Державний торговельно-економічний університет

Кафедра комп`ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка комп`ютерної гри на базі Unreal Engine 5»

Студента 2 курсу 4м групи
спеціальності 122
«Комп`ютерні науки»
освітньої програми
«Комп`ютерні науки»
другого (магістерського)
рівня _____ Бая Андрія
Сергійовича

Науковий керівник:
кандидат фізико-математичних
наук, доцент _____ Філімонова Тетяна
Олегівна

Гарант освітньої програми:
доктор фізико-математичних
наук, професор _____ Пурський Олег
Іванович

Київ 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
«20» грудня 2022р.

Завдання
на випускну кваліфікаційну роботу студенту

Баю Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1.Тема випускного кваліфікаційного проекту
«Розробка комп'ютерної гри на базі Unreal Engine 5»
Затверджена наказом ректора від *«06» грудня 2022 р. № 3284*
2. Строк здачі студентом закінченого проекту *26 листопада 2023 року*
3. Цільова установка та вихідні дані до проекту
Мета роботи: розробка комп'ютерної гри на базі Unreal Engine 5.
Об'єкт дослідження: процес розробки комп'ютерної гри на Unreal Engine 5.
Предмет дослідження: методологія розробки комп'ютерних ігор.
4. Перелік графічного матеріалу _____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Філімонова Т.О.	15.12.2021 р.	15.12.2021 р.
2	Філімонова Т.О.	15.12.2021 р.	15.12.2021 р.
3	Філімонова Т.О.	15.12.2021 р.	15.12.2021 р.

6. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. Постановка задачі та формування концепту гри

1.1.Постановка задачі

1.2.Розробка концепту

1.3. Способи розробки гри

Висновки по розділу

РОЗДІЛ 2. Розробка графічного контенту

2.1. Розробка 3D моделей

2.2. Ріггінг та анімація

2.2. Розробка графічного дизайну

Висновки по розділу

РОЗДІЛ 3. Розробка гри у Unreal Engine

3.1. Імпорт ассетів у движок

3.2. Розробка ігрових механік

3.3. Критерії оцінювання

3.4. Результати тестування

Висновки по розділу

ВИСНОВОК

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ По р.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*	*01.11.2022*	*01.11.2020*
2	*Розробка та затвердження завдання для випускної кваліфікаційної роботи*	*06.12.2022*	*05.12.2020*
3	*Вступ*	*01.06.2023*	
4	*РОЗДІЛ 1. Постановка задачі та формування концепту гри*	*25.06.2023*	
5	*РОЗДІЛ 2. Розробка графічного контенту*	*02.09.2023*	
6	*Підготовка статті у збірник наукових статей магістрів*	*09.09.2023*	
7	*РОЗДІЛ 3. Розробка гри у Unreal Engine*	*21.10.2023*	
8	*Висновки*	*02.11.2023*	
9	*Здача випускної кваліфікаційної роботина кафедру науковому керівнику*	*05.11.2023*	
10	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023*	
11	*Виправлення зауважень, зовнішнє рецензування випускного кваліфікаційної роботи*	*22.11.2023*	
12	*Представлення готової зшитої випускної кваліфікаційної роботи*	*26.11.2023*	
13	*Публічний захист випускної кваліфікаційної роботи*	*Згідно роботи ЕК*	

8. Дата видачі завдання «15» грудня 2022 р.

9. Керівник випускної кваліфікаційної роботи

Філімонова Т.О.

(прізвище, ініціали, підпис)

10. Гарант освітньої програми Пурський О.І.

(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент-дипломник

Бай А.С.

(прізвище, ініціали, підпис)

12. Відгук керівника випускної кваліфікаційної роботи

У випускній кваліфікаційній роботі розроблена гра на Unreal Engine 5. Створена логіка гри та ігрові механізми, проведено тести на правильність роботи і швидкодію. Робота оформлена згідно з правилами оформлення ВКР. Всі поставлені задачі виконані. Вважаю, що робота може бути допущена до захисту.

Керівник випускної кваліфікаційної роботи

04.12.2023 р.

(підпис, дата)

13. Висновок про випускну кваліфікаційну роботу

Випускний кваліфікаційний проект студента Бая А.С.

(прізвище, ініціали)

може бути допущений до захисту в екзаменаційній комісії.

Гарант освітньої програми Пурський О.І.

(підпис, прізвище, ініціали)

Завідувач кафедри Пурський О.І.

(підпис, прізвище, ініціали)

«_____» _____ 2023 р.

Вступ

Актуальність даної теми обумовлена швидким розвитком галузі комп'ютерних ігор та постійним зростанням очікувань гравців. Розробка комп'ютерних ігор є однією з найбільш актуальних і перспективних галузей сучасної індустрії розваг. Швидкий технологічний прогрес, постійне удосконалення графічних та 3D-редакторів, а також зростання популярності віртуальної реальності сприяють зростанню інтересу до індустрії. На фоні великої кількості ігрових проектів від численних студій, та перенасиченості ринку одноманітними іграми, що копіюють одне одну, відкриваються двері для інді-розробників, які зможуть зачепити гравця цікавим концептом, новими ігровими механіками, або незвичним поглядом на відомий жанр.

Мета роботи: розробка комп’ютерної гри на базі Unreal Engine 5. **Об’єкт дослідження:** процес розробки комп’ютерної гри на Unreal Engine **Предмет дослідження**: методологія розробки комп’ютерних ігор. **Методи дослідження:** Теоретичною основою дослідження є загальнонауковий аналітичний метод, а також системний підхід і праці провідних вчених з аналізу розвитку ринку відеоігор. Для практичного вирішення поставлених задач використовувалися такі методи: методи 3D моделювання, текстурування та анімації, методи blueprint-програмування, функціональне тестування та тестування продуктивності. **Наукова новизна** даної роботи полягає у вивченні та аналізі актуальних методів розробки комп'ютерних ігор на основі передового ігрового рушія Unreal Engine 5. Розгляд процесу створення гри на прикладі UE5, який включає в себе моделювання та текстурування ігрових ассетів, ріггінг та анімацію, розробку графічного контенту, ігрових механік, тестування, є внеском у розвиток методології розробки ігор. Також, робота висвітлює важливі аспекти в сучасній індустрії розваг, такі як використання новітніх технологій у розробці, висока конкуренція на ринку гри, та зростання інтересу до інді-розробників. Це сприяє розумінню та впровадженню інновацій у галузі розробки комп'ютерних ігор.

В даній роботі буде розглянуто процес розробки комп’ютерної гри на базі потужного движку Unreal Engine 5. Unreal Engine 5 (UE5) – це ігровий рушій від компанії Epic Games. Написаний на мові програмування C++, він дає розробникам гри широкий спектр інструментів та можливостей для творчості, і підходить для розробки ігор під більшість сучасних платформ і операційних систем.

Для цього необхідно буде виконати наступні завдання:

- Моделювання та текстурування ігрових ассетів;
- Ріггінг та анімація;
- Розробка графічного контенту;
- Розробка ігрових механік у рушії;
- Тестування.

Процес розробки буде умовно поділено на 3 етапи:
- Постановка задачі та формування концепту. На цьому етапі буде сформульована концепція гри, її стилістика та основні механіки, а також розглянуто програмне забезпечення, що буде використано в розробці.
- Розробка графічного контенту. Під графічним контентом розуміється будь-які елементи, якими будуть наповнені сцени – модульні елементи локацій, предмети оточення, персонажі, а також іконки, елементи ігрового інтерфейсу та інше. Також для персонажів гри будуть створені риг і анімація.
- Розробка гри у Unreal Engine. Цей етап передбачає інтеграцію створеного графічного контенту у рушій, розробку ігрових механік, встановлення критеріїв оцінювання гри та її тестування.

Гра буде створена на основі блупринтів (blueprint) – системи візуального програмування у рушії UE5, що представляють собою систему нодів з різноманітними даними (подіями та функціями).

Результатом випускної кваліфікаційної роботи стане повністю готова та функціонуюча гра, а також результати її тестування за критеріями, зазначеними у 3-му пункті розділу 3.

Результати дослідження були опубліковані в збірнику наукових студентських статей «Прикладні комп’ютерні технології».

РОЗДІЛ 1. Постановка задачі та формування концепту гри

1.1 Постановка задачі.

Розробка комп'ютерних ігор є однією з найбільш актуальних і перспективних галузей сучасної індустрії розваг. Швидкий технологічний прогрес, постійне удосконалення графічних та 3D-редакторів, а також зростання популярності віртуальної реальності сприяють зростанню інтересу до індустрії.

Пандемія COVID-19 мала значний вплив на глобальну індустрію розваг, зокрема на ринок відеоігор. Завдяки обмеженням та карантинним заходам, люди проводили значний час вдома, що значно збільшило попит на цифрові розваги, до яких відносяться відеоігри. Згідно PwC Global, з кожним роком дохід від ігор зростає, і згідно прогнозам це число буде лише рости [1]. В свою чергу, це призвело до ще більшого зростання обсягу ігрового ринку, і вже в 2022 році він складав $202.7 млрд. [2]. За оцінкою експертів з IMARC Group, у перспективі до 2028 року обсяг ринку виросте до $ 343.6 млрд., що є вагомим аргументом на користь актуальності розробки відеоігор.

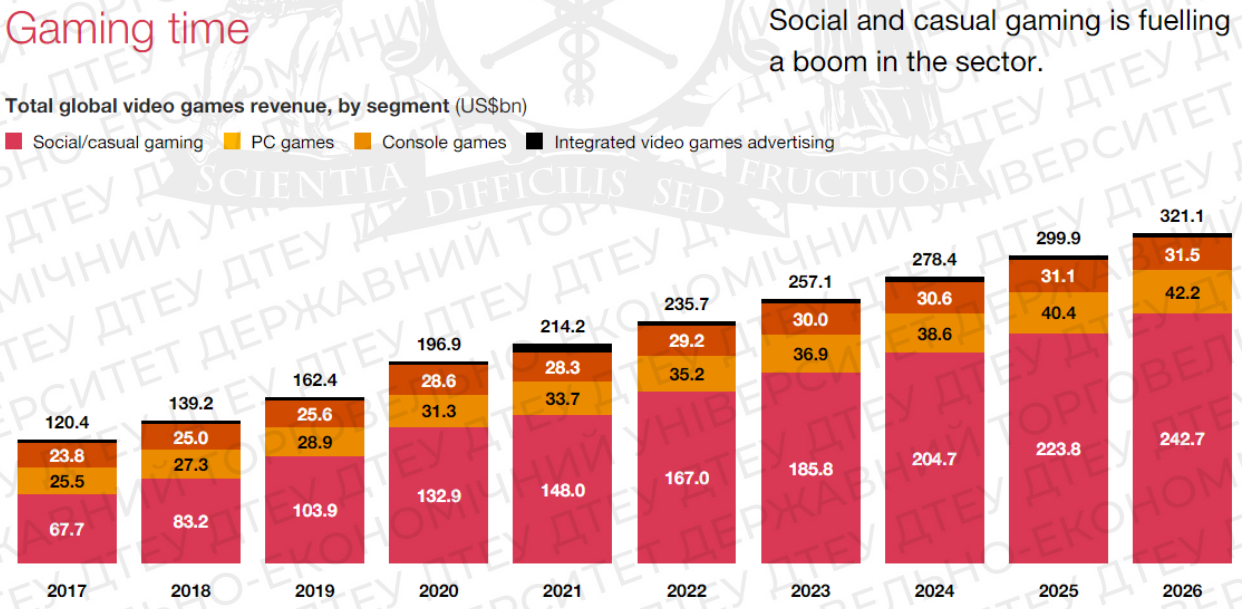

Рис. 1.1 Темпи зростання часу, проведеного за відеоіграми

І оскільки з’явився великий попит на цифрові розваги, це відкрило двері для багатьох розробників, в тому числі і для малих незалежних студій, чи навіть індивідуальних проектів. В такий час навіть маючи багатомільйонних бюджетів можна зробити цікавий продукт, який приверне увагу гравців.

Не секрет, що ринок комп'ютерних та мобільних ігор є перенасиченим одноманітними проектами, які не вносять нового у індустрію та не розвивають її. Це створює попит на цікаві проекти з незвичними ігровими механіками, впізнаваним візуальним стилем, або цікавою концепцією, яка розширює рамки існуючого жанру чи навіть стає основою для нового[2].

Задовольнити цей попит і є завданням дипломної роботи.

Для цього буде розроблена гра на базі Unreal Engine 5. Для того, щоб мати чітке уявлення про кінцевий результат, необхідно сформувати вимоги до гри. Отже, гра повинна:

Бути працездатною, тобто запускатись і працювати без помилок.
Відповідати концепту своїм дизайном і ігровими механіками.
Мати прийнятний для гри fps (frames per second) навіть на слабкому пристрої.
Мати яскравий візуальний дизайн, який приверне увагу та виділить проект серед конкурентів.
Мати цікаві ігрові механіки, які захоплять увагу гравця.
Бути простою в реалізації, адже розробником є одна людина.

1.2 Розробка концепту

Тепер, орієнтуючись на вказані вище вимоги, сформуємо концепт гри. Гра буде представляти собою гібрид шутера та Tower Defense, поєднуючи будування захисних турелей із стрільбою по хвилям противників.

Місцем дії гри є коридор покинутого бункера, заставлений барикадами. Один кінець коридору представляє собою масивні двері, через які проникають зомбі. Вони рухатимуться уздовж коридору, по напрямку до генератора, розташованого в іншому кінці коридору. Біля стін розташовані спеціальні місця під будівництво турелей. Їх будівництво та посилення буде одним з найважливіших елементів геймплею. Турелі будуть трьох видів, кожна з яких має свої особливості, переваги і недоліки.

Останнім елементом оборони є сам гравець. Під час хвилі його персонаж переміщується вздовж виступу на одній із стін, паралельно руху зомбі. Гравець озброєний гвинтівкою, яку так само як інші елементи оборони, можна апгрейдити.

Рис. 1.2 Приклад геймплею

Ігровий процес ділиться на 2 стадії, що чергуються – стадія підготовки та стадія бою.

Під час стадії підготовки гравець може вибрати, на що витратити енергію генератора – основний ресурс гри. Його можна витратити на будівництво турелей, розміщення пасток чи апгрейд зброї. Саме на цій стадії гравець вибирає стратегію, яка більше підходить його стилю гри – чи буде він робити ставку на власні сили, і вкладати енергію на зброю, чи відігравати інженера, який повністю покладається на механізми.

Об’єкти для встановлення/апгрейду є наступні:

Гвинтівка - Особиста зброя персонажа, при покращені завдає більше пошкоджень та/або стріляє швидше.

Турелі – автоматизовані орудійні платформи – є непідконтрольними гравцю об’єктами, які самостійно захищають генератор, націлюючись на найближчого ворога у своєму радіусі дії.

Барикади – перепони, що встановлюються в спеціально відведених для цього слотах. Вони мають параметр міцності, який зменшується від атак зомбі. Цей параметр не відновлюється між раундами автоматично, для ремонту барикад необхідно витрачати енергію. Також за енергію можна покращити її, збільшивши максимальний запас міцності. Перепону, яка під

час хвилі зомбі була знищена, можна встановити заново, але тоді всі її апгрейди будуть втрачені.

Пастки:

Пастки – це особливі засоби стримування, що розміщуються гравцем в спеціальних місцях. Вони спрацьовують, коли противник заходить в зону їх активації, і накладають певний ефект на всі цілі в зоні дії. Пастки є одноразовими, тобто знищуються при спрацьовуванні. На відміну від інших об’єктів, їх не можна покращувати.

Коли енергія витрачена, і гравець готовий до бою, починається стадія бою. В одному з кінців коридору спавняться зомбі, і прямують до генератора. Натрапляючи на перешкоду, вони починають ламати її, завдаючи їй пошкодження залежно від свого рівня атаки.

Хвиля зомбі оминає турелі та виступ, на якому знаходиться гравець, тому вони вважаються невразливими об’єктами і не мають смуги здоров’я.

Гравець програє, якщо міцність генератора впаде до нуля. Якщо ж хвиля зомбі була відбита, гра знов переходить у стадію підготовки, і починається новий раунд. Кожен наступний бій буде важчим за попередній, адже характеристики противників зростатимуть з кожним раундом.

1.3 Способи розробки гри

Перед початком розробки гри необхідно провести аналіз методів створення ігор та програмного забезпечення, яке може знадобитися у процесі.

Одним із найбільш відомих і розповсюджених ігрових рушіїв є Unreal Engine. Розроблений компанією Epic Games, цей рушій визнаний одним з найсильніших інструментів для створення комп'ютерних ігор. Його широкий спектр функціональності і потужні можливості роблять Unreal Engine популярним серед професіоналів галузі.

Особливості Unreal Engine суттєво впливають на якість ігрових продуктів, що розробляються з його використанням. Один з головних аспектів - це його графічна потужність. Unreal Engine пропонує вражаючу візуальну якість завдяки високорівневому рендерингу, фотореалістичним ефектам освітлення, динамічним шейдерам та реалістичним фізичним моделям. Це дозволяє розробникам створювати деталізовані світи з

вражаючими візуальними ефектами, що забезпечує глибоку іммерсію для гравців[3].

Крім того, Unreal Engine володіє потужними інструментами для розробки ігрових механік та фізики. Його система поведінки персонажів дозволяє створювати складні анімації, реалістичну фізику та інтерактивність у грі. Завдяки вбудованій системі штучного інтелекту, розробники можуть створювати розумних противників та непередбачувані ситуації, що збільшує геймплейну глибину та складність ігрового процесу.

Плюси:

- Графічна потужність: Unreal Engine пропонує вражаючу візуальну якість і можливості, що дозволяють створювати деталізовані світи з фотореалістичними ефектами.
- Інструменти для розробки ігрових механік: Unreal Engine має потужні інструменти для створення складних анімацій, фізики та інтерактивності у грі.
- Широкий функціонал: Рушій пропонує багато корисних функцій, таких як система штучного інтелекту, звуковий двигун, редактор матеріалів та багато інших.
- Велика спільнота розробників: Unreal Engine має активну спільноту розробників, яка допомагає обмінюватись знаннями, рішеннями та підтримкою.

Мінуси:

- Складність в освоєнні: Розробка на Unreal Engine вимагає певного рівня технічних знань і досвіду, що може бути викликом для початківців.
- Високі вимоги до обладнання: Для роботи з Unreal Engine потрібний потужний комп'ютер з відповідними характеристиками, що може бути обмеженням для деяких розробників.
- Ліцензійна вартість: Використання Unreal Engine може бути пов'язано з певними витратами, зокрема, залежно від типу ліцензії, яку обирає розробник.

Враховуючи ці плюси та мінуси, Unreal Engine є потужним інструментом для розробки комп'ютерних ігор, проте вимагає певного рівня знань та ресурсів для успішної реалізації ігрових проектів. [4]

Найближчим конкурентом Unreal Engine є рушій Unity. Unity є однією з найпопулярніших платформ для розробки комп'ютерних ігор і відома своєю гнучкістю та доступністю для розробників з різним рівнем досвіду.

Особливості Unity полягають у його кросплатформеності, що дозволяє розробляти ігри для різних платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність. Це дає розробникам широкий охоплення аудиторії та можливість публікації ігор на різних ринках.

Ще однією особливістю Unity є його вбудований магазин активів, де розробники можуть придбати готові ресурси, такі як моделі персонажів, текстури, звуки тощо, або навіть власноруч створити та продавати свої активи. Це спрощує процес розробки, зменшує витрати на виробництво і дозволяє швидше створювати ігри.

Плюси:

- Простота: має зрозумілий і простий в освоєнні редактор, який швидко освоїть навіть той, хто раніше ніколи не займався розробкою ігор.
- Кросплатформеність: Unity дозволяє розробляти ігри для різних платформ, забезпечуючи широку аудиторію та можливості публікації на різних ринках.
- Магазин активів: Розробники можуть придбати готові ресурси або продавати власні активи, що спрощує і прискорює процес розробки.
- Доступність: Unity відомий своєю доступністю для розробників з різним рівнем досвіду, дозволяючи широкому колу людей займатися розробкою ігор.

Мінуси:

- Обмежені можливості графічного рендерингу: У порівнянні з Unreal Engine, Unity може мати меншу графічну потужність і меншу кількість графічних ефектів.
- Висока залежність від скриптів: Unity використовує мову програмування C# для розробки, що може вимагати більшої кількості програмістських знань в порівнянні з візуальним програмуванням, що пропонує Unreal Engine.
- Вартість деяких додаткових функцій: Деякі розширення та функціональність Unity можуть бути доступні лише за плату, що може збільшити витрати на розробку.

Як бачимо, Unity є достойним конкурентом UE через свою доступність та універсальність. Але чому для дипломної роботи був вибраний саме Unreal Engine 5? Справа в системі блупринтів - системи візуального програмування Unreal Engine, що представляє собою систему нодів (вузлів) з різноманітними даними (подіями та функціями). За допомогою них розробник може створити гру не написавши жодної строки коду. Це дозволяє не-програмістам писати скрипти для ігор, перетаскуючи ноди, задавати їх властивості у інтерфейсі та поєднувати між собою [5].

Кожен вузол має один або декілька портів, які приймають чи повертають певний параметр. Порти зліва є портами входу, справа – виходу.

Таким чином, хоч Unreal Engine може бути важчим в освоєнні, та мати більше вимог до обладнання, вбудована система візуального програмування значно знижує поріг входження.

Варто також розглянути програмне забезпечення для 3D моделювання, яке буде використовуватись для створення ігрових ассетів. Найпопулярнішими 3D пакетами є Blender, 3DsMax і Maya.

Blender: Blender - це безкоштовний пакет для 3D-моделювання, анімації та візуалізації. Він має потужні інструменти для створення складних моделей, реалістичних анімацій та візуальних ефектів. Однією з основних особливостей Blender є його відкритий вихідний код, що дозволяє користувачам модифікувати та адаптувати програму під свої потреби. Іншою відмінною рисою Blender є система геометричних нод, що роблять можливим процедурну генерацію складної геометрії.

Плюси:

- Безкоштовність: Blender є безкоштовним програмним забезпеченням, що робить його найдоступнішим з усіх 3D програм.
- Відкритий вихідний код: Відкритий вихідний код Blender дозволяє розробникам модифікувати та розширювати його функціональні можливості.
- Функціональність: Blender має широкі можливості для моделювання, анімації, симуляції та візуалізації.

- Універсальність: може використовуватись для будь-якого виду робіт – від 3D-візуалізації інтер'єрів до створення ігрових ассетів та візуальних ефектів.

Мінуси:

- Крутий початковий поріг: Blender має дещо складну навігацію та інтерфейс, що може затримати початківців.
- Обмежена підтримка: У порівнянні з комерційними пакетами, Blender може мати меншу підтримку користувачів у вигляді документації та онлайн-курсів.

3ds Max: 3ds Max - це потужний пакет для 3D-моделювання, анімації та візуалізації, розроблений компанією Autodesk. Він має широкий спектр інструментів для створення складних моделей, фотореалістичних візуалізацій та анімаційних ефектів. Однією з основних особливостей 3ds Max є його висока інтеграція з іншими продуктами Autodesk, що дозволяє легко обмінюватися даними та працювати в єдиному екосистемі.

Плюси:

- Розширені можливості моделювання: 3ds Max має широкі можливості для створення детальних та складних моделей.
- Інтеграція з іншими програмами: 3ds Max легко інтегрується з іншими програмами Autodesk, що спрощує обробку зображень та співпрацю з іншими розробниками.

Мінуси:

- Вартість: 3ds Max є комерційним програмним забезпеченням, і може бути фінансово недосяжним для деяких користувачів.
- Вимоги до системи: 3ds Max вимагає потужних обчислювальних ресурсів, що може бути обмежуючим фактором для деяких користувачів.

Maya: Maya - це професійний пакет для 3D-моделювання, анімації та візуалізації, створений компанією Autodesk. Він широко використовується в галузі кіно, відеоігор, телебачення та інших секторах розваг. Maya відрізняється своїми потужними інструментами для анімації, візуалізації та симуляції, а також широкою підтримкою плагінів та розширень.

Плюси:

- Гнучкість та розширюваність: Maya надає широкі можливості для налаштування та розширення, завдяки плагінам та скриптам.
- Спеціалізовані інструменти: Maya пропонує спеціалізовані інструменти для секторів кіно, відеоігор та телебачення, що робить його популярним серед професіоналів.

Мінуси:

- Вартість: так як і 3DsMax, Maya є досить дорогим програмним забезпеченням, на даний момент ціна його місячної підписки становить $235/місяць.
- Вимоги до системи: Maya вимагає потужного обладнання та ресурсів, що може бути обмеженням для деяких користувачів.

Для дипломної роботи найбільш доцільно використовувати Blender, адже його доступність та універсальність є важливими перевагами в контексті невеликого інді-проекту. Blender давно увійшов у робочий процес багатьох невеликих студій, адже для них вигідніше використовувати безкоштовне ПЗ, яке не поступається якістю аналогам, а тому його можна справедливо назвати стандартом індустрії [6].

Задля виконання мети дипломної роботи використано наступне ПЗ:

- Zbrush (скульптинг персонажів)
- Blender (hard surface моделювання, ретопологія, розгортка)
- Substance 3D Painter (текстурування)
- Adobe Photoshop (Іконки та інші елементи інтерфейсу)
- Unreal Engine (левел дизайн, скрипти)

Оскільки методи створення моделей персонажів та об’єктів оточення відрізняються, це в свою чергу також знаходить відображення у використаному програмному забезпеченні. Для створення персонажів використовується 3D пакет Zbrush, розроблений компанією Pixologic. Ця програма є лідером у створенні високополігональних моделей, і широко використовується у кіно, мультиплікації, та ігровій сфері [7]. Через можливість стабільної роботи із десятками мільйонів полігонів, широкий набір пензлів та зручний інтерфейс, який ідеально підходить для роботи із графічним планшетом, Zbrush вважається стандартом індустрії для character-дизайну.

Adobe Substance 3d Painter є спеціалізованим ПЗ для текстурування трьохвимірних об’єктів. Він має потужну систему фарбування, яка дозволяє в режимі реального часу розфарбовувати модель, накладати інтелектуальні маски для надання текстурам деталізації, а також створенню нових матеріалів [8]. Тут будуть виконуватися два останні етапи, які передують імпорту моделі у рушій – запікання текстурних карт та власне створення текстур.

Окремо стоїть процес створення елементів інтерфейсу – іконок, головного меню, панелей та іншого. Це є елементи графічного дизайну, і для їх створення доречніше використовувати Adobe Photoshop – найпопулярніший редактор растрової графіки. Окрім створення колажів та редагування фотографій, Photoshop має широкий набір інструментів для малювання, і тому ідеально підходить для створення інтерфейсу гри. В дипломній роботі панелі інтерфейсу будуть створені з нуля, виключно інструментами Photoshop, в той час як іконки будуть виконані технологією overpaint, тобто малюванням поверх рендеру моделі. Це спрощує роботу, адже замість малювання персонажа потрібно лише доопрацювати зображення, надавши йому впізнаваний візуальний стиль.

Таким чином, для виконання мети дипломної роботи використано наступне ПЗ:

- Zbrush (скульптинг персонажів)
- Blender (hard surface моделювання, ретопологія, розгортка)
- Substance 3D Painter (текстурування)
- Adobe Photoshop (Іконки та інші елементи інтерфейсу)
- Unreal Engine (левел дизайн, скрипти)

Висновок

Отже, задачею випускної кваліфікаційної роботи є розробка гри на базі Unreal Engine 5. Цей рушій, не дивлячись на високі вимоги до обладнання має низький поріг входження для не-програмістів. Головною перевагою UE над його головним конкурентом Unity в контексті даної роботи є наявність в Unreal Engine блупринтів - системи візуального програмування, що представляє собою систему нодів з різноманітними даними .

Основні вимоги до практичної частини – простота у розробці, цікавий концепт та відповідність цьому концепту фінальної версії додатку. Гра буде

представляти собою гібрид шутера та Tower Defense, поєднуючи будування захисних турелей із стрільбою по хвилям противників.

Основні механіки – будівництво та апгрейд турелей, пасток та іншого під час стадії підготовки; відбиття атак противників на манер шутера під час стадії бою.

Для створення ігрових ассетів буде використовуватись 3D пакет Blender. Через свою доступність та універсальність він є найкращим вибором для невеликого інді проекту.

РОЗДІЛ 2. Розробка графічного контенту

2.1 Розробка 3D моделей

Розглянемо процес створення 3D моделей на прикладі розробки персонажа – головного героя, яким керує гравець.

Першим етапом створення будь-якого скульпту є блокінг, тобто створення болванки, яка формує силует моделі. Основними вимогами на цій стадії робочого процесу є утримання низького полікаунту, тобто кількості полігонів об’єкта. Причиною цього є те, що менш щільна полігональна сітка легше піддається деформації та згладжуванню, тож при створенні основних форм необхідно тримати низьку кількість полігонів [9].

За допомогою базових інструментів задаються основні форми та пропорції майбутнього персонажа. Це в першу чергу інструмент “Move”, який дозволяє «тягнути» меш, пензлі “Standart” та “Clay Buildup” – для додавання об’єму, і “DamStandart” – для створення щілей та поглиблень.

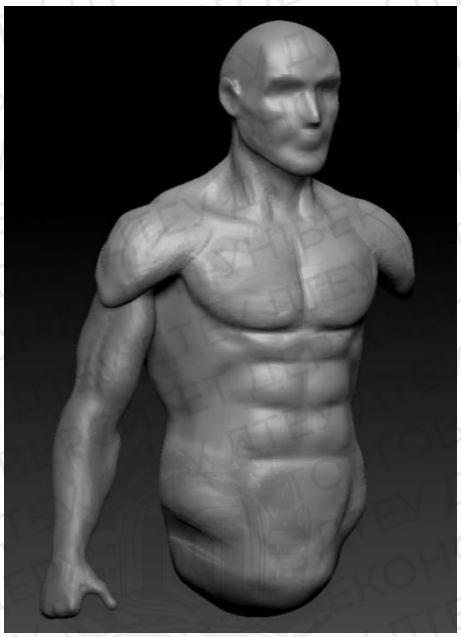

Рис. 2.1 Блокінг персонажа

Для блокінгу персонажа деякі частині тіла будуть створюватись окремо. Причиною для цього є те, що такі частини тіла, як торс, потребіють набагато меншої щільності полігональної сітки, ніж наприклад руки. Пальці рук, нігті, та інші деталі, є дуже малими за розміром, в той час як торс є набагато більшим за розміром, і не має таких малих деталей. І якщо робити і руки, і тіло одним об’єктом, то для всього об’єкту потрібно буде задати високий полукаунт, що в свою чергу є зайвим (оскільки для торсу непотрібно багато полігонів), а також

дасть більше невантаження на систему. Розділивши модель на декілька об’єктів, можна ефективніше використовувати ресурси процесора, що надасть можливість в подальшому більш комфортно працювати із складною геометрією.

На рис.2 видно, наскільки може відрізнятись щільність сітки на різних частинах тіла. На торсі (жовта полігрупа) ще видно, як просвічує поверхня моделі, в той час як рука (зелена полігрупа) повністю покрита чорним, тобто полігонів настільки багато, що відображення сітки затемнює поверхню моделі.

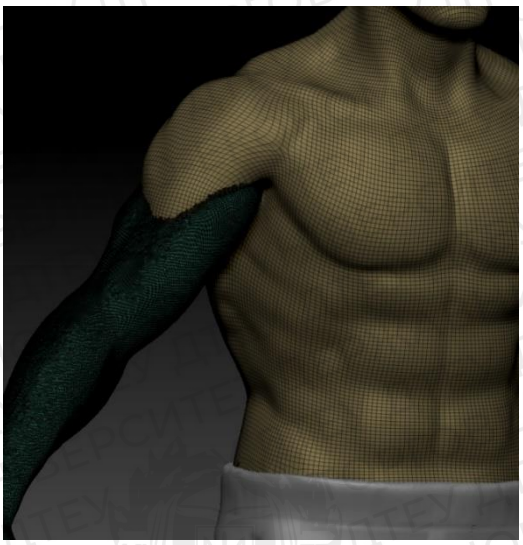

Рис. 2.2 Порівняння щільності сітки руки та торса

Після блокінгу необхідно проробити вторинні деталі, тобто проробити м’язи тіла, промалювати риси обличчя тощо. Надалі модель тіла буде використовуватись для створення інших елементів – одягу та шолому. Для цього за допомогою маски потрібно виділити область, з якої буде екструдитись (вичавлюватись) одяг. Після цього потрібно знов повернути сітку до відносно низького полікаунту – для цього використовуємо DynaMesh, який перебудовує сітку на основі наявної геометрії за заданим параметром “resolution”. Після цього одяг можна деформувати та промальовувати складки.

Рис. 2.3 Рендер скульпту персонажа

Готовий скульпт персонажа містить майже 7 мільйонів полігонів, і однієї такої моделі в сцені достатньо, щоб зламати гру, оскільки прорахування

такого полігонажу забере всі ресурси комп'ютера [10]. В цілях оптимізації необхідно зробити ретопологію, тобто створити нову модель, з набагато менш щільною сіткою, яка лише повторює силует скульпту. Як видно на рисунку 4, на низькополігональній версії (справа) видно окремі полігони, а на підборідді помітно яскраво виражені кути, через малу кількість граней.

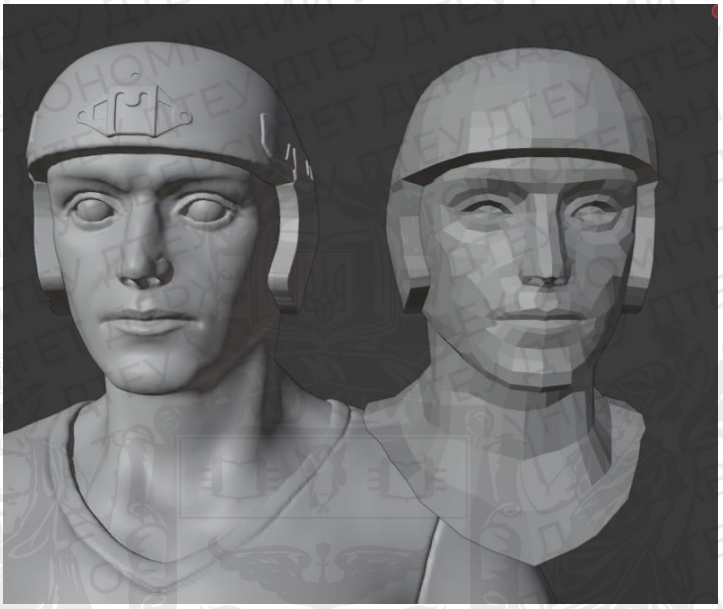

Рис. 2.4 Порівняння скульпту та ретопу

Ретопологію можна робити і всередині Zbrush, але Blender має вбудований аддон Bsurfaces, що робить процес набагато простішим. Як приклад, можна почати будувати нову топологію, намалювавши декілька ліній на поверхні моделі, після чого перетворити їх у геометрію. Окрім того, цей аддон автоматично налаштовує параметрі снапінгу (прив'язки, англ. snapping) таким чином, щоб нова геометрія «примагнічувалась» до скульпту, а також додає всі необхідні модифікатори.

Готову ретопологію необхідно розгорнути, тобто провести розгортку UVW координат. Цей процес передбачає створення на основі трьохвимірної моделі двомірного атласу, на який можна накладати текстуру. Для цього поверхню моделі покривають швами, по яким модель ділиться на частини, які «розтягують» на атласі. Основним завданням на цьому етапі є створення найменшої кількості швів при мінімальному розтягненні текстури.

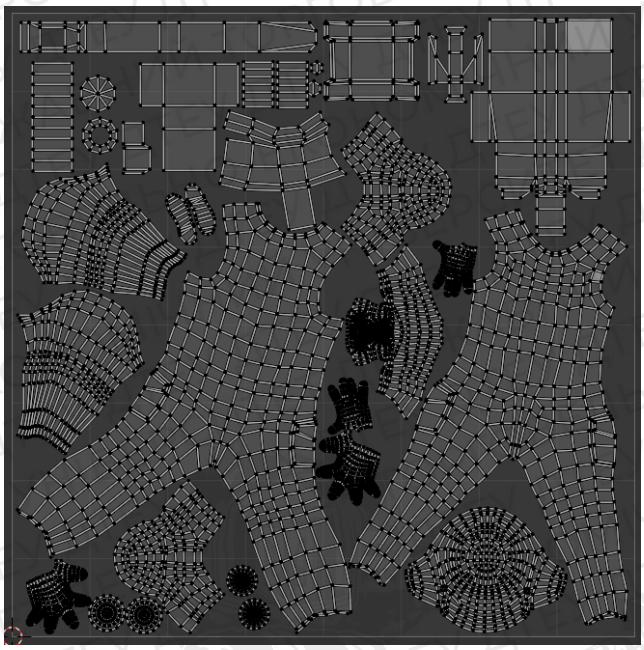

Рис. 2.5 UV-розгортка

Наступним кроком є текстурування моделі. Розгорнутий персонаж імпортується в Substance Painter, при створенні нового проекту вказуємо Normal Map Format – DirectX. Це означає, що мапа нормалей, яку ми створимо, матиме формат DirectX, а не OpenGL. Це необхідно тому, що рушій Unreal Engine використовує саме цей формат мап, і якщо вибрати невірний формат, то модель буде відображатися некоректно.

Перед початком фарбування моделі потрібно підготувати текстурні карти. Для цього всередині Substance Painter є функція “Bake mesh maps”. Ця функція створює на розгортці карти певний параметрів моделі, наприклад, Curvature, яка відображає викривлення моделі, і може використовуватися для виділення та покраски фасок, швів, та інших нерівностей.

В першу чергу, необхідно запікти карту нормалей, яка містить інформацію про високополігональну модель, тобто скульпт, і переносить його деталізацію на ретоп, дозволяючи досятгти високої деталізації при мінімальному полікаунті. Після цього можна запікати всі інші мапи. Важливо при цьому виставити в параметрах високу роздільну здатність, навіть якщо майбутня текстура буде значно менша, адже карта, запечена в 4к, і перероблена в 2к, буде виглядати набагато краще, ніж та, що одразу запечена в 2к.

Маючи готові текстурні карти, можна приступати к текстуруванню.

Фарбування моделі в Substance Painter базується на системі шарів, так само, як і в Adobe Photoshop. Нижніми шарами задається базовий колір, після чого наступні накладаються поверх за допомогою пензлів, заливок, масок, та інших інструментів. Окрім кольору, моделі задаються параметри roughnes (шорсткість), яка відповідає за відбиття/поглинання світла, metalness (металевість), що визначає кількість відображень та відблисків, та інші.

Спершу створимо декілька папок, для кожного елемента персонажа: для шкіри, для футболки, штанів, взуття та шолому, а також для зброї. Кожній з цих папок додаємо black mask, маску, яка за допомогою чорно-білого градієнту задає, чи буде відображатись цей шар/папка на даній поверхні, і якщо буде, то якою мірою. Таким чином ми ізолюємо ця папки одна від одної, і текстури не будуть перекривати одна одну.

Рис. 2.6 Модель, покрита базовими кольорами

Далі додаємо нові шари (fill layer), і покриваємо їх такою ж маскою, як і папки, але на цей раз вона використовується не для того, щоб ізолювати, а для того, щоб малювати ними поверх базового кольори, використовуючи пензлі та заливки.

Для прикладу розглянемо процес створення ластовтиння. Новому fill layer задається колір, темніший за загальний тон шкіри, на нього накладаєм чорну маску. Додаємо чотири елементи fill, тобто заливки. У grayscale канал перетягуємо 3D_simplex_noise, ratio_clouds_1, ratio_dirt_3 та ratio_dirt_4. Їм необхідно виставити режими змішування Normal, Multiply, Screen та Screen відповідно. Дані заливки відмічають на масці місця, де проявлятиметься даний шар. При цьому користувач може в будь-який час редагувати параметри balance та contrast, щоб змінити рисунок маски. Зверху додаємо

фільтр sharpen з параметром sharpen intensity 0.35. Далі додаємо елемент levels, який дасть контроль над маскою. Рухаючи повзунки маску можна зробити більш або менш яскравою та/або контрастною.

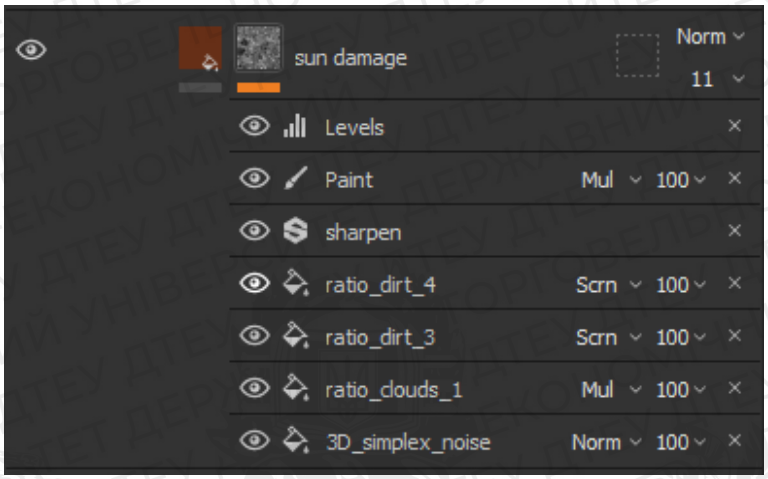

Рис. 2.7 Приклад налаштування fill layer.

Після створення текстур їх експортують у потрібному форматі у рушій або іншу 3D програму.

Рис. 2.8 Персонаж із текстурами

2.2 Ріггінг та анімація

Перед тим, як анімувати персонажа, для нього потрібно створити ріг, тобто скелет моделі, який визначатиме, які частини моделі і яким чином

рухатимуться. Особливістю рігу ігрових персонажів є те, що для них створюють 2 скелети – один, control rig, призначений для створення самих анімацій. Другий – deform rig, безпосередньо деформує модель згідно анімаціям, заданим control rig. Така система необхідна тому, що ігрові рушії не підтримують більшість тих функцій, які мають 3D пакети накшталт Blender, Maya та інші. Окрім того, при створенні control rig додається багато зайвих кісток, які не деформують поверхню, а використовуються як контролери. Приклад: налаштування інверсивної кінематики – для кожної кінцівки створюють по дві додаткові кістки (одна для контролю напрямку сгинання, і одна для рухання кінцівки). А тому потрібно запікати анімації на deform rig, аби не створювати зайву плутанину в ієрархії скелету.

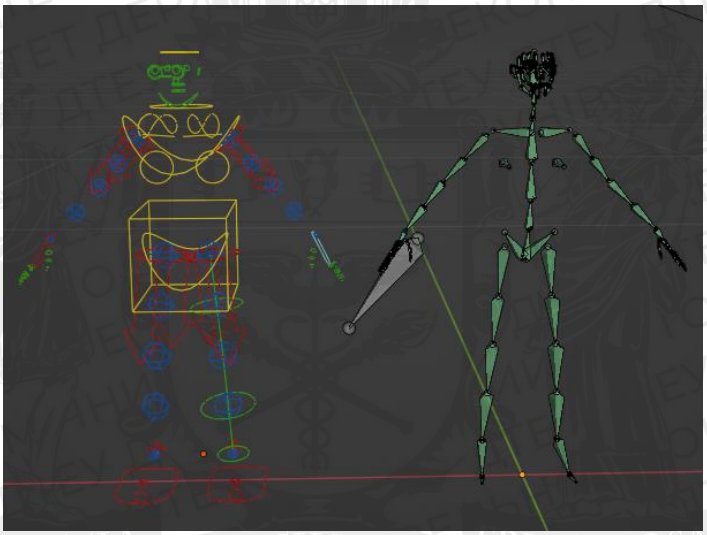

Рис. 2.9 Control rig (зліва) та deform rig (справа)

Для створення скелету моделі в Blender вбудований аддон rigify. Включити його можна, перейшовши у меню Edit/Preferences/Add-ons та знайшовши його у пошуку. Тепер, при доданні нового елементу у сцену, в группі armature з’явився Meta-Rig. Додаємо його у сцену, за необхідності масштабуємо. Необхідно розташувати кістки всередині моделі так, щоб вона коректно деформувалася. Також необхідно, щоб кістки кінцівок на місці суглобів з’єднувались під кутом, оскільки це необхідно для коректної роботи інверсивної кінематики.

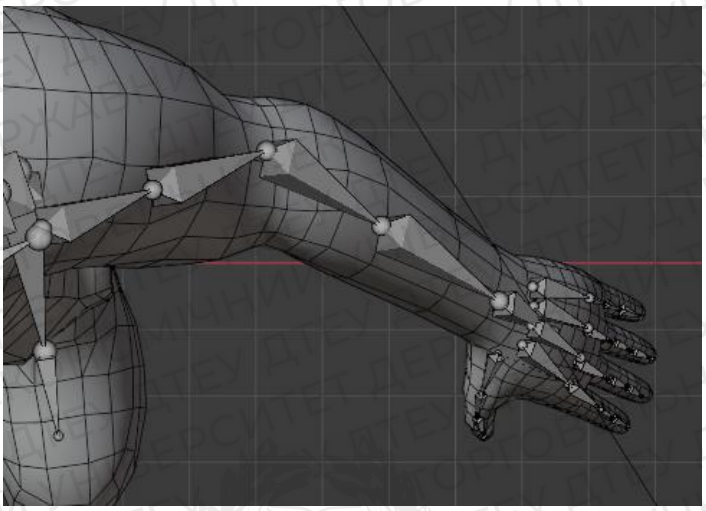

Рис. 2.10 Налаштування скелету моделі

Налаштувавши скелет, натискаємо «Generate Rig», отримуємо ріг на основі побудованого скелету. Вибираємо модель і ріг (саме в такій послідовності), і вибираємо Set patent to/Armature deform/With automatic weights. Таким чином модель прив'язується до рігу. Кожній кістці задається певна область, на яку вона впливає. Ця область задається у режимі Weight paint, який дозволяє малювати на поверхні модель синьо-червоний градієнт, який визначає, на які точки і як сильно впливає деформація даної кістки. Якщо поверхня пофарбована в синій колір, це означає, що вибрана кістка не має жодного впливу на неї, якщо в червоний – значить вплив максимальний. Інформація про вплив кожного елемента типу armature на модель зберігається в окрему вертексну групу, і користувач може в будь-який час їх редагувати.

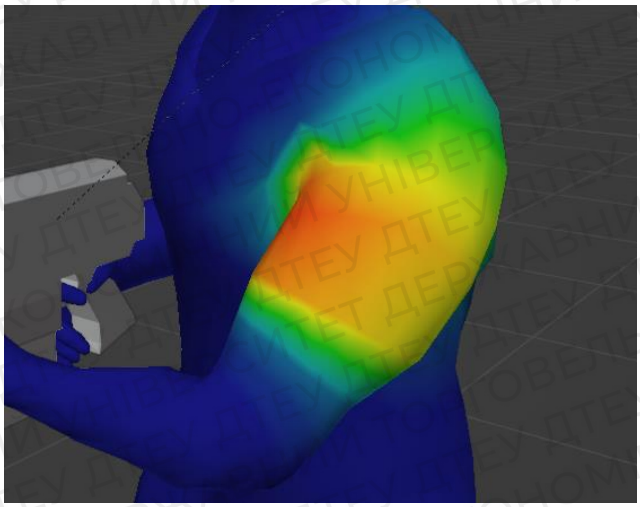

Рис. 2.11 Режим Weight Paint

Після створення рігу і прив'язки до нього моделі, модель починає повторювати рухи скелета. Але під час деяких рухів, наприклад, при сильному повороті кінцівок, бачимо, що модель сильно деформується. Для

виправлення подібних проблем використовують різні методи: іноді допомагає додання нових кісток, які змінюють динаміку скелета таким чином, що модель деформується більш коректно (прикладом є розділення передпліча на 3 кістки, з'єднані послідовно, для виправлення досить поширеної проблеми з перекручуванням руки під час повороту зап'ястя), також часто причиною проблеми є неправильний weigth painting, при якому кістка не отримує контроль над поверхнею, яку має деформувати, або деформує те, за що вона відповідати не повинна, а також використовують «ключі форми», shape keys. [11]

Рис. 2.12 Некоректне згинання руки

Виконаємо згинання руки персонажа у ліктьовому суглобі. Бачимо, що модель деформується некоректно – передпліччя входить у біцепс. Переходимо у Object data properties, додаємо новий shape key. Отримуємо базову форму персонажа, тобто основу, порівняно з якою поверхня буде змінюватись. Перший «ключ» ніколи не деформують, для цього додають нові. Створюємо shape key для згинання руки і виставляємо параметр value на 1. Після цього переходимо в режим редактування, і виправляємо недолік. Тепер ми можемо переключатись між базовою та виправленою формою моделі. Підвищуючи параметр value ближче до одиниці виправлення проявляється, а якщо навпаки – зникає. Далі натискаємо пкм на парамерт

value, і вибираємо add driver. Виставляємо наступні параметри:

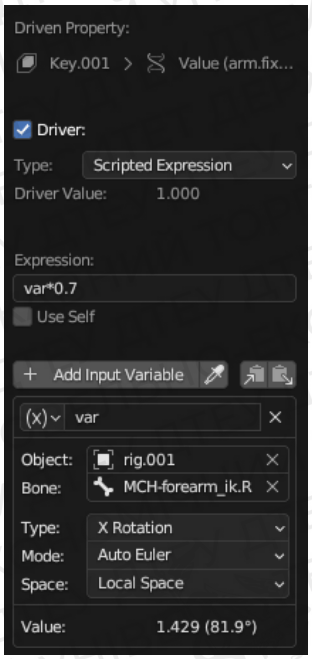

Рис. 2.13 Налаштування драйвера

Таким чином, ми прив’язуємо value до параметру повороту кістки передпліча таким чином, що при сгинанні руки до 82 градусів даний shape key повністю проявляється, а при прямій руці він відключений.

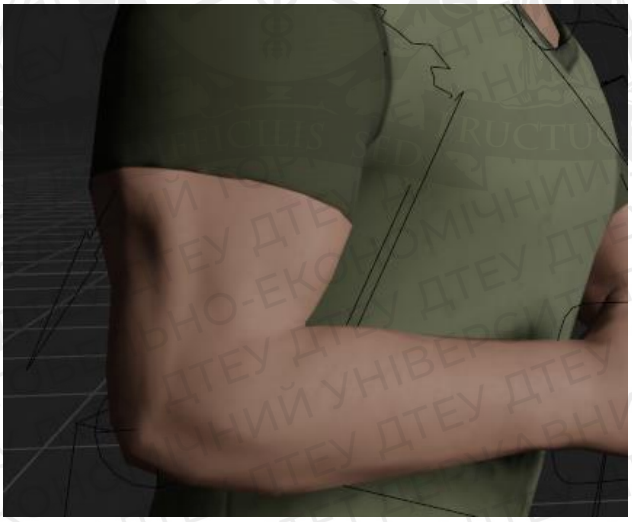

Рис. 2.14 Коректне сгинання руки при спрацьовуванні драйверу.

Для створення окремих анімацій необхідно перейти в Action Editor, і там додати нову анімацію. Всі розміщені на таймлайні keyframe-и будуть відноситись лише до цієї дії. Після цього на таймлайні (шкалі, на якій розмічені кадри анімації) розміщуються keyframe-и, тобто мітка, яка дає знак програмі, що на даному кадрі дана кістка мала наступні параметри. Серед параметрів можуть бути кут повороту, переміщення, масштабування кістки

тощо. Кількість фреймів в анамаціх задається користувачем, від цього залежить, наскільки довгою вона буде. Розмістивши два кейфрейми на шкалі, і запустивши анамацію, побачимо, що модель рухається від стану, заданого першим кейфреймом, до стану другого. Таким чином можна розмітити на шкалі всі необхідні трансформації.

Першим кроком є створення так званої king frame, тобто найбільш важливого кадру в анімації. Для прикладу візьмемо процес створення дії «sidewalk_left», тобто «ходьба вліво». В даному випадку king frame буде поза персонажа, при якому права нога приставляється до лівої. Розміщуємо цей кадр в центрі таймлайну, на 15 фреймі (тому що загалом ця анімація складається з 30-ти). На першому фреймі ставимо персонажа у початкову позицію, при якій персонаж просто стоїть, після чого поєднанням клавіш shift+d робимо дублікат цього фрейму на кінець анімації. Додаємо декілька допоміжних keyframe-ів, щоб зробити анімацію реалістичною, після чого приступаємо до зациклення анімації.

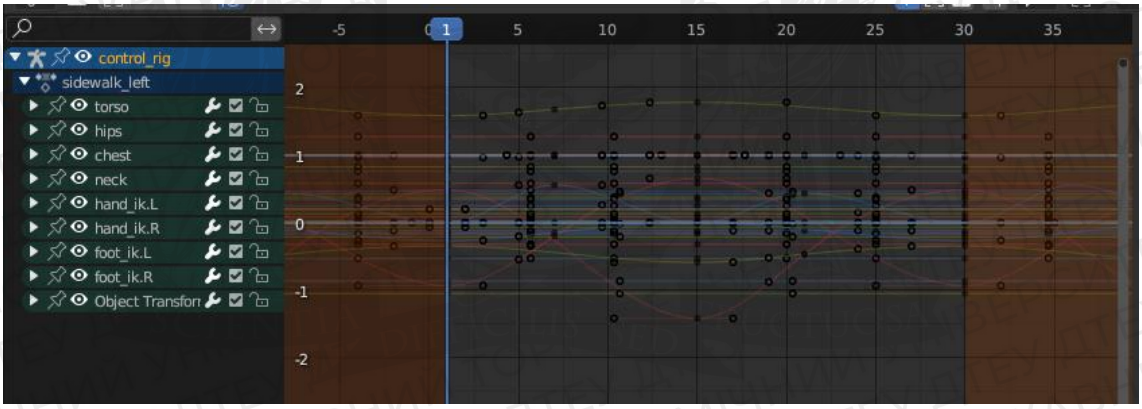

Рис. 2.15 Graph editor, зациклення анімації ходьби

Якщо анімація є повторюваною, як наприклад анімація ходьби, її треба зациклити. Для цього спочатку треба впевнитись, що перший та останній фрейм анімації є абсолютно ідентичними. Далі треба відкрити Graph Editor, і вибравши всі кістки, які задіяні в анімації, додати модифікатор Cycles. Ван створить дублікати кривих зміни параметрів перед і після анімації, що змінює форму кривих на першому і останньому кадрі так, ніби після останнього кадру знову йде перший.

Рис. 2.16 Готовий до експорту анімований персонаж

Таким чином створюємо всі необхідні анімації, і експортуємо їх разом із персонажем в рушій.

2.3 Розробка графічного дизайну

Створення графічного дизайну для даного проекту обмежується створенням елементів інтерфейсу, таких як іконки, меню та різноманітні позначки (приціл, стрілки тощо).

Розглянемо процес створення логотипу іконки гри (портрету), яка в майбутньому буде її портретом в магазинах додатків накшталт Play Market.

Візьмемо модель іншого персонажа цієї гри, зомбі, і зробимо рендер таким чином, щоб освітлення не відповідало забарвленню фона. Для цього в Blender перейдемо у Material editor, і відкриємо “World”. Бачимо налаштування шейдеру оточення і базовий набір нод: Background та World Output. Копіюємо ноду Background, змінивши її колір на інший, наприклад, червоний. Поєднуємо ці дві ноди за допомогою Mix Shader, в якості factor використовуємо вихід Is Camera Ray ноди Light Path. Таким чином, один Background освітлює саму модель, а інший – оточення.

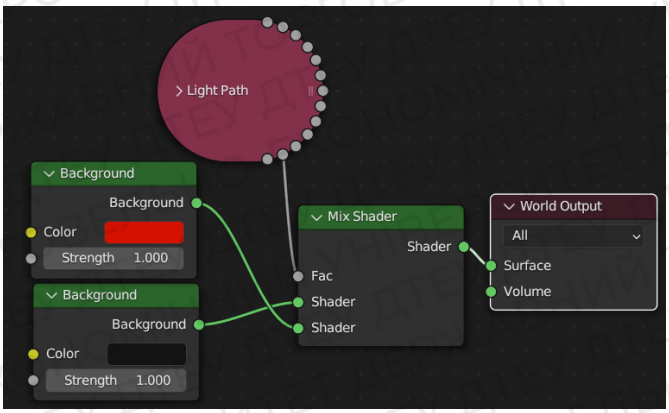

Рис. 2.17 Налаштування фону

Рендеримо дане зображення, і відкриваємо його в Adobe Photoshop. Вибираємо інструмент «чарівна паличка» та виділяємо червоний фон, вирізаємо його на новий шар. Вибираємо два віттінка синього кольору, і перетворюємо шар з фоном на градієнт. Після цього створюємо фігуру лінія, змінюємо її колір на чорний, і розміщуємо на краю зображення. Робимо 3 її копії, та переміщуємо до інших країв, таким чином отримуємо рамку.

Рис. 2.18 Іконка гри

Висновок

Створення графічного контенту для гри – це складний і багатофазовий процес, де кожен наступний крок може виконуватись в іншому програмному забезпеченні. Розглянуто процес створення моделі на прикладі розробки персонажа - головного героя. Його скульпт був виконаний в Zbrush, після чого його експортували в Blender для ретопології та розгортки. Після цього модель була затекстурована у Substance Painter, і повернута назад в Blender для анімації.

Особливостями робочого процесу створення 3D моделей для гри є необхідність оптимізації (низький полікаунт, використання мапи нормалей тощо), а також специфічна система створення скелету, при якій один використовується для створення анімацій, після чого з нього анімації "запікаються" на інший скелет, без зайвих кісток та контролерів.

Окрім того, було показано процес розробки іконки (портрету) гри, при якому задіяно як функціонал 3D програм (рендер моделі, маніпуляції з шейдером оточення), так і функціонал графічного редактора Adobe Photoshop.

РОЗДІЛ 3. Розробка гри у Unreal Engine

3.1 Імпорт ассетів у рушій

Розробка гри в середовищі Unreal Engine починається зі створення проекту та імпорту ассетів. В якості шаблону був вибраний Blank (пустий пресет), без стартового контенту та raytracing-у.

Після цього на диску створюється відповідна директорія, в яку буде завантажуватись весь контент. Далі треба зайти в Blender, і експортувати звідти моделі, вказавши в полі Transform параметри «Forward» - «X» та «UP» - Z. Для об'єктів, що мають анімації також необхідно вибрати «Bake

animations» та прибрати галочку «Add leaf bones». Після цього можна повертатись до Unreal Engine та імпортувати ассети. Для різних груп моделей створюються різні папки для більш зручної навігації, а також окремі папки для віджетів та анімацій.

Після цього створюються blueprint класи, які перетворять імпортовані моделі на об'єкти, які можна використовувати для побудови логіки гри.

Actor. Даний клас представляє собою об'єкт, який може бути розміщений або «заспавнений» у просторі. Використовується для різноманітних елементів оточення, зброї, снарядів тощо. Саме він був назначений таким об'єктам, як Generator та Landmine, оскільки вони не рухаються і не контролюються гравцем[12].

Pawn. Цей клас є видом Actor, що може бути відданий під контроль гравцю. Окрім того, він легко приймає ввідні дані, відтворює анімації, та робить багато речей подібно гравцю. Цей клас був назначений турелям, оскільки це дозволяє використовувати PawnSensing – компонент, який дозволяє створити логіку системи наведення на ціль. Окрім того цей клас має Enemy_AI, штучний інтелект для противників.

Character. Підвид Pawn, який більше пристосований до керування, та за замовченням має Capsule Component для колізії та Character Movement Component для реалізації логіки ходьби, був назначений персонажу, яким керує гравець.

3.2 Розробка ігрових механік

Керування персонажем

Створюємо клас GM_GameMode типу Game Mode Base, вибираємо BP_Char, який представляє собою клас гравця, як Default Pawn Class. У World Settings вибираємо GM_GameMode як GameModeOverride. Таким чином гравець на початку гри матиме контроль над персонажем класу BP_Char, тобто ігровий процес буде йти від імені цього класу.

Перед налаштування логіки керування в одній із директорій потрібно створити Mapping Context, де назначаються відповідні гарячі клавіші (WASD, пкм, лкм тощо). У Event Graph класу BP_Char після події EventPlay створюється посилання на Player Controller, який підключається до Enhanced Input Local Player Sub System; за умови, що контролер наявний (Is Valid?), додається вибраний Mapping Context. За допомогою подій InputAxis

прив’язується напрямок (вісі Y та X) капсули персонажа до параметра world direction ноди Add Movement Input. Таким чином, при натисканні клавіш WASD персонаж рухається у відповідну сторону. Схожим чином налаштовується обертання камери, де axis value події Axis Value, яка відповідає за оберт камери по одній з осей, прив’язується до Add Controller Yaw/Pitch Input.

В анімаційному блупринті параметри швидкості та кута повороту персонажа перетворюються на змінні, які використовуються для контролю Blendspace анімації, тобто файлу, який визначає, як одна анімація переходить в іншу. Таким чином, кут повороту визначає, яку саме анімацію ходьби використовує персонаж, а швидкість визначає, чи будет він рухатись взагалі.

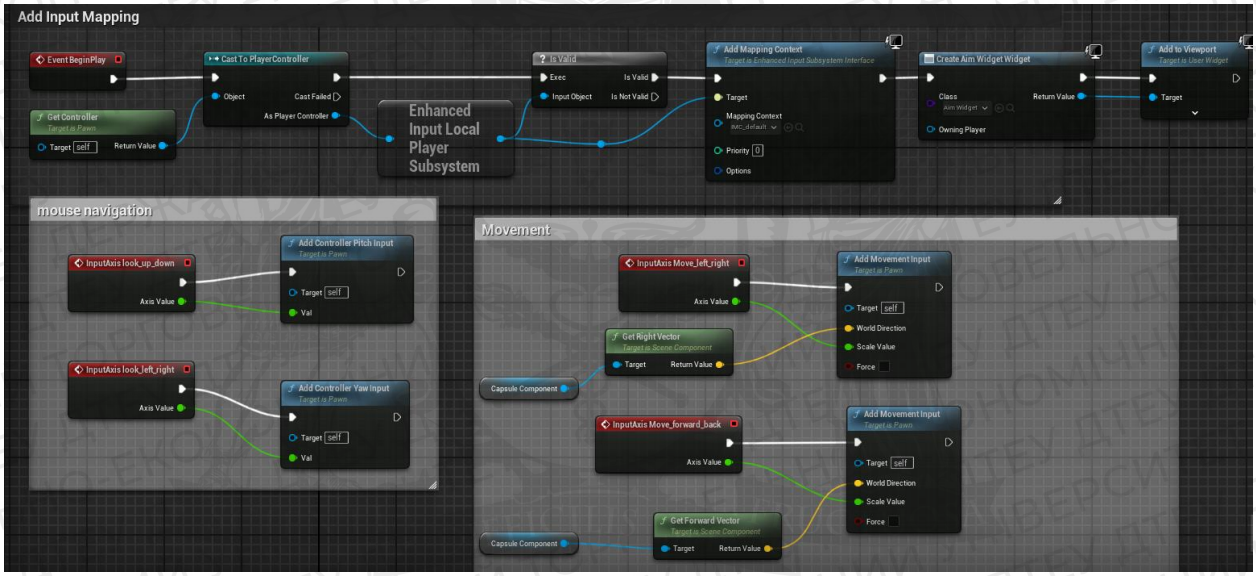

Рис. 3.1 Налаштування ходьби та повороту камери

Прицілювання та постріл персонажа

Enhanced Input Action IA_Fire (лкм), спрацьовуючи, викликає ноду Do Once, підключену до черги, яка пропускає виконання логіки далі, після чого після затримки (яку задає користувач) повертається в поле reset ноди Do Once. Таким чином, при затисканні лкм кількість пострілів буде обмежена. В іншому випадку постріл би спрацьовував десятки разів за секунду. Далі виконується нода Line Trace By Channel, яка прораховує траєкторію пострілу і генерує події, пов’язані з влученням. В якості параметру Start використовується вектор положення сокету, який відповідає дулу гвинтівки, та вектору, отриманому із параметра повороту камери. Якщо постріл попав у колізію, то викликається нода Apply Damage, яка накладає пошкодження до

об’єкта классу actor, що був зачеплений пострілом. Для пошкодження створюється окрема змінна.

В анімаційному блупринті створюємо State Machine. За допомогою неї відбувається перехід від одного стану до іншого, в даному випадку від Idle/Walking, тобто звичайного стану, коли гравець не прицілюється, до стану, коли він цілиться. Перехід відбуватиметься за допомогою логічної змінної IsAiming?, яка стає істинною, коли гравець настискає праву клавіши миші, викликаючи подію EnhancedInputAction. При зміні IsAiming? на true персонаж виконує анімацію прицілювання. Окрім того, в стані Aiming анімаційного блупринту скелет персонажа за допомогою ноди Layered Blend Per Bone ділиться на верх і низ, щоб анімація прицілювання не переривала анімацію ходьби.

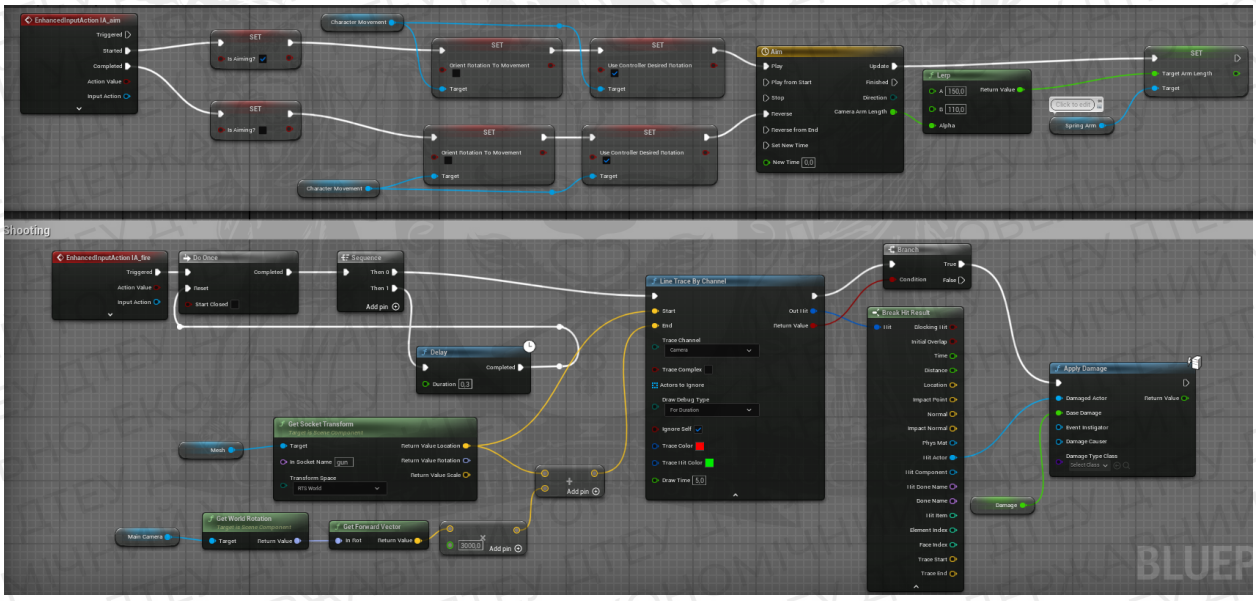

Рис. 3.2 Система прицілювання та стрільби персонажа

Штучний інтелект противників та смуги hp

Після спрацювання події BeginPlay, і невеликої затримки, викликається нода Get All Actors Of Class, яка передає копію об'єкту класу Target Point у поле target actor ноди AI Move To. В якості параметра Pawn – посилання на самого себе (тобто на клас Enemy_AI). При появі противника на карті він починає рухатись до Target Point, яка виставляється у редакторі. Але для того, щоб він міг прокладати маршрут, потрібно використати компонент NavMeshBondsVolumue. Розмістивши його на об'єктах, яким назначена колізія, він прораховує область, де можливе переміщення.

Для отримання пошкоджень до Event Graph треба додати подію Event Any Damage. Також створюється змінна, яка відповідає за кількість життя (хітпоінтів) противника, і при кожному виклику події від цієї змінної віднімаються кількість завданих пошкоджень. Тепер потрібно, щоб ці дані передавались у смугу здоров'я.

В сцену добавляється віджет, де створюється елемент progress bar. Параметр % заповненості цієї смуги повертається у вигляді змінної Zombie_Health. У блупринті Enemy_AI створюється функція UpdateHPBar, яка, посилючись на віджет, оновлює змінну Zombie_Health. Ця функція викликається кожен раз при спрацюванні Event Any Damage. Якщо змінна Zombie_Health опускається до нуля, програється анімація смерті, після чого противник зникає (Destroy Actor).

Якщо противник успішно дійшов до Target Point, викликається функція Zombie_Hit_Generator, яка посилається на блупрінт Generator (головний об'єкт, захищати який є метою гри), і завдає йому пошкодження.

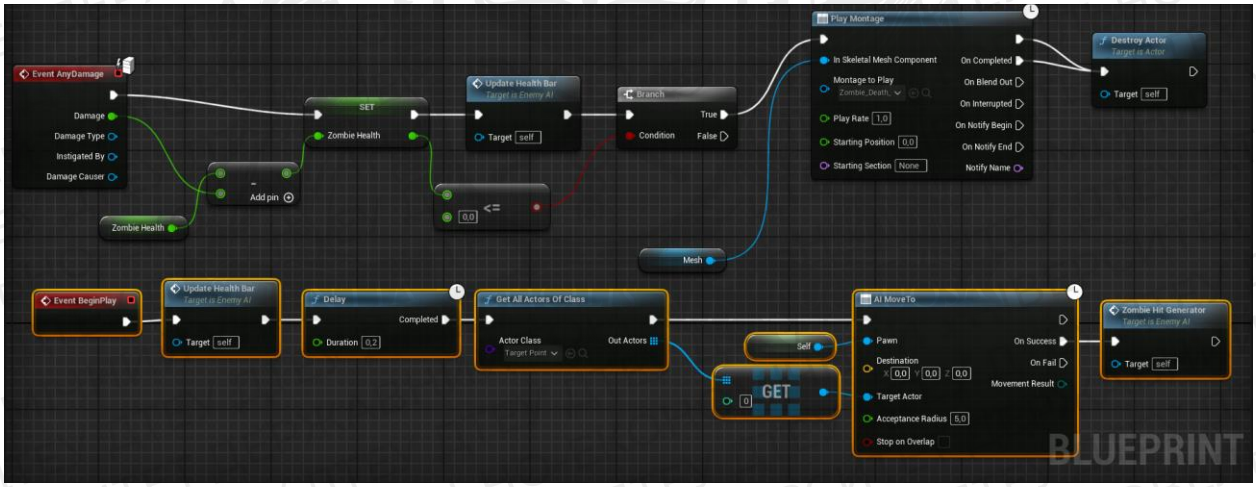

Рис. 3.3 Розробка штучного інтелекту противника

Турелі. Спавн та логіка роботи

У відповідній директорії створюється віджет Turret Selection, в який додаються 4 кнопки. Три з них які відповідають за розміщення одного з видів турелей, а одна – за закриття меню вибору. Кожна має подію Event On Clicked; при натисканні на кнопки, що відповідають за вибір турелі, викликається нода SpawnActor, і в якості параметра spawn transform їх задається місцезнаходження TowerPod – об'єкту, на який буде розміщуватись турель. Після цього логічна змінна CanSpawnATurret? набуває значення false, а сам віджет закривається. Четверта кнопка, TurretSelectionExit, відразу закриває віджет.

Створюється новий блупрінт TowerPod (поле, де розміщується турель) класу Actor, матеріал та Material Parameter Collection з такою самою назвою. В Material Parameter Collection створюється 5 векторних параметрів, кожен з яких несе певний колір – жовтий, зелений, червоний, чорний, та ще один чорний. Останній буде називатися PodColor, саме від буде відображались як base color матеріала, і буде змінюватися залежно від дій гравця.

У блупринті TowerPod додаємо події OnBeginCursorOver, OnEndCUrsorOver, OnClicked та OnReleased. При настанні однієї з цих подій викликається Get Vector Parameter Value, і записує відповідний колір у PodColor за допомогою SetParameterValue. Таким чином, при наведенні курсора на поле воно підсвічується жовтим, коли курсор перестає наводитись – повертається до дефолтного чорного; при натисканні на поле, за умови, що змінна CanSpawnATurret? має значення true, викликає віджет TurretSelection, якщо false – підсвічує поле червоним. Після вибору турелі або при відпусканні лкм поле знову повертається до чорного кольору.

Модель кожної турелі складається з двох частин – з основи та башти, яка здатна обертатися. Також до кожного з трьох класів турелей додається компонент PawnSensing, який виявляє цілі у певному радіусі. З цим компонентом пов'язана подія OnSeePawn, яка спрацьовую з періодичністю, визначеною параметром sensing interval. При виявленні цілі система перевіряє, чи є вона противником, і якщо є, націлює башту на ціль за допомогою Set World Rotation. Виявлення потрібного куту повороту відбувається завдяки порівнянню власного місцезнаходження та місцезнаходження цілі нодою Find Look At Rotation. Поворот здійснюється завдяки ноді Rinterp to, яка повертає поворот по осі Z (Yaw) у Set World Rotation та Set Actor Rotation, який слідую далі. Для того, щоб основа не поверталась разом із баштою, треба за допомогою Set World Rotation задати їй кут повороту 0.

Далі вводиться логічна змінна EnemyIsInRange?, яка за замовчуванням є false. Після виконання коду, який слідує за EventOnSeePawn, вона стає true, що робить можливим стрільбу турелі. Event Begin Play викликає ноду Set Timer By Event, яка підключається до кастомного івенту Turret Shooting, який буде спрацьовувати згідно заданому інтервалу. При кожному спрацьовуванні таймеру система перевіряє змінну EnemyIsInRange?, і якщо вона істинна, то створює об’єкт Ammo (відповідний типу турелі). В якості початкової точки спавну використовується кастомний scene component, параметри якого можна отримати за допомогою Get World Transform. Після створення патрона

система знову встановлює змінну EnemyIsInRange? як false, що дозволяє перевіряти наявність противників у полі зору перед кожним пострілом.

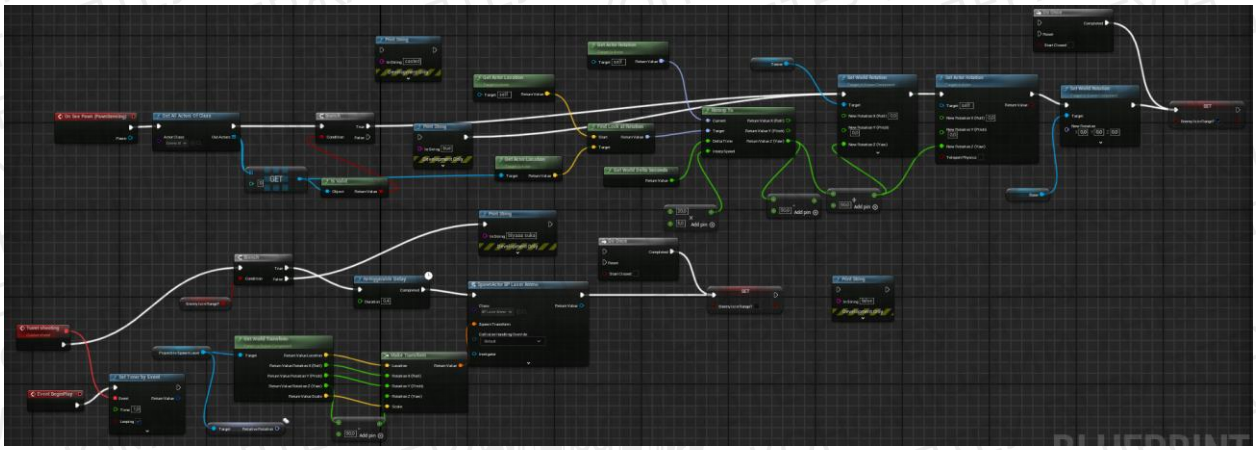

Рис. 3.4 Система наведення та стрільби турелі

Генератор та умови програшу

Для більшої наочності гравцю потрібно бачити, скільки хітпоінтів залишилось у генератора, адже якщо вони впадуть до нуля – гра закінчиться. Для цього створюється новий віджет WB_Generator_HP_Counter, в який додається 2 текстових поля, одне з яких є просто статичним надписом «Generator HP», а інше – лічильником, який оновлюється кожен раз, коли генератор отримує пошкодження. Текст лічильника прив'язується до функції Update Gen HP, яка перетворює зміст поля у числовий формат integer та записує туди змінну Generator Health класа Generator.

В блупринт генератора вводиться логічна змінна GameIsOver?, яка перевіряє, чи не є раунд програним. При кожному виклику події Any Damage система перевіряє, чи йде досі гра, і якщо так, то від змінної Generator Health віднімається число нанесених йому пошкоджень, після чого виклакається функція Update Generator HP, яка повертає оновлені дані у лічильник хітпоінтів. Якщо число Generator Health після цього буде менше або рівне нулю, то викликається функція Game Over, а змінна GameIsOver? стає true.

Функція Game Over створює на екрані віджет WB_GameOverScreen, який містить напівпрозорий червоний фон, який покриває весь екран, надпис «Game Over» та 2 кнопки – «Try Again» та «Quit», одна з яких починає гру заново, а інша – виходить з гри. Кожна кнопка має On Click Event, і при натисканні Try Again викликається нода Open Level By Name, де вказаний поточний рівень, а при натисканні Quit – нода Quit Game.

На початку раунду цей віджет додається до інтерфейсу, і йому передається значення Generator Health за замовчуванням. Після цього логічна змінна Set Show Mouse Cursor отримує значення false, що в свою чергу ховає курсор під час раунду.

Окрім того протягом раунду працює лічильник Set Timer By Event, підключений до кастомного івенту RoundIsOverCheck. Посилаючись на клас Spawn_AI, він виставляє змінну RoundIsOver? цього класу як true за умовою, що змінна Zombie Count класу Spawn_AI менша або рівна нулю, та змінна Generator Health більше нуля. Таким чином система перевіряє змінні перед переходом в новий раунд.

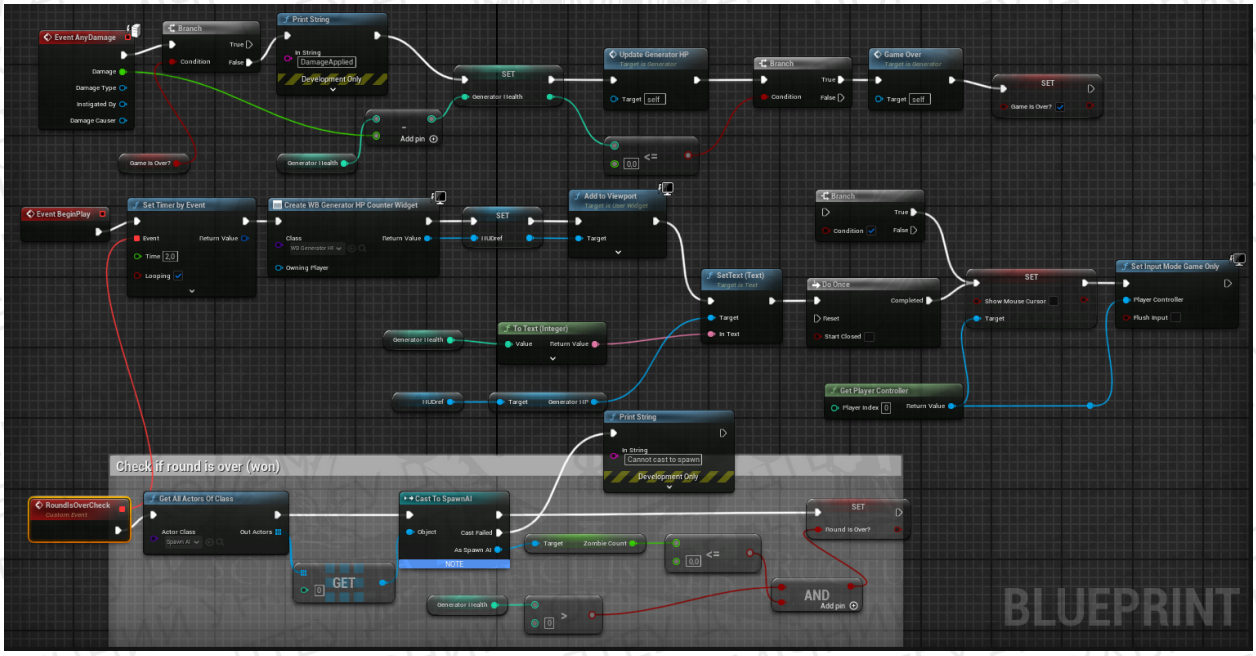

Рис. 3.5 Клас Generator

Система спавну та перехід у наступний раунд

Event Begin Play класу Spawn_AI викликає одразу два таймери Set Timer By Event. Перший підключається до кастомного івенту Spawn Enemy. У класа є змінна ZombieCount, і якщо вона більша нуля, то спрацьовує Spawn AI From Class, яка створює об'єкт класу Enemy_AI на місці SpawnPoint, при цьому віднімає 1 від числа ZombieCount. Коли ZombieCount стане рівним нулю,

логічна змінна ?WaveIsGone стане true.

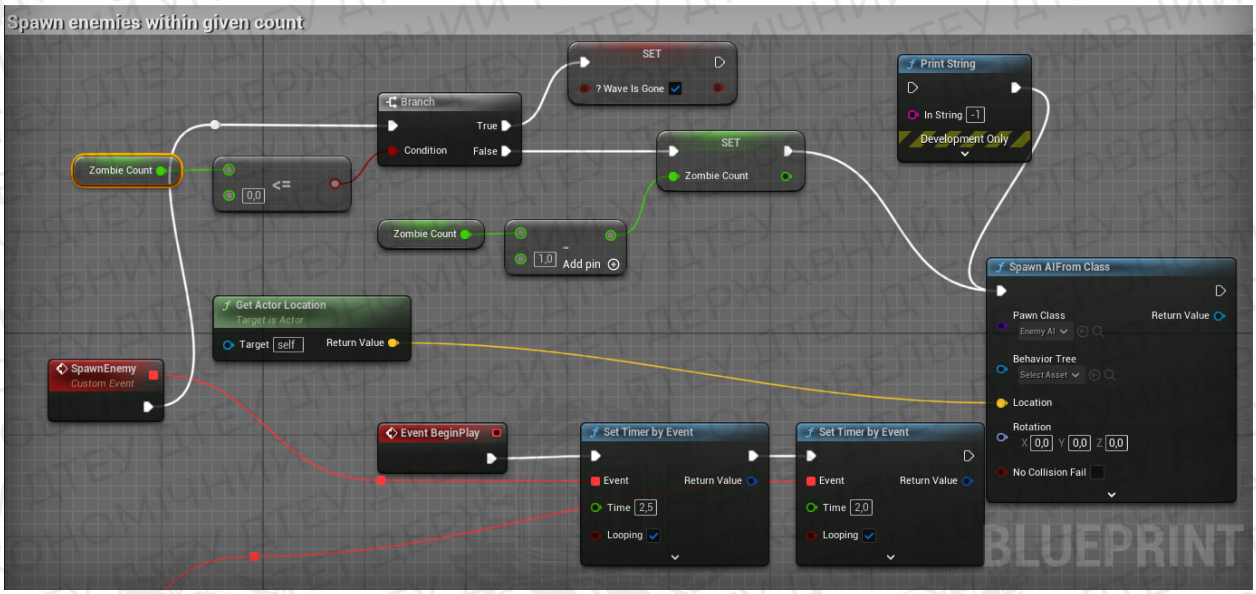

Рис. 3.6 Система спавну противника

Паралельно з цим спрацьовує інший таймер, підключений до кастомного івенту Check Round Win, який порівнює ряд змінних: якщо RoundISOver? та ?WaveIsGone мають значення true, і на рівні не лишилось валідних копій класу Enemy_AI, тобто якщо генератор уцілів, всі зомбі ліквідовані, і більше не будуть спавнитись, то раунд вважається виграним. В такому випадку за допомогою ноди Posses гравець бере під контроль об’єкт класу BP_UpgradeMode.

Після передачі контролю BP_UpgradeMode прибирає зайві віджети, та вмикає Show Mouse Cursor, Enable Click Events та Enable Mouse Over Events, що дозволяє курсору вільно взаємодіяти з об’єктами та віджетами, таким чином відкриваючи функції, які були недоступні під час раунду бою.

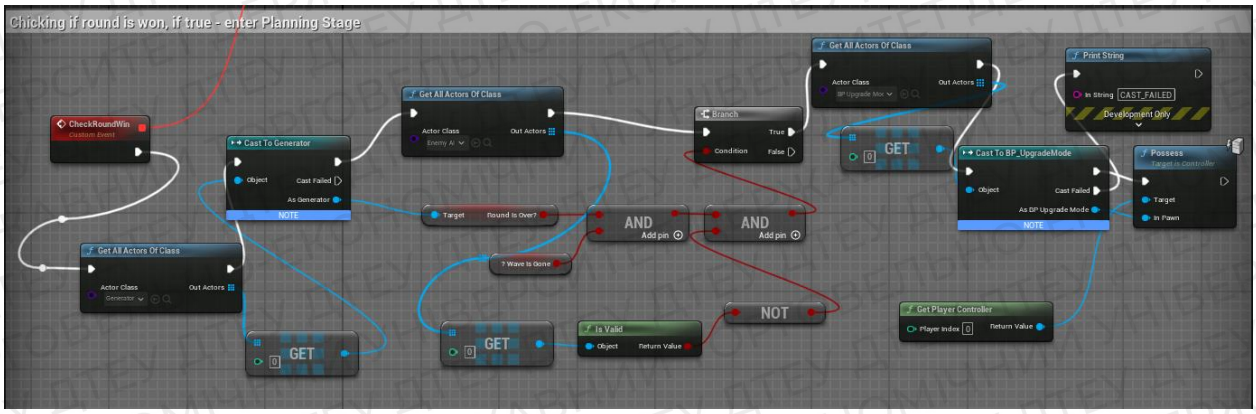

Рис. 3.7 Перевірка завершення раунду

3.3 Критерії оцінювання

Перед тим як розпочати тестування, необхідно детально розробити критерії оцінювання гри, враховуючи її унікальні особливості. Метою тестування даної гри є не лише перевірка працездатності, але й виявлення потенційних помилок та дефектів, які можуть впливати на коректну гру. Оскільки це одиночна гра, що поєднує в собі елементи шутера від третьої особи та стратегії в області захисту веж, критерії та завдання тестування повинні враховувати цю унікальну комбінацію.

Під час перевірки працездатності важливо з'ясувати, чи оптимізована гра належним чином, виявити можливі просадки кадрів на різних системах, визначити можливі зависання чи викидання. Додатково слід перевірити, якість геймплею, чи гравець може взаємодіяти з різними механіками (рух, стрільба, будівництво турелей тощо).

При виявленні потенційних багів слід уточнити, чи правильно виконуються задані механіки, зокрема, чи коректно працює система керування гравцем, його можливості стріляти, будувати турелі та взаємодіяти з противниками. Важливо переконатися, що гравець не має можливості застрягнути в різних частинах локації, щоб уникнути «провалів у текстурах»

Комплексне тестування гри на основі визначених критеріїв дозволить забезпечити високу якість та оптимальну функціональність продукту перед його випуском.

Отже, список механік, які тестуватимуться:
1) Рух персонажа:
Ходьба персонажа має відповідати заданим гарячим клавішам та правильно реагувати на зміну напрямку/швидкості, відображаючи відповідні анімації.
2) Спавн противників, їх рух до цілі та взяємодія з оточенням:
Противники мають з’являтись на заданому місці згідно прописаному інтервалу, і рухатись до об’єкту Target Point, оминаючи турелі чи інші об’єкти.
3) Стрільба гравця, вплив пострілу на смугу здоров’я противників:
Постріл має бути направлений до центра камери, де розташований віджет прицілу, та віднімати кількість здоровья, задану змінною Damage. Зміна кількості здоров’я цілі повинна відображатись на його смузі hp.
4) Стрільба турелей, вплив їх пострілів на смугу здоров’я противників
Турелі мають корректно реагувати на появу противників, наводячись на них і розпочинаючи стрільбу. Кожен постріл має генерувати відповідний снаряд, та віднімати задану відповідною змінною кількість здоров’я. Оновлення у кількості здоров’я цілі повинно відображатись на його смузі здоров’я.

5) Втрата хітпоінтів генератора, коли противники доходять до нього:
При успішному прибутті противника до Target Point, генератор втрачає кількість хітпоінтів, визначену змінною Zombie_Damage, після чого цей противник зникає з карти. Зміна хітпоінтів має відображатись у лічильнику відповідного віджета.
6) Програш раунду:
Коли хітпоінти генератора опустяться до 0, гра показує вікно Game Over.
7) Рестарт гри:
При натисканні кнопки Try Again рівень має завантажуватися заново.
8) Вихід з гри
При натисканні кнопки Quit гравець повинен виходити з гри.
9)Перехід у наступний раунд:
Якщо всі противники були ліквідовані, їх кількість у хвилі закінчилась, а хітпоінти генератора досі більше нуля, гравець переходить у стадію планування, в якій може будувати та апгрейдити різні елементи оборони. На цій стадії він не керує персонажем, а тому не може стріляти, але натомість може керувати курсором. Разом з появою курсору включаються Mouse Over та Click івенти.
10) Будівництво турелей:
При наведенні курсора на об’єкт TowerPod він має змінювати колір на жовтий, при натисканні лкм стає зеленим – яко турелі на цьому полі ще нема, і червоним, якщо вона є. Після вибору турелі, натисканні на віджет з хрестиком, або відведенні курсора, поле повертається до чорного кольору.

3.4 Результати тестування

Працездатність та швидкодія

Під час тестування був перевірений FPS гри на різних її етапах, і на кожному він тримав стабільні 60 кадрів на секунду. Зависань чи вильотів також виявлено не було. Під час тестування кількість противників була штучно збільшена вдвічі завдяки зменшенню інтервалу між спавном та збільшенню загального числа противників. Але навіть в цьому випадку FPS зменшувався на 2 кадри/секунду, що не є критичним.

Тестування механік

1) Рух персонажа:
Ходьба персонажа працює правильно, персонаж рухається відповідно заданим гарячим клавішам, та виконує відповідні анімації
2) Спавн противників, їх рух до цілі та взаємодія з оточенням:

Противники з’являються на spawn point згідно прописаному інтервалу, і одразу прямують до об’єкту Target Point. Проблем з пошуком маршруту чи зависань на місці не виявлено.

3) Стрільба гравця, вплив пострілу на смугу здоров’я противників:
Постріл направлений до віджету прицілу, нанесення пошкоджень працює і відображається правильно.

4) Стрільба турелей, вплив їх пострілів на смугу здоров’я противників
Турелі мають реагують на появу противників у зоні видимості, правильно наводяться і стріляють. Під час пострілу був виявлений баг, при якому пошкодження наносилися одразу 4 рази. Причиною багу стало те, що снаряд окрім capsule component класу Enemy_AI перетинає ще 3 колізії. Тому необхідно було після спрацьовування OnHitEvent додати перевірку класу, з яким зіткнувся снаряд, і викликати ноду Apply Damage тільки у випадку зіткнення з об’єктом класу Enemy_AI. Після даного виправлення механіка пострілу працює коректно.

5) Втрата хітпоінтів генератора, коли противники доходять до нього:
Противники, які дійшли до Target Point, наносять пошкодження генератору і зникають. Зміна хітпоінтів відображається у лічільнику у правій частині екрану.

6) Програш раунду:
Коли генератор втрачає всі хітпоінти, з’являється вікно Game Over.

7) Рестарт гри:
При натисканні кнопки Try Again рівень починається заново.

8) Вихід з гри
При натисканні кнопки Quit гравець виходить з гри.

9)Перехід у наступний раунд:
При знищенні всіх противників, та при умові додатнього числа хітпоінтів генератора, гра переходить у стадію планування. З’являється курсор, який генерує може взаємодіяти з об’єктами.

10) Будівництво турелей:
При виборі турелей чи простому наведенні курсора поле для турелі змінює колір на відповідний.

Висновок:

Розробка гри на Unreal Engine на блупринтах базується на взаємодії різноманітних блупринт-класів із іншими об’єктами, такими як Input Actions, тобто ввідні дії (приклад – гарячі клавіші), матеріали, анімації, візуальні ефекти тощо. У результаті розробки було створено систему управління персонажем, прицілювання та пострілу, прописано штучний інтелект для противників та турелей, реалізовано систему hp/пошкоджень, спавну противників, переходу між раундами, побудови турелей тощо.

Створені ігрові механики були протестовані на працездатність, так само як і швидкодія гри. Результати тестування свідчать про хорошу оптимізацію та коректну роботу розроблених механік, включаючи управління, взаємодію об'єктів та поведінку ШІ, а виявлені недоліки були виправлені у процесі тестування.

Висновки

Розробка комп'ютерних ігор є однією з найбільш актуальних і перспективних галузей сучасної індустрії розваг. Швидкий технологічний прогрес, постійне удосконалення графічних та 3D-редакторів, а також зростання популярності віртуальної реальності сприяють зростанню інтересу до індустрії. На фоні пандемії ігрова індустрія отримала новий поштовх до розвитку через карантинні обмеження, які закрили мільйони людей вдома.

Метою написання вкр є розробка гри на базі Unreal Engine 5. Для її виконання було виконано наступні завдання:

1) Моделювання ігрових ассетів
2) Текстуринг
3) Ріггінг та анімація
4) Розробка ігрових механік
5) Тестування

Розробка гри – це складний і багатофазовий процес, де кожен наступний крок може виконуватись в іншому програмному забезпеченні.

Під час виконання ВКР було використано наступне ПЗ:

- Unreal Engine 5
- Blender
- Zbrush
- Substance 3D Painter
- Adobe Photoshop.

Результатом стала готова і функціонуюча гра, гібрид жанрів Шутер та Tower Defence, яка відповідає заданому концепту, та успішно пройшла тестування на працездатність і швидкодію.

Список джерел:

1. How The Gaming Industry Has Leveled Up During The Pandemic
Bartosz Skwarczek, Forbes Jun, 17, 2021
https://www.forbes.com/sites/forbestechcouncil/2021/06/17/how-the-gaming-industry-has-leveled-up-during-the-pandemic/?sh=3d0c4ad2297c

2. How the Video Game Industry Is Changing
Andrew Beattie, October 31, 2021 [електронний ресурс]
https://www.investopedia.com/articles/investing/053115/how-video-game-industry-changing.asp

3. What Unreal Engine 5 Means for the Games Industry
RONNY BARRIER, MAY 17, 2022 [електронний ресурс]
https://www.ign.com/articles/what-unreal-engine-5-means-for-the-games-industry

4. Kitbash [електронний ресурс]
https://kitbash3d.com/a/blog/why-you-should-learn-unreal-engine-5-in-2023

5. How video games are made: the game development process
Nadia Stefyn, 05/09/2022 [електронний ресурс]
https://www.cgspectrum.com/blog/game-development-process

6. Nuclino [електронний ресурс]
https://www.nuclino.com/articles/video-game-development-process

7. The Complete Guide to Blender Graphics: Computer Modeling & Animation,
John M. Blain., [Навчальний посібник] April 16, 2012

8. Офіційний сайт компанії Maxon. [електронний ресурс]

https://www.maxon.net/en/about-maxon/history

9. Офіційний сайт Adobe. [електронний ресурс]

https://www.adobe.com/ua/products/substance3d-painter.html

10. THE SEVEN STAGES OF GAME DEVELOPMENT
Ross Bramble, GameMaker 10 May 2023

11. Blender документація [електронний ресурс]
https://docs.blender.org/manual/en/latest/

12. Exploring alternatives with Unreal Engine’s Blueprints Visual Scripting System
Eric Chu, Loutfouz Zaman, Volume 36, January 2021, 100388