КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»

(шифр і назва спеціальності)

«Прикладне програмування»

(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Комп'ютерна гра засобами Unreal Engine»

Виконав _____
(Підпис)
Коломоєць Андрій Євгенович
(прізвище, ім'я, по батькові)

Керівник _Сайко Володимир Григорович
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")
Завідувач кафедри _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2021

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

Назва теми: «Комп’ютерна гра засобами Unreal Engine»

Освітня програма: Прикладне програмування
Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Коломоєць Андрій Євгенович	

Назва роботи українською та англійською мовами

Комп’ютерна гра засобами Unreal Engine

Creating computer game using Unreal Engine

Мета бакалаврської кваліфікаційної роботи, завдання

Мета роботи: Створення комп’ютерної гри використовуючи ігровий рушій Unreal Engine.

План роботи:

1. Аналіз сучасних ігрових рушіїв
2. Аналіз процесу створення відеоігор
3. Розробка ключових компонентів комп’ютерної гри (Blueprints Visual Scripting)

ПІБ, ступінь, звання наукового керівника роботи: _____

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Номер	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	26.10.2020	
2.	Видача завдання кваліфікаційної роботи бакалавра	23.11.2020	заява
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	01.12.2020	
4.	Затвердження плану кваліфікаційної роботи бакалавра	18.02.2021	
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	25.02.2021	
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	05.03.2021	
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	09.04.2021	
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	07.05.2021	
9.	Подання роботи у першому варіанті	11.05.2021	
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	12.05.2021	
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	24.05.2021	
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	28.05.2021	
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	11.06.2021	
14.	Захист кваліфікаційної роботи бакалавра	22.06.2021	

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1 ст.
Завдання до дипломної роботи (календарний план проекту)	1 ст.
Відомість дипломної роботи	1 ст.
Анотація	1 ст.
Анотація (іноземною мовою-англійською)	1 ст.
Зміст	2 ст.
Словник термінів	1 ст.
Вступ	2 ст.
1. ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ КОМП'ЮТЕРНИХ ІГОР	12 ст.
2. РОЗРОБКА ГРИ У ЖАНРІ RPG	23 ст.
3. Аналіз створеної гри	2 ст.
Висновки	1 ст.
Перелік посилань	2 ст.
Додатки	6 ст.

				ДП XXXX 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.				Відомість дипломної роботи	Лист	Листів
Керівн.	Сайко В.Г.					
Н/контр.	Макаренко С.А.					
Зав.каф.	Плескач В.Л.					

Анотація (реферат)

Дипломна робота Коломойця Андрія Євгеновича, з теми «Комп'ютерна гра засобами Unreal Engine», спеціальності комп’ютерні науки 122, освітня програма - прикладне програмування, містить: 57 сторінок, 9 рисунків, 1 таблицю, 23 літературних джерел. Метою роботи є дослідження і розробка гри з використання ігрового рушія Unreal Engine. Об’єкт дослідження – комп’ютерні ігри. У результаті дослідження було проаналізовано нинішній стан індустрії відеоігор, технічні та програмні засоби цієї сфери та більш детально розглянуто засоби розробки в ігровому рушії Unreal Engine. Створено прототип комп’ютерної гри на основі проведених досліджень. Розробка велась за допомогою засобів двигуна Unreal Engine для платформи Windows

Ключові слова: комп’ютерна гра, Unreal Engine, розробка відеоігор.

.

Annotation (abstract)

Thesis of Kolomoiets Andrii Yevhenovych, on the topic «Creating computer game using Unreal Engine», specialty of computer sciences 122, educational program - applied programming, contains: 57 pages, 9 drawings, 1 table, 23 literary sources. The aim of the work is to research and develop a game by means of Unreal Engine game engine. The object of research is computer games. The study analyzed the current state of the video game industry, hardware and software in this area and researched more about development tools in the Unreal Engine. A prototype computer game based on research has been created. The development was carried out using the Unreal Engine for Windows platforms

Keywords: computer game, Unreal Engine, video game development.

ЗМІСТ

Словник термінів

Нижче наведено словник специфічних слів і термінів (професійні жаргонізми, абревіатури, скорочення) що можуть використовуватись у контексті розробки відеоігор.

FPS (First Person Shooter), шутер — жанр відеоігор, основу ігрового процесу якого є стрільба зі зброї по цілям.

RTS (Real Time Strategy) — піджанр стратегії, основу ігрового процесу якого є одночасне керування групами ігрових персонажів в режимі реального часу.

MMO (Massive Multiplayer Online) — жанр відеоігор, що розрахована на велику кількість одночасних гравців в онлайн режимі.

PRG (Role-Playing Game) — жанр відеоігор, в якому гравець повністю керує діями персонажа у чітко визначеному ігровому всесвіті

MMORPG (Massively Multiplayer Online Role-Playing Game) — жанр відеоігор, що об’єднує в собі аспекти MMO та RPG одночасно.

Спавн — процес створення ігрового об’єкту в ігровому просторі.

Рендеринг — процес створення графічного зображення певної моделі використовуючи комп’ютерні програми.

ВСТУП

Найперша відеогра в історії людства була створена в 1958 році. Це була проста гра у вигляді тенісу на двох, зіграти в неї можна було на аналоговому комп'ютері, який використовували для вираховування траєкторії часток із урахування опору вітру. Починаючи із 1970 року настає епоха спеціальних аркадних ігрових автоматів, а індустрія відеоігор починає неперервно розвиватися. В наш час індустрія відеоігор заробляє більше ніж 180 мільярдів доларів за рік. Потенційно прибуток ринків відеоігор може зрівнятися із прибутками спортивної індустрії.

Актуальність дослідження. На сьогоднішній день понад мільярд людей регулярно грає у відеоігри. Щорічно виходять сотні, а то й тисячі найрізноманітніших відеоігор. Масштаб ігор різниться від найпростішого тетрісу в командному вікні до цілих армій в гігантських штучних галактиках, жанр - від головоломок до рольових ігор, а бюджет розробки - від нуля до десятків мільйонів доларів.

Все це говорить про те, що є необхідність в розробці нових ігор, а ігрова індустрія активно продовжує розвиватися. У 2011 році в США відеоігри офіційно визнали видом мистецтва. Так, практично будь-яка гра містить в собі безліч аспектів. Двома фундаментальними з них є програмний код і ігровий контент. Крім двох названих аспектів можна виділити і інші - наприклад, дизайн гри, ігровий баланс, маркетинг та інші. Візуальний зворотний зв'язок є підмножиною контенту. У широкому сенсі ігровий контент — це все, що міститься в рамках гри, за винятком програмної логіки. Сюди входить вся графіка (від моделей персонажів і текстур неба до кнопок і логотипу в головному меню гри), звук, музика, сценарій, сюжет, персонажі, ігрові рівні та інше. Практично будь-яка нетривіальна відеогра містить більшість з перерахованих видів контенту.

Дана робота допомагає дослідити та зробити аналіз питань, що стосуються розробки комп'ютерних ігор використовуючи сучасні засоби розробки.

Практичну частину дипломної роботи можна охарактеризувати створенням комп’ютерної гри, що міститиме в собі всі базові компоненти сучасних ігор, використовуючи ігровий движок Unreal Engine.

Метою дипломної роботи є створення комп’ютерної гри використовуючи Unreal Engine. Для цього буде поступово розглянуто процес створення всіх компонентів відеогри.

Завдання дослідження:

- аналіз інструментів створення комп’ютерної гри;
- ознайомлення із головними принципами створення ігор;
- ознайомитися з процесом створення комп’ютерної гри від створення прототипу до проведення тестування;

Об’єктом дослідження у цій роботі є комп’ютерні ігри, а ***предметом дослідження*** – створення комп’ютерної гри засобами Unreal Engine.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ КОМП'ЮТЕРНИХ ІГОР

У відеоігри грають на аркадних ігрових автоматах, вдома на телевізорі чи персональному комп'ютері. Вони можуть бути упаковані у великі консолі, ігрові пакети, які можна відтворювати лише на обладнанні одного виробника (наприклад, Nintendo, Sega Genesis та Sony Playstation), на компакт-дисках, їх можуть продавати спеціалізовані інтернет магазини. Складені з програми, яка вказує комп'ютеру відображати певні візуальні та звукові ефекти, відеоігри використовують найсучасніші технології, щоб надати кінцевому користувачу максимум задоволення від процесу гри.

1.1 Основні жанри комп'ютерних ігор

Дизайн є ключовим аспектом створення всіх відеоігор. Зазвичай це робить команда досвідчених програмістів, сценаристів, художників та інших дизайнерів ігор. На цьому етапі розробки вони генерують технічні характеристики гри, які включають тип гри, ціль та графіку.

Хоча створення відеоігри рідко буває поетапним процесом, існує безліч завдань, які необхідно виконати на етапі розробки. На початку визначається тип і мета гри. Загалом, ігри належать до шести категорій або жанрів, включаючи боротьбу, стрільбу, стратегію, симуляції, пригоди, біг, платформери. Бойові ігри вимагають від гравців битви між собою або комп'ютером. На сьогодні вони найпопулярніші і охоплюють такі заголовки, як Mortal Kombat та Street Fighter. Стрілялки включають битви, в яких гравець намагається знищити ворожі танки, кораблі або літаки. Стратегічні ігри включають такі класичні ігри, як шахи, бридж або шашки. Симулятори — це ігри, які відтворюють реальні життєві ситуації, такі як політ чи водіння. Пригодницькі ігри — це комп'ютеризовані версії рольових фентезійних ігор. Платформери — це такі, як ігри Super Mario, в

яких персонаж намагається досягти мети, долаючи різні перешкоди на своєму шляху.

Серед найпопулярніших жанрів відеоігор можна виділити наступні:

- FPS (First Person Shooter) — шутер від першої особи. На сьогодні екшн-ігри є найпопулярнішим жанром ігор, а ігри FPS утворюють найпопулярніший піджанр в категорії екшн-ігор. Ігри FPS мають тривимірне середовище і зосереджені навколо бою на основі зброї в перспективі від першої особи, тобто так, щоб гравець бачив оточення таким, яким бачив би його персонаж. TPS (Third Person Shooter) або шутери від третьої особи схожі на ігри FPS, за винятком того, що камера гравця знаходиться позаду персонажа, яким вони керують.
- RTS (Real Time Strategy) — стратегія в режимі реального часу. У іграх RTS гравці керують підрозділами з ігрових персонажів під своїм контролем, щоб перемогти суперників і встановити контроль над ключовими зонами на карті. У більшості середовищ можна створити більше одиниць населення та побудувати цивільні та військові структури в рамках гри. Збір ресурсу, як правило, є головним ключем для досягнення цих цілей. Наприклад, контроль золотої шахти в грі дозволяє гравцеві фінансувати будівництво будівель та створення нових одиниць населення, що, в свою чергу, є корисним у майбутніх місіях. Завдання, які повинен виконувати гравець, щоб досягти успіху в грі RTS, як правило, ускладнюються в міру просування рівнів.
- MOBA (Multiplayer Online Battle Arena) — багатокористувацька онлайн-бойова арена. У іграх MOBA гравець керує одним персонажем у команді, яка змагається з іншими командами в оточенні. Завдання, як правило, полягає в знищенні комп'ютерних об'єктів та перемозі інших команд у навколишньому середовищі. У більшості ігор MOBA кожен гравець бере на себе певну роль у команді. У цих середовищах існують різні ролі. Наприклад, гравці ролі підтримки надають допомогу команді та її союзникам, наприклад, відволікаючи чи злегка завдаючи шкоди ворогам,

тим самим створюючи простір для дій союзників. Ігри MOBA — це поєднання екшн-ігор, рольових ігор та стратегічних ігор у реальному часі. MOBA часто називають A-RTS або екшн-стратегією у режимі реального часу, хоча слово стратегія в даному випадку стосується того, як команда співпрацює та розігрує карту, щоб досягти успіху.

- RPG (Role Playing Game) — рольова гра. У RPG гравець керує діями персонажа у чітко визначеному фантазійному всесвіті чи у всесвіті наукової фантастики. Гравці часто можуть робити те, що неможливо в реальному житті. Зазвичай мета полягає у тому, щоб гравець виконав серію завдань, щоб дійти до висновку центральної сюжетної лінії. У цих середовищах розвиток персонажа відбувається за допомогою елементів сюжету. Більшість RPG включають комплексну 3D графіку .
- MMO (Massively Multiplayer Online) — масова багатокористувацька мережа. MMO-ігри, як правило, мають величезний постійний відкритий світ, і тисячі користувачів, що грають на одному сервері. Для цих ігор потрібні мережеві платформи, такі як смартфони, комп’ютери або підключені до інтернету ігрові приставки. У MMO гравці можуть широко співпрацювати та взаємодіяти один з одним. Існує MMO для різних типів і жанрів ігрового процесу. MMO можуть бути FPS, RPG або RTS в різних галузях, таких як бойові, спортивні, гоночні, соціальні та інші.
- MMORPG (Massively Multiplayer Online Role Playing Games) масові багатокористувацькі онлайн-рольові ігри. MMORPG - це комбінація MMO та RPG. Вони мають дуже велику кількість гравців, які взаємодіють один з одним у постійно відкритому віртуальному світі. Гравець повністю бере на себе роль свого персонажа і може співпрацювати чи конкурувати з іншими гравцями у постійному світі гри. Світ продовжує існувати та розвиватися, поки гравець перебуває поза мережею або поза грою. Популярні MMORPG засновані на фантастичних темах, що включають елементи злочину, чаклунства та наукової фантастики. Деякі спільноти MMORPG розробили

власні субкультури, які мають свій власний жаргон, видатні особистості та соціальні правила.

Оскільки індустрія відеоігор продовжує швидко розвиватися і в усьому світі, ці ігрові жанри будуть продовжувати розвиватися. У міру зростання обчислювальної потужності та по мірі того, як технологія віртуальної реальності та доповненої реальності стає загальнодоступною, стануть можливими більш складні ігрові середовища.

В рамках виконання практичної частини кваліфікаційної роботи бакалавра, мною буде розроблена гра у жанрі RPG. Концепція рольових ігор класична, вона існувала ще до появи комп'ютерів. В наш час багато гравців по всьому світу вважають саме цей жанр комп'ютерних ігор своїм улюбленим. За останні роки найбільші AAA-релізи відносяться саме до цього жанру. Серед найяскравіших представників цього жанру можна виділити наступні ігри: Cyberpunk 2077, The Witcher, The Elder Scrolls, Dark Souls.

1.2 Ключові елементи при створенні відеогри

Після визначення типу гри та історії можна визначити формат гри. Формат стосується того, що гравець бачить під час гри. Існує безліч форматів, включаючи платформер, зверху-вниз (top-down), прокрутку (sidescroller), ізометричну, тривимірну (3D), та чисто текстовий формат. Ігри платформери — це ті, що мають вигляд збоку на персонажа гравця. Ігри зверху-вниз дають змогу спостерігати за персонажем гравця нібито з висоти пташиного польоту. Такий формат часто використовують для ігор з військовою тематикою. Ізометричний формат — це гра зверху-вниз, яка використовує фокуси перспективи, щоб створити ілюзію 3D. Формат текстових ігор має обмежену графіку і використовується лише для інтерактивної художньої літератури. Завдяки розвитку технологій 3D формат є найпопулярнішим у наш час. Його широко використовую при розробці комп'ютерних та консольних ігор, часто

використовую для мобільних ігор. Крім того цей формат є невід’ємною частиною для ігор з використанням технологій віртуальної чи доповненої реальності. Загалом, одна гра може містити в собі декілька форматів та ігрових режимів. Такий підхід дозволяє урізноманітнити ігровий процес, що в кінцевому результаті призводить до більшого залучення кінцевого користувача до ігрового світу, та, як наслідок, приносить більше задоволення під час гри.

Коли всі попередні елементи проектування гри визначені, можна розпочинати її програмування. Першим кроком у цьому процесі є складання блок-схеми, яка показує логічний прогрес комп'ютерної програми. Код, як правило, виробляється командою програмістів, кожна з яких працює на різній фазі гри, і на її виготовлення може піти до декількох років. Щоб пришвидшити процес кодування, раніше розроблені алгоритми часто модифікуються та адаптуються до нової гри. Такий підхід є більш ефективним, оскільки усуває необхідність постійно переписувати подібні програми та зменшує ймовірність серйозних помилок. Кожна дія може вимагати багатьох індивідуальних інструкцій, написаних програмістом, і приблизно 250 000 окремих команд написані для створення програми для відеоігор. Звук і графіку також потрібно програмувати окремо.

Візуальна складова гри також є дуже вагомою її складовою. Неважливо наскільки продуманими можуть бути ігрові механіки, якщо графіка у грі буде зроблена погано. Люди сприймають більшість інформації саме через зір, а отже графіці у іграх повинна бути відведена особлива увага. Над створенням гри зазвичай працюють групи художників, що створюють моделі персонажів, предметів та оточення загалом в одному визначеному стилі. Над створенням рівнів та загальної картини світу у грі працюють спеціальні дизайнері рівнів. Їх робота полягаю в тому, щоб зробити ігровий простір єдиним цілим. Порушення стилістики рівнів чи відхилення від сюжету може призвести до порушення враження від ігрового процесу. Добре скомпонований ігровий світ завжди залишає лише приємні враження від гри.

Ще одною складовою повноцінної відеогри є саунд-дизайн. Звуковий дизайн відеоігор — це мистецтво створення та додавання звукових елементів у відеогру. Це передбачає створення цілих бібліотек власних звукових ефектів, щоб надати грі відчуття реалізму та унікальності. Звукові ефекти повинні бути належним чином застосовані до зображень, які буде бачити гравець. Якщо ж звук гри не працює належним чином, наприклад, тихий вибух чи діалог, то враження від гри може зіпсуватись. Навіть найперші відеоігри, що використовували лише ті технічні ресурси, які мали, додавали звуки до гри, що робило її більш привабливою. Наприклад, у Понга звучали прості звукові сигнали, коли м'яч відскакував від платформ. Звукорежисер — це людина, яка генерує та обробляє звукові елементи для гри. Роль звукорежисера в ігровій індустрії багато в чому подібна до ролі телевізійного виробництва, театру та кіно. Без когось, хто насправді створював би музику та звукові ефекти для гри, кожному довелося б використовувати однакові звуки. Метою звукорежисера є співпраця з командою дизайнерів та аніматорів для створення насиченого звуку, який відповідає віртуальному досвіду, що розробляється. Звукорежисери також займаються тестуванням гри на пізніх стадіях розробки, щоб виявити будь-які звукові відхилення або помилки.

Етап тестування розробки ігор допомагає виявити основні проблеми проектування та програмування. Тестування можна виконувати різними способами. Програмісти можуть самі запустити гру і спробувати виявити грубі проблеми. Додатково використовуються професійні ігрові майстри. Це люди, які спеціально навчені грати в ігри та шукати тонкі помилки. Вони, як правило, самі працюють дизайнерами ігор і мають досвід роботи з багатьма типами ігор. Окрім пошуку помилок, тестувальники також дають критику та пропозиції щодо покращення гри. У деяких випадках розробники комп'ютерних ігор використовують для тестування ігор людей із загальної аудиторії. Це дає їм інформацію про прийняття їх продукту споживачами. Інформація, отримана на етапі тестування, переглядається, потім проводиться перепрограмування, доки гра не буде належним чином налаштована.

1.3 Сучасні ігрові рушії

Кожна створена гра працює на певному ігровому рушії. Ігровий рушій (або ігровий движок) — це набір програмних інструментів або API, побудованих для оптимізації розвитку відеоігор. Зазвичай ігрові рушії включають в себе ігровий цикл та механізми рендеринга. На відміну від простого використання мов програмування із графічними бібліотеками та спеціальним графічним програмним забезпеченням, ігрові рушії мають вбудовану систему оптимізації ігрового циклу, що також включає в себе оптимізацію рендерингу. Ігровий движок повинен мати можливість спростити такі важливі завдання, як:

- Фізика — ігровий досвід та фізика повинні мати ідеальний баланс між якістю моделювання та обмеженнями обчислювальної потужності для кінцевого користувача.
- Вхідні дані - це надзвичайно поширена проблема у крос-платформній розробці. Ігровий движок повинен надавати інструментарій для підтримки операцій вводу на будь-яких підтримуваних платформах.
- Обробка візуальних активів - освітлення, затінення, відображення текстур та глибина різкості вимагають менших зусиль програмування під час використання ігрових движків.

Ігровий движок повинен надавати можливість виконувати вищезазначені завдання із зменшеними зусиллями кодування. Використання ігрового рушія допомагає значно скоротити час розробки та дозволяє командам зосередитись на розробці своїх ігор, щоб забезпечити унікальний та особливий досвід користувачів. Незважаючи на це, в світі все ще існує велика кількість компаній і навіть незалежних команд, які створюють свої ігрові движки. Станом на 2021 рік найкращими ігровими рушіями вважаються наступні: Unreal Engine, Amazon Lumberyard, CryEngine, Unity, GameMaker: Studio, Godot.

Одним з найпопулярніших і широко використовуваних ігрових двигунів є Unreal Engine, який належить Epic Games. По суті, це багатоплатформенний

механізм розробки ігор, розроблений для підприємств будь-якого розміру, який допомагає використовувати технологію реального часу для перетворення ідей у залучення візуального вмісту. В 2019 році Epic Games придбала Quixel, який володіє величезною бібліотекою активів "фотограмметрія" реальних зображень, які можна використовувати для створення анімації та відеоігор. Користувачі Unreal Engine зможуть безкоштовно використовувати вбудовані інструменти Quixel (Bridge, Mixer) та всі ресурси бібліотеки Quixel's Megascans безкоштовно.

Як видно з назви продукту, Lumberyard — це комплексна пропозиція від Amazon. Це виключно тривимірний ігровий движок, призначений для створення ігор. Він пропонує режим попереднього перегляду VR, засоби візуального сценарію, а також інтеграцію із платформою Twitch. Завдяки веб-службам Amazon, безпечній хмарній платформі, розробка онлайн ігор на Lumberyard стає значно простішою. Amazon в цілому концентрується саме на онлайн сервісах. Lumberyard також підтримує Autodesk Maya та Adobe Photoshop.

CryEngine є середовищем для розробки комп'ютерних ігор, створений CryTek. Однією з його важливих особливостей є потужний механізм візуалізації, який характеризується своєю візуальною якістю. Продукт включає багато інструментів, важливих для сучасних жанрів розробки ігор, таких як фізичний двигун та інструменти штучного інтелекту. Через складний характер програмного забезпечення воно орієнтоване на команди певного розміру та користувачів з великими очікуваннями.

Unity — це мультиплатформенний ігровий движок. Цей ігровий рушій сьогодні є вибором багатьох великих організацій завдяки його чудовій функціональності, високоякісному контенту та можливості використовувати його для будь-якого типу гри. Він підтримує як 2D, так і 3D вміст. Завдяки універсальному редактору Unity сумісний з платформами Windows, Mac, Linux, IOS, Android, Switch, Xbox, PS4 та іншими. Зручний інтерфейс полегшує розробку та зменшує потребу в навчанні. Магазин активів Unity зберігає величезну колекцію інструментів та вмісту, що створюється щодня.

GameMaker: Studio став широко використовуватися, оскільки для його використання не потрібно вміти програмувати. Програмування ігор відбувається завдяки візуальним елементам, які можуть слугувати повною заміною мовам програмування. Такий підхід є актуальним для невеликих студій чи новачків. Серед найкращих ігор, зроблених за допомогою GameMaker є Spelunky, Hotline Miami та Super Crate Box. Основним недоліком GameMaker є його обмеженість у порівнянні із іншими ігровими рушіями.

Godot - це багатофункціональний ігровий рушій для створення 2D та 3D ігор з уніфікованим інтерфейсом. Ігри, створені на Godot, можна експортувати в один клік на ряд платформ, включаючи основні платформи для настільних ПК (Linux, macOS, Windows), а також мобільні (Android, iOS) та веб-платформи (HTML5). Godot є абсолютно безкоштовним та має відкритий код за ліцензією MIT. Ігри користувачів є повністю їхніми, навідміну від інших движків Godot нічого з них не отримує. Розвиток Godot повністю незалежний та керується громадою, надаючи можливість користувачам допомогти сформувати свій двигун відповідно до їхніх очікувань.

1.4 Базові Поняття в Unreal Engine

Unreal Engine — професійний ігровий движок. Він був розроблений компанією Epic Games у 1988 році. Спочатку він був розроблений як шутер від першої особи. В даний час він використовується для створення бойових ігор, RPG, RTS та інших MMORPG. В процесі програмування часто використовують мову C++. Цей движок користується популярністю серед розробників AAA ігор і використовується ними як інструмент розробки ігор. Дуже часто його порівнюють із Unity, і більшість новачків віддають перевагу саме Unity, але Unreal Engine також пропонує основні техніки розробки, які дуже допомагають новачкам.

Це високоякісний ігровий движок, який допомагає користувачеві створювати ігри. Він містить різні механізми, які забезпечують платформу для користувачів та допомагає їм запускати гру. Він має величезний систематичний набір інструментів та редакторів, які допомагають користувачам керувати своїми властивостями та модифікувати їх для створення ілюстрацій для своїх ігор. Механізм складається з частин, які включають графічний движок, онлайн-модуль, фізичний движок, звуковий механізм, вхідні дані та фреймворк ігрового процесу. Розробники обирають саме цей движок, тому що він має:

- Фото реалістичний рендеринг в режимі реального часу;
- Надійна багатокористувацька структура;
- VFX і моделювання частинок;
- Ефекти пост-обробки зображення якості на рівні фільмових;
- Розширені можливості для створення штучного інтелекту;
- Unreal звуковий движок;
- Безмежна розширюваність.

У ньому є кілька редакторів, які допомагають у розробці гри. При запуску він має Unreal Editor за замовчуванням. Цей редактор є головним редактором, який допомагає користувачеві переглядати та працювати з іншими підплатформами та редакторами.

Серед основних переваг Unreal Engine можна виділити:

- Використання передових технологій для роботи з графікою
- Концентрація багатьох елементів для розробки гри в одному місці
- Інтерфейс користувача Unreal Engine постійно оновлюється найновішими інструментами та опціями.
- Він має вбудовану технологію візуального програмування і використовує вузли, що називаються Blueprints. Ці вузли допомагають користувачам створювати ігри високого класу без написання сценаріїв та коду.

- Тут використовується мова програмування C ++, що саме робить програму програмою першого вибору розробника, оскільки C++ вважається найкращим вибором для програм, які потребують оптимізації.

Він посідає значне місце в ігровій індустрії, оскільки може виконувати багато видів діяльності, має безліч функцій та переваг. Він також постійно вдосконалює інтерфейс найновішими інструментами. Постійні вдосконалення дозволяють постійно завойовувати великий інтерес та попит.

1.4.1 Unreal Engine Marketplace

Unreal Engine Marketplace — це платформа електронної комерції, за допомогою якої творці контенту, що використовують UE4, спілкуються з розробниками, надаючи безліч готового до гри вмісту та коду. Люди, що продають контент таким чином, отримують 88% базової ціни за кожен продаж опублікованих продуктів

Окрім того, що продукти Marketplace можуть використовуватися для навчання, експериментів чи створення прототипів, можна також використовувати продукти Marketplace у своїх власних відвантажених продуктах. Проте не можна продавати або ліцензувати вміст Marketplace іншим розробникам для використання в їх продуктах, наприклад через веб-сайт або в механізмах електронної комерції, вбудованих в інструменти розробки 3D.

Unreal Engine Marketplace також містить розділ із безкоштовним контентом, проте такі продукти дозволено використовувати лише при роботі з Unreal Engine. До таких товарів відноситься набір сіток, анімацій та матеріалів Infinity Blade — Action-RPG 2010 року від студії ChAIR Entertainment. Крім того, кожного місяця компанія Epic Games пропонує до 5 товарів безкоштовно. Ці товари назавжди залишаються на аккаунті, та можуть бути використані при розробці власних продуктів. Натомість компанія Epic Games виплачує розробникам таких товарів повну суму за кожну покупку.

1.4.2 Blueprints Visual Scripting

Система Blueprints Visual Scripting— це система створення сценаріїв ігрових процесів, заснована на концепції використання інтерфейсу на основі вузла для створення елементів ігрового процесу з Unreal Editor. Як і у багатьох поширених мовах сценаріїв, він використовується для визначення об'єктно-орієнтованих класів або об'єктів у механізмі. Ця система надзвичайно гнучка і потужна, оскільки надає можливість дизайнерам використовувати практично весь спектр концепцій та інструментів, загалом доступних лише програмістам. Крім того, спеціальна розмітка Blueprint, доступна в реалізації C ++ від Unreal Engine, дозволяє програмістам створювати базові системи, які можуть бути розширені дизайнерами.

Висновки

Виходячи із проведених досліджень, Unreal Engine підходить для розробки будь-яких ігор, має багату бібліотеку активів, а отже ідеально підходить для створення гри в рамках виконання кваліфікаційної роботи бакалавра. Серед існуючих жанрів ігор було обрано жанр RPG, оскільки він є найбільш класичним та загальноприйнятим. Для виконання практичної частини роботи будуть використані активи Infinity Blade із бібліотеки активів Unreal Engine. За угодою, ці активи безкоштовні при використанні їх в іграх зроблених на Unreal Engine.

РОЗДІЛ 2. РОЗРОБКА ГРИ У ЖАНРІ RPG

В даному розділі проводиться опис роботи по розробці комп'ютерної гри засобами Unreal Engine. Для реалізації гри в жанрі RPG будуть розроблені наступі компоненти:

- Головний герой, механізм керування героєм;
- Розробка інтерфейсу користувача;
- Створення допоміжних реквізитів;
- Створення ворогів та їх штучного інтелекту;
- Створення рівню для гри;
- Тестування гри.

2.1 Головний герой

Головний герой — центральний персонаж, яким керує гравець під час проходження гри. Головним героєм можуть бути сам герой, оповідач, найкращий друг головного героя - поки вони є головними героями, які беруть участь в історії, взаємодіють із другорядними або допоміжними персонажами і на них особисто впливає головний конфлікт сюжету.

Головним героєм в створюваній грі є лицар, ціль якого боротися зі злом у своєму світі. Такий сюжет є класичним для багатьох ігор. Герой повинен вміти пересуватись. Також він повинен володіти мечем, для того щоб відбиватися від сил зла. Як і у класичних іграх в жанрі RPG, герой матиме певну кількість здоров'я та енергії. Енергія буде витрачатися під час використання ключових здібностей героя. Залежно від кількості здоров'я буде визначатися стан героя. Так, якщо кількість здоров'я впаде до нуля, то гра буде завершена.

В процесі гри герой зможе мати витратні ресурси, які можна буде використовувати під час гри. За класикою жанру, до таких ресурсів будуть відноситися зілля відновлення здоров'я та енергії.

Для боротьби герой використовуватиме зброю, яку можна буде змінити під час гри. Тип зброї визначатиме кількість очок пошкоджень, які герой зможе наносити під час атаки. Крім звичайних атак герой володітиме спеціальними вміннями, які допоможуть опанувати ситуацію в бою. Щоб використати вміння герой витрачатиме частину своєї енергії, після чого вміння стане недоступним для повторного використання на певний проміжок часу. Такі вміння будуть поступово відкриватися із підвищенням рівня героя.

2.1.1 Опис класу головного героя

Для того, щоб головним героєм можна було керувати, він повинен наслідувати базовий клас Character, що в свою чергу наслідує клас Actor. Подібно до фільмів, такі сукупність акторів утворюють єдину картину на рівнях гри. Актори можуть бути будь-чим: від статичних нерухомих об’єктів до повноцінних персонажів. Клас Actor підтримує тривимірні перетворення (переміщення, поворот та зміна розміру). Кожен Actor підтримує ієрархію компонентів (клас UComponent) та його перетворення визначається перетворенням основного компонента — Root. Actor також має доступ до функцій Tick(), яка оновлює його статус і може використовуватися для відтворення коду в кожному кадрі. Процес створення Actor називається спавн (Spawn).

Ієрархія компонентів головного героя виглядає наступним чином:

- Компонент капсульної колізії — компонент, що відповідає за виконання базових операцій з колізіями. Прикладом подібної операції є отримання пошкоджень: коли компонент, що відповідає за колізії, у ворожого об’єкта перетинається з цим компонентом, утворюється On Component Being Overlap() — подія, в якій можна запрограмувати логіку отримання пошкоджень.
- Компонент стрілки — вказує поточне направлення актору. Є невід’ємною частиною акторів, які розраховані на пересування у просторі, оскільки

вказує на так званий forward vector (або направляючий вектор). Цей вектор використовується для встановлення позиції актору в світових координатах.

- Скелетна сітка — необхідна частина для акторів, які повинні рухатися. На відміну від статичної сітки, скелетна сітка має скелет, який дозволяє розробляти різні анімації для сітки посилаючись на елементи скелету. Для визначення текстури сітки, їй потрібно мати посилання на використовуваний матеріал; інакше сітка буде визначати лише форму актора
- Камера — відповідає за те, що бачить кінцевий користувач. Камера буде слідкувати за об'єктом до якого вона прив'язана, копіюючи його перетворення.
- Компонент кронштейну — використовується разом із камерою для більш коректного її позиціонування. Кронштейн дозволяє запобігати випадків, коли камера губиться серед інших об'єктів виштовхуючи її з них.
- Компонент переміщення — також є ключовим компонентом для акторів, які мають переміщуватися у світовому просторі. Цей компонент дозволяє відслідковувати такі параметри як швидкість та прискорення, при чому параметр прискорення буде оновлюватися в кожному кадрі, опираючись на направляючий вектор та вхідний вектор, що задається командою пересування.

Всі ці чисто технічні складові дозволяють створити персонажа, за яким гравець зможе спостерігати, та який отримуватиме команди по переміщенню в світовому просторі із симуляцією фізики. Крім технічних компонентів для реалізації головного персонажу потрібні ще й його логічні компоненти, на яких будуватиметься загальний геймплей за цього персонажа. Для реалізації механізмів здоров'я, енергії та інших ключових механізмів, описаних в попередньому підрозділі, до класу головного героя були додані наступні поля: BaseTurnRate, BaseLookUpRate, MaxHealth, Health, MaxMana, Mana, Level, CurrentEXP, NextLevelEXP, DamageMod, HealthRegenRate, ManaRegenRate.

Поля BaseTurnRate та BaseLookUpRate — це суто технічні поля, вони відповідають за швидкість повороту персонажу та швидкість зміни кута погляду камери.

Поля MaxHealth та Health відповідають за максимальну та поточну кількість очок здоров'я відповідно. Значення Health не може перевищувати значення MaxHealth. Якщо значення Health стане нижче або дорівнювати нулю, то в рамках гри персонаж помре, а гравцю потрібно буде починати гру заново.

Аналогічно до полів здоров'я, поля Mana та MaxMana відповідають за енергію. Енергія необхідна для використовування спеціальних здібностей. Падіння показника поточного запасу енергії значитиме, що перед використанням здібностей необхідно відновити її запас.

Поля HealthRegenRate та ManaRegenRate відповідають за кількість очок відповідного ресурсу, що відновлюються пасивно з плином часу. Так, якщо швидкість регенерації дорівнюватиме 0.1 очок в секунду, а герой матиме 150 очок здоров'я, то повністю пасивно відновити свій запас здоров'я він зможе за 1500 секунд.

Поля Level, CurrentEXP, NextLevelExp відповідають за рівень персонажа. Якщо показник CurrentEXP стане більше або дорівнювати показнику NextLevelEXP, то показник Level збільшиться на одиницю, показник CurrentEXP залишить залишок або обнулиться, а всі інші показники персонажу перерахуються, включаючи NextLevelEXP. При переході на новий рівень у персонажа збільшуються максимальні показники здоров'я та енергії, після чого повністю відновлюються. Персонаж починає гру із першим рівнем, максимальний доступний рівень — п'ятий. На кожному рівні персонаж розблоковує одну із своїх здібностей.

Поле DamageMod впливає на кінцеву кількість очок пошкоджень, яку може нанести персонаж своїм ворогам. Базове значення 1, із рівнем збільшується на 0,1.

2.1.2 Загальний вигляд та анімація головного героя

Для створення головного персонажу була використана спеціальна скелетна сітка лицаря (рисунок 1.1). У порівнянні із звичайною скелетною сіткою, яка надається разом із самим Unreal Engine, ця сітка має більш складну систему ієрархії кісток, що дозволяє робити більш складні анімації. Матеріал, розроблений для цієї сітки дає чітко зрозуміти, що це саме лицар.

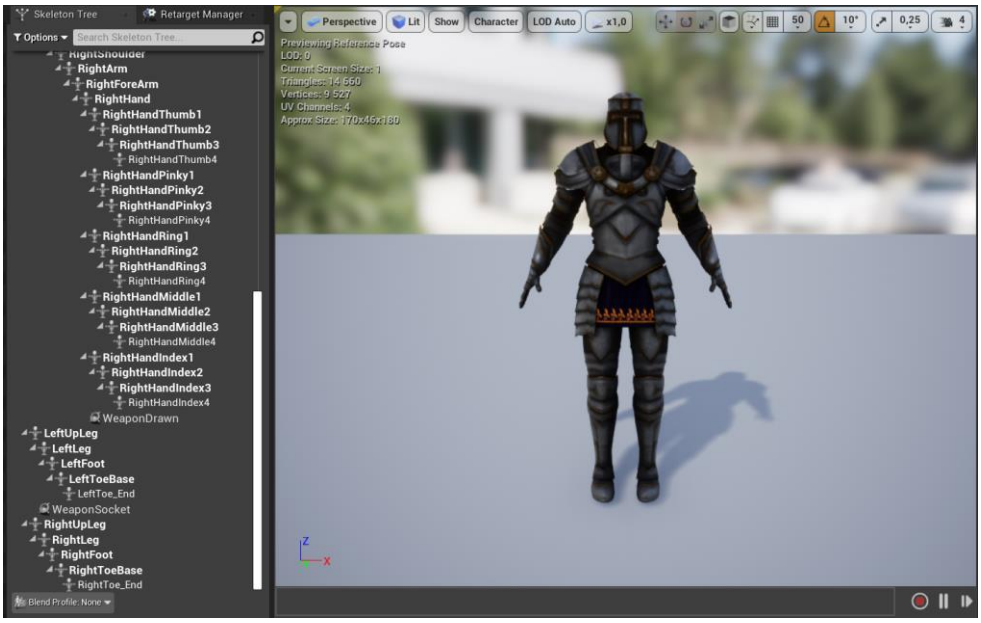


Рисунок 2.1 — Скелетна сітка лицаря

Ця скелетна сітка має набір анімацій, пов’язаних із різними діями, які потенційно може виконувати персонаж. Для того щоб знати в який час яку анімацію необхідно відтворити, в Unreal Engine використовується клас Animation Blueprint. Цей клас містить всю інформацію про анімації та скелетну сітку, в ньому можна задавати послідовність програвання анімацій. Проте якщо просто програвати анімації, то всі дії персонажу будуть схожі на слайд шоу. Щоб вони програвались плавно, із переходами та потенційним змішуванням декількох анімацій одночасно необхідно запрограмувати логіку переходу та змішування анімацій.

Для визначення стану персонажу та програвання відповідної анімації використовуються машини станів. По своїй суті машина станів являє собою

скінченний автомат. В цьому автоматі вказано такі стани: початковий, простоювання, пересування, початок стрибка, середина стрибка, кінець стрибка (рисунок 1.2). Кожен стан, окрім пересування, має одну вбудовану анімацію для програвання.

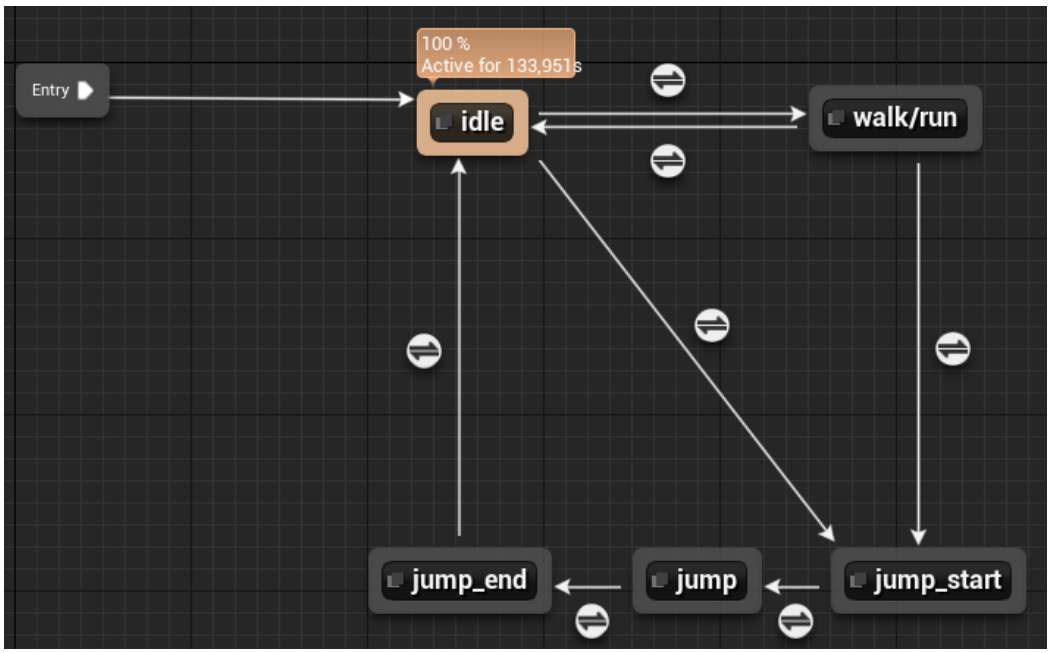


Рисунок 2.2 — Машина станів

Початковий стан э обов’язковим для машини станів. При спауні актору стан одразу переходить до наступного. Стан простоювання по суті є початковим і кінцевим одночасно: якщо ніяких команд не надходитиме, то буде обрано саме цей стан. Команда стрибка розбита на три стани, оскільки, як правило, є три окремі анімації для стрибка:

- Початок стрибка: програється як тільки надходить команда стрибка.
- Середина стрибка: програється одразу після попередньої анімації. Може повторюватися безліч разів.
- Кінець стрибка: програється як тільки персонаж торкається землі, анімація приземлення.

Оскільки ми ніколи не знаємо в якому напрямку буде рухатися наш персонаж, то для анімування пересування використовується спеціальна структура — Blend Space (рисунок 1.3). Ця структура дозволяє накопичувати

анімації, розташовувати їх на двомірній сітці, та програвати необхідну анімацію відповідно до вхідних параметрів.

Рисунок 2.3 — Blend Space для пересування

Для анімування пересування вхідними параметрами є швидкість на направлення. Опираючись на ці параметри можна точно вказати необхідну анімацію. Так, наприклад, якщо швидкість буде максимальною, а направлення дорівнюватиме 180°, то це значить, що персонаж бігом пересувається назад.

Для того, щоб машина станів працювала правильно, їй необхідно мати дані про нашого персонажа, а саме швидкість, направлення та чи знаходиться він у повітрі. Щоб отримати ці параметри у вкладці Event Graph виконується покадрове оновлення даних, відповідно до поточних даних персонажу.

Створити машину станів та структуру Blend Space недостатньо для гладкого пресування персонажу. Анімації будуть програватися правильно, але при спробі додати нову анімацію поверх тої, що вже відтворюється виникнуть проблеми. З метою їх уникнення виконується кешування анімацій та операція змішування анімацій.

Кешовані анімацій по своїй суті нагадують звичайні змінні — створюються з метою використання у подальших розрахунках. Кешування анімацій дозволяє мати декілька анімацій у запасі кожного кадру. Для гладкого пересування необхідно закешувати анімацію, що видає машина станів, та анімації які надходять із різних каналів, та змішати їх. Канали анімацій — потоки, що тримають інформацію про поточну анімацію в ньому. Так, наприклад, для реалізації анімації звичайної атаки використовується канал атаки, з якого

кешується потрібна нам анімація. Можна мати декілька каналів одночасно, та вирішувати, як саме необхідно відтворити всі анімації. Такий підхід дозволяє зробити анімації плавними та більш реалістичними.

Отримавши закешовані анімації, їх необхідно змішати. Змішування анімацій можливе завдяки скелетній структурі сітки. Обравши необхідну кістку в сітці можна розділити сітку на частину, що буде анімуватися ,наприклад, машиною станів, а інша — закешованою анімацію. Інтенсивність змішування відповідає за ступінь впливу кореневої анімації на вторинну.

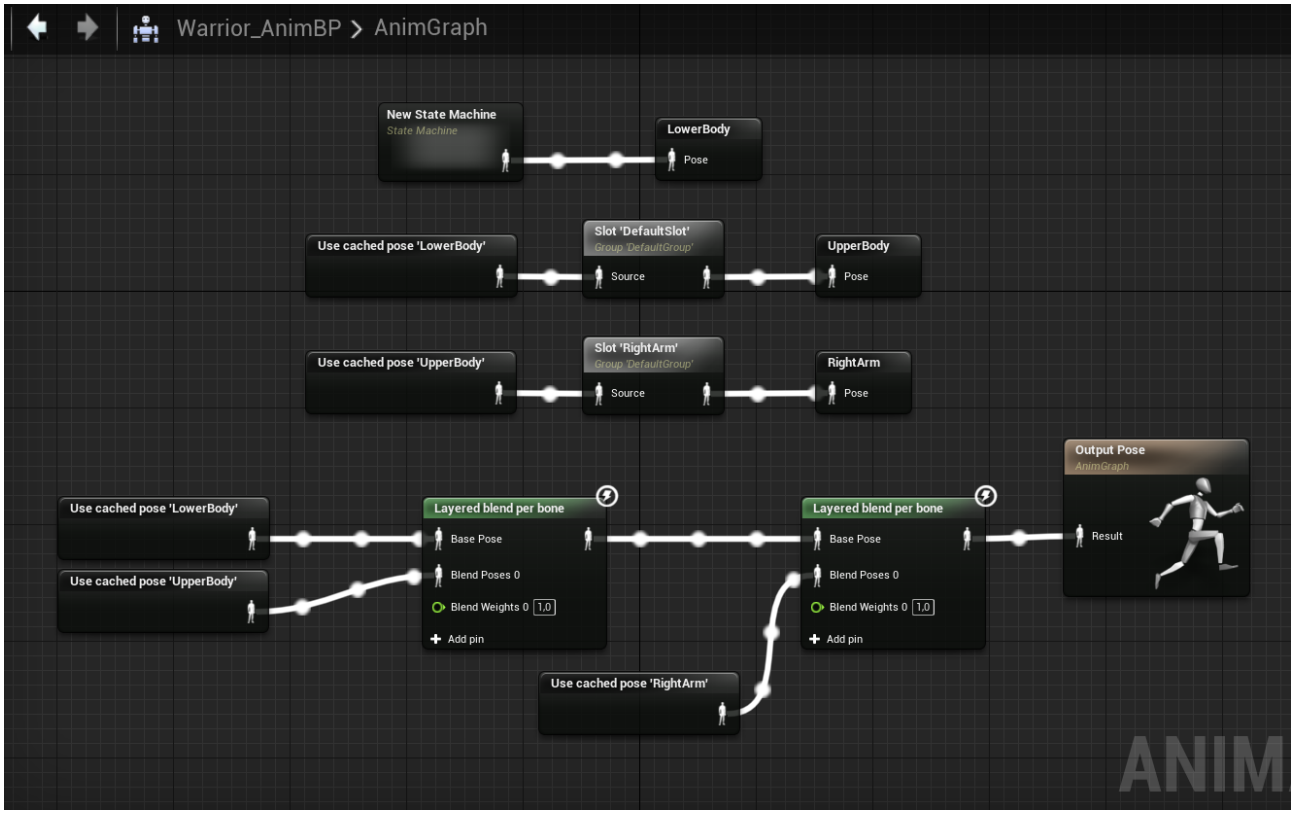


Рисунок 2.4 — Граф анімацій персонажу

На рисунку 1.3 показано, що для анімування персонажу спочатку кешується анімація машини станів як LowerBody, анімація з каналу DefaultSlot як UpperBody та анімація з каналу RightArm як RightArm. Після кешування всіх анімацій, вони по черзі змішуються, при чому кореневою анімацією завжди є анімація з кешу LowerBody. Кеш UpperBody надає анімації верхньої частини тіла, а кеш RightArm — анімації правої руки, це пізніше буде використано для анімування атаки. Якщо до каналів не надходить ніяких анімацій, то буде програватися коренева анімація, тобто анімація з машини станів. Будь-які

анімації, що надходять до інших двох каналів будуть додані до поточної анімації. Такий граф анімацій (рисунок 1.3) дозволяє персонажу плавно та неперервно відтворювати потрібні анімації, відповідно до його стану.

2.1.3 Зчитування подій, керування героєм

Гравець повинен керувати головним персонажем. В Unreal Engine є багато способів це реалізувати. В кожному проекті можна визначити свій унікальний список подій та прив’язати їх до декількох клавіш одночасно. Ці події можна використовувати в Blueprint класах акторів, для реалізації дій при натисканні. Для більш тонкого налаштування подій можна також підв’язувати клавіші в самому коді, а не в глобальних налаштуваннях проекту. Таким чином, операція, що буде виконуватися, буде залежати від її конкретної реалізації в керованому класі актору.

Unreal Engine дозволяє зробити проект кросплатформерним. Для налаштувань одного івенту для багатьох платформ необхідно вказати джерело виникнення цього івенту на різних пристроях вводу. Так, наприклад, для реалізації події стрибка можна прив’язати пробіл для клавіатури, стіки для геймпаду та трігери для пристроїв керування в віртуальній реальності (рисунок 2.5). За наявності декількох пристроїв керування, подія зможе бути спровокована будь-яким із них.

Після визначення методу реєстрації подій необхідно реалізувати логіку для виконання відповідних дій. Наприклад, якщо користувач натискає на пробіл, то керований персонаж повинен стрибнути. Логіка стрибка полягає в створенні відносно невеликого імпульсу вверх по координаті oz в світовому просторі. Після здійснення стрибка, персонаж відштовхнеться вгору, а приземлення відбудеться завдяки налаштуванням гравітації.

Рисунок 2.5 — Підв’язка події стрибка для різних пристроїв керування

Реалізація пересування персонажу в світовому просторі відбувається трохи складніше. Для цього необхідно визначити направляючий та правий вектор актору. Як видно із назви, правий вектор — це вектор, який завжди знаходиться під кутом в 90° до осі ox та oz, і завжди визначає праву сторону актору в локальних координатах. Для пересування актору необхідно також визначити характер пересування. Зазвичай, для цього в налаштуваннях проекту прив’язуються чотири клавіші, які визначають вектор в площині хоу. Одночасне натискання декількох таких кнопок призводить до підсумовування їх значень, що в результаті дає фінальний бажаний вектор для пересування. Відповідно до значення цього вектору, вбудований в актора направляючий вектор перераховується кожного кадру. Так, якщо вектор пересування дорівнюватиме нуль-вектору, тобто жодна з команд пересування не буде викликана, то персонаж не змінить своє положення в світовому просторі.

Для повноцінного керування головним героєм реалізовані наступні події: пересування, стрибок, атака, використання здібностей. Цей набір дозволяє стандартне пересування та бойові дії. Додатково були реалізовані події для взаємодії із навколишнім середовищем та інвентарем персонажу.

2.2 Графічний інтерфейс користувача

Будь яка подія в світі гри має або може мати деякий вплив на персонажа. Для відслідковування цього впливу в відеоіграх створюється окремий графічний інтерфейс користувача. HUD (Heads-Up Display), або прозорий дисплей, є невід'ємною частиною графічного інтерфейсу користувача. Завдяки HUD в іграх можна відстежувати стан персонажу та його основні характеристики.

В рамках проекту HUD складається із наступних компонентів:

- Шкала здоров'я;
- Шкала енергії;
- Шкала досвіду;
- Піктограми здібностей.

Всі шкали по своїй суті елементами класу Progress Bar. Їх значення та міра повноти напряму залежить від показників головного героя. Шкали здоров'я та енергії вказують на поточний рівень здоров'я та енергії відповідно. Ці шкали оновлюються кожного кадру. Шкала Досвіду явно демонструє кількість досвіду, необхідного для переходу на наступний рівень розвитку персонажу, та оновлюється, коли персонаж отримує досвід.

Піктограми здібностей вказують на доступність та готовність здібностей персонажу. Порожня піктограма вказує на недоступність здібності. При досягненні певного рівня піктограма буде вказувати на деяку здібність персонажу. Світла піктограма значить повну готовність здібності. Після використання здібності, вона на деякий час стає недоступною, а її піктограма стає затемненою. Процес переходу в стан готовності явно відображається на піктограмі поступовим її висвітленням.

2.2.1 Система інвентарю

В класичних іграх жанру RPG під час своїх подорожей персонаж може мати при собі певні витратні предмети. Для відслідковування наявності таких предметів використовується інвентар персонажу. Крім простого відслідковування кількості предметів, інвентар дозволяє взаємодіяти з ними. Прикладом предмету в інвентарі є зілля здоров'я: якщо в бою персонаж отримає занадто багато пошкоджень та його кількість очок здоров'я знизиться до критичного рівня, за наявності такого зілля в інвентарі, персонаж гравець зможе відкрити інвентар та використати зілля, щоб відновити частину очок здоров'я.

Описана в прикладі система інвентарю потребує наступні речі: можливість викликання графічного інтерфейсу інвентарю, можливість додавання предметів до інвентарю, можливість розпізнавання та взаємодії з предметами в середині інвентарю персонажа. По своїй суті інвентар є контейнером, який зберігає в собі певну кількість клітинок інвентарю (рисунок 2.6). В свою чергу така клітинка має містити в собі всю інформацію про предмет, який вона зберігає. Інформація, необхідна для повноцінної роботи із клітинками, утворює структуру та включає в себе такі поля:

- назва предмету;
- посилання на піктограму предмету;
- булева змінна, що вказує на можливість зберігання декількох подібних предметів в одній клітинці;
- посилання на клас актору предмету;
- опис предмету;
- максимальна кількість предметів однакового типу в клітинці.

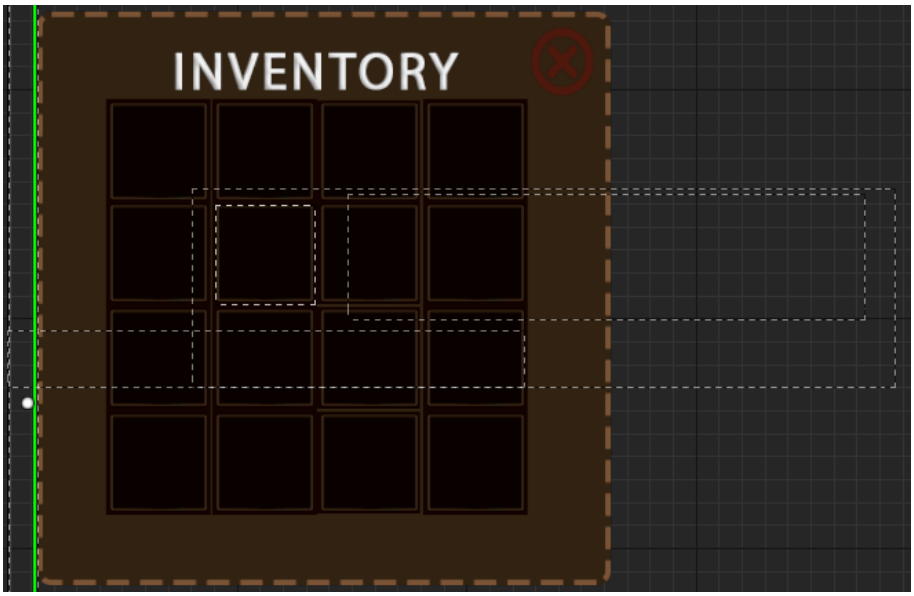


Рисунок 2.6 — Графічний інтерфейс інвентарю та структура клітинки

Для взаємодії з клітинками використовуються On Hover та On Click події. Із порожніми клітинками нічого не відбувається. Якщо користувач наведе курсор на заповнену клітинку, то під піктограмою предмету з’явиться опис його ефекту при використанні. При натисканні на піктограму предмету з’явиться меню для взаємодії в предметом із інвентарю. До таких взаємодій відноситься використання та викидання предмету. Використовуючи предмет, відбувається те, що написано в описі предмету, після чого із інвентарю вираховується один екземпляр предмету. При викиданні предмету із інвентарю поруч із самим персонажем з’явиться фізичний актор, що відповідає викинутому предмету, після чого із інвентарю вираховується один екземпляр предмету.

Наповнення інвентарю здійснюється за рахунок взаємодії із предметами в світі. Кожен такий предмет має два стани: фізичний, коли він існує в рамках світу, та прихований, коли він існує в рамках інвентарю. При взаємодії із такими фізичними об’єктами у світі, їх актор знищується, тобто вони перестають існувати як фізичні тіла, та додаються до інвентарю персонажу. Якщо кількість предметів одного типу в інвентарі перевищуватиме максимальну кількість однакового типу в клітинці, то вони будуть розподілені на декілька клітинок.

2.3 Допоміжні актори

Ігровий світ, в якому існує лише сам герой, є занадто порожнім для того, щоб бути повноцінною грою. В світі повинні бути предмети, із якими можна буде взаємодіяти. Крім того, кожен такий предмет повинен мати фізичного актора, щоб його можна було розташувати у якомусь місці на карті. Для створення гри реалізовані такі базові актори, як зброя та витратні ресурси. Також реалізовано скриню, як об’єкт взаємодії.

2.3.1 Прототип зброї

Зброя необхідна герої для реалізації бойової системи, без зброї герой не зможе виконувати базових атак. Оскільки головний герой — лицар, то в якості зброї він використовує меч. Скелетна сітка героя не має вбудованого меча, тому необхідно реалізувати його окремо. Сам герой має посилання на актора типу зброї як поле в класі.

Базовий клас зброї визначатиме загальну поведінку всіх можливих типів зброї. Цей клас також реалізовує інтерфейс Weapon Interaction. Сам актор зброї складається із статичної сітки, точкового джерела світла та компоненту кругового пересування. Сітка зброї використовується для визначення зовнішнього виду зброї. Точкове світло та обертання необхідні для візуального виділення об’єкту в світі. Знаходячись у світі актор зброї буде підсвічуватися та обертатись навколо своєї осі у повітрі. Гравець зможе з легкістю визначити, який саме об’єкт є зброєю.

Взаємодія з актором зброї полягає в підбиранні зброї з карти для подальшого її використання героєм, або заміну використовуваного типу зброї на інший. Прикріплення актору зброї до героя відбувається завдяки сокетам на скелетній сітці героя. Сокет – позначення спеціального місця на скелеті сітки як місця для взаємодії з іншими сітками. Після взаємодії із актором зброї в

світовому просторі, цільовий актор прикріплюється до вказаного сокету та переходить в локальні координати персонажу.

Головний герой має два сокети для зброї: для активної зброї, та для складеної зброї. Маючи ці сокети, необхідно підібрати необхідні анімації для використання меча. Логіка використання меча наступна:

- Якщо персонаж не має зброї, наступна взаємодія із актором зброї додасть її до сокету складеної зброї.
- Складена зброя буде активована при спробі атакувати, та перейде до сокету активної зброї.
- Зброя в активному стані дозволяє персонажу виконувати атаки.
- За бажанням персонаж може скласти зброю, що прикріпить її назад до сокету складеної зброї.

Всі ці дії вимагають окремого анімування: в машині станів не прописана логіка для використання зброї. Анімування зброї відбувається за допомогою спеціальних монтажів. Монтаж — спеціальна структура, яка містить в собі анімацію та дозволяє генерувати події на таймлайні цієї анімації. Наприклад, для складання зброї недостатньо програти саму анімацію, потрібно визначити в який час необхідно змінити сокет зброї, а використовуючи монтаж можна генерувати подію переходу під час програвання анімації.

Кожен створений монтаж має свій канал анімацій: монтажі атаки займають канал RightArm, а монтажі зміни стану зброї — DefaultSlot. Ці анімації змішуються із іншими анімаціями (розділ 2.1.2), не впливаючи на загальний стан персонажу.

Активна зброя дозволяє виконати персонажу серію до трьох атак поспіль. Кожна атака в серії наносить більше очок пошкоджень ніж попередня. Для виконання серії в монтажах анімацій вказані спеціальні вікна, які сприймають повторну активацію атаки як продовження попередньої атаки.

Створивши прототип зброї, можна створити багато різновидів зброї шляхом наслідування прототипу.

2.3.2 Витратні ресурси

Витратні ресурси — ресурси, які мають фізичного актора, а взаємодія з ними переносить їх до інвентарю (розділ 2.2.1). Такі актори реалізують інтерфейс Interaction. Подібно до зброї, витратні ресурси мають базовий клас, наслідуючи який можна зробити будь-яку кількість подібних ресурсів.

На відміну від базового класу актора зброї, базовий клас актору витратних ресурсів має вбудовану фізику. Сам актор складається із таких компонентів: статична сітка, сітка кубу, компонент сферичної колізії. Статична сітка визначає зовнішній вигляд актору. Сітка кубу є невидимою для гравця, та використовується для симуляції фізики. Фізика встановлюється для кожної сітки по різному. Базова статична сітка актору може не мати фізики, а стандартна сітка куба має свою вбудовану фізику. Поєднання двох сіток дозволяє симулювати фізику на будь-якому об’єкті. Компонент колізії визначає радіус, в якому із актором можна взаємодіяти. Необхідною умовою для взаємодії із таким актором є перетинання власного компоненту колізії із компонентом головного героя.

Через наслідування базового класу актору витратних матеріалів реалізовано клас зілля зцілення, та зілля відновлення енергії. Кожне зілля може мати максимум по три предмети в одній комірці інвентарю. При використанні зілля відновлюють відповідні ресурси головного героя.

2.3.3 Скриня

Скриня є об’єктом для взаємодії в навколишньому середовищі. При взаємодії скриня відкривається, та з неї падають певні предмети. Скриня є статичним об’єктом, та може бути відкрита лише один раз.

Актор скрині має дві статичні сітки, що відповідають за основу та кришку об’єкту скрині. Відкриття відбувається за допомогою компоненту обертального руху. При відкритті кришка скрині провертається на певну кількість градусів. Як і у інших акторів, з якими можна взаємодіяти, скриня має компонент колізії. Цей

компонент визначає відстань, з якої зі скринею можна взаємодіяти. Необхідною умовою для взаємодії із актором є перетинання власного компоненту колізії із компонентом головного героя.

Сам клас скрині містить поля з посиланнями на акторів, які випадуть з неї при відкритті. Незважаючи, на те, що в середині скриня порожня, вона не містить фізичних акторів всередині себе. Натомість, при відкритті скрині створюються нові актори на її координатах. Тип акторів визначається для кожної скрині індивідуально.

Візуальним індикатором можливості відкрити скриню є невеликий віджет, який з’являється коли персонаж знаходиться досить близько до неї. При натисканні кнопки взаємодії, скриня відкриється, після чого створяться нові актори. Ці актори при створенні мають невеликий імпульс, який виштовхує їх із скрині.

2.4 Вороги

Вороги представляють сили зла, з якими бореться головний герой. Вони будуть намагатися вбити головного героя впродовж гри. Кожен тип ворога реагує на гравця по-своєму.

2.4.1 Принцип створення ворогів

Кожен тип ворога є унікальним, але принцип їх створення є схожим. Кожен тип ворогів має свою скелетну сітку та набір анімацій. Подібно до персонажу головного героя, вороги повинні мати акторів із своїми скелетними сітками.

Прикладом одного з ворогів є павук. Актор павука має наступні компоненти: скелетна сітка, компонент капсульної колізії, компонент стрілки, компонент переміщення та компонент сприйняття інформації. На відміну від актору персонажу, актор павука не має компонентів, що пов’язані із камерою. Він не отримує ніяких прямих команд від користувача. Натомість він має

додатковий компонент, який симулює відчуття актору. Цей компонент використовується для створення штучного інтелекту, він дозволяє налаштовувати радіус слуху та зору актору. При отримуванні інформації про наявність збудників чуття, генеруються спеціальні події. Оброблюючи ці події можна вказувати реакцію на різні збудники, посилаючись на джерело збудження. Так, коли в радіус зору ворога потрапляє персонаж гравця, ворог намагатиметься наздогнати його.

Скелетні сітки ворогів можуть сильно відрізнятися від скелетної сітки головного персонажу (рисунок 2.7). Це означає, що набір анімацій створених для персонажу не підходить для використання при анімуванні ворогів — анімації виконуються завдяки визначеній структурі кісток. Кістики, необхідні для програвання певних анімацій, можуть бути відсутні або переставлені в інше місце, за рахунок чого анімація відтворюватиметься неправильно або взагалі не відтворюватиметься. Тому для кожної унікальної сітки ворога необхідно мати також набір анімацій для відповідного набору кісток.

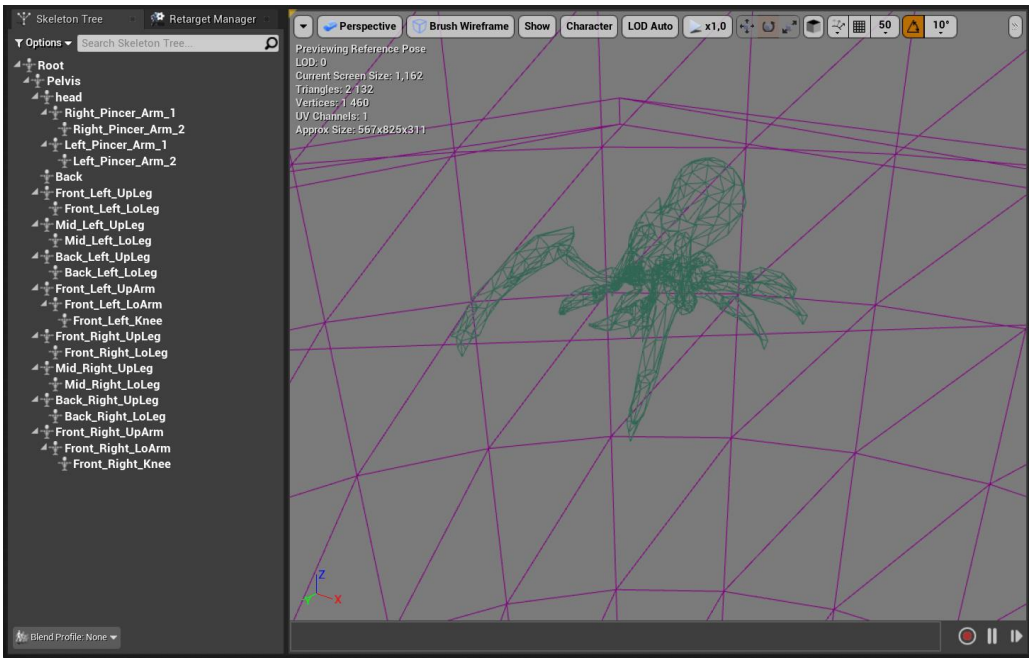


Рисунок 2.7 — Скелетна сітка павука

За наявності сітки, матеріалу сітки та анімацій сітки можна розробити проанімовану версію ворога. Процес анімування таких акторів схожий на процес анімування головного героя (розділ 2.1.2), за виключенням того, що для

відтворення анімацій та монтажів необхідно використовувати різні канали анімацій. Використання однакових каналів анімацій може призвести до конфліктів під час кешування анімацій.

2.4.2 Поведінкові дерева, реакція на героя

Маючи готового, проанімованого актора, необхідно розробити методи пересування для цього актора. Оскільки актори не керуються гравцем, їм потрібно визначити методи отримання інформації та команд на виконання відповідних дій. В Unreal Engine.

Необхідною умовою для створення штучного інтелекту для акторів є присвоєння їм компоненту, що відповідають за відчуття. Ці компоненти перевіряють наявність збудників в заданому радіусі або конусі, та, аналізуючи отриману інформацію, виконують запрограмовані дії.

Для визначення поведінки актору, який не контролюється гравцем використовуються поведінкові дерева. По своїй суті поведінкові дерева є скінченними автоматами.

Поведінкові дерева утворюються із трьох основних компонентів: кореня, селектора та послідовності. Корінь дерева – його початкова точка. Селектор — вузол, який виконує послідовності зліва направо при виконанні необхідних умов. Послідовність — вузол , який виконує всі вузли зліва направо. Для реалізації поведінкового дерева павука використано один селектор та три послідовності.

Кожна послідовність виконує завдання, через які можна маніпулювати павуком. Unreal Engine має набір стандартних завдань, але їх недостатньо для реалізації задуманої моделі поведінки павука. Додатково були розроблені наступні завдання:

- AttackPlayer: наказ виконати атаку;
- ContinuePath: наказ продовжити патрулювання;
- GetPlayerLocation: допоміжне завдання, отримує координати гравця;

- GetSplineLocation: допоміжне завдання, використовується для відслідковування маршруту патрулювання;
- GetStrafeLocation: допоміжне завдання, обирає точку справа чи зліва від актору для подальшого пересування до неї;
- PausePath: наказ призупинити патрулювання;
- RotateToPlayer: допоміжне завдання, отримує кут повороту до гравця;
- RotateToPathway: допоміжне завдання, отримує кут повороту до маршруту патрулювання.

Перша послідовність відповідає за поведінку павука в спокійному стані. Вважається, що павук знаходиться в спокійному стані, якщо він не бачить гравця. За таких умов необхідно відтворити певну поведінку переміщення павука. Він може патрулювати територію в очікуванні гравця, ходити до випадкового місця, або просто стояти на місці.

Для патрулювання місцевості необхідно вказати маршрут. Маршрут реалізовано у вигляді окремого актора, невидимого для гравця. Цей актор має сітку куба, сплайн та компонент колізії у вигляді сфери. Сплайн може бути відредагований у редакторі рівня та, власне, визначає маршрут. Сітка куба потрібна для прикріплення до неї елемента колізії, сам сплайн не може мати колізій. Куб сам по собі не має жодних колізій та є невидимим для гравця. Куб має власну швидкість та буде слідувати по траєкторії сплайну. Компонент колізії вказує на приближення актору, що патрулює. Якщо колізія відбувається, це значить що актор буде слідувати за кубом, а куб в свою чергу рухається по траєкторії сплайу. В іншому випадку колізія не відбувається, куб буде стояти на місці та чекати на актора, що патрулює.

Друга послідовність відповідає за слідування за персонажем гравця. Якщо компонент чуття визначить, що в радіусі зору павука є гравець, то павук почне наздоганяти його. Таким чином стан павука переходить із спокійного в стан погоні. Під час погоні потрібно розвернути актора в сторону гравця, поставити

маршрут на паузу, вказати швидкість із якою павук буде переміщатись та вказати умову виходу із стану. Умовою виходу зі стану є достатнє наближення до гравця.

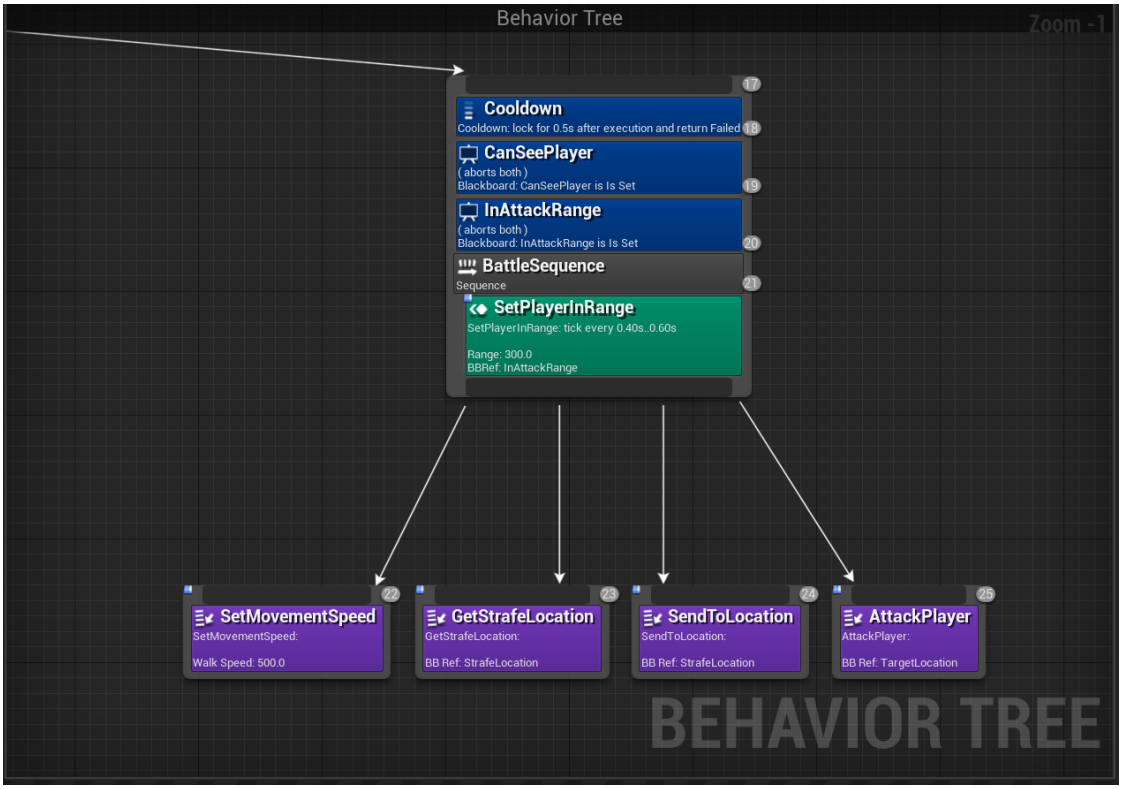


Рисунок 2.8 — Послідовність бою в поведінковому дереві павука

Третя послідовність відповідає за бойові дії павука. Коли павук зближається з персонажем, він переходить зі стану погоні в стан бою. В бойовому стані павук починає пересуватись навколо персонажу та виконує атаки. Анімації атаки виконані через монтажі. Якщо павук втрачає грація з поля зору, то він повертається до патрулювання місцевості.

Для виконання поведінкового дерева необхідно створити клас штучного інтелекту, в якому буде вказана команда виконання цього поведінкового дерева. Посилання на цей клас повинно бути присвоєне актори, який використовуватиме цей екземпляр класу штучного інтелекту.

2.5 Рівень для гри

Рівень у грі визначає ігровий простір для гравця. На рівні мають бути розташовані актори для їх відображення в грі. Сам рівень та його наповненість створюють спеціальні дизайнери рівнів.

Рівень складається із видимих статичних сіток, джерел світла та часток, невидимих компонентів. Статичні сітки визначають загальний ландшафт рівню та створюють сам ігровий простір. Джерела світла визначають властивості освітлення на рівні, рівень без джерел світла буде повністю чорним, в ньому нічого не буде видно. Джерела часток дозволяють визначають властивості різних візуальних ефектів на рівні, в тому числі погодних ефектів. Невидимі компоненти обмежують гравця в пересуванні на рівні (невидимі стіни) або визначають простір для спеціальних взаємодій, наприклад простір для пересування штучного інтелекту.

Рисунок 2.9 — Карта морозної печери

Для реалізації рівню гри було використано карту морозної печери (рисунок 2.9). Ця картань розташована на Unreal Engine Marketplace, та налічує в собі більше ніж вісімсот компонентів.

Висновки

Для реалізації гри в жанрі RPG засобами Unreal Engine було виконано наступні задачі: створення головного персонажу, персонажів ворогів та допоміжних акторів; реалізована обробка подій користувача та генерація власних подій; створено ігровий простір та рівень для гри.

Після проведення досліджень була розроблена правильна структура для анімування персонажів. Всі динамічні персонажі в грі є проанімованими. Персонаж гравця має налагоджену систему керування. Персонажі, що не керуються гравцем, мають розроблений штучний інтелект, що дозволяє їм виконувати дії в ігровому просторі.

Ігровий простір представлено у вигляді рівні. Самі рівні насичені акторами. Кожен актор, що має індивідуальні налаштування, був налагоджений індивідуально для кожного рівня.

РОЗДІЛ 3. АНАЛІЗ СТВОРЕНОЇ ГРИ

В розділі 3 ми проведемо аналіз основного функціоналу створеної гри. Як і будь-яка інша гра, розроблена гра повинна мати список системних вимог. Також проведемо розбір керування у грі та розглянемо її основні механіки.

3.1 Інструкція користувача

Під час гри, гравець контролює персонажа лицаря. Керування здійснюється за допомогою клавіатури та комп'ютерної миші. Мета гри — пройти всі рівні. Всього у грі є п'ять рівнів.

Пересування персонажу здійснюється через використання клавіш WASD для бігу та пробілу для стрибка. В ігровому просторі розташовані різні об'єкти для взаємодії. До них належать: зброя, витратні матеріали, скриня. Взаємодія із цими предметами виконується через клавішу E.

При взаємодії зі зброєю, персонаж змінює екіпіровану зброю на об'єкт взаємодії. За наявності зброї персонаж може виконувати звичайні атаки. Для їх виконання використовується ліва кнопка миші. Персонаж може виконувати серію до трьох атак поспіль. Персонаж може скласти зброю натиснувши на клавішу R. Крім звичайних атак, персонаж може використовувати спеціальні здібності використовуючи клавіші 1-5.

При взаємодії із витратним ресурсом, персонаж підбирає його. Ці ресурси зберігаються в інвентарі персонажу. Доступ до інвентарю здійснюється за допомогою клавіші I. В середині інвентарю персонаж може переглядати список всіх зібраних ресурсів, а також використовувати їх. Дію ефект при використанні ресурсу також описано в інвентарі.

Персонаж має показники здоров'я, енергії та досвіду. Здоров'я — головний ресурс персонажу. Коли показник здоров'я падає до нуля гра завершується та починається спочатку. Енергія — це ресурс, який використовується для використання здібностей. Ці два ресурси поступово відновлюються впродовж

гри. Показник досвіду визначає рівень персонажу під час гри. Персонаж отримує певну кількість очок досвіду коли перемагає ворога. Чим вище рівень персонажу, тим вище його показники. Відслідковувати за показниками персонажу можна за допомогою інтерфейсу користувача.

Основною перешкодою для персонажу на його шляху є ворожі сили зла. До ворогів належать павуки, вовки, тролі, бомбери та роботи. Кожен тип ворога має свої власні характеристики та поведінкові шаблони. В кінці шляху на гравця чекає бос гри — король кузні. Після перемоги над босом гра вважається пройденою.

3.2 Системні вимоги

Шляхом запуску на декількох машинах, були визначені рекомендовані параметри необхідні для стабільної роботи гри (таблиця 3.1)

Таблиця 3.1 — Рекомендовані системні вимоги

Характеристика	Вимога
Процесор	Intel® Core™ i5-4200H
Частота процесору	2.80 GHz
Операційна система	Windows 10
Тип системи	64-розрядна операційна система
Розмір оперативної пам’яті	4 Гб
Частота оперативної пам’яті	1600 МГц
Модель відеокарти	NVIDIA GeForce GTX 850M
Об’єм відеопам’яті	4 Гб

Розроблене програмне забезпечення може не працювати на комп’ютерах, системні параметри яких нижче зазначених. Проект було розроблено на Unreal Engine версії 4.25.4 та, у зв’язку з запланованим релізом Unreal Engine 5, може бути повністю перенесеним на оновлений движок. В такому випадку системні вимоги можуть відрізнятися від зазначених.

ВИСНОВКИ

В рамках виконання кваліфікаційної роботи бакалавра було розглянуто різні жанри комп’ютерних ігор. Були розглянуті різні ігрові рушії та описані їх переваги. Також було проведено аналіз найбільших переваг Unreal Engine серед інших ігрових рушіїв.

Досліджено систему Blueprints Visual Scripting, що є вбудованою системою візуального програмування в Unreal Engine. Завдяки цій системі реалізовано всі основні компоненти гри.

Було реалізовано власні об’єкти для створення гри. Результатом виконаної роботи є гра, яка включає в себе базовий функціонал заздалегідь продуманого ігрового процесу. Створено ігровий простір у вигляді п’яти рівнів, на яких розміщені 3D моделі ігрових об’єктів.

До базових функцій проекту належать:

- Пересування в середині ігрового простору.
- Реалізовано функціонал для взаємодії із навколишнім світом.
- Створено ряд типів ворогів.
- Розроблено системи штучного інтелекту для ворогів.
- Налаштовані анімації для відповідних об’єктів.

Були розглянуті різні методи створення відеоігор за допомогою ігрового рушія Unreal Engine, розглянуті різні інструменти, вбудовані в цей рушій. В результаті поставлена задача створення гри була досягнута, всі необхідні задачі були виконані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Takahashi D. More than 1.2 billion people are playing games [Електронний ресурс] / Dean Takahashi. – 2021. – Режим доступу до ресурсу: https://venturebeat.com/2021/01/27/driving-game-growth-a-day-one-roundup-of-the-all-digital-gamesbeat-event/.

2. Sewell B. Blueprints Visual Scripting for Unreal Engine / Brenden Sewell., 2015.

3. Cookson A. Unreal Engine 4 Game Development in 24 Hours, Sams Teach Yourself / A. Cookson, C. Crumpler, R. Dowling., 2016.

4. Tavakkoli A. Game Development and Simulation with Unreal Technology / Alireza Tavakkoli., 2015.

5. Moniem M. Unreal Engine Lighting and Rendering Essentials / Muhammad Moniem., 2015.

6. Stagner A. Building an RPG with Unreal 4.x / A. Stagner, S. Santello., 2016.

7. Carnall B. Unreal Engine 4.X By Example / Benjamin Carnall., 2016.

8. Lee J. Unreal Engine: Game Development from A to Z / J. Lee, J. Doran, N. Misra., 2016.

9. Jenkins J. Your Ultimate Guide to Character Development: 9 Steps to Creating Memorable Heroes [Електронний ресурс] / Jerry Jenkins. – 2021. – Режим доступу до ресурсу: https://jerryjenkins.com/character-development/.

10. MARKETPLACE FREQUENTLY ASKED QUESTIONS (FAQ) [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.unrealengine.com/en-US/marketplace-faq.

11. What is Unreal Engine? [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.educba.com/what-is-unreal-engine/.

12. UE4: Beginner's Step-by-Step to Creating Your First Level [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://www.worldofleveldesign.com/categories/ue4/ue4-step-by-step-first-simple-level.php.

13. Buckley D. Beginner's Guide to Game Development with Unreal Engine [Електронний ресурс] / Daniel Buckley. – 2020. – Режим доступу до ресурсу: https://gamedevacademy.org/unreal-engine-tutorial/.

14. Get Started with UE4 [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.unrealengine.com/en-US/Basics/GettingStarted/index.html.

15. Tran T. Unreal Engine 4 Blueprints Tutorial [Електронний ресурс] / Tommy Tran. – 2017. – Режим доступу до ресурсу: https://www.raywenderlich.com/663-unreal-engine-4-blueprints-tutorial#toc-anchor-001.

16. Godot Engine [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://summerofcode.withgoogle.com/archive/2020/organizations/4877284235804672/.

18. GAMEMAKER STUDIO 2 [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.yoyogames.com/ru/gamemaker.

19. What is CRYENGINE? [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.capterra.com/p/210664/CRYENGINE/.

20. What is a Game Engine? [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://gamescrye.com/blog/what-is-a-game-engine/.

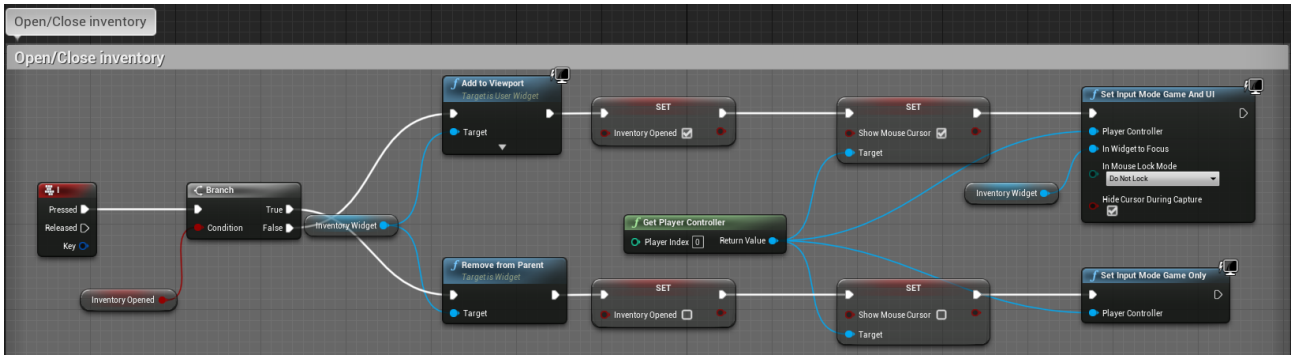
21. The Power of Video Game Engines: Every Game Developer's (Not-So-Secret) Weapon Share [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.gamedesigning.org/career/video-game-engines/.

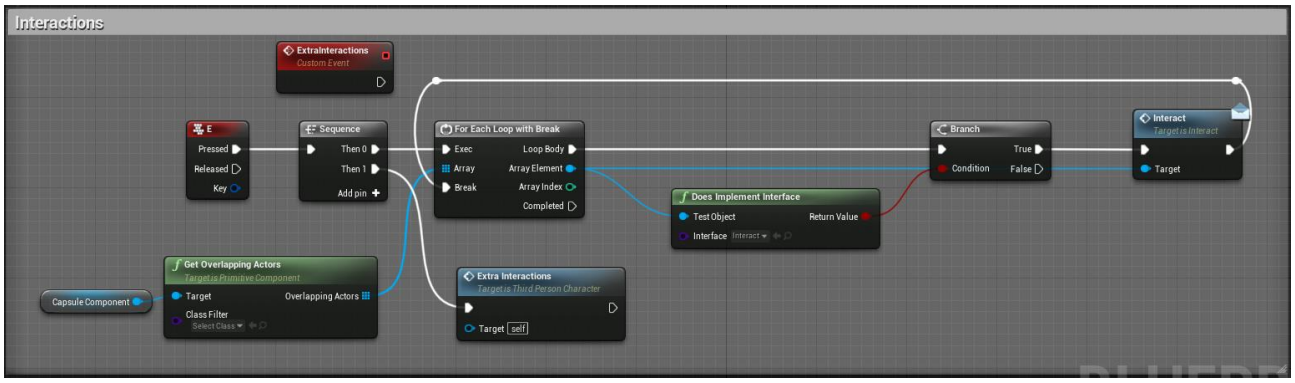
22. Dar R. Top 7 Gaming Engines You Should Consider for 2021 [Електронний ресурс] / Renana Dar. – 2021. – Режим доступу до ресурсу: https://www.incredibuild.com/blog/top-7-gaming-engines-you-should-consider-for-2020.

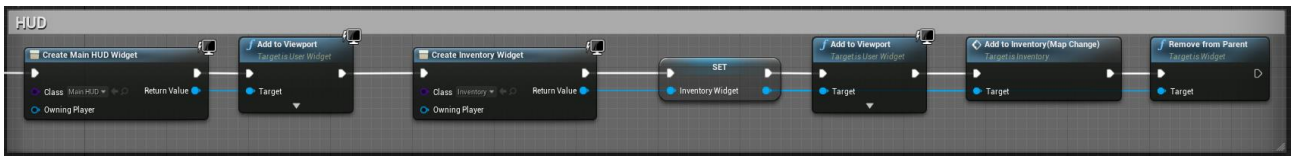
23. Perron B. The Video Game Theory Reader 2 / B. Perron, M. Wolf., 2009.

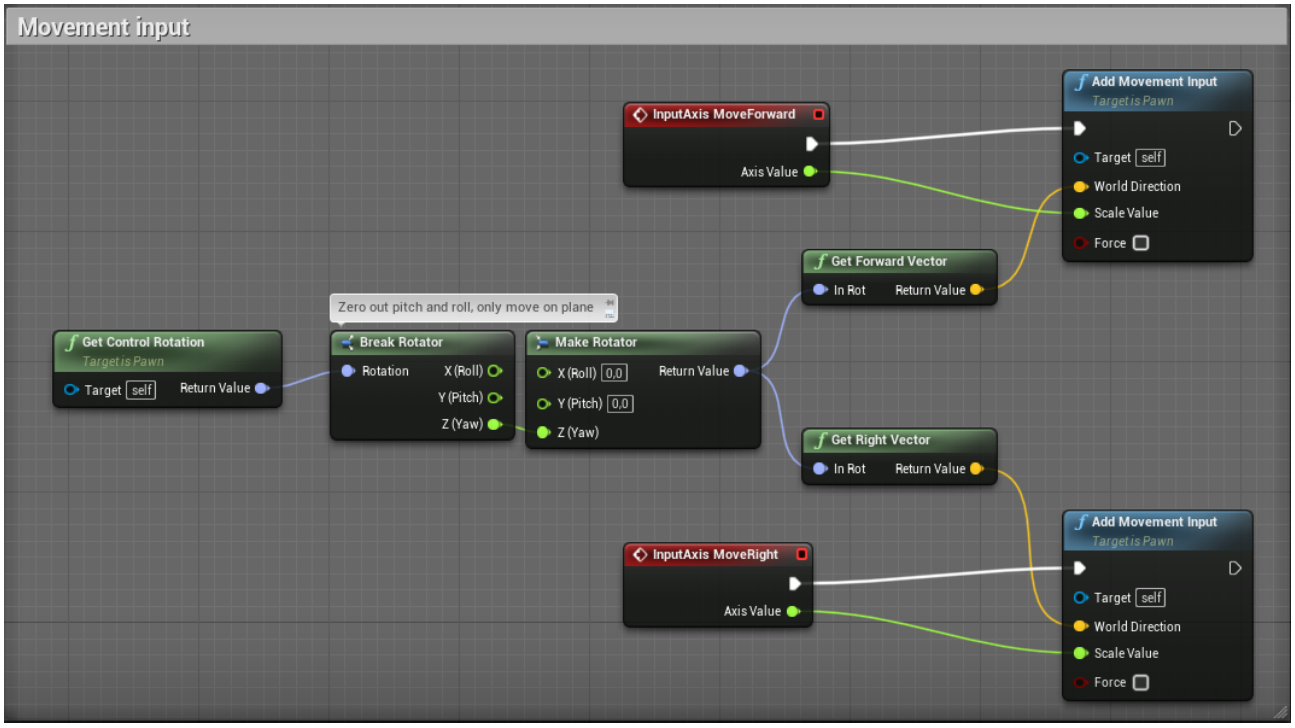
ДОДАТКИ

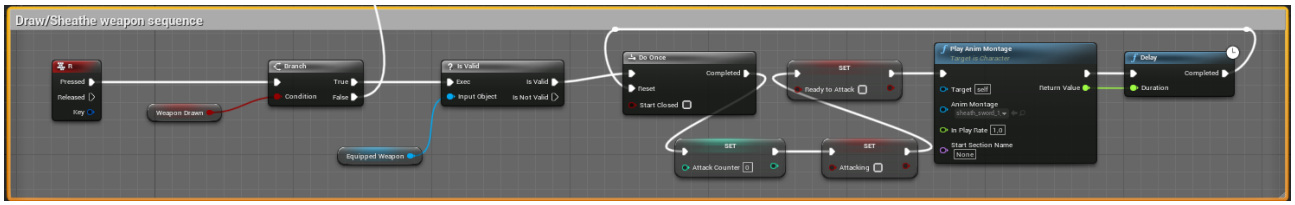
Додаток А. Керування головним персонажем

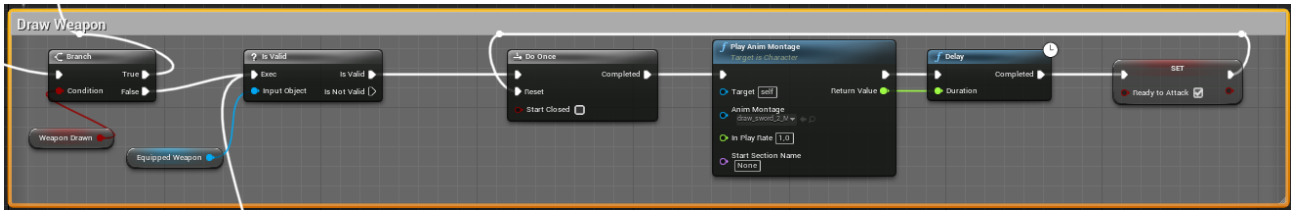


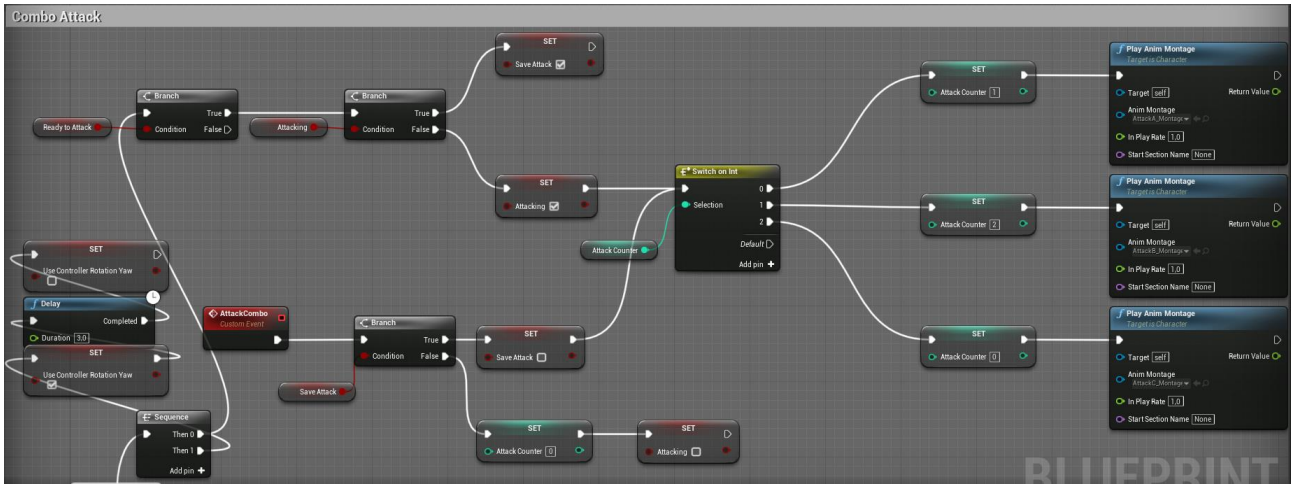


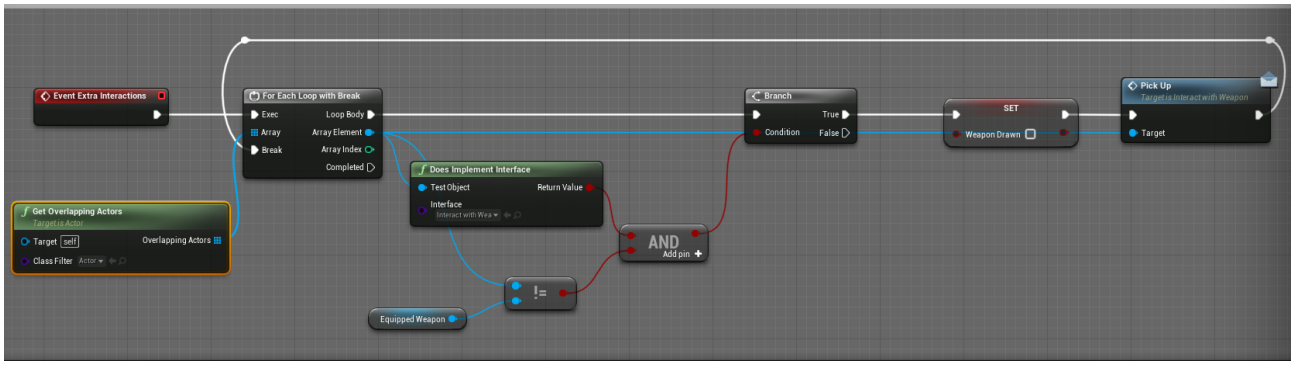


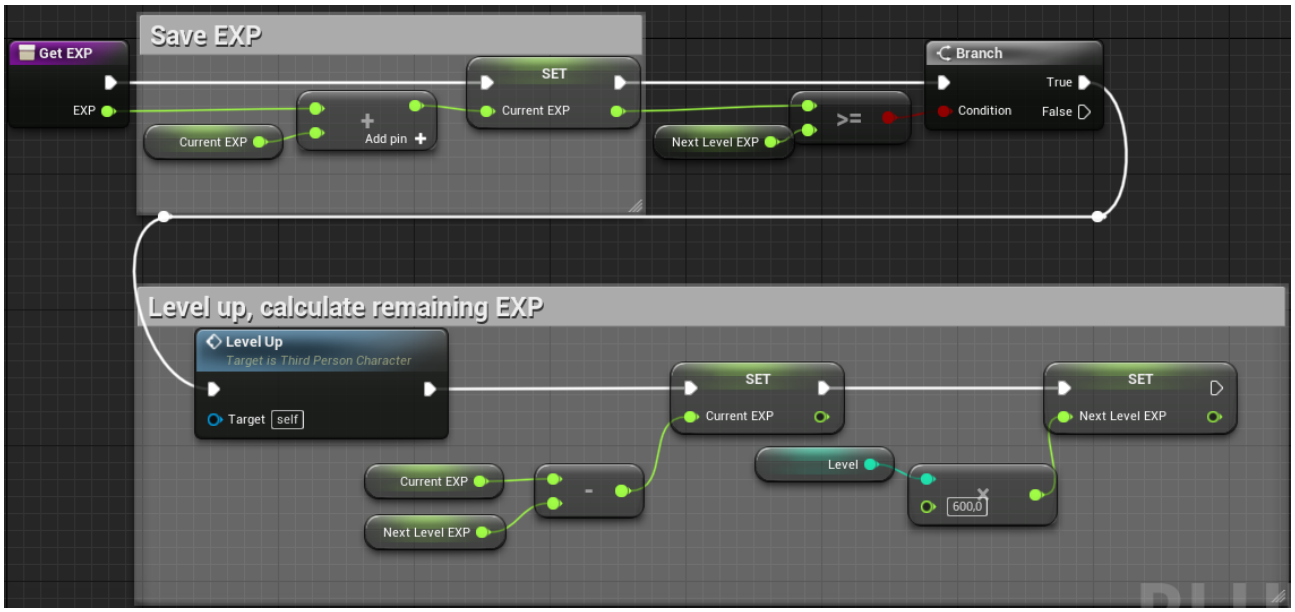


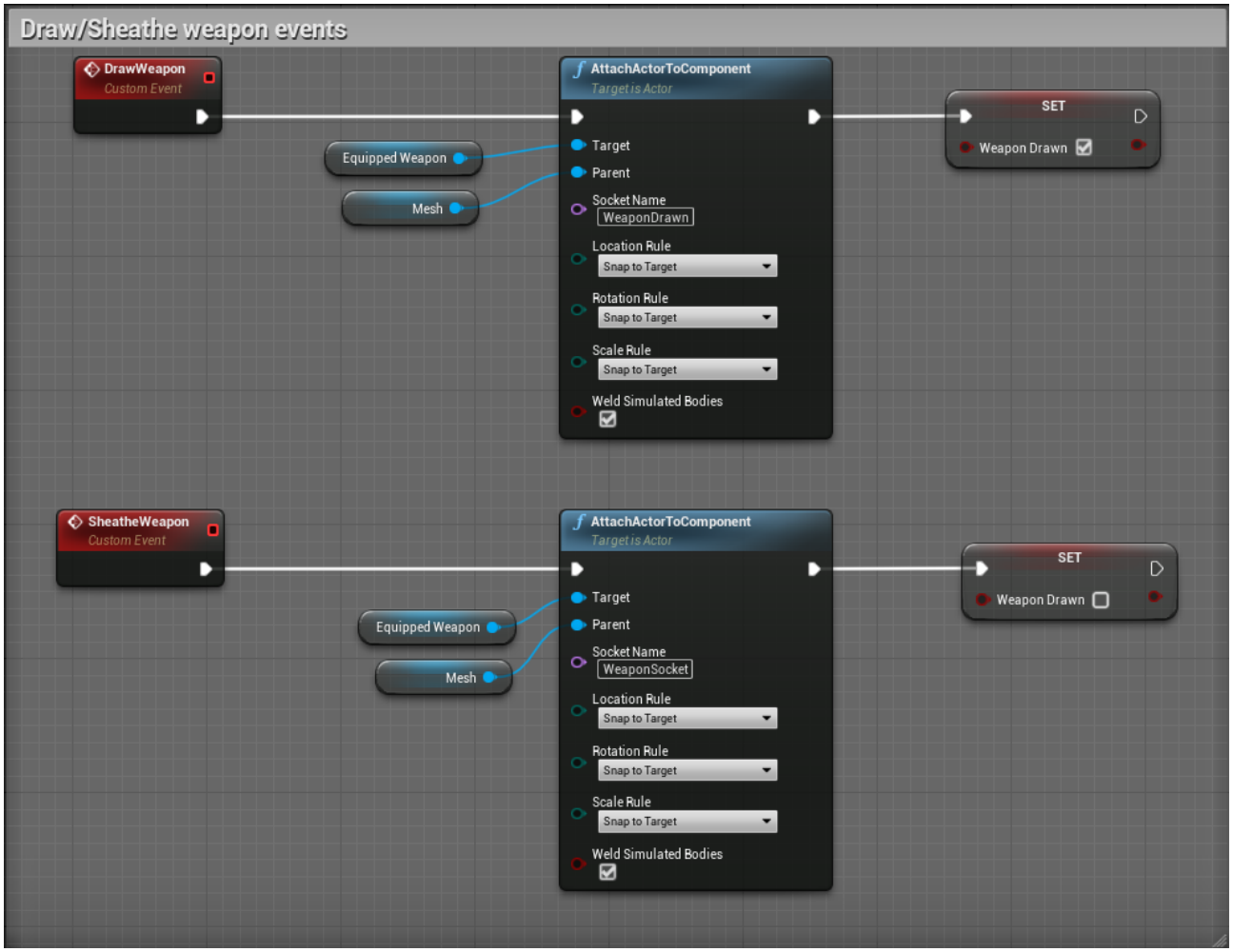

Draw/Sheathe weapon sequence
R
Pressed
Released
Key
Weapon Drawn
Branch
Condition
True
False
Is Valid
Input Object
Is Valid
Is Not Valid
Equipped Weapon
Do Once
Reset
Start Closed
Completed
SET
Ready to Attack
SET
Attack Counter
SET
Attacking
Play Anim Montage
Target is Character
Target
self
Anim Montage
In Play Rate
1.0
Start Section Name
None
Return Value
Delay
Duration
Completed


Draw Weapon
Branch
Condition
True
False
Weapon Drawn
Is Valid
Input Object
Is Valid
Is Not Valid
Equipped Weapon
Do Once
Reset
Start Closed
Completed
Play Anim Montage
Target is Character
Target
self
Anim Montage
In Play Rate
1.0
Start Section Name
None
Return Value
Delay
Duration
Completed
SET
Ready to Attack

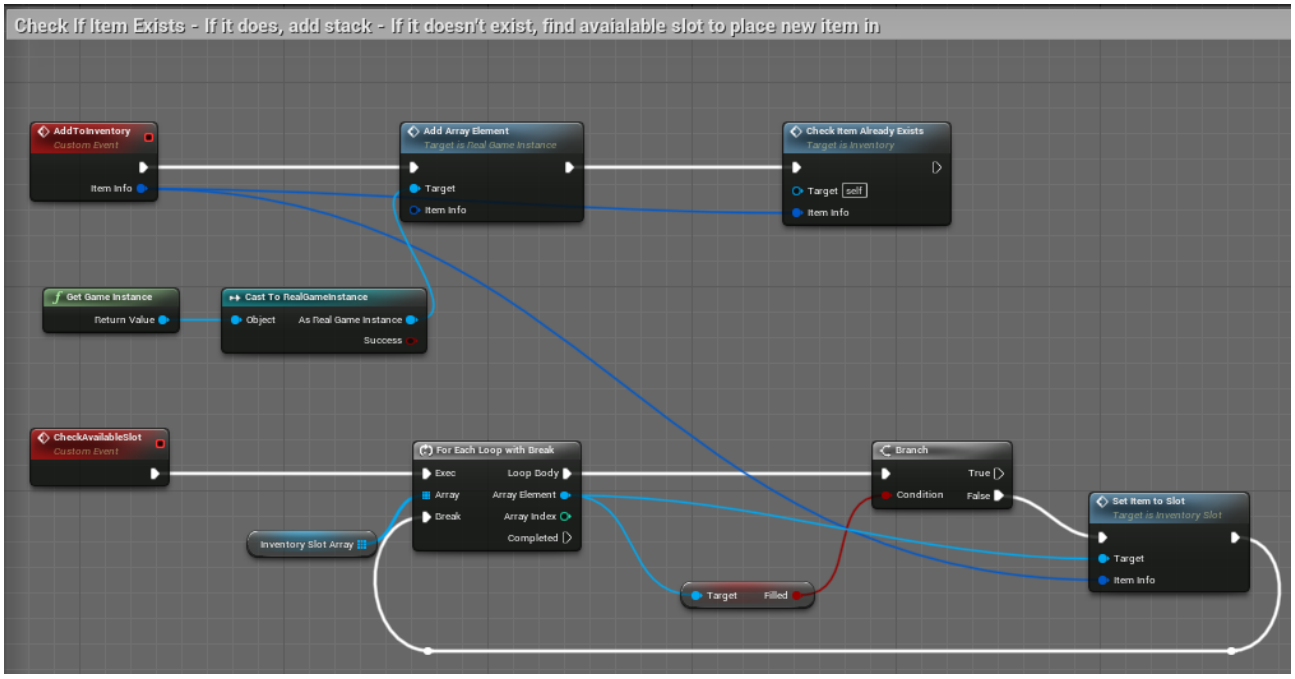

Combo Attack
Ready to Attack
Branch
Condition
True
False
Attacking
Branch
Condition
True
False
SET
Save Attack
SET
Attacking
Switch on Int
Selection
0
1
2
Default
Add pin
Attack Counter
SET
Attack Counter
1
SET
Attack Counter
2
SET
Attack Counter
0
Play Anim Montage
Target is Character
Target
self
Anim Montage
In Play Rate
1.0
Start Section Name
None
Return Value
SET
Use Controller Rotation Yaw
Delay
Duration
3.0
Completed
SET
Use Controller Rotation Yaw
AttackCombo
Custom Event
Branch
Condition
True
False
SET
Save Attack
Save Attack
Sequence
Then 0
Then 1
Add pin
SET
Attack Counter
0
SET
Attacking
BLUEPRINT

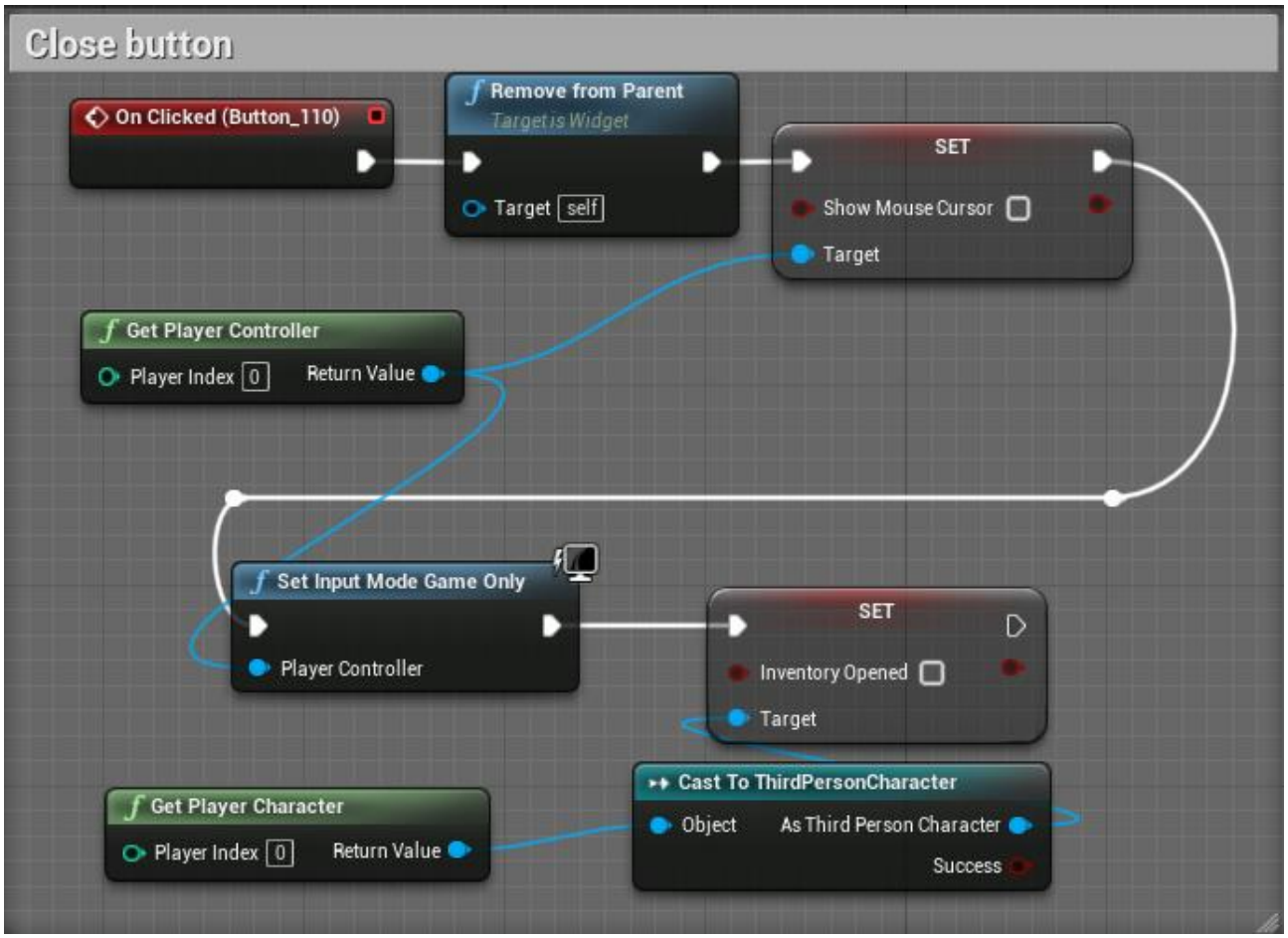

Event Extra Interactions
For Each Loop with Break
Exec
Array
Break
Loop Body
Array Element
Array Index
Completed
Get Overlapping Actors
Target is Actor
Target
self
Class Filter
Actor
Overlapping Actors
Does Implement Interface
Test Object
Interface
Return Value
AND
Add pin
Equipped Weapon
Branch
Condition
True
False
SET
Weapon Drawn
Pick Up
Target is Interact with Weapon
Target

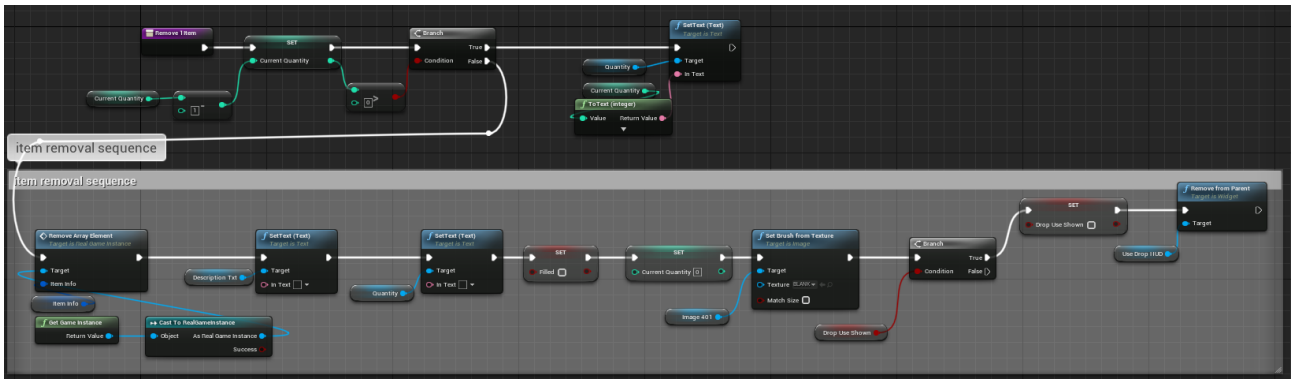

Save EXP
Get EXP
EXP
SET
Current EXP
Add pin
Current EXP
Next Level EXP
>=
Branch
True
Condition
False
Level up, calculate remaining EXP
Level Up
Target is Third Person Character
Target
self
SET
Current EXP
SET
Next Level EXP
Current EXP
Next Level EXP
Level
600,0

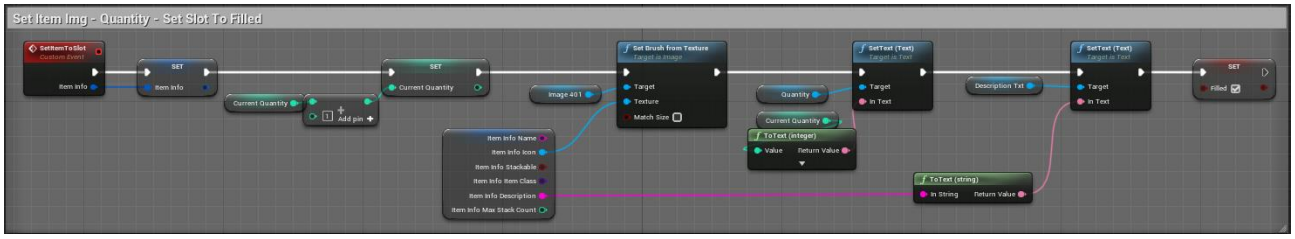

Draw/Sheathe weapon events
DrawWeapon
Custom Event
AttachActorToComponent
Target is Actor
Target
Parent
Socket Name
WeaponDrawn
Location Rule
Snap to Target
Rotation Rule
Snap to Target
Scale Rule
Snap to Target
Weld Simulated Bodies
Equipped Weapon
Mesh
SET
Weapon Drawn
SheatheWeapon
Custom Event
AttachActorToComponent
Target is Actor
Target
Parent
Socket Name
WeaponSocket
Location Rule
Snap to Target
Rotation Rule
Snap to Target
Scale Rule
Snap to Target
Weld Simulated Bodies
Equipped Weapon
Mesh
SET
Weapon Drawn

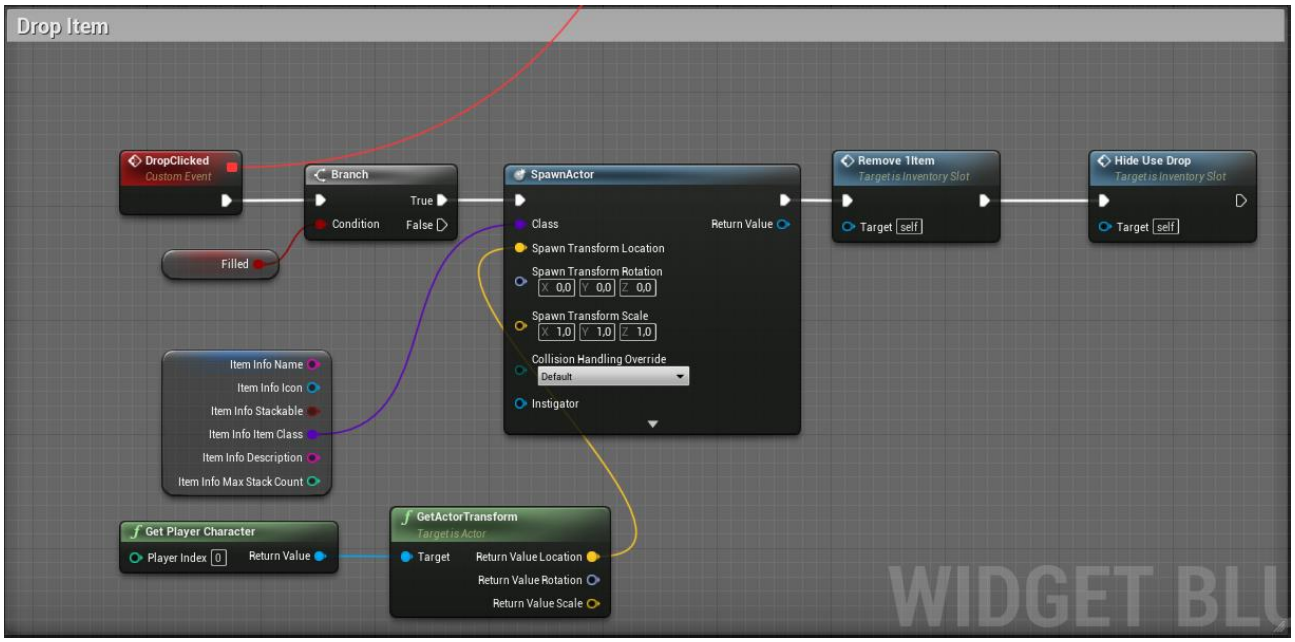
Додаток Б. Реалізація системи інвентарю

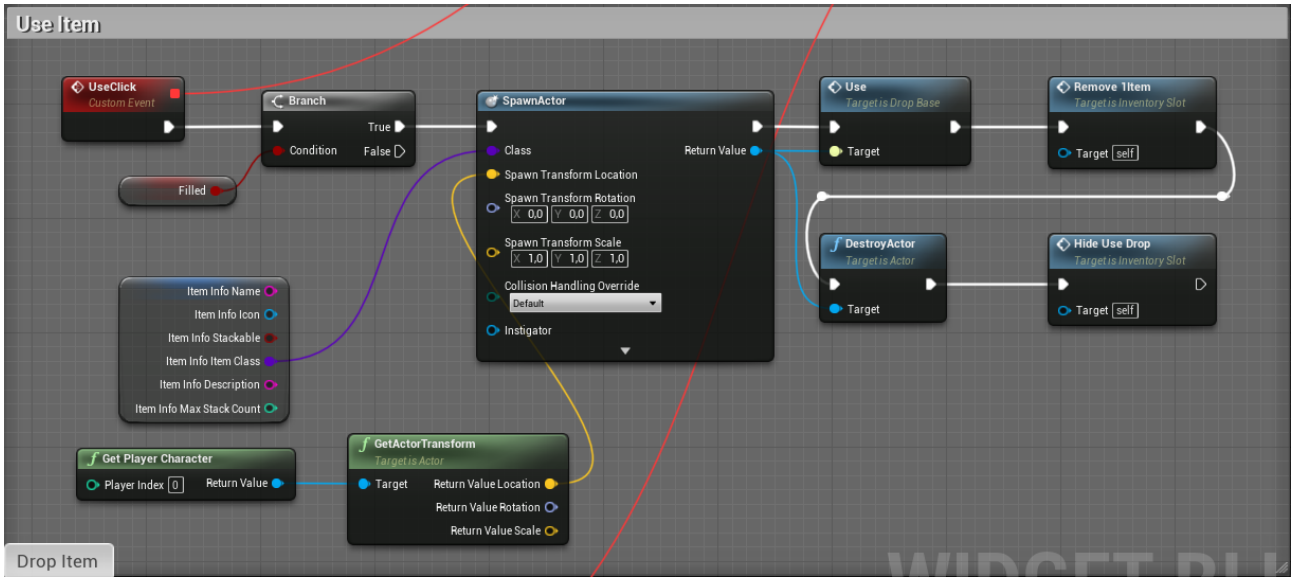


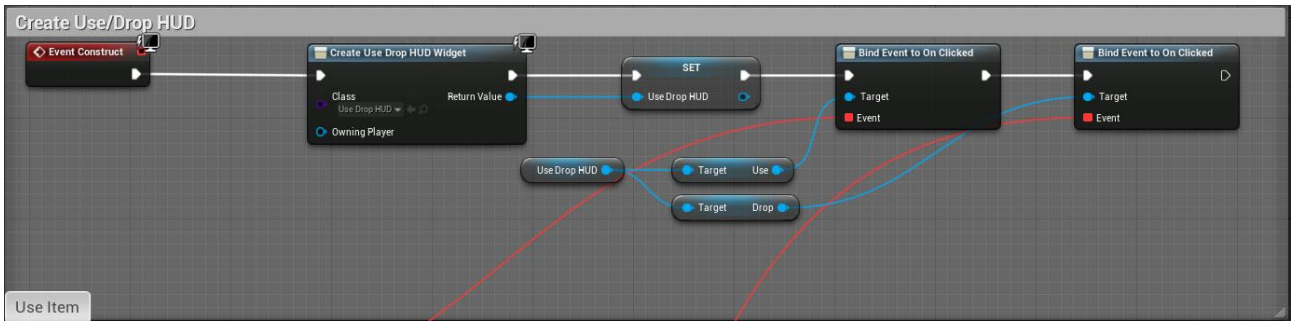


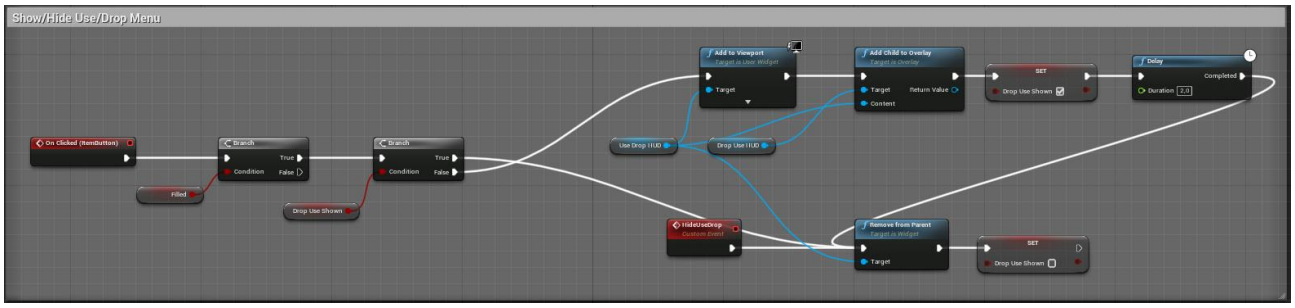

item removal sequence
item removal sequence


Set Item Img - Quantity - Set Slot To Filled


Drop Item
DropClicked
Custom Event
Branch
Condition
True
False
Filled
SpawnActor
Class
Return Value
Spawn Transform Location
Spawn Transform Rotation
Spawn Transform Scale
Collision Handling Override
Default
Instigator
Remove 1Item
Target is Inventory Slot
Target
self
Hide Use Drop
Target is Inventory Slot
Item Info Name
Item Info Icon
Item Info Stackable
Item Info Item Class
Item Info Description
Item Info Max Stack Count
Get Player Character
Player Index
Return Value
GetActorTransform
Target is Actor
Return Value Location
Return Value Rotation
Return Value Scale


Use Item
UseClick
Custom Event
Branch
Condition
True
False
Filled
SpawnActor
Class
Return Value
Spawn Transform Location
Spawn Transform Rotation
Spawn Transform Scale
Collision Handling Override
Default
Instigator
Use
Target is Drop Base
Target
Remove 1Item
Target is Inventory Slot
Target
self
DestroyActor
Target is Actor
Target
Hide Use Drop
Target is Inventory Slot
Target
self
Item Info Name
Item Info Icon
Item Info Stackable
Item Info Item Class
Item Info Description
Item Info Max Stack Count
Get Player Character
Player Index
Return Value
GetActorTransform
Target is Actor
Return Value Location
Return Value Rotation
Return Value Scale
Drop Item


Create Use/Drop HUD
Event Construct
Create Use Drop HUD Widget
Class
Use Drop HUD
Return Value
Owning Player
SET
Use Drop HUD
Bind Event to On Clicked
Target
Event
Bind Event to On Clicked
Target
Event
Use Drop HUD
Target
Use
Target
Drop
Use Item


Show/Hide Use/Drop Menu
On Clicked (ItemButton)
Branch
Condition
True
False
Filled
Branch
Condition
True
False
Drop Use Shown
Add to Viewport
Target
Add Child to Overlay
Target
Content
Return Value
SET
Drop Use Shown
Delay
Completed
Duration
Use Drop HUD
Drop Use HUD
HideUseDrop
Remove from Parent
Target
SET
Drop Use Shown