Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

«МОДЕЛЮВАННЯ ЛЮДИНО-МАШИННОЇ ВЗАЄМОДІЇ З УПРАВЛІННЯМ ЇЇ МЕТАПРАВИЛАМИ (ЗА ЗРАЗКОМ DON'T STARVE) (BACKEND)»

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проекту

№	П.І.Б.	Група
1	Кузьмін Антон Дмитрович	КН-П-201
2	Юркевіч Володимир Олександрович	КН-П-201
3	Руденко Андрій Олегович	КН-П-201
4	Андрющенко Владислав Павлович	КН-Д-201
5	Латій Марія Андріївна	КН-Д-201

Автор звіту

_____ Руденко Андрій

Одеса – 2024

АНОТАЦІЯ

Цей науковий документ присвячений ретельному аналізу та проектуванню програмної системи, спрямованої на ігрову індустрію з основним фокусом на швидке додавання нового контенту. В роботі детально розглянуті ключові етапи життєвого циклу розробки програмного забезпечення, включаючи аналіз вимог, проектування архітектури, реалізацію та подальшу підтримку системи.

Розробка програмної системи зосереджувалася на використанні передових інструментів і технологій таких як Unity а також його внутрішніх інструментів (Cinemachine, ScriptableObjects) і сторонніх пакетів (Photon), що сприяли створенню високоякісного та надійного програмного продукту. Особлива увага приділялася можливості швидкої ітерації та впровадження змін, що дозволяє підтримувати високий темп розвитку гри відповідно до змінюючихся вимог ринку та користувацьких очікувань.

Під час розробки системи виникали технічні виклики, такі як оптимізація продуктивності, управління ресурсами та мультиплеер.

Загальна структура документа включає аналіз основних викликів, з якими стикається ігрова індустрія, та практичні рекомендації щодо їх вирішення. Результати дослідження можуть бути корисними для фахівців із розробки програмного забезпечення, які працюють у галузі ігрової індустрії та інших галузях, де важлива швидкість і гнучкість у розробці програмних продуктів.

ЗМІСТ

АНОТАЦІЯ 1
ЗМІСТ 2
ВСТУП 2
ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ 3
Загальний опис. 3
Призначення. 4
Вимоги до функціоналу. 4
Архітектура. 5
Технології. 7
Команда проєкту. 7
РОЗДІЛ 1 8
РОЗДІЛ 2 11
РОЗДІЛ 3 13
РОЗДІЛ 4 15
ВИСНОВКИ 18
ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ НАВЕДЕНОЇ ЧАСТИНИ ПРОЄКТУ 19

ВСТУП

У сучасному світі інформаційних технологій програмні системи відіграють вирішальну роль у різних сферах діяльності. Від бізнесу та освіти до охорони здоров'я та розваг, програмне забезпечення стало невід'ємною частиною повсякденного життя, забезпечуючи автоматизацію процесів, зручність користування та ефективність виконання завдань. Однією з найбільш динамічних і швидкозростаючих сфер є ігрова індустрія, де програмні системи мають особливе значення. Процес створення ігрового програмного забезпечення є складним і багатогранним,

включаючи етапи аналізу, проектування, розробки та подальшої підтримки.

Цей документ зосереджений на аналізі програмної системи, проектування якої націлене на створення інструменту, що дозволить швидко додавати новий контент у гру.

Розробка є центральним етапом, на якому програмісти перетворюють проектну документацію в реальний програмний продукт. Використання сучасних мов програмування, фреймворків та інструментів значно спрощує цей процес. Особливу увагу приділено тому, щоб новий контент можна було легко інтегрувати в існуючу ігрову середу. Однак, не менш важливою є і подальша підтримка системи, яка включає виправлення помилок, оновлення та вдосконалення програмного забезпечення з урахуванням зворотного зв'язку від користувачів.

Під час розробки програмних систем неминуче виникають проблеми, пов'язані з різними аспектами роботи. Це можуть бути технічні складнощі, проблеми сумісності, обмеження ресурсів або невідповідність вимогам. Цей документ детально описує ці проблеми та пропонує шляхи їх вирішення, базуючись на реальному досвіді розробників та перевірених практиках.

Організація командної роботи є ключовим фактором успішної розробки програмних систем. Ефективна комунікація, розподіл ролей та обов'язків, використання методологій Agile або Scrum сприяють підвищенню продуктивності та якості кінцевого продукту.

Таким чином, цей документ містить всебічний аналіз програмної системи, створеної для ігрової індустрії, з акцентом на можливість швидкого додавання нового контенту в гру. Описані всі ключові етапи розробки та підтримки системи, проблеми та шляхи їх вирішення, а також методи організації командної роботи, що сприяють досягненню високих результатів у розробці програмного забезпечення.

У наступному розділі представлено технічне завдання проекту. У роботі розглянуто створення системи генерації збереження та загрузки ігрового пространсва, а також можливість мультиплеерної гри. У висновках підсумовано виконану роботу.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ

Моделювання людино-машинної взаємодії з управлінням її метаправилами (за зразком Don’t Starve)

Загальний опис.

Задачею програмного забезпечення (ПЗ) є створення системи взаємодії людини (гравця) та елементами ігрового середовища. Функцією системи є забезпечення інтерактивності довкілля: взаємодії гравця з елементами середовища, неігровими персонажами, з інтерфейсом користувача та іншими гравцями.

Кожна підсистема відповідає за окрему частину досвіду гравця, серед них: система ремесла - поєднання предметів оточення для отримання нових предметів, система досягнень - використання предметів для отримання винагороди, система дослідження внутрішньо ігрового контенту, система подолання перешкод для перевірки навичок та знань гравця про метаправила системи. Також система містить підсистеми, що забезпечуть створення ігрового оточення, з’єднання до 4 гравців через локальну мережу та мережу Інтернет для спільної взаємодії з єдиним ігровим оточенням та реалізації персоналізації та самовираження гравця в середині ігрової системи.

Призначення.

Даний проект розробляється для створення відеогри, яка надає гравцям можливість взаємодії з віртуальним світом та іншими гравцями. Головна мета гри полягає в наданні гравцям тривалого та захоплюючого ігрового досвіду через різноманітні ігрові механіки, такі як дослідження, ремесло, бій, кооперація та виконання завдань. Гра орієнтована на підтримку регулярного оновлення контенту, що дозволить зберігати інтерес гравців протягом довгого часу та забезпечити їх повернення до гри.

Вимоги до функціоналу.

Головне меню

Стартовий інтерфейс ПЗ (головна меню або стартова сцена) служить вихідною точкою для користувача та повинна містити можливість налаштувань звуку, налаштування кастомізації ігрового аватара користувача, створення власного ігрового оточення (світу) для одиночної гри, гри по локальній мережі та грі по мережі Інтернет, а також під’єднання до існуючих ігрових оточень в локальній мережі та мережі Інтернет.

Системи взаємодії

Системи взаємодії гравця з системою (ігрові механіки) забезпечують гравця задачами та цілями в середині ігрової системи (світу).

- Механіка комбінування предметів для отримання нових (система ремесла) заключається в пошуку предметів в ігровому світі та комбінуванні даних предметів. Комбінування предметів відбувається за допомогою спеціальних предметів ігрового оточення. Для розробників потрібно забезпечити зручний інструмент для додавання нових предметів та їх комбінацій
- Механіка взаємодії з активними об’єктами ігрового оточення (неігровими персонажами, НІП). Неігрові персонажи - набір

персонажів, керування якими здійснюється системою і взаємодія гравця з якими може різнитися залежно від цілі гравця та типу НІПа. Для розробників необхідно забезпечити зручний інструмент для додавання нових НІПів.

- Діалог. Це тип взаємодії з неігровими персонажами, в результаті якої гравець отримує завдання, які необхідно виконати всередині системи для отримання винагороди.
- Бій. Це тип взаємодії, суть якої полягає в змаганні гравця з системою, в якій гравцю, використовуючи наявні інструменти необхідно знизити показник НІПа до нуля раніше, ніж НІП зробить це у гравця. За перемогу в змаганні гравець отримує винагороду у вигляді предметів, а за поразку - покарання у вигляді втрати предметів та прогресу.
- Кооперація. Це тип взаємодії з неігровими персонажами, суть якої полягає у використанні спеціальних предметів з метою переведення НІПа під керування гравцем.

Багатокористувацький режим

Багатокористувацький режим - це спеціальний режим гри, який забезпечує з’єднання до 4 гравців на різних пристроях в рамках єдиного ігрового оточення. Даний режим має реалізовувати з’єднання по локальній мережі та мережі Інтернет.

Архітектура.

Архітектура програмного забезпечення відеогри включає в себе кілька підсистем, які взаємодіють між собою для забезпечення цілісного ігрового досвіду. Нижче наведено основні компоненти архітектури та їх опис:

Основні компоненти

Core (Ядро)

Опис: Основні механізми гри, включаючи головний цикл, керування станами гри, завантаження ігрових сцен тощо.

Функції:

- Менеджер станів гри
- Завантаження/збереження прогресу
- Керування ресурсами

Player (Гравець)

Опис: Всі аспекти, що стосуються гравця, включаючи управління аватаром, інвентарем тощо.

Функції:

- Контроль аватара
- Інвентар та крафтинг

Environment (Оточення)

Опис: Система, що створює та керує ігровим світом, включаючи генерацію рівнів, ресурси та інтерактивні об'єкти.

Функції:

- Генерація рівнів
- Розміщення ресурсів
- Керування об'єктами оточення

NPCs (Неігрові персонажі)

Опис: Логіка та поведінка всіх неігрових персонажів (НІПів), включаючи ворогів, союзників та нейтральних персонажів.

Функції:

- Логіка поведінки НІПів
- Система діалогів
- Взаємодія з гравцем (бій, кооперація)

Quests (Завдання)

Опис: Система завдань, яка забезпечує створення та керування квестами та місіями для гравця.

Функції:

- Створення квестів
- Відслідковування прогресу
- Винагороди за виконання завдань

Multiplayer (Багатокористувацький режим)

Опис: Інфраструктура для підключення гравців через локальну мережу та мережу Інтернет для спільної гри.

Функції:

- З'єднання гравців
- Синхронізація станів гри
- Керування сесіями

UI (Інтерфейс користувача)

Опис: Всі елементи інтерфейсу користувача, включаючи головне меню, інвентар, кастомізацію та інші елементи управління.

Функції:

- Головне меню
- Налаштування
- Інтерактивні елементи

Технології.

Зберігання вихідних кодів ПЗ здійснюється за допомогою системи контроля версій (вибір системи узгоджується групою проєкту).

Структура файлів та каталогів повинна відображати архітектуру ПЗ:

- Design
- Scripts
    - Core
    - Player
    - Environment
    - NPCs
    - Quests
    - Multiplayer
    - UI

Для розробки рекомендуються наступні технології створення:

- Ігровий процес: Ігровий рушій Unity та мова програмування C#
- Дизайн: Adobe Photoshop, Adobe Illustrator, FontEditor, Adobe InDesign
- Публікація: веб-ресурс itch.io та технологія WebGL

Команда проєкту.

Для виконання поставлених завдань робота команди передбачається окремо за кожною частиною ігрової функціональності.

При команді в 5 учасників троє учасників відповідають за створення програмних систем, 2 відповідають за візуальну частину наповнення ігрового контенту.

РОЗДІЛ 1
ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

В результаті ґрунтовного аналізу вимог технічного завдання до проєкту нашою командою було вирішено використовувати ігровий рушій Unity для розробки через його високу продуктивність, зручність використання, сучасні можливості розробки та обширну бібліотеку ресурсів спільноти. Unity дозволяє створювати якісні ігри з використанням багатьох інструментів, які спрощують процес розробки та забезпечують високу продуктивність кінцевого продукту.

Основа програмної архітектури складається з наступних компонентів:

- **Core**: Основні механізми гри, включаючи головний цикл, керування станами гри, завантаження ігрових сцен тощо.
- **Player**: Всі аспекти, що стосуються гравця, включаючи управління аватаром, інвентарем тощо.
- **Environment**: Система, що створює та керує ігровим світом, включаючи генерацію рівнів, ресурси та інтерактивні об'єкти.
- **NPCs**: Логіка та поведінка всіх неігрових персонажів (НІПів), включаючи ворогів, союзників та нейтральних персонажів.
- **Quests**: Система завдань, яка забезпечує створення та керування квестами та місіями для гравця.
- **Multiplayer**: Інфраструктура для підключення гравців через локальну мережу та мережу Інтернет для спільної гри.
- **UI**: Всі елементи інтерфейсу користувача, включаючи головне меню, інвентар, кастомізацію та інші елементи управління.
- **Design**: Представляє набір готових елементів інтерфейсу для головного меню, меню створення ігрового світу, меню підключення до гри через мережу та меню в середині гри, а саме: меню павзи, меню гравця, інвентар, показники.

Частина “Design” включає створення та налаштування елементів інтерфейсу, які забезпечують зручність користування грою. Для меню

гравця та інвентарю за допомогою інструментів Unity було додано анімації відкриття та закриття, що надає грі більш динамічного та привабливого вигляду. Також частина “Design” включає набір зображень (спрайтів) для аватара гравця, неігрових персонажів, елементів ігрового середовища та такі елементи брендингу, як логотип та шрифт. Важливим аспектом було забезпечення зручного та інтуїтивно зрозумілого інтерфейсу, який дозволяє гравцям легко взаємодіяти з грою.

Програмна частина проєкту була повністю реалізована з використанням мови C# та Unity Scripting API. Широко використовувались додаткові пакети та інструменти Unity, такі як Cinemachine для налаштування камери, що забезпечує плавне та якісне управління камерою в грі, а також сценарні об’єкти (Scriptable Objects), які спрощують зберігання і управління даними. Також для реалізації багатокористувацького режиму було використано сторонній пакет Photon. Цей пакет було обрано через його зручність та легкість впровадження до проєктів Unity, що дозволило швидко та ефективно реалізувати функціонал мережевої гри.

Для реалізації збережень ігрового прогресу та створених ігрових світів було використано технологію JSON. Це забезпечує зручний та гнучкий спосіб зберігання даних, що дозволяє легко зберігати та завантажувати стан гри.

При проєктуванні та розробці програмної архітектури широко застосовувались різні патерни об’єктно-орієнтованого проєктування, такі як “Машина станів” (State Machine) та Model-view-controller (MVC). Використання цих патернів дозволило створити структуровану та модульну архітектуру, яка полегшує підтримку та розширення проєкту. Головним принципом проєктування було дотримання принципів проєктування SOLID, що забезпечує високу якість коду, легкість його

розширення та підтримки, а також низьку зв’язаність (low coupling) між компонентами системи.

Командна робота була організована за допомогою елементів різних методологій керування проєктами, що забезпечило ефективність і злагодженість процесу розробки. Було використано елементи SCRUM з тижневими спринтами, що дозволило чітко планувати та контролювати виконання задач на короткі проміжки часу. Для відстежування потоку задач використовувалися дошки Kanban, які забезпечували візуалізацію процесу роботи і допомагали команді бачити прогрес виконання завдань в режимі реального часу.

Для прискорення розробки програмного коду, передачі досвіду між членами команди та уникнення такого явища як “вежа знань”, використовувався елемент екстремального програмування, а саме парне програмування. Цей підхід дозволив підвищити якість коду, зменшити кількість помилок та забезпечити безперервний обмін знаннями між розробниками. Парне програмування сприяло кращому розумінню коду всіма членами командита та збільшило гнучкість у випадку зміни складу команди чи ролей всередині неї.

Вихідні файли розробленого проєкту зберігаються у публічному репозиторії веб-сервісу GitHub, що дозволяє легко відстежувати зміни, співпрацювати з іншими розробниками та забезпечувати контроль версій. Доступ до готової користувальницької частини проєкту знаходиться на веб-сервісі itch.io, що дозволяє гравцям легко грати в гру прямо в бравзері, що забезпечено технологією WebGL. Репозиторій GitHub доступний за посиланням: GitHub, а доступ до гри на itch.io - за посиланням: itch.io.

РОЗДІЛ 2
ГЕНЕРАЦІЯ ІГРОВОГО СЕРЕДОВИЩА

Ця частина програми складається з різних блоків

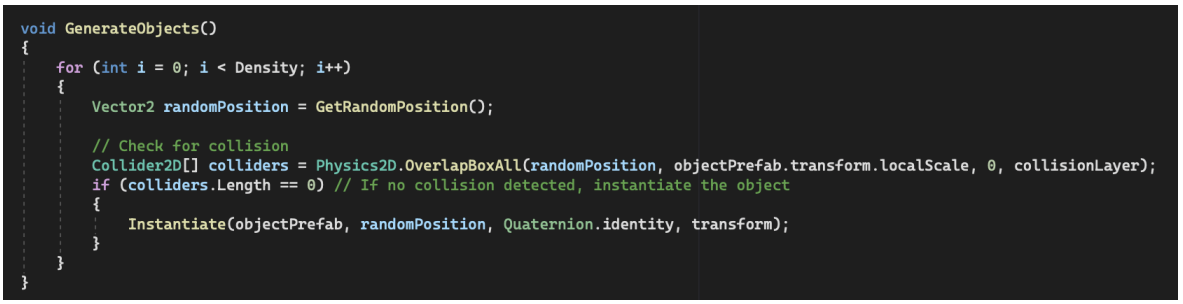
- генерація об'єктів середовища
- генерація та контроль NPC

Генерація об'єктів середовища

Для генерації об'єктів середовища було 2 варіанти розробки. Перший варіант мав на увазі підтримання одного шару та внесення змін до grid. Суть підходу полягає у зміні шару з біомами методом заміни вже існуючих, заповнених осередків біома на нове значення которе буде відповідати номеру елемента навколишнього середовища. У цього підходить є свої плюси і недоліки. Перший недолік це повільна швидкість генерації світу, так як перезапис масиву це досить таки не швидкий процес він досить негативно може вплинути на ебистродейсвие системи. Другий недолік - це візуальне відображення вже заповненої ігрової карти. Спрайти об'єктів оточення таких як кущі дерева і каміння самі собою різноманітні в розмірах і деякі їх частини є прозорими. У таких умовах осередок в якому харниться один з таких об'єктів в ігровому середовищі візуально відрізнятиметься від біома. Тобто під об'єктом не буде відображатися тип біома, в якому він знаходиться, що ламає цілісність і сприйняття гри. Але даних підхід дозволяє зберігати вже згенерований світ досить швидко і з відносно невеликою кількістю коду. Так як єдине, що треба зберігати, це один масив. Ще одна перевага - це уникнення колізій при гереації об'єктів оточення. Так як кожен об'єкт замінат тільки один осередок в масиві, другий об'єкт не може перекривати існуючий.

Другий це реалізація багатошарової системи, яка здатна швидко додавати нові об'єкти в ігровий простір. Даний підхід передбачає створення окремих шарів що беруть на себе генерацію об'єктів одного типу і розташування їх на карті світу. Як уже сказано вище, цей підхід набагато швидше для додавання нового коненту в ігровий світ, що було нашим пріоритетом. Єдиним завданням, що ускладнювало розробку, це

можливість об'єктів ставати на одні й ті ж координати, але на різних шарах. Рішення, яке було implemented це всім об'єктам виставити рівень шару “environment” і кожному об'єкту додати колайдери (рис. 1).

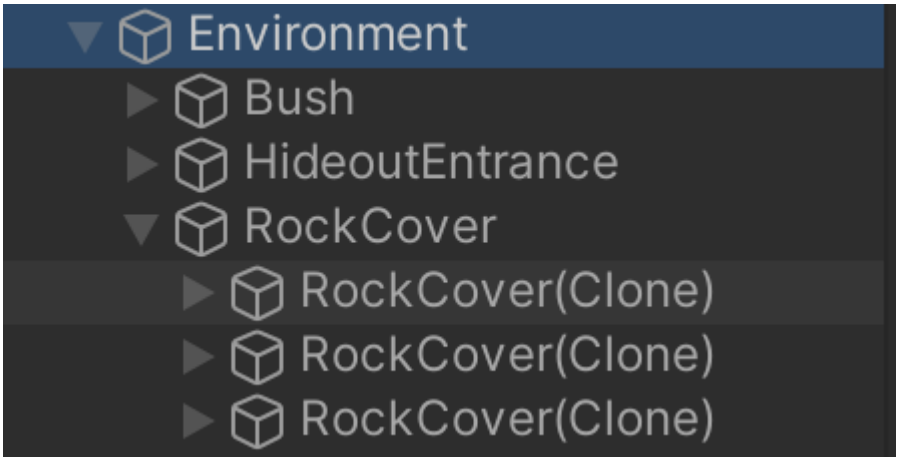


```
void GenerateObjects()
{
    for (int i = 0; i < Density; i++)
    {
        Vector2 randomPosition = GetRandomPosition();

        // Check for collision
        Collider2D[] colliders = Physics2D.OverlapBoxAll(randomPosition, objectPrefab.transform.localScale, 0, collisionLayer);
        if (colliders.Length == 0) // If no collision detected, instantiate the object
        {
            Instantiate(objectPrefab, randomPosition, Quaternion.identity, transform);
        }
    }
}
```

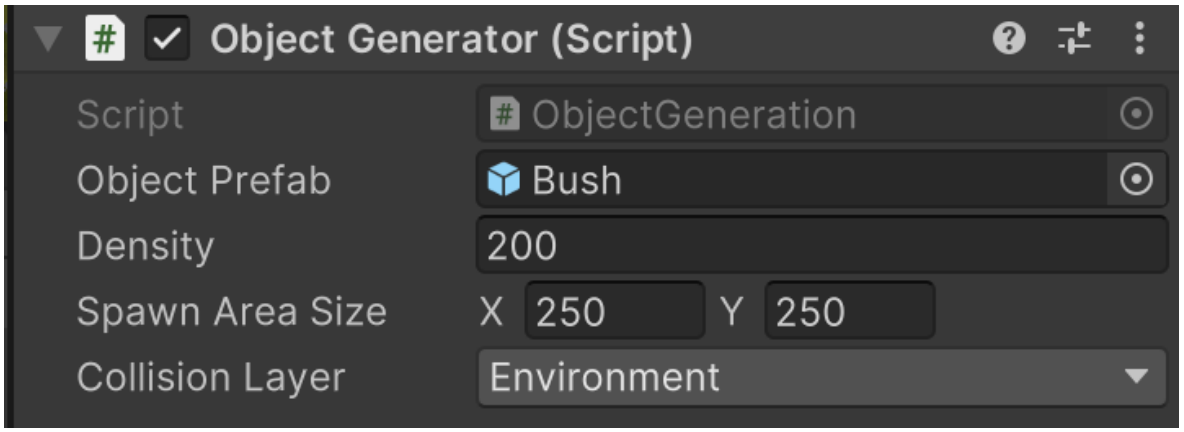
(рис. 1)

Організація об'єктів зроблена наступними образами. Кадий тип об'єднаний в один parent object, що тягне за собою компактнішу і зрозумілішу для сприйняття і контролю структуру. Всі об'єкти з шарами об'єднані в один parent object - environment (рис. 2).



(рис. 2)

Структура об'єкта дозволяє змінювати: префаб, щільність заполонення на карті, координати - які позначають межі спавна і тег шару на котлором будуть розміщуватися нові об'єкти (рис 3).



(рис. 3)

Генерація та контроль кількості NPC

Генерація і контроль кількості NPC здійснюється за рахунок скритпу який ставиться на об'єкт батько. Даний скрипт контролює NPC, максимальна кількість якого повинна згенеруватися і проміжок між генерацією нового об'єкта. Тобто, наприклад будиночок у селі виступає у ролі батьківського об'єкта і він генерує парубків.

```
[SerializeField]private GameObject enemyPrefab; // The enemy prefab to spawn
public Transform spawnPoint;   // The spawn point
[SerializeField] private int maxEnemies = 5;     // Maximum number of enemies on the map
[SerializeField] private float spawnInterval = 2f; // Time interval between spawns
```

(рис. 4)

РОЗДІЛ 3
ЗБЕРЕЖЕННЯ ТА ЗАВАНТАЖЕННЯ ИГОРОВОГО СЕРЕДОВИЩА

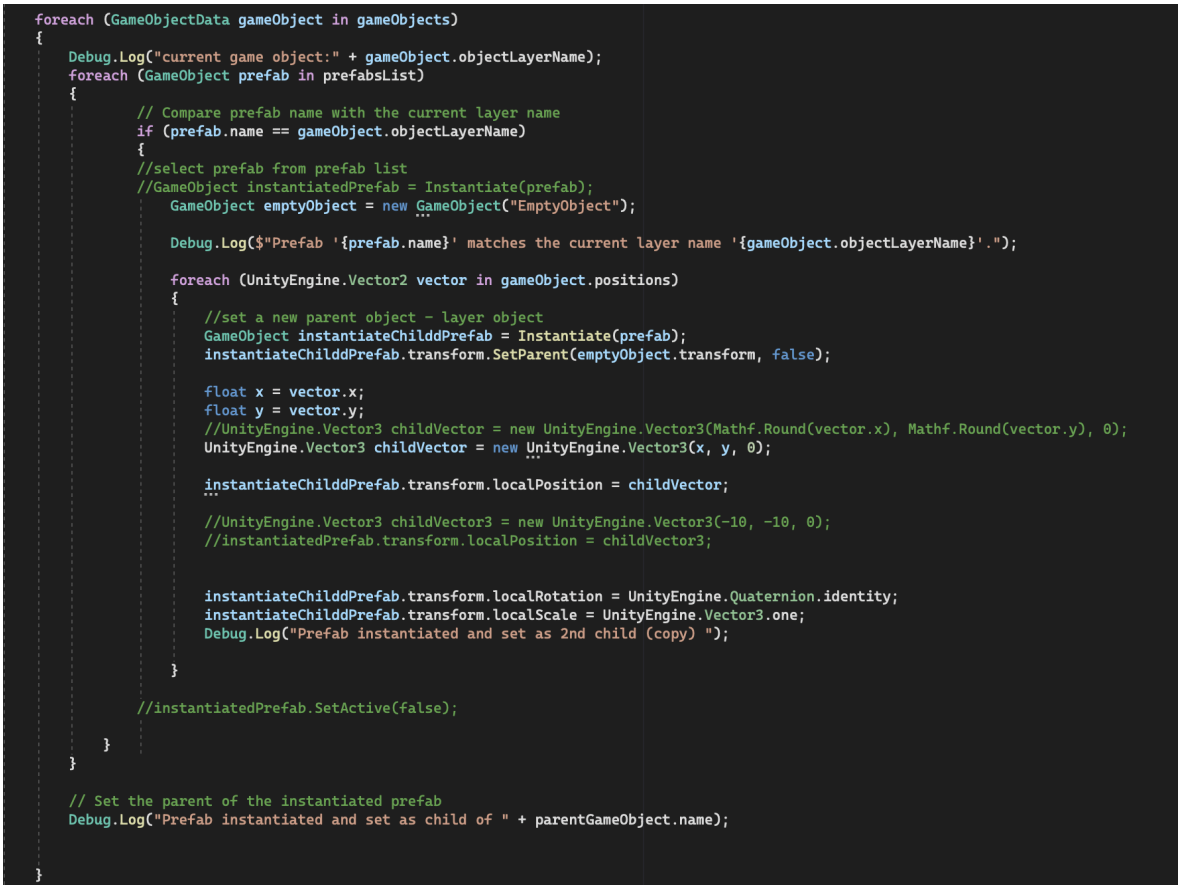
Заощадження та завантаження ігрового середовища здійснюється двома різними скриптами LevelSaverr і LevelLoader відповідно. Обидва скрипи використовують об'єкт GameObjectData який зберігає в собі 2 поля:

- List<Vector2> positions - позиції об'єктів
- string objectLayerName - назва шару на якому розташовані ці об'єкти

Завдяки структурі в яку організовано все ігрова середовище (рис. 2), нам не потрібно зберігати дуже багато значень для кожного об'єкта. Таким чином зменшуючи об'єм інформації, що зберігається, і швидкодіючі системи.

Робота LevelSaver здійснюється наступним зверненням. Він отримує всі об'єкти які розташовані на конкретному шарі і записує їх у json. Кожен новий запис створює окремий файл.

LevelLoader, у свою чергу, викликається тільки якщо в режимі гри вказано, що вона запускається зі збереження. У GameManager є bool поле isSave, яке відповість за статус гри. Коли це поле дорівнює true, то запускається функція, яка повністю очищає всі об'єкти з ігрової карти та заповнює її новими, зчитуючи з конкретного (вибраного гравцем) json файлу.



```
foreach (GameObjectData gameObject in gameObjects)
{
    Debug.Log("current game object:" + gameObject.objectLayerName);
    foreach (GameObject prefab in prefabsList)
    {
            // Compare prefab name with the current layer name
            if (prefab.name == gameObject.objectLayerName)
            {
            //select prefab from prefab list
            //GameObject instantiatedPrefab = Instantiate(prefab);
                GameObject emptyObject = new GameObject("EmptyObject");

                Debug.Log($"Prefab '{prefab.name}' matches the current layer name '{gameObject.objectLayerName}'.");

                foreach (UnityEngine.Vector2 vector in gameObject.positions)
                {
                    //set a new parent object - layer object
                    GameObject instantiateChilddPrefab = Instantiate(prefab);
                    instantiateChilddPrefab.transform.SetParent(emptyObject.transform, false);

                    float x = vector.x;
                    float y = vector.y;
                    //UnityEngine.Vector3 childVector = new UnityEngine.Vector3(Mathf.Round(vector.x), Mathf.Round(vector.y), 0);
                    UnityEngine.Vector3 childVector = new UnityEngine.Vector3(x, y, 0);

                    instantiateChilddPrefab.transform.localPosition = childVector;

                    //UnityEngine.Vector3 childVector3 = new UnityEngine.Vector3(-10, -10, 0);
                    //instantiatedPrefab.transform.localPosition = childVector3;


                    instantiateChilddPrefab.transform.localRotation = UnityEngine.Quaternion.identity;
                    instantiateChilddPrefab.transform.localScale = UnityEngine.Vector3.one;
                    Debug.Log("Prefab instantiated and set as 2nd child (copy) ");

                }

            //instantiatedPrefab.SetActive(false);

        }
    }

    // Set the parent of the instantiated prefab
    Debug.Log("Prefab instantiated and set as child of " + parentGameObject.name);


}
```

рис. 5

Даний алгоритм (рис. 5) виставляє нові об'єкти на вже збережені місця зберігаючи структуру, що вибудовується спочатку. Найбільшим завданням було отримати всі об'єкти префабів з папки. Рішенням було вручну додавати ті префаби, які треба зберігати. У майбутньому сподіваємося, що ми покращимо цю систему.

Так само в грі реалізована можливість видаляти збереження. Для цього в головному меню було створено ScrollView, в якому відображається актуальна інформація про те, що знаходиться в папці. Управління списком здійснюється аналогічним способом як і генерація нових об'єктів на карті, тобто спочатку всі об'єкти видаляються зі списку і завантажуються нові, актуальні. Для того щоб було зрозуміло з яким елементом гравець хоче

провзаємодіяти, кожному елементу при створенні надається індекс. При видаленні будь-якого елемента, індекси виставляються по-новому.

РОЗДІЛ 4
БАГАТОКОРИСТУВАЦЬКИЙ РЕЖИМ

Для створення мультиплеєра було вирішено використовувати бібліотеку Photon Engine, так як вона має досить великий ком'юніті та кількість доступних матеріалів. Однак найголовніша перевага – це можливість користувачам підключатися в мережі інтернет.

Photon дає вже готові класи та методи для налаштування мультиплеєра, що робить його дуже швидким у розробці. Так само весь обмін пакетами здійснюється Photon Суть роботи гравець підключається до сервера де він має можливість або створити або підключитися до кімнати, що вже існує. Для роботи необхідно 2 сцени, перша де він може ввести собі нікнейм і в цій сцені він підключається до лобі. Саме лобі є окремою сценою, на якому розташовані 2 панелі. На панелі з вибором доступних кімнат, розташований ScrollView, робота якого аналогічна з роботою назріли управління збережень, тобто очищення всіх елементів та виконання новими. Єдина відмінність це джерело інформації, якщо у першому випадку йде зчитування файлу, то в другому ми отримуємо список доступних кімнат безпосередньо від сервера. На даному етапі виявилася проблема, гравець 1 не бачив кімнати, створені гравцем 2. Виявилося, що у фотона є сервери в різних регіонах, і для коректної роботи необхідно було встановити один і той же регіон для всіх гравців.

Єдиним недоліком і лімітуючим фактором на даний момент це необхідність мати різні сцени для мультиплеєра та сінгплеєра. Що тягне за собою окреме налаштування цих сцен. Так само, одна з особливостей Photon, префаб гравця повинен перебувати в папці ресурсів, так що це

передбачає два або більше майже однакових варіантів. Однак відмінності все ж таки є, для коректної роботи та уникнення повторення коду, префаб онлайн гравця спостерігається від класу Player з override методів Start і Update. Як показано на фрагменті коду (рис. 6), у цих двох методах йде налаштування роботи камери та input гравця. Так як на сцені може бути 2 або більше об'єкта гравців, камера повинна розуміти за яким об'єктом вона повинна стежити. Аналогічна історія з гравцем input в методі update.

```
PhotonView view;
public CinemachineVirtualCamera virtualCamera;

// Start is called before the first frame update
protected override void Start()
{
    base.Start();
    view = GetComponent<PhotonView>();

    if (view.IsMine)
    {
        // Find or create the Cinemachine camera in the scene
        virtualCamera = FindObjectOfType<CinemachineVirtualCamera>();

        if (virtualCamera == null)
        {
            GameObject camObj = new GameObject("CinemachineCamera");
            virtualCamera = camObj.AddComponent<CinemachineVirtualCamera>();
        }

        // Set the camera to follow this player
        virtualCamera.Follow = transform;
        virtualCamera.LookAt = transform;
    }
}

// Update is called once per frame
protected override void Update()
{
    if (view.IsMine)
    {
        base.Update();
    }
}
```

рис. 6

У сцені з вибором доступних кімнат було реалізовано функціональність кастомізації свого персонажа. Найголовніше, що при виході з кімнати, і при приєднанні обрані персонаж залишається. Також гравець, який створив кімнату, має можливість стартувати гру, при цьому гра може запуститися за умови, що в лобі знаходиться більше одного гравця.

Так само в даній сцені було необхідно отримувати актуальну інформацію про зміну кількості доступних кімнат без прямого втручання гравця. Для вирішення цієї функціональності було переписано callback методу OnRoomListUpdate. Цей метод спрацьовує за умовчанням, коли відбуваються зміни.

```
public override void OnRoomListUpdate(List<RoomInfo> roomList)
{

    if (Time.time >= nextUpdateTime)
    {
        UpdateRoomList(roomList);
        nextUpdateTime = Time.time + timeBetweenUpdates;
    }


}
```

рис. 7

При виклику на жаль відбувався баг-метод міг викликатись два або 3 рази, так що було прийнято рішення поставити таймер який буде обмежувати рамки виклику даного методу.

ВИСНОВКИ

Під час виконання дипломної роботи було проведено ґрунтовне дослідження та реалізацію ігрового середовища, заснованого на

українській дохристиянській культурі та міфології. Основні результати такі:

1. **Гнучкість та масштабованість системи**: Використання JSON для збереження ігрового прогресу забезпечило зручність та гнучкість у зберіганні даних.

2. **Командна робота**: Впровадження елементів SCRUM з тижневими спринтами та використання дошок Kanban сприяло ефективній організації командної роботи. Парне програмування підвищило якість коду та забезпечило безперервний обмін знаннями між розробниками.

3. **Архітектурні рішення**: В процесі розробки широко застосовувалися об'єктно-орієнтовані патерни проєктування, що дозволило створити структуровану та модульну архітектуру. Принципи SOLID забезпечили високу якість коду, легкість його розширення та підтримки.

4. **Використання сучасних технологій**: Проєкт розроблявся за допомогою Unity, C#, Visual Studio, WebGL та інших інструментів. Для створення багатокористувацького режиму використовувався пакет Photon, який забезпечив зручність і ефективність реалізації мережевої гри.

5. **Розробка ігрового середовища та механік**: Проєкт ґрунтується на аналізі та розширенні ігрових механік, що базуються на популярній грі Don’t Starve. Було додано нові способи взаємодії з неігровими персонажами, розширено бойову систему та впроваджено кооперативну механіку, що дозволяє NPC допомагати гравцеві.

Загалом, виконана робота демонструє можливості створення якісного ігрового контенту з використанням сучасних технологій та методологій. Результати можуть бути застосовані для подальшого розвитку проєкту та впровадження нових ігрових механік.

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ НАВЕДЕНОЇ ЧАСТИНИ ПРОЄКТУ

№ п.п.	Ресурс	Призначення
1.	Unity; Розробник: Unity Technologies; Документація: посилання	Використано в якості ігрового рушія та середовища розробки
2.	Мова програмування C# Розробник: Microsoft Документація: посилання	Була обрана мовою програмування для реалізації ігрової логіки
3.	Visual Studio; Розробник: Microsoft Документація: посилання	Використано в якості головного інструмента редактору коду
4.	WebGL; Розробник: Khronos Group Документація: посилання	Використано для створення збірки та публікації на веб-сторінці
5.	itch.io Розробник: Leaf Corcoran Документація: посилання	Використано для публікації збірки проєкту
6.	Cinemachine; Розробник: Unity Technologies; Документація: посилання	Використано для налаштування ігрової камери
7.	Photon Engine;	Використано для

	Розробник: Exit Games; Документація: посилання	реалізації багатокористувацького режиму