МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «УНІВЕРСИТЕТ «КРОК»

Фаховий коледж Університету «КРОК»

ДИПЛОМНА РОБОТА

за темою

«Розробка гри на платформі Unreal Engine»

Студент 4 курсу групи ІПЗ-20к

Тхоржевський Михайло Олегович

(прізвище, ім'я та по-батькові керівника)

Керівник дипломної роботи

Кандидат технічних наук, доцент

(посада керівника)

Добришин Юрій Євгенович

(прізвище, ім'я та по-батькові керівника)

До захисту

(резолюція «До захисту»)

_____ 12.06.2024 _____

(підпис студента) (дата) (підпис викладача)

Київ, 2024 рік

ЗМІСТ

ВСТУП

У сучасному світі відеоігри є важливою складовою розважальної культури та індустрії розваг. Той, хто краще робить свою справу – буде отримувати кращу винагороду. Тому розробка гри на високоякісному движку є ключовим етапом у створенні успішного продукту. Unreal Engine, розроблений компанією Epic Games, на думку більшість висококваліфікованих розробників гри, є одним з найкращих програм у світі, для створення ігор абсолютно різних рівнів та для різних платформ.

Розробка повноцінного ігрового проекту складається з багатьох видів робіт. Лише одного програміста буде не достатньо для створення по справжньому якісного продукту. Також варто помітити, що не завжди компанії з великими бюджетами розробляють якісний продукт. Розробка гри - це складний процес, який включає в себе багато ключових аспектів, тому саме програміст, який займається саме функціональною частиною, не буде відігравати найголовнішу роль в розробці проекту. Тому що окрім нього також важлива візуальна частина, якою займається геймдизайнер. Роботи дуже багато тому, одна людина створити якісний продукт самотужки не зможе.

Створення гри можна порівняти з написанням віршів, казок, створення фільмів та якоюсь творчою діяльністю. Тому що, при створенні власної гри в тебе є простір для маневру, ти можеш створити абсолютно все – що ти забажаєш. Людина просто немає обмежень, тільки в фантазії, робочого матеріалу та особистих навичках. Хтось має змогу створити гру про яку мріяв ще з дитинства, хтось створює гру для слави та грошей, а для когось це просто хобі.

Метою дипломної роботи є створення власного продукту, отримання практичного досвіду в самостійній розробці власного проекту.

Дипломна робота складається зі вступу, чотирьох основних розділів, висновку, списку використаних джерел та додатків.

РОЗДІЛ 1 АНАЛІЗ АКТУАЛЬНОСТІ РОЗРОБКИ ГРИ НА UNREAL ENGINE, ОГЛЯД НАЯВНИХ РЕЗУЛЬТАТІВ

Unreal Engine на даний момент входить в список найпопулярніших ігрових двигунів для розробки ігор

Рис. 1.1 Статистика популярності

Тому щодо актуальності, питань бути не повинно. На думку багатьох професійних розробників гри для PC та VR платформ, найкращим вибором буде Unreal Engine. Тому що Unreal Engine набагато потужніше та універсальніше за своїх конкурентів.

Насправді Unreal Engine дуже потужний двигун і про нього можна достатньо довго розповідати, але варто виділити основні переваги Unreal Engine та конкретно чому саме його вибирають більшість.

Однієї основної відмінності Unreal Engine не має. Але є просто багато відмінностей, які кількістю та особливо високою якістю відрізняють Unreal Engine від своїх конкурентів.

Одною з унікальних можливостей цього двигуна є Blueprints. Blueprints – це вбудована система програмування без кодування, яка дозволяє розробникам створювати складні логіку гри, взаємодіючи з об'єктами та компонентами графічного інтерфейсу користувача. В народі цю функцію просто називають "Блюпринти". Ця функція надає новачкам або програмістам, які не сильно володіють C++ створювати власний продукт. Також ця система програмування може пришвидшувати загальний процес розробки, наприклад якщо вам просто десь треба трішки змінити швидкість пересування або створити абсолютно легку логіку чогось – краще зробити це в блюпрінтах. Тому що, це легше та швидше зробити в графічному інтерфейсі чим шукати певну строку коду де знаходяться потрібна нам інформація.

Звичайний приклад, замість того аби шукати це в коді легше просто натисну на потрібну нам змінну та праворуч в розділі Default Value в полі Sprin Speed ввести потрібні нам значення, як це зроблена на малюнку 1.2.

```
// MOVEMENT
// Set Walk and Sprint Speed for Character
WalkSpeed = 300.0f;
SprinSpeed = 600.0f;
isSprinting = false;
```

Рис. 1.2. Приклад коду

Рис. 1.3. Приклад Blueprint

Наступна потужна функція Unreal Engine – потужність графічної оболонки. В цьому двигуні настільки сильно розвинута ця характеристика, що іноді важко відрізнити віртуальний світ з реальністю. Якщо у вас ціль створити проект, де обов'язково треба висока якість графіки або світ, схожий до нашої реальності, тоді Unreal Engine на даний момент один з найкращих двигунів для цього.

На двох нижче поданих малюнках ви бачите можливості, які пропонує цей двигун. Такий рівень якості створення графіки мало хто може запропонувати.

Рис. 1.4. Приклад пустельної карти

Рис. 1.5. Приклад лісової карти

Unreal Engine також відомий своєю підтримкою для розробки віртуальної реальності (VR) та доповненої реальності (AR). Unreal Engine надає інтуїтивний інтерфейс розробки для створення VR та AR вмісту. Розробники можуть легко створювати та редагувати сцени, додавати віртуальні об'єкти та налаштовувати параметри. Інтуїтивний інтерфейс розробки існує не тільки для VR чи AR, а і також для всіх інших платформ, тому більшість вибирають Unreal Engine, бо на зрозумілому інтерфейсі краще та приємніше працювати.

Вже багато хто розуміє, що AR це буквально майбутнє. Це можливсть доповнити або зробити наш світ більш зручнішим для нас, і через те, що це набирає сильні обороти популярності, двигун Unreal Engine також стає популярним, якщо не для розробки ігор, то для роботи з AR.

Останньою перевагою в цьому списку буде – Мультиплатформенність. Мультиплатформенність зараз дуже важлива, бо це на пряму може впливати на потенційний дохід, який може отримати компанія через цей продукт. Чим більша кількість платформ, на яких доступний цей продукт – тим більше клієнтів, тим самим більше дохід.

Але варто попередити, що наприклад є двигуни, які були створенні саме для однієї платформи і їх сильно розвинули в цьому напрямку. Тому не на всіх платформах Unreal Engine може конкурувати з іншими. Наприклад Unity, він звісно не створений тільки для телефонних ігор, але саме в цьому напрямку Unity буде краще чим Unreal Engine. Хоча ми не можемо сказати, що на 100%, тому що Unreal Engine так само залишається потужним двигуном але для розробки ігор не треба таких великих потужностей, які має Unreal Engine, а ось Unity для цього підходить, бо він має саме стільки – скільки цього потрібно. Зазвичай просто ігри на телефонах це не такі складні та великі проекти як на комп'ютерах або інших платформах, тому їм і не треба потужний Unreal Engine.

Взагалом аналізуючи актуальність розробки гри на Unreal Engine, можна зробити висновок про його великий потенціал та привабливість до розробників.

РОЗДІЛ 2 АНАЛІЗ ІНФОРМАЦІЙНОГО ТА ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ, ОКРЕМИХ ПРОГРАМНИХ ЗАСОБІВ, ЩО ЗАСТОСОВУЄТЬСЯ ДЛЯ РОЗРОБКИ ІНДИВІДУАЛЬНОЇ ГРИ

Весь основний процес розробки гри відбувався в двигуні Unreal Engine. Так як Unreal Engine по справжньому потужний двигун, ресурсів він їсть він дуже багато. Тому для розробки гри на цьому двигуні бажано мати потужний комп'ютер. Звичайно, якщо немає можливості розробляти проект на потужному комп'ютері то можна трохи схитртити. В самому додатку Unreal Engine можна змінати багато чого, а саме графіку. Буквально цей двигун можна налаштувати під себе та свої можливості. Багато ресурсів їсть освітлення. Як порада, можна просто змінити загальну якість графіки. На малюнки нижче показано приклад.

Рис. 2.1. Налаштування загальної якості графіки

Через такий великий функціонал можна вважати Unreal Engine чудово технологічно забезпеченим.

Про технічне забезпечення можна узагальнити, що для того аби працювати в двигуні Unreal Engine бажано мати потужний персональний комп'ютер.

Щодо інформаційного забезпечення, загально відомої інформації, яка допомагає у створенні проекту дуже багато. Існує офіційна документація від розробників цього двигуну. Там можна дізнатися більш технічну інформацію.

Unreal Engine часто обновлюється і самі розробники особисто можуть допомагати у вирішенні проблем, які потенційно могли статися через певні недопрацювані функції або певні нововведення в двигун. Взаглом служба підтримки працює дуже добре. Розробники цього двигуна створили офіційний форум, де користувачі діляться досвідом, допомагають одне одному в певних помилках або разом намагаються вирішити ті помилки, які ще ніхто до них не вирішував (зазвичай це певні баги після обновлення версії двигуна).

Створити повноцінну гру в одному додатку – неможливо. Тому щодо програмного забезпечення тут можна дуже довго казати. Але варто виділити основні для базових потреб. Почнемо з малого, особисто на моєму досвіді дуже часто використовуються звичайні сайти. Наприклад сайт, де можна конвертувати типи файлів. Де та як це знадобилось? Коли я додав музику на головний екран, чи звуки на фоні, то файл з піснею повинен бути формату "WAV", а зазвичай коли завантажуєш пісню з інтернету, то файл зберігається у форматі MP3. Двигун не пропускає формат MP3 тому ось вам і приклад використання сайтів.

Рис 2.2. Конвертація файлу в формат WAV

Visual Studio, думаю це для всіх відомий застосунок, в якому прописується логіка будь-чого. Цікавий факт, я вже казав про Blueprints, а ось блюпрінти створенні завдяки коду, це теж саме програмування тільки замість коду існують таблички. В біль великих проектах рекомендується логіки розробляти в Visual Studio на мові програмування C++ , тому що тоді оптимізація гри буде краще. Але найкращий варіант – це поєднання Blueprints та C++.

Рис. 2.3 Інтегроване середовище розробки

Наступним додатком буде Blender. Також достатньо потужний додаток в своїй справі. В Blender можна створювати все, що ви бачите в грі, але найкраще використовувати Blender для редагування якихось об'єктів. В моїй практиці я часто використовував Blender для дрібного редагування (десь щось вирізати, додати якийсь елемент до персонажу аби змінити колір чогось) та змінити формат файлу(на fbx), який підходить для Unreal Engine.

Epic Games Launcher, двома словами - бібліотека всього. Якщо проект розробляється невеликою командою аби взагалі самотужки, то Epic Games Launcher це просто бібліотека всіх робочих матеріалів, які обов'язково треба для розробки. Там ви зможете знайти як безкоштовні матеріали так і платні. Платного контенту звичайно в рази більше, але для новачка і безкоштовного матеріалу буде більше чим достатньо для створення першого проекту.

Завдяки Epic Games Launcher геймдизайнери і не тільки можуть продавати свої роботи іншим людям, тим самим заробляти та створювати більше можливостей та простору в ідея для створення нових ігрових продуктів. Це по справжньому велика бібліотека всього чого треба для розробки ігор, а саме: персонажі, анімації, музичне супроводження, різних предметів та навіть повністю готових ігрових світів.

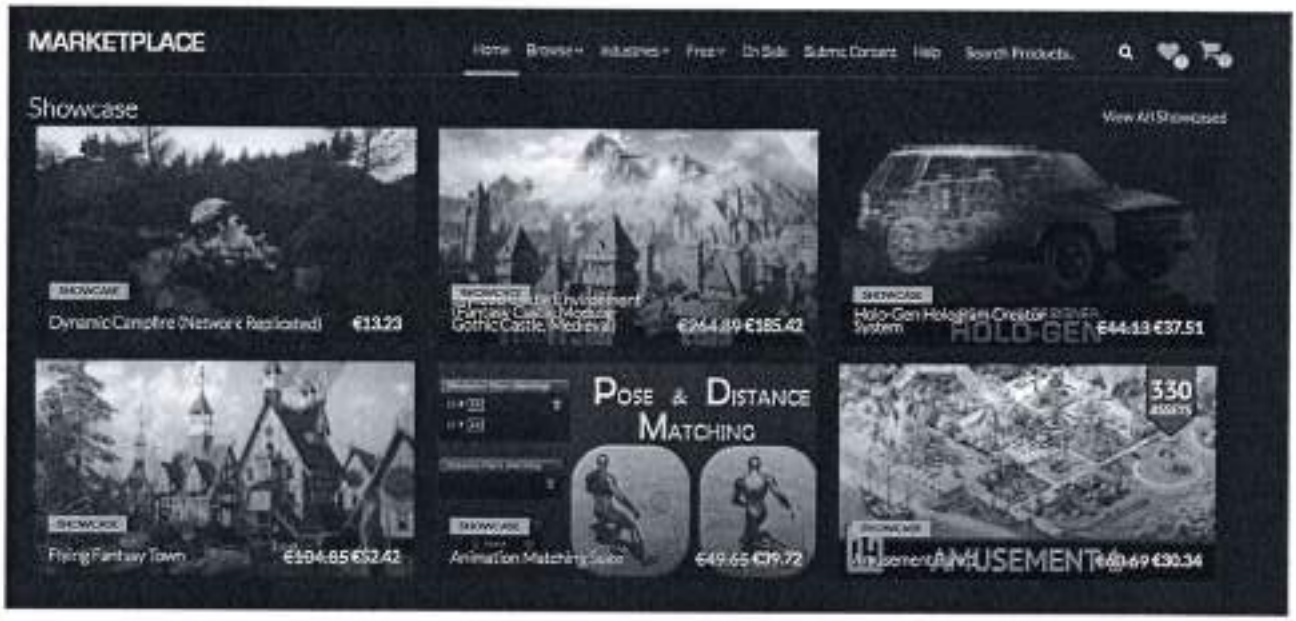

Рис 2.4. Торговий майданчик

В магазині Epic Games Launcher продаються плагіни, які можуть взагальному покращити швидкість та якісь розробки проекту.

Рис. 2.5. Приклад плагінів на торговому майданчику

РОЗДІЛ 3 ОПИС СТРУКТУРИ ГРИ, ХАРАКТЕРИСТИКА РОЗРОБЛЕНИХ ПЕВНИХ МОДУЛІВ ГРИ

3.1 Базова функціональна частина програми

Перше та найголовніше про що треба попередити коли йдеться мова про розробку гри на двигуні Unreal Engine, це те, що все будується на успадкуванні. Про це далі буде йти мова, але якщо коротко то розробниками двигуна були створенні батьківськи класи, які вже в собі мають базовий потрібний нам функціонал. Тому при створенні класу ми просто унаслідуємось від потрібного нам класу і переймаєм - те що нам потрібно.

Почнемо з функціональної частини.

Вся програма побудована на клас, які між собою дуже сильно пов'язані та іноді навіть залежать одне від одного. Якщо виникає якась проблема в одному класі, то з великою долею вірогідності виникнуть проблеми і в інших класах, які унаслідковуються від цього класу або мають певну взаємодію.

Для того аби почати розбір класів створених мною, варто дізнатися, що було створено розробниками двигуна для його користувачів. Тому що, без цієї бази буде важко зрозуміти для чого були створені мої класи та від чого я унаслідковувався.

1. Actor – це базовий клас для всіх об'єктів, які можуть бути розміщені на сцені, таких як персонажі, об'єкти навколишнього середовища, світло, камери тощо. Все що є на малюнку – це предмети, які походять від Actor.

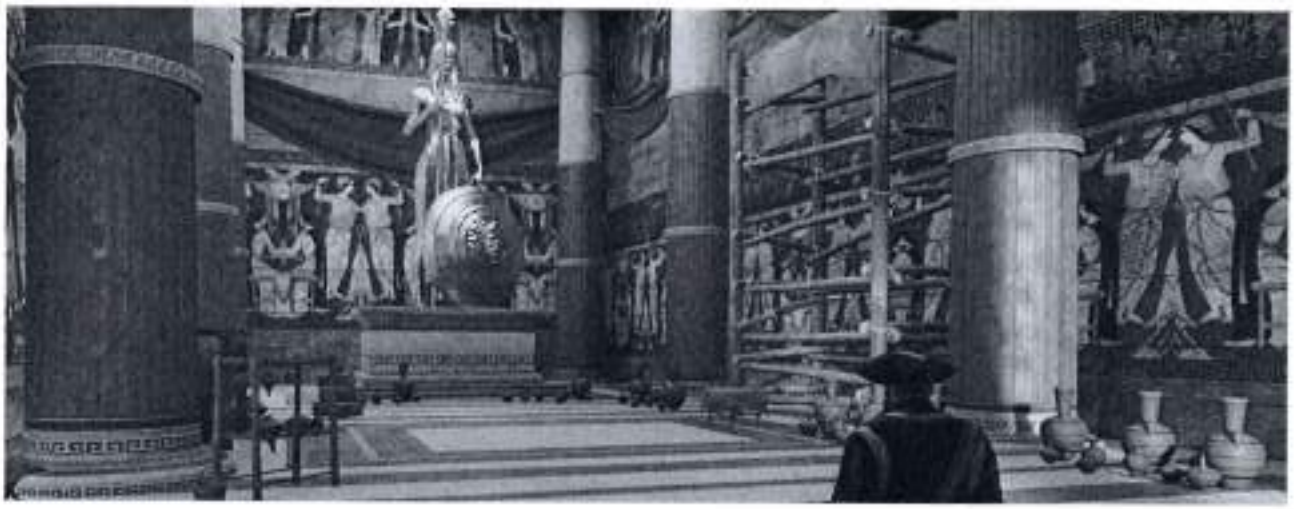

Рис 3.1. Приклад об’єктів на сцені

2. Pawn - є базовим класом для об'єктів, які можуть бути керовані гравцем або штучним інтелектом. Pawn може представляти різні ігрові об'єкти, такі як персонажі, транспортні засоби або будь-які інші об'єкти, які можуть рухатися по сцені гри. Він має можливість обробляти введення від гравця та взаємодіяти з іншими об'єктами у грі.

Легкий та звичайний приклад, ми керуємо нашим головним персонажем через це клас. Звісно не на прямо а через дочірні класи, які унаслідковуються від класу Pawn, але він є головним в цьому процесі. На малюнку нижче видно, як завдяки нашому роботі класу, ми можемо керувати нашим головним персонажем, в даному випадку стрибати. Там де треба якесь управління об’єктами на сцені або якийсь функціонал, потрібно використовувати даний клас.

Рис 3.2. Приклад роботи класу Pawn

Pawn володіє базовими характеристиками руху, такими як рух по землі, приземлення, стрибки, а також може мати властивості, специфічні для конкретного типу об'єкта, який він представляє. Наприклад, Pawn може мати можливість стріляти, літати, керувати швидкістю, машиною, вибірними характеристиками або будь-якими іншими унікальними властивостями, що залежать від конкретного типу об'єкта.

Pawn може бути керований гравцем за допомогою PlayerController, або штучним інтелектом за допомогою AIController.

3. Character та Player Controller. Схожі між собою, тому що працюють над персонажем. Character похідний від класу Pawn, призначений для роботи з персонажами, які мають власну анімацію та контроль над рухом. PlayerController – це клас, що відповідає за керування гравцем в грі, обробку введення та взаємодію з іншими системами. Player Contoller похідний від класу Pawn. Тому для того аби наш персонаж не був просто пам'ятником, який не рухається – треба йому додати можливість руху. Саме для цього і підходе наш Player Controller, велика частина функціоналу персонажа відбувається через Player Controller. На малюнку знизу приклад використання цього класу в нашому проекті.

Рис 3.3. Приклад використання класу Player Controller в коді

Прийнято, щоб головний персонаж унаслідковувався від класу Character, тому що заздалегість в цьому класі вже вкладено великий функціонал. На малюнки нижче показано.

```
22    UCLASS()
23  class MYPROJECT2_API AMainCharacter : public ACharacter
24    {
25        GENERATED_BODY()
26
27
28        /** MappingContext */
29        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
30        UInputMappingContext* DefaultMappingContext;
31
32        /** Jump Input Action */
33        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
34        UInputAction* JumpAction;
35
36        /** Move Input Action */
37        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
38        UInputAction* MoveAction;
```

Рис 3.4. Приклад унаслідування головного персонажа від класу Character

Використовування цих двох класів просто обов'язкове в розробці будь-якого проекту, тому що без них просто все буде важче та довше.

4. Клас GameModeBase є базовим класом для визначення правил гри. GameModeBase встановлює параметри гри, такі як початкові умови, цілі та способи завершення гри. Він відповідає за ініціалізацію гри, обробку подій гри та управління загальними аспектами гри.

Цей клас є одним з найважливіших, тому що жодний проект без цього класу існувати не зможе. В ньому прописується банально умови, при якій гра закінчується. Всі правила та все найголовніше з геймплею робиться саме в цьому класі. Якщо ціль створити гру, з великим та цікавим функціоналом, правилами гри та цікавою механікою, тоді без цього класу ніяк. Водночас, малі недопрацювання в цьому класі, можуть призвести до великих збоїв в програмі. Наприклад, якщо по умові гра повинна закінчуватись коли у персонажа не буде зовсім здоров'я. Ви зробили тип даних int для шкали здоров'я і через це система показує вам що у вас 0 здоров'я, хоча на справді у вас 0.4 здоров'я. І через це користувач гри не буде розуміти справжню картину гри, та геймод може працювати невідповідно. Цей клас по справжньому важливий та великий, який дуже сильно співпрацює з іншими не менш важливими класами, які максимально сильно доповнюють одне одного. На малюнку нижче можете побачити, як вони пов'язані між собою

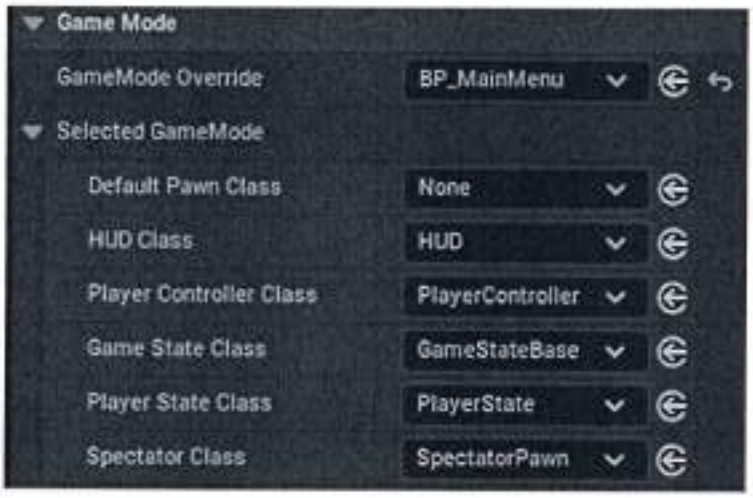

Рис. 3.5. Приклад можливостей класу Game Mode

GameModeBase може бути розширений та налаштований розробником гри згідно з потребами конкретної гри. Він визначає основний контекст і правила гри, що робить його ключовим компонентом для будь-якого проекту в Unreal Engine.

Почнемо опис функціоналу мого проекту.

Майже кожний клас, який був створений мною унаслідувався від якогось класу, який був створений до мене.

Розглянемо мого головного персонажа. Він був створений від класу Character.

```
22  UCLASS()
23  class MYPROJECT2_API AMainCharacter : public ACharacter
24  {
25      GENERATED_BODY()
```

Рис 3.6. Приклад створення класу головного героя

Це було зроблено аби перейнятий певний функціонал від батьківського класу. Можливо наступні персонажи будуть унаслідковуватись вже не від класу Character, а від класу AMainCharacter.

В класі AMainCharacter було реалізовано абсолютно всі дії, які відбувається на персонажем, а саме:

1. Було додано багато властивостей, які можуть бути змінені як у редакторі, так і під час виконання гри

```
28      /** MappingContext */
29      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
30      UInputMappingContext* DefaultMappingContext;
31
32      /** Jump Input Action */
33      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
34      UInputAction* JumpAction;
35
36      /** Move Input Action */
37      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
38      UInputAction* MoveAction;
39
40      /** Look Input Action */
41      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
42      UInputAction* LookAction;
43
44      /** Sprint Input Action */
45      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
46      UInputAction* SprintAction;
47
48      /** Heal Input Action */
49      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
50      UInputAction* HealAction;
51
52      /** Damaging Input Action */
53      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
54      UInputAction* DamagingtAction;
55
56      /** Damaging Input Action */
57      UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
58      UInputAction* DanceAction;
```

Рис 3.7. UPROPERTY властивості головного персонажа

В загалом, це було додано, щоб в самому редакторі гри додати можливості руху, стрибку, нанесення урону та лікування, можливості швидкого бігу та танці і так далі. Це все самостійно працювати не буде а тільки при співпраці з системою Enhanced Input System. Це новітня розробка від програмістів двигуна, для більш гнучкого біндінгу клавіш.

2. Реалізація біндінгу клавіш. На даному малюнки показана реалізацію біндінгу клавіш. Що взагалі таке “Біндінг клавіш”, біндінг клавіш - це процес призначення або пов'язання конкретної клавіші клавіатури або кнопки геймпада з певною дією або функцією в грі або програмі. Це дозволяє користувачеві керувати програмою або грою за допомогою

введення з клавіатури або геймпада. На малюнку 3.8. показано найновішу систему та спосіб біндінгу клавіш.

```
76  void AMainCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
77  {
78      // Set up action bindings
79      if (UEnhancedInputComponent* EnhancedInputComponent = Cast<UEnhancedInputComponent>(PlayerInputComponent)) {
80
81          // Jumping
82          EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Started, this, &ACharacter::Jump);
83          EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Completed, this, &ACharacter::StopJumping);
84
85          // Moving
86          EnhancedInputComponent->BindAction(MoveAction, ETriggerEvent::Triggered, this, &AMainCharacter::Move);
87
88          // Looking
89          EnhancedInputComponent->BindAction(LookAction, ETriggerEvent::Triggered, this, &AMainCharacter::Look);
90
91          // Sprint
92          EnhancedInputComponent->BindAction(SprintAction, ETriggerEvent::Started, this, &AMainCharacter::Sprint);
93          EnhancedInputComponent->BindAction(SprintAction, ETriggerEvent::Completed, this, &AMainCharacter::StopSprint);
94
95          // Heal
96          EnhancedInputComponent->BindAction(HealAction, ETriggerEvent::Started, this, &AMainCharacter::Heal);
97
98          //Damaging
99          EnhancedInputComponent->BindAction(DamagingtAction, ETriggerEvent::Started, this, &AMainCharacter::Damaging);
100         EnhancedInputComponent->BindAction(DanceAction, ETriggerEvent::Completed, this, &AMainCharacter::Dance);
101     }
102 }
```

Рис 3.8. Приклад використання Enhanced Input System

Як повідомили нас розробники цього двигуна, саме зараз варто переходити на цей спосіб, тому що в майбутніх версіях старий спосіб можуть просто прибрати. На малюнку позакано старий спосіб. Варто помітити, що в коді сильних змін немає, але в самому редакторі треба створити певні налаштування перед тим як почати прописувати код.

Щоб почати використовувати цю систему, попередньо треба в графічному редакторі створити Inpit Action Input(мал. 3.9.). Там і додаються певні налаштування для них, наприклад, клавіша буде працювати по якісь системі координат чи це просто бульове значення. Можливість поєднання двох клавіш в одну функцію, додавання тригерів та модифікаторів клавіши і так далі.

Рис 3.9. Варіанти Input Action

```
void ARPGCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    // Set up action bindings
    if (UEnhancedInputComponent* EnhancedInputComponent = Cast<UEnhancedInputComponent>(PlayerInputComponent)) {

        // Sprint
        PlayerInputComponent->BindAction("Sprint", IE_Pressed, this, &ARPGCharacter::Sprint);
        PlayerInputComponent->BindAction("Sprint", IE_Released, this, &ARPGCharacter::StopSprint);

        //Get Health
        PlayerInputComponent->BindAction("Heal", IE_Pressed, this, &ARPGCharacter::StartHealing);
        //Get Damage
        PlayerInputComponent->BindAction("Damage", IE_Pressed, this, &ARPGCharacter::StartDamage);
    }
    else
    {
        UE_LOG(LogTemplateCharacter, Error, TEXT("'%s' Failed to Find an Enhanced Input component! This template i
    }
}
```

Рис. 3.10. Приклад використання старої системи біндінгу клавіш

Далі мова буде йти про Game Mode Base. Це базовий геймод, в якому прописані всі правила та умови гри. На даному етапі розробки власної гри я не багато працював над Game Mode. На малюнку нижче показано як додається Bluepint нашого головного персонажа в наш клас. При запуску гри в налаштуваннях редактора можна вибирати який геймод буде запускатися, тому якщо ми вибираємо сааме цей геймод, тоді і одразу функціонал персонажа також буде запускатися. Бо разом з геймодом запускається Blueprint нашого персонажу.

```
#include "MyGameModeBase.h"

AMyGameModeBase::AMyGameModeBase() : Super()
{
    ConstructorHelpers::FClassFinder<APawn>MainCharacterPirate(TEXT("/Game/Characters/MainCharacter/Blueprints/MyMainCharacter"));
    DefaultPawnClass = MainCharacterPirate.Class;
    //  HUDClass = ACharacterHUD::StaticClass();
}
```

Рис. 3.11. Підключення Blueprint головного персонажа до Game Mode

Наступний етап, Blueprints - це візуальна система програмування в рамках ігрового движка Unreal Engine, яка дозволяє розробникам створювати геймплей, інтерфейси користувачів, логіку штучного інтелекту та багато іншого без потреби в кодуванні.

Одним з найголовніших файлі цього проекту є Blueprint головного персонажа. В ньому прописано багато функціоналу, додані анімації, багато різних налаштувань і так далі. Наприклад, на малюнку показано як зробити так, аби при натисканні клавіші 1 наш головний персонаж починав танцювати потрібний нам танець.

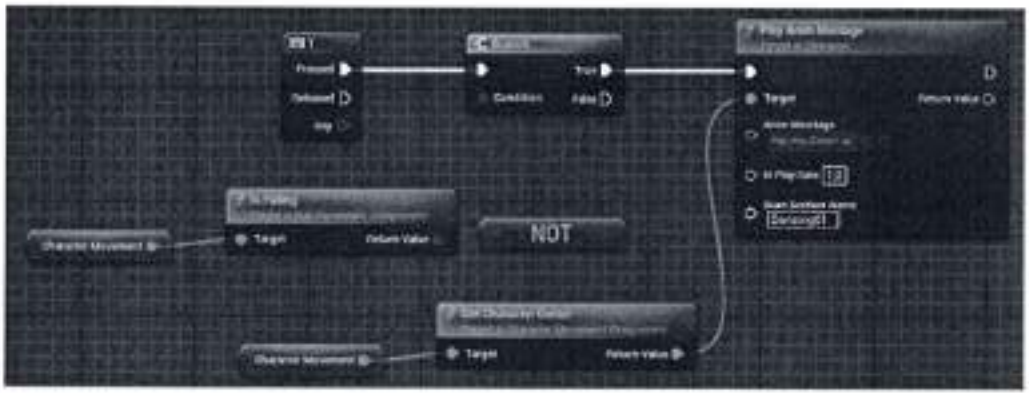

Рис. 3.12. Приклад функціоналу танцю

На малюнку нижче показано, як додати музику на задній фон під час основної гри.

Рис. 3.13. Приклад функціоналу додавання музики на задній фон гри

Blueprint надає зручний спосіб реалізації взаємодії з іншими елементами Unreal Engine, такими як об'єкти, персонажі, інтерфейси користувача, анімація та багато іншого. Він також дозволяє швидко прототипувати ідеї та проводити тестування без необхідності компіляції коду. Через Blueprint ми дуже легко та зручно можемо змінювати певні ігрові параметри, як на малюнку нижче. Це звісно малесенька частинка з всіх параметрів. По малюнку ми бачимо, що ми може змінити певні матеріои, наприклад нам не подобається забарвлення одяну і в нас є інший варіант для одяну, тому справа в два кліки мишкою ми можемо змінити цей параметр. Або нам не подобається теперешнє положення камери і ми хочемо його кудись змісти. Наприклад, ми підготували декілька варіантів Anim Class і можемо змінювати їх абсолютно з легкістю. Все теж саме ми можемо зробити в коді але навіщо, якщо це можна зробити за одну мить та з

легкістю. Базово, ми можемо змінюватии Mesh персонажів, якщо тестувати, який персонаж краще, якому більше підходить певний Anim Class і так далі.

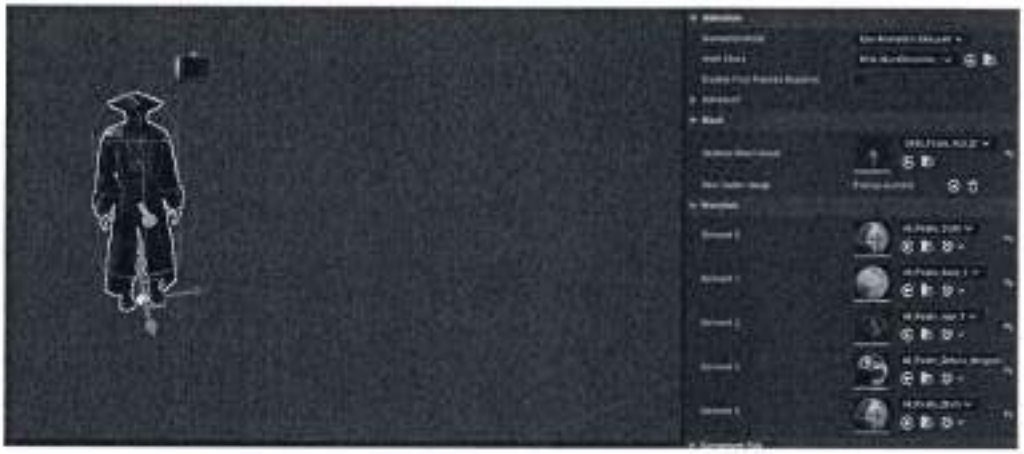

Рис. 3.14. Частина вікна Blueprint головного персонажа

В Blueprint дуже багато різних панелей, які полегшують та дають більш гнучкі варіанти вирішення певних задач. Приклади, на малюнках нижче.

Рис. 3.15. Приклад панель Blueprint

Рис. 3.16. Приклад іншої панелі Blueprint

3.2 Продвинута функціональна частина програми

3.2.1 Система зброї

Продвинута функціональна частина програми складається з багатьох модулів програмної частини. В цьому розділі буде розповідатися про те, як створити базову систему зброї, систему ворога та базовий ігровий режим. Завдяки продвинутій функціональній частині програми можливості ігрового процесу набувають кращих результатів. Додатковий функціонал – це цікаві

механіки гри, в свою чергу, якщо певна гра має власні унікальні механіки гри – то вона набуває унікальних рис.

У іграх система зброї є важливим аспектом геймплею, що значно впливає на загальне враження від гри. Майже кожна гра має систему зброї. Вона може включати різні види зброї, їх характеристики, механіку використання, систему прокачування та інші елементи.

Системи зброї в іграх різноманітні і можуть бути дуже інноваційними. Вони значно впливають на геймплей та досвід гравця, додаючи глибині та унікальності кожному проекту. Геймдизайнери постійно шукають нові способи зробити зброю цікавою та інтерактивною, що веде до постійного розвитку ігрової індустрії. Особливо чутливий розвиток системи зброї відбувся в двох напрямках, це реалізм та фентезі.

Розвиток реалізму в системах зброї у відеоіграх є складним і багатогранним процесом, який включає багато аспектів, таких як фізика, анімація, звук, механіка стрільби та моделювання пошкоджень. Кожен постріл повинен мати реалістичний ефект на приціл, щоб змушувати гравців контролювати віддачу і враховувати відстань до цілі. Також балістика, це один з ключових аспектів коли мова йде про реалістичність зброї. Потім фізика, фізика не тільки зброї, а і простору, руху персонажу, наприклад при швидкому бізі віддачі від зброї повинна бути більше чим від статичної пози персонажу, тому що при бізі звичайна людина менш контролює віддачу зброї чим від статичної пози.

У фентезійних іграх система зброї є не менш важливою, ніж у реалістичних шутерах, але її особливості часто відрізняються через включення магічних елементів, фантастичних істот та незвичайних матеріалів. В фентезі жанрі геймдизайнер може вже використовувати свою творчість, тому що немає жодних обмежень або правил, чого не можна сказати про реалізм. В цьому жанрі зброї зазвичай більший потенціал для розвитку, тому що немає обмежень і можна

придумувати абсолютно нові та різні ігрові механіки. Найголовніша відміність та можливо перевага від реалізму, це відсутність обмежень у фантзаії. Наприклад у грі може використовуватись зброя, що може змінювати свою форму під час бою. Це додає додатковий рівень тактичної варіативності, дозволяючи гравцям адаптуватися до різних ситуацій та ворогів.

В моєму проекті я зробив ставку на комбінування. Тобто в проекті присутній модулі з обох жанрів. Головний персонаж використовує реалістичні види зброї а його вороги більше до фантазійного жанру. Для того будь яка зброя мала якусь логіку – для цього треба створювати Blueprint класи. Вони нам надають можливість якісно інтегрувати зброю в експлуатація певним персонажем. На малюнку нижче видно як саме виглядають Blueprint класи різних об'єктів.

Рис. 3.17. Зображення Blueprint класів різних видів зброї

Тепер розглянемо об'єкти детальніше. Кожний з наших Blueprint має базові налаштування. Ці налаштування ми вже були попередньо визначенні. Ми бачимо, що зброя має такі змінні, як: Назва зброї, урон, розмір магазину, назві кістки персонажу до якої в майбутньому вона буде прикріплена і так далі.

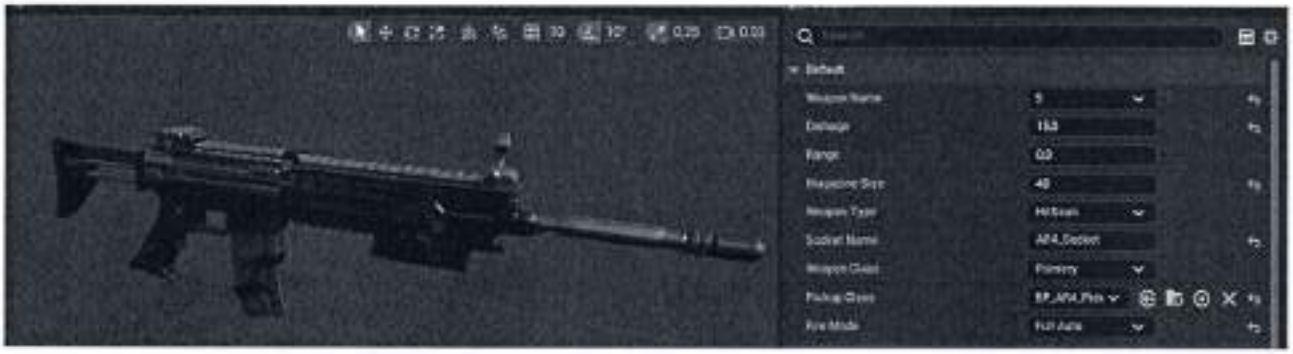

Рис. 3.18. Базові змінні класів

Для того аби краще реалізувати реалізм, важливо аби зброя мала анімацію, анімацію пострілу і так далі. Щоб анімація програвалась я треба підключити в нашому Blueprint. Для цього треба перейти в Event Graph та створити базову логіку, завдяки нотам які існують в середині. На малюнку нижче показано детально базову логіку реалізації підключення анімації до зброї.

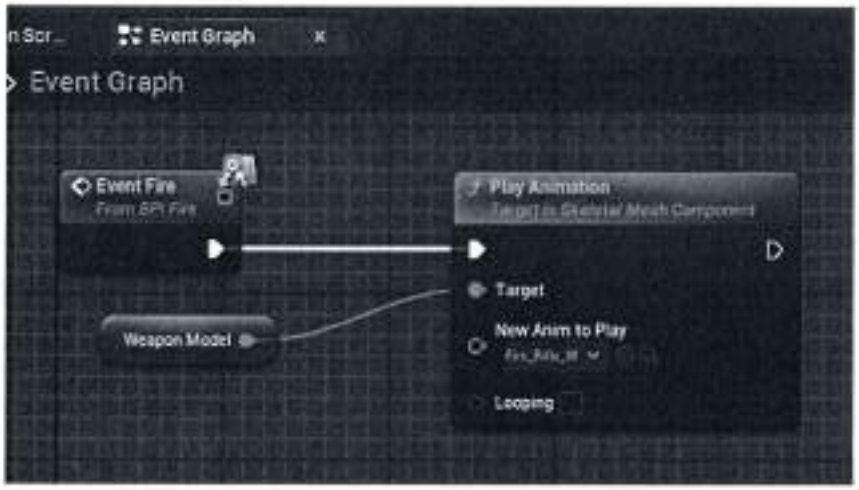

Рис. 3.19. Підключення анімацій до зброї

Ми використали Event Fire, тобто коли веде вогонь то тоді відіграється анімація. В середині ноди Play Animation ми вибираємо яку саме анімацію та до якої конкретно моделі зброї. Такий процес має кожна зброя.

В кожній системі зброї повинні бути батьківські класи від яких далі піде успадкування. Цей клас зазвичай називають BP_WeaponMaster. "BP" на початку пишеться, аби помітити що це саме Blueprint клас а не якийсь інший. Об'єктами цього класу якраз і є вже якась конкретна зброя. Це дуже зручний підхід створення системи озброєння, тому що таким чином ми створюємо одну змінну одразу для декількох видів озброєння. Тим самим ми використовуємо менше пам'яті та наша система становиться більш організованою та структурованою.

Але збалансована та добре спроектована система зброї не складається тільки з одних Blueprint класів. Також важливу роль відіграють перечислення

(Enum). Перечислення може допомогти структурувати та організувати різні типи зброї, рівні майстерності та інші пов'язані категорії. Звісно можна понастворювати змін але найкращий варіант залишається за створення списків для конкретних цілей, наприклад:

На малюнку знизу список назв зброї. Кожний елемент списку має власну унікальну назву.

Рис. 3.20. Дані з списку

На малюнку знизу список класів зброї. В нашій ігровій системі є два типу озброєння, основна зброя та додаткова.

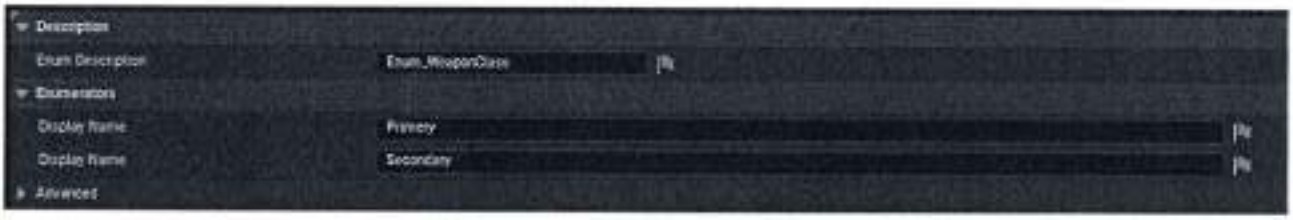

Рис. 3.21. Дані з іншого списку

3.2.1.1 Система підбирання зброї

Система підбирання зброї – це механіка в ігрі, коли гравець може збирати або підбирати різні види зброї, розташовані на рівні чи в грі загалом. Ця система може мати різні варіації залежно від типу гри та її механік. Система підбору зброї додає до ігрового процесу глибину і стратегічність, а також може стимулювати гравця до дослідження гри та використання різних видів зброї в залежності від ситуації. Система підбору зброї має відносно тіж самі базові функції що і Weapon Master. Це той самий батьківський клас від якого буде йти наслідування об'єктів,

а саме вже конкретних видів озброєння. На відміну від від Weapon Master система підбору зброї має вже розроблену логіку. На малюнку нижче продемонстровано логіку підбирання зброї з рівня.

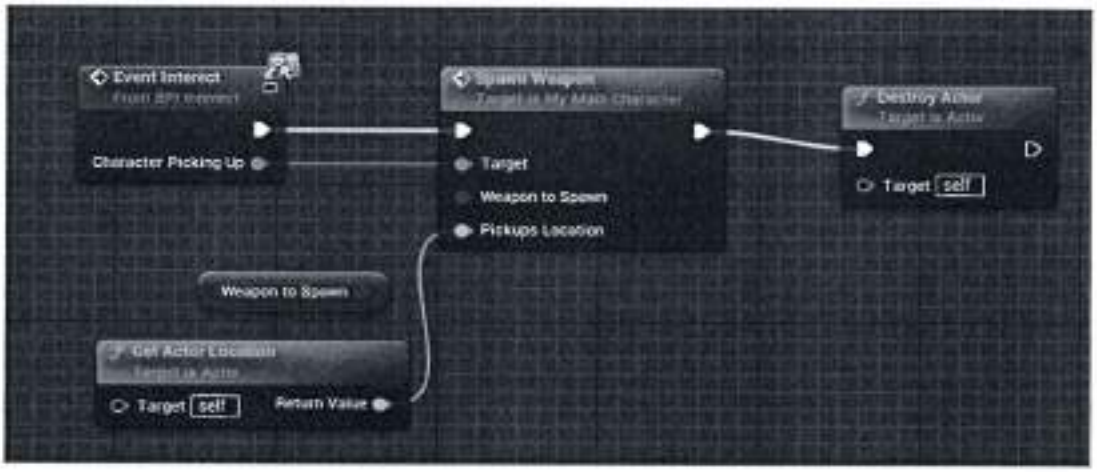

Рис. 3.22. Логіка підбирання зброї

Також наш Blueprint має одну цікаву функцію. Для того аби ми змогли підібрати зброю – нам треба зайти в радіус зброї, в якому ми зможемо підібрати її. Цей радіус треба власноруч створювати, для цього нам потрібно спочатку додати Staick Mesh (на фото нижче перейменовано на PickupObject) а потім вже саме до цього об'єкту додати Sphere Collision. Тобто наш радіус це розміри сфери(кола жовтим кольором) а колізія працює як датчик в нашому випадку. Коли персонаж перетинаю колізію, тоді відпрацьовує потрібна нам логіка. На малюнку нижче все продемонстровано.

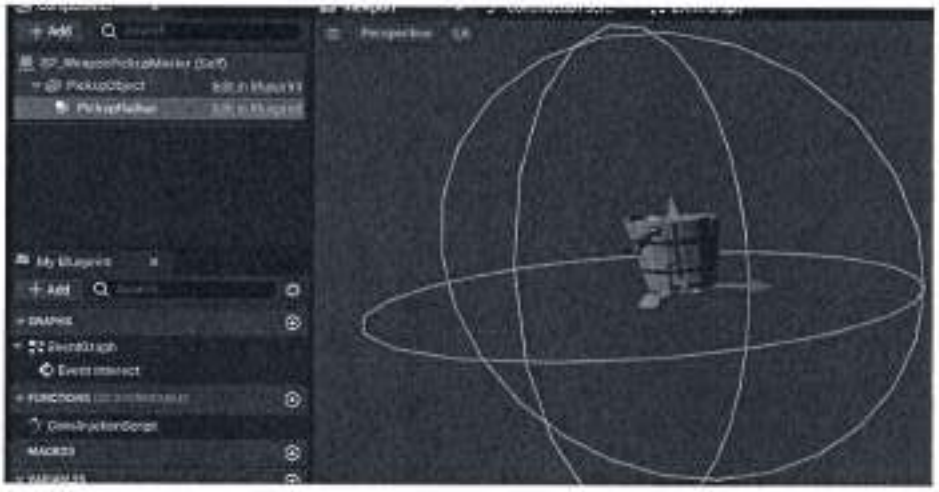

Рис. 3.23. Радіус підбирання

Так як це батьківський клас, значить в нього є об'єкти. Об'єкти цього класу це вже і є зброя, яку ми будемо використовувати протягом гри. Персонаж може взяти зброю якщо натисне клавішу "F", це означає що ми попердньо забіндили цю клавішу та створили базову логіку, завдяки які персонаж бере зброю та додає собі в інвентар. В залежності від класу зброї (класи зброї визначення в перечисленні Enum_WeaponClass), зброя займе конкретне місце в інвентарі. Наприклад якщо це основна зброя, то при натисканні клавіші "1" – персонаж візьми в руки основну зброю, якщо "2" – додаткову зброю. Цій функціональності не обійти без однієї додаткової змінної, яка і приймає значення з перечислення. Ця змінна має назву Equip Index, тобто індекс підібраного предмету. Ми бачимо, що логіка далі ділиться на дві частини, в залежності яке значення має змінна Equip Index.

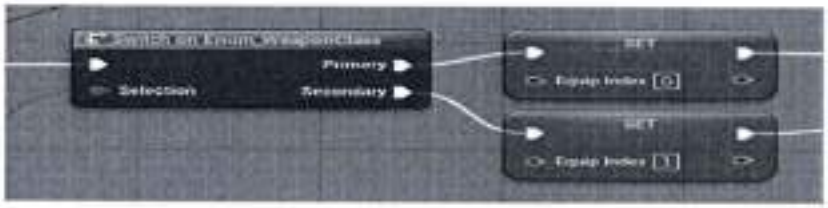

Рис. 3.24. Індекси видів озброєня

Варто додати, що ця система повинна бути достатньо універсальна та мати хорошу механіку, тому що зазвичай в іграх дуже багато об'єктів з яким персонаж повинен мати змогу працювати, кожний об'єкт має свої дані, доступи і так далі, тому ця механіка повинна бути на високому рівні розробки.

3.2.1.2 Створення зброї

Для того аби зброя була на карті, для початку її треба там зробити, тобто заспавнити. Ця механіка використовується в іграх, коли гравцеві дозволяється створювати або отримувати зброю у відведеному місці в грі або на рівні. Це може бути корисною функцією для тестування різних типів зброї, для розвитку гри або

для створення спеціальних сценаріїв. В загалом логіка цієї механіки досить велика, тому що треба вказувати багато параметрів того, що буде створено(заспавнено). Наприклад як було раніше сказано, що в залежності від Equip Index буде зрозуміло де конкретно в інвентарі буде знаходитись конкретний варіант зброї. Також від цієї змінної буде залежати яка конкретно зброя буде створено. Розробник попередньо визначає місця на карті де буде створена зброя. Можна зробити так, аби зброя випадково вибирала місце на карті, насправді дуже багато можливостей з цією механікою. На картинці знизу видно результат роботи створення зброї. В конкретному місці була зроблена конкретна зброя, в даному випадку вона доповнює сценарій гри. Завдяки цій механіці ми додатково можемо встановити логіку прикріплювання зброї до кістки персонажу. Ми можемо змінювати положення зброї в руці. Також можна кожну зброю персоналізувати під персонажа. Бувають моменти коли зброя залазить в меш персонажа, тому якщо правильно поставити зброю в руку та налаштувати під конкретні анімації – тоді все буде виглядає правильно.

Рис. 3.25. Результат роботи створення зброї

3.2.1.3 Система вогню

Система вогню з зброї - це комплексний механізм, який відповідає за моделювання вогневих ефектів, які виникають при використанні вогнепальної зброї у грі. Вона включає в себе реалістичне відтворення руху куль, пошкоджень, віддачі, а також вплив на оточуюче середовище та цілі.

Система повинна розраховувати траєкторію кулі від моменту пострілу до попадання у ціль або у поверхню. Це включає врахування фізичних законів, таких як гравітація, повітряний опір та вітер. Для цього існує така функція як Line Trace, тобто лінія, яка імітує траєкторію пуль. Якщо ці лінія перетинає якийсь об'єкт, то при бажанні з цим можна створити логіку нанесення пошкоджень. На малюнку нижче продемонстровано механіка Line Trace. Ми бачимо що вона достатньо не малесенька але є основні аспекти, які варто розповісти. Траєкторія вогню повинна брати свій початок, тому цей початок встановлюємо ми таким чином, на скелеті зброї ми створюємо додаткову кісту на називаємо її "Barrol". Це початкова точка виходу вогню з нашої зброї. В редакторі скелетів ми задаємо правильну позицію цієї кістки та правильний нахил, щоб струя вогню виходила рівна з дула зброї та йшла по траєкторії так, як нам треба. Для опрацювання вище сказаної інформацію на малюнку видно, що для цього ми використали функції "Get Socket Location" та "Get Socket Rotation". Перша для місцезнаходження початкової точки, друга для нахилу траєкторії. Відстань траєкторія вказано у функції "Line Trace by Channel", параметрами Start та End.

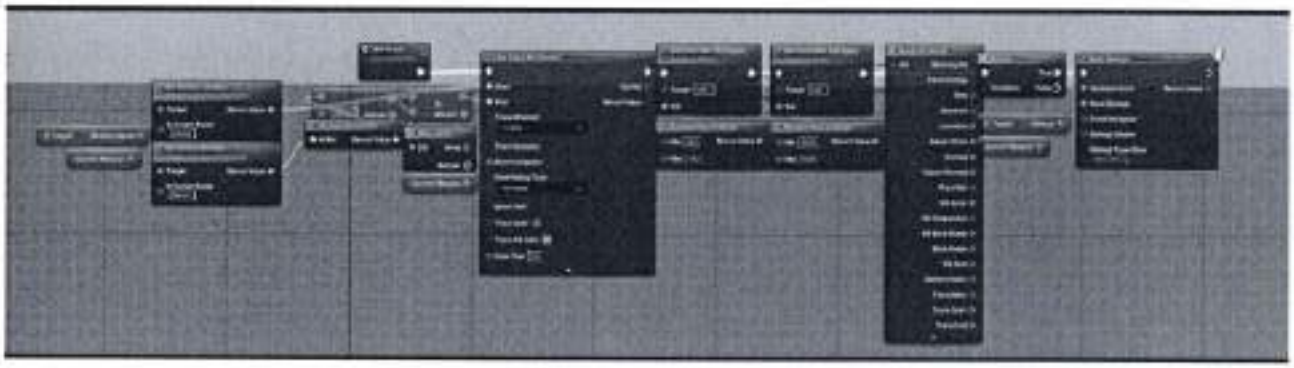

Рис. 3.26. Система вогню

Для створення реалістичного відчуття вогню з зброї система повинна відтворювати візуальні та звукові ефекти, такі як полум'я, дим, тріскіт палючих матеріалів та звук вогнепальних пострілів. Також для гарної візуалізація та покращення реалізму обов'язково треба додати віддачу від зброї та анімації роботи зброї. Найвищий рівень це коли анімації співпрацюють з окремим

деталями зброї, наприклад при повному перезарядженні зброї якась одна рука повинна прикріплюватись до магазину та в певний момент викидувати старий магазин та вставляти в зброю новий магазин. Якщо гра розробляється під платформу VR то там реалізм та професіоналізм на вищому рівні, тому під цю платформу анімації та робота з окремим деталями зброї звичайний процес. Під гру на комп'ютері можна це обходити, наприклад якщо правильно налаштувати камеру, тобто вибрати правильну локацію камери під час перезарядження, тоді зброя уходить з поля зору але треба, щоб анімації були доречні та в цьому випадку треба більш краще проробити звукову підтримку та інші схожі можливості. Важливою функцією системи вогню є перемикання режимів ведення вогню. Наприклад в кожної зброє власні специфічна налаштування, тому для такої механіки повинна бути можливість перемикатись між режимами. На малюнку нижче продемонстровано, як саме перемикається режим вогню в залежності від актуального виду озброєння.

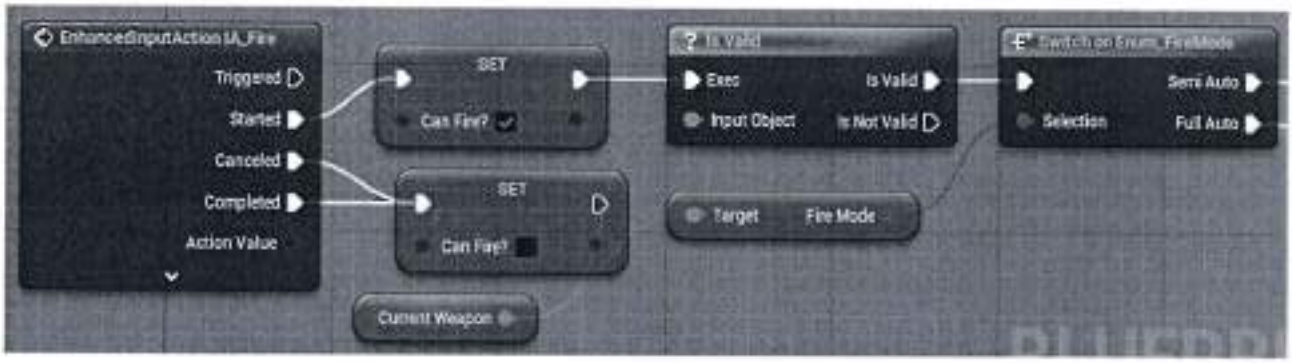

Рис. 3.27 Перемикання режиму вогню

Система повинна враховувати фактори, що впливають на точність пострілу, такі як стійкість гравця, тип зброї, відстань до цілі та інші, а також розсіювання кулі відносно точкової цілі.

3.2.1.4 Прицілювання

Прицілювання - це техніка в геймдевелопменті, яка використовується для забезпечення реалістичного прицілювання героя або персонажа в грі, коли гравець контролює рухи огляду (наприклад, мишкою або джойстиком). Це особливо важливо для ігор з вогнепальною зброєю, де точність стрільби залежить від того, наскільки точно гравець може прицілитися до цілі.

Основна ідея полягає в тому, що персонаж відповідно реагує на рухи огляду гравця, але з затримкою або корекцією, щоб забезпечити зручне та реалістичне прицілювання. Наприклад, якщо гравець прицілюється вгору, то персонаж може здійснити зміщення своєї голови або зміну позиції зброї відповідно до цього напрямку.

Існує багато варіантів як можна модифікувати цю механіку, наприклад якщо персонаж цілиться, то тоді можна зменшити швидкість повертання мишкою. Бувають ігрові моменти коли за сценарієм в певному місці швидкість руху персонажу зменшується, тоді буде логічно також зменшити швидкість миші.

Правильне налаштування цієї механіки дуже важливо, тому що це дуже впливає на комфорт гри. При неправильному налаштуванні може здаватися що мишка рухається відривками або буте незрозуміле трусіння прицілу, яке не дасть плавної картинки гри.

В розвинутих версіях цієї механіки розробники надають гравцям кастомізувати приціл особисто під себе. В загальному ця механіка потрібна, щоб компенсувати різницю між тим, як гравець прицілюється зі свого контролера або миші, і тим, як це відображається у рухах персонажа у грі. Або в деяких випадках навпаки, або максимально сильно ототожнити рухи контролера гравця.

В цій механіці є багато модулів, які треба створювати окремо одне від одного. Наприклад на малюнку нижче показано логіку, яка враховує пріоритети

виконання анімацій та умови при яких вони повинні відігратись.

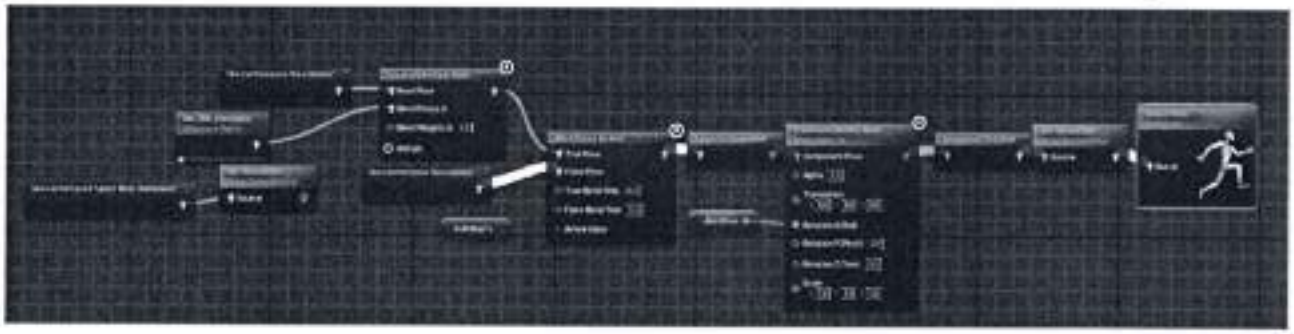

Рис. 3.28. Анімаційний Blend Space

Додатково на малюнку зверху продемонстровано рух частини тіла в напрямку за курсором гравця. Це додає більше реалістичності та плавності геймплею.

Знизу на малюнку продемонстровано, як відбувається зчитування двигуном математичних даних, які в результаті і прораховують в якому саме напрямку буде дивитись курсор. На фото видно функцію “Clamp (Float)”, вона потрібна для того аби встановити границі, по які наша мишка дійсна. Ця функція може значно на ігровий приціл додаючи різні варіації використання цієї механіки в грі. Тобто, як це працює конкретно, ми опускаємо приціл максимально вниз але він опускається до певного рівня, а камера персонажа не має таких обмежень. Тому зброя залигається на певному рівні а камера рухається далі, цим можна користуватись в певних ігрових моментах або навіть створюють сценарії гри, в яких використання цієї механіки є необхідністю.

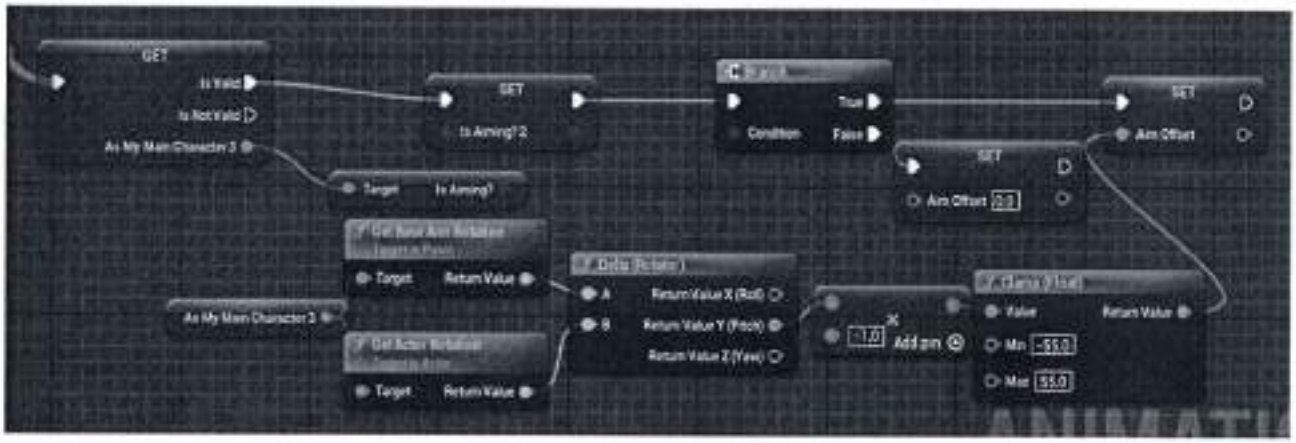

Рис. 3.29. Функціонал прицілювання

3.2.1.5 Віддача

Віддача - це фізичний ефект, що виникає при стрільбі з вогнепальної зброї. Коли куля вистрілюється, вона відштовхує зброю в протилежному напрямку від напрямку польоту кулі. Цей рух відбувається через вивільнення газів від пострілу та дії рухомих частин зброї. Це у випадку якщо гра націлена на реалізм. Зазвичай віддачу намагаються кастомізувати, тому що це дуже впливає на візуальну частину. Гравцями вважається, чим більше віддача - тим більше урон. Тому віддачу можна використовувати в різних цілях. Також в житті віддача не завжди виглядає так добре як в іграх. Бо на першому місці в розробників мета аби користувачу сподобалось.

Віддача може бути відчутною для стрілка і впливає на точність стрільби. Велика віддача може призвести до того, що гравець втратить контроль над зброєю, що ускладнює стрільбу на великі відстані або при стрільбі серіями пострілів. Розробники намагаються додавати віддачу де тільки можна, аби грати ставало важче, щоб у людей було більше захоплення грати, бо коли занадто легко то це нікому не цікаво.

У більш реалістичних іграх віддача зброї може бути моделювана відповідно до типу зброї, її калібру, використання аксесуарів для зменшення віддачі (наприклад, стабілізаторів), а також стану та навичок стрілка.

У багатьох іграх віддача є важливим геймплейним елементом, оскільки вона впливає на стратегію і техніку гравця. Гравці можуть навчитися контролювати віддачу, використовуючи різні методи, такі як короткі стрільби, регулювання позиції зброї та використання спеціальних аксесуарів.

При створенні механіки віддачи, зазвичай віддачу додають до правої руки. В розвинутих механіках віддача поширюється на праву руку та дві кості на спині, аби максимально плавно відігравалась анімація віддачі.

3.2.2 Вороги

Система ворогів у грі — це комплекс механізмів, які визначають поведінку, взаємодію, характеристики та зовнішній вигляд ворожих об'єктів у грі. Система ворогів у грі включає в себе дуже багато аспектів, іноді навіть більше чим головний персонаж. Насправді розробка головного персонажа звичайно відрізняється від розробки ворогів персонажа, але не дуже сильно. Тому що за основу беруться ті ж самі батьківськи класи, на перших етапах процес створення об'єктів ворогів майже такий самий, як процес створення головних персонажів. Основні елементи системи ворогів включають: штучний інтелект, анімації, взаємодія з гравцями, створення ворогів і так далі.

Створення великої кількості ворогів одночасно може вплинути на продуктивність гри, тому завжди необхідно використовувати техніки оптимізації, такі як обмеження зони спавну, зменшення деталізації моделей ворогів, які знаходяться далеко від гравця, та інші. Створення ворогів повинно бути обдумане заздалегідь, щоб зменшити навантаження на гру. Для цього треба використовувати фіксовані точки спавну, це створення ворогів у заздалегідь визначених місцях на рівні. Це підходить для рівнів з фіксованою структурою та передбачуваним сценарієм. Наприклад, в платформерах або в шутерах від першої особи з попередньо визначеними хвилями ворогів. Через те, що ми знаємо скільки конкретно точок створення на карті, ми знаємо приблизну кількість навантаження на комп'ютер. Існує також динамічний спавн, цей метод передбачає створення ворогів у відповідь на певні умови, такі як місцезнаходження гравця, прогрес в грі або час. Це робить гру більш непередбачуваною і динамічною. Цей метод більше навантажує систему чим попередній, тому що ми не можемо точно контролювати точну кількість ворогів.Реалізація системи спавну ворогів є важливою частиною розробки ігрового процесу. Використовуючи різні методи спавну, такі як фіксовані точки,

тригери та динамічний спавн, можна створити цікаві та різноманітні ігрові ситуації.

3.2.2.3 ШІ Ворогів

ШІ ворогів в грі може створювати враження живої та інтелектуальної поведінки. Відповідно до жанру гри, ШІ ворогів може мати різні функції, такі як переміщення по світу, пошук гравця, атаку та уникнення загроз. Для досягнення цих цілей нам знадобилося розробити маленьку систему штучного інтелекту яка буде включати в себе такі компоненти, як: навігація та переміщення, виявлення гравця, поведінка та прийняття рішень

Штучний інтелект ворога повинен вміти взаємодіяти з оточуючим світом, навігувати через нього та знаходити оптимальні дорогу до цілей. У Unreal Engine ми використали "NavMeshBoundsVolume"для генерації області роботи ворога, яка доступна для навігації. На малюнку нижче можна побачити як конкретно виглядає область роботи ворога. Також можна побачити як виглядає поле зору нашої моделі ворога. Додавання поля зору було створено в редакторі класу Blueprint, а саме через вбудований асет "PawnSensing". Ми можемо змінювати параметри цього асету, наприклад зменшити поле зору, або вказати конкретно, на що саме буде реагувати наш ворог, наприклад на звуки, конкретні об'єкти, частота реагування на об'єкт та швидкість реагування.

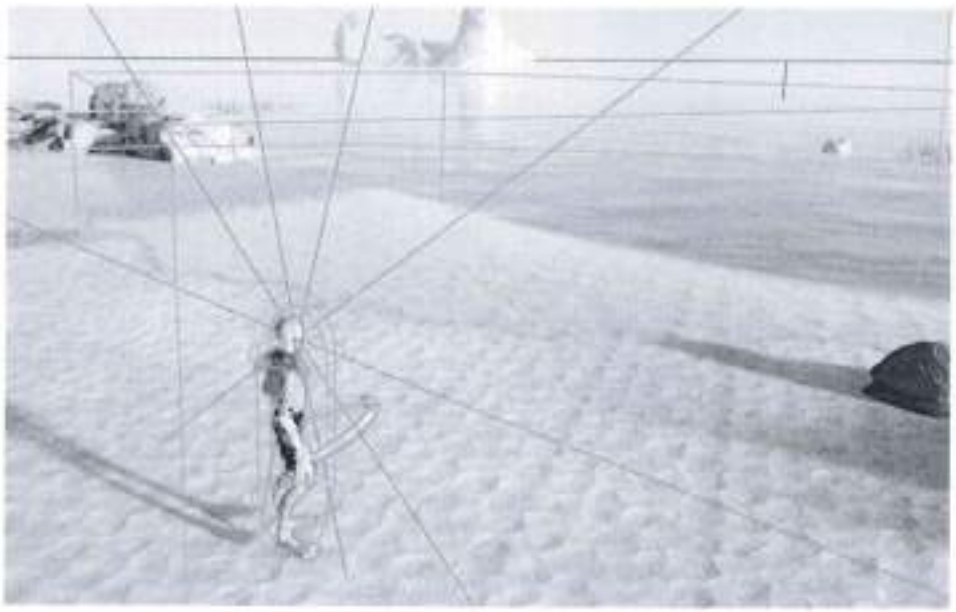

Рис. 3.30. Область роботи ворога

Поведінка ворога у випадку, якщо головний персонаж зайшов в область навігації ворога та ворог бачить нашого головного персонажа, реалізовується по різному. Але за основу береться реалізація функції "Follow Player" таким чином як на малюнку знизу. Тобто, якщо вище перечисленні умови відбулися, то ворог починає слідкувати за головним персонажем. У розробці цієї механіки ключову роль відіграє вбудована в двигун функція "AI Move To", а далі як параметри ми просто вказуємо, що хто буде виконувати цю функцію та хто ціль переслідування. Таким базовим чином реалізується функція переслідування головного персонажа при певних ігрових умов.

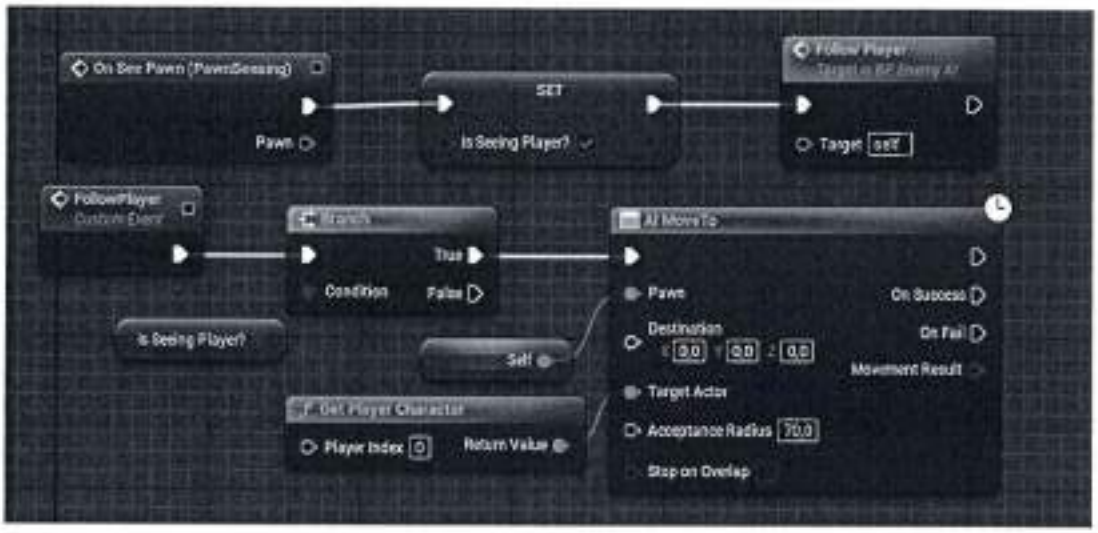

Рис. 3.31. Реалізація функції "Follow Player"

Для того аби поведінка всіх ворогів на рівні була відпрацьована належним чином, треба правильно розташовувати асет "NavMeshBoundsVolume". Тому що бувають випадки, коли в одній області декілька ворогів і при взаємодії з головним персонажем трапляються незрозумілі помилки. В таких випадках треба або дуже детально налаштовувати об'єкти ворогів, щоб вони не конфліктували одне з одним в одній зоні. Або просто поділити карту на відповіді

зони і щоб конкретна зона роботи належала конкретному об'єкту. В загалом це складний та творчий процес, що вимагає багато експериментів та налаштувань, щоб досягти бажаного результату.

3.2.2.3 Атака ворогів

Атака ворогів - важливий компонент штучного інтелекту в іграх, тому що вона забезпечує динамічність, захопливість і виклик для гравця. Розробка системи атаки ворогів в Unreal Engine включає кілька ключових етапів таких як, визначення типів атак, створення анімацій, детекцію зіткнень, розрахунок пошкоджень і реакцію гравця на атаку. Всі ці елементи мусять добре взаємодіяти між собою та бути добре налаштованими. Бо наприклад, якщо поставити звук удару на пів секунди раніше ніж сама анімація удару спрацює, то це буде виглядати дуже не професійно. Для того аби в загальному правильно працювала система атаки у ворога, головний персонаж повинен також мати певні налаштування аби співпрацювати з ворогом. Наприклад, у ворога все працює класно, є анімація, логіка атаки, логіка нанесення урону, правильно налаштована колізія зброї але в результаті ми бачимо що урон з головного персонажа після атаки ворога не знімається. Одна з причин цієї проблеми може бути якраз не співпрацьованість модулів двох персонажів. Якщо конкретно, то для того аби головний персонаж приймав урон, йому треба створити додатковий модуль програми, який буде відповідати за це. На малюнку нижче продемонстровано базу реалізацію цього функціоналу.

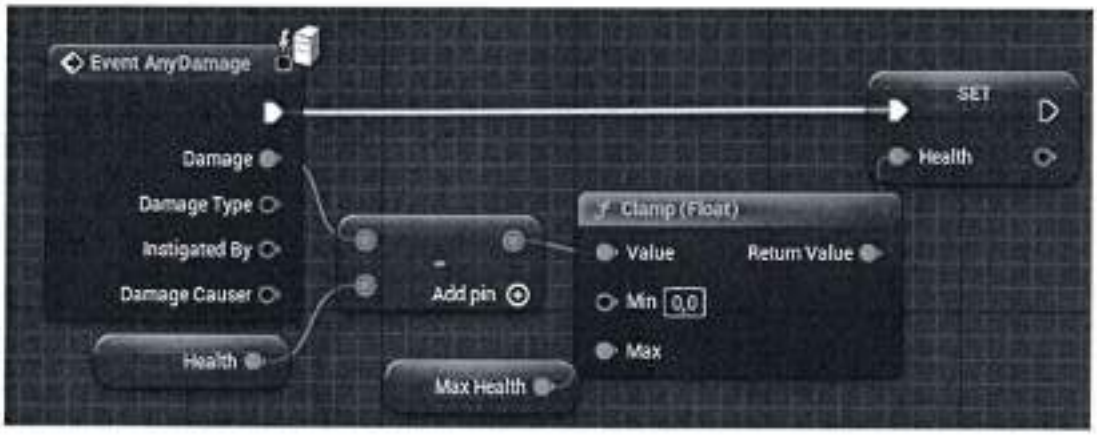

Рис. 3.32. Прийом пошкоджень

За основу використовується вбудована в двигун функція "Event AnyDamage". Яка відповідає за прийом урону від інших персонажів.

Процес реалізації механізму атаки складається з багатьох частин, але варто помітити основні, якраз на чому все ґрунтується. Для початку треба встановити умови, при яких буде виконуватися функція атаки. Для того аби наша функція спрацювала тоді коли треба, треба використати її в правильному місці, а саме після того як наш ворог добереться до точки до якої він рушив, тобто до нашого головного персонажу. Цей функціонал показано на малюнку нижче.

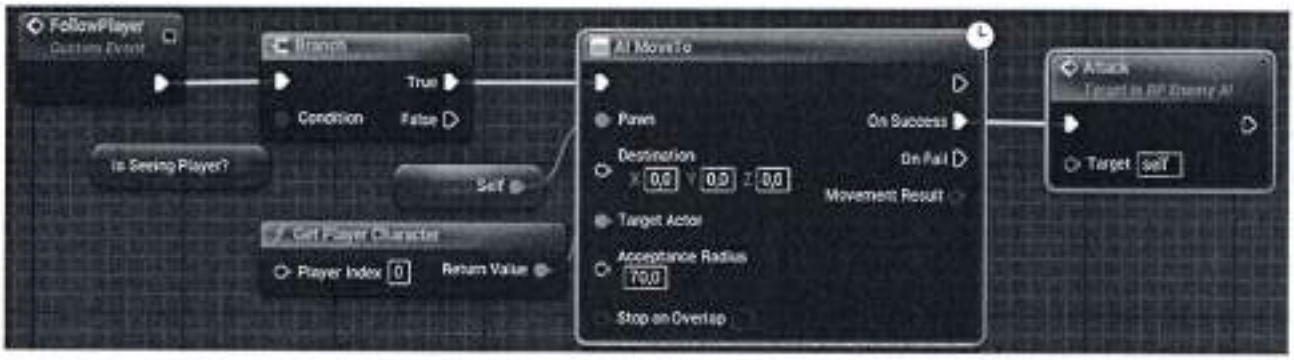

Рис. 3.33. Умови при яких відбудеться атака

Після виконаних зверху умов далі буде доречно використати цикл для проведення атак. Звісно для специфічних ворогів реалізація механіки атаки може відрізнятись. Наприклад бувають одноразові вороги, тобто їх ціль підбігти та

вдарити всього один раз і після цього зникнути, в таких випадках використання циклу буде не доречним. Але зазвичай в більшості випадків для реалізації середньостатистичного ворога в першу чергу використовується цикл.

В циклі спочатку вимикається можливість рух під час однієї ітерації циклу. Це робиться для того, аби анімація спрацьовувала правильно. Тому що якщо цього не зробити, то підуть баги. Наприклад, ворог підходе до головного персонажу та починає відігравати анімацію, але якщо головний персонаж в цей час почне рухатись то анімація не закінчиться а просто обірветься. Це руйнує візуальну картинку гри, плавність гри та комфорт геймплею.

Після того як ворог добіг до персонажу та зупинився, йде наступний ключовий етап. Відігрівання анімації, але не звичайної а модифікованої, тобто Anim Montag.

Після того як робота анімації завершилась, ми знову повертаємо нашому ворогу можливість руху. Ми це робимо завдяки функції "Set Movement Mode". В параметрах цієї функціє ми маємо змогу вибрати який саме режим встановити. Є багато різних варіантів переміщення але нам потрібно вибрати той, який був до того як ми вимкнули можливість руху, тобто Walk.

Після всього цього процесу наш цикл повертається назад, а саме вже на наступну його ітерацію. В наступній ітерації циклу спочатку йде перевірка чи ворог досі бачить нашого персонажа, якщо так весь описаний процес повторюється ще раз. Повність реалізована механіка атаки ворога можна детально роздивитись на малюнку нижче.

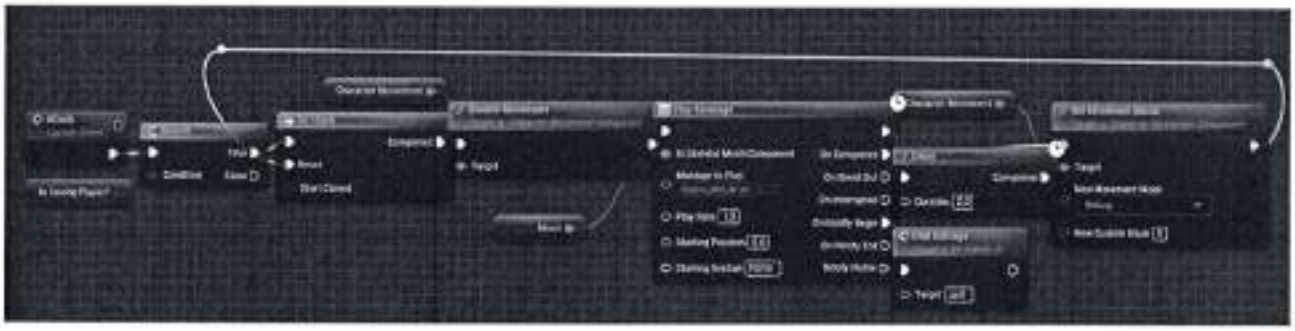

Рис. 3.34. Повноцінна механіка атаки ворога

Варто додати, що додатково відіграє важливу роль в побудуванні механіки атаки ворога те, чим саме буде відбуватися атака ворога, тобто це буде частина мешу персонажа, доданий об'єкт до персонажу чи щось інше. Це важливо враховувати, тому що треба знати радіус атаки, і під радіус атаки підлаштовувати інші параметри. Наприклад, якщо радіус атаки у ворога великий то, тоді йому не потрібна підбігати на коротку відстань. А якщо у ворога як основна зброя наприклад меч чи шабля, то треба підбігати на коротку відстань. Якщо це не враховувати то може вийте така ситуація, що ворог б'є по мешу головного персонажа, а через те що радіус атаки не прорахований – то буде так, що меч ворога сильно виходить це розміри мешу головного персонажу. Такі малі деталі паплюжать візуальну частину гри.

3.2.2.3 Пошкодження ворогів

Система завдання пошкоджень ворогами є критичним елементом геймплею, який впливає на взаємодію гравця з ворогами та загальний рівень виклику в грі. Ця система включає різні типи пошкоджень, методи їх завдання, інтеграцію з анімаціями та ефектами, а також тестування та налагодження для забезпечення збалансованого та захоплюючого ігрового процесу.

Для початку, треба зрозуміти на базову рівні коли саме ця функціональність повинна спрацьовувати. Для того аби цей ігровий механізм спрацював, потрібно щоб були сприятливі ігрові умові. Тобто, щоб ворог наніс пошкодження, треба створити область дії зброї або радіус атаки.

Для цього треба до головної зброї додати "Sphere Collision", ця сфера це просто радіус дії нашої зброї. Також ця сфера може експлуатуватися як датчик. Якщо головний персонаж попадає в область дії цієї сфери, то тоді ворог автоматично починає відігравати анімацію удари та наносити пошкодження. Це

може звучати логічно, але не варто забувати, що якщо погляд персонажа направлений в протилежну сторону від вас то він не буде реагувати на вам. Але якщо ви наблизетесь занадто близько до персонажу і попаде вже не під поле зору ворогу а під область дії сфери, то він почне відігравати анімацію та наносити урон.

Чому ця сфера динамічна, тобто змінює своє положення та прямує разом з зброєю. Тому що ця сфера кріпиться до зброї а зброя в свою чергу до кістки персонажа. І при програванні анімації для атаки, сфера рухається і так чином відбувається доторкання області дії сфери до головного персонажу.

На етапі розробці цю сферу можна та треба вмикати, для того аби взагалі бачити, чи наносить ворог пошкодження. Так треба зазначити, що ця сфера має ж колізію, тому бувають проблеми коли тебе одночасно атакують два вороги з однакового місцезнаходження, тому що сфери їхні перетинаються та через це відбувається конфлікт колізій. Для цього треба детально налаштовувати кожну сферу так, аби вона могла правильно співпрацювати з іншою сферою.

Для кращого розуміння цього функціоналу завдання пошкоджень ворогами, треба розібрати цей модуль на основні частини.

На малюнку нижче продемонстровано першу частину функціоналу цього механізму. Ми можемо побачити, що в першій частині ми більше працюємо з сферою, тобто ми заповнюємо функцію потрібними нам параметрами. Таким як "Get World Location" разом з радіусом атаки. Ці параметри треба аби двигун розумів, де саме повинна бути розташована сфера. Також важливим параметром є "Actor to Ignore". Цей параметр обов'язково треба вказувати як "Self", тому якщо цього параметру не вказати – то при спробі нанести пошкодження головному персонажу, ворог буде також наносити пошкодження самому собі.

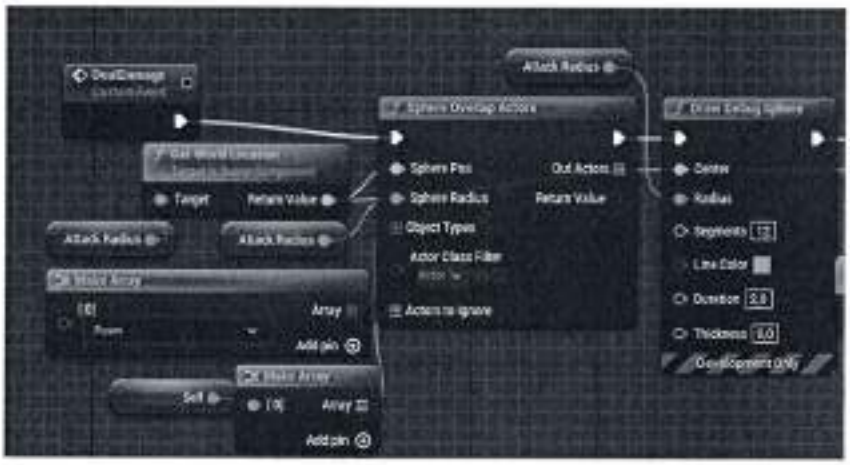

Рис. 3.35. Перша частина функціоналу

На малюнку видно, що після роботи над сферою далі йде функція "Draw Debug Sphere". Це треба для того аби намалювати нашу сферу після того як ворог вдарив по головному персонажу. Ця функція часто використовують програмісти на етапах розробки. В цій функції також є параметри, як було б бажано заповнити. Наприклад параметр радіус, покаже точний розмір сфери. Параметр "Duration" встановлює час в секундах скільки буде відображатись сфера. Параметр "Center" встановлює локацію сфери. Тому я думаю використання цієї функції на етапах розробки є просто необхідністю. Потім вже при шипінгу проекту цю функцію можна видалити.

На малюнку нижче видно, що в другій частині коду ми використовуємо цикл "For Each Loop". Але найголовнішою функцією другої частини функціоналу є функція "Apply Damage".

Рис. 3.36. Друга частина функціоналу

Вона використовується для того, щоб наносити пошкодження. Ця функція є вбудованою функцією двигуна. Розробники Unreal Engine реалізували її добре, тому в нас є багато варіантів використання цієї функції.

Система пошкоджень Unreal Engine дозволяє використовувати різні типи пошкоджень, такі як фізичні, магічні або статусні ефекти. Це досягається шляхом створення спеціальних класів, що успадковують "UDamageType".

Функція "Apply Damage' має багато дуже корисних та необхідних для заповнення параметрів. Наприклад параметр "Damaged Actor". Цей параметр відповідає за те, щоб показати хто саме буде отримувати пошкодження від нашого ворога, тобто ми. "Base Damage" надає інформацію, скільки саме життя буде зніматися за один удар. Параметр "Event Instigator", він потрібен для того, аби двигун розумів хто ініціював пошкодження (наприклад, контролер гравця). "Damage Causer" допомагає вказати хто конкретно завдає пошкодження.

В загалом функція завдання пошкоджень є важливим інструментом в Unreal Engine, що дозволяє реалізувати взаємодію між акторами в грі.

3.2.3 Меню завершення гри

Меню завершення гри є важливою частиною ігрового інтерфейсу. Воно надає гравцям зворотний зв'язок про закінчення гри, пропонує опції для перезапуску, повернення до головного меню або виходу з гри. Це меню має бути інтуїтивно зрозумілим, візуально привабливим і функціональним, забезпечуючи плавний перехід від закінчення гри до наступних дій гравця.

Меню завершення гри зазвичай включає кілька ключових компонентів:

1 Текст повідомлення про завершення гри: Повідомляє гравця про те, що гра закінчена. Це може бути просте повідомлення, наприклад, "Game Over" або більш детальне, включаючи причину завершення гри.

2 Кнопка перезапуску (Respawn): Дозволяє гравцю перезапустити гру з початку

3 Кнопка повернення до головного меню (Main Menu): Дозволяє гравцеві повернутися до головного меню гри.

На малюнку нижче продемонстровано меню закінчення гри.

Рис. 3.37. Меню закінчення гри

Розглянемо детальніше як створюється меню завершення гри. Для того аби створити меню завершення гри для початку треба створити відповідний Blueprint, але в цьому випадку Blueprint мати батьківський клас не “Character” як це зазвичай, а просто “User Widget”.

Зліва знизу в нас буде відображатись архітектура нашого Widget, це корисна панель, яка нам дуже знадобиться у використанні. Архітектура нашого вікна виглядає так, як показано на малюнку нижче.

Рис. 3.38. Архітектура вікна

На малюнку видно, що ієрархія будується зверху вниз, тобто першим та найголовнішим елементом нашого Widget це “Canvas Panel”.

“Canvas Panel”– це один із найважливіших та найгнучкіших елементів розмітки у Unreal Engine для створення користувацьких інтерфейсів (UI). Це контейнер для інших елементів UI, який дозволяє абсолютно позиціонувати їх у межах своїх границь. За допомогою “Canvas Panel” ви можете розміщувати, масштабувати та змінювати розмір елементів, щоб точно налаштувати інтерфейс гри.

“Canvas Panel” дозволяє задавати точні координати для кожного елемента, що міститься в ньому. Це особливо корисно для складних інтерфейсів, де точне розміщення має вирішальне значення.

Далі по ієрархії йде наш задній фон. Для заднього фону додано картинку, як має власні налаштування, наприклад прозорість та колір.

Наступний елемент це головний текс, який розташований в центрі та виділений червоним кольором. Він сигналізує про те, що гра завершилась.

Потім йде “Vertical Box”, це контейнерний елемент інтерфейсу в Unreal Engine, який дозволяє розміщувати дочірні елементи у вертикальному порядку. Це корисно для створення лінійних інтерфейсів, таких як меню, списки або

інформаційні панелі, де елементи повинні бути розташовані один під одним. Без цього елементу в нашій ієрархії ми б просто не змогли додати клавіші або зробити наше меню завершення гри більш функціональним. Також цей контейнер служить нам як маячок, який допомагає нам правильно розташувати кнопки, тобто забезпечуючи рівномірний інтервал між ними.

Перед останнім елементом нашої ієрархії є кнопка "Respawn", тобто відродження. Вона є важливим елементом у багатьох іграх, особливо в тих, де гравці можуть загинути та повинні мати можливість швидко повернутися до гри. Ця кнопка зазвичай розміщується в меню завершення гри або на екрані після смерті персонажа. Її натискання перезапускає гравця на певній точці відродження, дозволяючи продовжити гру. При натисканні цієї кнопки відбувається логіка, така як на малюнку нижче.

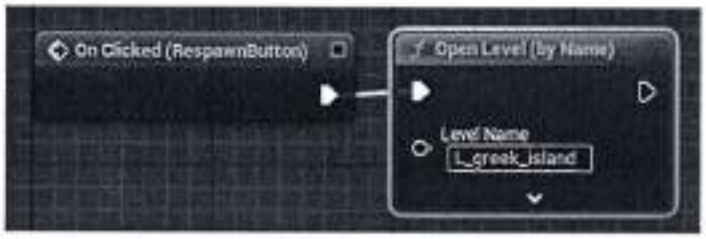

Рис. 3.39. Перезапуск гравця

При натисканні цієї кнопки ми вказуємо двигуну, що перенеси нас новий рівень. Насправді новий рівень це тільки для двигуна, для нас ні, тому що ми вже на ньому грали. На цьому рівні десь на карті стоїть "Player Start', саме там ми будемо відродженні як тільки нас завантажить на новий рівень.

Останній елемент ієрархії це ще одна кнопка, вона працює по абсолютно такому ж принципу. Єдина відмінність це назва рівня, куди нас буде завантажено. У випадку з кнопкою "Mein Menu" нас буде завантажено на рівень головного екрану гри.

Але просто створити меню завершення гри не достатньо, треба знати в якому місці та коли використати цей механізм. Тому в цьому проекті це меню

було використано одразу після того як наш головний персонаж вмирає. Вмирає наш персонаж тоді, коли в нього залишається 0 або менше балів життя. На малюнку знизу детально показано яким чином йде віднімання життів та коли саме та при яких умовах вмикається меню завершення гри.

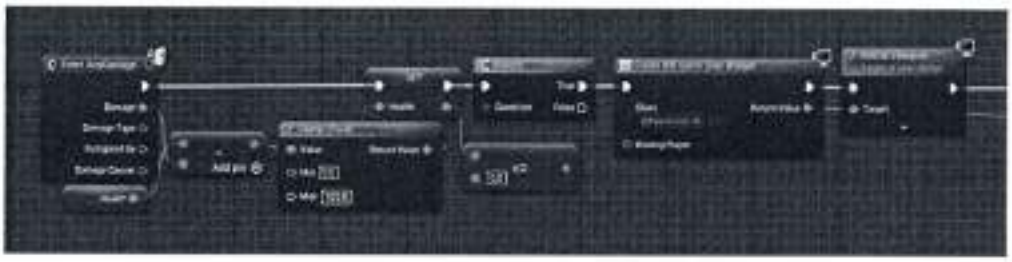

Рис. 3.40. Віднімання життів

Для того аби ми могли натиснути на кнопки, які є в меню, то нам треба перевести увагу двигуна з ігрового рівня на наш створений Widget. Це робиться завдяки логікі продемонстрованої знизу.

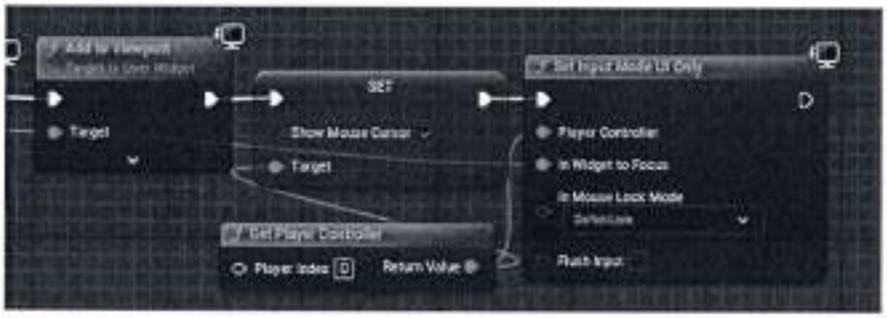

Рис. 3.41. Логіка активування курсора

3.3 Графічної частина гри

Переходимо до графічної структури гри.

Графічна структура гри - це архітектурний елемент, який описує організацію та взаємозв'язок графічних об'єктів у грі. Це включає в себе розміщення об'єктів у просторі, їхній зовнішній вигляд, взаємодію з освітленням, анімацію та інші аспекти, що стосуються візуальної реалізації гри.

Почнемо з базового ігрового простору, також по іншому їх називають рівнями. Простір - це 3D середовище, в якому розміщені графічні об'єкти.

Простір може бути статичним або динамічним і містити різні елементи декору, такі як дерева, будівлі, гори тощо. В даному проекту є всього два основних ігрових рівні, а саме головне меню та карта де відбуваються ігрові події. На малюнку 3.16. показано головне меню

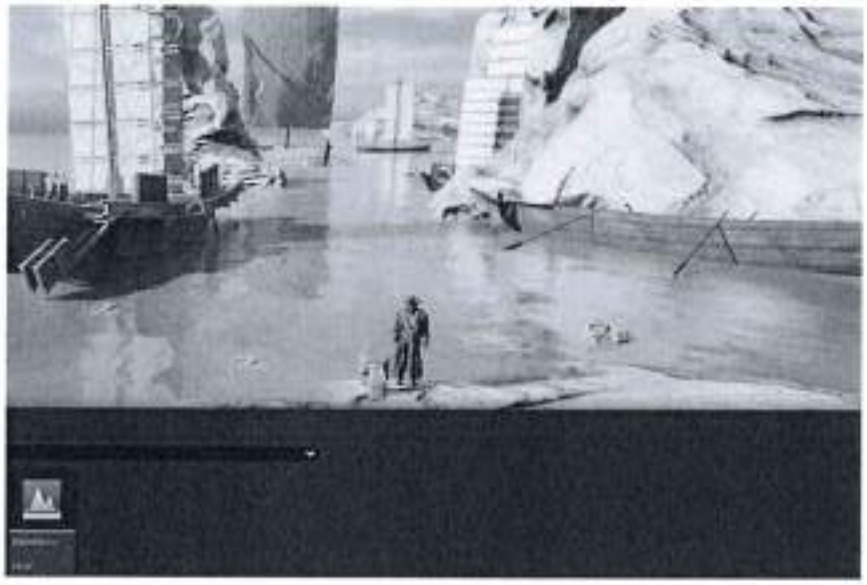

Рис. 3.42. Сцена головного екрану

На малюнку нижче показана карта, де відбувається весь головний ігровий процес. Ця карта містить в собі дуже багато різних об'єктів і достатньо високої якості. Перемикання між цими двома рівнями реалізовано в головному мені завдяки однієї клавіши, функціональність якої була реалізована в Widget Blueprint

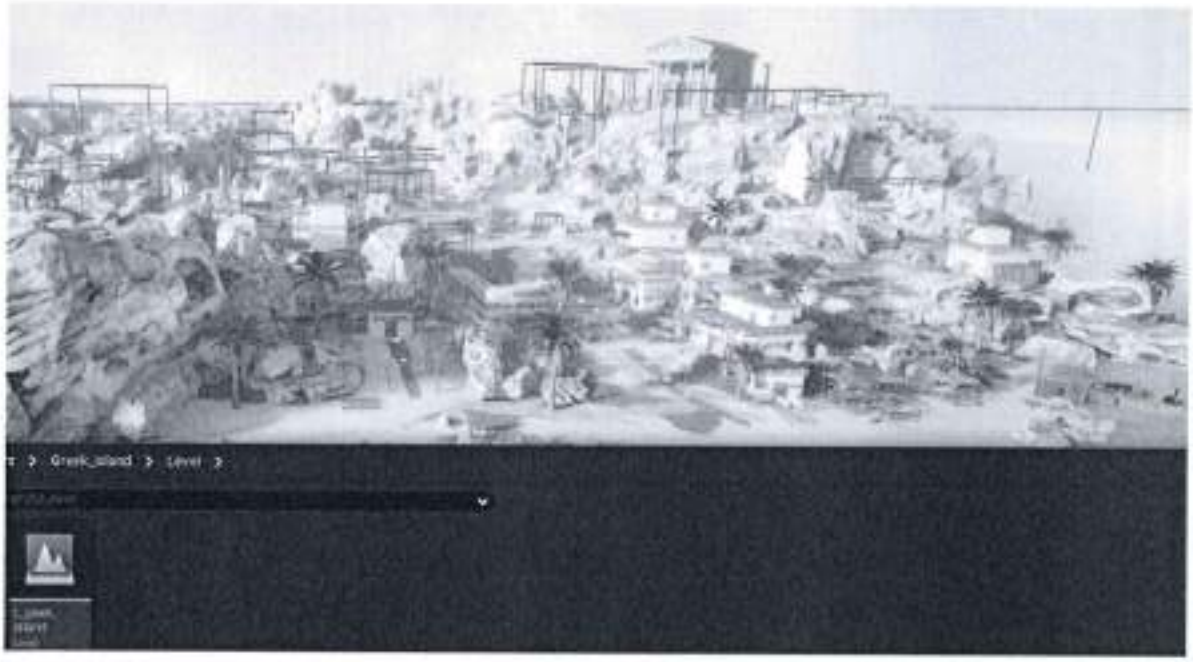

Рис. 3.43. Головна ігрова карта

Там де є великі світі, з великою кількістю об'єктів та з загальною високою якістю графіки – можуть бути проблем з оптимізацією. Тому для кращої оптимізацію проекту в самому двигуні розробники додали функцію, яка автоматично припиняє вимальовувати картину на певній відставні, яку ми самі встановимо. Це справді допомагає покращити оптимізацію проекту. Навіщо аби комп'ютер навантажував систему тим, що ви не бачите.

При створенні карти варто враховувати який саме об'єкт ви додаєте в проект. Варто дивитись на кількість кутів об'єкт. Тобто якість об'єкту. Якщо це всього одна маленька десь статуя, для виду і головне, до якої ви часто аби дуже близько підходити не будете – тоді не варто використовувати об'єкт з великою кількістю кутів. Наприклад, як на малюнках нижче. Зверніть особливу увагу на поле Triangles, чим більше число – тим важче об'єкт і тим більше комп'ютерних ресурсів воно потребує.

Рис. 3.44. Приклад об’єкту з великою кількістю кутів

Рис. 3.45. Приклад об’єкту з низькою кількістю кутів

Об'єкти в графічній структурі гри є основними елементами, які створюють інтерактивний світ гри. Вони можуть бути різноманітними за формою, розміром, функціоналом та способом взаємодії з гравцем.

Про об’єкти вже було сказано вище але додам ще трішки. Об’єкти це абсолютно все, що ви бачите в грі, персонаж, зброя, квітка, небо, море і так далі. Поєднання якості об’єкта з іншими властивостями на пряму залежать на якість картинки яку ви бачите. Правильне розміщення та використання цих об’єктів також має великий вплив. Програміст навряджи зможе створити класний віртуальний світ бо цьому треба вчитись окремо від кодування. Цим займаються Level-Designer. Програміст може просто десь трішки коректувати роботу Level-Designer.

У світі ігор об'єкти можуть мати найрізноманітніші форми і розміри, від невеликих предметів, таких як монетки або ключі, до великих структур, таких як будівлі або машини.

Об'єкти, розміщені в ігровому світі, можуть впливати на атмосферу гри, створюючи відчуття реалізму або фантазії, а також можуть впливати на настрій гравців.

На малюнку нижче видно правильно розташування об’єктів на карті. В даному випадку зберігається загальна атмосфера гри. Все взаємопов’язане за тематикою та атмосферою гри. Немає якогось одного об’єкту, який сильно відрізняється від інших.

Рис. 3.46. Приклад правильного розташування об’єктів на карті

Анімації є важливою складовою графічної структури гри і дозволяють надати руху та життя об'єктам і персонажам у грі. Анімації зазвичай створюються за допомогою ключових кадрів, де аніматор встановлює початкові та кінцеві позиції об'єкта або персонажа, а потім програма автоматично розраховує проміжні кадри для плавного переходу між ними.

Створенням анімацій займається аніматор.

В графічній структурі гри можуть бути різноманітні типи анімацій, такі як рухи персонажів, атаки, стрибки, зміни стану, анімації об'єктів тощо.

Самостійна робота з анімаціями насправді нелегкий процес. Програміст повинен розуміти будову скелету персонажа, які є типу будови скелетів, ієрархія скелету, вміти коректувати певні анімації, які були попередньо створенні аніматорами. Знати та використовувати ретаргетинг анімацій та скелетів.

Анімацій є велика кількість але є основі анімації, які використовуються майже в кожній грі, це наприклад анімації переміщення та атака.

Як взагалі додаються анімації? Анімації можна додавати в певних ситуаціях по різному. Наприклад анімації базового переміщення додаються на сітку, де враховується напрямок персонажу та швидкість переміщення. На малюнки нижче показано вікно, де додаються анімації переміщення на спеціальний макет. Параметри цієї сітки програміст самостійно встановлює та має можливість змінювати властивості.

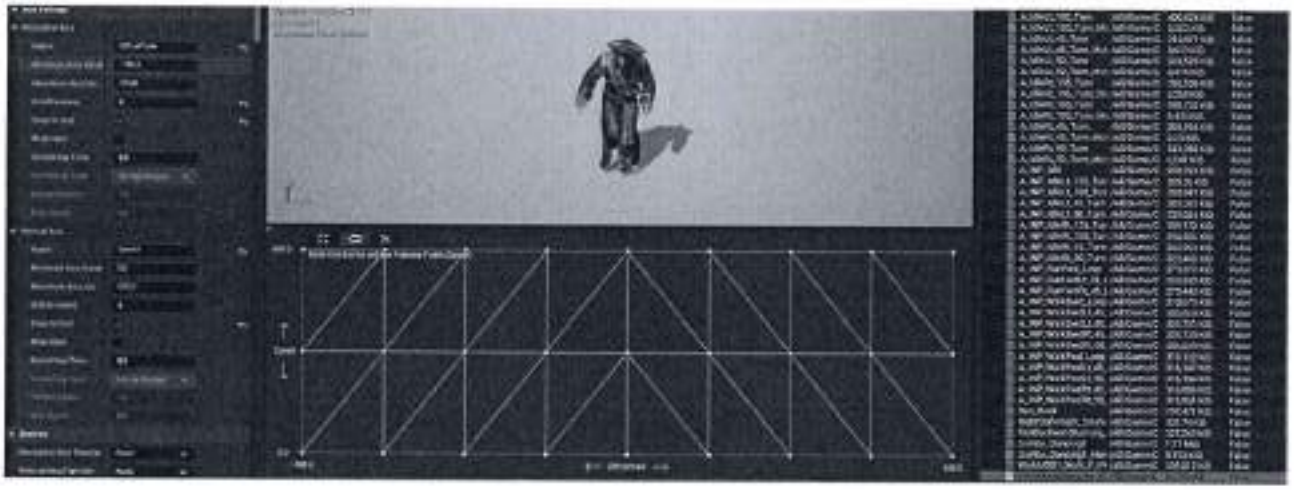

Рис. 3.47. Вікно налаштування анімацій

Інтерфейс в графічній структурі гри є важливим елементом, який забезпечує взаємодію гравців з грою та надає їм необхідну інформацію для успішної гри. Без інтерфейсів не обходиться жодна гра, тому що інтерфейс може служити як навігація по грі для гравця. Гарно створений інтерфейс, особливо якщо гарно взаємопов'язаний з ігровим процесом – дуже добре впливає на сприйнята та комфорт гри.

В даному проекті існує два інтерфейси. Це Widget Blueprint для головного меню та Widget Blueprint для ігрового процесу. На малюнку 3.22. показано Widget Blueprint головного меню. На малюнку 3.23. показано Widget Blueprint ігрового процесу.

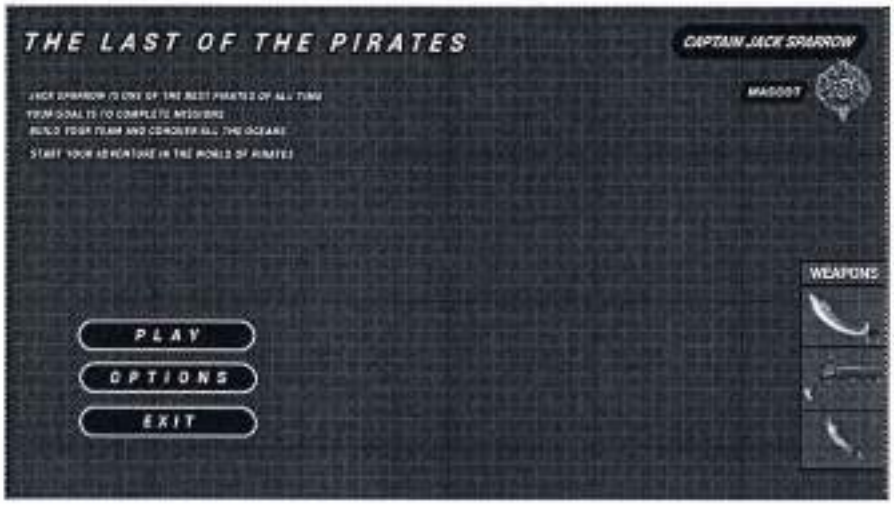

Рис. 3.48. Widget головного меню

Рис. 3.49. Widget ігрового процесу

Завдяки створеним клавішам в інтерфейсах (мал. X) гравець зможе переходити між ігровими рівнями. Також інтерфейси будуть покращувати загальний комфорт геймплею, а саме, віджети з кількістю патронів, яка саме зброя актуальна на даний момент, час гри, кількість гравців і так далі.

РОЗДІЛ 4 ІНСТРУКЦІЯ ДЛЯ ГРАВЦЯ ЩОДО ГЕЙМПЛЕЮ В ГРІ

Після того як ви запустите програму ви одразу опинитесь в головному меню гри. В головному меню у вас на вибір буде три клавіші: грати, налаштування та вийти. Зліва вище над клавішами ви зможете прочитати ваші ігрові завдання. Праворуч зверху стоїть ім'я персонажа та його власний талісман. Праворуч знизу доступна йому зброя. На малюнку нижче все детально показано.

Рис. 4.1. Головне ігрове меню

Після того як ви почнете ігровий режим вас перекинить на ігрову карту. Ви з'явитесь на березі острова.

Рис. 4.2. Ігровий процес

Зліва зверху ви побачите світлину вашого персонажа та поруч шкалу здоров'я та витривалості.

Персонаж витрачає свою витривалість тільки тоді коли біжить. Для того аби почати бігти треба натисну комбінацію клавіш Shift + W/S/A/D. Поки ви біжите ваша шкала витривалості буде зменшуватись поки не досягне нульового значення. Як тільки витривалість повністю закінчиться – персонаж не зможе далі бігти, тоді персонаж перейде в звичайну ходьбу. Поки витривалість не заповнена на 100% та персонаж не біжить – витривалість буде регенеруватися. Витривалість буде регенеруватися доки, поки шкала повністю не заповниться. Регенерування витривалості неможливе поки персонаж біжить або шкала дорівнює 100%.

Рис. 4.3. Зменшення витривалості

Рис. 4.4. Збільшення витривалості

Також зліва зверху знаходиться шкала здоров'я. Вона показує поточну кількість здоров'я персонажу. Натискаючи на клавішу J ви віднімаєте собі здоров'я. Натискаючи на клавішу H ви додаєте собі здоров'я. Віднімання здоров'я супроводжується червоною шкалою. Червона шкала показує, яку кількість здоров'я було втрачено. Це було додано для зручності, аби користувач міг орієнтуватись хто або що яку кількість здоров'я віднімає.

Рис. 4.5. Відображення кількості зменшення здоров'я

Якщо ви хочете стрибнути то треба натиснути клавішу Spacebar. Варто враховувати що при комбінації клавіш Shift + W/S/A/D + Spacebar персонаж буде стрибати довше ніж при звичайному натиску стрибка. Тому що, при швидкому бігу людина стрибає довше ніж при ходьбі чи з місця.

Рис. 4.6. Стрибок персонажа

При натисканні клавіш 1,2,3 буде програватись музика разом з танцем. Персонажу вимкнена можливість пересування під час програвання танцю, тому варто враховувати коли краще не натискати на цю клавішу. Кожен танець різний та музика до кожного танцю підібрана з смаком. Не варто вмикати перший танець поки не закінчився перший. На малюнках нижче показані три основних танці.

Рис. 4.7. Танець 1

Рис. 4.8. Танець 2

Рис. 4.9. Танець 3

В грі існує інвентар, тому на початку гри взяти зброю. На вибір дається два види зброї. Для того аби взяти зброю треба натиснути клавішу F.

Рис. 4.10. Озброєння

Зброю потрібно використовувати тоді, коли вам грозить небезпека. Небезпечно в грі становиться тоді, коли ви заходите в зону, яка охороняється ворогами. Ваша ціль не вмерти, тому вам прийдеться вести вогонь по ворогам.

Рис. 4.11. Епізод з зони яка охороняється

Якщо вас вб'ють то у вас відкриється меню, в якому ви зможете обрати що ви хочете далі, переграти чи повернутись в головне меню.

Рис. 4.12. Меню завершення гри

Знайдіть всіх жителів та врятуйте їх від ворогів.

Рис. 4.13. Ігрова місія

ВИСНОВОК

Під час виконання дипломної роботи я покращив практичні навички в розробці ігор на двигуні Unreal Engine, проектування проекту, навчився знаходити рішення труднощів, які виникають під час розробки проекту.

Завдячуючи цій дипломній роботі я зміг вивчити багато нових технологій, таких як: Retargeting персонажів, 8 Way Movement Animations, Retracing, Lumen, Weapon System, Enemy System, які будуть або вже використовуються в розробках ігор для комп'ютерів. Отримав досвід в створенні нового проекту, пройшов етапи від "Я не знаю з чого почати" до "Нарешті я завершив проект".

В процесі розробки власного проекту було проаналізовано багато ігрових модулів, які в загальному вважаються не досить легкими. Були добре про опрацьовані такі теми як, система зброї, а саме як створюється механізми стрільби, траєкторія пуль, вогневі режими, ігровий інвентар під різні класи зброї, прицілювання та віддача. Також була добре про опрацьована тема ворогів та штучний інтелект ворогів. В цьому ігровому модулі мною було створено базову логіку штучного інтелекту, тобто поведінку штучного інтелекту при певних ігрових умовах, прийнята рішень при певних умовах, умови при яких штучний інтелект перестає свою працю і так далі.

Додатково оволодів іншими додатками для розробки проектів, таких як "Git Hub", для ведення системи управління версіями. "Blender" для роботи над різними ігровими задачами, наприклад детальне редагування мешу персонажа.

Завдяки цій дипломній роботі, був створений план дій, який довів цей проект до свого логічного завершення. Цей план дій тепер я зможу використовувати у власних цілях, для того аби правильно підходити до розробки якогось проекту. Основне завдання дипломної роботи, що полягало в розробці гри на двигуні Unreal Engine, було виконано в повному обсязі та вчасно

СПИСОК ПОСИЛАНЬ

1. ChatGPT // Режим доступу: https://chat.openai.com/
2. Mixamo // Режим доступу: https://www.mixamo.com/#/
3. Reddit Unreal Engine // Режим доступу: https://www.reddit.com/r/unrealengine/
4. Stackoverflow // Режим доступу: https://stackoverflow.com/
5. Sketchfab // Режим доступу: https://sketchfab.com/feed
6. Unreal Engine 5.3 Documentation // Режим доступу: https://docs.unrealengine.com/5.3/en-US/
7. Unreal Engine Forum // Режим доступу: https://forums.unrealengine.com/categories?tag=unreal-engine
8. Unreal Engine Marketplace // Режим доступу: https://www.unrealengine.com/marketplace/en-US/free?count=20&sortBy=effectiveDate&sortDir=DESC&start=0
9. Vanas Online School // Режим доступу: https://www.vanas.ca/en/blog/top-5-video-game-engines
10. YouTube // Режим доступу: https://www.youtube.com/

ДОДАТКИ

Додаток А

Опис деяких частин коду гри

Реалізація функції переміщення для головного персонажа:

```
void AMainCharacter::Move(const FInputActionValue& Value)
{
	// input is a Vector2D
	FVector2D MovementVector = Value.Get<FVector2D>();

	if (Controller != nullptr)
	{
		// find out which way is forward
		const FRotator Rotation = Controller->GetControlRotation();
		const FRotator YawRotation(0, Rotation.Yaw, 0);

		// get forward vector
		const FVector ForwardDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);

		// get right vector
		const FVector RightDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

		// add movement
		AddMovementInput(ForwardDirection, MovementVector.Y);
		AddMovementInput(RightDirection, MovementVector.X);
	}
}
```

Реалізація функцій відображення здоров'я та урона на шкалах Progress Bar:

```
float UUserWidgetHealth::GetCharacterHealth()
{
	AMainCharacter* OurPlayer = Cast< AMainCharacter>(UGameplayStatics::GetPlayerCharacter(GetWorld(),0));
	float GetHealth = OurPlayer->Health/100.f;
	return GetHealth;
}

float UUserWidgetHealth::GetCharacterRedHealth()
{
	AMainCharacter* OurPlayer = Cast< AMainCharacter>(UGameplayStatics::GetPlayerCharacter(GetWorld(), 0));
	float GetBackHealth = OurPlayer->BackHealth / 100.f;
	return GetBackHealth;
}
```