Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп’ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

здобувача Афанасьєва Артема Олександровича
(ПІБ)

академічної групи 123-21-1
(шифр)

спеціальності 123 Комп’ютерна інженерія
(код і назва спеціальності)

за освітньо-професійною програмою 123 Комп’ютерна інженерія
(офіційна назва)

на тему “Комп'ютерна система з хмарним розміщенням для інтеграції та підтримки відеоігри "Forged from Ruins" ”
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	доц. Каштан В.Ю.			
спеціального розділу	доц. Каштан В.Ю.			
розділу розробка корпоративної мережі	доц. Каштан В.Ю.			

Рецензент				

Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро

2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)

_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)

"25" лютого 2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавр

здобувача Афанасьєва А.О. **академічної групи** 123-21-1
(прізвище та ініціали) (шифр)

спеціальності 123 «Комп'ютерна інженерія»

за освітньо-професійною програмою 123 «Комп'ютерна інженерія»
(офіційна назва)

на тему "Комп'ютерна система з хмарним розміщенням для інтеграції та підтримки відеоігри "Forged from Ruins" "

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 № 336-с

Розділ	Зміст	Термін виконання
Стан питання і постановка завдання	На основі матеріалів виробничих практик, інших науково-технічних джерел показати актуальність завдання, сформулювати мету та задачі виконання кваліфікаційної роботи	10.05.25
Розробка апаратної частини	Виконати технічне проєктування апаратної частини комп'ютерної системи з необхідними інженерними розрахунками	17.05.25
Розробка корпоративної мережі	Розрахувати й розподілити адреси вузлів комп'ютерної системи, розробити заходи з обмеження доступу до даних системи	24.05.25
Розробка компонента системи	Обґрунтувати технічні характеристики програми й розробити прототип відеогри «Forged From Ruins»	31.05.2025

Завдання видано _____ доц. Каштан В.Ю.
(підпис керівника) (прізвище, ініціали)

Дата видачі 25.02.2025

Дата подання до екзаменаційної комісії 16.06.2025

Прийнято до виконання _____ Афанасьєв А.О.

РЕФЕРАТ

Пояснювальна записка: 123 с., 43 рис., 4 табл., 2 дод., 18 джерел.

РОЗРОБКА 2D ГРИ, ІГРОВИЙ РУШІЙ UNITY, КОМП’ЮТЕРНА СИСТЕМА, ХМАРНЕ РОЗМІЩЕННЯ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ.

Об’єкт розробки − комп'ютерна система з хмарним розміщенням для інтеграції та підтримки відеоігри "Forged from Ruins".

Мета роботи – проведення аналізу індустрії відеоігор та визначення шляхів розробки прототипу відеогри у жанрі 2D-Roguelite з можливістю розміщення у хмарі, а також розробка гнучкої комп’ютерної мережі підприємства для підтримки продукту та забезпечення стабільної та ефективної роботи підприємства.

Здійснено розробку універсальної корпоративної мережі з врахуванням вимог до надійного рівня безпеки а також швидкої передачі даних. Система переважно орієнтована на підтримку роботи відеоігрової студії, яка займається розробкою відеоігор з можливістю хмарного розміщення та віддаленого доступу. В якості спеціального елемента системи розроблено прототип відеоігри у жанрі 2D Roguelite.

При розробці прототипу було реалізовано наступні ключові елементи: процедурна генерація ігрового досвіду, піксельний візуальний стиль, модульна архітектура побудови коду.

У комп’ютерній мережі було застосовано та налаштовано такі технології: визначені підмережі розділено на логічні сегменти за допомогою VLAN та реалізовано видачу IP адрес за DHCP відповідно до VLAN, між основними відділами компанії та її віддаленим офісом налаштовано захищений IPsec канал із застосування VPN, зв’язок основних відділів із зовнішнім світом відбувається з використанням технології NAT, для серверів компанії налаштовано статичні адреси, налаштовано DNS сервер та HTTPS / FTP сервер для імітації хмарного розміщення.

Корпоративну мережу було побудовано та протестовано із застосуванням програми Cisco Packet Tracer.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Roguelike – назва ігрового жанру що походить від слів Rogue (назва гри) та like (схожий);

Roguelite – назва ігрового піджанру Rogelike, що походить від слів Rogue (назва гри) та lite (полегшений);

Контент – інформаційне наповнення системи;

Механіка – набір правил і способів, який реалізує певним чином деяку частину інтерактивної взаємодії гравця і гри;

ПЗ – програмне забезпечення;

Забіг – переклад англійського терміна Run, який використовується у іграх жанру Roguelike та Roguelite;

Реграбельність – показник, що визначає ступінь бажання гравців зіграти у гру ще раз, навіть після проходження або витрати великої кількості часу;

Steam – онлайн платформа для продажу відеоігор;

Спрайт – двовимірне зображення, що використовується у комп’ютерній графіці, часто, як частина більшого зображення;

Тайл – невеликий графічний фрагмент квадратної форми, що розташовується у сітці;

Префаб – шаблон ігрового об’єкту;

Хакатон – захід під час якого розробники згуртовано працюють над вирішенням якоїсь проблеми або розробкою нового продукту.

Кооперативний – режим відеогри, в якому кілька людей грають одночасно для проходження гри;

Рендер – процес отримання та візуалізації графічного зображення на екрані;

ООП – об’єктно орієнтоване програмування;

Босс – термін, що позначає неймовірно потужного суперника;

ВСТУП

На сьогоднішній день відеоігри займають велике місце у сфері цифрових розваг, на рівні з кінематографом, музикою, соціальними мережами та іншими індустріями. Це обумовлюється широкою доступністю високопродуктивних пристроїв та комп’ютерів з можливостями, які не були доступні ще 30 років тому. Кожен власних сучасного смартфону, ПК або ноутбуку має технічну можливість насолоджуватися широким спектром відеоігрових продуктів. Незалежні розробники та невеликі студії мають високу частку ринку та відіграють важливу роль у формуванні нових тенденцій та розвитку жанрів.

Комп’ютерні мережі та сфера телекомунікацій також розвивали технології зв’язку для задоволення зростаючого попиту на швидкий та стабільний зв’язок на початку 2000-х років. У сучасному світі доступ до швидкого та стабільного підключення до мережі Інтернет є найлегшим за усі часи, що дозволяє розвиватися хмарним технологіям. Відеоігрова індустрія також не залишилась осторонь і почала реалізовувати проєкти з хмарного розміщення не тільки даних а і апаратних ресурсів, що дозволяє грати з будь-якого пристрою з достатньою швидкістю Інтернет-з’єднання, навіть у вимогливі ігри.

Актуальність цієї теми зумовлена постійним та стабільним зростанням розробників на ринку, наявним попитом на відеоігри, що пропонують унікальний досвід при кожному запуску. Інструментарій для розробки відеоігор стає доступнішим для широкого загалу з кожним роком, що знижує технічні бар’єри для розробників початківців. Таким чином, індустрія постійно зростає як у кількості зацікавлених клієнтів так і розробників, що готові випускати нові інноваційні проекти на ринок. Хмарний геймінг також не стоїть на місці, з кожним роком його частка на ринку зростає, що значно знижує поріг входу для звичайного користувача у світ відеоігор. Проте це приносить власні виклики, адже постійне транслювання якісного відеосигналу потребує великих мережевих ресурсів. Дана проблема буде адресована та врахована при побудові корпоративної мережі, а також запропоновано рішення у вигляді

високошвидкісного каналу зв’язку з використанням оптичних кабелів.

Мета цієї роботи – дослідити та описати фундаментальні принципи, механіки, механізми та дизайнерські рішення що знадобляться для побудови прототипу 2D-Roguelite гри. Буде проведено аналіз ігрового ринку, існуючих рішень та визначено, як організувати створення такої гри в рамках сучасних реалій розробки ПЗ. Іншою метою є розробка корпоративної мережі, що зможе підтримувати роботу підприємства з розробки відеоігор, а також підтримувати можливість хмарного розміщення та хмарного геймінгу з використанням актуальних технологій та протоколів захисту та зв’язку.

1 СТАН ПИТАННЯ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Актуальність побудови та розвитку відеоігрових проектів у жанрі 2D-Roguelite.

Історія жанру бере свій початок у 1980 році з випуски гри Rogue, яка однією з перших представила гравцям процедурно згенеровані ігрові елементи. Головними особливостями гри були: остаточна смерть ігрового персонажа, покрокова система бою, рух персонажа на основі сітки, випадкового згенеровані рівні та високий рівень механічної складності. Ці елементи стали основою жанру Roguelike, який став підґрунтям для багатьох адаптацій. Згідно до статті на ресурсі “Temple Of The Roguelike” [1], “справжні” представники жанру повинні мати 5 критичних аспектів:

− незворотні наслідки;

− орієнтованість на головного персонажа;

− процедурна генерація контенту;

− покроковість;

− чистота забігів.

Цей перелік означає, що гравець представляє єдиного персонажа, гра повинна бути побудована таким чином, що гравець отримує винагороду за використання своїх зростаючих знань та ступінь адаптації до процедурно згенерованих викликів, отримуючи кардинально інші результати порівняно із початковими забігами, зберігаючи при цьому однакові стартові умови запуску кожного нового забігу. Відеоігри продовжили розвиватися, і це не оминуло жанр Roguelike, який з роками еволюціонував та породив новий піджанр, який частково послабив вимоги до наслідування усіх історичних елементів жанру, проте зберіг відчуття непередбачуваності, складні виклики та реграбельність. Ігри жанру Roguelite відрізняються тим, що додають нові механіки та особливості до класичних елементів жанру, наприклад, можливість розвивати персонажа з кожним наступним забігом, що полегшує гру не тільки на рівні адаптації гравця, а і на механічному.

Жанр Roguelite пропонує досить вільні рамки для розробки 2D гри, яка буде балансувати складність, випадковість і поступовий зріст персонажа гравця. Розробляючи таку гру, метою є забезпечення ігрового досвіду, який винагороджує адаптивність, майстерність та наполегливість гравців. На ринку існує явна потреба та зацікавленість у нових продуктах в даному жанрі, наприклад, гра Balatro, що вийшла у 2024 році та перетворила класичну гру покер у чудового представника жанру Roguelite (рисунок 1.1), який був номінований на звання “Гра року” на церемонії The Game Awards. Іншим прикладом успішного продукту є гра Don’t Starve Together 2016 року випущена студією у 35 співробітників, яка згідно статистичних даних [2] продалася приблизним тиражем у 32.7 мільйонів копій та принесла розробникам 205 мільйонів доларів. При цьому навіть 9 років після свого випуску вона продовжує розвиватися та зберігати стабільну аудиторію гравців, які в середньому складають 28 тисяч одночасних користувачів щодня. Гра є кооперативним Roguelite у відкритому світі зі стилізованим унікальним візуальним стилем та елементами виживання персонажа у світі повному ворожих істот (рисунок 1.2).

Рисунок 1.1 – Приклад ігрового процесу Balatro

Ілюстрацію взято з офіційної сторінки продукту на платформі Steam

Рисунок 1.2 – Ілюстрація ігрового процесу Don’t Starve Together
Ілюстрацію взято з офіційної сторінки продукту на платформі Steam

1.2 Характеристика, структура, особливості розробки та роботи 2D-відеоігор

На відміну від ігор, які побудовані навколо тривимірного руху персонажа у середовищі, 2D-відеоігри обмежують рух гравця лише горизонтальною та вертикальною осями. Введення таких обмежень ставить перед розробником чіткі рамки для дизайну ігрового процесу, що заохочує більшу сфокусованість на основних механіках, що призводить до більш ефективного процесу розробки, тестування та шліфування продукту перед публікацією. Більшість 2D-ігор в якості візуального стилю використовують піксельну, векторну або мальовану графіку. Дані стилі є простими для опанування для розробників, що не мають дизайнерською спеціалізації але мають бажання втілювати своє дизайнерське бачення в життя. Простота візуального зображення впливає також і на сприйняття гравця, дані візуальні стилі часто асоціюються з роботами розробників початківців та малих студій, проте це не означає що вони є поганим вибором для

більших проектів.

Через свої обмеження, 2D-ігри часто мають прості для розуміння механіки, які водночас надають виклик гравцю та постійно перевіряють його знання та навички. Простота навігації персонажа у двовимірному просторі змушує розробників робити інші ігрові взаємодії, такі як: стрільба, використання ігрових предметів, ухилення та інші, більш вимогливими до навичок гравця.

Доступність – ще одна перевага двохвимірної графіки, тому що вона має низькі вимоги до апаратного забезпечення. Можливість відтворення на широкому спектрі пристроїв дозволяє розширити потенційну аудиторію та дозволяє випускати проект на різних ринках. Проте простота може спричинити відчуття однотипності, це призводить до необхідності створення ілюзії глибини за допомогою художнього стилю та спеціальних технік, для обману сприйняття використовується перспектива малюнку, кольори для імітації тіней та інші техніки [3]. Графічна частина проєкту зберігається у різних форматах в залежності від типу об’єкту. Дизайн персонажів часто зберігається у вигляді спрайтів або наборів спрайтів для створення анімацій та відображення на екрані, елементи рівнів такі як стіни, підлога та декорації часто групуються у єдиний файл який називається набором тайлів або атласом, в залежності від стилістики та перспективи візуального стилю.

Об’єкти у сучасних відеоіграх часто виконуються у вигляді префабів, які використовуються в якості багаторазових шаблонів, які часто використовуються у ігровому світі таких як персонажі, вороги, об’єкти для взаємодії або снаряди. Незалежно від свого місцезнаходження вони зберігають свій зв’язок з оригінальним префабом, що знаходиться у вигляді файлу або об’єкту у структурі гри.

На сьогоднішній день, розробка відеоігор переросла з простого хобі вузького кола програмістів у окрему сферу виробництва програмного забезпечення, це означає, що сучасний розробник має доступ до широкого спектру ресурсів для проєктування та побудови гри на будь-який смак. Набори графічних елементів, інтерфейсів, звуків, музики, анімацій, програмного коду та інструменти для

розробки знаходяться у вільному доступі для вільного використання або покупки, що дозволяє значно пришвидшити процес створення прототипу або фінального продукту. Існують цілі спільноти, що займаються розробкою ігор та активно обмінюються навчальними матеріалами, найкращими практиками, практичними та технічними рішеннями з написання окремих елементів коду. Такі спільноти також часто проводять хакатони з розробки ігор за певний проміжок часу, із заданою темою, стилістикою або жанром, що дозволяє отримати практичний досвід (рисунок 1.3).

Робота будь-якої гри є результатом постійної взаємодії багатьох систем та структурних елементів. Незалежно від використаних технологій, ігрових рушіїв та архітектури, структура будь-якої 2D-гри складається з декількох основних компонентів, що працюють синхронно у реальному часі.

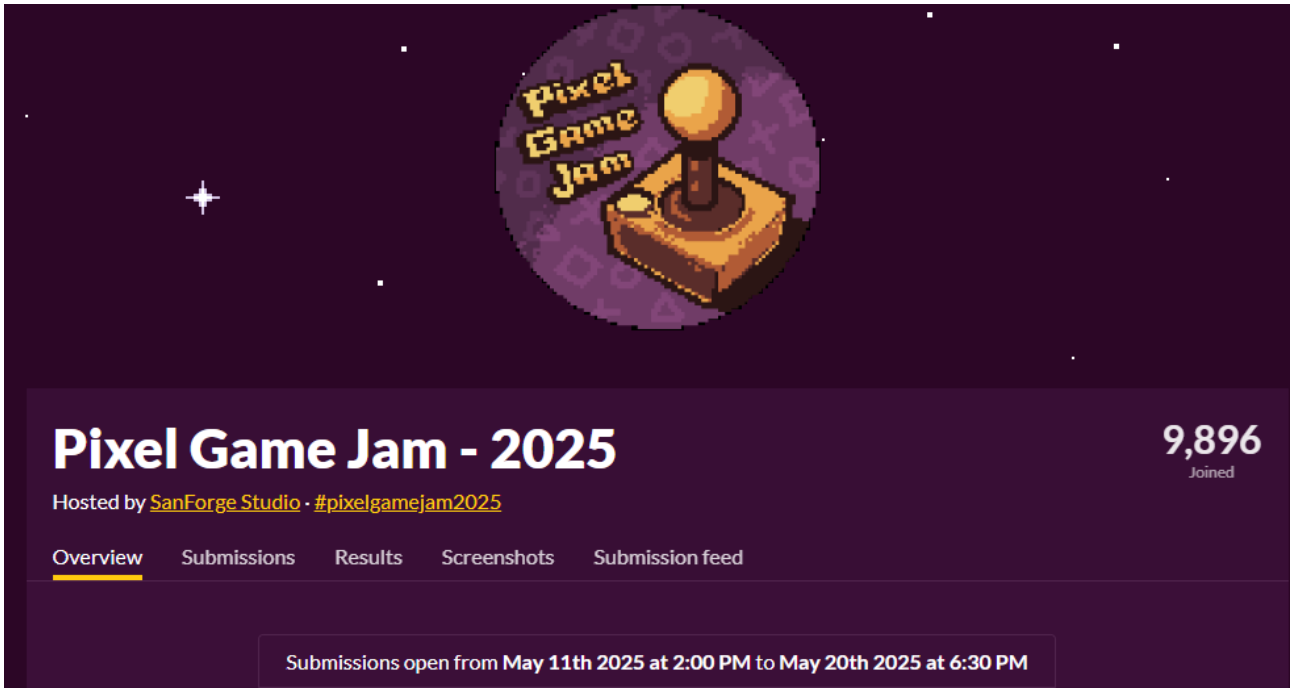

Рисунок 1.3 – 10 денний хакатон з розробки гри у піксельному стилі

Ілюстрацію взято з офіційної сторінки хакатону: https://itch.io/jam/-pixel-game-jam-2025

1.3 Огляд існуючих аналогів, технологій, архітектур та програмних рішень для виконання 2D-Roguelite проєкту

Розуміння ключових особливостей і тенденцій у розробці сучасних 2D-Roguelite проєктів, дозволить визначити основні елементи, які є важливими для сучасної гри цього жанру, а також вивчити успішні приклади реалізації аналогів.

Гра “Enter The Gungeon” розроблена компанією Dodge Roll була випущена у 2016 році та є одним з найвідоміших прикладів жанру 2D-Roguelite. Гра мала успіх серед широкої аудиторії через свій захопливий візуальний стиль, цікаве поєднання та креативне поєднання базових ігрових механік, а також загальну завершеність проєкту. Структурно, гра складається з багатьох рівнів, що називаються “поверхами”, кожний поверх складається з мережі кімнат, які процедурно розміщаються на рівні та поєднуються згенерованими коридорами. Цікавим є те, що самі кімнати зроблені заздалегідь вручну, це дозволяє забезпечити цікаві, продумані та складні сутички з противниками. Наповнення кімнат ворогами є частково випадковим, що робить проходження одних і тих самих кімнат цікавим навіть після багатьох забігів. Такий гібридний підхід до генерації ігрового досвіду залишає інструменти контролю в руках розробника, проте, за достатньої кількості попередньо створених кімнат, підтримується відчуття випадковості та унікальності кожного забігу. На кожному з рівнів також присутні кімнати, що генеруються з вірогідністю 100%, наприклад: лавка продавця, 2 кімнати зі скринями та інші унікальні локації. Перехід між рівнями можливий лише за умови перемоги над сильним супротивником – боссом, що знаходиться у кінці кожного рівня. Приклад генерації двох рівнів можна побачити на рисунку 1.4, де виділено кімнати, що мають генеруватися при кожному циклі, а також деякі кімнати, що випадково були обрані для обох спроб процедурної генерації.

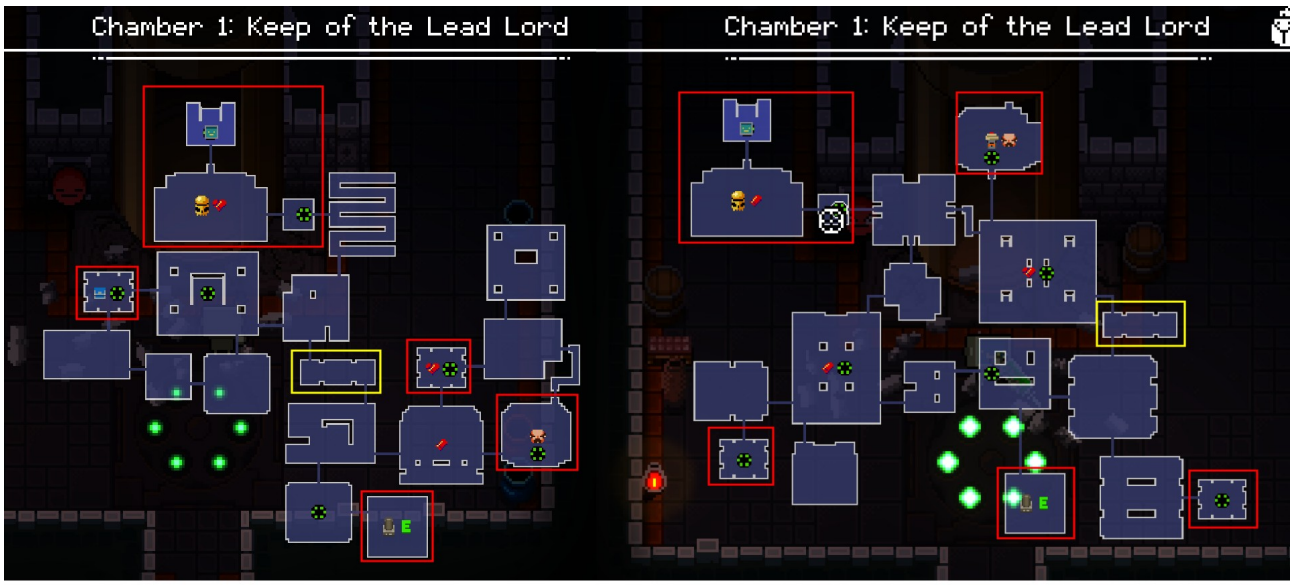

Рисунок 1.4 – Приклад процедурної генерації рівня у Enter The Gungeon

Ігровий процес зосереджений на високоінтенсивних та складних сутичках з супротивниками у більшості кімнат. Однією з основних механік для виживання є ухилення, що дозволяє отримати незначне вікно невразливості головного персонажа. Незважаючи на складність, сутички є чесними для гравця і перемога чи поразка залежать виключно від навичок гравця та не залежать від зовнішніх факторів. Ігровий баланс налаштований на постійне випробування навичок та підвищення ставок, що робить ігровий процес цікавим для вивчення та прогресування. У грі представлено сотні одиниць унікальної зброї та предметів, які мають унікальні ефекти застосування та поведінку. Розподіл зброї та предметів є випадковим, тому змушує гравця адаптувати свою стратегію проходження до наявної ситуації. Незважаючи на те, що випадковість відіграє значну роль у отриманні зброї, баланс гри побудований таким чином, що кожен предмет та одиниця зброї є механічно життєздатними. Налаштування ігрового балансу є одним із найважливіших аспектів у Roguelite іграх, адже через процедурну природу кожного забігу, розробник не може протестувати кожну ігрову ситуацію та підлаштувати її складність у ручному режимі. Для цього часто

розробникам доводиться проводити математичний аналіз систем гри та використовувати статистичні методи та аналіз вірогідностей для забезпечення чесного та захоплюючого ігрового досвіду для гравця. Існують і інші методи, наприклад балансування складності сутичок, балансування прогресії гравця та балансування між ігровими стратегіями [4].

Візуально гра виконана у піксельній стилістиці (рисунок 1.5), яка використовує яскраву палітру та високий рівень деталізації для представлення ігрового світу та його елементів. Продуманий розподіл кольорів між декораціями, персонажами, ворогами та снарядами, а також використання динамічного освітлення дозволяє гравцю орієнтуватися у ігровому хаосі, що відбувається на екрані.

Рисунок 1.5 – Стоп-кадр ігрового процесу Enter The Gungeon
Ілюстрацію взято з офіційної сторінки продукту на платформі Steam

Іншим цікавим прикладом для дослідження став проєкт “Slay The Spire” випущений у 2019 році студією MegaCrit. Вона є впливовою грою у світі 2D-Roguelite через своє поєднання структури Roguelite з механіками побудови

колоди, які належать до жанру колекційних карткових ігор. Вивчення Slay the Spire пропонує цінний досвід, як систематична випадковість, побудова ігрового процесу на основі глибокої стратегічності та мінімалістична естетика можуть бути використані для створення захопливого та тривалого ігрового досвіду.

Система процедурної генерації у грі побудована на основі мапи. Замість того, щоб переміщатися між кімнатами, як у “Enter The Gungeon”, гравці рухаються розгалуженими шляхами на карті, вибираючи з різних типів зустрічей, таких як битви, елітні битви, скарбниці, магазини та місця відпочинку (рисунок 1.6). Кожна мапа генерується наново на початку забігу, при цьому випадково розташовуючи вузли із зустрічами та нагороди за їх проходження.

Рисунок 1.6 – Приклад згенерованої мапи

Ілюстрацію взято з ресурсу: https://interfaceingame.com/

Slay The Spire є одним з представників модульної дизайн архітектури, що дозволила відносно невеликій команді розробників створити гру з великою глибиною механік та високою реграбельністю. Гра мала багато оновлень з новим ігровим контентом, які демонструють надійність цього дизайнерського рішення.

Гра полюбилась ігровим критикам та принесла розробникам комерційний успіх, що показує важливість креативного підходу до створення простих, але глибоких ігрових механік та налагодження ігрового процесу навколо них.

Сучасні розробники відеоігор в своїй більшості використовують спеціальне програмне забезпечення – ігрові рушії, для полегшення написання коду та побудови технічної бази. Вони абстрагують значну частину складних процесів з рендеру графіки, обчислювання фізичних взаємодій об'єктів та управління пам'яттю, дозволяючи розробнику зосереджуватись на написанні безпосередніх ігрових механік. Вибір ігрового рушія є одним із найважливіших технологічних рішень при створенні будь-якої гри, адже вірогідність заміни рушія у процесі розробки без втрати значної частини прогресу написання є мізерною, через високу різницю між архітектурами рушіїв. Для розробки 2D-гри в жанрі Roguelite найчастіше використовуються: Unity, Godot та GameMaker Studio (рисунок 1.7).

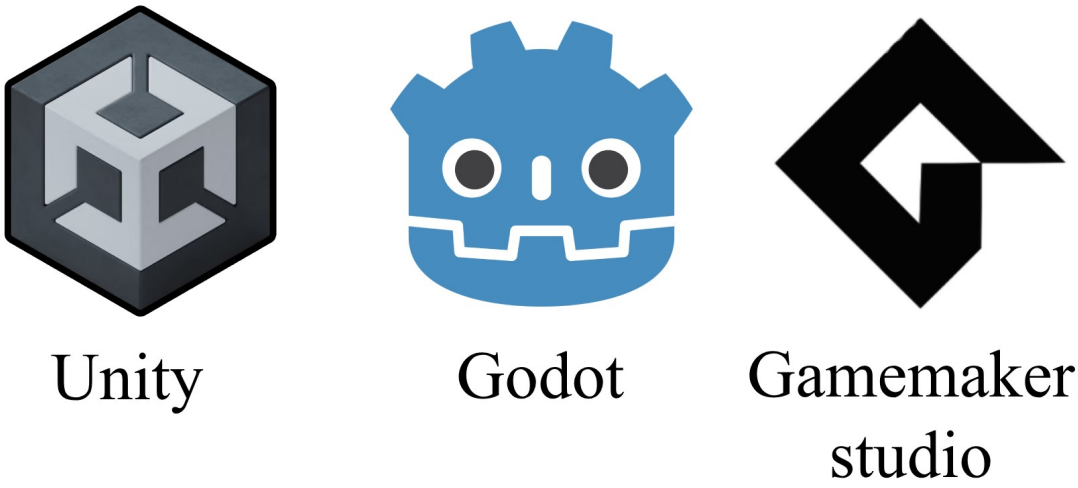

Рисунок 1.7 – Логотипи популярних ігрових рушіїв 2D розробки

Unity пропонує багатофункціональне середовище розробки, яке підтримує розробку ігрового проєкту будь-якого масштабу та жанру. Він має можливість розробки з врахуванням багатьох платформ, що дозволяє значно спростити вихід на нові ринки. Unity має широкий набір спеціалізованих інструментів для роботи з 2D-проєктами та їх елементами. Незважаючи на закритість деякої частини коду, відкрита частина має широку та деталізовану документацію для покращення

адаптації нових розробників. Найбільшою перевагою Unity є його сховище ігрових ресурсів. Компанія надає доступ до платформи, з якої будь-який розробник може завантажувати моделі, звуки, музику, візуальні ефекти, інструменти та інші елементи прямо у проєкт з вікна редактору. На сьогоднішній день платформа надає доступ до більше ніж 75000 ресурсів [5], як в безкоштовному так і платному форматі.

Рушій Godot є ПЗ з відкритим кодом, що здобув популярність серед малих розробників. Він має вбудовану підтримку 2D-рендеру, а також побудований на модульній інфраструктурі, що дозволяє легко вимикати непотрібні елементи рушія. Він має велику спільноту розробників, що постійно діляться власними знаннями та розробляє додаткові інструменти для рушія, проте ця підтримка є далекою від рівня Unity. Через відкритість коду, будь-яка гра створена на рушії Godot не потребує ліцензування, а отже розробник може заробити більше з продаж власного продукту ніж використовуючи рушій з закритим кодом.

GameMaker позиціонує себе як оптимальне рішення для розробки 2D-ігор. Він має дуже простий та інтуїтивний процес розробки, що дозволяє проводити розробку з неймовірною швидкістю. Через свою простоту, даний рушій має певні технічні обмеження, що не дозволяють будувати технічно складні проєкти на його основі. Рушій має закритий код та плату за ліцензування випущених на його основі ігор.

Ігрова архітектура слугує основою для побудови усіх ігрових систем. Для забезпечення модульності побудованого проєкту, що є важливим для ігор Roguelite, необхідно обрати відповідний архітектурний шаблон. Боб Найстром у своєму виступі на шоу Roguelike Celebration 2018 [6], обговорює різні архітектурні рішення, які можуть бути використані для організації коду Roguelike ігор. Серед них є модель «сутність-компонент-система»(ECS), яка є архітектурою, що фокусується на даних , а також інші системи організації коду що керуються об’єктами. В даному виступі було обговорено, що гарною практикою є поділ функцій на окремі компоненти, кожен з яких виконує свою визначену роботу, а також використання власних типів даних та стандартних шаблонів

програмування. Боб Найстром також надає своє бачення оновлених стандартизованих шаблонів програмування, таких як “Команда”, “Спостерігач”, “Одинак” та інших у сучасних реаліях відеоігрової розробки [7].

До задач побудови правильної архітектури також входить керування станами. В Roguelite іграх присутньо багато систем та елементів які мають множини станів, які в свою чергу впливають на загальний стан гри. Повинні бути враховані процедурні динамічно згенеровані сутності, наприклад кімнати, із забезпеченням їх правильної роботи у контексті всієї системи.

При побудові структури коду варто дослухатися деяких рекомендацій з дотримання чистоти та зрозумілості у коді, що наводяться у книзі “Clean Code” Роберта Мартіна та підсумовуються у роботі від Supergloo, Inc. [8]. Гарною практикою при використанні ООП вважається дотримування 5 принципів що формують абревіатуру SOLID розроблених все тим же Робертом Мартіном у 2000 році. Головною ідеєю SOLID є зменшення залежностей між компонентами коду. Через велику кількість залежностей код стає складніше підтримувати та модифікувати, а також його не можна використовувати за межами його контексту [9].

1.4 Характеристика підприємства та умов застосування комп’ютерної системи

Індустрія розробки відеоігор є однією з найбільш динамічних, технологічно насичених та конкурентних у сьогоденному світі. У контексті побудови інформаційно-мережевої інфраструктури, існує багато особливостей та фактів, які необхідно враховувати при побудові.

Сучасні компанії з розробки програмного забезпечення оперують великими обсягами даних у процесі своєї роботи, у випадку ігрових студій, такими даними можуть бути: 3D моделі, 2D спрайти, код продукту, тестові версії гри, звукові ефекти, музика та інші. Критичним фактором у таких випадках є швидкість передачі даних, адже від неї напряму залежить ефективність роботи усієї команди. Також слід зазначити, що усі ці дані, дуже часто, є приватною власністю

компанії та захищені авторськими правами. Розробка однієї відеогри може тривати роками, і несанкціоноване втручання у роботу компанії може відкласти вихід готового продукту, або ж повністю його зупинити. Саме тому важливим аспектом корпоративної мережі відеоігрової студії є її захищеність. Використання рекомендованих безпекових практик, надійних протоколів та технологій а також швидкісної передачі даних дозволяє компанії забезпечувати свою ефективну роботу, відсутність затримок з технічних причин. У роботі студій часто можна побачити використання хмарних сервісів для виконання таких задач як: хостинг ігрових серверів, системи онлайн дистрибуції відеоігор, зберігання даних та інших, тому забезпечення швидкого та надійного зв'язку є важливим для взаємодії з віддаленими сервісами.

Об'єктом впровадження стане відеоігрова студія, що має один великий офіс, у якому розташовано основну мережу підприємства, а також віддалену локацію в якому буде знаходитись один із відділів інженерів програмного забезпечення. Організаційно, компанія розділена на такі структурні підрозділи: маркетинг та управління, дизайнерський відділ, відділ розробки та технічний відділ. Відділи розробки та технічного обслуговування підпорядковуються головному інженеру, дизайнерський відділ має власного керівника – дизайн директора. Відділ маркетингу та управління знаходиться у прямому підпорядкуванні генерального директора студії, так само як і головний інженер та дизайн директор. Більш детальний розподіл відділів зображено на організаційній структурній схемі підприємства (рисунок 1.8).

У відділі розробки нараховується 342 працівників, що враховує молодший керівний склад та головного інженера, дизайнерський відділ складається з 142 осіб, маркетинг та управління мають 137 осіб та в технічний відділ нараховує 44 співробітники. При виборі та плануванні приміщень студії, необхідно врахувати площу для розміщення кожного зі співробітників, одне офісне місце повинно займати не менше 6 м.кв. Відстань між основним офісом компанії та її віддаленим офісом складає 3 кілометри по прямій.

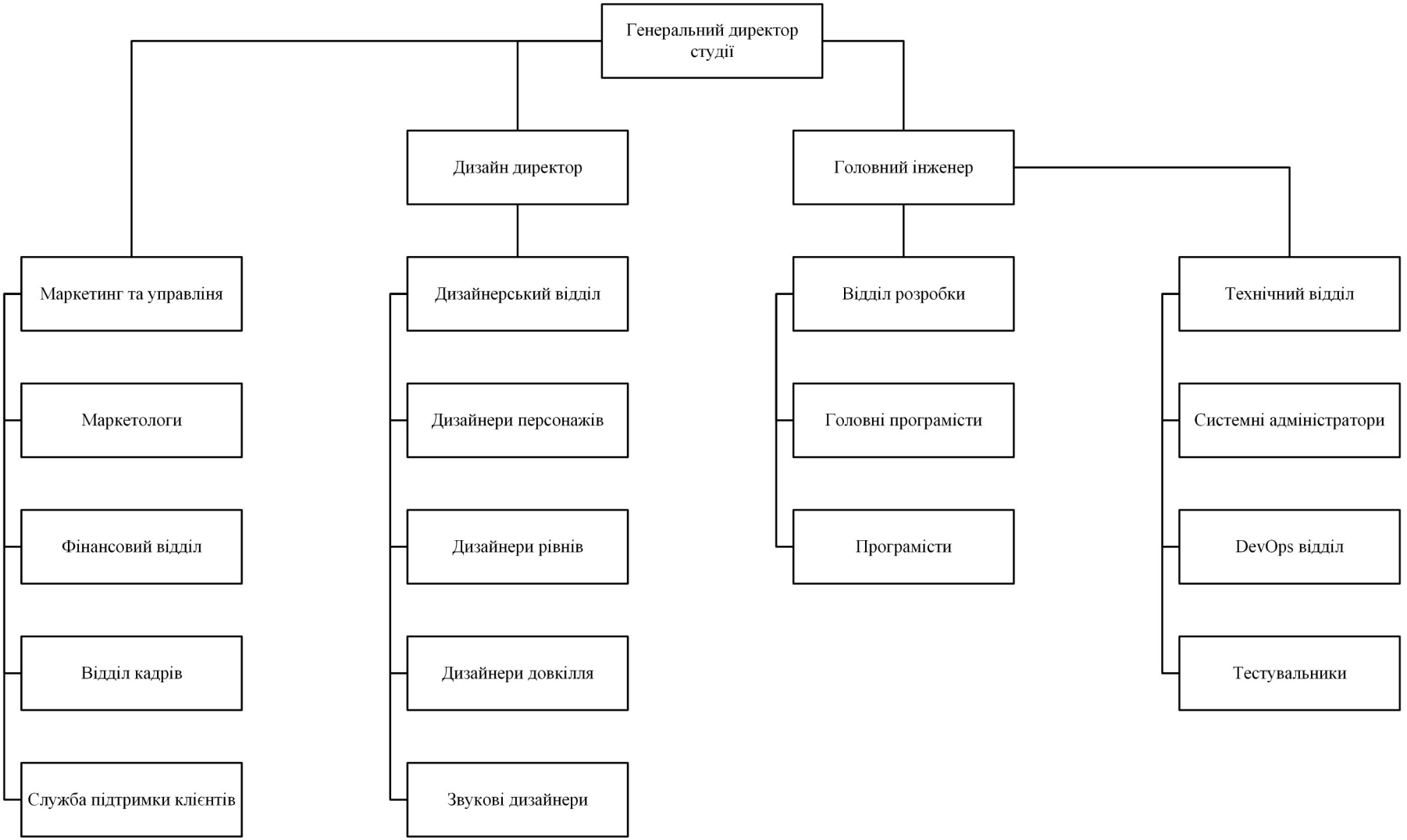

Рисунок 1.8 – Організаційна структура підприємства

Кожен підрозділ підприємства розташований у власному виділеному приміщенні. На плані основного триповерхового офісу компанії можна побачити розподіл відділів за поверхами та кімнатами. На рисунку 1.9 зображено план 1-го поверху, де знаходиться відділ маркетингу та управління, що включає генерального директора, а також відділ технічного забезпечення поруч із яким розташовано серверну кімнату.

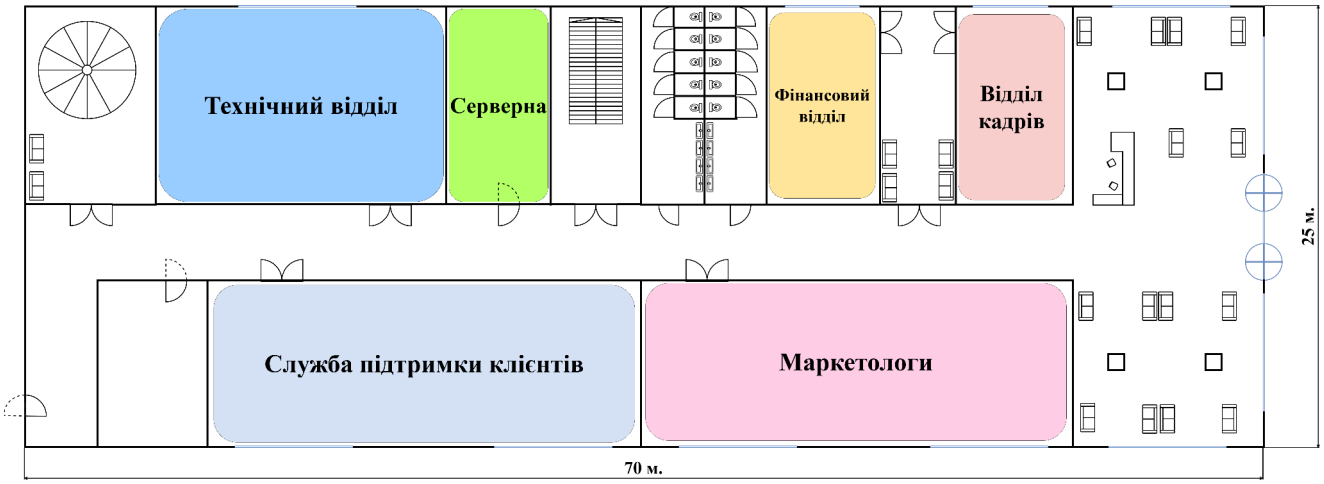

Рисунок 1.9 – План 1 поверху основного офісу

На другому поверсі розташовано відділ розробки у повному складі (рисунок 1.10).

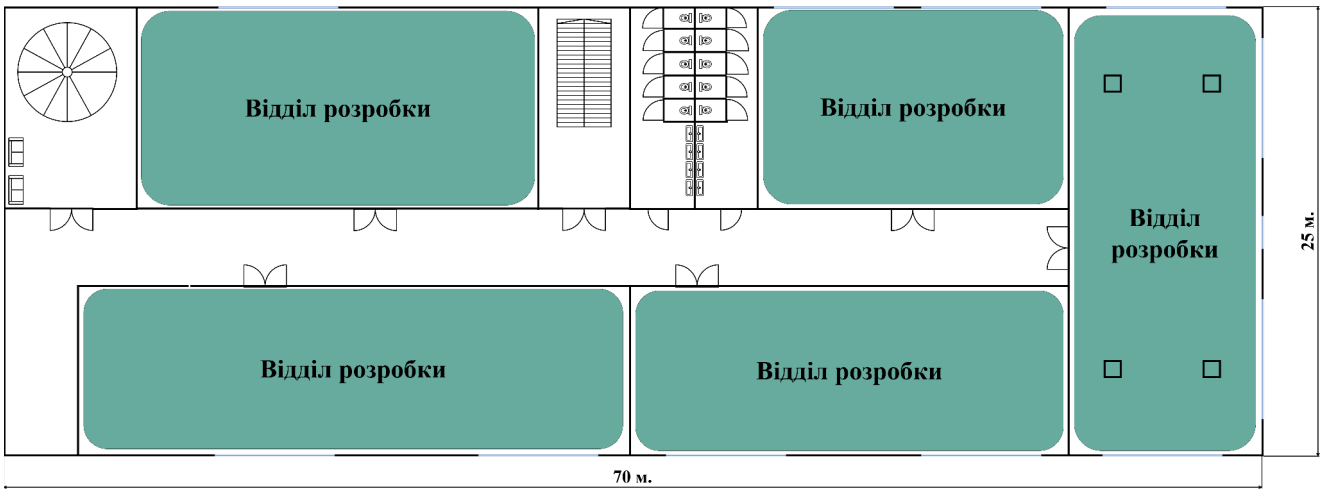

Рисунок 1.10 – План 2 поверху основного офісу

На третьому поверсі розташовується відділ дизайну, де кожний спеціалізований відділ має власне приміщення для роботи (рисунок 1.11).

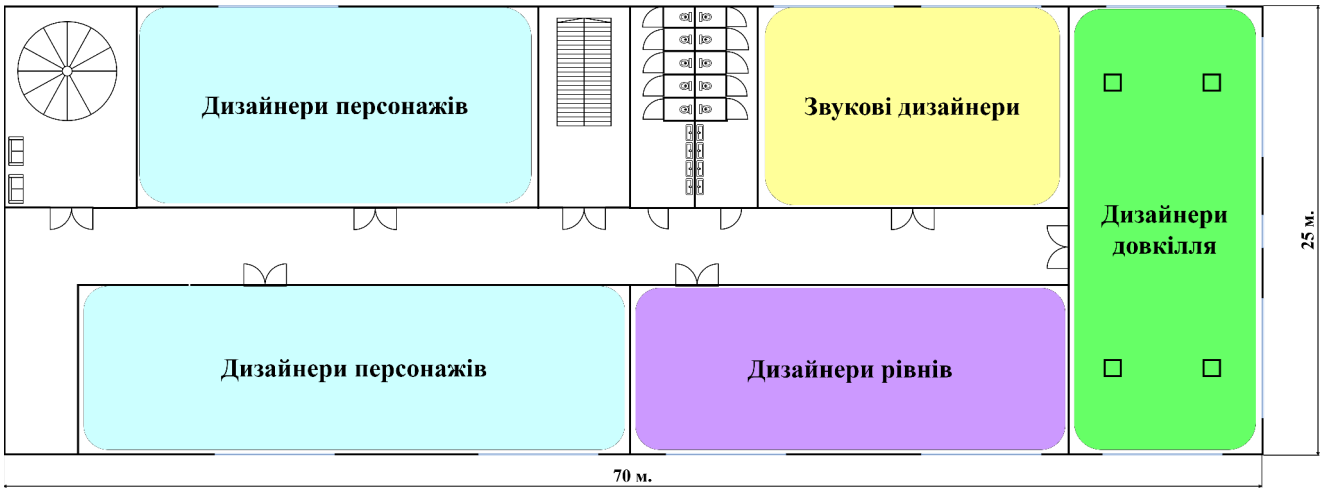

Рисунок 1.11 – План 3 поверху основного офісу

Віддалений офіс знаходиться на другому поверсі іншої офісної будівлі, в ньому розташовуються працівники відділу розробки (рисунок 1.12).

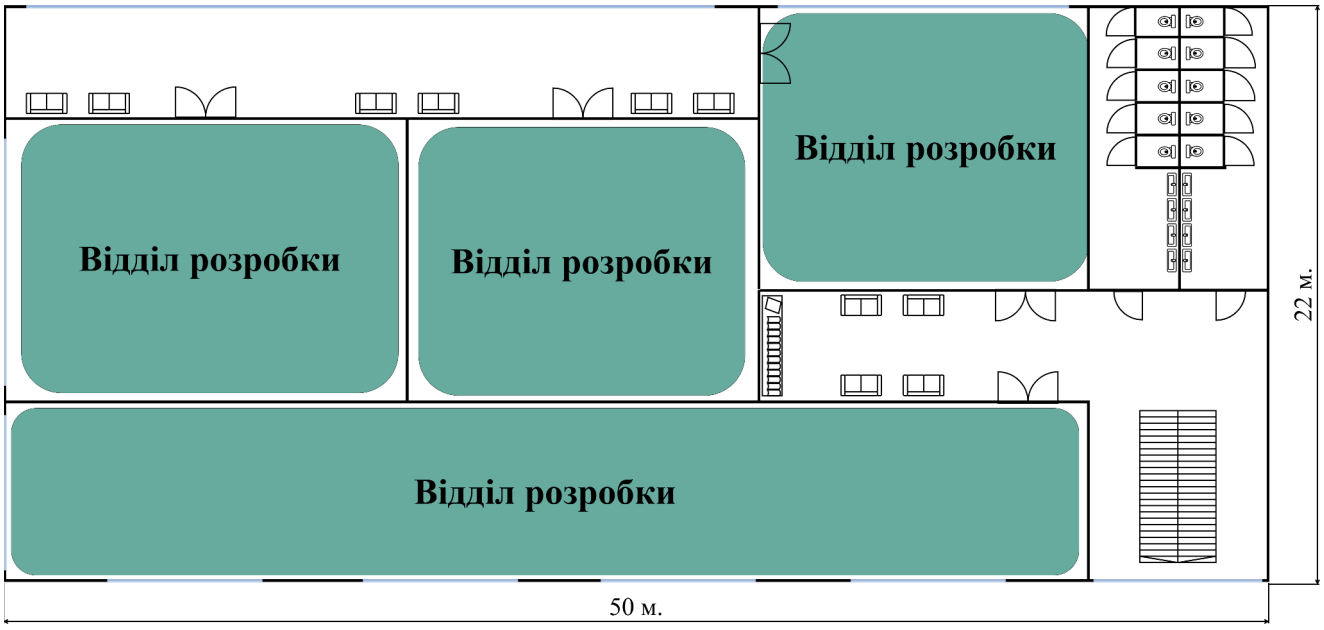

Рисунок 1.12 – План поверху віддаленого офісу

1.5 Технології керування, інформаційне забезпечення, визначення функцій системи

Побудова ефективних та захищених інформаційно-мережевих систем є комплексною задачею та потребує використання багатьох протоколів та технологій, кожна з яких відповідає за виконання певних функцій системи. Враховуючи характеристику галузі та структуру компанії, буде реалізовано наступний перелік функцій: забезпечення стабільного та швидкого зв’язку між підрозділами, забезпечення безпеки діяльності та даних компанії, налаштування власної серверної кімнати з реалізацією мережевих функцій та функції зберігання даних.

Інформаційне забезпечення системи повинне покривати широкий спектр задач, адже структура компанії є комплексною, через це кожен відділ потребує власне програмне забезпечення для виконання своїх функцій. Відділи постійно обмінюються інформацією та документами один з одним, для цього необхідно забезпечити працівників пакетом офісних застосунків, наприклад Office 365 або Libre Office. Обмін файлами та повідомленнями може проходити у корпоративному месенджері, Office 365 має для цього Microsoft Teams, проте є і інші рішення такі як Slack або Element. Відділ розробки та технічний відділ потребують доступу до системи контролю версій, для підтримки стабільного та однорідності коду розроблених продуктів компанії. Для цього можна можна розгорнути спільний репозиторій на основі технології Git в якому будуть зберігатися усі актуальні дані про проєкти а також історія внесених змін.

Керування, моніторинг та підтримка системи повинні проводитися на постійній основі співробітниками відділу технічного забезпечення. Моніторинг мережі може здійснюватись із застосуванням протоколу SNMP, а також протоколу NetFlow від компанії Cisco. Для автоматизації оновлень конфігурацій ПК можна використовувати Windows Active Directory для систем з ОС Windows, у випадку якщо у компанії застосовуються хости на основі ОС Linux або MacOS, можна використати будь-яку іншу службу каталогів на основі протоколу LDAP.

1.6 Огляд існуючих способів обробки та передачі інформації, принципів побудови, відомих рішень у галузі

Глобальна мережа на сьогоднішній день побудована на основі мережевого стеку протоколів TCP/IP. Даний стек включає велику кількість протоколів, що поділені на 4 рівні, кожен з яких виконує власні функції.

Канальний рівень передачі даних найчастіше є представленим протоколом Ethernet, який поділений на різні стандарти у відповідності до швидкості та надійності передачі даних, на локальному рівні найпопулярнішими серед них є: 100BASE-TX та 1000BASE-T. Така комбінація з кабелів у 100 та 1000 Мбіт/с дозволяє гнучко розподілити потреби Інтернет трафіку серед хостів та при цьому забезпечити безперервний доступ. В якості фізичних провідників для цих стандартів використовуються кабелі витої пари Cat5 для 100 Мбіт/с та Cat5E для 1000 Мбіт/с.

На мережевому рівні для адресації користувачів досі домінує протокол IPv4, незважаючи на свою моральну застарілість. Обладнання сумісне з протоколом IPv6 поступова входить до обігу, проте його покриття все ще не є достатнім для повного переходу глобальної мережі.

Транспортний рівень забезпечує передачу даних з використанням протоколів TCP та UDP. TCP використовується при необхідності надійного зв’язку та гарантованого доставлення повідомлень, UDP є простішим протоколом і відповідає виключно за надсилання даних, залишаючи відповідальність за їх повторну передачу та обробку помилок на протоколи вищого рівня.

Прикладний рівень пропонує найрізноманітнішу підбірку протоколів для вирішення широкого спектру задач. Найважливішими в контексті даної роботи є протоколи: DHCP, HTTPS, DNS, FTP, TFTP. HTTPS та DNS працюють у парі для забезпечення адресації сторінок у мережі Інтернет, DHCP виконує функцію автоматичного надання IP адрес хостам у мережі, FTP та TFTP здійснюють передачу файлів через мережу.

1.7 Аналітичний огляд існуючої топології та можливості її розвитку та покращення

Логічна топологія мережі згідно з поставленим завданням на кваліфікаційну роботу має наступний вигляд:

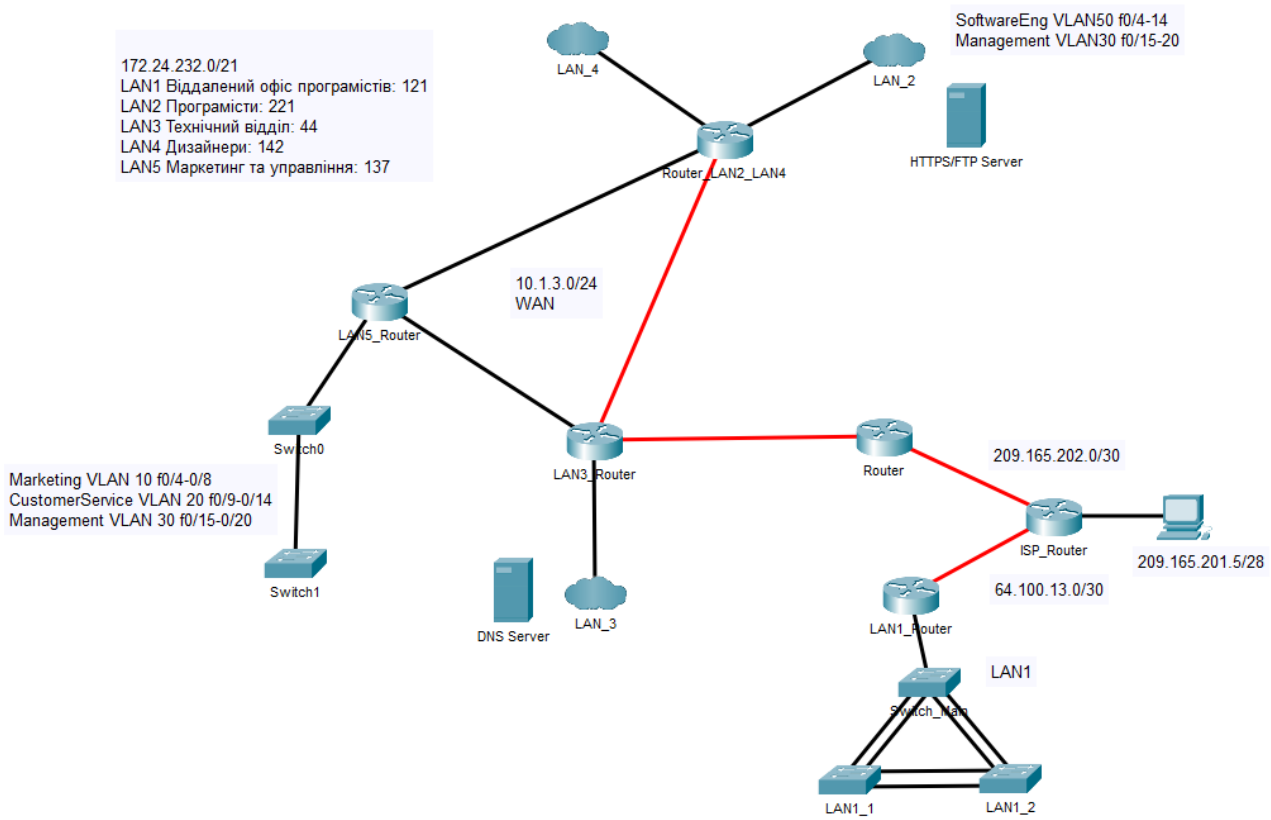

Рисунок 1.13 – Логічна структура корпоративної мережі

З даної схеми можна побачити, що у основному офісі знаходиться 4 підмережі, які розділені між відділами компанії. У локальній мережі №5 та локальній мережі №2 необхідно розгорнути віртуальні локальні мережі для більш ефективного розподілу трафіку та підвищення безпеки інформації, особливо у відділі маркетингу та управління. Для розташованих у мережі серверів потрібно забезпечити статичні адреси для глобальної доступності. У рамках розробки комп'ютерної системи необхідно буде розширити та протестувати покращену топології мережі, розробити технічні вимоги до системи а також специфікацію апаратних пристроїв.

1.8 Обґрунтування вибраного напрямку вирішення задачі

Процес проектування будь-якого програмного продукту завжди є пов’язаним з технічними та творчими викликами, це особливо стосується відеоігрової розробки. Існує багато визначень, які відрізняють відеоігри від інших продуктів, якщо об’єднати деякі з них, то можна визначити 10 ознак, які є важливими для відеогри [10]:

- в ігри грають зі своєї волі;
- у іграх є цілі;
- у іграх є конфлікт;
- у іграх існують правила;
- у іграх можна вигравати та програвати;
- ігри є інтерактивними;
- у іграх існують виклики;
- ігри можуть створювати власну внутрішню цінність;
- ігри повинні бути захопливі;
- ігри є закритими формальними системами.

Відсутність деяких з цих ознак є нормальним для гри, проте слідування ним підвищує шанси зробити цікавий та потрібний продукт, та підвищує розуміння, що вирізняє відеоігри від іншого ПЗ та цифрових розваг. Існує одна ознака яка об’єднує усі ігри – це вирішення проблем. Коли аспект вирішення проблем вилучається з гри, вона перестає бути грою і стає просто звичайним заняттям, як наприклад розмови з друзями або фізична активність.

При створенні гри, першочерговим питанням є, чому гравці грають у ігри? Чому вони виділяють час на те щоб запустити гру на своєму комп’ютері або телефоні замість перегляду кіно, відеоролику або походу до музею. Щоб зробити успішну гру необхідно розуміти відмінності цих форм проведення вільного часу та використовувати їх для створення найкращого ігрового досвіду. Багато гравців люблять грати в ігри, тому що вони надають виклики. Коли людина стикається з труднощами, а потім долає їх, вона чогось навчається. Неважливо, чи це завдання в підручнику з математики, чи в комп'ютерній грі. Складні ігри можуть бути

навчальним досвідом. Також гравці бажають отримувати власний унікальний досвід. Але ігри відрізняються від інших занять, таких як читання книги чи перегляд відео, оскільки вони надають гравцям щось для взаємодії, досвід реагування продукту на їхні дії. Як і в інших формах розваг, гравці можуть шукати певної емоційної винагороди, коли грають у відеогру. Емоції, які ігри здатні викликати у гравців, набагато сильніші, ніж ті, що можна відчути в інших медіа, де досвід менш захопливий і є має значно менше особистого залучення [11]. Існують і інші причини та стимули грати у відеоігри, проте ці є одними з найважливіших та впливових.

Отже, при розробці проєкту 2D-Roguelite гри важливо пам’ятати не тільки про основні особливості жанру, а і про ознаки відеоігор загалом. Даний ігровий жанр також ставить унікальні виклики для залучення гравця, необхідно затягнути гравця у ігровий цикл, що постійно змінюється завдяки процедурній генерації. Саме через це модульна структура побудови проєкту та його систем, є важливою, адже додавання нового контенту до гри та його оновлення є важливими елементами збереження уваги та зацікавленості гравця. Важливо зазначити, що 2D простір розробки було обрано не тільки через відносно простоту у порівнянні з повноцінною 3D-площиною, а і через актуальність для як для обраного жанру так і для індустрії в цілому, що можна побачити на основі приведених раніше прикладів.

В якості ігрового рушія для проєкту було обрано Unity 6.0, що було зумовлено деякими прагматичним факторами. По-перше, Unity пропонує широку підтримку та екосистему для створення 2D-проєктів, що дозволяє зробити процес розробки більш зрозумілим, з можливістю розробки для декількох платформ одночасно. По-друге, Unity має велику кількість різноманітних ресурсів для розробки ігор доступних за одним кліком на їх платформі, що дозволяє розробнику зосередитися на унікальних особливостях власного проєкту та використовувати ресурси та інструменти розроблені іншими авторами. По-третє, в даному контексті питання ліцензування готового продукту не є критичним, адже проєкт виконується в академічних цілях та буде представляти із себе прототип

який не буде отримувати прибуток.

В якості візуального стилю було обрано піксельну графіку. Незважаючи на свою давню історію та можливе відчуття застарілості, даний стиль все ще є популярним серед малих розробників та користується попитом серед користувачів. Даний вибір обумовлений не тільки попитом користувачів, а і тим фактом, що для розробника, створення дизайнів є легшою задачею ніж при використанні інших візуальних стилів. Практично будь-яка людина має можливість створювати базові дизайни без років навчання та підготовки. Іншою перевагою є зменшення загальної ваги ігрового проєкту, адже дизайни є неймовірно малими за розмірами, що також впливає і на продуктивність кінцевої гри [12].

Отже, в якості рішення з розробки прототипу 2D-Roguelite, було обрано рушій Unity з дотриманням модульної архітектури при написанні коду. В якості візуального стилю обрано піксельну стилістику, яка полегшить виготовлення дизайну гри та полегшить оптимізацію ігрового процесу та продуктивність. Ігровий процес буде побудований на комбінації принципів застосованих у “Slay The Spire” та “Enter The Gungeon”. На початку ігрового циклу процедурно генерується мапа, на якій будуть розташовуватися вершини різних типів. При виборі вершини з запланованою сутичкою із супротивником, буде процедурно згенеровано рівень, наповнений заздалегідь створеними кімнатами з ворогами. Таким чином гравець буде пересуватися вершинами до фінальної точки де буде розташовано босса рівня. Структура гри повинна відповідати Use-case діаграмі наведеній на рисунку 1.14

В якості платформи для розробки буде обрано збірку для систем Windows, а також WebGL для можливості хмарного розташування та можливості грати прямо з вікна браузера. Для хмарного розташування проєкту буде використано сторінку на ресурсі itch.io.

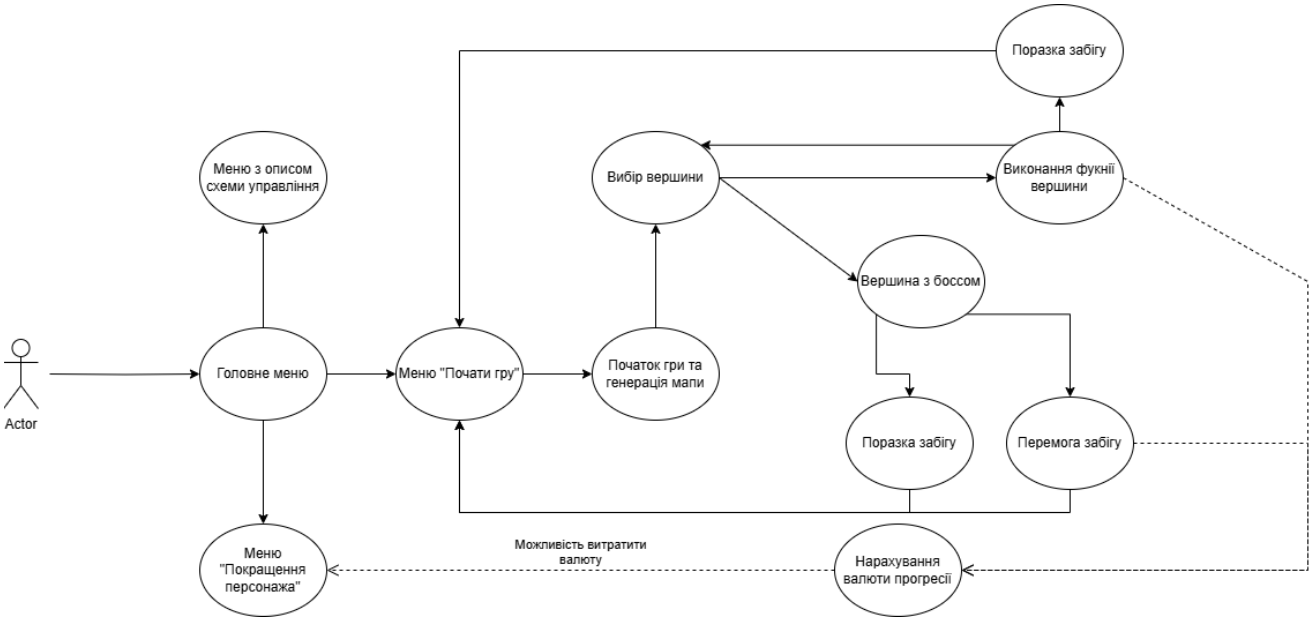

Рисунок 1.14 – Use-case діаграма ігрового процесу

Враховуючи поставлені задачі, реалізація комп’ютерної системи повинна базуватися на найбільш ефективних, надійних та популярних рішеннях у галузі. З’єднання локальних офісів через глобальну мережу буде проводитися за допомогою налаштованого VPN-тунелю за допомогою протоколу IPsec. Оптимальним рішенням у довгій перспективі буде розміщення основної серверної інфраструктури та систем зберігання даних локально у виділеній серверній. Менш важливі завдання, які мають схильність швидко масштабуватися у об’ємі необхідних для виконання ресурсів, будуть покладатися на хмарні сервіси. Архітектура мережі та її апаратне забезпечення повинні відповідати технічним вимогам поставленим до системи, а також задовільняти потреби потенційного розширення компанії. Локальні мережі в яких відбувається передача найважливіших даних повинні бути поділені на віртуальні локальні мережі для більшої безпеки та ефективності використання трафіку. Швидкість з’єднання до кожної з локальних мереж повинна відповідати стандарту Gigabit Ethernet зі швидкістю передачі 1000 Мбіт/с.

1.9 Мета і задачі роботи

Метою роботи є дослідження проблем та рішень, пов’язаних зі створенням відеоігрових проектів з використанням динамічного, процедурно згенерованого ігрового досвіду. Важливим аспектом є створення продукту, який буде залучати гравця та дозволить отримати унікальний та цікавий ігровий досвід. Мережева частина роботи переслідує мету визначення оптимальних рішень побудови надійних та ефективних корпоративних мереж для сучасних компаній з потребами у передачі та оперуванні великими обсягами даних.

Задачею є розробка прототипу 2D-гри у жанрі Roguelite з використанням ігрового рушія Unity та реалізації алгоритмів процедурної генерації ігрового досвіду. До розробки необхідно залучити сучасні практики написання ефективного та читабельного програмного коду. Стилістичне оформлення буде виконано у піксельному стилі. Для підтримки програмного продукту необхідно побудувати швидку та надійну комп’ютерну мережу з підтримкою хмарного розміщення даних, захищеною мережею з реалізацією VPN-тунелю між віддаленими офісами.

Налаштувати протокол динамічної маршрутизації між роутерами, а також налаштувати статичні адреси NAT для серверів компанії та PAT для інших хостів, заради забезпечення безпечного зв’язку із зовнішніми мережами.

У визначених підрозділах необхідно налаштувати VLAN мережі, та розділити їх з врахуванням структурних підрозділів.

Розробити архітектурне рішення корпоративної мережі та підібрати відповідне мережеве обладнання для побудови мережі та на його основі.

2 РОЗРОБКА АПАРАТНОЇ ЧАСТИНИ КОМП'ЮТЕРНОЇ СИСТЕМИ ПІДПРИЄМСТВА

2.1 Розробка технічних вимог до системи

Перед початком роботи над створення комп'ютерної системи, необхідно написати технічні вимоги згідно яких буде проводитись розробка. Виставлені технічні вимоги до системи, знаходяться у додатку А цієї роботи.

2.2 Розробка апаратної частини системи

2.2.1 Побудова схеми комплексу технічних засобів комп'ютерної системи

Враховуючи розроблені технічні вимоги, існуючу топологію мережі а також мету та завдання цієї кваліфікаційної роботи, було розроблено схему комплексу технічних засобів (КТЗ) комп'ютерної системи відеоігрової студії, вона зображена на рисунку 2.1.

Структурна схема розділена на рівні згідно до ієрархічної моделі мережі розробленої компанією Cisco. У моделі та на схемі визначено такі рівні:

– Рівень ядра – на цьому рівні знаходяться пристрої що формують ядро мережі та виконують швидке і надійне пересилання основного обсягу трафіку. На даному рівні знаходиться маршрутизатор та мережа провайдера (ISP).

– Рівень розподілу – на цьому рівні реалізуються маршрутизація для індивідуальних локальних мереж, на даному рівні знаходяться маршрутизатори основного та віддаленого офісу, що безпосередньо підключені до локальних мереж підприємства.

– Рівень доступу – рівень на якому відбувається комутація та розподіл пакетів для кінцевих користувачів. Мета даного рівня підключення вузлів до основної мережі, тому на цьому рівні знаходяться усі комутатори локальних мереж. Кінцеві вузли рівня доступу були виділені у окремий рівень хостів на схемі, в ньому знаходяться ПК та сервери компанії.

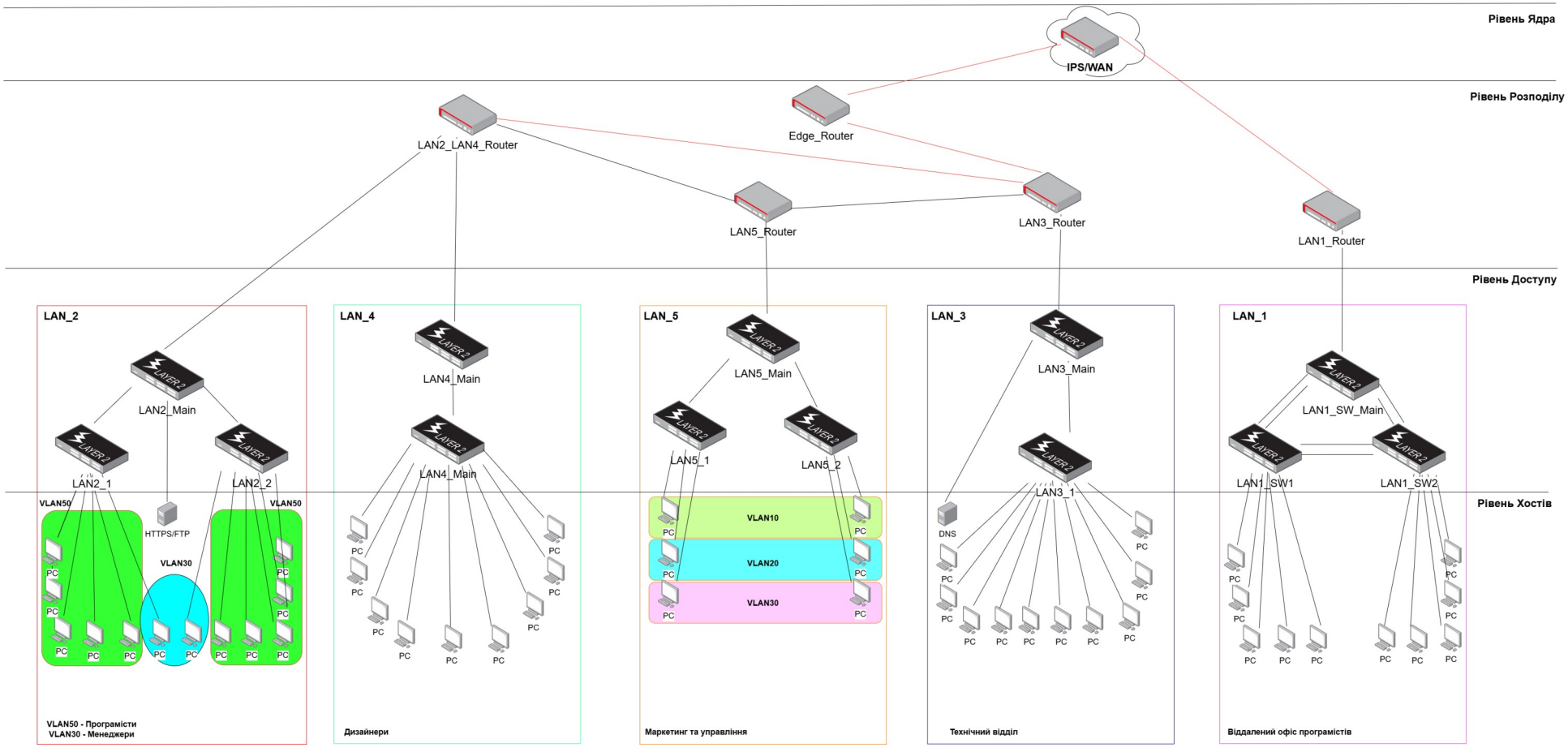

Рисунок 2.1 – Схема комплексу технічних заходів комп’ютерної системи

2.2.2 Розробка специфікації апаратних засобів комп’ютерної системи

Враховуючи технічні вимоги, розроблену схему (КТЗ) а також існуючу топологію мережі, необхідно підібрати відповідне обладнання для побудови мережі.

Розроблена схема КТЗ не враховує усі вузли, що знаходяться у комп’ютерній системі, проте основна мережева інфраструктура буде залишатися незмінною, для повної реалізації мережевої топології необхідно лише додати невраховані вузли та комутатори для їх підключення до головних комутаторів мереж які позначені назвою LAN#_Main.

Для маршрутизації у мережі будуть використовуватися роутери Cisco C2911-VSEC/K9. Дані роутери мають достатню кількість портів та вільних слотів для розширення, для реалізації існучої топології. Роутер має сучасні технології захисту, можливості налаштування IPsec VPN-тунелів, налаштування динамічної маршрутизації OSPF та NAT. У КТЗ та логічній топології він буде використовуватися для симуляції усіх роутерів.

Для кожної з мереж необхідно обрати головний комутатор, характеристики якого повинні бути наступними: 24 LAN порти GigabitEthernet з можливістю PoE для масштабування та підключення додаткових комутаторів та GigabitEthernet Uplink порти з можливістю підключення за допомогою витої пари. Під ці параметри підходить комутатор Cisco CBS220-24P-4G-EU. Він підтримує усі необхідні протоколи та технології, що повинні бути реалізовані у мережі. У КТЗ даний пристрій відповідає усім комутаторам зі словом Main у назві.

Комутатори що напряму підключаються до ПК працівників повинні мати 24 LAN порти FastEthernet та Uplink GigabitEthernet витої пари. Для цих цілей буде використано Cisco WS-C2960-24PST-L. На даному пристрої можлива реалізація VLAN та EtherChannel, та додаткові можливості PoE, що повністю задовільняє потреби комп’ютерної мережі.

Специфікація обладнання комп’ютерної системи наведена у таблиці 2.1 [13-14]. У стовпці кількість число позначає кількість використаних пристроїв у логічній топології, число в дужках позначає необхідну кількість згідно завдання.

Таблиця 2.1 – Загальна специфікація обладнання

Позиція	Найменування	Тип, марка, позначення	Одиниці виміру	Кількість	Примітки / Х-ки
1	**2**	**3**	**4**	**5**	**6**
Мережеве обладнання					
1.	Маршрутизатор Cisco C2911	C2911-VSEC/K9	од.	6	Має два слоти розширення HWIC-1GE-SFP Детальні х-ки: https://stack-systems.com.ua/marshrutizator-cisco-2911-k9
2.	Комутатор Cisco CBS220	CBS220-24P-4G-EU	од.	5	Детальні х-ки: https://comtrade.ua/ua/cisco-cbs220-24p-4g-eu/
3.	Комутатор Cisco WS-C2960	C2960-24PST-L	од.	8 (34)	Детальні х-ки: https://comtrade.ua/ua/cisco-ws-c2960-24pst-l/
Робочі станції					
4.	Настільний комп’ютер стандартної офісної конфігурації	Cobra Optimal I11.8.S2.INT.429	од.	19 (181)	Використовуються у LAN3 та LAN4. Процесор: Intel Core i3-10100 (3.6 - 4.3 ГГц). Оперативна пам’ять: 8ГБ. Пам’ять: 256 ГБ SSD.

Продовження таблиці 2.1

5.	Настільний комп’ютер підвищеної потужності	COBRA Advanced I14F.16.S4. 165.12991	од.	33 (484)	Використовуються у LAN1, LAN2 та LAN5 Процесор: Intel Core i5-10400F (2.9 - 4.3 Ггц). Графічний процесор: 4ГБ. Оперативна пам’ять: 16 ГБ. Пам’ять: 480 ГБ SSD.
Серверне обладнання					
6.	Сервер ARTLINE	ARTLINE Business T17 v18	од.	2	Постачається у власному кейсі, не потребує встановлення на стійку. HTTPS/FTP та DNS сервери. Процесор:Intel Core i5-11400 (2.6 – 4.4 Ггц) Оперативна пам’ять: 32 ГБ Пам’ять: 2 шт. 1ТБ HDD, 1 шт. 250 ГБ SSD
Обладнання для монтажу мережевих пристроїв					
7.	Шафа серверна настінна Gear 12U	GWMSN-12U-600-450	шт.	2 (6)	19 дюймова шафа для встановлення маршрутизаторів та комутаторів.

3 РОЗРОБКА КОРПОРАТИВНОЇ МЕРЕЖІ

3.1 Розрахунок налаштувань для заданої топології мережі

3.1.1 Розрахунок схеми адресації корпоративної мережі

Згідно до п. А.1.2.1 додатку А внутрішній мережі відеоігрової студії було виділено блок адрес 172.24.232.0/21, а також блок 10.3.1.0/24 для її маршрутизаторів. Адресний простір було розподілено за допомогою технологій CIDR та VLSM у відповідності до зазначеної кількості вузлів у таблиці 3.1, а також з врахуванням можливостей масштабування кожного з підрозділів.

Таблиця 3.1 – Кількість вузлів у кожній з підмереж

Назва мережі	LAN1	LAN2	LAN3	LAN4	LAN5
Кількість вузлів згідно завдання	121	221	44	142	137

Після поділу блоку локальних адрес, а також виділеного блоку маршрутизації та з урахуванням наданих адрес для зв’язку з ISP, маємо схему адресації, що наведена у таблиці 3.2.

Таблиця 3.2 – Схема адресації комп’ютерної системи

Підмережа	Розмір	Адреса	Маска	Діапазон адрес
1	2	3	4	5
LAN1	121	172.24.232.0	255.255.254.0	172.24.232.1 – 172.24.233.254
LAN2	221	172.24.234.0	255.255.254.0	172.24.234.1 – 172.24.235.254
LAN3	44	172.24.239.0	255.255.255.128	172.24.239.1 – 172.24.239.126
LAN4	142	172.24.236.0	255.255.254.0	172.24.236.1 – 172.24.237.254
LAN5	137	172.24.238	255.255.255.0	172.24.238.1 – 172.24.238.254
WAN1	2	10.1.3.0	255.255.255.252	10.1.3.1 - 10.1.3.2
WAN2	2	10.1.3.4	255.255.255.252	10.1.3.5 - 10.1.3.6

Продовження таблиці 3.2

1	2	3	4	5
WAN3	2	10.1.3.8	255.255.255.252	10.1.3.9 - 10.1.3.10
WAN4	2	10.1.3.12	255.255.255.252	10.1.3.13 – 10.1.3.14
WAN IPS	2	209.165.202.0	255.255.255.252	209.165.202.1 - 209.165.202.2
LAN IPS	2	209.165.201.0	255.255.255.240	209.165.201.1 - 209.165.201.14
WAN Remote	2	64.100.13.0	255.255.255.252	64.100.13.1 - 64.100.13.2
LAN2 VLAN30	254	172.24.235.0	255.255.255.0	172.24.235.1 – 172.24.235.254
LAN2 VLAN50	254	172.24.234.0	255.255.255.0	172.24.234.1 – 172.24.234.254
LAN2 VLAN99	30	172.24.236.0	255.255.255.224	172.24.236.1 – 172.24.236.30
LAN5 VLAN10	62	172.24.238.0	255.255.255.192	172.24.238.1 - 172.24.238.62
LAN5 VLAN20	62	172.24.238.64	255.255.255.192	172.24.238.65 - 172.24.238.126
LAN5 VLAN30	62	172.24.238.128	255.255.255.192	172.24.238.129 - 172.24.238.190
LAN5 VLAN 99	30	172.24.238.192	255.255.255.192	172.24.238.193 - 172.24.238.206

Після розподілу адрес на підмережі, необхідно створити таблицю адресації пристроїв мережі (табл. 3.3). У мережах із застосуванням технології VLAN на маршрутизаторах налаштовано віртуальні інтерфейси у відповідності до номеру VLAN, для якого вони будуть динамічно надавати адреси за протоколом DHCP. У мережах без VLAN DHCP налаштовано на фізично підключених до мережі інтерфейсах. Інтерфейси маршрутизаторів повинні мати перші адреси у кожній мережі, комутаторам надається друга адреса, за винятком мереж із VLAN, там мережевим пристроям надається окремий блок адрес. Серверам надаються статичні адреси з легкими для запам’ятовування адресами у кінці блоку, нариклад 50, 100, 200, інші вузли мережі отримують адреси за залишковим принципом.

Таблиця 3.3 – Схема адресації пристроїв мережі

Пристрій	Інтер-фейс	IP-адреса	Маска	Шлюз	VLAN	Інтерфейс підключеного пристрою
1	**2**	**3**	**4**	**5**	**6**	**7**
LAN1 Віддалений офіс програмістів						
Afanasiev_LAN1_ Router	G0/0	172.24.232.1	/23	-	-	G0/0
	G0/2/0	64.100.13.2	/30	-	-	G0/2/0
LAN1_Switch_Main	Vlan1	172.24.232.2	/23	172.24.232.1	-	G0/1
LAN1_Switch_1	Vlan1	172.24.232.3	/23	172.24.232.1	-	G0/1
LAN1_Switch_2	Vlan1	172.24.232.4	/23	172.24.232.1	-	G0/1
LAN1_PC (10 шт.)	NIC	172.24.232.5 - 172.24.232.15	/23	172.24.232.1	-	Fa0/1- 10
LAN2 Головний офіс відділу розробки						
Afanasiev_Router_ LAN2_LAN4	G1/0. 30	172.24.235.1	/25	-	30	G0/1
	G1/0. 50	172.24.234.1	/24	-	50	G0/1
	G1/0. 99	172.24.235.129	/27	-	99	G0/1
	G2/0	10.1.3.13	/30	-	-	G2/0
	G3/0	10.1.3.6	/30	-	-	G3/0
LAN2_Switch_Main	Vlan99	172.24.235.130	/27	172.24.235.129	99	G0/1
LAN2_Switch_1	Vlan99	172.24.235.131	/27	172.24.235.129	99	G0/1
LAN2_Switch_2	Vlan99	172.24.235.132	/27	172.24.235.129	99	G0/1
LAN2_PC_VLAN 50 (9 шт.)	NIC	172.24.234.2 - 172.24.234.10	/24	172.24.234.1	50	Fa0/4-14
LAN2_PC_VLAN 30 (2 шт.)	NIC	172.24.235.2 - 172.24.235.3	/25	172.24.235.1	30	Fa0/15-16
TFTP / HTTP Server	NIC	172.24.235.150	/27	172.24.235.129	99	Fa0/23
LAN3 Технічний відділ						
Afanasiev_LAN3_ Router	G0/0	172.24.239.1	/25	-	-	G0/1
	G1/0	10.1.3.9	/30	-	-	G1/0
	G3/0	10.1.3.5	/30	-	-	G3/0
	G4/0	10.1.3.2	/30	-	-	G0/3/0
LAN3_Switch_Main	Vlan1	172.24.239.2	/25	172.24.239.1	-	G0/1
LAN3_Switch_1	Vlan1	172.24.239.3	/25	172.24.239.1	-	G0/1

Продовження таблиці 3.3

1	2	3	4	5	6	7
LAN3_PC (10 шт.)	NIC	172.24.239.4 - 172.24.239.14	/25	172.24.239.1	-	Fa0/1-10
DNS Server	NIC	172.24.239.100	/25	172.24.239.1	-	Fa0/24
LAN4 Дизайнерський відділ						
Afanasiev_Router_ LAN2_LAN4	G0/0	172.24.236.1	/23	-	-	G0/1
LAN4_Switch_Main	Vlan1	172.24.236.2	/23	172.24.236.1	-	G0/1
LAN4_Switch_1	Vlan1	172.24.236.3	/23	172.24.236.1	-	G0/1
LAN4_PC (9 шт.)	NIC	172.24.236.4 - 172.24.236.13	23	172.24.236.1	-	Fa0/1-9
LAN5 Маркетинг та управління						
Afanasiev_LAN5_ Router	G0/0. 10	172.24.238.1	/26	-	10	G0/0
	G0/0. 20	172.24.238.65	/26	-	20	G0/0
	G0/0. 30	172.24.238.129	/26	-	30	G0/0
	G0/0. 99	172.24.238.193	/28	-	99	G0/0
	G1/0	10.1.3.10	/30	-	-	G1/0
	G2/0	10.1.3.14	/30	-	-	G2/0
LAN5_Switch_Main	Vlan99	172.24.238.194	/28	172.24.238.193	99	G0/1
LAN5_Switch_1	Vlan99	172.24.238.195	/28	172.24.238.193	99	G0/1
LAN5_Switch_2	Vlan99	172.24.238.196	/28	172.24.238.193	99	G0/1
LAN5_PC_VLAN10 (4 шт.)	NIC	172.24.238.2 - 172.24.238.6	/26	172.24.238.1	10	Fa0/4-8
LAN5_PC_VLAN20 (4 шт.)	NIC	172.24.238.66 - 172.24.238.80	/26	172.24.238.65	20	Fa0/9-13
LAN5_PC_VLAN30 (4 шт.)	NIC	172.24.238.130- 172.24.238.134	/26	172.24.238.65	30	Fa0/15-20
ISP та WAN						
ISP_Router	G0/0	209.165.201.1	/28	-	-	Fa0/1
	G2/0	209.165.202.1	/30	-	-	G0/2/0
	G3/0	64.100.13.1	/30	-	-	G0/2/0
PC0	NIC	209.165.201.2	/28	-	-	G0/0
Afanasiev_Edge_ Router	G0/2/0	209.165.202.2	/30	-	-	G2/0
	G0/3/0	10.3.1.1	/30	-	-	G4/0

3.1.2 Розробка топологічної схеми корпоративної мережі

Комп’ютерна мережа у відповідності до технічних вимог та наданого завдання, є поділеною на п’ять локальних мереж. Топологію мережі було побудовано з використанням топології “розширена зірка”. Маршрутизація у мережі компанії здійснюється з використанням 5 роутерів. Роутери мереж LAN1, LAN3 та LAN5 виконують функції маршрутизації для своїх відповідних мереж. Один із роутерів підключений одночасно до двох підсистем, LAN2 та LAN4. Крайній роутер мережі основного офісу повинен реалізовувати функції NAT та мати налаштований IPsec VPN-тунель до LAN1. Маршрутизатори підключені один до одного за допомогою стандарту GigabitEthernet з комбінованим використанням витої пари та оптичних кабелів. Локальні мережі підключені до маршрутизаторів за допомогою головних комутаторів та стандарту GigabitEthernet.

Комутатори поділяються на головні та зв’язуючі, головні виконуються функцію підключення до маршрутизатора та розподілу трафіку серед інших комутаторів, зв’язуючі комутатори напряму здійснюють підключенні вузлів до мережі.

Вузли мереж підключаються до комутаторів за допомогою витої пари стандарту FastEthernet, при цьому сервери мереж підключаються напряму до головних комутаторів мережі, при наявності VLAN для віддаленого управління комутаторами, статична IP-адреса серверу повинна знаходитися у блоці адреси виділеному для віддаленого управління.

Завершена схема логічної топології мережі зображена на рисунку 3.1.

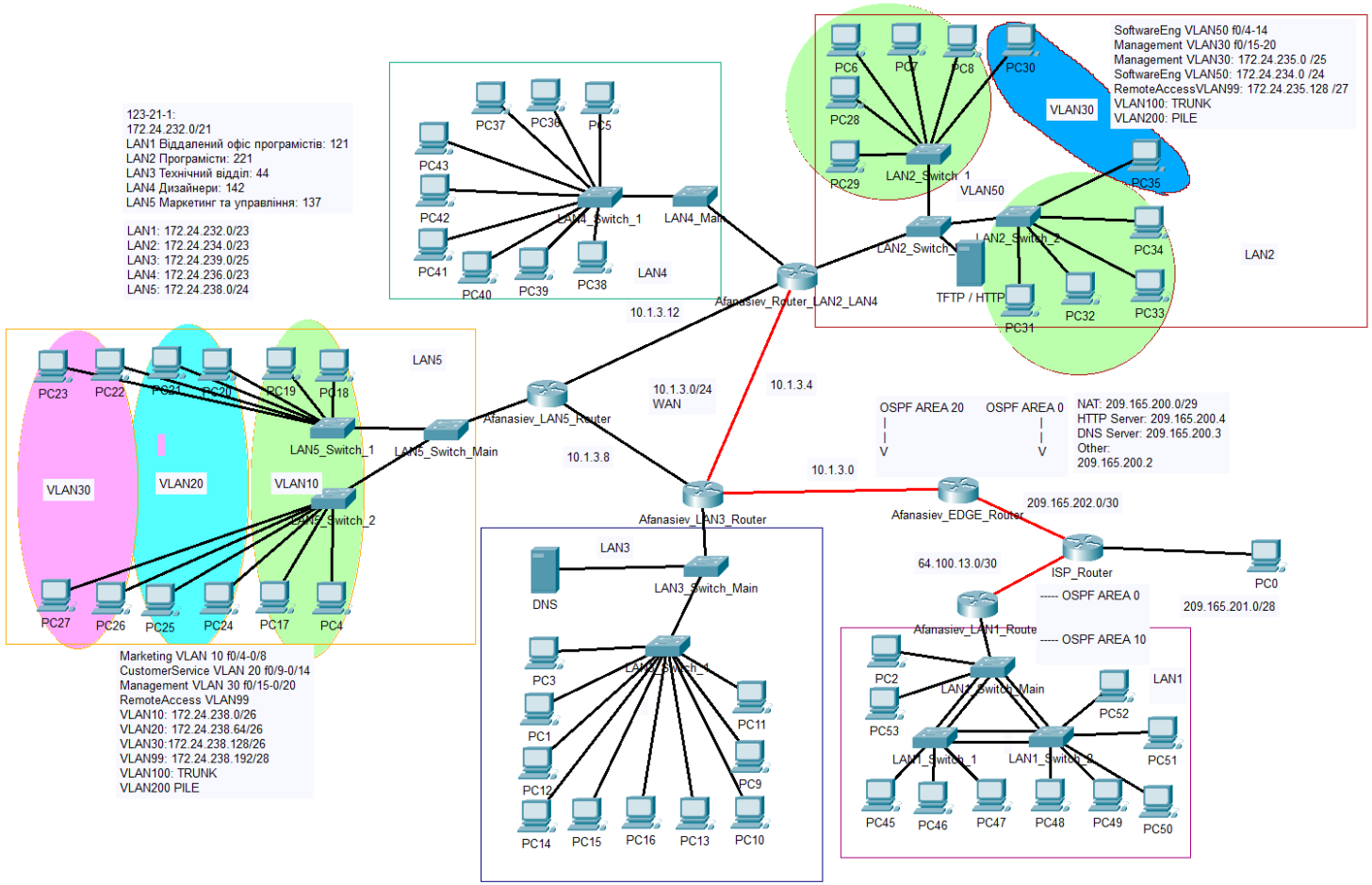

Рисунок 3.1 – Схема логічної топології мережі відеоігрової студії

3.2 Перевірка роботи комп’ютерної системи підприємства

3.2.1 Базове налаштування мережевих пристроїв

У набір базових налаштувань мережевих пристроїв комп’ютерної системи входять такі налаштування як:

– встановлення паролів для привілейованого режиму, консольного порту та ліній віддаленого доступу vty із застосуванням шифрування;

– зміна назви пристрою для кращого документування;

– налаштування IP-адрес на мережевих інтерфейсах;

Приклад виконаних команд для налаштування цих функцій на роутері ISP:

Router(config)#hostname ISP

ISP(config)#line console 0

ISP(config-line)#password afanasiev

ISP(config-line)#login

ISP(config-line)#exit

ISP(config)#enable secret afanasiev

ISP(config)#line vty 0 15

ISP(config-line)#password afanasiev

ISP(config-line)#login

ISP(config)# service password-encryption

ISP(config)#int gig2/0

ISP(config)#ip address 209.165.200.1 255.255.255.252

ISP(config)#int gig3/0

ISP(config)#ip address 64.100.13.1 255.255.255.252

ISP(config)#int gig0/0

ISP(config)#ip address 209.165.201.1 255.255.255.240

3.2.2 Налаштування маршрутизаторів корпоративної мережі

На маршрутизаторах комп’ютерної мережі необхідно налаштувати протоколи маршрутизації, для цього було використано комбінацію динамічного протоколу OSPF та статичних шляхів. При налаштуванні процесу OSPF на роутерах,

важливо щоб процеси на усіх пристроях мали однаковий ID процесу, також важливим є налаштування зон в яких знаходяться різні підсистеми мережі. Головний офіс компанії та його адреси повинні знаходитися у зоні №20, LAN1 віддаленого офісу має №10, інтерфейси та мережі що напряму підключені до ISP повинні мати 0 зону.

Приклади виконаних налаштувань на роутерах:

До мережі ISP додається статичний маршрути до мережі, яка буде використана при налаштуванні NAT на крайньому маршрутизаторі офісної мережі:

ISP(config)#ip route 209.165.200.0 255.255.255.248

Далі проводиться налаштування процесу OSPF та надання дозволу на поширення статичного маршрути з іншими маршрутизаторами:

ISP(config)#router ospf 1

ISP(config-router)#network 209.165.201.0 0.0.0.15 area 0

ISP(config-router)#network 209.165.202.0 0.0.0.3 area 0

ISP(config-router)#network 64.100.13.0 0.0.0.3 area 0

ISP(config)#redistribute static subnets

Приклад налаштування на внутрішніх роутерах основного офісу:

LAN3(config)#router ospf 1

LAN3(config-router)#network 172.24.239.0 0.0.0.127 area 20

LAN3(config-router)#network 10.1.3.0 0.0.0.3 area 20

LAN3(config-router)#network 10.1.3.4 0.0.0.3 area 20

LAN3(config-router)#network 10.1.3.8 0.0.0.3 area 20

Налаштування на крайньому роутері головного офісу:

Edge(config-router)#network 10.1.3.0 0.0.0.3 area 20

Edge(config-router)#network 209.165.202.0 0.0.0.3 area 0

Проведено додаткове налаштування маршруту за замовчення та дозвіл на його поширення за OSPF

Edge(config)#default-information originate

Edge(config)#ip route 0.0.0.0 0.0.0.0 g0/2/0

Після проведених налаштувань OSPF, таблицю маршрутизації на роутері ISP можна побачити на рисунку 3.2.

```
ISP#sh ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is 209.165.202.2 to network 0.0.0.0

     10.0.0.0/30 is subnetted, 4 subnets
O IA    10.1.3.0 [110/2] via 209.165.202.2, 02:37:54, GigabitEthernet2/0
O IA    10.1.3.4 [110/3] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    10.1.3.8 [110/3] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    10.1.3.12 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
     64.0.0.0/30 is subnetted, 1 subnets
C       64.100.13.0 is directly connected, GigabitEthernet3/0
     172.24.0.0/16 is variably subnetted, 10 subnets, 6 masks
O IA    172.24.232.0/23 [110/2] via 64.100.13.2, 02:37:54, GigabitEthernet3/0
O IA    172.24.234.0/24 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.235.0/25 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.235.128/27 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.236.0/23 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.238.0/26 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.238.64/26 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.238.128/26 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.238.192/28 [110/4] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
O IA    172.24.239.0/25 [110/3] via 209.165.202.2, 02:37:44, GigabitEthernet2/0
     209.165.200.0/29 is subnetted, 1 subnets
S       209.165.200.0 [1/0] via 209.165.202.2
     209.165.201.0/28 is subnetted, 1 subnets
C       209.165.201.0 is directly connected, GigabitEthernet0/0
     209.165.202.0/30 is subnetted, 1 subnets
C       209.165.202.0 is directly connected, GigabitEthernet2/0
O*E2 0.0.0.0/0 [110/1] via 209.165.202.2, 02:37:54, GigabitEthernet2/0
```

Рисунок 3.2 – Таблиця маршрутизації на ISP_Router

На роутерах, що напряму підключені до локальних підсистем налаштовано DHCP:

*LAN3(config-*if *)#ip dhcp pool LAN3*

*LAN3(config-*if *)#network 172.24.239.0 255.255.255.128*

LAN3(config-if)#default-router 172.24.239.1

LAN3(config-if)#dns-server 172.24.239.100

LAN3(config)#ip dhcp excluded-address 172.24.239.1

LAN3(config)#ip dhcp excluded-address 172.24.239.2

LAN3(config)#ip dhcp excluded-address 172.24.239.3

LAN3(config)#ip dhcp excluded-address 172.24.239.100

Приклад таблиці наданих за DHCP адрес зображено на рисунку 3.3.

```
LAN3(config)#do sh ip dhcp binding
IP address       Client-ID/              Lease expiration        Type
                 Hardware address
172.24.239.3     0090.2B40.9338          --                      Automatic
172.24.239.2     00D0.D322.5688          --                      Automatic
172.24.239.4     000B.BE4E.6056          --                      Automatic
172.24.239.8     0090.0C88.739C          --                      Automatic
172.24.239.7     00D0.979D.1038          --                      Automatic
172.24.239.6     0030.F2C2.79C2          --                      Automatic
172.24.239.10    0060.5CA5.6D51          --                      Automatic
172.24.239.9     00E0.F9E6.ADE7          --                      Automatic
172.24.239.5     00E0.F7E4.21D8          --                      Automatic
172.24.239.11    00D0.D39C.4920          --                      Automatic
```

Рисунок 3.3 – Таблиця DHCP на Afanasiev_LAN3_Router

3.2.3 Захист інформації в комп'ютерній системі

На комутаторах у локальних мережах LAN2 та LAN5 було налаштовано VLAN, у випадку LAN2 це три VLAN призначені для підключення користувачів, та 3 допоміжні VLAN, 99, 100 та 200. VLAN 99 виконує функцію об'єднання комутаторів мережі у одну віртуальну мережу для динамічного надання адрес. VLAN 100 є транковою мережею, за допомогою якої комутатори та маршрутизатор будуть обмінюватися даними про VLAN. VLAN 200 призначена для занесення неактивних портів для подальшого їх призначення до інших VLAN.

Приклад налаштування комутаторів для роботи з VLAN у мережі LAN2:

LAN5_Switch_Main(config)#vlan 10

LAN5_Switch_Main(config)#name Marketing

LAN5_Switch_Main(config)#vlan 20

LAN5_Switch_Main(config)#name CustomerService

LAN5_Switch_Main(config)#vlan 30

LAN5_Switch_Main(config)#name Management

LAN5_Switch_Main(config)#vlan 99

LAN5_Switch_Main(config)#name RemoteAccess

LAN5_Switch_Main(config)#vlan 100

LAN5_Switch_Main(config)#name TRUNK

LAN5_Switch_Main(config)#vlan 200

LAN5_Switch_Main(config)#name PILE

LAN5_Switch_Main(config)#int vlan 99

LAN5_Switch_Main(config)#ip address dhcp

LAN5_Switch_Main(config)#no shutdown

LAN5_Switch_Main(config)#int fa0/24

LAN5_Switch_Main(config)#switchport mode trunk

LAN5_Switch_Main(config)#switchport trunk allowed vlan all

LAN5_Switch_Main(config)#switchport trunk native vlan 100

На розподільчих комутаторах налаштовано порти доступу за наступним алгоритмом:

LAN5_Switch_1(config-if)#int range fa0/1-3

LAN5_Switch_1(config-if)#switchport mode access

LAN5_Switch_1(config-if)#switchport access vlan 200

LAN5_Switch_1(config-if)#int range fa0/4-14

LAN5_Switch_1(config-if)#switchport mode access

LAN5_Switch_1(config-if)#switchport access vlan 50

LAN5_Switch_1(config-if)#int range fa0/15-20

LAN5_Switch_1(config-if)#switchport mode access

LAN5_Switch_1(config-if)#switchport access vlan 30

LAN5_Switch_1(config-if)#int range fa0/21-23, gig0/1-2

LAN5_Switch_1(config-if)#switchport mode access

LAN5_Switch_1(config-if)#switchport access vlan 200

Для коректної роботи DHCP з віртуальними мережами, на маршрутизаторах необхідно провести налаштування віртуальних інтерфейсів з використанням протоколу dot1Q для надання такої можливості.

Приклад налаштування Afanasiev_LAN5_Router:

Afanasiev_LAN5_Router(config-if)#ip dhcp pool VLAN 10

Afanasiev_LAN5_Router(config-if)#network 172.24.238.0 255.255.255.192

Afanasiev_LAN5_Router(config-if)#default-router 172.24.238.1

Afanasiev_LAN5_Router(config-if)#dns-server 172.24.239.100

Afanasiev_LAN5_Router(config-if)#ip dhcp excluded-address 172.24.238.1

Afanasiev_LAN5_Router(config-if)#ip dhcp pool VLAN20

Afanasiev_LAN5_Router(config-if)#network 172.24.238.64 255.255.255.192

Afanasiev_LAN5_Router(config-if)#default-router 172.24.238.65

Afanasiev_LAN5_Router(config-if)#dns-server 172.24.239.100

Afanasiev_LAN5_Router(config-if)#ip dhcp excluded-address 172.24.238.65

Afanasiev_LAN5_Router(config-if)#ip dhcp pool VLAN30

Afanasiev_LAN5_Router(config-if)#network 172.24.238.128 255.255.255.192

Afanasiev_LAN5_Router(config-if)#default-router 172.24.238.129

Afanasiev_LAN5_Router(config-if)#dns-server 172.24.239.100

Afanasiev_LAN5_Router(config-if)#ip dhcp excluded-address 172.24.238.129

Afanasiev_LAN5_Router(config-if)#ip dhcp pool VLAN99

Afanasiev_LAN5_Router(config-if)#network 172.24.238.192 255.255.255.240

Afanasiev_LAN5_Router(config-if)#default-router 172.24.238.193

Afanasiev_LAN5_Router(config-if)#dns-server 172.24.239.100

Afanasiev_LAN5_Router(config-if)#ip dhcp excluded-address 172.24.238.193

Afanasiev_LAN5_Router(config-if)#int gig0/0.99

Afanasiev_LAN5_Router(config-if)#encapsulation dot1Q 99

Afanasiev_LAN5_Router(config-if)#ip address 172.24.238.193 255.255.255.240

Afanasiev_LAN5_Router(config-if)#int gig0/0.10

Afanasiev_LAN5_Router(config-if)#encapsulation dot1Q 10

Afanasiev_LAN5_Router(config-if)#ip address 172.24.238.1 255.255.255.192

Afanasiev_LAN5_Router(config-if)#int gig0/0.20

Afanasiev_LAN5_Router(config-if)#encapsulation dot1Q 20

Afanasiev_LAN5_Router(config-if)#ip address 172.24.238.65 255.255.255.192

Afanasiev_LAN5_Router(config-if)#int gig0/0.30

Afanasiev_LAN5_Router(config-if)#encapsulation dot1Q 30

Afanasiev_LAN5_Router(config-if)#ip address 172.24.238.129 255.255.255.192

Зміст налаштувань VLAN у LAN5 можна побачити на рисунку 3.4, зміст таблиці DHCP роутера LAN5 зображено на рисунку 3.5.

```
LAN5_1#sh vlan br

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active
10   Marketing                         active    Fa0/4, Fa0/5, Fa0/6, Fa0/7
                                                 Fa0/8
20   CustomerService                   active    Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                 Fa0/13, Fa0/14
30   Management                        active    Fa0/15, Fa0/16, Fa0/17, Fa0/18
                                                 Fa0/19, Fa0/20
99   RemoteAccess                      active
100  TRUNK                             active
200  PILE                              active    Fa0/1, Fa0/2, Fa0/3, Fa0/21
                                                 Fa0/22, Fa0/23, Gig0/1, Gig0/2
1002 fddi-default                      active
1003 token-ring-default                active
1004 fddinet-default                   active
1005 trnet-default                     active
```

Рисунок 3.4 – Налаштування VLAN на LAN5_Switch_1

```
LAN5#sh ip dhcp binding
IP address       Client-ID/              Lease expiration        Type
                 Hardware address
172.24.238.3     00D0.589B.1ED8            --                    Automatic
172.24.238.4     00E0.8F59.0C42            --                    Automatic
172.24.238.2     0004.9AA4.CA25            --                    Automatic
172.24.238.5     0002.4AC6.8C32            --                    Automatic
172.24.238.67    000A.F313.5961            --                    Automatic
172.24.238.66    0040.0B8E.1CC6            --                    Automatic
172.24.238.69    0005.5E9B.9447            --                    Automatic
172.24.238.68    0030.A3A7.2B9C            --                    Automatic
172.24.238.130   0009.7CD9.4310            --                    Automatic
172.24.238.131   0002.173A.0D49            --                    Automatic
172.24.238.132   000A.4153.993C            --                    Automatic
172.24.238.133   00D0.BC65.335C            --                    Automatic
172.24.238.194   00D0.D352.4801            --                    Automatic
172.24.238.195   000C.8564.0701            --                    Automatic
172.24.238.196   0004.9A70.4101            --                    Automatic
```

Рисунок 3.5 – Видані за DHCP адреси на роутері Afanasiev_LAN5

Останнім кроком налаштування безпекових функцій маршрутизаторі є розгортання NAT на крайньому роутері мережі основного офісу та встановлення захищеного IPsec тунелю між ним та підсистемою LAN1. Для серверів мережі було встановлено статичний NAT із визначеної мережі 209.165.200.0/29, де адреса з номером 4 надається HTTP серверу, 3 DNS серверу та 2 для перекриття PAT усіх інших вузлів.

Налаштування NAT на Afanasiev_Edge_Router:

Для початку визначаються внутрішні та зовнішні інтерфейси роутера:

Edge(config)#int g0/3/0

Edge(config)#ip nat inside

Edge(config)#int g0/2/0

Edge(config)#ip nat outside

Створюється список ACL, який налаштовується таким чином, що усі пакети які йдуть назовні проходять перетворення NAT, але пакети до мережі LAN1 залишаються незмінними:

Edge(config)#ip access-list extended 105

Edge(config)#deny ip 172.24.234.0 0.0.1.255 172.24.232.0 0.0.1.255

Edge(config)#deny ip 172.24.236.0 0.0.1.255 172.24.232.0 0.0.1.255

Edge(config)#deny ip 172.24.238.0 0.0.0.255 172.24.232.0 0.0.1.255

Edge(config)#deny ip 172.24.239.0 0.0.0.127 172.24.232.0 0.0.1.255

Edge(config)#permit ip 172.24.234.0 0.0.1.255 any

Edge(config)#permit ip 172.24.236.0 0.0.1.255 any

Edge(config)#permit ip 172.24.238.0 0.0.0.255 any

Edge(config)#permit ip 172.24.239.0 0.0.0.127 any

Після цього проходить налаштування PAT перекриття та статичних NAT перетворень:

Edge(config)#ip nat pool PAT_POOL 209.165.200.2 209.165.200.2 netmask 255.255.255.248

Edge(config)#ip nat inside source list 105 pool PAT_POOL overload

Edge(config)#ip nat inside source static 172.24.235.150 209.165.200.4

Edge(config)#ip nat inside source static 172.24.239.100 209.165.200.3

Останнім кроком налаштування мережі є створення IPsec тунелю між головним офісом та LAN1. Для цього необхідно налаштувати дві фази захищеного обміну. У першій фазі налаштовується канал за допомогою якого пристрої можуть безпечно обмінюватись інформацією про затвердження захищених шляхів та встановлення захищеного підключення. Друга фаза

налаштування відповідає за безпосереднє шифрування та надсилання даних між мережами. Перед виконанням будь-яких налаштувань необхідно встановити пакет безпекових функцій на кожному з роутерів командою:

Edge(config)#license boot module c2900 technology-package securityk9

Після цього опції для налаштування IPsec каналу стануть доступними і можна приступати до налаштування першої фази, де визначаються: номер політики обміну, протоколи шифрування між маршрутизаторами, вид автентифікації, номер протоколу Діффі-Геллмана для обміну криптографічними ключами.

Edge(config)#crypto isakmp policy 10

Edge(config)#encryption aes 256

Edge(config)#authentication pre-share

Edge(config)#group 5

Після цього визначається ключ автентифікації, та налаштовується друга фаза захищеного тунелю, з встановленням правил протоколів шифрування даних, мапа криптографічного шифрування, та адреси маршрутизатора до якого налаштовується тунель. Також налаштовується список адрес для яких буде застосовуватися криптографічний захист:

Edge(config)#ip access-list extended 110

Edge(config)#permit ip 172.24.234.0 0.0.1.255 172.24.232.0 0.0.1.255

Edge(config)#permit ip 172.24.236.0 0.0.1.255 172.24.232.0 0.0.1.255

Edge(config)#permit ip 172.24.238.0 0.0.0.255 172.24.232.0 0.0.1.255

Edge(config)#permit ip 172.24.239.0 0.0.0.127 172.24.232.0 0.0.1.255

Edge(config)#crypto isakmp key GAME address 64.100.13.2

Edge(config)#crypto ipsec transform-set VPN-SET esp-aes esp-sha-hmac

Edge(config)#crypto map VPN-MAP 10 ipsec-isakmp

Edge(config)#set peer 64.100.13.2

Edge(config)#set transform-set VPN-SET

Edge(config)#match address 110

Після налаштувань другої фази, на зовнішньому інтерфейсі маршрутизатора встановлюється раніше налаштована криптографічна мапа:

Edge(config)#int g0/2/0

Edge(config)#crypto map VPN-MAP

Для маршрутизатору мережі LAN1 виконано ідентичні налаштування зі зміненими адресами:

LAN1(config)#ip access-list extended 110

LAN1(config)#permit ip 172.24.232.0 0.0.1.255 172.24.234.0 0.0.1.255

LAN1(config)#permit ip 172.24.232.0 0.0.1.255 172.24.236.0 0.0.1.255

LAN1(config)#permit ip 172.24.232.0 0.0.1.255 172.24.238.0 0.0.0.255

LAN1(config)#permit ip 172.24.232.0 0.0.1.255 172.24.239.0 0.0.0.127

LAN1(config)#crypto isakmp key GAME address 209.165.202.2

LAN1(config)#crypto ipsec transform-set VPN-SET esp-aes esp-sha-hmac

LAN1(config)#crypto map VPN-MAP 10 ipsec-isakmp

LAN1(config)#set peer 209.165.202.2

LAN1(config)#set transform-set VPN-SET

LAN1(config)#match address 110

Останнім кроком налаштування мережі є підключення відповідних сервісів на серверах компанії. Налаштування DNS серверу зображені на рисунку 3.6, Налаштування HTTPS та FTP можна побачити на рисунку 3.7.

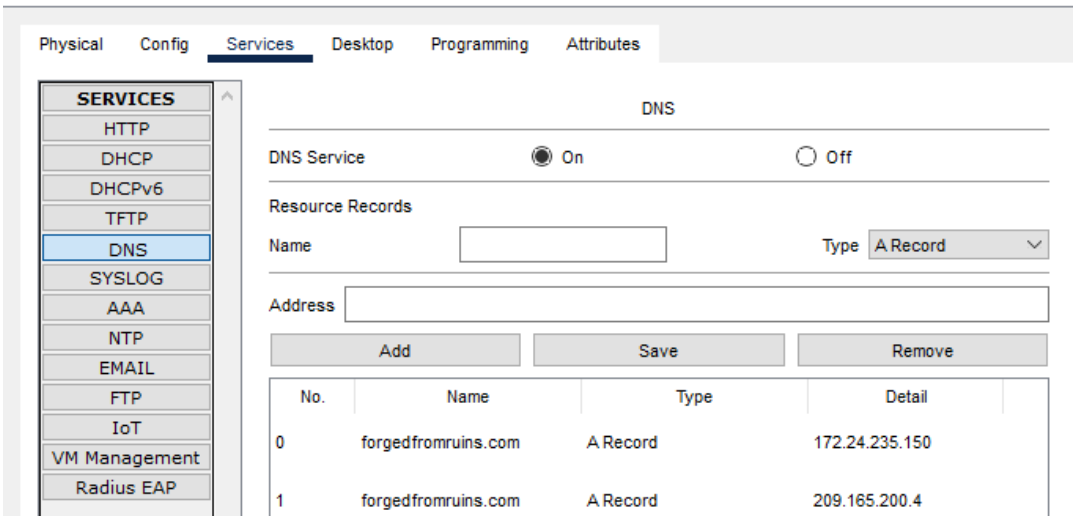

Рисунок 3.6 – Налаштування DNS серверу

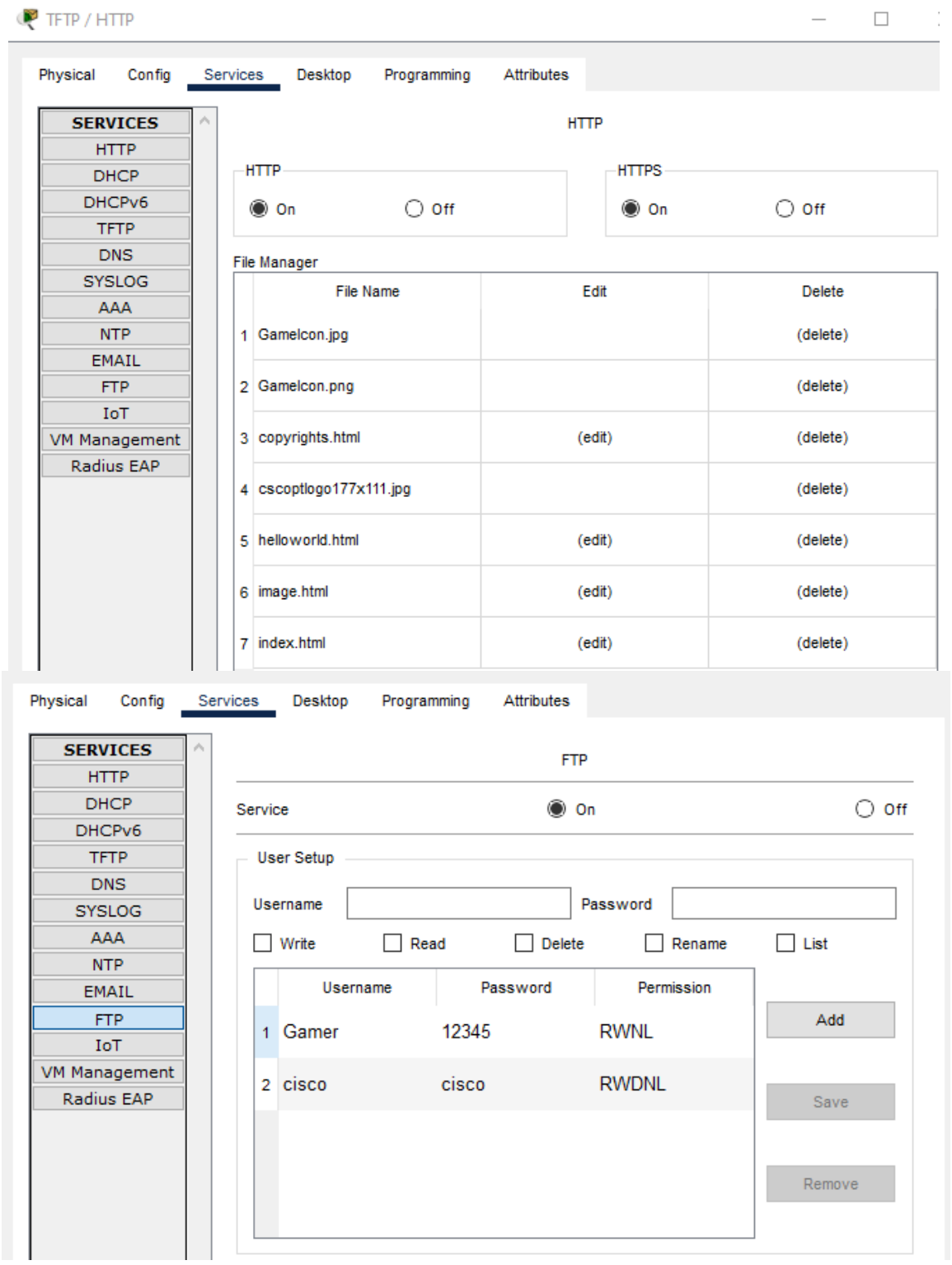

Рисунок 3.7 – Налаштування HTTP / FTP серверу

3.3 Перевірка роботи встановлених налаштувань

Після побудови логічної топології у Cisco Packet Tracer, та проведення зазначених вище налаштувань. Перевіримо базові налаштування маршрутизації шляхом надсилання пакетів між локальними мережами LAN2, LAN3, LAN4 та LAN5, зважаючи на рисунок 3.8, маршрутизація всередині головного офісу працює без помилок.

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit
	Successful	PC3	PC18	ICMP		0.000	N	0	(edit)
	Successful	PC18	PC38	ICMP		0.000	N	1	(edit)
	Successful	PC38	PC29	ICMP		0.000	N	2	(edit)
	Successful	PC29	PC3	ICMP		0.000	N	3	(edit)

Рисунок 3.8 – Результат надсилання пакетів між підсистемами

Перевіримо роботу NAT та перетворення статичних адрес серверів та PAT перекриття. Для цього було надіслано пакети з сервера та одного з вузлів на ПК у мережі ISP. Результати перетворень на маршрутизаторі Afanasiev_Edge_Router можна побачити на рисунках 3.9-3.10.

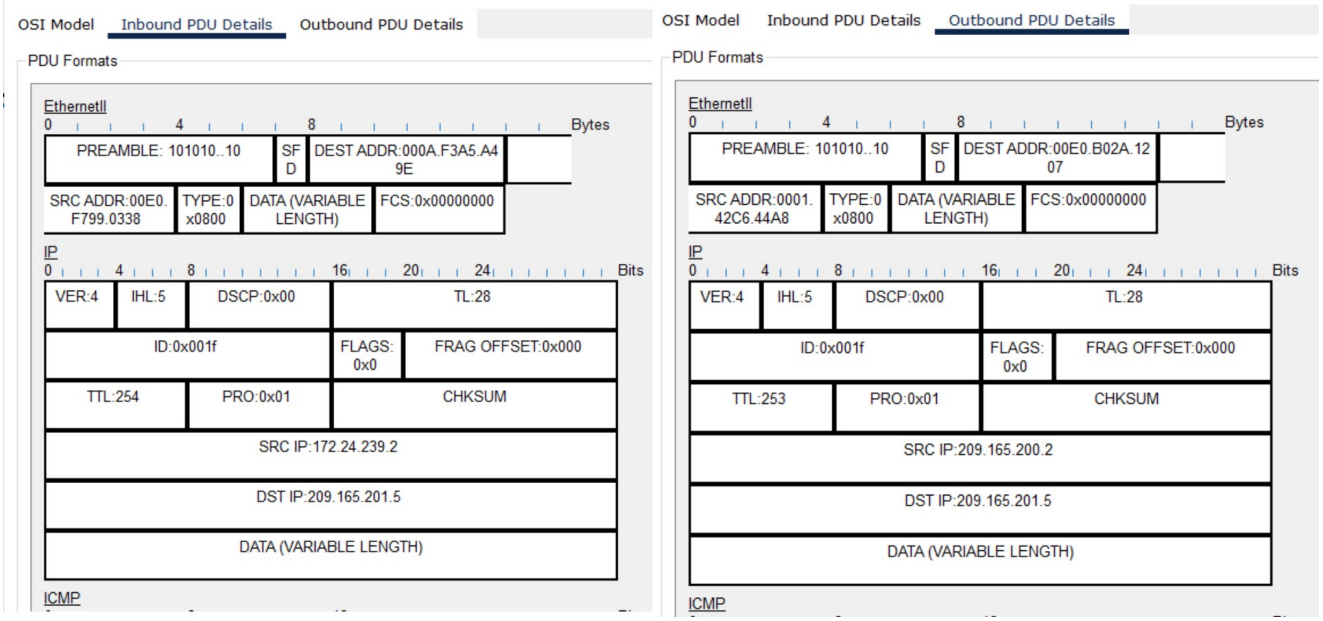

Рисунок 3.9 – Результат перетворення пакету вузла з використанням PAT

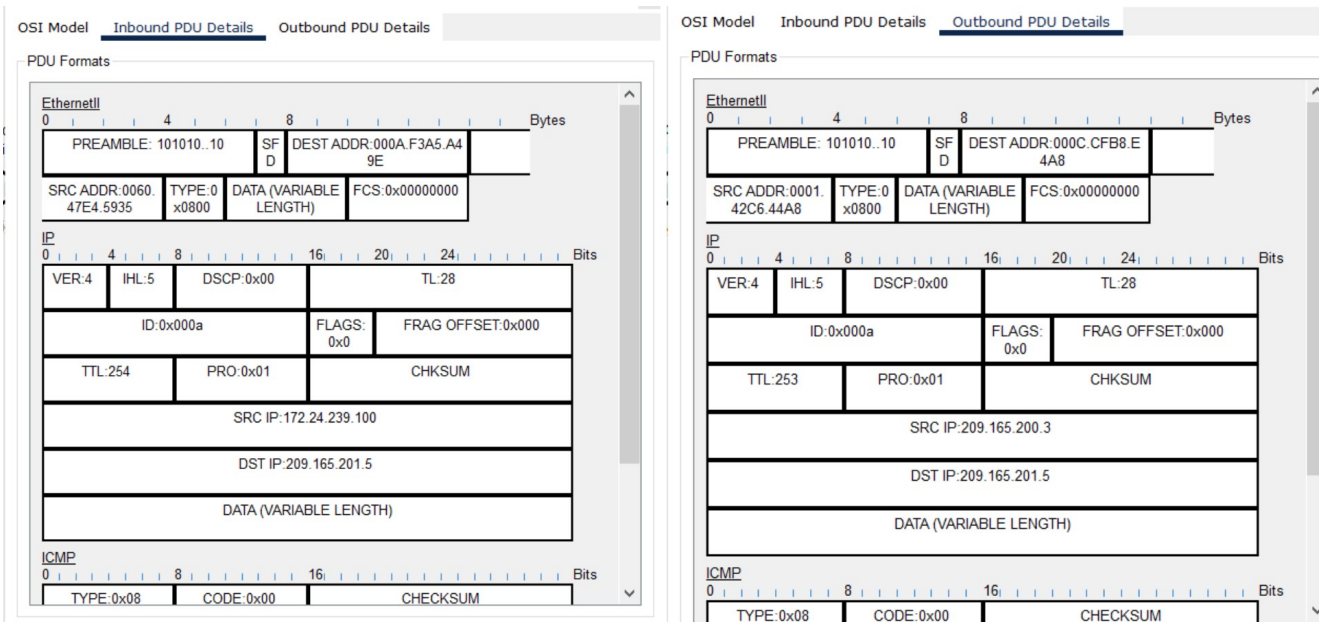

Рисунок 3.10 – Результат перетворення пакету серверу на статичну адресу NAT

Перевірено правильність налаштування IPsec VPN-тунелю між офісами, було надіслано пакет з мережі головного офісу до LAN1, результат перетворення пакету на роутері Afanasiev_Edge_Router можна побачити на рисунку 3.11. Додатково підтвердити встановлення захищеного зв'язку можна виконавши команду: *show crypto isakmp sa,* результат зображено на рисунку 3.12.

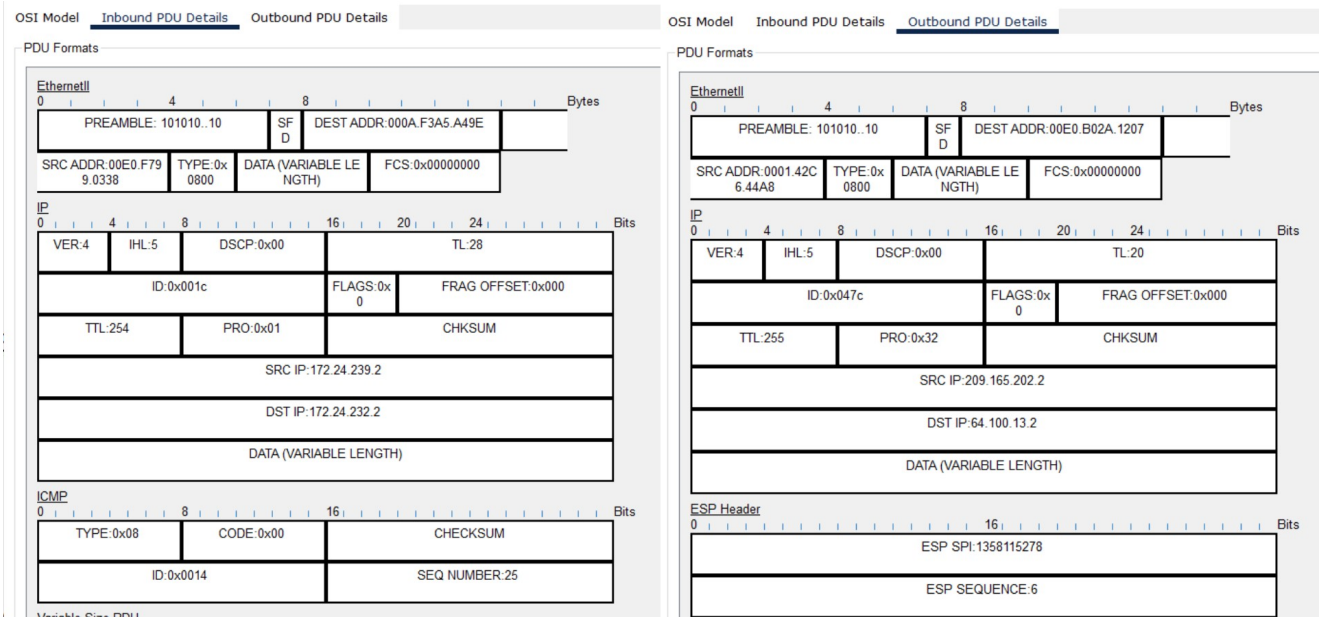

Рисунок 3.11 – Результат перетворення та шифрування пакету

```
EDGE#show crypto isakmp sa
IPv4 Crypto ISAKMP SA
dst             src             state          conn-id slot status
64.100.13.2     209.165.202.2   QM_IDLE           1012    0 ACTIVE
```

Рисунок 3.12 – Результат успішного IPsec з’єднання

Підсумовуючи, усі основні механізми комп’ютерною мережі визначені у технічних вимогах були реалізовані та працюють у відповідності до вимог.

4 РОЗРОБКА КОМПОНЕНТА СИСТЕМИ

4.1 Визначення базової архітектури проєкту

Визначення архітектури та планування основних елементів проєкту на початковому етапі його розробки є однією з найважливіших задач при розробці відеогри. Архітектура гри та формування методів взаємодії окремих модулів один із одним є складним завданням, проте окреслення цих речей з самого початку дозволить провести розробку продукту плавніше, адже внесення змін у фундаментальні принципи функціонування архітектури стає все складніше із кожним написаним рядком коду.

Враховуючи направлення проєкту, а саме 2D-Roguelite, визначимо його характерні особливості:

- вид зверху, що означає використання виключно осей X та Y для переміщення;
- система процедурної генерації рівнів;
- система контролю персонажа гравця;
- система управління ворогами та їх поведінкою;
- система користувацького інтерфейсу;
- система звукових ефектів;
- система інвентарю гравця, предметів та можливостей їх використання;
- система збереження прогресу гравця між ігровими сесіями;
- система випадкового генерування ігрових предметів;

Основна робота з рушієм Unity відбувається у сценах, вони є контейнерами для ігрових об’єктів, і в середині них знаходяться усі активні елементи гри. У проекті будуть використовуватися MonoBehaviour та ScriptableObject - це два фундаментальні класи в архітектурі Unity, кожен з яких виконує різні ролі в організації ігрової логіки та даних. Архітектура Unity побудована навколо GameObjects – це будь-які об’єкти, що знаходяться у ігровій сцені, отже будь-який активний об’єкт у Unity є GameObject. Це не стосується об’єктів, що існують у пам’яті рушія Unity, адже вони не є елементами сцени, і слугують для звернення

рушія до даних або об’єктів, що знаходяться у файлах гри.

MonoBehaviour - це основний базовий клас, від якого походять усі скрипти Unity, якщо вони призначені для приєднання до GameObject як компоненти. Він інтегрує написаний розробником C# код у компонентну архітектуру Unity, дозволяючи скрипту брати участь у циклах оновлення рушія, викликах фізики, подіях рендерингу та інших процесах. Класи, що успадковані від MonoBehaviour, можуть перевизначати стандартний набір методів, які Unity викликає у певні моменти під час виконання:

− Awake() − викликається одразу при завантаженні класу;

− OnEnable() − викликається кожного разу коли об’єкт стає активним;

− Start() − викликається перед виконанням першого Update(), але після викликів Awake() усіх інших класів;

− Update() − викликається кожен ігровий кадр.

− FixedUpdate() − викликається за зафіксованим інтервалом.

− OnDisable() − викликається кожного разу коли об’єкт стає неактивним.

− OnDestroy() − викликається при знищенні об’єкту.

ScriptableObject - це клас-контейнер даних, призначений для зберігання даних. На відміну від MonoBehaviour, він не вимагає приєднання до GameObject, натомість, екземпляри зберігаються як .asset файли у директорії Assets ігрового проекту. Оскільки дані зберігаються як файл, значення зберігаються між сеансами відтворення і можуть бути використані у різних сценах та об'єктах. Їх використання є ефективнішим для збереження пам’яті, адже кілька посилань на один і той самий ScriptableObject спільно використовують базовий набір даних, зменшуючи дублювання даних у пам’яті. Також це дозволяє зменшити використання жорстко закріплених даних у інших класах, що допомагає розділяти дані та поведінковий код, хоча це не означає що ScriptableObject не може мати власні функції та виконувати їх під час роботи проєкту.

4.2 Написання коду основних ігрових систем

4.2.1 Головний скрипт управління

Головним елементом роботи програми є клас GameManager, він є об’єктом шаблону “Одинак”. Будь-який наступний екземпляр GameManager негайно знищується, забезпечуючи існування лише одного екземпляру класу. Переходи між станами гри представлені змінною типу GameState, а поточний і попередній стан записуються у спеціальних полях. Щоразу, коли відбувається зміна стану, метод UpdateGameState(GameState newState) зображений на рисунку 4.1 оновлює ці поля, за необхідності, та транслює зміну через статичну подію OnGameStateChanged, підписавшись на яку за програмним шаблоном “Спостерігач”, інші скрипти можуть отримувати оновлення внутрішнього стану гри. Цей механізм синхронізує підсистеми не вдаючись до розрізнення скриптів, що її потребують.

```
public void UpdateGameState(GameState newState)
{
    OldState = State;
    State = newState;
    switch (newState)
    {
        case GameState.MainMenu:
            ResumeTimeFlow();
            DisablePlayer();
            break;
        case GameState.MapSelect:
            ResumeTimeFlow();
            _mapManager = GameObject.FindAnyObjectByType<MapManager>();
            DisablePlayer();
            break;
        case GameState.FightLevel:
            _levelManager = GameObject.FindAnyObjectByType<LevelGenerator>();
            ResumeTimeFlow();
            break;
        case GameState.RewardScreen:
             ResumeTimeFlow();
            break;
        case GameState.Shop:
            ResumeTimeFlow();
            break;
        case GameState.BossLevel:
            _levelManager = GameObject.FindAnyObjectByType<LevelGenerator>();
            ResumeTimeFlow();
            break;
        case GameState.WinScreen:
            break;
        case GameState.LoseScreen:
            break;
        case GameState.PauseScreen:
            break;
    }
    Debug.Log($"We are in {newState}");
    OnGameStateChanged?.Invoke(newState);
}
```

Рисунок 4.1 – Метод UpdateGameState класу GameManager

GameManager також реєструє обробники подій, такі як OnFightLoaded, що викликаються після завантаження сцени. В цьому обробнику також відбувається виклик генератора рівнів, який створює випадкову кількість кімнат від 2 до 6, Після завершення генерації кожний екземпляр класу Room, реєструє свою внутрішню подію OnRoomCleared, яка викликається при зачистці кімнати від ворогів, у менеджері, що дозволяє слідкувати за прогресом гравця на рівні та коли лічильник активних кімнат досягає 0, гравець повертається на сцену з картою рівнів. На початку існування GameManager також створюється постійний об’єкт гравця за допомогою використання заготованого префаба. Це дозволяє іншим скриптам завжди мати доступ до скриптів гравця, та його поточного місцезнаходження, а також запобігає дублюванню елементів та даних пов’язаних з інвентарем гравця та його параметрами, що можуть бути пов’язані з його постійним створенням у сцені та знищенням.

Інвентар і предмети гравця обробляються через посилання на ScriptableObject з класу InventorySO. На початку кожного забігу, виконуються такі методи як ResetLootbase() та SpawnStartingEquipment(), які забезпечують чисту генерацію предметів та надають гравцю стартове спорядження. Перемога чи поразка у грі визначається за допомогою визначених подій Enemy.OnBossDeath та PlayerController.OnPlayerDeath відповідно. Детальний код класу GameManager надано у Додатку Б.

4.2.2 Система генерації мапи рівнів

В основі проекту лежить процедурна система генерації мап, яка створює розгалужені шляхи, подібні до тих, що можна знайти в Slay the Spire [15]. Архітектура підтримує декілька орієнтацій, дозволяючи мапам рухатися в будь-якому кардинальному напрямку (зліва направо, справа наліво, зверху вниз або знизу вгору). Така гнучкість робить систему адаптивною до різних ігрових дизайнів та орієнтацій екрану. Проект включає декілька важливих компонентів Unity та сторонніх ресурсів. DOTween слугує анімаційним фреймворком, що

забезпечує плавні переходи та візуальні ефекти у всьому інтерфейсі мапи. Система використовує Newtonsoft.JSON для Unity для управління серіалізацією та десеріалізацією даних мапи, що дозволяє зберігати стани збереження між ігровими сесіями. Процедурна система генерації включає складне розміщення вузлів з можливостями рандомізації для запобігання повторюваності макетів.

Алгоритми усунення перехресних вузлів забезпечують чисті, прохідні шляхи без конфліктуючих з'єднань. Система підтримує конфігуровані типи вузлів, що дозволяє розробникам визначати різні типи зустрічей у різних точках карти. Генератор карти починає роботу з читання визначеного конфігураційного файлу MapConfig, де розташовані налаштування кількості вузлів, кількості шарів генерації, їх типів, розташування та вірогідностей появи на кожному кроці проходження. Процедура генерації починається з розміщення кожної вершини у визначеній координаті шару, потім для кожного первинного «маршруту» вибирається початкова вершина у шарі 0 і просувається шар за шаром: на кожному кроці вибирається вершина, індекс якої лежить у межах невеликого зсуву від попереднього вибору.

Після прокладення основних шляхів, алгоритм вводить додаткові зв’язки між шарами, що підвищує варіативність обирання шляху та дає можливість переходити з одного шляху на інший. Приклад такої генерації можна побачити на рисунку 4.2.

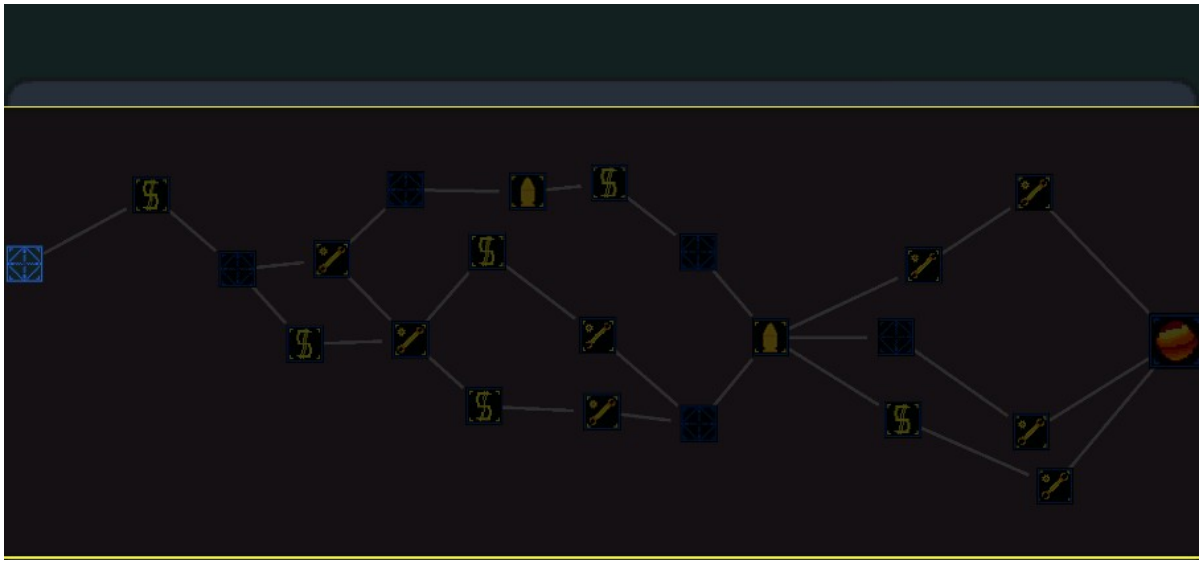

Рисунок 4.2 – Приклад генерації мапи рівнів

4.2.3 Система генерації процедурних ігрових рівнів

Для генерації рівнів на яких гравцю необхідно перемагати ворогів використовується скрипт LevelGenerator, повний код якого можна знайти у Додатку Б. Клас організовує процедурну побудову рівнів за шляхом відтворення заготованих раніше префабів кімнат. Після виклику функції GenerateLevel() клас запускає асинхронну операцію з генерування рівня. Першою операцією є видалення усіх об’єктів, що могли залишитися після попереднього виклику генерації, після цього на нульових координатах створюється заздалегідь заготована стартова кімната startRoom, яка зберігає у генераторі точку playerStartPosition, яка пізніше буде використана для виклику гравця на рівень.

Після створення кімнати, алгоритм шукає точки виходу та починає ітерувати цикл, який буде виконуватись доки кількість кімнат на рівні не досягне значення maxRooms або на рівні не залишиться жодного вільного виходу у жодній з кімнат. Отже, генератор випадково вибирає вихід, та шукає префаб кімнати який має вихід протилежний до обраного, для запобігання нескінченним спробам підібрати кімнату для створення, у циклі є обмежена кількість спроб підбору префабу для кожного з виходів, після чого він буде вважатися закритим. Для кожної кімнати що була успішно підібрана як підходяща для певного виходу, створюється тимчасовий екземпляр на нульових координатах, для розрахунку координат, які будуть надані кімнаті при її кінцевому створенні. Після цих обчислень, тимчасова кімната знищується, і кімната створюється на попередньо розрахованих координатах, після чого запускається перевірка, яка визначає, чи перекриває геометрія новоствореної кімнати вже існуючі на рівні кімнати. У випадку перетину, кімната відкидається як потенційний кандидат і цикл починається наново. У випадку проходження тесту на перетин, кімната активується та додається до списку розташованих кімнат placedRooms. Вихід що був залучений до під’єднання кімнати до попередньої, виключається з розгляду алгоритмом як вже зайнятий. Після завершення усіх ітерацій алгоритму, виконується активація навігаційної сітки у розміщених кімнатах. Дана сітка використовується ворогами для переміщення та навігації рівнем.

Приклад створеного за допомогою цього класу рівня з 40 кімнат можна побачити на рисунку 4.3.

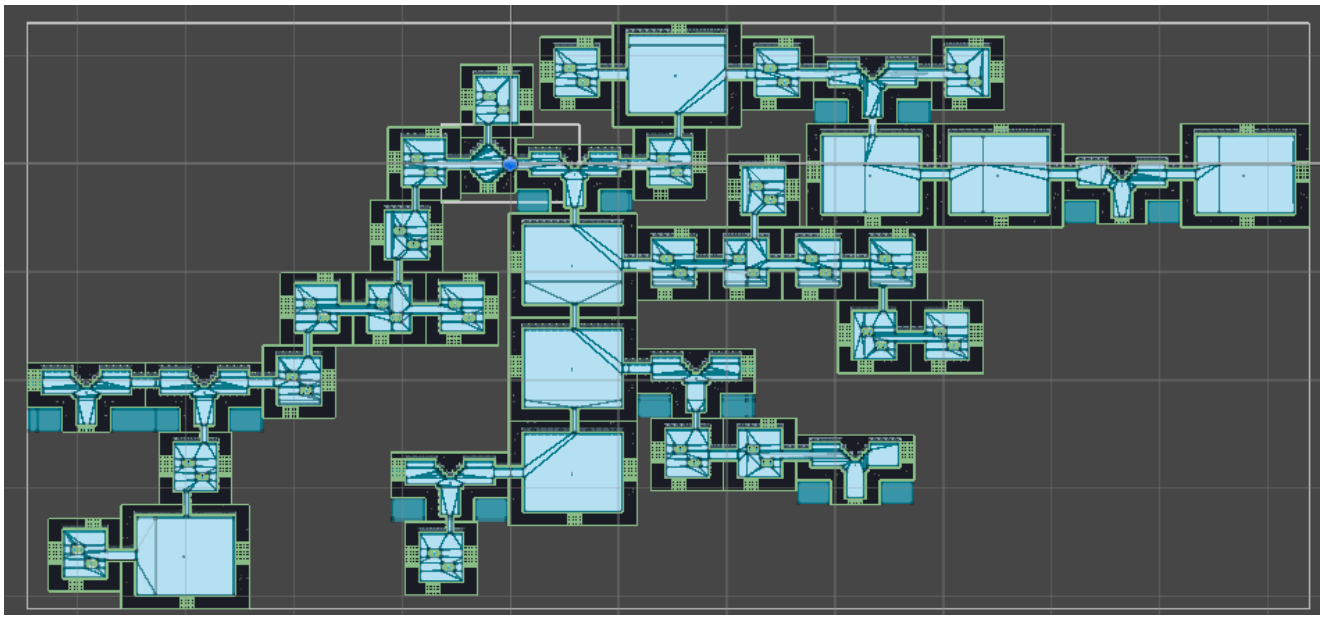

Рисунок 4.3 – Приклад генерації рівня

Префаб кожної з кімнат повинен мати схожу структуру побудови, для їхньої правильної генерації, цю структуру можна побачити на рисунку 4.4.

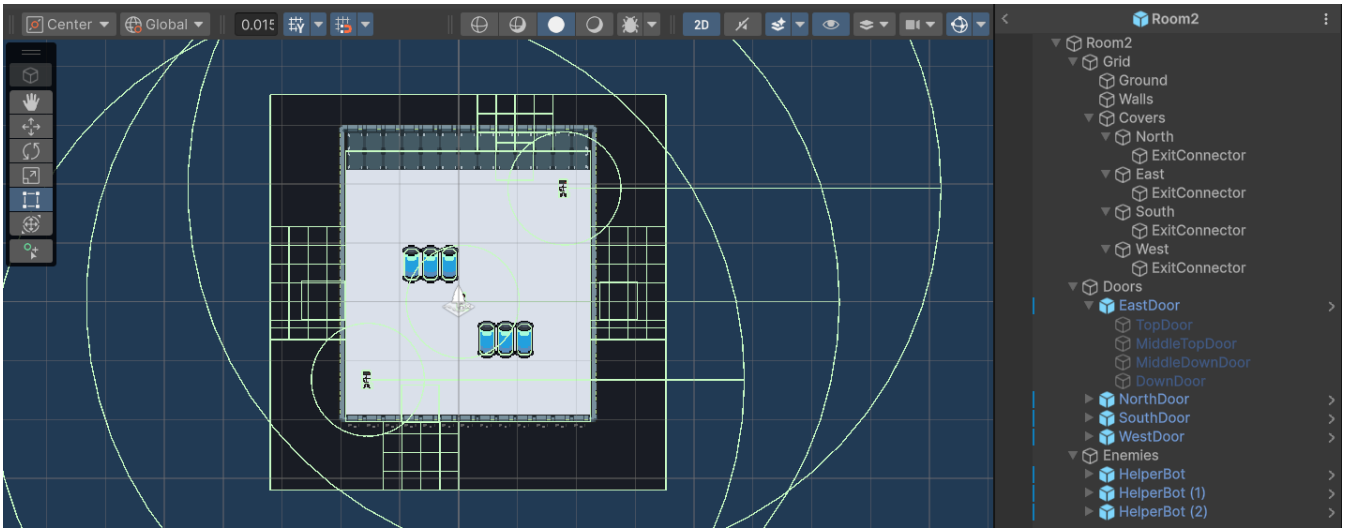

Рисунок 4.4 – Структура кімнати

Кожна кімната має елемент Grid, який використовується для рендеру ігрових тайлів, з яких складаються рівні. У кожній кімнаті є 3 типи карт тайлів: підлога,

стіни та накладки. Карти тайлів використовуються для розділу тайлів та надання їм певних особливостей в залежності від контексту. Наприклад тайли стін зупиняють кулі та гравця, адже це є їхнім призначенням, тому вони мають компонент Rigidbody2D, який використовується у Unity для фізичних взаємодій між об’єктами. Кожному об’єкту у Unity може бути призначений тег та рівень взаємодії на якому він знаходиться, що дозволяє гнучко налаштовувати комплексні взаємодії між об’єктами, наприклад: кулі гравця та супротивників знаходяться на різних рівнях взаємодії та ніяк напряму не впливають один на одного. У префабах кімнат рівні взаємодії використовуються для визначення які елементи кімнати будуть враховуватися при розрахунку їхніх меж генератором рівнів. Кожний елемент, який має фізичний колайдер та повинен бути врахований у обчисленнях, знаходиться на рівні Room та має тег CollidableWall. Таким чином, усі елементи що мають колайдери, але не знаходяться на рівні Room ігноруються генератором рівнів під час обробки фізичних меж кімнати.

Кожна кімната має визначену кількість входів/виходів що приймають участь у генерації, вони можуть відповідати одному із чотирьох напрямків компасу: північ, схід, південь, захід. Кількість виходів може бути меншої за визначену, але не більшою. Кожен вихід має визначену “накладку”, яка вимикається у разі активації виходу що відкриває прохід до іншої кімнати. Кожна з накладок є власною картою тайлів, а також має скрипт ExitTilemap, в якому зазначається напрямок виходу та точка під’єднання.

Для кожного виходу у кімнаті розташовані окремі об’єкти у вигляді дверей. У випадку якщо гравець заходить до кімнати і вона має ворогів, усі двері зачиняються та поки гравець не переможе ворогів, вийти з кімнати буде неможливо. Така взаємодія забезпечується скриптами DoorTrigger, який надсилає сигнал у разі знаходження гравця у визначеній зоні, після цього скрипт RoomDoor перевіряє умови активації та надсилає відповідні сигнали до скрипту Room, який у разі наявності ворогів у кімнаті починає їх активацію. Приклад налаштувань скриптів Room, RoomDoor, ExitTilemap для раніше зображеної структури кімнати ілюстровано на рисунку 4.5.

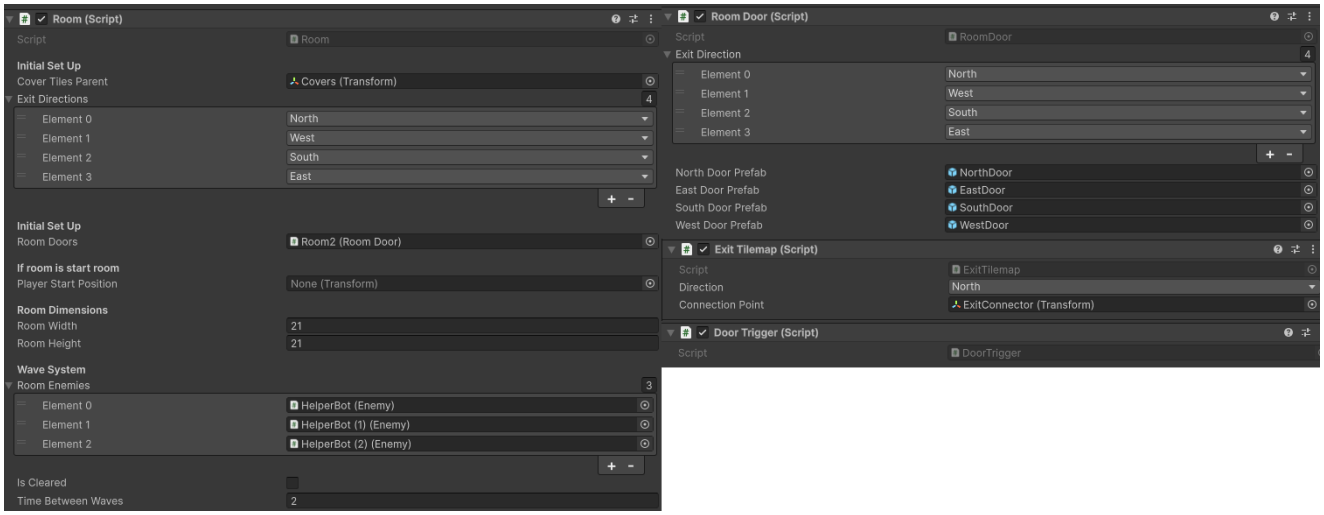

Рисунок 4.5 – Приклад налаштувань скриптів кімнати

4.2.4 Система управління персонажем

Система управління ігровим персонажем побудована на основі 4 скриптів: PlayerController, PlayerMovement, PlayerActions, GunSelector. PlayerController є центральним скриптом, в якому зібрано виклик усіх інших пов’язаних з гравцем функцій. Також, він керує взаємодією з інвентарем гравця та предметами що в ньому знаходяться. Код забезпечує такі можливості як: наявність інвентарю застосованого спорядження а також інвентарю запасних предметів. У коді йде перевірка застосованих до гравця модифікаторів параметрів, що можуть надходити від спорядження або пасивних предметів, що знаходяться у звичайному інвентарі. Контролер також відповідає за систему здоров’я гравця та її обробку, взаємодію з колайдерами ворожих пострілів та передачу даних параметрів до ScriptableObject PlayerAttributesSO, який зберігає стан параметрів гравця.

Скрипт PlayerMovement зчитує ввід інформації від периферійних пристроїв користувача, та відповідає за переміщення гравця, а саме надання його фізичному компоненту Rigidbody2D лінійного прискорення в залежності від параметрів зазначених у PlayerAttributesSO, також даний скрипт відповідає за обробку механіки ухилення гравця від ворожих куль. У випадку якщо гравець натисне кнопку “Пробіл” на клавіатурі, персонаж почне швидке прискорення у заданому

напрямку та у зазначений період гравець не буде зазнавати пошкоджень від зовнішніх загроз, проте він також не зможе вести власний вогонь чи вільно переміщуватись під час виконання маневру ухилення.

PlayerActions також зчитує вхідну інформацію від пристрою користувача, але виконує функцію контролю та взаємодії гравця з такими елементами системи як: стрільба та перезаряджання зброї. При потенційному доопрацюванню системи, він буде відповідати за взаємодію гравця з активними елементами світу, наприклад: контейнери зі спорядженням або станції лікування.

GunSelector відповідає за роботу спорядженої на гравці зброї. Інвентар спорядження має 2 відділи для основного озброєння, яке можна обмінювати один між одним за натисканням на клавішу “1” та використовувати для стрільби натисканням лівої кнопки миші. Також на гравці є відділ для додаткового озброєння, яке можна використати натисканням правої кнопки миші. GunSelector представляє з себе проміжний скрипт, завданням якого є налагодження взаємодії між системами інвентарю та системою зброї.

Детальний код класів PlayerController, PlayerMovement та PlayerActions можна знайти у Додатку Б.

4.2.5 Інвентарна система

Для забезпечення цікавості та відчуття новизни для кожного нового забігу гравця, у грі було реалізовано систему предметів та інвентарю. Дана система базується на відкритому коді [16], з внесенням деяких змін для потреб проєкту.

В основі інвентарної системи лежить клас ItemObject, який є об’єктом ScriptableObject та зберігає такі дані про предмет як: текст опис, спрайт зображення, рідкість появи, мінімальна та максимальна кількість у одному інвентарному слоті, необхідність попереднього розблокування. Під час виконання гри, даний клас виконує роль контейнера даних та забезпечує стабільність посилань інших класів на базові елементи предметів, що знаходяться у інвентарі. Наступним класом є клас Item, він виконує функцію збереження унікального ідентифікатора кожного предмету та його модифікаторів параметрів гравця, клас

ItemObject має поле типу Item з назвою Data.

Також існують класи, що наслідують від ItemObject – це PrimaryWeaponItem, SecondaryWeaponItem, PassiveItem, ActiveItem, кожен з них є розширеним класом, який є необхідним для відтворення, відповідно: основного озброєння, додаткового озброєння, пасивних та активних предметів.

Важливим класом для підтримки стабільного стану усіх предметів у грі незалежно від обставин є ItemDatabaseSO, це об’єкт ScriptableObject що виконує документування та оновлення даних про кожен ItemObject проєкту, робить він це за допомогою надання кожному об’єкту унікального ідентифікатора на початку будь-якого ігрового процесу. Код даного класу зображено на рисунку 4.6.

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

[CreateAssetMenu(fileName = "Database", menuName = "Inventory/ItemDatabase")]
Unity Script | 5 references
public class ItemDatabaseSO : ScriptableObject, ISerializationCallbackReceiver
{
    public ItemObject[] ItemObjects;

    [ContextMenu("Update ID's")]
    1 reference
    public void UpdateID()
    {
        for (int i = 0; i < ItemObjects.Length; i++)
        {
            if(ItemObjects[i].Data.Id != i)
            {
                ItemObjects[i].Data.Id = i;
                ItemObjects[i].InitializeType();
            }
        }
    }

    0 references
    public void OnAfterDeserialize()
    {
        UpdateID();
    }

    0 references
    public void OnBeforeSerialize()
    {
```

Рисунок 4.6 – Код класу ItemDatabaseSO

Останній ScriptableObject що реалізує систему інвентарю це InventorySO, який виконує функції зберігання та маніпулювання предметами гравця, що знаходяться у його змінній Container, класу Inventory. Inventory в свою чергу є класом що зберігає в собі об’єкти InventorySlot, кожен з яких є контейнером для індивідуальних об’єктів класу Item.

Для візуалізації та надання гравцю можливості фізично взаємодіяти з InventorySO існує скрипт UserInventoryInterface. Він реалізує функції виведення візуального представлення стану прив’язаного до нього об’єкту InventorySO. Даний клас також наслідується двома іншими класами: Static та DynamicInventoryInterface, які слугують для відображення статичних та динамічних інвентарних інтерфейсів відповідно.

Таким чином ієрархічну структуру класів, що забезпечують реалізацію інвентарної та речової системи у грі можна побачити на рисунку 4.7.

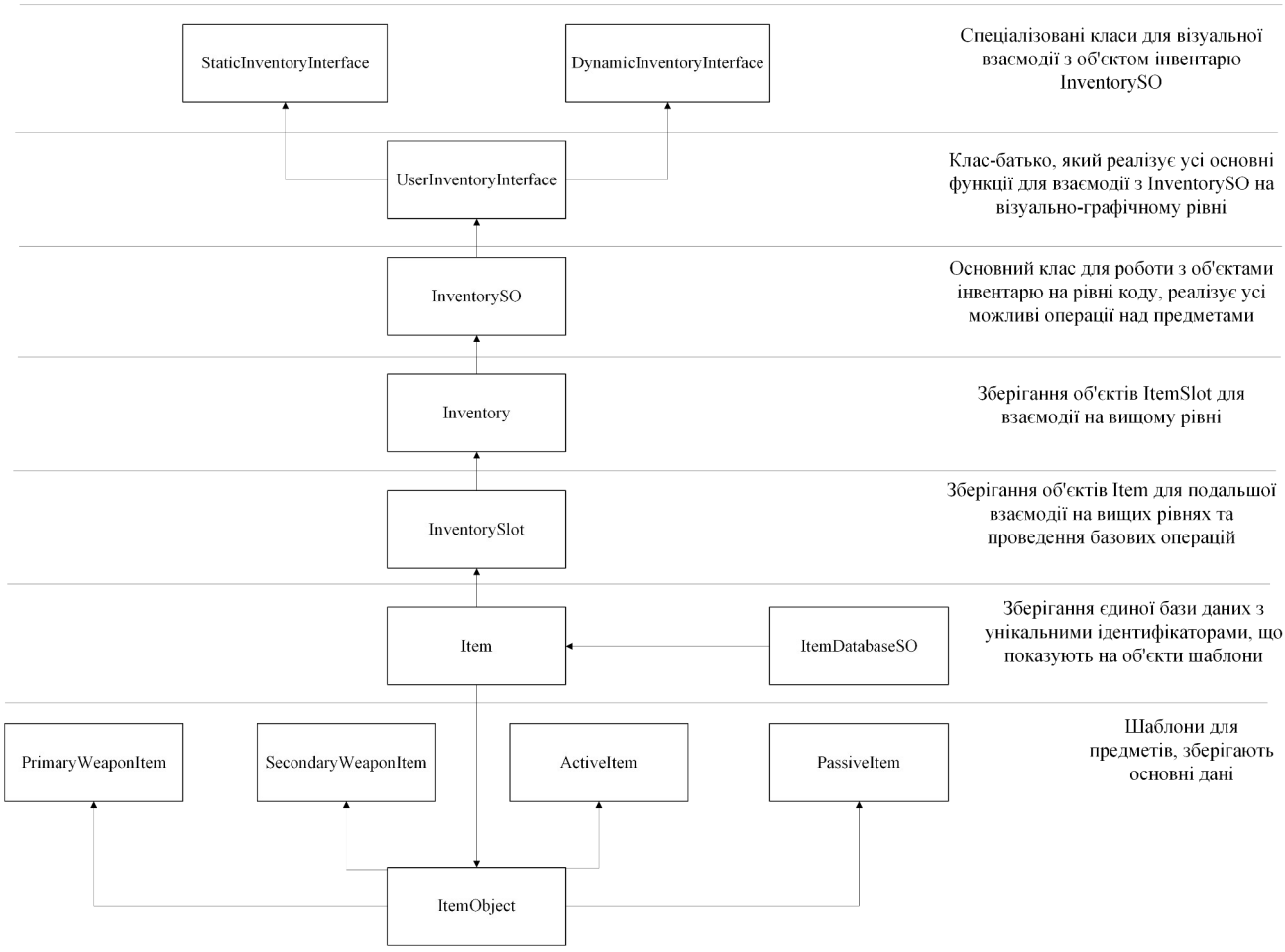

Рисунок 4.7 – Ієрархічна система класів інвентарної системи

4.2.6 Система бою та взаємодії з ворогами

Основним елементом бойової системи гри “Forged from Ruins” є можливість наносити шкоду супротивникам за допомогою снарядів. Для цього було розроблену систему зброї на основі відкритого коду [17].

Система реалізована на основі 4 скриптів ScriptableObject, кожен з яких виконує окремо визначену функцію. Центральним класом є GunScriptableObject, який власне і представляє об’єкт зброї. Він містить в собі дані про спрайт зброї, її позицію у світі, а також реалізує алгоритм стрільби за допомогою функції Shoot(). Також GunScriptableObject містить у собі посилання та використовує у своїй роботі 3 інші скрипти: AmmoConfig, ShootConfig, ProjectileConfig.

AmmoConfig зберігає дані про загальну кількість набоїв у зброї а також набої що заряджені до магазину. В коді реалізовано функції перезаряджання активного магазину, а також поповнення загального боєзапасу.

ShootConfig зберігає дані про технічні характеристики зброї, а саме її швидкострільність та точність, в цьому скрипті також зберігаються налаштування для зброї типу рушниця, кількості снарядів та їхнього розкиду при кожному пострілі.

ProjectileConfig який відповідає за логіку снарядів, їхніх параметрів а також динамічне створення у сцені. Механізм стрільби та створення снарядів реалізований на основі принципу ObjectPooling. У Unity існує можливість створювати об’єкти у сцені динамічно за допомогою функції Instantiate(), проте створення великої кількості об’єктів одночасно або у короткий проміжок часу може призвести до небажаної втрати продуктивності, для вирішення цієї проблеми існує методика “пулінгу”. Це техніка продуктивності, в якій фіксований набір багаторазових об'єктів створюється заздалегідь і періодично активуються та деактивуються за потреби, тим самим мінімізуючи виділення пам'яті, витрати на збір сміття рушієм і стрибки частоти кадрів.

Противники у грі реалізовані з використанням MonoBehaviour скрипту Enemy, який використовує модульну систему поведінки для управління логікою дій противника.

У кожного ворога є три стани в яких він може знаходитись: створення, переслідування та атака, де кожен стан виражений окремим класом типу ScriptableObject. Стан створення є вхідним, в ньому може прописуватись логіка появи ворога у сцені, або логіка його пасивного існування до виявлення гравця. Зі стану створення у стан переслідування ворог переходить при наближені гравця на визначену дистанцію, більш детальна логіка прописується у індивідуальних модулях поведінки. Стан атаки активується після наближення гравця на визначену дистанцію агресивної поведінки, види та логіка атаки описуються у класах що наслідують від базового класу стану.

Зміна станів відбувається за допомогою класу EnemyStateMachine яка має дві функції: ініціалізація та зміна стану на визначений. В якості аргументів обидві функції отримують клас EnemyState, який є базовим класом для інших станів. В ньому реалізовані наступні функції:

– EnterState() – описує логіку при вході у стан;

– ExitState() – описує логіку при виході зі стану;

– FrameActionsUpdate() – викликається у функції Update класу Enemy;

– PhysicalEngineUpdate() – викликається у функції FixedUpdate класу Enemy;

Дані функції є віртуальними, що означає – кожен клас спадкоємець має право змінювати описану у них логіку. Даний клас успадковується класами: AttackState, SpawnState, FollowState, які переписують логіку віртуальних функій та проводять виклик логіки, що визначена у раніше описаних класах ScriptableObject. Таким чином досягається високий рівень інкапсуляції коду, головний механізм функціонування не є доступним ззовні, але зміна поведінки пов’язаної з кожним окремим станом є можливою та гнучкою. Така структура коду дозволяє не тільки створювати різні варіації вже існуючих станів, але і додавати нові варіації з унікальними механіками, при цьому не змінюючи базові елементи системи. У прототипі було реалізовано два типи ворогів, звичайний ворог який атакує гравця коли він заходить у його радіус та босс гри, який є статичним і при цьому має 2 унікальні атаки, усе що відрізняє цих ворогів з перспективи розробника – заміна двох скриптів стану атаки та переслідування у інспекторі об’єкту.

4.2.7 Система користувацького інтерфейсу

Користувацький інтерфейс є важливим елементом взаємодії гравця з продуктом. Відсутність або погана реалізація інтерфейсу може заплутати користувача, або зовсім відторгнути його від взаємодії з проєктом. Саме тому розробка гарного інтерфейсу є значним елементом успіху будь-якої відеогри. В даному проєкті, головна роль управління інтерфейсом відведена класу UIManager, та декільком допоміжним скриптам меншого розміру.

UIManager гарантує, що відповідні елементи інтерфейсу будуть видимими або прихованими в залежності від стану гри. Для анімації елементів інтерфейсу використовується бібліотека DOTween. Клас виконаний за шаблоном “Одинак”, та взаємодіє з системою введення даних пристроями користувача та раніше створеним GameManager класом. Елементи інтерфейсу поділені на групи: HUD, Ігрові екрани та головне меню, і в залежності від сцени в якій знаходиться UIManager, активуються ті чи інші групи інтерфейсу. Реалізація даного механізму показана на рисунку 4.8.

```
private void ConfigureUIForScene(string sceneName)
{
    switch (sceneName)
    {
        case "MainMenu":
            SetHUDActive(false);
            SetGameScreensActive(false);
            SetMainMenuActive(true);
            break;

        case "Gameplay":
            SetHUDActive(true);
            SetGameScreensActive(true);
            SetMainMenuActive(false);
            break;
        case "Map":
            SetHUDActive(false);
            SetGameScreensActive(false);
            SetMainMenuActive(false);
            break;
        default:
            SetHUDActive(true);
            SetGameScreensActive(true);
            SetMainMenuActive(false);
            break;
    }
}
```

Рисунок 4.8 – Механізм активації груп інтерфейсів в залежності від сцени

В якості HUD елементів інтерфейсу реалізовано такі об’єкти як: індикатор рівня здоров’я персонажа, кількість набоїв у активній зброї, кількість валюти, яку можна витратити на придбання предметів.

У ігрових екранах знаходяться такі панелі як: запасний інвентар гравця, інвентар спорядження, інвентар контейнера з нагородою та екрани поразки, перемоги та паузи.

У головному меню гри існують такі підменю як: магазин, де можна витрати зароблену під час гри валюту, меню “Про гру” де перелічено використані сторонні ресурси під час розробки гри. Також в головному меню знаходяться кнопка “Старт”, “Вихід” та повзунок налаштування гучності звуків у грі.

Детальний код класу UIManager знаходиться у Додатку Б.

4.2.8 Система прогресії гравця

У грі реалізовано систему прогресії, кожен раз коли гравець знищує супротивника, він отримує винагороду у вигляді валюти, яку він може витратити на розблокування предметів у магазині в головному меню, після чого ці предмети будуть мати шанс з’явитися під час наступного забігу. Для реалізації цієї системи було зроблено два класу UnlockManager та LootDatabase. UnlockManager відповідає за збереження прогресу гравця, він зберігає дані про розблоковані речі та кількість валюти до спеціального внутрішнього класу Unity – PlayerPrefs. Даний клас є спеціальним файлом, що Unity зберігає локально на комп’ютері гравця, і підвантажує дані з нього за потреби. Таким чином, незалежно від того хто буде намагатися пограти у гру на хмарному сервісі, кожен гравець буде мати локальний файл з їхнім персональним прогресом. Клас LootDatabase відповідає за відслідковування предметів, що мають шанс бути згенерованими та вже були згенеровані під час поточного забігу. Даний клас підвантажує дані з PlayerPrefs та вже на основі збережених у LootDatabase даних, клас LootGenerator проводить випадкову видачу предметів гравцю.

4.3 Створення візуальної та звукової складових гри

4.3.1 Створення графічного представлення

Гра створена у стилі піксельної графіки, незважаючи на давню історію використання, рання графіка була піксельною вимушено, через обмеження апаратного забезпечення того часу. На сьогоднішній день, вибір даного виду графіки є усвідомленим під час розробки нового проєкту, через свою простоту та доступність, будь-який розробник може створити візуальне представлення своєї гри власними силами. Для створення піксельних спрайтів, можна використовувати будь-які графічні редактори, наприклад Photoshop, GIMP або Photopea, проте для цього виду графіку було створено декілька спеціалізованих редакторів. У розробці спрайтів цього проекту було використано програму Aseprite, яка чудово підходить для новачків та має багато функій що полегшують процес створення спрайтів та тайлів. Користувацький інтерфейс програми можна побачити на рисунку 4.9, приклади створених спрайтів для гри включають: спрайти для головного меню та його елементів (рисунок 4.10), спрайти для системи інвентарю (рисунок 4.11), спрайти тайлів для створення рівнів (див. рисунок 4.4)

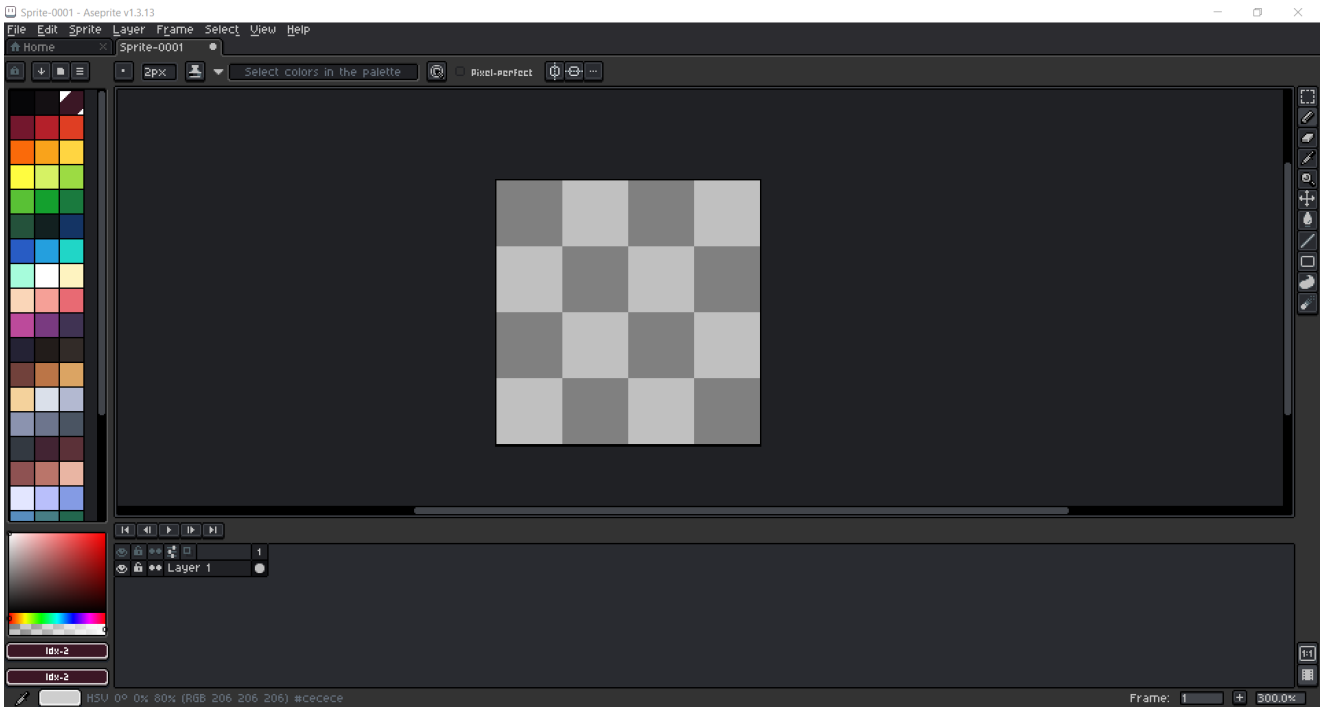

Рисунок 4.9 – Користувацький інтерфейс програми Aseprite

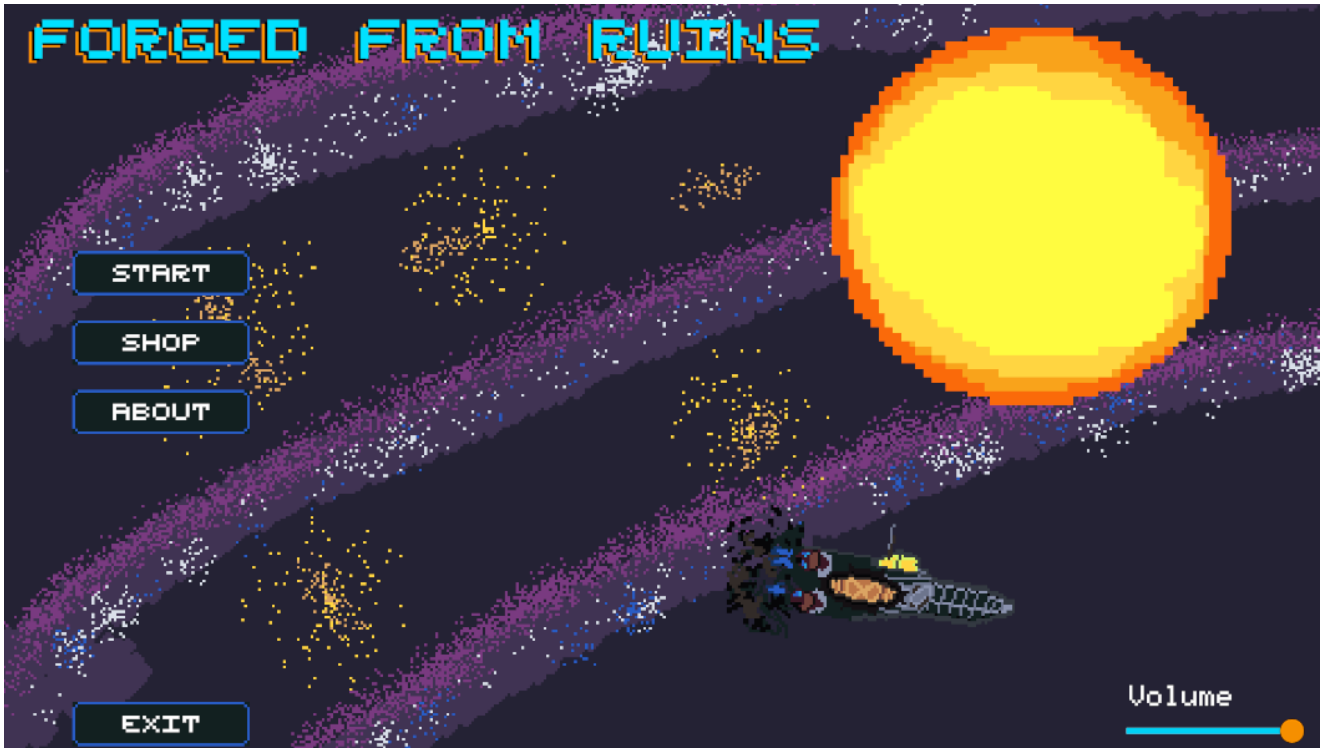

Рисунок 4.10 – Створені спрайти у головному меню

Рисунок 4.11 – Спрайти системи інвентарю

4.3.2 Звукові ефекти ігрового процесу

Звук залучає ще один орган чуття гравця під час ігрового процесу, що робить його більш залученим та дозволяє передавати інформацію гравцю не тільки за допомогою візуальних ефектів. Було використано звукові ефекти для отримання шкоди ворогом та гравцем [18], для ясності ігрового процесу та надання гравцю критичної інформації. Кожній зброї було надано унікальне звучання для покращення емоційної віддачі від гри. Босс гри має власні унікальні звукові ефекти для своїх атак, це створює необхідну емоційну напругу та дозволяє гравцю реагувати не тільки на візуальні ефекти а і на звукові.

Звуки гри відіграються за допомогою класу SoundFXManager, який має лише одну функцію, яка створює динамічно створює джерело звуку класу AudioSource, та приймає звуковий кліп від класу що її викликав, а також позицію у світі де повинен програватися звуковий ефект, третім аргументом функції є гучність кліпу. Увесь код класу зображений на рисунку 4.12.

```
using UnityEngine;

Unity Script (1 asset reference) | 8 references
public class SoundFXManager : MonoBehaviour
{
    public static SoundFXManager Instance;
    [SerializeField] private AudioSource soundFXObject;
    Unity Message | 0 references
    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy(this);
        }
    }

    7 references
    public void PlaySoundFXClip(AudioClip audioClip, Transform spawnPosition, float volume)
    {
        AudioSource audioSource = Instantiate(soundFXObject, spawnPosition.position, Quaternion.identity);

        audioSource.clip = audioClip;
        audioSource.volume = volume;
        audioSource.Play();
        float clipLength = audioSource.clip.length;
        Destroy(audioSource.gameObject, clipLength);

    }
}
```

Рисунок 4.12 – Код класу SoundFXManager

4.4 Перевірка роботи скомпільованої гри

Після розробки усіх систем та підсистем, гра проходить процес компіляції для платформи WebGL, після цього згенеровані файли проєкту поміщаються в архів, який буде використаний для розгортання гри на платформі для публікації ігр: https://itch.io/.

Після налаштування сторінки та завантаження архіву з ігровими файлами, готова сторінка має вигляд зображений на рисунку 4.13.

Рисунок 4.13 – Сторінка гри на веб-ресурсі

Перевірено роботу гри, натиском на кнопку “Run game” сторінка змінює своє наповнення та користувач очікує завантаження проєкту. Після завантаження гра є доступною до взаємодії (рисунок 4.14), усі налаштовані системи працюють та гравець має можливість пройти гру та перемогти фінального боса (рисунок 4.15) з чого можна зробити висновок, що основні елементи прототипу працюють.

Рисунок 4.14 – Головне меню

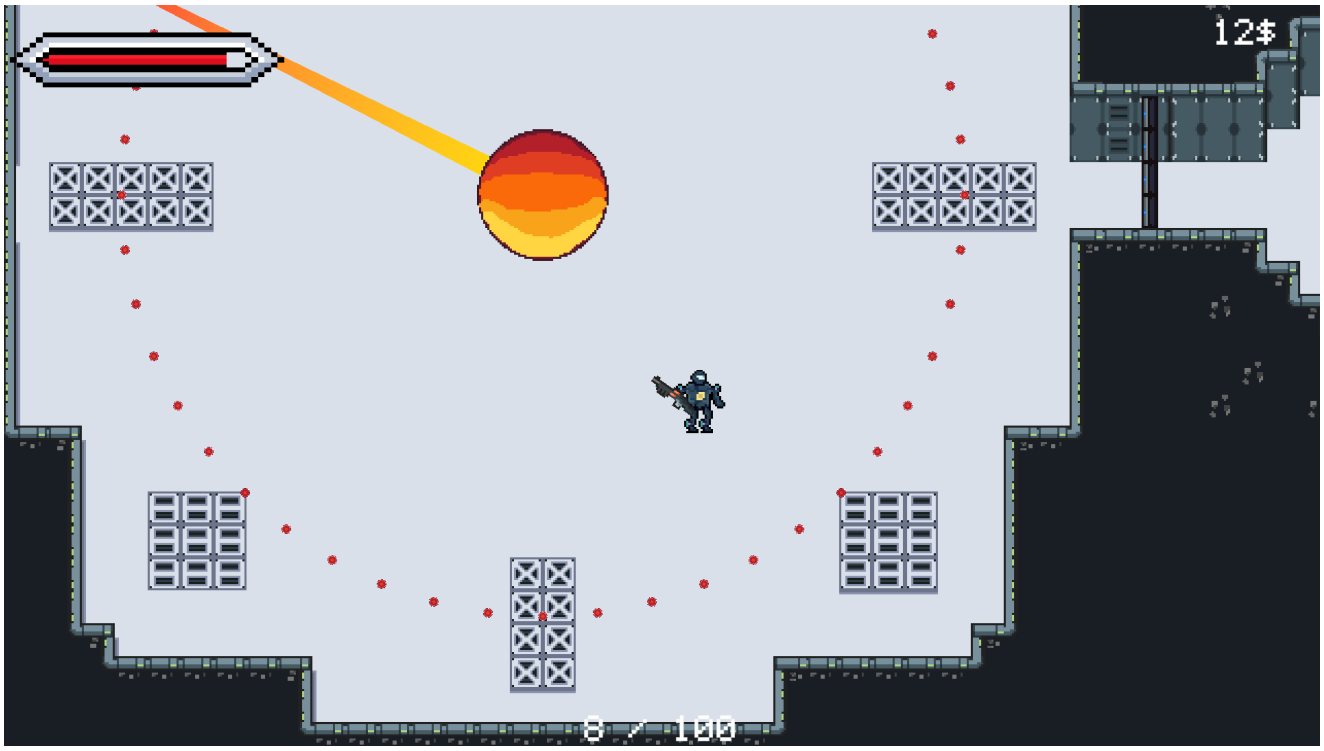

Рисунок 4.15 – Кімната фінального босса

ВИСНОВКИ

Під час виконання кваліфікаційної було проведено аналіз та надано характеристику відеоігровій індустрії, її особливостям та актуальності у контексті сьогодення. Основним об’єктом дослідження стали відеоігри жанру 2D-Roguelite, було вивчено основних представників, що існують на ринку.

Було досліджено питання структурування та ефективних архітектур для розробки 2D-проєктів, як в загальному так і для ігрового рушія Unity. Визначено основні напрямки на які необхідно звернути увагу при розробці, а саме: механізми процедурної генерації, дотримання ігрового балансу для підтримки зацікавленості гравця та одночасного надання йому виклику для подолання, вибір стилістичного оформлення та побудова цікаво ігрового циклу. На основі даних отриманих під час дослідження було побудовано прототип відеогри “Forged from Ruins”

Було розглянуто методи та сучасні практики побудови сучасних комп’ютерних мереж. Було змодельовано ефективну та захищену корпоративну мережу, яка задовільняє потреби сучасної відеоігрової студії. При налаштуванні мережевих пристроїв було використано актуальні технології та протоколи захисту, а також налаштовано ефективні методи маршрутизації між мережами.

Створена мережа не є ідеальною та має можливості для подальшої модернізації та доопрацювання. Розроблений прототип гри “Forged from Ruins” має потенціал для розвитку та розширення можливостей, а також доведення прототипу до стану завершеного продукту шляхом виправлення помилок у коді та додавання нових ігрових елементів.

Отже, за результатами кваліфікаційної роботи було розроблено ефективну та надійну комп’ютерну систему, з реалізацією корпоративної мережі розрахованої на використання відеоігровою студією з можливістю хмарного розміщення відеоігрових проєктів. Для хмарного розміщення було розроблено прототип 2D-Roguelite гри “Forged from Ruins” на основі ігрового рушія Unity для платформи WebGL.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Temple of the Roguelike. Core Traditional Roguelike Values [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – URL: https://blog.roguetemple.com/what-is-a-traditional-roguelike/ (дата звернення 29.04.2025)

2. Gamalytic. Статистика відеогри Don’t Starve Together [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – URL: https://gamalytic.com/game/322330 (дата звернення 29.04.2025)

3. Azzi M. Pixel Logic: Pixel Art Tutorials. [Електронний ресурс] / Michael Azzi. іл. Jenna Brown - [Б.м.]: Gumroad, 2022. - 242 с. – Режим доступу: https://michafrar.gumroad.com/l/pixel-logic (дата звернення: 29.04.2025).

4. Schreiber I. Game Balance / Ian Schreiber, Brenda Romero. - [Б.м.] : Taylor & Francis Group, 2021. - 766 с.

5. Unity. Офіційна сторінка магазину ігрових ресурсів [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – URL: https://assetstore.unity.com/publishing/community-building-resources (дата звернення 29.04.2025)

6. Roguelike Celebration. Bob Nystrom - Is There More to Game Architecture than ECS? [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – URL: https://www.youtube.com/watch?v=JxI3Eu5DPwE (дата звернення 29.04.2025)

7. Nystrom B. Game Programming Patterns [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – URL: https://gameprogrammingpatterns.com/design-patterns-revisited.html (дата звернення 29.04.2025)

8. Clean Code Summary: Agile Software Craftsmanship Guidelines. Developer Deconstructed. - [Б.м.] : Supergloo Inc. 2016. - 108 с.

9. Бранець І. Чому SOLID – важлива складова мислення програміста [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – URL: https://dou.ua/lenta/articles/solid-principles/ (дата звернення 29.04.2025)

10. Schell J. The Art of Game Design: A Book of lenses. / Jesse Schell. - [Б.м.] : Elsevier Inc. 2008. - 518 с.

11. Rouse III R. Game Design: Theory & Practice Second Edition. / Richard Rouse III. - Plano, Tex.: Wordware Publishing, Inc 2005. - 724 с.

12. Silber D. Pixel Art for Game Developers. / Daniel Silber. - [Б.м.]: A K Peters/CRC Press. 2016. - 252 с.

13. Мережеве обладнання Stack Systems [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – URL: https://stack-systems.com.ua/ (дата звернення 29.04.2025)

14. Мережеве обладнання ComTrade [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – URL: https://comtrade.ua/ (дата звернення 29.04.2025)

15. Vladimir Limarchenko. Slay The Spire map in Unity. Ліцензовано під MIT License [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – URL:https://github.com/silverua/slay-the-spire-map-in-unity (дата звернення 29.04.2025)

16. Sniffle6. Scriptable Object Based Inventory [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – URL:https://github.com/sniffle6/Scriptable-Object-Inventory (дата звернення 29.04.2025)

17. LlamAcademy. Scriptable Object Based Guns. Ліцензовано під MIT License [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – URL:https://github.com/llamacademy/scriptable-object-based-guns (дата звернення 29.04.2025)

18. Helton Yan. Retro Mecha SFX. Ліцензовано під CC BY 4.0 [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – URL:https://heltonyan.itch.io/retro-mecha-sfx (дата звернення 29.04.2025)

ДОДАТОК А

Технічні вимоги до Системи

А.1 Технічні вимоги до Системи

А.1.1 Вимоги до системи в цілому

А.1.1.1 Вимоги до структури і функціонування Системи

Повна назва Системи: Комп'ютерна система з хмарним розміщенням для інтеграції та підтримки відеоігри "Forged from Ruins". Система представляє собою корпоративну мережу відеоігрової студії і призначена для організації ефективного та надійного обміну даними між підрозділами компанії. Також одним із основних призначень Системи є підтримка хмарної інфраструктури для розміщення та підтримки відеоігри “Forged from Ruins”.

А.1.1.1.1. Перелік підсистем, їх призначення та основні характеристики, рівні ієрархії Системи.

Перелік підсистем складається з локальних мереж на які поділено підрозділи підприємства. Враховуючи організаційну структуру відеоігрової студії а також завдання кваліфікаційної роботи можна виділити такі підсистеми:

- підсистема віддаленого офісу програмістів (LAN_1);
- підсистема головного відділу програмістів (LAN_2);
- підсистема технічного відділу (LAN_3);
- підсистема дизайнерського відділу (LAN_4);
- підсистема відділу маркетингу та управління (LAN_5);

Підсистема віддаленого офісу повинна забезпечувати роботу 121 вузла. Основний офіс налічує 544 вузли, які розподілені наступним чином: 221 у відділі розробки, 44 у технічному відділі, 142 у відділі дизайну, 137 у відділі маркетингу та управління. У основному офісі також розташовані сервер DNS та сервер HTTPS / FTP.

Комплекс технічних засобів (КТЗ) Системи повинен складатися з таких ієрархічних рівнів:

1. Фізичний рівень – включає в себе комунікаційну інфраструктуру, кабелі, сервера, інше обладнання.

2. Мережевий рівень – включає в себе зв’язок між офісами, маршрутизацію, комутацію та захист шляхом налаштування VPN та сегментації на VLAN.

3. Прикладний рівень – включає в себе програмне забезпечення для роботи компанії, а також сервіс каталогів.

4. Адміністративний рівень – забезпечує централізоване адміністрування та моніторинг Системи.

А.1.1.1.2. Вимоги до способів і засобів зв'язку для інформаційного обміну між компонентами Системи

Способи і засоби зв’язку для інформаційного обміну між підсистемами повинні відповідати вимогам, визначеним КТЗ та іншим технічним вимогам, а також дозволяти інтеграцію широкої кількості пристроїв та обладнання для збереження масштабованості Системи.

А.1.1.1.3. Вимоги до характеристик взаємозв'язків створюваної Системи з суміжними системами

Система повинна бути відкрита до розширення та інтеграції, для цього повинно бути забезпечено підтримку стеку TCP/IP Системою. Система повинна бути захищена на усіх рівнях доступу, за допомогою налаштованої системи доступів та шифрування даних та трафіку. Система повинна бути продуктивною та надійною для взаємодії з зовнішніми хмарними платформами. На фізичному рівні зв’язок між елементами Системи повинен відповідати визначеним стандартам Ethernet та використовувати відповідні кабелі витої пари.

А.1.1.1.4. Вимоги до режимів функціонування Системи

Система повинна мати наступні режими функціонування:

1. Робочий – основний режим роботи, який забезпечує повний функціонал Системи, доступ до всіх підсистем та максимальну продуктивність 24/7.

2. Сервіс – режим планового технічного обслуговування, в якому відбувається вимкнення окремих компонентів Системи для їх обслуговування

3. Аварійний – режим в якому електроживлення Системи відбувається від безперебійного джерела живлення протягом обмеженого часу

А.1.1.1.5. Вимоги по діагностуванню Системи

Технічний відділ повинен забезпечити регулярне діагностування Системи за допомогою програмних засобів, з використанням віддаленого доступу до мережевих пристроїв та спеціалізованого програмного забезпечення. На фізичному рівні елементи Системи повинні проходити регулярний огляд та обслуговування згідно рекомендацій виробників обладнання.

А.1.1.1.6 Перспективи розвитку, модернізації Системи

Система має значний потенціал для можливого масштабування шляхом додавання нових пристроїв до існуючої мережі. Можливим є заміна мережевого обладнання Системи на більш сучасне або продуктивне. Система залишатиметься гнучкою до адаптації нових технологій та змін бізнес-моделі компанії.

А.1.1.2 Показники призначення

При нормальній роботі Системи:

− доступність основних систем та сервісів повинна бути >99.9% протягом усього року;

− середній час між відмовами мережевого обладнання(MTBF) > 100.000 годин;

− затримка сигналу в LAN мережі < 50 мс;

− затримка сигналу у WAN з’єднанні < 100 мс;

− пропускна здатність Інтернету: навантаження не повинно перевищувати 90%;

− втрата даних у разі інциденту не більше 15 хв;

− узгодженість даних між офісами >99.9%

А.1.1.3 Вимоги до експлуатації, технічного обслуговування, ремонту і збереження компонентів Системи

А.1.1.3.1 Умови і регламент експлуатації що повинні забезпечувати використання технічних засобів (ТЗ) системи з заданими технічними показниками

Умови експлуатації технічних засобів Системи формуються з вимог та задач Системи, а у відповідності до рекомендацій виробника апаратного забезпечення.

Кліматичні умови використання повинні знаходитися у межах 0-25°С та вологості 40-60% без конденсації вологи. Приміщення в яких знаходяться мережеве та серверне обладнання повинні бути обладнані вентиляцією або системами кондиціювання повітря. Доступ до мережевого та серверного обладнання повинен бути обмежений тільки для авторизованого персоналу.

Сервери та мережеве обладнання повинні проходити профілактичне обслуговування та тестування 1 раз/пів року, очищення обладнання від накопиченого пилу повинно проводитися 1 раз/2 місяці. Резервне джерело живлення повинно проходити тестування 1 раз/пів року та проходити оновлення батарей у визначений компанією виробником період.

А.1.1.3.2 Вимоги до параметрів мереж енергопостачання

Основне джерело живлення Системи повинно надавати струм з параметрами 220-230В та частотою 50 Гц. Обов’язковим є використання безперебійного джерела живлення для кожного з локальних відділів із забезпеченням автономної роботи критичного обладнання протягом 15 хвилин.

А.1.1.3.3 Вимоги до кількості, кваліфікації обслуговуючого персоналу і режимам його роботи

Основний технічний персонал, що складається зі спеціально навчених спеціалістів повинен щороку проводити інструктажі з безпеки, експлуатації та базового обслуговування ТЗ. Обслуговуючий персонал назначається зі складу відділу технічного обслуговування компанії.

А.1.1.3.4 Вимоги до складу, розміщенню й умовам збереження комплекту запасних виробів і приладів

До комплекту складу запасних виробів і приладів повинні входити усі необхідні для функціонування та обслуговування елементи для мережевого обладнання компанії, а також інших пристроїв. У складському приміщенні повинні обов’язково зберігатися: маршрутизатори, комутатори, запасні мережеві кабелі, запасні кабелі живлення, запасні ПК у зборі.

Зберігання здійснюється в окремому технічному приміщенні або шафі з обмеженим доступом. Запасні вироби повинні бути захищені від впливу сонячних променів, пилу, вібрацій.

А.1.1.3.5 Вимоги до регламенту обслуговування

У інструкціях з експлуатації необхідно визначити періодичність та регламент для проведення регулярного та позапланового обслуговування. Для визначення цих параметрів необхідно враховувати рекомендаційні матеріали з обслуговування апаратного забезпечення, що надаються виробником. Також відділ технічного забезпечення повинен забезпечити регулярне оновлення програмного забезпечення Системи.

А.1.1.4 Вимоги до патентної чистоти

Встановлення Системи та установка окремих її елементів не потребує покупки додаткових ліцензій, проте розробка відеогри потребує приписання авторства деяких елементів продукту їхнім власникам у відповідності до визначених ліцензійних документів.

А.1.1.5 Додаткові вимоги

А.1.1.5.1 Вимоги до Системи, пов'язані з особливими умовами її експлуатації

Система не передбачає особливих умов з експлуатації.

А.1.1.5.1 Вимоги до активного обладнання (функціонування, кількість портів та їх запас, варіанти встановлення, технічні вимоги)

Активне мережеве обладнання повинне забезпечувати безперервну роботу

24/7 без врахування часу на обслуговування. Комутатори та маршрутизатори повинні бути встановлені у стандартизованій шафі або стійці 19``. Комутатори до яких напряму підключається вузли Системи повинні мати 24 порти FastEthernet та мати запас портів у 20% (5 шт.) від поточної кількості. Центральні комутатори, основна функція яких передавати трафік до інших комутаторів, повинні мати 24 порти GigabitEthernet та підтримку PoE. Маршрутизатори повинні відповідати функціоналу визначеному у цих технічних вимогах та підтримувати усі визначені технології та протоколи.

А.1.1.5.2 Вимоги до кабель-каналів, інформаційним та електричним розеткам (тип, розмір, варіант розміщення)

Кабель-канали повинні бути виконані з пластику та мати клас захисту IP30, в залежності від кількості кабелів у каналі, повинен бути підібраний відповідний його розмір. Для прокладання каналів використовуються спеціалізовані плінтуси на підлозі, або кріплення під стелею. Інформаційні розетки повинні мати порт RJ-45 з підтримкою кабелів Cat 5. Електричні розетки необхідно належним чином заземлити та розташувати 4 розетки для кожного робочого місця.

А.1.1.5.3 Вимоги до комунікаційного обладнання і його розташування

У кожній підсистемі повинно бути розташована достатня кількість мережевого обладнання для підключення усіх вузлів, з врахуванням резервування портів. Комунікаційне обладнання повинно бути розміщеним у спеціалізованій шафі або стійці із обмеженим доступом сторонніх осіб до безпосереднього обладнання та кабелів що підключені до нього.

А.1.1.5.4 Вимоги до однорідності

Усі структурні підрозділи повинні бути обладнані ідентичним або сумісним мережевим обладнанням єдиного виробника мережевого обладнання. Кабельна інфраструктура повинна бути побудована на основі кабелів Cat 5 для підключення кінцевих вузлів, зв’язок між комутаторами та маршрутизаторами повинен використовувати кабелі Cat 5E, маршрутизатори поєднуються за допомогою оптичних або мідних кабелів стандарту GigabitEthernet.

А.1.1.5.6 Вимоги до резервування

Дані на серверах компанії а також важливі дані високопосадовців, працівників фінансового відділу, відділу кадрів повинні проходити регулярне резервування на резервні носії даних, що знаходяться під захистом у відділі технічного обслуговування.

А.1.2 Вимоги до функцій, виконуваних Системою

А.1.2.1. Перелік функцій, завдань або їх комплексів (в тому числі тим, що забезпечують взаємодію частин системи), що підлягають автоматизації

Система складається із п’яти підсистем LAN1-LAN5. Кількість вузлів для кожної з підсистем визначається пунктом 2.1.1.1.1 цих вимог. Мережі підсистем знаходяться мають блок адрес 172.24.232.0/21, даний блок необхідно розділити у відповідності до кількості вузлів у кожній з підсистем з врахуванням резервування та можливого масштабування у майбутньому.

Для маршрутизації у мережі основного офісу було виділено блок адрес 10.1.3.0/24.

Призначення адрес у Системі повинно відповідати наступним рекомендаціями:

− перші можливі для використання IP-адреси надаються інтерфейсам маршрутизаторів;

− другі можливі IP-адреси надаються комутаторам для можливості віддаленого доступу;

− у випадку застосування VLAN у підмережі, комутатори повинні знаходитися у власній VLAN та отримувати адреси з власного визначено блок адрес;

− сервери отримують адреси, які знаходяться як найближче до межі блоку адрес, при цьому є легкими для запам’ятання, наприклад 100 або 150;

− усі інші IP-адреси призначаються іншим вузлам за протоколом DHCP;

Базове налаштування пристроїв повинно включати:

− маршрутизатори мережі отримують назви за правилом: Afanasiev_підмережа до якої підключено пристрій_тип пристрою;

− на усіх пристроях необхідно встановити паролі до консолі vty та привілейованого режиму та зашифрувати, пароль для усіх режимів та пристроїв: afanasiev;

Налаштування маршрутизаторів повинні включати:

− базове налаштування інтерфейсів зв'язку;

− налаштований протокол маршрутизації OSPF з номером процесу 1;

− маршрутизатори що знаходяться у основному офісі повинні мати зону OSPF з номером 20, віддалений офіс знаходиться у зоні 10, інтерфейси що пов'язані з ISP повинні знаходитися у зоні 0.

− на крайньому маршрутизаторі у мережі основного офісу повинен бути налаштований статичний NAT для серверів компанії, та PAT для інших вузлів.

− між крайнім маршрутизатором у мережі основного офісу та LAN1 повинен бути налаштований IPsec тунель.

− на усіх маршрутизаторах повинна бути налаштована роздача IPv4 адрес для вузлів у своїх LAN мережах за протоколом DHCP. У мережах з VLAN, повинні бути налаштовані додаткові віртуальні інтерфейси та роздача IP-адрес відповідно до VLAN.

Налаштування комутаторів:

− у мережах LAN2 та LAN5 необхідно налаштувати VLAN для різних підрозділів.

− першим чотирьом портам не призначаються VLAN, вони назначаються динамічно при потребі підключення тих чи інших вузлів до певних VLAN. Чотири останні порти знаходяться у резерві.

− у мережі LAN1 необхідно налаштувати EtherChannel між комутаторами з дублюючими підключеннями.

А.1.2.2 Форми представлення вихідної інформації, вимоги одночасності виконання групи функцій

Усі підсистеми повинні виконуватися синхронно та одночасно для оптимального та ефективного функціонування Системи. Критичні функції не повинні блокувати діяльність одна одної та у разі потреби працювати у асинхронному режимі.

А.1.3 Вимоги до видів забезпечення

А.1.3.1 Математичне забезпечення

Математичне забезпечення Системи повинно включати алгоритми, методи і моделі, які є необхідними для коректного та оптимального функціонування її обчислювальних компонентів. Забезпечення повинне включати алгоритми маршрутизації та балансування навантаження для мережевих елементів. Повинні бути забезпечені механізми шифрування та хешування для захисту даних.

А.1.3.2 Інформаційне забезпечення

Інформаційне забезпечення Системи повинне забезпечувати стабільний обмін інформацією між компонентами та підсистемами. Реалізовані рішення повинні забезпечити можливість інтеграції Системи у глобальну мережу з можливістю подальшої модернізації та масштабування. Зберігання даних у мережі повинно бути організоване за допомогою центрального репозиторію, де будуть зберігатися усі дані що відносяться до проектів компанії. Також фінансові та інші дані повинні бути організовані та систематизовані за допомогою використання реляційних або нереляційних баз даних з регулярним резервуванням даних у відповідності п. А.1.1.5.6 цих вимог.

А.1.3.3 Лінгвістичне забезпечення

Все лінгвістичне забезпечення системи для організації взаємодії з користувачем повинне використовувати українську та англійську мови. Працівники компанії повинні використовувати ОС Windows 11 при роботі. Для серверної ОС може бути використаний Windows Server або один із серверних

дистрибутивів ОС Linux, наприклад Debian або Ubuntu Server.

Мережеве обладнання використовує програмне забезпечення надане або рекомендоване виробником у відповідності до моделі обладнання.

При роботі відділу розробки повинні використовуватися мови програмування C#/C++, інтегроване середовище розробки Visual Studio або Visual Studio Code, а також ігровий рушій Unity 6.0.

А.1.3.4 Технічне забезпечення

Для побудови Системи повинно бути використане мережеве обладнання компанії Cisco, яке задовольняє технічні вимоги. Обладнання повинне бути розміщене у відповідних умовах із забезпечення безперебійного живлення. Обладнання відділів маркетингу та управління, а також технічного відділу повинні знаходитися на рівні стандартного офісного ПК. Відділи дизайну та розробки повинні бути забезпечені робочими станціями підвищеної потужності із характеристиками не нижче: 4-ядерний процесор 4.5 Ггц, 16 ГБ оперативної пам’яті та дискретний графічний процесор на 4 ГБ відеопам’яті.

Серверне обладнання повинно забезпечувати достатній обсяг пам’яті для зберігання даних компанії, а також мати потужний процесор для обробки запитів. Підключення до мережі повинно здійснюватися за стандартом GigabitEthernet.

А.1.3.6 Організаційне забезпечення

За організацію функціонування Системи у компанії, відповідає визначений підрозділ технічного обслуговування, а саме системні адміністратори. За налаштування та роботу серверного обладнання у технічному відділі відповідають працівники DevOps. В кожному відділі компанії повинні бути назначені відповідальні за обладнання працівники з інших підрозділів, які пройдуть відповідне навчання для проведення базового обслуговування та підтримки роботи критичного обладнання. Для мінімізації ризику помилкових дій персоналу передбачено багаторівневий доступ до елементів Системи, підтвердження дій на критичних етапах, а також регулярне навчання користувачів та відповідальних осіб.

А.1.3.7 Методичне забезпечення

Перелік основних стандартів, що використовуються при проектуванні та побудові Системи:

− ДСТУ ISO/IEC 27001:2015;

− ДСТУ EN 50174 “Вимоги до проектування кабельних систем”

− ГОСТ 34.003-90.

− ГОСТ 34.201-89.

− ГОСТ 34.601-90.

− ГОСТ 34.602-89.

Також Системи повинна відповідати вимогам, що визначаються у внутрішніх документах компанії таких як наприклад: політика інформаційної безпеки, журнал конфігураційних файлів, журнал внесених змін до Системи. Рекомендовано також прийняти до уваги при побудові Системи міжнародні стандарти, що не є офіційно представленими в Україні.

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВІДЕОГРИ “FORGED FROM RUINS”

Текст програми

804.02070743.25003-01 12 01

Листів 30

2025

АНОТАЦІЯ

Дана програма містить в собі частину коду розробленої відеогри “Forged from Ruins” розробленої для хмарного розміщення у комп’ютерній системі підприємства.

Гра є представником жанру 2D-Roguelite з видом згори і реалізує такі системи як: процедурна генерація рівнів, управління загальним станом гри, управління графічним інтерфейсом користувача, керування діями гравця та інші системи що не включені в даний додаток.

Програма написана мовою програмування C# у середовищі розробки Visual Studio, для ігрового рушія Unity 6.0. Даний код може бути скомпільований для будь-якої платформи, що підтримується рушієм Unity.

ЗМІСТ

1 Файл GameManager.cs

```
using Map;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set; }
    public bool isPaused = false;
    public GameState State = GameState.MainMenu;
    private GameState OldState;
    public static event Action<GameState> OnGameStateChanged;

    private LevelGenerator _levelManager;
    private MapManager _mapManager;
    private GameObject activePlayerInstance;

    private List<Room> _allRooms = new List<Room>();
    private int _roomsRemaining;

    [Header("Inventory References")]
    [SerializeField] private InventorySO playerInventory;
    [SerializeField] private InventorySO rewardInventory;

    [Header("PlayerPrefab")]
    [SerializeField] private GameObject _player;

    [Header("Loot Settings")]
    [SerializeField] private float playerLuckModifier = 1f;
    [SerializeField] private LootGenerator _lootGenerator;

    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);

            _lootGenerator = GetComponent<LootGenerator>();
            if (_lootGenerator == null)
            {
                Debug.LogError("LootGenerator component not found on GameManager!");
            }

            CreatePersistentPlayer();
            PlayerController.OnPlayerDeath += OnLose;
            LevelGenerator.OnGenerationComplete += EnablePlayer;
            Enemy.OnBossDeath += OnWin;
        }
        else
        {
            Destroy(gameObject);
        }
```

```
}

private void CreatePersistentPlayer()
{
    // Create player as child of GameManager so it persists between scenes
    activePlayerInstance = Instantiate(_player, transform);
    activePlayerInstance.name = "PersistentPlayer";

    // Start with player disabled
    activePlayerInstance.SetActive(false);
}

public void UpdateGameState(GameState newState)
{
    OldState = State;
    State = newState;
    switch (newState)
    {
        case GameState.MainMenu:
            ResumeTimeFlow();
            DisablePlayer();
            break;
        case GameState.MapSelect:
            ResumeTimeFlow();
            _mapManager = GameObject.FindAnyObjectByType<MapManager>();
            DisablePlayer();
            break;
        case GameState.FightLevel:
            _levelManager = GameObject.FindAnyObjectByType<LevelGenerator>();
            ResumeTimeFlow();
            break;
        case GameState.RewardScreen:
             ResumeTimeFlow();
            break;
        case GameState.Shop:
            ResumeTimeFlow();
            break;
        case GameState.BossLevel:
            _levelManager = GameObject.FindAnyObjectByType<LevelGenerator>();
            ResumeTimeFlow();
            break;
        case GameState.WinScreen:
            break;
        case GameState.LoseScreen:
            break;
        case GameState.PauseScreen:
            break;
    }
    Debug.Log($"We are in {newState}");
    OnGameStateChanged?.Invoke(newState);
}

public void PauseGame()
{
    Time.timeScale = 0f;
    UpdateGameState(GameState.PauseScreen);
    isPaused = true;
}
```

```
public void UnPause()
    {
        Time.timeScale = 1f;
        UpdateGameState(OldState);
        isPaused = false;
    }

    public void ResumeTimeFlow()
    {
        Time.timeScale = 1f;
        isPaused = false;
    }

    public void GoToMainMenu()
    {
        UpdateGameState(GameState.MainMenu);
        SceneManager.LoadScene(0, LoadSceneMode.Single);
    }

    public void StartRun()
    {
        SceneManager.sceneLoaded += OnMapLoaded;

        if (activePlayerInstance == null)
        {
            Debug.LogWarning("Player instance was null, creating it now");
            CreatePersistentPlayer();
        }

        if (_lootGenerator == null)
        {
            _lootGenerator = GetComponent<LootGenerator>();
            Debug.LogWarning("LootGenerator was null, attempting to re-find component");
        }

        if (_lootGenerator != null)
        {
            Debug.Log($"LootGenerator null check: {_lootGenerator == null}");
            _lootGenerator.ResetLootbase();
            _lootGenerator.SpawnStartingEquipment();
        }
        else
        {
            Debug.LogError("LootGenerator is still null after attempting to find it!");
        }
        var controller = activePlayerInstance.GetComponent<PlayerController>();
        controller.ResetInventories();
        SceneManager.LoadScene(1, LoadSceneMode.Single);
    }

    private void OnMapLoaded(Scene scene, LoadSceneMode mode)
    {
        SceneManager.sceneLoaded -= OnMapLoaded;
        UpdateGameState(GameState.MapSelect);
        _mapManager.GenerateNewMap();
    }

    public void LoadMapSelect()
    {
        UpdateGameState(GameState.MapSelect);
```

```
    _allRooms.Clear();
    SceneManager.LoadSceneAsync(1, LoadSceneMode.Single);
}

public void LoadGameLevel()
{
    SceneManager.sceneLoaded += OnFightLoaded;
    SceneManager.LoadScene(2, LoadSceneMode.Single);
}
public void LoadBossLevel()
{
    SceneManager.sceneLoaded += OnBosstLoaded;
    SceneManager.LoadScene(2, LoadSceneMode.Single);
}

public void LoadRewardLevel()
{
    SceneManager.sceneLoaded += OnRewardLoaded;
    SceneManager.LoadScene(2, LoadSceneMode.Single);
}

public void OnHealNode()
{
    var controller = activePlayerInstance.GetComponent<PlayerController>();
    controller._playerStats.IncreaseHealth(50);
}

public void OnAmmoRefill()
{
    var gunSelector = activePlayerInstance.GetComponent<GunSelector>();
    gunSelector.ReloadActiveWeapons();
}
private void OnRewardLoaded(Scene scene, LoadSceneMode mode)
{
    SceneManager.sceneLoaded -= OnRewardLoaded;
    UpdateGameState(GameState.RewardScreen);
    StartCoroutine(DelayedClear());
}

private IEnumerator DelayedClear()
{
    yield return new WaitForEndOfFrame();
    _lootGenerator.ClearRewardInventory();
    _lootGenerator.GenerateLoot(2);
    EnablePlayer(this.transform);
}

private void OnFightLoaded(Scene scene, LoadSceneMode mode)
{
    SceneManager.sceneLoaded -= OnFightLoaded;
    UpdateGameState(GameState.FightLevel);
    _levelManager.maxRooms = UnityEngine.Random.Range(2, 6);
    _levelManager.SetDifficulty(GenerationSettings.Regular);
    LevelGenerator.OnGenerationComplete += LevelCompletionTracker;
    _levelManager.GenerateLevel();
}

private void OnBosstLoaded(Scene scene, LoadSceneMode mode)
{
    SceneManager.sceneLoaded -= OnBosstLoaded;
```

```
        UpdateGameState(GameState.BossLevel);
        _levelManager.maxRooms = 2;
        _levelManager.SetDifficulty(GenerationSettings.Boss);
        _levelManager.GenerateLevel();
    }

    public void LoadShopLevel()
    {
        UpdateGameState(GameState.Shop);
        SceneManager.LoadScene(2, LoadSceneMode.Single);
    }

    public void LevelCompletionTracker(Transform playerPos)
    {
        _allRooms.Clear();
        Room[] roomsInScene = FindObjectsByType<Room>(FindObjectsSortMode.InstanceID);
        foreach (Room room in roomsInScene)
        {
            if (room == null) continue;
            if (room.playerStartPosition) continue;

            room.OnRoomCleared -= HandleRoomClear;
            room.OnRoomCleared += HandleRoomClear;

            _allRooms.Add(room);
        }
        _roomsRemaining = _allRooms.Count;
        Debug.Log($"[RoomClearWatcher] Found {_roomsRemaining} rooms in this level.");


    }

    public void HandleRoomClear(Room room)
    {
        room.OnRoomCleared -= HandleRoomClear;

        _roomsRemaining--;
                Debug.Log($"[RoomClearWatcher] Room cleared: {room.name}. Rooms left:
{_roomsRemaining}");
        if (_roomsRemaining <= 0)
        {
            LoadMapSelect();
        }
    }

    private void OnWin(Enemy boss)
    {
        Time.timeScale = 0f;
        isPaused = true;
        UpdateGameState(GameState.WinScreen);
    }

    private void OnLose()
    {
        Time.timeScale = 0f;
        isPaused = true;
        UpdateGameState(GameState.LoseScreen);
    }
```

```
    private void EnablePlayer(Transform position)
    {
        if (activePlayerInstance != null)
        {
            // Move player to the spawn position
            activePlayerInstance.transform.position = position.position;
            activePlayerInstance.transform.rotation = position.rotation;

            // Enable the player
            activePlayerInstance.SetActive(true);

            Debug.Log("Player positioned and enabled at: " + position.position);
        }
        else
        {
            Debug.LogError("Persistent player instance is null!");
        }
    }

    private void DisablePlayer()
    {
        if (activePlayerInstance != null)
        {
            activePlayerInstance.SetActive(false);
        }
    }

    public GameObject GetPlayer()
    {
        return activePlayerInstance;
    }

    public void ExitGame()
    {
        Application.Quit();
    }

    private void OnDestroy()
    {
        PlayerController.OnPlayerDeath -= OnLose;
    }

}

public enum GameState
{
    MainMenu,
    MapSelect,
    FightLevel,
    RewardScreen,
    Shop,
    BossLevel,
    WinScreen,
    LoseScreen,
    PauseScreen
}
```

2 Файл LevelGenerator.cs

```
using NavMeshPlus.Components;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class LevelGenerator : MonoBehaviour
{
    [SerializeField] NavMeshSurface Surface2D;
    [Header("Room Settings")]
    public Room startRoom;
    [SerializeField] private List<Room> roomPrefabs;
    [SerializeField] private List<Room> hardPrefabs;
    [SerializeField] private List<Room> bossPrefabs;
    [SerializeField] public int maxRooms = 10;
           public GenerationSettings GeneratorMode { get; private set; } =
GenerationSettings.Regular;
    [Header("Collision Settings")]
    [SerializeField] private LayerMask roomLayerMask = 1 << 15;


    private List<Room> placedRooms = new List<Room>();
        private List<(Room room, ExitTilemap exit)> openExits = new List<(Room,
ExitTilemap)>();

    private const int MAX_EXIT_TRIES = 3;
       private Dictionary<(Room, ExitTilemap), int> exitTries = new Dictionary<(Room,
ExitTilemap), int>();

    public static Action<Transform> OnGenerationComplete;

    public void GenerateLevel()
    {
        StartCoroutine(GenerateLevelCoroutine());
    }

    IEnumerator GenerateLevelCoroutine()
    {
        #if UNITY_EDITOR
        for (int i = transform.childCount - 1; i >= 0; i--)
            DestroyImmediate(transform.GetChild(i).gameObject);
        #else
        for (int i = transform.childCount - 1; i >= 0; i--)
            Destroy(transform.GetChild(i).gameObject);
        #endif

        placedRooms.Clear();
        openExits.Clear();

        // Seed first room at world origin
        Room first = InstantiateRoom(startRoom, Vector3.zero);
        placedRooms.Add(first);
        Transform playerStart = first.playerStartPosition;
        // Collect its exits
```

```
foreach (var exit in first.GetComponentsInChildren<ExitTilemap>())
    openExits.Add((first, exit));

// Expand until maxRooms or no exits left
while (placedRooms.Count < maxRooms && openExits.Count > 0)
{
    // Inside the while loop, add:
    var (parentRoom, parentExit) = PopRandomExit();


    var key = (parentRoom, parentExit);
    exitTries.TryGetValue(key, out int tries);
    if (++tries > MAX_EXIT_TRIES)
    {
        // give up on this exit forever
        RemoveExit(parentRoom, parentExit);
        exitTries.Remove(key);
        continue;
    }
    exitTries[key] = tries;


    ExitDirection neededDir = Opposite(parentExit._direction);

    // Find a prefab with matching exit
    Room prefab = FindPrefabWithExit(neededDir);
    if (prefab == null)
    {
        continue;
    }

    // Locate the matching exit on a temp instance to get correct world offset
    Room temp = Instantiate(prefab, Vector3.zero, Quaternion.identity);

    yield return null;

    ExitTilemap tempExit = temp.GetComponentsInChildren<ExitTilemap>()
                               .FirstOrDefault(e => e._direction == neededDir);

    if (tempExit == null)
    {
        DestroyImmediate(temp.gameObject);
        continue;
    }
    Vector3 childMarkerWorld = tempExit.connectionPoint.position;
    Vector3 parentPoint = parentExit.connectionPoint.position;
    Vector3 spawnPos = parentPoint - childMarkerWorld;
    DestroyImmediate(temp.gameObject);

    // Instantiate disabled for overlap test
    Room newRoom = Instantiate(prefab, spawnPos, Quaternion.identity);
    newRoom.transform.SetParent(transform, false);

    Physics2D.SyncTransforms();
    // Overlap check using actual colliders
    if (CheckOverlap(newRoom))
    {
        DestroyImmediate(newRoom.gameObject);
        continue;
    }
```

```
            // Accept the room and finalize
            newRoom.gameObject.SetActive(true);
            placedRooms.Add(newRoom);

            // Disable connecting covers
            parentRoom.SetCoverActive(parentExit._direction, false);
            newRoom.SetCoverActive(neededDir, false);

            // Remove the used exit from further consideration
            RemoveExit(parentRoom, parentExit);

            // Enqueue all other exits of the new room
            foreach (var exit in newRoom.GetComponentsInChildren<ExitTilemap>())
            {
                if (exit._direction == neededDir) continue;
                openExits.Add((newRoom, exit));
            }

        }
        StartCoroutine(BuildNavMesh());
        OnGenerationComplete?.Invoke(playerStart);
    }

    IEnumerator BuildNavMesh()
    {
        yield return null;
        Physics2D.SyncTransforms();
        Surface2D.BuildNavMesh();
    }

    private Room InstantiateRoom(Room prefab, Vector3 position)
    {
        return Instantiate(prefab, position, Quaternion.identity, transform);
    }

    private (Room, ExitTilemap) PopRandomExit()
    {
        int idx = UnityEngine.Random.Range(0, openExits.Count);
        var entry = openExits[idx];
        openExits.RemoveAt(idx);
        return entry;
    }

    private void RemoveExit(Room room, ExitTilemap exit)
    {
        openExits.RemoveAll(x => x.room == room && x.exit == exit);
    }

    private Room FindPrefabWithExit(ExitDirection dir)
    {
        List<Room> matches = null;
        switch (this.GeneratorMode)
        {
            case GenerationSettings.Regular:
                matches = roomPrefabs
                    .Where(r => r.GetComponentsInChildren<ExitTilemap>()
                    .Any(e => e._direction == dir))
                    .ToList();
                break;
```

```
            case GenerationSettings.Hard:
                matches = hardPrefabs
                    .Where(r => r.GetComponentsInChildren<ExitTilemap>()
                    .Any(e => e._direction == dir))
                    .ToList();
                break;
            case GenerationSettings.Boss:
                matches = bossPrefabs
                    .Where(r => r.GetComponentsInChildren<ExitTilemap>()
                    .Any(e => e._direction == dir))
                    .ToList();
                break;
            default:
                matches = new List<Room>();
                break;
        }

        /*var matches = roomPrefabs
                    .Where(r => r.GetComponentsInChildren<ExitTilemap>()
                    .Any(e => e._direction == dir))
                    .ToList();*/


        // If none found, bail out
        if (matches.Count == 0)
            return null;

        // Pick one at random (uniform probability)
        int idx = UnityEngine.Random.Range(0, matches.Count);
        return matches[idx];
    }

    private ExitDirection Opposite(ExitDirection dir)
    {
        return dir switch
        {
            ExitDirection.North => ExitDirection.South,
            ExitDirection.South => ExitDirection.North,
            ExitDirection.East => ExitDirection.West,
            ExitDirection.West => ExitDirection.East,
            _ => dir,
        };
    }
    private bool CheckOverlap(Room room)
    {

        // Get only room structure colliders
        var roomCols = room.GetComponentsInChildren<Collider2D>()
            .Where(col => ((1 << col.gameObject.layer) & roomLayerMask) != 0)
            .ToArray();

        if (roomCols.Length == 0)
        {
            return false;
        }

        // Calculate bounds only from room structure
        Bounds bounds = roomCols[0].bounds;
        for (int i = 1; i < roomCols.Length; i++)
            bounds.Encapsulate(roomCols[i].bounds);
```

```
        Vector2 checkSize = new Vector2(bounds.size.x - 0.2f, bounds.size.y - 0.2f);

        // Perform overlap check
        Collider2D[] hits = Physics2D.OverlapBoxAll(
            bounds.center,
            checkSize,
            0f,
            roomLayerMask
        );

        // Count overlaps from different rooms
        int overlapCount = 0;
        foreach (var hit in hits)
        {
            Room hitRoom = hit.GetComponentInParent<Room>();

            if (hitRoom != null && hitRoom != room)
            {
                overlapCount++;
            }
        }

        bool hasOverlap = overlapCount > 0;
        return hasOverlap;
    }

    public void SetDifficulty(GenerationSettings setting)
    {
        GeneratorMode = setting;
    }

}

public enum GenerationSettings
{
    Regular,
    Hard,
    Boss
}
```

3 Файл PlayerController.cs

```
using System;
using System.Collections;
using Unity.VisualScripting;
using UnityEngine;

public class PlayerController : MonoBehaviour, IDamageable
{
    [SerializeField]
    PlayerMovement _movementController;
    [SerializeField]
    PlayerActions _playerActions;
    [SerializeField]
    Rigidbody2D _rigidbody;
    [SerializeField]
    public PlayerAttributesSO _playerStats;
    [SerializeField]
    InventorySO _inventory;
    [SerializeField]
    InventorySO _equipment;
    [SerializeField]
    AudioClip DamageClip;

    public bool isPlayerInvulnerable = false;

    public static Action OnPlayerDeath;

    public int MaxHealth {  get; set; }
    public int CurrentHealth { get; set; }

    private void Start()
    {
        for (int i = 0; i < _playerStats.Attributes.Length; i++)
        {
            _playerStats.Attributes[i].SetParent(this);
        }
    }

    public void OnBeforeSlotUpdate(InventorySlot slot)
    {
        Debug.Log($"=== BEFORE UPDATE ===");
        Debug.Log($"Slot Hash: {slot.GetHashCode()}");
        Debug.Log($"ItemObject: {(slot.ItemObject != null ? "Present" : "NULL")}");
        Debug.Log($"Item: {(slot.Item != null ? slot.Item.Id.ToString() : "NULL")}");
        Debug.Log($"Amount: {slot.Amount}");
        Debug.Log($"Inventory Type: {slot.Parent._inventory.InventoryType}");
        if (slot.ItemObject == null) return;
        switch (slot.Parent._inventory.InventoryType)
        {
            case InterfaceType.Inventory:
              if (_inventory.Database.ItemObjects[slot.Item.Id].Type == ItemType.Passive)
                {
                    Debug.Log($"Processing passive item removal: {slot.Item.Id}");
                    for (int i = 0; i < slot.Amount; i++)
                    {
                        foreach (var stat in slot.Item.Stats)
```

```
                    {
                        _playerStats.RemoveModifier(stat);
          Debug.Log($"  Removed modifier: {stat} (iteration {i + 1}/{slot.Amount})");
                    }
                }
    Debug.Log($"Removing {slot.Amount} stacks of modifiers for item {slot.Item.Id}");
            }
            break;
        case InterfaceType.Equipment:
            foreach (var stat in slot.Item.Stats)
                _playerStats.RemoveModifier(stat);
            break;
        case InterfaceType.Reward:
            break;
        default:
            break;
    }
    Debug.Log($"=== END BEFORE UPDATE ===\n");
}
public void OnAfterSlotUpdate(InventorySlot slot)
{
    Debug.Log($"=== AFTER UPDATE ===");
    Debug.Log($"Slot Hash: {slot.GetHashCode()}");
    Debug.Log($"ItemObject: {(slot.ItemObject != null ? "Present" : "NULL")}");
    Debug.Log($"Item: {(slot.Item != null ? slot.Item.Id.ToString() : "NULL")}");
    Debug.Log($"Amount: {slot.Amount}");
    Debug.Log($"Inventory Type: {slot.Parent._inventory.InventoryType}");
    if (slot.ItemObject == null) return;
    StartCoroutine(DelayedAfterUpdate(slot));

}

private IEnumerator DelayedAfterUpdate(InventorySlot slot)
{
    yield return null;
    switch (slot.Parent._inventory.InventoryType)
    {
        case InterfaceType.Inventory:
          if (_inventory.Database.ItemObjects[slot.Item.Id].Type == ItemType.Passive)
            {
             Debug.Log($"Processing passive item application: {slot.Item.Id}");
               for (int i = 0; i < slot.Amount; i++)
               {
                    foreach (var stat in slot.Item.Stats)
                    {
                        _playerStats.ApplyModifier(stat);
                        Debug.Log($"  Applied modifier:
                        {stat} (iteration {i + 1}/{slot.Amount})");
                    }
                }
            }
            else
            {
              Debug.Log("Not a passive item or item is null - skipping application");
            }
            break;
        case InterfaceType.Equipment:
            foreach (var stat in slot.Item.Stats)
                _playerStats.ApplyModifier(stat);
            break;
```

```
        case InterfaceType.Reward:
            break;
        default:
            break;
    }
    Debug.Log($"=== END AFTER UPDATE ===\n");

}

private void OnEnable()
{
    _playerStats.Health = _playerStats.GetStatValue(TypeOfStat.MaxHealth);
    MaxHealth = _playerStats.GetStatValue(TypeOfStat.MaxHealth);
    CurrentHealth = _playerStats.Health;

    _playerStats.healthChangeEvent += HandleDamage;
    for (int i = 0; i < _equipment.GetSlots.Length; i++)
    {
        _equipment.GetSlots[i].OnBeforeUpdate += OnBeforeSlotUpdate;
        _equipment.GetSlots[i].OnAfterUpdate += OnAfterSlotUpdate;
    }
    for (int i = 0; i < _inventory.GetSlots.Length; i++)
    {
        _inventory.GetSlots[i].OnBeforeUpdate += OnBeforeSlotUpdate;
        _inventory.GetSlots[i].OnAfterUpdate += OnAfterSlotUpdate;
    }
}

private void OnDisable()
{
    _playerStats.healthChangeEvent -= HandleDamage;
    for (int i = 0; i < _equipment.GetSlots.Length; i++)
    {
        _equipment.GetSlots[i].OnBeforeUpdate -= OnBeforeSlotUpdate;
        _equipment.GetSlots[i].OnAfterUpdate -= OnAfterSlotUpdate;
    }
    for (int i = 0; i < _inventory.GetSlots.Length; i++)
    {
        _inventory.GetSlots[i].OnBeforeUpdate -= OnBeforeSlotUpdate;
        _inventory.GetSlots[i].OnAfterUpdate -= OnAfterSlotUpdate;
    }
}

void Update()
{
    if(GameManager.Instance.State != GameState.PauseScreen)
    {
        _playerActions.HandlePlayerActions();
        _playerActions.HandleGunRotation();
        _movementController.HandleMovementInputs();
   }
}

private void FixedUpdate()
{
    _movementController.ProcessDodge();
    _movementController.Walking();
}

public void Damage(int damage)
```

```
{
    _playerStats.DecreaseHealth(damage);
    StartCoroutine(InvulnerabilityTimer());
}

public void HandleDamage(int value)
{
    CurrentHealth = value;
    if (value <= 0)
    {
        Death();
    }
    else
    {
        Debug.Log("Current health: " + CurrentHealth);
    }
}

private IEnumerator InvulnerabilityTimer()
{
    isPlayerInvulnerable = true;
    yield return new WaitForSeconds(_playerStats.InvulnerabilityDuration);
    isPlayerInvulnerable = false;
}

public void Death()
{
    Debug.Log($"[PlayerController] Death() called at Time={Time.time:F2}");
    OnPlayerDeath?.Invoke();
}

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("EnemyProjectile"))
    {
      Projectile hittingProjectile =
      collision.gameObject.GetComponentInParent<Projectile>();
        if (hittingProjectile != null)
        {
            if (isPlayerInvulnerable) return;
           SoundFXManager.Instance.PlaySoundFXClip(DamageClip, this.transform, 0.7f);
            _playerStats.DecreaseHealth(hittingProjectile.Damage);
            StartCoroutine(InvulnerabilityTimer());
            hittingProjectile.ObjectPool.Release(hittingProjectile);
        }
    }
    if (collision.CompareTag("SunLaser"))
    {
        LaserDamage laser = collision.gameObject.GetComponent<LaserDamage>();
        if (laser != null)
        {
            if (isPlayerInvulnerable) return;
           SoundFXManager.Instance.PlaySoundFXClip(DamageClip, this.transform, 0.7f);
            _playerStats.DecreaseHealth(laser.Damage);
            StartCoroutine(InvulnerabilityTimer());
        }
    }
}

public void StatModified(PlayerAttributes stat)
```

```
    {
        Debug.Log($"{stat.Type} got updated it is now {stat.Value.ModifiedValue}");
    }

    public void ResetInventories()
    {
        _equipment.Container.Clear();
        _inventory.Container.Clear();
        _equipment.AddItem(new Item(_equipment.Database.ItemObjects[3]), 1);
    }

}
```

4 Файл PlayerMovement.cs

```
using System.Collections;
using System.Drawing;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.Rendering;

public class PlayerMovement : MonoBehaviour
{
    public InputActionAsset InputActions;

    private InputAction m_moveAction;
    private InputAction m_lookAction;
    private InputAction m_dodgeAction;


    private Vector2 m_moveAmt;
    private Vector2 m_lookAmt;

    [SerializeField] Rigidbody2D _rigidbody;

    [SerializeField] PlayerAttributesSO _playerStats;
    private PlayerController _playerController;
    private int m_moveSpeed;
    [Header("Dodge Settings")]
    [SerializeField]
    [Tooltip("How long (seconds) the dash lasts.")]
    private float dodgeDuration = 0.2f;
    [SerializeField]
    [Tooltip("Time (seconds) before you can dash again.")]
    private float dodgeCooldown = 0.5f;
    [SerializeField]
    [Tooltip("Seconds of invulnerability during dash.")]
    float invulDuration = 0.2f;
    [SerializeField]
    [Tooltip("Layers that will collide with player")]
    private LayerMask collidableLayer;
    private bool m_canDodge = true;
    private bool m_isDodging = false;
    private float m_dodgeSpeed;
    private float m_dodgeDistance;
    private bool m_dodgePressed;
    [SerializeField]
    Collider2D _tileCollider;

    private void Awake()
    {
        m_moveAction = InputSystem.actions.FindAction("Move");
        m_lookAction = InputSystem.actions.FindAction("Look");
        m_dodgeAction = InputSystem.actions.FindAction("Dodge");
    }
    private void Start()
    {
        m_moveSpeed = _playerStats.GetStatValue(TypeOfStat.MoveSpeed);
        m_dodgeDistance = _playerStats.GetStatValue(TypeOfStat.DodgeDistance);
        _playerController = GetComponent<PlayerController>();
    }
```

```
public void HandleMovementInputs()
{
    if (m_isDodging) return;

    m_moveAmt = m_moveAction.ReadValue<Vector2>();
    m_lookAmt = m_lookAction.ReadValue<Vector2>();
    if (m_dodgeAction.WasPressedThisFrame() && m_canDodge && !m_isDodging)
    {
        m_dodgePressed = true;
    }
}

public void Walking()
{
    if (m_isDodging) return;
    _rigidbody.linearVelocity = m_moveAmt * m_moveSpeed;
}

public void ProcessDodge()
{
    if(m_dodgePressed && m_canDodge && !m_isDodging)
    {
        StartCoroutine(IDodge());
    }
    m_dodgePressed = false;
}

IEnumerator IDodge()
{
    m_canDodge = false;
    m_isDodging = true;

    var input = m_moveAmt;
    if (input.sqrMagnitude < 0.1f)
    {
        m_canDodge = true;
        m_isDodging = false;
        yield break;
    }
    StartCoroutine(IDodgeInvulnerability(invulDuration));

    var remainingDistance = m_dodgeDistance;
    Vector2 dodgeDir = input.normalized;
    m_dodgeSpeed = m_dodgeDistance / dodgeDuration;
    float elapsed = 0f;
    float skinWidth = 0.1f;

    Bounds bounds = _tileCollider.bounds;
    float collWidth = bounds.size.x;
    float collHeight = bounds.size.y;

    while (elapsed < dodgeDuration && remainingDistance > 0f)
    {
        // 1) Calculate how far we want to move *this* FixedUpdate
        float stepDistance = m_dodgeSpeed * Time.fixedDeltaTime;
        stepDistance = Mathf.Min(stepDistance, remainingDistance);

        Vector2 currentPos = _rigidbody.position;
        Vector2 perpendicular = new Vector2(-dodgeDir.y, dodgeDir.x);
        float minHitDistance = float.MaxValue;
```

```
        bool hitDetected = false;


        for (int i = -1; i <= 1; i++)
        {
            Vector2 rayStart = currentPos + perpendicular * (i * collWidth * 0.4f);
            RaycastHit2D hit = Physics2D.Raycast(
                rayStart,
                dodgeDir,
                stepDistance + skinWidth,
                collidableLayer
            );

            if (hit.collider != null)
            {
                hitDetected = true;
                minHitDistance = Mathf.Min(minHitDistance, hit.distance);
            }
        }


        float actualMove = hitDetected
            ? Mathf.Max(minHitDistance - skinWidth, 0f)
            : stepDistance;


        _rigidbody.MovePosition(_rigidbody.position + dodgeDir * actualMove);
        remainingDistance -= actualMove;


        elapsed += Time.fixedDeltaTime;
        yield return new WaitForFixedUpdate();
    }

    m_isDodging = false;
    yield return new WaitForSeconds(dodgeCooldown);
    m_canDodge = true;
}

IEnumerator IDodgeInvulnerability(float invulDuration)
{
    _playerController.isPlayerInvulnerable = true;
    yield return new WaitForSeconds(invulDuration);
    _playerController.isPlayerInvulnerable = false;
}

private void ChangeSpeed(int value)
{
    m_moveSpeed = value;
}

private void ChangeDodgeDistance(int value)
{
    m_dodgeDistance = value;
}

private void OnEnable()
{
    InputActions.FindActionMap("Player").Enable();
    _playerStats.moveSpeedChangeEvent += ChangeSpeed;
```

```
        _playerStats.dodgeDistanceChangeEvent += ChangeDodgeDistance;

    }

    private void OnDisable()
    {
        InputActions.FindActionMap("Player").Disable();
        _playerStats.moveSpeedChangeEvent -= ChangeSpeed;
        _playerStats.dodgeDistanceChangeEvent -= ChangeDodgeDistance;
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (m_isDodging)
        {
            m_isDodging = false;
            _rigidbody.linearVelocity = Vector2.zero;
        }
    }
}
```

5 Файл PlayerActions.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerActions : MonoBehaviour
{
    public InputActionAsset InputActions;
    [SerializeField]
    private GunSelector _gunSelector;
    [SerializeField]
    private bool AutoReload = true;

    private bool m_isReloading = false;

    private InputAction m_primaryAttackAction;
    private InputAction m_secondaryAttackAction;
    private InputAction m_itemUseAction;
    private InputAction m_interactionAction;
    private InputAction m_nextPrimaryWeaponAction;
    private InputAction m_previousPrimaryWeaponAction;
    private InputAction m_reloadAction;

    void Awake()
    {
        if (InputActions == null)
        {
            Debug.LogError("InputActionAsset is not assigned to PlayerActions script!");
            return;
        }
        m_primaryAttackAction = InputActions.FindAction("Attack");
        m_secondaryAttackAction = InputActions.FindAction("SecondaryAttack");
        m_itemUseAction = InputActions.FindAction("ItemUse");
        m_interactionAction = InputSystem.actions.FindAction("Interact");

        m_nextPrimaryWeaponAction = InputActions.FindAction("NextPrimaryWeapon");
        m_previousPrimaryWeaponAction = InputActions.FindAction("PreviousPrimaryWeapon");
        m_reloadAction = InputActions.FindAction("Reload");
    }

    private void OnEnable()
    {
        if (InputActions != null)
        {
            InputActions.Enable();
        }

        if (m_nextPrimaryWeaponAction != null)
            m_nextPrimaryWeaponAction.performed += OnNextPrimaryWeapon;
        if (m_previousPrimaryWeaponAction != null)
            m_previousPrimaryWeaponAction.performed += OnPreviousPrimaryWeapon;
    }

    private void OnDisable()
    {
        if (m_nextPrimaryWeaponAction != null)
            m_nextPrimaryWeaponAction.performed -= OnNextPrimaryWeapon;
        if (m_previousPrimaryWeaponAction != null)
```

```
        m_previousPrimaryWeaponAction.performed -= OnPreviousPrimaryWeapon;

    if (InputActions != null)
    {
        InputActions.Disable();
    }
}

private void OnNextPrimaryWeapon(InputAction.CallbackContext context)
{
    _gunSelector.SwitchToNextPrimaryWeapon();
}

private void OnPreviousPrimaryWeapon(InputAction.CallbackContext context)
{
    _gunSelector.SwitchToPreviousPrimaryWeapon();
}

public void HandlePlayerActions()
{
    if(m_isReloading) return;
    if (GameManager.Instance.State == GameState.PauseScreen) return;
    if (GameManager.Instance.State == GameState.RewardScreen) return;

    if (m_primaryAttackAction.IsPressed() && _gunSelector.HasPrimaryWeapon())
    {
        Debug.Log("Trying to shoot");
        if(_gunSelector.ActivePrimaryGun.AmmoConfig.CurrentClipAmmo > 0)
        {
            Debug.Log("Shooting");
            _gunSelector.ActivePrimaryGun.Shoot();
        }

    }
    if (m_secondaryAttackAction.IsPressed() && _gunSelector.HasSecondaryWeapon())
    {
        if (_gunSelector.ActiveSecondaryGun.AmmoConfig.CurrentClipAmmo > 0)
        {
            _gunSelector.ActiveSecondaryGun.Shoot();
        }
    }
    if(ShouldManualReload() || ShouldAutoReload() && !m_isReloading)
    {
        StartCoroutine(ReloadCoroutine());
    }
}

IEnumerator ReloadCoroutine()
{
    m_isReloading = true;
    float reloadTime = _gunSelector.ActivePrimaryGun.AmmoConfig.ReloadTime;

    yield return new WaitForSeconds(reloadTime);

    _gunSelector.ActivePrimaryGun.AmmoConfig.Reload();
    m_isReloading = false;
}

private bool ShouldManualReload()
{
```

```
        if (_gunSelector.ActivePrimaryGun == null) return false;
        return m_reloadAction.WasReleasedThisFrame()
            && _gunSelector.ActivePrimaryGun.CanReload();
    }

    private bool ShouldAutoReload()
    {
        if(_gunSelector.ActivePrimaryGun == null) return false;
        return AutoReload
            && _gunSelector.ActivePrimaryGun.AmmoConfig.CurrentClipAmmo == 0
            && _gunSelector.ActivePrimaryGun.CanReload();
    }
    public void HandleGunRotation()
    {
      Vector3 mousePosition =
      Camera.main.ScreenToWorldPoint(Mouse.current.position.ReadValue());
        Vector2 direction = (mousePosition - this.transform.position).normalized;
        float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        Quaternion targetRotation = Quaternion.Euler(0, 0, angle);

        bool isMouseRight = Mouse.current.position.ReadValue().x > Screen.width / 2.0f;


        if (_gunSelector.HasPrimaryWeapon())
        {
            _gunSelector.ActivePrimaryGun.ActiveRotation = targetRotation;

            _gunSelector.ActivePrimaryGun.SpriteRenderer.flipX = false;
            _gunSelector.ActivePrimaryGun.SpriteRenderer.flipY = !isMouseRight;
        }


        if (_gunSelector.HasSecondaryWeapon())
        {
            _gunSelector.ActiveSecondaryGun.ActiveRotation = targetRotation;

            _gunSelector.ActiveSecondaryGun.SpriteRenderer.flipX = false;
            _gunSelector.ActiveSecondaryGun.SpriteRenderer.flipY = !isMouseRight;
        }
    }
}
```

6 Файл UIManager.cs

```
using DG.Tweening;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography;
using System;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.SceneManagement;

public class UIManager : MonoBehaviour
{
    public static UIManager Instance { get; private set; }
    public InputActionAsset InputActions;

    [Header("HUD Elements")]
    [SerializeField] private AmmoDisplay _ammoCounter;


    [Header("Game Screens")]
    [SerializeField] private GameObject _equipmentScreen;
    [SerializeField] private GameObject _inventoryScreen;
    [SerializeField] private GameObject _pauseScreen;
    [SerializeField] private GameObject _rewardScreen;
    [SerializeField] private GameObject _winScreen;
    [SerializeField] private GameObject _loseScreen;
    [SerializeField]
    private int TIME_TO_OPEN;

    [Header("Menu Systems")]
    [SerializeField] private GameObject _mainMenu;

    [Header("UI Groups")]
    [SerializeField] private GameObject hudGroup;
    [SerializeField] private GameObject gameScreensGroup;

    [Header("Configuration")]
    [SerializeField] private List<string> fullyExcludedScenes;

    private bool isInventoryOpen = false;
    private InputAction m_openInventoryAction;
    private InputAction m_pauseAction;


    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject.transform.parent.gameObject);
            SceneManager.sceneLoaded += OnSceneLoaded;
            GameManager.OnGameStateChanged += OnGameEnd;
            GameManager.OnGameStateChanged += OnRewardScreen;

            if (InputActions == null)
            {
                      Debug.LogError("InputActionAsset is not assigned to PlayerActions
```

```
script!");
                return;
            }
            m_openInventoryAction = InputActions.FindAction("OpenInventory");
            m_pauseAction = InputActions.FindAction("Pause");


        }
        else
        {
            Destroy(gameObject);
        }
    }

    private void Update()
    {
     if(!isInventoryOpen      &&      m_openInventoryAction.WasPressedThisFrame()      &&
GameManager.Instance.State != GameState.PauseScreen)
        {
            Debug.Log("Opened Inventory");
            OpenEquipmentScreen();
            OpenInventoryScreen();
            StartCoroutine(WaitForInventory());
        }
        if    (isInventoryOpen    &&    m_openInventoryAction.WasPressedThisFrame()    &&
GameManager.Instance.State != GameState.PauseScreen)
        {
            Debug.Log("Closed Inventory");
            CloseEquipmentScreen();
            CloseInventoryScreen();
            StartCoroutine(WaitForInventory());
        }
        if (m_pauseAction.WasPressedThisFrame())
        {
            if (GameManager.Instance.State == GameState.PauseScreen)
            {
                ClosePauseScreen();
            }
            else
            {
                OpenPauseScreen();
            }
        }
    }

    IEnumerator WaitForInventory()
    {
        yield return new WaitForSeconds(TIME_TO_OPEN);
        if (!isInventoryOpen)
        {
            isInventoryOpen = true;
        }
        else
        {
            isInventoryOpen = false;
        }
    }

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
```

```
    ConfigureUIForScene(scene.name);
}

private void ConfigureUIForScene(string sceneName)
{
    switch (sceneName)
    {
        case "MainMenu":
            SetHUDActive(false);
            SetGameScreensActive(false);
            SetMainMenuActive(true);
            break;

        case "Gameplay":
            SetHUDActive(true);
            SetGameScreensActive(true);
            SetMainMenuActive(false);
            break;
        case "Map":
            SetHUDActive(false);
            SetGameScreensActive(false);
            SetMainMenuActive(false);
            break;
        default:
            SetHUDActive(true);
            SetGameScreensActive(true);
            SetMainMenuActive(false);
            break;
    }
}

public void SetHUDActive(bool active)
{
    if (hudGroup)
        hudGroup.SetActive(active);
}

public void SetGameScreensActive(bool active)
{
    if (gameScreensGroup)
        gameScreensGroup.SetActive(active);

    if (!active)
    {
        CloseAllGameScreens();
    }
}

public void SetMainMenuActive(bool active)
{
    if (_mainMenu != null)
    {
        Debug.Log($"Setting MainMenu active: {active},
        Current state: {_mainMenu.activeInHierarchy}");
        _mainMenu.SetActive(active);
        Debug.Log($"MainMenu state after change: {_mainMenu.activeInHierarchy}");
    }
    else
    {
        Debug.LogError("MainMenu GameObject reference is null!");
```

```
        }
    }

    private void CloseAllGameScreens()
    {
        CloseEquipmentScreen();
        CloseInventoryScreen();
        ClosePauseScreen();
        CloseRewardScreen();
    }

    private void OnGameEnd(GameState state)
    {
        if(state == GameState.LoseScreen)
        {
            _loseScreen.GetComponent<RectTransform>().anchoredPosition = Vector2.zero;
        }
        else if (state == GameState.WinScreen)
        {
            _winScreen.GetComponent<RectTransform>().anchoredPosition = Vector2.zero;
        }
        else
        {
        _loseScreen.GetComponent<RectTransform>().anchoredPosition = new Vector2(0, 900);
        _winScreen.GetComponent<RectTransform>().anchoredPosition = new Vector2(0, 900);
        }
    }

    private void OnRewardScreen(GameState state)
    {
        if (state == GameState.RewardScreen)
        {
            OpenRewardScreen();

        }
    }
    public void OpenEquipmentScreen() =>
_equipmentScreen.GetComponent<RectTransform>().DOAnchorPosX(-900, TIME_TO_OPEN);
    public void CloseEquipmentScreen() =>
_equipmentScreen.GetComponent<RectTransform>().DOAnchorPosX(-1900, TIME_TO_OPEN);
    public void OpenInventoryScreen() =>
_inventoryScreen.GetComponent<RectTransform>().DOAnchorPosX(900, TIME_TO_OPEN);
    public void CloseInventoryScreen() =>
_inventoryScreen.GetComponent<RectTransform>().DOAnchorPosX(1900, TIME_TO_OPEN);
    public void OpenPauseScreen()
    {
        _pauseScreen.GetComponent<RectTransform>().anchoredPosition = Vector2.zero; ;
        GameManager.Instance?.PauseGame();
    }
    public void ClosePauseScreen()
    {
       _pauseScreen.GetComponent<RectTransform>().anchoredPosition = new Vector2(0, 900);
        GameManager.Instance?.UnPause();
    }
    public void OpenRewardScreen() =>
_rewardScreen.gameObject.GetComponent<RectTransform>().DOAnchorPosY(350, TIME_TO_OPEN);
    public void CloseRewardScreen() =>
_rewardScreen.gameObject.GetComponent<RectTransform>().DOAnchorPosY(900, TIME_TO_OPEN);
```

```
private void OnDestroy()
    {
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }
}
```