

М. М. Сахно
В. О. Безруков
М. А. Томчук

Візуальне програмування в Unreal Engine

Вінницький національний технічний університет

Анотація:

У доповіді досліджено особливості використання візуального програмування в рушії Unreal Engine за допомогою системи Blueprints. Розглянуто архітектуру Unreal Engine, принципи роботи Blueprints та їх відмінності від традиційного програмування на C++. Проаналізовано переваги та недоліки візуального програмування, а також його вплив на продуктивність і швидкість розробки ігор. Визначено оптимальні випадки використання Blueprints у поєднанні з C++ для створення ефективних ігрових проєктів.

Ключові слова: Unreal Engine, Blueprints, візуальне програмування, C++, розробка ігор.

Abstract:

The report explores the peculiarities of using visual programming in the Unreal Engine using the Blueprints system. The architecture of the Unreal Engine, the principles of Blueprints and their differences from traditional C++ programming are considered. The advantages and disadvantages of visual programming are analyzed, as well as its impact on the performance and speed of game development. The optimal cases of using Blueprints in combination with C++ to create effective game projects are determined.

Keywords: Unreal Engine, Blueprints, visual programming, C++, game development.

Вступ

Сучасна ігрова індустрія постійно розвивається, вимагаючи від розробників все більш складних та реалістичних ігрових механік. В умовах зростаючої конкуренції та високих очікувань гравців важливо не лише створити унікальний ігровий досвід, але й оптимізувати процес розробки, зменшивши витрати часу та ресурсів. У цьому контексті візуальне програмування стало важливим інструментом для спрощення та прискорення процесу створення ігор. Unreal Engine є одним із найпотужніших ігрових рушіїв, який пропонує розробникам можливість створювати ігрові механіки за допомогою системи Blueprints — візуального інструменту програмування, що дозволяє реалізовувати складні алгоритми без написання коду мовою програмування C++. Система Blueprints надає розробникам можливість створювати логіку гри шляхом поєднання вузлів (nodes) у візуальному середовищі, що спрощує процес програмування та робить його доступним навіть для фахівців без глибоких знань у сфері програмування. Попри численні переваги, візуальне програмування в Unreal Engine має певні обмеження. Наприклад, продуктивність сценаріїв, створених за допомогою Blueprints, може бути нижчою порівняно з реалізацією тих самих алгоритмів на мові C++. Крім того, використання Blueprints у складних проєктах може призвести до збільшення складності налагодження та зниження читабельності коду. Таким чином, постає питання, як ефективно поєднувати можливості Blueprints та традиційного програмування на C++ для досягнення оптимальних результатів у розробці ігор.

Основна частина

Візуальне програмування в Unreal Engine є потужним інструментом, що спрощує створення ігрової логіки та відкриває можливості для швидкої розробки навіть без глибоких знань мов програмування. Його використання має низку суттєвих переваг, завдяки яким цей підхід стає популярним серед розробників різного рівня, однак разом із цим існують і певні недоліки, які можуть стати обмеженням у більш складних проєктах.

Однією з головних переваг візуального програмування є значне прискорення процесу розробки. Завдяки графічному інтерфейсу розробники можуть швидко створювати логіку гри, з'єднуючи візуальні блоки (Blueprints) замість написання коду. Це дозволяє економити час, оскільки не потрібно

витрачати багато зусиль на написання та тестування програмного коду, що особливо важливо для невеликих команд або інді-розробників, які працюють у швидкому темпі [3].

Ще однією важливою перевагою є зниження порогу входу для початківців. Люди, які тільки починають свій шлях у розробці ігор, можуть швидше освоїти основи створення ігрової логіки без необхідності вивчення складного синтаксису мов програмування. Це робить Unreal Engine доступнішим для широкого кола користувачів, включаючи художників, дизайнерів рівнів та інших спеціалістів, які хочуть працювати з ігровими механіками, але не мають достатнього досвіду у традиційному програмуванні [5].

Також візуальне програмування значно спрощує процес швидкого прототипування. Завдяки використанню Blueprints можна оперативнo тестувати різні механіки, змінювати параметри та експериментувати з ігровим процесом без необхідності написання великої кількості коду. Це особливо корисно на етапі розробки концепту, коли необхідно швидко оцінити, наскільки вдало працює певна ідея або механіка.

Однак, попри всі переваги, використання візуального програмування має і суттєві недоліки, які можуть обмежувати його застосування у великих проєктах. Одним із головних мінусів є зниження продуктивності при виконанні складних обчислень [1]. Оскільки Blueprints виконуються на вищому рівні абстракції порівняно з нативним кодом C++, вони можуть працювати повільніше, що впливає на загальну продуктивність гри. У випадках, коли гра містить складну фізику, штучний інтелект або велику кількість об'єктів, використання виключно візуального програмування може призвести до підвищеного навантаження на процесор і зменшення частоти кадрів.

Ще однією серйозною проблемою є труднощі з налагодженням та оптимізацією. У міру зростання проєкту кількість візуальних блоків може значно збільшуватися, що робить логіку складнішою для сприйняття та аналізу [4]. Якщо в коді C++ можна легко відстежити всі залежності та взаємозв'язки між функціями, то у Blueprint-графі велика кількість вузлів може швидко перетворитися на хаотичну мережу, де складно знайти джерело помилки. Це ускладнює налагодження та збільшує ризик появи неочікуваних багів, які можуть бути важко відстежуваними.

Крім того, хоч візуальне програмування і зручне для початкових етапів розробки, у великих комерційних проєктах часто доводиться переходити на традиційне програмування на C++ для досягнення кращої продуктивності та контролю над оптимізацією [7]. Деякі функції та механізми значно ефективніше реалізовувати саме на рівні коду, а використання Blueprints у таких випадках може лише ускладнити роботу. Тому досвідчені розробники зазвичай поєднують обидва підходи: Blueprints використовують для швидкого створення загальної логіки та інтеграції, а C++ – для написання продуктивного коду, який виконує критично важливі операції.

Механізм роботи системи Blueprints у Unreal Engine є одним із ключових інструментів для розробки ігрової логіки, що дозволяє створювати складні механізми без необхідності писати код. Використання Blueprints дає змогу швидко експериментувати з ігровими механіками, налаштовувати поведінку об'єктів і розширювати функціональність гри без ризику порушити загальну структуру проєкту. Це особливо корисно для розробників, які не мають досвіду в програмуванні, але хочуть працювати з інтерактивними елементами гри [8].

Основою Blueprints є вузли (Nodes), які виконують різні функції, такі як математичні обчислення, управління потоком логіки, виклики подій та взаємодія між об'єктами [2]. Вузли з'єднуються між собою через спеціальні лінії, що визначають послідовність виконання операцій. Це дозволяє створювати складні алгоритми шляхом побудови логічних ланцюгів без необхідності писати код вручну. Вузли поділяються на різні категорії, включаючи функціональні вузли, оператори, події та інші елементи, які допомагають визначати поведінку об'єктів у грі [9].

Змінні (Variables) використовуються для зберігання даних, які можуть змінюватися під час виконання гри. Вони можуть мати різні типи, включаючи цілі числа, числа з плаваючою комою, рядки, об'єкти, структури даних та масиви. Використання змінних дає змогу зберігати і передавати інформацію між різними частинами Blueprint, що дозволяє створювати динамічні та адаптивні системи. Наприклад, змінні можуть використовуватися для зберігання стану персонажа, параметрів гри або статистики гравця.

Функції (Functions) у Blueprints дозволяють створювати повторно використововані блоки логіки, які можна викликати з різних частин програми. Це допомагає структурувати логіку, зменшити дублювання коду і спростити налагодження. Функції можуть мати входні параметри та повертати результати, що дозволяє створювати складні обчислення та маніпуляції з даними. Наприклад, можна

створити функцію для розрахунку шкоди, яку отримує персонаж при зіткненні з ворогом, і використовувати її щоразу, коли відбувається атака [9].

Макроси (Macros) схожі на функції, але відрізняються тим, що вони безпосередньо вставляються в граф логіки кожного разу, коли викликаються. Це означає, що макроси можуть містити більш складні логічні структури без необхідності передавати вхідні параметри та повертати результати. Вони часто використовуються для створення повторюваних блоків логіки, які можуть бути легко змінені в одному місці, що автоматично оновлює всі їхні екземпляри в проекті [1].

Події (Events) є ще одним ключовим елементом Blueprints, оскільки вони дозволяють реагувати на різні дії та зміни у грі. Події можуть бути викликані внутрішньою логікою гри або взаємодією гравця. Наприклад, подія може спрацьовувати при натисканні клавіші, зіткненні персонажа з об'єктом або зміні певного параметра гри. Події також можуть бути викликані з коду на C++, що дозволяє легко інтегрувати Blueprints із низькорівневими функціями гри [1].

Створення логіки в Blueprints ґрунтується на поєднанні цих елементів для реалізації різних механік. Наприклад, якщо потрібно створити систему ворогів, можна використати події для визначення моменту, коли ворог помічає гравця, функції для обробки руху і атаки, змінні для збереження параметрів здоров'я та поведінки, а вузли для реалізації всіх необхідних операцій. Це дозволяє створювати складні системи без необхідності писати код.

Однак Blueprints не є ізольованою системою, і вони можуть взаємодіяти з кодом на C++. Це дозволяє розробникам комбінувати візуальне програмування з традиційним кодом для досягнення максимальної продуктивності та гнучкості [2]. Взаємодія між Blueprints і C++ відбувається за допомогою спеціальних класів, методів і подій, які можна викликати в обох напрямках. Наприклад, класи, написані на C++, можуть бути використані у Blueprints як батьківські класи для створення нових об'єктів, а також можуть надавати доступ до функцій, які неможливо реалізувати лише за допомогою візуального програмування.

Якщо необхідно оптимізувати певну частину гри, що використовує складні обчислення, наприклад, фізичні симуляції або штучний інтелект, можна написати відповідний код на C++, а потім викликати його в Blueprints для забезпечення інтеграції з іншими частинами гри [4]. Це дозволяє отримати високий рівень продуктивності, уникаючи надмірного використання Blueprints у місцях, де потрібні складні розрахунки.

Використання Blueprints спрощує процес розробки, дозволяючи швидко тестувати ігрову логіку без необхідності перезавантаження гри. Візуальний підхід допомагає краще розуміти структуру логіки і легко вносити зміни. Розробники можуть експериментувати з різними параметрами, налаштовувати механіки гри та змінювати взаємодію між об'єктами без ризику порушити загальну структуру проекту [5]. Це особливо важливо в командній розробці, де різні спеціалісти можуть працювати над своєю частиною гри без потреби глибоко занурюватися в код.

Приклад реалізації ігрової логіки у Blueprints охоплює створення базового управління персонажем, механіку бою, взаємодію з об'єктами гри та додаткові функції, що роблять ігровий процес більш динамічним і захопливим [6]. Управління персонажем реалізується через Blueprint-сценарії, що дозволяють задати обробку введення з клавіатури чи геймпада, налаштувати рухи у всіх напрямках, а також додати стрибки, присідання, спринт та інші дії. Щоб персонаж виглядав природно, використовується система анімацій, яка забезпечує плавні переходи між станами, такими як ходьба, біг, стрибок чи падіння. Крім того, можна додати систему витривалості, яка обмежуватиме кількість спринтів або спеціальних рухів, що робить геймплей більш збалансованим [3].

Одним із ключових елементів гри є механіка бою, яка може включати ближній і дальній бій. Для ближнього бою створюються хітбокси, що активуються під час анімації атаки та перевіряють, чи знаходиться ворог у зоні ураження. Якщо так, то об'єкту ворога завдається шкода, і він реагує відповідною анімацією [7]. Дальній бій реалізується через систему снарядів, які запускаються в напрямку цілі, перевіряють зіткнення та, у разі потрапляння, спричиняють відповідний ефект, наприклад, вибух або відштовхування. Для більш глибокої бойової системи можна реалізувати блокування атак, комбо-удари та особливі здібності персонажа, які активуються за певних умов, наприклад, при наповненні шкали енергії.

Взаємодія з об'єктами гри також відіграє важливу роль у створенні живого світу. Двері можуть відкриватися при наблизенні персонажа або після натискання кнопки, а важелі можуть змінювати стан механізмів у світі. Наприклад, натискання кнопки може активувати платформу, яка піднімається, або відкрити прихований прохід [6]. Пастки додають грі складності, адже персонаж має уникати або

знешкоджувати загрози, такі як падаючі камені, гострі шипи, електричні бар'єри чи рухомі леза. За допомогою Blueprints можна реалізувати складні системи, наприклад, розумних ворогів, які реагують на шум або слідують за персонажем, використовуючи систему штучного інтелекту.

Окрім основних механік, можна додати інвентар, систему збору предметів і квестову систему. Інвентар дозволяє персонажу зберігати знайдені предмети, такі як зброя, броня, зілля або ключі від дверей. У квестовій системі можна реалізувати завдання, наприклад, знайти певний об'єкт, знищити ворога чи допомогти неігровому персонажу. Для цього використовуються змінні та логічні перевірки, що визначають, коли завдання виконане, і які події запускаються після цього. Також можна створити систему діалогів, що дозволяє спілкуватися з NPC і робити вибір, який впливає на подальший розвиток подій у грі [4].

Ще одним важливим елементом є система збереження та завантаження прогресу, яка дозволяє гравцеві продовжувати гру з того моменту, де він зупинився. У Blueprints можна реалізувати збереження положення персонажа, його здоров'я, інвентарю, виконаних квестів та стану світу, що забезпечує безперервність ігрового процесу. У комплексі всі ці механіки створюють цікаву та інтерактивну гру, де персонаж може досліджувати світ, взаємодіяти з об'єктами, битися з ворогами та виконувати різноманітні завдання, а вся логіка реалізується без використання коду, лише засобами Blueprints [9].

Оптимізація та розширення функціоналу проектів в Unreal Engine — це ключові етапи розробки, які дозволяють забезпечити високу продуктивність, ефективність та гнучкість при створенні складних ігор і додатків. Unreal Engine надає потужні інструменти, зокрема Blueprints і C++, які можна використовувати разом для досягнення найкращих результатів у розробці. Використання Blueprints дає змогу розробникам працювати з візуальним інтерфейсом, який дозволяє безпосередньо взаємодіяти з логікою гри без потреби в написанні великої кількості коду [3]. Це дозволяє прискорити процес розробки, оскільки Blueprints дозволяють створювати базову механіку гри.

Проте для складніших і більш ресурсомістких завдань використання тільки Blueprints може бути недостатнім, оскільки цей підхід має обмеження в плані продуктивності. Коли проект потребує оптимізації або високої обчислювальної потужності, необхідно використовувати C++, який дозволяє гнучко налаштовувати продуктивність гри на низькому рівні [5]. C++ дає доступ до потужних можливостей Unreal Engine, таких як низькорівневий контроль над пам'яттю, високошвидкісна обробка даних і доступ до апаратних ресурсів.

У процесі розробки проекту на Unreal Engine часто виникає необхідність у інтеграції різних підсистем і компонентів, таких як фізика, анімації, графіка та звукові ефекти. Blueprints дозволяють швидко підключати та налаштовувати ці компоненти через API Unreal Engine без написання значного обсягу коду [1]. Це особливо корисно на ранніх етапах розробки, коли важливо швидко протестувати ідеї та функціональність. Через Blueprints розробник може підключити всі необхідні підсистеми, не глибоко занурюючись у специфікації API, що прискорює тестування і створення базових механік гри. Але коли виникає потреба у складних налаштуваннях або у виконанні високопродуктивних обчислень, слід звернутися до C++, щоб створити необхідну функціональність на більш низькому рівні [1]. Це дозволяє отримати максимальну продуктивність і оптимізувати ресурси, що особливо важливо в складних проектах з великими обсягами даних або високими вимогами до обчислювальних потужностей.

Для того, щоб зробити інтеграцію між C++ та Blueprints ще більш ефективною, Unreal Engine надає можливість створення користувацьких Nodes у Blueprints. Користувацькі Nodes дозволяють розробникам створювати спеціалізовані функції на C++, які можна безпосередньо використовувати в графічному інтерфейсі Blueprints. Це дає змогу легко створювати і налаштовувати складні системи та алгоритми без необхідності переходити між різними середовищами програмування [8]. Такий підхід дозволяє створювати більш складні ігрові механізми, інтегруючи їх у простіший для використання інтерфейс, що зменшує час на розробку і спрощує підтримку проекту.

Інтеграція з API Unreal Engine через Blueprints також дозволяє розширювати функціональність проекту без необхідності вручну писати великий обсяг коду. Зокрема, це дозволяє без проблем підключати зовнішні бібліотеки або використовувати внутрішні функції движка для роботи з фізикою, освітленням, анімаціями, звуками та іншими аспектами, що полегшує процес розробки і дозволяє зосередитись на більш важливих задачах. Така інтеграція допомагає зберегти час і ресурси, а також покращує стабільність проекту [4].

В загальному, поєднання Blueprints і C++ в Unreal Engine створює ідеальні умови для розробки складних і високопродуктивних проєктів. Це дозволяє розробникам швидко створювати основні функції за допомогою Blueprints, а потім використовувати C++ для реалізації більш складних та оптимізованих рішень, що дає максимальну продуктивність і зручність для підтримки проєкту на всіх етапах його розробки [6]. На рисунку 1 зображено систему Blueprints в Unreal Engine.

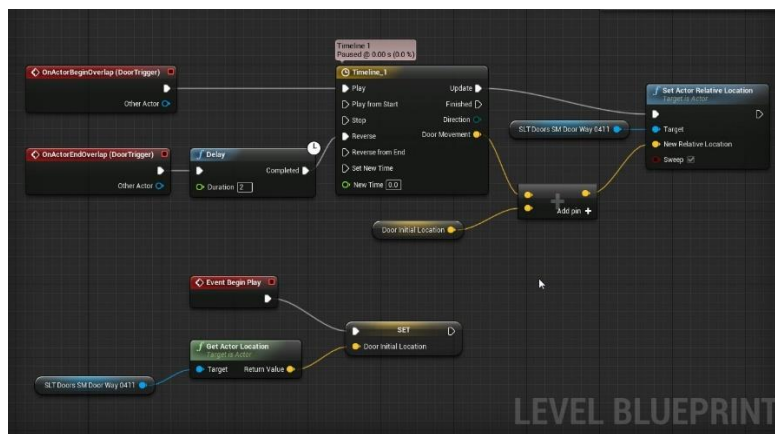


Рисунок 1 – система BluePrints в Unreal Engine

Система Blueprints в Unreal Engine має багато спільного з іншими системами візуального програмування, зокрема Unity Visual Scripting. Візуальне програмування в Unity також реалізоване за допомогою вузлів, проте воно базується на принципах графічного інтерфейсу, де логіка будується шляхом створення діаграм. Основною відмінністю між Blueprints та Unity Visual Scripting є рівень інтеграції з рушієм та продуктивність. Blueprints у Unreal Engine працюють на низькому рівні і безпосередньо генерують C++ код, що забезпечує вищу продуктивність у порівнянні з Unity Visual Scripting, який базується на C# і працює через рівень абстракції [1].

Blueprints також мають більш зручний інтерфейс для створення складних ігрових механік, особливо у випадках роботи з фізикою та анімацією. У Unity Visual Scripting для цього потрібно створювати окремі компоненти і прописувати додаткові налаштування [3]. У випадку Unreal Engine Blueprints дозволяють реалізувати подібні задачі шляхом простого перетягування та з'єднання вузлів у візуальному середовищі. Завдяки цьому Unreal Engine є більш привабливим для розробників, які прагнуть швидко створювати прототипи та реалізовувати складні механіки без написання коду вручну.

Однією з ключових відмінностей між візуальним програмуванням у Blueprints та традиційним програмуванням на C++ є продуктивність виконання коду [2]. Оскільки Blueprints генерують код на рівні C++ і працюють через додатковий рівень абстракції, виконання коду, створеного за допомогою Blueprints, може бути дещо повільнішим порівняно з прямим написанням коду на C++. Особливо це стає помітним у випадках реалізації складних алгоритмів або обробки великої кількості об'єктів у реальному часі. Наприклад, при роботі з великими обсягами фізики чи обчисленням штучного інтелекту продуктивність Blueprints може поступатися чистому коду C++ [7].

У випадках, коли продуктивність має критичне значення, розробники часто комбінують Blueprints із написанням коду на C++. Наприклад, основна логіка гри може бути реалізована на C++, тоді як візуальні ефекти, поведінка об'єктів та анімація можуть бути реалізовані за допомогою Blueprints [4]. Це дозволяє поєднати високу продуктивність коду C++ із зручністю налаштування через Blueprints.

Візуальне програмування у Blueprints значно пришвидшує процес створення прототипів та реалізації базової логіки гри. Інтуїтивно зрозумілий інтерфейс та можливість створювати логіку гри шляхом з'єднання вузлів дозволяють швидко експериментувати з ігровими механіками та вносити зміни без потреби в компіляції коду. У Blueprints реалізовано систему Hot Reload, яка дозволяє відразу бачити зміни в грі без необхідності перезапуску рушія [1].

З іншого боку, написання коду на C++ вимагає більше часу на налаштування середовища, компіляцію та відлагодження коду. Проте код на C++ зазвичай є більш ефективним і точним, що забезпечує стабільність гри на складних етапах розробки [9]. У результаті для швидкого створення прототипів та тестування механік перевагу має Blueprints, тоді як для оптимізації та фінальної реалізації гри ефективніше використовувати C++.

Blueprints забезпечують високу гнучкість у налаштуваннях та створенні логіки гри на ранніх етапах розробки. Візуальне середовище дозволяє легко налаштовувати об'єкти, створювати анімації, ефекти та обробляти події через подієву модель. Однак при створенні великих ігор з великою кількістю об'єктів логіка, побудована на Blueprints, може стати занадто складною та важкою для підтримки. У таких випадках краще реалізовувати базову логіку на C++ та використовувати Blueprints лише для налаштувань і взаємодії між об'єктами [5].

Наприклад, в Unreal Engine часто застосовується підхід, коли логіка створення об'єктів і керування пам'яттю реалізується через C++, тоді як анімації, візуальні ефекти та взаємодія об'єктів реалізуються через Blueprints. Це дозволяє досягти балансу між продуктивністю та зручністю розробки. У великих проєктах використання C++ для базової логіки забезпечує масштабованість та ефективність, тоді як Blueprints дозволяють швидко вносити зміни та налаштовувати поведінку окремих об'єктів [2].

Mortal Kombat 11 була створена студією NetherRealm на модифікованій версії Unreal Engine 3. В процесі розробки частково використовувалися Blueprints для створення деяких аспектів ігрової логіки та налаштування анімацій, що значно прискорило процес тестування та внесення змін без необхідності перекомпіляції коду. Це дозволило розробникам швидко налаштовувати бойові механіки та баланс персонажів, а також створювати сценарії для кат-сцен і взаємодії персонажів.

Використання Blueprints прискорило створення навчального режиму та тонке налаштування бойової системи, оскільки дизайнери могли працювати з логікою гри без залучення програмістів. Однак деякі недоліки включали збільшене споживання пам'яті через використання Blueprints, що могло впливати на продуктивність на консолях попереднього покоління. Крім того, складні сценарії в Blueprints були важко керованими через заплутану структуру вузлів, що ускладнювало підтримку та оптимізацію на пізніх етапах розробки.

Одним із викликів під час розробки було поєднання коду на C++ із логікою, створеною через Blueprints. Це створювало потенційні конфлікти під час налаштування складних механік бою. Наприклад, деякі комбінації атак могли некоректно працювати через конфлікти між кодом і Blueprint-сценаріями. Розробники вирішували цю проблему шляхом чіткого розподілу відповідальності між Blueprints і C++: критично важливі механіки залишалися в C++, тоді як вторинні налаштування (анімації, ефекти, логіка UI) реалізовувалися через Blueprints.

Висновки

Візуальне програмування в Unreal Engine за допомогою системи Blueprints справило значний вплив на процес розробки ігор, особливо у сфері прискорення створення ігрової логіки та прототипування. Завдяки Blueprints розробники отримали можливість швидко реалізовувати складні механіки без необхідності глибоких знань програмування на C++. Це дозволило дизайнерам ігрового процесу самостійно створювати і тестувати ігрову логіку, зменшуючи навантаження на програмістів і скорочуючи загальний час розробки.

Порівняння Blueprints із традиційним програмуванням на C++ показало, що Blueprints забезпечують вищу швидкість створення та тестування ігрової логіки за рахунок простоти використання та візуального відображення логіки. Це робить їх зручними для швидких змін і налагодження, особливо на етапі прототипування та початкової розробки. Однак з точки зору продуктивності, код, написаний на C++, має перевагу, оскільки він виконується швидше і дозволяє більш ефективно керувати використанням пам'яті та ресурсів. У складних ігрових проєктах розробники часто використовують комбінацію Blueprints і C++ для досягнення оптимального балансу між швидкістю розробки та продуктивністю гри.

Для досягнення максимальної ефективності в розробці ігор на Unreal Engine рекомендується поєднувати Blueprints та C++ залежно від етапу розробки та типу завдання. Blueprints ідеально підходять для швидкого створення прототипів, налаштування механік та створення анімацій, оскільки вони дозволяють дизайнерам ігрового процесу працювати з логікою гри без залучення програмістів. Це дозволяє швидко тестувати різні ідеї та вносити зміни в реальному часі. Однак у фінальних стадіях розробки або при створенні продуктивно інтенсивних механік (наприклад, обробки фізики, штучного інтелекту чи складних мережевих систем) доцільно використовувати C++ для підвищення продуктивності та кращого керування ресурсами. Оптимальним підходом є створення базової логіки та функціоналу в C++, а використання Blueprints – для більш гнучких налаштувань і швидких змін.

Таким чином, комбінація обох підходів дозволяє досягти високої продуктивності та гнучкості під час розробки складних ігрових проєктів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Unreal Engine Documentation [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com>
2. Epic Games – Unreal Engine [Електронний ресурс]. – Режим доступу: <https://www.unrealengine.com>
3. Blueprints vs. C++: Which should I use for Unreal Engine game development? [Електронний ресурс]. – Режим доступу: <https://www.wholetomato.com/blog/2024/10/23/c-versus-blueprints-which-should-i-use-for-unreal-engine-game-development/>
4. Unity чи Unreal Engine — плюси і мінуси [Електронний ресурс]. – Режим доступу: <https://gamedev.dou.ua/articles/unity-or-unreal-engine/>
5. Blueprint VS C++ in Unreal Engine 5, Which One Should You Choose [Відео]. – Режим доступу: <https://www.youtube.com/watch?v=vpysMjReREw>
6. Розробка ігор на Unreal Engine 5 [Електронний ресурс]. – Режим доступу: <https://avada-media.ua/services/razrabotka-igr-na-unreal-engine-5/>
7. Порівняння мов програмування C++ та BLUEPRINTS при розробці відеогри на UNREAL ENGINE [Електронний ресурс]. – Режим доступу: <https://orcid.org/0000-0001-5826-0751>
8. Курс Unreal Engine [Електронний ресурс]. – Режим доступу: <https://gamestudio.n-ix.com/unreal-engine-course/>
9. Unreal Engine Blueprints for Beginners [Електронний ресурс]. – Режим доступу: <https://gamedev.tutsplus.com/tutorials/unreal-engine-blueprints-for-beginners--cms-20447>

Сахно Михайло Миколайович – студент групи 2КІ-236, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: sahnomihajlo51@gmail.com

Безруков Владислав Олександрович – студент групи 2КІ-236, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: bezrukovvladyslav8578212@gmail.com

Томчук Микола Антонович – канд. техн. наук, доцент кафедри обчислювальної техніки, Вінницький національний технічний університет, м. Вінниця, e-mail: tomchuk@vntu.edu.ua.

Sakhno Mykhailo M. - – students, 2CE-23B, Faculty of information Technologies and Computer Engineering, Vinnytsa National Technical University, email: sahnomihajlo51@gmail.com

Bezrukov Vladyslav O. - – students, 2CE-23B, Faculty of information Technologies and Computer Engineering, Vinnytsa National Technical University, email: bezrukovvladyslav8578212@gmail.com

Tomchuk Mykola A. – Cand. Sc. (Eng.), Assistant Professor at the Department of Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: tomchuk@vntu.edu.ua.