МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н.Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему Розробка комп'ютерної гри на рушії Unreal Engine 5 з використанням штучного інтелекту

Виконав: студентка 4 курсу, групи МФ-42
спеціальність 122 «Комп’ютерні науки»
освітньо-професійна програма «Інформатика»
Седюк Є. В.

Керівник: Меняйлов Є. С.

Харків – 2023 року

ЗМІСТ

1. ВСТУП

1.1. Формулювання мети роботи, задач та обґрунтування актуальності теми

Метою даної роботи є розробка комп'ютерної гри на рушії Unreal Engine 5 з використанням штучного інтелекту. Основною задачею є створення інтелектуальної системи, яка забезпечить взаємодію між гравцем та ігровим світом, здатна здійснювати прийняття рішень на основі зібраної інформації, та динамічно адаптуватися до дій гравця.

Актуальність теми полягає в тому, що комп'ютерні ігри з штучним інтелектом є однією з найбільш перспективних галузей в галузі інформаційних технологій, оскільки дозволяють реалізувати складні алгоритми та взаємодію між гравцем та ігровим світом, що забезпечує більш інтенсивний та захоплюючий геймплей. Unreal Engine 5, з своїми потужними можливостями для розробки ігор, є ідеальним вибором для реалізації даного проекту.

Задачі розробки комп'ютерної гри на рушії Unreal Engine 5 з використанням штучного інтелекту можуть бути такими:

- Визначити тип гри (головоломка, платформер, RPG, пригодницька гра тощо).
- Розробити графічні ресурси, включаючи архітектуру рівнів, 3D-моделі персонажів та предметів, текстури, анімації тощо.
- Розробити механіку гри, включаючи управління персонажами, взаємодію з ігровим світом, ворожнечу між персонажами тощо.
- Розробити систему штучного інтелекту, яка відповідатиме за поведінку комп'ютерних персонажів у грі.
- Розробити інтерфейс користувача, включаючи меню, інформаційні панелі, кнопки тощо.

- Перевірити функціональність гри, включаючи тестування механіки, штучного інтелекту та інтерфейсу користувача.
- Виправити помилки та недоліки гри, які були виявлені під час тестування.
- Оптимізувати гру для запуску на різних платформах, включаючи ПК та ігрові консолі.

Крім того, можна також розробити додаткові функції, які доповнять і покращать гру, наприклад, мультиплеєр, мережеву гру, підтримку різних мов тощо.

Розробка комп'ютерної гри з штучним інтелектом також має практичне значення, оскільки її можна використовувати для навчання та тренування майбутніх фахівців в галузі інформаційних технологій, а також як розважальний продукт для широкої аудиторії.

1.2. Стислий огляд відомих результатів в області дослідження

Останні версії Unreal Engine 5 надають розробникам ігор багато можливостей для створення реалістичних графічних ефектів та деталізації ігрового світу. Окрім того, Unreal Engine 5 підтримує високошвидкісне завантаження даних та можливість створювати масштабні ігрові світи з динамічною світлодіодною системою.

У галузі дослідження штучного інтелекту в ігровій розробці було досягнуто значних успіхів у використанні методів машинного навчання для покращення поведінки NPC (неприхильних персонажів) та управління групою NPC. Такі методи, як Q-навчання, дерева поведінки, глибинне навчання та навчання з підсиленням, використовуються для розв'язання завдань з покращення поведінки NPC, таких як навчання NPC рухатися в напрямку цілі, уникнення перешкод та взаємодії з гравцем.

Дослідження в області штучного інтелекту в ігровій розробці також зосереджені на розробці системи, яка забезпечує більш реалістичну

поведінку гравця. Наприклад, методи машинного навчання можуть бути використані для навчання NPC розпізнавати та відрізняти різні види предметів та поводитися відповідно.

У цілому, область дослідження штучного інтелекту в ігровій розробці є активною та продуктивною, а використання Unreal Engine 5 може значно полегшити процес розробки та покращити графічну якість та інші аспекти гри.

1.3. Відомості про одержані результати та їх новизна.

На сьогоднішній день, розробка комп'ютерних ігор на рушії Unreal Engine 5 з використанням штучного інтелекту є досить актуальною та перспективною галуззю досліджень.

Деякі з найбільш відомих результатів в цій області включають розробку систем штучного інтелекту, які забезпечують більш реалістичне поведінку персонажів в грі, а також використання нейромереж для генерації вмісту, такого як текстури та анімація. Крім того, дослідження у галузі машинного навчання та глибинного навчання дозволили створити системи, які можуть навчатися взаємодіяти з гравцями, а також розпізнавати та вивчати їхні звички та поведінку, що дозволяє забезпечити більш індивідуальний та персоналізований геймплей.

Новизна результатів полягає в тому, що з кожним роком розвиток технологій дозволяє створювати все більш реалістичні та інноваційні ігри з використанням штучного інтелекту. Відкриття нових можливостей для створення більш складних та цікавих ігор стимулює розвиток технологій та досліджень у цій галузі.

На основі розроблених ігор на рушії Unreal Engine 5 з використанням штучного інтелекту, було отримано наступні результати:

- Реалізовано систему штучного інтелекту для некерованих персонажів в грі, що дозволяє їм виконувати різноманітні дії, такі як рух, атаку, уникнення ворожих атак тощо.
- Використано нові можливості рушія Unreal Engine 5, зокрема високодеталізовану графіку з використанням технології Nanite, що дозволило створити візуально привабливу гру з деталізованими об'єктами та текстурами.
- Розроблено різноманітні локації в грі, які відображають різні світи та атмосфери.
- Відточено навички розробки іммерсивного геймплею, що передає гравцю атмосферу гри та робить його втягнутим у її світ.
- Розроблено механіку гри, що дозволяє гравцю взаємодіяти з персонажами, предметами та оточенням, що збільшує рівень іммерсії та реалізму гри.

Отримані результати є новизною в галузі розробки ігор на рушії Unreal Engine 5 з використанням штучного інтелекту. Розроблена гра містить в собі нові можливості технологій рушія Unreal Engine 5 та дозволяє гравцеві відчути іммерсивний світ гри з детальною графікою та різноманітними персонажами та локаціями. Реалізація системи штучного інтелекту для персонажів гри дозволяє їм діяти більш реалістично та взаємодіяти з гравцем, що збільшує ігровий досвід.

2. ОСНОВНА ЧАСТИНА

2.1. Постановка задачі

Розробка гри на движку може бути досить ємним завданням, включаючи великі можливості движка, тому необхідно побудувати чіткий план роботи якого необхідно дотримуватися. Ми розглянемо основні етапи розробки ти проектування.

- Проектування гри: В цьому етапі ми визначаємо концепцію гри, які ігрові механіки та елементи повинні бути в грі та як вони пов'язані між собою.

- Створення моделей: В цьому етапі ми створюємо 3D-моделі головних персонажів та ігрових об'єктів.

- Створення текстур та матеріалів: Наступним кроком є створення текстур та матеріалів для наших 3D-моделей. Ми використовуємо можливості рушія і його вбудованні матеріали.

- Розробка дерев поведінки: Використовуючи інструменти Unreal Engine 5, ми розробляємо дерева поведінки для наших персонажів. Ми створюємо складні поведінкові моделі для персонажів, які забезпечують реалістичне поведінку персонажів в грі.

- Розробка User Interface: На цьому етапі ми розробляємо інтерфейс гри, з яким буде взаємодіяти гравець для його зручності використання проекту.

- Програмування: Наступним кроком є програмування гри. Ми використовуємо мову програмування C++ для створення складних ігрових механік та реалізації логіки дерев поведінки.

- Тестування та налагодження: Фінальний етап розробки, у якому ми перевіряємо гру на можливі помилки та недоліки.

2.2 Розвинутий огляд сучасного стану справ в області

Ігровий рушій (англ. game engine) - це програмне забезпечення, що використовується для створення відеоігор та інтерактивних додатків. Рушій ігор забезпечує розробникам ігор готовий набір інструментів для відображення графіки, реалізації фізики, звуків та музики, а також обробки вхідних сигналів, управління персонажами, зберігання даних та інші можливості, які потрібні для створення відеоігор.

Ігрові рушії можуть бути написані на різних мовах програмування, таких як C++, C#, Java, Python, Lua та інші. Вибір мови програмування залежить від вимог до продуктивності, зручності розробки та інших факторів.

Наприклад, Unreal Engine використовує мову програмування C++, яка забезпечує високу продуктивність та дозволяє розробляти складні проекти з великою кількістю об'єктів та ефектів. Unity використовує мову програмування C#, яка є більш зручною для розробки, має менше обмежень та спрощує взаємодію з компонентами рушія.

Деякі ігрові рушії також підтримують сценарні мови, які дозволяють легко налаштовувати гру без потреби у глибокому розумінні мов програмування. Наприклад, Unity підтримує мову програмування Boo та скриптову мову JavaScript.

Крім того, розробники можуть створювати власні модулі та плагіни для ігрових рушіїв на мовах програмування, які їм зручні. Це дає можливість розширювати функціональність рушія та налаштовувати його під потреби розробника.

Отже, ігрові рушії можуть бути написані на різних мовах програмування, залежно від вимог до продуктивності та зручності розробки.

Використання мов програмування дозволяє розробникам створювати складні ігри та розширювати функціональність ігрового рушія.

Ігрові рушії зазвичай мають графічний інтерфейс та складаються зі складових частин, таких як рушій фізики, рушій штучного інтелекту, система відображення графіки, система звуку, система управління користувача та інші. Розробники використовують ці складові частини, щоб створити свої власні ігрові механіки та ефекти, що відображаються на екрані.

Ігрові рушії можуть бути розроблені як спеціальне програмне забезпечення для конкретної відеоігри, так і як готовий набір інструментів, що дозволяє розробникам створювати відеоігри різних жанрів та типів.

На початку історії розробки ігор, розробники часто писали свої власні рушії з нуля. Це було дуже складно та витратно у плані часу та ресурсів, адже створення рушія - це великий проект, який потребує високого рівня знань у програмуванні, графічному дизайні та інших суміжних областях.

З часом, з'явилися готові рушії, такі як Unity та Unreal Engine, які можна використовувати для створення ігор без необхідності писати свій власний рушій. Готові рушії забезпечують базовий функціонал, такий як роботу з графікою, фізикою, звуком та іншими елементами гри, що зменшує витрати часу та ресурсів на створення гри.

З появою готових ігрових рушіїв створення комп'ютерних ігор стало більш доступним і швидким процесом. Люди більше не повинні бути експертами в програмуванні рушіїв або мати досвід у розробці компіляторів, оптимізації пам'яті та інших складних технологій, що пов'язані з розробкою ігрових движків.

Готові рушії дозволяють командам розробників швидко розпочати роботу над проектом, використовуючи вже наявну інфраструктуру та

інструменти, з якими розробники можуть легко працювати. Крім того, готові рушії забезпечують стандартизацію у розробці ігор, що полегшує співпрацю між розробниками та знижує час, необхідний для створення гри.

Також використання готового рушія дає розробникам змогу зосередитися на креативних аспектах розробки гри, таких як геймплей, історія, дизайн рівнів тощо, замість того, щоб тратити час на розробку технічних деталей.

Усе це робить готові рушії незамінними інструментами у створенні сучасних комп'ютерних ігор, що дозволяє розробникам більш ефективно використовувати свій час та ресурси, щоб створити більш вражаючі та ексклюзивні ігрові досвіди.

Існує безліч різних рушіїв для створення комп'ютерних ігор, але деякі з них відомі своєю популярністю та широким спектром можливостей. Ось декілька найвідоміших з них:

- Unity - це один з найпопулярніших рушіїв для створення ігор, який використовується багатьма розробниками з усього світу. Unity пропонує зручний інтерфейс, велику кількість інструментів для розробки, а також підтримку різних платформ.

- Unreal Engine - це інший популярний рушій, який використовується для розробки відеоігор. Unreal Engine пропонує вражаючі графічні можливості, зручний інтерфейс та велику кількість функцій для розробки ігор.

- CryEngine - цей рушій відомий своїми графічними можливостями та підтримкою фізики. CryEngine використовують для створення ігор з великими відкритими світами та детальною графікою.

- GameMaker - цей рушій зазвичай використовують для створення 2D-ігор. Він пропонує зручний інтерфейс, легкий для вивчення та велику кількість інструментів для створення простих ігор.

- Godot - цей рушій відкритого коду пропонує безкоштовну розробку ігор з відкритими джерелами. Godot підтримує різні платформи та мови програмування.

Ці рушії є дуже популярними серед розробників ігор, але також є інші рушії, які також можуть бути корисними для розробки ігор в залежності від специфіки проекту.

Розглянемо два найпопулярніші рушії, які активно використовуються для створення ігор:

1. **Unity** - це популярний рушій для створення комп'ютерних ігор, який використовується розробниками з усього світу. Unity пропонує велику кількість інструментів та функцій для розробки ігор, що робить його дуже зручним для роботи з ним.

   Основні можливості Unity включають:

   Кросплатформна підтримка - Unity дозволяє створювати ігри для різних платформ, таких як Windows, macOS, Linux, iOS, Android та ін.

   Графічні можливості - Unity має потужний рушій графіки, який дозволяє створювати як 2D, так і 3D графіку. Рушій пропонує велику кількість готових рішень для розробки графіки, таких як різноманітні ефекти, освітлення, текстури та ін.

Фізика - Unity має вбудований фізичний рушій, який дозволяє створювати реалістичні фізичні ефекти в іграх. Рушій пропонує різноманітні інструменти для роботи з фізикою, такі як різні типи колізій, рух об'єктів та ін.

Анімація - Unity має потужний інструментарій для роботи з анімацією. Розробники можуть створювати різноманітні анімаційні ефекти, рухи персонажів тощо.

Мови програмування - Unity підтримує різні мови програмування, такі як C#, JavaScript та Boo. Це дозволяє розробникам використовувати свої вміння та знання для створення ігор.

Розвиваючийся екосистема - Unity має активну спільноту розробників та безліч різних ресурсів, таких як форуми, документація та безкоштовні пакети ресурсів, які можуть бути використані в іграх. Крім того, Unity постійно оновлюється та розвивається, що дозволяє розробникам створювати ігри з новітніми технологіями.

Мобільна розробка - Unity дуже популярний серед розробників мобільних ігор. Рушій має вбудовані інструменти для розробки мобільних ігор, такі як оптимізація для мобільних платформ та інструменти для роботи з різними датчиками.

Віртуальна реальність та доповнена реальність - Unity підтримує розробку ігор для віртуальної та доповненої реальності. Рушій має вбудовані інструменти для роботи з різними гарнітурами віртуальної реальності, такими як Oculus та HTC Vive.

Unity дозволяє створювати ігри різних жанрів, таких як стратегії, рольові ігри, екшн, гонки та інші. Рушій має велику кількість готових рішень для розробки ігор, що дозволяє розробникам сконцентруватися на розробці геймплею та інших важливих аспектах гри.

2. **Unreal Engine 5** - це остання версія відомого ігрового рушія, розробленого компанією Epic Games. UE5 вийшов у травні 2021 року і він має кілька значних покращень у порівнянні з попередніми версіями рушія.

Unreal Engine також має потужну підтримку віртуальної реальності, що дозволяє розробникам створювати ігри та додатки VR високої якості. Рушій підтримує різні мови програмування, зокрема C++, Blueprint та Python, що дозволяє розробникам вибирати той інструмент, який відповідає їхнім потребам.

Unreal Engine також активно використовується для розробки не тільки ігор, але і симуляторів, тренажерів, візуалізації архітектурних проектів та багатьох інших додатків. Завдяки своїм можливостям та гнучкості, Unreal Engine є потужним інструментом для розробки різноманітних додатків з відмінними графічними та звуковими ефектами.

Unreal Engine 5 має різноманітні можливості для реалізації штучного інтелекту (ШІ) у комп'ютерних іграх. Найбільш популярні підходи до реалізації ШІ в UE5 включають в себе використання поведінкових дерев, машини станів та нейронних мереж.

- Поведінкові дерева - це графічний спосіб визначення поведінки ігрових об'єктів в залежності від їх стану та навколишнього

середовища. В UE5 це можна реалізувати за допомогою вбудованого інструменту Behaviour Tree, який дає можливість описувати взаємодію між ігровими об'єктами в форматі графа.

- Машина станів - це ще один популярний підхід до реалізації ШІ, де кожен об'єкт може перебувати в різних станах, які залежать від певних умов. В UE5 для реалізації машини станів можна використовувати вбудований інструмент State Machine.

- Нейронні мережі - це різновид штучного інтелекту, який моделює роботу мозку, дозволяючи ігровим об'єктам "навчатися" і змінювати свої дії з часом. В UE5 реалізація нейронних мереж можлива за допомогою вбудованої підтримки TensorFlow та PyTorch, що дає можливість інтегрувати нейронні мережі в ігрові об'єкти та ефекти.

Крім цього, UE5 також підтримує різні алгоритми ШІ, такі як алгоритми пошуку шляху, алгоритми пошуку оптимальної стратегії та інші.

2.3 Опис моделі вимог до інформаційної системи

Для опису моделі вимог необхідно ввести і пояснити деякі терміни, які надалі будуть використовуватися у роботі:

- NPC (Non-Player Character) - це термін, що використовується у комп'ютерних іграх. NPC означає персонажів, які контролюються комп'ютером або штучним інтелектом, а не гравцем. NPC виконують різні ролі в грі, такі як союзники, супротивники та інші, та взаємодіють з гравцем або з іншими NPC відповідно до заданої логіки та поведінки.

- Дерево поведінки (Behavior Tree) - це інструмент штучного інтелекту в ігрових рушіях, який використовується для організації та управління поведінкою персонажів у іграх. Воно представляє собою ієрархічну структуру з вузлами, що визначають різні дії, стани та логіку взаємодії персонажа з оточуючим світом. Дерево поведінки надає зручний спосіб визначати, як персонаж повинен взаємодіяти з грою, робити рішення та керувати своїми діями.

Тож, перейдемо до опису моделі вимог до інформаційної системи:

Функціональні вимоги:

Реалістичне моделювання поведінки NPC: Штучний інтелект повинен забезпечувати NPC здатністю до розумного прийняття рішень, реагування на змінення умови гри та виконання відповідних дій.

Контроль NPC: Гравець повинен мати можливість взаємодіяти з NPC.
Персоналізація NPC: Штучний інтелект повинен дозволяти розробникам налаштовувати характеристики NPC, їхні вміння, поведінку та взаємодію з гравцем.

Вимоги до штучного інтелекту:

Планування поведінки: Штучний інтелект повинен мати здатність до планування своєї діяльності, враховуючи обмеження, цілі та контекст гри.

Розпізнавання середовища: Штучний інтелект повинен мати можливість аналізувати середовище гри, виявляти перешкоди, об'єкти та інших NPC, і враховувати цю інформацію при прийнятті рішень.

Адаптація до гравця: Штучний інтелект повинен здатися адаптуватися до стилю гри гравця, враховуючи його дії.

Вимоги до графіки та анімації:

Реалістична зовнішність: NPC повинні мати реалістичні моделі, текстури та анімації, щоб краще імітувати людську поведінку та виразність.

Здатність до взаємодії: NPC повинні мати анімаційні рухи, що дозволяють їм взаємодіяти з гравцем.

Вимоги до дерева поведінки:

Система повинна підтримувати розробку і редагування дерев поведінки для NPC.

Дерева поведінки повинні бути гнучкими і можливими до налаштування для відтворення різних видів поведінки.

Вузли та логіка дерева поведінки: Система повинна надавати широкий набір вузлів, таких як умови, дії, послідовності, вибори та інші, для реалізації різноманітних поведінок NPC. Дерева поведінки повинні підтримувати можливість вкладення та ієрархічної організації вузлів.

Параметризація та конфігурація: Система повинна дозволяти налаштовувати параметри вузлів та поведінки для досягнення різних результатів. Розробникам повинно бути забезпечено зручне

конфігурування дерев поведінки для різних NPC без необхідності великого обсягу кодування.

Взаємодія з гравцем: Дерева поведінки повинні дозволяти NPC взаємодіяти з гравцем, включаючи можливість розпізнавання дій гравця, виконання відповідних реакцій та взаємодії з інтерфейсом гри.

Управління пріоритетами та станами: Дерева поведінки повинні мати механізми для управління пріоритетами різних вузлів та станів, щоб забезпечити відповідність поведінки NPC змінюються умовам гри.

Перевірка та налагодження: Система повинна надавати зручні інструменти для перевірки та налагодження дерев поведінки, включаючи візуалізацію виконання та стеження за станами.

Модель вимог для інтерфейсу

Головне меню: Вимагається наявність інтуїтивно зрозумілого та зручного головного меню, де гравець може виконувати різні дії, такі як Start гри, вихід з гри тощо.

Ігрові панелі: Вимагається наявність ігрових панелей, які надають гравцю доступ до важливої інформації під час гри, такої як життя головного героя тощо.

Дизайн рівня:

Розміщення об'єктів: Вимагається визначення розміщення різних об'єктів у грі, таких як перешкоди, платформи, розміщення NPC тощо.

Баланс складності: Вимагається досягнення збалансованої складності рівня, щоб гравець мав виклик, але не відчував надмірної фрустрації.

2.4 Опис функціональності системи

Для розробки були використані такі інструменти та середовища розробки:

Unreal Editor - це робоче середовище, яке надається разом з рушієм Unreal Engine. Він призначений для розробки, налаштування та створення ігрових вмісту. Unreal Editor надає потужні інструменти, які допомагають розробникам створювати вражаючі графічні ігри та візуалізації. (Рис. 1)

Основні складові Unreal Editor включають:

- Вікна перегляду: Unreal Editor має різні вікна перегляду, такі як вікно сцени, вікно контенту, вікно матеріалів тощо. Ці вікна дозволяють розробникам переглядати, редагувати та взаємодіяти з різними аспектами гри, включаючи об'єкти сцени, текстури, анімації та інше.

- Панелі інструментів: Unreal Editor має різні панелі інструментів, які містять набори інструментів та опцій для редагування ігрового вмісту. Ці панелі надають широкий спектр функціональності, включаючи малювання сцени, розміщення об'єктів, створення матеріалів та багато іншого.

- Редактор сцен: Редактор сцен у Unreal Editor дозволяє розробникам створювати та редагувати віртуальні світи, додавати об'єкти, налаштовувати освітлення, фізику, колізії та інші аспекти сцени. Він надає потужні інструменти для створення динамічних та реалістичних ігрових оточень.

- Редактор матеріалів: Редактор матеріалів дозволяє розробникам створювати та редагувати матеріали, які використовуються для текстурування об'єктів у грі. Він надає широкий набір інструментів для створення складних шейдерів, текстурних ефектів та інших візуальних елементів.

- Навігатор ресурсів: Навігатор ресурсів дозволяє розробникам керувати та організовувати різні ресурси гри, такі як моделі, текстури, звуки, анімації та інше. Він допомагає знаходити, переглядати та імпортувати ресурси у проект.

(Рис. 1)

Blueprints - візуальна система скриптування, яка входить до складу рушія Unreal Engine. Вона дозволяє розробникам створювати логіку гри, взаємодію об'єктів та програмувати різні компоненти.

Blueprints використовує графічний інтерфейс (Рис. 2), де розробники можуть створювати схеми з вузлів, які представляють різні дії, умови та обробники подій. Кожен вузол має свою функціональність і взаємодіє з іншими вузлами, утворюючи послідовності дій та реакції в грі.

Основні переваги використання Blueprints включають:

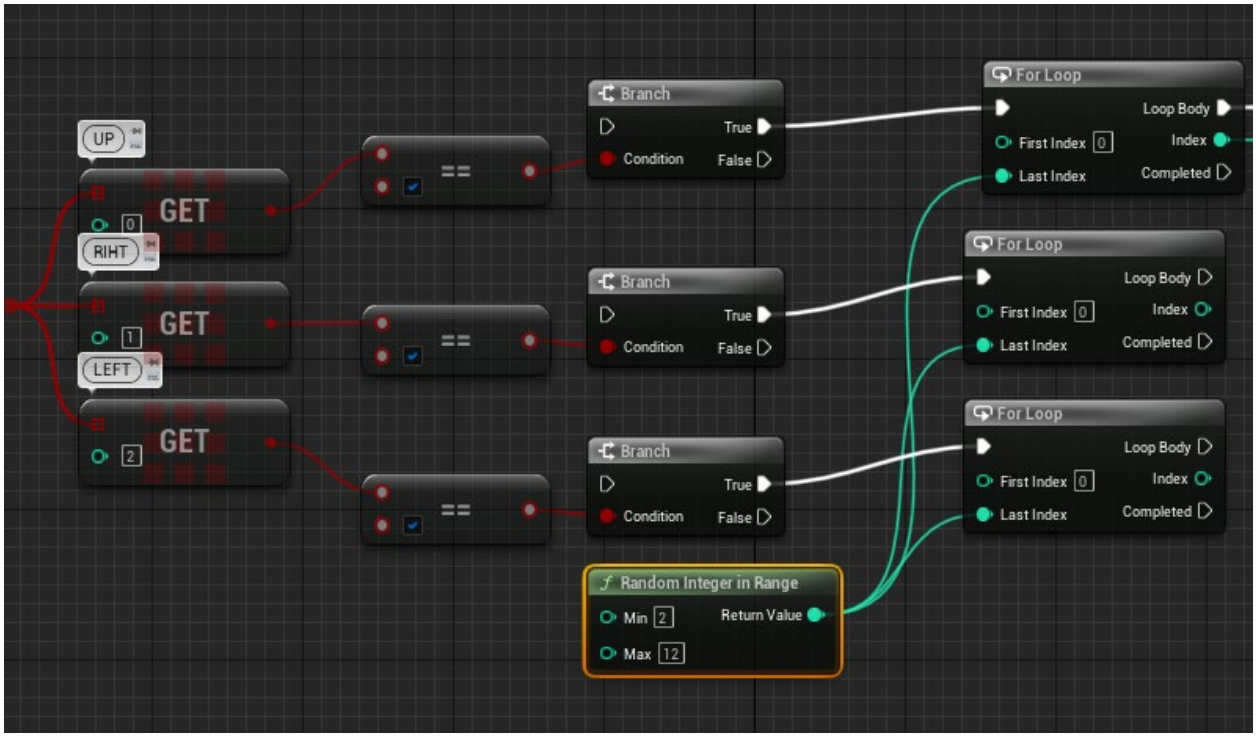
- Візуальний підхід: Blueprints дозволяють розробникам виразити свої ідеї та логіку за допомогою графічних елементів. Це зрозуміліше для багатьох розробників, які не мають досвіду у програмуванні.

- Швидкий прототипування: Завдяки графічному інтерфейсу, Blueprints дозволяють швидко створювати та змінювати функціональність гри без необхідності компіляції коду. Це полегшує прототипування та ітеративний процес розробки.

- Візуальна логіка: Blueprints надають інтуїтивний спосіб виразити складну логіку гри за допомогою зв'язаних вузлів. Розробники можуть створювати умови, цикли, обробники подій та інші компоненти, що дозволяють гнучко контролювати поведінку об'єктів у грі.

- Інтеграція з кодом: Blueprints можуть бути поєднані зі звичайним кодом на мові C++. Це дозволяє розробникам використовувати кращі

можливості Blueprints для визначення логіки гри, а при потребі використовувати мову C++ для оптимізації або додаткових функцій.



(Рис. 2)

Rider - це інтегроване середовище розробки (IDE), розроблене компанією JetBrains, яке спеціалізується на розробці програмного забезпечення. Rider призначений для роботи з різноманітними технологіями програмування, включаючи C#, C++, Unity і багато інших.

Основні особливості і можливості Rider включають:

- Мультиплатформеність: Rider підтримує роботу на різних операційних системах, таких як Windows, macOS та Linux. Це дозволяє розробникам використовувати свою улюблену IDE на платформі свого вибору.

- Розумне автодоповнення: Rider надає потужне автодоповнення коду, яке допомагає розробникам швидко і точно писати код. Воно враховує контекст та типи даних, що дозволяє прискорити процес написання коду.

- Візуальні інструменти: Rider має широкий набір візуальних інструментів для дебагування, профілювання та аналізу коду. Вони допомагають знайти й усунути помилки, покращити продуктивність та якість коду.

- Rider підтримує мову програмування C++, яка є основною мовою розробки в Unreal Engine. Розробники можуть використовувати Rider для написання коду на C++ для своїх проектів на Unreal Engine, використовуючи всі переваги інтелектуального автодоповнення, аналізу коду та інших функцій IDE.

- Завдяки гнучкому інтерфейсу Rider, можна встановити плагіни або розширення, розроблені спільнотою або третіми сторонами, які можуть покращити інтеграцію з Unreal Engine. Наприклад, можуть бути доступні плагіни для підсвічування синтаксису Unreal Engine, інтеграції з системою збирання проекту або інших специфічних інструментів.

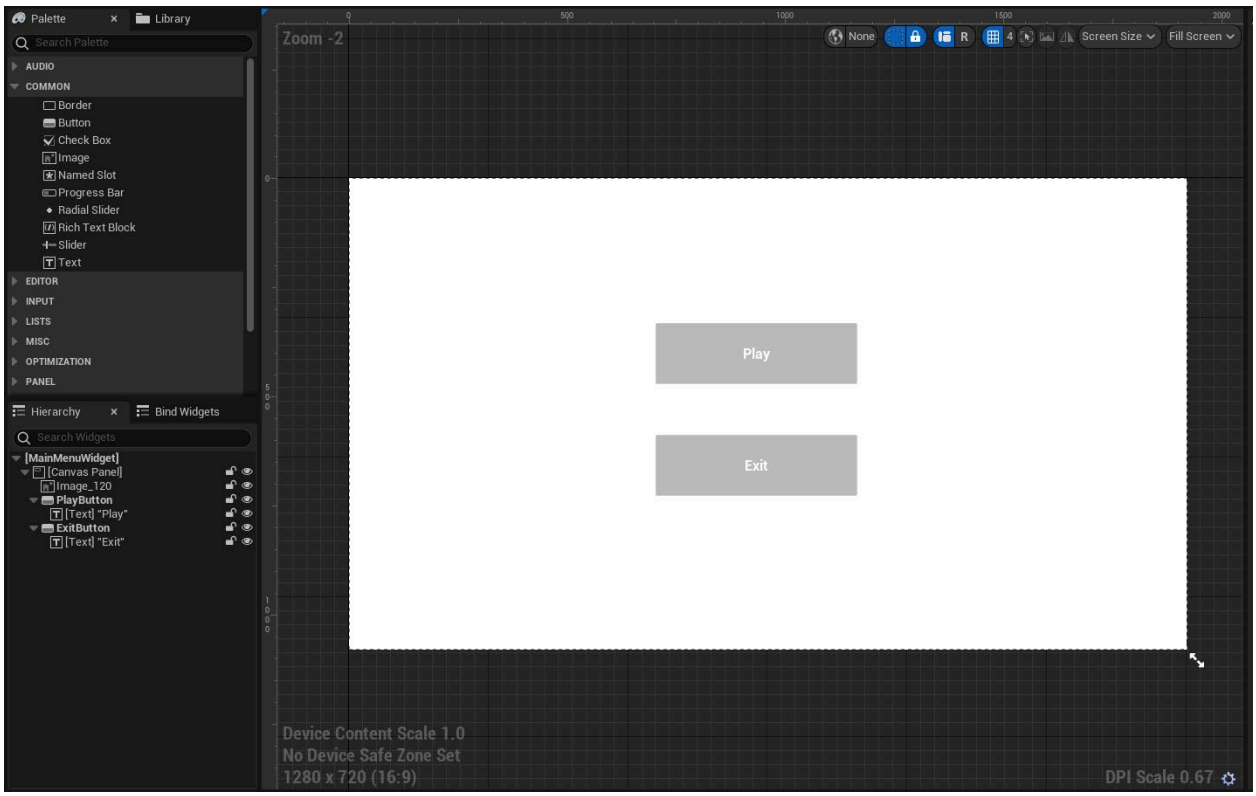
1. Розробка інтерфейсу

Перша сцена, яку бачить гравець відкриваючи гру це головне меню.
Розробку головного меню розділимо для зручності на дві частини: ініціалізація та імплементація.

Ініціалізація: Створюємо клас головного меню UMainMenuWidget. У цьому класі ми оголошуємо дві змінні StartButton і ExitButton, які будуть зв’язані з кнопками у головному меню. Також, оголосимо дві функції Start та Exit, та додатково оголосимо функцію NativeConstruct, яку ми будемо заміщати з базового класу UUserWidget

Імплементація: У функції плей ми викликаємо статичну функцію Unreal Engine, яка викликає наступну сцену гри. У функції Exit ми також викликаємо статичну функцію Unreal Engine виходу з гри, яка завершує роботу програми. Далі, заміщуємо функцію NativeConstruct, яка є конструктором і викликаємо цю функцію від імені базового класу. Далі ми зв’язуємо функції плей так Exit з делегатами кнопок плей та Exit відповідно.

Для подальшої розробки відкриваємо едітор. Створюємо блюпрінт який буде інтерфейсом ігрового меню. До цього блюпрінта додаємо дві кнопки та називаємо їх так само, як і змінні у класі UMainMenuWidget. Власноруч корегуємо їх знаходження відносно сцени і отримаємо працюючий макет. (Рис. 3)



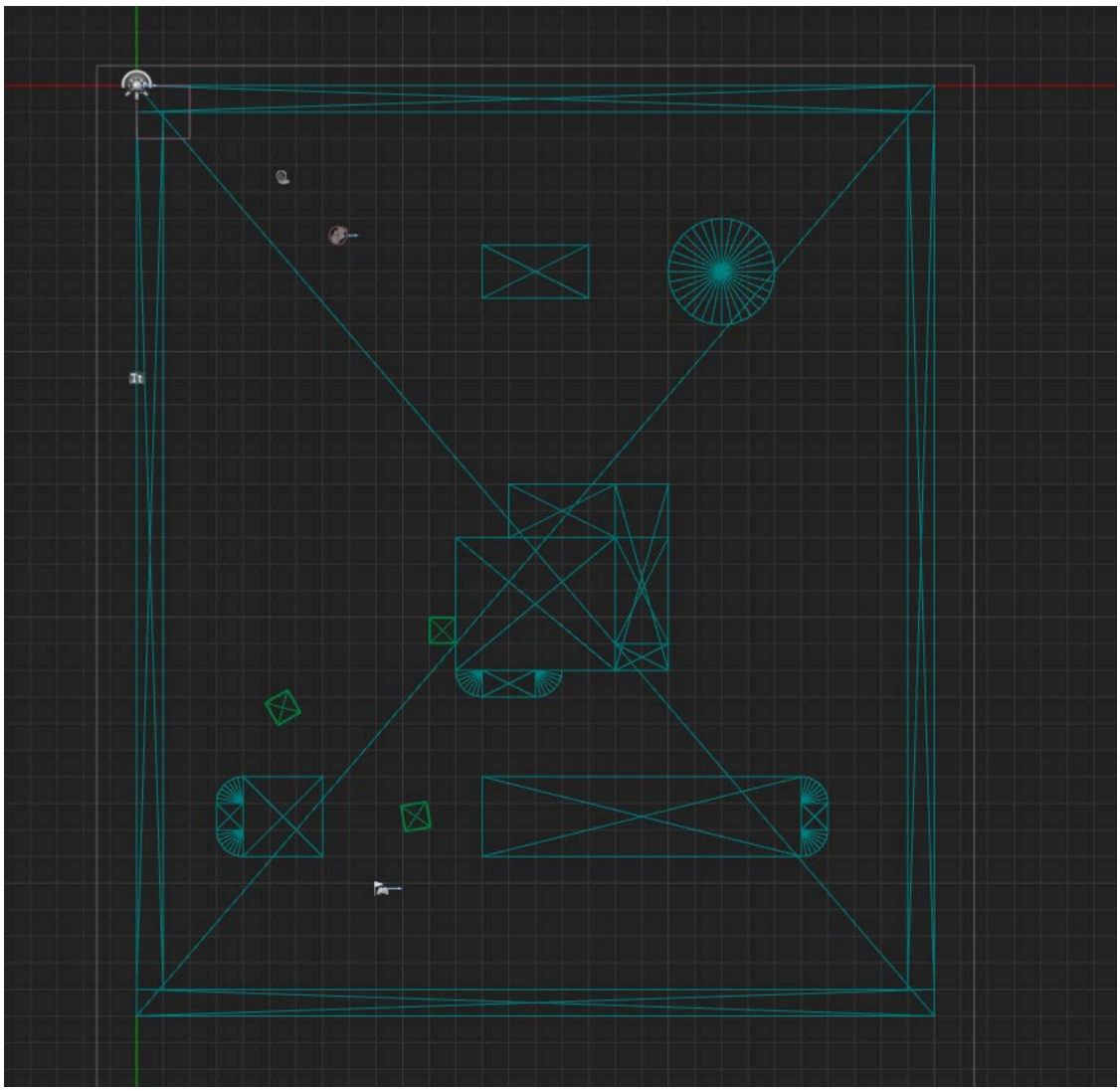
(Рис. 3)

2. Розробка сцени гри та рівня

Наступним кроком є створення рівня, на якому відбуватиметься процес гри.

Для цього знову скористаємось Unreal Editorom, щоб створити нову сцену, на якій спроектуємо рівень. У налаштуваннях рівнів вибираємо камеру від третьої особи і приступаємо до складання рівня.

Для цього я накреслила макет рівня, щоби далі перенести його в 3Д за допомогою інструментів, які надає Unreal Editor. (Рис. 4)



(Рис. 4)

Рівень являє собою довгий лабіринт, зі стартовою точкою, де гравець починає гру. Також на рівні розташовані точки де будуть розміщені NPC, різні особливості ландшафту, які ускладнюють процес гри і демонструють можливості NPC аналізувати ситуацію, простір і обирати кращі можливі шляхи для досягнення мети. Ландшафт являє собою різні височини з виступами, або зі сходами.

Після готового макету далі я приступаю до редагування текстур, щоб створити приємніший вид рівня і гравець відчував себе комфортно та зрозуміло у цьому оточенні.

Після створення рівня я можу перейти до його тестування. Я перевіряю процес гри, виявляю помилки чи недоліки та вношу необхідні зміни та удосконалення. Цей процес включав повторне редагування, налагодження та оптимізацію рівня, щоб забезпечити його функціональність та виправити можливі помилки, щоб запобігти ситуаціям, де гравець може «пройти» скрізь текстуру, або застрягти у ній, те ж саме стосується і NPC.

3. Створення контролера

Для створення контролера над гравцем та NPS на C++, я створюю клас контролера, обираючи тип класу AController і AAIController, де AController буде визначати контролер гравця, а AAIController для NPS та задаю їх імена.

У створеному класі визнаю його успадкування від базового класу контролера, наприклад, AController і AAIController. Це надає контролеру необхідні основні функції та можливості.
Далі я налаштовую функції та поведінки: Визнаю функції та методи, які будуть відповідати за різні аспекти контролера, такі як обробка введення гравця, рух, взаємодія тощо.
У створеному контролері я реалізую віртуальну функцію OnPossess, ця функція викликається коли до цього контролера було призначено Pawn яким він буде управляти. У цій функції ми запускаємо виконання BehaviorTree.
Також додаю систему «зору», яка у Unreal Engine називається AISense_Sight, що є частиною «системою сприйняття» у рушії.

Я створюю функцію OnTargetDetected у якій я буду визначати змінну CanSeePlayer у Black Board. Ця функція закріпляється до делегату системи «зору».

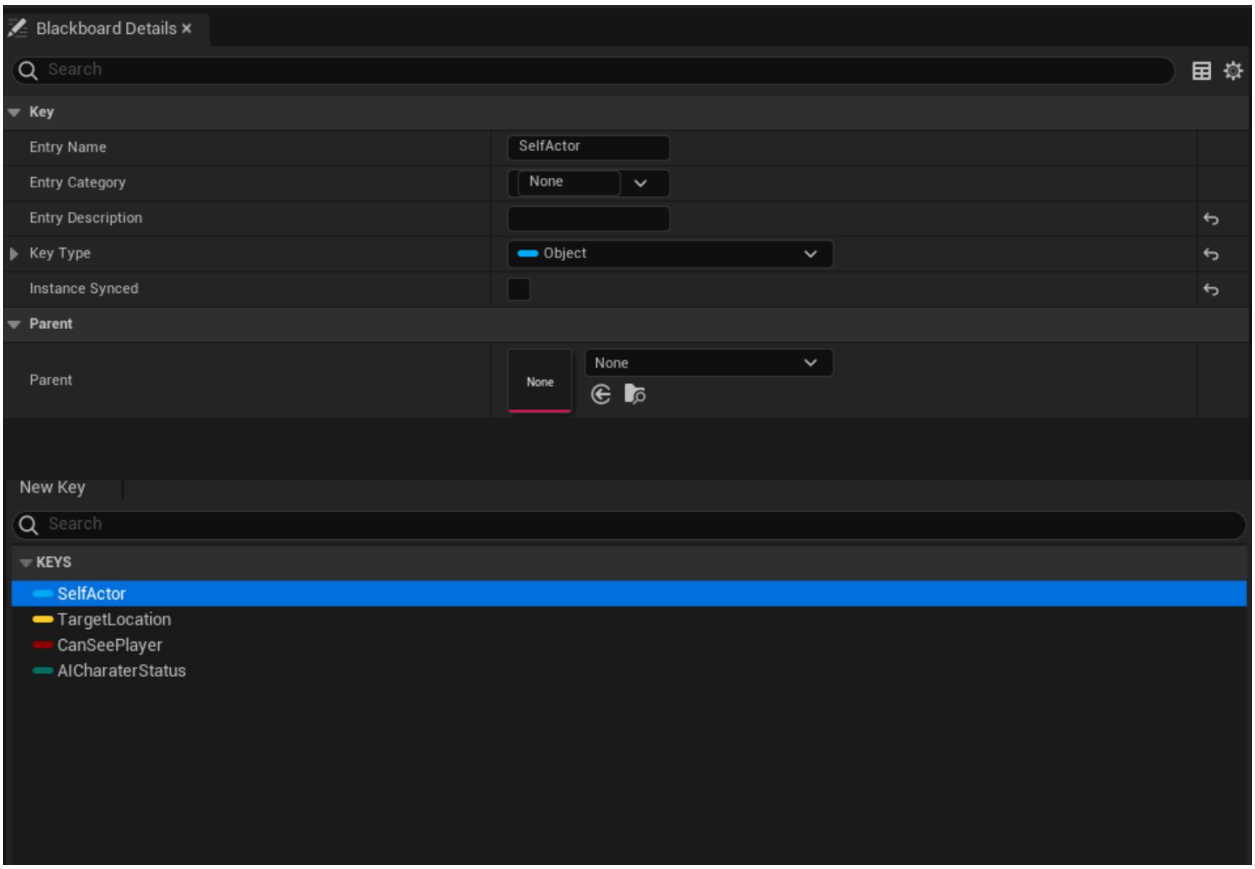
Тож, мій створений контролер призначений для керування певним персонажем, тому я підключаю його до персонажа за допомогою методу SetPawn(). Це дозволяє контролеру взаємодіяти з персонажем та керувати його рухом та поведінкою.

Останнім етапом є тестування та налагодження. Після написання коду контролера важливо виконати тестування та налагодження, щоб переконатися, що він працює належним чином. Тож, я перевіряю правильність реагування контролера на ведення гравця та його взаємодію з грою. На рисунку можна побачити результат вже готового для використання контролеру **(Рис)**

4. Розробка штучного інтелекту та логіки поведінки NPC

Початком для створення дерева поведінки є створення базового дерева поведінки у Blueprints, який містить основну функціональність для керування послідовністю виконання функцій у дереві поведінки. Також паралельно із Blueprints, я створюю компонент BlackBoard, що забезпечує зберігання та обмін даними між різними системами штучного інтелекту або іншими компонентами гри. Blackboard є частиною системи поведінки штучного інтелекту в Unreal Engine. Blackboard дозволяє мені визначати та зберігати ключові значення, які використовуються в системі штучного інтелекту. Ці значення представляють цілі, стан, ресурси, розташування об'єктів тощо. Вони можуть бути поновлюваними та доступними для читання з різних елементів штучного інтелекту, що дозволяє системі штучного інтелекту адаптуватися до змінних умов та приймати рішення на основі актуальних даних.

Також, BlackBoard взаємодіє із C++, бо C++ може викликати функції API, які дозволяють отримувати значення з Blackboard. Я роблю це, використовуючи методи, такі як GetValueAsInt, GetValueAsString, GetValueAsVector та інші, які дозволяють отримати значення за ключем з Blackboard; C++ може використовувати функції API для оновлення значень в Blackboard, за допомогою методів, таких як SetValueAsInt, SetValueAsString, SetValueAsVector та інші, які дозволяють мені встановити нове значення за ключем в Blackboard; C++ може додавати та видаляти ключі з Blackboard. Я роблю це за допомогою методів, таких як AddKey, RemoveKey та інші, які дозволяють динамічно керувати списком ключів у Blackboard. (Рис. 5)



(Рис. 5)

Логіку дерева я прописую на мові C++, а саме логіку поведінки кожного вузла, його функції та перевірки умов. Одна з великих переваг використання C++ класів в Unreal Engine полягає в можливості посилатися на C++ код з Blueprints. Це дозволяє використовувати написаний мною в C++ функціонал у Blueprints. Тож я створюю власні C++ функції та класи, які потім використаю як Blueprints для розширення їх функціональності або реалізації складнішої логіки.

Ця комбінація C++ і Blueprints дозволяє максимально використовувати переваги обох підходів: швидкість розробки з Blueprints та гнучкість та контроль з C++. Це дозволяє мені розширювати та налаштовувати дерева поведінки, використовуючи графічні можливості Blueprints та швидкість мови C++.

Використання Blueprints особливо корисно для прототипування та ітеративного розроблення, що дозволяє мені швидко тестувати та вносити зміни у поведінку без компіляції коду.

Я починаю реалізацію дерева поведінки зі створення асету, який я успадковую від класу BehaviorTree. У створеному асеті у вкладці «Graph» є перша нода «Root», що ж за замовченням у кожному дереві поведінки. Від цієї ноди будується усе дерево поведінки.

Існує декілька видів нод, які я використовую: Selector та Sequence.
Selector виконує розгалуження зліва направо і зазвичай використовується для вибору між піддеревами. Селектори припиняють рух між піддеревами, коли вони знаходять піддерево, яке вони успішно виконують. Наприклад, якщо штучний інтелект успішно переслідує гравця, він залишатиметься в цій гілці, доки не завершиться його виконання, а потім підніметься до батьківського композиту селектора, щоб продовжити процес прийняття рішень.

Sequence виконує розгалуження зліва направо і частіше використовується для виконання черги піддерев по порядку. На відміну від селекторів, послідовність продовжує виконувати свої дочірні елементи, доки не досягне вузла, який повернув «failed».

Тож, я реалізую декілька функцій для NPC: Переслідування, патрулювання, пошук.

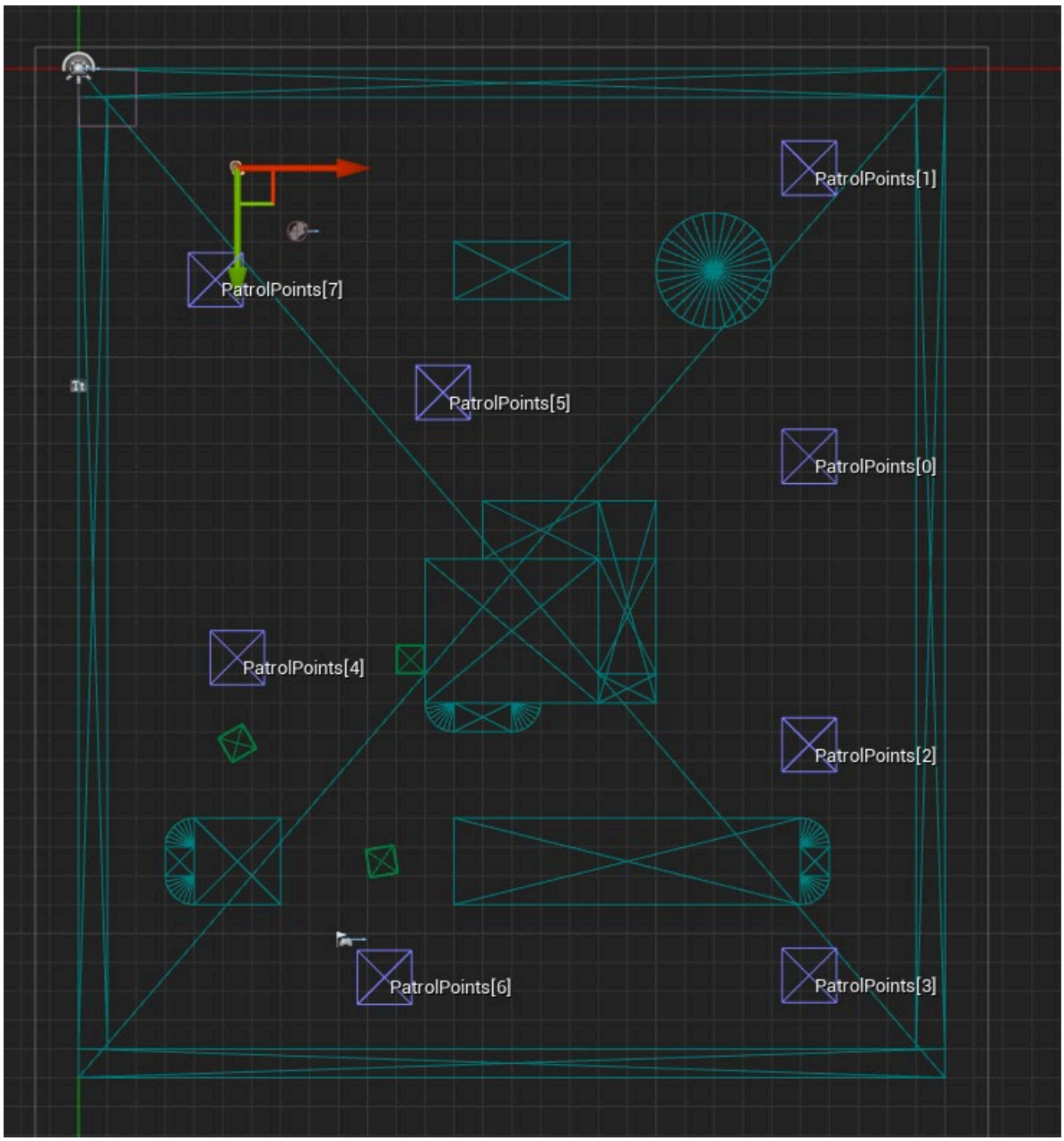
Переслідування: у C++ я створюю клас BTTask_ChasePlayer, який успадковується від класу BTTask_BlackboardBase. У створеному класі реалізуємо віртуальну функцію ExecuteTask, у якій дістаємо значення TargetLocation з Блекборд та наказуємо NPC рухатися до TargetLocation. TargetLocation я попередньо змінила у завданні UBTTask_FindPlayerLocation. За допомогою цих двох завдань я реалізую перше піддерево, яке буде відповідати за переслідування гравця. Для цього від ноди Root я створюю ноду Selector, та від цього Selector я створюю ноду Sequence, додаємо декоратор, який перевіряє значення CanSeePlayer на правдивість у Black Board, тобто, якщо NPC бачить гравця, то він буде виконувати саме цю послідовність. Також додаємо один сервіс, який я вже реалізувала раніше, він змінює швидкість переміщення NPC, коли запускається ця послідовність. Від Sequence створюємо ще дві ноди: перша буде вказувати на завдання Find Player Location, а друга на завдання Chase Player.

Патрулювання та пошук: Від найближчого від початку Selector створюємо ноду, яка також є Selector і називаю її «PlayerNotSeen», і розміщуємо її лівіше за попереднє піддерево. До цього нового Selector додаємо декоратор, який також перевіряє змінну CanSeePlayer, але тепер на неправдивість, тобто, цей селектор буде виконувати свої піддерева, якщо NPC не бачить гравця. Від цього селектору, зліва створюємо Sequence та додаємо декоратор, який

перевіряє змінну Black Board на те, чи було переслідування гравця останньою задачею цього NPC. До цього Sequence створюємо задачу MoveTo, та вказуємо TargetLocation, як ціль цієї задачі, тобто NPC буде рухатися до останньої точки де бачив гравця. Далі, правіше створюємо інший Sequence та назвемо його «Searching», додаємо декоратор Loop та назначимо кількість повторень на потрібну кількість разів, тобто, мі зазначили що послідовність Searching буде виконуватися зазначену кількість разів. Для реалізації послідовності Searching, реалізую UBTTask_FindRandomLocation за прикладом UBTTask_ChasePlayer. Після реалізації цієї створюємо від ноди Searching наліво реалізовану задачу FindRandomLocation, та ще правіше створюємо ноду MoveTo та вказуємо ту точку яку згенерувала задача FindRandomLocation і створюємо задачу Wait ще правіше, та зазначаємо таймер на потрібний час. Тобто, я зробила те, коли NPC втрачає гравця зі свого поля зору, то він йде до останньої точки, де бачив гравця, та зазначений час шукає гравця у зазначеній зоні пошуку.

Повертаюсь до Selector «PlayerNotSeen» та правіше від попереднього створюю інший Sequence, який називаю «Patrolling». До цього Sequence прикріпляю сервіс, який змінює швидкість руку NPC.

Для реалізації патрулювання створюю додаткового актора. Створюю клас у C++, який називаю «APatrolPath», який успадковую від AActor, щоб цей клас можливо було розмістити на рівні гри. У цьому класі створюю масив векторів та дві функції: GetPatrolPoint, щоб отримати точку з масивом за індексом, та Num, яка буде повертати кількість елементів масиву. Розміщую цього актора на рівні гри і тепер ми можемо просто додавати точки у масив та розміщувати їх на рівні потрібним чином, створюючи маршрут патрулювання для NPC. (Рис. 6)

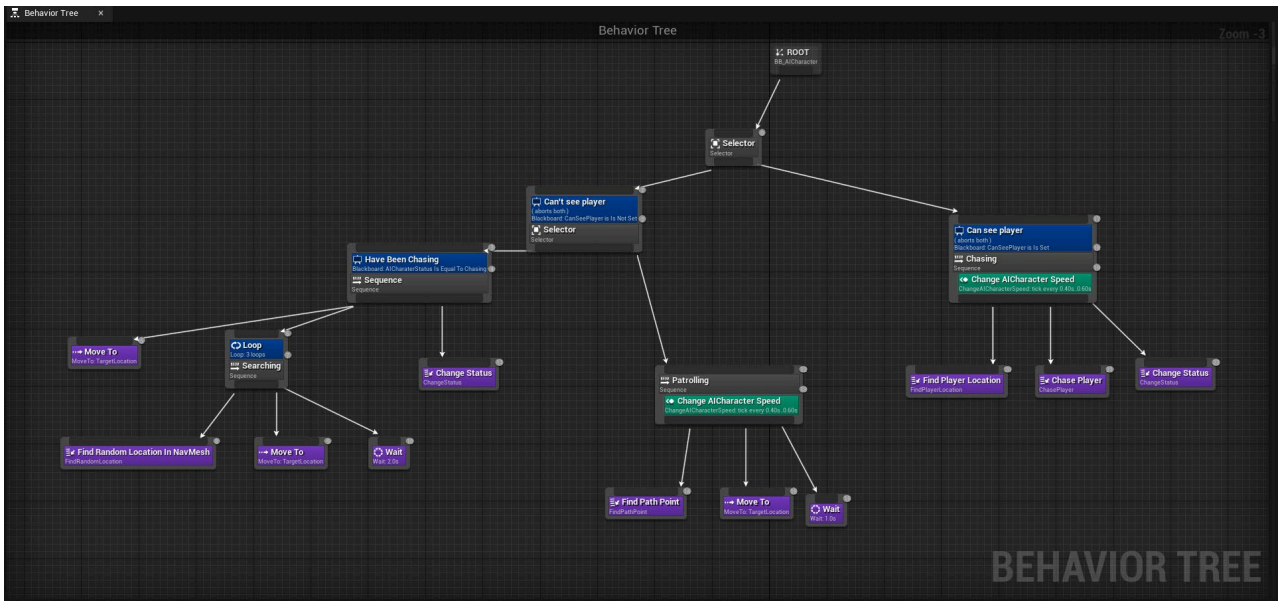


(Рис. 6)

Далі я реалізую задачу і називаю її UBTTask_FindPathPoint. Реалізую її за прикладом вже створених попередньо задач.

Від Sequence «Patroling» зліва створюємо задачу FindPathPoint, далі створюємо ноду MoveTo та вказуємо точку, яку повернула попередня задача. І завершуючи додаємо ноду Wait за прикладом вже створеної раніше.

В підсумку маємо створене дерево, яке можна бачити на рисунку. (Рис 7)



(Рис. 7)

Створене дерево можна вважати базовим деревом поведінки для усіх NPC, бо це дерево має базову поведінку для будь-якого NPC у грі, тобто, для створення більш специфічних, наприклад, які можуть атакувати, або ставити якісь перешкоди ми все одно будемо спадкуватись від цього дерева.

Управління деревом поведінки за допомогою класів C++ дозволяє мені розділити логіку на більш специфічні компоненти та використовувати їх у Blueprints. Тож, я використовую базові класи Blueprints для загальних дій, а потім створюю спеціалізовані класи для конкретних типів взаємодії або поведінки.

Вже з написаними класами C++ я приступаю до оптимізації, використання прямих викликів методів та оптимізацію алгоритмів, що можуть бути важкими або неможливими якщо писати тільки на Blueprints.

2.5 Результати тестування

Під час тестування гри на Unreal Engine 5 з використанням штучного інтелекту були отримані наступні результати:

- Продуктивність та швидкість: UE5 пропонує оптимізації та нові алгоритми, які дозволяють досягти високої продуктивності гри. Під час тестування було зафіксовано плавну роботу гри з високою кількістю полігонів, без помітних затримок або проблем зі швидкістю кадрів.

- Перевірка на відповідність специфікаціям: Використання дерев поведінки дозволило перевірити, чи відповідає поведінка персонажів вимогам специфікацій гри. Тестування включало перевірку правильності реакції персонажів на різні сценарії та дії гравця.

- Керування персонажем: Дерева поведінки забезпечили ефективне та зручне керування персонажем гравця. Відтворювані були різні сценарії поведінки, такі як рух та взаємодія з оточуючим рівнем.

- Адаптивність та гнучкість: За допомогою дерев поведінки було можливо створювати адаптивну поведінку NPS, що реагують на змінювання події у грі.

- Масштабованість та повторне використання: Дерева поведінки є модульними. Це означає, що їх можна використовувати для різних персонажів у грі без необхідності повторно розробляти логіку. Це спрощує розширення гри та дозволяє ефективно управляти поведінкою великої кількості персонажів.

3. ВИСНОВКИ

У результаті розробки гри на рушії Unreal Engine 5 з використанням штучного інтелекту можна зробити наступні висновки:

Використання дерев поведінки стало ефективним інструментом для моделювання складної поведінки персонажів у грі. Дерева поведінки дозволяють виразно виражати різноманітні дії, умови та залежності між ними.

Графічний інтерфейс редагування Blueprints забезпечує зручне та інтуїтивно зрозуміле середовище для розробки дерев поведінки. Розробники можуть візуально створювати, налаштовувати та тестувати поведінку персонажів, що спрощує процес розробки та зменшує залежність від програмування на мові C++.

Використання мови програмування C++ у поєднанні з деревами поведінки дозволяє розширювати можливості і кастомізувати елементи поведінки. Розробники можуть додавати власні функції, створювати спеціалізовану логіку та розширювати функціонал гри з використанням мови C++.

Розробка гри з використанням дерев поведінки на рушії Unreal Engine 5 вимагає від розробників певного рівня знань і розуміння принципів роботи дерев поведінки. Однак, після оволодіння необхідними навичками, цей підхід дозволяє ефективно реалізувати різноманітну та складну поведінку персонажів.

Отже, розробка гри на рушії Unreal Engine 5 з використанням дерев поведінки виявилася успішною та результативною. Використання цього

підходу дозволило створити ігровий досвід з реалістичною, інтелектуальною поведінкою персонажів, що покращує геймплей та надає гравцям нові можливості та емоції.

4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sewell B., Romero M. Blueprints Visual Scripting for Unreal Engine 5: Unleash the true power of Blueprints to create impressive games and applications in UE5 / ed. by L. Cataldi. 3rd ed. Packt Publishing, 2022. 568 p.

2. Newton P. L., Feng J. Unreal Engine 4 AI Programming Essentials. Packt Publishing, 2016. 188 p

3. Sewell B., Romero M. Blueprints Visual Scripting for Unreal Engine 5: Unleash the true power of Blueprints to create impressive games and applications in UE5 / ed. by L. Cataldi. 3rd ed. Packt Publishing, 2022. 568 p.

4. Ulibarri S. S. Unreal Engine C++ the Ultimate Developer's Handbook: Learn C++ and Unreal Engine by Creating a Complete Action Game. Independently published, 2020. 534 p.

5. Lynn M., Sharif C. Game Development with Unreal Engine 5: Learn the Basics of Game Development in Unreal Engine 5. BPB Publications, 2022. 336 p.

6. N4713. Working Draft, Standard for Programming Language C++. Effective from 2017-11-27. Official edition. 1448 p. URL: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf

7. Unreal Engine 5 Documentation. Official edition. URL: https://docs.unrealengine.com/5.0/en-US/