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.08.23.000.ДП

ШВЕЦЯ МАКСИМА СЕРГІЙОВИЧА

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 Shady Adventure з різними видами ворогів

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

Дипломник _____ (Швець М.С.)

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 4/80

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" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Швецю Максиму Сергійовичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка гри Shady Adventure з різними видами ворогів*

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

2. Термін здачі закінченого проекту

3. Вихідні данні до проекту *Використання програмного рушія Unity; Використання мови програмування C# та бібліотек Unity для розробки гри; Розробка основних механік ігрового жанру платформер; Реалізація різних видів ворогів; Реалізація проходження різних рівнів гри; Реалізація використання магічних заклинань; Реалізація бойової системи; Реалізація інтерфейсу для гри.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити) *Аналіз аналогічних ігрових проектів; Вибір ігрового програмного рушія для розробки проекту; Вибір інструментів розробки; Визначення основних ігрових механік; Проєктування основних ігрових механік; Проектування різних видів ворогів; Реалізація основних ігрових механік розробляємої гри; Створення ворогів та їх поведінки; Тестування розробленої гри.*

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів) *Огляд аналогічних проєктів в ігровій індустрії; Обрання засобів розробки; Основні ігрові механіки; Проектування та реалізація основних механік гри; Проектування різних видів ворогів; Реалізація ворогів; Реалізація ігрових рівнів; Тестування гри; Скріншоти розробленої гри.*

6. Консультанти по проекту, із зазначенням розділів проекту, 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 _____

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

Завдання прийняв до виконання *Швець М.С.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проєкту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проєктування*	*14.05.2025*	
2	*Проведення розробки ігрового концепту*	*16.05.2025*	
3	*Аналіз та обрання ігрового жанру*	*17.05.2025*	
4	*Обрання програмного забезпечення для розробки*	*20.05.2025*	
5	*Проектування основних механік гри*	*22.05.2025*	
6	*Проектування різних видів ворогів*	*28.05.2025*	
7	*Реалізація ігрових механік та видів ворогів*	*01.06.2025*	
8	*Реалізація графічної частини гри та її інтерфейсу*	*03.06.2025*	
9	*Відлагодження модулів гри*	*06.06.2025*	
10	*Тестування працездатності елементів гри*	*10.06.2025*	
11	*Виправлення виявлених помилок*	*11.06.2025*	
12	*Аналіз результатів, підготовка слайдів презентації*	*12.06.2025*	
13	*Економічні розрахунки та питання з охорони праці*	*13.06.2025*	
14	*Підготовка графічної частини проекту*	*14.06.2025*	
15	*Підготовка проекту до захисту та тестування гри*	*16.06.2025*	

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 23 000. 00 ДП	Дипломний проєкт		
А4			РП 08. 23 001. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 23 000. 00 ДП		
Розробив		Швець М.С.		16.06	Розробка гри Shady Adventure з різними видами ворогів	Літ.	Аркуш 4 / Аркушів 80
Перевірив		Джабраілов Д.В.		16.06		Н Д П	
Н. контр.		Петрашова В.І.		16.06.25		ВСП "ОТФК ОНТУ" гр.4РП-08	
Затверд.		Кривченко Ю.В.		16.06			

ЗМІСТ

ВСТУП

Індустрія комп'ютерних ігор продовжує активно розвиватись, займаючи важливе місце в глобальному IT-секторі. Стрімке зростання ринку відеоігор зумовлене як високим попитом на інтерактивні продукти, так і постійним удосконаленням технологій створення ігор. Сьогодні розробкою ігор займаються як великі студії, так і інді-розробники, які використовують сучасні інструменти, здатні реалізовувати повноцінні проєкти самостійно. Значною мірою це стало можливим через розвиток програмних рушіїв, які спрощують процес створення ігор. Раніше повноцінний ігровий проєкт потребував великої команди, але зараз одна людина, володіюча базовими знаннями програмування та дизайну, може створити власний повноцінний проєкт. Для цього існують умовно безкоштовні інструменти, які дозволяють реалізувати повний цикл розробки – від задуму до публікації.

Завдяки цьому у студентів і початківців з'явилась реальна можливість не лише ознайомитись із процесом створення гри, а й спробувати свої сили у практичній реалізації власного проєкту. Це дозволяє краще зрозуміти структуру ігрових систем, основи архітектури, принципи побудови геймплею, а також складності, з якими стикається розробник під час реалізації проєкту.

Темою цього дипломного проєкту є «Розробка гри Shady Adventure з різними видами ворогів». Проєкт охоплює ключові аспекти створення ігор: проєктування структури, розробку ігрових механік, використання власних 2D зображень, анімацій та інтерфейсу, а також налаштування логіки взаємодії між гравцем і ворогами. Такий підхід дозволяє не лише попрактикуватись в розробці, а й отримати реальне уявлення про складність цього процесу.

Результатом стане повноцінний ігровий продукт, який продемонструє здобуті навички, а також може стати базою для подальшого розвитку. Реалізація гри з різними видами ворогів також дозволяє глибше зануритись в тематику поведінкових шаблонів, логіки та баланс ігрового процесу, що є актуальними питаннями сучасної гейм-розробки.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз аналогічних ігрових проєктів

1.1.1 Загальний аналіз жанру

Перед початком розробки проєкту потрібно визначитись з основними етапами розробки ігор. Їх може бути багато та в різних проєктах можуть відрізнятися одне від одного, але в загальному сенсі вони складаються із деяких складових.

На початку обирається ігровий жанр та з'ясовується чи є на нього запит. Можна створити дуже довгий список, якщо перелічувати усі ігрові жанри, але все ж існують основні ігрові жанри та похідні від них, які можуть комбінувати різні ігрові жанри чи мати свої особливості реалізації механік. У ігрових жанрах відрізняються ігровими механіками, темпом гри, реіграбельністю, візуальним інтерфейсом та елементами ігрового процесу.

У сучасній ігровій індустрії багато ігор реалізують велику кількість видів ворогів для урізноманітнення ігрового процесу та підвищення інтересу гравця. Така особливість зустрічається в багатьох жанрах. Вибір жанру напряму впливає на механіки гри, стиль боїв, поведінку ворогів та можливості гравця.

Одним із найпоширеніших жанрів, у якому реалізується велика кількість видів ворогів, є платформер. Платформери мають чітку структуру рівнів, простий геймплей і часто використовують різноманітних ворогів, що змінюються залежно від локації або складності. Було розглянуто велику кількість аналогічних проєктів різних жанрів, серед яких в даній роботі можна виділити два проєкти, які лягли в основу дипломного проєкту.

У сучасних проєктах цей жанр часто об'єднують з процедурною генерацією, високою складністю та багатьма варіантами проходження, завдяки вибору, який дається гравцю, зазвичай зброї.

Останнім етапом є визначення ігрових елементів та механік та способи їх реалізацій, це буде відігравати важливу роль у зацікавлені гравців в грі. Використання стандартних механік жанру може привернути увагу любителів

цього жанру, та робити унікальні механіки, яких ще не було в цьому жанрі це ризик, тому що гравцям може не сподобатись вона, але також це навпаки може привернути увагу завдяки своїй унікальності. Визначення того як буде реалізований проєкт теж є важливою частиною роботи, тому що правильно спроектована архітектура справді полегшує розробку та підтримку проєкту та покращує його продуктивність.

1.1.2 Аналіз особливостей жанру гри «Rogue Legacy»

Перша комп'ютерна гра жанру платформер з різними видами ворогів для мене стала «Rogue Legacy», тому я почав свій аналіз з цієї гри. Гра офіційно вийшла у 2013 році та швидко здобула популярність серед гравців через свою різноманітність.

У межах аналізу жанру платформер зосередимо увагу саме на тих аспектах цієї гри, які безпосередньо стосуються ключових механік цього жанру. Особливо важливою є механіка платформ, які дозволяють гравцю стрибати через них знизу вверх та щоб спуститися з них потрібно гравцю натиснути клавішу, яка відповідає за цю функцію. Така механіка дає гравцю варіативності у вертикальному переміщенні та є типовим елементом жанру платформер. В цій грі є багато видів ворогів та до кожного потрібен свій підхід. Для цього гравець може вибирати різну зброю, броню та магію. Основною механікою самої гри є те, що після загибелі персонажа гра генерує наново локацію з випадковим порядком кімнат. Завдяки цьому гравець може запам'ятати будову окремих кімнату та розташування кожного ворога у них, але не буде знати, яка кімната буде наступна під час нового проходження. Саме ці особливості гри та її механіки будуть враховані при розробці та проєктуванні дипломного проєкту.

1.1.3 Аналіз особливостей жанру гри «Dead Cells»

«Dead Cells» – також гра з елементами платформера та різними видами ворогів, яка вийшла у 2018 році на всі платформи та набула великої популярності через особливі механіки та свій рівень складності. Її бойова система і динамічний геймплей стали орієнтиром для багатьох інді-розробників.

Як і з попередньою грою розберемо ті механіки, які є ключовими для цієї гри. По-перше в грі також є платформи, які допомагають гравцю в переміщенні на локації. По-друге є велика кількість ворогів з якими стикається гравець та кожен з яких має свою поведінку, саме тому в грі є велика кількість зброї та заклинань, які гравець вибирає в протягом гри та у будь-який момент може їх змінити. По-третє, гра також наново генерує випадковий рівень після загибелі персонажа, і гравець починає спочатку. Проте ця гра зробила це інакше та рівень не складається з окремих кімнат, які створенні в випадковому порядку, а гра генерує суцільну локацію випадковим чином.

Особливу увагу розробники приділили чіткості керування, що робить геймплей надзвичайно динамічним. Велике значення має й швидкість реакції гравця, тому що багато ворогів атакують раптово та потребують миттєвої відповіді. Крім того, гра має яскравий візуальний стиль, що створює відчуття живого, але небезпечного світу. Збалансоване поєднання бойової системи та платформінгу формує унікальний досвід при проходженні. Саме завдяки цьому поєднанню ця гра викликає у гравців інтерес та залишається актуальною навіть через роки після виходу. Скріншот гри зображено на рис. 1.1.

Рисунок 1.1. Скріншот гри «Dead Cells»

1.1.4 Результати аналізу

Після розгляду ряду аналогічних проєктів було обрано жанр платформер для дипломного проєкту, тому що саме цей жанр найбільш ефективно дозволяє реалізувати ідею гри з великою кількістю ворогів. У межах даного жанру є як сталі шаблони, так і унікальні особливості, властиві кожній окремій грі. До сталих елементів жанру належать: механіка платформ, різні типи ворогів, проти яких потрібен індивідуальний підхід, система здоров'я персонажа з можливістю його відновлення, а також втрата прогресу після загибелі персонажа та початок гри заново. Серед унікальних механік можна виділити можливості вибору зброї та магії, а також генерацію рівнів із випадковим порядком кімнат при кожному новому проходженні. Виходячи з цього можна визначити, що необхідно реалізувати у дипломному проєкті.

1.2 Вибір ігрового програмного рушія для розробки проєкту

Кожна розробка ігрового проєкту основана на виборі програмного забезпечення для вирішення питання технологічної реалізації. Ключовим етапом початку розробки є вибір ігрового рушія, так як саме воно впливає на вибір технологій, архітектурні рішення, способи реалізації ігрових механік та багато інших аспектів. Часто саме рушій визначає, як функціонуватиме майбутній проєкт і чи вдалося взагалі реалізувати задумане. У випадку роботи в команді або в складі якоїсь команди, тоді програмне забезпечення буде продиктоване місцем праці. Тому, краще за все вивчати, ті програмні рушії, які широко використовуються.

Ігрові рушії можна поділити на безкоштовні, умовно безкоштовні та платні. До останніх найчастіше відносяться ті ігрові рушії, які були створенні крупними ігровими компаніями для внутрішніх проєктів, наприклад RAGE Engine (Rockstar Games) або REDengine (CD Projekt). Вони не призначені для відкритого доступу і не підходять для індивідуальних чи навчальних проєктів. Безкоштовні найчастіше розповсюджуються у вигляді бібліотек із мінімалістичним інструментарієм для розробки, що може визивати досить

серйозні проблеми для розробки, приклади таких рушіїв – Love2D, MonoGame або Raylib. Умовно безкоштовні рушії мають велику кількість інструментарію для розробки та не потребують плати за використання, при виконанні окремих умов, тому даний варіант є одним з найкращих. До таких належать Unreal Engine, CryEngine та Unity. Огляд їх переваг:

Unreal Engine – рушій, орієнтований на створення високоякісних 3D-проєктів із передовою графікою. Пропонує власну візуальну мову програмування Blueprints, що дозволяє працювати без глибоких знань коду. Незважаючи на підтримку 2D, вона реалізована слабкіше, ніж в тому ж Unity. Підходить радше для складних проєктів, де потрібна реалістична фізика, світло та детальність.

Cry Engine – це потужний ігровий рушій від компанії Crytek. Який вирізняється високою якістю графіки, розвиненою системою освітлення та підтримкою складних симуляцій. Незважаючи на свою технічну перевагу, цей рушій має низьку недоліків для починаючих розробників: складний інтерфейс, обмежена документація та менш активна спільнота в порівнянні з іншими рушіями.

Unity – один із найпопулярніших ігрових рушіїв, особливо серед інді-розробників. Простота використання та доступність роблять його напевно найкращим вибором як для новачків, так і для досвідчених розробників. Unity надає зручне графічне середовище, велику кількість готових ресурсів і компонентів, а також дає змогу створити логіку та взаємодію в грі за допомогою потужної мови програмування C#. Однією з ключових переваг Unity є його модульність і гнучкість, що дозволяє адаптувати рушій під різні типи ігор – від простих 2D-платформерів до складних 3D-проєктів. Це значно розширює можливості розробника в процесі створення ігор. У Unity є одна з ключових особливостей це його кросплатформенність. Авжеж основними платформами залишаються ПК та мобільні пристрої, і саме на цих платформах Unity займає провідні позиції серед рушіїв завдяки особливостям, які були названі раніше. Умовою для умовно безкоштовного використання є те, що зароблена

розробником сума не досягає двохсот тисяч доларів за останні дванадцять місяців.

Окрім того, у Unity є велика підтримка з боку спільноти. Величезна кількість форумів, відеоуроків, офіційної документації та прикладів, що дозволяє швидко вирішувати технічні труднощі та вивчати нові можливості рушія. Цей активний обмін знаннями часто допомагає знайти оптимальні рішення навіть у складних ситуаціях . Це особливо корисно для розробників-початківців. Активна спільнота також значно зменшує бар'єр входу в розробку, навіть складні речі можна освоїти крок за кроком, спираючись на чужий досвід. На рис. 1.2 зображено сайт Unity documentation, де можна знайти офіційний посібник, приклади проєктів та коду.

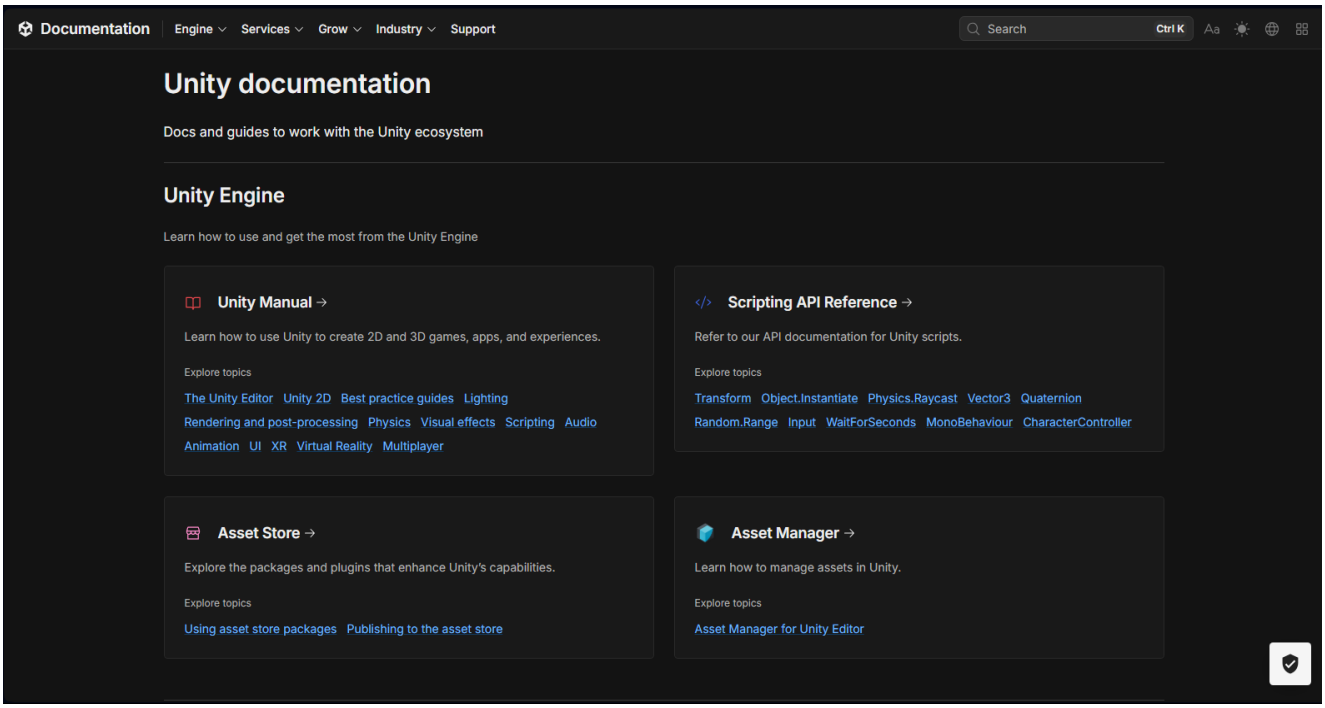


Рисунок 1.2. Сайт Unity documentation

Після порівняння найпопулярніших рушіїв для створення ігор було обрано Unity, оскільки він надає найкраще співвідношення зручності, функціональності та гнучкості саме для проєктів у жанрі 2D платформер. Також важливим чинником стала велика кількість навчальних матеріалів та активна спільнота, що дозволяє швидко знаходити рішення у процесі розробки. Unity також має потужний редактор сцен і систем компонентів дозволяє ефективно реалізувати ідеї навіть за обмежених ресурсів.

1.3 Вибір інструментів розробки

Наступним важливим етапом підготовки є вибір інструментів. Програмне забезпечення відіграє ключову роль у процесі створення гри, адже кожен інструмент має свої особливості, переваги та обмеження. Також розробники досить часто мають власні вподобання щодо редакторів, у яких вони вже мають досвід і добре орієнтуються у функціоналі. В більшості випадків навіть великі студії не встановлюють жорстких вимог до вибору допоміжного інструментарію, наприклад, для редактору коду розробникам дозволяється працювати з тим, що їм зручно. Вибір правильного інструментарію на початковому етапі дозволить уникнути багатьох проблем у майбутньому, від труднощів з інтеграцією компонентів до втрати часу на пошук сумісних рішень. Під час розробки дипломного проєкту мені потрібно було реалізувати як програмну, так і графічну складову.

Для реалізації програмної частини я обрав Microsoft Visual Studio, оскільки вона надає повну інтеграцію з Unity та оптимізована для роботи з мовою програмування C#. Це середовище дозволяє зручно організувати роботу з класами, методами, об’єктами сцени та зовнішніми бібліотеками. Також Visual Studio є офіційно рекомендованим редактором для Unity і має широку підтримку серед розробників. Крім того, середовище пропонує розширені можливості відлагодження, підказками в коді та перевірки на помилки в режимі реального часу, що особливо корисно якщо вперше створюєш проєкт та при частій зміні логіки гри.

Для графічного наповнення проєкту я обрав Adobe Photoshop. Так як вже є досвід роботи з цим редактором і його функціоналу достатньо для створення простих спрайтів та їх анімацій. Photoshop підтримує роботу з шарами, масками, а також має великий вибір інструментів трансформації, що дозволяє швидко адаптувати спрайти до потреб гри. Оскільки графіка в грі на даному етапі не потребує високого рівня деталізації, цей редактор повністю задовольняє вимоги проєкту. Крім того, зручність інтерфейсу та можливість швидко експортувати готові зображення у потрібних форматах робить його практичним вибором.

1.4 Визначення основних ігрових механік

Темою дипломного проєкту є розробка гри з різними видами ворогів. З ігор розглянути вище було з’ясовано, які основні механіки потрібно реалізувати. Також був визначений ігровий програмний рушій та виходячи з цього можна розпочати розробку концепту цієї гри.

Перше – метод створення ігрового простору. Одним із важливих аспектів розробки гри є створення ігрового простору, особливо способу реалізації підлоги та рівня загалом. Для цього можна використовувати окремі об’єкти або тайлсет (англ. tileset). Використання окремих об’єктів є легким варіантом, однак може призвести до навантаження на продуктивність. У свою чергу тайлсет дозволяє зручно й оптимізовано створювати великі ділянки карти, зменшуючи кількість об’єктів на сцені, але в свою чергу потребують попереднього налаштування. Тому для проєкту був вибраний спосіб за допомогою тайлсету.

Друге – система отримання пошкодження. В грі буде використовуватись стандартна система для сучасних ігор, у персонажа є певний рівень здоров’я та якщо він отримає пошкодження, тоді рівень здоров’я понижається, і якщо здоров’я опуститься до 0, тоді гравець програє. Також потрібно звернути увагу на систему перезапуску після програшу і як в іграх, які були розглянуті була вибрана система повного перезапуску. Це означає, що після поразки гравець починає гру з самого початку, втрачаючи усі набуті предмети та прогрес. Такий підхід підвищує складність та досить часто використовується в іграх.

Третє – види ворогів, кожен з ворогів повинен вимагати від гравця певного підходу. Наприклад, одні вороги будуть поводитись агресивно та намагатися наблизитись до гравця та наносити шкоду при контакті, змушуючи його ухилятися та наносити атаки з дистанції. Інші – навпаки, тримають гравця на дистанції й завдають шкоду здалеку за допомогою снарядів, тим самим змушуючи гравця ухилятись від їх снарядів та самому наближатись до них, щоб нанести атаку. Також, для таких ігор є стандартним додавати перешкоду, яка не є ворогом, але яка наносить шкоду, якщо доторкнутись до неї.

Четверте – система зброї, яка надає гравцеві можливість обирати стиль гри відповідно до власних уподобань. У грі має бути передбачено кілька видів зброї – від швидкої, яка наносить мало шкоди до повільної, але потужної. Також потрібно розробити магічні заклинання для атак на відстані. Це дозволить адаптувати тактику до різних ситуацій. Одних ворогів зручніше атакувати зблизька, а інших тримати на відстані. Також треба звернути увагу на систему отримання предметів. Для дипломного проєкту було вирішено що предмети будуть видаватись після знищення всіх ворогів на сцені. У грі буде певна кількість сцен та гра буде вважатиметься пройденою коли гравець збере всі доступні предмети.

П’яте – система інвентаря, яка дозволяє гравцю зберігати отримані предмети та знаряджати потрібну йому зброю чи заклинання. Гравець може, натиснувши на певну клавішу на клавіатурі, відкрити інвентар та переглянути чи спорядити предмети в ньому. Також для зручного використання заклинань потрібно зробити панель швидкого доступу, яка дозволяє гравцю миттєво перемикатись та використовувати обрані заклинання. Це забезпечить зручність під час бою, оскільки не буде потреби постійно відкривати інвентар.

1.5 Проєктування основних ігрових механік

1.5.1 Проєктування предметів

Було вирішено створити 3 види предметів: заклинання, зброя та пасивна здібність. Вони повинні унаслідуватись від класу предмет, який зберігає інформацію, яка відноситься до кожного з типів предметів, таких як: зображення, назва, тип предмета. А вже кожен з типів предмета має свій клас, де зберігається інформація, яка відноситься для його класу. Для заклинань це шкода та перезарядка, пасивна здібність зберігає інформацію, яку здібність вона має, а зброя зберігає інформацію про шкоду, перезарядку та дальність атаки. Такий підхід дозволить зберігати централізовано спільні дані, що спрощує додавання нових типів предметів у майбутньому. Структурну схему зберігання даних предметів зображено на рисунку 1.3.

Рисунок 1.3. Структурна схема зберігання даних предметів

У ігор, які були розглянуті раніше був досить важливий вибір для гравця, зброї та заклинань, але в цих іграх магія була скоріше другорядним. Тому було вирішено зробити у дипломному проєкті магію як основну зброю, але щоб гравець не використовував нескінченно одне й теж заклинання, буде зроблено декілька різних заклинань між якими гравець зможе перемикатись. Авжеж потрібно зробити якесь обмеження на використання, і між двома варіантами: щоб заклинання коштували ману чи у заклинань буде перезарядка в декілька секунд, було вирішено додати усім заклинанням перезарядку, тому чим потужніше заклинання, тим довшим буде його перезарядка. Через це гравець буде в першу чергу використовувати заклинання і тільки після цього використовувати зброю ближнього бою.

Перше заклинання – метеор. Він запускається в напрямку куди дивиться персонаж, з початковим імпульсом, що спрямовує метеор вверх і вперед від персонажа. Після чого під дією гравітації починає падати вниз, все так же віддаляючись вперед від персонажа. Це створює характерну дугоподібну траєкторію польоту. Метеор має середню шкоду и 6 секунд перезарядки. Це зумовлено тим, що гравцеві потрібно запам’ятати траєкторію польоту заклинання та попасти ним, перебуваючи на відповідній відстані від ворога.

Друге заклинання – вогняний снаряд. Гравець може вибирати траєкторію польоту снаряду за допомогою миші, але снаряд також піддається дії гравітації. Вогняний снаряд має середню шкоду та 8 секунд перезарядки. Правильно

кинутий снаряд може далеко пролетіти на завдати шкоду декільком ворогам на великій відстані, тому це заклинання має трохи більшу перезарядку.

Третє заклинання – промінь світла. Гравець випускає промінь, який летить рівно та гравець може вибирати напрямок за допомогою миші. Промінь має високу шкоду, 12 секунд перезарядки, але влучає тільки в першого ворога в якого попаде. Тому це заклинання краще використовувати на одного сильного ворога.

Четверте заклинання – тимчасова платформа. Гравець створює тимчасову платформу у себе під ногами, вона працює як і звичайна платформа, це значить що через неї можна пройти знизу вверх. Також платформа буде знищувати ворожі снаряди, щоб можна були використовувати це заклинання як захист. Платформа існує 7.5 секунд та має перезарядку 15 секунд.

Останнє заклинання – лікування. Дуже потрібне заклинання, враховуючи те, що програш в грі означає що потрібно починати гру з самого початку. Лікування відновлює персонажу 50% здоров'я, що дає гравцю право на помилку. Це заклинання досить сильне і відновлює велику кількість здоров'я, через що її перезарядка 50 секунд.

Далі проєктування зброї, було вирішено зробити просту систему зброї, яка буде відрізнятись збільшенням шкоди, дальності атаки та чим більші ці показники, тим більша буде перезарядка атаки. Основна ідея полягає в тому, що чим сильнішою є зброя тим повільніше вона перезаряджається. Така система дозволяє уникнути зловживання однією сильною зброєю та змушує гравця планувати тактику своєї гри.

У зброї повинно бути різна назва, яка відображатиме силу зброї. Слабка зброя називатиметься «Тіньовий уламок», наноситиме 10 шкоди, коротку дальність атаки, але 0.5 секунд затримки між атаками. Трохи сильніша зброя називається «Тіньове ікло», завдає 15 шкоди, має трохи більшу дальність атаки та перезарядку 0.7 секунд. Середньою зброєю є «Тіньовий клинок», який завдає 25 шкоди, має середню дальність атаки та перезарядку 1.5 секунд. Найсильніша зброя називається «Тіньовий руйнівник», що буде підкреслювати його силу, наноситиме 50 шкоди, велику дальність атаки, але 5 секунд перезарядки атаки не

дасть гравцю можливість битися лише ним. Також, потрібні якась зброя, яка буде між ними. Цей підхід дозволяє створити варіативний геймплей, де гравець обирає зброю залежно від свого стилю гри чи ситуації на рівні.

Також для варіативності в проходженні гри, було вирішено створити пасивні здібності. Гравець зможе спорядити та мати ефект тільки однієї такої здібності.

Першою такою здібністю є прискорення, яке збільшить швидкість персонажа на 35%. Це надає гравцеві перевагу в мобільності – він зможе наближатись до ворогів або, навпаки віддалятись від них залежно від ситуації.

Друга здібність – подвійний стрибок. Така здібність дасть гравцю можливість краще маневрувати у повітрі, що особливо корисно для уникнення шипів, ворожих атак або снарядів.

Третьою є збільшення максимального здоров'я. Ця здібність підвищує витривалість персонажа, що дозволяє допустити більше помилок без фатальних наслідків. Крім того, вона посилює ефект заклинання лікування, оскільки воно відновлює 50% від максимального запасу здоров'я.

Четвертою є збільшення шкоди від атак ближнього бою на 50%. Це дозволяє гравцеві обрати зброю ближнього бою основним інструментом атаки, а заклинання допоміжним засобом. Через що, гравець зможе експериментувати з різними типами зброї. Спорядити сильну зброю, щоб мати ще більше шкоди та додаткова дальність атаки від цієї зброї дасть можливість наносити шкоду з безпечної відстані. Або обрати середню зброю, яка має меншу перезарядку та зі спорядженою здібністю шкода зброї буде вище середнього, що дасть можливість наносити сильні та швидкі атаки.

Останньою здібністю є посилення заклинань на 35%. Усі заклинання мають своє посилення від цієї здібності:

-метеор, вогняний снаряд та промінь світла отримують додаткову шкоду.

-тимчасова платформа отримує додатковий час існування, що також зменшить час очікування між її зникненням та повторним використанням.

-заклинання лікування буде додатково відновлювати більше здоров'я.

1.5.2 Проєктування ігрового інтерфейсу

Також окрема потрібно розглянути питання ігрового інтерфейсу, який потрібно розробити на етапі проєктування. Було вирішено, що буде інтерфейс, який гравець бачитиме весь час, а також інтерфейс, який буде відображатись при певних обставинах.

До інтерфейсу, який гравець буде бачити весь час відноситься: шкала здоров'я та панель швидкого доступу. При проєктуванні ігрового інтерфейсу потрібно приділяти особливу увагу як функціональності, так і зручності сприйняття. Концепт ігрового інтерфейсу шкали здоров'я та панелі швидкого доступу зображено на рис. 1.4.

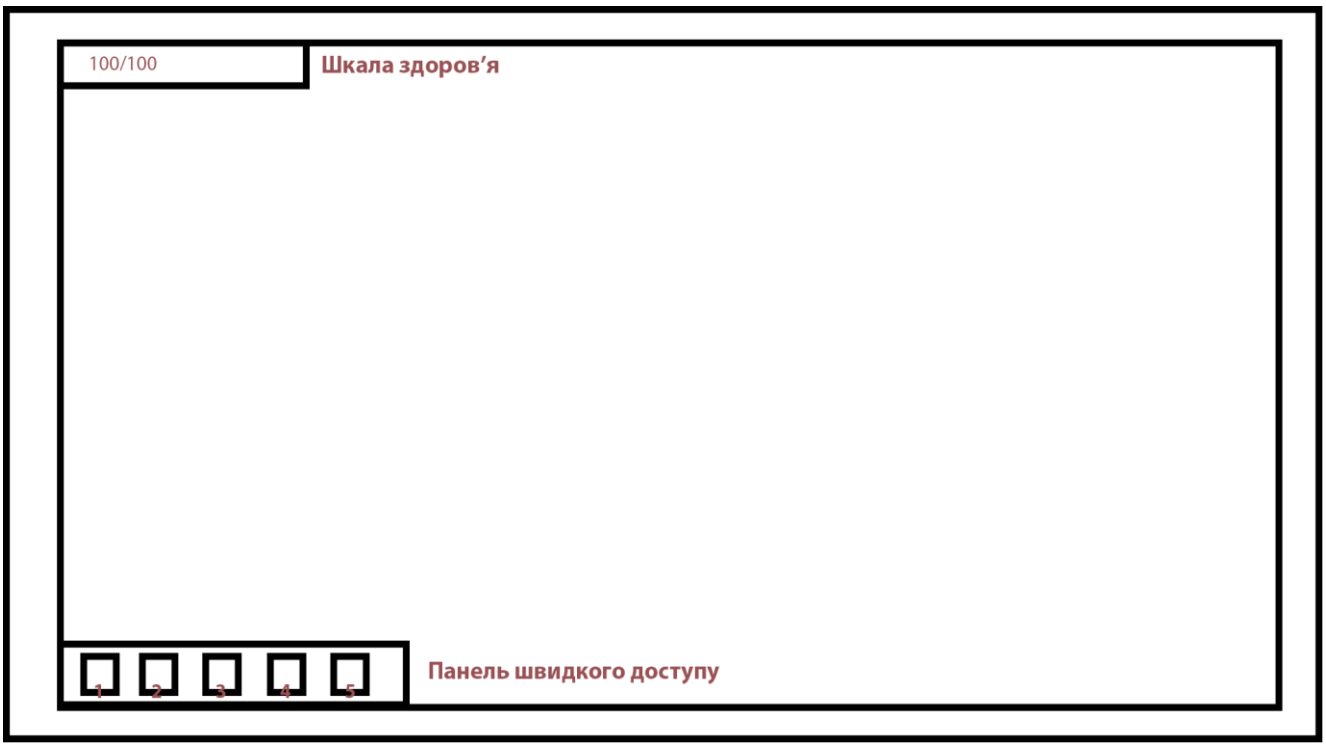


Рисунок 1.4. Концепт ігрового інтерфейсу

Саме завдяки інтерфейсу гравець може отримувати зворотній зв'язок від гри, бачити стан персонажа, доступні предмети, тощо. Шкала здоров'я отримує від гравця інформацію про поточне здоров'я та максимальний запас здоров'я та відображає його у реальному часі. А панель швидкого доступу, у свою чергу, відображатиме спорядженні заклинання, їх розташування в слотах та стан перезарядки. Це дозволить гравцю швидко перемикатись між заклинаннями та

бачити скільки залишилось до повної перезарядки. Такий підхід не тільки підвищить зручність управління, а й підтримує динамічність геймплею.

До інтерфейсу, який буде відображатись при певних діях будуть:

-повний інвентар, який відображається, якщо гравець натисне клавішу Tab.

-панель вибору предметів, яка відображається, якщо гравець знищить всіх ворогів на сцені і дасть для нього вибір з трьох випадкових предметів.

-екран програшу, який відображається, при смерті персонажа.

Було вирішено зробити повний інвентар, який відкривається посередині екрану та блокує можливість пересуватись, атакувати та використовувати заклинання персонажу, поки інвентар відкритий. У інвентаря повинно бути достатньо місця, щоб зберігати всі предмети та мати окремі слоти для зброї, заклинань та пасивної здібності. Концепт інтерфейсу інвентаря зображено на рис. 1.5.

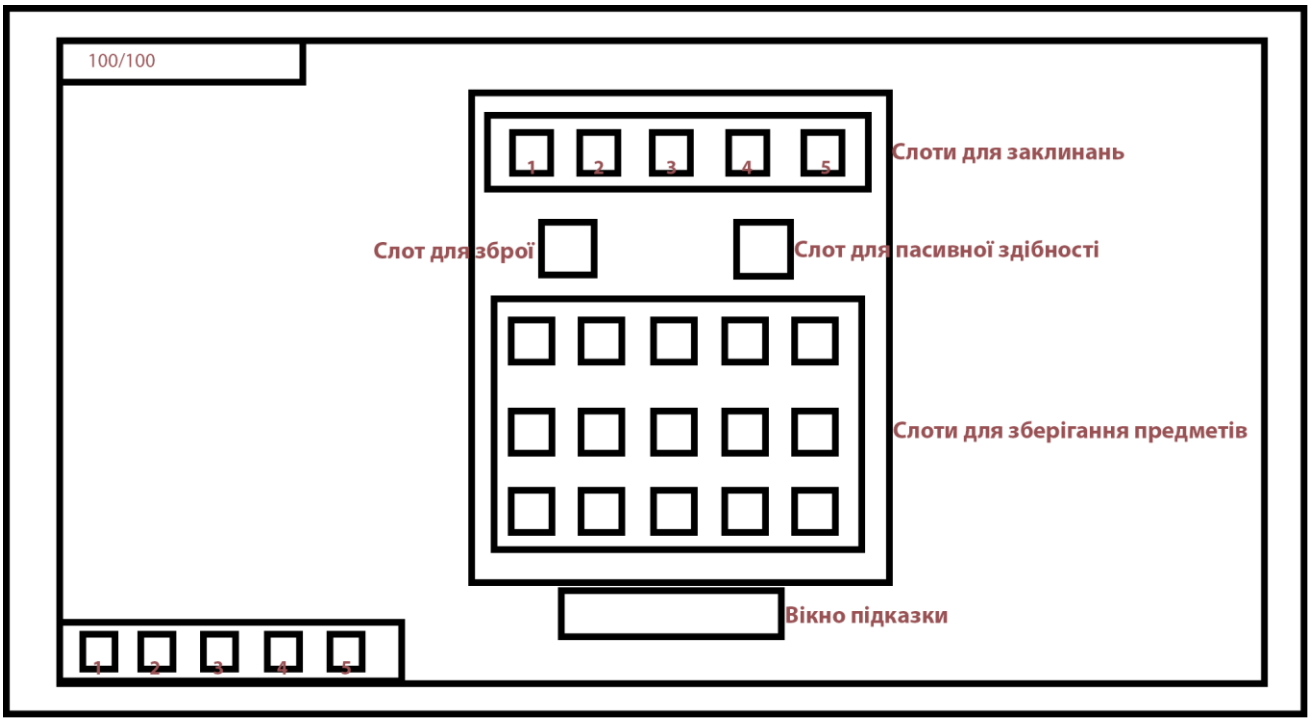


Рисунок 1.5. Концепт інтерфейсу інвентаря

Для повноцінної роботи інвентаря потрібні наступні елементи: слоти для зберігання усіх предметів, які гравець отримує під час проходження гри, а також спеціальні слоти для спорядження зброї, пасивної здібності та заклинань. Також правильним рішенням буде додати вікно підказки, яке буде з’являтись при

наведенні курсора на предмет і вікно буде відображати детальну інформацію цього предмету, такі як: назва, тип, шкода, перезарядка, дальність атаки, опис.

Слоти для всіх предметів, коли гравець буде отримувати предмет, він буде видаватись саме в ці слоти. Для коректної роботи інвентаря потрібно, щоб кожен слот приймав певний тип предмету. Ці слоти можуть приймати усі типи предметів, в свою чергу слоти для зброї, пасивної здібності та заклинань будуть перевіряти та приймати предмети тільки свого типу. Це значить, що слоти повинні мати 4 варіанти: які приймають усі типи, тільки зброя, тільки пасивні здібності та тільки заклинання.

Unity працює з англійською мовою, тому перевірка того, що може в собі зберігати слот буде розподілена на: Any, Spell, Weapon, Passive. Коли гравець споряджає заклинання, вони починають відображатись у панелі швидкого доступу, що дозволяє гравцю використовувати 5 заклинань, перемикаючись між ними. Коли споряджає зброю, у нього міняється дальність ближнього бою, відносно дальності атаки зброї, перезарядка атаки та шкода. Коли гравець споряджає пасивну здібність, він отримує одне з посилень, до поки здібність не буде знята з екіпірування.

Наступним елементом інтерфейсу була розроблена панель вибору предметів, що з’являється після завершення кожної сцени. Було вирішено, що гравцю надаватиметься вибір із трьох випадкових, але ще не отриманих предметів. Через що, гравець зможе обирати саме те посилення, яке найкраще доповнить його стиль гри. Завдяки цьому, інвентар поступово заповнюється відповідно до вибору гравця, а не випадковості, що робить ігровий процес більш цікавим і гравець має вибір, який сприяє на посилення персонажа.

Після знищення всіх ворогів на сцені, з’являється панель вибору предметів, яка блокує персонажу можливість пересуватись, атакувати та використовувати заклинання, до поки гравець не зробить вибір одного з предметів, після чого панель зникає та відновлює рух персонажа. Концепт інтерфейсу панелі вибору предмета зображено на рис. 1.6.

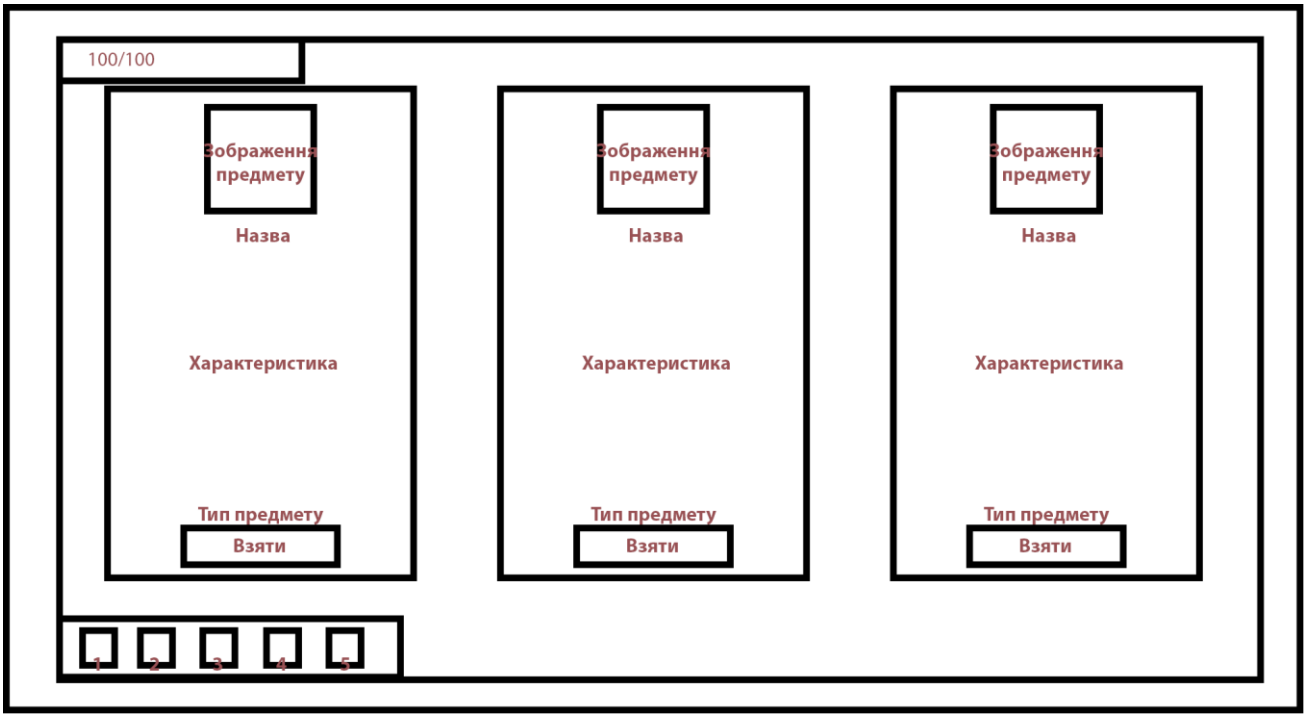


Рисунок 1.6. Концепт інтерфейсу панелі вибору предмета

Останнім елементом інтерфейсу є екран програшу, який з'являється у випадку, коли здоров'я персонажа знизиться до нуля. Основне завдання цього елементу інтерфейсу – чітко повідомити гравцю про програш, а також запропонувати можливість перезапуску чи виходу. Спершу потрібно заблокувати управління персонажем, щоб гравець не міг рухатись або виконувати дії після смерті. Після цього запускається анімація смерті у персонажа, яка додає візуального завершення бою, і лише потім з'являється екран програшу. Також, щоб перекрити видимість сцени та сфокусувати увагу гравця на екрані програшу, весь екран, окрім потрібних елементів, буде набувати червонуватий відтінок. Для правильної роботи екрану програшу йому потрібні наступні елементи: напис «Програш», кнопка перезапуску та кнопка виходу.

1.6 Проєктування різних видів ворогів

Щоб уникнути проблем під час реалізації ігрового процесу, дуже важливо ще на етапі проєктування чітко визначити логіку поведінки ворогів. Завчасне продумування типів ворогів та їх механік дозволяє забезпечити більш структурований підхід до розробки гри.

У всіх скриптів ворогів повинен бути батьківський клас, який буде зберігати всю інформацію, яка відноситься до всіх ворогів. Було вирішено назвати клас «Entity» та унаслідувати від нього класи ворогів та ігрового персонажа. Батьківський клас буде зберігати зміні та методи, які відносяться, як до гравця так і до ворогів. Будуть використовуватись такі зміні як: currentHealth, Rigidbody2D, SpriteRenderer, Animator. А до батьківських методів відноситься Awake, в якому отримують доступ до компонентів Rigidbody2D та спрайта, TakeDamage, завдяки якому і ворог і гравець будуть визивати цей метод при отриманні шкоди, Flip, який буде розвертати дочірній спрайт.

Такий метод спрощує розробку та оновлення проєкту тим, що коли потрібно змінити один й той самий метод у всіх ворогів, потрібно його змінювати лише в батьківському класі. Це також зменшує дублювання коду, через що орієнтуватись в ньому стає простіше і під час відлагодження коду всі зміні коду знаходяться в одному місці. Блок-схему наслідування класів зображено на рис. 1.7.

Рисунок 1.7. Блок-схема наслідування класів

Для початку потрібно додати перешкоду, яка теж буде наносити шкоду персонажу при зіткненні, зазвичай такими перешкодами є шипи, тому у грі будуть шипи, які повинні мати схожий колір з підлогою рівня, але також зберігати різницю в відтінках. Це змусить гравця бути уважним, спостерігати за середовищем.

Тепер потрібно розробити активних ворогів, змушуючи гравця зосереджувати свою увагу на них, через що гравець може випадково натрапити на шипи. Першим таким ворогом буде слайм, який досить часто зустрічається в таких іграх. У нього буде радіус з якого він помічає гравця, після чого слайм підготовлюється до стрибка та стрибає в сторону гравця, наносячи шкоду при контакті. У цього ворога повинно бути середнє значення здоров'я та між його атаками повинен бути певний час, що створить ситуацію, де гравцю потрібно нанести декілька атак слайму та щоб ухилитись від його стрибка потрібно відходити в сторону, через що гравець може наткнутись на шипи чи інших ворогів. Схожу структуру поведінки має кожен з ворогів.

Другим ворогом буде маг, який буде випускати магічний снаряд по горизонталі в сторону гравця. Маг буде їх випускати незалежно від того як далеко гравець від нього, але умова того що він може стріляти лише по горизонталі дає магу змогу лише заважати гравцю пересуватись на тій самій висоті, де і стоїть маг. Маг не зможе пересуватись та не буде наносити шкоду при контакті, що робить мага слабким ворогом, якщо він один, але якщо маг стоїть далеко та між ним та гравцем є ще декілька ворогів, тоді це дасть йому змогу перешкодити гравцю у вільному пересуванні та змусить гравця ухилятись від магічних снарядів.

Третій ворог буде кажан, який має радіус зору та менший радіус атаки. Кажан буде літати та атакувати гравця на відстані, але цей ворог буде мати середню дальність атаки і через те, що радіус зору більша ніж дальність атаки, кажан повинен буде підлітати до гравця перш ніж атакувати, що дасть гравцю час зреагувати. На відміну від мага кажан може випускати снаряд у будь-яку сторону, але має меншу дальність атаки та невелику швидкість польоту. Також

кажан наносить шкоду при контакті з гравцем, що створює для гравця перешкоду в повітрі.

Останнім ворогом буде примара, який повинен переслідувати гравця та мати шкоду при контакті. Його особливістю є здатність літати, великий радіус виявлення гравця, великий запас здоров'я, а також поступове пришвидшення при наближенні до гравця. Щоб уникнути ситуації, коли примара застрягає в гравцеві через свою високу швидкість в близькості з гравцем, після завдання шкоди примара буде зупинятись на декілька секунд, що дасть змогу гравцеві збільшити відстань між ними чи завдати достатньо велику шкоду примарі.

1.7 Реалізація основних ігрових механік розробляємої гри

1.7.1 Створення нового 2D проєкту

Для створення проєкту використовується відповідна програма UnityHub, який є центральним інструментом для керування всіма аспектами роботи з Unity. Основними функціями потрібними для створення дипломного проєкту є управління версіями, створення та керування проєктами. UnityHub дозволяє легко створювати нові проєкти на основі шаблонів, можна обрати шаблон для 2D гри. Можна побачити деякі шаблони для створення проєктів, які завантажують ігрові об'єкти та налаштування сцени.

Після натискання «Create project» проєкт буде створено автоматично, разом з усіма необхідними файлами, директоріями та папками. Також автоматично буде створена пуста сцена, із стандартним набором об'єктів – головна камера та освітлення. Також будуть підключенні пакети для роботи з 2D об'єктами.

1.7.2 Створення та реалізація тайлсету

В розділі «Визначення основних ігрових механік» були розглянуті методи створення ігрового простору та вибраний варіант створення за допомогою тайлсету. Завдяки цьому можна швидко формувати рівні, використовуючи повторювані графічні елементи, що значно полегшує створення великої

кількості сцен, зменшує навантаження на систему та забезпечує візуальну цілісність гри.

Для початку потрібно створити зображення тайлсету – набір невеликих квадратних елементів (тайлів), з яких буде формуватися рівень гри. Було вирішено створювати всі спрайти власноруч, що дозволяє повністю контролювати стиль та візуальну єдність гри. Тому потрібно зрозуміти, як правильно створити власний тайлсет.

Головна вимога до малювання полягає у створенні тайлів, які безшовно переходять один в одного. Для цього межі кожного тайлу повинні візуально збігатись з межами усіх тайлів, які можуть бути для нього сусідніми. Такий підхід дозволяє створити зображення суцільної поверхні, де немає видимих «швів». Тому потрібно одразу продумати повний набір тайлів: горизонтальні, вертикальні, краї та вигини – усе, що може знадобитись для побудови різноманітного ігрового середовища. Це дозволяє гнучко створювати рівні без потреби повертатись до малювання нових тайлів у майбутньому. Мій готовий тайлсет зображено на рис. 1.8.

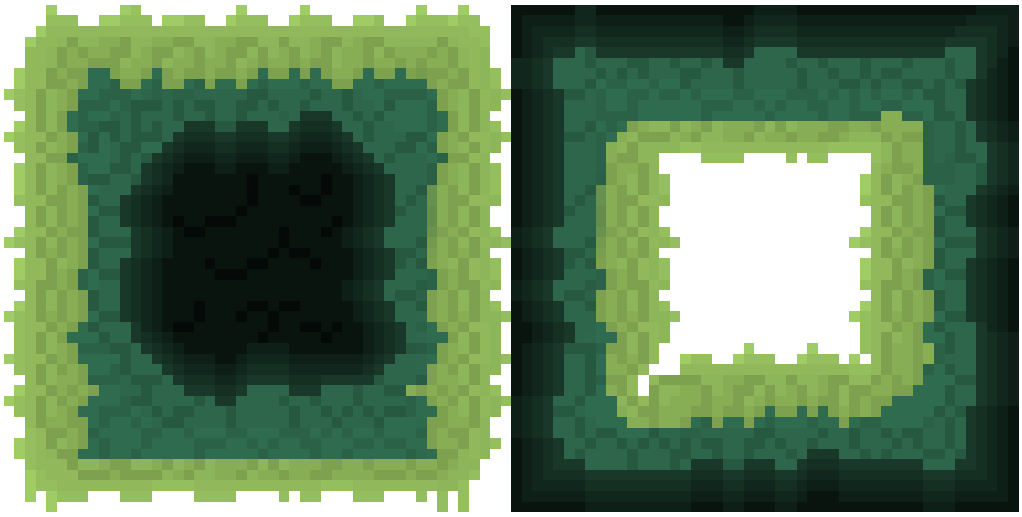


Рисунок 1.8. Готовий тайлсет

Тепер в Unity імпортую це зображення, в папці Assets створюю папку Sprites та пересуваю зображення в неї. Для налаштування, натискаю на зображення та в інспекторі ставлю наступні параметри: Sprite Mode: Multiple,

Pixels Per Unit: 16. Далі натиснув на кнопку «Open Sprite Editor», щоб налаштувати коллайдери. Було вирушено зробити коллайдери у кожного тайлу трохи меншими з відкритої сторони, для того щоб персонаж стояв в траві, а не над нею. Теж саме було зроблено і з кутами для їх правильного функціоналу. Вікно Sprite Editor зображено на рис. 1.9.

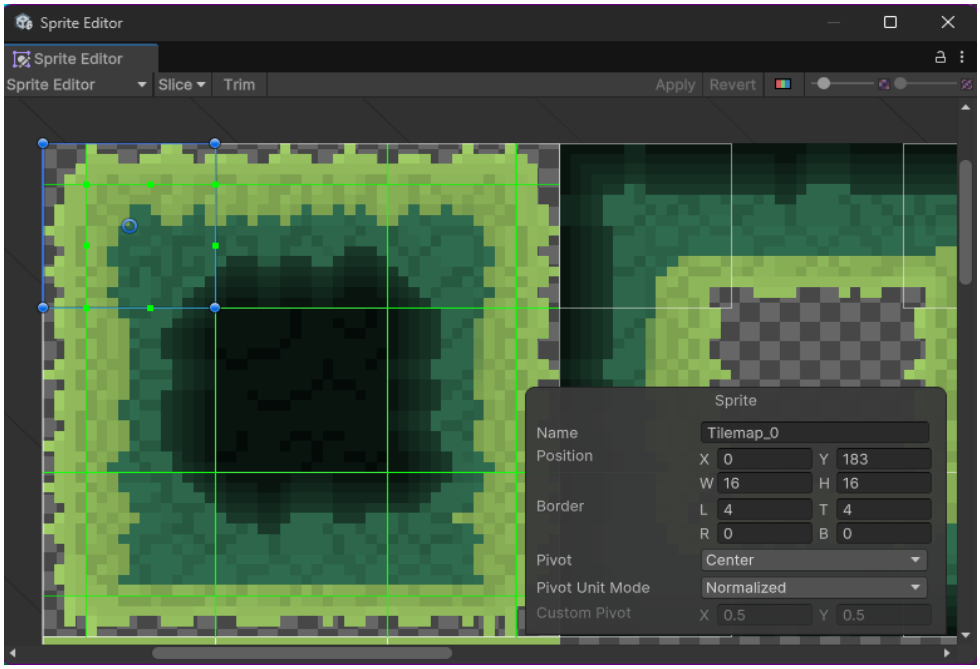


Рисунок 1.9. Вікно Sprite Editor

Щоб розпочати побудову рівня, потрібно створити палітру тайлів. Відкриваю вікно Window>2D>Tile Palette, в цьому вікні натискаю на кнопку «Create New Palette» та обираю формат Rectangle. Далі пересуваю всі спрайти тайлів в палітру. Палітра готова, щоб її використовувати потрібно на сцені створити тайлмапу. Для цього створюю новий об’єкт на сцені GameObject>2D Object>Tilemap>Rectangular, Unity автоматично створить «Grid» і «Tilemap». І щоб на тайлмапі міг стояти персонаж до нього потрібно додати компонент TilemapCollider2D. Тепер можна створювати рівні виділяючи тайли в палітрі та ставити їх на сцені.

Але Unity також дозволяє використовувати «Tilemap Rule Tile», використовуючи визначений тип тайла, який автоматично змінює вигляд залежно від сусідніх тайлів, наприклад виглядати як кут, якщо тайл стоїть на краю. Що дасть можливість ще швидше створювати велику кількість сцен,

використовуючи один тайл, щоб намалювати, те як повинен виглядати рівень. Для цього потрібно встановити пакет «2D Tilemap Extras» через Unity Package Manager. В папці Assets створити Rule Tile, після чого вибрати тайл, який буде змінюватись і останнє – налаштувати правила, вказавши, як має виглядати тайл у залежності від оточення. Всього 13 тайлів в тайлсеті, тому повинно бути 13 правил для цього. Готовий Rule Tile зображено на рис. 1.10.

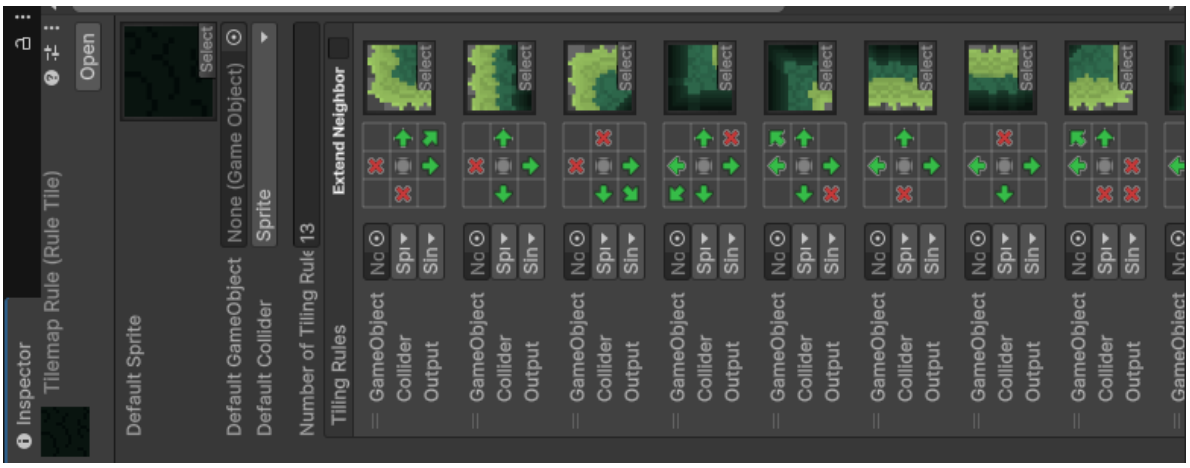


Рисунок 1.10. Готовий Rule Tile

Після чого додаю його до палітри та використовуючи його можна швидко створювати велику кількість сцен.

1.7.3 Створення персонажу та основних механік управління

Після створення ігрового простору за допомогою тайлсету, наступним етапом є реалізація ігрового персонажа. Саме ігровий персонаж буде основною точкою взаємодії гравця з ігровим світом. Для початку створюю спрайт персонажа, без анімацій. Імпортую його в проєкт і в інспекторі ставлю наступні налаштування для цього і всіх наступних спрайтів: Filter Mode: Point (no filter), Compression: None. Це потрібно щоб Unity не розмивало піксельний спрайт персонажа.

Для створення персонажа на сцені, було вирішено зробити персонажа як пустий об’єкт-контейнер з усіма компонентами та зі спрайтом у дочірньому елементі, що дозволить легко змінювати спрайт, не зачіпаючи логіку чи фізику. У об’єкт-контейнер додаю основні компоненти: Rigidbody2D, CapsuleCollider2D, Player_Controller. Rigidbody2D – надає фізику об’єкту, тому об’єкт починає

підкорятись гравітації, інерції, прискоренню та зіткненням. CapsuleCollider2D – це невидима форма, яка визначає, де об’єкт може взаємодіяти з іншими об’єктами. Без цього компоненту об’єкт буде проходити крізь один одного. Форма коллайдера була вибрана у вигляді капсуля, тому що саме він більш за все схожий на форму персонажа. Player_Controller – власний скрипт управління, який дозволяє персонажу рухатись вліво-вправо та стрибати. На рис. 1.11 зображено блок-схему логіки класу переміщення гравця. Нижче зображено фрагмент коду:

```
private Rigidbody2D rb;
private float moveSpeed = 6.5f;
private float jumpForce = 30f;
private bool isGrounded;
void Awake() {
    rb = GetComponent<Rigidbody2D>(); }
void Update() {
    if (Input.GetButton("Horizontal"))
      Run();
    if(Input.GetButtonDown("Jump")&&isGrounded)
      Jump();
    CheckGround(); }
private void Run() {
  float moveInput = Input.GetAxis("Horizontal");
  Vector3 dir = new Vector3(moveInput, 0, 0).normalized;
  transform.position+=dir*moveSpeed*moveSpeedBoost* Time.deltaTime; }
private void Jump() {
  rb.linearVelocity = new Vector2(rb.linearVelocity.x, 0);
  rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse); }
private void CheckGround() {
  Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position,
0.3f, groundLayer);
   isGrounded = collider.Length > 0; }
```

Метод Awake викликається одразу після створення об’єкта, ще до старту гри. Через нього отримую компонент Rigidbody2D, щоб надалі ним користуватись.

Метод Update викликається кожного кадру гри. Саме в ньому будуть оброблятись введення з клавіатури та визначати чи торкається персонаж землі.

Метод Run переміщує персонажа по осі X. Залежно від натиснутої клавіші значення Input.GetAxis("Horizontal") стає -1 або 1.

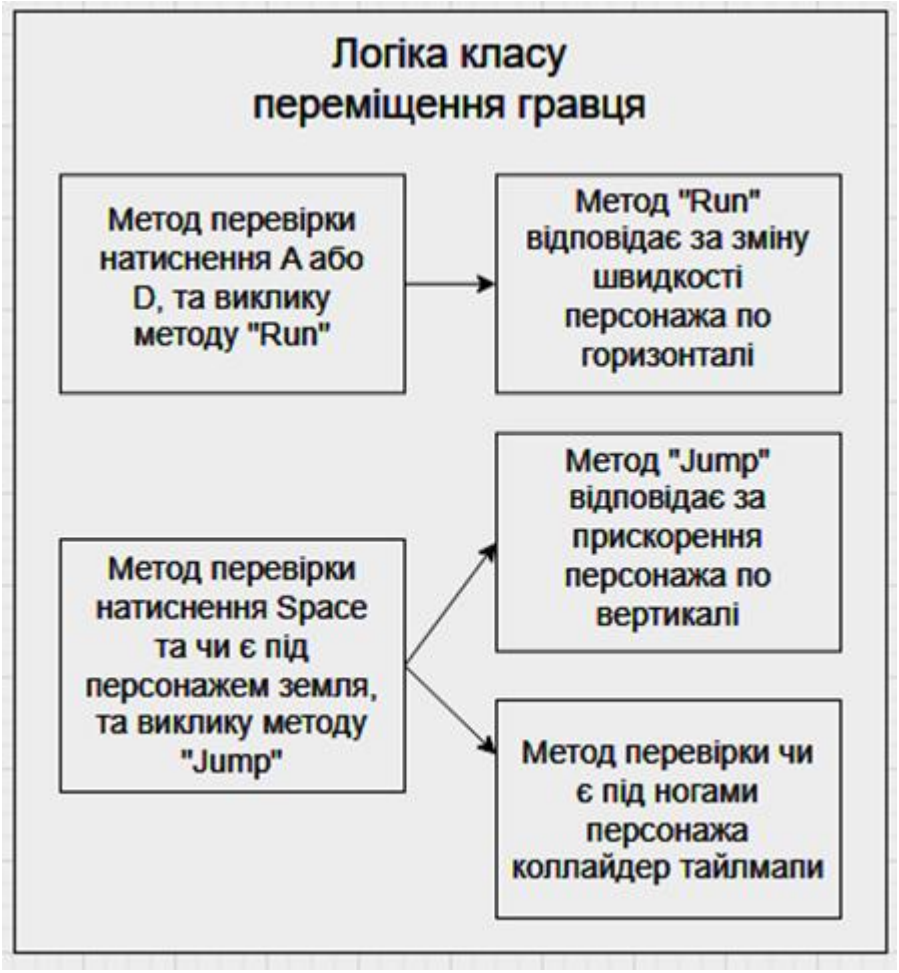


Рисунок 1.11. Блок-схема логіки класу переміщення гравця

Метод Jump додає силу вверх з типом імпульсу, що миттєво впливає на тіло. Перед стрибком вертикальна швидкість обнуляється, щоб уникнути накопичення прискорення під час повторних стрибків.

Метод CheckGround перевіряє, чи є під персонажем коллайдери землі. Якщо коллайдерів більше нуля, тоді isGrounded = true.

Після реалізації базової логіки управління необхідно додати анімацій для персонажа. Кожен стан персонажа містить набір кадрів анімації, які поєднуються в Unity у Sprite Animator. Покадрова анімація є досить часто використовується в іграх з піксельною графікою. Після створення кожної анімації імпортую всі кадри до Assets. На даний момент всього 3 анімації: Idle, Run, Jump. На рис. 1.12 зображено кадри анімації персонажа.

Для того щоб додати їх до персонажа, виділяю дочірній об’єкт Sprite та відкриваю вікно Animation, яке визивається сполученням клавіш CTRL+6.

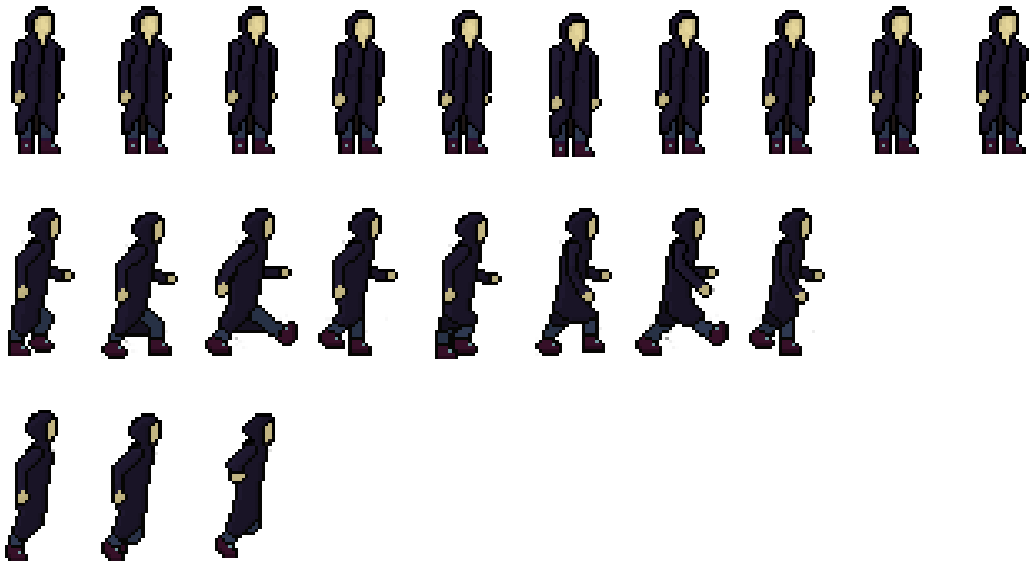


Рисунок 1.12. Кадри анімації персонажа

Створюю нову анімацію для цього об’єкту, називаю її та перетягую всі кадри однієї з анімацій та налаштовую їх на доріжці аніматора. Головне розставити кадри правильно на таймлайні, щоб анімація не було занадто швидкою чи повільною.

Також в вікні аніматора можна відіграти вибрану анімацію, щоб зрозуміти як вона буде виглядати. Після створення 3 анімацій потрібно зробити переходи у вікні Animator та переключати стани в скрипті. Алгоритм визначення анімації персонажа зображено на рис. 1.13.

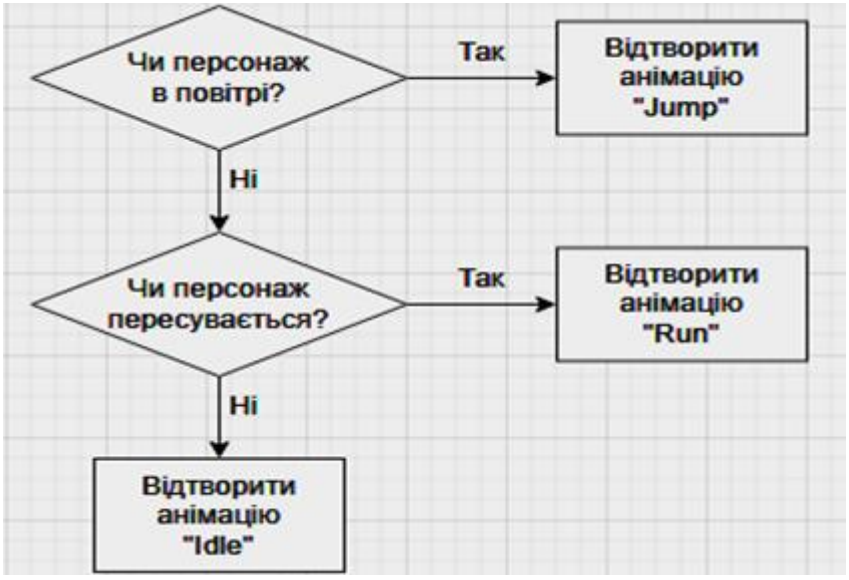


Рисунок 1.13. Алгоритм визначення анімації персонажа

Тепер додаю логіку перемикання між станами в клас персонажа. В скрипті персонажа потрібно отримати компонент Animator у дочірнього об’єкта Sprite, для цього додаю до скрипта наступне:

```
private Transform spriteTransform;
private Animator anim;
void Awake()
{  spriteTransform = transform.Find("Sprite");
        anim = spriteTransform.GetComponent<Animator>(); }
public enum States
{  idle,
run,
jump }
private States State
{  get { return (States)anim.GetInteger("state"); }
set { anim.SetInteger("state", (int)value); } }
```

Компонент Animator буде отримувати стан персонажа від скрипта. Наступне налаштування переходів анімацій. Анімація стрибка не повинна бути зацикленою, тому знаходжу її в Assets та відключаю в неї Loop Time. Переходжу в вікно Animator та налаштовую переходи за допомогою значення state. На рис. 1.14 зображено готові переходи між анімаціями.

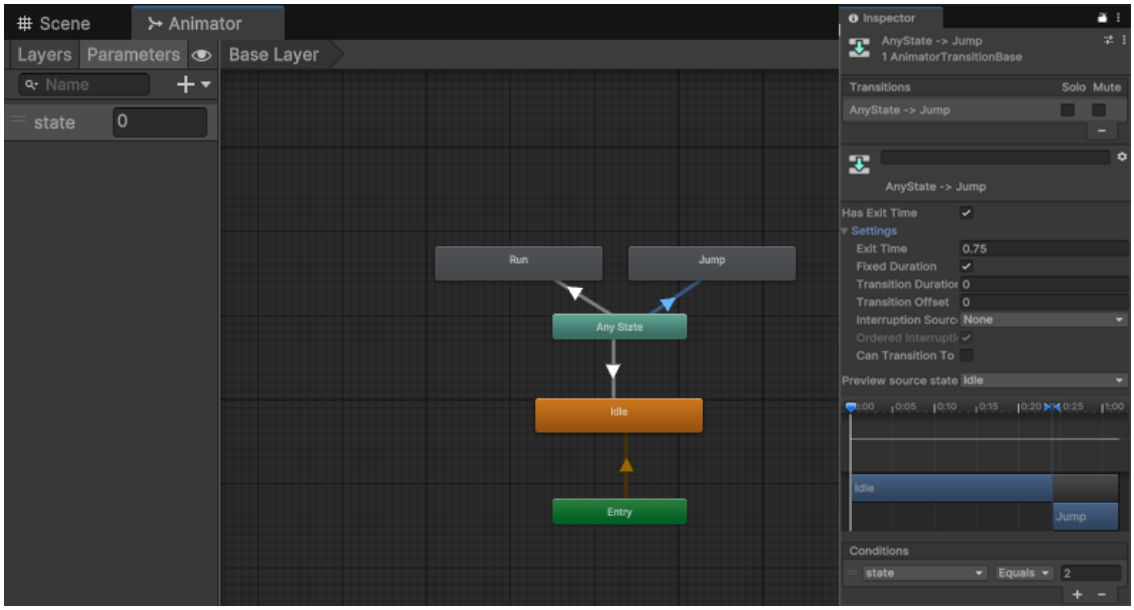


Рисунок 1.14. Готові переходи між анімаціями

Тепер потрібно зробити так, щоб гравець міг розвертатись. Для цього обновив скрипт, додавши функцію розвороту:

```
private bool facingRight = true;
```

```
private void Flip()
{  facingRight = !facingRight;
transform.Rotate(0f, 180f, 0f); }
```
Та додав перевірку в метод Run:
```
if ((moveInput < 0 && facingRight) || (moveInput > 0 && !facingRight))
Flip();
```

Тепер у персонажа є всі основні функції управління. Також було вирішено додати трохи зручності в механіку стрибка, зараз персонаж стрибає лише якщо під його ногами є земля і натискається клавіша стрибку, що може визвати незручність, якщо клавішу стрибка натиснули трохи раніше того як персонаж приземлиться. Тому було додано дві змінні, які будуть відраховувати час, якщо гравець натиснув клавішу стрибка перед приземленням і через невеликий проміжок часу персонаж торкається землі, то персонаж одразу ж стрибне. Гравець може навіть не помічати цього, але це спростить управління та уникати проблем коли гравець натиснув клавішу, а персонаж не стрибнув. Друга змінна буде відраховувати час, коли гравець торкається землі і якщо він натисне клавішу стрибка після невеликого проміжку часу після того як персонаж зійде з краю, то персонаж все одно стрибне. Це дозволить простіше стрибати з країв і дозволить уникати ситуацій, коли клавішу стрибка натиснули саме в той момент коли персонаж перестав торкатись землі. Фрагмент коду зображений нижче:

```
private float timerForJump;
private float airJumpTimer;
void Update(){
  if(isGrounded)
      timerForJump = Time.time;
  if (!isGrounded && Input.GetButtonDown("Jump"))
      airJumpTimer = Time.time;
if((isGrounded || Time.time - timerForJump < 0.15) &&
(Input.GetButtonDown ("Jump") || (airJumpTimer > 0 && Time.time -
airJumpTimer < 0.15)))
    Jump(); }
```

1.7.4 Створення ігрового інтерфейсу та системи взаємодії з предметами

Наступним етапом розробки стало додавання інтерфейсу. . Для цього у сцену додав об’єкт Canvas, який використовується для розміщення UI-елементів. Першим елементом став індикатор здоров’я, реалізований за допомогою

компонентів Image та Slider. Для початку додаю пустий об’єкт з компонентом Slider та власним скриптом. До нього додаю 3 дочірні об’єкти:

-Fill – зображення одного кольору, яке буде зменшатись відносно поточного здоров’я персонажа.

-Border – зображення рамки індикатора здоров’я.

-Text – текст, який відображає поточне та максимальне здоров’я персонажа.

Border і Text повинні бути вище ніж Fill. До скрипта батьківського об’єкта додаю наступне:

```
using UnityEngine.UI;
public class HealthBar : MonoBehaviour
{  [SerializeField] private Slider slider;
[SerializeField] private Gradient gradient;
[SerializeField] private Image fill;
[SerializeField] private Text healthText;
public void SetHeath(int maxHP, int currentHP)
{  slider.maxValue = maxHP;
slider.value = currentHP;
fill.color = gradient.Evaluate(slider.normalizedValue);
healthText.text = currentHP + " / " + maxHP; } }
```

Тепер в інспекторі у індикатора здоров’я встановлюю до значень Image fill та Text healthText дочірні об’єкти, а для Gradient gradient встановлюю градієнт від червоного до зеленого, щоб коли здоров’я повне смужка індикатора була зеленого кольору, а чим нижче до 0 тим червонішою вона ставала. В значення Slider slider встановлюю компонент Slider, в якому встановлений об’єкт Fill, через що чим нижче здоров’я персонажа, тим менше стає смужка індикатора. Для того щоб індикатор здоров’я працював потрібно в скрипті ігрового персонажа викликати метод відображення здоров’я, вказуючи максимальне та поточне здоров’я персонажа.

Наступним елементом інтерфейсу є інвентар. Для початку створюю пустий батьківський об’єкт, в якому будуть всі дочірні об’єкти. Далі додаю до пустого об’єкта дочірній об’єкт, сіре напівпрозоре зображення, яке позаду інвентаря та буде перекривати зображення ігрової сцени, далі додаю зображення інвентаря, до якого додаю 4 дочірні об’єкти: InventorySlotsGrid, SpellsSlotsGrid,

WeaponSlotGrid, PassiveSlotGrid. До кожного з цих об’єктів додаю компонент «Grid Layout Group» та налаштовую його. Ці пусті об’єкти використовуються як сітка для слотів.

Далі створюю слот, до якого додаю той же компонент, щоб предмет в слоті завжди був посередині. Далі створюю предмет, для цього використовую ScriptableObject. Це спеціальний клас, який дозволяє зберігати дані окремо від сцени або об’єктів. Можна представити його як контейнер для інформації, який не потрібно прив’язувати до конкретного об’єкту на сцені. Для коректного розподілу на заклинання, зброю та пасивні здібності було вирішено створити батьківський клас Item, який буде унаслідуватись від ScriptableObject та класи Spell, Weapon та Passive, які будуть унаслідуватись від Item. В класі Item буде зберігатись інформація, яку мають всі предмети, такі як зображення, назва та тип, а в інших класах буде зберігатись інформація, яка відноситься конкретно до цього типу предмета. Нижче приведений код цих класів:

```
[CreateAssetMenu(fileName = "New Item", menuName = "Inventory/Item")]
public class Item : ScriptableObject {
public string itemName;
public Sprite icon;
public ItemType itemType; }
public enum ItemType {
Spell,
Weapon,
Passive }
public class Spell : Item {
public int damage;
public float cooldown; }
public class Weapon : Item {
public int damage;
public float attackCooldown;
public float attackRange; }
public class Passive : Item
{public string description;}
```

CreateAssetMenu дозволяє створювати об’єкти типу ScriptableObject прямо в меню Unity. Тепер для коректної роботи інвентаря потрібно створити 3 нових скрипти: InventoryManager, InventorySlot та DraggableItems. Структурна блок-схема логіки інвентаря зображено на рис. 1.15.

InventoryManager буде у пустого об’єкта на сцені, який буде знаходити перший незаповнений слот в інвентарі та додавати до нього предмет.

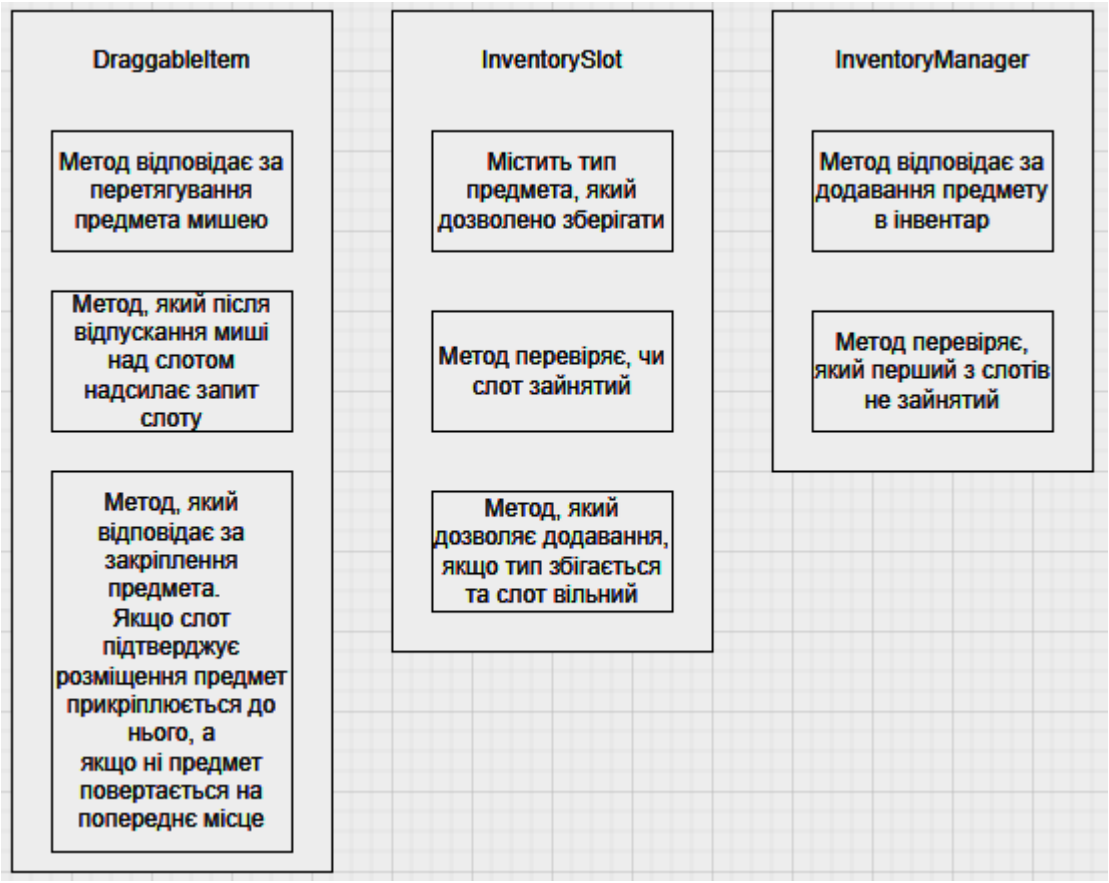


Рисунок 1.15. Структурна блок-схема логіки інвентаря

InventorySlot буде у кожного окремого слота в інвентарі, він буде перевіряти чи предмет підходить по типу для слота, якщо так, тоді прийматиме предмет, як дочірній об’єкт. DraggableItems – скрипт, який буде у кожного предмету, він буде перевіряти чи курсор перетягує його, пересуватись за курсором та перевіряти коли курсор відпускає предмет і якщо під предметом не має пустого слота інвентаря, тоді предмет повертається на місце, де був до пересування, а якщо під ним буде пустий слот інвентаря, тоді предмет стає дочірнім об’єктом для слота і через те, що кожен слот має компонент Grid Layout Group, предмет завжди буде посередині слота.

Скрипти предмета та слота потрібно створювати та перевіряти разом. Для початку створив скрипт предмета:

```
public class DraggableItems : MonoBehaviour, IBeginDragHandler,
IDragHandler, IEndDragHandler {
    public Image iconImage;
```

```
private Item itemData;
public Transform parentAfterDrag;
public void Initialize(Item newItem, TooltipUI tooltipUI) {
    itemData = newItem;
    iconImage.sprite = newItem.icon;
    tooltip = tooltipUI; }
public void OnBeginDrag(PointerEventData eventData) {
    parentAfterDrag = transform.parent;
   transform.SetParent(transform.root);
   transform.SetAsLastSibling();
   iconImage.raycastTarget = false; }
public void OnDrag(PointerEventData eventData) {
   transform.position = Input.mousePosition; }
   public void OnEndDrag(PointerEventData eventData) {
   transform.SetParent(parentAfterDrag);
   iconImage.raycastTarget = true; }
public Item ItemData()
   {return itemData;} }
```

Цей скрипт реалізує інтерфейси для обробки подій перетягування. Цей скрипт також має метод доступу, який повертає значення поля itemData. Одразу створюю скрипт для слотів:

```
public class InventorySlot : MonoBehaviour, IDropHandler {
  [SerializeField] private SlotType allowedType;
  public void OnDrop(PointerEventData eventData) {
    if (transform.childCount == 0)  {
      GameObject dropped = eventData.pointerDrag;
      DraggableItems draggableItems = dropped.GetComponent
<DraggableItems>();
      if (IsItemAllowed(draggableItems))
        draggableItems.parentAfterDrag = transform;  } }
  private bool IsItemAllowed(DraggableItems item) {
    if (allowedType == SlotType.Any) return true;
    return allowedType.ToString() == item.ItemData().itemType.ToString();}
  public enum SlotType { Any, Spell, Weapon, Passive } }
```

Після цього додаю ці скрипти на предмет та слот, роблю їх префабами та видаляю предмет зі сцени, тому що предмет буде додаватись на сцену за допомогою InventoryManager. Вставляю в сітку для слотів префаби слотів, створюючи основний інвентар з 15 слотів, 5 слотів для заклинань і по 1 слоту для зброї та пасивної здібності і у кожного слота назначаю його тип.

Наступним кроком потрібно зробити InventoryManager та вікно підказки, яке буде відображати знизу екрана інформацію предмета на який наведено курсор. Для цього спочатку створюю новий елемент інтерфейсу, який має текст, який буде змінюватись відносно інформації предмета та власний скрипт, який отримує інформацію від предмета та змінює свій текст та колір. Для цього потрібно до скрипта предмета додати інтерфейс IPointerEnterHandler, IPointerExitHandler та реалізувати його в коді:

```
public void OnPointerEnter(PointerEventData eventData)
{  tooltip.ShowTooltip(itemData); }
public void OnPointerExit(PointerEventData eventData)
{  tooltip.HideTooltip(); }
```

Тепер потрібно створити InventoryManager, для цього в його скрипті створюю змінні для префаба предмета та масив слотів інвентаря, щоб менеджер міг знайти перший вільний слот перевіряючи, які з слотів зайняті:

```
public void AddItem(Item item) {
for (int i = 0; i < inventorySlots.Length; i++) {
  InventorySlot slot = inventorySlots[i];
DraggableItems itemSlot = slot.GetComponentInChildren
<DraggableItems>();
    if (itemSlot == null) {
      SpawnItem(item, slot);
      return; } } }
```

В метод створення предмета передається в який слот потрібно створити предмет та який саме предмет створити.

Наступними кроками стане створення панелі швидкого доступу, яка отримує інформацію, які заклинання споряджені та відображає їх і зробити так, щоб гравець міг використовувати споряджені предмети. Для початку створюю панель швидкого доступу, для цього створюю новий елемент інтерфейсу, в ньому сітку для слотів і 5 слотів. Всередині кожного слоту створюю зображення, яку буде змінюватись відносно заклинання в слоті та сіре напівпрозоре зображення, яке буде повністю відображатись після використання заклинання та поступово зменшуватись відносно перезарядки вибраного заклинання, тим самим відображаючи стан його перезарядки. В скрипті слота в панелі швидкого доступу повинні бути наступні змінні:

-зображення самого слота, щоб змінювати його колір.

-2 кольори для вибраного слота та не вибраного для того, щоб гравець розумів, який слот зараз вибраний.

-змінна для слоту, який відповідає за слот в панелі швидкого доступу, наприклад перший слот в панелі має змінну першого слота в інвентарі з споряджених заклинань.

-зображення дочірнього об’єкту, який відповідає за відображення іконки заклинання.

-зображення дочірнього об’єкту, який відповідає за відображення перезарядки вибраного заклинання.

-змінна з компонентом Slider, щоб зображення перезарядки поступово зменшувалось.

В скрипті слота додав метод, який перевіряє чи є у відповідному слоті інвентаря предмет, якщо ні, тоді зображення заклинання просто не відображається, але якщо є, тоді скрипт отримує інформацію про предмет та змінює іконку на зображення заклинання. Також цей скрипт повинен мати метод, який перезаряджає заклинання після його використання та перевіряє чи перезарядка завершилась. Метод перезарядки буде викликатись з скрипта гравця.

Тепер панель швидкого доступу отримує всю інформацію про заклинання, яке споряджено в інвентаря. Останнім кроком в цьому підрозділі буде реалізація використання споряджених предметів гравцем. Для цього в скрипті гравця потрібні змінні для:

-слотів з панелі швидкого доступу.

-слота зброї в інвентарі.

-слота пасивної здібності в інвентарі.

-пустого об’єкта в якому знаходяться дочірні елементи інвентаря.

Для перемикання між слотами в панелі швидкого доступу, потрібно зчитувати натискання клавіш 1-5 і перемикати вибраний слот.

```
void Update() {
  if (Input.inputString != null) {
    bool isNumber = int.TryParse(Input.inputString, out int number);
```

```
        if (isNumber && number > 0 && number < 6)
          ChangeSelectedSlot(number - 1); } }
    void ChangeSelectedSlot(int newValue) {
      if (selectedSlot >= 0)
        spellBarSlots[selectedSlot].Deselect();
      spellBarSlots[newValue].Select();
      selectedSlot = newValue; }
```

Методи Select та Deselect в скриптах слотів панелі швидкого доступу змінюють колір. Тепер потрібно додати можливість відкривати інвентар на клавішу Tab, для цього при натисканні клавіші Tab буде вмикати чи вимикати об’єкт інвентар на сцені. Після цього гравець зможе відкривати інвентар пересувати в ньому предмети, споряджати їх і тепер потрібно реалізувати їх використання.

Було вирішено почати з заклинань. На рис. 1.16 зображено блок-схему логіки використання заклинань гравцем. Скрипт персонажа отримує назву заклинання у вибраному слоті панелі швидкого доступу, тепер потрібно робити перевірку, чи натискає гравець праву кнопку миші та яка назва заклинання в вибраному слоті, після чого виконувати метод, в якому гравець використовує це заклинання.

Вирішив почати з заклинання метеора:

```
    if (Input.GetMouseButtonDown(1) && !isInventoryOpen && !isCasting) {
      if (selectedSpell == "Meteor" && spellBarSlots[selectedSlot].IsReady()) {
        StartCoroutine(CastMeteor());
        spellBarSlots[selectedSlot].TriggerCooldown(); } }
    IEnumerator CastMeteor() {
      isCasting = true;
         State = States.cast1;
      yield return new WaitForSeconds(0.2f);
      isCasting = false;
      GameObject meteor = Instantiate(meteorPrefab, spellPoint.position,
Quaternion.identity);
          Vector2 launchDir = facingRight ? Vector2.right : Vector2.left;
      meteor.GetComponent<Meteor>().Launch(launchDir, spellAmp); }
```

Для використання цього методу персонажу потрібно додати змінні з точкою, звідки буде випускатись метеор та префабом метеора, в якому є скрипт, який задає напрямок польоту.

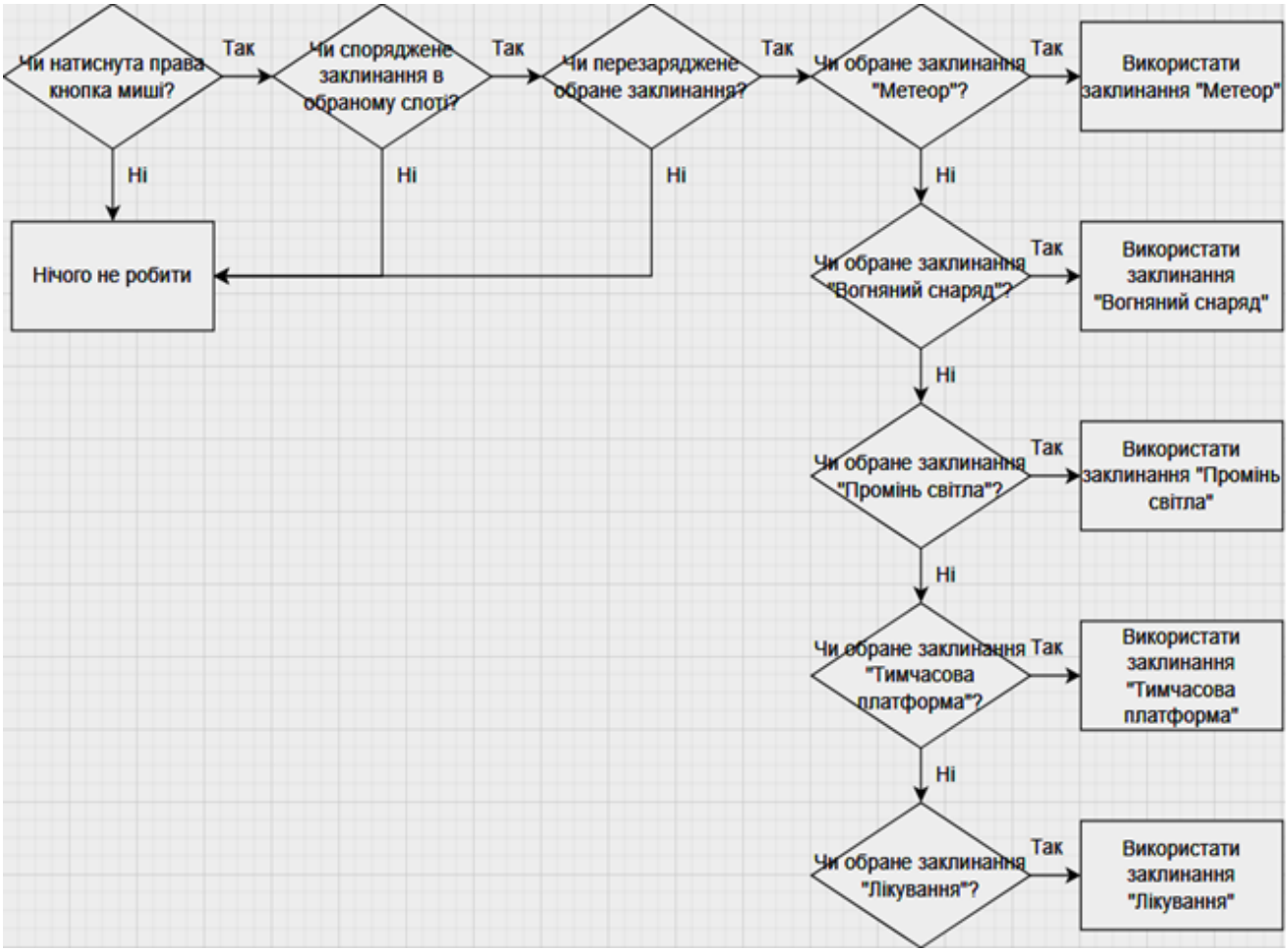


Рисунок 1.16. Блок-схема логіки використання заклинань гравцем

Скрипт метеора наведено нижче:

```
public void Launch(Vector2 direction, float _spellAmp) {
    Destroy(gameObject, 2.5f);
    spellAmp = _spellAmp;
    rb = GetComponent<Rigidbody2D>();
    rb.AddForce(new Vector2(direction.x * launchForce, upwardForce),
ForceMode2D.Impulse); }
```

Також є метод, який наносить шкоду при зіткненні з коллайдером об’єкта з тегом «Enemy». Виклик у 4 інших заклинань такий самий, по назві, тому розглянемо тільки чим вони відрізняються. У заклинання вогняного снаряду за напрямок береться положення курсору на екрані і якщо гравець випускає снаряд за спину персонажа, тоді виконується метод розвороту.

У заклинання промінь світла теж курсор береться за напрямок, але персонаж випускає не префаб, а за допомогою метода RayCast випускає промінь.

В коді це реалізовано наступним чином:

```
private IEnumerator ShootRayOfLight() {
```

```
        isCasting = true;
        State = States.cast1;
           yield return new WaitForSeconds(0.2f);
           rayOfLight.enabled = true;
           isCasting = false;
           rayOfLight.SetPosition(0, spellPoint.position);
           int layerMask = (1 << LayerMask.NameToLayer("Enemy"))
      RaycastHit2D hitInfo = Physics2D.Raycast(spellPoint.position, direction, 50f,
layerMask);
        if (hitInfo) {
             Entity enemy = hitInfo.transform.GetComponent<Entity>();
             if (enemy.CompareTag("Enemy"))
            enemy.GetDamage(Mathf.RoundToInt(rayOfLightData.damage));
          rayOfLight.SetPosition(1, hitInfo.point); }
        else
          rayOfLight.SetPosition(1, (Vector2)spellPoint.position + direction * 50);
        rayOfLight.enabled = false; }
```

Якщо промінь стикається з будь-яким об’єктом на шарі «Enemy», тоді промінь малюється від персонажа до точки зіткнення та ворог, в якого поцілив луч отримує шкоду.

У заклинання тимчасова платформа, як і з першими двома викликається префаб платформи під персонажем. В скрипті платформи є лише метод виклику, в якому вказується тривалість її дії і метод знищення ворожих снарядів при зіткненні.

І останнє заклинання лікування, воно додає до поточного здоров’я 50% від максимального здоров’я. Наступний тип предмету є зброя, для його реалізування скрипт гравця отримує інформацію чи споряджена в інвентарі зброя, якщо так, тоді скрипт персонажа отримує інформацію про цю зброю. Для атаки було створено окремий дочірній для персонажа об’єкт «DamageCollider» з компонентом коллайдер та власним скриптом, в якому отримує інформацію чи гравець атакує та шкоду зброї. Якщо гравець атакує, тоді всі об’єкти з тегом Enemy, які пересікаються з коллайдером зброї персонажа, отримують шкоду.

В скрипті гравця створюю метод, який міняє положення та розмір Damage Collider’а відносно дальності атаки спорядженої зброї:

```
attackCollider.localPosition = new Vector3(AttackColliderPosition(),1.46f, 0);
attackCollider.localScale = new Vector3(attackRange, 1, 1);
```

```
private float AttackColliderPosition() {
  switch (attackRange) {
case 1.2f: return 0.75f;
case 1.5f: return 0.9f;
case 2f: return 1.2f;
default: return 0.6f; } }
```

Останнім типом предметів є пасивна здібність. Для їх реалізації скрипт персонажа отримує інформацію чи споряджена пасивна здібність, якщо так, тоді скрипт отримує назву пасивної здібності та персонаж отримує певне посилення. Нижче можна побачити фрагмент коду для реалізації пасивних здібностей:

```
maxJumps = selectedPassive == "Double Jump" ? 1 : 0;
moveSpeedBoost = selectedPassive == "Speed Boost" ? 1.35f : 1f;
spellAmp = selectedPassive == "Arcane Mastery" ? 1.35f : 1f;
attackAmp = selectedPassive == "Power Strike" ? 1.5f : 1f;
maxHP = selectedPassive == "Vitality Boost" ? 150 : 100;
```

Тепер персонаж може отримати додатковий стрибок, прискорення, посилення заклинань, посилення ближнього бою чи підвищення максимального здоров’я.

1.7.5 Реалізація вибору нового предмету

Для реалізації вибору нових предметів потрібно створити інтерфейс та менеджер для цього. Створив об’єкт інтерфейсу, який буде слугувати сіткою для трьох «карток» з предметами. Далі створив об’єкт на якому є зображення та текст, який буде змінюватись відносно предмету, який буде відображатись та кнопку «Взяти». Також до «картки» додаю скрипт, який буде змінювати текст та іконку на об’єкті. Роблю цей об’єкт префабом та створюю новий пустий об’єкт на сцені з назвою RewardManager, який має власний скрипт. В цей скрипт додаю префаб «карточки», а також додаю список предметів(ScriptableObject), з яких випадково будуть вибрані 3 предмети. Нижче зображено фрагмент скрипта:

```
public void ShowRewards() {
  if (availableItems.Count == 0)
            return;
  rewardPanel.SetActive(true);
  isPickUIActive = true;
  foreach (Transform child in rewardSlotsGrid)
    Destroy(child.gameObject);
```

```
List<Item> chosenItems = availableItems.OrderBy(x => Random
.value).Take(3).ToList();
    foreach (var item in chosenItems) {
      GameObject slot = Instantiate(rewardSlotPrefab, rewardSlotsGrid);
      RewardSlotUI ui = slot.GetComponent<RewardSlotUI>();
        ui.Setup(item, () => OnItemSelected(item)); } }
  private void OnItemSelected(Item selectedItem) {
    inventoryManager.AddItem(selectedItem);
    availableItems.Remove(selectedItem);
      rewardPanel.SetActive(false); }
```

Після вибору предмета, він видаляється зі списку та додається в інвентар. Останнім етапом до отримання предмету є зробити виклик методу і як було написано в розділі проєктування, вибір предмету дається після знищення всіх ворогів на сцені. Тому наступний крок це зробити сцени та переходи між ними.

1.7.6 Розробка рівнів та переходів між ними

Для розробки сцен потрібно розібратись в тому префаби яких об’єктів будуть використовуватись в кожній сцені та які об’єкти повинні переходити між сценами. Для кожного з рівнів було створено префаб CinemachineCamera. Це об’єкт з інструменту Unity, який дає можливість автоматично стежити за гравцем та ставити кордони, за які камера не буде заходити. Для цього було створено префаб пустого об’єкту-тригеру з коллайдером та названий «Frame», коллайдер цього об’єкта розтягую та створюю рамки для кожної сцени, щоб камера не показувала гравцеві що знаходиться за рівнем.

Також для префабів об’єктів, які повинні бути в кожній сцені відносяться:

- спрайти заднього фону, для цього було намальовано два види заднього фону для вертикальних чи горизонтальних рівнів.

- точка де починається рівень, вона потрібна тому що, коли персонаж переходить між сценами, він опиняється на новій сцені на тих самих координатах на яких вийшов з минулої сцени.

- невидимі стінки, яка буде закривати прохід на наступний рівень, поки гравець не знищить всіх ворогів.

- вихід, який буде переносити персонажа на наступну сцену, коли він знищив всіх ворогів.

До об’єктів, які будуть переноситись між сценами належать вже знайомі: InventoryManager, RewardManager, Canvas, Player та EventSystem, який відповідає за відстежування натискання кнопок миші по елементам інтерфейсу. А також два нових об’єкти: RoomsManager та ReloadManager, які відповідають за вибір випадкового рівня без повторів та повного перезапуску чи виходу в меню. На рис. 1.17 зображено список створених сцен.

Для того, щоб об’єкти не зникали при переході між рівнями потрібно щоб вони були у стартовій сцені та додати до них скрипт з наступною командою:

DontDestroyOnLoad(gameObject);

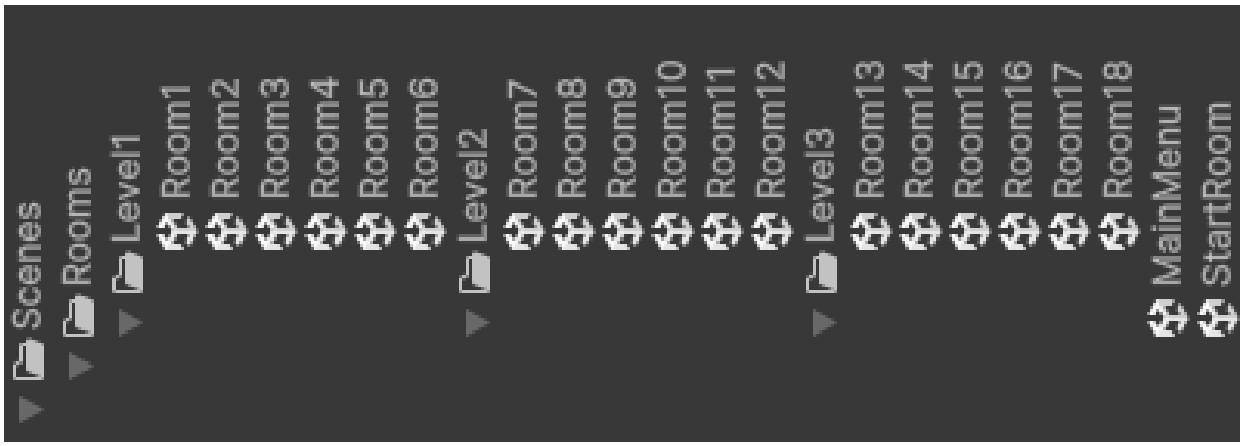


Рисунок 1.17. Список створених сцен

RoomsManager має 3 списки в яких по 6 сцен. Кожен список відповідає за окрему складність і якщо в списку залишається менше двох сцен, тоді складність підвищується. Це було зроблено, тому що було вирішено, що гравець не повинен тільки через випадковість зустріти на початку гри ту ж кімнату, яку він може зустріти вкінці гри, тому список сцен від 1 до 18, перший рівень складності 1-6, другий рівень 7-12, а третій 13-18. При заході в нову сцену RoomsManager одразу передає об’єкту Exit, який буде наступний рівень. Скрипт виходу написаний нижче:

```
    private void Start() {
    rewardManager = GameObject.FindWithTag("RewardManager")
.GetComponent <RewardManager>();
    roomManager = GameObject.FindWithTag("RoomsManager").GetComponent
<RoomsManager>();
     nextSceneName = roomManager.GetRandomRoom();  }
```

```
private void Update() {
  if(CanExitRoom() && !rewardGiven) {
      rewardGiven = true;
    rewardManager.ShowRewards(); } }
private void OnTriggerEnter2D(Collider2D other) {
  if (other.CompareTag("Player") && CanExitRoom())
    SceneManager.LoadScene(nextSceneName); }
private bool CanExitRoom() {
    return GameObject.FindGameObjectsWithTag("Enemy").Length == 0; }
```

Для ReloadManager потрібно створити простий інтерфейс програшу, на якому буде напис «Game Over» та дві кнопки «Рестарт» та «Вийти в головне меню». При натисканні на обидві кнопки будуть знищуватись всі об’єкти після чого сцена зміниться або на стартову кімнату або на головне меню. Екран програшу повинен відображатись тільки після смерті персонажа, тому в його скрипті створюю метод в якому блокується пересування та взаємодія. Фрагмент коду зображено нижче:

```
   private IEnumerator Death(){
State = States.die;
yield return new WaitForSeconds(1.2f);
GetComponent<Collider2D>().enabled = false;
deathScreen.SetActive(true); }
```

1.8 Створення ворогів та їх поведінки

У розділі 1.6 «Проєктування різних видів ворогів» було детально розглянуто типи ворогів, їхню роль у геймплеї та особливості поведінки. Для початку створив клас, від якого будуть унаслідуватись скрипти ворогів та гравця. В ньому будуть всі основні змінні та методи отримання шкоди, розвороту, Awake, в якому об’єкт буде отримувати компоненти по типу: Rigidbody2D, groundLayer, Sprite, тощо. В методі отримання шкоди буде просто відніматись від поточного здоров’я шкода, а також потрібно додати візуальну зміну, щоб гравець зрозумів, що він або ворог отримав шкоду. Для цього в скрипті Entity від якого унаслідуються скрипти гравця та ворогів додав дві змінні кольорів: notHurtColor = Color.white, білий колір означає, що спрайт буде звичайного кольору, hurtColor = new Color(1f, 0.37f, 0.37f, 0.7f), що робить спрайт трохи

червоним та напівпрозорим. Далі потрібно отримати доступ до зміни кольору спрайта, для цього додаю до скрипта:

```
private Color hurtColor = new Color(1f, 0.37f, 0.37f, 0.7f), notHurtColor = Color.white;
protected Transform spriteTransform;
protected SpriteRenderer sprite;
```

Після чого додаю метод отримання шкоди, в якому буде даватись час між можливістю отримувати шкоду, щоб об’єкт не отримував шкоду кожен кадр та метод, який міняє колір спрайта на деякий час:

```
public virtual void GetDamage(int damage) {
if (Time.time - lastTimeGetHit > getDamagedCooldown)
{  StartCoroutine(Hurt());
currentHP -= damage;
lastTimeGetHit = Time.time; } }
protected virtual IEnumerator Hurt()
{  sprite.color = hurtColor;
yield return new WaitForSeconds(0.8f);
sprite.color = notHurtColor; }
```

Було використано корутину. Корутин – це метод, який не виконується одразу повністю, а може призупинятись, чекати певний час або умову, і продовжувати далі. Обов’язковою умовою використання корутину є використання певної умови, наприклад «yield return new WaitForSeconds(0.8f)», через яку метод очікує 0.8 секунд та визивати метод за допомогою команди «StartCoroutine(Hurt())».

Тепер коли є метод отримання шкоди потрібно додати джерело шкоди. Для початку було вирішено додати перешкоду, яка буде наносити шкоду персонажу. Для цього створюю простий спрайт без анімацій у вигляді шипів, додаю до нього коллайдер у вигляді круга, в інспекторі у коллайдера увімкнув параметр «isTrigger» та додаю скрипт, який наносить шкоду, якщо в коллайдер цього об’єкта увійде коллайдер об’єкта з тегом «Player»:

```
private int damage = 10;
private void OnTriggerStay2D(Collider2D other)
{  if (other.CompareTag("Player"))
{  Entity playerDamaged = other.GetComponent<Entity>();
playerDamaged.GetDamage(damage); } }
```

Наступний крок створити ворога слайм, який повинен стрибати на гравця, коли той є в полі зору слайма. Структурну блок-схему класу слайма зображено на рис. 1.18.

Створюю спрайт та анімацію для слайма та імпортую їх в Unity. У слайма буде два коллайдери. Перший є тригером для отримання шкоди у гравця, другий для взаємодії з землею. Слайм був зроблений таким же чином, як і ігровий персонаж. Спрайт це дочірній об’єкт до пустого об’єкта з усіма основними компонентами та скриптом управління. Щоб слайм міг стрибати на гравця потрібно, щоб скрипт отримував місцезнаходження ігрового персонажа, для цього в скрипт, який унаслідувався від Entity додаю наступне:

```
    Transform player;
    protected override void Awake()
{   base.Awake();
player = GameObject.FindGameObjectWithTag("Player").transform; }
void Update()
{   Vector2 target = new Vector2(player.position.x, rb.position.y);  }
```

Рисунок 1.18. Структурна блок-схема класу слайма

Тепер потрібно додати слайму радіус зору та можливість зробити дугоподібний стрибок в точку, де знаходиться гравець. Для цього були додані наступні змінні та методи:

```
    private float jumpForce = 6f;
```

```
private float aggressiveRadius = 6f;
private int damage = 25;
private float attackCooldown = 2.5f;
private float lastTimeAttacked;
private bool isAttacking;
void Update()
{  if (Vector2.Distance(transform.position, player.position) <=
aggressiveRadius && isGrounded&&Time.time - lastTimeAttacked >
attackCooldown&& !isAttacking)
StartCoroutine(Preparing()); }
private IEnumerator Preparing()
{  isAttacking = true;
yield return new WaitForSeconds(1f);
Attack();
isAttacking = false; }
private void Attack()
{  Vector2 startPos = transform.position;
Vector2 targetPos = player.position;
float gravity = Mathf.Abs(Physics2D.gravity.y * rb.gravityScale);
float timeToReachTarget = Mathf.Sqrt(2 * jumpForce / gravity) * 2;
Vector2 jumpVelocity = new Vector2((targetPos.x - startPos.x) /
timeToReachTarget, Mathf.Sqrt(2 * gravity * jumpForce));
rb.linearVelocity = jumpVelocity;
lastTimeAttacked = Time.time; }
```

В результаті отримую слайма, який має дальність зору і якщо в нього заходить гравець слайм починає підготовку до стрибка і через 1 секунду стрибає на позицію ігрового персонажа. І як з скриптом шипів додав метод, який наносить шкоду, якщо гравець входить у коллайдер слайма. Далі, як і з ігровим персонажем імпортую кадри анімацій та додаю до спрайта слайма додаю 3 анімації для кожного з станів: Idle, Preparing, Jump. На рис. 1.19 зображено покадрову анімацію слайма.

Рисунок 1.19. Кадри анімації слайма

Усі налаштування анімацій такі ж, як у анімації персонажа. Останнім елементом є додавання функції розвороту. Так як цей метод є в батьківському класі, тоді в скрипті слайма потрібно тільки визвати його за допомогою наступної перевірки:

```
if ((player.position.x < transform.position.x &&
facingRight)||(player.position.x > transform.position.x && !facingRight))
Flip();
```

Тепер ворог слайм повністю готовий і так як всі наступні вороги будуть створенні аналогічним способом, буде описане лише те, що в їх скриптах відрізняється.

Наступним ворогом є маг. Маг відрізняється тим, що стоїть на місці, персонаж не отримує від нього шкоди при контакті та може проходити крізь нього і те, що маг випускає снаряди по горизонталі в сторону персонажа. Для початку створюю префаб снаряда. Префаб – це шаблон ігрового об’єкта, для початку створюю об’єкт на сцені, налаштовую його і потім перетягую його з Hierarchy в папку Prefabs в Assets. У префаба снаряда повинен бути CircleCollider2D, Rigidbody2D та власний скрипт, в якому можна надати снаряду напрямок. Окрім нанесення шкоди при вході гравця в коллайдер потрібно додати змінні зі значеннями шкоди, швидкості та метод, який завдає напрямок. Нижче зображено фрагмент коду:

```
private int damage = 15;
private int speed = 10;
private Rigidbody2D rb;
public void Launch(Vector2 direction)
{  Destroy(gameObject, 2.5f);
rb = GetComponent<Rigidbody2D>();
rb.linearVelocity = direction.normalized * speed; }
```

Тепер зберігаю цей снаряд, як префаб та додаю до скрипта мага, за допомогою серіалізованих полів, що дає змогу зробити змінну з закритим доступом, яку можна назначити в інспекторі. Нижче можна побачити фрагмент коду, в якому використовується префаб снаряду для атаки та дочірнього для мага об’єкт, точка запуску снаряда.

```
[SerializeField] private GameObject projectilePrefab;
```

```
[SerializeField] private Transform spellPoint;
private IEnumerator Shoot()
{  isAttacking = true;
State = States.attack;
yield return new WaitForSeconds(1.2f);
isAttacking = false;
GameObject projectile = Instantiate(projectilePrefab, spellPoint.position,
Quaternion.identity);
Vector2 direction = facingRight ? Vector2.right : Vector2.left;
projectile.GetComponent<Projectile>().Launch(direction); }
```

Наступним ворогом є кажан, його особливість це те, що він літає і також випускає снаряди, але вже не тільки по горизонталі, а в направлені гравця. Для того, щоб кажан зміг літати в Rigidbody2D ставлю значення Gravity Scale на 0. Як було написано в розділі проєктування, кажан має радіус зору та радіус атаки, який менше по розмірам ніж перший і якщо гравець входить в радіус зору, але знаходиться все ще за радіусом атаки, тоді кажан наближається до гравця. Для цього до скрипта кажана додаю наступне:

```
private float aggressiveRadius = 15f;
private float attackRadius = 6f;
void Update() {
  float distance = Vector2.Distance(transform.position, player.position);
  if (distance < aggressiveRadius && !isAttacking) {
    if (distance > attackRadius)
      transform.position=Vector2.MoveTowards(transform.position,
player.position, speed * Time.deltaTime);
    else if (Time.time - lastTimeAttacked > attackCooldown) {
      StartCoroutine(Shoot());
      lastTimeAttacked = Time.time; } } }
```

В методі запуску снаряда виконуються ті ж дії, що і в скрипті мага, лише напрямок снаряда задається за допомогою віднімання координат ігрового персонажа від координат кажана.

Останній ворог це примара. Її особливість це те, що вона літає, має великий радіус зору та прискорюється тим більше, чим ближче до ігрового персонажа. Також при нанесенні шкоди персонажу примара на деякий час зупиняється. Для цього в її скрипт було додано наступне:

```
private float speed;
private bool isAggressive;
```

```
    private bool isStopped;
    void Update() {
      float distance = Vector2.Distance(transform.position, player.position);
      if (distance < 5f)
        speed = 7f;
      else if (distance < 12f)
        speed = 4f;
      else if (distance < 22f)
        speed = 1.5f;
      if (isAggressive)
        transform.position = Vector2.MoveTowards(transform.position,player
.position, speed * Time.deltaTime); }
    private void OnTriggerEnter2D(Collider2D other){
      if (other.CompareTag("Player")) {
        Entity playerDamaged = other.GetComponent<Entity>();
        playerDamaged.GetDamage(damage);
        StartCoroutine(Stop()); } }
    private IEnumerator Stop(){
      isStopped = true;
      yield return new WaitForSeconds(1.8f);
      isStopped = false; }
```

Примара має 3 значення швидкості та чим ближче до гравця, тим більша швидкість. Всі види ворогів готові, тепер потрібно зробити кожного префабом та використовувати для будування рівнів.

1.9 Тестування розробленої гри

Після реалізації всіх основних ігрових механік дуже важливим кроком є комплексне тестування всіх ігрових елементів. Саме на цьому етапі виявляються помилки, які могли залишитись непоміченими під час реалізації окремих компонентів. На цьому етапі перевіряється не лише коректність роботи окремих компонентів, а й їхня взаємодія в межах цілісної системи.

Під час тестування мені потрібно перевірити такі ключові механіки гри як:

- керування персонажем: рух, стрибки, атаки, використання заклинань.

- робота інвентарю та панелі швидкого доступу: коректне додавання предметів, коректну роботу слотів, які приймають лише певний тип предмету, коректне використання заклинань та їх перезарядку в панелі швидкого доступу

та коректну роботу пасивних здібностей. На рис 1.20 зображено тестування отримання нового предмета в інвентар.

- поведінка ворогів: відстеження гравця, нанесення шкоди, отримання шкоди від атак та заклинань гравця.

- інтерфейс: відображення коректної інформації, правильне відображення інтерфейсу.

- переходи між рівнями: коректне відстежування знищення всіх ворогів, коректний вибір випадкових рівнів.

Рисунок 1.20. Тестування отримання нового предмету в інвентар

Через те, що розробка гри на ігровому програмному рушії Unity, процес тестування можна виконувати безпосередньо в самому редакторі. Основною метою тестування було проведення перевірки готовності поточного проєкту. За результатами тестування критичних помилок, які могли б суттєво вплинути на ігровий процес або призвести до краху програми, не було виявлено. Всі ключові системи функціонують стабільно та згідно очікуванням.

На рис. 1.21 зображено тестування заклинань в грі.

Рисунок 1.21. Тестування заклинань в грі

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проєкті розроблено програмний продукт (ПП) – гра «Shady Adventure» з різними видами ворогів. В роботі реалізовано повний цикл створення програмного продукту.

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.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам’яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

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.

Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога.

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

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

Таблиця 2.1. Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, умовн. 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
9. ПП оптимізаційних розрахунків	1300 – 4200

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
5.00	306
6.00	330

На підставі отриманого значення, по таблиці 2.2, визначається укрупнена норма часу на розробку аналога програмного забезпечення, яка коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8:

$T^a p = 229*0.8= 183,2$ (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.

Для розрахунку необхідні наступні коефіцієнти:

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

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

Кт – 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 ПО, що має аналоги програмних продуктів, то код ступеня новизни для мого ПО – В, а значення коефіцієнта $К_н$=0,7. По таблиці 2.4, знаючи код ступеня новизни, тепер можна визначити значення питомих коефіцієнтів трудомісткості (по таблиці 2.3):

L_1=0,12;

L_2=0,11;

L_3=0,61;

Таблиця 2.5. Коефіцієнт 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 від 40 до 60 відсотків існуючих функцій, це значить, що $К_т$=0,7. Тепер потрібно розрахувати трудомісткість технічного завдання, розробки технічного проекту, робочого проекту по кожному етапу окремо:

$$T_{тз}=Ta*L_1*K_н=183,2\ *0,12*0,7=\ 15,39\ (люд/годин) \quad (2.1)$$

$$T_{тп}=Ta*L_2*K_н=183.2\ \ *0,11*0,7\ =\ 17,42\ (люд/годин) \quad (2.2)$$

$$T_{рп}=Ta*L_3*K_н*K_т=183,2\ *0,61*0,7*0,7=54,76\ (люд/годин) \quad (2.3)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап. $N_{тз}$= 2 (стр), $N_{тп}$= 18 (стр), $N_{рп}$= 32 (стр), $N_{пз}$= 50 (стр) – технічне завдання, розробка технічного проєкт, розробка робочого проєкт, пояснювальна записка відповідно. Розрахунок зведений у таблицю 2.6

На основі таблиці 2.6 розраховую тривалість розробки в роках:

$$T_{пп}=T/(8.0*0.73*360)=249,27/(8*0.73*360)= 0,12(р), \quad (2.4)$$

де 8.0 – тривалість робочого дня;
0.73 – коефіцієнт перекладу в календарні дні;

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	Розробка ПП	Контроль керівника	Нормоконтроль
1.ТЗ	$T_{Ртз}$=15,39	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}=0,15*2=0,30$
2.Розробка ТП	$T_{Ртп}$=17,42	$T_{кк}=0,7*N_{тп}=0,7*18=12,6$	$T_{нк}=0,15*N_{тп}=0,15*18=2,7$
3.Розробка РП	$T_{Ррп}$=54,76	$T_{кк}=0,7*N_{рп}=0,7*32=22,4$	$T_{нк}=0,15*N_{рп}=0,15*32=4,8$
4.Розробка ПЗ	$T_{пз}=1,5*N_{пз}=1,5*50$ = 75	$T_{кк}=0,7*N_{тз}=0,7*50=35$	$T_{нк}=0,15*N_{пз}=0,15*50$ =7,5
Усього, в т.ч.:	$T_{пп}$=249,27		
- на розробку	$\Sigma T_р$=162,57		
- контроль керівника		$\Sigma T_{кк}$=71,4	
- нормоконтроль			$\Sigma T_{нк}$=15,3

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 ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2024» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2025 року - 8000 гривень; мінімальну погодинну тарифну ставку – 46 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудоміст. робіт, години	Погодинна тар. ставка, грн.	Розрахунок, грн.
1.Розробка ПП	162,57	46	7478,22
2.Контроль керівника	71,4	46	3284,4
3.Нормоконт-роль	15,3	46	703,8
Усього	-	-	ΣЗо=11466,42

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

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування атеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір А4		100	1,5	150
Транспортно-заготівельні витрати 10%				$В_{тр\_з}=0,1*В_{м1}=15$
Усього				$В_м=В_{м1}+В_{тр\_з}=165$

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

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	165	$В_м$ (див. табл. 2.7)
2. Основна заробітна плата	11466,42	$З_о$ (див. табл. 2.6)
3.Додаткова заробітна плата	1146,6	$Зд = 0,1 \times Зо = 0,1*11466,42$
4.Відрахування до єдиного фонду соціального внеску	2774,8	$Вє.с.в. = 0,22 \times (Зо + Зд)$ $=0,22*(11466,42+1146,6)$
5. Накладні витрати	4586,5	$Внак. = 0,4 \times Зо = 0,4*11466,42$
6. Повна собівартість	20139,32	$С_{пов} = В_м + З_о + З_д + Вє.с.в. + Внак. =$ $165+11466,42+1146,6+2774,8+4586,5$

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

$$П=(С_{пов}*Р)/100= (20139,32*10)/100= 2013,9 \text{ (грн)}, \quad (2.5)$$

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

Оптимальна ціна (кошторисна вартість) визначається по формулі:

$$Ц_о = С_{пов}+П= 20139,32+2013,9=22153,22 \text{ (грн)}, \quad (2.6)$$

Податок на додану вартість визначається по наступній формулі:

$$ПДВ = 0.3 * Ц_о = 0,3*22153,22= 6645,96 \text{ (грн)}, \quad (2.7)$$

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

$$Ц_р = Ц_о+ПДВ = 22153,22+6645,96= 28799,18 \text{ (грн)}. \quad (2.8)$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Основи

Техніка безпеки та охорона праці – це важлива складова організації сучасного виробничого процесу, що забезпечує фізичну безпеку працівника та створює сприятливі умови для ефективної професійної діяльності. Саме від дотримання вимог з охорони праці залежить стабільність технічного циклу та працездатність персоналу.

У цьому розділі буде розглянуто особливості умов праці програміста, для безпечної роботи працівника під час розробки гри «Shady Adventure» з різними видами ворогів.

3.2Аналіз умов праці розробника програмного забезпечення

Сучасна діяльність розробника програмного забезпечення передбачає тривале перебування за комп’ютером, що супроводжується високою зоровою, емоційною та розумовою напругою. Така праця в основному є статичною та виконується сидячи із високою концентрацією, що може призводити до втоми, перенапруження зору та загального зниження працездатності. Небезпечний фактор – це вплив, який може спричинити травму або різке погіршення здоров’я працівника під час роботи.

До основних елементів, які визначають безпечність умов праці програміста належать: правильна організація робочого місця, достатній рівень освітлення, відповідний мікроклімат у приміщенні, ергономічність меблів, наявність перерв для відпочинку, дотримання норм електробезпеки та протипожежного захисту. Ще важливу роль відіграє психологічна стабільність працівника, що може погіршуватись у разі перевтоми, перевищення робочого навантаження або при недотриманні режиму праці та відпочинку.

Усі ці аспекти необхідно враховувати при організації робочого процесу аби зменшити вплив шкідливих факторів і створити сприятливі умови для ефективної та безпечної праці.

3.3 Організація робочого місця працівника

Правильна організація робочого місця програміста має велике значення для підтримки здоров'я та підвищення продуктивності. Висота робочої поверхні робочого столу має регулюватися в межах 680-800 мм, а ширина і глибина – забезпечувати можливість виконання операцій у зоні досяжності моторного поля рекомендовані розміри: 600-1400мм, глибина – 800-1000мм. Робочий стіл повинен мати простір для ніг заввишки не менше ніж 600мм, завширшки не менше ніж 500мм, завглибшки не менше ніж 450мм, на рівні простягнутої ноги не менше ніж 650мм. Робочий стілець має бути підйомно-поворотним, регульованим за висотою, з кутом і нахилу сидіння та спинки і за відстанню від спинки до переднього краю сидіння поверхня сидіння має бути плоскою, передній край – заокругленим. Висота поверхні сидіння має регулюватися в межах 400-500мм, а ширина і глибина становити не менше ніж 400мм. Кут нахилу сидіння – до 15 градусів вперед і до 5 градусів назад. Регулювання за кожним із параметрів має здійснюватися незалежно, легко і надійно фіксуватися. На рис. 3.1 зображено ергономічні вимоги.

Рисунок 3.1 Ергономічні вимоги до робочого місця працівника

Монітор має розташовуватися на оптимальній відстані від очей користувача, що становить 600-700мм, але не ближче ніж за 600мм з урахуванням розміру літерно-цифрових знаків і символів. Розташування екрана

монітору має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30 градусів до нормальної лінії погляду працівника. Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, звернутого до працюючого.

3.4 Вимоги до приміщення для роботи

Для внутрішнього оздоблення приміщень з персональними комп‘ютерами слід використовувати дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі 0,7-0,8, для стін 0,5-0,6. Покриття підлоги повинне бути матовим з коефіцієнтом відбиття 0,3-0,5. Поверхня підлоги має бути рівною, неслизькою, з антистатичними властивостями. Віконні прорізи приміщень для роботи з персональними комп’ютерами мають бути обладнані регульованими пристроями (жалюзі, завіски, зовнішні козирки). Площа на одне робоче місце для користувачів повинна складати не менше 6 кв.м, а об’єм – не менше 20,0 куб.м.

3.4.1 Освітлення

Для освітлення приміщення, у якому працює користувач ПК, використовується змішане освітлення, тобто сполучення природного й штучного освітлення. Для загального освітлення приміщення використовуються газорозрядні лампи типу ЛД. Норма для необхідної освітленості робочого місця становить 300-500 лк. Якщо ці значення освітленості неможливо забезпечити системою загального освітлення, допускається використовувати місцеве освітлення. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана має не перевищувати 300лк

3.4.2 Електробезпека

Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, застосовувати негорючу ізоляцію. У приміщенні, де одночасно експлуатуються понад п’ять

персональних комп’ютерів і периферійних пристроїв, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця операторів (водопровідні труби, батареї опалення, кабелі із заземленим відкритим екраном) мають бути надійно захищені діелектричними щитками або сітками з метою недопущення потрапляння працівника під напругу.

3.4.3 Шум

При розумовій праці, яка вимагає зосередженості припустимий рівень шуму становить 50дБ. Устаткування, що становить джерело шуму (АЦП, принтери тощо), слід розташовувати поза приміщеннями, де знаходяться робочі місця. Для забезпечення допустимих рівнів шуму на робочих місцях слід застосовувати засоби звукопоглинання, вибір яких має обґрунтовуватись спеціальними інженерно-акустичними розрахунками.

3.4.4 Мікроклімат

Приміщення для роботи з персональними комп’ютерами мають бути обладнані системами опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. У приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря. Температура повітря 22-25 ^{0}C, вологість повітря – 40-60%, швидкість руху повітря – 0,1-0,2 м/с.

Для підтримки допустимих значень мікроклімату та концентрації позитивних та негативних іонів необхідно передбачати установки або прилади зволоження та/або штучної іонізації, кондиціювання повітря.

3.5 Пожежна безпека

Для кожного офісного приміщення, як правило, розробляється Інструкція дій персоналу у разі виникнення пожежі. При цьому окремих наказів на кожне приміщення керівник не видає. Такі інструкції необхідно вивішувати на видимих місцях, з ними мають бути ознайомлені всі працівники об’єкта.

Ці інструкції мають вивчатись під час проведення протипожежних інструктажів. В офісах з постійним або тимчасовим перебуванням на них 100 і більше осіб або таких, що мають хоча б одне окреме офісне приміщення із одночасним перебуванням 50 і більше осіб, котрі мають два поверхи і більше, у разі одночасного перебування на поверсі більше 25 осіб, а для одноповерхових – більше 50 осіб, мають бути розроблені і вивішені на видимих місцях плани евакуації людей на випадок пожежі. На рис. 3.2 зображено приклад плану евакуації людей при пожежі

Для ліквідації невеликих осередків пожеж, а також для гасіння пожеж у початковій стадії їх розвитку силами персоналу об'єктів, застосовуються первинні засобі пожежогасіння. Це вогнегасники (вуглекислотні та порошкові), пожежний інвентар (покривала з негорючого полотна, ящики з піском, бочки з водою), пожежний інвентар.

Приміщення, де розміщені робочі місця, мають бути оснащені системою автоматичної пожежної сигналізації і вогнегасниками відповідно до вимог чинного законодавства України. Проходи до засобів пожежогасіння мають бути вільними. У приміщеннях, в яких розташовані робочі місця, слід щоденно робити вологе прибирання.

Рисунок 3.2. Приклад плану евакуації людей при пожежі

ВИСНОВКИ

У процесі виконання дипломного проєкт було проведено аналіз аналогічних вже існуючих проєктів з різними видами ворогів, що дозволило визначити найбільш відповідний жанр – платформер, саме у якому найчастіше використовуються різні види ворогів. Саме на аналізі існуючих проєктів було визначені основні механіки, які потрібно реалізувати в проєкті.

На етапі проєктування були визначенні ключові ігрові системи, такі як бойові механіки, інвентар, вигляд інтерфейсу та поведінка ворогів. Розроблено загальну архітектуру гри з створенням базових класів та їх зв'язки і логіки взаємодій. Саме це дозволило реалізувати злагоджену структуру проєкт, що легко розширюється та підтримується.

У процесі реалізації використовувались сучасні інструменти розробки, а саме: ігровий програмний рушій Unity, мова C#, середовище Visual Studio та графічний редактор Photoshop, в якому були створенні всі графічні елементи, які використовуються в цьому проєкті.

Протягом розробки проводилось тестування кожного доданого елементу та одразу виконувалась відладка, у випадку помилки. Саме тому при останньому етапі тестування всіх елементів одразу проєкт не мав помилок, які могли вплинути на стабільність геймплею. Усі основні функції працюють відповідно задумки, а ігрові механіки коректна взаємодіють між собою. Останнє, що потрібно було тестувати та змінювати є баланс. Баланс предметів, а саме співвідношення шкоди, зручність використання та перезарядки. А також баланс ворогів, їх співвідношення шкоди, поведінки та здоров'я.

В результаті роботи була створена 2D-гра у жанрі платформер з різними видами ворогів, предметами, системою інвентаря та бойовою системою. Створення проєкту дозволив засвоїти основні аспекти геймдеву: аналіз схожих продуктів, вибір інструментів розробки, проєктування, створення графіки, реалізації логіки та тестування. Усі ці етапи були пройдені та це дозволило засвоїти важливі технічні навики.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Подільський О. В. Проектування комп'ютерних ігор для навчання / О. В. Подільський, О. А. Блажко. — Одеса: ФОП «Побута», 2018. — 212 с.;
2. Андрієнко В.М. Програмування мовою C#: Навчальний посібник. — Київ: НАУ, 2020. — 240 с.
3. Красножон В.М. Інформаційні системи та технології в геймдеві. — Запоріжжя: ЗНУ, 2020. — 174 с.
4. Лозинський А. В. Технології розробки ігрових застосунків: навчальний посібник. — Івано-Франківськ: ІФНТУНГ, 2021. — 128 с.
5. Черних І. І. Розробка 2D-гри на платформі Unity / І. І. Чернов. — Миколаїв : ЧНУ ім. Петра Могили, 2022. — 49 с.;
6. Бондаренко О.С. Unity: основи розробки 2D ігор. — Дніпро: ДНУ, 2022. — 190 с.
7. Завадський Ю.С. Проектування і розробка комп'ютерних ігор: навчальний посібник. — Львів: НУ "ЛП", 2021. — 160 с.
8. Кравченко І. П. Графічне програмування в Unity: теорія та практика. — Черкаси: ЧДТУ, 2020. — 150 с.
9. Черних О. П., Гряник Г. В., Батирєв Р. В., Комендант О. О. Інженерія комп'ютерних ігрових технологій / Черних О. П. та ін. — Харків: НТУ «ХПІ», 2024. — 86 с.;
10. Джозеф Хокінг Unity в дії / Джозеф Хокінг., 2018. – 335 с.;
11. Ляшенко О.М., Електронний навчальний посібник «Розробка комп'ютерних ігор за допомогою Unity 3D», Херсон: видавництво ФОП Вишемирський В.С., 2018.;
12. Unity Technologies. Unity Manual [Електронний навчальний посібник]. — Unity Technologies, 2025., https://docs.unity.com/en-us. (Дата звернення 28.05.2025)

ДОДАТОК А. Лістинг коду основних модулів гри мовою C#

Скрипт Entity

```
using UnityEngine;
using System.Collections;
public class Entity : MonoBehaviour
{
    ...
    protected virtual void Awake()
    {
        facingRight = true;
        groundLayer = LayerMask.GetMask("Ground");
        rb = GetComponent<Rigidbody2D>();
        spriteTransform = transform.Find("Sprite");
        sprite = spriteTransform.GetComponent<SpriteRenderer>();
        anim = spriteTransform.GetComponent<Animator>();
    }
    public virtual void GetDamage(int damage)
    {
        if (Time.time - lastTimeGetHit > getDamagedCooldown)
        {
            StartCoroutine(Hurt());
            currentHP -= damage;
            lastTimeGetHit = Time.time;
        }
    }
    protected virtual IEnumerator Hurt()
    {
        sprite.color = hurtColor;
        yield return new WaitForSeconds(0.8f);
        sprite.color = notHurtColor;
    }
    protected virtual void Flip()
    {
        facingRight = !facingRight;
        transform.Rotate(0f, 180f, 0f);
    }
}
```

Скрипт Player_Controller

```
using System.Collections;
using UnityEngine;

public class Player_Controller : Entity
{
    ...
    void Update()
    {
        ...
        mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        direction = (mousePos - (Vector2)spellPoint.position).normalized;
        isInventoryOpen = inventoryUI.activeSelf;
        selectedSpell = spellBarSlots[selectedSlot].GetSpellName();
        if (Input.GetMouseButtonDown(1) && !isInventoryOpen && !isCasting && !isPickUIActive)
        {
            if (selectedSpell == "Temporary platform" && spellBarSlots[selectedSlot].IsReady())
            {
                SpawnTemporalyPlatform();
                spellBarSlots[selectedSlot].TriggerCooldown();
```

```
            }
            ...
        }
        if (Input.GetMouseButtonDown(0) && !isCasting && !isInventoryOpen && !isPickUIActive && hasWeapon && Time.time - lastTimeAttack >= attackCooldown)
        {
            StartCoroutine(AttackAnimation());
        }
        if (Input.GetKeyDown(KeyCode.Tab) && !isPickUIActive)
            inventoryUI.SetActive(!isInventoryOpen);
        ...
        if (Input.inputString != null)
        {
            bool isNumber = int.TryParse(Input.inputString, out int number);
            if (isNumber && number > 0 && number < 6)
                ChangeSelectedSlot(number - 1);
        }
        CheckGround();
        UpdateWeapon();
        UpdatePassive();
    }
    ...
    private IEnumerator AttackAnimation()
    {
        isCasting = true;
        State = States.attack;
        yield return new WaitForSeconds(0.2f);
        isAttacking = true;
        yield return new WaitForSeconds(0.3f);
        isAttacking = false;
        isCasting = false;
        lastTimeAttack = Time.time;
    }
    ...
    private IEnumerator Death()
    {
        inventoryUI.SetActive(false);
        State = States.die;
        yield return new WaitForSeconds(1.2f);
        rb.bodyType = RigidbodyType2D.Kinematic;
        GetComponent<Collider2D>().enabled = false;
        deathScreen.SetActive(true);
    }
}
```

Скрипт Slime_Controller

```
using UnityEngine;
using System.Collections;
public class Slime_Controller : Entity
{
    ...
    private void Update()
    {
        ...
        if ((player.position.x < transform.position.x && facingRight)||(player.position.x > transform.position.x && !facingRight))
            Flip();
        if (Vector2.Distance(transform.position, player.position) <= aggressiveRadius && isGrounded && Time.time - lastTimeAttacked > attackCooldown && !isAttacking)
            StartCoroutine(Preparing());
```

```
            CheckGround();
        }
    }
        ...
        private void OnTriggerStay2D(Collider2D other)
        {
            if (other.CompareTag("Player"))
            {
                Entity playerDamaged = other.GetComponent<Entity>();
                playerDamaged.GetDamage(damage);
            }
        }
        ...
    }
```

Скрипт Mage_Controller

```
    ...
    public class Mage_Controller : Entity
    {
        ...
        void Update()
        {
            if (Time.time - lastTimeAttacked > attackCooldown)
            {
                StartCoroutine(Shoot());
                lastTimeAttacked = Time.time;
            }
            ...
        }
        private IEnumerator Shoot()
        {
            isAttacking = true;
            State = States.attack;
            yield return new WaitForSeconds(1.2f);
            isAttacking = false;
            GameObject projectile = Instantiate(projectilePrefab, spellPoint.position, Quaternion.identity);
            Vector2 direction = facingRight ? Vector2.right : Vector2.left;
            projectile.GetComponent<Projectile>().Launch(direction);
        }
        ...
    }
```

Скрипт Bat_Controller

```
    ...
    public class Bat_Controller : Entity
    {
        ...
        void Update()
        {
            float distance = Vector2.Distance(transform.position, player.position);
            if (distance < aggressiveRadius && !isAttacking)
            {
                if (distance > attackRadius)
                    transform.position = Vector2.MoveTowards(transform.position, player.position, speed *
Time.deltaTime);
                else if (Time.time - lastTimeAttacked > attackCooldown)
                {
                    StartCoroutine(Shoot());
                    lastTimeAttacked = Time.time;
                }
```

```
        }
        ...
    }
    ...
}
```

Скрипт Ghost_Controller

```
...
public class Ghost_Controller : Entity
{
    ...
    void Update()
    {
        float distance = Vector2.Distance(transform.position, player.position);
        if (distance < aggressiveRadius3)
            speed = 7f;
        else if (distance < aggressiveRadius2)
            speed = 4f;
        else if (distance < aggressiveRadius1)
            speed = 1.5f;
        if (distance < aggressiveRadius1 && !isStopped)
            isAggressive = true;
        else
            isAggressive = false;
        if (isAggressive)
            transform.position = Vector2.MoveTowards(transform.position, player.position, speed * Time.deltaTime);
        ...
    }
    ...
}
```

Скрипт RoomsManager

```
using System.Collections.Generic;
using UnityEngine;
public class RoomsManager : MonoBehaviour
{
    private List<string> level1Rooms, level2Rooms, level3Rooms;
    private void Awake()
    {
        level1Rooms = new List<string> {"Room1", "Room2", "Room3", "Room4", "Room5", "Room6"};
        level2Rooms = new List<string> { "Room7", "Room8", "Room9", "Room10", "Room11", "Room12" };
        level3Rooms = new List<string>{ "Room13","Room14", "Room15", "Room16", "Room17", "Room18" };
    }
    public string GetRandomRoom()
    {
        if(level1Rooms.Count > 2)
        {
            int index = Random.Range(0, level1Rooms.Count);
            string selectedRoom = level1Rooms[index];
            level1Rooms.RemoveAt(index);
            return selectedRoom;
        }
        else if (level2Rooms.Count > 2)
        {
            int index = Random.Range(0, level2Rooms.Count);
            string selectedRoom = level2Rooms[index];
            level2Rooms.RemoveAt(index);
            return selectedRoom;
        }
```

```
        else
        {
            int index = Random.Range(0, level3Rooms.Count);
            string selectedRoom = level3Rooms[index];
            level3Rooms.RemoveAt(index);
            return selectedRoom;
        }
    }
}
```

Скрипт RoomsManager

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class LevelExit : MonoBehaviour
{
    private string nextSceneName;
    private RoomsManager roomManager;
    private RewardManager rewardManager;
    private bool rewardGiven;
    private void Start()
    {
        rewardManager =GameObject.FindWithTag("RewardManager").GetComponent<RewardManager>();
        roomManager = GameObject.FindWithTag("RoomsManager").GetComponent<RoomsManager>();
        nextSceneName = roomManager.GetRandomRoom();
    }
    private void Update()
    {
        if(CanExitRoom() && !rewardGiven)
        {
            rewardGiven = true;
            rewardManager.ShowRewards();
        }
    }
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player") && CanExitRoom())
            SceneManager.LoadScene(nextSceneName);
    }
    private bool CanExitRoom()
    {
        return GameObject.FindGameObjectsWithTag("Enemy").Length == 0;
    }
}
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка гри Shady Adventure з різними видами ворогів

ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТА ГРУПИ 4РП-08: ШВЕЦЯ МАКСИМА СЕРГІЙОВИЧА

КЕРІВНИК: ДЖАБРАІЛОВ Д. В.

Огляд аналогічних проєктів в ігровій індустрії

Скріншот гри «Rogue legacy»

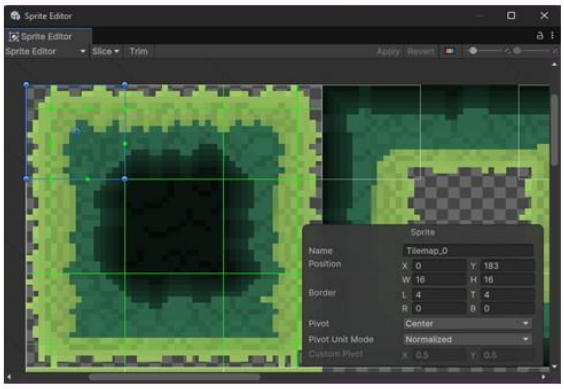
Скріншот гри «Dead cells»

Обрання засобів розробки

Основні ігрові механіки

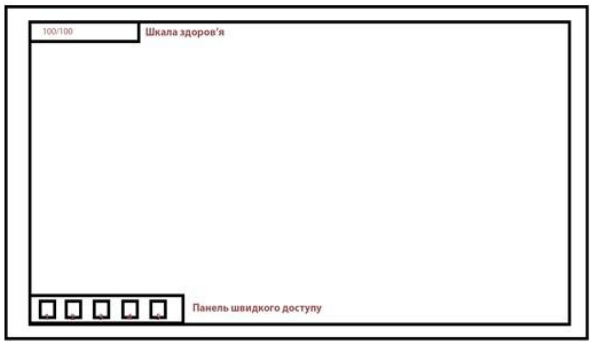
Створення ігрового простору	За допомогою Tilemap та Rule Tile будуть створюватись велика кількість сцен
Різні види ворогів	Для різноманітності геймплею були створенні різні види ворогів
Система предметів	Гра має велику кількість предметів, вони мають 3 види: зброя, заклинання та пасивні здібності
Система інвентаря	Гравець має можливість вибирати предмети, пересувати їх в інвентарі, а також споряджати до 5 заклинань

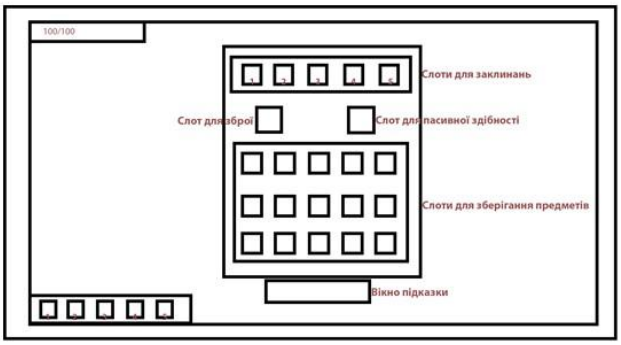
Проєктування та реалізація основних механік гри



Створення власного тайлсету
Використання його в Unity

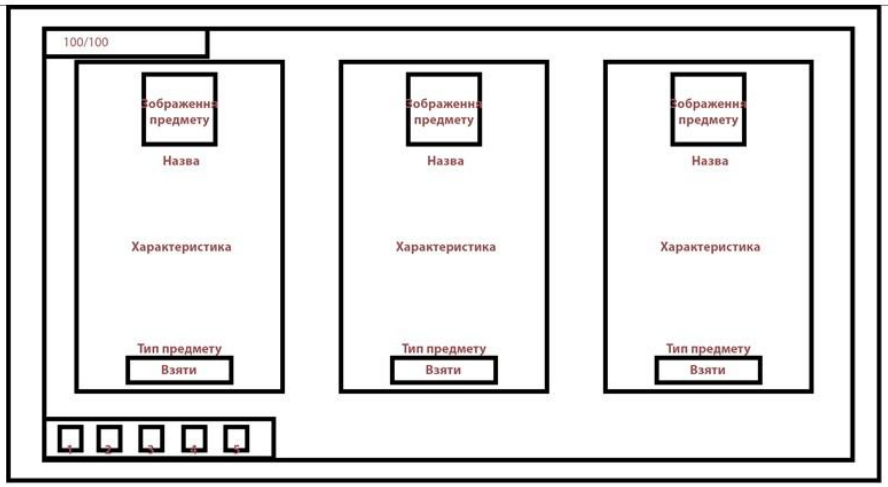
Проєктування та реалізація основних механік гри





Проєктування ігрового інтерфейсу та
інтерфейсу інвентаря

Проєктування та реалізація основних механік гри



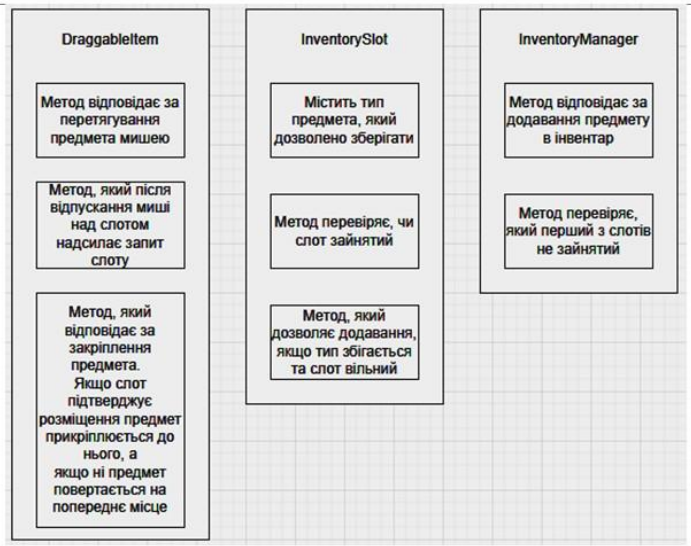
Концепт обрання одного з трьох випадкових предметів

Проєктування та реалізація основних механік гри

Структура класів предметів

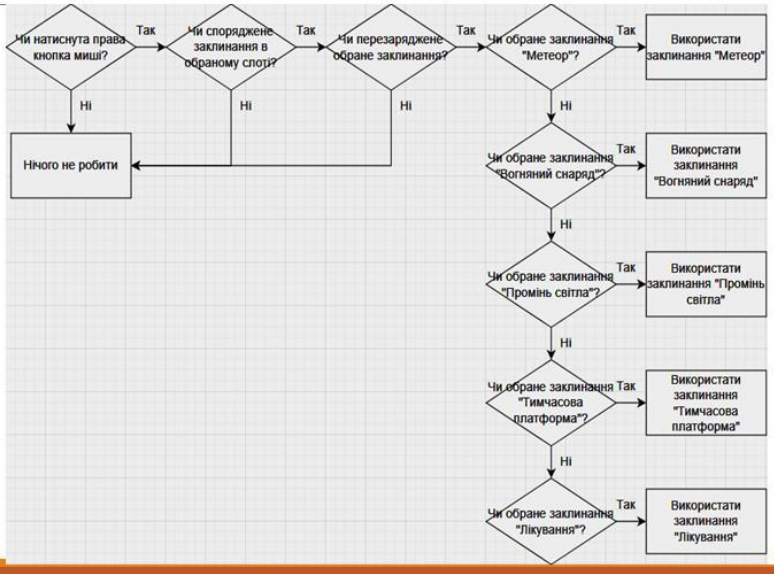
Проєктування та реалізація основних механік гри

Структурна блок-схема логіки інвентаря



Проєктування та реалізація основних механік гри

Логіка використання заклинань гравцем



Проєктування різних видів ворогів

Структура батьківського та дочірніх класів

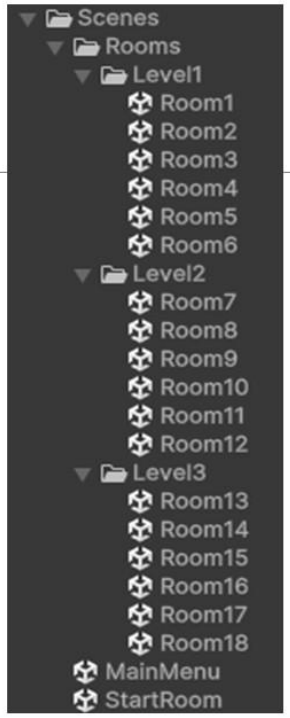
Логіка поведінки одного з видів ворогів – слайма

Реалізація ворогів

Анімація одного з видів ворогів – слайма

Реалізація ігрових рівнів

Список створених сцен, які використовуються в проєкті



Тестування та скріншоти розробленої гри

Тестування отримання нового предмету

Тестування та скріншоти розробленої гри

Тестування заклинань в грі

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Швець Максим Сергійович

(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 Shady Adventure з різними видами ворогів*

Обсяг розрахунково-пояснювальної записки *83* сторінок

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

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення ігрових продуктів в цілому та розробці ігор на ігровому програмному рушії Unity зокрема.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У основному розділі розглянуті питання проблематики розробки гри на ігровому програмному рушії Unity, сформовано концепцію гри згідно до теми дипломного проекту та завданню, виконано проектування основних аспектів розробляємої гри. За допомогою відповідного програмного забезпечення реалізовані всі намічені роботи з ігровим процесом.

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)

Обґрунтовано вибір рушія та жанру гри

Реалізована якісна візуальна складова проекту

Детально описано етапи розробки гри

Продумана організація міжсценових переходів

д) основні недоліки дипломного проекту (роботи)

Не реалізовано звукового супроводу. Через різноманітність інтерфейсних елементів існує ризик появи невідповідностей у взаємодії або труднощів із керуванням переходами між ними. Присутнє дублювання логіки в окремих модулях. Наявні деякі помилки оформлення ПЗ

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:

« *22* » *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проєкт здобувача (здобувачки) освіти

відділення комп'ютерних систем

Швець Максим Сергійович

(прізвище, ім`я та по батькові)

Спеціальність: *121 " Інженерія програмного забезпечення "*

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

Тема дипломного проєкту: *«Розробка гри Shady Adventure з різними видами ворогів»*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЄКТУ

а) обсяг і якість виконання проєкту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проєкт виконано відповідно технічному завданню. Пояснювальна записка містить 83 сторінки. У пояснювальній записці виконано опис етапів розробки гри з різними видами ворогів з елементами жанру платформер, а також питання пов'язані з концептуалізацією та проектуванням проекту. Графічна частина складається з 15 слайдів мультимедійної презентації. Якість виконання пояснювальної записки добра, якість графічної частини відмінна, розробку виконано в повному обсязі.*

б) самостійність роботи над проєктом: *Протягом всього строку дипломного проєктування та переддипломної практики здобувач освіти Швець М. С. 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 Швець М. С. 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 Швець М. С. ознайомився з основами геймдизайну, опанував інструменти середовища Unity та навички програмування на мові C#. Успішно реалізував ігрову логіку, механіки управління персонажем та поведінку різних типів ворогів.

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові керівника дипломного проєкту _____

Джабраілов Д.В.

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 ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Швець М. С.
здобувач освіти гр. 4РП-08, та

Джабраілов Д.В.,
керівник дипломного проекту,

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

«Розробка гри Shady Adventure з різними видами ворогів» (автор роботи – Швець М. С., 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 _____ / Швець М. С. /

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

Швеця Максима Сергійовича

на тему *Розробка гри Shady Adventure з різними видами ворогів*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з некритичними 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 14,47%, коефіцієнт цитування – 1,24%.*

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

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Швеця М.С.* (П.І.Б.)

проведена «*16*» *червня* 20*25* 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.)

Звіт подібності

метадані

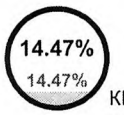
Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка гри Shady Adventure з різними видами ворогів

Автор
Швець Максим Сергійович
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

14620
Кількість слів

111908
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		15
Інтервали		0
Мікропробіли		1
Білі знаки		0
Парафрази (SmartMarks)	a	68

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/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	113 0.77 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	102 0.70 %
3	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	97 0.66 %
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	75 0.51 %
5	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	69 0.47 %

6	https://thepresentation.ru/obzh/m%D1%96krokl%D1%96mat-robochogo-m%D1%96stsya-1	68 0.47 %
7	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	66 0.45 %
8	https://thepresentation.ru/obzh/m%D1%96krokl%D1%96mat-robochogo-m%D1%96stsya-1	65 0.44 %
9	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	64 0.44 %
10	https://ts.kiev.ua/orhanizatsiini-zakhody-shchodo-zabezpechennia-pozhezhnoi-bezpeky/	56 0.38 %

з домашньої бази даних (0.29 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	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")	23 (3) 0.16 %
2	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")	12 (1) 0.08 %
3	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")	7 (1) 0.05 %

з програми обміну базами даних (0.68 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	МР_К23-2м_Послєдов Д.М. [redacted] 1/10/2025 University of Customs and Finance (University of Customs and Finance)	94 (9) 0.64 %
2	Розробка прототипу комп'ютерної 3D гри за допомогою середовища Unity [redacted] 6/2/2021 National University "Zaporizhzhia Polytechnic" (Кафедра "Програмні засоби")	5 (1) 0.03 %

з Інтернету (13.50 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**398 (13) 2.72 %**
2	**https://thepresentation.ru/obzh/m%D1%96krokl%D1%96mat-robochogo-m%D1%96stsya-1**	**370 (11) 2.53 %**
3	**https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download**	**179 (2) 1.22 %**
4	**https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download**	**175 (9) 1.20 %**
5	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content**	**119 (4) 0.81 %**
6	**https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download**	**113 (2) 0.77 %**

7	**https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download**	**85 (5) 0.58 %**
8	**https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download**	**84 (5) 0.57 %**
9	**http://norma.org.ua/document/legislation/law30/4.php**	**82 (5) 0.56 %**
10	**https://ts.kiev.ua/orhanizatsiini-zakhody-shchodo-zabezpechennia-pozhezhnoi-bezpeky/**	**56 (1) 0.38 %**
11	**https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download**	**48 (2) 0.33 %**
12	https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download	48 (3) 0.33 %
13	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content**	**35 (2) 0.24 %**
14	**https://sud.ua/uk/news/ukraine/289701-trebovaniya-proizvodstvennoy-sanitarii-k-rabochemu-mestu-v-gostruda-napomnili-nyuansy**	**32 (1) 0.22 %**
15	**https://dspace.nuft.edu.ua/bitstreams/91176c36-88b7-4a48-9100-5abc2e07bad1/download**	**28 (1) 0.19 %**
16	https://stackoverflow.com/questions/72584011/enemy-cannot-be-damaged-if-not-using-static-variable-for-health	24 (4) 0.16 %
17	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	21 (1) 0.14 %
18	https://ualombards.com.ua/2019/07/16/shchodo_vikonannya_vimog_zakonodavstva_z_pozhezhno_bezpeki/	13 (2) 0.09 %
19	https://dspace.onu.edu.ua/bitstreams/265b8303-36af-4573-99cf-32e9481b262e/download	12 (2) 0.08 %
20	https://essuir.sumdu.edu.ua/bitstream/123456789/88822/1/Yatsenko_bac_rob.pdf	11 (2) 0.08 %
21	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	11 (1) 0.08 %
22	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	10 (1) 0.07 %
23	https://rkwkgmlakd1.tistory.com/	10 (1) 0.07 %
24	https://card-file.ontu.edu.ua/server/api/core/bitstreams/21ac499a-a9e9-4137-810c-5f21a0318048/content	5 (1) 0.03 %
25	https://kvpubd.kiev.ua/wp-content/uploads/2020/06/gr.-15-01.06.2020-urok-virobnichogo-navchannya.pdf	5 (1) 0.03 %

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.08.23.000.ДП

ШВЕЦЯ
МАКСИМА СЕРГІЙОВИЧА

м. Одеса
2025 р.
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ