2.4.2 Секція реактора

На цій секції будуть розташований реактор, який підтримує життєздатність двигуна. При його пошкодженні спалахує пожежа. При виведенні зі строю, зупиняється відповідний двигун. При втраті цієї секції, втрачається секція з відповідним двигуном. При втраті всіх реакторів гра закінчується поразкою.

2.4.3 Секція двигуна

Від двигуна корабля напряму залежить його корабля. Пошкодження двигуна сповільнює корабель, а критична поломка зупинить переміщення корабля. При втраті двигуна, швидкість корабля зменшується вдвічі. Всього на кораблі дві секції з двигунами. При втраті двох двигунів гра закінчується поразкою.

2.4.4 Секція охолодження

На цій секції будуть розташовані системи охолодження реактора. На кораблі дві такі секції, по кожному на реактор. При поломці, реактори час від

Рисунок В.12 – Сторінка 12 дизайн-документу

часу почнуть перегріватися, шо буде викликати їх середню поломку. Якщо ж реактори вже перегріті, то це буде викликати критичну їх поламку.

При критичній поломці, або втраті цієї секції, реактор буде ще частіше перегріватися.

2.4.5 Склад

Секція з усім знаряддям для лагодження, розхідниками та зброєю.

2.4.6 Секція життєзабезпечення

Секція, яка відповідає за подачу кисню на кораблі. При частковій поломці кисень зникає на всіх відсіках корабля, окрім палуби, хабу та самої секція життєзабезпечення. При критичній поломці, або втраті відсіку кисень зникає усюди.

2.4.7 Генераторна

Секція з генератором, який відповідає за освітлення корабля. При поломці, світло на кораблі почне блимати, а при критичній поломці вимкнеться повністю. При від'єднані цієї секції, світло на кораблі вже ніколи не повернеться.

2.4.8 Хаб

Велика основна секція корабля. Звідси можна перейти до інших секцій.

2.4.9 Докер

Невелика секція корабля, призначена для стикування до інших кораблів. Через неї на борт будуть потрапляти загарбники (у більшості випадках). Також через неї виконується торгівля у магазинах. Не може бути розгерметизована.

Рисунок В.13 – Сторінка 13 дизайн-документу

2.4.10 Посадковий відсік

Звичайний відсік, через який команда висаджується на землі.

2.5 Зброя

2.5.1 Пістолет

Урон	10
Магазин	16
Скорострільність	3 пулі/сек
Режим стрільби	По 1 пулі

2.5.2 Напівавтоматичний автомат

Урон	14
Магазин	27
Скорострільність	3 пулі/сек
Режим стрільби	По 3 пулі

Рисунок В.14 – Сторінка 14 дизайн-документу

2.5.3 Гвинтівка

Урон	15
Магазин	30
Скорострільність	5 пулі/сек
Режим стрільби	Автоматичний

Рисунок В.15 – Сторінка 15 дизайн-документу

3. Референси

3.1 Гравці

Кожен гравець матиме свій унікальний колір підсвітки.

Рисунок В.16 – Сторінка 16 дизайн-документу

4. Інтерфейс

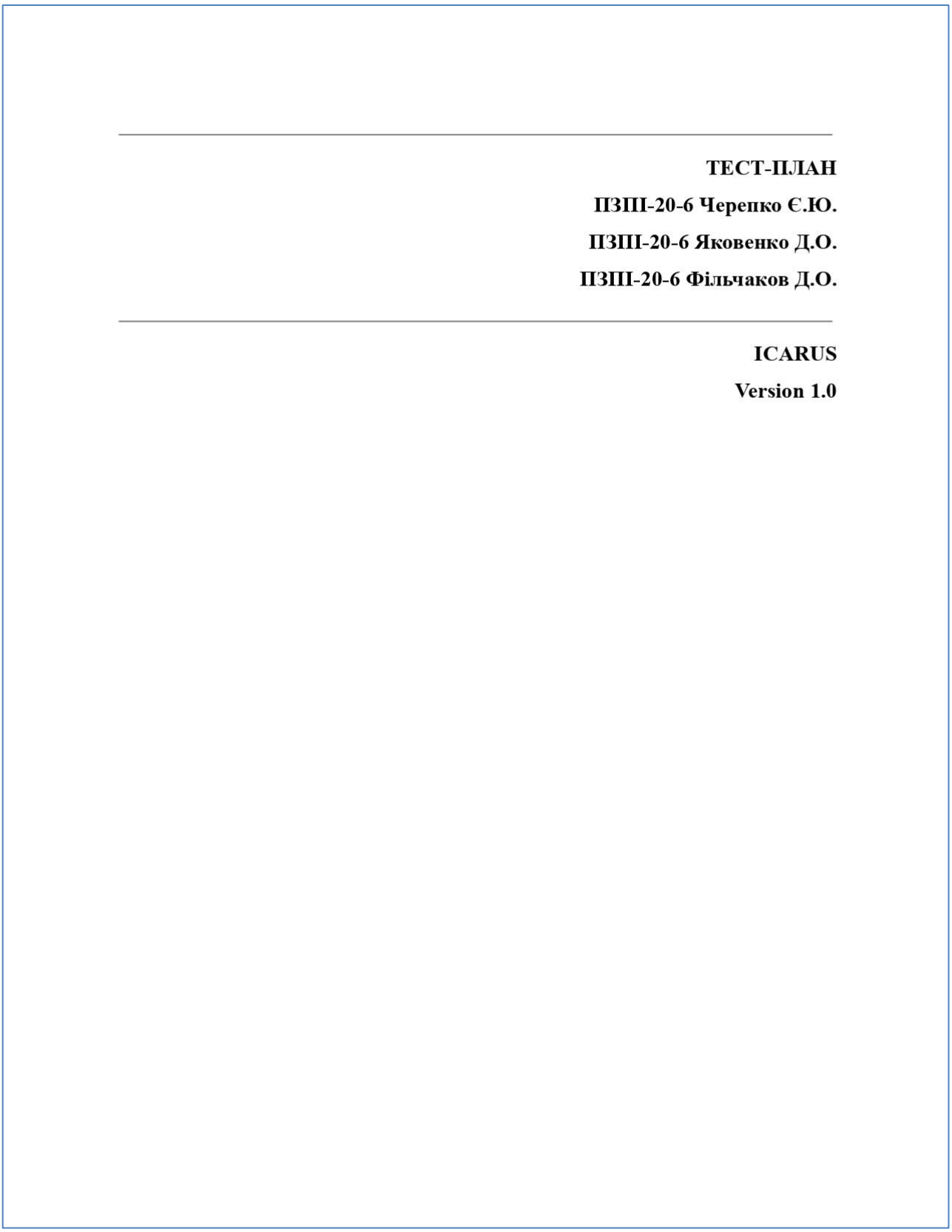
4.1 Основний геймплей

На екрані у гравця відображається інформація про стан його здоров'я (у верхньому лівому куті червоний статус бар), стан балонів з киснем (у верхньому лівому куті синій статус бар), стан здоров'я та балонів з киснем команди (у верхньому правому куті), залишок заряду його поточного предмета в руці предмету, якщо він міє заряди (внизу ліворуч), обраний предмет (внизу праворуч), індикатор шляху, який корабль подала, і який йому ще треба подолати, для успішного проходження рівня (зелений статус бар зверху).

Рисунок В.17 – Сторінка 17 дизайн-документу

ДОДАТОК Г

Тест-план



ТЕСТ-ПЛАН

ПЗПІ-20-6 Черепко Є.Ю.

ПЗПІ-20-6 Яковенко Д.О.

ПЗПІ-20-6 Фільчаков Д.О.

ICARUS

Version 1.0

Рисунок Г.1 – Сторінка 1 тест-плану

Revision History

Date	Version	Description	Author
26/02/2024	1.0	Initial revision	Черепко Є.Ю. Яковенко Д.О. Фільчаков Д.О.

Рисунок Г.2 – Сторінка 2 тест-плану

Table of Contents

1. Вступ (Introduction) 4
1.1 Мета (Purpose) 4
1.2 Довідкова інформація (Background) 4
1.3 Галузь застосування (Scope) 5
1.4 Визначення проекту (Project Identification) 8
2. Вимоги до тестування (Requirements for Test) 9
3. Стратегія тестування (Test Strategy) 9
3.1 Типи тестування (Testing Types) 10
3.1.1 Дані і БД Інтеграційне тестування (Data and Database Integrity Testing) 10
3.1.2 Функціоанльне тестування (Function Testing) 10
3.1.3 Бізнес-цикл тестування (Business Cycle Testing) 11
3.1.4 Тестування інтерфейсу користувача (User Interface Testing) 11
3.1.5 Тестування продуктивності (Performance Profiling) 12
3.1.6 Завантажувальне тестування (Load Testing) 12
3.1.7 Стресове тестування (Stress Testing) 13
3.1.8 Навантажувальне тестування (Volume Testing) 13
3.1.9 Тестування безпеки і контролю доступу (Security and Access Control Testing) 13
3.1.10 Тестування відмовостійкості та відновлення (Failover and Recovery Testing) 13
3.1.11 Тестування конфігурації (Configuration Testing) 14
3.1.12 Тестування інсталяції (Installation Testing) 16
3.2 Інструменти (Tools) 16
4. Ресурси (Resources) 16
4.1 Ролі (Roles) 16
4.2 Система (System) 19
Задачі проекту (Appendix A Project Tasks) 20

Рисунок Г.3 – Сторінка 3 тест-плану

Test Plan

1. Вступ (Introduction)

1.1 Мета (Purpose)

Метою складання Тест Плану є опис процесу тестування комп'ютерної гри Icarus, що є нашою дипломною роботою. Розглянемо визначення існуючої інформації про гру, визначення рекомендованих вимог до тестування, опис стратегій тестування, рекомендацій щодо використання, визначення необхідних ресурсів і забезпечення оцінки випробувань, перелік тестових елементів гри.

1.2 Довідкова інформація (Background)

У якості об’єкта тестування було обрано гра Icarus, що є нашою дипломною роботою.

В ході тестування необхідно буде перевірити найважливіші складові гри . А саме:

1. Графічний інтерфейс;
2. Анімації персонажів;
3. Взаємодія персонажа з оточенням;
4. Ворогів;
5. Геймплей у різних режимах(одиночний\split-screen\multiplayer);

Саме ці функції є основою онлайн гри, що тестується.

Тестування гри буде виконано в ручному режимі. Якщо в результаті тестування буде знайдено 75 багів, то тест можна вважати вдалим.

Рисунок Г.4 – Сторінка 4 тест-плану

1.3 Галузь застосування (Scope)

Повний набір складових частин, що будуть протестовані:

1. Графічний інтерфейс:

– Головне меню;
– Меню налаштувань;
– Меню створення сесії;
– Меню обору ролі в split-screen;
– Меню обору ролі в мультиплеєрі;
– Магазин;
– Екран завантаження гри;
– HUD персонажа;
– Меню паузи.

2. Анімації персонажів:

– Анімація пересування персонажа;
– Анімації персонажа з різними предметами;
– Анімації персонажа при використанні споживаних предметів;
– Анімація пересування ворогів;
– Анімація атак ворогів;

3. Мультиплеер:

– Авторизація;
– Створення сесії;
– Під'єднання до сесії;
– Реплікація пересування;
– Реплікація станів персонажів;
– Реплікація модулів;

Рисунок Г.5 – Сторінка 5 тест-плану

– Реплікація ворогів;
– Реплікація предметів;
– Реплікація магазину;
– Реплікація інтерфейсу в лобі;
– Реплікація поломок;

4. Штучний інтелект ворогів:
– перевірка реакції ворогів на персонажів;
– перевірка реакції ворогів на модулі;
– перевірка коректності пошуку ворогом цілі;
– коректність пересування ворогів по навігаційній сітці;
– перевірка узгодженості атак ворогів між собою;

5. Магазин:
– коректність списання ігрової валюти та начислення предметів, або ресурсу;
– коректність оновлення наявних предметів з переходом на новий рівень;
– коректність відродження загиблих персонажів при їх викупі;

6. VFX
– Коректність відображення частиць.
– Оптимізованість візуальних ефектів.
– Коректність взаємодії частиць з навколишнім середовищем.

7. Баланс
– перевірка балансу характеристик персонажів;
– перевірка балансу характеристик ворогів;

Рисунок Г.6 – Сторінка 6 тест-плану

– перевірка балансу систему токенів атаки;
– перевірка балансу атак ворогів: швидкості, частоти, сили особливих ефектів;
– перевірка балансу характеристик модулів;
– перевірка збалансованості алгоритму поломок;
– перевірка балансу зброї;
– перевірка балансу предметів починки;
– перевірка балансу споживаних предметів;
– перевірка балансу пожеж;
– перевірка балансу витрачання кисню;

8. Геймплей у різних режимах:
– Вибір класу персонажа в split-screen;
– Вибір класу персонажа в мультиплеєрі;
– Геймплей в однокористувацькому режимі;
– Геймплей в split-screen;
– Геймплей в мультиплеєрі;
– Спавн персонажей;
– Винекнення поламок;
– Спавн ворогів;
– Респавн;
– Виграш та програш;

Ризиком, або непередбачуваною обставиною, що може вплинути на тестування може бути відмова роботи Epic Online Services з технічних причин або технічні роботи на сервері.

Рисунок Г.7 – Сторінка 7 тест-плану

1.4 Визначення проекту (Project Identification)

Документ і версія / дата) (Document and version / date)	Створено або Доступно (Created or Available)	Поступило або перегляну то (Received or Reviewed)
Специфікація вимог (Requirements Specification)	Так	Ні
Функціональна специфікація (Functional Specification)	Так	Так
Звіти використання - випадок (Use-Case Reports)	Ні	Ні
План проекту (Project Plan)	Так	Ні
Специфікація дизайну (Design Specifications)	Так	Так
Прототип (Prototype)	Так	Ні
Керівництво користувача (User's Manuals)	Ні	Ні
Бізнес модель (Business Model or Flow)	Ні	Ні
Модель даних (Data Model or Flow)	Ні	Ні
Бізнес-функції (Business Functions and Rules)	Так	Ні

Рисунок Г.8 – Сторінка 8 тест-плану

Оцінка ризиків (Project or Business Risk Assessment)	Ні	Ні

2. Вимоги до тестування (Requirements for Test)

В ході тестування необхідно буде перевірити найважливіші складові гри . А саме:

1. Графічний інтерфейс;
2. Анімації персонажів;
3. Взаємодія персонажа з оточенням;
4. Ворогів;
5. Геймплей у різних режимах(одиночний\split-screen\multiplayer);

Саме ці функції є основою онлайн гри, що тестується.

Тестування гри буде виконано в ручному режимі. Якщо в результаті тестування буде знайдено 75 багів, то тест можна вважати вдалим.

3. Стратегія тестування (Test Strategy)

Для тестування було прийнято рішення використати наступні типи:

- Функціональне тестування;
- Тестування інтерфейсу користувача;
- Тестування продуктивності;
- Тестування відмовостійкості;
- Тестування конфігурації;

Інтеграційне тестування, бізнес-цикл тестування, стресове тестування, тестування безпеки і контролю доступу, тестування навантаження, завантажувальне тестування та тестування інсталяції не будуть проведені з

Рисунок Г.9 – Сторінка 9 тест-плану

ряду причин, що будуть наведені нижче у відповідних пунктах.

3.1 Типи тестування (Testing Types)

3.1.1 *Дані і БД Інтеграційне тестування (Data and Database Integrity Testing)*

Цей тип тестування неможливо провести через відсутність БД. Від тесту відмовляємося, через непотрібність проведення.

3.1.2 *Функціоанльне тестування (Function Testing)*

Було прийнято рішення обрати цей тип тестування з метою перевірки правильності прийняття і обробки даних, а також належного виконання бізнес-правил. Тестування буде проведено за методом «білої скрині», тобто перевірки гри, занурюючись у код, щоб зрозуміти яка частина коду працює некоректно.

Мета випробування (Test Objective):	Виявити баги
Технічний прийом (Technique):	Методом «білої скрині» перевірити кожну функцію, виявити дефекти.
Критерії завершення (Completion Criteria):	Кожна складова гри протестована, виявлені помилки записані у баг-репорт
Спеціальні рекомендації (Special Considerations):	Зафіксувати процес тестування на відео

Рисунок Г.10 – Сторінка 10 тест-плану

3.1.3 *Бізнес-цикл тестування (Business Cycle Testing)*

Бізнес цикл тестування не буде використовуватися, оскільки ми не маємо необхідної документації. Цей тип тестування потребує документації, де визначені бізнес-процеси, що представляють собою сценарії щоденного використання. В нашому випадку такої документації немає у наявності. Від цього тестування відмовляємося.

3.1.4 *Тестування інтерфейсу користувача (User Interface Testing)*

Тестування інтерфейсу користувача було обрано з метою перевірки взаємодії користувача з інтерфейсом гри. В ході тестування необхідно перевірити зручність та правильність роботи інтерфейсу, а також виявити візуальні баги та баги анімацій.

Мета випробування (Test Objective):	Перевірити всі доступні користувачеві сторінки, діалогові вікна та елементи ігрового інтерфейсу під час геймплею.
Технічний прийом (Technique):	Створення та редагування випробувань для кожного елемента, щоб перевірити коректність його роботи та наявність візуальних помилок.
Критерії завершення (Completion Criteria):	Кожен елемент інтерфейсу був перевірений, виявлені візуальні баги занесені у баг-репорт.

Рисунок Г.11 – Сторінка 11 тест-плану

3.1.5 *Тестування продуктивності (Performance Profiling)*

Тестування продуктивності було обрано з метою виміру оцінки часу відгуку, перевірки швидкості транзакції, й інших вимог. Тестування продуктивності реалізується і виконується згідно з профілем, а також розглядається залежно від робочого навантаження або апаратної конфігурації.

Мета випробування (Test Objective):	Перевірка продуктивності різних функцій та час їх відгуку, порівняння з нормою.
Технічний прийом (Technique):	- В ході тестування проводити додаткові перевірки на наявність проблем з продуктивністю та багів, що викликають «лаги»; - Тестування проводити на конфігурації, що відповідає рекомендованим системним потребам, або краще.
Критерії завершення (Completion Criteria):	В ході перевірки всі показники продуктивності знаходяться в межах норми, під час тестування не виникає збоїв.

3.1.6 *Завантажувальне тестування (Load Testing)*

Рисунок Г.12 – Сторінка 12 тест-плану

Завантажувальне тестування не буде проведено, через неможливість залучити до тестування максимально можливу кількість користувачів. Цей вид тестування передбачає перевірку роботи гри при великій кількості користувачів.

3.1.7 *Стресове тестування (Stress Testing)*

Тестування неможливо провести через відсутності виділених серверів, оскільки вся мережева взаємодія відбувається через сервери, які хостят гравці, і стоїть обмеження на 4 гравців на один сервер.

3.1.8 *Навантажувальне тестування (Volume Testing)*

Навантажувальне тестування неможливо для проведення, оскільки гра не передбачає поля вводу де, є можливість ввести велику кількість даних для тестування. Крім того цей вид тестування не є обов'язковим для функцій, що необхідно перевірити.

3.1.9 *Тестування безпеки і контролю доступу (Security and Access Control Testing)*

Даний тип тестування неможливо провести через відсутність різних ролей у грі. Існує лише роль звичайного гравця.

3.1.10 *Тестування відмовостійкості та відновлення (Failover and Recovery Testing)*

Було прийнято рішення обрати цей тип тестування з метою перевірки

Рисунок Г.13 – Сторінка 13 тест-плану

відмовостійкості гри, при втрачанні з’єднання з сервером, або з клієнтом.

Мета випробування (Test Objective):	Перевірити всі етапи гри, на яких можуть виникнути проблеми з мережею.
Технічний прийом (Technique):	Створення та редагування випробувань для кожного етапу, щоб перевірити коректність його роботи.
Критерії завершення (Completion Criteria):	Кожен випадок був перевірений, виявлено баги, та занесено їх в баг-репорт.

3.1.11 *Тестування конфігурації (Configuration Testing)*

Конфігурація тестування перевіряє через тест роботу гри при різних умовах апаратної конфігурації. Цей тип тестування буде проведений з метою перевірки коректності роботи гри на мінімальній та рекомендованій конфігурації.

Рисунок Г.14 – Сторінка 14 тест-плану

Мета випробування (Test Objective):	Тестування проводиться належним чином на мінімальній та рекомендованій конфігурації.
Технічний прийом (Technique):	Проведення тестових випадків на різних конфігураціях та порівняння результатів виконання однакових функцій. Виявлення відмінностей у результатах, що будуть вважатися багами.
Критерії завершення (Completion Criteria):	Робота гри на різних конфігураціях перевірена, виявлені баги занесені в баг-репорт.
Спеціальні рекомендації (Special Considerations):	Для тестування використати наближену до Мінімальна: - Процесор: amd ryzen 5 1600; - Відеокарта: GTX 1650; - Озу: 8гб; А також конфігурацію, що буде вище за рекомендована: - Процесор: amd ryzen 5 3600; - Відеокарта: GTX 2070 - Озу: 16гб;

Рисунок Г.15 – Сторінка 15 тест-плану

3.1.12 *Тестування інсталяції (Installation Testing)*

Тестування інсталяції не потребується для тестування зазначених вище функцій, крім того інсталяція відноситься до тестування цифрового магазину, що не розглядається в тест плані.

3.2 Інструменти (Tools)

The following tools will be employed for this project:

	Tool
Test Management	Microsoft Excel
Defect Tracking	Jira
ASQ Tool for performance testing	FPS Monitor
Створення тест кейсів (Test case creation)	Microsoft Excel
Відстеження тест кейсів(Test case tracking)	Microsoft Excel

4. Ресурси (Resources)

4.1 Ролі (Roles)

Ця таблиця показує, кадрові забезпечення для проекту.

Рисунок Г.16 – Сторінка 16 тест-плану

Людський ресурс (Human Resources)		
Працівник (Worker)	Конкретні обов'язки або Коментарі (Specific Responsibilities or Comments)	Конкретні обов'язки або Коментарі (Specific Responsibilities or Comments)
Тест-менеджер, Менеджер з тестування проекту (Test Manager, Test Project Manager)	1	Забезпечує управління наглядом. Обов'язки: - технічна підтримка, - придбання відповідних ресурсів, - забезпечення управлінської звітності

Рисунок Г.17 – Сторінка 17 тест-плану

Конструктор тестів (Test Designer)	1	Визначення, пріоритетів, і реалізація тестів. Обов'язки: - створення плану тестування, - генерація тестових моделей, - оцінка ефективності тестових зусиль.
Тестувальник (Tester)	1	Виконання тестів. Обов'язки: - виконання тестів, - журнал результатів, - відновлення в журналі реєстрації після помилок, - документ зміни.
Виконавець (Implementer)	1	Реалізує модульні тести і тестові класи Обов'язки: - створює тестові класи і пакети, - виконує тестові моделі

Рисунок Г.18 – Сторінка 18 тест-плану

4.2 Система (System)

Нижче в таблиці представлені системні ресурси для тестування гри.

System Resources	
Resource	Name / Type
Апаратна конфігурація	Мінімальна: - Процесор: amd ryzen 5 1600; - Відеокарта: GTX 1650; - Озу: 8гб;
	Рекомендована: - Процесор: amd ryzen 5 3600; - Відеокарта: GTX 2070 - Озу: 16гб;

Рисунок Г.19 – Сторінка 19 тест-плану

Задачі проекту (Appendix A Project Tasks)

Нижче наведені завдання, пов'язані з тестом:

1. Спланувати тест
– визначити вимоги до тесту;
– розробити стратегію тестування;
– створюємо розклад;
– генеруємо план тестування.
2. Проектування тесту
– визначаємо та описуємо тестові кейси;
3. Реалізація тесту
– записувати або програмувати тестові скрипти;
– визначити специфічну для тестів функціональність в моделі проектування та реалізації;
– створення зовнішніх наборів даних.
4. Виконання тесту
– виконати тестові процедури;
– реєстрація дефектів.
5. Оцінити тест
– аналіз дефектів;
– визначити, чи були досягнуті критерії завершення тесту та критерії успіху.

Рисунок Г.20 – Сторінка 20 тест-плану

ДОДАТОК Д

Приклад програмного коду Header BPC_HitTrace

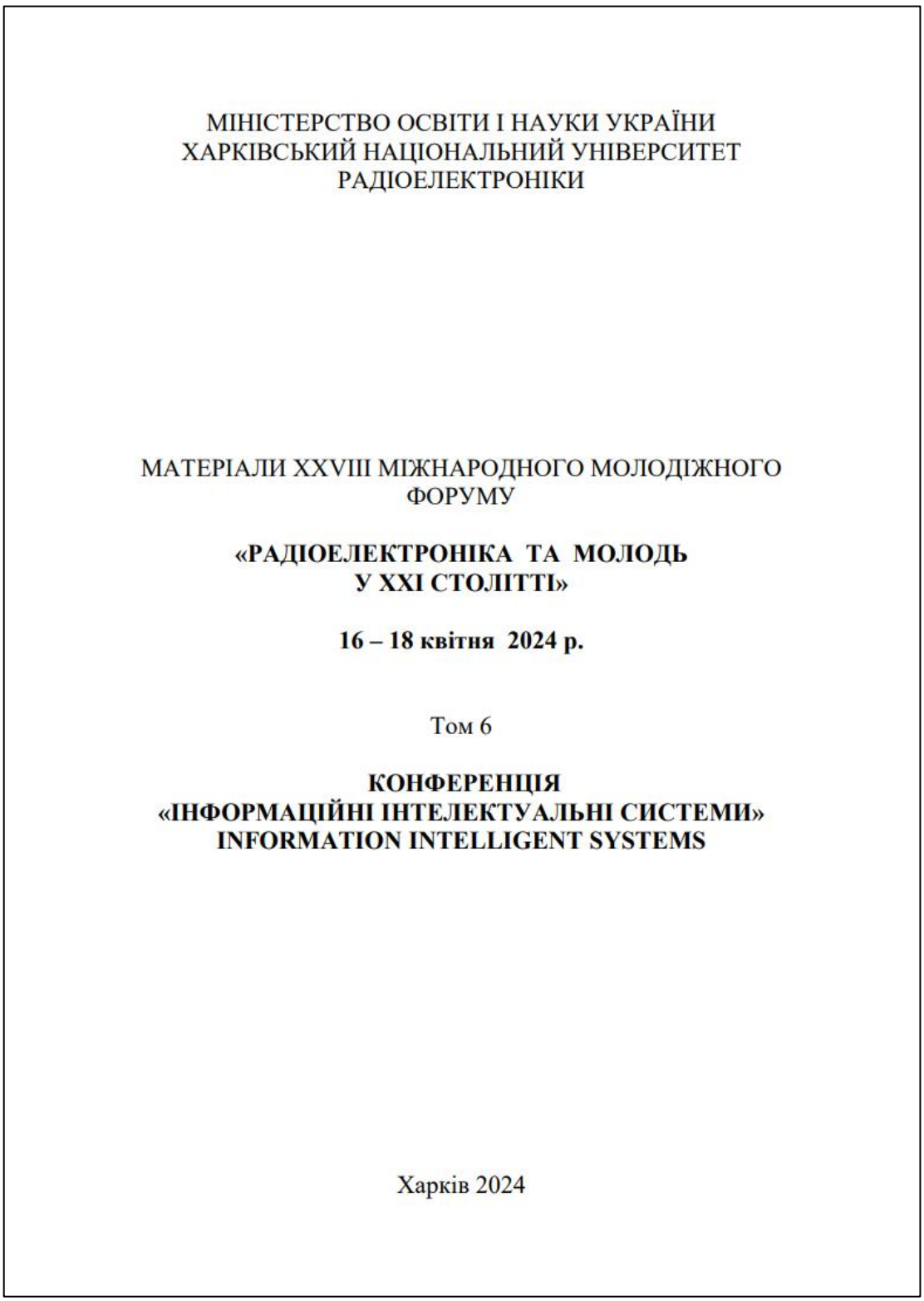
```
/** Responsible for executing traces between provided sockets and
    dispatching events when the actor is hit by hit with unique index
    */
UCLASS(Blueprintable, BlueprintType)
class UBPC_HitTrace : public UActorComponent
{
    GENERATED_BODY()
protected:
    /** Main function which executed trace for single trace data */
    UFUNCTION(BlueprintCallable)
    void CollisionTrace(FS_HitTraceData traceData, AActor* HitActor,
    int32 AttackIndex, FHitResult HitResult, TArray<AActor*>
    HitActors, FS_ActorsArray ActorsWrapper);

    /** Trace between all points */
    UFUNCTION(BlueprintCallable)
    void MultiPartsTrace();
public:
    /** Add new trace data to points array */
    UFUNCTION(BlueprintCallable)
    void AddTraceData(const FS_HitTraceData& TraceData);

    /** Remove trace data from points array */
    UFUNCTION(BlueprintCallable)
    void RemoveTraceData(const int32& AttackIndex, bool Removed);

    /** Returns true if there is at least one trace data in the
    array, false overwise */
    UFUNCTION(BlueprintPure)
    void IsTracing(bool& IsTracing);

    /** Processes hit results from trace by adding new actors to
    AlreadyHitActors array and calling OnHit dispatcher event */
    UFUNCTION(BlueprintCallable)
    void ProcessHitResults(int32 AttackIndex, UPARAM(ref)
    TArray<FHitResult>& HitResults, FS_ActorsArray ActorsWrapper,
    FHitResult HitResult, TArray<AActor*> HitActors);

    /** Array of actors to ignore while tracing */
    UPROPERTY(EditDefaultsOnly, Category="Default")
    TArray<AActor*> ActorsToIgnore;
public:
    /** Event is despatched when the target is hitted by trace first
    time per hit */
    DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnHit, FHitResult,
    HitResult, int32, AttackIndex);
    UPROPERTY(BlueprintAssignable, EditDefaultsOnly,
    Category="Default")
    FOnHit OnHit;
```

```
    /** Determines if the debug shape should be drawn while tracing
    */
    UPROPERTY(BlueprintReadWrite, EditAnywhere, Category="Debug")
    TEnumAsByte<EDrawDebugTrace::Type> Draw Debug Type;
private:
    /** Determines how often trace executed */
    UPROPERTY(EditDefaultsOnly, Category="Default")
    double UpdateTime;

    /** Array of all trace points between which trace should be done
    */
    UPROPERTY(EditDefaultsOnly, Category="Default")
    TArray<FS_HitTraceData> TracePointsArray;

    /** Timer which executes MultiPartsTrace function */
    UPROPERTY(EditDefaultsOnly, Category="Default")
    FTimerHandle DataTracedTimer;
public:
    /** The time which determines how long debug shape should be
    displayed */
    UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Debug")
    float DrawTime;

    /** Event is dispatched when the last trace data is removed from
    TracePointsArray */
    DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnAllTracesEnd);
    UPROPERTY(BlueprintAssignable, EditDefaultsOnly,
    Category="Default")
    FOnAllTracesEnd OnAllTracesEnd;

    /** Event is dispatched when the first trace data is added to
    TracePointsArray */
    DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnFirstTraceStart);
    UPROPERTY(BlueprintAssignable, EditDefaultsOnly,
    Category="Default")
    FOnFirstTraceStart OnFirstTraceStart;

    /** Determines if the owner of component should be ignored in
    traces */
    UPROPERTY(BlueprintReadWrite, EditDefaultsOnly,
    Category="Default")
    bool bIgnoreSelf;
private:
    /** Map stores index of attack and all hitted actors during it in
    a struct */
    UPROPERTY(EditDefaultsOnly, Category="Default")
    TMap<int32,FS_ActorsArray> AttackHittedActors;

    /** Trace channel on which trace is executed */
    UPROPERTY(EditDefaultsOnly, Category="Default")
    TEnumAsByte<ETraceTypeQuery> Trace Channel;
public:
    /** Pointer to the component which containing sockets between
    which trace should be done */
    UPROPERTY(BlueprintReadWrite, EditDefaultsOnly,
    Category="Default")
    TObjectPtr<USceneComponent> SocketsComponent;
```

```
private:
    /** Please add a variable description */
    UPROPERTY(EditDefaultsOnly, Category="Default")
    TMap<int32,UBP_ActorArray_C*> AttackHittedActors_0;
};
```

ДОДАТОК Е

Тези доповіді для науково-практичної інтернет-конференції



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ XXVIII МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

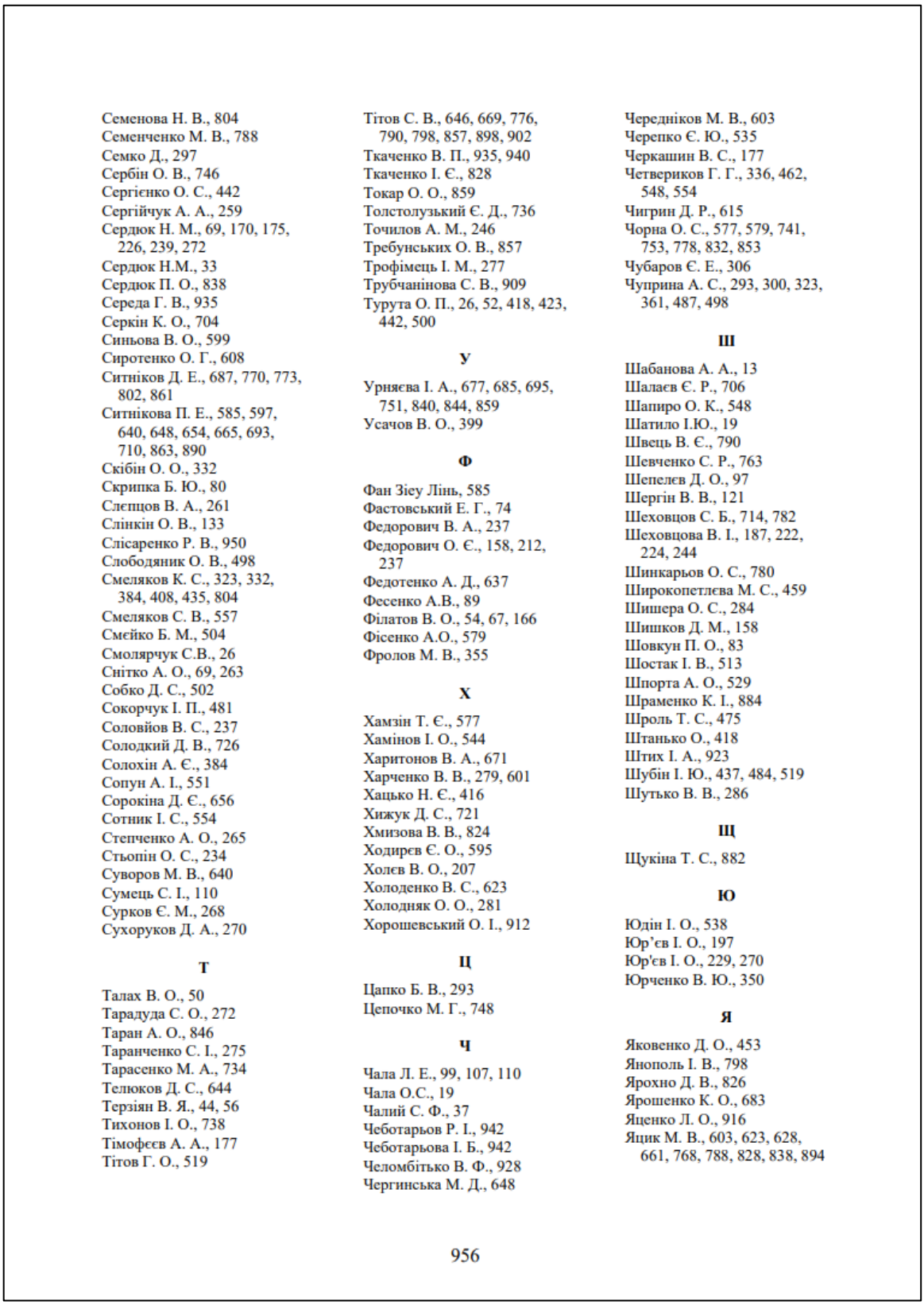
16 – 18 квітня 2024 р.

Том 6

КОНФЕРЕНЦІЯ
«ІНФОРМАЦІЙНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ»
INFORMATION INTELLIGENT SYSTEMS

Харків 2024

Рисунок Д.1 – Обкладинка збірника



Семенова Н. В., 804
Семенченко М. В., 788
Семко Д., 297
Сербін О. В., 746
Сергієнко О. С., 442
Сергійчук А. А., 259
Сердюк Н. М., 69, 170, 175, 226, 239, 272
Сердюк Н.М., 33
Сердюк П. О., 838
Середа Г. В., 935
Серкін К. О., 704
Синьова В. О., 599
Сиротенко О. Г., 608
Ситніков Д. Е., 687, 770, 773, 802, 861
Ситнікова П. Е., 585, 597, 640, 648, 654, 665, 693, 710, 863, 890
Скібін О. О., 332
Скрипка Б. Ю., 80
Слєпцов В. А., 261
Слінкін О. В., 133
Слісаренко Р. В., 950
Слободяник О. В., 498
Смеляков К. С., 323, 332, 384, 408, 435, 804
Смеляков С. В., 557
Смєйко Б. М., 504
Смолярчук С.В., 26
Снітко А. О., 69, 263
Собко Д. С., 502
Сокорчук І. П., 481
Соловйов В. С., 237
Солодкий Д. В., 726
Солохін А. Є., 384
Сопун А. І., 551
Сорокіна Д. Є., 656
Сотник І. С., 554
Степченко А. О., 265
Стьопін О. С., 234
Суворов М. В., 640
Сумець С. І., 110
Сурков Є. М., 268
Сухоруков Д. А., 270

Т

Талах В. О., 50
Тарадуда С. О., 272
Таран А. О., 846
Таранченко С. І., 275
Тарасенко М. А., 734
Телюков Д. С., 644
Терзіян В. Я., 44, 56
Тихонов І. О., 738
Тімофєєв А. А., 177
Тітов Г. О., 519
Тітов С. В., 646, 669, 776, 790, 798, 857, 898, 902
Ткаченко В. П., 935, 940
Ткаченко І. Є., 828
Токар О. О., 859
Толстолузький Є. Д., 736
Точилов А. М., 246
Требунських О. В., 857
Трофімець І. М., 277
Трубчанінова С. В., 909
Турута О. П., 26, 52, 418, 423, 442, 500

У

Урняєва І. А., 677, 685, 695, 751, 840, 844, 859
Усачов В. О., 399

Ф

Фан Зіеу Лінь, 585
Фастовський Е. Г., 74
Федорович В. А., 237
Федорович О. Є., 158, 212, 237
Федотенко А. Д., 637
Фесенко А.В., 89
Філатов В. О., 54, 67, 166
Фісенко А.О., 579
Фролов М. В., 355

Х

Хамзін Т. Є., 577
Хамінов І. О., 544
Харитонов В. А., 671
Харченко В. В., 279, 601
Хацько Н. Є., 416
Хижук Д. С., 721
Хмизова В. В., 824
Ходирєв Є. О., 595
Холєв В. О., 207
Холоденко В. С., 623
Холодняк О. О., 281
Хорошевський О. І., 912

Ц

Цапко Б. В., 293
Цепочко М. Г., 748

Ч

Чала Л. Е., 99, 107, 110
Чала О.С., 19
Чалий С. Ф., 37
Чеботарьов Р. І., 942
Чеботарьова І. Б., 942
Челомбітько В. Ф., 928
Чергинська М. Д., 648
Чередніков М. В., 603
Черепко Є. Ю., 535
Черкашин В. С., 177
Четвериков Г. Г., 336, 462, 548, 554
Чигрин Д. Р., 615
Чорна О. С., 577, 579, 741, 753, 778, 832, 853
Чубаров Є. Е., 306
Чуприна А. С., 293, 300, 323, 361, 487, 498

Ш

Шабанова А. А., 13
Шалаєв Є. Р., 706
Шапиро О. К., 548
Шатило І.Ю., 19
Швець В. Є., 790
Шевченко С. Р., 763
Шепелєв Д. О., 97
Шергін В. В., 121
Шеховцов С. Б., 714, 782
Шеховцова В. І., 187, 222, 224, 244
Шинкарьов О. С., 780
Широкопетлєва М. С., 459
Шишера О. С., 284
Шишков Д. М., 158
Шовкун П. О., 83
Шостак І. В., 513
Шпорта А. О., 529
Шраменко К. І., 884
Шроль Т. С., 475
Штанько О., 418
Штих І. А., 923
Шубін І. Ю., 437, 484, 519
Шутько В. В., 286

Щ

Щукіна Т. С., 882

Ю

Юдін І. О., 538
Юр'єв І. О., 197
Юр'єв І. О., 229, 270
Юрченко В. Ю., 350

Я

Яковенко Д. О., 453
Янополь І. В., 798
Ярохно Д. В., 826
Ярошенко К. О., 683
Яценко Л. О., 916
Яцик М. В., 603, 623, 628, 661, 768, 788, 828, 838, 894

Рисунок Д.2 – Алфавітний покажчик збірника

УДК 004.92 DOI: https://doi.org/10.30837/IYF.IIS.2024.535

МАСШТАБУВАННЯ VFX ЯК СПОСІБ ОПТИМІЗАЦІЇ В UNREAL ENGINE

Черепко Є. Ю.
Науковий керівник – ст. викл. Новіков Ю. С.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна
email: yevhen.cherepko@nure.ua

An important part of developing any visual effects in the media is their optimization. It is a key point, because optimization expresses the quality of the product, affects its perception and evaluation by the public. One of the most powerful ways to optimize visual effects in the Unreal Engine is through the scaling system. It allows you to flexibly cool down VFX depending on the distance to them or the load of systems. That reduces the load on the main engine processes which leads to better performance results.

Важливою частиною розробки будь-яких візуальних ефектів у медіа є їх оптимізація. Вона є ключовим моментом, адже оптимізованість виражає якість продукту, впливає на його відчуття та оцінку публіки. Процес оптимізація полягає в використанні різних технік при розробці ефектів, щоб зменшити загальне споживання ресурсів систем.

Незважаючи на платформу чи рівень потужності пристрою, кінцевий час обробки кожного відмальованого кадру буде залежати від багатьох факторів. Зокрема в Unreal Engine ці фактори впливають на три центральні системи – процес гри, процес рендеру та на графічний процесор [1]. Важливо зазначити, що результат систем передається послідовно, від системи до системи, при цьому вони виконуються паралельно одна одній. Таким чином, час обробки одного кадру зазвичай буде дорівнювати найбільшому часу відпрацювання з трьох вищезазначених систем.

Одним з основних факторів, які напряму впливають на завантаженість GPU є перемалювання. Перемалювання, або ціна шейдерів пікселів визначається об’ємом екранного простору, який займається ефектом та його кількість інструкцій, які витрачаються на обробку. Він напряму пов’язаний з прозорими матеріалами, адже кожен їх шар потребує правильного сортування та змішування текселів при рендері [2], що значно підвищує час обробки.

Спрощення матеріалів, та зміна їх типу змішування є прямими варіантами зменшення кінцевої кількості перемалювань пікселів. Однак, є ситуації коли технічний художник повинен використати складний матеріал, при цьому зберігаючи оптимальний рівень продуктивності системи. І на цьому етапі в силу входять профілі масштабування.

Профілі масштабування – це бібліотеки поведінок «охолодження» систем залежно від визначених умов. Під охолодженням мається на увазі

Рисунок Д.3 – Матеріал тез (сторінка 1)

спрощення, або виключення систем для збереження ресурсів пристрою користувача. Unreal Engine дозволяє використовувати ці профілі для систем частинок Niagara, як на глобальному рівні, так і на рівні окремих систем чи випромінювачів.

Основні налаштування профілів включають: тип охолодження – система вмирає при виконанні умови, або впадає у сон; частота перевірки умови; пріоритет вибору системи, яка збереже життя. Для визначення умов охолодження можна використати відстань до системи, їх кількість, або залежність кожного з них від поточних споживаних ресурсів апаратного приладу та інші. Умови профілів задаються для цільових рівнів графіки та платформ.

Доцільним варіантом використання масштабування є його впровадження з відстанню до ефекту, адже збільшуючи відстань загальна кількість пікселів заціплених ефектом зменшуються, однак, складність перемалювання збільшується, через більш щільне шарування частинок. Крім того, навантаження зберігається і на процесі рендерингу, адже кожна частинка повинна бути оброблена для передачі даних на GPU.

Так, для відмалювання 10 систем Niagara, кожна з яких породжує 100 000 частинок можна визначити наступну залежність завантаженості систем гри від відстані до систем Niagara:

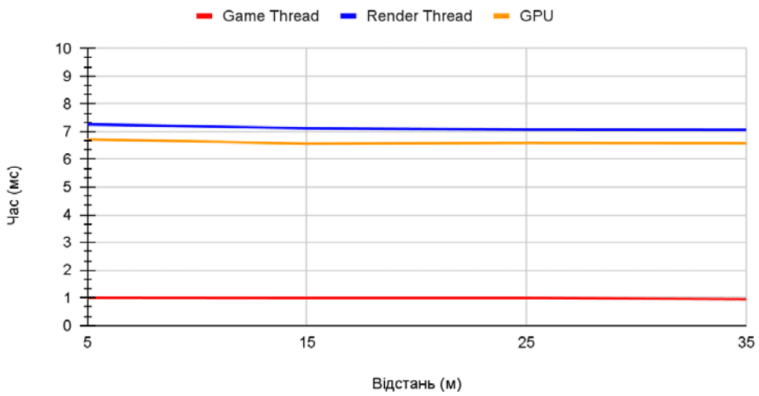


Рисунок 1 – Залежність часу обробки кадру від відстані
в початковому вигляді

З діаграми на рисунку 1 можна побачити майже сталу завантаженість систем з відстанню.

Повний потенціалом масштабування можна скористатися напряму у випромінювачах частинок. Наприклад, можна створити лінійну залежність між дистанцією до об’єкта та кількістю породжуваних частинок. При цьому з відстанню можна збільшувати розмір частинок. Це зменшить кількість перемалювання, зберігаючи плавність і щільність ефекту з відстанню. Наведена діаграма на рисунку 2 зображує час витрачений

Рисунок Д.4 – Матеріал тез (сторінка 2)

кожної системою з відстанню де кількість породжуваних частинок кожної системи зменшується з 100 000 до 100 лінійною залежністю від дистанції:

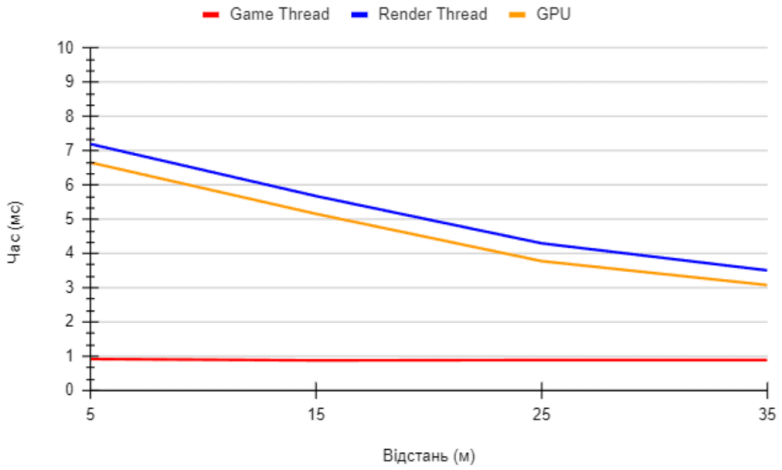


Рисунок 2 – Залежність часу обробки кадру від відстані після налаштування масштабування

З результатів маємо майже лінійну криву, де з відстанню зменшується час обробки кадру на ігровому та рендер потоці, приблизно у два рази. Враховуючи показники з пустою системою Niagara, де час рендеру складає біля 1,5 секунди, загальний час рендеру системи Niagara зменшився у 2,65 рази.

Таким чином, пошук балансу між візуальною складовою та спрощеннями візуальних ефектів є ключовим фактором при розробці медіа. Масштабування є потужним інструментом для оптимізації ефектів залежно від поточних ресурсів пристрою, платформи, або відстані до об’єкта. Це сприяє зменшенню динамічних витрат основних систем Unreal Engine, без вагомих збитків візуальної складової продукту при належному налаштуванні інструменту.

Список використаних джерел:

1. VFX Optimization Guide // Unreal Engine Documentation. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Optimization/Results/ (дата звернення: 05.03.2024).

2. Матвєєв Д.І., Лановий О.Ф., Методи спрощення опрацювання систем симуляції незалежних часток на основі середовища Unreal Engine 4 // Електронне моделювання, 2023, Том 45, № 2, с. 95-107.

Рисунок Д.5 – Матеріал тез (сторінка 3)