МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проєкт

здобувача освіти денної форми навчання
РП.08.16.000.ДП

ПЕСАРА ІВАНА ПЕТРОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**
Освітньо-професійна програма: «**Розробка програмного забезпечення»**
Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту на тему:

Розробка гри Vondering knight з різноманітними механіками ближнього бою

Проєктний матеріал складається з пояснювальної записки на 75 сторінках та графічного (презентаційного) матеріалу на 15 аркушах (слайдах)

Дипломник _____ (Песар І.П.)
Керівник _____ (Джабраілов Д.В.)

Консультанти:
з економічного розділу _____ (Канський М.Ю.)
з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)
з нормоконтролю _____ (Петрашова В.І.)
старший консультант _____ (Кривченко Ю.В.)

До захисту допущений
Голова циклової комісії _____ (Кривченко Ю.В.)
Завідувач відділення _____ (Краснокутська К.Г.)

Захист «21» 06 2025 р. Протокол ЕК № 1
Оцінка ЕК 4/87

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"12" 04 2025 р.

ЗАВДАННЯ

на дипломний проєкт

Здобувачеві освіти *Песару Івану Петровичу*

(прізвище, ім'я, по батькові)

1. Тема проєкту *Розробка гри Vondering Knight з різноманітними механіками ближнього бою*

затверджена наказом по коледжу від "14" листопада 2024р. № 246

2. Термін здачі закінченого проєкту

3. Вихідні данні до проєкту *Використання програмного рушія Godot Engine;Використання GodotScript для розробки гри;Реалізація основних ігрових механік для гри жанру top-down slasher;У грі мають бути реалізовані різні механіки ближнього бою для гравця;Гра повинна реалізовувати функцію видання ігрових квестів для гравця;Гравець повинен мати можливість вільно переміщуватись по ігровому рівню;В грі має бути реалізована механіка розвитку ігрового персонажу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналіз та обрання ігрового жанру для гри Vondering Knight;Обрання засобів розробки; Проєктування основних ігрових механік;Проєктування механік ближнього бою;Розробка основних ігрових механік гри;Розробка механік ближнього бою для гри;Розробка ігрового світу;Реалізація квестового персонажу.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Особливості обраного ігрового жанру;Вибір засобів розробки гри;Основні ігрові механіки; Механіки ближнього бою;Процес розробки гри;Розробка квестового персонажу;Тестування; Скріншоти розробленої гри.

6. Консультанти по проєкту, із зазначенням розділів проєкту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Джабраілов Д.В.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 02.06.2025

Керівник *Джабраілов Д.В.* _____
(підпис)

Завдання прийняв до виконання *Песар І.П.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проєкту	Термін виконання етапів дипломного проєкту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проєктування*	*14.05.2025*	виконав
2	*Аналіз ігрового процесу гри та вибір ігрового жанру*	*16.05.2025*	виконав
3	*Огляд та обрання програмних рішень для розробки*	*17.05.2025*	виконав
4	*Розробка концепту гри для розробки*	*20.05.2025*	виконав
5	*Проєктування основних та бойових ігрових механік*	*22.05.2025*	виконав
6	*Реалізація основних ігрових механік*	*28.05.2025*	виконав
7	*Реалізація механік ближнього бою*	*01.06.2025*	виконав
8	*Реалізація та імплементація квестового персонажу*	*03.06.2025*	виконав
9	*Відлагодження елементів гри*	*06.06.2025*	виконав
10	*Тестування працездатності елементів гри*	*10.06.2025*	виконав
11	*Виправлення виявлених помилок*	*11.06.2025*	виконав
12	*Аналіз результатів, підготовка слайдів презентації*	*12.06.2025*	виконав
13	*Економічні розрахунки та питання з охорони праці*	*13.06.2025*	виконав
14	*Підготовка графічної частини проєкту*	*14.06.2025*	виконав
15	*Підготовка проєкту до захисту та тестування гри*	*16.06.2025*	виконав

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 16 000. 00 ДП	Дипломний проєкт		
А4			РП 08. 16 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 16 000. 00 ДП		
Розробив		Песар І.П.		16.06	Розробка гри Vondering Knight з різноманітними механіками ближнього бою	Літ.	Аркуш 4 / Аркушів 75
Перевірив		Джабраілов Д.В.		16.06		Н Д П	
Н. контр.		Петрашова В.І.		16.06		ВСП "ОТФК ОНТУ" гр.4РП-08	
Затверд.		Кривченко Ю.В.					

ЗМІСТ

ВСТУП

Галузь розробки комп'ютерних ігор давно вийшла за межі розваги й перетворилась на повноцінний напрям в інформаційних технологіях. Вона охоплює складну екосистему, де програмування, графіка, звук, дизайн та логіка геймплею формують цілісний продукт. Із розвитком інструментів і поширенням відкритих технологій розробка стала доступною не лише великим студіям, але й окремим розробникам, що дозволило значно урізноманітнити типи ігор і форми їх подачі.

Особливий вплив на демократизацію виробництва ігор мали безкоштовні рушії та візуальні редактори, серед яких вирізняється Godot Engine – система з відкритим кодом, що надає широкі можливості для створення двовимірних та тривимірних ігор. Спрощення порогу входу, підтримка GDScript, зручність побудови сцен та модульна структура рушія дали змогу розробникам зосередитись на ідеї та механіці гри, а не на технічних складнощах її реалізації.

Окремо варто відзначити популяризацію піксельної графіки, яка в умовах обмежених ресурсів дозволяє досягти високого стилістичного ефекту. Поєднання такої графіки з механіками ближнього бою, натхненними класикою жанру top-down, відкриває поле для дослідження особливостей керування персонажем, балансу ігрових систем та організації візуального простору.

У цьому контексті розробка гри Vondering Knight стала не лише практикою створення повноцінного грального продукту, а й способом дослідити функціональні можливості рушія Godot, оптимізувати роботу з GDScript і створити візуальний стиль за допомогою інструмента Aseprite. Проєкт охоплює повний цикл: формування концепції, розробку механік, візуальне оформлення, інтеграцію контенту та тестування. Саме такий підхід дав змогу систематизувати досвід і краще зрозуміти вимоги, які ставить сучасна розробка до кожного етапу створення гри.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз та обрання ігрового жанру для гри Vondering Knight

1.1.1 Визначення цільової ігрової концепції

Процес розробки будь-якої гри починається з формування чіткої ігрової концепції – внутрішньо цілісного бачення того, що саме має становити основу ігрового досвіду. У випадку гри Vondering Knight базова ідея виникла як спроба поєднати динаміку бою, елемент пригодницького дослідження та візуальну стилізацію, що відсилає до естетики класичних піксельних проєктів. Із самого початку була визначена ключова ідея гри, гравець має поступово проходити гру, реагуючи на дії ворогів і змінні умови бою, а ігрова сцена має надавати достатньо простору для маневру, водночас залишаючись структурно зрозумілою.

Цільова ігрова концепція будувалася навколо центрального героя – лицаря, який виконує квести у світі, у якому знаходяться ворожі істоти. Важливою складовою задуму є бойові зіткнення, у яких ключовим являється ближній бій. Отже, першочергова увага приділялася механіці ударів, поведінці ворогів та створенню простору, у якому можливо втілити ці елементи природно й динамічно. Це створило потребу у форматі ігрового рівня, де кожна сутичка має значення, а не зводиться до механічного повторення.

Особливість обраного підходу полягала у фокусі на зчитуваності ситуацій під час гри: гравець завжди повинен мати змогу чітко бачити, де він перебуває, звідки насувається загроза, і як краще зреагувати на поточну ситуацію. Це потребувало зваженого підходу до вибору перспективи камери, композиції сцени, візуальної стилізації й ритму геймплею. Крім того, було враховано важливість інтуїтивного управління рухом та атакою, оскільки бойова система має бути достатньо гнучкою для різних тактик, але при цьому залишатися простою для освоєння.

Із практичної точки зору концепція мала відповідати обмеженням обраного рушія та доступних інструментів. Тому ще до початку технічного

Проєктування був зроблений акцент на доцільності реалізації візуальної частини у піксельній стилістиці, яка не потребує складної 3D-анімації або рендерінгу, а також дозволяє зосередитися на розробці чіткої бойової логіки та поведінки ворогів. У межах цієї концепції передбачалося, що ігрова сцена буде динамічно реагувати на дії гравця: об’єкти ворогів, їхнє розміщення, а також завдання на рівні формуватимуться з урахуванням зони досяжності та місця перебування персонажа.

Цільова концепція в підсумку набула форми бойової 2D гри з елементами квесту, орієнтованої на ближній бій і напружені сутички. Цей напрям став відправною точкою для всіх подальших рішень, включаючи вибір жанру, ігрових механік, візуального стилю та технічної реалізації.

1.1.2 Обґрунтування вибору жанру top-down action

Вибір жанру гри визначає загальну динаміку геймплею, структуру ігрового процесу та очікування від гравця. У випадку Vondering Knight ключовим завданням було створення інтенсивного, але контрольованого бойового досвіду, де гравець постійно перебуває в активному русі, а сприйняття простору та загроз має першочергове значення. У цьому контексті жанр top-down action виявився найбільш відповідним як з геймплейної, так і з технічної точки зору.

На відміну від платформерів, де переважну роль відіграє вертикальна навігація, top-down підхід забезпечує повну свободу руху по горизонтальній площині. Це дозволяє реалізувати бойову систему з максимальною кількістю варіацій напрямку атак, позиціювання відносно ворогів. У контексті ближнього бою це особливо важливо: атакувати можна з будь-якого боку, а вороги можуть пересуватися у будь-якому напрямку, створюючи природну необхідність для гравця орієнтуватися у просторі. З позиції візуального контролю жанр top-down надає стратегічну перевагу: камера завжди розміщується над персонажем, забезпечуючи огляд усієї найближчої зони дії. Це дозволяє гравцеві миттєво зчитувати ситуацію на полі бою, що критично для швидких рішень у бойових зіткненнях. Такий формат знижує потребу в додаткових підказках чи інтерфейсних елементах, адже вся інформація про загрози – прямо перед очима.

Серед прикладів, які демонструють ефективність жанру, можна згадати The Legend of Zelda: A Link to the Past – класичну гру з видом зверху, у якій реалізовано динамічні бої, дослідження просторів та гнучку бойову систему. Візуальна структура ігрового світу в цій грі дає змогу гравцеві орієнтуватися в просторі без зайвого навантаження на інтерфейс, а камера ефективно передає відчуття контролю над ситуацією, на рис. 1.1 зображено скріншот цієї гри. Подібний підхід, адаптований до сучасних інструментів та стилістики, лягає в основу концепції Vondering Knight.

Рисунок 1.1. Скріншот гри The Legend of Zelda: A Link to the Past

Також top-down action чудово масштабується – він дозволяє реалізовувати як вузькі, коридорні сцени з обмеженим маневром, так і більш відкриті локації з великою кількістю варіантів руху. У випадку гри Vondering Knight це відкритий світ, структурно створений у вигляді єдиної великої арени. Жанр також дозволяє легко змінювати темп гри, чергуючи інтенсивні бої з дослідженням світу.

Із практичного боку цей жанр добре узгоджується з інструментарієм рушія Godot Engine. Вбудовані можливості рушія дозволяють швидко реалізовувати рух у двовимірному просторі, підтримують роботу з tilemap-ами, легко налаштовуються колізії, і зручно реалізовується логіка зору камери та

розміщення об’єктів на сцені. Усе це дозволяє уникнути зайвих складнощів, зосередившись на механіках, які становлять серцевину ігрового досвіду – тобто ближніх боях. Вибір саме action-піджанру, а не RPG чи adventure, також був усвідомленим. В акценті на бойові зіткнення треба було реалізувати геймплей насиченим та ритмічним, де в основі – точне позиціювання й розуміння ворога. Це не передбачає складної системи розвитку персонажа або глибокої діалогової структури – навпаки, мінімалізм в інтерфейсі та увага до чистої механіки створює досвід, близький до аркадного, але з достатньою глибиною для відчуття майстерності.

Жанр top-down action став природним вибором у межах сформованої ігрової концепції. Він узгоджується із запланованими механіками ближнього бою, дозволяючи втілити інтенсивну гру з акцентом на контроль простору, маневрування та зорове сприйняття ситуації.

1.1.3 Врахування переваг жанру для створення гри

Жанр top-down action має низку технічних і концептуальних переваг, які особливо актуальні в умовах індивідуальної розробки гри. Завдяки використанню перспективи згори, під час розробки було отримано повний контроль над візуальним простором сцени. Це спрощує не тільки побудову рівня, а й логіку навігації, взаємодії з оточенням та реалізацію механік бою, прогресії або квестів.

Однією з ключових переваг такого жанру є висока читабельність ігрової сцени. Камера, що постійно показує гравця у центрі уваги, зменшує ризик плутанини в бою або втрати орієнтації. Завдяки цьому можливо створювати геймплей, зберігаючи при цьому візуальну зрозумілість для гравця.

Top-down підхід дозволяє уникнути складної просторової побудови сцен, властивої тривимірним іграм. Водночас він зберігає потенціал для глибоких механік. Наприклад, можна реалізувати різні типи супротивників із різною поведінкою, складні патерни атак або продуману навігацію між зонами, не вдаючись до складної фізики або об’ємного освітлення. Крім того, двовимірне

подання дає змогу зосередитись на стилізованій графіці, не витрачаючи ресурси на високополігональні моделі чи дорогі анімації.

Обраний жанр забезпечує баланс між технічною досяжністю та глибоким ігровим потенціалом. Він дозволяє створювати масштабний ігровий досвід без надмірних витрат ресурсів, зберігаючи чіткість управління, гнучкість дизайну і виразність візуального стилю.

1.2 Обрання засобів розробки

У процесі створення гри Vondering Knight ключовим етапом стало визначення технологічної основи, на якій буде реалізовано всі аспекти проєкту – від графіки до логіки геймплею. Обрання інструментів напряму впливає не лише на темпи розробки, а й на технічні можливості реалізації задуму.

1.2.1 Використання Godot Engine як основного рушія

У процесі розробки гри Vondering Knight вибір рушія визначався не лише популярністю або доступністю, а й відповідністю до специфіки проєкту, яка передбачає двовимірну перспективу, активну бойову систему й модульну структуру сцени. Серед усіх доступних варіантів найбільш органічним виявився саме Godot Engine, який поєднує простоту інтеграції, відкритий вихідний код і чітку орієнтацію на 2D-ігри, на рис. 1.2 зображено керування проєктами у Godot Engine.

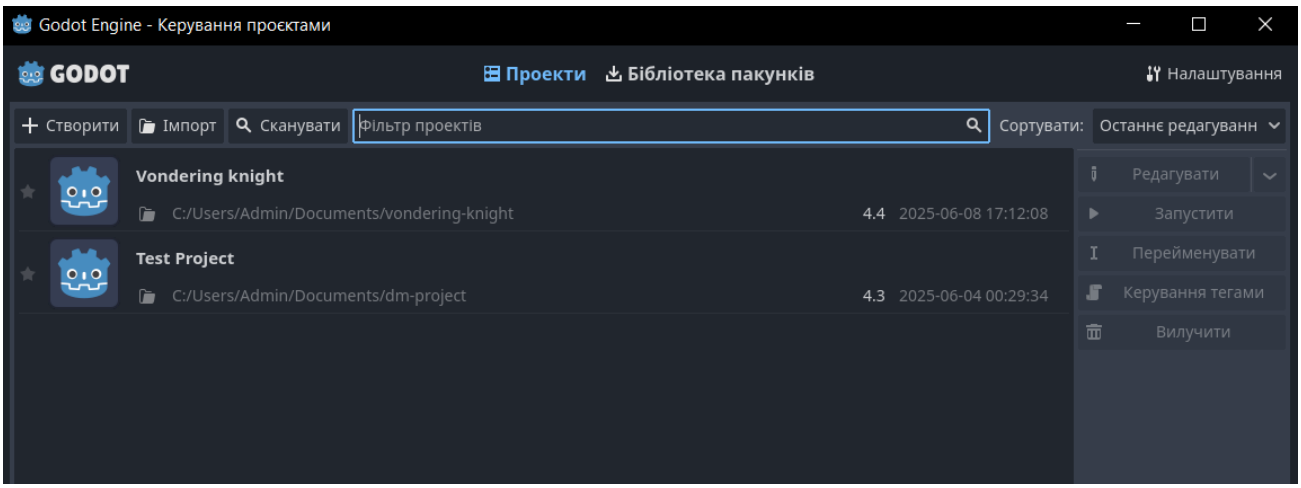


Рисунок 1.2. Скріншот керування проєктами у Godot Engine

Ключовим аргументом стало зручне сценове моделювання, що лежить в основі архітектури рушія. У Godot усі елементи гри – від гравця до найпростішого декоративного об’єкта – є вузлами, які можна поєднувати у дерева. Це дозволяє створювати складну поведінку з простих елементів, швидко змінювати логіку, структуру чи візуальне оформлення без потреби повністю перебудовувати ієрархію. Такий підхід особливо ефективний при роботі з інтерактивним середовищем і багаторазово використовуваними елементами, NPC, моби чи об’єкти ландшафту.

Важливою перевагою є також підтримка 2D-фізики. Godot надає власний механізм обробки колізій, зручну систему шарів і масок, які дозволяють чітко розмежувати взаємодію між об’єктами. У випадку з грою, де від фізичної точності залежить результат бою (удари, зіткнення зі стінами, траєкторії руху, вороги), ця можливість є критичною.

Ще однією перевагою є легкість у створенні анімацій за допомогою AnimationSprite2D. Завдяки цьому вся логіка рухів, атак, реакцій ворогів і ефектів була реалізована без потреби у зовнішніх редакторах чи складних підключеннях, на рис. 1.3 зображено як створюються сцени та виглядають визули.

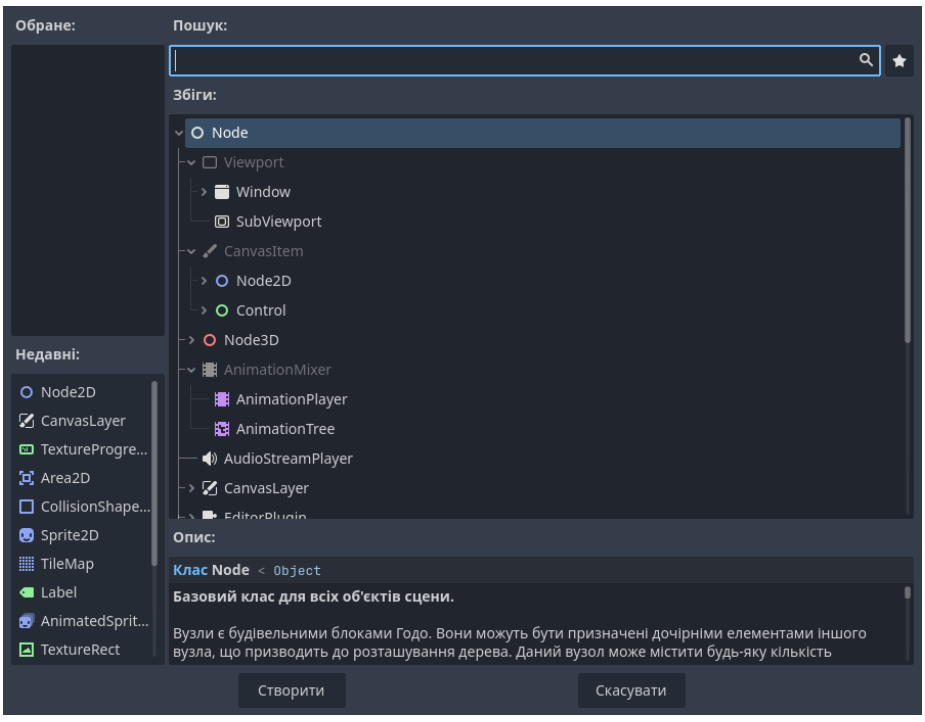


Рисунок 1.3. Скріншот структури створення сцени та вигляду вузлів

Крім того, рушій дозволяє експортувати проєкт на різні платформи, включно з Windows, Linux, Android і HTML5, без необхідності кардинально змінювати структуру проєкту або писати платформозалежний код. Це відкриває можливість подальшого розширення гри без істотних технічних обмежень. Godot Engine став не просто інструментом, а структурною основою всього проєкту, яка забезпечила необхідний баланс між функціональністю, гнучкістю й продуктивністю при реалізації ключових ігрових механік у межах 2D action-жанру з видом зверху. На рис. 1.4 зображено інтерфейс Godot Engine.

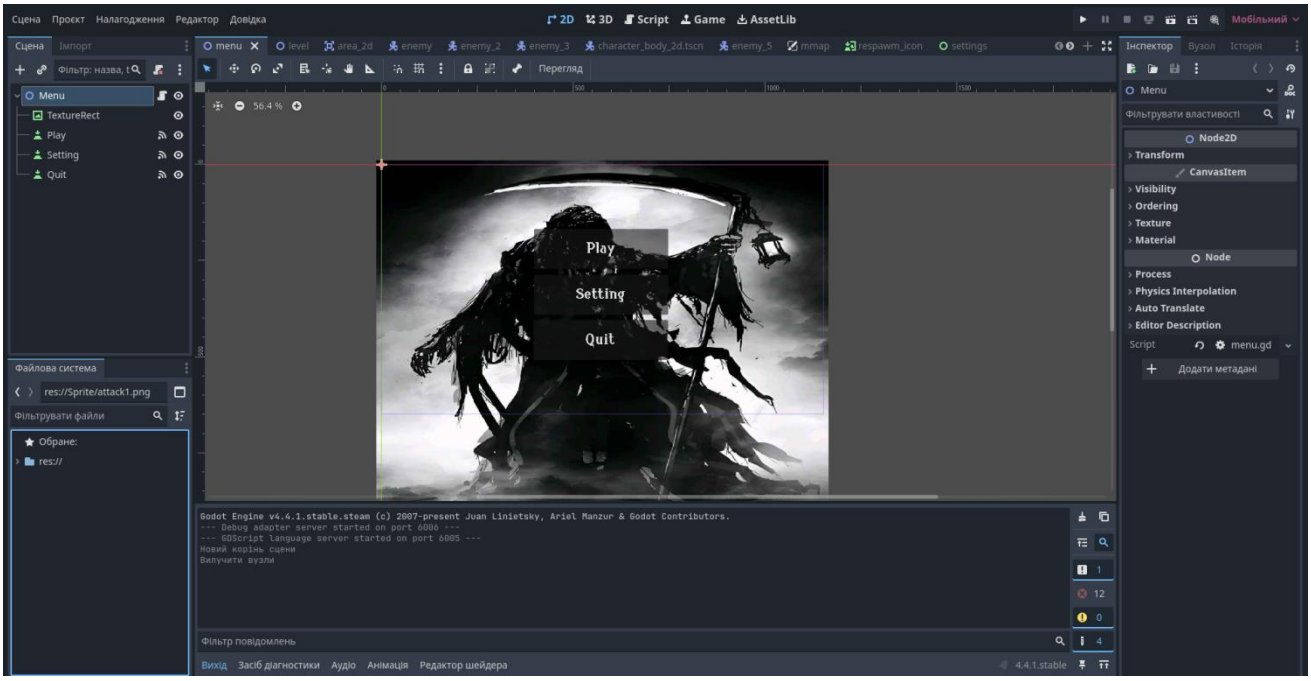


Рисунок 1.4. Скріншот інтерфейсу Godot Engine

1.2.2 Використання GDScript для реалізації логіки гри

Під час розробки гри Vondering Knight у середовищі Godot було обрано мову програмування GDScript як основний інструмент для реалізації логіки ігрового процесу. Такий вибір зумовлений, перш за все, органічною інтеграцією цієї мови з усіма аспектами рушія, її простотою, подібністю до Python і одночасною оптимізацією під специфіку роботи в Godot Engine.

Однією з головних переваг GDScript є його синтаксична лаконічність і високий рівень читабельності, що особливо важливо при роботі над проєктом, де часто відбувається зміна та доповнення логіки. Завдяки цьому написання, читання та налагодження коду відбувається швидко й інтуїтивно. Наприклад,

базові дії з обробки руху гравця, анімацій, обробки зіткнень або активації ворогів реалізуються у вигляді коротких і зрозумілих функцій. Це значно прискорює ітеративну розробку – можна швидко змінювати поведінку об'єктів, не заглиблюючись у складні архітектури чи бібліотеки, приклад коду функцій зображено на рис. 1.5.

```
func update_health_bar():
    if health_bar:
        health_bar.max_value = max_hp
        health_bar.value = current_hp

func apply_knockback(from_position: Vector2, force: float = knockback_strength):
    var direction = (global_position - from_position).normalized()
    knockback_vector = direction * force
    knockback_time_left = knockback_timer

func apply_burning(damage: int, interval: float, duration: float):
    var ticks = int(duration / interval)
    burn_timer.start(interval)
    burn_timer.set_meta("damage", damage)
    burn_timer.set_meta("ticks_left", ticks)
```

Рисунок 1.5. Скріншот прикладу функцій у коді GDScript

Ще одним аргументом на користь GDScript стало те, що ця мова є скриптовою, тобто дозволяє запускати та тестувати зміни в коді без необхідності довгого перекомпілювання проєкту. Це дало змогу зосередитися на вдосконаленні геймплейних механік та миттєво перевіряти результати змін без затримок. Такий підхід особливо важливий при створенні ітеративних механік ближнього бою, які потребували постійного налаштування таймінгів, радіусів атаки, а також поведінки ворогів у реакції на дії гравця.

Варто також зазначити, що велика кількість офіційної документації та відкритих прикладів на GDScript значно полегшує розв'язання типових завдань. Це дозволило зекономити час на реалізації базових алгоритмів і сконцентруватися на унікальних деталях гри.

GDScript став ефективним і гнучким інструментом для написання логіки гри Vondering Knight. Його синтаксична зручність, миттєвий запуск змін, глибока інтеграція з Godot Engine зробили цю мову оптимальним вибором для

проєкту, де гнучкість, швидкість і зрозумілість коду мали вирішальне значення. На рис. 1.6 зображено приклад коду GDScript.

```
91      if current_hp <= 0:
92          die()
93
94  func apply_knockback(from_position: Vector2):
95      var direction = (global_position - from_position).normalized()
96      knockback_vector = direction * knockback_strength
97      knockback_time_left = knockback_timer
98
99  func apply_burning(damage: int, interval: float, duration: float):
100     var ticks = int(duration / interval)
101     burn_timer.start(interval)
102     burn_timer.set_meta("damage", damage)
103     burn_timer.set_meta("ticks_left", ticks)
104
```

Рисунок 1.6. Скріншот прикладу коду GDScript

1.2.3 Застосування Aseprite для створення піксельної графіки

Візуальний стиль гри Vondering Knight базується на піксельній графіці з виглядом зверху (top-down), що вимагає точного й послідовного опрацювання кожного спрайту. Для створення всіх графічних елементів у проєкті використовувалась програма Aseprite, яка спеціалізується саме на розробці піксельного арту та анімації.

Однією з головних причин вибору Aseprite стала її зручна та інтуїтивно зрозуміла система роботи з шарами, таймлайнами та палітрами кольорів. Завдяки цьому вдалося ефективно створювати анімаційнії для персонажа, зокрема – рух, атаки, смерть, а також анімації для ворогів. Ще одною важливою перевагою Aseprite є підтримка функції onion skinning, її приклад зображено на рис. 1.7, яка дозволяє бачити попередні й наступні кадри при створенні анімації.

Рисунок 1.7. Скріншот прикладу функції onion skinning в Aseprite

Крім того, Aseprite дозволяє швидко коригувати кольорові схеми та працювати з обмеженими палітрами, що важливо для підтримки стилістичної цілісності гри. У випадку Vondering Knight використовувалась палітра endesga 32, на рис. 1.8 зображено її вигляд, вона підкреслює атмосферу небезпеки. Aseprite дозволив підтримувати цю стилістику на всіх етапах створення візуальних елементів – від іконок до інтерфейсу користувача. Ще одним практичним моментом стала можливість експортувати анімації у форматах, сумісних з Godot Engine. Під час розробки були використані як sprite, так і окремі PNG-файли з анімаціями, що імпортувались у Godot як AnimationFrames. Завдяки цьому інтеграція графіки в рушій відбувалась без втрат якості й без додаткової обробки.

Рисунок 1.8. Скріншот палітри endesga 32

У підсумку, Aseprite став незамінним інструментом для створення піксельної графіки у Vondering Knight. Його вузька спеціалізація, гнучкі функції для анімації та повна сумісність із форматом 2D-ігор дозволили підтримувати високу якість візуального стилю проєкту й ефективно організувати робочий процес під час розробки графіки. На рис. 1.9 зображено інтерфейс Aseprite.

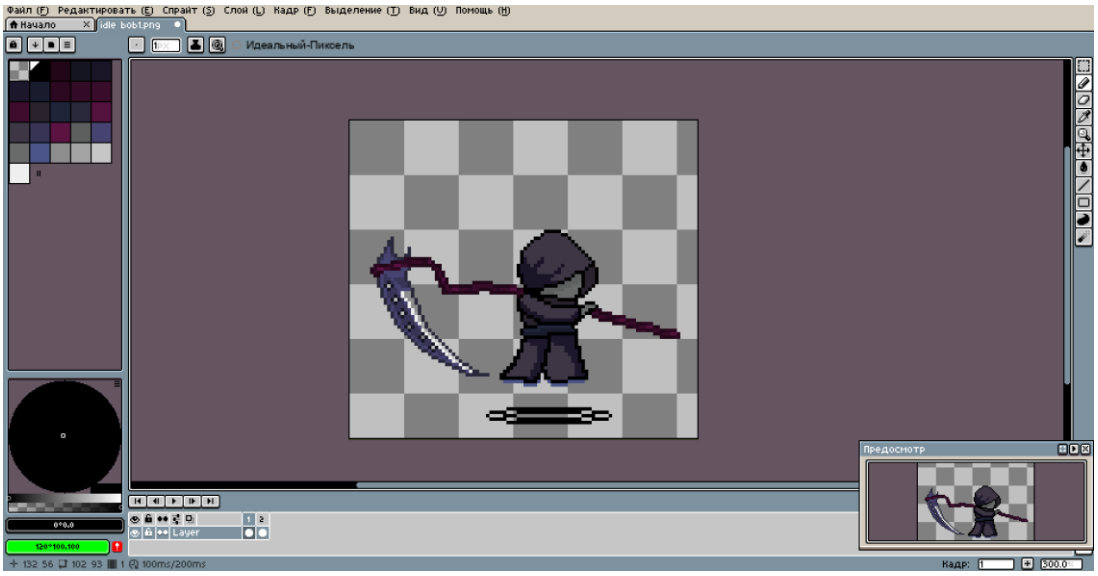


Рисунок 1.9. Скріншот інтерфейсу Aseprite

1.3 Проєктування основних ігрових механік

Проєктування ігрових механік є основоположним етапом у процесі створення будь-якої гри, оскільки саме механіки визначають те, яким буде досвід взаємодії користувача з ігровим світом. У випадку Vondering Knight важливим було поєднання простоти реалізації та глибини ігрових рішень, що відповідало формату індивідуального проєкту й жанру 2D top-down action. При цьому акцент був зроблений на ближньому бої, мінімалізмі інтерфейсу та зчитуваності ситуацій у реальному часі.

Основною задачею стало створення структури, яка дозволяє реалізувати всі компоненти гри – гравця, ворогів, бойові дії, взаємодію з NPC, інтерфейс та ігровий світ, у межах цілісної та гнучкої архітектури. Для цього було обрано вузлову модель рушія Godot, яка дала змогу модульно проектувати кожну частину гри.

1.3.1 Сценова модель та архітектура Godot як основа проєктування

Однією з ключових переваг рушія Godot, яка визначає підхід до Проєктування ігрової логіки, є його сцено-вузлова архітектура. Саме ця модель дозволила у грі Vondering Knight організувати весь процес розробки у структурований, логічний і гнучкий спосіб. На відміну від класичного підходу з єдиним об’єктом, який містить увесь код і функціональність, Godot пропонує

концепцію поділу гри на сцени та вузли, де кожна сцена є ієрархією вузлів, що виконують окремі ролі. Сцена в Godot – це не просто рівень або екран, а будь-який логічно завершений компонент гри: гравець, ворог, NPC, меню, інтерфейс, екран смерті. Кожна з таких сцен може бути незалежною одиницею, що містить свою структуру вузлів, логіку, сигнали, змінні й навіть вбудовані підсцени. Такий підхід дає змогу мислити не об’ктами, а структурами, де логіка проєкту розподілена між багатьма простими й керованими компонентами.

Наприклад, ігровий персонаж у Vondering Knight є окремою сценою, яка складається з вузлів, на рис. 1.10 зображено як це виглядає у Godot Engine. Структура сцени:

- CharacterBody2D
- CollisionShape2D
- AnimatedSprite2D
- Camera2D
- Area2D
- CanvasLayer

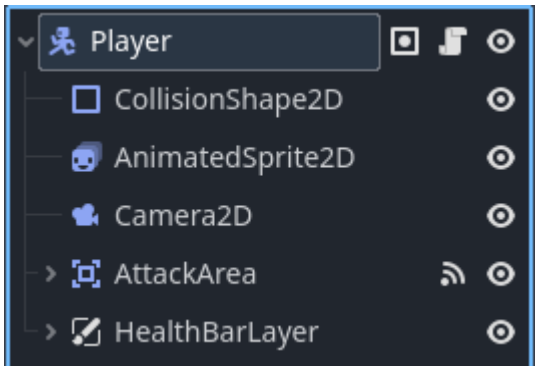


Рисунок 1.10. Скріншот структури сцени гравця

Усе це організовано як дерево, де батьківські вузли можуть містити дочірні, і так формується цілісна логічна структура об’єкта. Кожен вузол має свої властивості та методи, а також може використовувати сигнали, які дозволяють йому повідомляти інші вузли про події. Такий підхід є модульним, що означає: змінивши один вузол, немає потреби змінювати всю сцену. Наприклад, заміна AnimatedSprite2D на іншу анімацію або модифікація логіки

атаки не потребує повного переписування гравця. Це критично важливо для швидкого проєктування у процесі розробки.

Ще однією перевагою Godot є сигнальна система, яка дозволяє вузлам спілкуватися між собою через події. Наприклад, коли ворог помирає, він посилає сигнал enemy_killed, який перехоплюється глобальним контролером квестів, що, у свою чергу, оновлює статус завдання, що і було реалізовано у грі. Така подієва модель дозволяє розділити логіку між окремими вузлами й уникнути тісної залежності між ними, що особливо важливо в умовах індивідуальної розробки. У межах сцено-вузлової архітектури реалізовано не лише ігрові об’єкти, а й інтерфейс користувача (HUD), квестова система, спрайти атаки, система здоров’я, мінікарта. Наприклад, елементи інтерфейсу розміщено в окремому CanvasLayer, який не змінюється разом із камерою й завжди залишається в одному положенні на екрані. Такий підхід дозволяє зручно відображати шкалу здоров’я, індикатори атаки, підказки для квестів і зворотний зв’язок від дій гравця.

Завдяки цьому, сцено-вузлова структура не лише формально організовує проєкт, а й задає спосіб мислення при розробці: кожна ідея реалізується у вигляді сцени, кожен об’єкт – вузол, кожна взаємодія – сигнал. Це дозволило зберегти логічну цілісність гри, зменшити складність розробки, прискорити тестування та гарантувати гнучкість для подальших змін без руйнування існуючої структури.

1.3.2 Система керування в Godot як основа інтерактивності

Система керування є критичним компонентом будь-якої гри, оскільки саме вона встановлює прямий зв’язок між гравцем і внутрішнім світом проєкту. У грі Vondering Knight управління відіграє ключову роль, адже вся динаміка бою, руху, атак й взаємодії побудована навколо прямого контролю дій персонажа. Для реалізації інтерактивності було використано вбудовану систему обробки введення в Godot Engine, яка надає широкий набір можливостей як для базових, так і для складніших сценаріїв взаємодії.

Першим етапом стало налаштування мапи абстрактних дій, через які обробляються команди гравця. Замість безпосередньої перевірки натискання клавіш (KEY_W, KEY_D), у Godot вводяться абстрактні назви: "Up", "Down", "attack", на рис. 1.11 зображено як це виглядає. Кожна з них асоціюється з конкретною клавішею які можуть бути легко змінені без редагування коду. Це дозволяє швидко адаптувати керування, не змінюючи логіку поведінки персонажа.

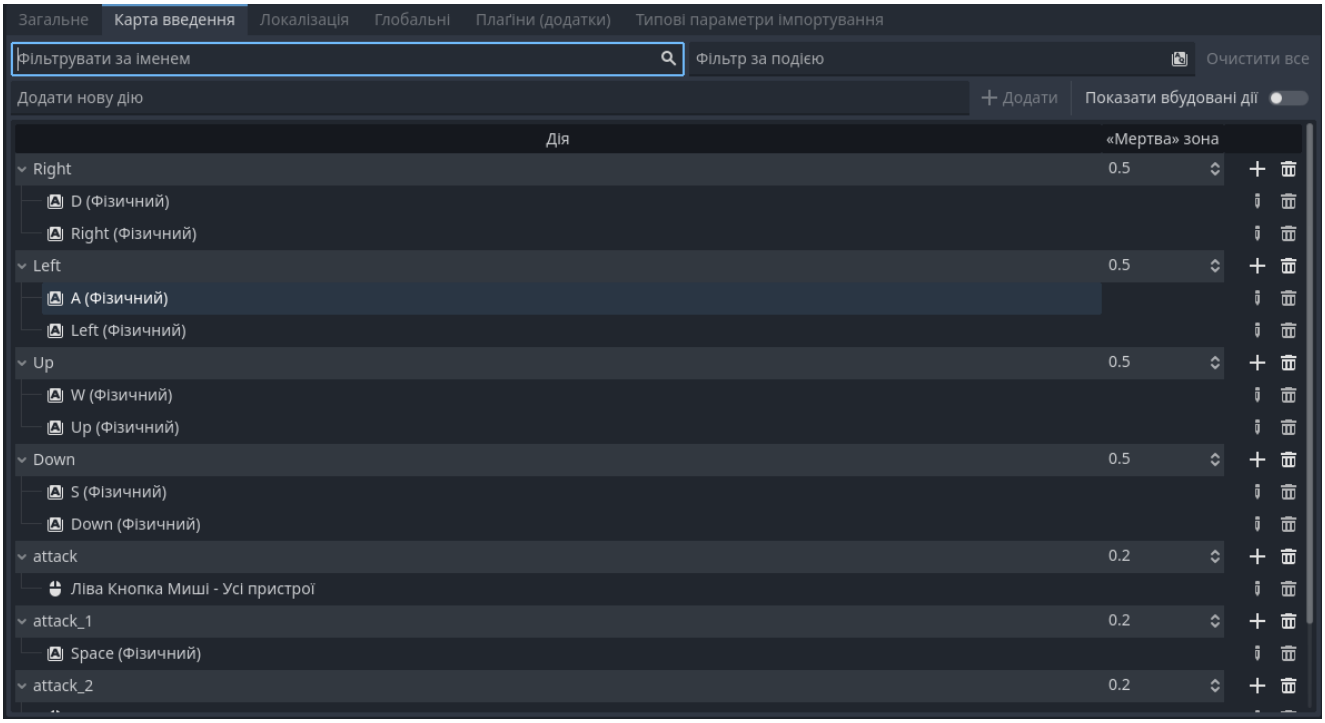


Рисунок 1.11. Скріншот карти введення

У коді для обробки дій використовується метод Input.get_action_strength(), який повертає значення сили натиснення певної дії. У грі Vondering Knight це дозволило реалізувати плавний рух гравця в усіх чотирьох напрямках, а також по діагоналі з однаковою швидкістю. Рух обробляється в методі _physics_process(delta), який виконується кожного кадру симуляції фізики. Сам алгоритм побудований на оновленні вектора швидкості (velocity) в залежності від натиснених клавіш, після чого використовується метод move_and_slide() для переміщення персонажа з урахуванням фізичних зіткнень, на рис. 1.12 зображено який цей код має вигляд.

```
var input_vector = Vector2(
    Input.get_action_strength("Right") - Input.get_action_strength("Left"),
    Input.get_action_strength("Down") - Input.get_action_strength("Up")
).normalized()

velocity = input_vector * SPEED
move_and_slide()
update_animation(input_vector)
```

Рисунок 1.12. Скріншот коду керування гравцем

Переміщення є не просто зміщенням координат, а повноцінною фізичною взаємодією, де враховується зіткнення з об’єктами, межами карти, NPC або ворогами. Це забезпечує точний контроль над поведінкою персонажа й знижує ризик непередбачуваних сценаріїв.

Ще одним важливим компонентом є управління анімацією, яке безпосередньо пов’язане з системою керування. Наприклад, залежно від дії гравця, перемикається відповідна анімація: рух, атака, статика. Це реалізовано через AnimatedSprite2D, у якому автоматично активується потрібний кадр анімації при зміні дії. Це забезпечує плавний і правдоподібний вигляд персонажа, синхронізуючи візуальну частину з логікою.

У сукупності, система керування в Godot дозволила побудувати чіткий, гнучкий і легко адаптовуваний механізм управління, який повністю контролюється через логічні дії, абстрактні команди і внутрішню фізичну модель. Завдяки цьому гра залишає враження плавної, контрольованої та інтуїтивно зрозумілої, що є критично важливим для динамічного жанру top-down action. Незалежно від складності сцени чи кількості ворогів, гравець завжди має змогу чітко контролювати дії свого персонажа, своєчасно реагувати на загрози і взаємодіяти з ігровим світом.

1.4 Проєктування механік ближнього бою

Створення ближнього бою як основної бойової механіки в грі виникла ще на ранньому етапі розробки гри, коли стало зрозуміло, що весь ігровий процес має будуватись навколо прямого фізичного зіткнення гравця з ворогами. Вибір саме ближнього бою був обумовлений атмосферою гри – стилістикою, у якій

основна зброя персонажа за задумом – це коса. Інші варіанти, такі як стрільба чи магія, з самого початку розглядалися як додаткові або другорядні, але потім не були реалізовані для того щоб не перевантажувати геймплей.

Коли почалась розробка, для зручності була створена блок-схема яку зображено на рис. 1.13, фокус був на тому, щоб механіка ближнього бою не просто існувала формально, а справді визначала темп гри. Хотілося, щоб кожен удар мав відчуття ваги – тобто був не лише візуально помітним, але й передавав відчуття контролю над ситуацією. Тому перше, що треба було зробити це щоб атака виглядала керованою, з чіткою прив’язкою до напрямку руху персонажа.

Рисунок 1.13. Блок-схема проєктування механік ближнього бою

У грі камера дивиться зверху, тому всі дії гравця сприймаються дуже чітко й відверто – будь-яка неузгодженість між напрямком атаки та анімацією була б помітною. Це створювало вимогу чітко синхронізувати візуальне відображення удару з його ефективною частиною – тобто моментом нанесення шкоди. Розуміння цього вплинуло на подальший підхід до структури бойової системи: вона мала бути не просто інтегрованою в анімацію, а залежати від неї логічно.

У межах цього ж етапу довелося обмірковувати систему перезарядки між ударами. Занадто швидкі удари перетворювали гру на спам-клікер, а надто повільні – робили бойову систему нудною. Знайти баланс між цими крайнощами було завданням, яке вирішувалося експериментальним шляхом: спочатку обирався приблизний інтервал, далі він підлаштовувався під швидкість ворогів, відстань між ними та загальну динаміку гри.

Тож коли все було продумано треба було спочатку створити анімації в Aseprite для кожної із атак, на рис. 1.14 зображено всі анімації.

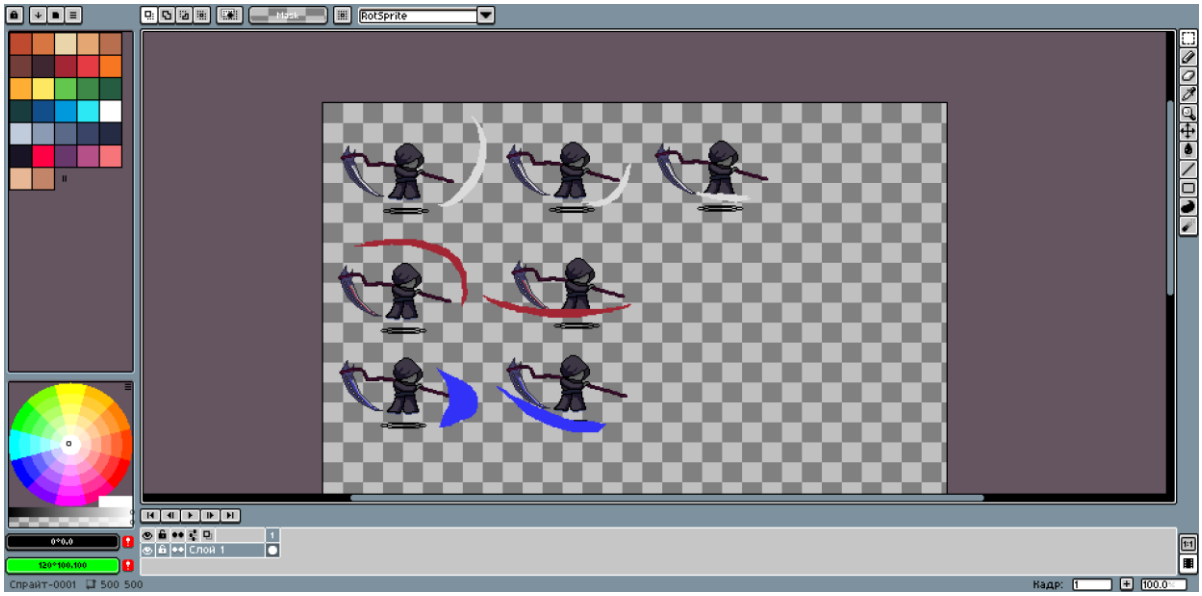


Рисунок 1.14. Скріншот спрайтів всіх анімацій ближнього бою

Ще одна річ, яку треба було враховував, – це зворотний зв’язок від атаки. Щоб після удару було чітко зрозуміло, чи він влучив. Тобто крім візуального ефекту удару, мали з’являтись якісь реакції від ворогів, якісь зміни у фізичному положенні (наприклад, відштовхування). Воно було реалізовано за допомогою body.apply_knockback(global_position, knockback_strength), коли гравець

підходить достаньо близько до ворога, та натискає атаку, це створює реакцію у вигляді поштовху – ворог фізично відкидається назад.

Другою механікою було обрано атаку з ефектом підпалу ворога, за допомогою body.apply_burning(FIRE_DAMAGE + FIRE_INTERVAL, FIRE_DURATION), так само коли гравець атакує в зоні досяжності атаки ворог умовно підпалюється та починає отримувати періодичну шкоду.

Третю механіку треба було зробити такою щоб гравцю давалась можливість більше контролювати дистанцію з ворогом, та отримувати можливість для маневру, тож була розроблена атака яка уповільнює ворога, даючи гравцеві перевертати хід бою, це було реалізовано за допомогою body.apply_slow(SLOW_DURATION).

Також для виводу шкоди використовувався метод take_damage, у виклику show_damage_number(amount) за допомогою коду який зображено на рис. 1.15.

```
func show_damage_number(amount: int):
    var label = DAMAGE_NUMBER_SCENE.instantiate()
    label.text = str(amount)
    get_parent().add_child(label)
    label.global_position = global_position
```

Рисунок 1.15. Скріншот коду виводу шкоди

Також треба було вирішити, чи повинен удар бути спрямований лише в бік, у якому гравець рухається, чи варто додати можливість атакувати під час стояння на місці. Зрештою, було прийнято рішення, що атака завжди відбувається в останньому напрямку руху персонажа. Це дозволяє гравцеві зупинитись, зорієнтуватись і завдати точного удару без обов'язкової необхідності рухатись під час атаки.

Загалом проєктування ближнього бою було етапом, який багато в чому визначив стиль усієї гри. Він задає динаміку, ритм і основний підхід до геймплею. На цьому етапі ще не було повного конкретного реалізованого у коду – всі рішення приймались концептуально, з невеликим розумінням того як це буде виглядати у коді, та з написанням деяких основних моментів.

У процесі формування механік ближнього бою постала ще одна важлива задача – визначити, як ці атаки будуть взаємодіяти з ворогами. Уже на етапі Проєктування стало зрозуміло, що простого факту зіткнення буде недостатньо. Потрібна чітка система перевірки попадання, яка б враховувала не тільки позицію гравця, а й конкретний момент удару та просторовий радіус дії зброї. Так виникла ідея про використання Area2D, у якому був CollisionShape2D який виступав окремою колізійною областю для удару, яка з'являється лише в момент атаки і лише в тому напрямку, у якому спрямована дія, на рис. 1.16 зображено як це вигладало у момент проєктування.

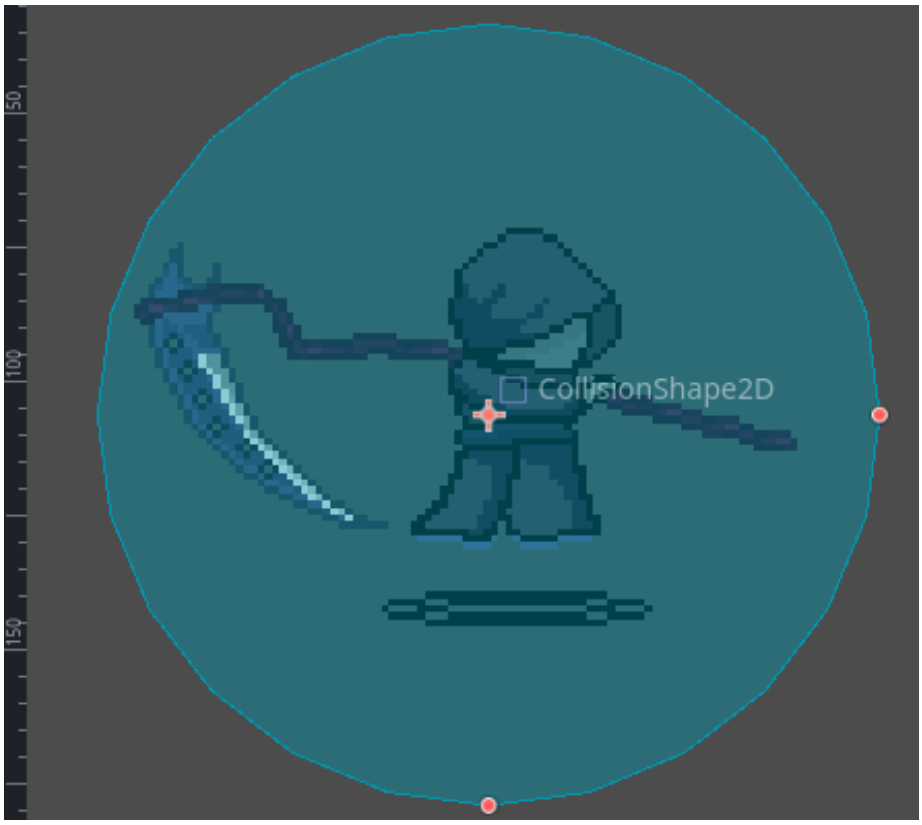


Рисунок 1.16. Скріншот області у якій відбувається атака

Це рішення дало змогу чітко відокремити момент атаки від руху персонажа, і водночас дозволило будувати систему ворогів, яка реагує саме на факт потрапляння в цю зону, а не на загальну присутність гравця поряд. Такий підхід також відкрив можливості для подальшого створення нових ігрових механік.

На рівні проєктування важливо було також продумати та створити ворогів, для того щоб у момент розробки повністю сфокусуватися на розробці атак

гравця. Зважаючи на те що час був обмежений, а також для того щоб не дуже перевантажувати геймплей, було створено 5 ворогів. Під час роздумів, для цікавості ігрового процесу ворогів було вирішено поступово ускладнювати, для того щоб бої не були однотипними та надто легкими в подальшому. Тож на початку треба було розробити кожного ворога та спрайти до них, на рис. 1.17 зображено спрайти усіх ворогів.

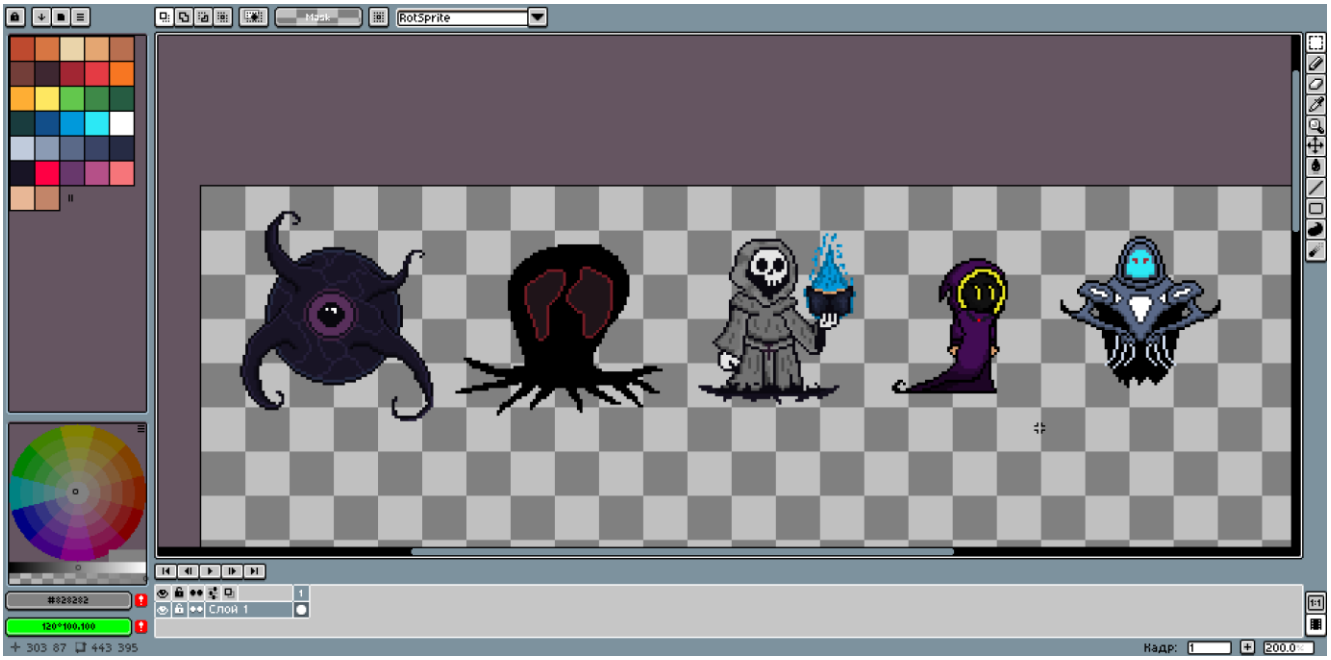


Рисунок 1.17. Скріншот спрайтів усіх ворогів

Також ворогам треба було створити анімації атаки, руху, смерті та простого стояння на місці. Атака у ворогів була розроблена за такою самою системою як і у гравця, за допомогою Area2D, зони досяжності атаки, для кожного ворога також була створена спрайт анімація, всі анімації атаки зображено на рис. 1.18.

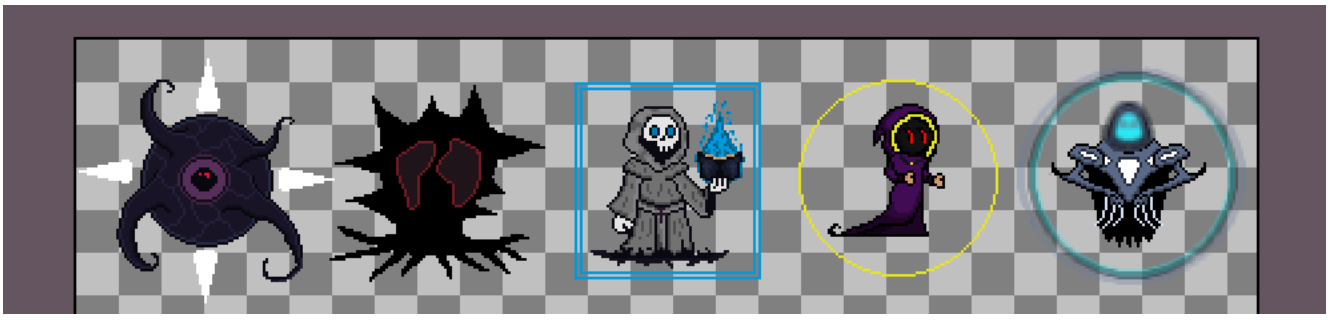


Рисунок 1.18. Скріншот спрайтів анімації атаки

Також треба було розробити те як гравець буде отримувати шкоду, для цього на рівні проєктування треба було створити лінію здоров'я гравця, HealthBar було створено за допомогою вузла TextureProgressBar який знаходиться у CanvasLayer. У Godot це реалізовується за допомогою двух іконок, рамки та заповнення (здоров'я), рамка використовується як візуал, а заповнення змінюється у коді, також така сама але спрощена система створена і ворогам, але без рамки, для того щоб вороги не мали вигляд гравця.

Для того щоб почати реалізовувати цю систему у грі треба було створити спрайти, такі щоб гравцю було зрозуміло скільки в нього здоро'я та де взагалі в інтерфейсі це знаходиться, тож були створені спрайти, які зображені на рис. 1.19.

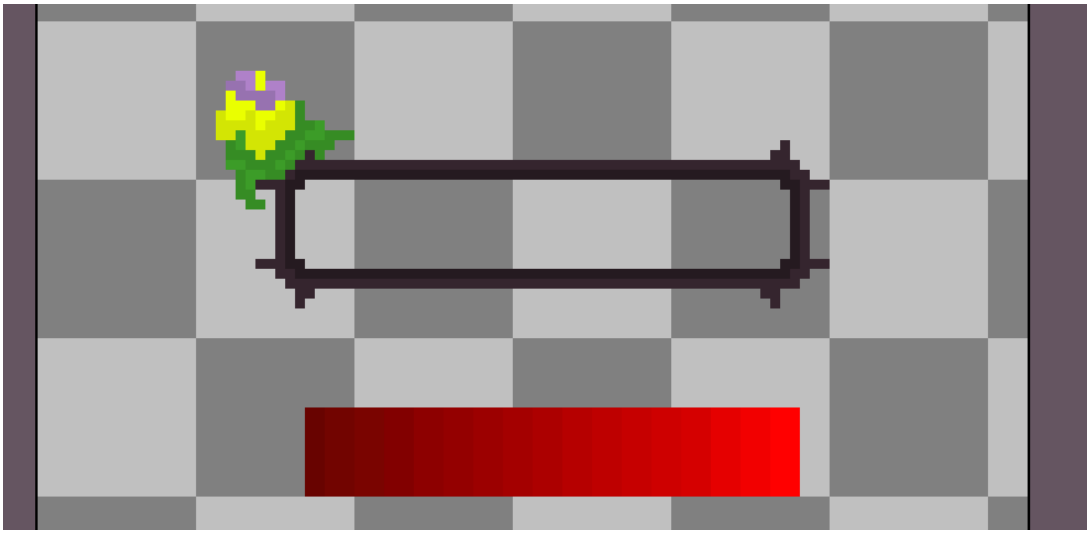


Рисунок 1.19. Скріншот спрайтів HealthBar

Коли все було готово був розроблений простий код у якому використовується функція update_health_bar(), яка за допомогою hp_bar.max_value = max_hp та hp_bar.value = current_hp, змінює здоров'я від заданої йому шкоди. Також на етапі проєктування було прийняте рішення, для ускладнення геймплею, не поновлювати здоров'я гравця після виконання квесту, це зроблено для того щоб гра була складною не зважаючи на обмежену кількість ворогів.

1.5 Розробка основних ігрових механік гри

Процес реалізації ігрових механік у Vondering Knight починався зі створення основної сцени, яка виконувала функцію центрального ігрового

простору. Кожна нова сцена яка створювалась окремо від основної в подальшому прикріплялась до основної, що створило чітку ієрархію, на рис. 1.20 зображено структуру основної сцени.

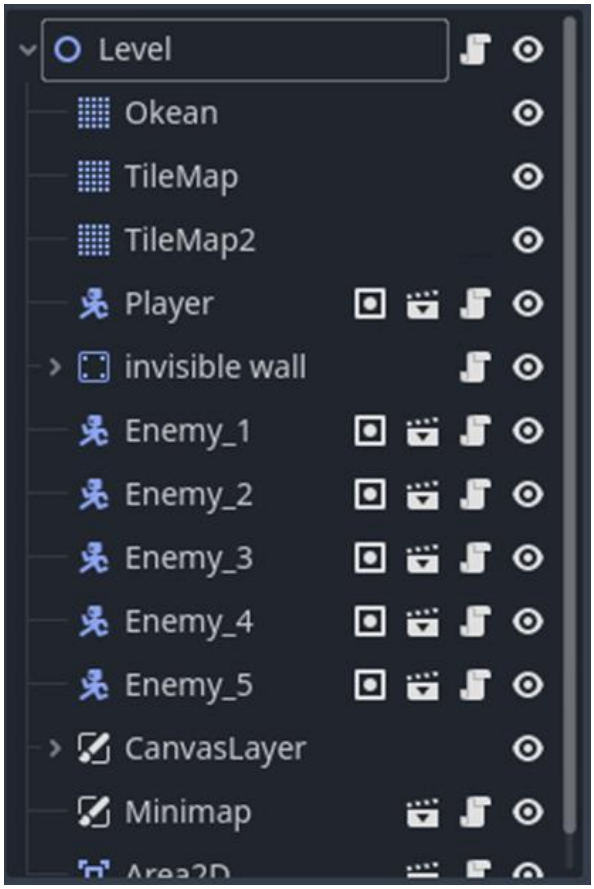


Рисунок 1.20. Скріншот структури основної сцени

Також треба було розробити головне меню гри, щоб коли гравець запускав гру, його не перекодувало одразу у рівень, а у вікно з кнопками де він міг почати грати, зайти в налаштування або вийти з гри.

Його реалізовано як окрему сцену на основі вузла Node2D, що виконує функцію контейнера для всіх візуальних та функціональних елементів. До нього були додані допоміжні вузли: TextureRect для виведення фонового зображення та три кнопки Button – Play, Setting, Quit. Кожна з кнопок має свою обробку натискання, реалізовану через вбудовану систему сигналів Godot. Функціонал був закладений у скрипт, прикріплений до кореневого вузла Node2D. Обробнику реалізовано через код який зображено на рис. 1.21.

```
4  func _on_quit_pressed() -> void:
5      get_tree().quit()
6
7  func _on_setting_pressed() -> void:
8      get_tree().change_scene_to_file("res://Settings.tscn")
9
10 func _on_play_pressed() -> void:
11     get_tree().change_scene_to_file("res://level.tscn")
```

Рисунок 1.21. Скріншот коду головного меню

Кнопка Play ініціює запуск основної ігрової сцени (level.tscn), що відкриває гравцеві доступ до активного проходження. Кнопка Quit завершує виконання гри, викликаючи метод get_tree().quit(), а Setting відкриває нову сцену з налаштуваннями, яка наразі виконує лише формальну роль. Кожна з кнопок підключена до відповідного сигналу pressed, що гарантує миттєве реагування на взаємодію без додаткової перевірки стану. Вся сцена головного меню побудована так, щоб залишатися максимально простою та легко масштабованою в майбутньому. Вузлова структура дозволяє за потреби розширити меню – додати нові кнопки, ефекти наведення або анімації без необхідності повністю переробляти логіку. Розміщення елементів на екрані реалізовано вручну у редакторі, з урахуванням центру екрана, що забезпечує симетрію та зручність навігації для гравця.

Меню слугує стартовою точкою проєкту, поєднуючи заголовкову сцену з основною ігровою логікою, на рис. 1.22 зображено який вигляд має головне меню гри.

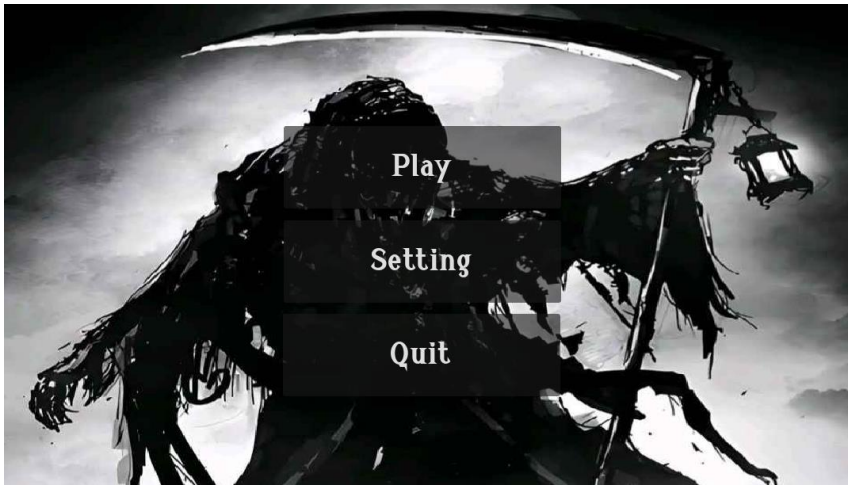


Рисунок 1.22. Скріншот головного ігрового меню

Також був створений гравець як самостійна сцена з усією необхідною логікою. Основою його фізичної моделі став вузол CharacterBody2D, що автоматично забезпечує обробку зіткнень і руху на основі фізичних правил. Об’єкт реагує на введення користувача, обчислює вектор руху та за допомогою функції move_and_slide() переміщується по тайлмапу, не проходячи крізь стіни або інші тіла.

Камера в грі реалізована за допомогою Camera2D, яка динамічно слідкує за гравцем. Завдяки налаштуванню властивості “current”, камера автоматично фокусується на координатах персонажа і переміщується разом із ним. У поєднанні з правильно підібраним масштабом це створювало відчуття цілісного ігрового світу, в якому гравець завжди перебуває в центрі подій. Простір навколо розкривався поступово, по мірі просування гравця, а система меж дозволяла уникнути виходу камери за край карти, також до камери була додана система умовної невеликої затримку відгуку, для того щоб камера відчувалася “живою”, на рис. 1.23 зображено як камера пересувається за гравцем.

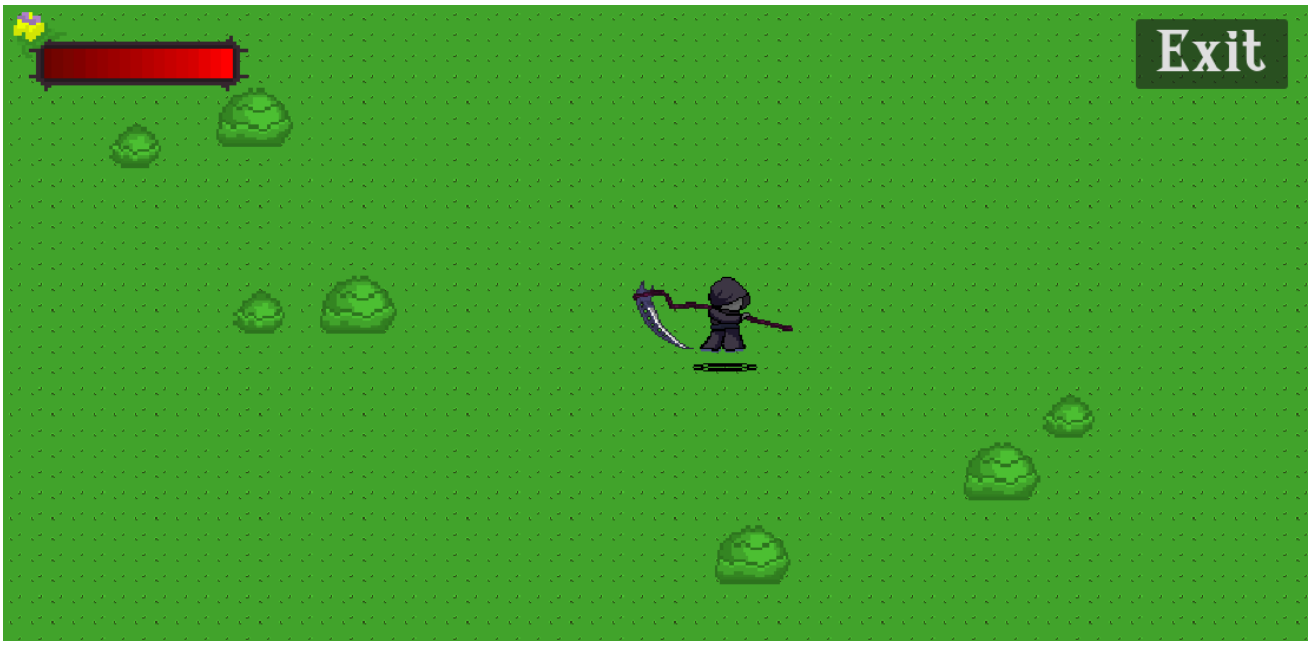


Рисунок 1.23. Скріншот пересування камери за гравцем

Після реалізації основ гравця та його руху, наступним кроком стала інтеграція ворогів. Усі супротивники створювались як окремі сцени з власною логікою пересування, атак і взаємодії.

Вороги реалізовані також через вузол CharacterBody2D, і використовують спрощену систему навігації, сама структура сцени ворогів була ідентична один до одного, її зображено рис. 1.24.

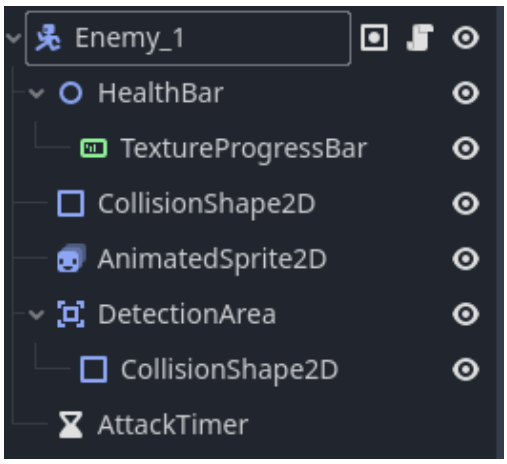


Рисунок 1.24. Скріншот структури сцени ворога

Окрім базової поведінки ворогів, у грі була реалізована система агро-зони – це окрема Area2D, прикріплена до кожного ворога, яка визначає радіус виявлення гравця, на рис. 1.25 зображено як ця зона виглядає. У момент, коли гравець потрапляє в цю зону, ворог отримує сигнал body_entered, після чого починає рух у напрямку до цілі. Якщо гравець залишає межі агро-зони (body_exited), переслідування припиняється, і ворог повертається у початковий стан. Такий підхід дозволив зберігати природну поведінку ворогів, уникнути їхньої постійної активності, зменшити навантаження на процесор та надати гравцю можливість переосмислювати свою стратегію бою.

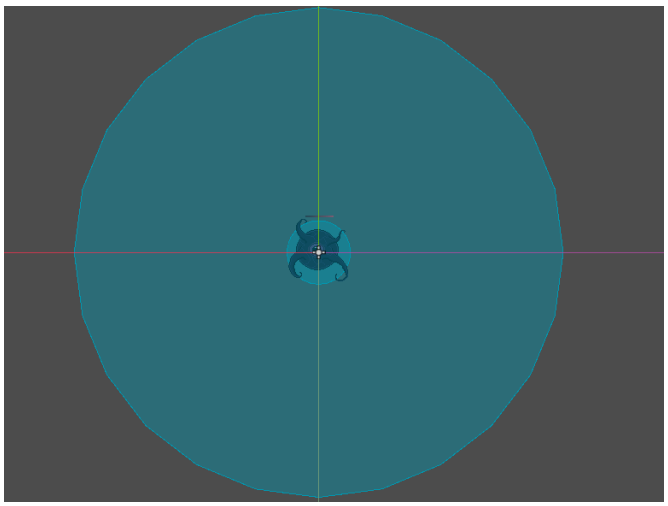


Рисунок 1.25. Скріншот зони виявлення гравця у ворога

Ще одним важливим елементом є реалізація логіки смерті як гравця, так і ворогів. Коли кількість здоров'я (current_hp) падає до нуля, викликається функція die(), яка обробляє всі події, пов'язані зі знищенням об'єкта. У випадку ворога – запускається коротка анімація смерті, а сам об'єкт видаляється зі сцени за допомогою queue_free(). Одночасно надсилається сигнал до глобальної системи квесту, що враховує факт знищення ворога. Для гравця смерть активує окрему сцену поразки, зупиняє рух, блокує управління та очищує інтерфейс. Це дозволяє миттєво завершити поточний сеанс гри, не зберігаючи прогрес, що створює враження виклику й необхідності діяти уважно, на рис. 1.26 зображено анімацію смерті гравця та ворогів.

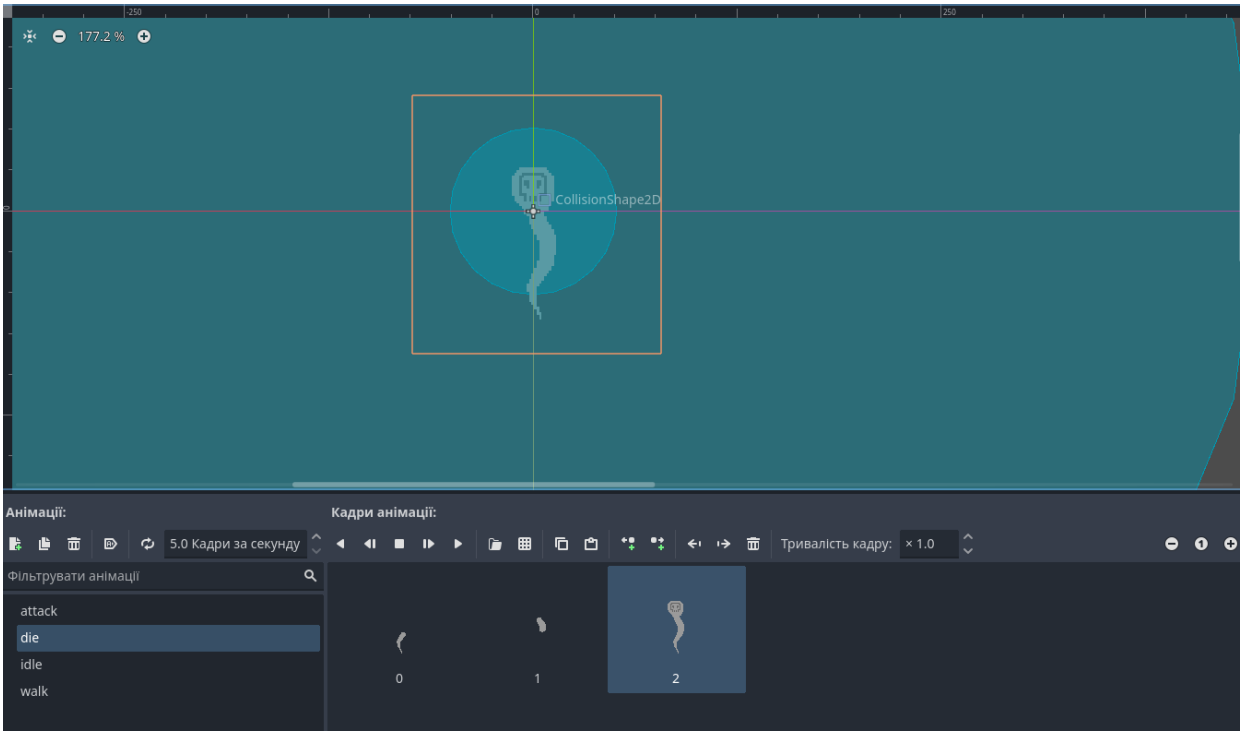


Рисунок 1.26. Скріншот анімації смерті гравця та ворогів

1.6 Розробка механік ближнього бою для гри

Побудова системи атаки гравця стала однією з ключових задач у проєкті, оскільки основна бойова механіка базується саме на ближньому бою. Початковою задачею було забезпечити просту й водночас динамічну бойову систему, яка б дозволяла гравцю швидко реагувати на загрози та ефективно взаємодіяти з ворогами. Ідея полягала в тому, щоб уникати складних комбінацій

або комбо-ланцюгів і натомість зосередитися на чіткій синхронізації між анімацією атаки, її тривалістю та моментом нанесення шкоди, як і у минулих розділах для зручності була розроблена блок-схема, яку зображено на рис. 1.27.

Окремо було допрацьовано зону, в якій може відбуватись ураження ворогів, вона стала активуватися лише на короткий момент під час удару. Це дозволило уникнути ситуацій, коли ворог отримував шкоду або задовго до початку анімації, або після її завершення. Такий підхід забезпечив візуальну й ігрову точність.

Ця логіка атаки стала фундаментом для всієї бойової системи гри, адже саме з нього починається кожна бойова взаємодія між гравцем і ворогами. Її якісна реалізація дозволила побудувати далі усі інші бойові функції, такі як обробка отримання шкоди, смерті ворогів, квестових перевірок.

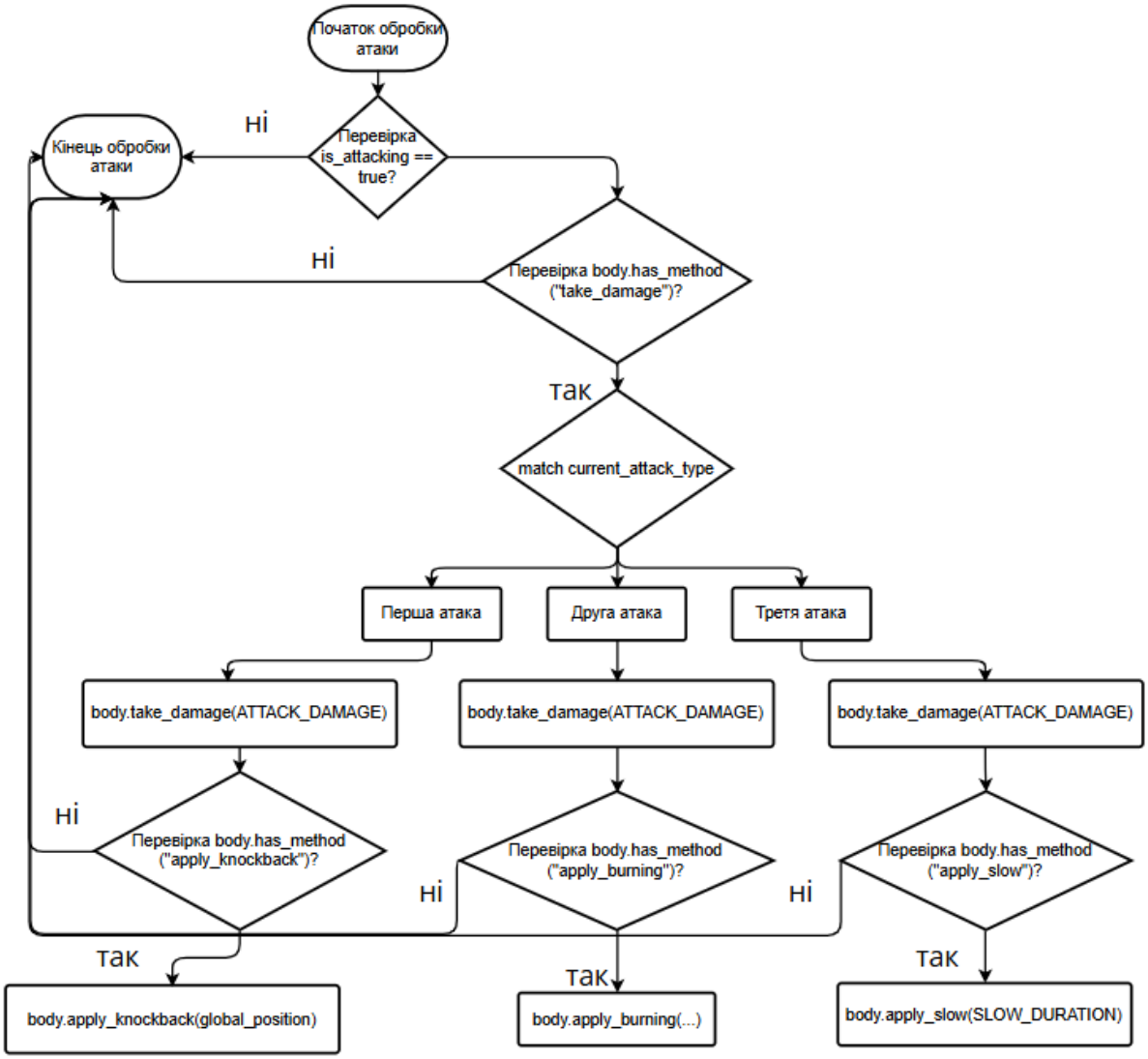


Рисунок 1.27. Блок-схема обробки атаки

1.6.1 Звичайна атака з відштовхуванням ворога

У реалізації гри Vondering Knight базова атака гравця об’єднує простий механізм ближнього бою з візуальною анімацією та фізичним ефектом відштовхування ворога. Ця атака є основою всіх бойових дій і виконується через активне виявлення об’єктів у зоні ураження під час анімації.

Головний персонаж створений на основі CharacterBody2D, що забезпечує обробку руху та взаємодії з фізичним середовищем. Для реалізації атаки використано анімований спрайт (AnimatedSprite2D) і зону атаки (AttackArea), що представлена вузлом Area2D із активованим monitoring лише на час удару. При натисканні клавіші атаки в коді викликається функція start_attack(), яка активує відповідну анімацію та вмикає AttackArea.

Реакція на влучання реалізована в обробнику _on_attack_area_body_entered(), який перевіряє наявність у об’єкта методу take_damage і, якщо атака є звичайною ("attack"), викликає його. Одночасно перевіряється, чи підтримує ворог метод apply_knockback, який, у разі наявності, використовується для застосування сили відштовхування, його зображено на рис. 1.28.

```
if is_attacking and body.has_method("take_damage"):
    match current_attack_type:
        "attack":
            body.take_damage(ATTACK_DAMAGE)
            if body.has_method("apply_knockback"):
                body.apply_knockback(global_position, knockback_strength + attack_knockback_bonus)
```

Рисунок 1.28. Скріншот коду який перевіряє можливість нанесення першої атаки

Сам механізм відштовхування реалізується у ворога через функцію apply_knockback(from_position), яка розраховує нормалізований напрям від гравця до ворога та задає knockback_vector, що використовується в _physics_process для зміщення тіла ворога протягом короткого часу який зображено на рис. 1.29.

```
func apply_knockback(from_position: Vector2, force: float = knockback_strength):
    var direction = (global_position - from_position).normalized()
    knockback_vector = direction * force
    knockback_time_left = knockback_timer
```

Рисунок 1.29. Скріншот коду відштовхування ворога на певний час

Як персонаж так і ворог, отримуючи удар, зміщується назад відповідно до напряму атаки. Завдяки таймеру knockback_time_left ефект обмежується коротким періодом часу, що зберігає контрольованість фізики та запобігає порушенню поведінки ворога.

Загальна система побудована на розділенні відповідальностей: гравець ініціює атаку та передає координати, а ворог самостійно визначає, як реагувати. Це дозволяє створювати ворогів з різними варіантами поведінки при влучанні, не змінюючи логіку гравця. Базова атака, є не лише механічним елементом бою, а й фундаментом для подальшого розширення системи ближнього бою, скріншот анімації у Godot Engine зображено на рис. 1.30.

Рисунок 1.30. Скріншот анімації першої атаки

1.6.2 Вогняна атака з ефектом підпалу

Другий тип атаки у грі Vondering Knight реалізує статус-ефект «підпалювання», що завдає шкоди ворогам протягом певного періоду часу після контакту з ударом. У цій атаці немає прямого фізичного впливу у вигляді відштовхування, але її сила – в поступовому виснаженні здоров'я ворога, що підсилює тактичну глибину бою.

Як і в базовій атаці, удар починається з виклику функції start_attack("attack_1"), що активує відповідну анімацію та тимчасово вмикає AttackArea. Після виявлення ворога у зоні дії система перевіряє, чи вказано в змінній current_attack_type значення "attack_1", та чи має ворог метод apply_burning. Якщо так, викликається ця функція з параметрами, які задають інтенсивність та тривалість ефекту, код зображено на рис. 1.31.

```
"attack_1":
    if body.has_method("apply_burning"):
        body.apply_burning(
            FIRE_DAMAGE + fire_damage_bonus,
            FIRE_INTERVAL,
            FIRE_DURATION
```

Рисунок 1.31. Скріншот коду який перевіряє можливість нанесення другої атаки

Ворог починає отримувати шкоду з фіксованою частотою (FIRE_INTERVAL), протягом певного часу (FIRE_DURATION). Усі ці параметри визначені як константи у коді гравця, які зображені на рисунку 1.32.

```
const FIRE_DAMAGE = 8
const FIRE_DURATION = 3.0
const FIRE_INTERVAL = 1.0
```

Рисунок 1.32. Скріншот коду параметрів другої атаки

Це дозволяє легко змінювати баланс гри без втручання в саму логіку. Також, застосування ефекту відбувається не в момент самої атаки, а через передачу керування до методу ворога, який реалізує сам процес завдання періодичної шкоди. Гравець виконує лише тригерну функцію, а подальші дії повністю залежать від внутрішньої реалізації ворога.

Перевага цього підходу – повна модульність. Ефекти накладаються тільки на тих ворогів, які підтримують відповідну поведінку. Якщо ворог не має методу apply_burning, жодних змін не відбудеться. Такий принцип забезпечує гнучкість у створенні нових типів ворогів, не порушуючи вже написану логіку гравця.

Статусний ефект дозволяє відійти від ворога після активації атаки й уникнути прямого контакту, що має стратегічну цінність у складніших ситуаціях бою, на рис. 1.33 зображено анімацію цієї атаки.

Рисунок 1.33. Скріншот анімації другої атаки

1.6.3 Водяна атака з ефектом уповільнення руху ворога

Третій варіант ближньої атаки у грі Vondering Knight має специфічну функціональність – вона не завдає прямої шкоди, але тимчасово зменшує швидкість руху ворога. Цей тип удару особливо ефективний у тактичному плані, дозволяючи гравцю отримати простір для маневру або встигнути змінити позицію.

Механіка активації загалом ідентична до попередніх типів атак: під час натискання відповідної клавіші викликається функція start_attack("attack_2"), яка запускає анімацію та активує зону attack_area. У момент контакту з тілом ворога перевіряється значення current_attack_type, і в разі відповідності "attack_2" викликається метод apply_slow, якщо ворог підтримує таку поведінку, на рис. 1.34 зображено цей код.

```
"attack_2":
    if body.has_method("apply_slow"):
        body.apply_slow(SLOW_DURATION + slow_duration_bonus)
```

Рисунок 1.34. Скріншот коду який перевіряє можливість нанесення третьої атаки

Параметр SLOW_DURATION, що задає тривалість ефекту, визначений як константа:

const SLOW_DURATION = 2.0

У результаті на ворога накладається статус, який на визначений час зменшує його швидкість. Цей ефект не залежить від анімації або позиції гравця після удару – ефект діє самостійно, як тільки було підтверджено контакт з ворогом, на рис. 1.35 зображено анімацію останьої атаки.

Ця атака не має вбудованої шкоди чи ефекту відштовхування, але її сила полягає у контрольному потенціалі. Завдяки уповільненню можна ізолювати ворога, знизити загрозу або створити момент для відступу. Подібно до атаки з підпалом, ця реалізація використовує принцип: гравець лише ініціює ефект, а вся логіка реалізується на боці ворога. Такий підхід підтримує масштабованість. Якщо у грі з'являються нові вороги, які мають свою реакцію на уповільнення, їхня поведінка може бути реалізована незалежно від логіки гравця. Завдяки цьому механіка залишається універсальною, але водночас адаптивною до специфіки кожного ворога.

Рисунок 1.35. Скріншот анімації третьої атаки

1.7 Розробка ігрового світу

Розробка ігрового світу починалася зі створення загальної концепції рівня, який мав не лише виконувати функцію арени для бою з ворогами, але й водночас передавати візуальне відчуття цілісності, замкненості простору, в якому відбувається дія. З самого початку було вирішено, що основна карта складатиметься з одного острова, оточеного водою. Це рішення давало змогу

логічно обмежити територію пересування гравця і уникнути складних рішень із реалізації країв карти або додаткових бар'єрів.

На початку розробки була створена блок-схема для розуміння, що має бути розроблено, реалізовано, як має виглядати світ та що у ньому має знаходитись. Саму блок-схему зображено на рис. 1.36. У блок-схемі є:

- Концепція рівня;
- Інструменти створення карти;
- Формування ігрової карти;
- Структура основної сцени;
- Налаштування взаємодії;
- Вода та декоративний елемент;

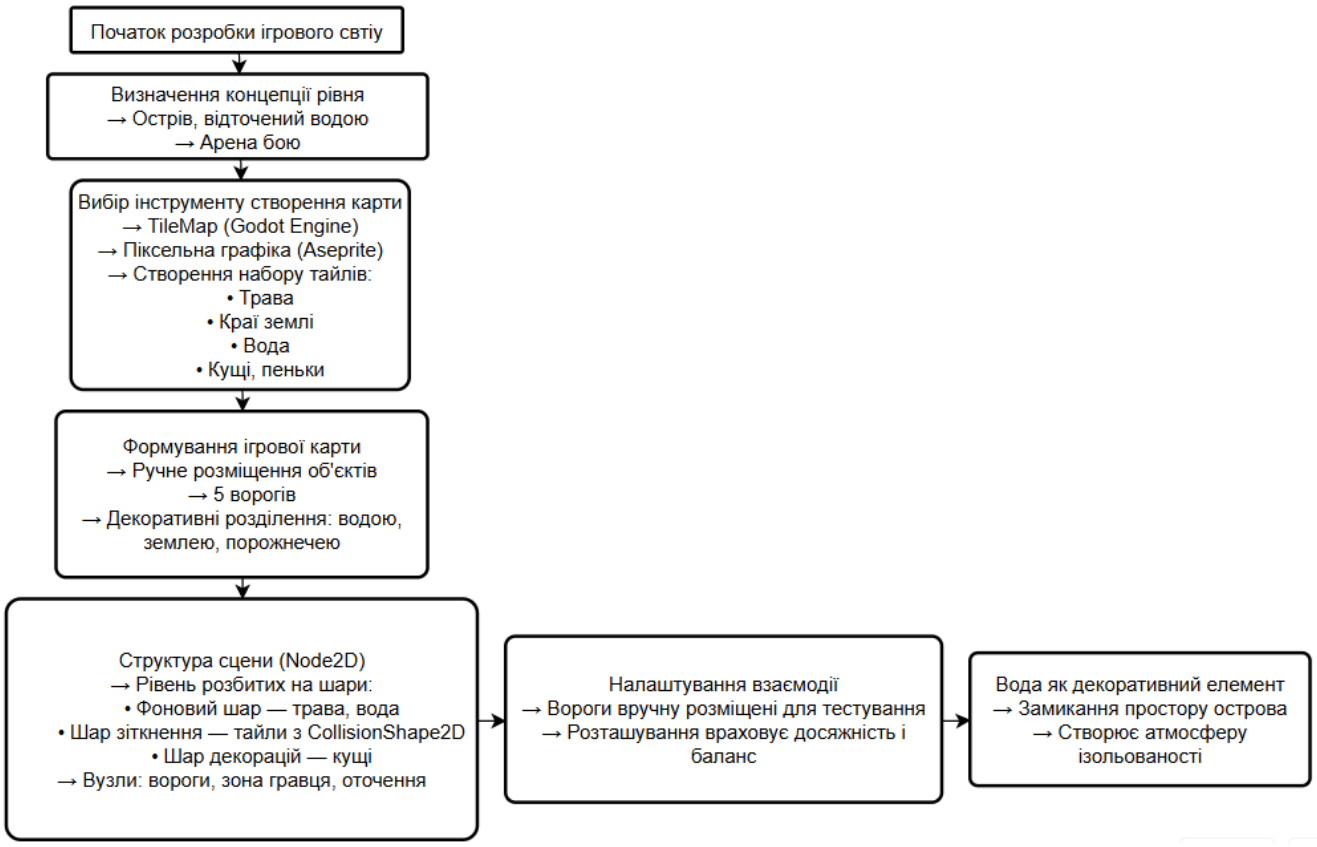


Рисунок 1.36. Блок-схема розробки ігрового світу

Для побудови світу було обрано підхід із використанням TileMap – основного інструмента Godot Engine для формування тайлових рівнів. Тайли були створені в Aseprite вручну, з урахуванням піксельної стилістики гри. Основу становив набір блоків для трави, країв землі, води та декоративних

елементів, як-от кущі. Це дало змогу швидко формувати карту, дотримуючись єдиного стилю, при цьому візуально розділяти ігрові області. На рис. 1.37 зображено готові створені тайли.

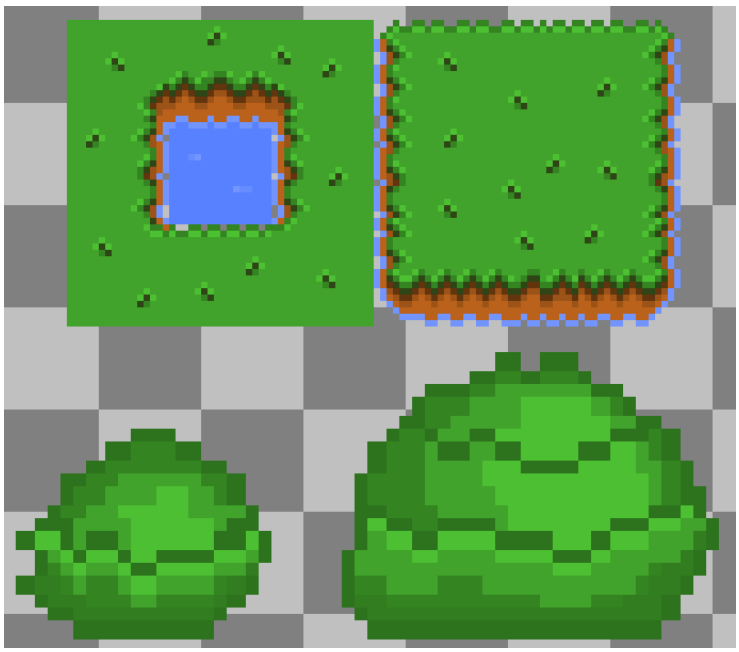


Рисунок 1.37. Скріншот готових спрайтів тайлмапи та оточення

Початково планувалося реалізувати кілька зон для різних ворогів, але через обмеженість часу було зосереджено увагу на одній карті з п’ятьма заздалегідь розставленими ворогами. Кожен з них мав перебувати в різних частинах, щоб запобігти одночасному зіткненню з двома ворогами одночасно. На рис. 1.38 зображено схему структури, як виглядає світ, що знаходиться у ньому та камеру гравця, яка пересувається з ним.

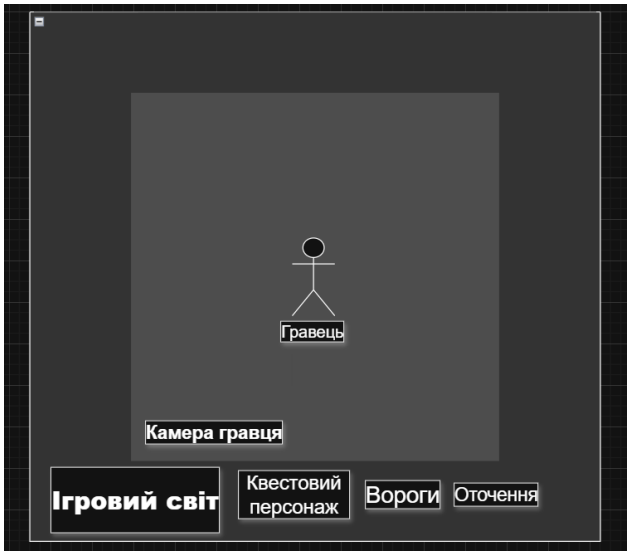


Рисунок 1.38. Схема вигляду ігрового світу і камери гравця

Окрема увага була приділена структурі сцени ігрового світу. Основна сцена рівня складалася з кількох умовних шарів, кожен з яких відповідав за певний тип об’єктів: фоновий шар, шар зіткнень, шар декорацій та об’єктів взаємодії. Така структура дозволила чітко відділити візуальні елементи від технічних ігрових вузлів і уникнути непередбачуваних конфліктів між ними.

Фоновий шар містив виключно тайли, що не впливали на геймплей: трава, край берега, декоративні кущі. Це дозволяло точно налаштувати, які з тайлів блокують рух персонажа, а які – лише декоративні. Наприклад, вода на краю острова мала зіткнення, що повністю обмежувало вихід гравця за межі карти, створюючи умовний “невидимий бар’єр”, який зображено на рис. 1.39.

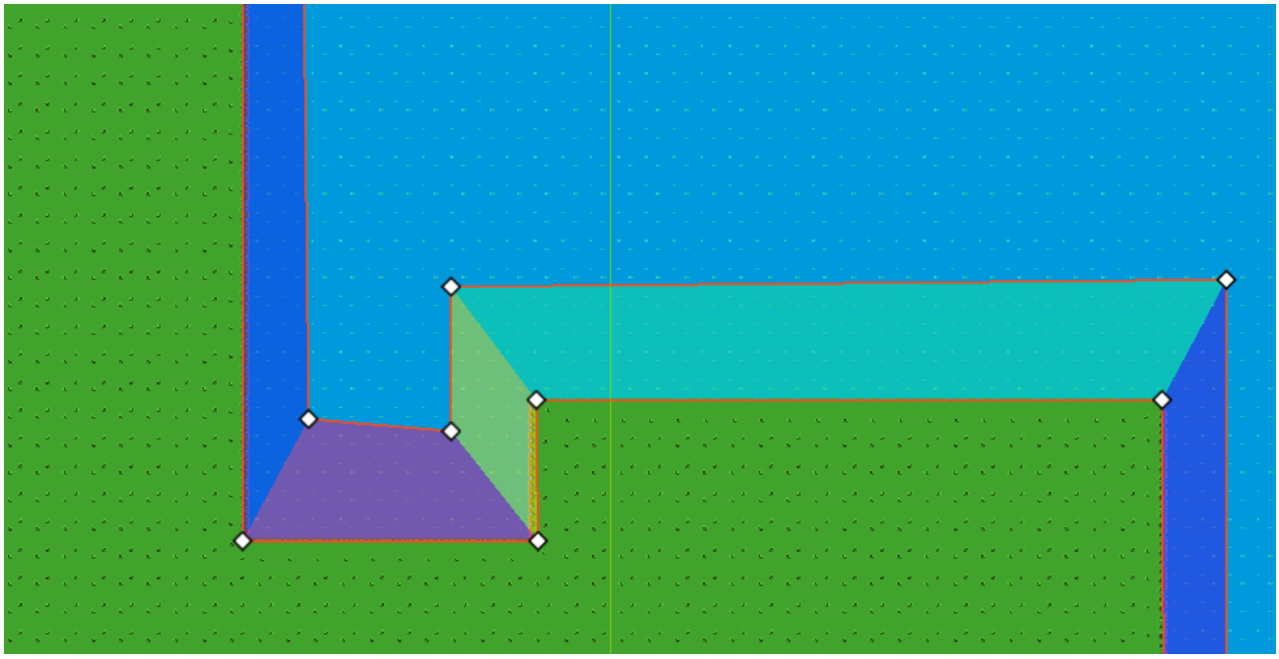


Рисунок 1.39. Скріншот реалізації невидимого бар’єру в грі

Шар декорацій включав у себе об’єкти, які розміщувалися як окремі вузли. Це були кущі які не мали фізичну взаємодію, але візуально збагачували сцену. Навіть декоративний елемент використовувався для формування ігрового ландшафту. Усі об’єкти ігрового світу було організовано в ієрархії у межах головного Node2D. В основній сцені знаходилися вороги, тайлова карта, об’єкти оточення, зона гравця, а також логічні області для скриптів. Таке структурування забезпечило зручність навігації у сцені та полегшило розробку. Крім того, ігрова камера, прив’язана до персонажа, була обмежена межами острова. Це дозволило

уникнути демонстрації порожнього простору поза межами карти та зберегти фокус на активній зоні.

Важливою частиною сцени була також навколишня вода, реалізована як декоративний фон. Вона не мала глибини чи анімацій, але її рівномірне оточення всієї сцени візуально замикало простір. Це рішення підтримувало атмосферу ізольованості та концентрувало увагу на подіях у центрі карти. На рис. 1.40 зображено всі умовні шари та світ загалом, видом зверху.

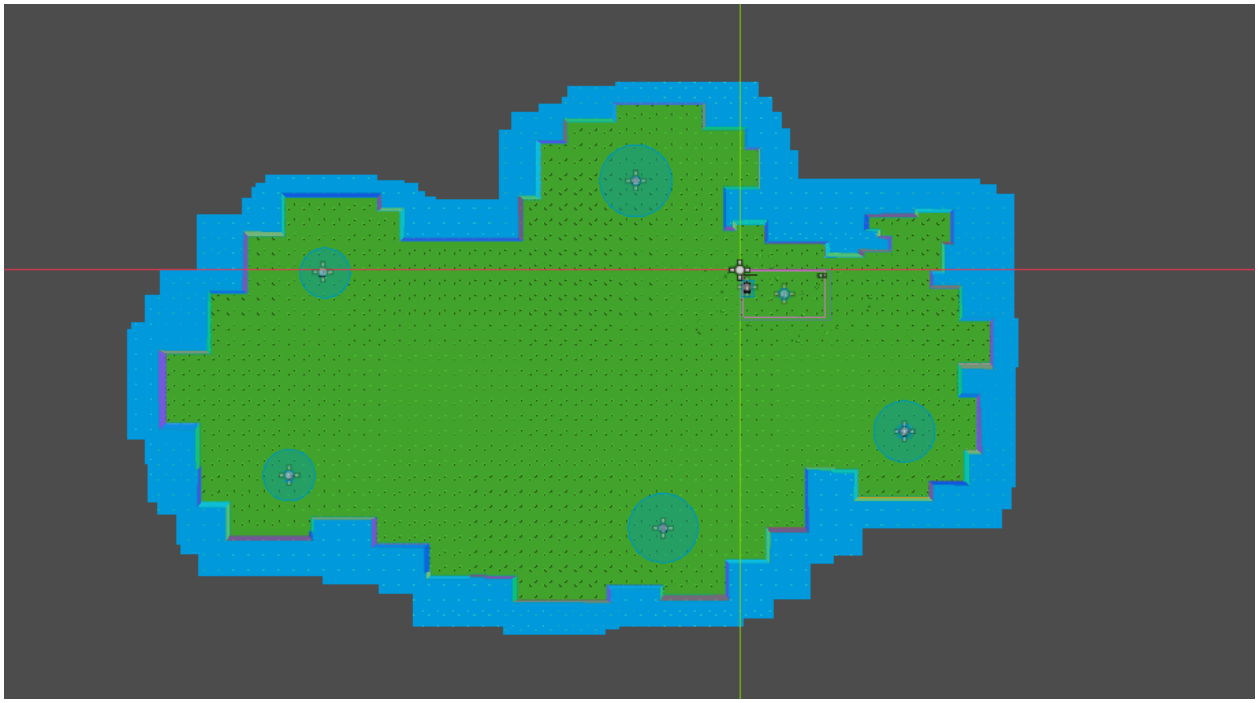


Рисунок 1.40. Скріншот ігрового світу видом зверху

Також для зручності пересування та орієнтації в просторі була розроблена міні-карта, яка допомагає гравцю завчасно побачити ворога, та відреагувати, або просто знайти його. Для початку треба була створена нова сцена CanvasLayer з назвою Minimap яка служила головною, до неї прикріпили вузол SubViewportContainer, а до нього SubViewport який в свою чергу виконував функції умовного контейнеру для іконки гравця і ворогів на міні-карті. Спрайт рамки був доданий за допомогою вузла TextureRect під назвою MinimapFrame. На рис. 1.41 зображено структуру сцени Minimap.

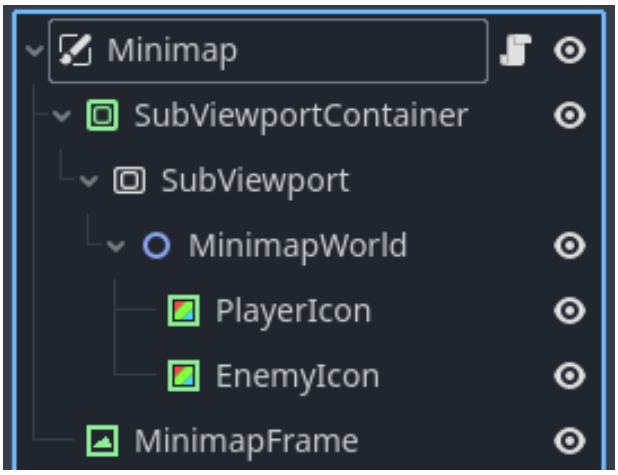


Рисунок 1.41. Скріншот структури сцени міні-карти

Перед написанням коду треба було створити дві нові групи player та enemies, для того щоб у коді можна було розрізняти гравця та ворогів, після цього був написаний і сам код, на рис. 1.42 зображено основну візуальну частину коду, де реалізовується відображення гравця і горогів.

```
func _process(_delta):
    update_player_icon()
    update_enemy_icons()

func update_player_icon():
    if is_instance_valid(player):
        player_icon.position = minimap_center_offset

func update_enemy_icons():
    for enemy in enemy_icons.keys():
        if is_instance_valid(enemy):
            var relative_position = (enemy.global_position - player.global_position) * MINIMAP_SCALE
            enemy_icons[enemy].position = relative_position + minimap_center_offset
        else:
            enemy_icons[enemy].queue_free()
            enemy_icons.erase(enemy)
```

Рисунок 1.42. Скріншот коду візуального відображення гравця і ворогів

1.8 Реалізація квестового персонажу

Ідея створити квестового персонажа з’явилась ще на етапі формування загальної структури гри. Потрібно було вбудувати механіку, яка б задавала гравцю чітку послідовність цілей, поступово знайомила з ворогами, дозволяла контролювати прогрес та додавала систему прокачки. Найпростіше і водночас найефективніше рішення – це статичний персонаж, який видає квести в обмін на

дії. Такий персонаж став логічним центром між боями й сюжетом. Візуальний образ був обраний швидко: старий маг з довгою сивою бородою, який сидить у дерев'яній споруді, на рис. 1.43 зображено спрайт цього персонажу. Його зовнішність і положення не змінюються – це свідомо прийняте рішення, щоб не ускладнювати гру зайвими анімаціями та акцентувати увагу саме на його функціональності як носія завдань.

Рисунок 1.43. Скріншот спрайту квестового персонажу

Його особливістю також було те що він не казав чітко де знаходиться ворог, він тільки давав підказку як цей ворог виглядає, на рис. 1.44 зображено як це відбувається, такий підхід не давав гравцю швидко пройти гру, а також додавав до гри новий елемент інтерактивності.

Рисунок 1.44. Скріншот демонстрації короткої підказки у квесті

Також треба було продумати та розробити те як гравець буде отримувати квест, це мало бути легко та інтуїтивно зрозуміло. Було розроблено систему де гравець, потрапляючи в зону дії, тригерить перемикач, що активує появу літери В. У момент натискання клавіші виконується перевірка, чи можна запустити новий квест, і якщо так – запускається логіка, яка призначає конкретного ворога

як ціль. Після його виконання з’являється повідомлення про виконаний квест та нагорода яку отримує гравець, після цього, гравець може взяти новий квест.

Кожен ворог, якого потрібно перемогти, описаний в унікальний спосіб, тож такий підхід дозволяє легко змінювати або розширювати список завдань – достатньо лише додати новий запис у масив quests. Крім того, у майбутньому така система може бути адаптована під більш складні завдання з умовами, таймерами чи фільтрами доступу. Загалом таких завдань п’ять, кожне – з новим ворогом. Після завершення останнього, квестовий персонаж більше не може видати нове завдання, а вітає гравця з перемогою.

Також нагороду реалізовано в виді прокачки гравця, кожний виконаний квест, збільшує відштовхування з першої атаки, шкоду з другої та уповільнення з третьої. Це зроблено тому що кожен наступний ворог стає сильнішим та небезпечнішим. На рис. 1.45 зображено як це виглядає.

```
Квест: Вбий другого ворога. Це якась незрозуміла субстанція!
Квест виконаний!
Бонуси оновлено після квесту: 1
```

Рисунок 1.45. Скріншот демонстарії виконання квесту та отримання бонусу

Логіка реалізована через вузол Area2D, який має кілька дочірніх елементів: CollisionShape2D для визначення зони взаємодії, AnimatedSprite2D для відображення персонажа та Label, який використовується для відображення літери B, на рис. 1.46 зображено структуру сцени.

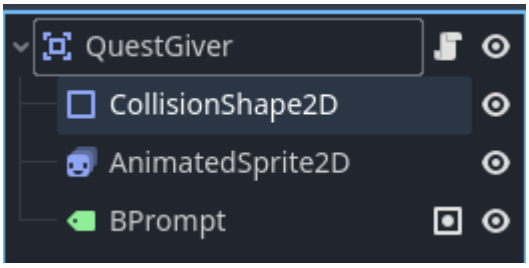


Рисунок 1.46. Скріншот сцени квестового персонажу

Не менш важливою частиною був код, який це все реалізовує, для реалізації відстежування знаходження гравця в зоні взаємодії було використання введення булевої змінної player_in_range яке дозволило це реалізувати. Змінна current_quest фіксувала поточний номер активного квесту. Всі квести зберігалися

у масиві словників, де кожен словник містив назву ворога та опис завдання на рис. 1.47 зображено цей скрипт.

```
var player_in_range: bool = false
var current_quest: int = 0

var quests: Array[Dictionary] = [
    {
        "enemy_name": "Enemy_1",
        "description": "Вбий першого ворога. Він схожий на пекельну сферу!"
    },
    {
        "enemy_name": "Enemy_2",
        "description": "Вбий другого ворога. Це якась незрозуміла субстанція!"
    },
    {
        "enemy_name": "Enemy_3",
        "description": "Вбий третього ворога. Маг який хоче тобі помститися!"
    },
    {
        "enemy_name": "Enemy_4",
        "description": "Вбий четвертого ворога. Цей скелет виглядає моторошно!"
    },
    {
```

Рисунок 1.47. Скріншот скрипту зони взаємодії та відстежування квестів

Не зважаючи на те що квестовий персонаж є статичною моделью, залишати його взагалі без жодної анімації не можна було, це би робило його взагалі ненатхненним об’єктом, тож було додано idle анімацію, яка активувалась через _ready(), також так реалізовується текстова підказка та встановлюються з’єднання сигналів body_entered та body_exited. Це дозволяє виявляти, коли гравець входить у зону дії NPC, скріншот коду зображено на рис. 1.48.

```
func _ready() -> void:
    animated_sprite.play("idle")
    b_prompt.visible = false
    connect("body_entered", Callable(self, "_on_body_entered"))
    connect("body_exited", Callable(self, "_on_body_exited"))
```

Рисунок 1.48. Скріншот коду idle анімації та виявлення зони дії NPC

Перевірку виконання квестів було реалізовано у start_quest(), спочатку перевіряється, чи ще залишилися невиконані квести. Якщо так, то береться словник з інформацією про наступного ворога. Потім серед усіх ворогів на сцені

виконується пошук за іменем. Якщо потрібного ворога знайдено, йому встановлюється мета quest_target, і додається сигнал tree_exited – щоб дізнатися момент його знищення, код цієї перевірки зображено на рис. 1.49.

```
func start_quest() -> void:
    if current_quest >= quests.size():
        print("Всі квести виконані!")
        return

    var quest: Dictionary = quests[current_quest]
    print("Квест: ", quest["description"])

    var enemy_name_expected: String = quest["enemy_name"]
    var enemies = get_tree().get_nodes_in_group("enemies")

    for enemy in enemies:
        if enemy.name == enemy_name_expected and not enemy.has_meta("quest_target"):
            enemy.set_meta("quest_target", true)
            enemy.connect("tree_exited", Callable(self, "_on_enemy_killed"))
            return

    print("Ворог не знайдений!")
```

Рисунок 1.49. Скріншот коду перевірки виконання квестів

Коли ворог знищується (зникає з дерева сцени), викликається _on_enemy_killed(). У цьому методі гравцеві також у вигляді нагороди збільшується статистика через метод increase_stats_after_quest(), і змінна current_quest збільшується, дозволяючи перейти до наступного квесту, на рис. 1.50 зображено код цієї частини.

```
func _on_enemy_killed() -> void:
    print("Квест виконаний!")

    var player = get_tree().get_first_node_in_group("player")
    if player:
        player.increase_stats_after_quest(current_quest)

    current_quest += 1
```

Рисунок 1.50. Скріншот коду виконання квесту

Цей NPC став важливим елементом структури гри, завдяки якому відбувається прогресія, сюжетний рух та поступове ускладнення.

Його створення дозволило об’єднати бойові механіки та елементи сюжету без введення складної діалогової системи. Це було особливо важливо з огляду на обмеження візуального стилю гри та бажання уникнути перевантаження сцени зайвими елементами. Гравець завжди має перед собою чітку мету, яка не потребує додаткових пояснень. Момент, коли на екрані з’являється підказка з літерою B, сигналізує про можливість взаємодії. Це інтуїтивно зрозуміло, не потребує навчання і в той самий час не порушує загального ігрового ритму. Не хотілося робити звичайну аркаду з постійними боями без мети, квестовий персонаж став не лише джерелом завдань, а й умовним провідником по ігровому світу, тож всі завдання які ставились на початку розробки квестового персонажу були реалізовані, повну взаємодію з квестовим персонажем під час гри зображено на рис. 1.51.

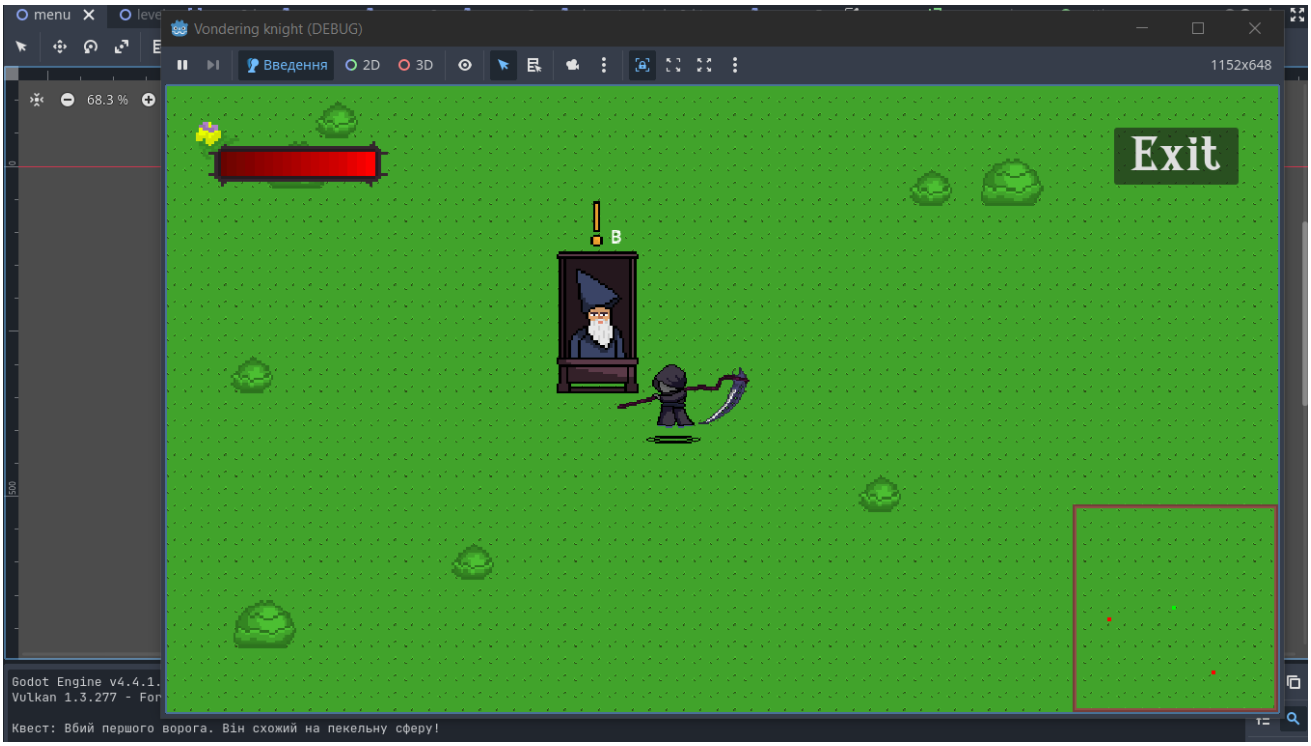


Рисунок 1.51. Скріншот взаємодії з квестовим персонажем

У підсумку, реалізований проект не лише продемонстрував основні етапи створення гри у жанрі top-down action, а й дав змогу на практиці опрацювати повний цикл розробки – від ідеї до робочого прототипу. Отриманий досвід дозволяє надалі масштабувати гру.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проєкті розроблено та реалізовано 2D-гру у жанрі top-down action на програмному рушії Godot Engine. Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

Проведемо розрахунки визначення трудомісткості розробки даного програмного продукту.

2.2 Розрахунок ціни програмного продукту нормативним методом

2.2.1 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку.

У табл. 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт.

Таблиця 2.1. Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП СУБД	2500 – 9800
2. Комплексні системи ведення БД	950 – 7430
3. ПП введення інформації	1060 – 5750
4. ПП оптимізації розрахунків	1300 – 4200
5. ПП автоматизації засобів по каталогу	680 – 7000
6. ПП автоматизованих розрахунків	1300 – 8600
7. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800
8. ПП організації обчислювального процесу	13000 – 10200

Зеленим коліром помічено потрібний варіант.

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл. 2.2.

Таблиця 2.2. Трудомісткість

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262
4.00	283

овіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк = 0,7÷0,8):

$T^a p$ = 229 х 0,8 = 183,2 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

- Розробка технічного завдання:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \qquad (2.1)$$

- Розробка технічного проєкту:

$$T_{ТП} = T^a p \times L_2 \times K_H \qquad (2.2)$$

- Розробка робочого проєкту:

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \qquad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

L_i – питома вага і-го етапу розробки (див. табл. 2.3.);

$K_н$ – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.4.);

$K_т$ – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.5.).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

Тому що розробка системи є ПП, що має аналоги програмних продуктів, то код ступеня новизни для мого ПП – В, а значення коефіцієнта $К_н$ = 0,7. По табл. 2.5, знаючи код ступеня новизни, тепер можна визначити значення питомих коефіцієнтів трудомісткості:

L_1 = 0,12;

L_2 = 0,11;

L_3 = 0,61;

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

У розробленому програмному продукті використовується понад 60 відсотків типових рішень, це значить, що $К_т$ = 0,6.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання:

$$Т_{тз} = Т^а * L_1 * К_н = 183,2*0,12*0,7 = 15,40 \text{ (люд/годин)}$$

Трудомісткість розробки технічного проєкту:

$$T_{тп} = T^{а}*L_2*К_н = 183.2*0,11*0,7 = 14,11 \text{ (люд/годин)}$$

Трудомісткість розробки робочого проєкту:

$$T_{рп} = T^{а}*L_3*К_н*К_т = 183,2*0,61*0,7*0,6 = 46,94 \text{ (люд/годин)}$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: $N_{тз}$ = 2 (стор.), $N_{тп}$ = 21 (стор.), $N_{рп}$ = 21 (стор.), $N_{пз}$ = 31 (стр.). Розрахунок зведений у табл. 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=15,40	$T_{кк}$=0,7*$N_{тз}$= 0,7*2=1,4	$T_{нк}$ =0,15*$N_{тз}$=0,15*2=0,3
2.Розробка ТП	$T_{Ртп}$=14,11	$T_{кк}$=0,7*$N_{тп}$=0,7*21=14,7	$T_{нк}$=0,15*$N_{тп}$=0,15*21=3,15
3.Розробка РП	$T_{Ррп}$=46,94	$T_{кк}$=0,7*$N_{рп}$=0,7*21=14,7	$T_{нк}$=0,15*$N_{рп}$=0,15*21=3,15
4.Розробка ПЗ	$T_{Рпз}$=1,5*$N_{пз}$= 1,5*31 =46,5	$T_{кк}$=0,7*$N_{тз}$=0,7*31=21,7	$T_{нк}$=0,15*$N_{пз}$=0,15*31 =4,65
Усього, в т.ч.:	186,7		
- на розробку	$\Sigma T_р$=122,95		
- контроль керівника		$\Sigma T_{кк}$=52,5	
- нормоконтроль			$\Sigma T_{нк}$=11,25

2.2.2 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПП. Розрахунок основної заробітної плати виконавців приведений у табл. 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2025» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2025 року – 8000 гривень; мінімальну погодинну тарифну ставку – 46.20 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудоміст. робіт, години	Погодинна тар. ставка, грн.	Розрахунок, грн.
1.Розробка ПП	122,95	46.20	5680,29
2.Контроль керівника	52,5	46.20	2425,5
3.Нормоконт-роль	11,25	46.20	519,75
Усього ($З_о$)	-	-	$\Sigma З_о$ = 8625,54

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в табл. 2.8.

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	А4	100	0,4	40,0
Разом	-	-	-	$В_{м\,і}$=40,0
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1 =$ 0,1*72=4
Усього				$В_м=В_{мі}+В_{тр\_з}$=44

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в табл. 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	44	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	8625,54	$З_о$ (див. табл. 2.7)
3. Додаткова заробітна плата	862,55	$Зд = 0{,}1 \times Зо$ =8625,54*0,10
4. Відрахування до єдиного фонду соціального внеску	2087,38	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22*(8625,54+862,55)
5. Накладні витрати	3450,22	$Внак. = 0{,}4 \times Зо$ =0,4*8625,54
6. Повна собівартість	15069,69	$С_{пов}=В_м+З_о+З_д+$Вє.с.в.+Внак.= 44+8625,54+862,55+2087,38+ 3450,22

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$П = (С_п * Р)/100$ (2.4)

Де Р – плановий рівень рентабельності (10-15%).

П = (15069,69*10)/100 = 1506,97 грн.

Оптова ціна визначається по формулі:

$Ц_о = С_{пов} + П$ = 15069,69 + 1506,97 = 16576,66 грн. (2.5)

Податок на додану вартість визначається по наступній формулі:

$ПДВ = 0.2 * Ц_о$ = 0,2 * 16576,66 = 3315,33 грн. (2.6)

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$Ц_р = Ц_о + ПДВ$ = 16576,66 +3315,33 = 19891,99 грн. (2.7)

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Умови і чинники, що впливають на безпеку праці розробника

Професійна діяльність розробника програмного забезпечення передбачає тривалу роботу за комп'ютером у статичній позі та високий рівень психоемоційного навантаження. Хоча така робота не є небезпечною з точки зору фізичних ризиків, вона супроводжується рядом чинників, які можуть негативно впливати на стан здоров'я та працездатність.

До основних шкідливих впливів належать: напруження органів зору через фокусування на екрані, м'язове напруження та дискомфорт внаслідок незручної пози, недостатній повітрообмін і погане освітлення. Важливими також є вплив шуму від техніки, накопичення статичної електрики, наявність електричних приладів без захисного заземлення та надмірне використання подовжувачів. Небезпечні ситуації можуть виникати при несправності електрообладнання або порушенні правил його експлуатації. Пошкодження кабелів, перевантаження розеток, відсутність засобів захисту збільшують ризик коротких замикань або ураження електричним струмом.

Шкідливі чинники можуть проявлятись поступово: зниження зору, болі в спині й шиї, синдром хронічної втоми, підвищена дратівливість, зниження концентрації. Ці ефекти посилюються за умов відсутності регулярних перерв, погано організованого робочого простору або неправильного положення тіла під час роботи. Умови праці програміста потребують постійної уваги до мікроклімату, ергономіки, освітлення, електробезпеки та режиму праці.

3.2 Вимоги до організації простору та мікроклімату робочого середовища

Раціональна організація простору та параметрів навколишнього середовища є основою безпечної та ефективної праці розробника. Робоче місце має бути розташоване в окремому приміщенні з вентиляцією. Рекомендована площа – не менше 6 м², об'єм повітря – не менше 20 м³ на одного працівника.

3.2.1 Температурний режим і вологість

Температура повітря повинна підтримуватись у межах 22–25 °C, вологість 40–60 відсотків. Відхилення від цих норм може призвести до дискомфорту, сонливості або погіршення самопочуття.

3.2.2 Повітрообмін і вентиляція

Приміщення має бути обладнане природною або механічною вентиляцією. У разі використання кондиціонерів важливо уникати протягів і забезпечити рівномірний розподіл повітря.

3.2.3 Освітлення робочої зони

Робоче місце повинне мати змішане освітлення: природне – через вікна з захисними жалюзі, штучне – з освітленістю не менше 300 лк.

3.2.4 Рівень шуму

Максимально допустимий рівень шуму – до 50 дБ. Для зменшення шумового фону використовують звукопоглинаючі матеріали та техніку з низьким рівнем звукового навантаження.

3.2.5 Колірна гамма та оздоблення

Стіни та меблі повинні мати спокійні, матові кольори без блиску. Яскраві контрасти, складні візерунки чи глянцеві поверхні не допускаються.

3.2.6 Психоемоційна складова середовища

Комфортна атмосфера впливає на загальний настрій працівника. Позитивно сприймаються природні кольори, рослини, порядок на робочому місці, а також наявність умов для короткого відпочинку.

3.3 Ергономіка та технічне облаштування робочого місця програміста

Ефективність та безпечність праці програміста значною мірою залежить від правильного облаштування його робочого місця. Основні вимоги стосуються зручності положення тіла, розміщення обладнання та якості меблів. При

недотриманні цих вимог зростає ризик професійних розладів – болю в спині, порушення кровообігу, зорового перенапруження.

3.3.1 Вимоги до комп'ютерного місця

Монітор має бути встановлений на відстані 50–70 см від очей, нижній край екрана – на рівні погляду або трохи нижче. Клавіатура – на відстані 10–30 см від краю столу, миша – поруч, на одному рівні.

Розміщення обладнання має забезпечувати пряму поставу та рівномірний розподіл навантаження. Всі елементи керування повинні знаходитись у зоні досяжності без необхідності повертати корпус або нахилятися.

3.3.2 Меблі та обладнання

Стіл має бути заввишки 700–750 мм, з матовою поверхнею. Глибина – не менше 800 мм, щоб монітор був на достатній відстані від очей. Стілець повинен регулюватися по висоті, мати опору для попереку, підлокітники й стійку основу. Для профілактики втоми рекомендується наявність підставки для ніг, а також періодичне змінювання пози тіла під час роботи. Усі меблі мають відповідати антропометричним параметрам користувача.

3.4 Технічна безпека та запобігання аварійним ситуаціям

Робоче місце програміста включає численні електротехнічні пристрої, тому питання безпеки під час їх експлуатації мають пріоритетне значення. Важливо враховувати як захист від електричного струму, так і попередження ризиків, пов'язаних із пожежами та накопиченням статичної електрики.

3.4.1 Електробезпека

Всі пристрої, що підключаються до мережі, мають бути справними, заземленими та сертифікованими. Забороняється використовувати кабелі з пошкодженою ізоляцією, підключати декілька потужних пристроїв до одного подовжувача без запобіжників. Усі розетки повинні мати захист від коротких замикань. За наявності сторонніх запахів, іскор або перегріву корпусу

обладнання – живлення має бути негайно вимкнено, на рис. 3.1 зображено який може мати вигляд перегрітий ноутбук.

Рисунок 3.1 Вигляд перегрітого ноутбука

3.4.2 Статична електрика

При терті синтетичних матеріалів або недостатній вологості повітря може накопичуватись статичний заряд, що становить небезпеку для чутливої техніки. Для зменшення цього впливу використовують антистатичні килимки, заземлення поверхонь і підтримують вологість не нижче 40 відсотків.

Також не рекомендується працювати в одязі зі 100 відсоткової синтетики. Перевагу слід надавати натуральним або змішаним тканинам.

3.4.3 Пожежна безпека

Усі приміщення, де розміщені комп’ютери та інше офісне обладнання, повинні відповідати категорії «Д» за пожежною безпекою, тобто належати до приміщень з низьким ризиком займання. У таких кімнатах обов’язково мають бути доступними засоби пожежогасіння – вуглекислотні або порошкові вогнегасники, схеми евакуації, інструкції з безпечного залишення приміщення. Важливо забезпечити вільний доступ до вимикачів живлення, щитків та евакуаційних проходів. Наявність пожежної сигналізації або автономних димових сповіщувачів значно зменшує ризик пізнього реагування у випадку пожежі.

Робоче місце програміста, насичене електронною технікою, потенційно належить до зони підвищеної пожежної небезпеки. Перевантаження електромережі, використання несправних подовжувачів або неправильна експлуатація обладнання можуть призвести до короткого замикання та загоряння. Щоб уникнути таких ситуацій, слід регулярно перевіряти стан розеток, шнурів живлення, вилок, а також не допускати підключення кількох потужних пристроїв до одного джерела.

Кожен працівник повинен бути проінструктований щодо дій у надзвичайній ситуації, знати, де розташовані вогнегасники та як ними користуватись. У разі виявлення загоряння першочергово слід вимкнути живлення, викликати пожежну службу, а гасіння проводити лише за відсутності прямої загрози для життя.

Схема евакуації повинна бути розміщена на видному місці з чітко зазначеним напрямком виходу та місцем збору персоналу. Наявність чітких інструкцій та регулярне інформування працівників дозволяють діяти впевнено й уникати паніки в критичних ситуаціях, на рис 3.2 зображено який вигляд може мати схема евакуації.

Рисунок 3.2. Вигляд схеми евакуації

ВИСНОВКИ

У межах дипломного проєкту було створено повноцінну 2D-гру Vondering Knight у жанрі top-down action з виглядом зверху, яка поєднує бойові дії, квестову систему та стилізовану піксельну графіку. Була побудована логічна завершена ігрова структура, де гравець проходить послідовні бої із ворогами в межах ізольованої локації.

Одним із центральних напрямів проєкту стала розробка бойової системи, яка включає три типи атак: звичайну з відштовхуванням ворога, вогняну з ефектом підпалу та водяну з уповільненням. Ці механіки були реалізовані засобами GDScript з урахуванням фізичних взаємодій, анімацій і системи сигналів.

Під час розробки була створена сцено-вузлова структура, яка забезпечила чіткий поділ між логікою гравця, ворогів, середовища та інтерфейсу. Усі об’єкти організовані в групи, а система сигналів дозволила ефективно обробляти зіткнення, отримання шкоди, респавн і події, пов’язані з прогресом гравця.

Інтерфейс гри охоплює шкалу здоров’я, індикатори стану, відображення нанесеної шкоди та мінікарту. Усі графічні ресурси створені вручну в Aseprite, що надало грі виразного візуального стилю. Тайлова карта та анімоване оточення сформували завершене ігрове середовище, у якому гравець орієнтується завдяки зрозумілій побудові простору. Квестова система реалізована у вигляді послідовних завдань, що формуються через спеціального NPC, який стежить за знищенням потрібних ворогів. Після виконання кожного завдання гравець отримує покращення характеристик, що підсилює відчуття прогресії.

Створена гра охоплює всі основні компоненти завершеного ігрового прототипу. Проєкт є прикладом самостійного вирішення задач у сфері геймдизайну, програмування, проєктування взаємодії та візуального стилю. Усі реалізовані системи можуть бути розширені, доповнені новими рівнями, механіками або перетворені в повноцінний реліз.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ляшенко. О.М. Розробка комп'ютерних ігор. – Херсон: ФОП Вишемирський В.С., 2018. – 220 с.
2. Інститут прикладної педагогіки. Технології створення освітніх комп'ютерних ігор та XR-застосунків: конспект лекцій. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 194 с.
3. Колектив авторів (ІПП, КПІ). Основи розробки комп'ютерних ігор: електронний посібник. – Київ: ІПП, 2020. – 120 с.
4. Серебріян О.Ю. Проєктування комп'ютерних ігор для навчання: методичний довідник. – Одеса: ОНПУ, 2017. – 23 с.
5. Навчальна дисципліна «Основи комп'ютерних ігор та ігрових програм». – Харків: КПІ-KHARKOV, 2021. – ~60 с.
6. Робоча програма дисципліни «Ігровий дизайн». – Київ: Університет Грінченка, 2024. – 120 с.
7. Шерстюк В.Г. Основи розробки комп'ютерних ігор: електронний навчальний посібник. – Херсон: ФОП Вишемирський В.С., 2018. – 210 с.
8. Кормановський С.І., Слободянюк О.В., Пащенко В.Н. Інженерна та комп'ютерна графіка: навчальний посібник. – Вінниця: ВНТУ, 2006. – 118 с.
9. Офіційна документація Godot Engine [Електронний ресурс]. – Режим доступу: https://docs.godotengine.org/ (Дата звернення: 14.04.2025).
10. Офіційний сайт Aseprite [Електронний ресурс]. – Режим доступу: https://www.aseprite.org/ (Дата звернення: 12.04.2025).
11. Сервіс створення діаграм diagrams.net [Електронний ресурс]. – Режим доступу: https://app.diagrams.net/ (Дата звернення: 20.05.2025).
12. Онлайн-сервіс моделювання Visual Paradigm Online [Електронний ресурс]. – Режим доступу: https://online.visual-paradigm.com (Дата звернення: 20.05.2025).

ДОДАТОК А. Лістинг основних сцен гри мовою GDScript

Скрипт player.gd

```
extends CharacterBody2D

const SPEED = 200
const ATTACK_DAMAGE = 10
const FIRE_DAMAGE = 8
const FIRE_DURATION = 3.0
const FIRE_INTERVAL = 1.0
const SLOW_DURATION = 2.0

@export var knockback_strength := 200
@export var knockback_timer := 0.2

var attack_knockback_bonus := 0
var fire_damage_bonus := 0
var slow_duration_bonus := 0.0

const HP_BG_TEXTURE = preload("res://Sprite/Hp_bg.png")
const HP_FILL_TEXTURE = preload("res://Sprite/Hp_fill.png")
const DAMAGE_NUMBER_SCENE = preload("res://label.tscn")
const RESPAWN_ICON_SCENE = preload("res://respawm_icon.tscn")

var max_hp = 100
var current_hp = 100
var is_attacking = false
var knockback_vector = Vector2.ZERO
var knockback_time_left := 0.0
var current_attack_type = ""

@onready var animated_sprite = $AnimatedSprite2D
@onready var hp_bar = $HealthBarLayer/HealthBar
@onready var attack_area = $AttackArea

func _ready():
  hp_bar.texture_under = HP_BG_TEXTURE
  hp_bar.texture_progress = HP_FILL_TEXTURE
  update_health_bar()
  attack_area.monitoring = false

func _physics_process(delta):
  if knockback_time_left > 0:
        velocity = knockback_vector
        move_and_slide()
        knockback_time_left -= delta
        return

  if is_attacking:
        velocity = Vector2.ZERO
        move_and_slide()
        return

  var input_vector = Vector2(
        Input.get_action_strength("Right") - Input.get_action_strength("Left"),
        Input.get_action_strength("Down") - Input.get_action_strength("Up")
  ).normalized()
```

```
    velocity = input_vector * SPEED
    move_and_slide()
    update_animation(input_vector)

    if not is_attacking:
        if Input.is_action_just_pressed("attack"):
            current_attack_type = "attack"
            start_attack("attack")
        elif Input.is_action_just_pressed("attack_1"):
            current_attack_type = "attack_1"
            start_attack("attack_1")
        elif Input.is_action_just_pressed("attack_2"):
            current_attack_type = "attack_2"
            start_attack("attack_2")

func update_animation(direction: Vector2):
    if is_attacking:
        return
    elif direction == Vector2.ZERO:
        animated_sprite.play("idle")
    else:
        animated_sprite.play("walk")
        if direction.x != 0:
            animated_sprite.flip_h = direction.x < 0

func start_attack(animation_name: String):
    is_attacking = true
    animated_sprite.play(animation_name)
    attack_area.monitoring = true
    await animated_sprite.animation_finished
    attack_area.monitoring = false
    is_attacking = false
    current_attack_type = ""

func _on_attack_area_body_entered(body: Node2D):
    if body == self:
        return
    if is_attacking and body.has_method("take_damage"):
        match current_attack_type:
            "attack":
                body.take_damage(ATTACK_DAMAGE)
                if body.has_method("apply_knockback"):
                    body.apply_knockback(global_position, knockback_strength + attack_knockback_bonus)
            "attack_1":
                if body.has_method("apply_burning"):
                    body.apply_burning(
                        FIRE_DAMAGE + fire_damage_bonus,
                        FIRE_INTERVAL,
                        FIRE_DURATION
                    )
            "attack_2":
                if body.has_method("apply_slow"):
                    body.apply_slow(SLOW_DURATION + slow_duration_bonus)

func take_damage(amount: int):
    current_hp = max(current_hp - amount, 0)
    show_damage_number(amount)
    update_health_bar()
    if current_hp == 0:
```

```
        die()

func apply_knockback(from_position: Vector2, force: float = knockback_strength):
  var direction = (global_position - from_position).normalized()
  knockback_vector = direction * force
  knockback_time_left = knockback_timer

func update_health_bar():
  hp_bar.max_value = max_hp
  hp_bar.value = current_hp

func die():
  animated_sprite.play("die")
  spawn_respawn_icon()
  queue_free()

func show_damage_number(amount: int):
  var label = DAMAGE_NUMBER_SCENE.instantiate()
  label.text = str(amount)
  get_parent().add_child(label)
  label.global_position = global_position
func spawn_respawn_icon():
  var icon = RESPAWN_ICON_SCENE.instantiate()
  icon.global_position = global_position
  get_parent().add_child(icon)
func increase_stats_after_quest(quest_index: int):
  match quest_index:
        0:
              attack_knockback_bonus += 20
        1:
              fire_damage_bonus += 2
        2:
              slow_duration_bonus += 0.2
        3:
              knockback_strength += 50
  print("Бонуси оновлено після квесту: ", quest_index)
```

Скрипт enemy.gd

```
extends CharacterBody2D

const BASE_SPEED = 50
const ATTACK_RANGE = 100
const DAMAGE = 20
const DAMAGE_NUMBER_SCENE = preload("res://label.tscn")

@export var max_hp := 100
@export var knockback_strength := 150
@export var knockback_timer := 0.2
@export var player_path: NodePath

var speed := BASE_SPEED
var current_hp = max_hp
var player_in_range = false
var is_dying = false
var knockback_vector = Vector2.ZERO
var knockback_time_left := 0.0

var burn_timer: Timer
var slow_timer: Timer
var original_speed := BASE_SPEED
```

```
@onready var player = get_node(player_path)
@onready var animated_sprite = $AnimatedSprite2D
@onready var detection_area = $DetectionArea
@onready var attack_timer = $AttackTimer
@onready var health_bar = $HealthBar/TextureProgressBar

func _ready():
  current_hp = max_hp
  detection_area.connect("body_entered", _on_body_entered)
  detection_area.connect("body_exited", _on_body_exited)
  attack_timer.one_shot = true
  animated_sprite.play("idle")

  burn_timer = Timer.new()
  burn_timer.one_shot = false
  burn_timer.connect("timeout", _on_burn_tick)
  add_child(burn_timer)

  slow_timer = Timer.new()
  slow_timer.one_shot = true
  slow_timer.connect("timeout", _on_slow_end)
  add_child(slow_timer)
  update_health_bar()
func _physics_process(delta):
  if is_dying or player == null:
        return

  if knockback_time_left > 0:
        velocity = knockback_vector
        move_and_slide()
        knockback_time_left -= delta
        return

  if not player_in_range:
        animated_sprite.play("idle")
        return
  var distance = global_position.distance_to(player.global_position)
  if distance > ATTACK_RANGE:
        var direction = (player.global_position - global_position).normalized()
        velocity = direction * speed
        move_and_slide()
        animated_sprite.play("walk")
        animated_sprite.flip_h = direction.x < 0
  else:
        velocity = Vector2.ZERO
        move_and_slide()
        attack()

func attack():
  if not attack_timer.is_stopped():
        return
  animated_sprite.play("attack")
  if player.has_method("take_damage"):
        player.take_damage(DAMAGE)
        if player.has_method("apply_knockback"):
              player.apply_knockback(global_position)
  attack_timer.start(1.0)
func take_damage(amount: int):
  if is_dying:
```

```
            return
    current_hp -= amount
    show_damage_number(amount)
    update_health_bar()
    if current_hp <= 0:
            die()
func update_health_bar():
    if health_bar:
            health_bar.max_value = max_hp
            health_bar.value = current_hp
func apply_knockback(from_position: Vector2, force: float = knockback_strength):
    var direction = (global_position - from_position).normalized()
    knockback_vector = direction * force
    knockback_time_left = knockback_timer

func apply_burning(damage: int, interval: float, duration: float):
    var ticks = int(duration / interval)
    burn_timer.start(interval)
    burn_timer.set_meta("damage", damage)
    burn_timer.set_meta("ticks_left", ticks)

func _on_burn_tick():
    var ticks_left = burn_timer.get_meta("ticks_left")
    var damage = burn_timer.get_meta("damage")
    take_damage(damage)
    ticks_left -= 1
    if ticks_left <= 0:
            burn_timer.stop()
    else:
            burn_timer.set_meta("ticks_left", ticks_left)
func apply_slow(duration: float):
    speed = speed / 2
    slow_timer.start(duration)

func _on_slow_end():
    speed = original_speed

func die():
    is_dying = true
    animated_sprite.play("die")
    await animated_sprite.animation_finished

    if player != null and player.has_method("apply_quest_reward"):
            player.apply_quest_reward()

    queue_free()

func _on_body_entered(body):
    if body.name == "Player":
            player_in_range = true

func _on_body_exited(body):
    if body.name == "Player":
            player_in_range = false

func show_damage_number(amount: int):
    var label = DAMAGE_NUMBER_SCENE.instantiate()
    label.text = str(amount)
    get_parent().add_child(label)
    label.global_position = global_position
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка гри Vondering Knight з різноманітними механіками ближнього бою

ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТА ГРУПИ 4РП-08: ПЕСАРА ІВАНА ПЕТРОВИЧА
КЕРІВНИК: ДЖАБРАЇЛОВ Д.В.

Особливості обраного жанру

Вигляд top-down action гри

The Legend of Zelda: A Link to the Past

Вибір засобів розробки гри

Godot Engine

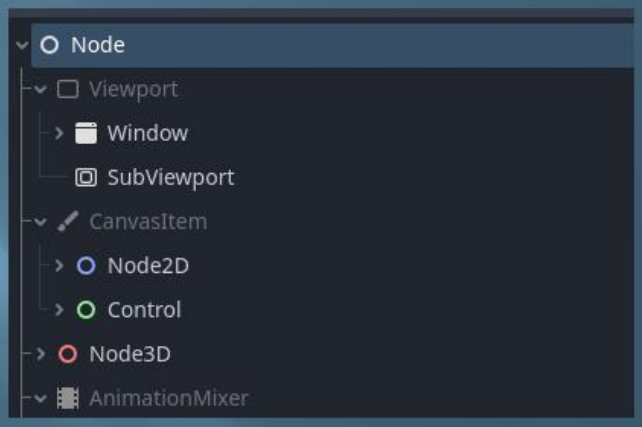
```gdscript
func start_attack(animation_name: String):
    is_attacking = true
    animated_sprite.play(animation_name)
    attack_area.monitoring = true
    await animated_sprite.animation_finished
    attack_area.monitoring = false
    is_attacking = false
    current_attack_type = ""
```

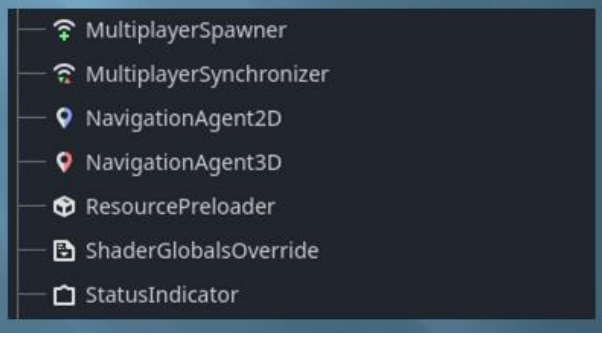
```gdscript
func take_damage(amount: int):
    current_hp = max(current_hp - amount, 0)
    show_damage_number(amount)
    update_health_bar()
    if current_hp == 0:
        die()

func apply_knockback(from_position: Vector2, force: float = knockback_strength):
    var direction = (global_position - from_position).normalized()
    knockback_vector = direction * force
    knockback_time_left = knockback_timer
```

Приклад GDScript

Вибір засобів розробки гри

Aseprite



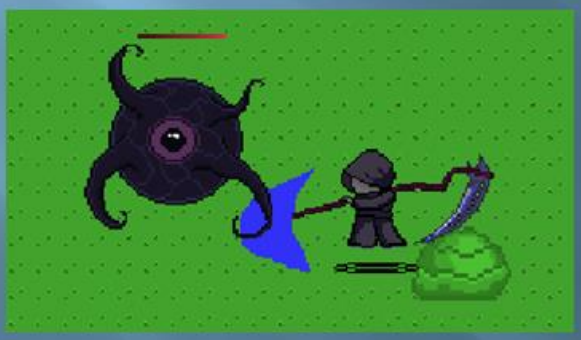


Вузли в Godot Engine

Механіки ближнього бою

Перша атака

Друга атака



Третя атака

Область атаки гравця

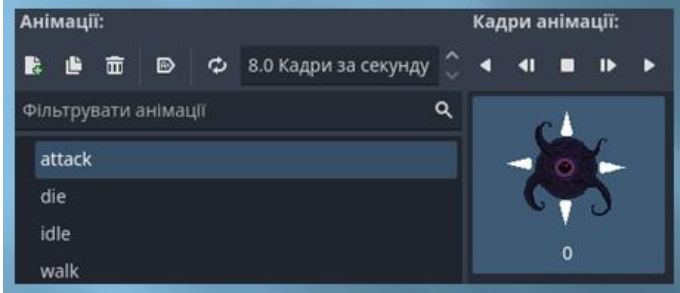
Механіки ближнього бою

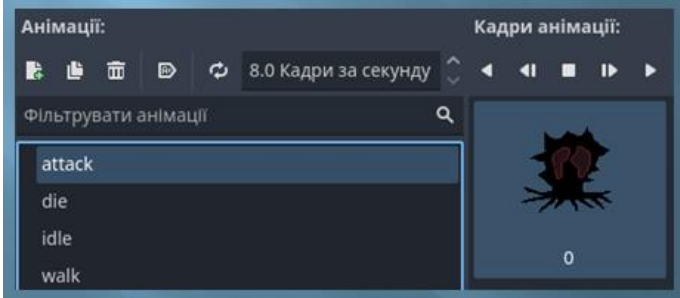
Код ближнього бою гравця

```
func _on_attack_area_body_entered(body: Node2D):
    if body == self:
        return
    if is_attacking and body.has_method("take_damage"):
        match current_attack_type:
            "attack":
                body.take_damage(ATTACK_DAMAGE)
                if body.has_method("apply_knockback"):
                    body.apply_knockback(global_position, knockback_strength + attack_knockback_bonus)
            "attack_1":
                if body.has_method("apply_burning"):
                    body.apply_burning(
                        FIRE_DAMAGE + fire_damage_bonus,
                        FIRE_INTERVAL,
                        FIRE_DURATION
                    )
            "attack_2":
                if body.has_method("apply_slow"):
                    body.apply_slow(SLOW_DURATION + slow_duration_bonus)
```

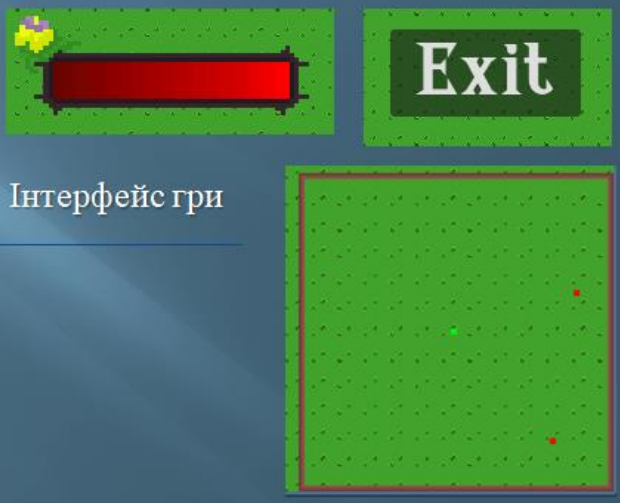
```
    if not is_attacking:
        if Input.is_action_just_pressed("attack"):
            current_attack_type = "attack"
            start_attack("attack")
        elif Input.is_action_just_pressed("attack_1"):
            current_attack_type = "attack_1"
            start_attack("attack_1")
        elif Input.is_action_just_pressed("attack_2"):
            current_attack_type = "attack_2"
            start_attack("attack_2")
```

Процес розробки гри



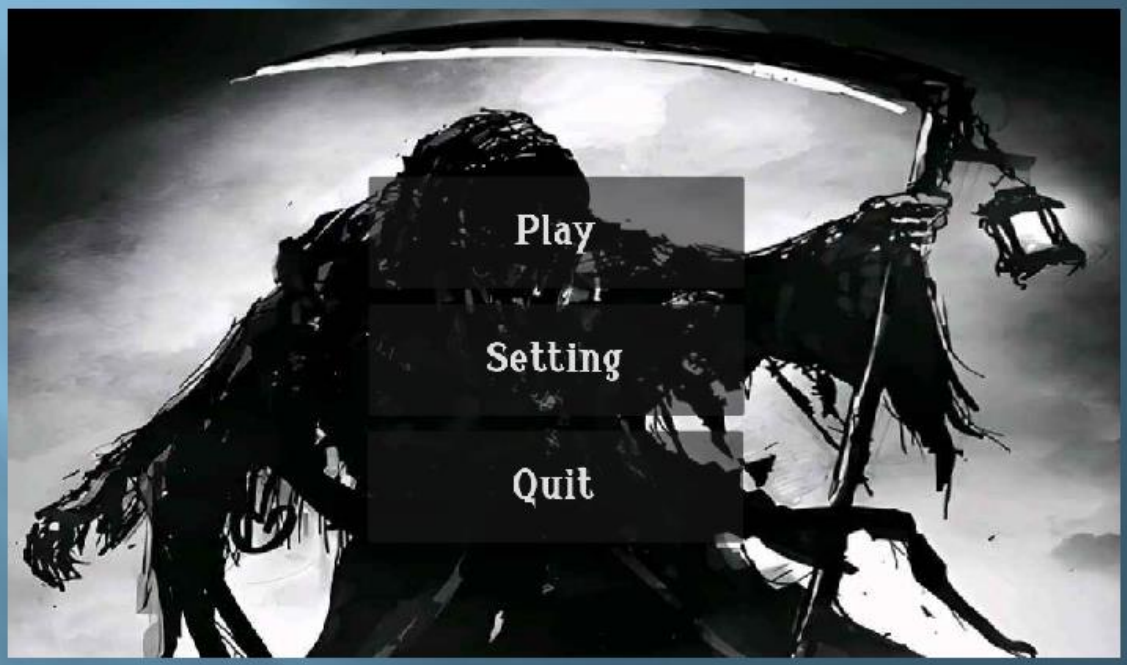


Анімації ворогів



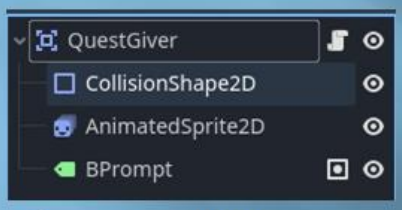
Інтерфейс гри

Процес розробки гри



Головне ігрове меню

Розробка квестового персонажу



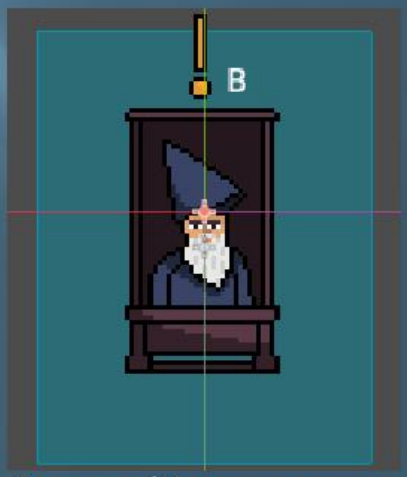
Структура сцени

```
Квест: Вбий другого ворога. Це якась незрозуміла субстанція!
Квест виконаний!
Бонуси оновлено після квесту: 1
```

```
Квест: Вбий першого ворога. Він схожий на пекельну сферу!
```

Вигляд квестів

Спрайт квестового персонажу



Зона в якій можна взяти квест

Розробка квестового персонажу

```gdscript
@onready var animated_sprite: AnimatedSprite2D = $AnimatedSprite2D
@onready var b_prompt: Label = $BPrompt

var player_in_range: bool = false
var current_quest: int = 0
```

```gdscript
var quests: Array[Dictionary] = [
    {
        "enemy_name": "Enemy_1",
        "description": "Вбий першого ворога. Він схожий на пекельну сферу!"
    },
```

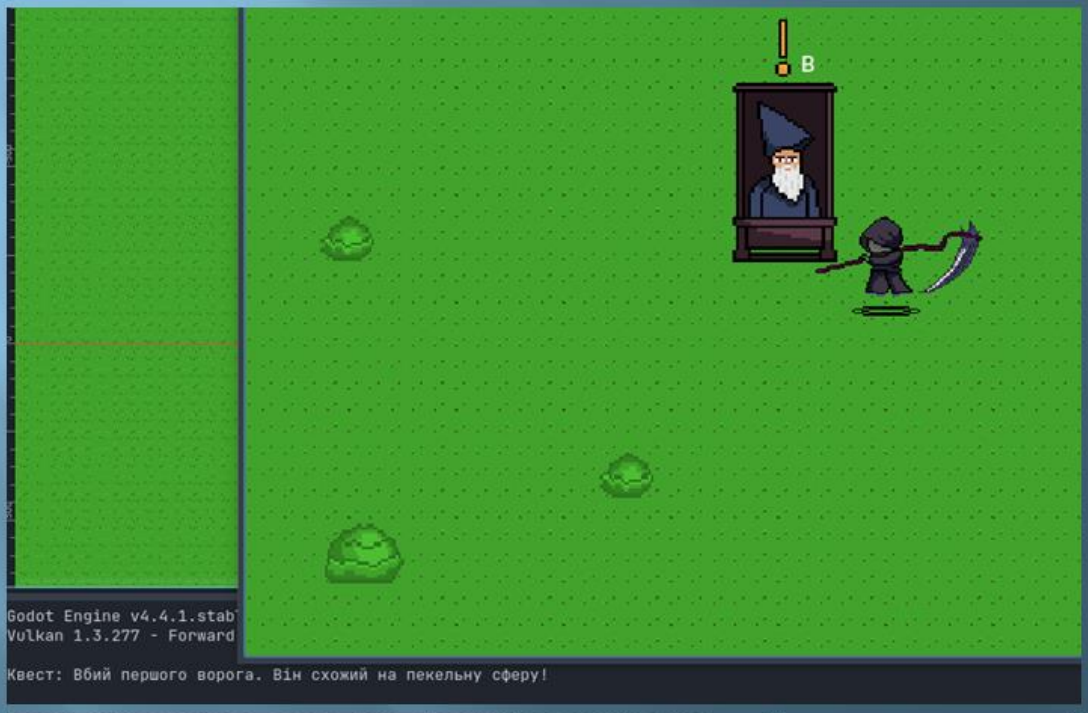
```gdscript
func _on_body_entered(body: Node) -> void:
    if body.is_in_group("player"):
        player_in_range = true
        b_prompt.visible = true

func _on_body_exited(body: Node) -> void:
    if body.is_in_group("player"):
        player_in_range = false
        b_prompt.visible = false
```

Код квестового персонажу

Тестування

Тестування ближнього бою

Тестування



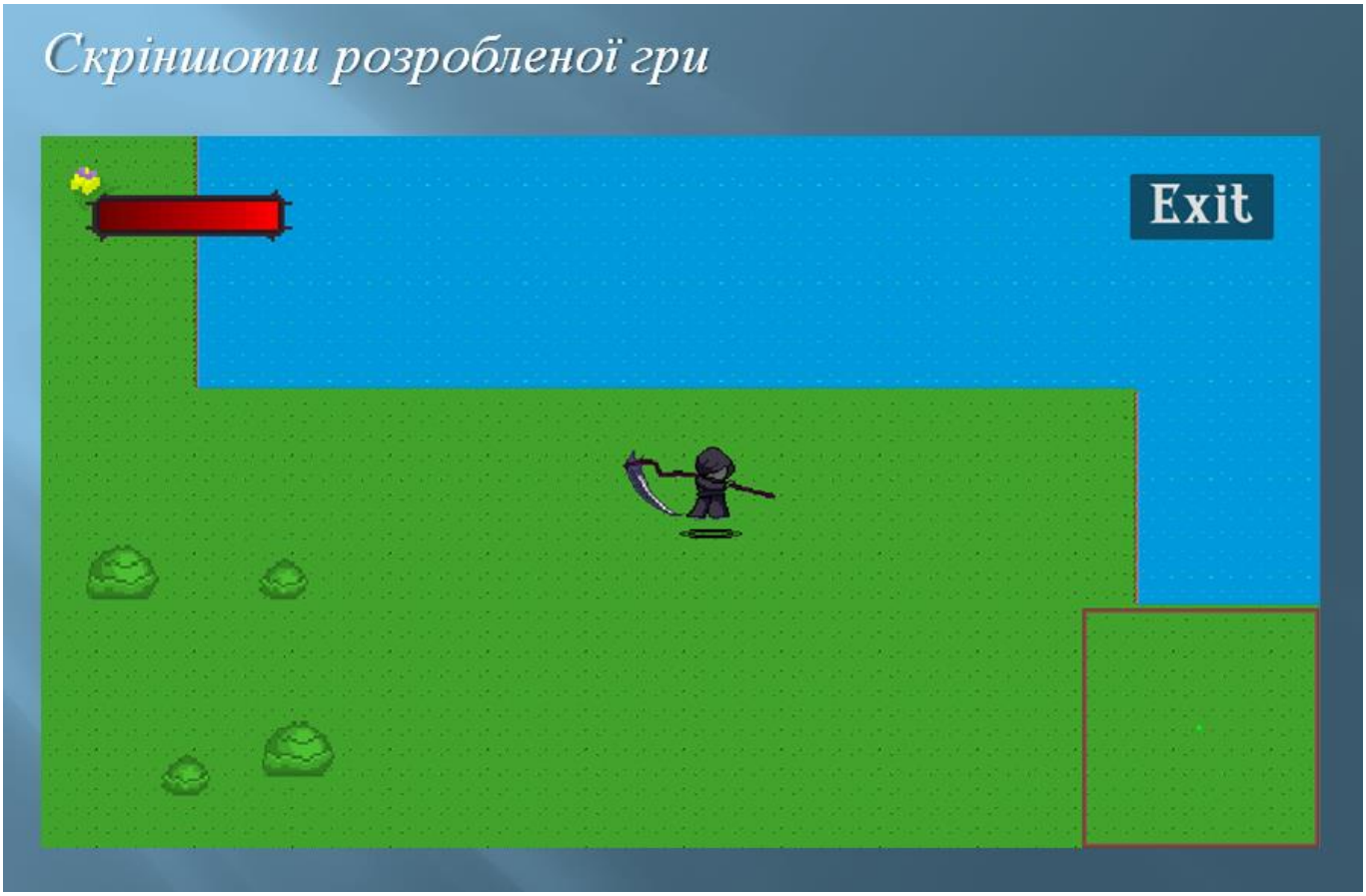

```
Квест: Вбий останнього ворога. Будь обережний в останній битві
Квест виконаний!
Бонуси оновлено після квесту: 4
Всі квести виконані!
```

Тестування взаємодії з квестовим персонажем та проходження гри

Скріншоти розробленої гри

Скріншоти розробленої гри

Скріншоти розробленої гри



ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Песара Івана Петровича

(прізвище, ім`я та по батькові)

Спеціальність *121 " Інженерія програмного забезпечення "*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Джабраілов Дмитро Володимирович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка гри Vondering Knight з різноманітними механіками ближнього бою*

Обсяг розрахунково-пояснювальної записки *75* сторінок

Обсяг графічної (презентаційної) частини *15* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню *Представлений дипломний проєкт повністю відповідає виданому завданню та заявленій меті. Тематика роботи є актуальною в межах галузі цифрових медіа та розробки ігор, а обрана технологія – використання рушія Godot Engine для створення 2D-гри, повністю узгоджується з технічними вимогами проєкту.*

б) характеристика виконання кожного розділу дипломного проекту (роботи) *Дипломна робота складається зі вступу, трьох основних розділів, висновку і переліку використаних джерел. Проведено аналітичний розгляд бойових механік, способів реалізації візуального стилю та побудови логіки ігрової взаємодії. Виконано проектування та розробку ближнього бою, ефектів атак, а також ігрового світу з використанням тайлової карти. Реалізовано повноцінну квестову систему з видачею завдань, перевіркою умов і прокачуванням гравця.*

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) *Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено*

г) перелік позитивних якостей дипломного проекту (роботи)

Грамотно обґрунтовано вибір рушія та жанру гри;

Реалізовано оригінальні бойові механіки з ефектами;

Розроблено власну систему квестів та послідовного прогресу;

Забезпечено цілісність ігрового прототипу з інтерфейсом і логікою.

д) основні недоліки дипломного проекту (роботи)

Ігровий процес обмежено однією локацією. Відсутній редактор рівнів та можливість змінювати карту. Відсутність детальної інформації про балансування ігрового процесу. Деякі рішення (наприклад, використання окремих методів для кожного типу атаки) треба було оптимізувати для зменшення дублювання коду

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові рецензента *к.т.н. Рудніченко Микола Дмитрович*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

« *20* » *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проєкт здобувача (здобувачки) освіти

відділення комп'ютерних систем

Песара Івана Петровича

(прізвище, ім`я та по батькові)

Спеціальність: *121 "Інженерія програмного забезпечення"*

Освітньо-професійна програма: *«Розробка програмного забезпечення»*

Тема дипломного проєкту: *"Розробка гри Vondering Knight з різноманітними механіками ближнього бою"*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЄКТУ

а) обсяг і якість виконання проєкту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проєкт виконано відповідно технічному завданню. Пояснювальна записка містить 75 сторінки. У пояснювальній записці виконано опис етапів розробки гри, проєктування її основниї механік, проєктування механік ближнього бою та розробка цих механік. Графічна частина складається з 15 слайдів мультимедійної презентації, які також містять слайди, передбачені технічним завданням. Якість виконання пояснювальної записки відмінна, якість графічної частини добра, розробку виконано в повному обсязі.*

б) самостійність роботи над проєктом: *Протягом всього строку дипломного проєктування та переддипломної практики здобувач освіти Песар І.П. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Песар Іван Петрович під час роботи над дипломним проєктом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проєкту.*

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проєктування здобувач освіти Песар Іван Петрович мав змогу самостійно приймати рішення з реалізації елементів і модулів гри, та показав вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові керівника дипломного проєкту

Джабраілов Дмитро Володимирович

Місце роботи і посада керівника дипломного проєкту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії

Підпис

«16» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЄКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Песар І.П.
здобувач освіти гр. 4РП-08, та

Джабраілов Д.В.,
керівник дипломного проєкту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проєкту фахового молодшого бакалавра на тему:

«Розробка гри Vondering Knight з різноманітними механіками ближнього бою» (автор роботи – Песар І.П., керівник роботи – Джабраілов Д.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Песар І.П. /

Керівник _____ / Джабраілов Д.В. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4РП-08

Песаря Івана Петровича

на тему *Розробка гри Vondering knight з різноманітними механіками ближнього бою*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з деякими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 14.06.2025 р. значення коефіцієнту подібності в роботі становить 8,71 %, коефіцієнт цитування – 1,92%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Песаря І.П.* (П.І.Б.)

проведена « *16* » *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/14/2025
Дата редагування 6/14/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка гри Vondering knight з різноманітними механіками ближнього бою

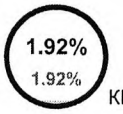
Автор
Песар Іван Петрович
Науковий керівник / Експерт
Джабраілов Дмитро Володимирович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1



КЦ

25
Довжина фрази для коефіцієнта подібності 2

13576
Кількість слів

95749
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	6
Інтервали	A→	361
Мікропробіли	·	11
Білі знаки	ß	1
Парафрази (SmartMarks)	a	50

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	83 0.61 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	68 0.50 %
3	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	68 0.50 %
4	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	61 0.45 %
5	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	50 0.37 %

6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	40 0.29 %
7	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	37 0.27 %
8	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	37 0.27 %
9	https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download	35 0.26 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	29 0.21 %

з домашньої бази даних (0.63 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка web-застосунку для генерації повідомлень із використанням технологій штучного інтелекту [redacted] 6/14/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	31 (5) 0.23 %
2	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	28 (4) 0.21 %
3	Створення web-застосунку цифрового помічника з використанням Open AI [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	27 (3) 0.20 %

з програми обміну базами даних (0.08 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	1 2 3 розділ.docx [redacted] 6/25/2023 Ternopil Professional College of Ternopil Ivan Puluj National Technical University (ЦК комп'ютерної інженерії)	11 (1) 0.08 %

з Інтернету (8.00 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download**	**470 (18) 3.46 %**
2	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**268 (12) 1.97 %**
3	**https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download**	**74 (3) 0.55 %**
4	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	71 (6) 0.52 %
5	https://forum.godotengine.org/t/animation-only-plays-the-first-frame-when-button-pressed/73102	48 (6) 0.35 %
6	https://card-file.ontu.edu.ua/bitstreams/dfa57ac3-98fa-4c22-86e7-0549d1254d89/download	47 (3) 0.35 %
7	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	21 (1) 0.15 %
8	https://cloud.tencent.com/developer/ask/sof/107516032	19 (2) 0.14 %

9	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	18 (3) 0.13 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/25984f4d-6b53-4b0e-8100-83971ae848a1/content	15 (2) 0.11 %
11	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	10 (1) 0.07 %
12	https://card-file.ontu.edu.ua/server/api/core/bitstreams/c63b91ba-d04f-4715-890d-b16277695c7e/content	10 (1) 0.07 %
13	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	5 (1) 0.04 %
14	https://card-file.ontu.edu.ua/server/api/core/bitstreams/21ac499a-a9e9-4137-810c-5f21a0318048/content	5 (1) 0.04 %
15	https://card-file.ontu.edu.ua/bitstreams/c1f3e592-1123-419d-b14a-4c28662f0f1e/download	5 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення» Освітньо-професійна програма: **«Розробка програмного забезпечення»**
Група: 4РП- 08

Дипломний проєкт
здобувача освіти денної форми навчання РП.08.16.000.ДП

ПЕСАРА
ІВАНА ПЕТРОВИЧА

м. Одеса 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: «Розробка програмного забезпечення»
Група 4РП- **08**

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проєкту на тему:
(
_____
_____
)Розробка гри Vondering knight з різноманітними механіками ближнього бою

Проєктний матеріал складається з пояснювальної записки на 80 сторінках та графічного (презентаційного) матеріалу на 15 аркушах (слайдах)

Дипломник (Песар І.П.) Керівник (Джабраілов Д.В.)

Консультанти: з економічного розділу (Канський М.Ю.) з розділу **охорони праці та техніки безпеки (Чорновол Н.І.)** з нормоконтролю (Петрашова **В.І.) старший** консультант (Кривченко Ю.В.)

До **захисту допущений Голова циклової комісії** (Кривченко Ю.В.)
Завідувач відділення (Краснокутська К.Г.)

Захист « » 2025 р. **Протокол** ЕК No **Оцінка** ЕК

Секретар ЕК

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ