



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

Випускна кваліфікаційна робота бакалавра

**Моделювання між об'єктної взаємодії обмеженої екосистеми з не
обмеженим ресурсом (на прикладі Vampire Survivors)**

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

Виконавці проекту

№	П.І.Б.	Група
1	Гичак Данило Миколайович	КН-П-212
2	Власіхін Іван Сергійович	КН-П-212

Автор звіту

_____Власіхін Іван Сергійович

Ментор	Доцент кафедри ІТ та ФП, к.т.н., Масрі М.М.
--------	--

Одеса – 2025

АНОТАЦІЯ

Дипломна робота присвячена створенню гри у жанрі action-survivor з елементами RPG, натхненої механіками проєкту *Vampire Survivors*, але з унікальною архітектурою та реалізацією, орієнтованою на гнучкість, масштабованість та легкість розширення. Основною метою роботи стало проєктування програмної структури, яка дозволяє швидко та безперешкодно додавати нові ігрові механіки, персонажів, вміння, ворогів та інші елементи контенту без необхідності змінювати існуючий код.

Всі основні елементи гри реалізовано вручну: логіка персонажа, бойова система, прокачування, механіка ворогів, хвилі появи, звуковий супровід та інтерактивний інтерфейс. Особливу увагу приділено архітектурним рішенням — було використано об'єктно-орієнтоване програмування та шаблони проєктування, такі як Singleton, Strategy, Component та частково Observer, що забезпечило структурованість і модульність коду. Це дало змогу забезпечити просту інтеграцію нового контенту навіть розробникам, які вперше ознайомлюються з проєктом.

Графічна складова виконана на основі безкоштовних відкритих ресурсів та нейромережевих генераторів. У зв'язку з відсутністю дизайнера у команді, було прийнято рішення використовувати готові спрайти та елементи інтерфейсу з відкритих джерел.

Проєкт реалізовано з використанням рушія Unity та мови програмування C#. Вся логіка, включно з анімаціями, рухом камери та освітленням, написана вручну без використання стандартних шаблонів або візуальних тулів Unity. Звукові ефекти керуються через окремий аудіоконтролер, що дозволяє редагувати звуки без зміни коду.

Командна робота організовувалася через онлайн-платформу Discord, а керування завданнями — за допомогою канбан-дошки. Такий підхід забезпечив чіткий розподіл ролей, прозорість у виконанні завдань та ефективну внутрішню комунікацію.

Результатом роботи став функціональний, гнучкий і зручний для розширення

ігровий продукт, який може слугувати як основа для подальшої розробки повноцінного комерційного проєкту або освітнього інструмента з програмування ігрових механік.

ЗМІСТ

АНОТАЦІЯ	2
ЗМІСТ	4
ВСТУП	5
Загальний опис.	6
Призначення.	7
Вимоги до функціоналу.	8
Архітектура.	9
Технології.	10
Етапи розробки.	10
Вимоги до якості.	10
РОЗДІЛ 1	11
РОЗДІЛ 2	14
РОЗДІЛ 3	17
РОЗДІЛ 4	20
РОЗДІЛ 5	25
РОЗДІЛ 6	27
ВИСНОВКИ	28
ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ НАВЕДЕНОЇ ЧАСТИНИ ПРОЄКТУ	
30	

ВСТУП

У ХХІ столітті відеоігри зайняли чільне місце серед провідних галузей цифрових технологій, перетворившись не лише на джерело розваг, але й на важливий напрямок для вивчення, творчості та професійної реалізації. Сучасні ігри вражають своєю складністю, глибиною і технологічним рівнем реалізації — від графіки до сценаріїв, від штучного інтелекту до мережевої взаємодії між гравцями. У зв'язку з цим питання побудови надійної, гнучкої та масштабованої архітектури програмного забезпечення набуває особливої актуальності.

Розробка навіть невеликого ігрового проєкту вимагає від команди не тільки знання технічних засобів, а й уміння проєктувати систему таким чином, щоб її можна було безболісно розширювати: додавати нових ворогів, нові здібності персонажів, інтерфейсні елементи, механіки та візуальні ефекти. Одним із ключових викликів у таких проєктах є уникнення жорсткої залежності між модулями та дотримання принципів модульності та повторного використання коду.

Дипломна робота зосереджена на розробці гри у жанрі *action-survivor*, натхненої *Vampire Survivors*, з акцентом на архітектурні рішення та чистоту коду. У процесі створення застосовано патерни об'єктно-орієнтованого програмування, ручну реалізацію усіх ключових систем (рух, атака, система ворогів, прокачування тощо) та активне використання можливостей рушія Unity. Основним завданням було зробити систему відкритою для подальшого розширення та модифікацій навіть без глибокого занурення в архітектуру.

У роботі описано ключові етапи створення проєкту: проєктування системи керування гравцем і ворогами, реалізацію бойових механік, організацію інтерфейсу користувача, підключення аудіо та оптимізацію процесу командної роботи. Також розглянуто труднощі, з якими зіткнулась команда, та шляхи їх подолання.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ

Проектування модульної ігрової архітектури з динамічним розширенням функціональності (на прикладі гри Vampire Survival)

Загальний опис

Завданням програмного забезпечення (ПЗ) є створення інтерактивної однокористувацької аркадної гри у жанрі виживання з елементами RPG-прокачки, побудованої за принципом поступового ускладнення ігрового процесу. Гравець керує персонажем, який автоматично атакує хвилі ворогів, накопичує досвід, вибирає покращення, намагаючись вижити якомога довше.

Гра створюється на рушії Unity з використанням C# і використовує хмарну платформу PlayFab для збереження прогресу гравця, авторизації, ведення таблиці рекордів та зберігання валюти (монет), заробленої під час забігів.

Основною функцією системи є моделювання динамічної людино-машинної взаємодії в умовах високої інтенсивності, де ігрове середовище реагує на дії гравця через спаун ворогів, підвищення складності, надання винагород та розвиток персонажа. Особливу увагу приділено плавності керування, інтуїтивності інтерфейсу та збереженню мотивації гравця через прогресію та досягнення.

Гра включає кілька взаємопов'язаних підсистем, які формують єдиний цілісний геймплей:

Система бою – автоматична атака ворогів на основі певного інтервалу часу, який можна покращувати. Вороги з'являються хвилями, мають різну швидкість, здоров'я і типи поведінки. Перемога над ворогами приносить досвід та ресурси.

Система прогресії персонажа – гравець підвищує рівень, обираючи покращення (наприклад, збільшення дальності атаки, додаткову шкоду, нові типи зброї, пасивні ефекти тощо). Це стимулює формування індивідуальної тактики кожного забігу.

Система монетизації через монети – за кожний завершений забіг (незалежно від виграшу чи програшу) гравець отримує певну кількість монет. Монети

служують внутрішньоігровою валютою та зберігаються у профілі PlayFab.

Система рекордів та змагання – у профілі PlayFab зберігаються дані про найкращі результати гравця. Також реалізована таблиця лідерів, що стимулює конкуренцію між гравцями.

Система вибору контенту – гра пропонує 2 ігрові локації (із власним візуальним оформленням) та 2 персонажі на вибір. Кожен персонаж має унікальні базові характеристики (наприклад, більшу швидкість або більше здоров'я).

Інтерфейс користувача – реалізовано головне меню, інтерфейс бою, меню вибору покращень, екран результатів. Усі компоненти підтримують інтерактивність, зворотній зв'язок та адаптивність.

Система авторизації через PlayFab – користувач має можливість створити акаунт або увійти через вже існуючий. Дані зберігаються у хмарі та дозволяють гравцеві зберігати прогрес, навіть якщо він змінює пристрій.

Таким чином, ПЗ виконує роль інтерактивної системи, яка керує як ігровим процесом у режимі реального часу, так і довготривалим прогресом гравця. Воно формує мотиваційний цикл, де гравець щоразу намагається досягти кращого результату, отримати більше монет, відкрити нові можливості та поліпшити свої попередні досягнення, зберігаючи при цьому індивідуальний стиль гри. Гнучкість системи дозволяє розширювати її у майбутньому: додавати нові персонажі, карти, типи ворогів та внутрішньоігрові цілі.

Призначення

Гра призначена для платформи ПК (Windows). Вона орієнтована на фанатів аркад, rogue-lite, auto-attack-ігор, vampires-like, а також на досвід користувача з елементами RPG-прокачки та колекціонування досягнень.

Вимоги до функціоналу

Головне меню

- Запуск гри
- Вибір персонажа (2 варіанти)
- Вибір локації (2 доступні мапи)
- Авторизація через PlayFab (реєстрація/вхід)
- Перегляд таблиці рекордів
- Налаштування (звук)

Системи взаємодії

- **Головний ігровий процес**

Гравець керує персонажем у режимі top-down. Персонаж автоматично атакує ворогів. Гравець ухиляється, збирає досвід, монети, та виживає максимально довго.

- **Система хвиль**

Вороги з'являються групами з поступовим збільшенням складності. Наприкінці — поява босів.

- **Система досвіду та прокачки**

За кожного вбитого ворога гравець отримує досвід. При досягненні нового рівня — вибір одного з трьох покращень (швидкість, атака, дальність, броня, нова зброя тощо).

- **Система монет**

Після завершення забігу гравець отримує монети. Їх кількість залежить від тривалості гри, кількості вбитих ворогів та досягнутого рівня.

Монети зберігаються в базі даних PlayFab та використовуються для подальших покупок (опційно).

- **Система рекордів**

Максимальний час виживання та кількість вбитих ворогів зберігаються у профілі PlayFab. Реалізується таблиця лідерів (топ-10).

Архітектура

Основні компоненти

- **Ядро**

- Головний ігровий цикл
- Менеджер сцен та станів
- Завантаження/збереження через PlayFab

- **Гравець**

- Вибір персонажа
- Автоматична атака
- Прокачка та збір ресурсів
- Життя, шкода, апгрейди

- **Середовище**

- Генерація карт (2 варіанти: "ліс" і "пустка")
- Оточення: декоративні об'єкти та перешкоди

- **Вороги**

- Різні типи: звичайні, елітні, боси
- ШІ-поведінка: прямий рух до гравця
- Система зростаючої складності

- **PlayFab**

- Авторизація
- Збереження монет, рекордів, прогресу
- Таблиця лідерів

Інтерфейс користувача (UI)

- Головне меню
- Панель гравця (здоров'я, досвід, таймер, монети)
- Меню покращень

- Екран завершення гри (результати, отримані монети)

Технології

- Движок: Unity
- Мова програмування: C#
- База даних: PlayFab SDK
- Графіка: піксель-арт, спрайти
- Звук: безкоштовні звукові ефекти (атаки, смерть, апгрейд)
- Публікація: ПК

Етапи розробки

Етап	Тривалість	Опис
	Ь	
1. Архітектура та каркас	1–2 тижні	Менеджери сцен, UI, PlayFab
2. Ігрова логіка	2–3 тижні	Управління, вороги, атаки
3. Система прокачки	1–2 тижні	Апгрейди, досвід
4. Монети та PlayFab	1–2 тижні	Авторизація, збереження
5. Кінцева збірка	1 тиждень	Баланс, оптимізація, тестування

Вимоги до якості:

- FPS не нижче 30 на середніх ПК
- Відсутність критичних помилок
- Правильна інтеграція з PlayFab
- Повноцінна ігрова петля "виживання–нагорода–повтор"

РОЗДІЛ 1

ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

Після детального аналізу вимог технічного завдання команда розробників прийняла рішення використовувати ігровий рушій Unity, оскільки він вирізняється високою продуктивністю, зручністю у використанні, широкими можливостями для розробки та підтримкою мови програмування C#.

Основа програмної архітектури складається з наступних компонентів:

- **Game Manager:** Основні механізми гри, включаючи таймер гри, керування станами гри, завантаження ігрових сцен, виклик різних вікон тощо.
- **Player:** Всі аспекти, що стосуються гравця, включаючи управління аватаром, розвиток, візуальна частина тощо.
- **World Manager:** Система, що керує та містить декілька підсистем таких як: ігрова камера, створенням ворогів на карті, та розташуванням анімаційних об'єктів в світі.
- **Enemy:** Логіка та поведінка всіх неігрових персонажів, ворогів.
- **Audio:** Всі елементи звукового супроводу, включаючи музику, звукові ефекти інтерфейсу та гравця тощо.
- **UI:** Елементи інтерфейсу користувача, включаючи головне меню, пауза , вибір мапи тощо.
- **Design:** Частина **Design** охоплює створення та налаштування інтерфейсних елементів для головного меню, розвитку головного героя, а також внутрішньоігрових меню — меню паузи, меню гравця та панелі показників.

Оскільки команда не мала у своєму складі графічного дизайнера, для реалізації візуальної складової проєкту були використані графічні ресурси з відкритим доступом. Це стосується як інтерфейсних елементів, так і зображень (спрайтів) для аватарів гравців, неігрових персонажів, об'єктів ігрового середовища, а також базових елементів брендингу (логотип, шрифт тощо).

Особлива увага приділялася зручності та інтуїтивності інтерфейсу, щоб забезпечити комфортну взаємодію користувача з грою на всіх етапах геймплею.

Програмну частину проєкту було повністю реалізовано з використанням мови програмування C# та Unity Scripting API. Основна функціональність розроблялася власноруч, без застосування сторонніх шаблонів або готових бібліотек, таких як Asset Store-пакети для управління камерою, освітленням або геймплейними елементами. Усі ключові компоненти, включаючи логіку руху камери, обробку освітлення, управління персонажами та внутрішні ігрові системи, були написані вручну, що дозволило досягти високого рівня контролю над архітектурою проєкту.

Разом з тим, у роботі використовувались базові компоненти Unity, зокрема стандартні об'єкти сцени, системи колізій, UI-система та інструменти роботи з анімацією. Для організації та збереження даних активно застосовувалися Scriptable Objects та prefabs, які значно спростили процес додавання нового контенту.

Під час проєктування програмної архітектури активно застосовувалися об'єктно-орієнтовані принципи та патерни проєктування, зокрема Singleton, Strategy, Component та частково Observer. Такі рішення дозволили створити гнучку, масштабовану й легко підтримувану систему.

Ключові ігрові компоненти, як-от керування гравцем або менеджер гри, реалізовані через патерн Singleton, що забезпечує глобальний доступ та гарантує єдиний екземпляр класу. Система скілів побудована за принципом патерну Strategy: кожен скіл є окремим класом зі своєю поведінкою, що дозволяє легко додавати нові типи атак (наприклад, ближній бій, ударна атака чи зона дії) без зміни існуючого коду.

Архітектура також використовує компонентний підхід, властивий Unity, що дозволяє чітко розділити відповідальність між логікою об'єкта та його візуальним або фізичним представленням. Частково реалізовано

патерн Observer для взаємодії між системами, наприклад, оновлення інтерфейсу користувача при зміні здоров'я або досвіду гравця.

Завдяки такому підходу будь-який розробник, навіть без попереднього ознайомлення з усім кодом, зможе швидко додати нову пасивну здатність, ворога або інший ігровий елемент. Проєкт спроектовано з акцентом на розширюваність, що робить його зручним для подальшої розробки й масштабування.

Командна робота була організована з використанням методології Kanban, що забезпечувало прозорість та ефективність управління завданнями. Для візуалізації робочого процесу застосовувалась Kanban-дошка, яка дозволяла відстежувати статус виконання задач у реальному часі та координувати діяльність команди.

Для оперативної комунікації та обміну інформацією команда використовувала платформу Discord, що сприяло швидкому вирішенню питань і підтримці злагодженості в роботі.

Вихідний код розробленого проєкту розміщено у публічному репозиторії на платформі GitHub, що забезпечує зручне відстеження змін, колаборацію з іншими розробниками та контроль версій. Готова користувацька версія гри доступна на веб-сервісі itch.io.

РОЗДІЛ 2

ГРАВЕЦЬ ТА ВЗАЄМОДІЯ З СЕРЕДОВИЩЕМ

Клас `PlayerController`:

`PlayerController` відповідає за управління ігровим персонажем у проєкті та є центральним елементом геймплею. Він забезпечує інтеграцію різних аспектів ігрового процесу: рух персонажа, керування здоров'ям, накопичення досвіду, застосування пасивних характеристик і навичок, взаємодію з інтерфейсом та отримання пошкоджень.

Для забезпечення глобального доступу та унікальності екземпляра класу використовується патерн `Singleton` через статичний інстанс `PlayerController.instance`. Це дозволяє іншим компонентам проєкту легко звертатися до керування гравцем, не створюючи зайвих посилань.

Об'єктно-орієнтований підхід та патерни

Клас побудований з використанням принципів ООП: логіка поділена на окремі методи, використовується інкапсуляція даних і відповідальностей. Для роботи з ігровими навичками застосовується патерн `Strategy` — кожен скілл представлений як окремий клас-нащадок базового класу `Skill` з власною поведінкою і префабом, що дозволяє легко додавати нові типи навичок без зміни коду `PlayerController`.

Компонентний підхід `Unity` реалізується через розділення логіки та візуального представлення: наприклад, логіка руху та поведінки гравця відокремлена від візуального контролера анімації, а взаємодія з UI реалізована через окремий контролер `UIController.instance`.

Частково застосовується патерн `Observer` — при зміні здоров'я або досвіду гравця викликаються методи оновлення UI, що дозволяє підтримувати синхронізацію між логікою гри та інтерфейсом.

Основний функціонал

Рух персонажа: обробляється у методах Update() та FixedUpdate() з урахуванням введення користувача. Напрямок руху зберігається у змінній playerMoveDirection. Анімація руху та віддзеркалення спрайта також оновлюються залежно від напрямку руху.

Здоров'я та імунітет: керування поточним і максимальним здоров'ям гравця, імплементація короткочасного імунітету після отримання пошкоджень з візуальним ефектом мерехтіння.

Досвід і рівні: система накопичення досвіду, перехід на наступний рівень із викликом вибору навичок або пасивних статів через UI.

Пасивні характеристики: реалізовано набір параметрів, які модифікують базові властивості персонажа (захист, множники шкоди і швидкості атаки, шанс критичного удару), що дозволяє гнучко налаштовувати ігровий баланс.

Взаємодія з UI: усі зміни стану здоров'я, досвіду, рівня миттєво відображаються у відповідних елементах інтерфейсу за допомогою глобального інстанса UIController.

Розширюваність та підтримуваність

Архітектура класу спроектована з акцентом на легкість розширення:

Додавання нових скілів або пасивних здібностей можливе без модифікації основного коду класу — достатньо створити новий клас-нащадок Skill або структуру PassiveStat і додати їх у відповідні колекції.

Завдяки Singleton-інстансам (наприклад, `GameManager.instance`, `UIController.instance`) забезпечується зручний доступ до загальних ресурсів і сервісів, що полегшує інтеграцію різних компонентів.

РОЗДІЛ 3

Ігровий дизайн

Ігровий дизайн є одним із ключових аспектів розробки будь-якої відеогри, оскільки саме він формує взаємодію гравця з віртуальним світом, його візуальне сприйняття та загальний геймплейний досвід. У цьому розділі детально описані основні компоненти ігрового дизайну проєкту: інтерфейс користувача (UI), ігровий всесвіт, а також дизайн і анімація ворогів та головного персонажа. Всі ці елементи були створені з урахуванням можливостей команди, обмежень часу, а також доступних ресурсів.

Ігровий світ

Ігровий світ реалізований у вигляді двовимірної (2D) площини, що є класичним підходом для ігор жанру survival та action. Для формування візуального оформлення рівнів використовувалася палетка тайлів — набір невеликих графічних елементів, які компонуються у суцільний ігровий ландшафт. Цей метод дозволяє створювати великі й різноманітні ігрові локації за допомогою повторного використання базових тайлів, що істотно знижує обсяг необхідних ресурсів і спрощує дизайн рівнів.

Для розробки палетки тайлів було використано тайлсети, отримані з відкритих джерел в Інтернеті, які відповідають загальному стилю гри. Такий підхід обумовлений відсутністю власних художніх ресурсів у команді, але при цьому дозволив забезпечити цілісний і привабливий вигляд світу. Завдяки використанню тайлів, реалізовано різні типи поверхонь — від трав'янистих полів до кам'янистих ущелин — що створює відчуття живого, насиченого середовища.

Особливу увагу було приділено технічній стороні: усі тайли ретельно організовані у тайлмапи Unity, що забезпечує легкість редагування рівнів та швидке масштабування світу у разі необхідності. Крім того, такий підхід

відкриває широкі можливості для подальшого розвитку — додавання нових тайлів, створення складніших локацій або інтеграція процедурної генерації.

Інтерфейс користувача (UI)

Інтерфейс користувача є невід’ємною частиною ігрового досвіду, оскільки він безпосередньо впливає на зручність і комфорт взаємодії гравця з грою. В розробці UI було зроблено основний акцент на простоту, інтуїтивність і мінімалістичність, що відповідає жанровим вимогам survival і дозволяє не відволікати користувача від ігрового процесу.

Всі елементи інтерфейсу, такі як кнопки, панелі, індикатори здоров’я і досвіду, створювалися вручну в середовищі Unity. Однак через відсутність у команді дизайнера графічні ресурси для інтерфейсу були запозичені з відкритих джерел. Це дозволило швидко сформувати функціональний та привабливий UI без втрати якості.

Особливою рисою інтерфейсу є використання анімацій відкриття та закриття панелей, що створює більш динамічний та живий вигляд. Для анімації застосовувався стандартний інструментарій Unity Animator, що спрощує подальше редагування та масштабування анімаційних ефектів.

Ще одним цікавим аспектом є застосування фонових зображень в меню, яке було створено за допомогою нейронних мереж. Використання штучного інтелекту дозволило отримати унікальні арт-образи, які не повторюються в інших проектах, надаючи грі сучасний і свіжий візуальний стиль.

Дизайн персонажів і ворогів

Головний персонаж та вороги представлені у вигляді спрайтів, які були знайдені на відкритих ресурсах і ретельно підібрані відповідно до загальної естетики гри. Відсутність власного художника в команді змусила звернутися до готових рішень, що водночас дозволило зекономити час та ресурси.

Для реалізації рухів і анімацій персонажів застосовано Unity Animator —

потужний інструмент, який дозволяє створювати плавні переходи між різними анімаційними станами. Це надало персонажам живості і динамічності, що дуже важливо для підтримки ігрової атмосфери.

Ворогів у грі декілька типів, кожен із яких має унікальний набір анімацій та поведінкових моделей, реалізованих через окремі префаби і скрипти. Такий підхід дозволяє легко додавати нових ворогів, змінювати їх поведінку або створювати складніші моделі без необхідності переписувати існуючий код.

РОЗДІЛ 4

Система розвитку персонажа та прогресії

Загальна концепція системи

Розроблена система розвитку персонажа базується на модульній архітектурі з використанням сучасних патернів проектування програмного забезпечення. Система забезпечує гнучке та масштабоване рішення для керування прогресією гравця, включаючи активні навички та пасивні характеристики.

Архітектурні рішення та патерни проектування

1. Singleton Pattern (Одинак)

Реалізовано для забезпечення глобального доступу до ключових компонентів гри:

- **PlayerController.instance** - централізоване керування станом персонажа
- **GameManager.instance** - глобальне управління ігровим процесом
- **UpgradeManager.instance** - координація системи покращень

Переваги реалізації:

- Гарантія єдиності екземпляра критично важливих компонентів
- Спрощений доступ до глобального стану з будь-якої частини коду
- Централізоване керування ресурсами та конфігурацією

2. Strategy Pattern (Стратегія)

Використовується для реалізації різних типів навичок через наслідування від базового класу Skill:

- **AreaSkills** - навички зонального ураження
- **MeleeSkills** - навички ближнього бою
- **StrikeSkills** - навички цільового ураження

Архітектурна структура:

Skill (базовий клас)

└─ AreaSkills

└─ MeleeSkills

└─ StrikeSkills

Переваги підходу:

- Легке додавання нових типів навичок без модифікації існуючого коду
- Інкапсуляція специфічної логіки кожного типу навички
- Уніфікований інтерфейс для всіх типів навичок

3. Component Pattern (Компонентний підхід)

Реалізовано через розділення відповідальності між логікою навички та її візуальним представленням:

- **Skill класи** - містять логіку активації та параметри
- **SkillPrefab класи** - відповідають за поведінку префабів та взаємодію з ігровим світом

Структура взаємодії:

MeleeSkills (логіка) → MeleeSkillsPrefab (представлення)

AreaSkills (логіка) → AreaSkillsPrefab (представлення)

StrikeSkills (логіка) → StrikeSkillsPrefab (представлення)

4. Factory Pattern (Фабрика)

Впроваджено для створення покращень персонажа:

- **PassiveStatFactory** - генерація пасивних характеристик
- **SkillUpgradeFactory** - створення покращень навичок

Функціональність фабрик:

- Інкапсуляція логіки створення складних об'єктів
- Централізована конфігурація доступних покращень
- Гнучкий алгоритм вибору покращень з урахуванням поточного стану персонажа

5. Command Pattern (Команда)

Реалізовано через інтерфейс IUpgrade для уніфікації всіх типів покращень:

```
public interface IUpgrade
```

```
{
```

```
    string GetName();
```

```
    string GetDescription();
```

```
Sprite GetIcon();  
void Apply(PlayerController player);  
}
```

Переваги реалізації:

- Єдиний інтерфейс для всіх типів покращень
- Можливість розширення функціоналу (відміна дій, логування тощо)
- Спрощення логіки вибору та застосування покращень

6. Adapter Pattern (Адаптер)

Використовується для інтеграції існуючих навичок у нову систему покращень:

```
public class SkillUpgradeAdapter : IUpgrade  
{  
    private Skill skill;  
}
```

Система пасивних характеристик

Типи пасивних характеристик:

1. **Health** - збільшення максимального здоров'я
2. **Speed** - підвищення швидкості пересування
3. **Defense** - збільшення захисту від урону
4. **Damage** - множник базового урону
5. **AttackSpeed** - швидкість атаки
6. **CritChance** - шанс критичного удару
7. **Experience** - множник отримуваного досвіду
8. **Regeneration** - регенерація здоров'я

Масштабування характеристик:

Система включає прогресивне масштабування, де кожен наступний рівень покращення дає збільшений ефект:

```
private float GetScalingMultiplier()  
{  
    return 1f + (currentLevel * 0.1f); // +10% за кожен рівень
```

}

Механіка прогресії персонажа

Система рівнів:

- **Динамічне обчислення досвіду** для наступного рівня
- **Прогресивне зростання вимог** до досвіду
- **Максимальний рівень** з можливістю налаштування

Логіка вибору покращень:

Рівень 1, 3, 5, 7... → Активні навички

Рівень 2, 4, 6, 8... → Пасивні характеристики

Переваги архітектурного рішення

Масштабованість:

- **Легке додавання нових навичок** через наслідування від базового класу
- **Просте розширення пасивних характеристик** через enum та switch-case
- **Гнучка система фабрик** для генерації контенту

Підтримуваність:

- **Чітке розділення відповідальності** між компонентами
- **Слабка зв'язаність** модулів
- **Уніфіковані інтерфейси** для різних типів функціональності

Гнучкість:

- **Конфігурація через Inspector Unity** без зміни коду
- **Динамічна зміна параметрів** під час виконання
- **Модульна архітектура** дозволяє незалежну розробку компонентів

Розширюваність:

- **Додавання нових типів покращень** через інтерфейс IUpgrade
- **Створення нових фабрик** для різних типів контенту
- **Інтеграція з існуючими системами** через адаптери

Технічна реалізація критичних аспектів

Оптимізація продуктивності:

- **Object Pooling** для префабів навичок
- **Кешування компонентів** для зменшення викликів GetComponent
- **Ефективне управління колекціями** з використанням List<T>

Обробка урону з пасивними характеристиками:

```
float baseDamage = skill.stats[skill.skillLevel].damage;
```

```
float finalDamage = baseDamage * PlayerController.instance.damageMultiplier;
```

```
if (Random.value < PlayerController.instance.critChance)
```

```
{
```

```
    finalDamage *= 2f;
```

```
}
```

Система імунітету та захисту:

```
float finalDamage = Mathf.Max(1f, damage - defense);
```

Висновки

Розроблена архітектура системи розвитку персонажа демонструє ефективне використання сучасних патернів проектування для створення гнучкого, масштабованого та підтримуваного рішення. Система забезпечує:

1. **Модульність** - кожен компонент має чітко визначені обов'язки
2. **Розширюваність** - легке додавання нового контенту без модифікації існуючого коду
3. **Підтримуваність** - зрозуміла структура та слабка зв'язаність компонентів
4. **Продуктивність** - оптимізовані алгоритми та ефективне використання ресурсів

Дана архітектура може служити основою для розробки складних RPG-систем та легко адаптуватися під різні ігрові жанри, що підтверджує її універсальність та якість технічного рішення.

РОЗДІЛ 5

Вороги

Система ворогів у грі була реалізована повністю вручну, без використання сторонніх або вбудованих шаблонів штучного інтелекту. Такий підхід дозволив досягти максимальної гнучкості: додавання нових ворогів, зміна їх поведінки або характеристик відбувається швидко, без необхідності змінювати вже написаний код. Усі властивості, включаючи швидкість, здоров'я, силу атаки та досвід, який отримує гравець після знищення ворога, винесені в параметри, доступні для редагування безпосередньо в інспекторі Unity.

Типи ворогів

У грі передбачено два основних типи ворогів:

- **Звичайні вороги** – з'являються постійно впродовж ігрового процесу хвилями, мають базовий рівень складності;
- **Бос** – ворог з підвищеними показниками здоров'я, шкоди та швидкості, який з'являється на більш пізніх етапах гри як виклик для гравця.

Поведінка ворогів

Логіка руху та поведінки ворогів реалізована в класі `EnemyScript`. У методі `FixedUpdate()` кожен ворог постійно визначає координати головного героя та рухається у його напрямку. При цьому враховується фліп спрайту в залежності від сторони наближення, що створює природну анімацію переслідування.

Система нанесення шкоди реалізована через перевірку зіткнення з гравцем (`OnCollisionStay2D`). У разі контакту з гравцем ворог завдає фіксованої кількості шкоди, враховуючи показники захисту, реалізовані в `PlayerController`.

Після отримання шкоди ворог "відкидається" на короткий час, що досягається через зміну знаку швидкості руху на певну кількість кадрів. Це додає грі тактильності та чіткого зворотного зв'язку в бою.

При знищенні ворог:

- Відтворює ефект знищення (`destroEffect`);
- Видає певну кількість досвіду гравцю;
- Відтворює звуковий ефект смерті;
- Видаляється зі сцени.

Характеристики ворога

Кожен ворог має набір основних параметрів, які легко змінюються в інспекторі:

- `moveSpeed` — швидкість пересування;
- `damage` — кількість шкоди при зіткненні;

- health — загальне здоров'я ворога;
- experienceToGive — досвід, який отримує гравець за перемогу;
- pushTime — тривалість відкату після удару.

Завдяки такій структурі, додавання нового типу ворога з іншими характеристиками займає мінімум часу і не вимагає змін у логіці гри.

Система появи ворогів

Окремим скриптом EnemySpawnerScript реалізовано хвильову систему появи ворогів. Вона заснована на серії структурованих об'єктів Wave, кожен з яких містить:

- Префаб ворога;
- Час до наступного спавну (spawnInterval);
- Загальну кількість ворогів у хвилі (enemiesPerWave);
- Поточний прогрес появи (spawnedEnemyCount).

Завдяки методам Update() та SpawnEnemy(), гра поступово збільшує складність, скорочуючи інтервали між хвилями та змінюючи кількість ворогів. Це забезпечує динаміку і зростаючий виклик для гравця. Всі вороги з'являються у випадковій точці за межами поля зору гравця (RandomSpawnPoint()), що дозволяє уникнути "телепортації" перед очима та робить гру більш реалістичною.

Гнучкість системи

Оскільки всі елементи логіки — як ворогів, так і їх появи — були реалізовані вручну, система максимально адаптована до майбутніх змін:

- Нового ворога можна додати за кілька хвилин, скопіювавши існуючий префаб та змінивши його параметри.
- Легко змінювати темп і складність гри за рахунок правок у хвилях.
- Уся логіка інкапсульована та добре відокремлена: скрипти ворога не залежать від внутрішньої реалізації інших частин гри.

РОЗДІЛ 6

Аудіо

Аудіосупровід є важливою складовою будь-якої гри, оскільки саме він значною мірою формує атмосферу, підсилює емоційне сприйняття подій та надає гравцю зворотній зв'язок у процесі взаємодії з ігровим світом.

У рамках реалізації звукової системи в проєкті було прийнято рішення використовувати **звукові ефекти з відкритих джерел**, що дозволило швидко наповнити гру різноманітними звуками без залучення сторонніх звукорежисерів або платних бібліотек. Усі ефекти були ретельно підібрані відповідно до дій гравця та подій у грі: атака, отримання шкоди, вибір навичок, натискання кнопок, пауза тощо.

Центральним елементом аудіосистеми є спеціально створений **клас `AudioController`**, який відповідає за збереження, організацію та відтворення звуків у будь-який момент ігрового процесу. Клас реалізований як синглтон (instance), що забезпечує глобальний доступ до його методів з будь-якого іншого класу гри.

Відтворення звуків

Для відтворення звуків у класі передбачено два основні методи:

- `PlaySound(AudioSource sound)` — стандартне відтворення звуку;
- `PlayModifiedSound(AudioSource sound)` — відтворення з випадковою зміною pitch (висоти звуку), що створює різноманітність, особливо ефективно при повторюваних діях, як-от удари чи звуки смерті ворогів.

Ці методи викликаються у відповідних місцях гри: при взаємодії гравця з інтерфейсом, при атаках, втраті здоров'я тощо. Завдяки цьому, аудіо завжди синхронізовано з діями та подіями в грі, підвищуючи рівень занурення у процес.

ВИСНОВКИ

У процесі виконання дипломної роботи було здійснено повноцінну розробку функціонального прототипу гри жанру action-survivor з елементами RPG. Проєкт базувався на аналізі популярної гри Vampire Survivors, але при цьому мав власну архітектуру та унікальні особливості реалізації, що сприяють масштабованості, розширюваності та гнучкості системи. Основні результати роботи включають:

1. Створення повноцінного ігрового середовища:

Реалізовано двовимірний світ з використанням тайлової мапи, де ігровий простір розширюється навколо гравця. Всі елементи середовища інтегровані вручну через Unity Tilemap, з використанням сторонніх відкритих ресурсів (тайлсетів) та кастомізованих графічних елементів.

2. Розробка ігрової логіки та архітектури:

Було створено модульну архітектуру на основі об'єктно-орієнтованого програмування (ООП) та впроваджено низку архітектурних патернів, зокрема Singleton, Strategy, Observer та Component. Це дозволяє легко додавати нові скіли, ворогів, покращення чи ігрові об'єкти без втручання у вже існуючий код.

3. Гнучкість у роботі з контентом:

Весь функціонал реалізовано вручну, включаючи бойову систему, анімації, взаємодію з UI, появу ворогів та механіку прокачування. Новий контент (напр. ворог, скіл, пасивна характеристика) можна інтегрувати за лічені хвилини, що є свідченням хорошої структуризації коду.

4. Командна робота та організація процесу:

Розробка проєкту здійснювалася в команді із застосуванням Kanban-дошки та щоденної комунікації через Discord. Це дозволило підтримувати постійний зв'язок між учасниками, відслідковувати виконання завдань у реальному часі та забезпечити злагоджену розробку навіть за відсутності окремих фахівців

5. Візуальний та звуковий супровід:

Візуальні елементи інтерфейсу створені вручну, а графічні ресурси (спрайти, фони, анімації) адаптовані з відкритих джерел або згенеровані за допомогою нейромереж. Звукові ефекти також отримані з відкритих ресурсів і реалізовані через окремий клас `AudioController`, що дозволяє легко змінювати й адаптувати звуки через редактор Unity.

Розроблений ігровий проєкт демонструє здатність створення сучасної, гнучкої та масштабованої гри з нуля з мінімальним використанням сторонніх інструментів. Уся логіка реалізована вручну з урахуванням принципів хорошого проєктування, що дозволяє не лише підтримувати ігровий код у належному стані, але й продовжувати розвиток гри в майбутньому без суттєвих витрат ресурсів. Отримані результати можуть слугувати основою для подальших ітерацій, включення кооперативного режиму, нових механік та візуального покращення.

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.	PlayFab; Розробник: Microsoft Документація: посилання	Використано для баз даних
5.	itch.io Розробник: Leaf Corcoran Документація: посилання	Використано для публікації збірки проекту