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)»

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

Виконавці проекту

№	П.І.Б.	Група
1	Юркевіч Володимир Олександрович	КН-П-201
2	Кузьмін Антон Дмитрович	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

_____ Юркевіч Володимир Олександрович

Одеса – 2024

АНОТАЦІЯ

Дипломна робота присвячена аналізу та розробці ігрового середовища, проєктуванню різних ігрових механік та створенню програмної архітектури, яка дозволяє швидко додавати новий ігровий контент.

Основою для дослідження став проєкт Don’t Starve, ключовими механіками якого є дослідження ігрового світу та комбінування предметів (ремесло). Проєкт доповнено новими способами взаємодії з неігровими персонажами, зокрема розширено бойову систему та введено механіку кооперації, за якою неігрові персонажі допомагають гравцю. Додатково розроблено систему виконання завдань у ігровому середовищі за винагороду.

Тематика ігрового контенту ґрунтується на українській дохристиянській культурі та міфології. Актуальність продукту визначається зростанням інтересу до української культури як в Україні, так і за її межами, що зумовлено агресією Російської Федерації проти України.

Метою дослідження було визначення узагальнених характеристик ігрового контенту та аналіз способів їх гнучкого та швидкого інтегрування в ігрове середовище за допомогою ігрового рушія та середовища розробки Unity, а також його внутрішніх інструментів (Cinemachine, ScriptableObjects) і сторонніх пакетів (Photon).

Для розробки було обрано рушій Unity через його велику бібліотеку безкоштовних ресурсів, які є необхідними для створення повноцінного ігрового продукту, але виходять за межі компетенцій команди, наприклад, звукове супроводження.

Організація командної роботи та відстеження виконаних завдань здійснювалися за допомогою дошок Trello. У процесі розробки широко використовувалася методологія екстремального програмування, зокрема

парне програмування, що дозволило прискорити процес розробки та забезпечити обмін досвідом і знаннями серед команди програмістів.

ЗМІСТ

ВСТУП

У сучасному світі індустрія інтерактивних розваг перетворилася з локального захоплення на масовий феномен, який охоплює сотні мільйонів людей по всьому світу. Сектор продовжує динамічно розвиватися, впроваджуючи нові напрями, інновації та створюючи робочі місця. Відеоігри сьогодні демонструють свою культурну та мистецьку цінність, пропонуючи захопливі світи та неймовірні історії, доповнені цікавим ігровим процесом.

Жодна відеогра не існує без програмного коду, який оживляє її елементи. Сучасні тенденції вимагають від розробників створення архітектури, що дозволяє швидко додавати новий вміст без зміни основного коду. Це особливо актуально для ігор, які потребують довгострокової підтримки та оновлень.

Враховуючи ці вимоги, наша команда зосередилася на розробці архітектури, яка дозволяє легко інтегрувати нові елементи, забезпечуючи гравцям постійно свіжий досвід.

Цей документ включає аналіз програмної системи, проєктування архітектури, процес розробки та майбутню підтримку. Також описано проблеми, що виникли під час розробки, і шляхи їх вирішення. Додатково наведені методи організації командної роботи над проєктом.

У наступному розділі представлено технічне завдання проєкту. У роботі розглядається створення системи керування ігровим аватаром, його характеристиками, інвентарем та взаємодією з оточенням, а також поведінка неігрових персонажів і система ремесел. 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 проєкт 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

Ядро

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

Функції:

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

Гравець

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

Функції:

- Контроль аватара
- Інвентар та ремесло

Середовище

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

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)

Завдання

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

Функції:

- Створення квестів
- Відслідковування прогресу
- Винагороди за виконання завдань

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

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
ГРАВЕЦЬ ТА ВЗАЄМОДІЯ З СЕРЕДОВИЩЕМ

Клас Player відповідає за керування ігровим персонажем у проекті. Він є центральним елементом гри, забезпечуючи інтеграцію різних аспектів ігрового процесу, включаючи взаємодію з оточенням, керування здоров'ям, маною та інвентарем. Клас реалізує кілька інтерфейсів, що дозволяє йому виконувати різні дії.

Клас Player успадковує базовий клас ItemDropper і реалізує кілька інтерфейсів: IMoveable, IDamageable, IAttack, IInteract, та ICoverable. Це дозволяє об'єкту виконувати відповідні дії та забезпечує гнучкість і модульність.

Спадкування та інтерфейси

- ItemDropper: базовий клас, що дозволяє гравцю викидати предмети з інвентарю в ігрове середовище. Використання цього класу дозволяє уникнути дублювання коду.
- IMoveable: інтерфейс, що забезпечує наявність компоненту Rigidbody2D, необхідного для переміщення об’єкта в ігровому просторі. Метод Move() приймає напрямок та швидкість руху і виконує цей рух.
- ICoverable: інтерфейс, що дозволяє гравцю взаємодіяти з об’єктами, які дозволяють сховати ігрового аватара від ворожих неігрових персонажів.
- IInteract: інтерфейс, що дозволяє гравцю взаємодіяти з об’єктами ігрового середовища, які реалізують інтерфейс IInteractable.

Методи та функціонал

- Awake(): ініціалізує стан машини станів і встановлює початкові значення для здоров'я, голоду та мани. Підписується на події інвентаря та гри.

- Start(): налаштовує індикатори здоров'я, голоду та мани, а також менеджер амулетів і контролер інвентаря.
- Update(): виконує оновлення поточного стану машини станів, якщо гравець не в діалозі.
- FixedUpdate(): виконує фізичне оновлення поточного стану машини станів, якщо гравець не в діалозі.
- OnDestroy(): відписується від подій інвентаря та гри, зупиняє всі корутини.

Клас Player використовує патерн "Машина станів" для керування різними станами гравця. Кожен стан має свій клас, що реалізує специфічну логіку:

- PlayerIdleState: стан спокою, в якому гравець може керувати аватаром за допомогою клавіш WASD.
- PlayerMoveToPointState: стан, що дозволяє гравцю направити ігрового аватара в певну точку за допомогою миші.
- PlayerLockToInteractState: стан, в який гравець переходить за допомогою клавіші E. У цьому стані автоматично визначається найближчий елемент, що реалізує інтерфейс IInteractable, та ігровий аватар розпочинає рух до цього об’єкта і виконує взаємодію, щойно дійде до нього.
- PlayerLockToTargetState: стан, схожий до попереднього, але з деякими відмінностями. Цей стан потрібен для реалізації бойової системи, активується натисканням кнопки F. Замість об’єктів IInteractable шукає найближчий об’єкт IDamageable. Також у цьому стані реалізована перевірка, чи не є знайдений об’єкт союзником гравця.
- PlayerUnderCoverState: спеціальний стан гравця, що дозволяє приховати ігровий аватар від ворожих неігрових персонажів.

Умови переходу між станами обробляються в базовому класі PlayerState, що дозволяє уникнути дублювання коду в кожному конкретному стані. Цей підхід дозволяє зберегти код чистим і легко підтримуваним, розділяючи логіку різних станів на окремі класи.

Клас Player взаємодіє з контролером інвентаря через події:

- OnItemSelected(ItemSO item): обробляє вибір предмета.
- OnItemDeselected(): обробляє зняття вибраного предмета.
- OnItemPickedUp(ItemSO item, int quantity): обробляє підбір предмета.

Клас реалізує логіку здоров'я та голоду:

- ChangeHealth(float value): змінює рівень здоров'я гравця.
- ChangeHunger(float value): змінює рівень голоду.
- HungerCountdown(): корутина, що зменшує рівень голоду з часом.

Клас реалізує логіку атаки та отримання ушкоджень:

- Attack(IDamageable target): виконує атаку на ціль.
- GetDamage(float damage, GameObject attacker): обробляє отримання ушкоджень.
- CalculateDamageValue(float damage): розраховує величину ушкоджень з урахуванням амулетів.

Клас реалізує логіку використання магії:

- AddSpell(SpellSO spell): додає заклинання.
- RemoveSpell(SpellSO spell): видаляє заклинання.
- ActivateSpell(SpellSO spell): активує заклинання.

Клас реалізує різні види взаємодії з оточенням:

- Interact(IInteractable target): взаємодіє з об’єктом.
- Cover(ICover cover): ховається за об’єктом.
- LeaveCover(): залишає укриття.

Клас Player є комплексним і багатофункціональним компонентом гри, що забезпечує керування всіма аспектами ігрового персонажа. Використання патернів проектування, таких як "Машина станів", забезпечує модульність та підтримуваність коду. Реалізація різних інтерфейсів дозволяє легко розширювати функціональність класу та забезпечує гнучкість у взаємодії з іншими об'єктами ігрового світу.

РОЗДІЛ 3
ІНВЕНТАР ТА РЕМЕСЛО

Цей розділ функціональності складається з трьох частин: предмети, інвентар та ремесло.

Система предметів є основним елементом цього розділу. Кореневим елементом всіх предметів у грі виступає сценарний об’єкт, який зберігає всю інформацію про цей предмет. Ця інформація включає:

- Назву
- Інформативний та художній опис
- Рецепт створення
- Тип
- Можливі ефекти
- Зображення (спрайт)
- Максимальну кількість, що можна зберігати в одній комірці інвентарю

Всі предмети у грі поділяються на чотири типи:

- **Їжа (Food)**: предмети, які гравець може одноразово використати для відновлення показника голоду.
- **Оберіг (Amulet)**: предмети, які гравець може екіпірувати на ігрового аватара, щоб отримати покращені показники.
- **Зілля (Potions)**: предмети, які гравець може одноразово використати для відновлення одного або декількох показників.
- **Компоненти (Components)**: предмети, які не мають безпосереднього ефекту, але потрібні гравцю в системі ремесла для отримання оберегів та зіль.

Для розміщення предметів в ігровому середовищі створено клас Item, що наслідує клас UnityEngine.MonoBehaviour та реалізує інтерфейс IInteractable. Цей клас містить поля для кількості та сценарного об’єкту з інформацією про предмет. Взаємодія гравця з об'єктом реалізована через

інтерфейс IInteractable, що дозволяє гравцю додати предмет в свій інвентар. Також в класі Item створено метод InitItem, що дозволяє створювати предмети в ігровому середовищі під час виконання. За допомогою інструментів Unity створено заготовлений об’єкт (Prefab), до компонентів якого додано скрипт Item.

Інвентар реалізовано за допомогою патерну Model-View-Controller (MVC). Модель в даному випадку представлена у вигляді сценарного об’єкту, що містить перелік предметів, збережених у інвентарі, та загальної кількості комірок. Гравець може обрати певний предмет в інвентарі та використати його, натиснувши клавішу R, або викинути, натиснувши клавішу Q. Для синхронізації обраного предмета між класом-контролером та класом гравця використовується механізм подій та патерн Спостерігач. Також засобами Unity було додано анімацію відкриття та закриття панелі інвентарю.

Система ремесла реалізується через два предмети оточення, які знаходяться в окремій сцені - HideoutScene.

- EnchantmentTable: За допомогою цього об'єкта гравець має можливість комбінувати знайдені предмети та отримувати амулети.
- Cauldron: За допомогою цього об'єкта гравець може створювати зілля.

Обидва об’єкти мають в компонентах клас Crafter, який забезпечує цю функціональність. Цей клас містить перелік сценарних об’єктів, що описують рецепти, які доступні гравцю в цьому об’єкті. Сценарний об’єкт рецепту складається з переліку трьох предметів, які можна поєднати, та сценарного об’єкту предмету, що отримується в результаті.

Таким чином, система предметів, інвентарю та ремесла тісно пов'язані між собою і забезпечують основні механіки гри, дозволяючи гравцю взаємодіяти з предметами, створювати нові та ефективно керувати своїм інвентарем.

РОЗДІЛ 4
НЕІГРОВІ ПЕРСОНАЖІ

Базовий клас для всіх неігрових персонажів (NPC) називається NPCBase. Як і клас гравця, цей клас спадкується від класу ItemDropper та реалізує ті ж самі інтерфейси, за виключенням IInteract та ICover.

Також цей клас має Поле NPCDescriptionSO, яке містить сценарний об’єкт з описом конкретного НІПа. Цей об'єкт включає:

- Базову кількість очок здоров’я.
- Базову кількість очок шкоди.
- Тип.
- Ставлення до гравця.
- Перелік сценарних об’єктів, що описують НІПів для атаки.
- Перелік сценарних об’єктів з описами предметів, які гравець може отримати за перемоги над НІПом та шанс їх випадання.
- Назву.
- Інформативний та художній описи.
- Зображення (спрайт).

Для неігрових персонажів також реалізована окрема машина станів. Особливістю цієї машини станів є винесення обробки стану в сценарні об’єкти. Це дозволяє розділити конкретних неігрових персонажів і логіку станів, що призводить до уникнення дублювання коду в ситуаціях, коли обробка певного стану різних НІПів не відрізняється. У разі необхідності, така структура дозволяє легко доповнити нову обробку стану.

Створено також спеціальний клас BefriendableNPC, який є нащадком базового класу NPCBase. Цей клас розширює базовий функціонал особливим станом, який доступний тільки для тих неігрових персонажів, яких гравець може залучити собі на допомогу. Умови для цього є різними для кожного конкретного НІПа. В цьому додатковому стані НІП починає допомагати гравцю.

Базовий клас NPCBase та його похідні дозволяють ефективно керувати поведінкою неігрових персонажів в грі, забезпечуючи різноманітні взаємодії з гравцем та іншими елементами ігрового середовища. Використання машини станів та сценарних об’єктів дозволяє легко розширювати та модифікувати поведінку НІПів, що забезпечує гнучкість та масштабованість системи.

ВИСНОВКИ

У процесі виконання дипломної роботи було проведено всебічне дослідження та реалізацію ігрового середовища, що ґрунтується на українській дохристиянській культурі та міфології. Основними результатами є наступні:

1. **Розробка ігрового середовища та механік**: В основі проєкту лежить аналіз і розширення ігрових механік, базуючись на популярній грі Don’t Starve. Додано нові способи взаємодії з неігровими персонажами, розширено бойову систему та введено механіку кооперації, що дозволяє НІПам допомагати гравцю.
2. **Використання сучасних технологій**: Розробка проєкту здійснювалася з використанням Unity, C#, Visual Studio, WebGL та інших інструментів. Для створення багатокористувацького режиму застосовано пакет Photon, що забезпечив зручність і ефективність реалізації мережевої гри.
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. **Командна робота**: Впровадження елементів SCRUM з тижневими спринтами та використання дошок Kanban сприяло ефективній організації командної роботи. Парне програмування дозволило підвищити якість коду та забезпечити безперервний обмін знаннями між розробниками.
5. **Гнучкість та масштабованість системи**: Використання машини станів та сценарних об’єктів дозволило легко розширювати та модифікувати поведінку НІПів, що забезпечує гнучкість та

масштабованість системи. Застосування JSON для збереження ігрового прогресу забезпечило зручність та гнучкість у зберіганні даних.

В цілому, проведена робота демонструє можливості створення якісного ігрового контенту з використанням сучасних технологій та методологій. Результати можуть бути використані 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