

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

**«БАГАТОКОРИСТУВАЛЬНИЦЬКА ОНЛАЙН-ГРА
В ЖАНРІ “THIRD PERSON SHOOTER”»**

Рівень “бакалавр”

Галузь знань 12 “Інформаційні технології”

Спеціальність 122 “Комп’ютерні науки”

Виконав здобувач 4 курсу

Курчук Роман Олександрович

Керівник:

к.т.н., доцент

Задерейко Олександр Владиславович

Рецензент:

к.т.н., доцент

Трофименко Олена Григорівна

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ”

Факультет кібербезпеки та інформаційних технологій

Кафедра інформаційних технологій

Галузь знань 12 Інформаційні технології

Спеціальність 122 “Комп’ютерні науки”

Назва освітньої програми “Комп’ютерні науки”

ЗАТВЕРДЖУЮ

Завідувач кафедри інформаційних технологій

доцент Наталія Логінова _____

“ ____ ” _____ 2024 року

ЗАВДАННЯ

на кваліфікаційну (бакалаврську) роботу
здобувачу Курчуку Роману Олександровичу

1. Тема роботи: “Багатокористувальницька онлайн-гра в жанрі «third person shooter»”.

Керівник роботи: к.т.н, доцент Задерейко Олександр Владиславович
затверджені наказом НУ «ОЮА» від «02» квітня 2024 року № 809-06

2. Строк подання здобувачем роботи «20» травня 2024 року

3. Дата видачі завдання «15» вересня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз вимог до ігрового проєкту	02.10.2023	
2	Проектування програмного продукту	03.11.2023	
3	Реалізація базового функціоналу	04.12.2023	
4	Створення меню налаштувань	20.01.2024	
5	Реалізація мультиплеєрної частини	14.02.2024	
6	Вибір персонажа	18.03.2024	
7	Штучний інтелект ботів	20.04.2024	
8	Реалізація режимів гри та додаткових функцій	03.05.2024	
9	Оформлення звітної документації	17.05.2024	

Здобувач

Керівник роботи

АНОТАЦІЯ

Курчук Р. О. Багатокористувальницька онлайн-гра в жанрі «third person shooter». – Кваліфікаційна робота бакалавра за спеціальністю 122 “Комп’ютерні науки”, освітньою програмою “Комп’ютерні науки”.

Кваліфікаційна робота викладена на 76 сторінках основного тексту та 83 сторінках загального тексту, містить 3 розділи, 54 рисунки, 11 таблиць, 2 додатки, 11 використаних джерел.

Метою роботи є розробка багатокористувальницького шутера від третьої особи з можливістю проведення ігрової сесії шляхом підключення через локальну мережу засобами ігрового рушія Unreal Engine версії 5.3.

Об’єктом дослідження є багатокористувальницькі ігри.

Предмет дослідження – засоби розробки проєкту багатокористувальницької гри засобами рушія Unreal Engine.

У кваліфікаційній роботі розроблено проєкт багатокористувальницького шутера від третьої особи з базовим для жанру функціоналом, можливістю створення ігрових сесій, підключення до вже наявних і вибору типу сесії та режиму гри; досліджено специфіку розробки багатокористувальницьких ігор шляхом використання різних засобів розробки рушія Unreal Engine.

Ключові слова: рушій, гра, персонаж, клас, метод, функція, Blueprint.

ANNOTATION

Kurchuk R. O. Multiplayer online game in the genre of "third person shooter".

The qualification work is laid out on 76 pages of the main text and 83 pages of the general text, contains 3 chapters, 54 pictures, 11 tables, 2 appendixes, 11 used sources.

The goal of the work is the development of a third-person multiplayer shooter with the possibility of holding a game session by connecting through a local network using the Unreal Engine version 5.3 game engine.

The object of the study is the multiplayer games.

The subject of research is the means of developing a multiplayer game project using the Unreal Engine.

In the qualifying work, a third-person multiplayer shooter project was developed with basic functionality for the genre, the ability to create game sessions, connect to already existing ones, and select the type of session and game mode; the specifics of developing multiplayer games by using various Unreal Engine development tools have been studied.

Keywords: engine, game, character, class, method, function, Blueprint.

ЗМІСТ

ВСТУП.....	6
1 АНАЛІЗ ВИМОГ ДО ІГРОВОГО ПРОЄКТУ	7
1.1 Специфіка розробки проєкту	7
1.2 Аналіз наявних програмних аналогів	7
1.3 Розробка функціональних вимог проєкту	11
1.4 Вибір засобів розробки.....	13
1.5 Маркетингові інструменти створення та просування.....	15
1.6 Висновки до розділу 1	16
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ	18
2.1 Архітектура проєкту	18
2.2 Моделювання програмної системи	19
2.3 Висновки до розділу 2	23
3 РЕАЛІЗАЦІЯ ПРОЄКТУ	24
3.1 Реалізація базового функціоналу	24
3.2 Створення меню налаштувань гри.....	32
3.3 Реалізація мультиплеєрної частини	36
3.4 Вибір зброї та ігровий інтерфейс	44
3.5 Вибір персонажа	48
3.6 Штучний інтелект ботів	56
3.7 Налаштування параметрів переродження персонажа.....	59
3.8 Реалізація режимів гри та додаткових функцій.....	60
3.9 Висновки до розділу 3	72
ВИСНОВКИ.....	74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	76
ДОДАТКИ.....	77
Додаток А. Приклади Blueprint-функцій проєкту	77
Додаток Б. Лістинг програмного коду вихідного файлу та файлу заголовку класу Character	80

ВСТУП

З кожним днем індустрія розробки ігор розвивається, з'являються нові інструменти, які дозволяють максимально наблизити ігровий проєкт до реальності. Покращується графіка, візуальні ефекти, фізику предметів та тіл, міміку персонажів тощо. Розробка ігор – складний, але захоплюючий процес, який дозволяє розробникам розкрити свою творчість та потребує високих технічних навичок, а також вміння ефективно працювати в команді.

Кваліфікаційна робота бакалавра присвячена розробці мультиплеєрного ігрового проєкту в жанрі “шутер від третьої особи”.

Об’єкт дослідження – багатокористувальницькі ігри.

Предмет дослідження – засоби розробки проєкту багатокористувальницької гри засобами рушія Unreal Engine.

Метою роботи є розробка багатокористувальницького шутера від третьої особи з можливістю проведення ігрової сесії шляхом підключення через локальну мережу засобами ігрового рушія Unreal Engine версії 5.3.

Відповідно до мети завданнями дипломної роботи є:

1. реалізація основних функцій, притаманних жанру шутера: рух, стрільба, прицілювання, можливість вбивати суперників;
2. реалізація функціоналу, необхідного для підтримки мультиплеєрного режиму гри;
3. створення зрозумілого та дружнього для користувачів інтерфейсу;
4. створення дизайну персонажів та ігрових рівнів (наприклад, моделі та скіни персонажів);
5. створення базового штучного інтелекту для керування ботами;
6. реалізація можливості налаштування графічних налаштувань гри.

Методика дослідження – аналітична з використанням комп’ютерних технологій.

Практична значущість результатів дослідження: створення підґрунтя для реалізації повноцінної гри в жанрі “third person shooter”.

1 АНАЛІЗ ВИМОГ ДО ІГРОВОГО ПРОЄКТУ

1.1 Специфіка розробки проєкту

Під час аналізу вимог до проєкту описується поведінка розроблюваної системи. Специфікація вимог проєкту містить основні складові розроблюваного проєкту, детальний опис вимог до функціоналу певних частин та проєкту в цілому, а також оформлення якісного дизайну проєкту.

Розробка мультиплеєрного шутера є складним процесом, який поєднує декілька етапів:

- геймдизайн: на даному етапі необхідно визначити геймплейні механіки проєктованої гри (наприклад, рух та стрільба), продумати баланс між предметами, зброєю тощо для забезпечення чесної гри та зпроєктувати ігрові рівні;
- програмування (реалізація): реалізуються визначені механіки, розробляється серверна частина для підтримки мультиплеєрного режиму, створюється штучний інтелект тощо;
- візуальний етап (дизайн): розробляються та реалізуються моделі персонажів, предметів, ландшафт, архітектура карт, анімації персонажів та оточення, візуальні ефекти;
- звуковий етап: створюється звукова частина гри: звуки оточення, озвучка персонажів, саундтрек.

1.2 Аналіз наявних програмних аналогів

Шутери від третьої особи є менш популярними, ніж шутери від першої особи, відповідно конкуренція в даному жанрі також є меншою. В більшості шутерів від третьої особи є свої унікальні особливості, які виділяють їх на фоні конкурентів. Тож для визначення конкурентоспроможності та доцільності створення проєкту потрібно порівняти його з наявними прикладами, визначивши переваги та недоліки в порівнянні з конкурентами. Для цього необхідно провести SWOT-аналіз проєктів-конкурентів.

Створюваний проєкт порівнюватиметься з чотирма найпопулярнішими представниками шутерів від третьої особи: Fortnite, Gears 5, Mass Effect 3, PlayerUnknown's Battlegrounds (PUBG) та Tom Clancy's The Division 2.

Таблиця 1.1 – SWOT-аналіз гри Fortnite

Сильні сторони	Слабкі сторони	Можливості	Ризики
Популярність: дана гра є однією з найпопулярніших в світі	Монотонність: геймплей не змінюється та не розширюється з виходом оновлень, що часто призводить до втрати цікавості гравців	Розширення ком'юніті: подальший розвиток та регулярна реклама сприяє збільшенню кількості гравців	Конкуренція: інші ігри жанру можуть зменшити популярність Fortnite
Безкоштовний доступ: гра поширюється безкоштовно		Створення нових режимів гри: введення в гру нових режимів сприяє збільшенню зацікавленості гравців та зменшенню монотонності геймплею	Баланс гри: неправильний баланс вмінь чи предметів може призвести до порушення балансу гри
Кросплатформенність: гра функціонує на багатьох популярних платформах (Windows, Xbox, Playstation, Android та інші)		Колаборації: колаборація з іншими популярними іграми чи художніми творами сприяє залученню зацікавлених гравців	Юридичні проблеми: проблеми з авторськими правами на танці чи вміння можуть привести до проблем та зниження популярності
Регулярні оновлення: в гру постійно додається новий контент (вміння персонажів, скіни тощо)			Технічні проблеми: баги, глітчї та фрізи можуть негативно вплинути на геймплей

Таблиця 1.2 – SWOT-аналіз гри Gears 5

Сильні сторони	Слабкі сторони	Можливості	Ризики
Графіка та атмосфера: гра має вражаючу графіку та військову атмосферу	Незбалансованість карт: деякі карти погано придатні для певних режимів	Портування: портування на інші платформи збільшить кількість гравців	Конкуренція: інші подібні ігри можуть зменшити популярність гри
Багатовимірний геймплей: гра має багато різних режимів, як одиночних, так і мультиплеєрних, а також кооперативні	Часткова кросплатформенність: гра доступна лише на двох платформах, тому недоступна для користувачів таких платформ, як Playstation та Nintendo Switch	Рекламні кампанії та події: регулярна реклама та введення особливих подій збільшить популярність гри	Баланс гри: неправильно збалансовані елементи гри можуть вплинути на сприйняття її гравцями
Кросплатформенність: гра функціонує на платформах Windows та Xbox		Оновлення: створення нових карт чи режимів гри може сприяти збільшенню кількості гравців	Технічні проблеми: баги, глітчї та фрізи можуть негативно вплинути на геймплей:

Таблиця 1.3 – SWOT-аналіз гри Mass Effect 3

Сильні сторони	Слабкі сторони	Можливості	Ризики
Наратив та сюжет: гра має глибокий наратив та захоплюючий сюжет	Спірність фіналу: кінець гри став дуже суперечливим, що стало предметом обговорень та критики	Випуск контенту: оновлення гри може привернути увагу нових гравців	Конкуренція: ігри подібного жанру та з подібним сетингом можуть конкурувати за увагу гравців
Інноваційність геймплею: гра відзначилась введенням великої кількості нових механік (наприклад, бойові підрозділи).	Вплив DLC: багато ключових аспектів сюжету було реалізовано у виді платних DLC, що призводить до неповноти сюжету для гравців, які не придбали доповнення	Розвиток спільноти: підтримка та розвиток спільноти гравців може збільшити взаємодію гравців з розробниками та привернути увагу до даної гри	Недоліки гри: спірний фінал, проблеми зі стабільністю та необхідність придбання DLC для повного розуміння історії можуть вплинути на рішення гравців щодо цієї гри
Свобода вибору: гра має багато варіантів розвитку сюжету в залежності від обраних гравцем варіантів діалогів	Стабільність: деякі версії гри мають велику кількість багів		Зміна тенденції: падіння популярності ігор в космічному сетингу може вплинути на продажі та популярність гри
Ігровий світ: гра має масштабний ігровий світ, наповнений різними активностями			

Таблиця 1.4 – SWOT-аналіз гри PUBG

Сильні сторони	Слабкі сторони	Можливості	Загрози
Популярність: гра є одною з найпопулярніших боєвих ігор в світі	Оптимізація: на деяких платформах часто присутні проблеми, такі як баги чи фрізи	Кіберспорт: PUBG має всі шанси розвинути до дисципліни кіберспорту, що гарантує нові спонсорські пропозиції	Копіювання: конкуренти можуть спробувати скопіювати основу PUBG, що призведе до розподілу аудиторії
Геймплей: гра має динамічний геймплей, а режим “Королівська битва” дає можливість гравцям проявити себе в стратегії і тактиці	Монетизація: велика частка ком’юніті незадоволена моделлю монетизації гри	Розширення аудиторії: розширюючи доступність гри та створюючи новий контент можливо залучити більшу кількість гравців	Зміни в галузі: зміна технологій та смаків гравців може вплинути на популярність гри
Мультиплатформенність: гра доступна на всіх популярних платформах			

Таблиця 1.5 – SWOT-аналіз гри Tom Clancy's The Division 2

Сильні сторони	Слабкі сторони	Можливості	Загрози
Місії та розвиток персонажа: гра містить багато різноманітних місій та завдань, після проходження яких покращуються екіпірування та навички персонажа	Монотонність: деякі місії можуть здатися гравцям монотонними через повторення локацій та дій	Розширення світу: розробники можуть розширити ігровий світ, зробивши геймплей динамічнішим	Конкуренція: підвищення популярності інших ігор може вплинути на популярність гри
Геймплей: гра має динамічний та захоплюючий геймплей, а також розвинутий кооператив	Брак розвитку історії: хоча гра має цікавий світ та атмосферу, в деяких моментах присутній брак історії гри та персонажів	Підтримка спільноти: залучення гравців до процесу розробки може допомогти в підтримці гри	Баланс: введення в гру незбалансованих предметів чи навичок персонажа може вплинути на ігровий процес
Змістовні оновлення: розробники регулярно випускають нові доповнення та оновлення			
Графіка: гра має гарну та реалістичну графіку			

Всі розглянуті конкуренти мають свої сильні та слабкі сторони, в тому числі подібні. Виділимо основні плюси та мінуси всіх конкурентів:

- Fortnite доступний безкоштовно і є динамічним шутером, але має доволі монотонний ігровий процес;
- Gears 5 є дуже атмосферним шутером з великою кількістю різних режимів гри, але доступний лише на Windows та Xbox і має проблеми з балансом деяких карт у певних режимах;
- Mass Effect 3 виділяється глибоким і розвинутим сюжетом та був іноваційним під час виходу, але в мультиплеєрному режимі має лише кооперативний режим, а також деякі проблеми зі стабільністю певних версій; PUBG в даний момент дуже популярний і має захоплюючий та динамічний геймплей в режимі “Королівська битва”, але, водночас, не має інших режимів гри і з плином часу збільшує фокус на мікротранзакціях та інших внутрішньоігрових покупках;

– Tom Clancy's The Division 2 є гідним продовженням першої частини, з реалістичною графікою, динамічним геймплеєм та різноманітністю місій, але деякі місії можуть бути монотонними та неповними в плані історії.

Аналіз наявних конкурентів дозволив дійти висновку, що створення мультиплеєрного шутера від третьої особи є вельми актуальним. Проте для забезпечення його конкурентоспроможності під час розробки треба забезпечити:

- різноманітність геймплею: гра не має бути монотонною та залежати від тенденцій ігрового ринку. Це забезпечується введенням нових різноманітних режимів, персонажів, зброї тощо;
- динамічність ігрового процесу: забезпечення швидких, захоплюючих битв гарантує привернення уваги гравців до продукту;
- створення та розвиток спільноти гравців: постійна взаємодія з ком'юніті гри та залучення гравців до процесів розробки та підтримки гри забезпечить позитивний відгук геймерів та привернення уваги більшої кількості гравців;
- кросплатформність: спільнота гри буде набагато більшою, якщо кінцевий продукт буде доступний на більшості популярних платформ;
- врахування недоліків конкурентів: SWOT-аналіз конкурентів може допомогти змодельовати гру, яка враховуватиме сильні сторони та уникатиме слабких, в результаті чого матиме всі шанси стати популярнішою.

1.3 Розробка функціональних вимог проєкту

Для успішної реалізації та подальшої підтримки необхідно визначити точні вимоги до функціоналу та оформлення проєкту, що дозволить уникнути можливих проблем під час реалізації. В табл. 1.6 визначено вимоги до функціоналу проєкту.

В табл. 1.7 визначено вимоги до дизайну проєкту.

Таблиця 1.6 – Опис функціоналу проекту

Функція	Опис
Передігрова взаємодія	
Вибір профіля	Гравець має можливість створювати профілі з унікальними іменами та змінювати їх в будь-який момент перед грою
Створення сесії	Кожен гравець може створити ігрову сесію (одиначну для самотійної гри або мультиплеєрну для гри з іншими гравцями по локальній чи глобальній мережі)
Приєднання до сесії	Гравець має можливість знайти вже створені сесії та приєднатись до будь-якої з них (в тому числі під час гри)
Налаштування гри	Запущену гру можна налаштувати, встановивши потрібний рівень якості графічних показників (якість тіней, світла тощо), розширення та режим відображення вікна гри
Налаштування сесії	Гравець має можливість налаштувати будь-яку створену ним сесію, обравши мапу для гри, режим, кількість гравців (команд), час тривання раунду, затримку респауна, кількість очок, необхідних для перемоги тощо
Спілкування з іншими гравцями	Гравці мають можливість спілкуватись між собою в чаті лобі
Кастомізація персонажа	Гравець має можливість обрати модель персонажа, змінивши його характеристики, а також змінити його зовнішній вигляд за допомогою скінів
Ігрова взаємодія	
Переміщення персонажа	Гравець може переміщувати персонажа ігровим світом, використовуючи пристрій введення
Стрільба	Основний метод боротьби з супротивниками з використанням вогнепальної зброї, який здійснюється шляхом використання пристрою введення
Гранати	Допоміжний метод боротьби з супротивниками з використанням металевих снарядів
Ближній бій	Додатковий метод боротьби з супротивниками з використанням рукопашної атаки, яка здійснюється шляхом використання пристрою введення
Вибір зброї	Гравець може під час гри змінювати зброю шляхом використання назначеної клавіші або колеса зброї
Підбір предметів	Гравець може підбирати деякі предмети, розташовані в ігровому світі та використовувати його в ігровому процесі для поповнення інвентаря, здоров'я тощо

Таблиця 1.7 – Опис вимог до дизайну проекту

Вимога	Опис
Дизайн меню	Необхідно створити яскравий дизайн головного меню, меню налаштувань та меню мультиплеєра: фонове зображення, дизайн кнопок та слайдерів тощо
Дизайн лобі	Лобі має мати цікавий та зрозумілий дизайн, містити доступ до всіх налаштувань ігрової сесії, елемент з відображенням приєднаних гравців (наприклад, рамку з іменами або моделі гравців в самому лобі); також в лобі має бути чат для спілкування, розташований так, щоб не перекривати інші елементи та не заважати доступу до них
Дизайн мап	Кожна мапа повинна мати унікальний та пропрацьований дизайн, який відповідає виду та опису
Дизайн персонажів	Персонажі повинні мати дизайн, який відрізнятиметься в залежності від моделі чи скіна та повинен відповідати опису та/чи характеристиці
Дизайн предметів	Дизайн предметів повинен бути логічним та відповідати призначенню предмета (наприклад: предмети для відновлення здоров'я можуть мати вигляд серця, зброя має бути стилізована подібно реальним прототипам (за деякими виключеннями), дизайн боєприпасів має відповідати зброї, до якої вони підходять)

1.4 Вибір засобів розробки

Для створення ігрових проєктів використовуються ігрові рушії – центральна частина будь-якої відеогри, яка відповідає за всю технічну частину; він дозволяє полегшити розробку гри шляхом систематизації внутрішньої структури [1].

Загалом є два шляхи для реалізації свого ігрового проєкту:

- створення власного ігрового рушія, який використовуватиметься для реалізації власних проєктів;
- використання наявного ігрового рушія (можливо, з впровадженням модифікацій під власні потреби).

Для створюваного ігрового проєкту доцільніше використовувати наявний ігровий рушій, оскільки це забезпечить більшу стабільність реалізації, забезпечить великою кількістю допоміжних матеріалів та позбавить розробника від проблем створення нового рушія.

Є багато різних ігрових рушіїв як спеціалізованих для конкретних проєктів, так і для широкого використання. Для створення мультиплеєрного шутера від третьої особи доцільно вибрати один з трьох рушіїв: Unreal Engine, Unity та CryEngine. Щоб вибрати найбільш вдалий варіант треба провести аналіз сильних та слабких сторін кожного з них.

Unreal Engine (<https://www.unrealengine.com/>) – рушій, створений та підтримуваний компанією Epic Games. Він використовує мову програмування C++ та підтримує більшість наявних платформ [2].

Таблиця 1.8 – Переваги та недоліки рушія Unreal Engine

Переваги	Недоліки
Потужність: на даний момент Unreal є найбільш потужним та розвинутим ігровим рушієм, який виділяється високим рівнем реалістичності	Висока ресурсозатратність: створення графічно складних ігор вимагає значних ресурсів комп'ютера, тому можливі проблеми при використанні менш потужних систем
Багатофункціональність: рушій має широкий функціонал, в тому числі вбудовані інструменти для розробки ігор, візуалізації, анімації, що цілком дозволяє створювати власні проєкти без використання сторонніх інструментів	

Переваги	Недоліки
Спільнота та підтримка: рушій має велике ком'юніті, яке постійно підтримується великою кількістю розробників	Складність вивчення: порівняно з іншими рушіями, Unreal має вищий поріг вступу і може бути складним для освоєння новачками
Документація: Unreal Engine має широку, детальну та зрозумілу документацію, велику кількість курсів та гайдів	
Blueprint: рушій містить систему візуального блокового програмування Blueprint, що дозволить знизити складність та прискорити створення ігор	

Unity (<https://unity.com/>) – найпопулярніший рушій для створенні ігор, який використовує C# та підтримує багато різних платформ.

Таблиця 1.9 – Переваги та недоліки рушія Unity

Переваги	Недоліки
Легкість вивчення: Unity вважається більш легким та доступним для вивчення в порівнянні з Unreal та CryEngine	Обмеженість: рушій може гарантувати хорошу графіку тільки при доробці, але все одно не здатний конкурувати з Unreal
Активна спільнота: Unity має велику спільноту користувачів, які забезпечують рушій великою базою підтримки	Обмеженість безкоштовної версії: багато функцій недоступні в безкоштовній версії, що значно знижує можливості для розробників з обмеженим бюджетом
Ресурси: для Unity створено багато різних курсів, документації та матеріалів, які використовуються для створення власних продуктів	Зрозумілість матеріалів: більшість матеріалів для вивчення рушія недостатньо детальні та доступні, що може створити проблеми для недосвідчених розробників
	Технічні проблеми: Unity має багато проблем та багів, які можуть значно вплинути на процес розробки та якість кінцевого продукту
	Сумісність версій: проекти, створені на одній версії рушія, в більшості випадків несумісні з іншими версіями, що може вплинути на процес розробки

CryEngine (<https://www.cryengine.com/>) – ігровий рушій, створений компанією Crytek, який використовує як C++, так і C# для розробки.

Таблиця 1.10 – Переваги та недоліки CryEngine

Переваги	Недоліки
Графічна потужність: CryEngine також є потужним та графічно розвинутим рушієм	Складність використання: як і Unreal, CryEngine може бути складним у використанні та для вивченні недосвідченими розробниками
Рівень реалізму: рушій відомий здатністю до створення реалістичного середовища та ефектів	Обмежена підтримка: в порівнянні з Unreal та Unity, рушій має меншу спільноту та обмежену базу підтримки та матеріалів

CryEngine є кросплатформним, хоча список підтримуваних платформ значно менше, ніж в Unity та Unreal.

Аналіз результатів порівняння ігрових рушіїв дозволив визначитися з кращим варіантом для реалізації поставленої задачі. Це рушій Unreal Engine, позаяк він є найпотужнішим і технічно найрозвинутішим, що дозволить створити гру з хорошою реалістичною графікою. Також цей рушій має розширений функціонал, тому його можливо використовувати без доповнень та іншого додаткового контенту.

Unreal Engine надає два основних інструменти для створення ігрових проєктів – C++ класи, створені користувачем за допомогою коду, та система блокового програмування Blueprint, яка дозволяє будувати функціонал без написання коду. В кожного з цих інструментів є свої переваги та недоліки, тому найкращим методом реалізації проєкту доцільно вважати комбінування елементів C++ та Blueprint.

1.5 Маркетингові інструменти створення та просування

Створення маркетингової стратегії для просування та реалізації продукту варто почати зі створення карти клієнта. Карта клієнта містить таку інформацію:

- 1) демографічна інформація:
 - a) вік: 15-35 років;
 - b) стать: будь-яка;
 - c) географічне розташування: весь світ з фокусом на країни з великим ринком геймерів (США, Китай тощо);
- 2) поведінкові шаблони:
 - a) геймери, які цінують соціальну взаємодію та змагання;
 - b) геймери, зацікавлені в нових враженнях;
 - c) геймери, які віддають перевагу мультиплеєрним іграм;
 - d) геймери, яким важлива високоякісна графіка;
- 3) інтереси та потреби:
 - a) бажання випробувати нові рівні з різноманітним середовищем;

- b) потреба в регулярних оновленнях та введенні нового контенту;
- c) потреба в різних режимах гри.

Під час аналізу ринку було встановлено, що останнім часом збільшується попит на ігри від третьої особи, особливо шутери. Такі ігри, як Fortnite та PUBG зараз на піку популярності, але обидві гри доволі монотонні, тому гравці потребують нових ігор з новими геймплейними рішеннями. Ігри, які виділяються якісною графікою, мають геймплейні особливості та забезпечують взаємодію між гравцями, є затребуваними та цінуються на ігровому ринку.

Стратегії пошукової оптимізації та просування проєкту в соціальних мережах містять такі аспекти:

- 1) SEO:
 - a) використання ключових слів, пов'язаних з жанром проєкту, на сайті та ігровому контенті;
 - b) оптимізація заголовків та описів для привертання уваги геймерів, які шукають нові ігри.
- 2) SMM:
 - a) присутність та активний розвиток соціальних мереж (Facebook, YouTube, TikTok тощо);
 - b) розміщення ігрового контенту та оновлень на власному сайті, створення рекламних роликів, проведення трансляцій геймплею;
 - c) активна взаємодія зі спільнотою, регулярне проведення спеціальних подій для підтримання зацікавленості гравців.

1.6 Висновки до розділу 1

В цьому розділі було проведено аналіз як функціональних, так і дизайнерських вимог до проєкту.

При розгляді специфіки розробки ігрового проєкту встановлено основні етапи розробки багатокористувальницького шутера від третьої особи: геймдизайн, реалізація, дизайн і саундтрек.

Проведено SWOT-аналіз і розглянуто переваги та недоліки популярних представників обраного жанру: Fortnite, Gears 5, Mass Effect 3, PUBG та Tom Clancy's The Division 2. Аналіз наявних аналогів дозволив встановити, що створення мультиплеєрного ігрового проєкту в жанрі шутера від третьої особи є доцільним та затребуваним.

Визначено та розроблено такі функціональні вимоги до проєкту: передігрова взаємодія – вибір профіля, створення сесій та приєднання до вже створених, налаштування гри та сесій, спілкування з іншими гравцями та кастомізація персонажа; ігрова взаємодія – переміщення персонажа, атака (стрільба, гранати та ближній бій), підбір предметів, зміна зброї.

Під час вибору засобів розробки проаналізовано три найпоширеніших рушія для реалізації подібного проєкту, розглянуто та порівняно переваги та недоліки рушіїв, обрано Unreal Engine як основний інструмент для реалізації проєкту.

Створено маркетингову стратегію розвитку та поширення проєкту, яка складається з: карти клієнта, результатів аналізу ринку, стратегії оптимізації пошуку та поширення в соціальних мережах.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Архітектура проєкту

В табл. 2.1 визначено основні складові ігрового проєкту.

Таблиця 2.1. Основні складові розроблюваного проєкту

Термін	Опис
Головне меню	Сцена, яка створюється першою одразу після запуску проєкту; меню використовується для навігації та використання деяких функцій проєкту: налаштування гри, запуск сесії або вихід з гри
Меню налаштувань	Сцена, доступ до якої здійснюється з головного меню або під час гри через меню паузи; меню використовується для встановлення налаштувань гри: якість графіки, тіней, рівень звуку, розширення екрану тощо
Меню мультиплеєра	Сцена, доступ до якої здійснюється з головного меню; меню використовується для створення сесій та приєднання то вже створених, а також вибору типу підключення (локальна чи глобальна мережа)
Лобі	Сцена, яка слідує за меню і використовується для налаштування сесії, приєднання гравців, спілкування між гравцями в чаті та запуску сесії
Екран завантаження	Допоміжна сцена, яка з'являється між основними на час завантаження рівня
Ігровий рівень	Сцена, на якій відбувається основний ігровий процес; кожен ігровий рівень має мати унікальний дизайн
Персонаж	Ігрова модель, яка керується безпосередньо гравцем з певними особливостями та характеристиками, які впливають на ігровий процес
Бот	Ігрова модель з певними особливостями та характеристиками, яка керується штучним інтелектом. Боти використовуються найчастіше в якості противників
Зброя	Основний інструмент протистояння ворогам; кожен персонаж має дефолтну зброю та може підбирати нову під час гри

Головне меню містить такі елементи: кнопка для початку одиночної сесії, кнопка для переходу в меню мультиплеєра, кнопка переходу до меню налаштувань гри, кнопка виходу з гри, а також меню вибору профіля (усталено приховане) і кнопку для його відображення. Меню налаштувань містить вісім слайдерів для встановлення необхідного рівня якості графічних показників гри, один випадаючий список для вибору значення розширення екрану, три кнопки для вибору режиму відображення гри (віконний, повноекранний віконний чи повноекранний) і кнопку для повернення в головне меню. Меню мультиплеєра містить кнопки для створення та пошуку сесій, список знайдених сесій (усталено приховане; відображається після натискання кнопки пошуку), кнопку для зміни

типу підключення (локальний/глобальний), а також кнопку повернення в головне меню.

Ігрове лобі містить такі елементи:

- кнопку для переходу в меню кастомізації персонажа, яке, в свою чергу, містить кнопки вибору для всіх чотирьох персонажів та скінів для них, рамку з відображенням характеристик персонажа, кнопку для повернення в лобі;
- рамку з кнопками вибору ігрового рівня та інтро мапи;
- кнопки старту гри та повернення в меню;
- кнопки для вибору режиму гри;
- кнопка для переходу до меню налаштувань режиму, яке, в свою чергу, містить 4-5 випадні списки (залежно від режиму гри) для основних налаштувань режиму та кнопку повернення в лобі;
- чек-бокс (прапорець) для встановлення використання безшовного переходу;
- рамка з відображенням статистики гравця;
- чат;
- рамка з іменами приєднаних гравців.

2.2 Моделювання програмної системи

У роботі змодельовано варіанти взаємодії складових проєкту між собою, а також між проєктом та гравцем. На рис. 2.1 наведено діаграму прецедентів, яка описує повний запланований функціонал, доступний гравцям.

На рис. 2.2 подано ієрархію складових частин проєкту, яка відображає зв'язки між ними, тобто показує, з якої частини проєкту в яку гравець може потрапити.

Головне меню є першою сценою, доступною гравцеві. З головного меню гравець може перейти в лобі одиночної сесії для початку одиночної гри, меню мультиплеєра для початку мультиплеєрної сесії або приєднання до неї та меню налаштувань гри.

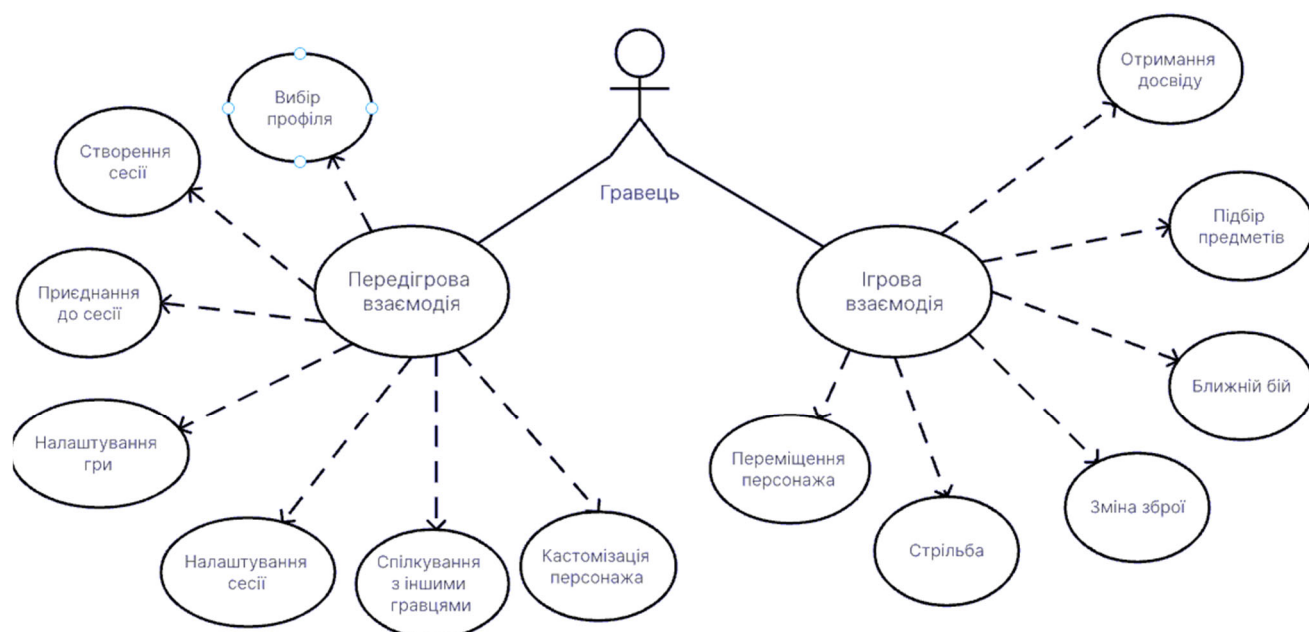


Рисунок 2.1 – Діаграма варіантів використання функціоналу гри

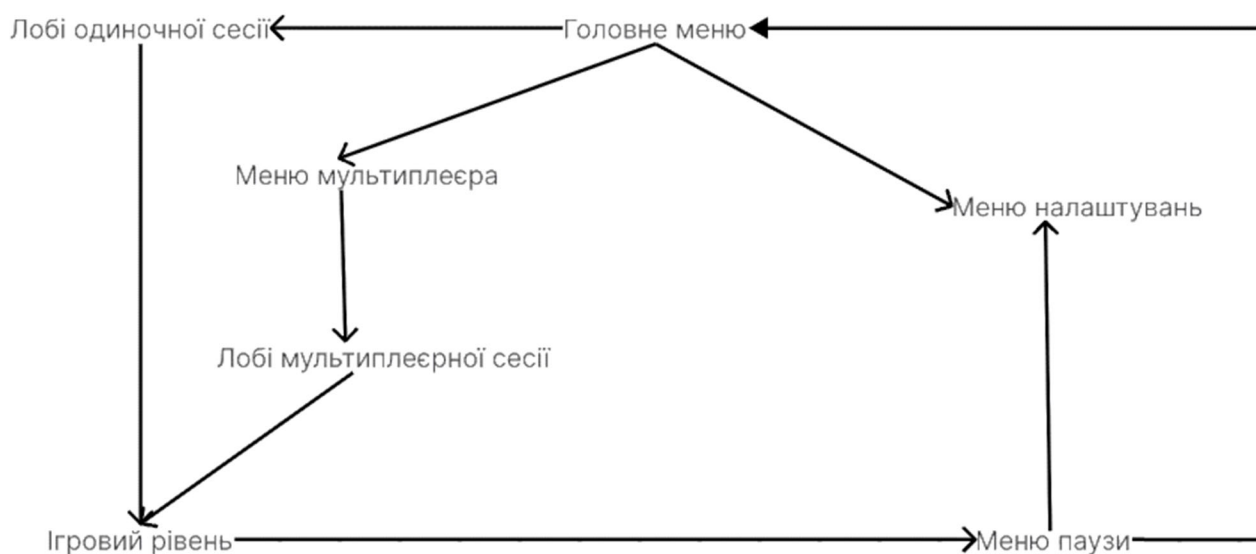


Рисунок 2.2 – Ієрархія складових проєкту

З меню мультиплеєра гравець може повернутись в головне меню або перейти в лобі сесії. З лобі одиночної та мультиплеєрної сесії гравець може повернутись в головне меню та меню мультиплеєра відповідно, або розпочати гру, перейшовши на ігровий рівень. З ігрового рівня гравець може потрапити в

меню паузи, з якого, в свою чергу, може потрапити в меню налаштувань гри або головне меню.

На рис. 2.3 зображено діаграму активностей для процесу пострілу.

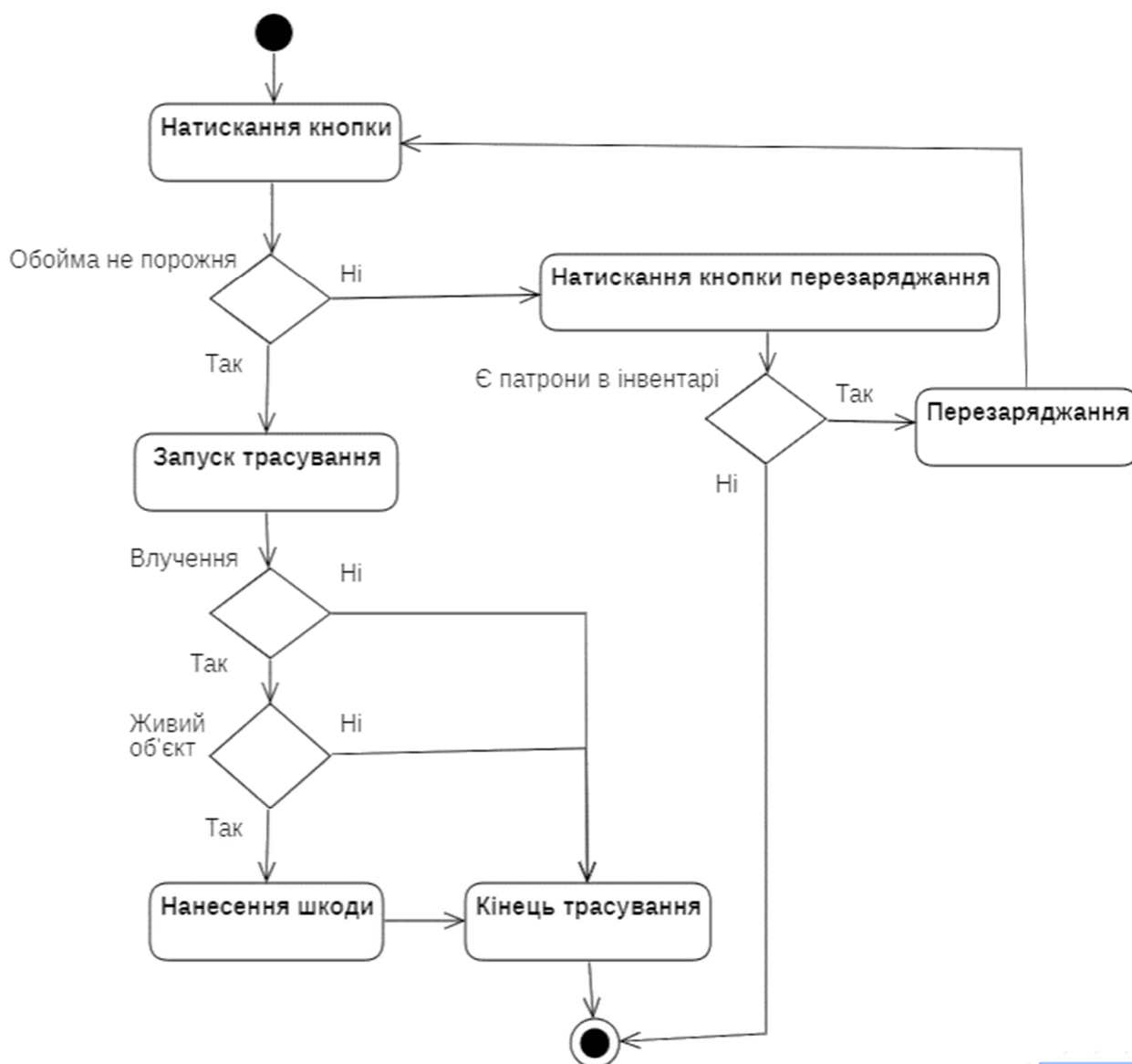


Рисунок 2.3 – Діаграма активностей для процесу пострілу

Процес починається з натискання кнопки гравцем, після чого перевіряється, чи є патрони в обоймі зброї. Якщо патронів нема, гравець повинен натиснути кнопку перезарядження, після чого зброя перезаряджається і необхідно знову натиснути кнопку стрільби (у разі відсутності патронів в інвентарі гравця процес закінчується, так як гравець не зможе стріляти). Якщо обойма не порожня, запускається трасування – відслідковування маршруту польоту снаряду. Якщо

снаряд влучає в живий об'єкт, йому наноситься шкода та закінчується трасування (в інших випадках відразу закінчується трасування).

На рис. 2.4 зображено діаграму активностей функціонування ботів.

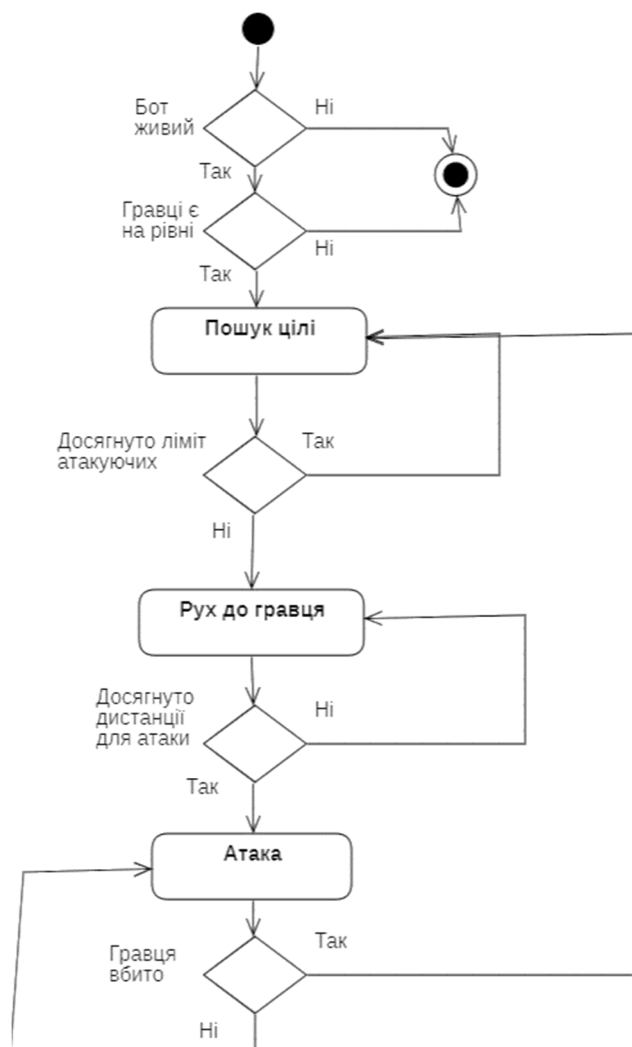


Рисунок 2.4 – Activity діаграма функціонування бота

Процес починається з перевірки статусу бота та наявності гравців на рівні. Якщо бот загинув або на рівні не залишилось гравців – процес закінчується. В іншому випадку починається пошук цілі, в якій не досягнуто ліміту атакуючих ботів. Якщо ціль знайдено – бот рухається до неї, доки не досягне необхідної дистанції для проведення атаки. Бот атакуватиме гравця, поки він не загине, після чого описаний цикл почнеться заново. Цикл пошуку цілі продовжуватиметься поки бот живий або на рівні є гравці.

2.3 Висновки до розділу 2

В даному розділі було спроектовано програмний застосунок. Визначено архітектуру проєкту, яка складається з таких елементів:

- головне меню;
- меню налаштувань гри;
- меню мультиплеєра;
- лобі (як для одиночної, так і для мультиплеєрної сесії);
- екран завантаження – відображається між лобі та ігровим рівнем;
- ігровий рівень.

Деякі елементи архітектури мають додаткові елементи для реалізації додаткових функцій або забезпечення зручності доступу до основних.

Змодельовано програмну систему проєкту: розглянуто основний та додатковий функціонал, доступний гравцям, складено ієрархію складових проєкту та методи доступу гравця до них. Розглянуто поведінку деяких основних функцій проєкту.

3 РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Реалізація базового функціоналу

Для початку реалізації проєкту було створено шаблон проєкту Blank типу Blueprint. Після створення проєкту створено стартову мапу, на якій відбуватиметься гра.

В створюваній грі використовувалися деякі ресурси з гри Lira – стартова гра Unreal Engine 5, яка доступна безкоштовно з Epic Games Launcher.

Після міграції необхідних ресурсів було додано входи (Inputs) для того, щоб при натисканні певної клавіші відбувалась певна дія (Project Settings → Input). Додано такі Axis Mappings: Move Forward (рух вперед – назад, клавіші W та S), Move Right (рух вправо – вліво, клавіші A та D), Turn (поворот вправо – вліво за допомогою руху миші) та LookUp (поворот вгору – вниз за допомогою руху миші).

Для створення персонажа було створено об'єкт класу Character. Даний об'єкт складається з таких компонентів: Capsule Component (капсульний компонент колізії), Arrow Component (стрілка, яка вказує напрямок повороту об'єкта) та Mesh (меш об'єкта). Спочатку додається меш, натиснувши на Mesh та обравши потрібний нам компонент в налаштуваннях об'єкта, після чого розвертається в напрямку стрілки та опускається в капсулу. Також в правому верхньому куті переназначено батьківський клас на створений власноручно клас C++ (лістинг подано в додатку Б).

В графі подій було створено графи подій для руху та обертання. Графи складаються з вузлів, які мають певний функціонал. Додавання вузлів відбувається за допомогою кліку правою кнопкою миші в середовищі графу та обираючи потрібного вузла. Тож:

- для подій Move Forward та Move Right додано вузол Add Movement Input (додає введення руху вздовж заданого вектора у світовому просторі), у вхід Scale Value якого передається значення Axis Value події. Для направлення руху потрібно передати вектор – для події Move Forward передається вектор за

допомогою вузла Get Actor Forward Vector (повертає вектор актора, направлений прямо), а для Move Right – за допомогою Get Actor Right Vector (повертає вектор актора, направлений вправо);

– для подій Turn та LookUp під'єднуються вузли Add Controller Yaw Input та Add Controller Pitch Input відповідно, у вхід Val вузлів передаються значення Axis Value відповідних подій.

Створені графи переміщення подано на рис. 3.1.

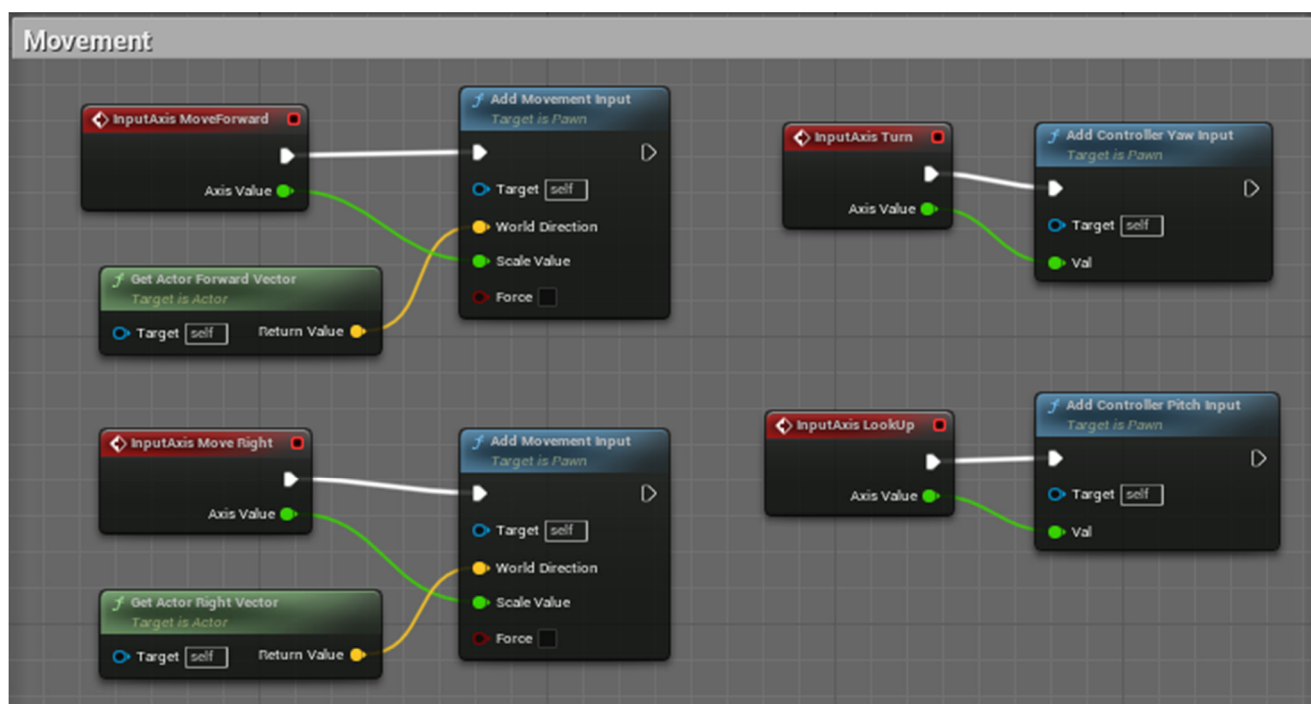


Рисунок 3.1 – Переміщення персонажа

Для персонажа необхідно додати камеру для відслідковування переміщення та інших подій. Для цього було додано компонент SpringArm, після чого додано саме камеру.

Після базового налаштування було створено анімацію руху. Для цього створено простір для роботи з анімаціями (Animation → Blend Space). Сет анімацій створюється шляхом комбінування різних анімацій з різними швидкостями програвання. Створений Blend Space подано на рис. 3.2.

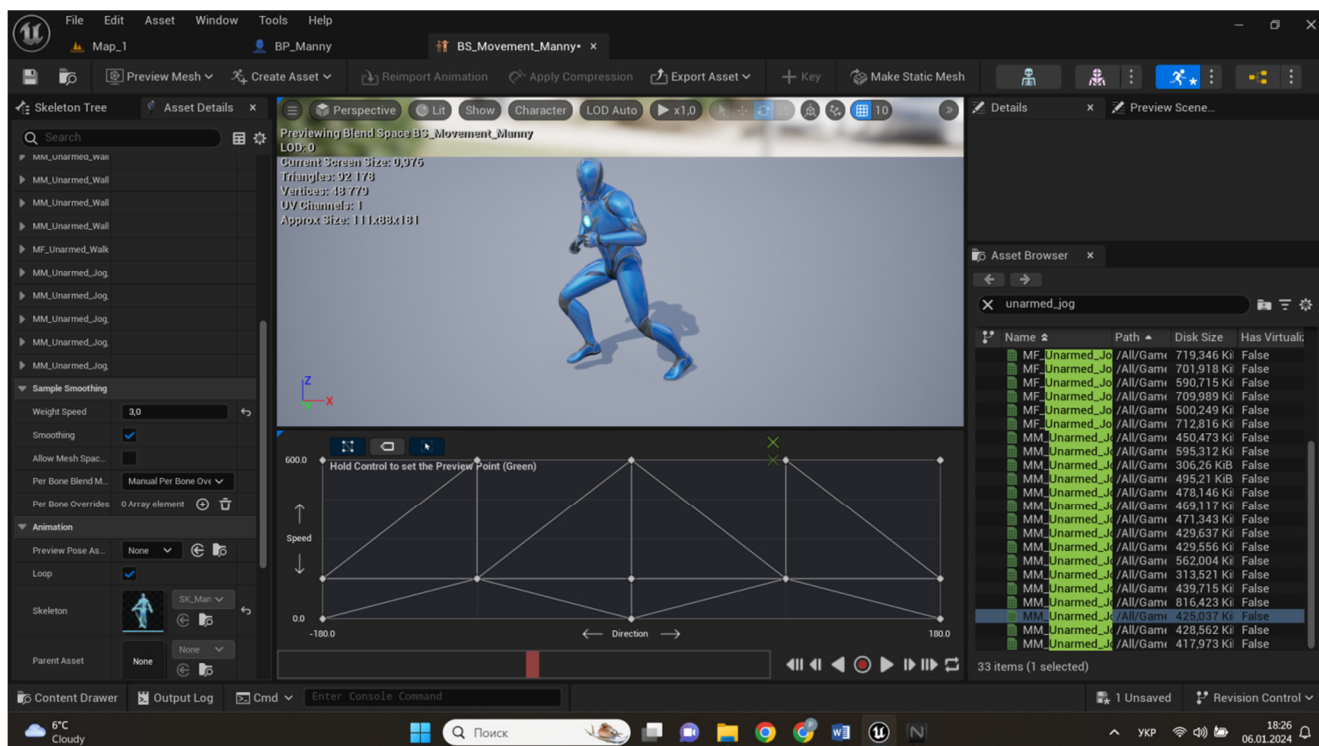


Рисунок 3.2 – Blend Space анімації руху

Для програвання анімацій під час гри необхідно створити Blueprint для анімації (Animation → Animation Blueprint). В середовище перетягується з браузера контенту створений Blend Space. Він має два входи – Direction (напрямок) та Speed (швидкість), які перетворюються на змінні. В графі подій додано подію Blueprint BeginPlay (активується, коли починає працювати даний Blueprint). Від вузла Try Get Pawn Owner взято каст [3] на Blueprint персонажа та поміщено його вихід As BP Manny в змінну Character. Для події Blueprint Update Animation (оновлення анімації) створено таку логіку: за допомогою вузла Is Valid перевіряється, чи доступна змінна Character, якщо так – встановлюються нові значення змінних Speed (зі змінної Character проведено вузли Get Velocity → Vector Length і присвоєно результат) та Direction (за допомогою вузла Get Base Aim Rotation отримано базовий поворот персонажа, передано його в Calculate Direction і присвоєно результат). Створений граф подано на рис. 3.3.

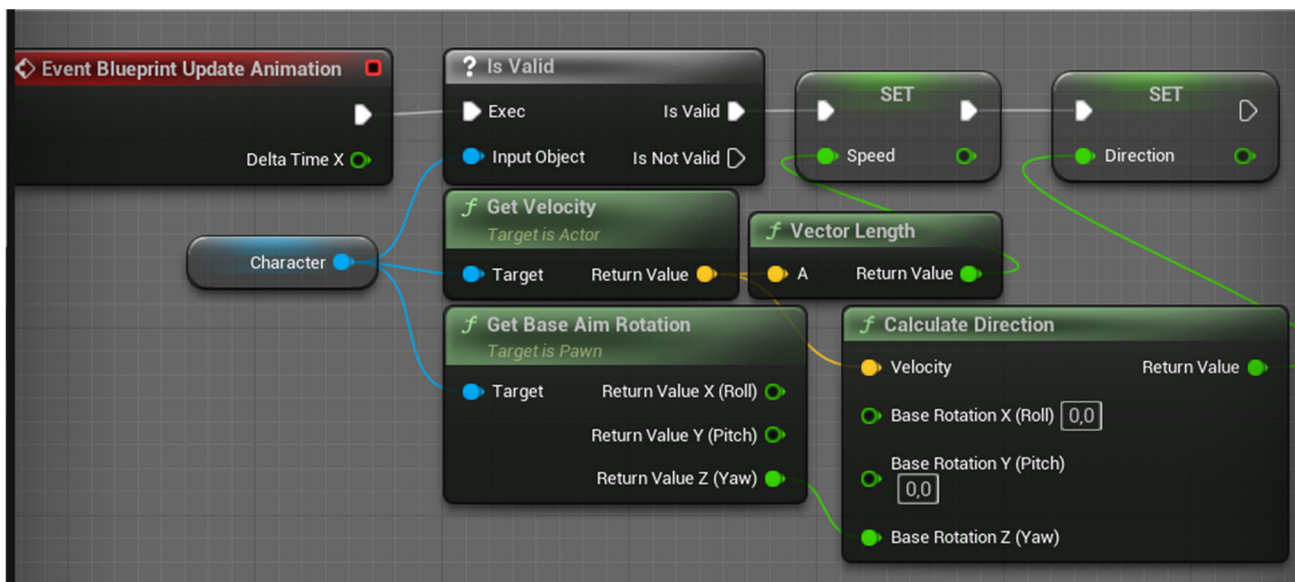


Рисунок 3.3 – Граф подій анімації руху

Повернувшись до Blueprint персонажа, в налаштуваннях меша обирається клас анімації (опція Anim Class), який щойно створили. Це необхідно для підключення створених анімацій до персонажа.

Для спавна персонажа та початку гри треба створити GameMode – компонент, який містить основні правила ігрового процесу [4]. В його налаштуваннях можна назначити класи ігрових об'єктів усталено. В Default Pawn Class назначено створеного персонажа, також очищен Default HUD, натиснувши кнопку Clear. Далі в налаштуваннях проєкту → меню Maps&Mods → опція Default GameMode оберемо MyGameMode. Тепер при запуску гри на сцені з'являтиметься персонаж, яким ми можемо керувати.

Далі було створено механіку прицілювання. В Project Settings → Input, створено новий Action Mapping та назначено на нього праву кнопку миші. В Blueprint персонажа створено нову змінну IsADS типу Boolean – використовуючи її відслідковуватиметься, чи прицілюється гравець.

В Blueprint анімації створено таку ж змінну. В графі подій взято змінну IsADS з Character та передано в Set IsADS. В графі анімацій від'єднано створену анімацію від Output Pose та збережено у вигляді нової пози за допомогою New Save cached pose. До графу додано сет анімацій прицілювання з пістолета, значення Pitch записано в змінну та додано базову позу. За допомогою вузла

Layered blend per bone створено змішування даної анімації з анімацією руху (отримано за допомогою Use cached pose “Locomotion”). Далі за допомогою Blend Poses by bool визначається, чи потрібно змішувати анімації. В Active Value (критерій визначення) назначено змінну IsADS, True Pose – змішування анімацій, False Pose – звичайна анімація руху. Дана логіка приєднана до Output Pose. Далі в налаштуваннях змішування в Layer Setup → Index → Branch Filters додано три компонента – кістки скелета, в яких будуть змішуватись анімації. Необхідно вказати назви кісток та глибину змішування: spine_01 – 4, thigh_l – -1 та thigh_r – -1.

Необхідно активувати реплікацію для передавання змін одного гравця в клієнт інших. Для цього в графі подій Blueprint персонажа в налаштуваннях вузла Set IsADS змінено значення опції Replication на Replicated. Тепер зміни клієнтів передаватимуться серверу. Але зміни на сервері не передаються в клієнт. Для цього було додано Switch Has Authority перед Set змінної. Даний вузол перевіряє, чи є даний гравець сервером. Якщо так (Authority), змінюється IsADS. Для клієнтів створено кастомну подію, яка змінює IsADS на сервері. Встановлено значення Replicated на Run On Server (запуск на сервері). Змінений граф подано на рис. 3.4.

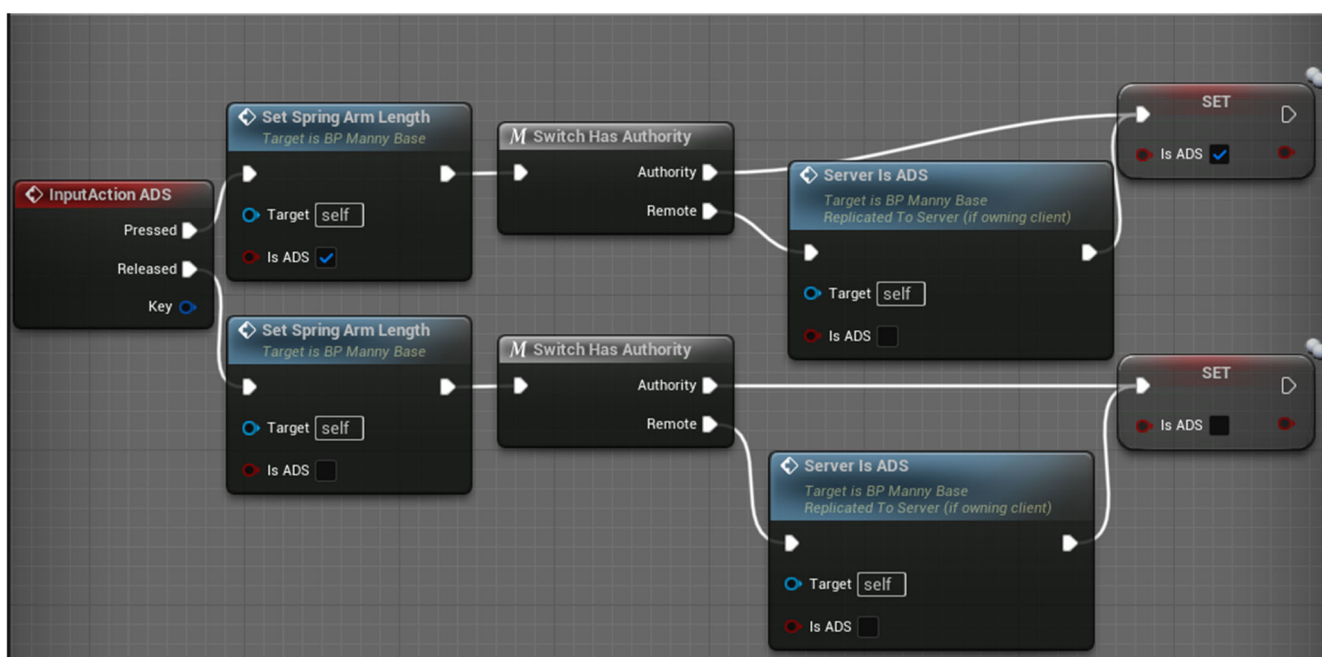


Рисунок 3.4 – Реплікація прицілювання

Далі було реалізовано зброю, меші для якої також взято з Lira. Для початку взято пістолет, гвинтівку та дробовик.

В Blueprint персонажа до наявного меша додано новий меш – зброя. Налаштовано меш пістолета – в меню Sockets → Parent Socket обрано weapon_r – пістолет прикріплюється до правої руки персонажа. Далі додано новий Action Mapping та назначено на ліву кнопку миші. Дана подія викликається в графі подій персонажа та під'єднується до вузла Spawn Emitter Attached, який створює об'єкт системи частинок у вказаному сегменті меша під час пострілу. Наступний вузол – Play 2D Sound, який програв звук під час пострілу (рис. 3.5).

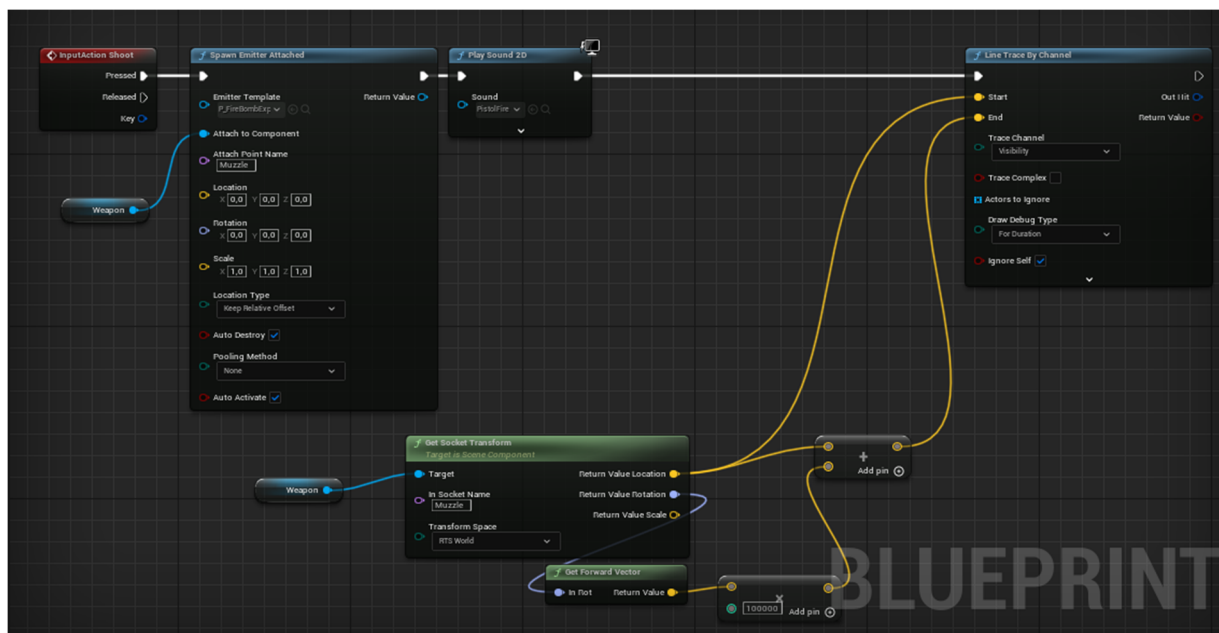


Рисунок 3.5 – Постріл з пістолета

Далі було створено GameInstance – менеджер високого рівня для екземпляра запущеної гри, який ми використаємо для зберігання даних гравця [5].

Для початку реалізації головного меню створено GameModeBase для меню, а також Character та Controller. В створеному GameMode інші створені класи назначено класами усталено. Також створено два нових рівня: порожній для меню та Basic для лобі, для обох в World Settings → GameMode назначено створений клас. Далі необхідно створити віджет меню – його створено в User Widget (рис. 3.6).



Рисунок 3.6 – Створений віджет меню

В `GameInstance` створено нову функцію `UI_ShowMainMenu` для відображення меню. За допомогою `Is Valid` перевіряється, чи існує віджет меню. Якщо так – він додається на екран, якщо ні – створюється, записується в змінну та додається на екран. Далі встановлюється `Set Input UI Only` для фокусування на інтерфейсі та відображається курсор миші.

В `Blueprint` рівня меню створено каст на створений `GameInstance` та викликається щойно створена функція відображення меню. Дану логіку приєднано до `BeginPlay`. Меню з'являється на екрані.

В `GameInstance` створено нову кастомну подію `Start SingleGame` та нову змінну `IsSingle` типу `Boolean`. Подія встановить змінній значення `True` та відкриє рівень `Map_1`.

В графі подій віджета меню викликаються події `OnClicked` кнопок `Single Player` та `Quit`. Спочатку на `Event Construct` береться каст на `GameInstance` та записується в змінну, після чого на натискання кнопки `Single Player` викликається функція `Start Single Game` з `GameInstance`. На натискання кнопки `Quit` викликається вузол `Quit Game`, який завершує гру. Перейшовши на `Map_1` та в `World Settings` назначено `MyGameMode` як `GameMode` усталено. Далі було створено новий контролер гравця та підключено до `GameMode`.

У створеному контролері при запуску перевіряється, чи є даний контролер локальним. Якщо так, встановлюється режим введення тільки для гри та приховується курсор миші.

Для реалізації логіки зберігання даних гравця було створено нову структуру з назвою PlayerInfo. В структурі додано нову змінну PlayerName типу Text, після чого створено об'єкт SaveGame з назвою SG_PlayerInfo. В ньому створено нову змінну з іменем та типом створеної структури.

Зберігання гри відбувається в GameInstance. В ньому створено функцію Checked for Saved Info, яка спочатку завантажує дані зі слоту за допомогою вузла Load Game from Slot, вхід Slot Game якого поміщено в змінну PlayerSlot. Створено нову функцію LoadPlayerInfo, яка завантажує дані зі слота та за допомогою каста на SaveGame отримує PlayerInfo та записує її в змінну.

Було створено ще одну функцію – Save Info, яка зберігає дані в SG_PlayerInfo, який потім записує в слот зберігання (рис. 3.7).

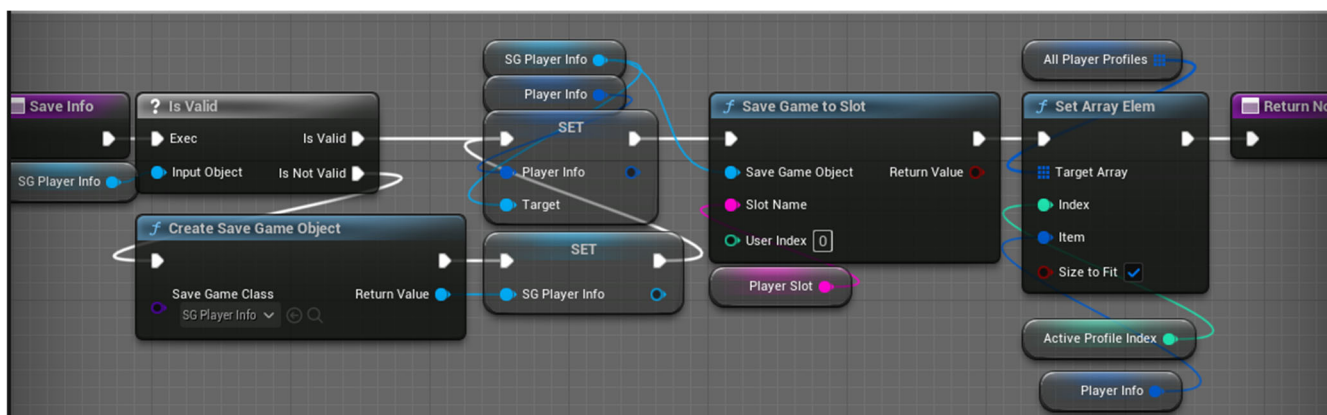


Рисунок 3.7 – Функція Save Info

У функції Checked for Saved Info за допомогою Branch перевіряється, чи завантажились дані зі слоту. Якщо так, викликається функція Load Player Info, якщо ні – Save Info (рис. 3.8).

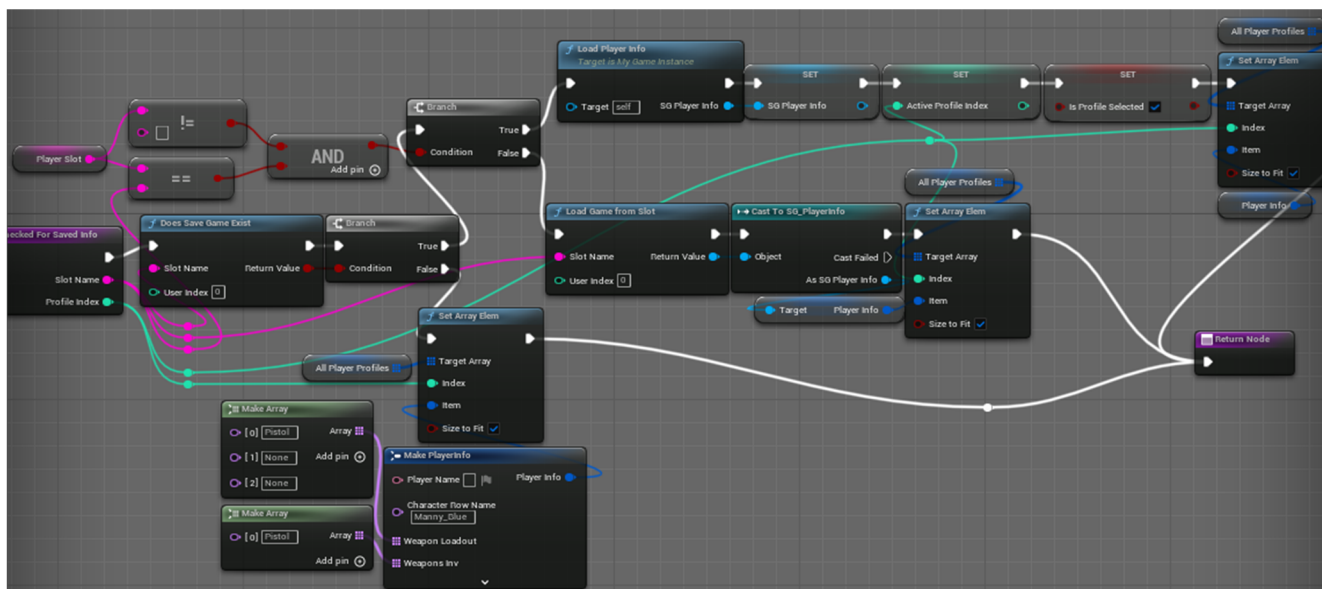


Рисунок 3.8 – Функція *Checked for Saved Info*

Далі у віджет меню було додано рядок введення імені гравця (Editable Text). В GameInstance створеною кастомну подію Change Player Name, яка оновлює ім'я гравця, яке буде введено в рядок в меню. Також змінено функцію UI_ShowMainMenu шляхом додавання відображення поточного імені персонажа.

У графі подій викликається подія On Text Committed створеного рядка введення, яка викликає подію Change Player Name з GameInstance.

3.2 Створення меню налаштувань гри

Для початку створення меню налаштувань було створено віджет Option, який використовується для створення налаштувань. Віджет складається з тексту та слайдера, розташованих в горизонтальній коробці.

У графі подій на Event Construct змінюється текст текстового поля, в нього передається нова змінна Option Name. Далі створено диспетчер подій з назвою On Value Changed. Диспетчери подій дозволяють Blueprint-класу повідомляти рівень про свій стан [6]. Викликається подія слайдера On Value Changed, яка запускається, коли змінюється значення слайдера. За допомогою Sequence розгалужено граф: перше розгалуження – значення, яке передає подія, записано в

змінну; друге розгалуження – перевіряється, чи дорівнює старе значення новому. Якщо ні – викликається диспетчер подій.

Також створено віджет Option1, який замість слайдера містить випадаючий список. Створений віджет використовується для вибору розширення екрану. В графі замість On Value Changed викликається On Selection Changed списку, яка викликає однойменний диспетчер подій, який також треба створити.

Віджет Settings – основний віджет меню налаштувань. Він складається з восьми віджетів Option та віджета Option1, які відповідають дев'ятьом основним налаштуванням графіки: тіні, відбиття, освітлення, візуальні ефекти, текстури, затінення, листя, згладжування та розширення екрану, а також трьох кнопок для встановлення режиму екрану, кнопки прийняття та повернення усталених значень. Також в верхньому лівому куті знаходиться кнопка повернення в меню.

В графі подій для показу поточних налаштувань створено кастомну подію Show Current Settings, яка викликається на Event Construct. Вона бере поточні налаштування та за допомогою Sequence створює розгалуження паралельних подій (рис. 3.9).

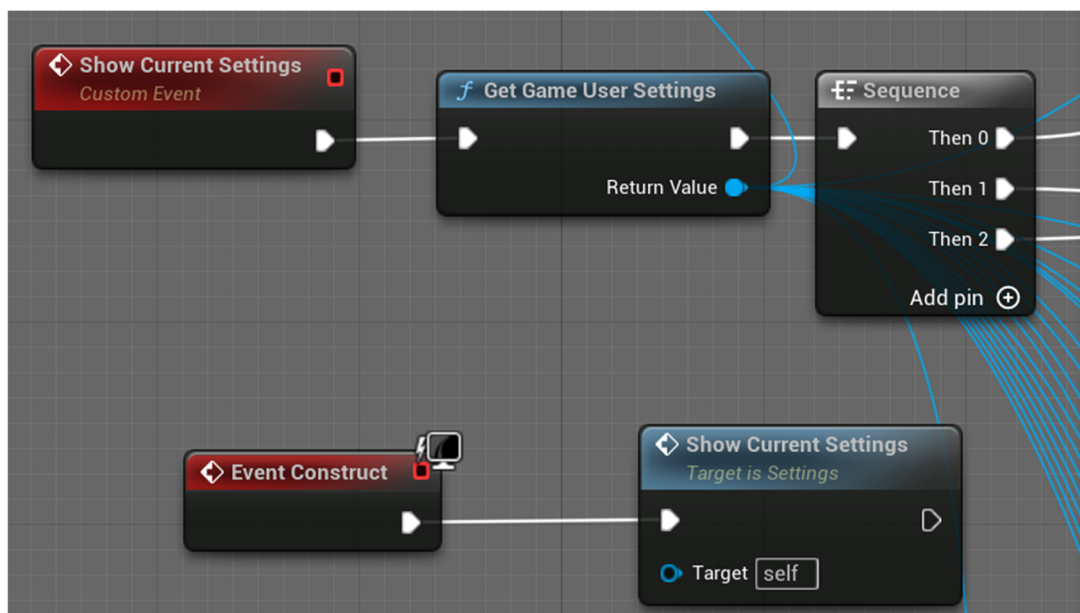


Рисунок 3.9 – Подія Show Current Settings

Всього подія містить три паралельні розгалуження:

– перше – виведення поточних налаштувань графіки. За допомогою Load Settings завантажуються налаштування, далі за допомогою гетерів значення присвоюються слайдерам відповідних опцій (рис. 3.10);

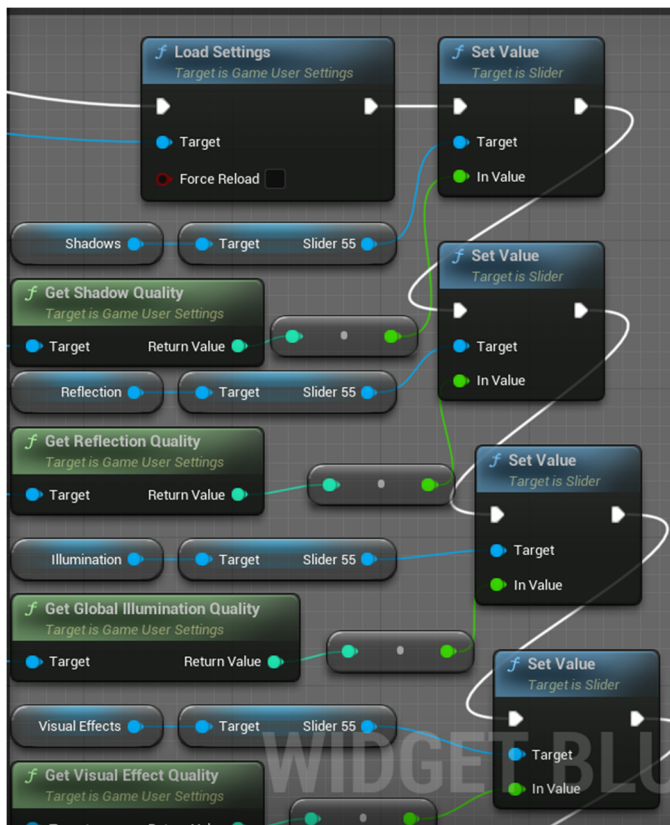


Рисунок 3.10 – Поточні налаштування графіки (частина графу)

– друге – поточний режим екрану. Get Fullscreen Mode отримує поточний режим екрану, далі за допомогою Switch on та створеної функції Enable Buttons кнопки стають неактивними, залежно від поточного режиму екрану (рис. 3.11 та рис. 3.12);

– третє – заповнення випадаючого списку можливими значеннями розширення та поточне розширення екрану. Clear Options очищає список від значень, далі отримуються всі можливі значення розширення екрану та додаються до списку за допомогою For Each Loop. Після заповнення значеннями з Game User Settings береться поточне розширення, конвертується в рядок та виводиться в список як обрана опція (рис. 3.13).

The diagram illustrates a Bolt visual script for enabling buttons. It consists of the following components and connections:

- Enable Buttons Node:** The starting node, which has three outputs: **Full** (red dot), **Full Windowed** (red dot), and **Windowed** (red dot).
- Full Button Node:** A button node with a **Full Button** label. Its **Target** input (blue dot) is connected to the **Full** output of the **Enable Buttons** node. Its **In Is Enabled** input (red dot) is connected to the **Full** output of the **Enable Buttons** node.
- FWButton Node:** A button node with a **FWButton** label. Its **Target** input (blue dot) is connected to the **Full Windowed** output of the **Enable Buttons** node. Its **In Is Enabled** input (red dot) is connected to the **Full Windowed** output of the **Enable Buttons** node.
- Windowed Button Node:** A button node with a **Windowed Button** label. Its **Target** input (blue dot) is connected to the **Windowed** output of the **Enable Buttons** node. Its **In Is Enabled** input (red dot) is connected to the **Windowed** output of the **Enable Buttons** node.
- Return Node:** The final node in the sequence, which receives the output from the **Windowed Button** node.

The script is designed to enable the **Full**, **FW**, and **Windowed** buttons in sequence, with each button's **In Is Enabled** input being connected to its corresponding output from the **Enable Buttons** node.

The screenshot displays a Node-RED workflow for managing screen resolution options in a game. The workflow is organized into two main sections: 'Supported Resolutions' and 'Current Screen Resolution'.

Supported Resolutions Section:

- Clear Options:** A function node that clears the dropdown menu. Its target is a combobox string.
- Get Supported Fullscreen Resolutions:** A function node that fetches supported resolutions. Its return value is an array of resolutions.
- For Each Loop:** A loop node that iterates over the array of resolutions. The loop body contains:
 - Convert Resolution:** A function node that converts a resolution (e.g., "1920x1080") into a string format (e.g., "1920x1080").
 - Add Option:** A function node that adds the converted resolution string to the dropdown menu.

Current Screen Resolution Section:

- Get Screen Resolution:** A function node that fetches the current screen resolution from the game's user settings.
- Convert Resolution:** A function node that converts the current resolution into a string format.
- Set Selected Option:** A function node that sets the selected option in the dropdown menu to the current resolution string.

The workflow uses a combobox string to represent the dropdown menu and a string to represent the selected option. The 'Add Option' and 'Set Selected Option' nodes are both targeted to the same combobox string.

Рисунок 3.13 – Поточне розширення екрану

Встановлення налаштувань реалізовано в такий спосіб:

- встановлення розширення екрану: викликається подія списку розширень On Selection Changed та встановлюється обрана опцію як поточне розширення;
- встановлення режиму екрану: викликаються події On Clicked трьох кнопок для зміни режиму, за допомогою Set Fullscreen Mode змінюється режим екрану, інші кнопки стають неактивними;
- встановлення налаштувань графіки: викликаються події On Value Changed для всіх опцій та встановлюється обране значення в якості поточного за допомогою сетерів. Також на подію On Clicked застосовуються обрані налаштування;
- повернення усталених налаштувань: на подію On Clicked кнопки Default отримуються поточні налаштування, повертаються усталені значення.

3.3 Реалізація мультиплеєрної частини

В цьому підрозділі описано реалізацію мультиплеєрної логіки проєкту. Спочатку було створено меню мультиплеєра, яке використовується для створення та пошуку сесій, шляхом дублювання головного меню. В головному меню видалено кнопку Host Game, а кнопку Search Game перейменовано в Multiplayer. Також дублікат перетворено в повноцінне меню мультиплеєра – додано кнопки для створення та приєднання до сесії, а також кнопки зміни типу приєднання та повернення в меню.

В GameInstance створено нову функцію Change Connection Type, яка визначає, чи є приєднання локальним та змінну IsLAN типу Boolean, в яку записується результат. В меню мультиплеєра створено функцію Set Connection Type Text для зміни тексту, який виводить тип з'єднання. Функція викликається з Event Construct.

Далі було реалізовано створення сесій та приєднання до них. В GameInstance створено кастомну подію CreateMPSession для створення власної сесії. Спочатку змінній IsSingle присвоюється значення False (так як це

багатокористувальницька сесія), потім власне створюється сесія, під'єднавши контролер та змінну IsLAN для визначення типу з'єднання. Якщо сесію створено, відкривається рівень лобі (реалізація якого описана далі; в опціях необхідно вказати “listen” для того, щоб інші гравці змогли приєднатись), якщо ні – виводиться повідомлення про невдачу.

Також створено кастомну подію JoinMPSession для приєднання до сесії, яка має вхід SessionResult типу Blueprint Session Result для отримання результату приєднання. Аналогічно створенню сесії, встановлюється значення False змінної IsSingle, потім гравця приєднує до сесії, вказавши контролер та передавши результат сесії, а також виводиться повідомлення у разі невдалого приєднання.

В меню мультиплеєра для створення сесії викликається подія CreateMPSession на On Clicked кнопки Host Game. Для приєднання до сесії в меню мультиплеєра було додано ScrollBox – це список доступних серверів. Далі створено новий віджет ServerInfo – він має вигляд кнопки та містить відомості про сервер (назву, кількість гравців та пінг). В графі подій даного віджета створено дві змінні – Session Result типу Blueprint Session Result та Multiplayer Menu однойменного типу. Далі створено кастомну подію Update Server Info для отримання даних про сервер з Session Result (рис. 3.14).

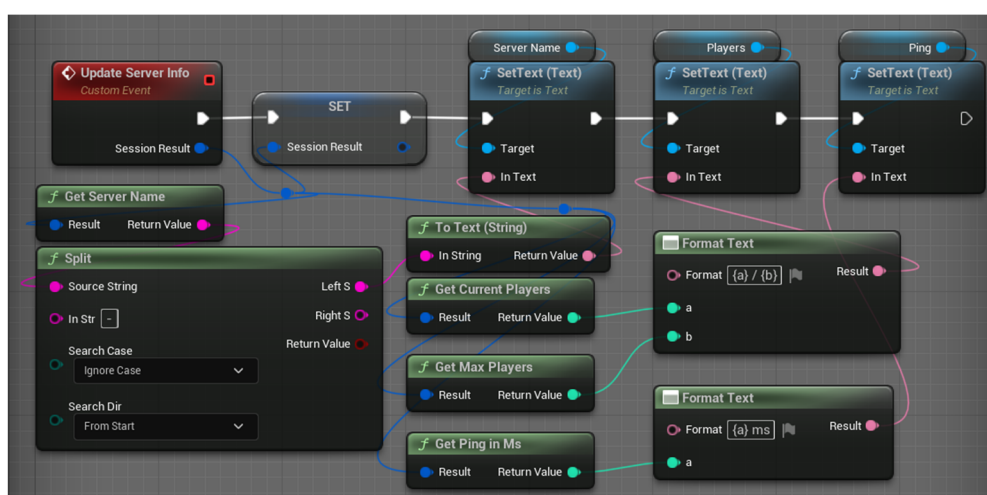


Рисунок 3.14 – Отримання даних про сервер

Також на подію On Clicked кнопки сервера гравець приєднається до нього, а також на час завантаження стають неактивними кнопки меню.

В меню мультиплеєра створено граф події On Clicked кнопки Search Game. Спочатку список серверів стає видимим (усталено він прихований), після чого ScrollBox та масив серверів очищуються, кнопки меню стають неактивними та починається пошук сесій. Якщо виникне помилка – виводиться повідомлення про невдачу, кнопки меню стають активними. Якщо пошук пройшов успішно – перевіряється, чи є порожнім масив результатів. Якщо так – аналогічно виникненню помилки, якщо ні – за допомогою For Each Loop створюється для кожного знайденого сервера віджет з інформацією та додається в список.

Також в Event Construct додано очищення ScrollBox та масиву серверів, встановлення правильного тексту кнопки Search Game, активація кнопок меню та приховування списку серверів.

Створено меню лобі, шляхом дублювання головного меню, з якого було видалено все, залишивши дві кнопки для старту гри та виходу в меню, також додано поле з чотирма рядками, які міститимуть імена приєднаних гравців. Далі в контролері меню створено кастомну подію End Client Session, яка завершуватиме сесію та перенаправлятиме гравця в головне меню.

В контролері меню додано нову змінну, яка містить масив контролерів меню всіх приєднаних гравців. В графі подій меню лобі викличемо подію натискання кнопки Main Menu. Спочатку перевіряється, чи є рівень, на якому зараз знаходиться гравець, сервером. Якщо ні – рівень знищується та відкривається головне меню. Якщо так – береться каст на GameModeMenu, з нього береться масив контролерів гравців та за допомогою For Each Loop сесія завершується для кожного гравця (рис. 3.15).

Для створення графу початку гри в GameInstance створено кастомну подію для переходу на обраний рівень. В меню лобі створено граф події натискання кнопки Start Game, яка, взявши каст на GameInstance та GameModeMenu, викликає подію Travel to Map із переданою з GameMode кількістю приєднаних гравців.

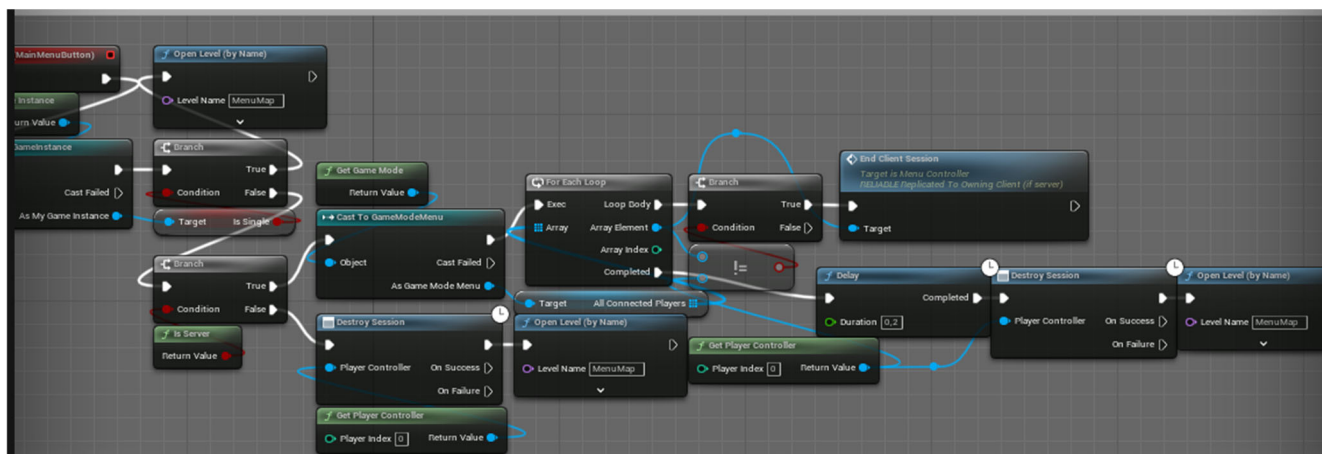


Рисунок 3.15 – Перехід з лобі в меню

Також в Event Construct створюються масиви усіх імен гравців та рамок, які їх містять, а також змінюється текст та стає неактивною кнопка Start Game, якщо клієнт гравця не є сервером.

В контролері меню створено функцію для відображення меню лобі, подібну функції відображення головного меню, яка викликається в кастомній події CreateLobbyMenu з графу подій контролера. Також на BeginPlay перевіряється, чи є контролер гравця локальним. Якщо так – встановлюється фокус на інтерфейсі. В GameModeMenu викликається подія Handle Starting New Player, яка працює, коли до гри приєднується новий гравець. Вона додає нового гравця до масиву гравців та створює меню лобі, якщо він знаходиться на рівні лобі.

Далі було реалізовано завантаження та виведення в список імен гравців. В контролері меню створено функцію Load Saved Profile, яка завантажує профіль гравця або створює його, якщо він недоступний. Дана функція викликається в кастомній події контролері.

В меню лобі створено дві функції – Update Player Names оновлює список гравців (рис. 3.16), а Clear Player Names очищає список (рис. 3.17).

В контролері створено кастомну подію для оновлення списку імен гравців, після чого в GameModeMenu створено кастомну подію, яка оновлює лобі (рис. 3.18).

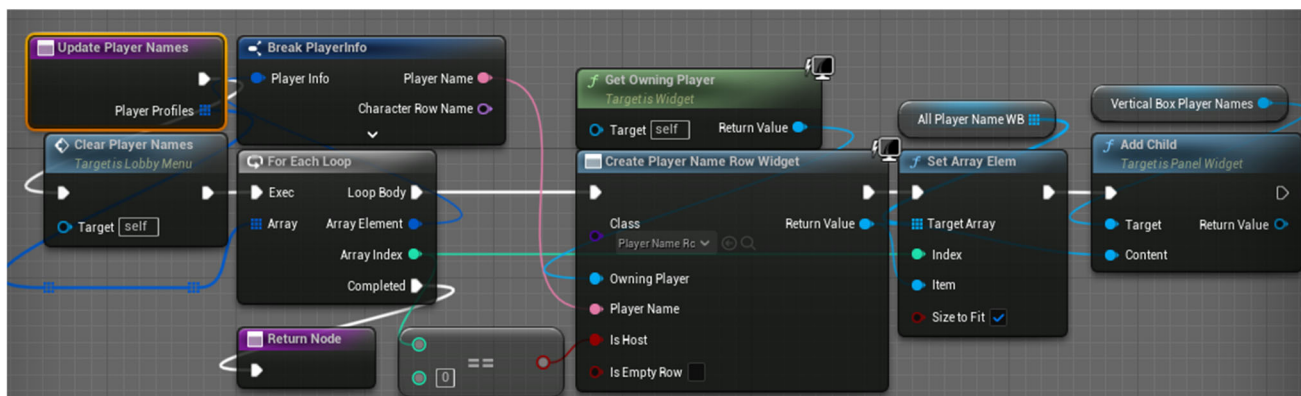


Рисунок 3.16 – Update Player Names

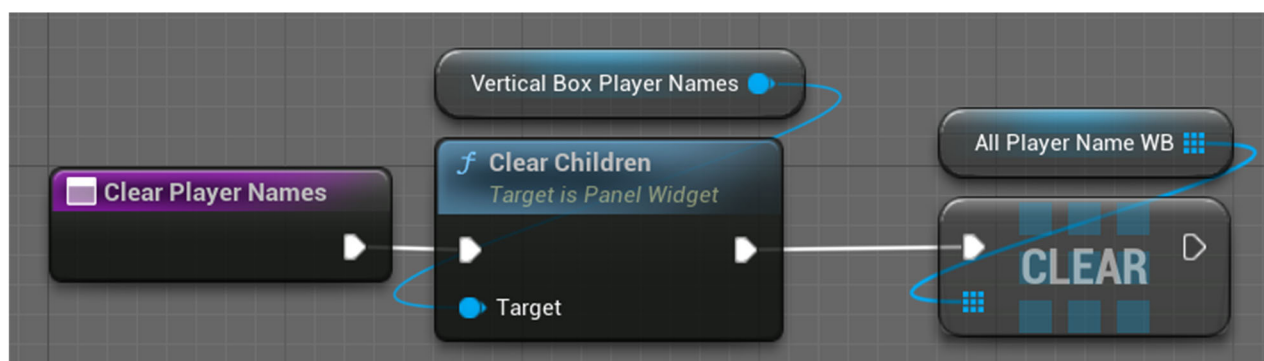


Рисунок 3.17 – Clear Player Names

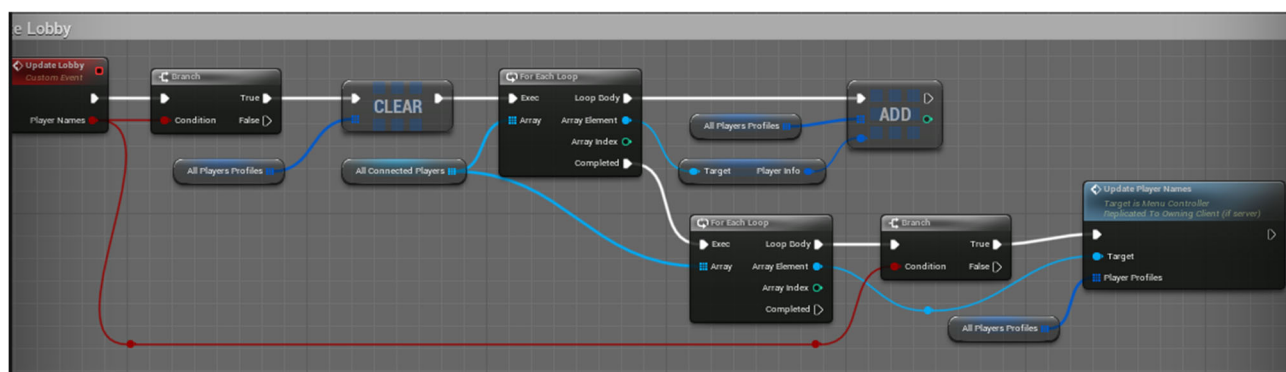


Рисунок 3.18 – Update Lobby

В контролері меню створено кастомну серверну подію для оновлення лобі, яка викликається після Load Saved Profile.

Далі було реалізовано плавний перехід між лобі та ігровим рівнем. Для початку в меню лобі було додано новий компонент – CheckBox для активації/деактивації плавного переходу. Далі на подію зміни CheckBox

встановлюється значення вбудованої змінної Use Seamless Travel в залежності від активності CheckBox.

Для використання Seamless Travel необхідно створити Travel Map. Після створення в Project Settings → Maps&Mods створену мапу встановлено в опції Transition Map. Також до графа BeginPlay в контролері гравця додано Remove All Widgets, так як після переходу меню лобі залишається на екрані. Далі було створено віджет завантаження.

В контролері меню було продубльовано функцію UI_Show_LobbyMenu та замінено референси меню лобі на референси екрана завантаження – ця функція відображає екран завантаження та викликається в графі з кастомної події. В меню лобі змінено граф початку гри. Після кастування на GameModeMenu додано цикл, який створює екран завантаження для всіх приєднаних гравців. Після завершення циклу відбувається перехід на рівень.

Далі було додано можливість вибирати мапу для гри. Спочатку створено структуру LevelsInfo, яка містить дані про всі ігрові рівні. Структура містить три змінні: назву мапи, рядок переміщення та прев'ю. Далі було створено Data Table, яка містить дану структуру – вона містить чотири записи.

В меню лобі додано новий компонент – меню вибору мапи (яке складається з назви мапи, прев'ю та списку вибору мап), а також створено новий віджет LevelName – рядок з назвою мапи. В графі подій створеного віджету додано три змінні – назва мапи (тип Text), назва рядка мапи в Data Table (тип Name) та меню лобі (тип Lobby Menu Object Reference). В меню лобі створено дві нові змінні – обраний рядок рівня (тип String) та масив всіх рівнів (тип LevelName Object Reference, Array). Продовжено подію BeginPlay: додано очищення масиву рядків та випадального списку рівнів, отримання даних з таблиці, створення рядків з отриманими даними та додавання їх до списку рівнів.

В контролері меню створено кастомну подію, яка викликає оновлення обраного варіанту, після чого в GameModeMenu та викликати мемо дану подію при необхідності після оновлення імен гравців з події Update Lobby. В меню лобі

створено кастомну подію, яка оновлює обраний рівень на сервері (рисунок 3.19). Створена подія викликається на натискання кнопки рівня у віджеті LevelName.

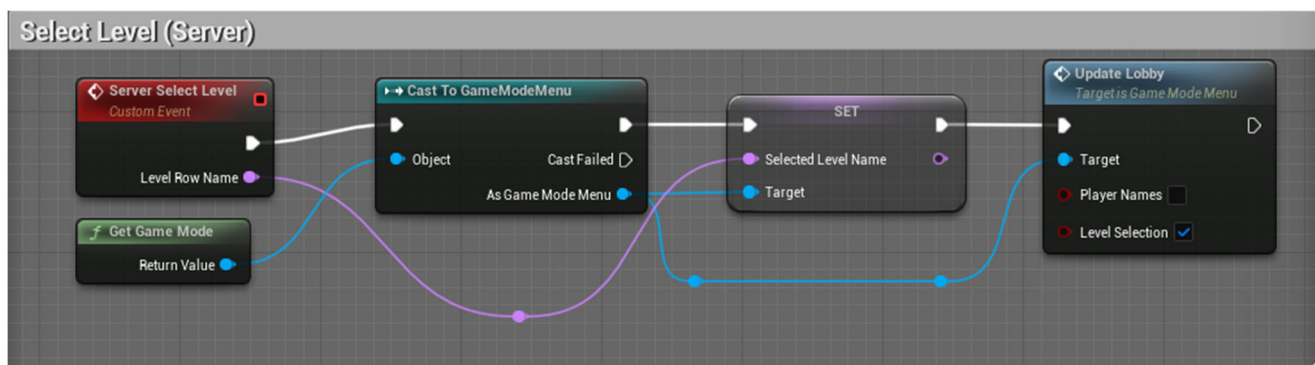


Рисунок 3.19 – Server Select Level

На всі ігрові рівні додано по чотири Player Start (так як максимальна кількість гравців – 4) та кожному вказано тег від 0 до 3. В GameMode створено граф події Handle Starting New Player, яка додає контролер приєднаного гравця в масив всіх контролерів, далі бере кількість приєднаних гравців з GameInstance та перевіряє, чи є сесія одиночною. Якщо так – встановлюється кількість гравців (1), після чого гравець спавниться на нульовий Player Start. Якщо ні – перевіряється, чи рівна кількість приєднаних гравців розміру масиву приєднаних контролерів. Якщо так – запускається цикл після затримки в 0.5 секунди, який респаунить гравців на відповідні Player Start.

Далі створено віджет, який використовується в якості HUD (heads-up display – частина інтерфейсу користувача, яка відображається на передньому плані ігрового простору). В графі створено кастомну подію, яка вестиме посекундний відлік до початку раунду та прибиратиме його через три секунди.

В контролері створено функцію для створення ігрового HUD, яка аналогічна функції створення головного меню. Змінено граф події BeginPlay: видалення віджетів розташовано після перевірки контролера, після чого створюється ігровий HUD. В Blueprint гравця створено кастомну подію, яка відключає керування – вона викликається в події Possessed.

В контролері створено дві кастомні події для підключення керування та ведення відліку до початку гри. В GameMode створено кастомну подію для старту рівня, яка виводить для всіх приєднаних гравців відлік та повертає керування після його закінчення. Викликається дана подія з події спавна гравців після закінчення циклу (для багатокористувальницької сесії) та після спавна гравця (для одиночної сесії).

Далі було створено віджет меню паузи, шляхом дублювання головного меню. З дубліката видалено рядок введення імені гравця та змінено кнопки Single Player та Multiplayer на Resume Game та Back to Main Menu відповідно. Також додано Widget Switcher та віджет налаштувань для переходу до меню налаштувань. В GameInstance створено нову функцію для створення меню паузи, аналогічну іншим функціям створення меню. В контролері створено кастомну подію для показу ігрового інтерфейсу. В GameInstance створено нову функцію Toggle Pause Game, яка виводить меню паузи, а також активує паузу в одиночній грі (рис. 3.20).

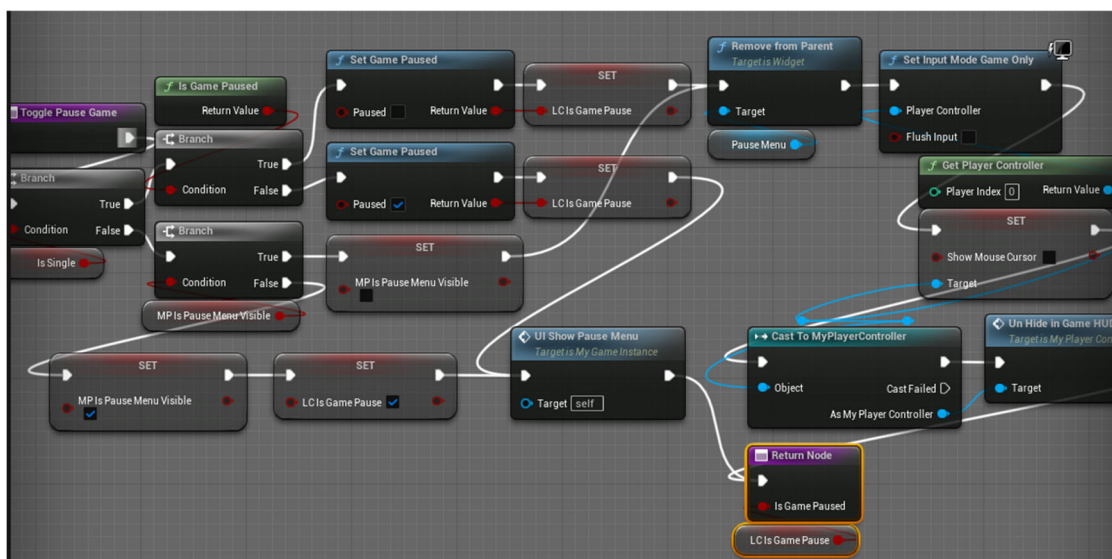


Рисунок 3.20 – Toggle Pause Menu

Після створення функцій назначено паузу на клавішу Escape в Action Mappings (Project Settings). Потім в Blueprint персонажа на подію BeginPlay

отримується ігровий HUD з контролера та записується в змінну. Далі було створено граф подій для натискання Escape.

В меню паузи створено логіку для кнопок повернення в меню та виходу з гри, аналогічно відповідним кнопкам в меню лобі та головному меню (для цього в контролері створено подію End Client Session, подібну однойменній події контролера меню, а також кастування на контролер меню змінено на кастування на звичайний контролер, інакше логіка працювати не буде).

Далі було створено Player State. Даний об'єкт створюється для кожного гравця на сервері та містить інформацію про гравця, яка є важливою для мультиплеєра: ім'я, рахунок і тому подібне [7]. В ньому створено нову змінну, яка містить структуру з даними гравця, а також подію Copy Properties, яка записує дані структури в новий Player State. В GameModeMenu доповнено граф події Update Lobby для того, щоб оновлювати Player State після переходу на інший рівень. Таким чином, дані гравця не буде втрачено.

У віджеті ігрового інтерфейсу в лівий нижній кут додано чотири текстових поля для відображення імен гравців. Створено граф для події Construct: створюється масив імен гравців, за допомогою циклу видаляються непотрібні (якщо приєдналося менше чотирьох гравців), після видалення за допомогою іншого циклу виводяться імена приєднаних гравців.

3.4 Вибір зброї та ігровий інтерфейс

Було реалізовано можливість гравця змінювати зброю. Для початку було створено монтажні анімації вибору зброї, в них змінено слот на Upper Body, а також створено структуру, яка містить дані про зброю. Далі створено Data Table, в яку записано всі дані структури для всіх видів зброї.

В Animation Blueprint створено змінні, які містять анімації стрільби та Blend Space для них, а також поточний Blend Space (BS). Внесено зміни в оновлення анімації – перед запуском монтажу обертання встановлюється поточний BS (ADS, якщо персонаж веде прицільний вогонь; Hipfire, якщо стріляє від бедра). Також внесено відповідні зміни в граф анімацій.

В Blueprint персонажа додано в інвентар зброю: спочатку додамо дві змінні (масив імен зброї в інвентарі та індекс активної зброї) та створено функцію, яка додає зброю з вказаною назвою та слотом в інвентар. Ця функція викликається в BeginPlay тричі (для додавання трьох видів зброї в три слоти інвентаря).

Створено функцію для реплікації змінних, які відповідають за дизайн зброї (меш, звук, анімації і т.п.). Створено події для вибору зброї (серверну та Multicast), серверна подія викликається на BeginPlay.

В Project Settings назначено вибір зброї на клавішу Q. Створено події для перемикання зброї, які запускають анімацію обирання відповідної зброї (рис. 3.21). Для перемикання зброї викликається подія натискання клавіші Q (рис. 3.22).

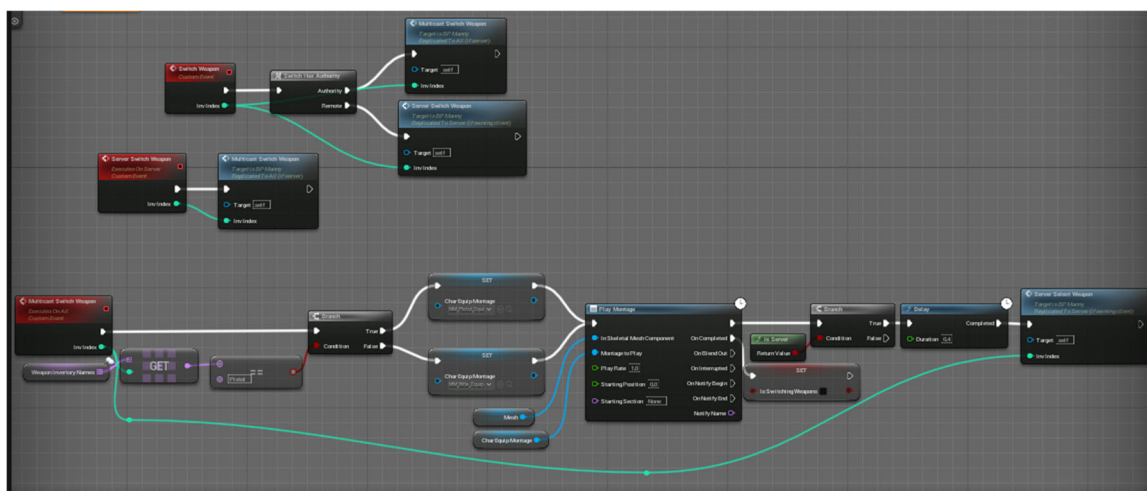


Рисунок 3.21 – Події Switch Weapons

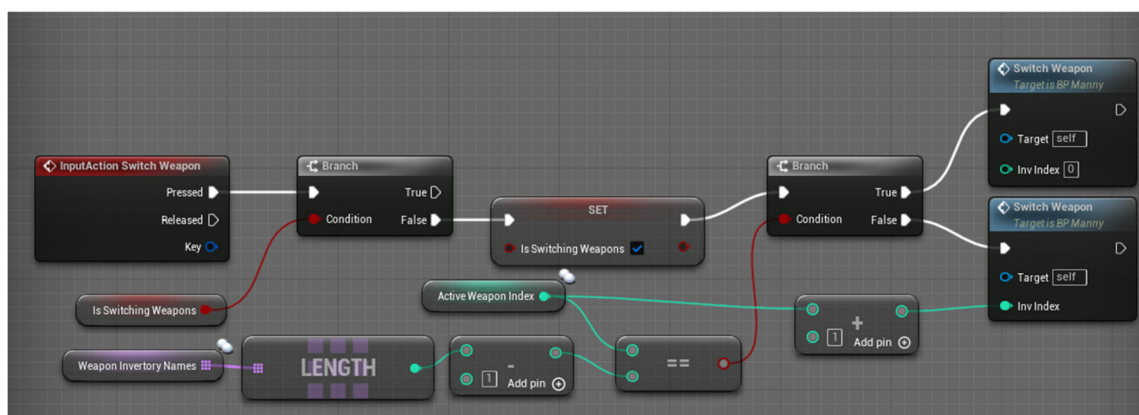


Рисунок 3.22 – Перемикання зброї

Допрацьовано реплікацію стрільби, так як на клієнті не чуто звуку стрільби і не видно ефектів. Створено дві нові змінні типу Boolean для відслідковування стрільби та натискання кнопки стрільби. Далі створено функцію для стрільби (подано в додатку А, рис. 1-3) та події для реалізації стрільби, де використовується створена функція (рис. 3.23).

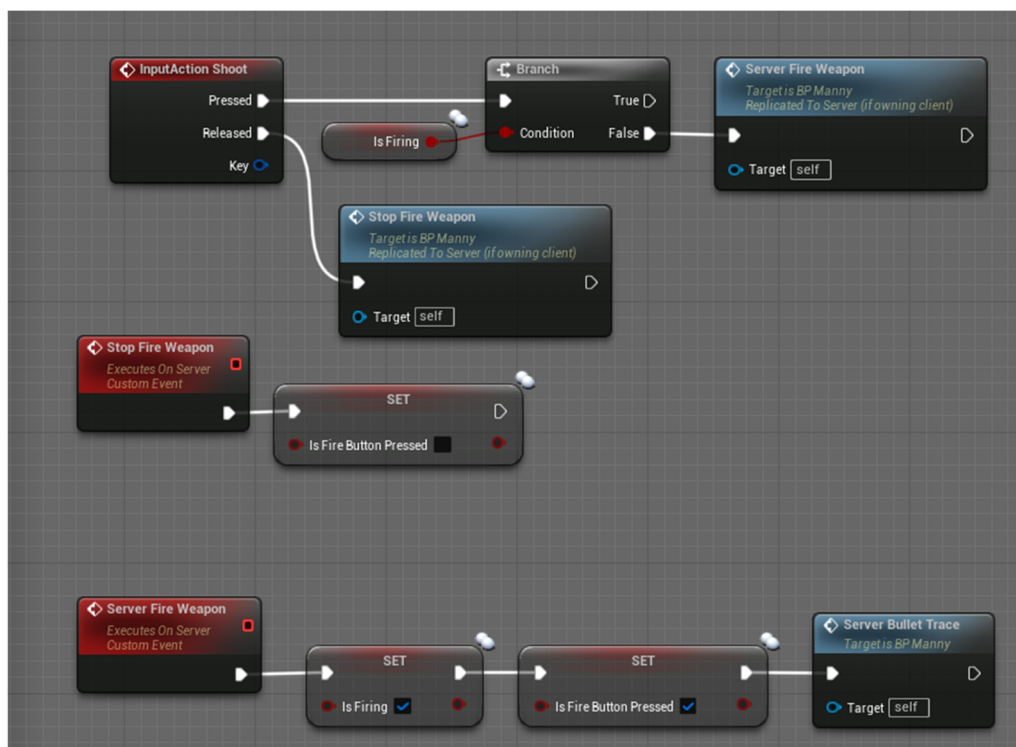


Рисунок 3.23 – Змінена подія стрільби

Також створено подію для запуску інших подій стрільби (ефекти та звук), яка викликається з події Server Fire Weapon.

Реалізовано створення ігрового інтерфейсу. В наявний віджет інтерфейсу додано приціл та рамку з назвою зброї і кількістю патронів. В графі створено функції для оновлення назви зброї та кількості патронів в інтерфейсі. В Blueprint персонажа створено чотири змінні-масиви, які містять максимальну та поточну кількість патронів в обоймі і максимальну та поточну кількість патронів в інвентарі. Внесено зміни в функцію додавання зброї в інвентар, шляхом додавання записування кількості патронів в інвентарі та обоймі у відповідні масиви (максимальної та поточної кількості) з Data Table. Також внесено зміни в

функцію Set Weapon Cosmetics, в якій викликаються функції оновлення, створені раніше в HUD, після перевірки на локальність контролю.

Реалізовано зміну кількості патронів в обоймі зброї та інвентарі гравця. Створено кастомну подію для оновлення кількості патронів в обоймі, після чого створено функцію для витрати патрона при стрільбі (рис. 3.24). Ця функція викликається в Server Fire Weapon перед трасуванням.

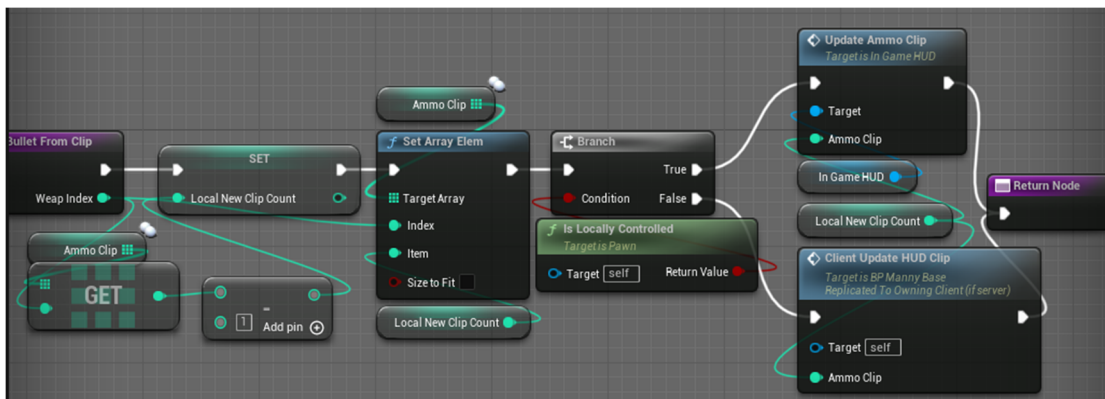


Рисунок 3.24 – Server Take Bullet from Clip

Далі створено функцію, яка перевіряє кількість патронів та визначає, чи може гравець стріляти та перезаряджатись (рис. 3.25). Вона викликається в події `Server Fire Weapon` на початку та після пострілу.

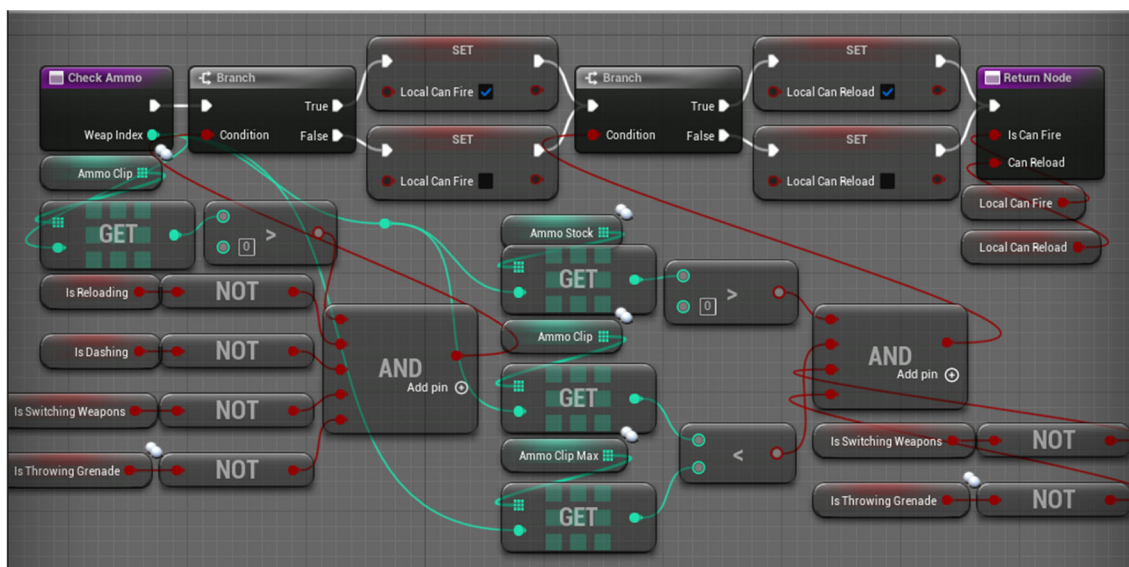


Рисунок 3.25 – Check Ammo

Створено подію оновлення патронів в інвертарі, аналогічну події оновлення кількості патронів в обоймі, після чого створено функцію для перезаряджання (подано в додатку А, рис. 4-5) та змінено подію перезаряджання (рис. 3.26).

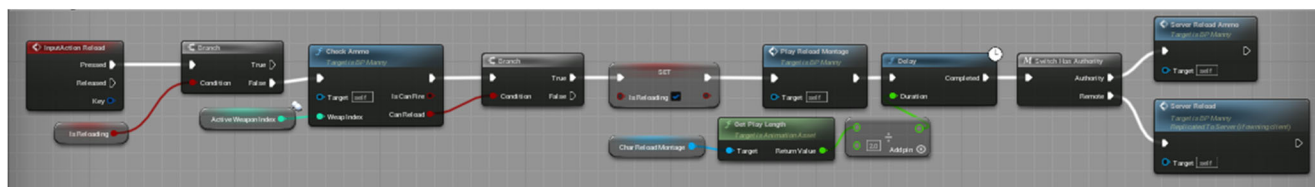


Рисунок 3.26 – Перезаряджання

Далі в контролері назначено HUD на подію On Posses, а також видалено відповідний функціонал з BeginPlay в Blueprint персонажа.

3.5 Вибір персонажа

Якщо в грі всі персонажі будуть однаковими, важче орієнтуватись та визначати, хто перед тобою. Тому було реалізовано вибір персонажа перед початком гри.

Створено новий об'єкт Actor – персонаж лобі. В ньому додано меш персонажа та текст, який розміщується перед ним (ім'я гравця). Створено функцію для встановлення імені гравця, потім в графі створено кастомні Multicast події для вибору меша, імені та анімації. Далі створено ще один об'єкт Actor – точка спавна гравця в лобі, в який було додано Billboard (щоб бачити місцезнаходження об'єкта) та булеву змінну Is Spot Used для відслідковування використання точки. Було розміщено чотири точки перед камерою на рівні лобі. В контролері меню створено дві змінні, які містять персонажа лобі та його точку спавна.

В GameModeMenu внесено зміни в Update Lobby шляхом видалення перевірки на початку та заміни в циклі Add на Set Array Element. Створено функцію для визначення використання точок спавна (рис. 3.27).

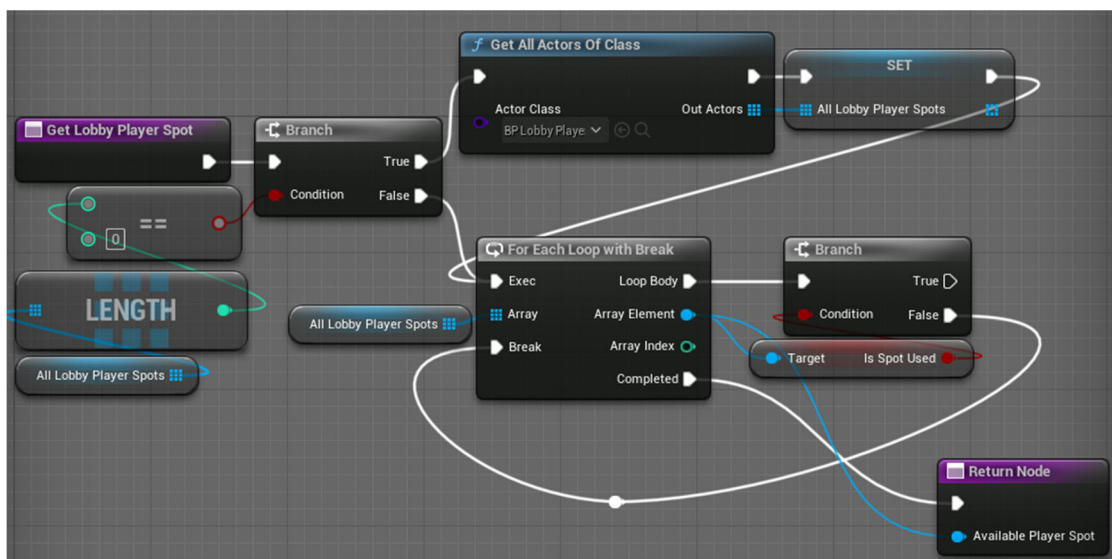


Рисунок 3.27 – Get Lobby Player Spot

Створено подію для оновлення обраного персонажа, яка створює персонажа в новій точці (якщо його нема) та оновлює дані персонажа (поки що тільки ім'я). Ця подія викликається в події Update Lobby (рис. 3.28).

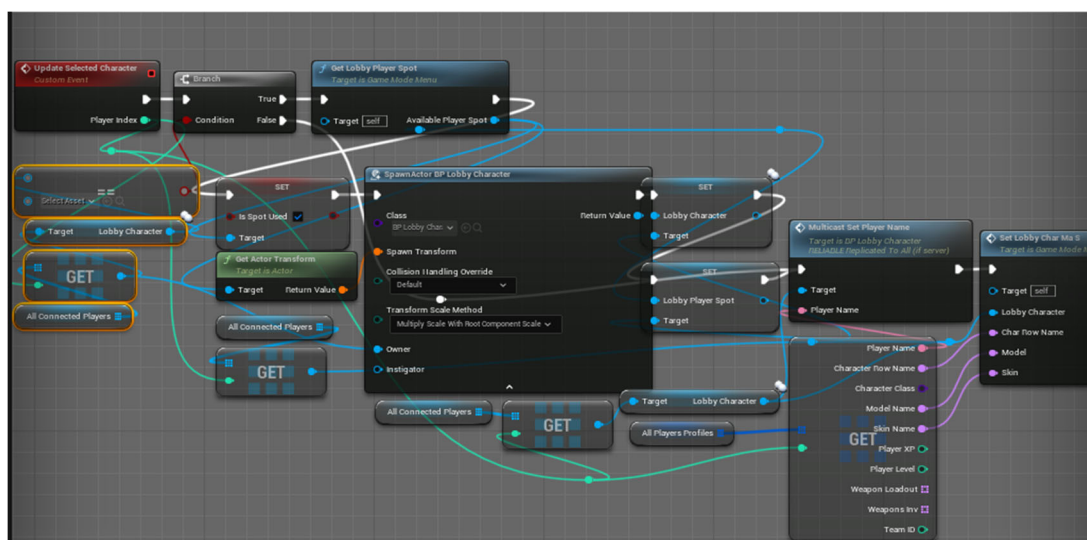


Рисунок 3.28 – Update Selected Character

Було створено дублікати меша та всіх потрібних матеріалів різних кольорів (синій та червоний є усталено, створено жовтий та чорний). Створено чотири дочірніх об'єкта для персонажа – це будуть чотири персонажі різних кольорів. Створено структуру, яка містить дані персонажа (меш та анімацію лобі) і

таблицю, яка містить структури з даними всіх персонажів. Також додано до структури та таблиці з даними гравця назву та клас персонажа. В GameModeMenu доповнено подію оновлення персонажа шляхом додавання подій встановлення меша та анімації.

Для початку створення меню вибору персонажа було реалізовано зміну ракурсу камери при переході в меню вибору. Для цього створено новий об'єкт Astar, який містить стрілку. В контролері меню створено подію для встановлення цілі камери лобі на сервері та змінено функцію відображення лобі шляхом додавання аналогічного функціоналу для сервера та клієнта. Продубльовано функцію відображення лобі та змінено референси меню лобі на референси меню вибору, таким чином було створено функцію відображення меню вибору персонажа. Аналогічним способом створено серверну подію встановлення цілі камери в меню вибору.

Створено віджет меню вибору персонажа, який містить чотири кнопки для зміни персонажа та кнопку повернення до лобі. В події Construct кнопки вибору персонажа додаються в змінну типу Map [8], створюється та встановлюється стиль кнопки обраного персонажа, яка стає неактивною. Також створено функцію для зміни стилю кнопок обраного та необраних персонажів і подію оновлення персонажа для контролера. Створена функція викликається на подію натискання кожної з кнопок персонажа, вказавши відповідні вхідні дані.

Обраного персонажа необхідно зберегти в SaveGame. Для цього в контролері створено серверну подію для оновлення інформації про гравця, в якій викликається подія Update Character Controller та функція, яка оновлює та зберігає дані. В меню лобі додано кнопку переходу в меню вибору персонажа та створено відповідний граф події On Clicked.

В контролері гри отримуються збережені дані гравця з SaveGame в події BeginPlay. Було створено граф функції SpawnDefaultPawnFor, яку можна знайти в меню Override (рис. 3.29).

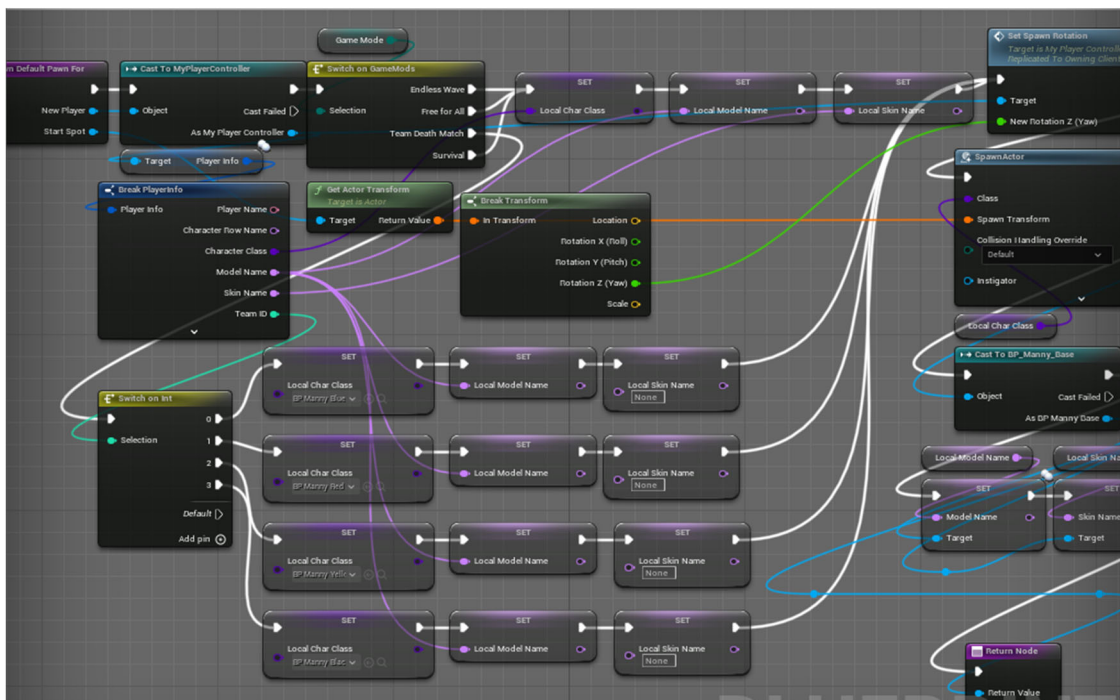


Рисунок 3.29 – *SpawnDefaultPawnFor*

В структуру та таблицю даних персонажа додано три змінних, які містять характеристики персонажа (здоров'я, витривалість та силу). У віджеті меню вибору персонажа додано рамку з характеристиками, які відображаються у вигляді стрічки прогресу. Створено функцію для оновлення характеристик персонажа, яка викликається при створенні меню (Construct) та натисканні кнопок вибору персонажа (функція Character Button Pressed).

Ці характеристики присвоюються власне персонажу (в батьківський об'єкт). Спочатку створено змінні характеристик, після чого в Construction Script в них записуються отримані з Data Table характеристики. В Construction Script усіх дочірніх об'єктів встановлено відповідні назви рядків таблиці перед виконанням батьківського скрипта.

Далі було додано можливість обирати статть персонажа та змінювати колір тіла. Створено дублікати меша Quinn та матеріалів різних кольорів. Спочатку було створено структуру, яка містить відомості про використання слотів матеріалу (в кожного персонажа два слоти матеріалу) та поміщена в іншу структуру, для якої, в свою чергу, створено Data Table. В таблицю записано всі меші та матеріали, які вони використовують. В меню вибору персонажа створено

меню кастомізації за допомогою Widget Switcher шляхом дублювання та зміни меню вибору. Також створено окремий віджет – кнопку кастомізації, яка використовується для вибору статі та шкіна персонажа. В Construct визначається текст кнопки (назву моделі, якщо це кнопка зміни статі; назву шкіна, якщо це кнопка зміни шкіна) та оновлюється структура та об'єкт SaveGame, в які додаються назви шкіна та моделі персонажа.

В меню кастомізації створено функції для відкриття та закриття меню, які викликаються на натискання відповідних кнопок. Функція відкриття меню витягує необхідні дані, записуючи їх в змінні (масив створених кнопок, мапа моделей та шкінів, масиви ключів моделей та шкінів) та створює відповідні кнопки зміни статі та шкіна. Функція закриття меню видаляє всі створені кнопки та очищає масиви даних, створені функцією відкриття. В контролері меню створено функції зберігання обраних моделей та шкіна шляхом дублювання функції Save Selected Character. В меню вибору персонажа створено функцію, яка змінюватиме шкін або модель при натисканні на відповідну кнопку (рис. 3.30).

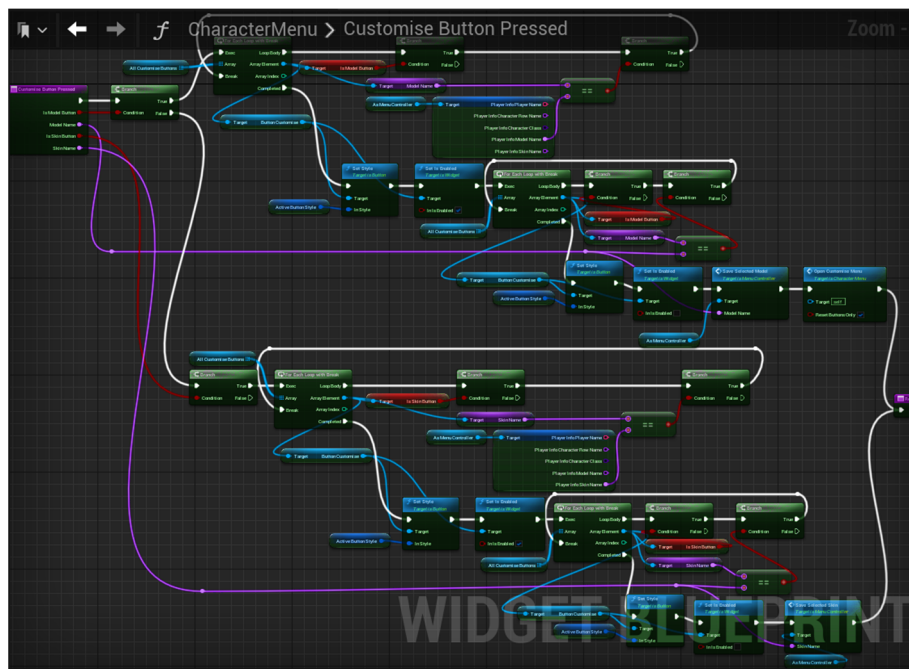


Рисунок 3.30 – Customise Button Pressed

В GameModeMenu змінено подію Update Selected Character, в яку додано зміну шкіна (для цього також створено Multicast подію для зміни шкіна в персонажі

лобі). Для того, щоб обраний скін з'являвся під час гри, в базовому об'єкті персонажа створено змінні, які містять назви моделі та скіна, та функцію для зміни моделі та скіна (рис. 3.31). Ця функція викликається на початку події BeginPlay. В GameMode змінено функцію Spawn Default Pawn For, в яку додано зміну скіна та моделі.

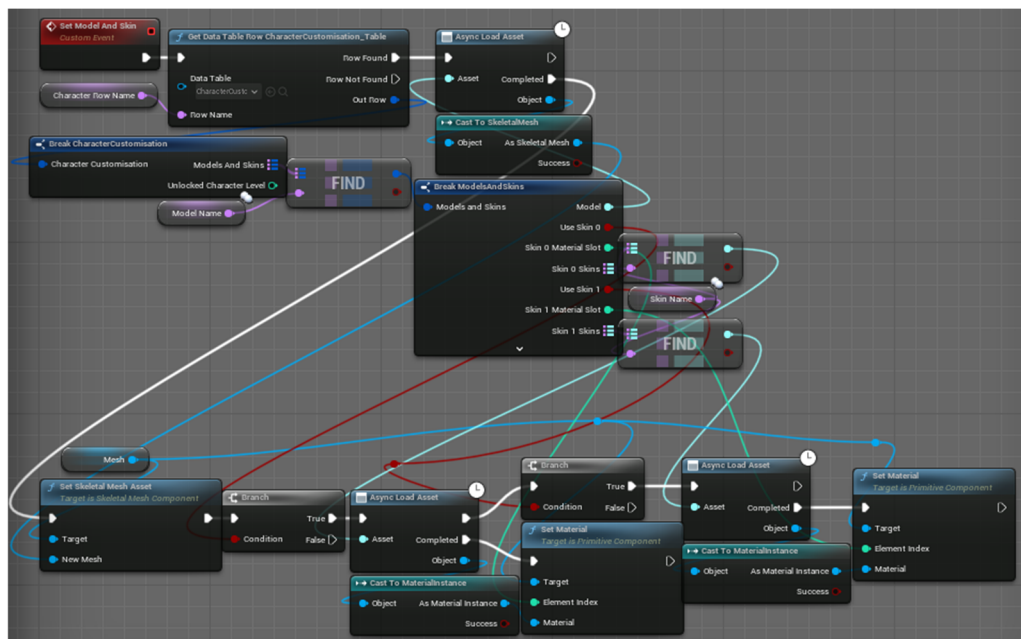


Рисунок 3.31 – Set Model and Skin

Для різноманітності геймплею в гру було додано рівні (ранги гравця), для підвищення яких потрібно отримувати досвід (вбиваючи інших гравців чи ботів).

Створено структуру, яка містить масиви кількості досвіду, необхідного для отримання рангу, та текстур рангів. Кожен масив складається з одинадцяти елементів (10 рангів + відсутність рангу). Також в структуру з даними гравця додано дві змінні – кількість досвіду та ранг гравця. У віджеті HUD додано ранг, його інтро та кількість досвіду. Створено дві функції для оновлення кількості досвіду та ранга гравця, які беруть значення зі структури даних гравця та виводять їх на екран. Далі було створено об'єкт класу Blueprint Interface – це інтерфейс досвіду. В ньому створено функцію Add XP. Даний інтерфейс додано до базового персонажа (опція Implemented Interfaces).

Далі було створено об'єкт класу Actor Component, який використовується для реалізації певних геймплейних функцій, пов'язаних з персонажем. Створено

функцію, яка записує в змінну дані про рівень та досвід гравця, отримані з відповідної структури. Далі було створено функцію, яка визначає, чи може гравець отримати наступний ранг та підвищує його, якщо може.

Створено функцію, яка оновлює досвід та ранг в контролері, а також в контролері створено кастомну подію для оновлення даних гравця в SaveGame. В Actor Component створено кастомну подію для ініціалізації об'єкта, яка записує в змінні досвід та рівень гравця, отримані з контролера та оновлює їх в контролері при необхідності, а також подію для додавання досвіду та підвищення рівня (рис. 3.32). Створений об'єкт додано до базового персонажа, подію ініціалізації викликано в BeginPlay.

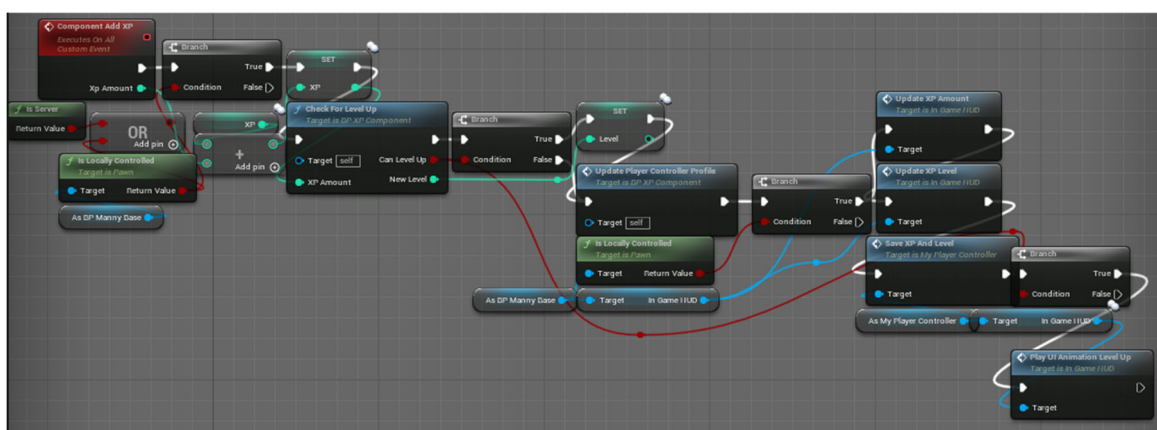


Рисунок 3.32 – Add XP

В Game Instance створено подію для скидання досвіду гравця шляхом дублювання та зміни події зміни імені (з даних гравця передається лише ім'я), після чого в головному меню було додано кнопку скидання досвіду та викликано створену подію на подію натискання кнопки.

Далі було створено новий віджет, який містить інформацію про ранги. Створено функцію для виведення на екран рангу та кількості досвіду для його отримання, яка викликається в Construct. Створений віджет додано в головне меню та меню лобі.

Далі було реалізовано відкриття зміни статі, персонажів та шкінів після здобуття певного рангу. Спочатку було додано в структури кастомізації та моделей нові змінні: в першу – ранг для розблокування персонажа; в другу –

3.6 Штучний інтелект ботів

Для забезпечення цікавого та динамічного геймплею в одиначній грі було реалізовано ботів – персонажів, яка керуються комп’ютером. Для їх створення використано безкоштовний пакет Paragon Minions з Epic Games Store. Для обраного меша було створено Blend Space, після чого було створено Animation Blueprint. Далі було створено два об’єкти – AI контролер (об’єкт, який використовується для назначення управління персонажами, керованими ШІ [9]) та Character (базовий персонаж для ботів). В графі створено нову функцію для визначення цілі, яку атакуватиме бот (рис. 3.35).

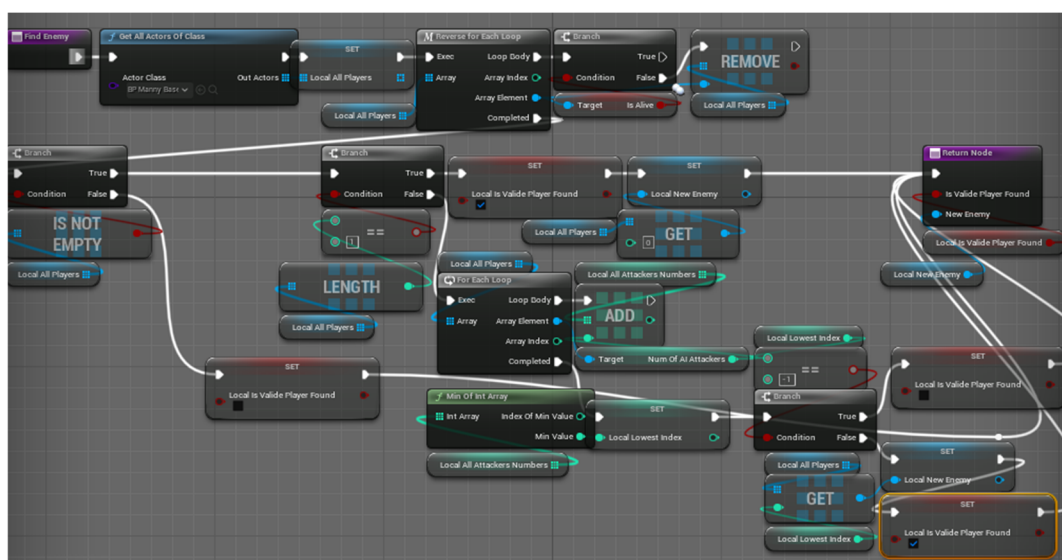


Рисунок 3.35 – Find Enemy

Далі було створено дві кастомні події: перша переміщує бота до гравця за допомогою AI Move To, якщо бот живий, а друга знаходить ціль та переміщує бота до неї, коли бот з’явиться на мапі. Щоб боти могли переміщуватись по мапі, необхідно додати навігаційну сітку [10].

Далі було реалізовано нанесення гравцем шкоди ботам. Для цього спочатку було створено новий канал трасування в Project Settings → Collision → Trace Channels. В Presets встановлено ігнорування каналу об’єктами Pawn та блокування об’єктами Character Mesh. В базовому персонажі створений канал трасування назначено в потрібних вузлах функції трасування.

Створено інтерфейс, який використовується для нанесення шкоди боту. Функція інтерфейсу має два входи та три виходи: вхідні дані – атакуючий гравець та кількість нанесеної ним шкоди; вихідні – визначення стану бота (живий/мертвий), кількість досвіду, який отримає гравець та визначення можливості нанесення шкоди.

В ВР бота створено кастомну подію для смерті, яка знищуватиме бота, якщо наявний атакуючий гравець. Далі було створено граф функції інтерфейсу, яка визначає стан бота і можливість нанесення шкоди та віднімає від кількості здоров'я кількість нанесеної шкоди.

В ВР персонажа створено нову функцію Server Process Hit, яка викликатиме функцію нанесення шкоди, якщо трасування заблоковано ботом. Викликається функція в іншій функції – Server Bullet Trace. Далі було створено Enumeration, який містить всі можливі напрямки нанесення шкоди.

Допрацьовано функцію інтерфейсу в ВР бота шляхом додавання визначення напрямку нанесення шкоди, який передається в подію смерті бота. Далі було створено монтаж анімацій смерті бота (всього їх 5), після чого в ВР бота створено нову функцію для встановлення монтажу анімації смерті в залежності від напрямку нанесення шкоди. Створено кастомну Multicast подію, яка відтворює обраний за допомогою створеної функції монтаж; вона викликається в події смерті бота. Також створено подію для активації Ragdoll фізики (рис. 3.36).

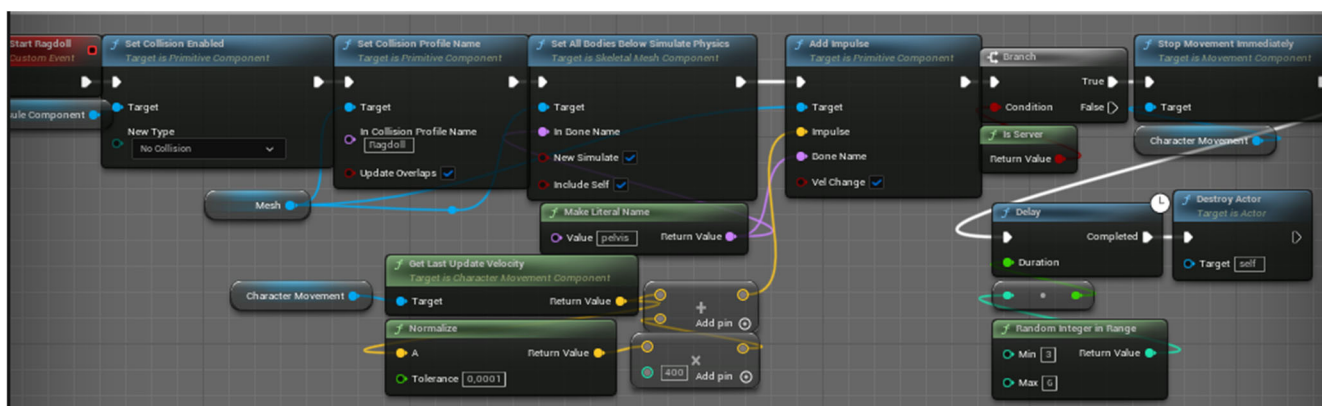


Рисунок 3.36 – Start Ragdoll (боту)

В AI контролері додано новий об'єкт – AI Perception та налаштовано потрібним чином (встановлено пресет, розпізнавання об'єктів). Він використовується для встановлення сприйняття ботом ігрового світу. Далі в директорії гри у файлі DefaultGame.ini в папці Config вписано такі рядки:

```
[/Script/AIModule.AISense_Sight]
bAutoRegisterAllPawnsAsSources=false
```

Після перезапуску проєкту в базовому персонажі додано об'єкт AI Perception Stimuli Source, до якого встановлено пресет, аналогічно AI Perception контролера.

Створено нову функцію, яка встановлює ціль бота в даному об'єкті та в контролері (рис. 3.37). Викликається в події On Spawned.

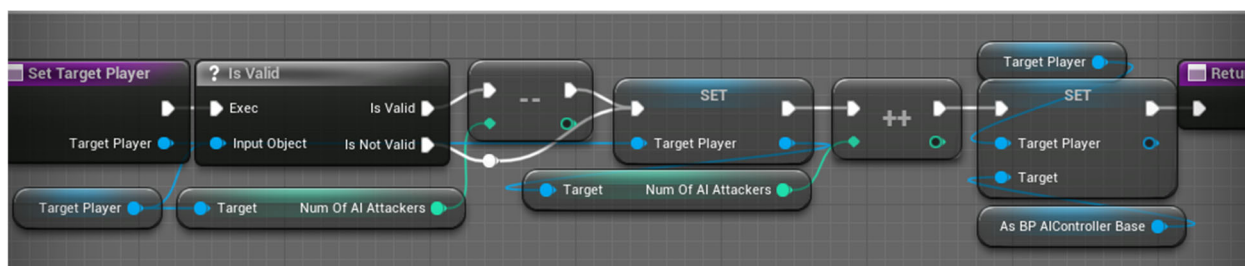


Рисунок 3.37– Set Target Player

В AI контролері створено граф події On Target Perception Updated для оновлення цілі бота. Далі було реалізовано ближню атаку ботів. Створено точку спавна бота (об'єкт Actor), після чого в об'єкті бота додано Box Collision та розташовано перед ботом. Створено подію на видалення тригера, яка викликається в події спавна та смерті. Далі було створено декілька подій для запуску та зупинки програвання монтажу атаки, які викликаються при початку та кінці перетину Box Collision з гравцем. Також створено функцію трасування для відслідковування ударів.

Створено новий інтерфейс, який додано до базового персонажа – він використовується для нанесення шкоди ботом гравцю. Функція інтерфейсу викликається в функції трасування бота. В ігровому інтерфейсі додано рядок з кількістю здоров'я, після чого створено функцію для його оновлення.

В ігровий інтерфейс додано накладання ефекту отримання шкоди – спочатку було додано малюнок в сам інтерфейс, далі створено анімацію його появи та зникнення при пораненні.

3.7 Налаштування параметрів переродження персонажа

В даному підрозділі описано реалізацію смерті та респава персонажів. Змінено функцію інтерфейсу шляхом додавання перевірки на наявність НР в гравця та створено функцію для програвання монтажу смерті. Також створено подію для запуску Ragdoll-фізики з використанням кастомної C++ функції. Реалізацію всіх функцій з використанням C++ подано в додатку Б.

Створено новий віджет – меню для респава та виходу після смерті, який складається з кнопок респава, спостереження та виходу в меню. При виході в меню з'являється меню підтвердження у випадку, якщо гравець є сервером, з якого можна вийти з гри чи повернутись до неї. В контролері створено функцію для відображення меню респава. Далі в GameMode було створено функцію для респава, подібну функції Spawn Default Pawn For, та подію Respawn After Dying для видалення померлого та створення нового персонажа в грі. Створена подія викликається в новій серверній події контролера. В меню респава викликається Respawn After Dying на клік кнопки респава.

Для реалізації режиму спостереження було додано до базового персонажа новий компонент Spring Arm, до якого, в свою чергу, додамо дочірній компонент. Створений компонент розміщено в місці розташування камери. В меню смерті додано рамку з кнопками контролю спостереження (перемикання між гравцями та респава). Створено дві функції для оновлення імені гравця, за яким ведеться спостереження та показу/приховування частин меню.

В контролері створено нові функції для оновлення цілі спостереження (рис. 3.38) та початку спостереження за нею. Далі було створено графи подій для натискання клавіш Q (попередній гравець), E (наступний гравець) та R (респава) і ланцюжок подій для спостереження за гравцем.

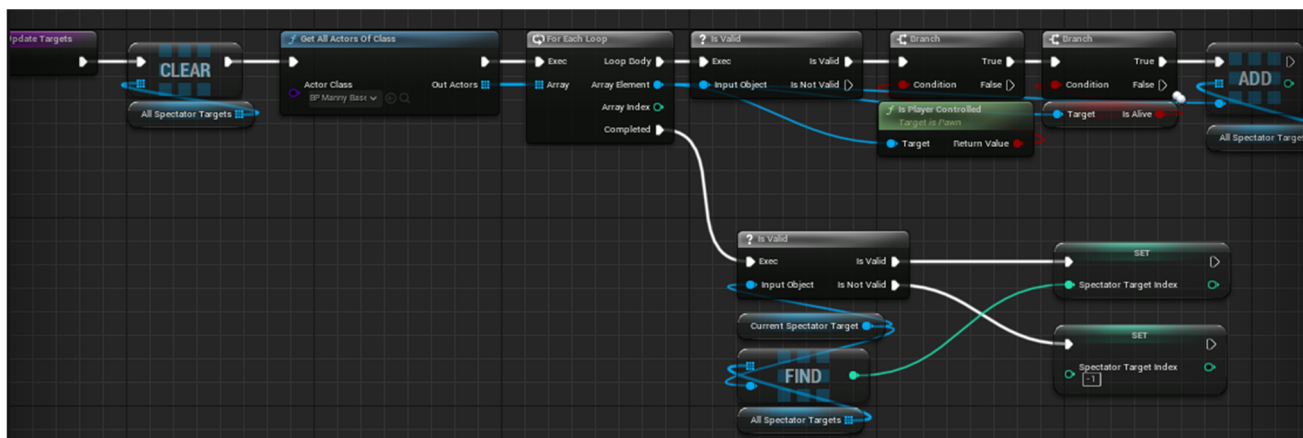


Рисунок 3.38 – Update Spectate Target

3.8 Реалізація режимів гри та додаткових функцій

Було реалізовано нескінченний хвильовий спавн ботів. Спочатку було додано на мапу декілька точок спавна та додано до них тег, після чого в GameMode створено подію для початку хвилі, яка викликається в події запуску рівня і яка, в свою чергу, викликає дві події – спавн хвилі (подано в додатку А, рис. 6-7) та очікування наступної хвилі після закінчення поточної, а також дві допоміжні функції для визначення кількості ботів у хвилі та кількості їх здоров'я. Також в ВР бота додано видалення об'єкта після смерті в події запуску Ragdoll-фізики. Таким чином було реалізовано перший режим гри – Endless Wave.

Створено об'єкт Enumeration, який містить назви всіх режимів гри. В GameMode створено функцію для встановлення налаштувань гри (вибраний режим, бесшовний перехід тощо), яка викликається в Begin Play. В HUD було відключено респаун гравця при грі в режимі Endless Wave.

Створено меню закінчення гри, в якому створено функцію для виведення кількості пережитих хвиль. В контролері змінено функцію виведення меню смерті шляхом додавання перевірки кількості померлих гравців (меню смерті видаляється, якщо всі гравці померли) та створено подію для виведення екрану завершення гри. В GameMode створено подію для відображення екрану завершення гри та переміщення гравця в лобі або меню (якщо це одиночна сесія), а також функцію, яка викликає дану подію, якщо всі гравці загинули.

Реалізовано можливість підбору гравцем зброї, яка знаходиться на мапі. Створено новий інтерфейс з чотирма функціями – взаємодія з актором, виведення назви зброї, встановлення видимості тексту та авто підбір. Також додано новий Input та об'єкт колізії (а також пресет для нього) в налаштуваннях проєкту.

В базовому персонажі додано об'єкт Sphere Collision для виявлення речей, які можна підібрати. Далі було створено події початку та кінця колізії для сфери та меша персонажа та подію взаємодії з актором. В HUD додано рамку з назвою предмета та створено функцію для відображення рамки та встановлення тексту; також в контролері створено подію для виклику даної функції.

Далі було створено структуру, яка містить інформацію про предмети, з якими гравець взаємодіє і таблицю для неї. Також створено простий віджет з одним текстовим полем (назва предмета) та базовий об'єкт для предметів, який містить статичний та скелетний меші, а також тригер у вигляді Sphere Collision і створений віджет. До об'єкта було додано потрібний інтерфейс та створено графі функцій інтерфейсу та граф Begin Play (рис. 3.39).

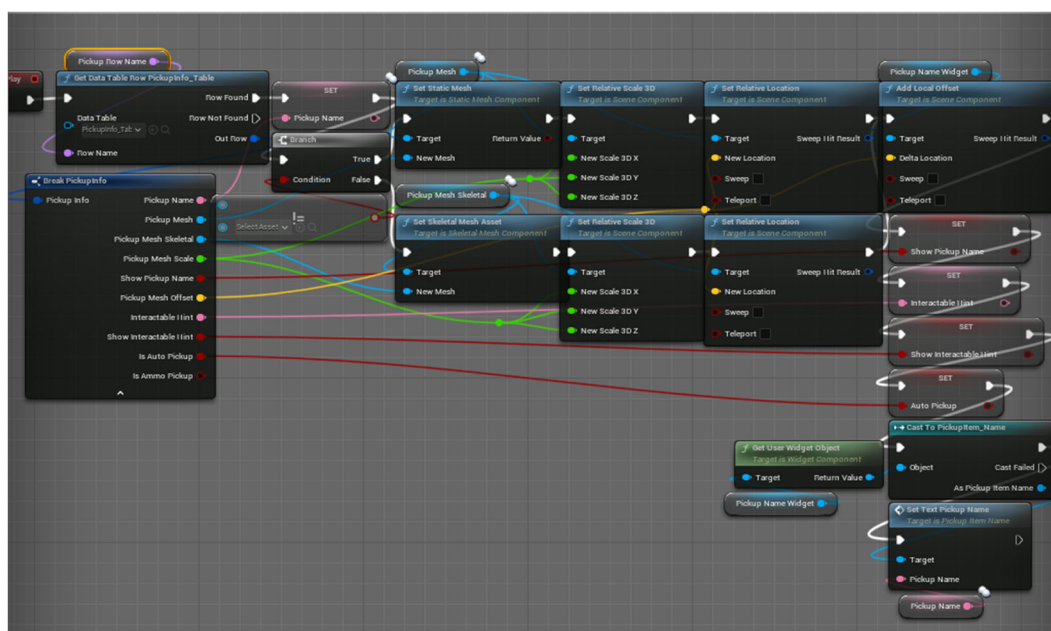


Рисунок 3.39 – Begin Play (BP_Pickup_Base)

Було створено дочірній об'єкт від базового – це об'єкт для підбирання зброї. В ньому створено граф функції інтерфейсу для взаємодії з актором, який додає

підібрану зброю в останній слот інвентарю, обирає його в грі та видаляє об'єкт для підбору.

В персонажі створено функцію для визначення можливості зміни зброї. Далі в таблицю предметів було додано новий запис – боєприпаси для гвинтівки. В базовому персонажі створено функцію, яка перевіряє інвентар гравця та визначає, чи може гравець підібрати зброю. В об'єкті зброї створено масштабну функцію Pick Up, яка перевіряє предмет, який було підібрано, та оновлює потрібні дані гравця, після чого змінено граф взаємодії шляхом його заміни на виклик даної функції.

Реалізовано зберігання профілю з використанням слотів. Спочатку було змінено завантаження та зберігання профілю в усіх об'єктах, використовуючи змінну для назви слота. Створено новий об'єкт Save Game, який містить ігрові налаштування, та створено змінну, яка містить активний слот профілю. В Game Instance створено масиви з усіма слотами зберігання та структурами даних профілю, а також функцію для завантаження налаштувань зі створеного об'єкта. Далі було внесено необхідні зміни в функцію Checked for Saved Info та створено функцію для завантаження всіх профілів і функцію для зміни профіля, яка змінює всі змінні, пов'язані з профілем гравця, записуючи в них значення профіля з обраним індексом. Функція завантаження налаштувань викликається в події Init, також було створено кастомну подію для створення нового профілю – вона викликається в віджеті при створенні нового профілю.

Створено віджет вибору профіля. Він має вигляд кнопки, яка містить ім'я гравця, його досвід, рівень та ранг. В головному меню додано нове меню вибору та створення профілю (усталено воно сховане та відкривається при натисканні на кнопку). Далі створено функцію для оновлення даних всіх профілів, відповідний функціонал кнопок та поля введення імені, а також функцію натискання кнопки (віджета) профілю.

Далі було реалізовано статус готовності гравця в лобі. В Game Mode меню додано масив булевих значень, які містять відомості про готовність гравців розпочати гру, а також кастомну подію для оновлення значень в масиві при зміні

статусу. В контролері меню створено події, які оновлюють статус гравців на сервері та в клієнтах.

У меню лобі додано текст, який позначає готовність гравця, розмістивши чотири екземпляри напроти імен гравців. Змінено подію натискання кнопки старту: якщо її натисне гравець-сервер – продовжуємо наявну логіку, якщо клієнт – викликатиметься нова функція Ready Button Pressed. Також створено функцію Update Players Ready, яка викликатиметься в події оновлення статусу гравця в клієнтах з контролера меню (рис. 3.40).

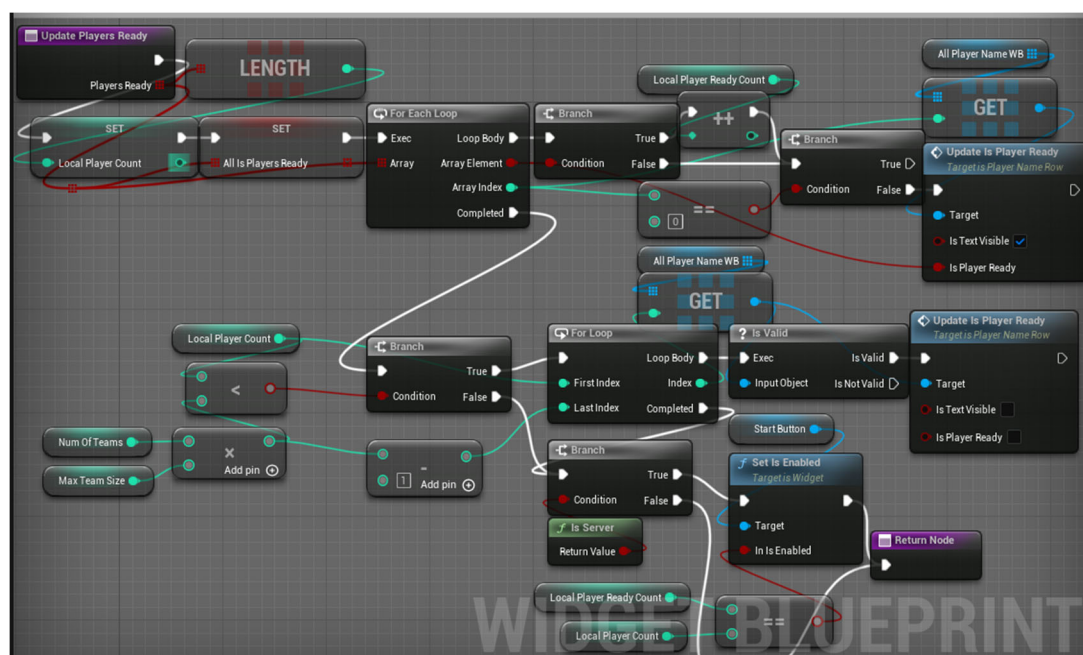


Рисунок 3.40 – Update Players Ready

Перед початком реалізації вибору режиму гри було додано перевірку на доступність введеного гравцем імені в меню створення профілю. В меню створено нову функцію, яка перевіряє введене ім'я на наявність недопустимих символів (наприклад, знаку оклику). Вона викликається в події зміни тексту в полі введення імені.

Далі було реалізовано вибір режиму гри. В лобі додано рамку з назвою режиму та кнопками для його вибору (кнопки активні лише для сервера, так як тільки він може змінювати режим гри). В меню лобі створено функцію для вибору режиму, яка змінює режим як у віджеті, так і в Game Instance (це необхідно для

функціонування гри), використовуючи створену просту подію з Game Mode меню, а також функцію для оновлення назви режиму в лобі – ця функція викликається з реплікованої події в контролері меню, яка, в свою чергу, викликається в події Update Lobby. Далі було створено події натискання кнопок зміни режиму, які змінюють обраний режим на попередній та наступний, зменшуючи чи збільшуючи індекс режиму відповідно.

Далі було створено стан гри (Game State). Стан гри використовується для запису та відслідковування подій, які пов'язані з певними правилами (наприклад, тривалість раунду) [4]. Створено об'єкт класу Game State та усталено встановлено станом гри в обох об'єктах Game Mode.

Змінено меню лобі для одиночної гри. Спочатку в Game Instance було створено кастомну подію для відкриття меню лобі. У віджеті меню лобі змінено події натискання кнопки старту та Construct шляхом додавання перевірки змінної Is Single (одиночна гра чи мультиплеєр) – при натисканні кнопки в одиночній грі викликається створена раніше подія (яка також викликається в головному меню при натисканні кнопки старту одиночної гри), а в Construct видаляються непотрібні частини меню (такі як імена інших гравців).

В ігровому інтерфейсі створено функцію Update Player Names для оновлення імен гравців. В Game State створено масив імен гравців та встановлено варіант RepNotify в опції Replication. Це потрібно для того, щоб створити функцію, яка викликатиметься як на сервері, так і в клієнтах щоразу, коли змінна оновлюється [11]. Дана функція викликає функцію оновлення імен гравців з ігрового інтерфейсу через контролер. Також створено функцію для оновлення імен в самому контролері, яка викликається при запуску рівня та виходу гравця з сесії.

В меню лобі додано новий компонент – чат, в якому гравці можуть спілкуватись, знаходячись в лобі. Створено функцію, яка створює повідомлення, додає його до чату та пролистує чат донизу (якщо повідомлення виходить за межі екрану). В контролері меню створено подію для оновлення чату, також в Game Mode меню створено функцію для відправки повідомлення, яка використовує

дану подію; функція викликатиметься в серверній події з контролера меню. В лобі викликається подія On Text Committed текстового поля чату, яка надсилатиме повідомлення (рис. 3.41).

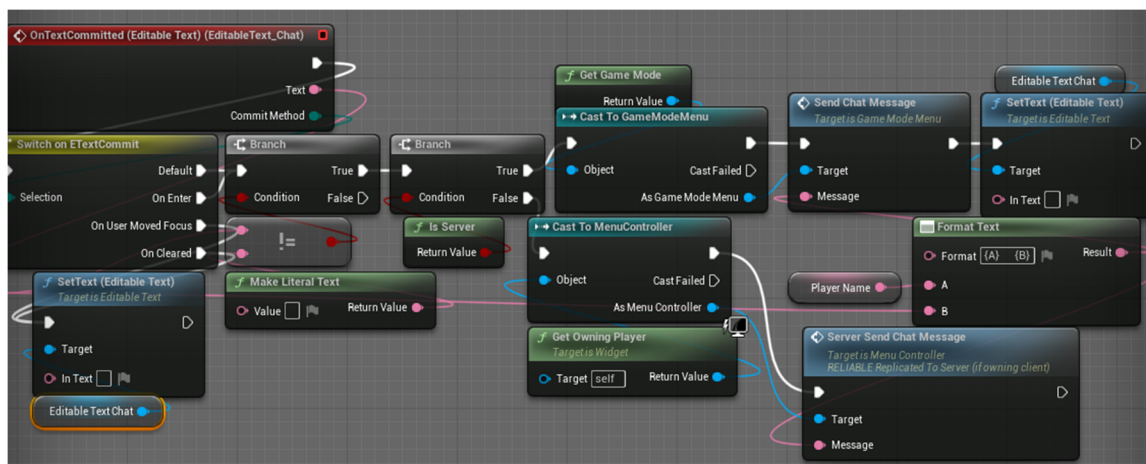


Рисунок 3.41 – Надсилання повідомлення в чат

Перший PvP режим, який було реалізовано – Free for All (всі проти всіх). Даний режим означає відсутність команд – кожен гравець грає один проти всіх інших гравців.

Назву цього режиму було додано в Enumerator, також створено структуру, яка містить три масиви (тривалість раунду, ліміт очок та час очікування респауна) і таблицю з даними обох режимів. В меню лобі додано кнопку переходу в меню налаштувань режиму гри та саме меню (усталено воно приховане), які будуть доступні лише серверу. В меню лобі створено функцію для оновлення встановлених налаштувань, яка викликається в функції вибору режиму гри. Також було створено граф натискання кнопки прийняття налаштувань (рис. 3.42). В Game Mode змінено функцію Set Game Options шляхом додавання нового режиму гри та запису налаштувань режиму в нові змінні.

В ігровому інтерфейсі додано таймер для відслідковування тривалості раунду. Створено функцію для оновлення таймеру, яка розбиває час до кінця раунду на хвилини та секунди. Далі в контролері було створено дві репліковані події для запуску гри та оновлення таймера.

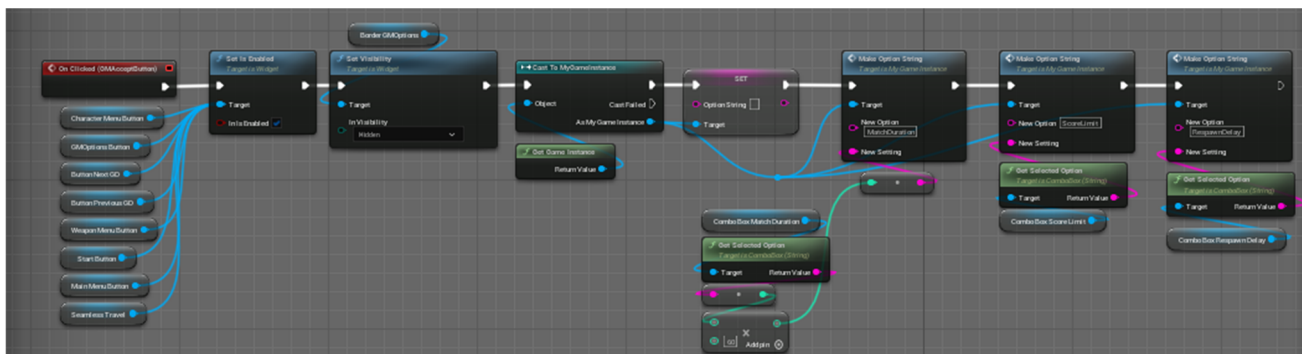


Рисунок 3.42 – Встановлення налаштувань режиму гри

В Game Mode створено подію для початку гри, подібну до події початку гри в режимі Endless Wave, яка викликається в Handle Starting New Player, якщо обрано режим Free for All. В Game State створено змінну, яка містить час, який залишився до закінчення раунду; реплікацію встановлено RepNotify. В функції оновлення змінної викликається функція оновлення з ігрового інтерфейсу. В Game Mode створено подію для зворотнього відліку, яка викликається в події старту рівня.

Також було реалізовано спавн (респаун) гравців на мапі в режимі FreeForAll та нанесення шкоди по гравцям. Спочатку було додано на мапу чотири нових точки спавна гравців. В теги точок спавна додано назву режиму, розділивши наявні точки спавна по режиму гри. В Game Mode створено функцію, яка визначає, які точки спавна можна використовувати в залежності від обраного режиму гри. Також створено подію для затримки точки респауна, після чого граф події Respawn After Dying було перетворено в функцію, також додавши в неї перевірку можливості використання точки респауна. В функції Server Process Hit додано перевірку на наявність інтерфейсу гравця та викликано функцію отримання шкоди в PvP режимі при його наявності.

Також було відключено режим спостерігання для режиму FreeForAll. Для цього в контролері змінено подію, яка визначає доступність режиму спостерігання. Створено дві булеві змінні для позначення доступності спостерігання та визначення режиму (мультиплеєрний чи ні), яким встановлено відповідні значення в залежності від обраного режиму гри.

В проєкт було додано таблицю лідерів з інформацією про кількість вбивств та смертей гравців. Спочатку в Player State в BeginPlay завантажується профіль гравця, з контролера береться структура з інформацією гравця та записується в змінну, після чого зі структури береться ім'я гравця, яке також записується в змінну (обидві змінні репліковані). В Game Mode створено функцію, яка при вбивстві гравця збільшує кількість вбивств в контролері та Player State для вбивці та аналогічно збільшує кількість смертей для вбитого, а також функцію для оновлення інформації в Game State. В базовому персонажі весь граф події отримання шкоди було перетворено в функцію. Створено функцію отримання шкоди в PvP-режимі шляхом дублювання та зміни функції отримання шкоди. В події отримання шкоди перевіряється, в якому режимі знаходиться гравець та викликається потрібна функція.

Створено нові віджети:

- рядок таблиці лідерів містить ім'я гравця, кількість його смертей та вбивств;
- таблиця лідерів містить назву режиму гри, таймер раунду, ліміт очок та таблицю, яка заповнюватиметься віджетами рядка.

Створено функцію для оновлення таблиці та подію Construct, яка заповнює всі необхідні поля відповідними значеннями. Далі в контролері було створено функцію для виведення/приховування таблиці та новий Input для виведення таблиці натисканням клавіші Tab.

В базовому персонажі подію, яка відповідає за смерть персонажа та виведення меню смерті, було розділено на дві: перша відтворює монтаж смерті, а друга виводить меню смерті після затримки. Виклик нової події було додано в обидві функції отримання шкоди.

Створено екран завершення гри при досягненні будь-яким гравцем необхідної кількості очок. У віджеті закінчення гри створено подію Construct, яка виводить відповідний текст в залежності від результату гри (переміг чи програв, досягнуто ліміт очок чи закінчився час раунду). В контролері створено декілька нових подій: для відключення керування та для виведення таблиці лідерів. В

Game Mode створено подію для завершення гри в PvP-режимі, подібну до відповідної події для Endless Wave. Також створено функцію для знаходження переможця шляхом пошуку максимальної кількості вбивств в масиві контролерів гравців. Викликається дана функція в події таймера після його закінчення та очищення, а також в функції обчислення кількості смертей та вбивств.

Наступний реалізований режим – командний смертельний бій (або Team Deathmatch). Суть режиму полягає в розбитті гравців на команди, в яких є одна ціль – знищити всіх гравців команди-суперника.

В налаштування режиму гри (в меню лобі) додано нову опцію – максимальний розмір команди. Далі було створено функцію, яка передає обрані налаштування в Game Instance та викликається при натисканні кнопки прийняття налаштувань. Також створено функцію для оновлення кількості гравців, яка множить кількість команд на кількість гравців в командах, вираховуючи таким чином загальну кількість гравців; дана функція викликається з контролера меню в події оновлення режиму гри.

Створено новий віджет – рядок з іменем гравця, який містить саме ім'я гравця та повідомлення про готовність. В ньому створено подію для оновлення повідомлення про готовність гравця. Далі було замінено рамку з імена гравців на нову, так як тепер для виведення імен гравців використовується новий віджет, також змінено функцію оновлення імен.

У віджет було додано нові елементи – опцію вибору кількості команд та кнопки для зміни команди. Змінено деякі події та функції, додавши передавання налаштувань для нового режиму. В функцію натискання кнопки готовності додано активацію/деактивацію кнопок вибору команди в залежності від готовності гравця. Далі було створено функцію, яка визначає видимість кнопок зміни команди в залежності від обраного режиму гри; викликається дана функція в функції оновлення кількості гравців. Також створено функцію для створення та заповнення списків команд, яка викликається в функції оновлення кількості гравців, і функцію оновлення ID команд. В Game Mode меню створено функцію для визначення можливості приєднання до обраної команди та подію для

встановлення ID обраної команди (рис. 3.43). В контролері меню створено серверну подію для оновлення ID команд, яка викликатиме створену подію.

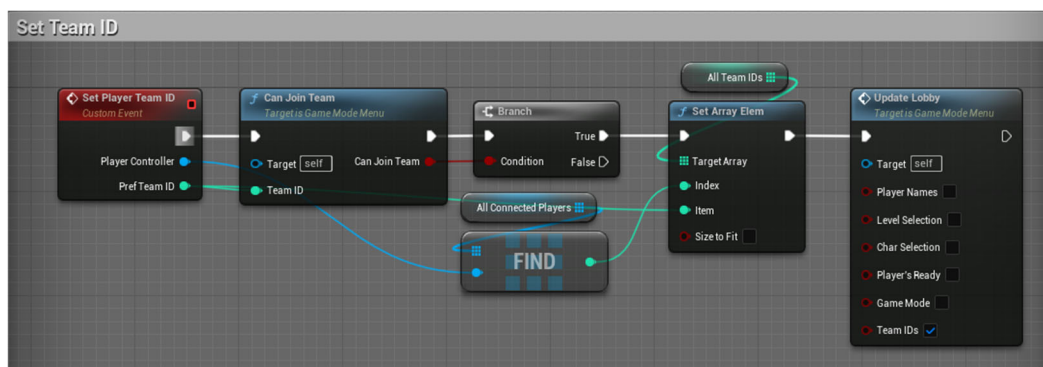


Рисунок 3.43 – Set Player Team ID

В GM меню повністю змінено подію вибору режиму гри, створивши для цього функцію, яка додаватиме гравців в команди, якщо це можливо.

Далі було створено функціонал для формування команд з ввімкеним автопідбором. Спочатку було створено функцію для формування команд, після чого змінено подію старту матчу (натискання кнопки старту), додавши формування команд та виведення в чат зворотнього відліку до початку гри. В контролері меню було продубльовано та змінено одну з функцій збереження, створивши таким чином функцію збереження ID команди; також створено подію, яка викликає дану функцію і яка, в свою чергу, викликається в новій функції збереження ID команд в Game Mode меню. Нова функція викликається в події початку гри.

В Game Mode створено функцію для спавна персонажів в режимі Team Deathmatch, яка по чергово спавнить гравців команд у відповідних точках спавна. Також змінено функцію респауна після смерті, додавши вибір точок спавна в залежності від обраної команди (для режиму Team Deathmatch). В базовому персонажі створено нові змінні в Begin Play – булеву змінну режиму TDM та ID команди гравця, після чого було змінено функції Server Process Hit та нанесення шкоди в PvP режимі, додавши перевірку ID команди – таким чином недоступний вогонь по своїм. В контролері змінено функцію оновлення цілей для

спостерігання, додавши перевірку ID команди для режиму TDM – таким чином гравець може спостерігати за гравцями лише своєї команди.

У віджеті закінчення гри змінено подію Construct, додавши розгалуження для виведення потрібного тексту в залежності від режиму гри. В Game Mode змінено функцію пошуку переможця (додавши розгалуження та повернення гравця (FFA) або команди (TDM) переможця), а також події виведення екрану закінчення гри та функцію вбивства гравця в PvP режимі. Також додано нарахування досвіду за вбивство гравців в PvP режимах в функції вбивства гравця.

Далі було додано дві нових дії – стрибок та присід. Для стрибка в базовому персонажі при натисканні клавіші викликається подія Jump та подія Stop Jumping при відпусканні клавіші (обидві події є вбудованими). Також змінено AVR, додавши програвання потрібних анімацій. Для присіду в базовому персонажі створено події для початку та закінчення присіду, які встановлюють швидкість руху персонажа та відтворюють потрібні монтажні в залежності від поточного стану. Далі було створено Blend Space скелета персонажа для присіду, після чого в AVR додано змішування поз для переміщення персонажа, а також перевірку на присід для програвання потрібних монтажів в функцію програвання монтажу повороту.

Далі було реалізовано ближній бій. Додавши нову дію в налаштуваннях проєкту, було змінено структуру з даними зброї та відповідну таблицю, додавши монтаж ближньої атаки. В базовому персонажі додано події проведення ближньої атаки та програвання монтажу.

Також в проєкті реалізовано допоміжне озброєння – гранати. Додавши дію метання гранати в налаштуваннях проєкту, було створено новий BP класу Actor, в якому додано три компонента: статичний меш, капсульну колізію та рух снаряда. В створеному об'єкті додано такі події:

- подія влучення колізії, яка зупиняє рух гранати та приписує гранаті шкоду в деякому радіусі;

– подія зупинки руху, яка відтворює ефекти та звук, після чого видаляє об'єкт.

Для позначення місця кидка гранати використовується ефект з безкоштовного пакету асетів Infinity Blade Effects, доступного для завантаження в Epic Games Store. Було створено та налаштовано для роботи зі сплайн-мешем матеріал, який використовується для позначення місця кидка. В базовому персонажі додано новий компонент Scene Root – маркер кидання гранати. В графі створено три нові події: для початку, закінчення та оновлення трасування кидання гранати, а також дві події (серверну та репліковану) для програвання монтажу кидання гранати. Далі було створено події кидання гранати, які створюють гранату та проводять трасування її кидка, для яких створено змінну ID команди в ВР гранати (для визначення союзників), також створено подію нанесення радіальної шкоди, яка викликає функцію отримання шкоди в PvP режимі.

В ВР бота створено подію Radial Damage, яка наносить шкоду боту та додає гравцю досвід за вбивство бота. Також в базовому персонажі додано перевірку декількох умов в події кидання гранати (для того, щоб гравець не міг кинути гранату під час перезаряджання, зміни зброї та ривка).

Далі в ігровий інтерфейс (ІІ) було додано кількість гранат. В графі створено функцію для оновлення тексту з кількістю гранат. В базовому персонажі створено подію для оновлення ІІ, а також функцію, яка визначає можливість кидка гранати, зменшує кількість гранат та оновлює ІІ (якщо можливо).

В структуру налаштувань режиму гри було додано дві булеві змінні для визначення типу сесії (одиначна чи мультиплеєр) та оновлено таблицю з даними режимів гри. В меню лобі створено нову функцію для фільтрування режимів гри, яка викликається в Construct (таким чином в грі, в залежності від типу сесії, будуть доступні обрані режими). Подібним чином змінено структуру та таблицю з даними мап, вказавши режими гри, для яких вони доступні.

Далі було додано новий рівень шаблону Open World, який буде доступний лише для нового режиму гри. Дані про неї додано в таблицю, також додано в Enumerator новий режим гри – Survival (виживання). Змінено всі фреймворки,

переглянувши наявні функції, пов'язані з вибором режиму гри, та додавши до них новий режим.

Було також реалізовано можливість відновлювати НР персонажа. Це відбувається за допомогою бонусів, які підбираються на рівні. Створено дочірній клас базового класу об'єкта для підбору, до нього додано систему частинок. В графі викликається функція інтерфейсу для взаємодії об'єкта з гравцем, яка збільшує здоров'я гравця на задану при розташуванні на мапі об'єкта кількість (якщо здоров'я гравця не повне; рис. 3.44).

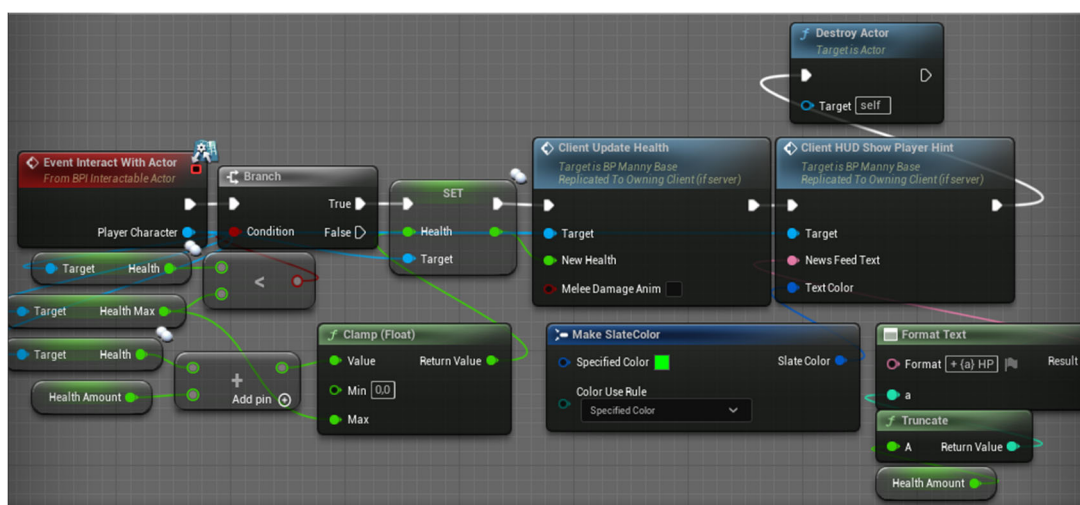


Рисунок 3.44 – Лікування гравця

Бонуси можуть лікувати гравця на різну кулькуість НР, тож для кращого розуміння приблизної кількості відновлюваного НР бонуси мають різне масштабування в залежності від кількості відновлюваного НР.

3.9 Висновки до розділу 3

В даному розділі було реалізовано всі функціональні та дизайнерські вимоги проєкту.

Було реалізовано базовий функціонал шутера від третьої особи, а саме: переміщення персонажа, стрільба, прицілювання, нанесення шкоди супротивникам. Створено функціонуюче меню налаштувань, за допомогою якого можна налаштувати графічні налаштування гри.

Реалізовано мультиплеєрну частину, яка складається з: створення власних ігрових сесій (як одиночних, так і мультиплеєрних), приєднання до вже існуючих сесій, створених іншими гравцями, налаштування власноруч створених сесій (вибір режиму гри, ігрової мапи, тривалості матчу, затримки респауна тощо).

Створено зрозумілий інтерфейс користувача, який містить елементи для відображення всіх необхідних для ігрового процесу функцій (кількість патронів, здоров'я, наявна зброя і т.п.). Реалізовано вибір зброї та персонажа, а також його кастомізація.

Реалізовано базовий ігровий штучний інтелект ботів для забезпечення базового функціоналу – переміщення бота по ігровій мапі, пошук цілі для атаки, атака гравця, а також смерть від рук гравця. Також реалізовано смерть персонажа (як від рук бота, так і від рук інших гравців-супротивників) та його переродження в спеціальних точках на мапі, яке залежить від обраного режиму гри.

В проєкті було реалізовано декілька режимів гри, як командних (team death match та endless wave), так і одиночних (free for all та survival), а також додаткові функції, притаманні обраному жанру (такі як лікування персонажа, підбір зброї та набоїв на ігровому рівні).

ВИСНОВКИ

У кваліфікаційній роботі бакалавра розроблено проєкт комп'ютерної багатокористувальницької гри в жанрі “third person shooter” з використанням ігрового рушія Unreal Engine.

При розгляді специфіки розробки ігрового проєкту встановлено основні етапи розробки багатокористувальницького шутера від третьої особи. Проведено SWOT-аналіз і розглянуто переваги та недоліки популярних представників вибраного жанру. Аналіз наявних аналогів дозволив встановити, що створення мультиплеєрного ігрового проєкту в жанрі шутера від третьої особи є доцільним.

Визначено та розроблено функціональні вимоги до проєкту. Під час вибору засобів розробки проаналізовано три найпоширеніших рушія для реалізації подібного проєкту, розглянуто та порівняно переваги та недоліки рушіїв, вибрано Unreal Engine як основний інструмент для реалізації проєкту. Створено маркетингову стратегію розвитку та поширення проєкту.

Було спроектовано програмний застосунок. Визначено архітектуру проєкту, яка складається головного меню, меню налаштувань гри, меню мультиплеєра, ігрового лобі, екрану завантаження та ігрового рівня, а також додаткових елементів. Змодельовано програмну систему проєкту: розглянуто основний та додатковий функціонал, складено ієрархію складових проєкту та методи доступу гравця до них. Розглянуто поведінку деяких основних функцій проєкту.

Проєкт створено засобами мови програмування C++ та сучасних інструментів створення відеоігор, які надає високотехнологічний ігровий рушія Unreal Engine 5. Було реалізовано базовий функціонал шутера від третьої особи. Створено функціонуюче меню налаштувань, за допомогою якого можна налаштувати графічні налаштування гри.

Реалізовано мультиплеєрну частину, яка відповідає за забезпечення стабільного ігрового процесу з підтримкою до 20 гравців (в залежності від режиму гри) в рамках одної сесії.

Створено зрозумілий інтерфейс користувача, який містить всі елементи для відображення необхідних для ігрового процесу функцій. Реалізовано вибір зброї та персонажа, а також його кастомізація.

Реалізовано базовий ігровий штучний інтелект ботів для забезпечення базового функціоналу. Також реалізовано смерть персонажа (як від рук бота, так і від рук інших гравців-супротивників) та його переродження в спеціальних точках на мапі.

Реалізовано декілька режимів гри, як командних (team death match та endless wave), так і одиночних (free for all та survival), а також додаткові функції, притаманні обраному жанру.

Розроблений проєкт при правильному та масштабному доопрацюванні складе конкуренцію відомим та популярним представникам даного жанру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ігровий рушій – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Ігровий_рушій
2. Unreal Engine 5.4 Documentation. URL: <https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-4-documentation>
3. Casting Quick Start Guide | Unreal Engine 4.27 Documentation. URL: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ActorCommunication/CastingQuickStart/>
4. Game Mode and Game State | Unreal Engine 4.26 Documentation. URL: <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/GameMode/>
5. UGameInstance | Unreal Engine Documentation. URL: <https://docs.unrealengine.com/4.27/en-US/API/Runtime/Engine/Engine/UGameInstance/>
6. Event Dispatcher | Unreal Engine Documentation. URL: <https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/UserGuide/EventDispatcher/>
7. A Player State | Unreal Engine Documentation. URL: <https://docs.unrealengine.com/4.27/en-US/API/Runtime/Engine/GameFramework/APlayerState/>
8. Map | Unreal Engine Documentation. URL: <https://docs.unrealengine.com/4.26/en-US/BlueprintAPI/Utilities/Map/>
9. AAI Controller | Unreal Engine Documentation. URL: <https://docs.unrealengine.com/4.27/en-US/API/Runtime/AIModule/AIController/>
10. Трофименко О. Г., Шиманський Є.І. Впровадження методів машинного навчання у розроблення ігрового бота. *Інфокомунікації – сучасність та майбутнє*: матер. IX міжнар. наук.-практ. конф. (12–15 листопада 2019 р.). Одеса: ОНАЗ ім. О.С. Попова. С. 376–378. ISBN 978-617-582-067-4
11. Replicating Variables in Blueprints | Unreal Engine Documentation. URL: <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Networking/Actors/ReplicatingVariables/>

ДОДАТКИ

Додаток А. Приклади Blueprint-функцій проєкту

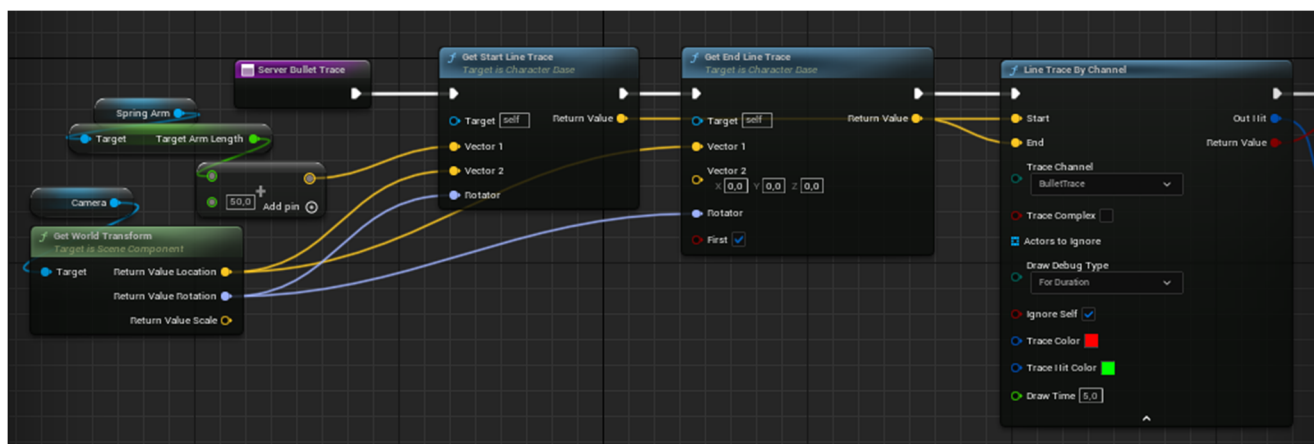


Рисунок А.1 – Функція Server Bullet Trace (частина 1)

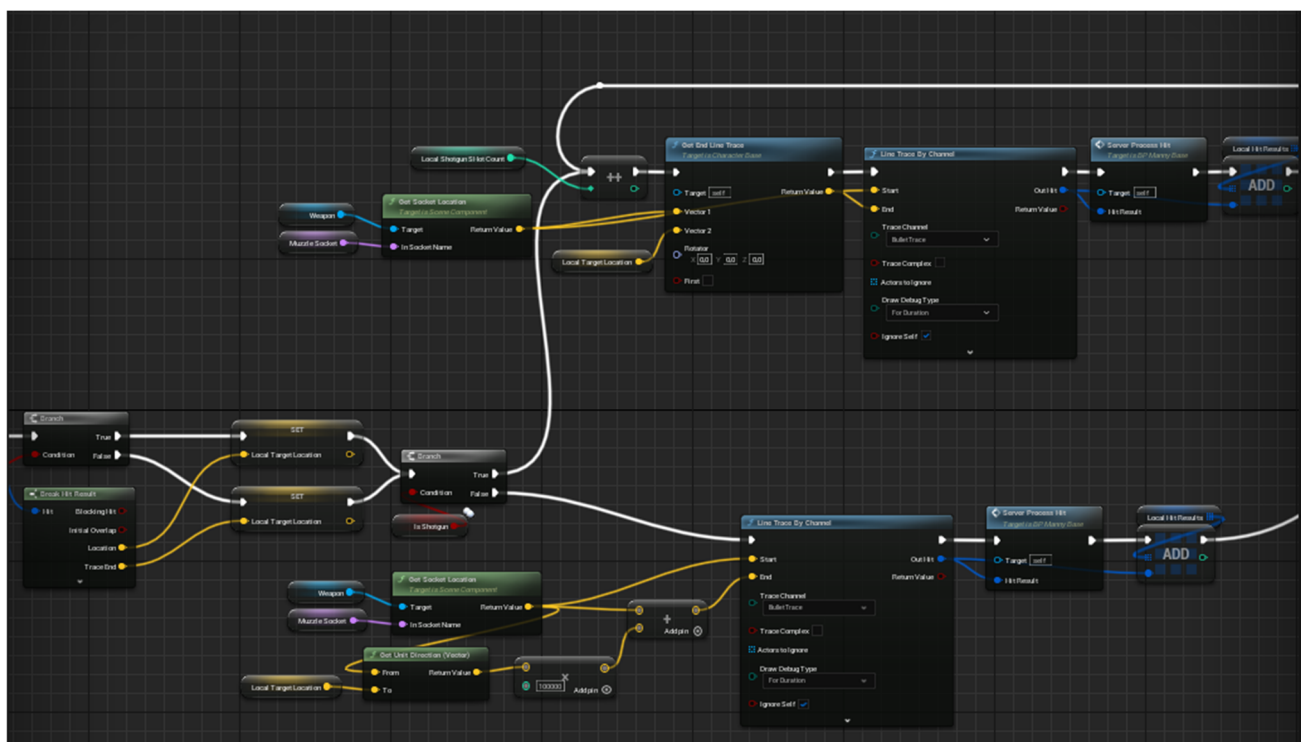


Рисунок А.2 – Функція *Server Bullet Trace* (частина 2)

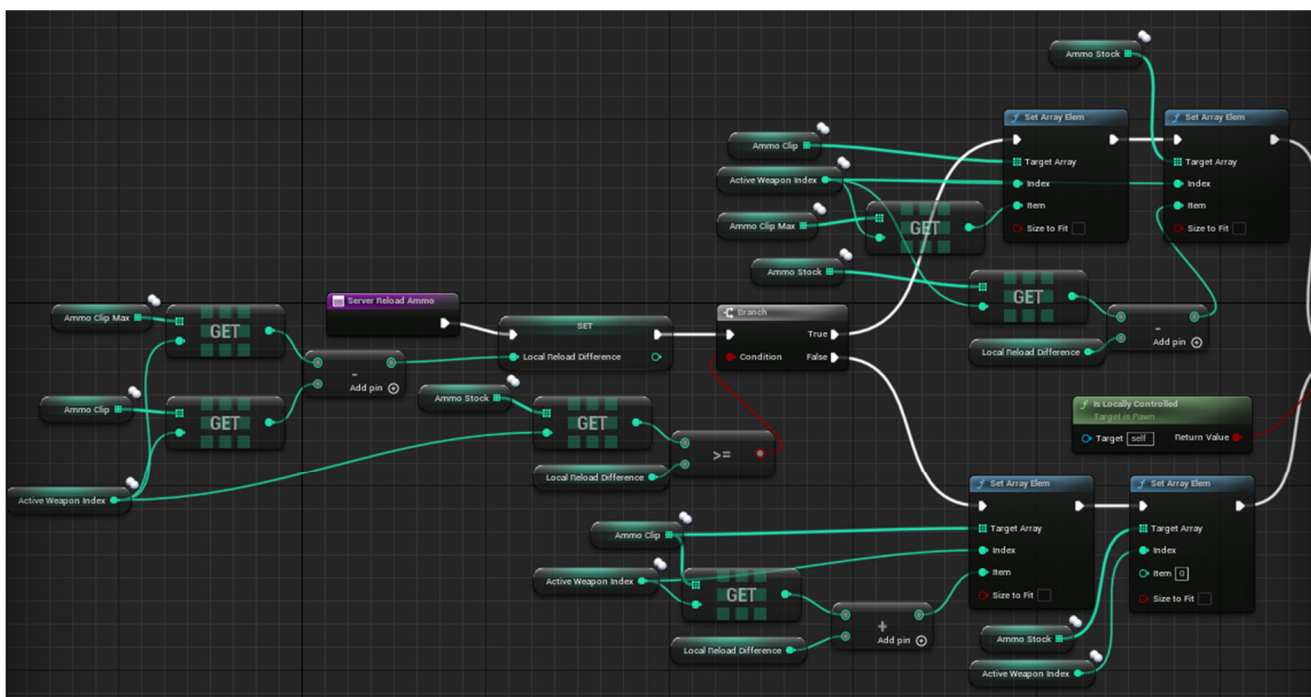


Рисунок А.3 – Функція Server Reload Аммо (частина 1)

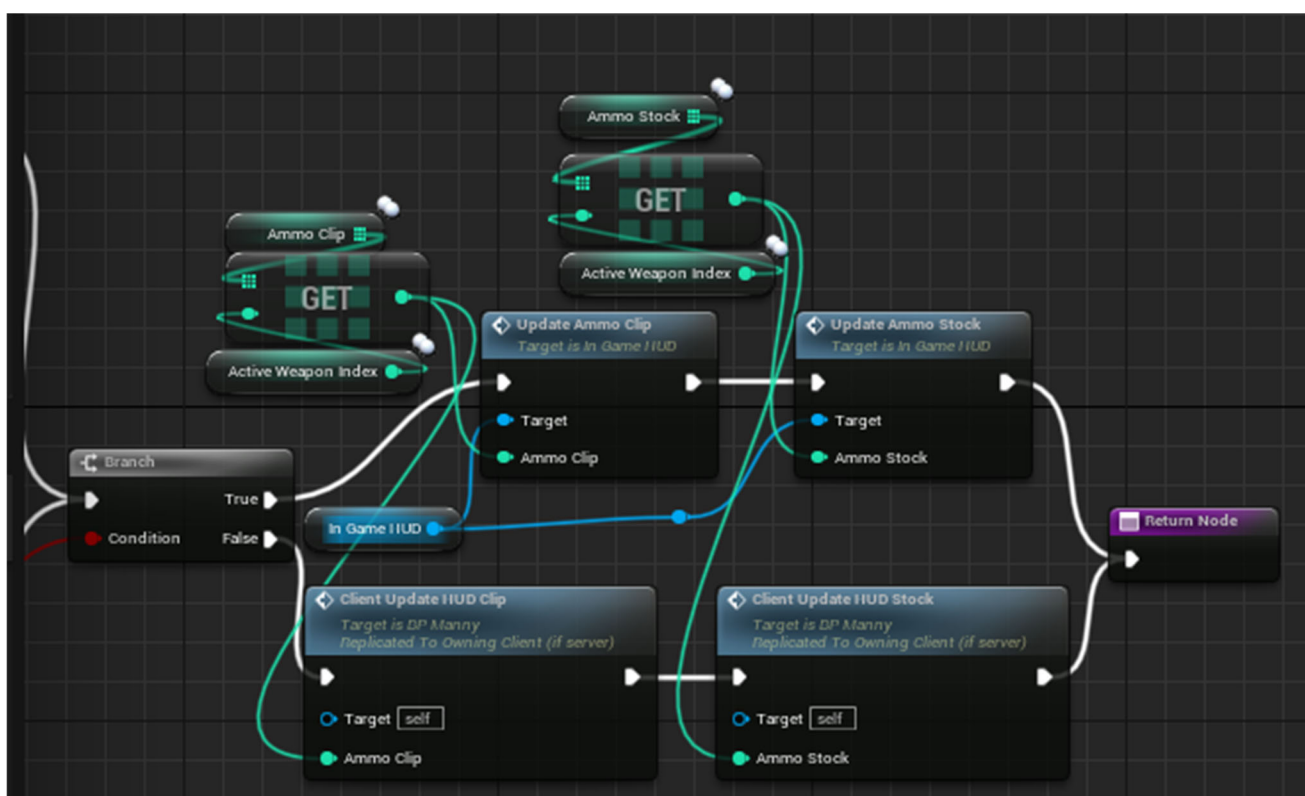


Рисунок А.4 – Функція Server Reload Аммо (частина 2)

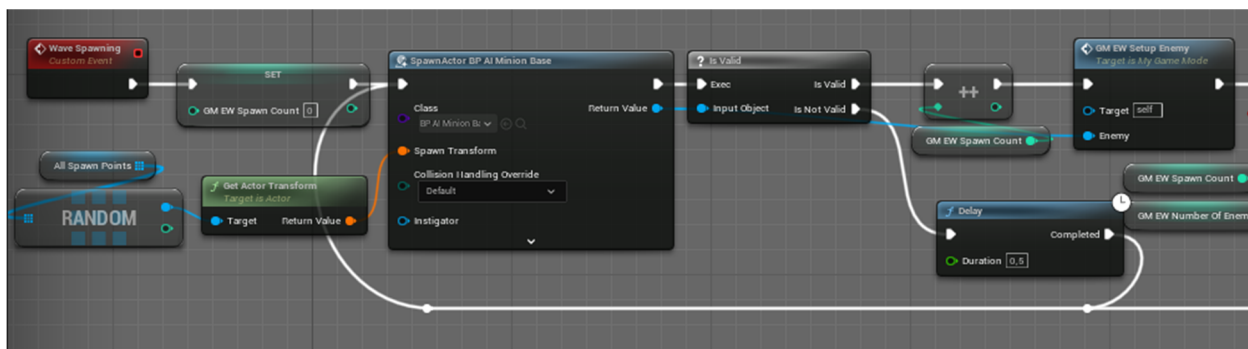


Рисунок А.5 – Подія Wave Spawning (частина 1)

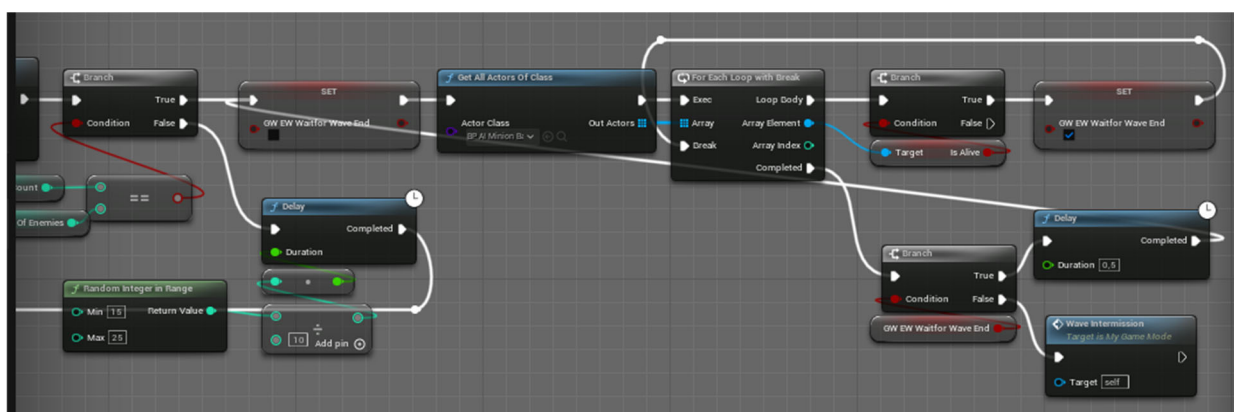


Рисунок А.6 – Подія Wave Spawning (частина 2)

Додаток Б. Лістинг програмного коду вихідного файлу та файлу заголовку класу Character

Файл .h:

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "CharacterBase.generated.h"

UCLASS()
class DIPLOMA_SHOOTER_API ACharacterBase : public ACharacter
{
    GENERATED_BODY()

public:
    // Sets default values for this character's properties
    ACharacterBase();

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    // Called to bind functionality to input
    virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override;

    UFUNCTION(BlueprintCallable)
    void AddDashImpulse(FName Moving);

    UFUNCTION(BlueprintCallable)
    void StartRagdoll();

    UFUNCTION(BlueprintCallable)
    float GetNewHealth(float Health, float Damage);

    UFUNCTION(BlueprintCallable)
    float GetDamageYaw(FVector Attacker);

    UFUNCTION(BlueprintCallable)
    FName GetDirection(float DamageYaw);

    UFUNCTION(BlueprintCallable)
    FVector GetStartLineTrace(FVector Vector1, FVector Vector2,
FRotator Rotator);
```

```

        UFUNCTION(BlueprintCallable)
        FVector GetEndLineTrace(FVector Vector1, FVector Vector2,
        FRotator Rotator, bool First);

        UFUNCTION(BlueprintCallable)
        FVector GetEndTraceBox(USkeletalMeshComponent* meshchar,
        UCameraComponent* camerachar);

        UFUNCTION(BlueprintCallable)
        FVector GetGrenadeVelocity(UCameraComponent* cam, int
        VelocityFW, int VelocityUW);

};

```

Файл .cpp:

```

#include "CharacterBase.h"
#include "GameFramework/CharacterMovementComponent.h"
#include "Engine/World.h"
#include "Components/CapsuleComponent.h"
#include "Kismet/KismetMathLibrary.h"
#include "Math/UnrealMath.h"
#include "Components/SceneComponent.h"
#include "Camera/CameraComponent.h"

// Sets default values
ACharacterBase::ACharacterBase()
{
    // Set this character to call Tick() every frame.  You can turn
    this off to improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;
}

// Called when the game starts or when spawned
void ACharacterBase::BeginPlay()
{
    Super::BeginPlay();
}

// Called every frame
void ACharacterBase::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
}

// Called to bind functionality to input

```

```

void ACharacterBase::SetupPlayerInputComponent(UInputComponent*
PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);
}

void ACharacterBase::AddDashImpulse(FName Moving)
{
    if (Moving == "None" || Moving == "Forward") {this-
>GetCharacterMovement()-
>AddImpulse((GetActorForwardVector()*1000)+(0,0,250));}
    else if (Moving == "Backward") { this->GetCharacterMovement()-
>AddImpulse((GetActorForwardVector() * (-1000))+(0,0,250)); }
    else if (Moving == "Left") { this->GetCharacterMovement()-
>AddImpulse((GetActorRightVector() * (-1000)) + (0, 0, 250));}
    else if (Moving == "Right") { this->GetCharacterMovement()-
>AddImpulse((GetActorRightVector() * 1000) + (0, 0, 250)); }
}

void ACharacterBase::StartRagdoll()
{
    this->GetCapsuleComponent()-
>SetCollisionEnabled(ECollisionEnabled::NoCollision);
    this->GetMesh()->SetCollisionProfileName("Ragdoll", true);
    this->GetMesh()->SetAllBodiesSimulatePhysics(true);
    this->GetMesh()->AddImpulse(this->GetCharacterMovement()-
>GetLastUpdateVelocity() + (this->GetCharacterMovement()-
>GetLastUpdateVelocity() * (150, 150, 150)));
}

float ACharacterBase::GetNewHealth(float Health, float Damage)
{
    float Result = Health - Damage;
    return FMath::Clamp(Result, 0.0f, 1000.0f);
}

float ACharacterBase::GetDamageYaw(FVector Attacker)
{
    float AttackerYaw =
UKismetMathLibrary::MakeRotFromZ(UKismetMathLibrary::GetDirectionUni
tVector(Attacker, this->GetActorLocation())).Yaw;
    float Yaw = this->GetActorRotation().Yaw - 90;
    return UKismetMathLibrary::NormalizeAxis(AttackerYaw - Yaw);
}

FName ACharacterBase::GetDirection(float DamageYaw)
{
    if (DamageYaw < 0)
    {
        if (DamageYaw >= -135 && DamageYaw < -45) { return "Left";
    }
        else

```

```

        {
            if (DamageYaw >= -45) { return "Front"; }
            else { return "Back"; }
        }
    }
else
{
    if (DamageYaw > 45 && DamageYaw <= 135) { return "Right"; }

    else
    {
        if (DamageYaw <= 45) { return "Front"; }
        else { return "Back"; }
    }
}
}

FVector ACharacterBase::GetStartLineTrace(FVector Vector1, FVector
Vector2, FRotator Rotator)
{
    return Vector2 +
    (UKismetMathLibrary::GetForwardVector(Rotator)*Vector1);
}

FVector ACharacterBase::GetEndLineTrace(FVector Vector1, FVector
Vector2, FRotator Rotator, bool First)
{
    if (First == true) { return Vector1 +
    (UKismetMathLibrary::GetForwardVector(Rotator) * 100000); }
    else { return Vector1 +
    (UKismetMathLibrary::RandomUnitVectorInConeInDegrees(UKismetMathLibr
ary::GetDirectionUnitVector(Vector1, Vector2), 3.0f)*100000); }
}

FVector ACharacterBase::GetEndTraceBox(USkeletalMeshComponent*
meshchar, UCameraComponent* camerachar)
{
    return meshchar->GetSocketLocation("upperarm_r") + (camerachar-
>GetForwardVector()*100);
}

FVector ACharacterBase::GetGrenadeVelocity(UCameraComponent* cam,
int VelocityFW, int VelocityUW)
{
    return (cam->GetForwardVector() * VelocityFW) + (cam-
>GetUpVector()*VelocityUW);
}

```