Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Проектування та розробка тривимірної гри*
(назва кваліфікаційної роботи згідно наказу ОНТУ)
жанру «Slasher». Програмна частина

Здобувача *Бурукова О.В.*
(прізвище, ініціали)
4 курсу *542* групи

Керівники: *ст. викл. Жуковецька С.Л.*
(посада, прізвище та ініціали)
к.т.н.,доц. Шестопалов С.В.
(посада, прізвище та ініціали)

Консультанти: _____
(посада, прізвище та ініціали)
д.е.н., проф. Басюркіна Н.Й.
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *10.06* 2023 р., протокол № 8
Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса - 2023 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Розробка ігор та інтерактивних медіа у віртуальній реальності*

ЗАТВЕРДЖУЮ
Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« 10 » *серпня* 2022 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Бурукова Олександра Васильовича

1. Тема роботи *Проектування та розробка тривимірної гри жанру «Slasher». Програмна частина*

Затверджена наказом університету від « *10* » *серпня* 2022 р., наказ № *440-03*

2 Термін здачі здобувачем закінченої роботи *5 червня 2023 р.*

3. Вихідні дані роботи

1. Редактор «Microsoft Office». 2. Середовище розробки «Visual Studio 2019». 3. Ігровий двигун «Unity 3D». 4. Ассети для Unity.

4. Перелік питань, які потрібно розробити

1. Аналіз предметної області. 2. Розгляд існуючих аналогів. 3. Представлення проектної документації. 4. Розробка програмної частини гри. 5. Розрахунок економічних показників. 6. Розгляд охорони праці. 8. Загальні висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайд 1. Мета, об'єкт, предмет, новизна. Слайд 2. Актуальність. Слайд 3. Особливості жанру. Слайд 4. Аналоги. Слайд 5. Ключові особливості гри (USP). Слайд 6. Опис гри. Слайд 7. Штучний інтелект. Слайд 8. Прототипування гри. Слайд 9. Економічні розрахунки. Слайд 10. Загальні висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Басюркіна Н.Й., д.е.н., проф.*		
Охорона праці	*Жуковецька С.Л., ст. викл.*		
Нормоконтроль	*Жуковецька С.Л., ст. викл.*		

7. Дата видачі завдання *30.09.2022*

Керівники _____ *Світлана ЖУКОВЕЦЬКА*

_____ *Сергій ШЕСТОПАЛОВ*

Завдання прийняв до виконання _____ *Олександр БУРУКОВ*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз предметної області*	*10.11.2022*	
2.	*Аналіз існуючих аналогів*	*20.11.2022*	
3.	*Розробка концептуального документу*	*01.01.2023*	
4.	*Розробка дизайнерського документу*	*15.01.2023*	
5.	*Розробка програмної частини*	*10.02.2023*	
6.	*Розробка економічної частини*	*10.04.2023*	
7.	*Розробка розділу охорони праці*	*25.04.2023*	
8.	*Оформлення пояснювальної записки*	*15.05.2023*	
9.	*Підготовка графічного матеріалу.*	*25.05.2023*	

Керівники роботи _____ *Світлана ЖУКОВЕЦЬКА*

_____ *Сергій ШЕСТОПАЛОВ*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач - дипломник _____ *Олександр БУРУКОВ*

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці програмної частини гри жанру «*Slasher*». Жанр «*Slasher*» відомий своєю акцентованістю на бойових сценах та динамічних поєдинках, тому правильна реалізація програмної частини є ключовою для створення задоволення гравця від ігрового процесу. Також, слешери вимагають точності та швидкості реакції, тому програмна частина має забезпечити плавну і відчутну взаємодію гравця з персонажем.

В першому розділі розглянуті особливості розробки ігор жанру «*Slasher»*, проведено аналіз існуючих аналогів.

В другому розділі сформоване технічне завдання, розроблений пайплайн роботи. Розроблено концептуальний та дизайнерський документ з точки зору програмної частини.

У третьому розділі обгрунтовано вибір засобів та описано процес розробки програмної частини гри.

У четвертому розділі проведена оцінка ефективності розробки гри жанру «*Slasher*», описано маркетинговий, науково-технічний, економічний, соціальний та екологічний ефект від розробки проекту. У п'ятому розділі розглянуто питання охорони праці.

Результатом роботи є демонстраційна версія гри в двигуні *Unity*, в якій релізовано переміщення персонажа, та битву з босом, що володіє штучним інтелектом.

Ключові слова: двигун *Unity,* бойова система, штучний інтелект, механіки.

ABSTRACT

The qualification work is devoted to the development of the program part of the game "Slasher". The "Slasher" genre is known for its emphasis on combat scenes and dynamic fights, so the correct implementation of the software part is key to creating player satisfaction from the gameplay. Also, slashers require precision and speed of reaction, so the software part should provide a smooth and tangible interaction between the player and the character.

The first section discusses the peculiarities of developing games of the "Slasher" genre, analyzes existing analogues.

In the second section, the technical task is formed and the work plan is developed. A conceptual and design document in terms of the program part was developed.

The third section substantiates the choice of tools and describes the process of developing the software part of the game.

The fourth section evaluates the effectiveness of the development of the game genre "Slasher", describes the marketing, scientific and technical, economic, social and environmental effects of the project. The fifth section deals with labor protection issues.

The result of the work is a demo version of the game in the Unity engine, which features character movement and a battle with a boss with artificial intelligence.

***Keywords**: Unity engine, combat system, artificial intelligence, mechanics.*

ЗМІСТ

стор.

ВСТУП

Розробка комп'ютерних ігор є однією з найпопулярніших та швидкозростаючих галузей сучасної розважальної індустрії. Ігрова індустрія постійно розвивається та вдосконалюється, пропонуючи гравцям нові захоплюючі виміри і враження. Одним із популярних жанрів комп'ютерних ігор є «*Slasher*», що відрізняється динамічним та екшен-орієнтованим геймплеєм.

Жанр «*Slasher*» відомий своєю високою динамікою, напруженими бойовими сценами та епічними поєдинками. У процесі проектування та розробки тривимірної гри жанру «*Slasher*» програмна частина відіграє вирішальну роль у створенні незабутнього геймплею та ігрового досвіду. Комп'ютерні ігри цього жанру надають гравцям можливість відчути себе непереможними бійцями, що виконують захоплюючі комбо-удари та здійснюють швидкі реакції на ворожі атаки.

Проектування програмної частини ігор жанру «*Slasher*» включає в себе розробку складної системи керування, реалізацію різноманітних рухів та анімацій персонажів, а також створення інтенсивних бойових механік та інтерактивного середовища. Команда розробників зосереджена на досягненні високої якості графічного оформлення, реалістичного звукового супроводу та плавного геймплею, що забезпечує гравцям максимально іммерсивне враження від гри.

Метою кваліфікаційного проекту є проектування та розробка тривимірної гри жанру «Slasher».

Об’єкт кваліфікаційної роботи: процес проектування і розробки програмної частини гри жанру «*Slasher*».

Предмет кваліфікаційної роботи: методи проектування і розробки програмної частини гри жанру «*Slasher*».

Завдання на кваліфікаційну роботу:

1. Дослідження жанру «*Slasher*», його особливості бойової системи та програмування. Дослідження існуючих аналогів.
2. Постановка задачі, та розробка дизайнерського документа, який включає технічний опис концепції та функціональної специфікації.
3. Розробка фізичної взаємодії об'єктів у грі.
4. Розробка бойової системи у грі.
5. Розробка штучного інтелекту ворогів у грі.
6. Об'єднання всіх частини в готові префаби.

Новизна кваліфікаційної роботи: використання патерну *State Machine* для керування об'єктами через їхні стани, розробка системи зброї та використання методики *CapsuleFloat* для контроля знаходження на поверхні.

Публікації за темою кваліфікаційної роботи:

1. Буруков О.В., Жуковецька С.Л. Характерні механіки комп'ютерних ігор жанру «*Slasher*». Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації / Матеріали II Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 29-30 вересня 2022 р. С. 104-105.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні жанри комп’ютерних ігор

Комп'ютерні ігри представляють собою важливу сферу творчої діяльності, яка заснована на програмному мистецтві і спрямована на створення віртуальних світів для взаємодії з користувачами.

Кожен жанр комп'ютерних ігор має свої характерні особливості та неповторний ігровий досвід, який він пропонує. Ці жанри формують різноманітну палітру ігрових світів, у які гравці можуть зануритися і отримати задоволення від різних видів віртуальної взаємодії. Від швидкої та інтенсивної екшен-гри до глибоких дослідницьких пригод, від тактичних стратегій до захопливих рольових ігор – кожен жанр запрошує гравців до унікальної подорожі, де вони можуть проявити свої навички, розвинути стратегічне мислення або просто насолодитися емоційною історією, сплетеною навколо персонажів і подій ігрового світу. Жанри ігор уособлюють нескінченні можливості та надихають гравців на дослідження, розвиток і взаємодію в захопливому віртуальному просторі. Ось деякі з найпоширеніших жанрів:

- екшен – жанр, у якому гра швидка, інтенсивна і наповнена захопливими сутичками, захопливою стрілянино̆ю та фізичною активністю;
- пригодницькі ігри – ці ігри фокусуються на розв'язанні головоломок, дослідженні світу, інтеракції з персонажами та розвитку сюжету;
- рольові ігри (*RPG*) – у *RPG* іграх гравці приймають роль вигаданого персонажа і керують його розвитком, прокачуванням навичок і виконують завдання у фантастичному або казковому світі;
- стратегії – ці ігри вимагають від гравців розроблення та реалізації стратегій, щоб досягти певних цілей, як–от управління містом, армією або цивілізацією;

– симулятори – симулятори дають змогу гравцям імітувати реальні або вигадані ситуації, як–от управління містом, авіасимулятори, управління фермою та інші;
– головоломки – цей жанр фокусується на розв'язанні головоломок, логічних завдань і головоломок, які вимагають від гравця аналітичного мислення і творчого підходу;
– шутери – жанр, у якому основна увага приділяється стрілянині та боям із використанням вогнепальної зброї;
– спортивні ігри – ігри, що моделюють різні види спорту, які дають змогу гравцям змагатися один з одним або проти комп'ютерних супротивників.

Це лише деякі приклади жанрів комп'ютерних ігор, і в реальності існує безліч комбінацій і варіацій, а також розвиток нових жанрів з плином часу. Кожен жанр пропонує свій унікальний ігровий досвід і може приваблювати різні аудиторії гравців.

1.2 Процес розробки ігор

Розробка комп'ютерних ігор – це творчий і складний процес, який вимагає поєднання майстерності, інновацій та технічної експертизи. Вона є мистецтвом, що втілює в собі уяву та віртуальні світи, здатні занурити гравців в унікальні пригоди та емоційні переживання. Особливістю розробки ігор є необхідність балансування між естетикою, геймплеєм, технічними можливостями та потребами аудиторії. Кожна гра являє собою унікальний світ, створений командою розробників з метою захоплення, розваги та залучення гравців. Вона вимагає уваги до деталей, ретельного планування і тестування, а також постійного прагнення до інновацій та вдосконалення. Розробка комп'ютерних ігор – це процес, який змішує технології та мистецтво, щоб створити магічні та захопливі віртуальні світи, здатні захопити уяву і стати джерелом радості та вражень для мільйонів гравців по всьому світу.

Основні етапи розробки комп'ютерних ігор включають:

1. Натхнення і концепція. На цьому етапі зароджуються ідеї та народжуються концепції, натхненні уявою розробників. Тут формуються основні характеристики та концептуальна основа гри.

2. Проектування і розробка. Цей етап включає в себе розробку геймплея, рівнів, персонажів і сюжетної лінії. Розробники створюють дизайн гри, визначають її механіку і створюють прототипи.

3. Мистецтво і дизайн. Тут створюється візуальний стиль гри, включно з графікою, анімацією, звуковим оформленням і музикою. Розробники прагнуть до створення привабливих і захопливих візуальних та аудіоефектів.

4. Програмування та розробка: Цей етап включає в себе програмування ігрової логіки, штучного інтелекту, управління персонажами та інших технічних аспектів гри. Тут відбувається створення програмного коду і тестування гри на різних платформах.

5. Тестування та налагодження: Важливий етап, на якому гра проходить через інтенсивне тестування, щоб виявити та виправити помилки, поліпшити ігровий баланс і забезпечити гладке функціонування.

6. Реліз і маркетинг: Після завершення розробки гра готується до релізу, включно зі створенням маркетингових матеріалів, складанням прес–релізів і плануванням стратегії просування.

7. Підтримка та оновлення: Після релізу розробники продовжують підтримувати гру, випускаючи оновлення та виправляючи помилки, взаємодіючи зі спільнотою гравців.

Однією з ключових особливостей розробки комп'ютерних ігор є використання ігрових рушіїв (*game engines*). Ігровий рушій – це програмне забезпечення, яке надає розробникам інструменти та функціональність для створення і запуску ігор. Він слугує основою, на якій будується весь ігровий процес, від графіки та анімації до фізики та звуку.

Безліч кампаній і студій розробляють свої власні ігрові рушії, щоб мати повний контроль над функціональністю і можливостями своїх ігор. Такі внутрішні рушії зазвичай налаштовуються та оптимізуються під конкретні проєкти, даючи змогу розробникам реалізувати свої унікальні ідеї та концепції.

Однак існують і популярні ігрові рушії, які широко використовуються в індустрії та доступні для використання комерційними та незалежними розробниками. Деякі з них включають *Unity*, *Unreal Engine*, *CryEngine* і *Godot*.

Unity – потужний і гнучкий ігровий рушій, який підтримує розробку ігор для різних платформ, включно з комп'ютерами, консолями та мобільними пристроями. Він має широкий набір інструментів для створення графіки, анімації, фізики та ігрової логіки. *Unity* також надає можливості для створення багатокористувацьких і віртуальної реальності (*VR*) ігор.

Unreal Engine – потужний і візуально вражаючий ігровий рушій, розроблений компанією *Epic Games*. Він пропонує широкий спектр інструментів для створення високоякісних графічних ефектів, реалістичної фізики та складної ігрової механіки. *Unreal Engine* також підтримує багатокористувацькі проєкти та розробку ігор у віртуальній і доповненій реальності.

CryEngine – ще один потужний ігровий рушій, відомий своєю вражаючою графікою та фотореалістичними можливостями. Він надає розробникам засоби для створення красивих оточень, динамічного освітлення та деталізованих моделей персонажів.

Godot – безкоштовний і відкритий ігровий рушій з активною спільнотою розробників. Він забезпечує інтуїтивний інтерфейс і зручні інструменти для створення *2D* і *3D* ігор. *Godot* також підтримує безліч платформ і є доступним вибором для розробників-початківців.

Використання ігрових рушіїв спрощує і прискорює процес розробки ігор, забезпечуючи готові інструменти та функції, а також підтримку різних платформ. Вони являють собою силу за спиною розробників, допомагаючи їм перетворити свої ідеї на захопливі ігрові світи.

1.3 Особливості жанру «*Slasher*»

1.3.1 Поняття жанру «*Slasher*»

Жанр ігор «*Slasher*» являє собою захопливу і динамічну категорію, яка приваблює безліч гравців своєю інтенсивністю і вибуховими битвами. Слешери, також відомі як ігри з бійками і боями, акцентують увагу на умілих рухах, швидкій реакції і нещадних сутичках. Ці ігри пропонують гравцям можливість стати великими воїнами, які майстерно володіють зброєю та навичками бою.

Однією з головних особливостей слешерів є їхній динамічний і проривний геймплей. Замість повільних і стратегічних боїв, слешери пропонують швидкі й ефектні атаки, комбо та спеціальні прийоми, які дають змогу гравцеві потужно розмахувати зброєю і наносити вражаючі удари. Це створює відчуття сили і переваги, а також надає грі ритм і динаміку.

Візуальна привабливість також відіграє важливу роль у слешерах. Вони часто пропонують барвисті та стильні світи, виконані в яскравих кольорах і з вражаючою графікою. Бойові сцени оформлені із застосуванням спеціальних ефектів, що створюють відчуття епічної битви і захопливих візуальних моментів.

Невід'ємною частиною слешерів є система прокачування і розвитку персонажа. Гравці мають можливість покращувати своїх героїв, відкривати нові прийоми, атаки та здібності, що дає змогу створювати унікальні стилі гри та адаптуватися до різних ситуацій. Це сприяє глибині та варіативності геймплею, даючи змогу кожному гравцеві налаштувати свого персонажа під свої вподобання.

Сюжетні елементи також відіграють важливу роль у слешерах. Хоча основний фокус зазвичай зосереджений на боях і сутичках, деякі ігри пропонують цікаві та захопливі історії, які доповнюють дію і надають грі емоційної глибини.

Слешери є жанром, у якому гравці можуть зануритися в інтенсивні бої, випробувати епічні моменти та відчути себе справжніми героями. Їхній динамічний геймплей, візуальна пишність і системи прокачування роблять слешери захопливими і дають змогу кожному гравцеві створити свою унікальну історію у світі бійок і битв.

1.3.2 Бойова система в жанрі «*Slasher*»

Однією з ключових особливостей слешерів є їхня глибока бойова система, що вимагає від гравця майстерності та точності. У них гравці можуть використовувати різноманітні атаки, комбо і спеціальні прийоми, щоб руйнувати ворогів з неймовірною силою і стилем. Парування також відіграє важливу роль у слешерах, даючи змогу гравцям блокувати та відбивати атаки супротивників з точним часом і навичками. Це створює відчуття глибини і контролю в битвах, де кожен рух має значення і може визначити результат битви.

Атака в слешерах є ключовим елементом геймплея. Гравці можуть використовувати різні види зброї, від мечів і клинків до магічних артефактів, щоб наносити удари противникам. Бойова система зазвичай різноманітна і дає змогу комбінувати різні атаки та прийоми для створення приголомшливих комбінацій і ефектів. Це дає змогу гравцям експериментувати, розвивати свій стиль бою і знаходити унікальні стратегії для подолання складних суперників.

Штучний інтелект ворогів є також важливим аспектом слешерів. Вороги володіють різними тактиками та стилями бою, що створює виклик і вимагає від гравців адаптації та вміння читати їхні дії. Вони можуть використовувати удари, блокування та уникати атак, щоб створити реалістичні та захопливі сутички. Деякі слешери також пропонують босів, які є особливими супротивниками з унікальними атаками та обороною, представляючи особливе випробування для гравців.

З огляду на вище сказане, у слешерах, де важлива миттєва реакція і відсутність затримок в управлінні, плавність і комфортність ігрового контролю мають вирішальне значення. Ідеальним прикладом виняткового управління слугує серія ігор *Ninja Gaiden*, яка є однією з найкращих у жанрі слешерів. У цих іграх бойова механіка відточена до досконалості, і управління відіграє важливу роль у цьому процесі. Герой бездоганно реагує на команди гравця, виконуючи кожну дію точно і негайно. Анімація в іграх серії *Ninja Gaiden* є плавною, детальною і приємною для ока. Такий самий підхід до керування можна спостерігати в іграх студії *Platinum Games*, зокрема в *Bayonetta* та *Metal Gear Rising*. Герої цих ігор також бездоганно підкоряються вказівкам гравця, миттєво переходячи від комбінацій до ухилень і блокування, створюючи фізичну й естетичну насолоду в боях. Керування в цих іграх є грамотно виконаним елементом, що дає змогу гравцеві інтуїтивно виконувати складні комбінації.

Metal Gear Rising, наприклад, завдяки своїй невпинній природі, надає плавний перехід від одного прийому до іншого, забезпечуючи неймовірну динаміку в боях. Незважаючи на різноманітність і швидкість дій у цих іграх, правильне освоєння механіки та зосередженість на екрані нагороджують гравця захопливим, адреналіновим видовищем. Жанр слешерів із високою динамікою вимагає уваги та зосередженості, але водночас пропонує неперевершене задоволення від швидких і барвистих битв.

1.4 Особливості програмування в жанрі «*Slasher*»

Програмування ігор – це мистецтво, в якому програмісти творять віртуальні світи, наповнюючи їх захопливими пригодами. Особлива увага приділяється жанру слешерів, де акцент робиться на епічних боях і динамічній бойовій системі. Розробка слешерів вимагає особливих навичок і креативного підходу до програмування, щоб створити унікальні та захопливі ігрові механіки, здатні занурити гравця в захопливий світ битв.

1.4.1 Мистецтво програмування

У світі програмування слешерів розробники стикаються з непростими викликами та завданнями, що вимагають особливої творчості та майстерності. Один із головних аспектів, на якому зосереджується увага, це створення переконливої та плавної бойової системи, де гравці можуть повністю зануритися в ігровий світ, вільно взаємодіючи з навколишнім середовищем і супротивниками. У цьому разі необхідно ретельно моделювати фізику, анімації та механіку атак.

Наприклад, розробляючи бойову систему в грі-слешері, програмісти повинні прагнути до створення реалістичних фізичних моделей, які враховують вплив сили, швидкості та маси на рух персонажів і об'єктів у грі. Правильно налаштована фізика дає змогу створити відчуття ваги та сили в ударах і атаках, роблячи бойові дії більш правдоподібними та задовільними для гравців.

Крім того, розробники повинні також звертати увагу на створення плавних і реалістичних анімацій, які відповідають діям персонажів. Добре анімовані атаки та рухи надають грі відчуття жвавості та додають візуальної привабливості. Наприклад, під час розроблення анімації ударів мечем, програмісти мають врахувати деталі, такі як траєкторія руху меча, реакція противника на удар та анімацію переходу між різними атаками.

Також важливою частиною розробки слешерів є робота над механіками атак. Розробники мають створювати різноманітні комбінації атак і рухів, щоб гравці мали свободу у виборі своєї бойової стратегії. Наприклад, у грі можна передбачити можливість комбінування різних атакувальних прийомів і блокувань, що дасть змогу гравцям створювати унікальні комбінації та підходи до боїв. Слід врахувати, що балансування механік атак і їхня сполучуваність є невід'ємною частиною розробки, щоб надати гравцям цікаві та різноманітні можливості в бою.

1.4.2 Бойова динаміка та управління

Важливою особливістю слешерів є їхня динамічна бойова система, яка підкреслює швидкість, реакцію та координацію гравця. Програмісти повинні приділяти особливу увагу розробці інтуїтивного управління, яке дає змогу гравцям вільно комбінувати атаки, блокувати та парирувати ворогів, а також використовувати спеціальні прийоми та вміння.

Наприклад, під час проєктування управління для гри-слешера, розробники повинні прагнути до створення системи, яка максимально відповідає інтуїтивним очікуванням гравців. Це може включати в себе використання поєднань кнопок або жестів, щоб гравці могли виконувати різні атаки та дії на основі своїх намірів. Грамотне керування дає змогу гравцям зосередитися на самій грі, мінімізуючи складність і тимчасові витрати на освоєння контролів.

Безперервне вдосконалення та оптимізація управління є ключовими аспектами розробки бойової системи в слешерах. Розробники повинні враховувати відгуки та реакції гравців, щоб зробити управління ще більш чуйним і зручним. Наприклад, якщо багато гравців зазнають труднощів під час виконання певних комбінацій атак, розробники можуть проаналізувати та переглянути управління, щоб спростити їхнє виконання або запропонувати альтернативні варіанти.

1.4.3 Штучний інтелект ворогів

У світі слешерів гравці стикаються з різноманітними ворогами, кожен з яких має свій унікальний стиль бою і тактику. Завдання програмістів полягає у створенні інтелектуальної системи, яка дає змогу ворогам реагувати на дії гравця, адаптуватися до його стратегії та надавати реальний виклик. Адже штучний інтелект ворогів має бути налаштований таким чином, щоб забезпечити захопливі бої та зберігати баланс між складністю та доступністю гри.

Наприклад, розробляючи штучний інтелект для ворогів у слешерах, програмісти повинні врахувати безліч чинників. Вороги повинні аналізувати дії гравця, передбачати його наміри і приймати відповідні рішення в бою. Це може включати в себе зміну своєї тактики, вибір оптимальних атак або блокувань, і навіть співпрацю між ворогами для підвищення складності бою.

Під час розробки штучного інтелекту важливо знайти баланс між суперниками, щоб гравцям було цікаво битися з ними, незважаючи на їхню складність. Занадто прості вороги можуть зробити гру нудною і не викликати відчуття прогресу, а надто складні можуть стати перешкодою для гравців, відлякати їх і викликати розчарування. Таким чином, тонке налаштування та балансування штучного інтелекту ворогів є невід'ємною частиною розробки слешерів, щоб створити захопливі та динамічні бої.

1.5 Існуючи аналоги

1.5.1 *NieR: Automata*

NieR: Automata – це унікальна гра жанру слешер, розроблена студією *PlatinumGames* і видана *Square Enix* у 2017 році. Вона пропонує захопливу і глибоку історію, переплетену з інтенсивним і динамічним геймплеєм. У грі гравцеві належить узяти на себе роль андроїдних бойових юнітів, які б'ються з машинами в постапокаліптичному світі. Графіка й атмосфера гри чудово передають атмосферу жаху та відчаю, створюючи неповторну атмосферу.

Бойова геймплейна складова *NieR: Automata* вирізняється високою інтенсивністю і динамікою, що робить її ідеальним прикладом слешера (рис. 1.1)[1]. Гравцеві доступні різноманітні комбо-атаки , які можна здійснювати за допомогою зброї ближнього бою, а також використовувати стрілецьку зброю для атак на відстані. Протягом боїв гравцеві доведеться проявляти рефлекси і вміння ухилятися від атак супротивників, щоб максимально ефективно впоратися з ними. Крім того, у грі передбачені боси, які становлять особливий

виклик і вимагають особливої стратегії та навичок (рис. 1.2)[1]. Бої в *NieR: Automata* пропонують насичений і захопливий досвід, який дає змогу гравцеві повністю зануритися в інтенсивну дію та насолодитися епічними сутичками.

Рис. 1.1 – Скріншот ігрового процесу гри *NieR: Automata*

Рис. 1.2 – Скріншот боса гри *NieR: Automata*

1.5.2 *Trek to Yomi*

Trek to Yomi – це захоплива гра жанру слешер, розроблена студією *Flying Wild Hog* і видана *Devolver Digital*. Гра пропонує захопливу історію про самурая, який вступає в сутичку зі своїми ворогами, щоб захистити свою рідну країну. Особливістю *Trek to Yomi* є його унікальний арт-стиль, який відтворює

атмосферу класичних фільмів про самураїв, створюючи неперевершену візуальну привабливість.

У бойовій геймплейній складовій *Trek to Yomi* демонструється велика увага до деталей, що характерно для жанру слешер. Гравцеві дозволяється виконувати складні комбінації атак за допомогою меча, використовувати блокування для захисту від ворожих атак і вчасно ухилятися, щоб уникнути смертельних ударів (рис. 1.3)[2]. Крім того, гра пропонує різноманітні ворожі типи, кожен з яких вимагає від гравця особливих підходів і стратегій. Завдяки плавному керуванню та реалістичній фізиці руху гравця, бої в *Trek to Yomi* стають захопливими та емоційно насиченими, даруючи неперевершені враження від боротьби самурая за своє майбутнє.

Рис. 1.3 – Скріншот ігрового процесу гри *Trek to Yomi*

1.5.3 *Thymesia*

Thymesia – захоплива гра жанру *Souls-like*, розроблена студією *OverBorder* і видана *Team17*. Гра розповідає історію про хороброго героя, який змушений вийти на бій із хворобою, що поширюється фантастичним світом. У *Thymesia* графіка темна й атмосферна, створюючи містичну і похмуру ауру, що допомагає заглибитися у вигаданий всесвіт.

У бойовій геймплейній складовій *Thymesia* використовуються елементи, характерні для жанру слешер. Гравець може виконувати швидкі та потужні

атаки своєю зброєю, використовувати блокування для захисту та розумно маневрувати, щоб уникнути ворожих атак (рис. 1.4)[3]. Однак, *Thymesia* також пропонує унікальну механіку, пов'язану з хворобою, яка впливає на геймплей. Гравцеві доводиться використовувати інфекцію, щоб отримувати певні переваги в бою, однак це супроводжується ризиками. Бої в *Thymesia* є тактичними та напруженими, де кожен крок має значення, а правильне використання навичок та реакція на ворожі дії є ключем до успіху.

Рис. 1.4 – Скріншот ігрового процесу гри *Thymesia*

1.5.4 *Devil May Cry 5*

Devil May Cry 5 - захоплююча гра жанру слешер, розроблена *Capcom*. Гра є п'ятою частиною в популярній серії *Devil May Cry*, яка славиться своїм швидким темпом, стильним бойовим мистецтвом і яскравими персонажами. У *Devil May Cry 5* гравець бере на себе роль вогненного мисливця за іменем Данте, який має знищити демонічну загрозу і врятувати світ від знищення.

Боєвий геймплей *Devil May Cry 5* є ключовим аспектом гри. Вона пропонує широкий арсенал зброї і навичок, які гравець може використовувати для вражаючих комбінацій та стильних атак (рис. 1.5)[4]. Головний акцент робиться

на швидкості, плавності рухів і точності управління, що дозволяє гравцям виконувати різноманітні комбо і розвивати свої бойові навички. У грі також присутній система рейтингів, яка оцінює стильність і ефективність бою гравця, надаючи стимул до досягнення високих результатів (рис. 1.6)[4].

Загалом, *Devil May Cry 5* є відмінним прикладом гри жанру слешер, де бойовий геймплей є серцевиною досвіду. Його екшн-насичений світ, багатий на вражаючі атаки та елементи стилю, забезпечує гравцям незабутній бойовий досвід, що доповнюється захоплюючою історією і запаморочливою графікою.

Рис. 1.5 – Скріншот ігрового процесу гри *Devil May Cry 5*

Рис. 1.6 – Скріншот системи рейтінга гри *Devil May Cry 5*

Висновок до першого розділу

В розділі розглянуті основні жанри комп’ютерних ігор, та особливості їх розробки. Також розглянуті особливості бойової системи та програмування жанру слешер. Визначено, що для жанру слешер бойова механіка дуже важлива частина, та має бути швидкою і динамічною.

РОЗДІЛ 2
ПРОЕКТУВАННЯ

2.1 Постановка завдання

Кваліфікаційна робота є частиною одного комплексного проекту, метою якого є проектування та розробка тривимірної гри жанру «*Slasher*». Дана кваліфікаційна робота охоплює програмну частину проекту.

Об'єктом розробки є тривимірна гра жанру «*Slasher*». Ця гра створюється для широкої аудиторії геймерів, яка зацікавлена в захопливому та динамічному ігровому досвіді.

Основною метою розробки є створення захопливого геймплею, в якому гравці зможуть відчути себе справжніми воїнами, здатними виконувати ефектні комбо-удари та випробувати свою швидкість і реакцію. Гра пропонуватиме напружені бойові сцени, які поглинуть гравців у захопливий світ різноманітних битв.

Метою даної кваліфікаційної роботи є створення програмної частини тривимірної гри жанру «*Slasher*».

Основні вимоги:

1. Загальні вимоги. Має бути створена програмна частина тривимірної гри жанру «*Slasher*».
2. Вимоги до результату. Результатом роботи має бути розроблений функціонал гри властивий жанру «*Slasher*».
3. Вимоги до функціоналу. Розробити фізичну взаємодію об'єктів, бойову систему, штучний інтелект ворогів, побудувати ігровий рівень та керування персонажем.
4. Вимоги до використаних технологій. Розробка гри має здійснюватися в ігровому рушії *Unity*. Реалізація ігрової логіки та функціоналу здійснюється мовою програмування високого рівня *C#* у середовищі розробки *Visual Studio*.

2.2 Концепція

2.2.1 Основні особливості гри

Ключові особливості гри (*USP*):

1. Винищення духів. Гра пропонує захопливи й динамічні бої проти духів, їх можна знищувати за допомогою ефектних комбо-ударів та різноманітних бойових прийомів. Духів можна вбити завдяки спеціальному виду зброї і різних спеціальних умінь;

2. Унікальні міфічні боси. Кожен бос вимагає до себе індивідуальний підхід і вибір тактики. У міру дослідження гри можна зустріти підказки стосовно слабкості босів;

3. Безліч загадок і несподіваних подій у світі гри, що вимагають уважності та швидкої реакції, протягом усього проходження.

2.2.2 Опис гри

Бій у грі відбувається таким чином. Гравець атакує ворога клинком, використовуючи різні атаки та навички. Гравець може парирувати деякі ворожі атаки, що дає велику перевагу при правильному використанні. Так само в грі є ухилення, що дають кадри невразливості на початку застосування. Вороги можуть атакувати зброєю ближнього і дальнього бою, а також особливими навичками, завдаючи різного виду шкоди. Шкода може бути фізичною, від простих атак, а може бути періодичною, як отрута. Так само є особливі атаки, що накладають різні негативні ефекти.

Під час взаємодії гравця зі святинею, він отримує характеристику «Стимул», маючи яку, при отриманні смертельної шкоди персонаж відновлює частину хп і отримує бонус сили, а значення стимулу опускається до нуля.

На локаціях можна натрапити на різні загадки, вирішивши які гравець отримує додаткову інформацію, або особливі предмети. Також можна зустріти спеціальні події, які вимагають від гравця в будь-який момент бути до них готовим, за їхнє виконання дається здебільшого валютна нагорода.

2.2.3 Порівняння та передумови створення

Хоча гра «*Awakenin*» містить оригінальні рішення у своєму жанрі, вона також успішно впроваджує найкращі особливості обраних прикладів у свою концепцію:

– *NieR: Automata* – схожа бойова механіка, і система зброї;
– *Elden Ring* – схожа механіка ухилення і парирування. Схожий ШІ ворогів.

2.2.4 Системні вимоги

Гра орієнтована на середню конфігурацію 6 річної давності.

Таблиця 2.1

Системні вимоги

Вимоги	Мінімальні	Рекомендовані
Операційна система	*Windows 7/10*	
Процесор	*Intel Core i3 2100*	*Intel Core i5 4670*
ОЗП	*2 GB*	*4 GB*
Вільне місце на *HDD*	*10 GB*	*10 GB*
Відеокарта	*NVIDIA GeForce GTX 770*	*NVIDIA GeForce GTX 980*
Звукова карта	*DirectX11 supported*	
Управління	Миша, клавіатура	

2.3 Функціональна специфікація

2.3.1 Життєва та бойова модель

Кожна істота має певний рівень здоров'я ($HEALTH_{stat}$), яке може змінюватися під впливом різних джерел, поточний стан здоров'я ($HEALTH_{curr}$)

відображається в інтерфейсі гравця. Істота володіє рівнем фізичного захисту ($PHYSDEFENSE_{stat}$) і зброєю, у якої є базова шкода ($BASEDAMAGE_{stat}$) та множник шкоди різних типів атак ($ATCKTYPEDMGMULT_{stat}$). Так само існують предмети, що дають відновлення певної кількості поточного здоров'я ($HEAL_{stat}$) та поступову регенерацію ($REGEN_{stat}$).

2.3.2 Формули

Зміна рівня $HEALTH_{curr}$ від фізичних атак:

$$HEALTH_{curr} = HEALTH_{curr} - BASEDAMAGE_{stat} * ATCKTYPEDMGMULT_{stat} * (100 / (100 + PHYSDEFENSE_{stat})) \quad (2.1)$$

Зміна рівня $HEALTH_{curr}$ від відновлення здоров'я:

$$HEALTH_{curr} = HEALTH_{curr} + HEAL_{stat} \quad (2.2)$$

Зміна рівня $HEALTH_{curr}$ від регенерації здоров'я протягом часу T:

$$HEALTH_{curr} = HEALTH_{curr} + REGEN_{stat} * T \quad (2.3)$$

2.3.3 Головний герой

Таблиця 2.2

Характеристики головного героя

Параметр	Значення
Здоров'я ($HEALTH_{stat}$)	100
Фізичний захист ($PHYSDEFENSE_{stat}$)	40

Таблиця 2.3

Управління головним героєм

Клавіша	Функція
ЛКМ	Звичайна атака
ПКМ	Блок
ЛКМ + *Shift*	Посилена атака
ПКМ + *Shift*	Парування
W/A/S/D / + *Shift*	Біг / Спринт
Space	Ривок
R	Використати предмет
F	Взаємодія

Стани перебування персонажа (рис. 2.1).

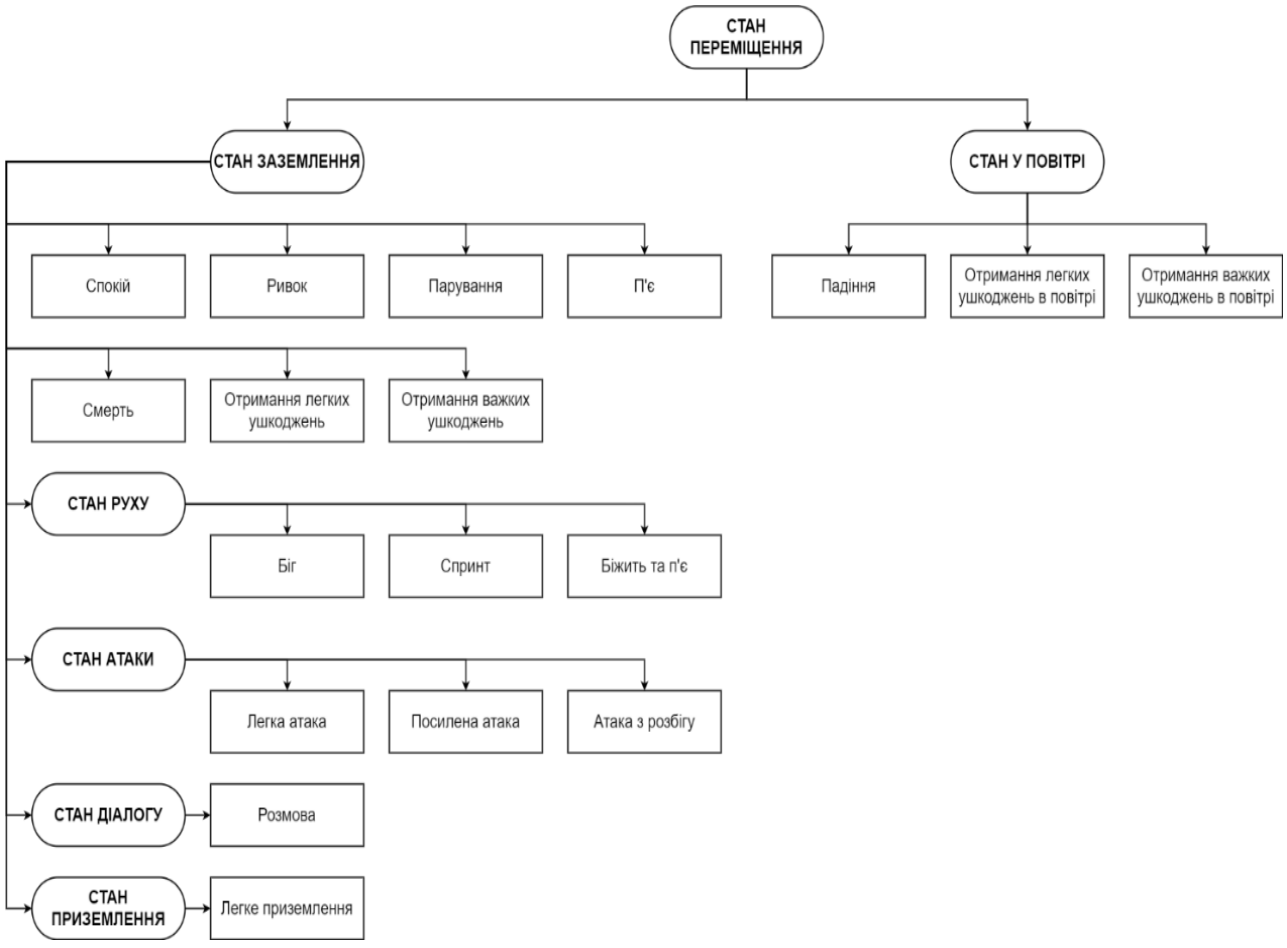


Рис. 2.1 – Стани перебування персонажа

2.3.4 Нікчемний дух

Таблиця 2.4

Характеристики істоти

Параметр	Значення
Здоров'я ($HEALTH_{stat}$)	30
Фізичний захист ($PHYSDEFENSE_{stat}$)	0
Базова шкода ($BASEDAMAGE_{stat}$)	5
Легка атака ($ATCKTYPEDMGMULT_{stat}$)	1

Істота може перебувати у наступних станах:

- спокій;
- біг;
- легка атака;
- отримання легких пошкоджень;
- отримання важких пошкоджень;
- відпарирован;
- падіння;
- легке приземлення;
- смерть.

2.3.5 Зрілий дух

Таблиця 2.5

Характеристики істоти

Параметр	Значення
Здоров'я ($HEALTH_{stat}$)	80
Фізичний захист ($PHYSDEFENSE_{stat}$)	20
Базова шкода ($BASEDAMAGE_{stat}$)	10
Легка атака ($ATCKTYPEDMGMULT_{stat}$)	1
Атака корінням ($ATCKTYPEDMGMULT_{stat}$)	0.8

Істота може перебувати у наступних станах:

- спокій;
- біг;
- легка атака;
- атака корінням;
- отримання легких пошкоджень;
- отримання важких пошкоджень;
- відпарирован;
- падіння;
- легке приземлення;
- смерть.

2.3.6 Дух лісу

Таблиця 2.6

Характеристики істоти

Параметр	Значення
Здоров'я ($HEALTH_{stat}$)	200
Фізичний захист ($PHYSDEFENSE_{stat}$)	100
Базова шкода ($BASEDAMAGE_{stat}$)	15
Комбо атака 1 ($ATCKTYPEDMGMULT_{stat}$)	1
Комбо атака 2 ($ATCKTYPEDMGMULT_{stat}$)	1
Посилена атака ($ATCKTYPEDMGMULT_{stat}$)	2
Висхідна атака ($ATCKTYPEDMGMULT_{stat}$)	1.5
Атака з розгону ($ATCKTYPEDMGMULT_{stat}$)	4
Масова атака ($ATCKTYPEDMGMULT_{stat}$)	2

Істота може перебувати у наступних станах:

- спокій;
- біг;
- спринт;
- бойовий клич;
- комбо атака 1;
- комбо атака 2;
- посилена атака;
- висхідна атака;
- атака з розгону;
- масована атака;
- отримання легких пошкоджень;
- отримання важких пошкоджень;
- відпарирован;
- падіння;
- легке приземлення;
- смерть.

2.3.7 Стародавній дух Природи

Таблиця 2.7

Характеристики істоти

Параметр	Значення
Здоров'я ($HEALTH_{stat}$)	500
Фізичний захист ($PHYSDEFENSE_{stat}$)	0
Базова шкода ($BASEDAMAGE_{stat}$)	20
Вистріл шипами ($ATCKTYPEDMGMULT_{stat}$)	0.5
Комбо атака 1 ($ATCKTYPEDMGMULT_{stat}$)	1

Продовження таблиці 2.7

Параметр	Значення
Комбо атака 2 (*ATCKTYPEDMGMULT*$_{stat}$)	1
Посилена атака (*ATCKTYPEDMGMULT*$_{stat}$)	2
Атака з стрибка (*ATCKTYPEDMGMULT*$_{stat}$)	3
Атака захопленням (*ATCKTYPEDMGMULT*$_{stat}$)	4
Масова атака корінням (*ATCKTYPEDMGMULT*$_{stat}$)	2

Істота може перебувати у наступних станах:

- спокій;
- біг;
- стрибок;
- бойовий клич;
- вистріл шипами;
- комбо атака 1;
- комбо атака 2;
- посилена атака;
- атака з стрибка;
- атака захопленням;
- масована атака корінням;
- отримання легких пошкоджень;
- отримання важких пошкоджень;
- відпарирован;
- падіння;
- легке приземлення;
- смерть.

2.3.8 Катана

Таблиця 2.8

Характеристики зброї

Параметр	Значення
Базова шкода (*BASEDAMAGE*$_{stat}$)	10
Легка атака 1 (*ATCKTYPEDMGMULT*$_{stat}$)	1
Легка атака 2 (*ATCKTYPEDMGMULT*$_{stat}$)	1.5
Легка атака 3 (*ATCKTYPEDMGMULT*$_{stat}$)	1.5
Посилена атака (*ATCKTYPEDMGMULT*$_{stat}$)	2
Атака з розгону (*ATCKTYPEDMGMULT*$_{stat}$)	1.5

2.3.9 Шпилька

Гравець може надіти одну зі шпильок, яка дає пасивні бонуси.

Таблиця 2.9

Характеристики зброї

Назва	Характеристики
Зелена шпилька	Дає гравцю пасивний ефект збільшення стійкості, а саме збільшує рівень захисту (*PHYSDEFENSE*$_{stat}$) на 40%.
Синя шпилька	Дає гравцю пасивний ефект підвищення сил, а саме збільшує базову шкоду (*BASEDAMAGE*$_{stat}$) на 20%.
Червона шпилька	Дає гравцю пасивну регенерацію здоров'я (*HEALTH*$_{curr}$), а саме 0.2 здоров'я /сек.
Золота шпилька	Дає гравцю усі пасівні здібності інших шпильок, а саме збільшення рівеня захисту на 20%, збільшення базової шкоди на 10% та пасивну регенерацію у значенні 0.1 здоров'я /сек.

2.3.10 Зілля

Гравець може використати обране зілля, яке дає тимчасові значні бонуси.

Таблиця 2.10

Характеристики зброї

Назва	Характеристики
Зілля зеленого кольору	Дає гравцю тимчасовий бонус збільшення стійкості, а саме збільшує рівень захисту ($PHYSDEFENSE_{stat}$) на 150% протягом 120 секунд.
Зілля синього кольору	Дає гравцю тимчасовий бонус підвищення сили, а саме збільшує базову шкоду ($BASEDAMAGE_{stat}$) на 50% протягом 120 секунд.
Зілля червоного кольору	Моментально відновлює гравцеві певну кількість здоров'я ($HEALTH_{curr}$), а саме 25% від максимального здоров'я ($HEALTH_{stat}$).
Зілля жовтого кольору	Дає гравцеві моментальне повне знезараження всіх негативних ефектів.

2.4 Штучний інтелект

Штучний інтелект (ШІ) в слешерах відіграє важливу роль у створенні захоплюючих та динамічних бойових сценаріїв. Його фокус полягає в тому, як реалістично ворог виконує свої дії, враховуючи умови, в яких він знаходиться. В демонстраційній версії гри, штучний інтелект був реалізований для керування поведінкою ворожих персонажів.

Реалізація ШІ ворога повинна передбачати:

– патрулювання території;

– виявлення противника;

– погоня за противником з огляду на перешкоди на шляху;

– битва;

– відступ у разі втрати противника з поля зору.

Ворог спочатку перебуває в не бойовому стані, патрулюючи місцевість, або стоячи на місці. Потім при виявленні гравця, залежно від відстані і перешкод між ними, підходить до гравця, або починає атакувати. Вибір атак в основному ґрунтується на дистанції між ворогом і гравцем, перезарядці атак і здібностей, кількості енергії, що залишилася для використання атак і навичок, певних діях гравця (використання зілля, закінчення стану ривка тощо). Втративши ворога з поля зору або занадто відсторонившись від базової точки, повертається назад.

Finite state machine діаграма ШІ (рис. 2.2).

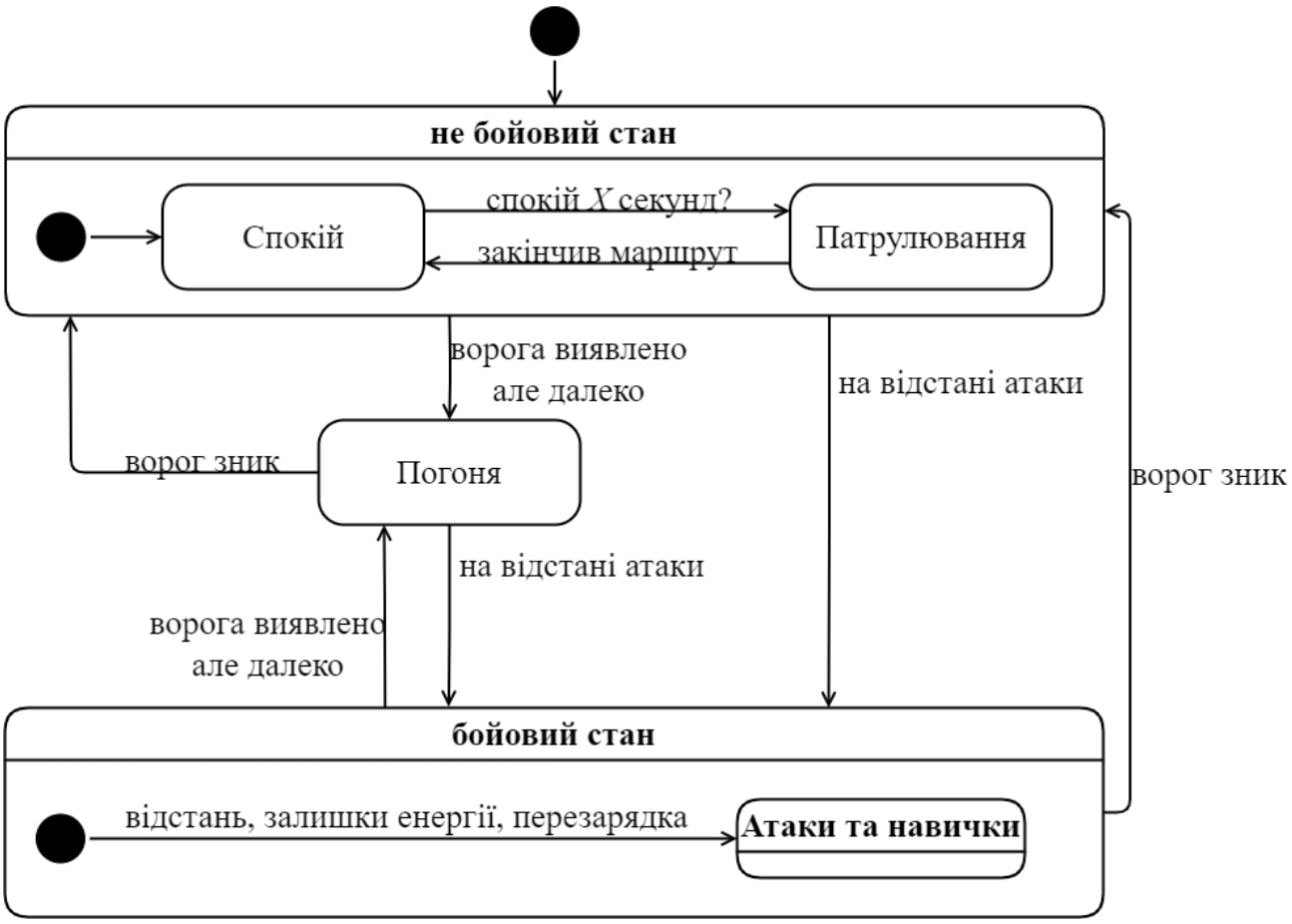


Рис. 2.2 – *Finite state machine* діаграма ШІ

Висновок до другого розділу

В розділі сформовано завдання, та розроблене технічне завдання. Розроблена проектна документація гри з точки зору програмної частини. Розглянуті концепція та функціональна специфікація.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ ГРИ

3.1 Вибір засобів розробки

Для розроблення програмної частини гри було ухвалено рішення використовувати ігровий рушій *Unity 2021.3.21f1*. *Unity* – середовище розроблення комп'ютерних ігор, є кросплатформеним, що дає змогу створювати додатки, які працюють на великій кількості різних платформ. В *Unity* об'єднані різні програмні засоби, які використовуються при створенні ПЗ – текстовий редактор, компілятор, налагоджувач і тд. Розрахунки фізики в *Unity* виконує фізичний рушій *PhysX* від *NVIDIA* для *3D* фізики. Графічний *API – DirectX*.

Так само великою перевагою *Unity* є наявність величезної бібліотеки ассетів і плагінів. У проєкті використовуються деякі додаткові плагіни, а саме:

1. *ProBuilder*: створення, редагування та текстурування власної геометрії в *Unity*. Використовується для прототипування рівнів і різних тестів взаємодії з оточенням.
2. *Post Processing*: набір ефектів і фільтрів зображень, які можна застосовувати до камер для поліпшення візуального оформлення ігор.

Мовою програмування для написання скриптів було обрано *C#*. *C#* – це об'єктно-орієнтована мова програмування, розроблена компанією *Microsoft*. Вона є однією з основних мов програмування, що використовуються для розроблення ігор і додатків на платформі *Unity*.

C# має безліч переваг для розроблення ігор, таких як інтуїтивний синтаксис, автоматичне керування пам'яттю, широкий набір бібліотек та інструментів, а також підтримка багатопотоковості й асинхронного програмування. Він також має хорошу продуктивність.

Як зовнішній редактор сценаріїв було обрано середовище написання коду *Microsoft Visual Studio*. Воно легко інтегрується в редактор *Unity*, і є продуктом однієї компанії з мовою *C#*, тому є основним інтегрованим середовищем розробки для неї.

3.2 Початок розробки

3.2.1 Імпортування ассетів

На першому етапі необхідно завантажити готові ассети, надані з попередньої роботи. З них були взяті модель персонажа (рис. 3.1) та необхідні анімації руху, атак, падіння, приземлення, спокою, ривка, парирування, отримання шкоди та смерті.

Рис. 3.1 – Модель головного персонажа

Так само була взята модель боса (рис. 3.2) та необхідні анімації руху, атак, отримання шкоди та смерті.

Рис. 3.2 – Модель боса

Отримані три моделі формату *FBX*, та необхідні текстурні карти для них. Щоб імпортувати в сцену модель, необхідно налаштувати її, створити необхідні матеріали, та встановити їх у модель (рис. 3.3). Матеріалам необхідно назначити текстурі карти.

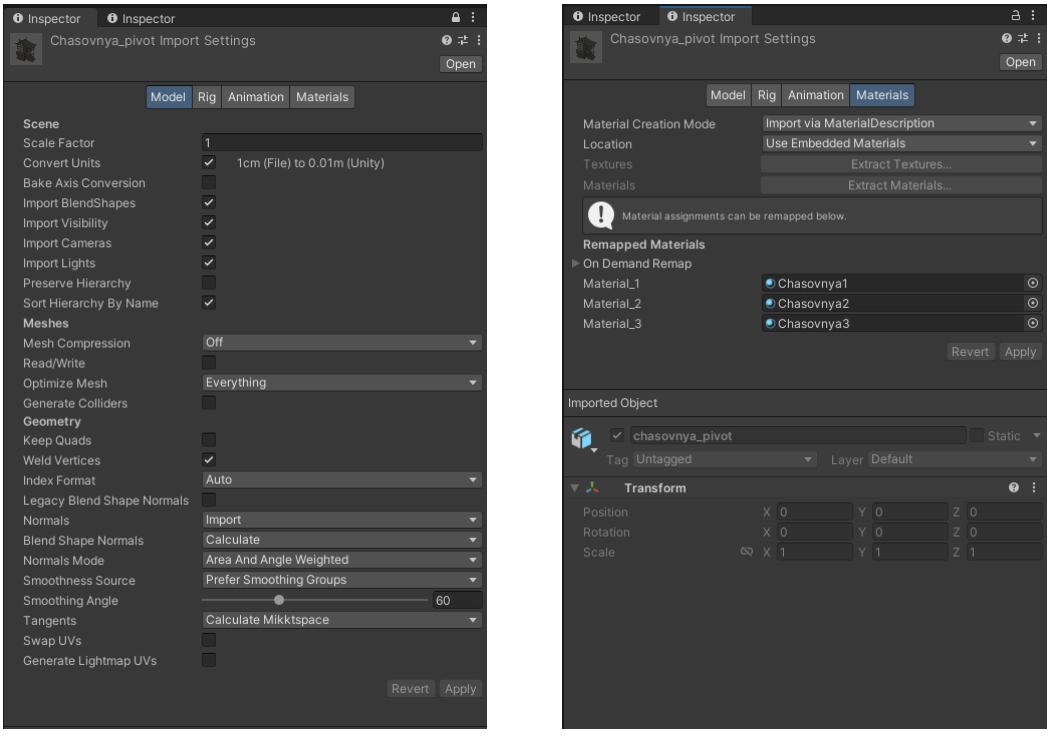


Рис. 3.3 – Налаштування моделі каплиці

3.2.2 Створення головної сцени

Створюється нова сцена з назвою *Lvl1*. Імпортовані моделі виставляються на необхідні місця:

– катана прив’язується до правої руки персонажа (рис. 3.4,а);

– щит прив’язується до лівої руки персонажа (рис. 3.4,б);

– каплиця буде знаходитися в середині сцени.

а) б)

Рис. 3.4 – Зброя персонажа: а) – катана; б) – щит

Далі створюється ліс. Для створення ландшафту використовується *Terrain.* Для підлоги террейну була використана тайлова текстура лісної землі, а для елементів оточення, знайдених серед ассетів онлайн-бібліотеки *Unity store*, зроблено пензлі для їхнього розташування (рис. 3.5).

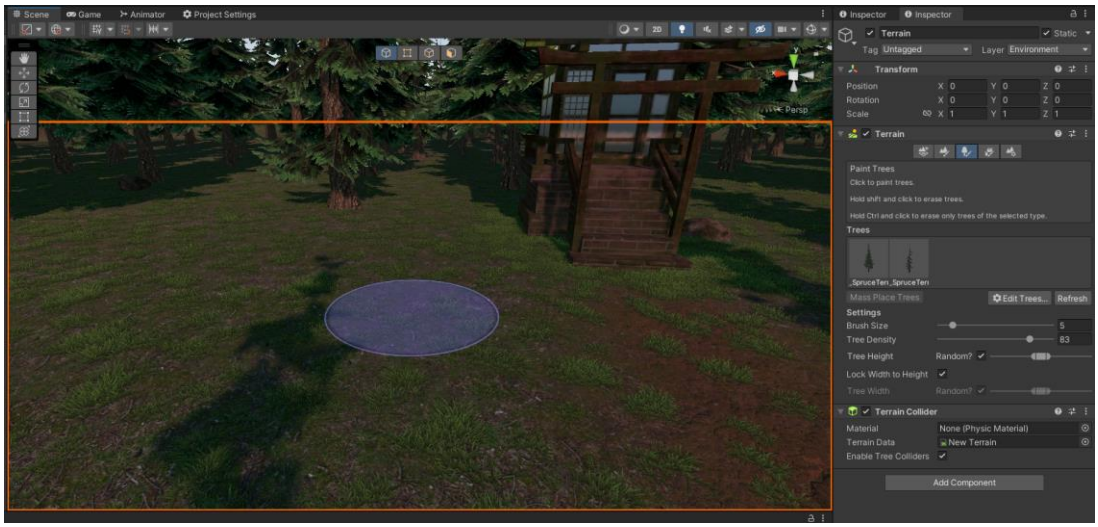


Рис. 3.5 – Головна сцена гри

3.2.3 Фізичні властивості об’єктів

На другому етапі прототипування гри, необхідно надати ігровим об'єктам фізичні властивості. Для початку необхідно надати всім об'єктам оточення, які вважаються для істот поверхнею, колайдер. Так само бажано заздалегідь усім об'єктам поверхні встановити певний *Layer*, для полегшення управління і взаємодії між ними.

Після встановлення компонента *Box Collider* на об'єкти оточення, необхідно додати компонент *Capsule Collider* усім істотам, а також головний для них компонент *Rigidbody*, який дасть змогу рушію працювати з об'єктом, як із фізичною одиницею (рис. 3.6).

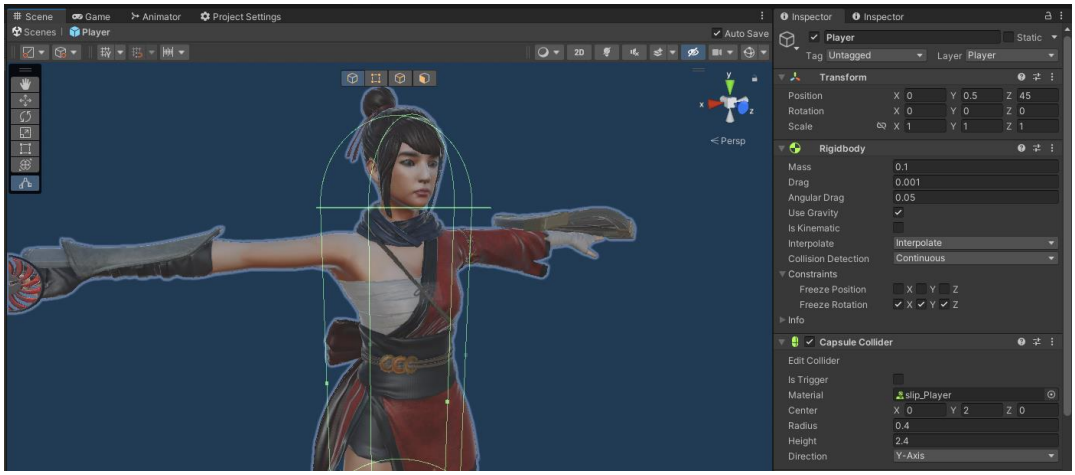


Рис. 3.6 – Компоненти фізичної взаємодії гравця

При простому використанні колайдера в істот виникає низка недоліків у їхньому пересуванні та взаємодії з оточенням. Для розв'язання цих проблем використовується методика *CapsuleFloat*. Її суть полягає у наступному: капсульний колайдер встановлюють не на всю висоту істоти, залишаючи невелику відстань між нижньою точкою колайдера і поверхнею. Щоб запобігти провалюванню істоти під текстури, до цієї відстані постійно застосовується сила, яка піднімає або опускає істоту залежно від відстані до поверхні. Таким чином, завдяки точно обчисленої силі, істота завжди залишається на поверхні, немов у неї є повноцінний колайдер (рис. 3.7).

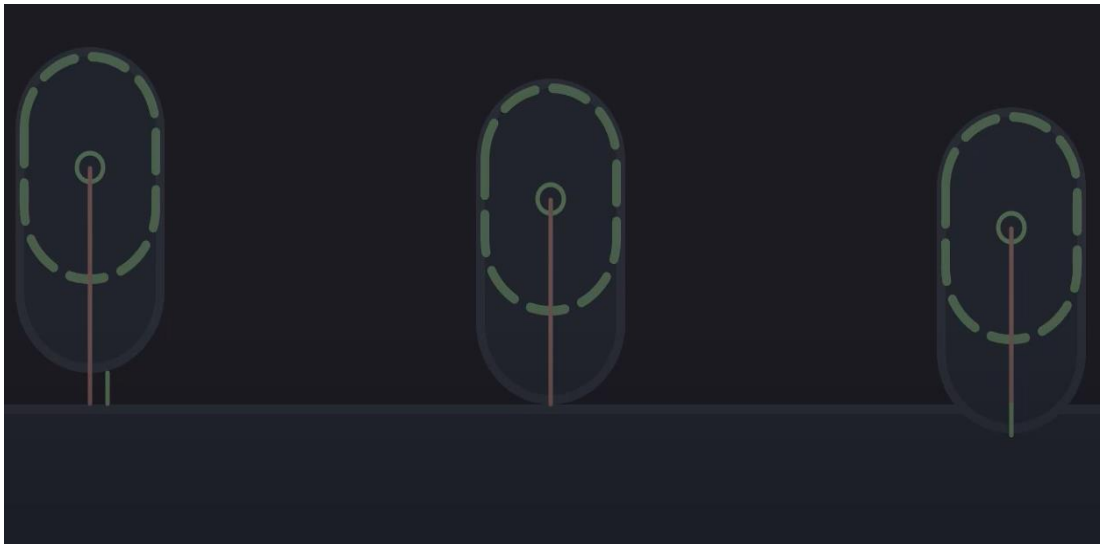


Рис. 3.7 – Схематична демонстрація методики *CapsuleFloat*

Код для налаштування капсульного колайдера (рис. 3.8), класа *Collider Utility*, та його компонентів, наданий у додатку А.

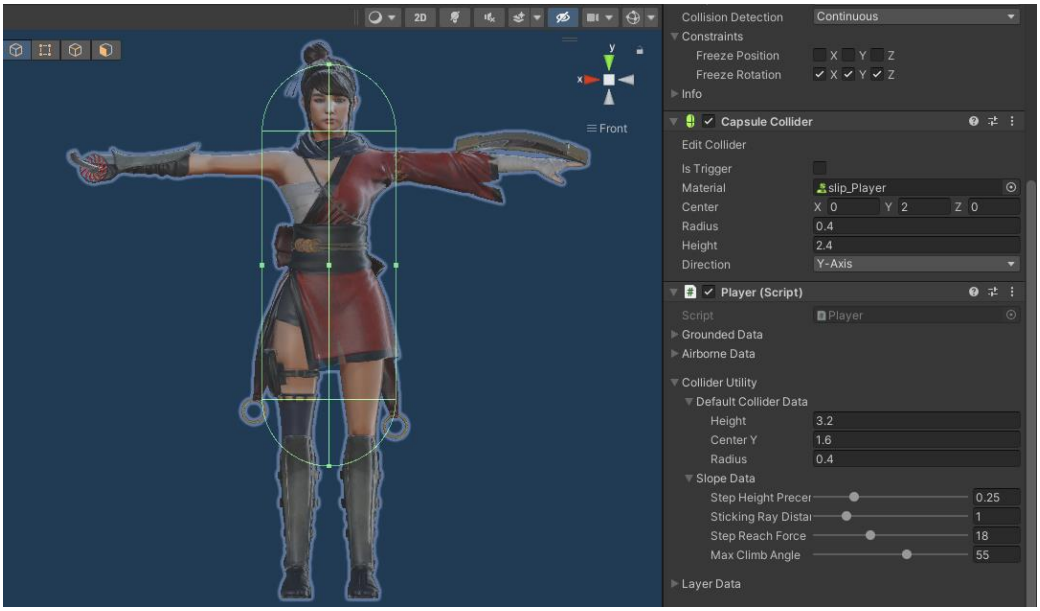


Рис. 3.8 – *CapsuleFloat* система гравця

Такі маніпуляції з колайдером дають змогу досягти таких результатів:

– об'єкт може вільно переміщатися різними поверхнями, не застрягаючи на маленьких перешкодах, адже він просто підніматиметься ними, немов плаваючи по хвилях;
– можливість підніматися по сходах, які не перевищують задану висоту;

– під час швидкого руху в гору або спуску по сходах, об'єкт не відривається від поверхні, якщо відстань до неї не перевищує зазначеного значення.

Ще однією проблемою є можливість підняття на поверхню з кутом нахилу до 90 градусів. Розв'язання цієї проблеми досить просте: ми кидаємо промінь, за допомогою методу *RayCast*, суворо вниз, і під час потрапляння на поверхню, визначаємо кут нахилу цієї поверхні. Якщо кут нахилу перевищує допустиме значення для проходження, вимикається ручне керування об'єктом і він відштовхується у зворотний бік, немов ковзає.

Таким чином, у разі виявлення неприпустимого кута нахилу поверхні, об'єкту автоматично забороняється підніматися і він відштовхується назад, забезпечуючи природну поведінку і запобігаючи неправильному переміщенню на крутих поверхнях.

Код методу *CapsuleFloat*:

```
Vector3 capsuleColliderCenterInWorldSpace =
    stateMachine.player.ColliderUtility.CapsuleColliderData.
    Collider.bounds.center;
Ray downwardsRayFromCapsuleCenter = new
    Ray(capsuleColliderCenterInWorldSpace,
    Vector3.down);
if (Physics.SphereCast(downwardsRayFromCapsuleCenter,
    stateMachine.player.ColliderUtility.DefaultColliderData.Radius -
    0.05f, out RaycastHit hit, GetFloatRayDistance(),
    stateMachine.player.LayerData.GroundLayer,
    QueryTriggerInteraction.Ignore))
{
  float groundAngle = Vector3.Angle(hit.normal,
    -downwardsRayFromCapsuleCenter.direction);
  stateMachine.reusableData.GroundAngel = groundAngle;
  float slopeAngle =
      Vector3.Angle(stateMachine.player.transform.forward.normalized,
      hit.normal);
  if (slopeAngle > 90)
  {
    stateMachine.reusableData.SlopeSpeedModifier =
        stateMachine.player.GroundedData.SlopeSpeedAngle.Evaluate(groun
        dAngle);
  }
  else
  {
    stateMachine.reusableData.SlopeSpeedModifier = 1 + (1 -
        stateMachine.player.GroundedData.SlopeSpeedAngle.Evaluate(groun
        dAngle));
```

```
    }
    float distanceToFloatingPoint =
        stateMachine.player.ColliderUtility.CapsuleColliderData.
        ColliderCenterInLocalSpace.y *
        stateMachine.player.transform.localScale.y - hit.distance + 0.05f
        - stateMachine.player.ColliderUtility.
        DefaultColliderData.Radius;
    if (distanceToFloatingPoint == 0)
    {
      return;
    }
    float amountToLift = distanceToFloatingPoint *
        stateMachine.player.ColliderUtility.SlopeData.StepReachForce -
        GetPlayerVerticalVelocity().y;
    stateMachine.player.rb.AddForce(new Vector3(0f, amountToLift, 0f),
        ForceMode.VelocityChange);
    if (groundAngle >=
        stateMachine.player.ColliderUtility.SlopeData.MaxClimbAngle)
    {
      if (stateMachine.reusableData.ShouldSlip)
      {
        stateMachine.reusableData.PlayerMovingControl = false;
        Vector3 slideForce = stateMachine.player.GroundedData.Slip_speed
            * (Vector3.ProjectOnPlane(hit.normal,
            Vector3.down).normalized + new Vector3(0, -0.7f, 0));
        stateMachine.player.rb.AddForce(slideForce –
             GetPlayerHorizontalVelocity(), ForceMode.VelocityChange);
        stateMachine.reusableData.AfterSlip = true;
      }
      else
      {
        stateMachine.player.StartCoroutine(ShouldSlipTrue());
      }
    }
    else
    {
      stateMachine.player.StopCoroutine(ShouldSlipTrue());
      if (stateMachine.reusableData.AfterSlip)
      {
        stateMachine.reusableData.AfterSlip = false;
        stateMachine.player.StartCoroutine(ShouldSlipFalse());
      }
    }
  }
```

Рух істот відбувається завдяки фізиці *Unity*. Рух поверхнею здійснюється шляхом застосування імпульсу до твердого тіла. При цьому від сили імпульсу віднімається поточне прискорення, що забезпечує плавний і рівномірний рух об'єкта, не змінюючи його прискорення безпосередньо.

У разі руху в повітрі, сила імпульсу розраховується з використанням рекурентної формули, що призводить до поступового зменшення швидкості

об'єкта в геометричній прогресії. У разі використання цього підходу, швидкість руху об'єкта в повітрі поступово зменшується з кожним наступним імпульсом.

Слід врахувати, що всі фізичні операції в коді виконуються в методі *FixedUpdate*, який автоматично викликається *Unity* з постійною частотою. Використання методу *FixedUpdate* забезпечує точність і стабільність фізичних взаємодій у грі. Він гарантує, що фізичні розрахунки будуть проводитися з постійною і регулярною частотою, що важливо для досягнення передбачуваності та послідовності фізичної поведінки об'єктів.

3.3 Машина станів

На наступному етапі розробки гри, реалізується патерн проектування *State Machine* для станів основного персонажа, та ворогів. Цей патерн являє собою організацію об’єкта в різні стани відповідно до певних умов. В основі патерну лежить така концепція:

1. Машина станів складається з фіксованого набору станів, у яких може перебувати об’єкт. Наприклад, це може бути стан бігу, атаки та інші. Кожен стан представляє певну поведінку або дію персонажа.

2. Об’єкт може перебувати тільки в одному стані одночасно. Це означає, що об’єкт, наприклад, не може одночасно бігати і стояти.

3. Кожен стан має набір переходів, які пов'язуються з певними подіями. Коли відбувається певна подія, машина станів переходить із поточного стану в новий стан відповідно до заданих переходів. Це дає змогу об’єкту реагувати на зміни в навколишньому середовищі або взаємодію з іншими об'єктами в грі.

Є інтерфейс стану (*IState*), який визначає набір полів та методів, присутніх у кожному стані. У загальному випадку можна виділити три основні аспекти для кожного класу стану (рис. 3.9):

– вхід (*Enter*): це момент, коли сутність переходить у стан і виконує одноразові дії, необхідні при вході в цей стан;

– вихід (*Exit*): це момент, коли сутність переходить у стан і виконує одноразові дії, необхідні при виході з цього стану;
– цикл оновлення (*Update*, *LateUpdate*, *PhysicsUpdate*): у цьому методі міститься основна логіка оновлення, яку можна розбити на цикл оновлення фізики, цикл оновлення в кожному кадрі, і цикл оновлення після кадру.

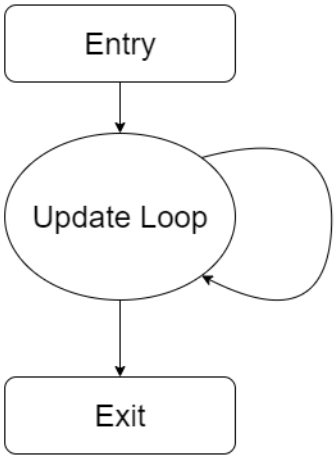


Рис. 3.9 – Алгоритм поведінки стану

Для управління станами, створен абстрактний клас машини станів (*StateMachine*), який визначає набір полів та методів для перемикання станів. Цей клас є основою для створення гнучких машин станів, які можуть бути адаптовані під різні потреби.

```
public abstract class StateMachine
{
    protected IState currentState;

    public void ChangeState(IState newState)
    {
        currentState?.Exit();

        currentState = newState;

        currentState.Enter();
    }

    public void Update()
    {
        currentState?.Update();
    }
```

```
    public void LateUpdate()
    {
        currentState?.LateUpdate();
    }

    public void PhysicsUpdate()
    {
        currentState?.PhysicsUpdate();
    }
}
```

Стани реалізуються ієрархічно, при цьому групи кінцевих станів, що мають схожі властивості, успадковуються від батьківських надстанів. Наприклад, стани бігу й атаки виникають тільки в тому разі, коли персонаж перебуває на поверхні, тому вони матимуть спільний надстан заземлення.

На рисунку 3.10 представлено діаграму класів головного персонажа та його машини станів.

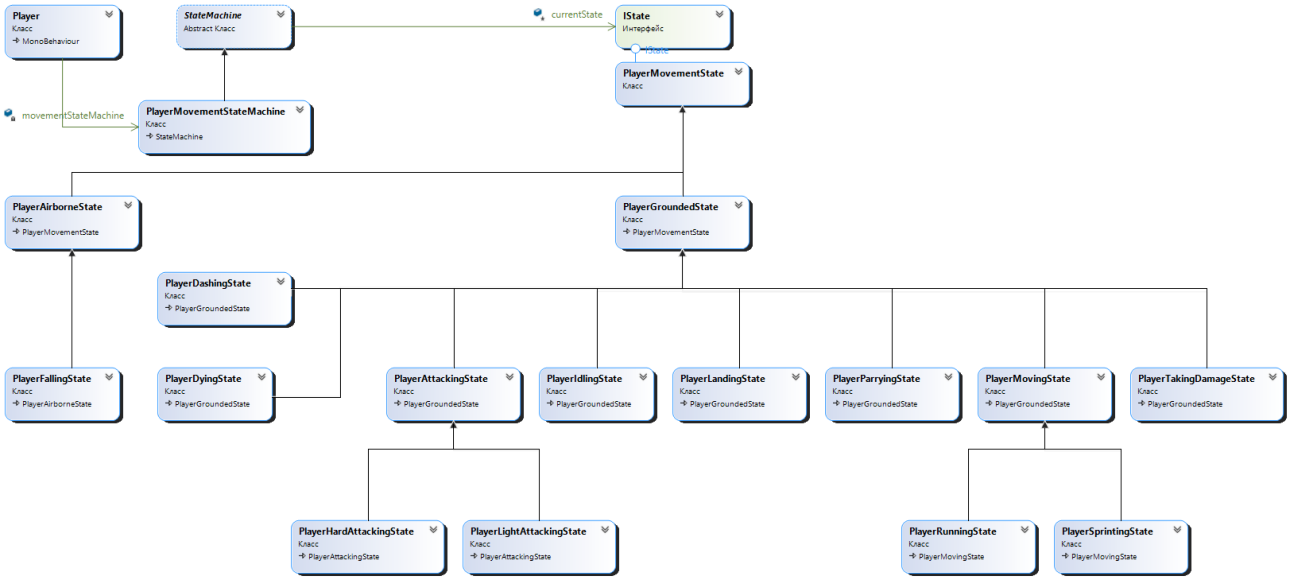


Рис. 3.10 – Діаграма класів машини станів, персонажа

На рисунку 3.11 представлено діаграму класів боса та його машини станів.

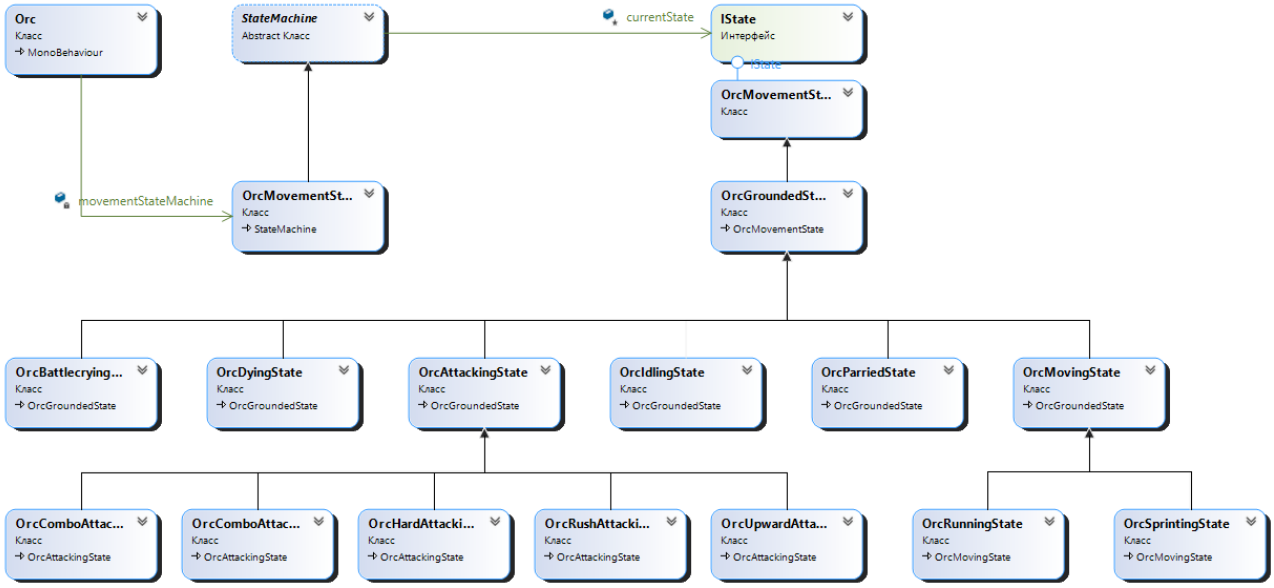


Рис. 3.11 – Діаграма класів машини станів, боса

3.4 Бойова система

3.4.1 Пошкодження об'єктів

Наступним етапом розробки є створення бойової системи взаємодії між істотами, зокрема гравцем і босом. Істоти мають властивість отримувати і завдавати шкоди, за винятком деяких випадків. Істота, яка може отримувати шкоду, реалізує інтерфейс *IDamageable*.

```
public interface IDamageable
{
    public float HealthPoint { get; }

    public void RecieveDamage(float damage);
    public void Die();

    public bool IsParrying();
    public bool IsInvulnerable();

    public float GetCurrentHP();
    public float GetMaxHP();
}
```

Так само істоти, які можуть завдавати шкоди, реалізують інтерфейс *IAttacker*.

```
public interface IAttacker
{
    public void DealDamage(Collider collider);

    public float GetBaseDamage();
}
```

Персонаж гравця реалізує ці інтерфейси в класі *PlayerCombatSystem*. Повний код класу *PlayerCombatSystem* надано в додатку Б. Так само, повний код класу боса *OrcCombatSystem* надано в додатку В.

Під час нанесення шкоди відбувається перевірка попадання, здійснювана безпосередньо влучанням у хітбокс цілі. Потім проводиться обчислення шкоди, яку персонаж повинен завдати залежно від зброї і типу застосовуваної атаки. Далі відбувається отримання компонента, що реалізує інтерфейс *IDamageable*, і перевірка, чи не парирує ціль на даний момент і чи може дана атака бути парированою. Якщо ці умови виконуються, ціль отримує шкоду. У разі, якщо ціль зуміла спарувати удар, персонаж буде парирований (рис. 3.12).

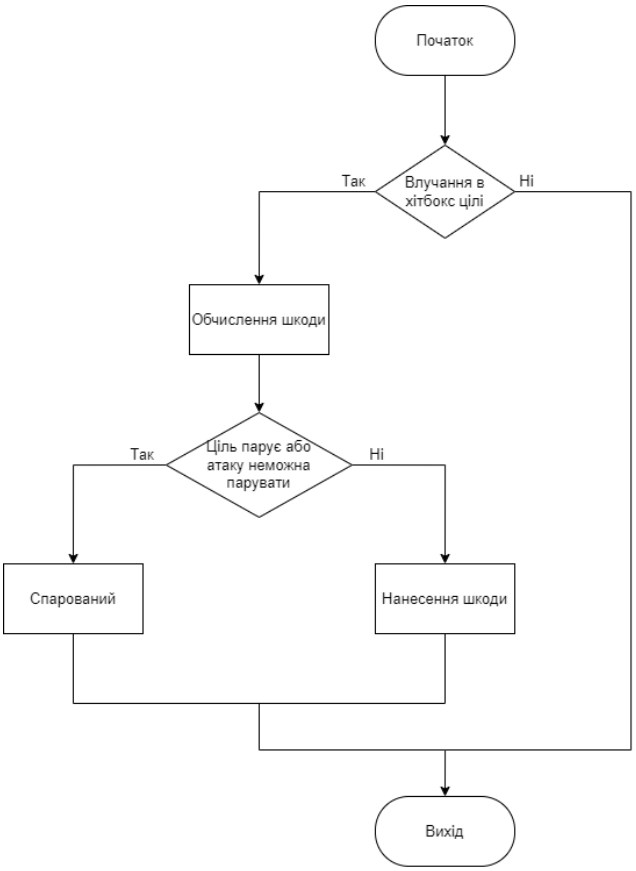


Рис. 3.12 – Алгоритм нанесення шкоди

Під час отримання шкоди здійснюється перевірка стану персонажа на невразливість. Якщо персонаж є невразливим, ніяких дій не відбувається. Якщо персонаж вразливий, то йому завдається шкода, впливаючи на його властивість *HealthPoint*, яка, при зменшенні до нуля, призводить до стану смерті. При отриманні шкоди також програються різні візуальні ефекти, і виникає стан отримання шкоди, який тимчасово обмежує здібності персонажа (рис. 3.13).

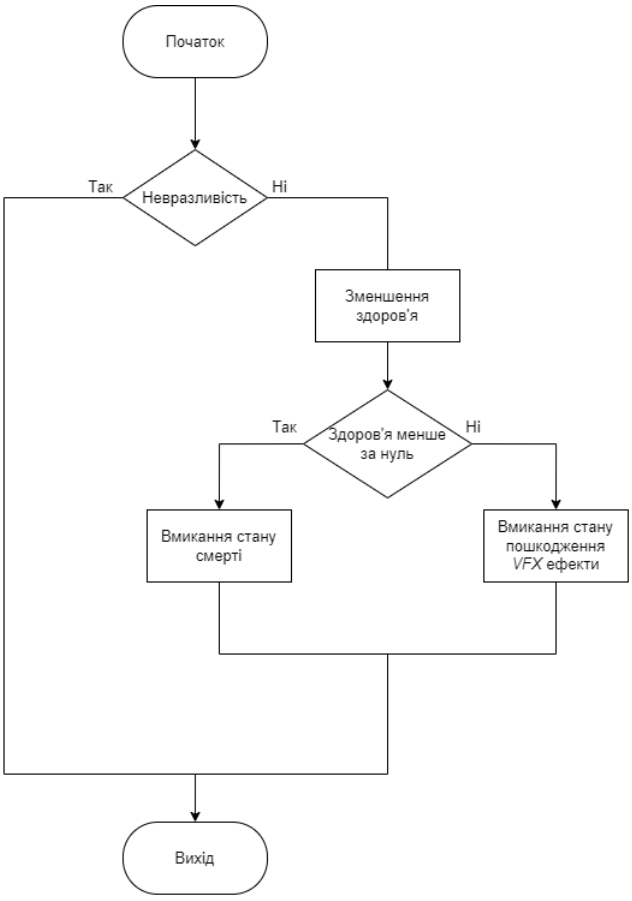


Рис. 3.13 – Алгоритм отримання шкоди

3.4.2 Система зброї

Уся шкода, яку завдають у грі, відбувається за рахунок зброї. Абстрактний клас *Weapon* є батьківським для всіх видів зброї. Він містить структуру, яка містить необхідні дані для кожного виду атаки. Крім того, клас містить базову шкоду зброї та її назву.

```
public abstract class Weapon : MonoBehaviour
{
    [Serializable]
    public struct AttackParameters
    {
        public int seriesNum;
        public float damageMultiplier;
        public float enduranceCost;
        public AnimationCurve MoveAndAttack;
        public bool unParrying;
    }

    public string Name;

    [SerializeField] protected float BaseDamage;

    public float GetBaseDamage()
    {
        return BaseDamage;
    }

}
```

Клас певної зброї, що реалізує абстрактний клас *Weapon*, містить різні види атак, характерні для цієї зброї, та їхні відповідні параметри. Крім того, клас містить екземпляр класу для хешування анімації атак, що дає змогу зберігати та керувати анімаціями, безпосередньо пов'язаними з атаками (рис. 3.14).

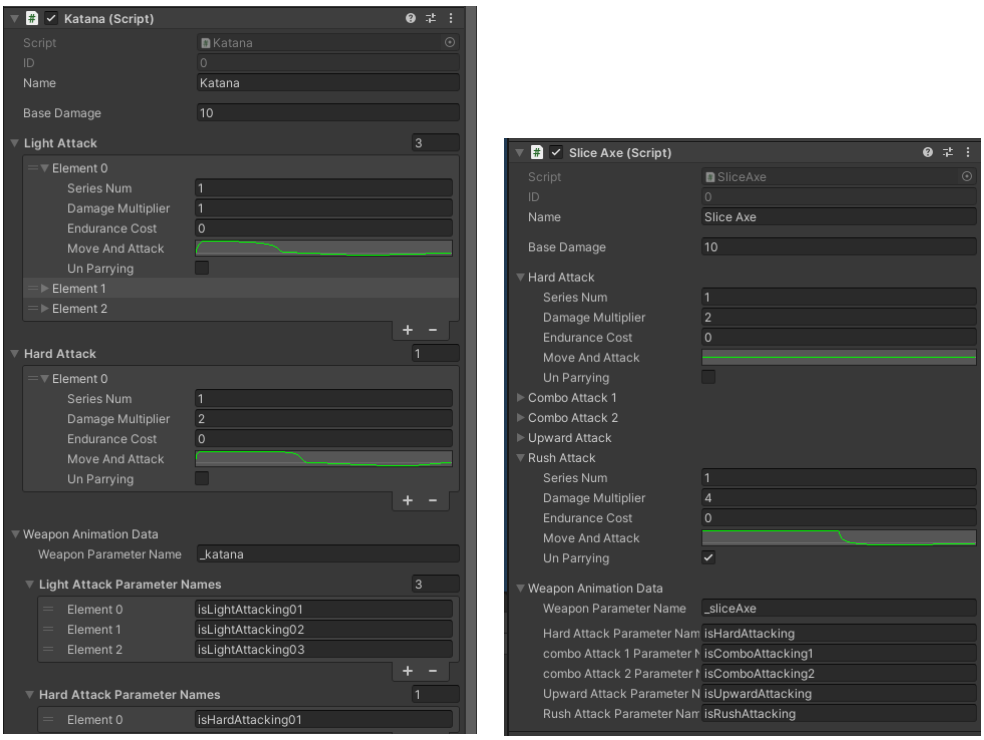
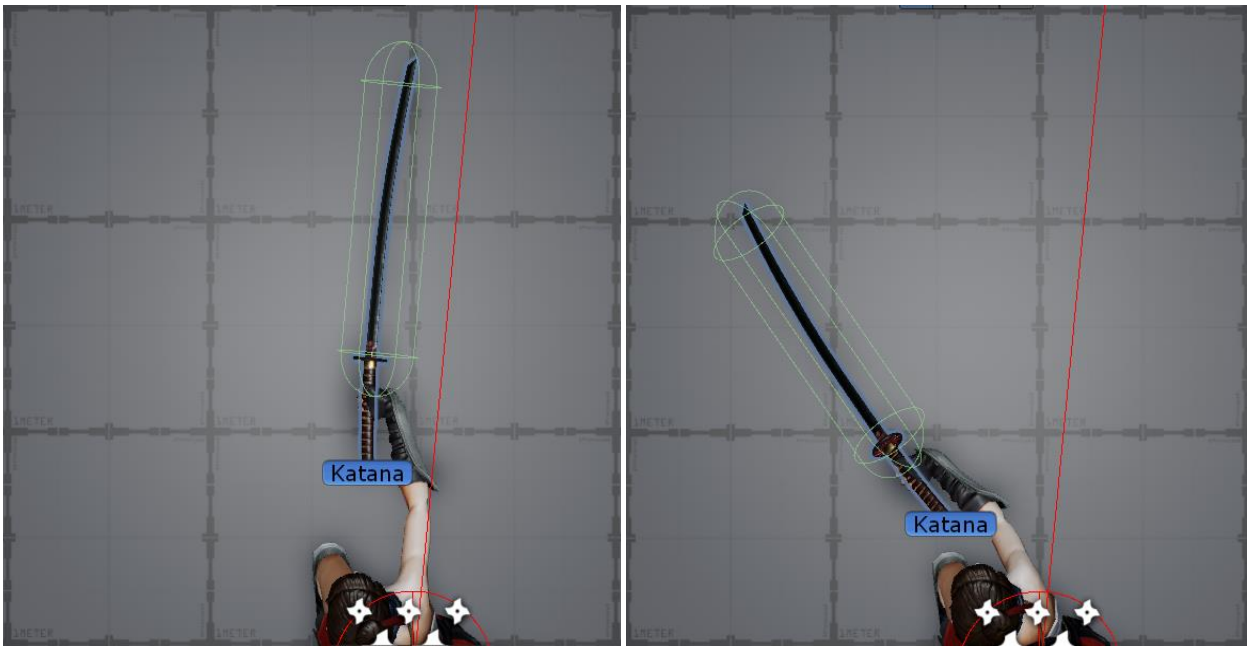


Рис. 3.14 – Компоненти катани та сокири

Для визначення влучання в противника під час атаки необхідно додати на зброю компонент *Capsule Collider* і ввімкнути параметр *Is Trigger*. За замовчуванням колайдер вимкнений і активується в момент удару зброї. Однак, такий підхід спричиняє серйозну проблему: помах і удар зброї здійснюються за допомогою анімації персонажа, унаслідок чого колайдер, відповідальний за визначення зіткнень із хітбоксом цілі, буде переміщатися по кадрах, аналогічно анімації. На рис. 3.15,а зображено перший кадр удару, тоді як на рис. 3.15,б – другий кадр. Помітно, що зброя здійснює стрибок на значну відстань, що означає, що якщо на шляху між першим і другим кадром перебуває ворог, то колайдер не зможе його зачепити.



а) б)

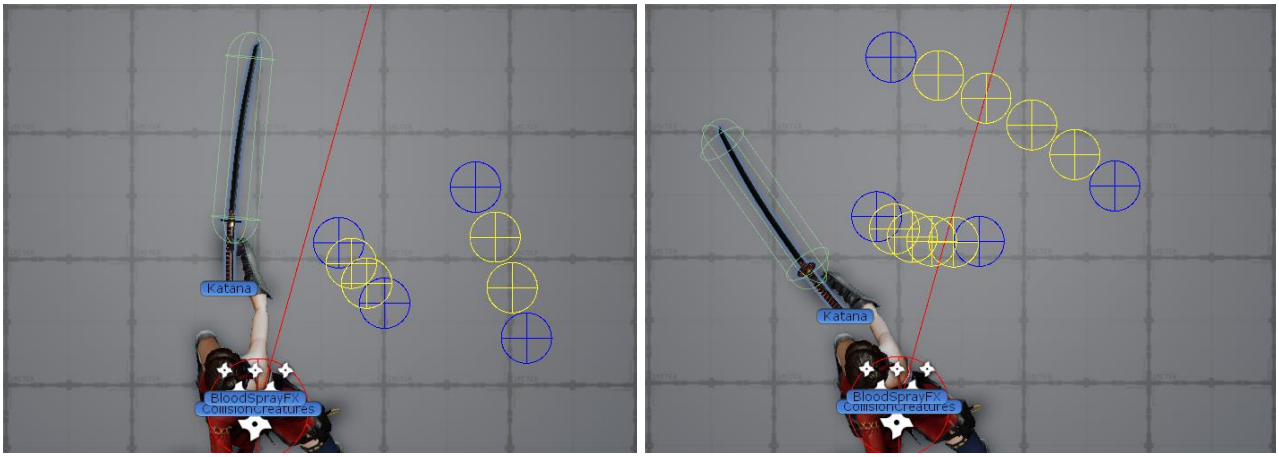
Рис. 3.15 – Анімація атаки: а) – перший кадр; б) – другий кадр

Для вирішення цієї проблеми пропонується додати скрипт *WeaponFillerColl* до зброї. Суть його роботи полягає в тому, щоб добудовувати між поточним і попереднім положенням колайдера мінімальну, для майже повного покриття площі, кількість нових колайдерів. Заповнення колайдерів відбувається в методі *FillTrail* з використанням лінійної інтерполяції.

```
private LinkedList<BufferObj> FillTrail(BufferObj from,BufferObj to)
    {
        LinkedList<BufferObj> fillerList = new LinkedList<BufferObj>();

        float distance = Math.Abs((to.position -
            from.position).magnitude);

        if(distance > weaponCollCaps.radius)
        {
            int steps = Mathf.CeilToInt(distance /
                weaponCollCaps.radius);

            float stepsAmount = 1 / (float)(steps + 1);
            float stepsValue = 0;

            for(int i = 0; i < steps; i++)
            {
                stepsValue += stepsAmount;

                BufferObj tmpBo = new BufferObj();
                tmpBo.position = Vector3.Lerp(from.position,
                    to.position, stepsValue);
                tmpBo.rotation = Quaternion.Lerp(from.rotation,
                    to.rotation, stepsValue);
                tmpBo.radius = weaponCollCaps.radius /
                    multiplierScaleParent;
                tmpBo.height = weaponCollCaps.height /
                    multiplierScaleParent;

                fillerList.AddFirst(tmpBo);
            }
        }
        return fillerList;
    }
```

Важливо зазначити, що під час використання такого підходу сам колайдер зброї завжди залишається відключеним і використовується тільки як шаблон для створення фантомних колайдерів з тими самими розмірами і положенням. Списки, що містять інформацію про положення основних і додаткових колайдерів, перевіряють потрапляння в ціль у методі *CheckTrail* з використанням методу *OverlapCapsule*. Тому в самій грі ці колайдери не відображаються і можуть вважатися фантомними. Повний код класу *WeaponFillerColl* наданий у додатку Д.

У результаті, розглядаючи рисунок 3.16, можна помітити, що між першим і другим кадром додаються парні жовті сфери, які являють собою верхні та нижні вершини фантомних капсульних колайдерів.



а) б)

Рис. 3.16 – Анімація фантомної атаки: а) – перший кадр; б) – другий кадр

3.5 Керування аніматором

Для реалізації перемикання анімацій в об'єкта, необхідно створити *Animator Controller*. У цьому контролері слід додати всі необхідні анімації та створити відповідні підстани, що дадуть змогу структурувати анімації в певні групи (рис. 3.17 – 3.19).

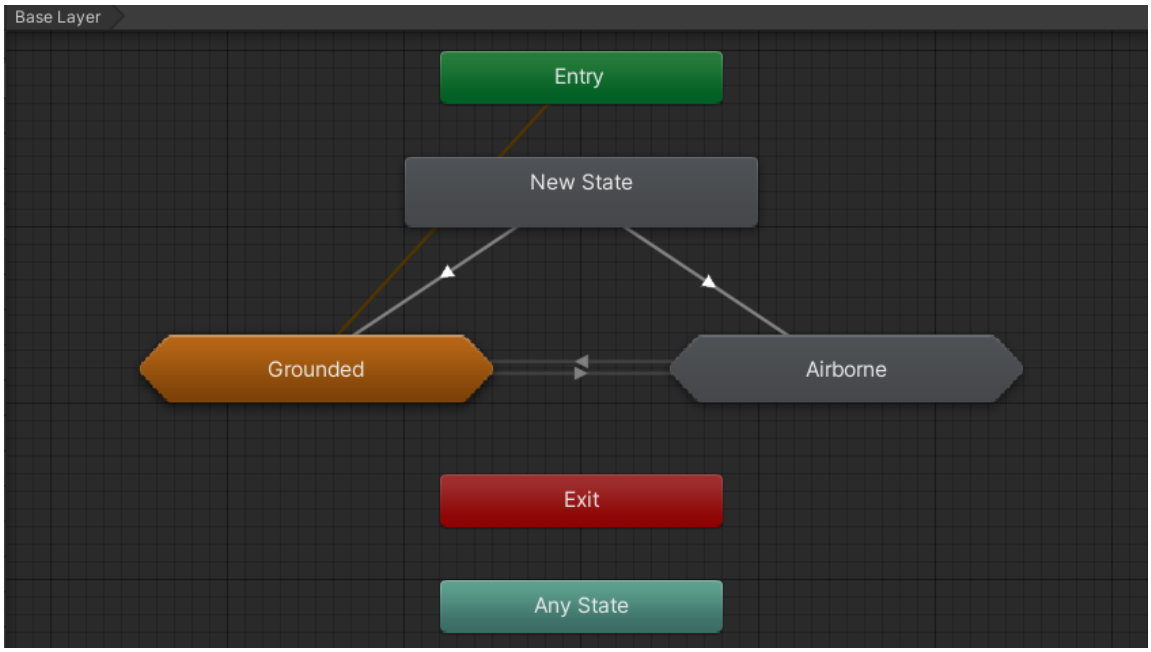


Рис. 3.17 – Зв'язки анімацій у базовому шарі

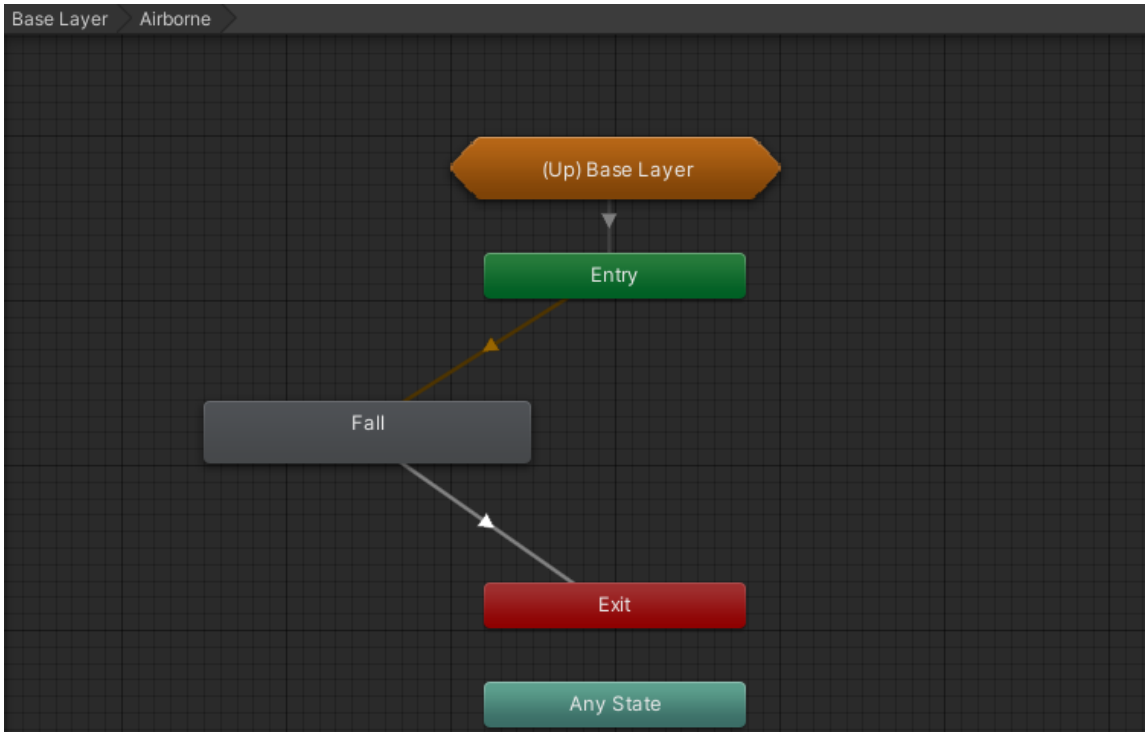


Рис. 3.18 – Зв'язки анімацій у *Airborne sub-state*

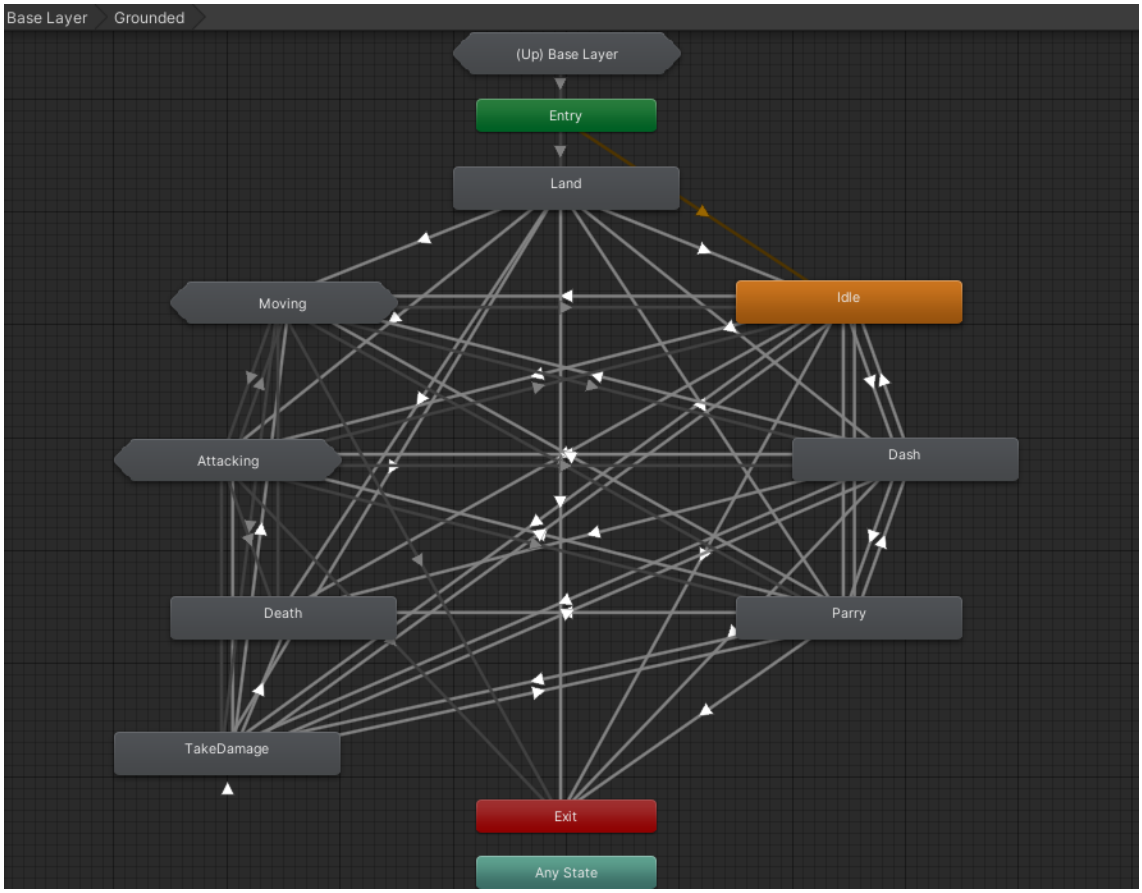


Рис. 3.19 – Зв'язки анімацій у *Grounded sub-state*

Для налаштування переходів використовуються параметри типу *bool* (рис. 3.20). Значення цих параметрів змінюються в скрипті персонажа залежно від поточного стану.

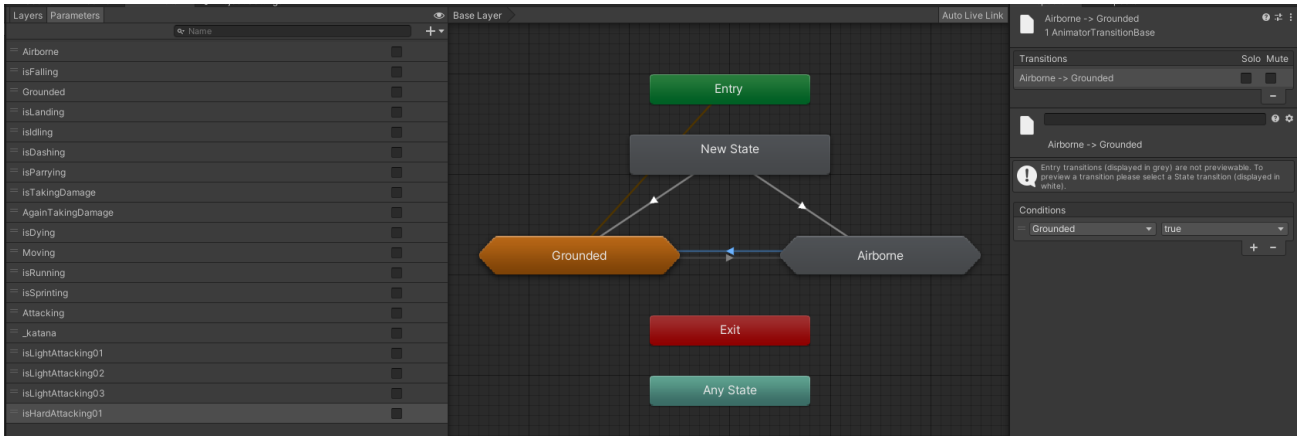
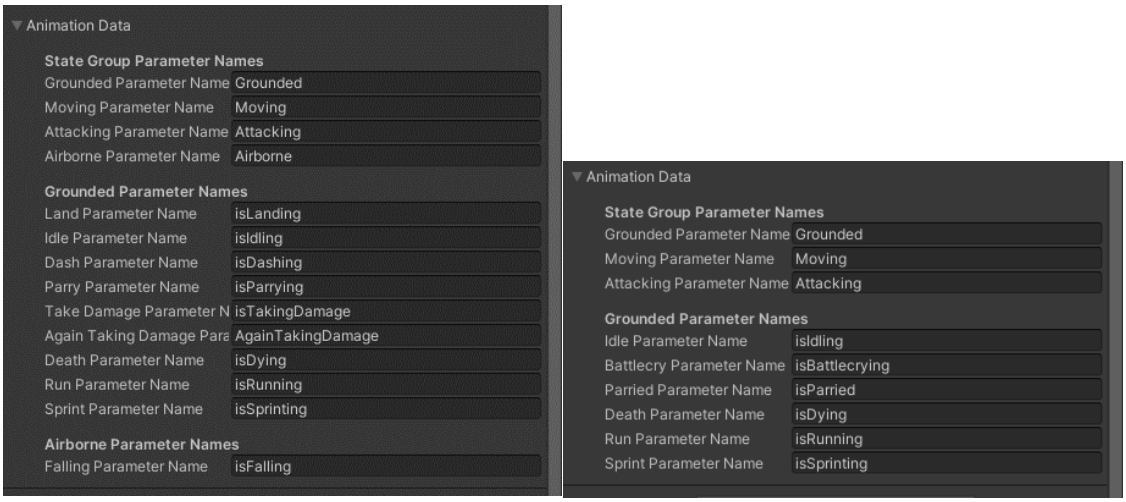


Рис. 3.20 – Параметри для переходу між станами

Для встановлення значення параметра в аніматорі зі скрипта, у методі встановлення значення (*SetBool*) необхідно вказати відповідне строкове ім'я цього параметра. Під час виклику методу *SetBool*, це строкове ім'я піддається хешуванню в режимі реального часу для пошуку відповідного параметра в аніматорі. Ця операція є ресурсномісткою, і також можливі помилки під час зазначення неправильного імені, які можуть бути складними для відстеження. Тому був використаний окремий клас, який здійснює хешування значень усіх параметрів під час першого запуску та зберігає їх у змінних (рис. 3.21).



а) б)

Рис. 3.21 – Компонент *AnimationData*: а) – персонаж гравця; б) – бос

3.6 Штучний інтелект ворогів

Наступний етап розробки включає розробку штучного інтелекту (ШІ) для ворогів, зокрема боса. Для того щоб бос взаємодіяв із гравцем, він повинен мати ШІ поведінки в різних ситуаціях.

Реалізація ШІ являє собою автономне керування переходами станів, виходячи з положення в навколишньому середовищі та поведінкою гравця. Основна логіка для переходів полягає в наступному: спочатку бос перебуває в стані спокою, оглядаючи все навколо себе, для виявлення противника (рис. 3.22).

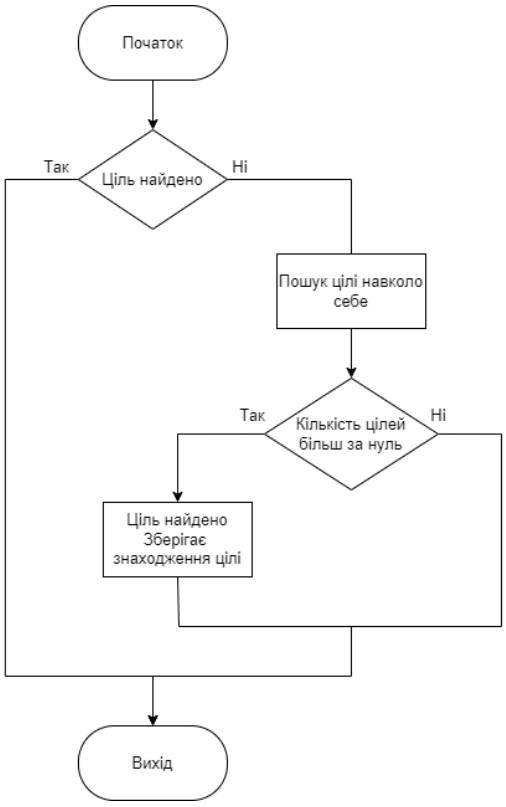


Рис. 3.22 – Алгоритм ШІ пошуку цілі

Коли ціль виявлено, бос починає вирішувати що робити далі. Рішення про наступний стан визначається кількома факторами: відстань між босом і ціллю визначає, чи повинен бос бігти в напрямку цілі або йти до неї, або атакувати її, або стояти на місці. Так само атаки мають перезарядку, щоб запобігти нескінченним атакам боса по цілі. Якщо здоров'я боса знижується до нуля вперше, він переходить до другої стадії, в якій додається нова атака. Ця атака

використовується тільки один раз після переходу на другу стадію, та щоразу коли бос починає бігти (рис. 3.23).

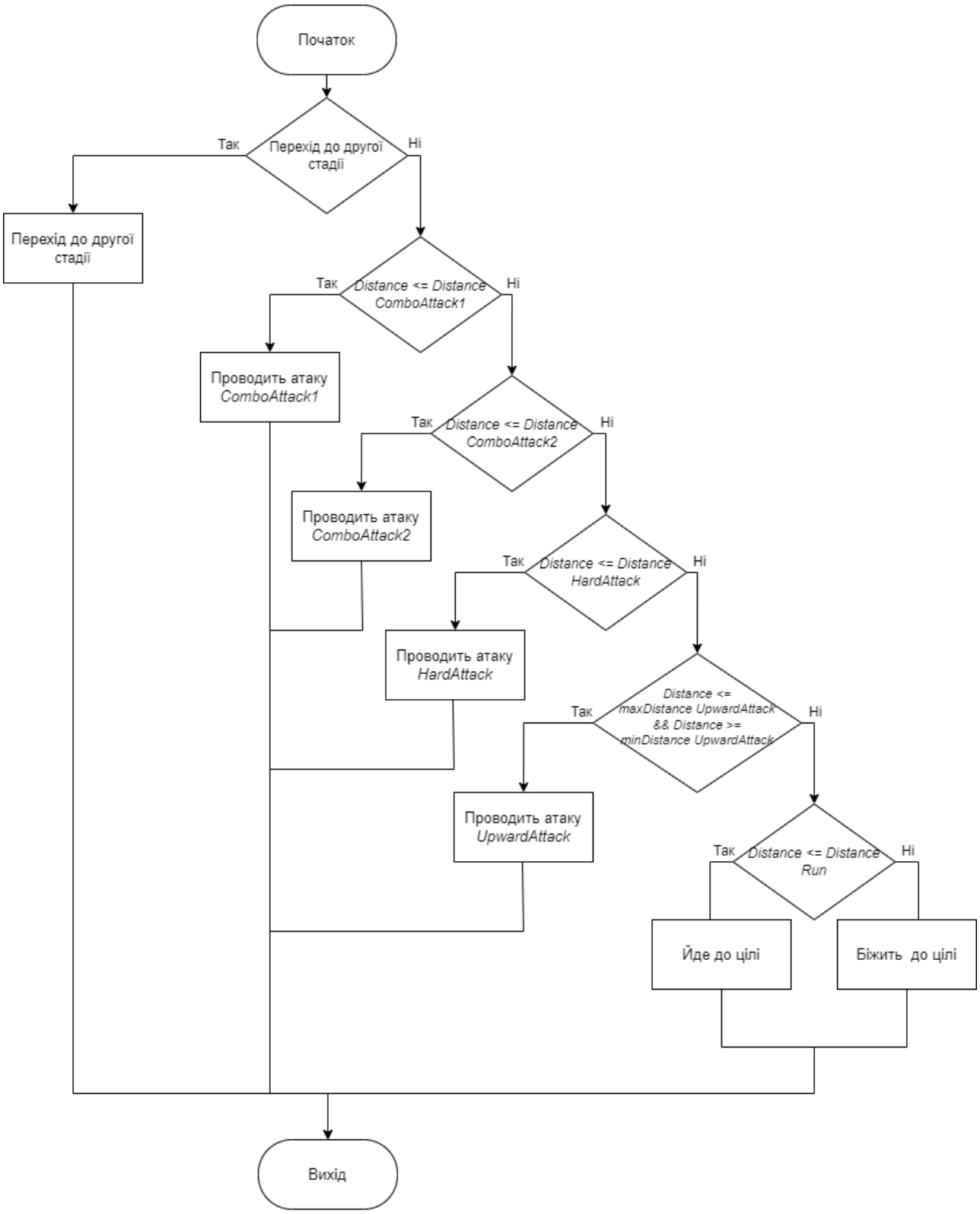


Рис. 3.23 – Алгоритм бою ШІ

Налаштування радіусів атаки, виявлення противника, та інших параметрів здійснюється в окремому класі *OrcAiData*. У цьому класі можна налаштувати радіус дії атаки, радіус переходу в стан бігу або радіус переходу в стан ходьби, а також встановити час перезаряджання атак. Усі налаштування радіусів можуть бути продемонстровані в редакторі (рис. 3.24).

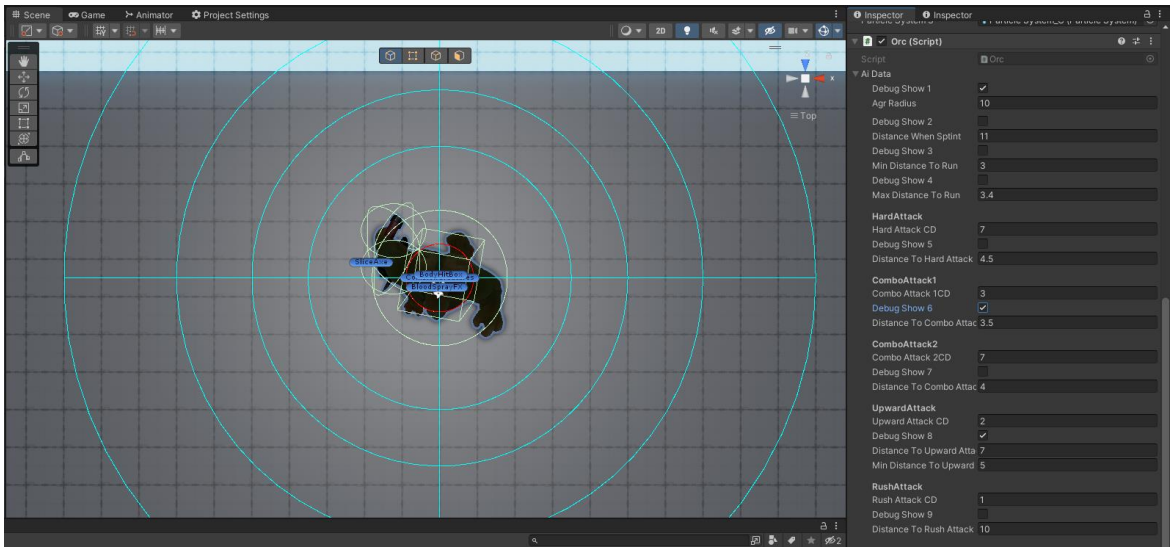


Рис. 3.24 – Компонент *AiData*

3.7 Підготовка сцени

Враховуючи всі попередні етапи, додано префаби всіх об'єктів. Так само визначено їхнє взаємне розташування на схемі рисунка 3.25.

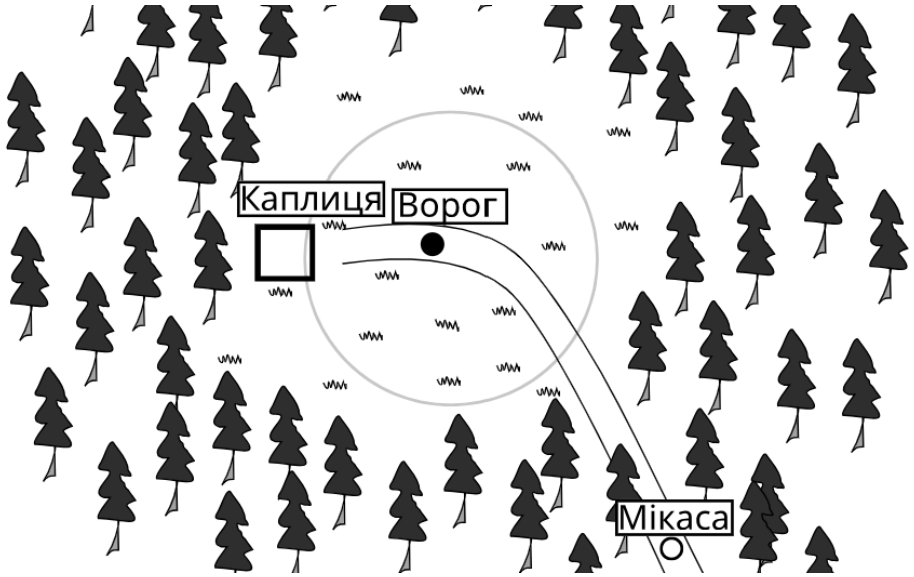


Рис. 3.25 – Схема основної мапи

Розташований на схемі об'єкт Мікаса. Має такі компоненти: *Rigidbody*; *Capsule Collider*; *Player Combat System*; *Player*. Додатково містить у собі катану та щит. Повний префаб представлений на рисунку 3.26.

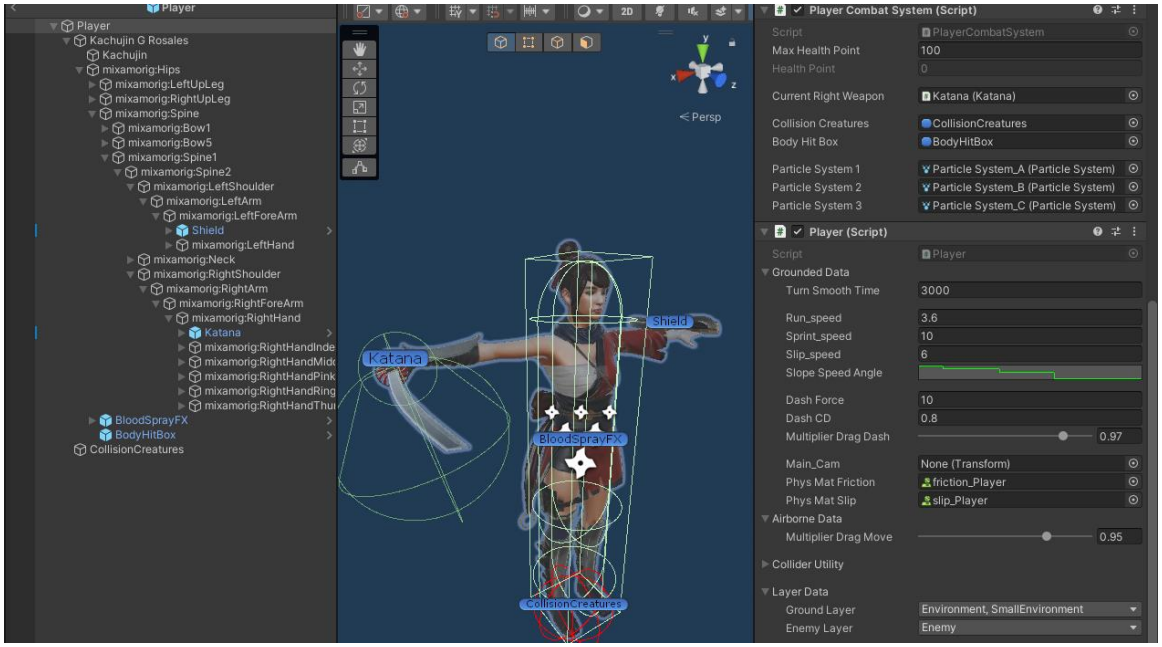


Рис. 3.26 – Префаб головного персонажа

Розташований на схемі об'єкт Ворог. Має такі компоненти: *Rigidbody*; *Capsule Collider*; *Orc Combat System*; *Orc*. Додатково містить у собі сокиру. Повний префаб представлений на рисунку 3.27.

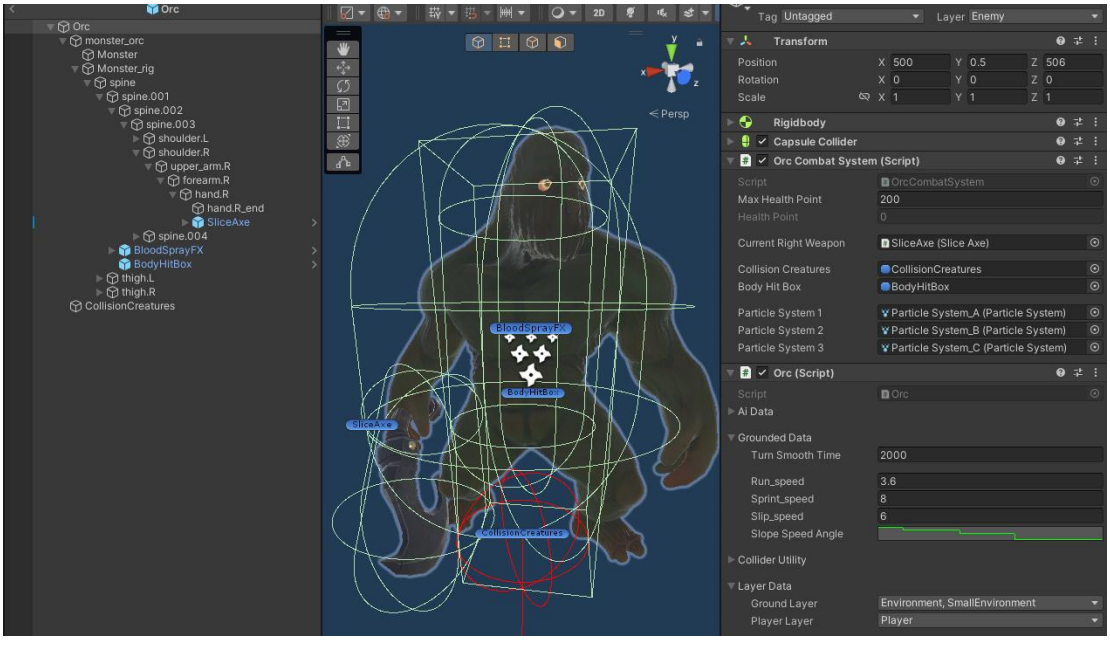


Рис. 3.27 – Префаб боса

Розташований на схемі об'єкт Каплиця. Повний префаб представлений на рисунку 3.28.

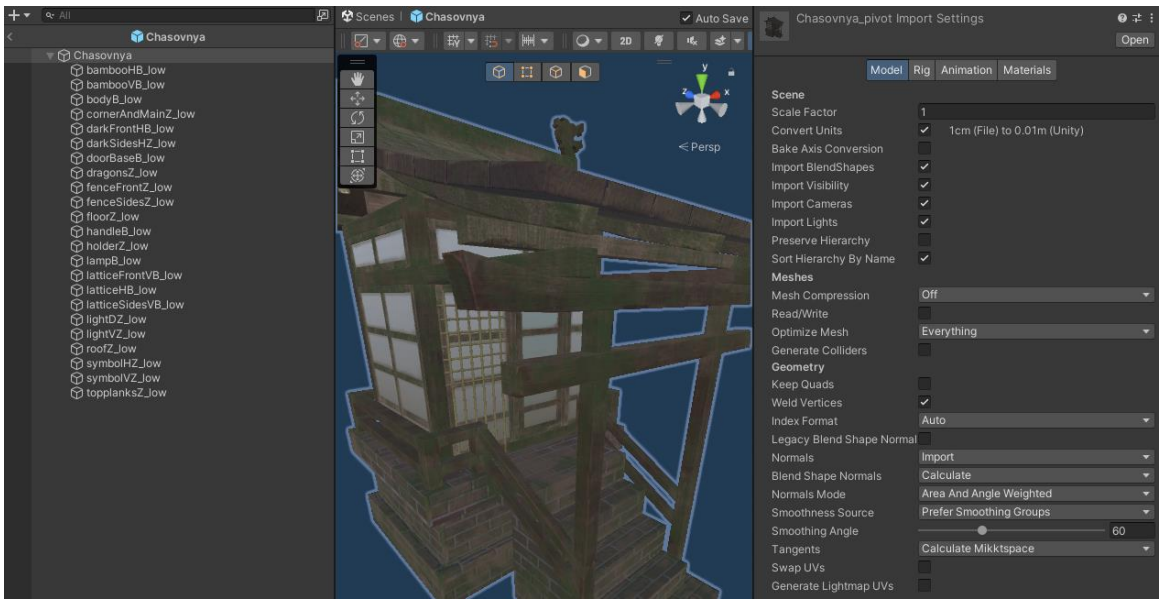


Рис. 3.28 – Префаб каплиці

Реалізація схеми на сцені (рис. 3.29).

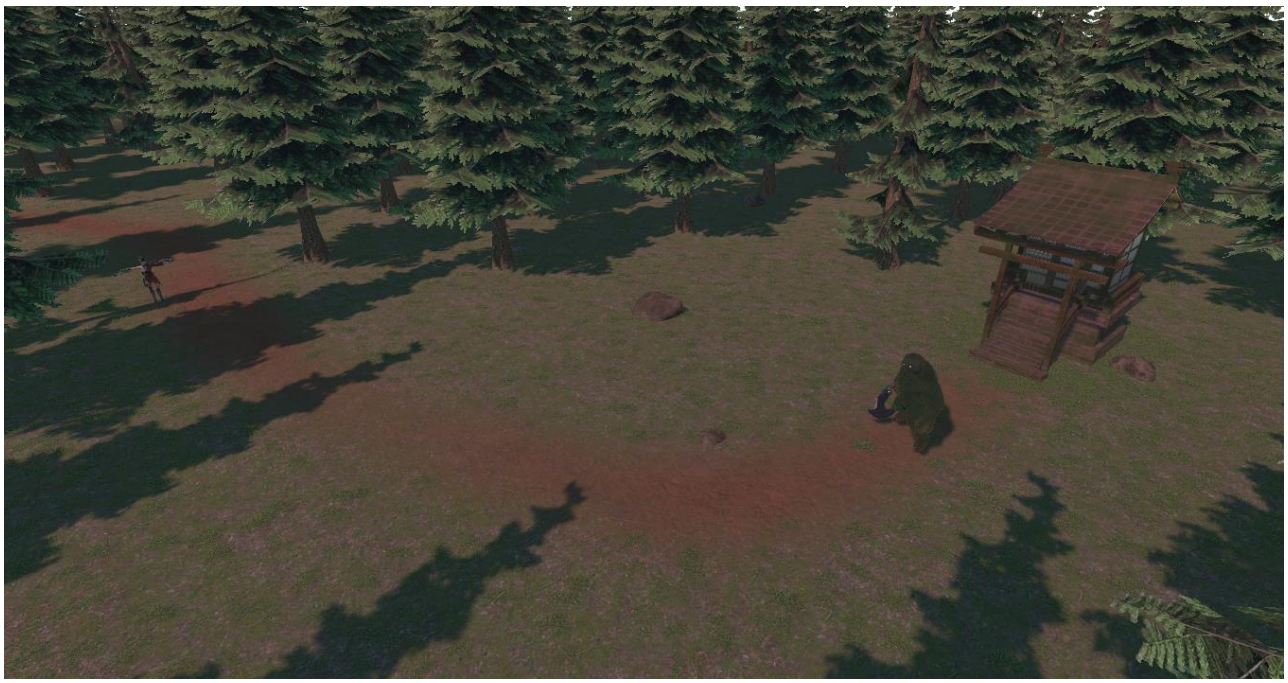


Рис. 3.29 – Фінальна головна сцена

3.8 Керування камерою

Камера є одним із ключових елементів у розробці ігор, оскільки вона відображає весь віртуальний світ на екрані, а керування нею становить істотну частину взаємодії гравця з грою.

На рисунку 3.30 показано компонент камери *CamController*.

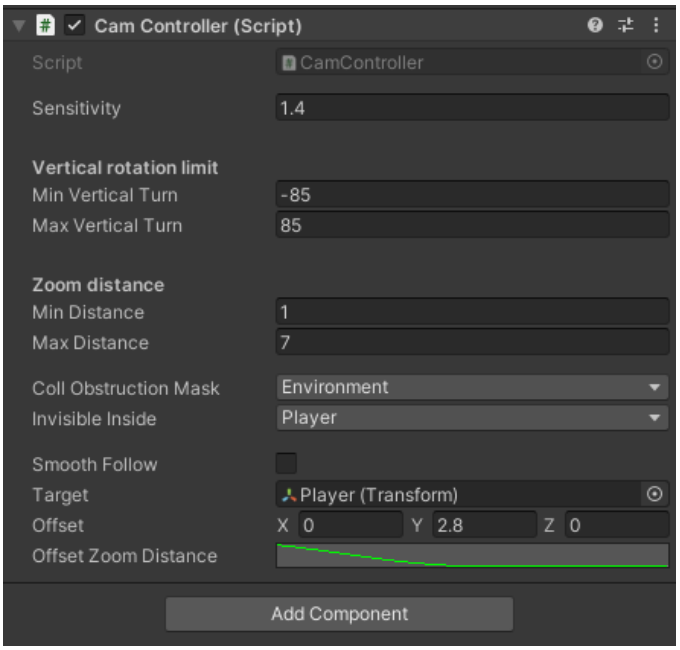


Рис. 3.30 – Компонент *CamController*

Поворот камери здійснюється за допомогою руху миші. Для неї можна налаштувати чутливість. Слід зазначити, що є обмеження на вертикальний кут повороту камери.

Наближення і віддалення камери здійснюється за допомогою колеса миші. Максимальне наближення та віддалення також можна налаштувати. Слід зазначити, що в разі сильного наближення камери її фокус трохи зміщується вгору, щоб голова персонажа була по центру.

Ще однією важливою механікою камери є зіткнення камери з оточенням у грі. Це необхідно, щоб камера не проходила крізь текстури. Реалізація цієї функціональності полягає у наступному: пускається промінь за допомогою *BoxCast*, з центру фокусу камери встановленим на персонажі. Потім

вираховується половинний вектор розміру прямокутного паралелепіпеда камери, для розміру променя, що кидається. Цей промінь спрямований у бік камери. Далі вказується шар об'єктів, з якими камера має стикатися. У разі виявлення об'єкта оточення між камерою і персонажем, камера автоматично наближається на таку відстань, щоб опинитися перед цим об'єктом.

```
Vector3 CameraHalfExtends
    {
        get
        {
            Vector3 halfExtends;
            halfExtends.y = cam.nearClipPlane * Mathf.Tan(0.5f *
                Mathf.Deg2Rad * cam.fieldOfView);
            halfExtends.x = halfExtends.y * cam.aspect;
            halfExtends.z = 0f;
            return halfExtends;
        }
    }

private void CamCollision()
    {
        Vector3 lookDirection = transform.rotation * Vector3.forward;

        if (Physics.BoxCast(transform.position, CameraHalfExtends, -
             lookDirection, out RaycastHit hit, transform.rotation,
             distance, CollObstructionMask))
        {
            distanceColl = hit.distance - 0.1f;
            first = true;
        }
        else
        {
            if (first)
            {
                distance = distanceColl;
                first = false;
            }
            distanceColl = distance;
        }
    }
```

Повний код класу *CamController* наданий у додатку Е.

На рисунку 3.31 видно, що камера зіткнулася зі стіною, при цьому відстань між камерою і персонажем має дорівнювати 7 (змінна *distance*).

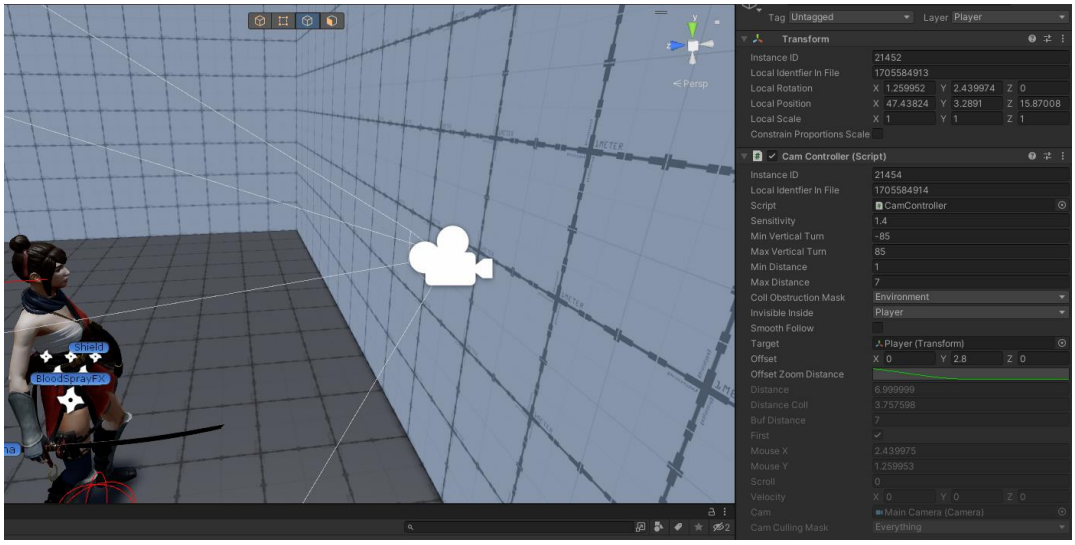


Рис. 3.31 – Зіткнення камери

Висновок до третього розділу

У цьому розділі розглянуто вибір засобів для розробки, таких як ігровий рушій, мова програмування і зовнішній редактор коду. Також повністю описано етапи проектування гри, такі як імпортування ассетів, фізичні властивості об'єктів, машина станів, налаштування анімацій, бойова система, штучний інтелект, розташування рівня та поведінка камери в грі.

РОЗДІЛ 4
ЕКОНОМІЧНІ РОЗРАХУНКИ

«В умовах відкритої ринкової економіки розширюється діапазон оцінки ефективності науково-технічних розробок, а отже, збільшується кількість основних видів ефективності НДДКР, які необхідно визначити з метою цієї оцінки»[5]. До них належать:

1. Науково-технічний ефект. Розробка гри розширює підхід до бойової системи.

2. Економічний ефект. Розробка гри має на увазі отримання відносно швидкого прибутку при закінченні розробки технології. Прибуток залежить напряму від просування гри та якості зробленої продукції, отриманий шляхом продажу електронних копій, або внутрішньоігрових транзакцій.

3. Соціальний ефект. Створена гра приносить суспільству як позитивні, так і негативні наслідки. Переваги полягають у позитивному проведенні часу, та розвитку корисних якостей у гравця. Недоліки ж полягають у занадто довгому проведені часу в грі, що може мати негативний ефект для здоров’я.

4. Маркетинговий ефект. Ігри цього жанру досить відомі, тому використання нових підходів для розробки, допускають доцільність створення нової гри.

5. Екологічний ефект. Розробка гри екологічно безпечна, оскільки вона майже не використовує природні ресурси.

Для оцінки потенційних витрат на розробку гри, необхідно провести аналіз аналогів на світовому ринку та розрахувати приблизні витрати на розборку, публікацію та маркетинг гри.

Видатними гравцями на світовому ринку комп'ютерних ігор різних жанрів є платформи, такі як *Steam* та *Epic Games*. Завдяки великій кількості схожих ігор, конкуренція в цій сфері надзвичайно висока. Також ігри які потребляють багато часу для проходження, можуть бути недооціненими. Тому вибір гри

здійснюється користувачами дуже ретельно. В таких умовах ефективна реклама є невід'ємною складовою успіху.

У разі невеликих фінансових можливостей інді розробників, хорошим рішенням буде реклама у стримерів і блогерів середньої популярності, основний контент яких пов'язаний, чи перетинається з темою ігор. Вони позитивно ставляться до інді розробників, і якщо їм сподобається продукт, можуть навіть безкоштовно записати його проходження для своїх глядачів.

У середнього блогера реклама гри буде коштувати приблизно 200 - 1000 доларів (7400 - 37000 гривні), в залежності від кількості підписників. У час випуску гри на ринок, бюджет реклами становить приблизно 3000 доларів (111000 гривень), Однак він може бути значно більшим, навіть в сотні разів.

Для того отримати можливість розташування гри на платформі *Steam*, потрібно сплатити внесок у розмірі 100 доларів (3700 гривень).

Також потрібно враховувати витрати на оплату праці працівникам. Наприклад, для розробки базової гри жанру «Слешер» можуть знадобитися один програміст, один розробник *Unity*, один *3D*-художник та один саунд-дизайнер. Якщо припустити, що робота над грою триватиме протягом року, а середня місячна заробітна плата для цих спеціалістів становить відповідно 60000, 40000, 35000 і 20000 гривень, то загальна сума витрат на оплату праці протягом року складатиме 1860000 гривень.

Для розрахунку витрат нашого проекту буде використано середні витрати на одного розробника та одного 3д художника, що будуть робити гру з нуля, використовуючи деякі безкоштовні ассети та анімації, на ноутбуках хорошої комплектації протягом року, та рекламою продукту протягом двох місяців.

Витрати такі:

– електроенергія: за рік роботи за середнім ноутбуком буде спожито 740 кВт енергії, що буде становити 1065 гривень, на одного;
– ноутбук: для розробки гри жанру «Слешер» рекомендовано використовувати ноутбуки із хорошою комплектацією. Подібний пристрій коштує в середньому 38000 гривні;

– інтернет: за рік використання тарифу інтернету в 100 мбіт/с у провайдера Чорне Море буде витрачено 2820 гривень;
– заробітна плата працівникам: для того, щоб виконати даний проект достатньо одного розробника рівня *Junior*, та одного 3д художника середнього рівня, що станом на 04.05.2023 буде коштувати в середньому 37500 гривень в місяць. Зображення, деякі ассети та анімації будуть використовуватися безкоштовні, тож плати вони не потребують. Так як гру будуть розробляти власники гри, плати вони не потребують;
– внесок за публікацію гри на платформі *Steam*. Сума внеску 100 доларів, що станом на 04.05.2023 становить 3656,86 гривень;
– реклама в блогерів та стримерів протягом двох місяців. Вирішено витратити 3000 доларів в місяць, тобто 6000 доларів за два місяці. Це 222000 гривні.

Підсумовуючи, загальні витрати на гру протягом року становлять 83541,86 гривні, при умові, що гру робить власник гри, в іншому випадку витрати становлять 983541,86 грн. Потім після випуску гри необхідно витратити 222000 гривні для реклами самої гри.

«Науково-технічну ефективність (НТЕ) результатів прикладних робіт визначають на основі показників науково-технічного рівня. Оцінка науково-технічної ефективності НДДКР відбувається на основі показника ($О_{НТЕ}$), який представляє собою ступінь досягнення максимально можливого рівня, значення якого дорівнює 1 (одиниці):

$$O_{НТЕ} = K^{Ф}_{НТЕ} / K^{П}_{НТЕ}, \quad (4.1)$$

де $K^{Ф}_{НТЕ}$ – показник (коефіцієнт) фактичного рівня науково-технічної ефективності;

$K^{П}_{НТЕ}$ – показник (коефіцієнт) потенціально можливого рівня науково-технічної ефективності (дорівнює одиниці).

Значення показника $K^{Ф}_{НТЕ}$ визначають на основі шкали експертних оцінок (табл. 4.1).

Таблиця 4.1

Шкала експертних оцінок для виміру рівня науково-технічної ефективності проектів

№	Групи показників	Характеристика показників	Інтервал рейтингового числа	Коефіцієнт значущості показників
1	Науково-технічний рівень	Перевищує кращі світові аналоги	10	0,35
		Відповідає світовому рівню	7 – 9	
		Нижче кращих світових аналогів	5 – 6	
		Перевищує кращі вітчизняні аналоги	3 – 4	
		Відповідає вітчизняному рівню	1 – 2	
		Нижче вітчизняного рівня	0	
2	Перспективність	Першочергова значущість	8 – 10	0,35
		Значущий	5 – 7	
		Корисний	1 – 4	
3	Потенційний масштаб практичного використання	Світовий ринок	10	0,20
		Галузі національної економіки	7 – 9	
		Галузь (регіон)	3 – 6	
		Окремі підприємства (об’єднання)	1 – 2	
4	Ступінь вірогідності досягнення позитивних результатів	Великий	10	0,10
		Середній	5 – 9	
		Малий	1 – 4	

Примітка**:** об'єкт оцінки і аналог(и), які порівнюють за однаковими показниками, наведеними у співставленому вигляді відхилення в значеннях кожного з показників, мають бути однаковими для варіантів, що порівнюються.

Проведення оцінки.

Визначають $К^{Ф}_{НТЕ}$ на основі експертної оцінки науково-технічного рівня розробки.

З цією метою:

− розроблюють перелік специфічних показників, необхідних для виміру науково-технічного рівня розробки;

− формують групу аналогів, які реалізовані на світовому і вітчизняному ринках;

− здійснюють відповідні розрахунки для співставлення показників і визначення балів по табл. 4.1.

До числа специфічних показників відносять:

− для нової техніки: продуктивність, споживання інженерних ресурсів на виробітку одиниці продукції, потреба в робочих, які обслуговують обладнання, експлуатаційні витрати на одиницю продукції;

− для нових матеріалів і речовин: вміст корисних речовин для виробітки готової продукції, питома вага відходів у загальному обсязі переробленої сировини, вартість одиниці … нового матеріалу;

− для нових технологій: якість виробленої продукції, енергоємність і трудомісткість продукції, собівартість одиниці продукції.

З метою спрощення визначення $К^{Ф}_{НТЕ}$ у табл. 4.2 не введено показника витрат на одиницю продукції»[5].

Таблиця 4.2

Порівняльні показники для виконання оцінки НТЕ

ПОКАЗНИКИ	Варіанти технології	
	розробленої	співвідносної (аналога)
Рівень новизни	світовий	–

«На основі співставлення даних таблиці встановлюють бали по характеристиках чотирьох груп і на цій основі розраховують значення інтегрального показника НТЕ:

$$НТЕ = \Sigma Б_i \times К_i^3, \quad (4.2)$$

де $i = 1 \div 4$;

Бі – бали (рейтингове число);

К – коефіцієнт значущості показників;

Рівень науково-технічної ефективності НДДКР розраховано на основі наведених даних прикладу (табл. 4.3)»[5].

Таблиця 4.3

Експертна оцінка і розрахунок величини інтегрального показника НТЕ

№	Групи показників	Рейтинг експертів			Середня за експертними оцінками	НТЕ
		1	2	3		
1	Науково-технічний рівень	8	8	9	8,33	2,92 (8,33 х 0,35)
2	Перспективність	8	8	7	7,67	2,68 (7,67 х 0,35)
3	Потенційний масштаб практичного використання	9	9	10	9,33	1,87 (9,33 х 0,20)
4	Ступінь вірогідності досягнення позитивних результатів	6	7	5	6	0,6 (6 х 0,10)
Всього						8,07

$$НТЕ=8,33\cdot0,35+7,67\cdot0,35+9,33\cdot0,20+6\cdot0,10=2,92+2,68+1,87+0,6=8,07$$

Отриманий результат слід порівняти з максимально можливим значенням, яке дорівнює 10 балам (10·0,35 + 10·0,35 + 10·0,2 + 10·0,1).

Отже, оцінка рівня НТЕ може бути зроблена за допомогою інтегрального коефіцієнта оцінки НТЕ ($К_{НТЕ}$):

$$K_{НТЕ} = \frac{\text{НТЕ}}{10} \cdot 100\ \%$$

На основі даних табл. 3.3 можна дійти до висновку, що $K_{НТЕ}$ відповідає 80,7 %, тобто:

$$\frac{8.07}{10} * 100\% = 80,7\ \%$$

«В тому випадку, коли значення $K_{НТЕ}$ перевищує середнє значення, яке дорівнює 5,0, має бути зроблено висновок про достатній рівень НТЕ:

- цілком достатній 5,0 - 6,0;
- достатній 6,1 - 8,0;
- достатньо високий 8,1 - 9,0;
- високий 9,1 - 10.

Таким чином, рівень НТЕ технології можна визнати достатньо високим. Отже, розроблену технологію пропонується впроваджувати у виробництво»[5].

Висновки до четвертого розділу

Зробивши оцінку ефективності розробки гри жанру «*Slasher*», можна зробити висновок, що розробка даної гри є вигідною, на що вказує коефіцент науково-технічної ефективності.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ

5.1 Основні положення охорони праці

Охорона праці визначається як система заходів, що охоплює законодавчі, організаційно-технічні, соціально-економічні, санітарно-гігієнічні і лікувально-профілактичні заходи, спрямовані на збереження життя, здоров'я і працездатності працівників під час трудової діяльності. Охорона праці має на меті забезпечити безпеку і здоров'я працівників, а також створити комфортні умови праці, що сприятимуть максимальній продуктивності. Об'єктом управління охороною праці є діяльність підрозділів, служб і колективу підприємства з метою забезпечення безпечних та здорових умов праці.

У цій кваліфікаційній роботі досліджуються питання охорони праці щодо місця праці, де здійснюється розробка гри і умови виконання роботи.

5.2 Недоліки та умови роботи за комп’ютером

Дисплей комп'ютера може бути джерелом різних видів випромінювання, таких як електростатичні поля, слабкі електромагнітні випромінювання в низькочастотному, наднизькочастотному і високочастотному діапазонах (2 Гц - 400 кГц), рентгенівські промені, ультрафіолетове і інфрачервоне випромінювання, а також видиме світло. Хоча вплив цих видів випромінювання на організм людини ще не повністю досліджений, відомо, що він має наслідки (експерименти на тваринах показують вплив слабких електромагнітних полів низької і наднизької частоти на біологічні об'єкти, особливо на мозок).

Довготривала неправильна поза, коли розробник сидить перед екраном дисплея, може спричиняти втому та виникнення болю в спині, шиї та плечових суглобах.

Робота з клавіатурою веде до неприємних відчуттів в ліктьових суглобах, передпліччі, зап'ястях, кистях і пальцях рук.

Операторська діяльність передбачає переважно візуальне сприйняття інформації, відображеної на моніторі, і тому зоровий апарат людей, які працюють з ПК, піддається значному навантаженню. Фактори, які найбільше впливають на зір, включають:

- недосконаліть методів створення зображення на моніторі, включаючи недостатньо оптимальні параметри схем розгортки електронно-променевої трубки (ЕПТ);
- несумісність параметрів монітора і графічного адаптера, низька роздільна здатність монітора, розфокусування, незведення променів та інші технічні характеристики монітора;
- недоцільна організація робочого місця, що спричиняє відблиски на екрані, недостатнє освітлення робочих місць і недотримання необхідної відстані між очима оператора і екраном;
- віконні прорізи приміщень з ПК повинні бути обладнані регульованими пристроями (жалюзі, завіски, зовнішні козирки), а внутрішнє оздоблення поверхонь має бути виконано з дифузно-відбивних матеріалів;
- освітлення приміщень, де встановлені ПК, повинне включаючи природне і штучне освітлення.

Рекомендована площа приміщення для роботи з ПК має бути не менше 6,0 кв. м., а об'єм - не менше 20,0 куб. м. Забороняється використання полімерних матеріалів (деревноволокнисті плити, миттєві шпалери, синтетичні матеріали і т.д.), які виділяють шкідливі хімічні речовини, для внутрішнього оздоблення приміщень з ПК. Застосування полімерних матеріалів можливо тільки за наявності дозволу відповідних органів та установ державної санітарно-епідеміологічної служби. Приміщення, де встановлені ПК, мають бути оснащені шафами, полицями, стелажами та іншими меблями з урахуванням вимог до площі приміщень.

5.3 Електробезпека

Електробезпека представляє собою систему організаційних та технічних заходів, спрямованих на захист людей від потенційно шкідливого та небезпечного впливу електричного струму, електричної дуги, електромагнітного поля та статичної електрики.

Забезпечення електробезпеки на підприємстві досягається шляхом дотримання вимог, викладених у відповідних актах законодавства, таких як: Правила безпечної експлуатації електроустановок споживачів (наказ Держнаглядохоронпраці від 09.01.1998 № 4); Правила безпечної експлуатації електроустановок (наказ Держнаглядохоронпраці України від 06.10.1997 № 257), що поширюються на працівників, які працюють з електроустановками Міністерства енергетики України; Правила технічної експлуатації електроустановок споживачів (наказ Мінпаливенерго України від 25.07.2006 № 258, зі змінами); Правила експлуатації електро-захисних засобів (наказ Міністерства праці та соціальної політики України від 05.06.2001 № 253); Правила улаштування електроустановок (наказ Міністерства енергетики та вугільної промисловості України від 24.07.2017 № 476); а також ДСТУ 2843-94 «Електротехніка. Основні поняття. Терміни та визначення». Останній документ встановлює терміни та визначення основних понять у галузі електробезпеки.

5.4 Пожежна безпека при роботі з комп’ютером

Сучасні комп'ютери мають високу щільність розташування елементів електронних систем. Близьке розташування сполучних проводів і комунікаційних кабелів створює ризик. При проходженні електричного струму через них виділяється значна кількість тепла, що призводить до підвищення температури в окремих вузлах до 80-100 °С. Це може спричинити оплавлення ізоляції або оголення проводів, що веде до короткого замикання та перевантаження елементів електронних схем. Перегрівання елементів може

призвести до виникнення іскор та пожежі. Для відведення тепла використовуються системи кондиціонування і вентиляції повітря. Проте, ці системи можуть створювати додаткову пожежну небезпеку, оскільки забезпечують подачу кисню, який може сприяти швидкому розповсюдженню вогню.

Отже, важливо дотримуватись заходів пожежної безпеки, уникати використання легкозаймистих матеріалів та здійснювати обережність при обслуговуванні, ремонті та профілактичних роботах, а також правильно прокладати кабелі для запобігання загорянню. Для машинних залів рекомендується використовувати негорючі або слабогорючі матеріали для прокладання кабелів під технологічними знімними підлогами, з вогнестійкістю не менше 0,5 години.

Висновок до п’ятого розділу

Було розглянуто питання, пов'язані з охороною праці на місці, де відбувається безпосереднє виконання роботи, включаючи розробку гри та умови праці. Необхідні норми пожежної безпеки при роботі з комп'ютером також були визначені.

ЗАГАЛЬНІ ВИСНОВКИ

В процесі виконання даної роботи було досягнуто наступні основні результати:

1. Зроблений аналіз особливості бойової системи та програмування гри жанру «*Slasher*».
2. Розроблений дизайнерський документ, який включає технічний опис концепції та функціональної специфікації.
3. Завантажені необхідні асети, та створено нову сцену.
4. Розроблена модель фізичної взаємодії між об'єктами, та використана методика *CapsuleFloat*, що дає змогу переміщатися по дрібних перешкодах і сходах.
5. Реалізовано патерн машини станів, що являє собою структуру кінцевих станів, кожен з яких виконує необхідні для стану дії та має певні умови переходів в інші стани.
6. Розроблена бойова система та система зброї, багатофункціональна і гнучка в налаштуванні, що унеможливлює можливість не зареєструвати потрапляння по ворогу атакою.
7. Розроблений штучний інтелект ворогів, що дає змогу приймати ворогам рішення виходячи з дій гравця.

Загалом, створена гра «*Slasher*» є успішним результатом проектування та розробки. Вона втілює унікальні ідеї та пропонує захопливий геймплей.

Гра має потенціал для подальшого розвитку, який може відбуватися у різних напрямках. Зокрема, можливість вдосконалення ШІ та бойової системи, збільшення кількості ігрових механік, покращення оптімізації гри та додавання збереження прогресу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Nier: Automata* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Nier:_Automata*.
2. *Trek to Yomi* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Trek_to_Yomi*.
3. *Thymesia* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Thymesia*.
4. *Devil May Cry 5* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Devil_May_Cry_5*.
5. Методичні вказівки до оцінки науково-технічної ефективності розробки нової технології, нового обладнання та інших інновацій. Для студентів всіх спеціальностей СВО «бакалавр» і «магістр» денної і заочної форм навчання. Укладачі Басюркіна Н.Й., Свистун Т.В. Одеса: ОНАТУ, 2022 р. 18 с.
6. *Unity (game engine)* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Unity_(game_engine)*.
7. *Hack 'n' Slash* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Hack_%27n%27_Slash*.
8. *Hack and slash* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Hack_and_slash*.
9. *Action-adventure game* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Action-adventure_game*.
10. *Steam (service)* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Steam_(service)*.
11. *Windows* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://ru.wikipedia.org/wiki/Windows*.
12. Шаблони проектування – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://gamedev.dou.ua/blogs/how-to-make-games-with-unity*.

13. *Unreal Engine* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Unreal_Engine*.
14. *CryEngine* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/CryEngine*.
15. *Godot (game engine)* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Godot_(game_engine)*.
16. *Ninja Gaiden (NES video game)* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Ninja_Gaiden_(NES_video_game)*.
17. *Bayonetta (video game)* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Bayonetta_(video_game)*.
18. *Metal Gear Rising: Revengeance* – Вікіпедія [Електроний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Metal_Gear_Rising:_Revengeance*.
19. *Unity Documentation* [Електроний ресурс]. – Режим доступу: *https://docs.unity.com*.
20. *State pattern* [Електроний ресурс]. – Режим доступу: *https://gameprogrammingpatterns.com/state.html*.
21. *Unity tutorials* [Електроний ресурс]. – Режим доступу: *https://catlikecoding.com/unity/tutorials*.
22. *Hack'n'slash game tutorial* [Електроний ресурс]. – Режим доступу: *https://medium.com/c-sharp-progarmming/making-a-hackn-slash-game-in-unity-c-6ec315e75816*.
23. *Joseph Hocking. Unity in Action: Multiplatform Game Development in C# with Unity 5. – Manning Publications*: 2018. – 370 с.: Іл.
24. *Jeremy Sumner Wycherley Gibson. Introduction to Game Design, Prototyping, and Development. – Addison-Wesley Educational Publishers Inc*: 2014. – 908 с.: Іл.
25. *HERBERT SCHILDT. C# 4.0: The Complete Reference – McGraw Hill Publications*: 2010. – 976 с.: Іл.