ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ

ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ГРИ “MIGHTY TOAD” ЖАНРУ ПЛАТФОРМЕР НА ОСНОВІ ДВИГУНА UNREAL ENGINE**»

Виконав: студент 5 курсу, групи ППЗ– 51

спеціальності:

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Єрмак Б.С.

(прізвище та ініціали)

Керівник Негоденко О.В.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Нормоконтроль

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного

забезпечення

_____О.В. Негоденко

“ ” _____ 2022 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Єрмаку Богдану Станіславовичу

(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка гри “Mighty Toad” жанру платформер на основі двигуна Unreal Engine»
   Керівник роботи зав. кафедри ІПЗ Негоденко О.В.
   (прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
   затверджені наказом вищого навчального закладу від “18” лютого 2022 року №__.

2. Строк подання студентом роботи 03.06.2022.
3. Вихідні дані до роботи:
   3.1. Положення побудови гри;
   3.2. Методи побудови гри;
   3.3. Розробка моделі гри;
   3.4. Науково-технічна література;
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1. Загальні положення побудови гри;
   4.2. Аналіз технології і методів побудови гри;
   4.3. Висновки

5. Перелік графічного матеріалу:
5.1. Титульний слайд;
5.2. Мета, об’єкт та предмет дослідження;
5.3. Актуальність роботи;
5.4. Аналоги;
5.5. Технічне завдання;
5.6. Програмні засоби реалізації;
5.7. Апробація результатів дослідження;
5.8. Висновки;

6. Дата видачі завдання: 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалавської роботи	Строк виконання етапів роботи	Примітка
1	Підготовка Розділу 1	11.04-17.04	Виконано
2	Підготовка Розділу 2	18.04-21.04	Виконано
3	Підготовка Розділу 3	22.04-07.05	Виконано
4	Висновки, презентація	07.05-15.05	Виконано
5	Попередній захист диплому	16.05-01.06	
6	Здача роботи	03.06	

Студент _____ Єрмак Б.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 64 стр., 7 рис., 49 джерел

Ключові слова: ГРА, ВІДЕОІГРИ, ПЛАТФОРМЕР, UNREAL ENGINE, C++, Blueprint

Об'єкт дослідження – процес розробки двовимірної комп'ютерної гри у жанрі платформер.

Предмет дослідження – методи, прийоми та інформаційні технології розробки ігрового програмного забезпечення.

Мета роботи – розробити та вдосконалити комп'ютерну гру, з можливістю її використання на платформі Windows.

Методи дослідження – концепція технології створення комп'ютерної гри, розбиття її на рівні, та опис розроблених ігрових механік для можливості більш активної взаємодії персонажа з навколишнім середовищем, які були описані в ході розробки.

Для реалізації мети було сформульовано та вирішено наступні завдання:

1. Провести аналіз схожих робіт.

2. Дослідити середовище розробки Unreal Editor та мову C++.

3. Розробити гру для проходження тестування.

4. Провести тестування. Предметом дослідження є гра платформер.

Практичне значення отриманих результатів полягає у написанні функціоналу для роботи з інтерполяцією візуального контенту гри Mighty Toad з використання ігрового рушія Unreal Engine та мови програмування C++ для платформи Windows. Також реалізовану гру можна поширювати серед гравців, виклавши її до магазинів ком'пютерних ігор.

В роботі розглянуто основні етапи створення ігор і досліджено можливості технічних засобів для розробки гри.

Розроблено модель застосунку та функціональні вимоги до гри та підібрані методи інтерполювання для їх реалізації.

Галузь використання – завдяки розвитку IT індустрії, будь-який користувач може завантажити та використовувати гру Mighty Toad в жанрі 2D платформер.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок карантинних обмежень спричинених глобальною пандемією 2019-2022рр. люди стали проводити вдома набагато більше часу. Перебування вдома означало, що кількість людей, які грають у ігри, різко зросла. З 2019 по 2021 роки кількість завантажень ігор у світі зросла на 75%, а час перегляду стрімів ігор збільшився на 45%. У 2020 році світовий ринок ігор досяг 162,32 мільярда доларів. Очікується, що до 2026 року ця цифра зросте до 295,63 мільярда доларів. Іншими словами, галузь процвітає і неймовірно конкурентоспроможна.

В дипломній роботі обрано рушій Unreal Engine, що розробляється і підримується компанією Epic Games. Написаний мовою C++, рушій дозволяє створювати ігри для більшості операційний систем персональних комп'ютерів, консолей компаній Microsoft, Nintendo, Sony та мобільних платформ. Його використовували у своїх проектах розробники BioShock, Borderlands та, з останніх новин, нова версія рушія Unreal Engine 5 буде використана для розробки The Witcher 4.

Жанр платформер був на піку популярності наприкінці минулого сторіччя, коли вийшли такі вже класичні ігри як Donkey Kong, Mario, Sonic, Metroid та Crash Bandicoot. Зараз цей жанр переживає нову хвилю популярності, завдяки Cuphead, Limbo, серіям Rayman та Ori, а також розквіту інді-сцени розробки ігор.

Мета та завдання. Метою є розробка та вдосконалення комп'ютерної гри, з можливістю її використання на платформі Windows.

Гра Mighty Toad повинна мати наступні характеристики:

- двовимірний світ;
- можливість контролювати персонажа за допомогою клавіатури або геймпаду;
- рівень має складатися з статичних та динамічних платформ;
- персонаж має обмежену кількість спроб на подолання рівня;
- персонаж має засоби для знешкодження ворогів та перешкод;

Об'єктом роботи виступає процес розробки двовимірного ігрового застосунку у жанрі «платформер» для Microsoft Windows.

Предмет – методи, прийоми та інформаційні технології розробки ігрового програмного забезпечення.

Методи дослідження – концепція технології створення комп'ютерної гри, розбиття її на рівні, та опис розроблених ігрових механік для можливості більш активної взаємодії персонажа з навколишнім середовищем, які були описані в ході розробки.

Практичне значення отриманих результатів полягає у написанні функціоналу для роботи з інтерполяцією візуального контенту гри Mighty Toad з використання ігрового рушія Unreal Engine та мови програмування C++ для платформи Windows. Також реалізовану гру можна поширювати серед гравців, виклавши її до магазинів ком'пютерних ігор.

Структура роботи. Структуру роботи складають: вступ, три розділи, висновки, перелік використаної літератури та додатки.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Відеоігри

Хоча відеоігри сьогодні можна знайти в будинках по всьому світу, вони насправді почали свою діяльність у дослідницьких лабораторіях вчених. Перший визнаний зразок ігрової машини був представлений доктором Едвардом Улером Кондоном на Всесвітній виставці в Нью-Йорку в 1940 році. У гру, засновану на стародавній математичній грі Нім, грали близько 50 000 людей протягом шести місяців її існування, при чому комп'ютер, як повідомляється, виграв понад 90 відсотків ігор. У 1952 році британський професор А.С. Дуглас створив OXO, також відомий як хрестики і нулі, в рамках своєї докторської дисертації в Кембриджському університеті. А в 1958 році Вільям Хігінботам створив Tennis for Two на великому аналоговому комп'ютері та екрані підключеного осцилографа для щорічного дня відвідувача Брукхейвенської національної лабораторії в Аптоні, штат Нью-Йорк. У 1962 році Стів Рассел з Массачусетського технологічного інституту винайшов Spacewar!, комп'ютерну космічну бойову відеогру для PDP-1 (Programmed Data Processor-1), тодішнього передового комп'ютера, який здебільшого зустрічався в університетах. Це була перша відеогра, в яку можна було грати на кількох комп'ютерних установках.

У 1967 році розробники компанії Sanders Associates, Inc. на чолі з Ральфом Баером винайшли прототип багатокористувацької багатопрограмної системи відеоігор, у яку можна було б грати на телевізорі. Вона була відомою як «Коричнева коробка». Вона була мікросхемою з вакуумними трубками, яку можна було під'єднати до телевізора і дозволяла двом користувачам керувати кубами, які ганялися один за одним на екрані. «Коричневу коробку» можна запрограмувати для гри в різноманітні ігри, включаючи пінг-понг, шашки та чотири спортивні ігри. Використовуючи передові технології для цього часу, додані аксесуари включали світловий пістолет для гри в мішень і спеціальне пристосування для гри в гольф. За даними Національного музею американської історії, Баер згадував: «У ту хвилину, коли ми грали в пінг-понг, ми зрозуміли, що у нас є продукт. Раніше ми були не дуже впевнені». Баер, якого іноді називають батьком відеоігор, ліцензував свій пристрій компанії Magnavox, яка продала систему споживачам як Odyssey, першу домашню консоль для відеоігор, у 1972 році. Вона на кілька місяців випередила Atari, яку часто помилково вважають першою ігровою консоллю. Протягом наступних кількох років примітивна консоль Odyssey стала некомерційною і зникла з продажу. Проте одна з 28 ігор Odyssey стала

джерелом натхнення для Pong, першої аркадної відеоігри, яку компанія Atari випустила в 1972 році. Sega і Taito були першими компаніями, які викликали інтерес громадськості до аркадних ігор, коли вони випустили електромеханічні ігри Periscope і Crown Special Soccer в 1966 і 1967 роках проте Atari (заснована Ноланом Бушнеллом, хрещеним батьком ігор) стала першою ігровою компанією, яка дійсно встановила еталон для великомасштабної ігрової спільноти.

Наприкінці 1970-х років ряд мережевих ресторанів у США почали встановлювати відеоігри, щоб отримати вигоду від гарячого нового захоплення. Характер ігор викликав конкуренцію серед гравців, які могли записувати свої найкращі результати своїми ініціалами і були рішуче налаштовані позначити своє місце у верхній частині списку. У цей момент багатокористувацькі ігри були обмежені гравцями, які змагалися на одному екрані. Перший приклад гравців, які змагалися на окремих екранах, з'явився в 1973 році з "Empire" — стратегічною покроковою грою для восьми гравців, яка була створена для мережевої системи PLATO. PLATO (Programmed Logic for Automatic Teaching Operation) була однією з перших узагальнених комп'ютерних систем навчання, спочатку створених Університетом штату Іллінойс, а потім перейнята Control Data (CDC), яка побудувала машини, на яких працювала система. Згідно з журналами використання системи PLATO, користувачі провели близько 300 000 годин, граючи в Empire в період з 1978 по 1985 рік. У 1973 році Джим Бавері випустив Spasim для PLATO — космічного шутера для 32 гравців, який вважається першим прикладом багатокористувацьких 3D-ігор. . Хоча доступ до PLATO був обмежений для великих організацій, таких як університети — і Atari — які могли дозволити собі комп'ютери та з'єднання, необхідні для приєднання до мережі, PLATO є одним із перших кроків на технологічному шляху до Інтернету та онлайн-ігор для кількох гравців, оскільки ми знати це сьогодні. У цей момент ігри були популярні серед молодого покоління і були спільним заняттям, оскільки люди змагалися за високі результати в іграх. Однак більшість людей не вважали б, що чотири з кожних п'яти американських домогосподарств мають ігрову систему як ймовірну реальність.

Окрім того, що ігрові приставки стали популярними в комерційних центрах і мережевих ресторанах США, на початку 1970-х років також стали реальністю персональні комп'ютери та ігрові консолі масового виробництва. Технологічні досягнення, такі як винахід Intel першого в світі мікропроцесора, привели до створення таких ігор, як Gunfight в 1975 році, першого прикладу багатокористувацького бойового шутера від людини до людини. Незважаючи на те, що Gunfight був далеким від Call of Duty, він був великою справою, коли вперше вийшов на аркади. Він прийшов із новим стилем гри, використовуючи

один джойстик для керування рухом, а інший — для напряму стрільби — чогось ніколи раніше не бачили.

У 1975 році Atari випустила домашню версію Pong, яка була настільки ж успішною, як і її аркадний аналог. У 1977 році Atari випустила Atari VCS (пізніше відому як Atari 2600), але виявила, що продажі повільні, продавши лише 250 000 машин за перший рік, а потім 550 000 у 1978 році — значно нижче очікуваних показників. Низькі продажі звинувачували в тому, що американці все ще звикали до ідеї кольорових телевізорів вдома, консолі були дорогі, а люди втомилися від Pong, найпопулярнішої гри Atari. Коли він був випущений, Atari VCS був розроблений лише для гри в 10 простих ігор, таких як Pong, Outlaw і Tank. Однак консоль включала зовнішній слот ПЗУ, куди можна було підключати ігрові картриджі; Потенціал був швидко виявлений програмістами з усього світу, які створювали ігри, які значно перевершували оригінальну консоль.

У міру того, як домашні та аркадні ігри процвітали, розвивався і ігрове співтовариство. Наприкінці 1970-х і на початку 1980-х були випущені журнали для любителів, такі як Creative Computing (1974), Computer and Video Games (1981) і Computer Gaming World (1981). Ці журнали створювали відчуття спільноти та пропонували канал, за допомогою якого геймери могли залучатися до товариства.

Бум відеоігор, викликаний Space Invaders, призвів до появи величезної кількості нових компаній і консолей, що призвело до періоду насичення ринку. Занадто багато ігрових консолей і надто мало цікавих, які залучають нові ігри, щоб на них грати, врешті-решт привели до краху відеоігор у Північній Америці 1983 року, який приніс величезні втрати та вантажівки непопулярних ігор низької якості, похованих у пустелі, щоб позбутися від них. Ігрова індустрія потребувала змін. Більш-менш у той самий час, коли консолі почали отримувати погану пресу, домашні комп'ютери, такі як Commodore Vic-20, Commodore 64 та Apple II, почали зростати в популярності. Ці нові домашні комп'ютерні системи були доступними для середнього американця, вони продавалися приблизно за 300 доларів на початку 1980-х років (близько 860 доларів на сьогоднішній день), і рекламувалися як «розумний» варіант для всієї родини. Ці домашні комп'ютери мали набагато потужніші процесори, ніж консолі попереднього покоління; це відкрило двері до нового рівня ігор із більш складними, менш лінійними іграми. Вони також запропонували технологію, необхідну геймерам для створення власних ігор з кодом BASIC. Навіть Білл Гейтс розробив гру під назвою Donkey (проста гра, яка включала ухилення від віслюків на шосе за кермом спортивного автомобіля). Цікаво, що гру повернули з мертвих як додаток для iOS у 2012 році. Хоча в той час суперники з Apple описували гру як «грубу та бентежну», Гейтс включив гру,

щоб надихнути користувачів на розробку власних ігор і програм за допомогою інтегрованої програми BASIC. Такі журнали, як Computer and Video Games і Gaming World, надавали вихідний код BASIC для ігор і допоміжних програм, які можна було вводити на ранніх ПК. Ігри, програми та коди читачів приймались та поширювались. На додаток до того, що ранні комп'ютери надали можливість для більшої кількості людей створювати власну гру за допомогою коду, ранні комп'ютери також проклали шлях до багатокористувацьких ігор, що стало ключовою віхою в еволюції ігрової спільноти. Ранні комп'ютери, такі як Macintosh, і деякі консолі, такі як Atari ST, дозволяли користувачам підключати свої пристрої до інших гравців ще наприкінці 1980-х років. У 1987 році MidiMaze був випущений на Atari ST і включав функцію, за допомогою якої можна було з'єднати до 16 консолей, підключивши порт MIDI-OUT одного комп'ютера до порту MIDI-IN наступного комп'ютера. Хоча багато користувачів повідомляли, що більше чотирьох гравців одночасно різко сповільнювали гру та зробили її нестабільною, це був перший крок до ідеї смертельного поєдинку, популярність якого вибухнула з виходом Doom у 1993 році і є однією з найпопулярніших сьогодні видом ігор.

Багатокористувацькі ігри через мережі справді пішли з випуском Pathway to Darkness у 1993 році, і народилася «LAN Party». Ігри через локальну мережу стали більш популярними з випуском Marathon на Macintosh в 1994 році і особливо після того, як у 1996 році з'явився в магазинах шутер від першої особи Quake. До цього моменту випуск Windows 95 і доступних карт Ethernet привів до подальшого розширення мережі на ПК з Windows та зростанню популярності багатокористувацьких локальних ігор. Справжня революція в іграх відбулася, коли локальні мережі, а згодом і Інтернет, відкрили багатокористувацькі ігри. Багатокористувацькі ігри вивели ігрове співтовариство на новий рівень, оскільки дозволили фанатам змагатися та взаємодіяти з різних комп'ютерів, що покращило соціальний аспект ігор. Цей ключовий крок заклав основу для широкомасштабних інтерактивних ігор, якими зараз користуються сучасні геймери. 30 квітня 1993 року ЦЕРН передав програмне забезпечення World Wide Web у суспільне надбання, але пройдуть роки, перш ніж Інтернет стане достатньо потужним, щоб пристосувати до ігор, якими ми його знаємо сьогодні.

Домашня індустрія відеоігор почала відновлюватися в 1985 році, коли Nintendo Entertainment System (NES), яка називалась Famicom в Японії, прийшла до Сполучених Штатів. NES покращила 8-бітну графіку, кольори, звук та ігровий процес порівняно з попередніми консолями. Nintendo, японська компанія, яка починала як виробник гральних карт у 1889 році, випустила ряд важливих франшиз відеоігор, які існують і сьогодні, таких як

Super Mario Bros., The Legend of Zelda і Metroid. Крім того, Nintendo ввела різні правила щодо сторонніх ігор, розроблених для її системи, допомагаючи боротися з поспіхом, низькоякісним програмним забезпеченням. Сторонні розробники випустили багато інших довготривалих франшиз, таких як Mega Man від Capcom, Castlevania від Konami, Final Fantasy від Square і Dragon Quest від Enix (Square і Enix пізніше об'єднаються, щоб утворити Square Enix в 2003 році). У 1989 році Nintendo знову зробила хвилю, популяризуючи портативні ігри, випустивши свій 8-розрядний пристрій для відеоігор Game Boy і гру Tetris, яка часто входила у комплект. Протягом наступних 25 років Nintendo випустить ряд успішних наступників Game Boy, включаючи Game Boy Color у 1998 році, Nintendo DS у 2004 році, Nintendo 3DS у 2011 році та Nintendo Switch у 2017 році.

Задовго до того, як ігрові гіганти Sega і Nintendo перейшли в сферу онлайн-ігор, багато інженерів намагалися використовувати потужність телефонних ліній для передачі інформації між консолями. Вільям фон Майстер представив новаторську технологію передачі модемом для Atari 2600 на виставці Consumer Electronics Show (CES) у Лас-Вегасі в 1982 році. Новий пристрій, CVC GameLine, дозволив користувачам завантажувати програмне забезпечення та ігри за допомогою фіксованого телефонного з’єднання та картриджа, які можна було б підключити до їхньої консолі Atari. Пристрій дозволяв користувачам «завантажувати» кілька ігор від програмістів по всьому світу, в які можна було грати безкоштовно до восьми разів; це також дозволило користувачам завантажувати безкоштовні ігри на їхні дні народження. На жаль, пристрій не зміг отримати підтримку від провідних виробників ігор того часу, і він був завданий смертельного удару катастрофою 1983 року.

У 1989 році Sega випустила свою 16-розрядну консоль Genesis як наступник своєї Sega Master System 1986 року, яка не змогла адекватно конкурувати з NES. Завдяки своїй технологічній перевазі над NES, розумному маркетингу та випуску в 1991 році гри Sonic the Hedgehog, Genesis досяг значних успіхів у порівнянні зі своїм старшим суперником. У 1991 році Nintendo випустила свою 16-розрядну консоль Super NES, розпочавши першу справжню «консольну війну». На початку-середині 1990-х було випущено безліч популярних ігор на обох консолях, включаючи нові франшизи, такі як Street Fighter II і Mortal Kombat, файтинг, який зображував кров у версії гри для Genesis. У відповідь на насильницьку гру (а також на слухання в Конгресі про насильницькі відеоігри) Sega створила раду з оцінки відеоігор у 1993 році, щоб надати описове маркування для кожної гри, що продається на домашній консолі Sega. Пізніше рада створила Загальногалузеву Раду з Оцінок Програмного Забезпечення для Розваг (Entertainment Software Rating Board),

яка й досі використовується для оцінювання відеоігор на основі вмісту. У середині 1990-х відеоігри вийшли на великий екран із виходом фільму Super Mario Bros. у 1993 році, за яким протягом наступних двох років вийшли стрічки по серіям Street Fighter і Mortal Kombat. З тих пір було випущено безліч фільмів, заснованих на відеоіграх. Маючи набагато більшу бібліотеку ігор, нижчу ціну та успішний маркетинг, Genesis до цього часу випередив SNES у Північній Америці. Але Sega не змогла досягти подібного успіху в Японії.

Справжні успіхи в «онлайнових» іграх не відбулися до випуску 16-розрядних консолей 4-го покоління на початку 1990-х років, після того, як Інтернет, як ми знаємо, став частиною суспільного надбання в 1993 році. У 1995 році Nintendo випустила Satellaview , периферійний супутниковий модем для консолі Nintendo Super Famicom. Технологія дозволила користувачам завантажувати ігри, новини та підказки з чітами безпосередньо на свою консоль за допомогою супутників. Трансляції тривали до 2000 року, але технологія так і не вийшла з Японії на світовий ринок. У період з 1993 по 1996 рік Sega, Nintendo і Atari робили ряд спроб проникнути в «онлайн-ігри» за допомогою кабельних провайдерів, але жодна з них не досягла успіху через повільні можливості Інтернету та проблеми з кабельними провайдерами. Лише після випуску Sega Dreamcast, першої у світі консолі з підтримкою Інтернету, у 2000 році, у онлайн-іграх, які ми знаємо сьогодні, були досягнуті реальні успіхи. Dreamcast постачався з вбудованим модемом 56 Кбіт/с і копією останнього браузера PlanetWeb, що робить ігри в Інтернеті основною частиною його налаштування, а не просто химерним доповненням, яким користується меншість користувачів.

Завдяки стрибку в розвитку комп’ютерних технологій п’яте покоління відеоігор започаткувало тривимірну еру ігор. У 1995 році Sega випустила свою систему Saturn, першу 32-розрядну консоль, яка запускала ігри на компакт-дисках, а не на картриджах, на п'ять місяців раніше запланованого. Цей крок мав перевершити перший набіг Sony у відеоігри, Playstation, яка продавалася на 100 доларів менше, ніж Saturn, коли вона була запущена пізніше того ж року. Наступного року Nintendo випустила свою 64-розрядну систему на основі картриджів – Nintendo 64. Хоча Sega та Nintendo кожна випустила свою частку високорейтингових 3D-ігор від бренду, таких як Virtua Fighter на Saturn і Super Mario 64 на Nintendo 64, відомі компанії з відеоігор не могли конкурувати з сильними сторонніми компаніями, які випускали ігри для Sony. Компанія домінувала на ринку відеоігор і продовжить це робити в наступному поколінні. Фактично, Playstation 2, випущена в 2000 році і здатна грати в оригінальні ігри Playstation, стане найбільш продаваною ігровою консоллю всіх часів. Playstation 2, яка була першою консоллю, яка використовувала DVD, зіткнулася з Sega Dreamcast (випущена в 1999 році), Nintendo Gamecube

(2001) і Xbox від Microsoft (2001). Dreamcast, який, на думку багатьох, випередив свій час і одна з найкращих консолей, коли-небудь створених з кількох причин, включаючи можливість онлайн-ігор, стала комерційним провалом, який поклав край консольним зусиллям Sega. Sega вимкнула систему в 2001 році, ставши відтепер компанією стороннього розробника програмного забезпечення.

У 2005 і 2006 роках Xbox 360 від Microsoft, Playstation 3 від Sony і Wii від Nintendo започаткували сучасну епоху ігор високої чіткості. Хоча Playstation 3 — єдина на той час система для відтворення Blu-ray — сама по собі була успішною, Sony вперше зіткнулася з жорсткою конкуренцією з боку своїх конкурентів. Xbox 360, який мав подібні графічні можливості до Playstation 3, отримав похвалу за свою екосистему онлайн-ігор і в 2007 році отримав набагато більше нагород критиків гри, ніж інші платформи; у ньому також була представлена Microsoft Kinect, найсучасніша система захоплення руху, яка пропонувала інший спосіб грати у відеоігри (хоча Kinect ніколи не був популярним серед основних гравців або розробників ігор). І незважаючи на те, що технологічно поступається двом іншим системам, Wii перевершила свою конкуренцію в продажах. Його чутливі до руху пульти дистанційного керування зробили ігри активнішими, ніж будь-коли раніше, допомагаючи привернути увагу значно більшої частини широкої громадськості, включаючи людей у будинках пристарілих.

З початку 2000-х років можливості Інтернету вибухнули, а технологія комп’ютерних процесорів удосконалювалася з такими швидкими темпами, що кожна нова партія ігор, графіки та консолей, здається, вириває з води попереднє покоління. Вартість технологій, серверів та Інтернету знизилася настільки, що Інтернет із блискавичною швидкістю тепер доступний і звичний, а 3,2 мільярда людей по всьому світу мають доступ до Інтернету. Згідно зі звітом індустрії комп’ютерних і відеоігор ESA за 2015 рік, щонайменше 1,5 мільярда людей з доступом до Інтернету грають у відеоігри. Інтернет-магазини, такі як Xbox Live Marketplace і Wii Shop Channel, повністю змінили те, як люди купують ігри, оновлюють програмне забезпечення, спілкуються та взаємодіють з іншими гравцями, а мережеві послуги, як-от Sony PSN, допомогли онлайн-іграм для кількох гравців досягти неймовірно нових висот. Технологія дозволяє мільйонам людей у всьому світі насолоджуватися грою як спільним заняттям. Нещодавній звіт ESA про ігри показав, що 54% геймерів вважають, що їхнє хобі допомагає їм спілкуватися з друзями, а 45% використовують ігри як спосіб провести час із сім’єю. На момент випуску Xbox 360 багатокористувацькі онлайн-ігри були невід’ємною частиною цього досвіду (особливо ігри «deathmatch», у які грали проти мільйонів однолітків у всьому світі для таких ігор, як Call of Duty Modern

Warfare). Нині багато ігор мають онлайн-компонент, який значно покращує ігровий процес та інтерактивність, часто витісняючи важливість офлайн-ігрових цілей гравця.

Ближче до кінця десятиліття і початку наступного відеоігри поширилися на платформи соціальних мереж, як-от Facebook, і мобільні пристрої, такі як iPhone, охоплюючи більш казуальну ігрову аудиторію. Як повідомляється, Rovio, компанія, яка створила гру для мобільних пристроїв Angry Birds (а пізніше й мультфільм Angry Birds), заробила колосальні 200 мільйонів доларів у 2012 році. У 2011 році Skylanders: Spyro's Adventure представила відеоігри у фізичний світ. Гра вимагала від гравців розміщувати пластикові іграшкові фігурки (продаються окремо) на аксесуар, який зчитує NFC-мітки іграшок, щоб залучити персонажів у гру. У наступні кілька років ми побачимо кілька сиквелів та інших гібридів іграшкових і відеоігор, наприклад Disney Infinity, в якому зображені персонажі Disney. З тих пір, як смартфони та магазини додатків вийшли на ринок у 2007 році, ігри зазнали ще однієї швидкої еволюції, яка змінила не лише спосіб гри людей, а й внесла ігри в основну поп-культуру у такий спосіб, якого ще не було. Швидкий розвиток мобільних технологій за останнє десятиліття призвів до вибуху мобільних ігор. Цей величезний зсув ігрової індустрії в бік мобільного ринку, особливо в Південно-Східній Азії, не тільки розширив демографічні показники ігор, але й висунув ігри на передній план уваги ЗМІ. Подібно до перших шанувальників ігор, які приєднуються до нішевих форумів, сьогоднішні користувачі об'єдналися навколо мобільних ігор, а Інтернет, журнали та соціальні мережі сповнені коментарів про нові ігри та галузеві плітки. Як завжди, блоги та форуми геймерів наповнені новими порадами щодо ігор, а такі сайти, як Macworld, Ars Technica і TouchArcade, пропонують ігри від менш відомих незалежних розробників, а також традиційних ігрових компаній.

Восьме покоління відеоігор почалося з випуску Nintendo Wii U у 2012 році, а потім Playstation 4 та Xbox One у 2013 році. Незважаючи на наявність пульта дистанційного керування з сенсорним екраном, який дозволяв грати в ігри поза телевізором та мати можливість грати в ігри Wii , Wii U був комерційним провалом — протилежністю його конкурентам — і був припинений у 2017 році. У 2016 році Sony випустила більш потужну версію своєї консолі під назвою Playstation 4 Pro, першу консоль, здатну виводити відео 4K. На початку 2017 року Nintendo випустила свого наступника Wii U, Nintendo Switch, єдину систему, яка дозволяє як телевізійні, так і портативні ігри. Microsoft випустить свою консоль Xbox One X з підтримкою 4K наприкінці 2017 року.

Перехід до мобільних технологій визначив недавню главу ігор, але хоча ігри в дорозі добре підходять для насиченого життя міленіалів, ігри на

мобільних пристроях також мають свої обмеження. Екрани телефонів невеликі, а швидкість процесора та внутрішня пам’ять на більшості мобільних телефонів обмежують можливості гри. Згідно з нещодавньою статтею VentureBeat, мобільні ігри вже стають свідками свого першого спаду. Зростання доходів сповільнилося, а витрати на ведення бізнесу та витрати на розподіл різко зросли за останні кілька років.

Хоча мобільні ігри спричинили смерть портативних ігрових пристроїв, консолі все ще процвітають, і кожне нове покоління консолей вітає нову еру технологій та можливостей. Дві галузі, які цілком можуть зіграти ключову роль у майбутньому ігор, - це віртуальна реальність і технологія штучного інтелекту. Компанію віртуальної реальності Oculus придбала Facebook у 2014 році та випустила гарнітуру Rift у 2016 році. Гарнітура ідеально підходить для використання в індустрії відеоігор і потенційно дозволить геймерам «жити» в інтерактивному, захоплюючому 3D-світі. Можливості для створення повністю інтерактивних, динамічних «світів» для MMORPG, у яких гравці могли б пересуватися, взаємодіяти з іншими гравцями та відчувати цифровий ландшафт у абсолютно новому вимірі, можуть бути в межах досяжності. За останні кілька років у світі штучного інтелекту, що обробляє мову, відбулося багато досягнень. У 2014 році Google придбала Deep Mind, IBM придбала AlchemyAPI, провідного постачальника технологій глибокого навчання; У жовтні 2015 року Apple здійснила два придбання штучного інтелекту менш ніж за тиждень. Дві галузі, які розробляються, – точність для технології розпізнавання голосу та відкритий діалог з комп’ютерами. Ці досягнення можуть означати дивовижну нову главу для ігор — особливо в поєднанні з віртуальною реальністю, оскільки вони можуть дозволити іграм взаємодіяти з персонажами в іграх, які зможуть відповідати на запитання та команди з розумними і, здавалося б, природними відповідями. У світі шутерів від першої особи, спортивних ігор і стратегічних ігор гравці могли ефективно керувати комп’ютером для виконання ігрових завдань, оскільки комп’ютер міг би розуміти команди через гарнітуру завдяки прогресу в точності розпізнавання голосу. Якщо зміни, які відбулися за останнє століття, будуть продовжуватися, схоже, що ігри в 2025 році будуть майже невпізнанними. Незважаючи на те, що Angry Birds стала популярною з моменту її випуску в 2011 році, навряд чи її згадують так тепло, як Space Invaders або Pong. Протягом свого розвитку в іграх спостерігалися численні тенденції, які йшли на спад і наплив, а потім їх повністю замінила інша технологія. Наступний розділ про ігри ще не зрозумілий, але що б не сталося, воно обов’язково буде цікавою.

1.2. Ігри-платформери

Платформер — це відеогра, в якій гравці керують персонажем, який бігає та стрибає на платформи, підлоги, виступи, сходи чи інші об’єкти, зображені на одному або прокручуваному (горизонтальному чи вертикальному) ігровому екрані. Його часто відносять до піджанру екшн-ігор. Перші ігри на платформі були розроблені на початку 1980-х років, що робить їх одним із найперших жанрів відеоігор, але термін «платформна гра» або «платформер» використовувався лише через кілька років для опису ігор. Багато істориків і шанувальників ігор вважають випуск Space Panic у 1980 році першою справжньою платформерною грою, тоді як інші вважають випуск 1981 року Donkey Kong від Nintendo першим. Хоча обговорюються, яка гра насправді започаткувала жанр платформи, зрозуміло, що ранні класики, такі як Donkey Kong, Space Panic і Mario Bros, були дуже впливовими, і всі вони доклали руку до формування жанру. Окрім того, що це один із перших і найпопулярніших жанрів відеоігор, він також є одним із жанрів, які поєднують елементи з іншого жанру, такі як левелінг та здібності персонажів, які можна знайти в рольових іграх. Є багато інших прикладів, коли платформенна гра також містить елементи з інших жанрів.

Одноекранні платформні ігри, як випливає з назви, граються на одному ігровому екрані і зазвичай містять перешкоди, яких гравець повинен уникати, і мету, яку він або вона намагається досягти. Найкращим прикладом одноекранної платформної гри є Donkey Kong, де Маріо подорожує вгору і вниз по сталевих платформах, ухиляючись і стрибаючи з бочки, які кидають на нього. Після того, як завдання на одному екрані виконано, гравець переходить на інший екран або залишається на тому ж екрані, але в обох випадках мета та цілі для наступного екрана зазвичай стають складнішими. Інші добре відомі одноекранні платформні ігри включають Burgertime, Elevator Action і Miner 2049er.

Платформери з боковою та вертикальною прокруткою можна визначити за екраном гри з прокруткою та фоном, який рухається, коли гравець рухається до одного краю ігрового екрана. Багато з цих ігор на платформі прокрутки також можна охарактеризувати кількома рівнями. Гравці подорожуватимуть по екрану, збираючи предмети, перемагаючи ворогів і виконуючи різні завдання, доки рівень не буде завершено. Після завершення вони переходять на наступний, зазвичай більш складний рівень, і продовжують. У багатьох із цих платформних ігор також кожен рівень закінчується битвою з босами, цих босів потрібно перемогти, перш ніж перейти на наступний рівень або екран. Кілька прикладів цих ігор на платформі прокрутки включають класичні ігри, такі як Super Mario Bros, Castlevania, Sonic the Hedgehog та Pitfall!

Незважаючи на меншу присутність на загальному ринку ігор, деякі платформні ігри продовжують бути успішними в сьомому поколінні консолей. У 2007 році були випущені Super Mario Galaxy і Ratchet & Clank Future: Tools of Destruction, що викликало позитивну реакцію критиків і шанувальників. Super Mario Galaxy була нагороджена найкращою грою 2007 року на популярних ігрових веб-сайтах, включаючи GameSpot, IGN і GameTrailers, і на той момент була найпопулярнішою грою всіх часів за версією GameRankings. У 2008 році LittleBigPlanet поєднала традиційну механіку 2D-платформи з моделюванням фізики та контентом, створеним користувачами, заробивши значні продажі та критичну реакцію. Electronic Arts випустила Mirror's Edge, яка поєднувала ігровий процес платформи з камерою від першої особи, але уникала рекламування гри як платформера через асоціацію, яку лейбл розробив з іграми, орієнтованими на молодшу аудиторію. 2D і 3D стилі ігрового процесу платформи; ця формула також використовувалася в Sonic Colors і Sonic Generations. Дві платформні ігри Crash Bandicoot також були випущені в 2007 і 2008 роках. Популярність 2D-платформерів знову почала зростати в 2010-х роках. В останні роки Nintendo відродила цей жанр. Новий Super Mario Bros. був випущений в 2006 році і розійшовся тиражем 30 мільйонів копій по всьому світу; це найбільш продавана гра для Nintendo DS і четверта найбільш продавана відеогра всіх часів без комплекту. Super Mario Galaxy було продано понад вісім мільйонів одиниць, тоді як Super Paper Mario, Super Mario 64 DS, Sonic Rush, Yoshi's Island DS, Kirby Super Star Ultra і Kirby: Squeak Squad також мають високі продажі, і цей жанр залишається активним.

Окрім власних успіхів Nintendo у різних платформенних іграх, зростання інді-ігор наприкінці 2000-х і в 2010-х роках допомагає стимулювати ринок платформних ігор. Загальним елементом цих нових платформних інді-ігор було посилення уваги на оповіді та розробці інноваційних елементів, яких не було в попередніх іграх платформи. У 2009 році незалежний розробник Frozenbyte випустив Trine, гру на платформі 2.5D, яка змішувала традиційні елементи з більш сучасними фізичними головоломками. Гра мала комерційний успіх, у підсумку було продано понад 1,1 мільйона копій. Це породило продовження, Trine 2, яке було випущено в 2011 році. У 2017 році було випущено ряд 3D-платформерів, що викликало спекуляцію в ЗМІ щодо оновлення жанру. Ці ігри включали Yooka-Laylee і A Hat in Time, обидві з яких були профінансовані на веб-сайті Kickstarter. Випуск Super Mario Odyssey на Nintendo Switch, який був оновленим поверненням до відкритого стилю гри, популяризованого Super Mario 64, є однією з найбільш продаваних і рецензованих ігор в історії франшизи. Super Lucky's Tale, на той час ексклюзив для Xbox One і HD-ремастер Voodoo Vince для Microsoft Windows і Xbox One. Snake Pass вважали «головоломкою-платформером без кнопки стрибка». Збірник Crash Bandicoot N. Sane Trilogy для PlayStation 4 був проданий

тиражем понад два мільйони копій по всьому світу, і деякі критики відзначили підвищену складність оригінальних ігор PlayStation. Приблизно в цей час з'явилися ремейки інших класичних 3D-платформерів, а саме трилогія Spyro Reignited, SpongeBob SquarePants: Battle for Bikini Bottom – Rehydrated (2020) і ремейк MediEvil 1998 року для PS4. Astro Bot Rescue Mission (2018) — це ексклюзивна гра для PlayStation VR, за якою слідує Astro's Playroom (2020), яка є попередньо встановленою грою на PlayStation 5. Серію Skylanders від Activision можна також вважати прабатьком франшиз Crash Bandicoot і Spyro, що повертаються. до прожектора. Версія Skylanders: Giants (2012) для Nintendo 3DS, відмінна від свого консольного аналога, була першою в серії, яка містила більш традиційні елементи платформи. Sackboy: A Big Adventure (2020), розроблена Sumo Digital, була однією з перших ігор PlayStation 5. Crash Bandicoot 4: It's About Time також був випущений у 2020 році на похвалу критиків. Super Mario 3D World + Bowser's Fury (2021) був перевипуском оригінальної гри Wii U з абсолютно новою та меншою кампанією, яка подолала розрив між її ігровим процесом і Odyssey. Ratchet & Clank: Rift Apart (2021) була однією з перших ексклюзивних ігор для PlayStation 5, створених Insomniac під лейблом «Sony Game Studios». Фінансований Kickstarter Psychonauts 2 був випущений 25 серпня 2021 року і отримав схвалення критиків.

1.3. Розробка ігор

Розробка ігор – це процес створення ігор від концепту до фінального продукту. Подібно до виробничої лінії, конвеєр розробки ігор допомагає організувати потік роботи, щоб кожен знав, що і коли їм потрібно поставити. Конвеєр також допомагає керувати графіком розробки гри та бюджетом, зменшуючи неефективність та вузькі місця. Хоча конвейери різняться між проектами та студіями, процес досить подібний, незалежно від того, працюєте ви над AAA, інді чи мобільною грою. Гра постійно розвивається, і речі, які звучали чудово в теорії, можуть не працювати так добре в реальності. Отже, конвеєр не обов'язково є лінійним процесом. Робота має бути надіслана на творче затвердження і часто може бути відправлена назад на доопрацювання. Конвеєри повинні бути достатньо гнучкими, щоб врахувати зміни та зміни курсу. Розробка відеоігор зазвичай поділяється на 3 етапи: підготовка, виробництво та постпродакшн.

Підготовка - з цього починається кожен проект. По суті, попередня підготовка визначає, про що йдеться у грі, чому вона повинна бути створена та що потрібно для її створення. У вас може бути чудова ідея для типу гри,

історії, яку ви хочете втілити в життя, або ви можете створити таку, яка використовує певний тип технології (наприклад, VR, новий контролер або консоль). Під час підготовки до виробництва знаходять відповіді на такі запитання:

- Про що гра?
- Хто глядач?
- Чи є для цього ринок? Яка конкуренція?
- На якій платформі він буде опублікований?
- Як це буде монетизуватися? Чи буде він продаватися на платформі чи безкоштовно грати з покупками в грі?
- Скільки часу знадобиться на розвиток?
- Які кадри та ресурси для цього знадобляться?
- Який кошторисний бюджет?

Цей етап може тривати від тижня до року, залежно від типу проекту, наявних ресурсів і фінансів, і зазвичай займає до 20% загального часу виробництва. На даний момент команда досить мала. Тут може бути продюсер, програміст, концепт-художник (або якщо ви працюєте з однією особою, ви будете робити більшу частину цього!). Продюсер відеоігор займається бізнес-аспектом проекту, зокрема фінансовими. Вони керують бюджетом і розробляють маркетингові стратегії для продажу продукту. Концепт-художник задає тон проекту на початку, розробляючи ілюстрації та ескізи. Ці ранні візуальні елементи допомагають сформувати мову гри, надаючи кожному, хто працює над проектом, візуальний посібник із загального вигляду та відчуття. Інформація, зібрана на цьому етапі підготовки до виробництва, становить основу дизайн документа гри.

Документ дизайну гри по суті є північною зіркою гри. Це живий документ, який допомагає кожному зрозуміти та приєднатися до більшого бачення проекту. Дизайн документ включає такі речі, як:

- Ідея чи концепція
- Жанр
- Історія та персонажі
- Основна механіка гри
- Ігровий процес
- Рівень і дизайн світу
- Мистецтво та/або ескізи
- Стратегія монетизації

Як живий документ, дизайн докумет постійно оновлюється та вдосконалюється протягом усього виробництва. Це може бути пов’язано з

технічними чи фінансовими обмеженнями або просто усвідомленням того, що все виглядає, грає чи працює не так добре, як ви сподівалися спочатку. Багато людей, особливо дрібні розробники, люблять використовувати більш гнучкі методи розробки, які не стосуються процесів і документації, а більше просто створення речей. Однак більші студії вважають за краще інший підхід. EA, Microsoft, Sony, Ubisoft та інші великі ігрові компанії керуються процесами і вимагають великої документації. Це велика частина того, як вони знову і знову досягали успіху. Дизайн документ тримає проект організованим, допомагає визначити потенційні ризики і дає змогу заздалегідь побачити, кого, можливо, доведеться найняти або аутсорсувати, щоб втілити проект у життя. Ваша ідея гри може здатися досить простою, але як тільки ви викладете її в дизайн документ, ви незабаром зрозумієте, наскільки великий і важкий ваш проект. Проекти без плану, швидше за все, будуть перевищувати закладений час та бюджет. Ще одна причина мати дизайн документ — допомогти запустити в розробку та фінансувати гру. Потенційні інвестори захочуть побачити надійний план, перш ніж інвестувати. Нарешті, дизайн документ допоможе продати продукт, коли він буде готовий до випуску.

Прототип відеоігри — це необроблений тест, який перевіряє функціональність, користувацький досвід, ігровий процес, механіку та напрямок мистецтва. Прототипування відбувається під час попереднього виробництва, щоб перевірити, чи спрацює ідея гри, і чи варто її продовжити. Багато ідей не проходять цей етап. Команда часто починає з дизайну на папері, щоб швидко, легко та економічно ефективно перевірити теорії та розробити багато нюансів гри чи серії систем. Хоча ідеї, психологія, теорії та інші глибокі метафори мислення важливі, гру можна розробляти лише в голові чи на папері. Більшість ідей для ігор потрібно торкнутися, відчути, відтворити та перевірити якомога раніше. Мета полягає в тому, щоб якнайшвидше запустити прототип, щоб перевірити, чи справді ідеї закладені у гру працюють, і чи вона настільки весела, як розробники сподівалися. Прототипування також може виявити несподівані проблеми, які потенційно можуть змінити весь хід проекту. Важливо, щоб інші нерозробники також тестували прототип, тому що речі, які є очевидними для одних, можуть не бути для інших. Асети-заповнювачі використовуються для економії часу та грошей. Ці низькоякісні асети замінюють такі речі, як зброя та реквізит, під час раннього етапу тестування, і якщо вони схвалені, вони замінюються остаточними високоякісними версіями пізніше. Асети-заповнювачі можна придбати або знайти безкоштовно в Інтернеті за допомогою програмного забезпечення для розробки ігор. Вони, як правило, мають досить базові форми, але також можуть бути трохи більш просунутими, наприклад, цей пакет асетів Soul: Cave від Epic Games для Unreal Engine 4:

Рис. 1.3.1 – Малюнок асетів Soul: Cave

Рис. 1.3.2 – Пакет асетів Soul: Cave

Виробництво — це найдовша стадія розробки ігор. Тривалість від 1 до 4 років, виробницто – це те, де гра дійсно починає формуватися. Історія вдосконалена, асети (персонажі, істоти, реквізит та середовище) створені, правила гри встановлюються, рівні та світи будуються, код написаний та багато іншого. Майже все у відеогрі є свідомим рішенням. Сюди входять кожен персонаж, середовище, об’єкт, а також зовнішній вигляд, кольори, звуки, рівень складності, правила та система нарахування балів. Однак початкові ідеї не завжди так добре втілюються в реальність, тому в міру того, як робота виконується, гра постійно тестується та вдосконалюється.

Розглянемо основні етапи виробництва ігор і деякі з ключових завдань розробки відеоігор, пам’ятаючи, що меншим командам потрібно буде виконувати кілька ролей, тоді як у більшій студії буде більше співробітників, багато з яких спеціалізуються на особливий аспект виробництва. Протягом процесу розробки гри необхідно досягти кількох етапів:

- **Прототип**: це початковий тест гри (який відбувається під час попереднього виробництва і детально описано вище). Деякі ігри можуть ніколи не пройти цей етап.
- **Перший придатний для гри білд**: дає набагато краще уявлення про зовнішній вигляд та ігровий процес. Хоча це ще далеко від остаточного варіанту, асетиі замінюються об’єктами більш високої якості, а також додаються ілюстрації.
- **Вертикальний зріз**: вертикальний зріз — це повністю відтворюваний зразок, який можна використовувати для презентації гри студіям або інвесторам. Від кількох хвилин до півгодини, вертикальний зріз забезпечує повний досвід від гри.
- **Пре-альфа**: більшість вмісту розроблено на стадії попереднього альфа-версії. На цьому етапі розробки гри потрібно буде прийняти деякі важливі рішення. Вміст може бути вирізано, або потрібно буде додати нові елементи, щоб покращити ігровий процес.
- **Альфа**: всі основні функції додано і гра повністю доступна від початку до кінця. Можливо, деякі елементи, як-от арт-об’єкти, все одно потрібно буде додати, але елементи керування та функціональні можливості мають працювати належним чином. Тестери контролю якості перевірятимуть, чи все працює безперебійно, і повідомлятимуть команді про помилки.
- **Бета-версія**: на цьому етапі весь вміст і асети інтегровані, і команда повинна бути зосереджена на оптимізації, а не на додаванні нових функцій.
- **Відправка на золото**: гра є остаточною і готова до відправки у видавництво та опублікування.

Після завершення виробництва та відправлення гри процес розробки гри продовжується, коли деякі члени команди переходять на технічне обслуговування (виправлення помилок, створення виправлень) або створення бонусного або завантажуваного контенту (DLC). Інші можуть перейти до продовження або наступного проекту. Можна провести огляд, щоб обговорити, що спрацювало/не спрацювало, і визначити, що можна було б зробити краще наступного разу. Вся проектна документація, активи та код доопрацьовуються, збираються та зберігаються, якщо вони знадобляться в майбутньому.

РОЗДІЛ 2. ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1. Мова програмування С++

Історія мови програмування С++ сягає 1979 року, коли Б'ярн Страуструп працював над докторською роботою. Однією з мов, з якою Страуструп мав можливість працювати, була мова під назвою Simula, яка, як випливає з назви, є мовою, в першу чергу, призначеною для моделювання. Мова Simula 67 - варіант, з яким працював Страуструп - вважається першою мовою, яка підтримує парадигму об'єктно-орієнтованого програмування. Страуструп виявив, що ця парадигма була дуже корисною для розробки програмного забезпечення, однак мова Simula була занадто повільною для практичного використання.

Невдовзі після цього він розпочав роботу над «С з класами», яка, як випливає з назви, мала бути наднабором мови С. Його метою було додати об’єктно-орієнтоване програмування в мову С, яка була і залишається мовою, що користується пошаною за її переносимість, не жертвуючи швидкістю або низькорівневою функціональністю. Його мова включала класи, базову спадковість, вбудовування, аргументи функцій за замовчуванням і сильну перевірку типів на додаток до всіх можливостей мови С.

Перший компілятор С з класами називався Cfront, який був похідним від компілятора С під назвою CPre. Це була програма, розроблена для перекладу коду С з класами на звичайний С. Досить цікавим моментом, на який варто звернути увагу, є те, що Cfront був написаний здебільшого на С з класами, що робило його компілятором із власним хостингом (компілятором, який може компілювати сам себе). Згодом Cfront був залишений у 1993 році після того, як стало важко інтегрувати в нього нові функції, а саме винятки С++. Тим не менш, Cfront зробив величезний вплив на реалізації майбутніх компіляторів і на операційну систему Unix.

У 1983 році назву мови було змінено з С з класами на С++. Оператор ++ у мові С — це оператор збільшення змінної, який дає деяке уявлення про те, як Страуструп ставився до мови. Приблизно в цей час було додано багато нових функцій, найпримітнішими з яких є віртуальні функції, перевантаження функцій, посилання із символом &, ключове слово const та однорядкові коментарі з використанням двох прямих слешей (що є функцією, взятою з мови BCPL). У 1985 році було опубліковано посилання Страуструпа на мову під назвою «Мова програмування С++». Того ж року С++ був реалізований як комерційний продукт. Мова ще не була офіційно стандартизована, що робить

книгу дуже важливим довідником. У 1989 році мова була знову оновлена, щоб включити захищені та статичні члени, а також успадкування від кількох класів. У 1990 році було випущено Анотований довідковий посібник C++. У тому ж році компілятор Borland Turbo C++ буде випущений як комерційний продукт. Turbo C++ додав безліч додаткових бібліотек, які мали б значний вплив на розвиток C++. Хоча останній стабільний випуск Turbo C++ був у 2006 році, компілятор все ще широко використовується.

У 1998 році комітет стандартів C++ опублікував перший міжнародний стандарт C++ ISO/IEC 14882:1998, який буде неофіційно відомий як C++98. Вважалося, що анотований довідковий посібник із C++ мав великий вплив на розробку стандарту. Була також включена стандартна бібліотека шаблонів, яка розпочала свою концептуальну розробку в 1979 році. У 2003 році комітет відповів на численні проблеми, про які повідомлялося в стандарті 1998 року, і переглянув його відповідно. Змінена мова отримала назву C++03. У 2005 році комітет стандартів C++ опублікував технічний звіт (отриманий під назвою TR1), в якому детально описувалися різні функції, які вони планували додати до останнього стандарту C++. Новий стандарт неофіційно отримав назву C++0x, оскільки очікувалося, що він буде випущений десь до кінця першого десятиліття. Як не дивно, але новий стандарт не буде випущений до середини 2011 року. До того часу було опубліковано кілька технічних звітів, і деякі компілятори почали додавати експериментальну підтримку нових функцій.

У середині 2011 року новий стандарт C++ (C++11) був завершений. Проект бібліотеки Boost зробив значний вплив на новий стандарт, і деякі з нових модулів були отримані безпосередньо з відповідних бібліотек Boost. Деякі з нових функцій включали підтримку регулярних виразів (докладні відомості про регулярні вирази можна знайти тут), повну бібліотеку рандомізації, нову бібліотеку часу C++, підтримку atomics, стандартну бібліотеку потоків (якої до 2011 року не вистачало як C, так і C++). , новий синтаксис циклу for, що забезпечує функціональність, подібну до циклів foreach в деяких інших мовах, ключове слово auto, нові класи контейнерів, кращу підтримку об’єднань і списків ініціалізації масивів, а також змінні шаблони.

Філософія мови програмування C++ була викладена в «Дизайн та еволюція C++» (1994) Б'ярна Страуструпа. Розуміння філософії чи правил C++ допомагає нам зрозуміти, чому певні речі в C++ є такими, якими вони є. Ось короткий зміст філософії C++:

- Програміст повинен мати право вибирати свою власну парадигму програмування (процедурне програмування, об’єктно-орієнтоване програмування, загальне програмування та абстракція даних)

- Мова повинна бути сумісна з C. Перехід з C на C++ не повинен бути складним.
- Кожна функція в мові повинна бути побудована для загальної мети. Функції не повинні бути специфічними для платформи.
- Мова має бути розроблена для роботи без складного середовища програмування (код C++ можна навіть написати на простому блокноті).
- Мова має бути статично типізованою та для загальних цілей. Він повинен бути таким же портативним, як і C, тобто код, написаний на одному комп'ютері, повинен бути використаний на іншому комп'ютері без будь-яких змін у коді.
- Мова повинна надавати програмістам можливість зробити власний вибір (вибір між різними типами змінних, виділення та звільнення пам'яті відповідно до потреб програми тощо), навіть якщо це збільшує ймовірність того, що програміст вибере неправильний вибір.
- Мова не повинна сповільнювати програму або споживати простір (накладні витрати) для функцій, які не використовуються в коді.
- Під C++ не повинно бути жодної мови, крім мови асемблера.

Інтерфейс — це структура програмування, яка описує поведінку класу в C++ без визначення конкретної реалізації цього класу. Наприклад, якщо у нас є клас Car і клас Scooter, то кожен з цих класів повинен мати дію (функцію) LightsOn(). Оскільки обидва класи мають однакову дію, ми можемо створити інтерфейс для скорочення коду та підвищення ефективності програми. Але те, як буде вмикатися світло, буде залежати від реалізації функції LightsOn() у кожному класі. Іншими словами, обидва класи матимуть батьківський клас, який містить у собі функцію LightsOn(). Але конкретні подробиці того, як вмикатимуться світло, будуть пояснені у відповідних класах. Деякі з найважливіших аспектів інтерфейсу:

- Це має бути легко зрозуміти
- Він не повинен бути схильним до помилок
- Це має сприяти ефективному використанню програми

Інтерфейси та абстрактні класи більш-менш передають ту саму ідею. Ось чому інтерфейси C++ реалізуються за допомогою абстрактних класів. Клас із чистою віртуальною функцією називається абстрактним класом. Програміст може створити чисту віртуальну функцію, написавши «= 0» в оголошенні функції. Інтерфейс або абстрактний клас використовується як базовий клас для інших класів, які успадковують абстрактний клас. Він забезпечує загальну функціональність класу, щоб інші класи могли легко використовувати чисті віртуальні функції класу.

Стандартна бібліотека в C++ заснована на конвенціях, введених у стандартній бібліотеці шаблонів (STL) і стандартній бібліотеці C з деякими змінами в ній. Бібліотека стандартних шаблонів надає різні корисні функції, такі як контейнери (наприклад, вектори), ітератори (узагальнені покажчики) для доступу до контейнерів, як-от масиви, і алгоритми для виконання різних операцій (наприклад, сортування та пошуку). Програміст може використовувати ці шаблони для написання загальних алгоритмів, які можуть працювати з будь-яким контейнером. Оскільки STL складається з набору класів шаблонів (векторів, масивів, черги тощо), це узагальнена бібліотека, яка не залежить від типів даних. Це свого роду план, який містить усі параметризовані компоненти. Щоб працювати зі стандартною бібліотекою шаблонів, ми повинні знати, як працювати з різними компонентами та можливостями класів шаблонів. Програміст може отримати доступ до різних функцій стандартної бібліотеки в C++, використовуючи шістдесят дев'ять стандартних заголовків, наданих в C++ (дев'ятнадцять з яких більше не використовуються). Ці функції можна оголосити за допомогою простору імен std у коді. Використання стандартної бібліотеки допомагає нам уникнути написання коду з нуля. Це допомагає заощадити час, оскільки багато речей, необхідних для коду, вже є в стандартній бібліотеці C++.

У порівнянні з C, C++ представив багато нових функцій у мові. Деякі з важливих особливостей C++:

1. **Простота**: однією з причин, чому C++ є першою мовою програмування багатьох програмістів, є те, що C++ є простим і легким для вивчення. Незважаючи на те, що C++ зручний для початківців, він широко використовується для створення численних просунутих програм.
2. **Об’єктно-орієнтованість**: C++ – це об’єктно-орієнтована мова програмування. Об’єкти полегшують розробку та обслуговування програмного забезпечення. За допомогою цих об’єктів ми можемо виконувати абстракцію даних, інкапсуляцію даних, успадкування, поліморфізм та приховування даних.
3. **Динамічне виділення пам'яті**: C++ підтримує динамічне виділення пам'яті. За допомогою вказівників у C++ можена динамічно розподіляти пам’ять.
4. **Вказівники**: вказівник — це змінна, яка зберігає адресу іншої змінної. C++ підтримує використання вказівників. Вони використовуються для взаємодії з місцем розташування будь-якої змінної у пам’яті комп’ютера.
5. **Широка бібліотека**: C++ пропонує величезну бібліотеку, повну вбудованих функцій, які полегшують програмістові кодування.

Доступ до функцій бібліотеки можна отримати за допомогою різних файлів заголовків.

6. **Компільованість**: на відміну від інших мов, таких як Python і Java, які базуються на інтерпретаторі, C++ є мовою, заснованою на компіляторі. Отже, C++ набагато швидше, ніж Python або Java.
7. **Перевантаження оператора**: C++ підтримує перевантаження операторів. Це означає, що C++ може надати операторам особливе значення для будь-якого даного типу даних. Він надає програмістам можливість створення нових визначень операторів C++.
8. **Чутливий до регістру**: як і C, C++ чутливий до регістру. Це означає, що C++ по-різному трактує малі та великі літери.
9. **Мова програмування високого рівня**: на відміну від C, яка є мовою програмування середнього рівня, C++ є мовою високого рівня. Це полегшує програмісту роботу на C++, оскільки ми можемо тісно пов’язати його з англійською мовою.
10. **Обробка винятків**: C++ підтримує обробку винятків. Це допомагає програмісту вловити помилку(и), якщо в програмі виникає помилка.
11. **Переносимість**: програми на C++ можна виконувати на різних машинах з незначними змінами в коді або без них. Але C++ не залежить від платформи. Припустимо, ми зібрали програму в операційній системі Windows. У цьому випадку цей скомпільований файл (файл .exe) не працюватиме в операційній системі Mac. Але файл .cpp, створений у Windows, буде ідеально працювати в операційній системі Mac.

2.2. Ігровий рушій Unreal Engine

Unreal Engine — ігровий рушій 3D комп’ютерної графіки, розроблений Epic Games, вперше показаний у шутері від першої особи Unreal 1998 року. Спочатку він був розроблений для шутерів від першої особи для ПК, але з тих пір використовувався в різних жанрах тривимірних ігор і знайшов поширення в інших галузях, особливо в кіно- та телевізійній індустрії. Написаний на C++, Unreal Engine має високий ступінь портативності, підтримуючи широкий спектр настільних, мобільних, консольних і віртуальних платформ.

Останнє покоління Unreal Engine 5 було запущено в квітні 2022 року. Як і його попередник, випущений у березні 2014 року, його вихідний код доступний на GitHub після реєстрації облікового запису, а комерційне використання

надається на основі моделі роялті. Epic відмовляється від своїх роялті за ігри до тих пір, поки розробники не отримають 1 мільйон доларів США прибутку, а комісія не буде знята, якщо розробники публікують у Epic Games Store. Epic включив функції придбаних компаній, як-от Quixel, у рушій, чому, як вважають, допомогли доходи Fortnite.

Перше покоління Unreal Engine було розроблено Тімом Суїні, засновником Epic Games. Створивши інструменти для редагування для своїх умовно-безкоштовних ігор ZZT (1991) і Jill of the Jungle (1992), Суїні почав писати рушій у 1995 році для виробництва гри, яка пізніше стала шутером від першої особи, відомою як Unreal. Після багатьох років розробки, вона дебютувала з випуском гри в 1998 році, хоча MicroProse і Legend Entertainment мали доступ до технології набагато раніше, ліцензувавши її в 1996 році. Згідно з інтерв'ю, Суїні написав 90 відсотків коду в рушії, включаючи графіку, інструменти та мережеві підсистеми. Спочатку рушій повністю покладався на програмне відтворення, тобто графічні обчислення оброблялися центральним процесором. Однак з часом він зміг скористатися можливостями, наданими виділеними відеокартами, зосередившись на Glide API, спеціально розробленому для прискорювачів 3dfx. Хоча OpenGL і Direct3D підтримувалися, вони працювали повільніше порівняно з Glide через недоліки в управлінні текстурами в той час. Суіні особливо розкритикував якість драйверів OpenGL для споживчого обладнання, описуючи їх як «надзвичайно проблематичні, помилкові та неперевірені» та позначив код у реалізації як «страшний», на відміну від більш простої та чистішої підтримки Direct3D. Що стосується аудіо, Epic використовувала Galaxy Sound System, програмне забезпечення, створене на мові асемблера, яке інтегрувало технології EAX і Aureal, і дозволяло використовувати трекерну музику, що давало дизайнерам рівнів гнучкість у тому, як саундтрек гри відтворювався в певний момент. в картах. Стів Полдж, автор плагіна Reaper Bots для Quake, запрограмував систему штучного інтелекту на основі знань, які він отримав під час роботи в IBM, розробляючи протоколи маршрутизаторів. За словами Суіні, найскладнішим у програмуванні рушія був рендер, оскільки йому довелося кілька разів переписувати його основний алгоритм під час розробки, хоча він виявив менш «гламурною» інфраструктуру, що з'єднує всі підсистеми. Незважаючи на те, що він потребує значних особистих зусиль, він сказав, що рушій був його улюбленим проектом в Epic, додавши: «Написання першого Unreal Engine було 3,5-річним першим оглядом сотень унікальних тем у програмному забезпеченні та було неймовірно просвітницьким». Серед його функцій були виявлення зіткнень, кольорове освітлення та обмежена форма фільтрації текстур. Рушій також інтегрував редактор рівнів UnrealEd, який підтримував конструктивні операції геометрії твердого тіла в реальному часі ще в 1996 році, дозволяючи дизайнерам рівнів змінювати макет рівня на льоту.

Незважаючи на те, що Unreal був розроблений для конкуренції з id Software (розробник Doom і Quake), співзасновник Джон Кармак похвалив гру за використання 16-бітового кольору і зазначив, що в ній реалізовані візуальні ефекти, такі як об'ємний туман. «Я сумніваюся, що будь-яка важлива гра відтепер буде розроблена з урахуванням 8-бітового кольору. Unreal зробив важливу справу, просунувшись до прямого кольору, і це дає художникам набагато більше свободи», – сказав він у статті, написаній Джефф Кейлі для GameSpot. «Світлі розквіти [сфери світла], об'єми туману та композиційне небо – це були кроки, які я планував зробити, але Епіс досягла цього першою з Unreal», – сказав він, додавши: «Рушій Unreal підняв планку того, що очікують від екшн-ігор. Візуальні ефекти, які вперше побачили в грі, стануть очікуваними від майбутніх ігор». Unreal був відомий своїми графічними інноваціями, але в 1999 році в інтерв'ю Eurogamer Суїні визнав, що багато аспектів гри були недороблені, посилаючись на скарги геймерів на високі системні вимоги та проблеми з онлайн-грою. Епіс розглянув ці моменти під час розробки Unreal Tournament, включивши кілька покращень у рушій, призначений для оптимізації продуктивності на недорогих машинах та покращення мережевого коду, а також удосконалюючи штучний інтелект для ботів для відображення координації в командних середовищах. Гра також поставлялася з підвищеною якістю зображення з підтримкою алгоритму стиснення S3TC, що дозволяє створювати 24-бітні текстури високої роздільної здатності без шкоди для продуктивності. На додаток до доступності в Microsoft Windows, Linux, Mac і Unix, рушій був портований через Unreal Tournament на PlayStation 2 і, за допомогою Secret Level, на Dreamcast. Наприкінці 1999 року The New York Times вказала, що було шістнадцять зовнішніх проектів з використанням технології Епіс, включаючи Deus Ex, The Wheel of Time і Duke Nukem Forever, останній з яких спочатку був заснований на рушії Quake II. На відміну від id Software, компанія Епіс пропонувала не лише вихідний код, Епіс надала підтримку ліцензіатам і зустрічалася з ними, щоб обговорити вдосконалення системи розробки ігор. Незважаючи на те, що виробництво коштувало близько 3 мільйонів доларів США та ліцензія на суму до 350 000 доларів, Епіс надала гравцям можливість модифікувати свої ігри за допомогою включення UnrealEd та мови сценаріїв під назвою UnrealScript, що створило спільноту ентузіастів навколо рушія, створеного, щоб бути розширюваним на протязі декількох поколінь ігор. «Велика мета з технологією Unreal весь час полягала в тому, щоб створити базу коду, яку можна було б розширити та покращити за допомогою багатьох поколінь ігор. Щоб досягти цієї мети, потрібно було підтримувати технологію досить загального призначення, писати чистий код і розробляти рушій, щоб він був дуже розширюваним. Ранні плани розробки розширюваного рушія кількох поколінь дали нам велику перевагу в ліцензуванні технології, коли вона

досягла завершення. Після того, як ми уклали кілька ліцензійних угод, ми зрозуміли, що це законний бізнес. Відтоді це стало основним компонентом нашої стратегії.» - Тім Суїні, Maximum PC, 1998.

Рис. 2.2.1 – Скріншот гри Гаррі Поттер і Філософський камінь, розробленої на Unreal Engine

У жовтні 1998 року IGN на основі інтерв'ю з філією Voodoo Extreme повідомила, що Суіні проводить дослідження для свого рушія наступного покоління. Оскільки розробка почалася через рік, друга версія дебютувала в 2002 році разом із American's Army, безкоштовним багатокористувацьким шутером, розробленим армією США як пристрій для вербування. Незабаром після цього Epic випустила Unreal Championship на Xbox, причому це була одна з перших ігор, яка використовує Xbox Live від Microsoft. Хоча це покоління було засноване на своєму попереднику, воно відчуло помітний прогрес у термінах візуалізації, а також нові вдосконалення набору інструментів. Здатний запускати рівні майже в 100 разів більш детально, ніж у Unreal, рушій інтегрував різноманітні функції, включаючи інструмент кінематографічного редагування, системи частинок, плагіни експорту для 3D Studio Max і Maya, а також систему скелетної анімації, яку вперше продемонстрували у версії Unreal Tournament для PlayStation 2. Крім того, інтерфейс користувача UnrealEd був переписаний на C++ за допомогою набору інструментів wxWidgets, який, за словами Суіні, був «найкращим доступним» на той час. Epic використовував фізичний рушій Karma, стороннє програмне забезпечення від британської студії Math Engine, щоб управляти фізичним моделюванням, таким як зіткнення гравців у ragdoll та довільна динаміка твердого тіла. З Unreal Tournament 2004 був успішно реалізований

ігровий процес на базі автомобілів, що дозволило широкомасштабні битви. Хоча Unreal Tournament 2003 підтримував фізику транспортних засобів через рушій Karma, як продемонструвала тестова карта з «наспіх сконструйованим транспортним засобом», лише після того, як Psyonix створив модифікацію базового коду Epic, гра отримала повністю закодовані транспортні засоби. Вражений їхніми зусиллями, Epic вирішив включити його у свого наступника як новий режим гри під назвою Onslaught, найнявши Psyonix як підрядника. Psyonix пізніше розробить Rocket League, перш ніж бути придбаним Epic у 2019 році. Спеціалізована версія UE2 під назвою UE2X була розроблена для Unreal Championship 2: The Liandri Conflict на оригінальній платформі Xbox з оптимізацією, характерною для цієї консолі. У березні 2011 року Ubisoft Montreal повідомила, що UE2 успішно працює на Nintendo 3DS через Tom Clancy's Splinter Cell 3D. «3DS є потужним, і ми можемо запустити Unreal Engine на цій консолі, що є досить вражаючим для портативної машини, і 3D не впливає на продуктивність (завдяки моїм дивовижним програмістам)», — сказали в Ubisoft.

Рис. 2.2.2 – Скріншот гри Killing Floor,

розробленої на Unreal Engine 2

Скріншоти Unreal Engine 3 були представлені в липні 2004 року, коли рушій уже розроблявся понад 18 місяців. Він був заснований на першому поколінні, але містив нові функції. «Основні архітектурні рішення, які бачать програмісти об’єктно-орієнтованого дизайну, підхід до написання сценаріїв на основі даних і досить модульний підхід до підсистем все ще залишаються [з Unreal Engine 1]. Але частини гри, які дійсно помітні для геймерів – рендерер,

фізична система, звукова система та інструменти – все помітно нове і значно потужніше», – сказав Суїні. На відміну від Unreal Engine 2, Unreal Engine 3 був розроблений для використання переваг повністю програмованого обладнання шейдерів. Усі розрахунки освітлення та тіні були зроблені на піксель, а не на вершину. Що стосується візуалізації, Unreal Engine 3 забезпечив підтримку рендерера високого динамічного діапазону з коректним гамма. Першими іграми, випущеними з використанням Unreal Engine 3, були Gears of War для Xbox 360 і RoboBlitz для Windows, обидві були випущені 7 листопада 2006 року. Спочатку Unreal Engine 3 підтримував лише платформи Windows, PlayStation 3 і Xbox 360, тоді як iOS (вперше продемонстровано за допомогою Epic Citadel) і Android були додані пізніше в 2010 році, причому Infinity Blade став першою назвою для iOS, а Dungeon Defenders — першою для Android. У 2011 році було оголошено, що рушій підтримуватиме Adobe Flash Player 11 через API-інтерфейси з апаратним прискоренням Stage 3D і що використовується в двох іграх Wii U, Batman: Arkham City і Aliens: Colonial Marines. У 2013 році Epic об'єдналася з Mozilla, щоб вивести Unreal Engine 3 в Інтернет; використовуючи підмову asm.js і компілятор Emscripten, вони змогли портувати рушій за чотири дні. Протягом усього терміну існування UE3 були включені значні оновлення, включаючи покращене середовище, що руйнується, динаміку м’якого тіла, симуляцію великого натовпу, функціональність iOS, інтеграцію Steamworks, глобальне рішення для освітлення в реальному часі та стереоскопічний 3D на Xbox 360 через TriOviz for Games Technology. Підтримка DirectX 11 була продемонстрована з демонстрацією Samaritan, яка була представлена на конференції розробників ігор у 2011 році та створена Epic Games у тісному партнерстві з Nvidia, з інженерами, що працюють по всій країні, щоб підняти графіку в режимі реального часу на новий рівень. Хоча Unreal Engine 3 був досить відкритим для моддерів, можливість публікувати та продавати ігри, що означали використання UE3, була обмежена ліцензіями движка. Однак у листопаді 2009 року Epic випустила безкоштовну версію SDK UE3, яка називається Unreal Development Kit (UDK), яка доступна для широкої публіки. У грудні 2010 року комплект було оновлено, щоб включити підтримку створення ігор і програм для iOS. Сумісність з OS X з’явилася у вересні 2011 року.

Рис. 2.2.3 – Скріншот з демо Samaritan, якою презентували Unreal Engine 3

У серпні 2005 року Марк Рейн, віце-президент Epic Games, повідомив, що Unreal Engine 4 розроблявся протягом двох років. "Люди цього не усвідомлюють, але ми вже два роки в розробці Unreal Engine 4. Це Звичайно, у нього ще немає повної команди, це лише один хлопець, і ви, напевно, можете здогадатися, хто цей хлопець", - сказав він C&VG. Виступаючи в інтерв'ю на початку 2008 року, Суїні заявив, що він був, по суті, єдиною людиною, яка працювала над рушієм, хоча він підтвердив, що його відділ досліджень і розробок почне розширюватися пізніше того ж року, розробляючи рушій паралельно з зусиллями команди UE3. . У лютому 2012 року Рейн сказав, що «люди будуть шоковані пізніше цього року, коли побачать Unreal Engine 4»; Epic представила UE4 обмеженій кількості учасників на конференції розробників ігор у 2012 році, а відео рушія, яке демонструє технічний художник Алан Віллард, було опубліковано 7 червня 2012 року на GameTrailers TV. Однією з головних функцій, запланованих для UE4, було глобальне освітлення в режимі реального часу за допомогою трасування воксельних конусів, що виключає попередньо обчислене освітлення. Однак ця функція, яка називається Sparse Voxel Octree Global Illumination (SVOGI) і продемонстрована в демонстрації Elemental, була замінена подібним, але менш дорогим алгоритмом через проблеми з продуктивністю. UE4 також включає нову систему візуальних сценаріїв "Blueprints" (наступник "Kismet" UE3), яка дозволяє швидко розвивати ігрову логіку без використання коду, що призводить до зменшення розриву між технічними художниками, дизайнерами та програмістами. 19 березня 2014 року на конференції

розробників ігор (GDC) Epic Games випустила Unreal Engine 4 через нову модель ліцензування. За щомісячну підписку вартістю 19 доларів США розробники отримали доступ до повної версії рушія, включаючи вихідний код C++, який можна було завантажити через GitHub. На будь-який випущений продукт нараховувалося 5% роялті від валового доходу. Першою грою, випущеною на Unreal Engine 4, була Daylight, розроблена з раннім доступом до рушія і випущена 29 квітня 2014 року. 4 вересня 2014 року Epic безкоштовно випустила Unreal Engine 4 для шкіл та університетів, включаючи особисті копії для студентів, які навчаються в акредитованих програмах розробки відеоігор, інформатики, мистецтва, архітектури, моделювання та візуалізації. Epic відкрила Unreal Engine Marketplace для придбання ігрових асетів. 19 лютого 2015 року Epic запустила Unreal Dev Grants, фонд розвитку в 5 мільйонів доларів, який має на меті надавати гранти для творчих проектів, які використовують Unreal Engine 4. У березні 2015 року Epic випустила Unreal Engine 4 разом з усіма майбутніми оновленнями безкоштовно для всіх користувачів. Натомість Epic встановила вибірковий графік роялті, вимагаючи 5% доходу за продукти, які приносять більше 3000 доларів на квартал. Суїні заявив, що, коли вони перейшли на модель підписки в 2014 році, використання Unreal зросло в 10 разів і завдяки багатьом меншим розробникам, і вважав, що завдяки цій новій схемі ціноутворення вони будуть використовувати ще більше. Намагаючись залучити розробників Unreal Engine, Oculus VR оголосила в жовтні 2016 року, що сплачуватиме гонорар за всі ігри Oculus Rift на базі Unreal, опубліковані в їхньому магазині, до перших 5 мільйонів доларів валового доходу за гру. Щоб підготуватися до випуску свого безкоштовного режиму Battle Royale у Fortnite у вересні 2017 року, Epic довелося внести ряд модифікацій Unreal Engine, які допомогли йому обробляти велику кількість (до 100) підключень до одного сервера. зберігаючи високу пропускну здатність і покращуючи відтворення великого відкритого ігрового світу. Epic заявила, що включатиме ці зміни в майбутні оновлення Unreal Engine. Після відкриття Epic Games Store у грудні 2018 року, Epic не стягуватиме 5% комісії з ігор, які використовують движок Unreal Engine і випущені через Epic Games Stores, поглинаючи цю вартість як частину базової 12% знижки, яку отримує Epic. на покриття інших витрат. Починаючи з 13 травня 2020 р. і ретроактивно до 1 січня 2020 р., сума звільнення від роялті збільшується до 1 000 000 доларів США валового доходу протягом усього життя проекту.

Рис. 2.2.4 – Скріншот з демо, яким

презентували Unreal Engine 4

Unreal Engine 5 був представлений 13 травня 2020 року, він підтримує всі існуючі системи, включаючи консолі наступного покоління PlayStation 5 і Xbox Series X/S. Роботи над рушієм почалися приблизно за два роки до його анонсу. Він був випущений у ранньому доступі 26 травня 2021 року та офіційно запущений для розробників 5 квітня 2022 року. Однією з його основних особливостей є Nanite, рушій, який дозволяє імпортувати в ігри високодеталізований вихідний фотографічний матеріал. Технологія віртуалізованої геометрії Nanite дозволяє Epic скористатися перевагами свого минулого придбання Quixel, найбільшої у світі бібліотеки фотограмметрії з 2019 року. Метою Unreal Engine 5 було максимально полегшити розробникам створення детальних ігрових світів без необхідності витрачати зайвий час на створення нових деталізованих асетів. Nanite може імпортувати майже будь-яке інше тривимірне представлення об’єктів і середовищ, включаючи моделі ZBrush і CAD, що дозволяє використовувати асети з якістю кіновиробництва. Nanite автоматично обробляє рівні деталізації (LOD) цих імпортованих об’єктів, що відповідають цільовій платформі та віддаленню малювання — завдання, яке в іншому випадку художник мав би виконати. Lumen — ще один компонент, описаний як «повністю динамічне глобальне рішення для освітлення, яке негайно реагує на зміни сцени та освітлення». Lumen усуває потребу художникам і розробникам створювати карту світла для певної сцени, але натомість обчислює відображення світла і тіні на льоту, таким чином дозволяючи в режимі реального часу поведінку джерел світла. Віртуальні карти тіней — ще один компонент, доданий в Unreal Engine 5, який описується

як «новий метод відображення тіней, що використовується для забезпечення послідовного затінювання з високою роздільною здатністю, яке працює з об’єктами якості плівки та великими, динамічно освітленими відкритими світами». Віртуальні карти тіней відрізняються від звичайної реалізації карт тіней надзвичайно високою роздільною здатністю, більш детальними тінями та відсутністю тіней, що з’являються і виходять, що можна знайти в більш поширеній техніці карт тіней через каскади тіней. Додаткові компоненти включають Niagara для динаміки рідини і частинок і Chaos для фізичного рушія. З потенційно десятками мільярдів полігонів, присутніми на одному екрані з роздільною здатністю 4K, Epic також розробила Unreal Engine 5, щоб скористатися перевагами майбутніх рішень високошвидкісного зберігання даних з консольним обладнанням наступного покоління, яке використовуватиме поєднання оперативної пам’яті та твердотільні накопичувачі. Epic тісно співпрацювала з Sony в оптимізації Unreal Engine 5 для PlayStation 5 та над архітектурою зберігання консолі. Щоб продемонструвати легкість створення деталізованого світу з мінімальними зусиллями, у травні 2020 року на презентації рушія було продемонстровано демонстрацію під назвою «Lumen in the Land of Nanite», що працює на PlayStation 5, яка була створена переважно шляхом вилучення ресурсів із бібліотеки Quixel та використання Nanite, Lumen та інші компоненти Unreal Engine 5 для створення фотореалістичної печери, яку можна було б досліджувати. Epic підтвердила, що Unreal Engine 5 також буде повністю підтримуватися на Xbox Series X, але під час анонсу була зосереджена на PlayStation 5 в результаті їхньої роботи з Sony в попередні роки. Epic планує використовувати Fortnite як тестовий стенд для Unreal Engine 5, щоб продемонструвати, що цей рушій може зробити для індустрії, і гра була введена для використання Unreal Engine 5 у грудні 2021 року. Senua's Saga: Hellblade II від Ninja Theory також стане однією з перших ігор з використанням Unreal Engine 5. The Matrix Awakens, прив’язаний до випуску фільму Матриця: Відродження, був розроблений Epic для подальшої демонстрації Unreal Engine 5 разом з іншими технологіями, які вони придбали протягом 2020 та 2021 років, включаючи їх MetaHuman Creator, розроблений та інтегрований в Unreal Engine 5 з технологіями 3Lateral, Cubic Motion і Quixel. Додаткові функції, заплановані для Unreal Engine 5, надходять завдяки придбанням і партнерствам Epic. MetaHuman Creator — це проект, заснований на технології трьох компаній, придбаних Epic — 3Lateral, Cubic Motion і Quixel — щоб дозволити розробникам швидко створювати реалістичні людські персонажі, які потім можна експортувати для використання в Unreal. Завдяки партнерству з Cesium Epic планує запропонувати безкоштовний плагін для надання тривимірних геопросторових даних для користувачів Unreal, що дозволить їм відтворити будь-яку частину нанесеної на карту поверхні Землі. Epic

включатиме RealityCapture, продукт, який вона придбала завдяки придбанню Capturing Reality, яка може створювати 3D-моделі будь-якого об’єкта з колекції фотографій, зроблених з різних ракурсів, а також різні інструменти проміжного програмного забезпечення, запропоновані Epic Game Tools. Unreal Engine 5 збереже поточну модель роялті, при цьому розробники повертатимуть 5% валового доходу Epic Games, хоча ця плата знята для тих, хто випускає свої ігри в Epic Games Store. Крім того, Epic оголосила разом із Unreal Engine 5, що вони не отримають жодної плати з ігор, які використовують будь-яку версію Unreal Engine за перший валовий дохід у розмірі 1 мільйона доларів США, який має зворотну силу до 1 січня 2020 року.

Рис. 2.2.5 – Скріншот з демо The Matrix Awakens,

розроблена на Unreal Engine 5

UnrealScript (часто скорочено до UScript) була нативною мовою сценаріїв Unreal Engine, яка використовувалася для створення ігрового коду та ігрових подій до випуску Unreal Engine 4. Ця мова була розроблена для простого програмування ігор високого рівня. Інтерпретатор UnrealScript був запрограмований Суїні, який також створив попередню мову сценаріїв для гри, ZZT-оор. Подібно до Java, UnrealScript був об’єктно-орієнтованим без множинного успадкування (усі класи успадковуються від загального класу Object), а класи були визначені в окремих файлах, названих відповідно до класу, який вони визначають. На відміну від Java, UnrealScript не мав обгорток для примітивних типів. Інтерфейси підтримувалися лише в Unreal Engine 3 та

кількох іграх на Unreal Engine 2. UnrealScript підтримує перевантаження операторів, але не перевантаження методів, за винятком необов'язкових параметрів. На конференції розробників ігор у 2012 році Epic оголосила, що UnrealScript буде видалено з Unreal Engine 4 на користь C++. Візуальні сценарії підтримуватимуться системою Visual Scripting Blueprints, заміною попередньої системи візуальних сценаріїв Kismet. Verse — це нова мова сценаріїв для Unreal Engine, яка, як очікується, буде випущена у 2022 році та реалізована у Fortnite. Саймон Пейтон Джонс, відомий своїм внеском у мову програмування Haskell, був найнятий Epic Games у грудні 2021 року як інженерний співробітник для роботи над Verse.

Unreal Engine знайшов застосування у створенні фільмів для створення віртуальних декорацій, які можуть відстежувати рух камери навколо акторів і об'єктів і відображатися в режимі реального часу на великих світлодіодних екранах і системах атмосферного освітлення. Це дозволяє створювати кадри в реальному часі, негайне редагування віртуальних декорацій за потреби, а також можливість знімати кілька сцен за короткий період, просто змінюючи віртуальний світ за акторами. Було визнано, що загальний зовнішній вигляд виглядає більш природним, ніж типові ефекти хромакея. Серед постановок з використанням цих технологій були телесеріали «Мандалорець» і «Світ заходу». Джон Фавро та підрозділ Industrial Light & Magic Lucasfilm працювали з Epic над розробкою технології StageCraft для «Мандалорця» на основі подібного підходу, який Фавро використовував у «Королі Леві». Потім Фавро поділився цим технологічним підходом з Джонатаном Ноланом і Лізою Джой, продюсерами Westworld. У шоу вже розглядалося використання віртуальних наборів раніше і було створено деякі технології, але в третьому сезоні інтегровано використання Unreal Engine, як і в StageCraft. Orca Studios, іспанська компанія, співпрацює з Epic над створенням кількох студій для віртуальних зйомок, подібних до підходу StageCraft з Unreal Engine, що надає віртуальні декорації, особливо під час пандемії COVID-19, яка обмежувала подорожі. У січні 2021 року Deadline Hollywood оголосив, що Epic використовує частину своїх Epic MegaGrants, щоб вперше підтримати повнометражний анімаційний фільм «Гільгамеш», який буде повністю випущено в Unreal Engine анімаційними студіями Hook Up, DuermeVela та FilmSharks. У рамках розширення MegaGrants Epic також профінансувала 45 додаткових проектів приблизно з 2020 року для створення фільмів і короткометражних фільмів на Unreal Engine.

2.3. Система візуального скриптингу Blueprint

Система Visual Scripting Blueprint в Unreal Engine — це повна система сценаріїв ігрового процесу, заснована на концепції використання інтерфейсу на основі вузлів для створення елементів ігрового процесу з Unreal Editor. Як і в багатьох поширених мовах сценаріїв, він використовується для визначення об'єктно-орієнтованих (ОО) класів або об'єктів у механізмі. Коли ви використовуєте UE4, ви часто помітите, що об'єкти, визначені за допомогою Blueprint, у розмовній мові називаються просто «Креслення». Ця система надзвичайно гнучка та потужна, оскільки надає можливість дизайнерам використовувати практично весь спектр концепцій та інструментів, які зазвичай доступні лише програмістам. Крім того, спеціальна розмітка Blueprint, доступна в реалізації Unreal Engine на C++, дозволяє програмістам створювати базові системи, які можуть бути розширені дизайнерами. Чи означає це, що Blueprint є заміною UnrealScript? Так і ні. Ігрове програмування та все, для чого використовувався UnrealScript в минулому, все ще можна обробляти за допомогою коду на C++. У той же час, хоча Blueprints не призначені як заміна UnrealScript, вони служать багатьом з тих же цілей, які виконував UnrealScript, наприклад:

- Розширення класів
- Зберігання та зміна властивостей за замовчуванням
- Керування екземпляром підоб'єкта (наприклад, компонентів) для класів

Очікується, що програмісти ігрового процесу створять базові класи, які надають корисний набір функцій і властивостей, які можуть використовувати та розширювати блупрінти, створені з цих базових класів. Блупрінти можуть бути одним із кількох типів, кожен із яких має власне специфічне використання від створення нових типів до подій рівня сценаріїв до визначення інтерфейсів або макросів, які будуть використовуватися іншими блупрінтами.

Клас Blueprint, який часто скорочується як Blueprint, є асетом, який дозволяє розробникам вмісту легко додавати функціональні можливості поверх існуючих ігрових класів. Блупрінти створюються всередині Unreal Editor візуально, а не шляхом введення коду, і зберігаються як асети в пакеті контенту. Вони, по суті, визначають новий клас або тип актора, який потім можна помістити на рівень як екземпляри, які ведуть себе як будь-який інший тип актора.

Blueprint, що містить лише дані, — це клас Blueprint, який містить лише код (у вигляді графіків вузлів), змінні та компоненти, успадковані від його батьківського елемента. Вони дозволяють налаштовувати та змінювати ці успадковані властивості, але не можна додавати нові елементи. Вони, по

суті, є заміною архетипів і можуть використовуватися, щоб дозволити дизайнерам налаштовувати властивості або встановлювати елементи з варіаціями. Blueprint, що містить лише дані, редагується в компактному редакторі властивостей, але також може бути «перетворений» на повні креслення шляхом простого додавання коду, змінних або компонентів за допомогою повного редактора Blueprint.

Blueprint рівня — це спеціалізований тип Blueprint, який діє як глобальний графік подій на рівні всього рівня. Кожен рівень у вашому проекті має свій власний блупрінт рівня, створений за замовчуванням, який можна редагувати в редакторі Unreal, однак нові блупрінти рівня не можна створити через інтерфейс редактора. Події, що стосуються рівня в цілому, або окремих екземплярів Акторів у межах рівня, використовуються для запуску послідовностей дій у формі викликів функцій або операцій керування потоком. Ті, хто знайомий з Unreal Engine 3, повинні бути добре знайомі з цією концепцією, оскільки вона дуже схожа на те, як Kismet працював у Unreal Engine 3. Блупрінти рівня також забезпечують механізм контролю для потокової передачі рівня та секвенсора, а також для прив’язування подій до акторів, розміщених на рівні.

Інтерфейс Blueprint – це набір однієї або кількох функцій – лише ім’я, без реалізації – які можна додати до інших Blueprint. Будь-який блупрінт, до якого додано інтерфейс, гарантовано матиме ці функції. Функціям інтерфейсу можна надати функціональні можливості в кожному з блупрінтів, які його додали. Це, по суті, схоже на концепцію інтерфейсу в загальному програмуванні, яка дозволяє використовувати декілька різних типів об’єктів для спільного використання та доступу до них через загальний інтерфейс. Простіше кажучи, інтерфейси Blueprint дозволяють ділитися різними кресленнями та передавати дані один одному.

Бібліотека макросів Blueprint — це контейнер, який містить набір макросів або автономних графіків, які можна розмістити як вузли в інших кресленнях. Вони можуть заощадити час, оскільки вони можуть зберігати зазвичай використовувані послідовності вузлів разом із входами та виходами як для виконання, так і для передачі даних. Макроси використовуються спільно для всіх графіків, які посилаються на них, але вони автоматично розгортаються в графіки, як ніби вони були згорнутим вузлом під час компіляції. Це означає, що бібліотеки макросів Blueprint не потрібно компілювати. Однак зміни до макросу відображаються лише в графіках, які посилаються на цей макрос, коли план, що містить ці графіки, перекомпілюється.

Утиліта Blueprint (або скорочено Blutility) — це Blueprint, призначений лише для редактора, який можна використовувати для виконання дій редактора або розширення функціональних можливостей редактора. Вони можуть показувати події без параметрів у вигляді кнопок в інтерфейсі користувача та мають можливість виконувати будь-які функції, доступні для Blueprints, і діяти на поточний набір вибраних акторів у вікні перегляду.

Функціональність Blueprints визначається за допомогою різних елементів; деякі з них доступні за замовчуванням, а інші можна додавати за потреби. Вони надають можливість визначати компоненти, виконувати операції ініціалізації та налаштування, реагувати на події, організовувати та модульувати операції, визначати властивості тощо.

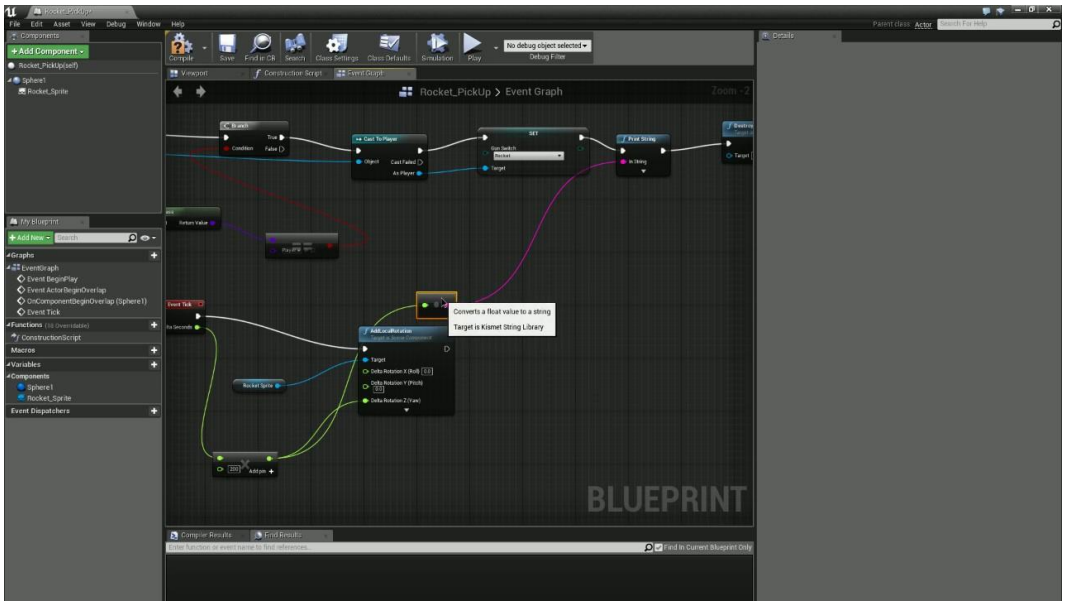


Рис. 2.3.1 – Скріншот вікна редагування Blueprint

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Планування розробки проекту

Після визначення вимог дипломного проекту та вибору програмних засобів, був сформований список задач, які неохідно реалізувати:

- Знайти асети для персонажа та рівня
- Створити проект в Unreal Editor з використанням фреймворку Paper2D
- Імпортувати знайдені асети у проект
- Розбити асети для рівня на тайлмапи
- Створити анімації з імпортованих асетів персонажа і оточення
- Створити клас персонажа
- Створити рівень
- Імплементувати подвійний стрибок для персонажа
- Додати рухомі платформи на рівень
- Додати відштовхуючі платформи
- Додати ворогів
- Додати фаєрболи
- Створити головне меню та меню паузи
- Додати HUD на екран гри
- Створити клас для предметів колекціонування
- Додати монети на рівень
- Додати пастки на рівень

Для відстежування виконання задач, вони були додані на дошку в системі Trello, яка часто використовується невеликими командами розробників для подібних задач.

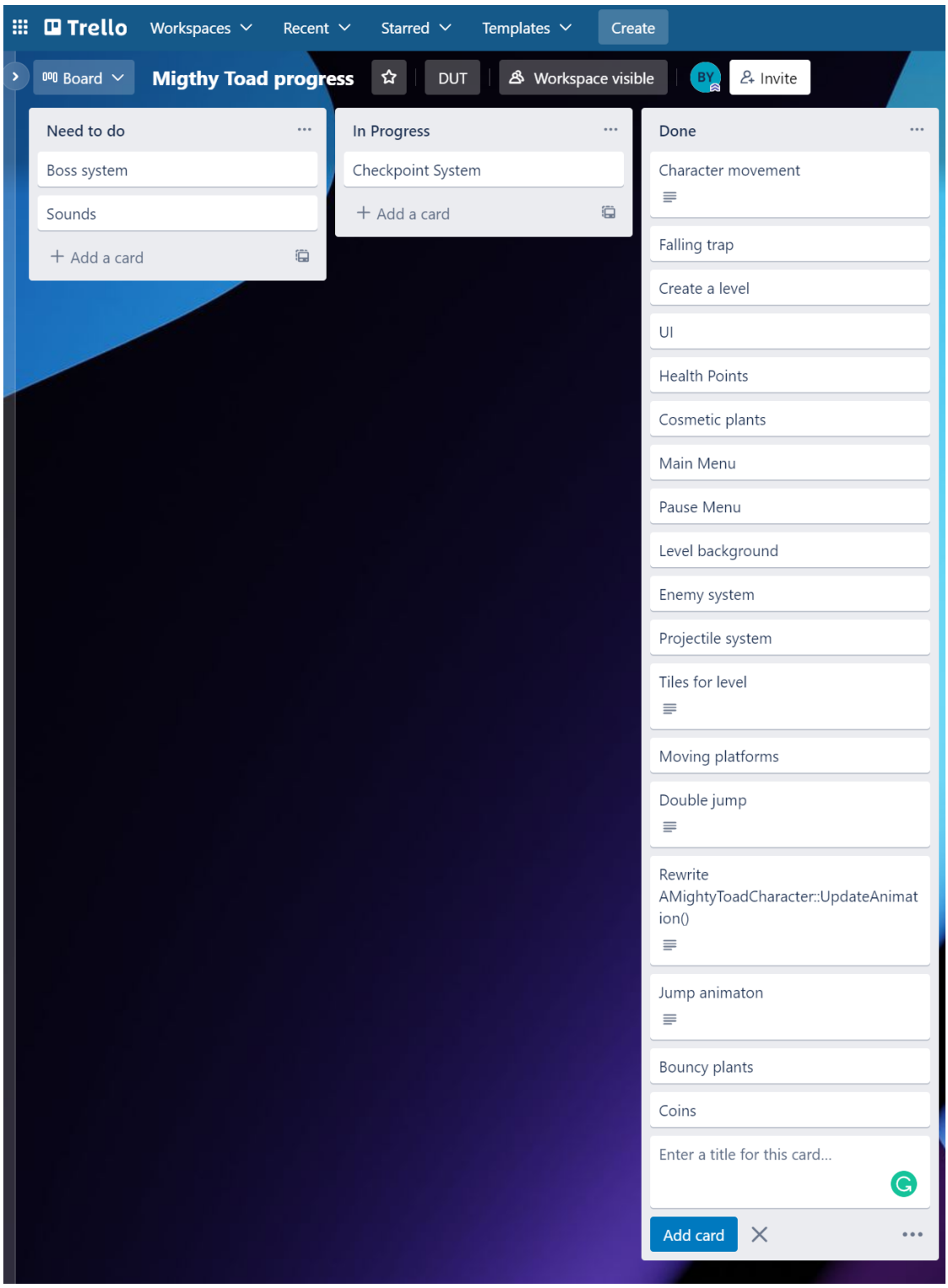


Рис. 3.1.1. – Скріншот дошки Trello дипломного проекту

3.2. Початок роботи з Unreal Engine

Після того як необхідні асети були знайдені, їх треба було імпортувати у новий проект проект. При створенні проекту в Unreal Engine, програма дає вибрати, створити абсолютно пустий проект чи проект на основі прототипу.

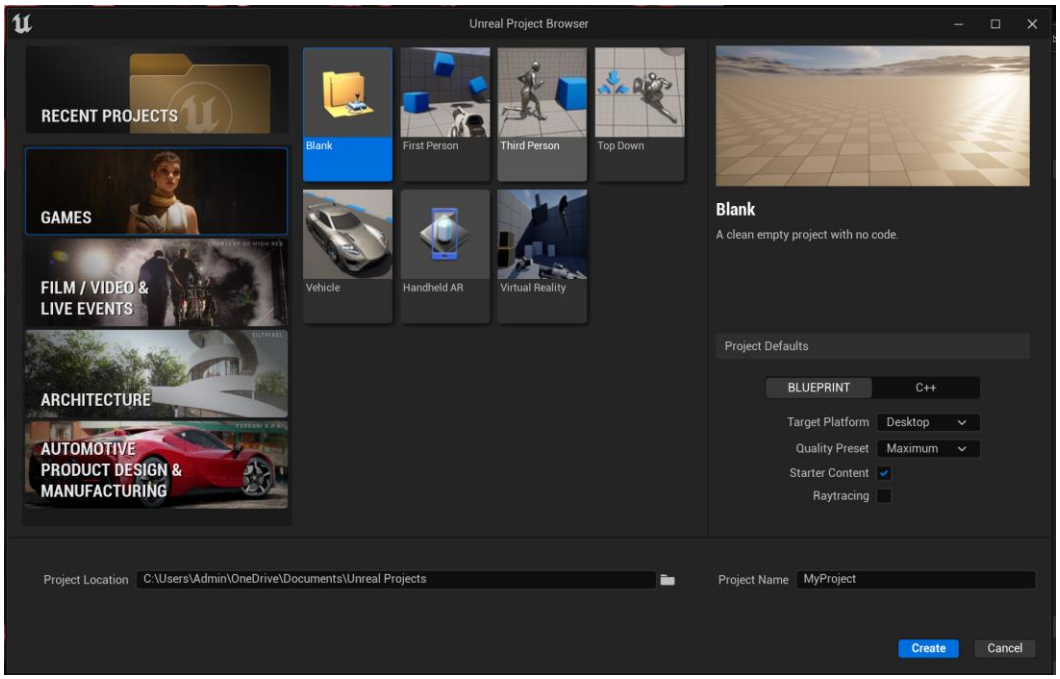


Рис. 3.2.1 – Вікно створення нового проекту

Враховуючи, що дипломний проект являє собою 2D-платформер, було обрано прототип на основі фреймворку Paper2D, що додало до проеку базову модель персонажа, тестовий рівень та базовий клас для 2D-гри.

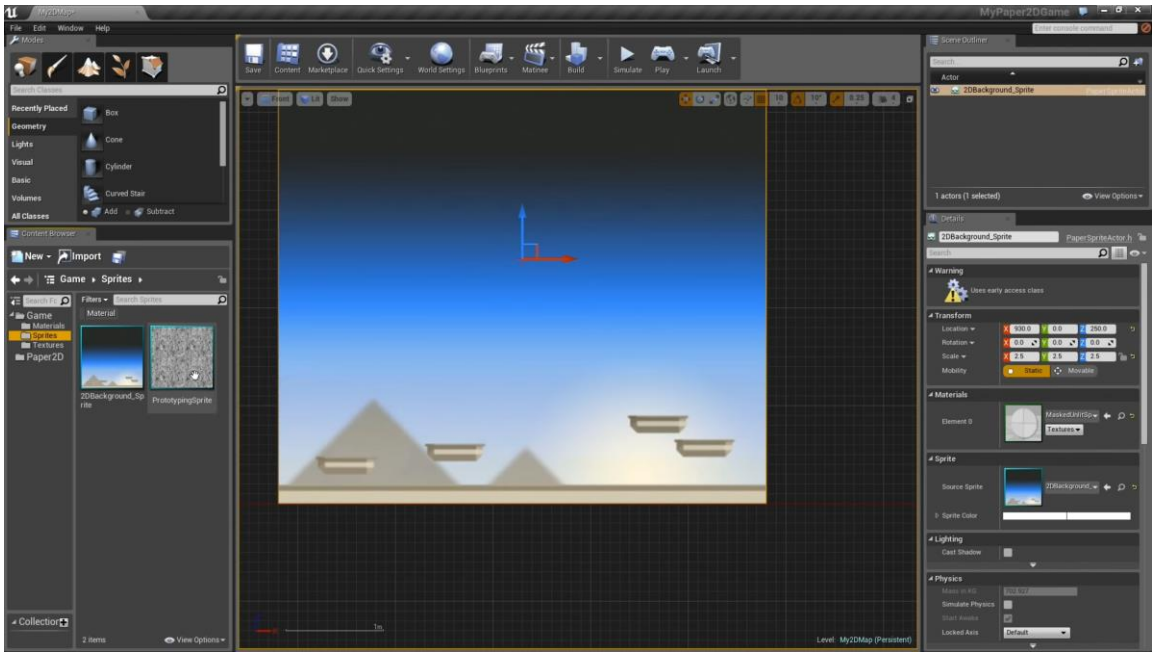


Рис. 3.2.2 – Основне вікно Unreal Editor, з базовим рівнем

Далі збережені асети були імпортовані у проект без проблем, за допомогою технології Drag’n’Drop.

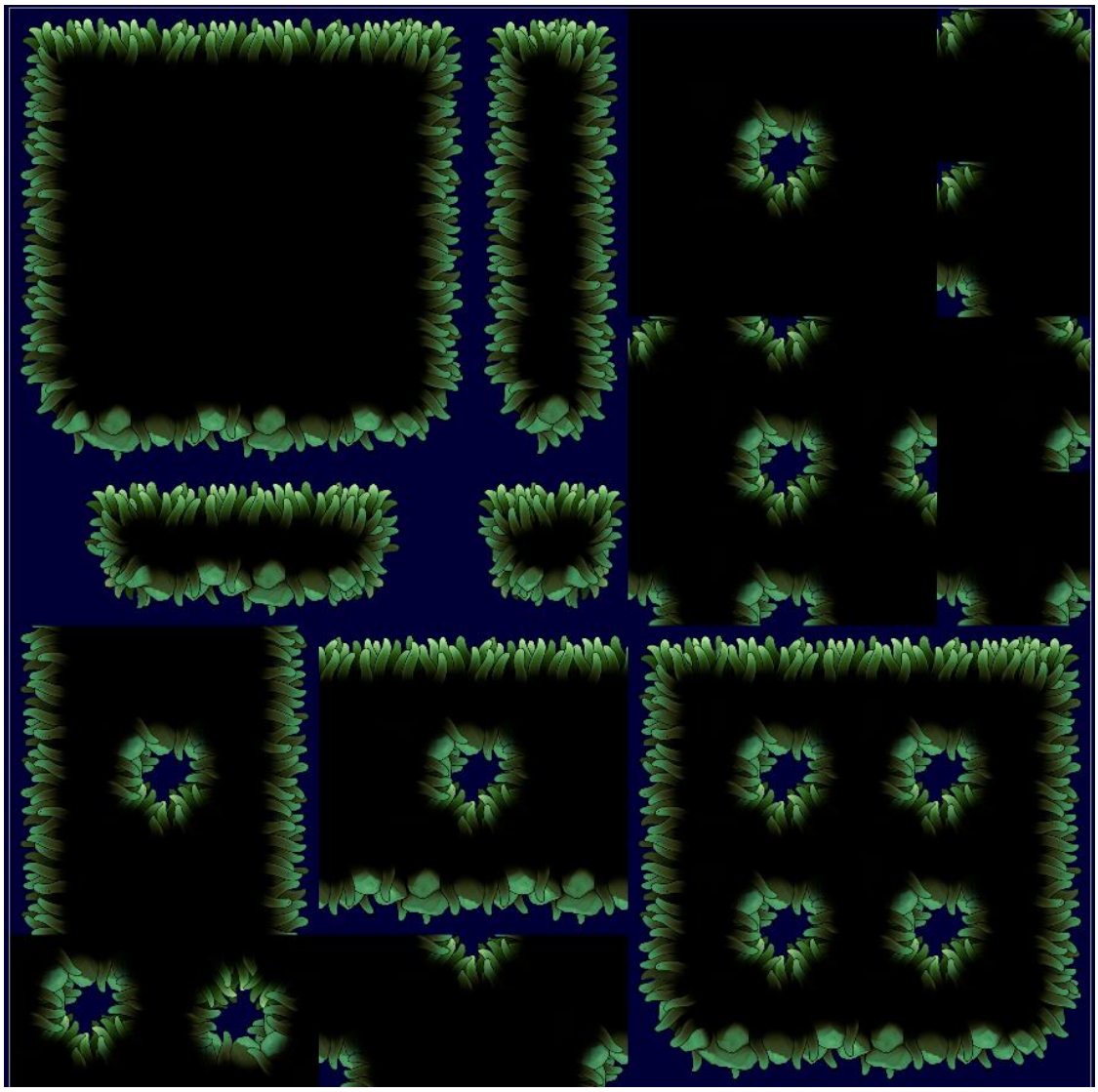


Рис. 3.2.3 – Імпортований асет з блоками, потрібними для побудови рівня

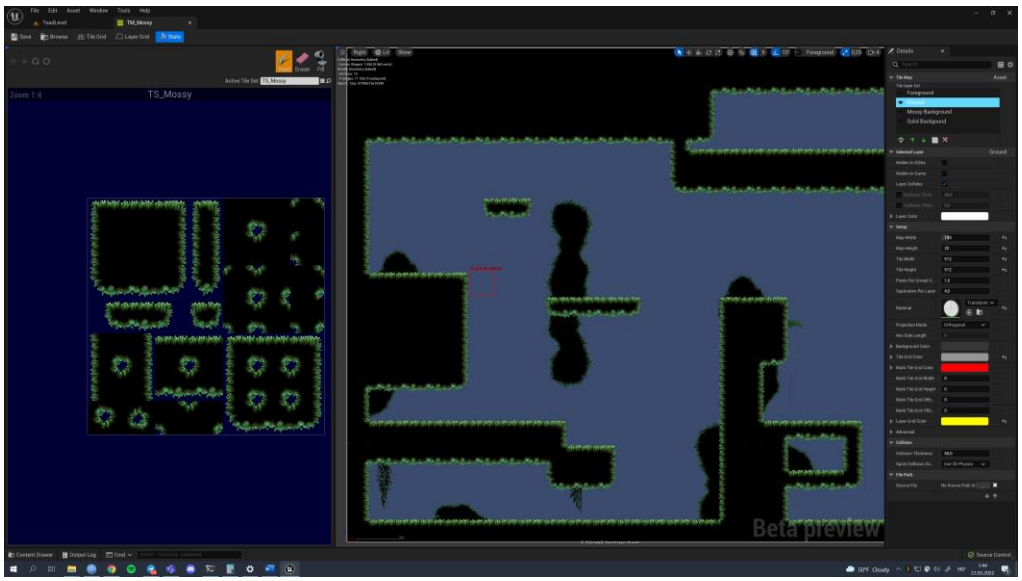


Рис. 3.2.4 – Вікно побудови рівня за допомогою тайлсетів

3.3. Робота з класами в Unreal Engine

Всі класи в Unreal Engine подібно до мови програмування Java наслідуються від базового класу (UObject, у випадку UE). Далі класи поділяються на Акторів та Компоненти. Актори – класи, які можна додати на рівень, а компоненти – частини функціоналу, які додаються до акторів. Клас головного персонажу буде наслідуватися від APaperCharacter – класу, який має компоненти для керування актором, камеру та модель для відображення на рівні.

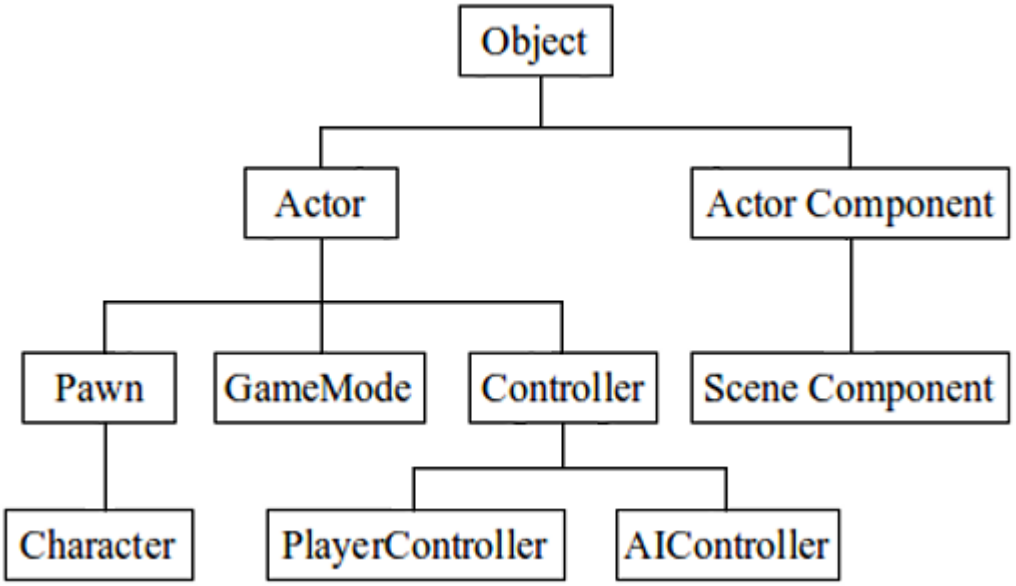


Рис. 3.3.1 – Базові класи Unreal Engine

В APaperCharacter вже реалізована система анімацій, яка включає режим спокою, бігу та поворот. Персонаж платформеру повинен вміти стрибати, тому у класі AmightyToadCharacter був перевантажений метод UpdateAnimation, щоб додати підтримку анімації стрибка.

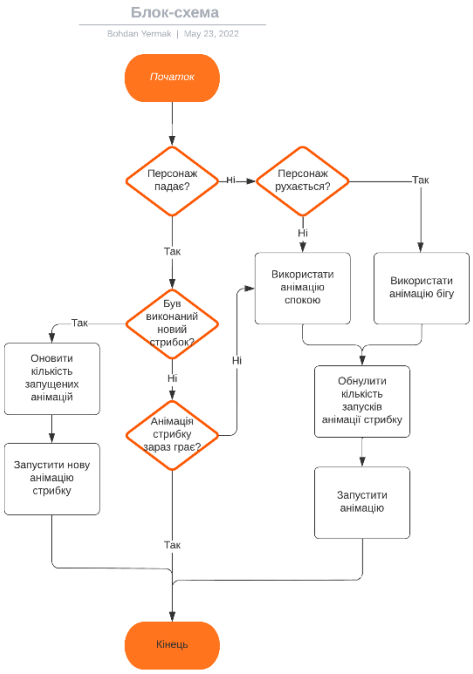


Рис. 3.3.2 – Блок-схема нового алгоритму вибору анімації для персонажа

3.4. Робота з Blueprint

Для створення рухомих та відштовхуючих платформ було вирішено використати систему візуального скриптингу Blueprint. Це дозволить швидко створити потрібний ігровий об’єкт без використання C++.

Спочатку створюється Blueprint-клас, що наслідується від Actor, так як об’єкти цього класу будуть розташовані «на сцені» (всередені рівня). До створеного об’єкту потрібно додати необхідні компоненти. У випадку рухомої платформи, це компонент зіткнення, що дозволить персонажу стояти на платформі та компонент картинки, яка буде відображати платформу на рівні.

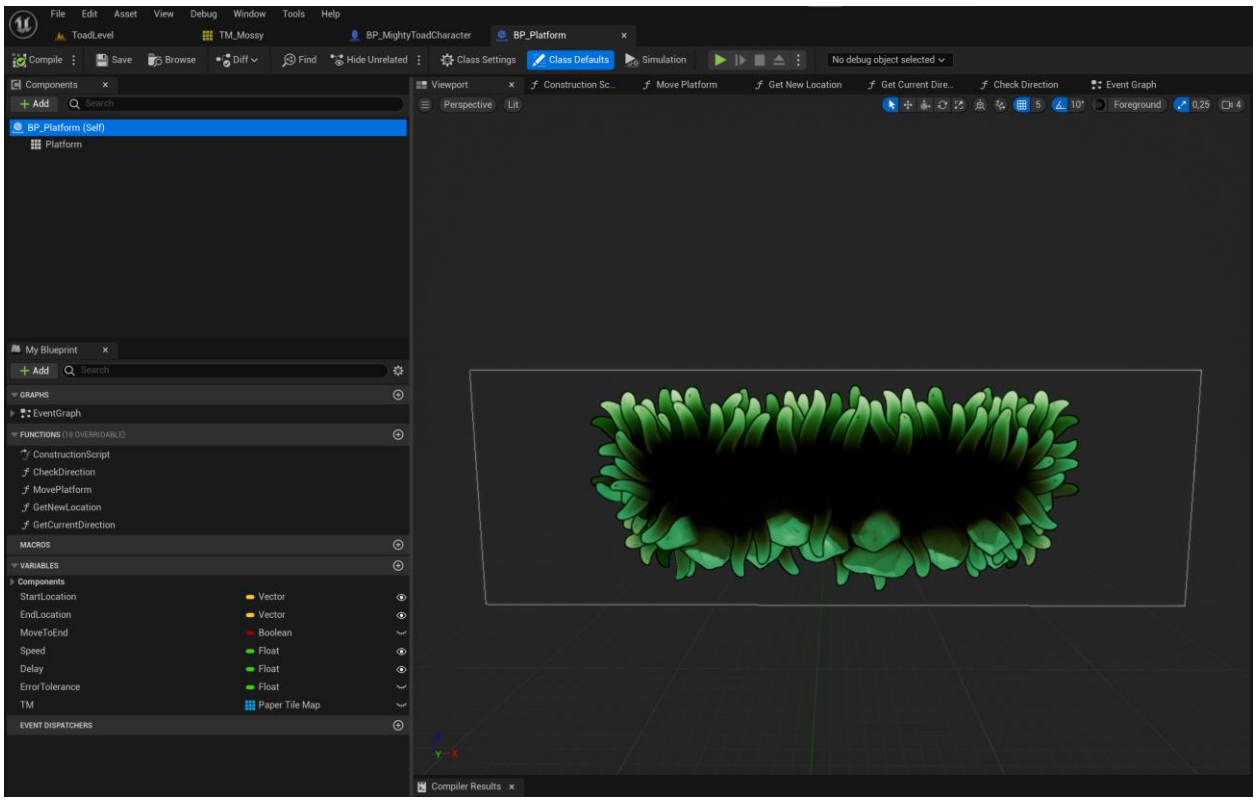


Рис. 3.4.1 – Вікно додавання компонентів до Blueprint-класу

Після того, як неохідні компоненти додані, необхідно додати функціонал, щоб платформа рухалась. Платформа буде рухатись між двома точками на рівні, які задаються при розміщенні об’єкту всередину. Для визначення пройденої відстані за певний проміжок часу було використано алгоритм лінійної інтерполяції.

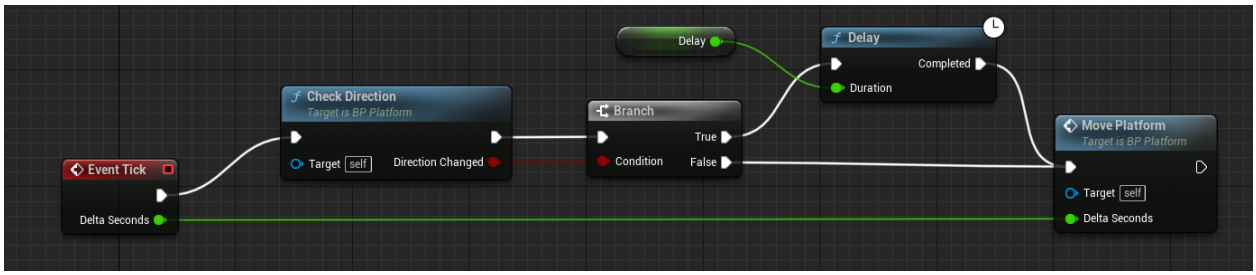


Рис. 3.4.2. – Основна функція для руху платформи

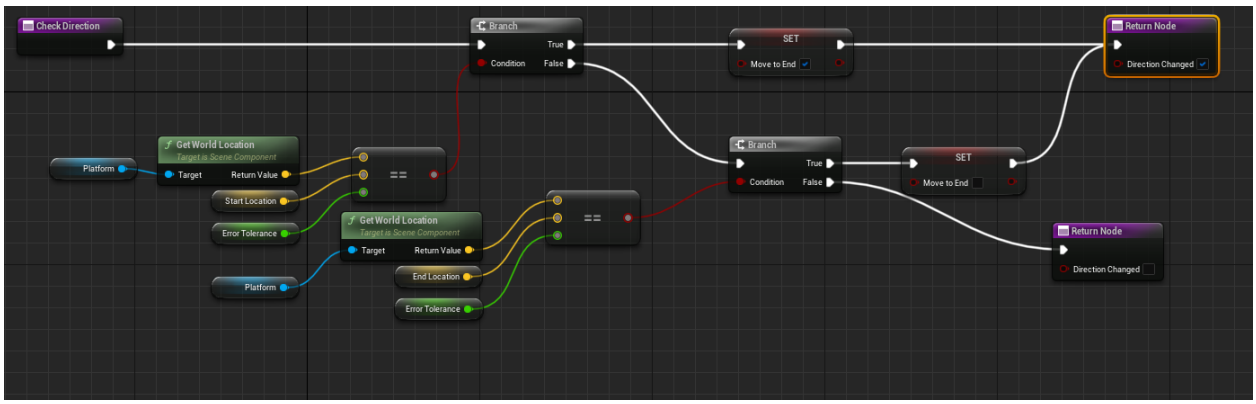


Рис. 3.4.3 – Функція CheckDirection

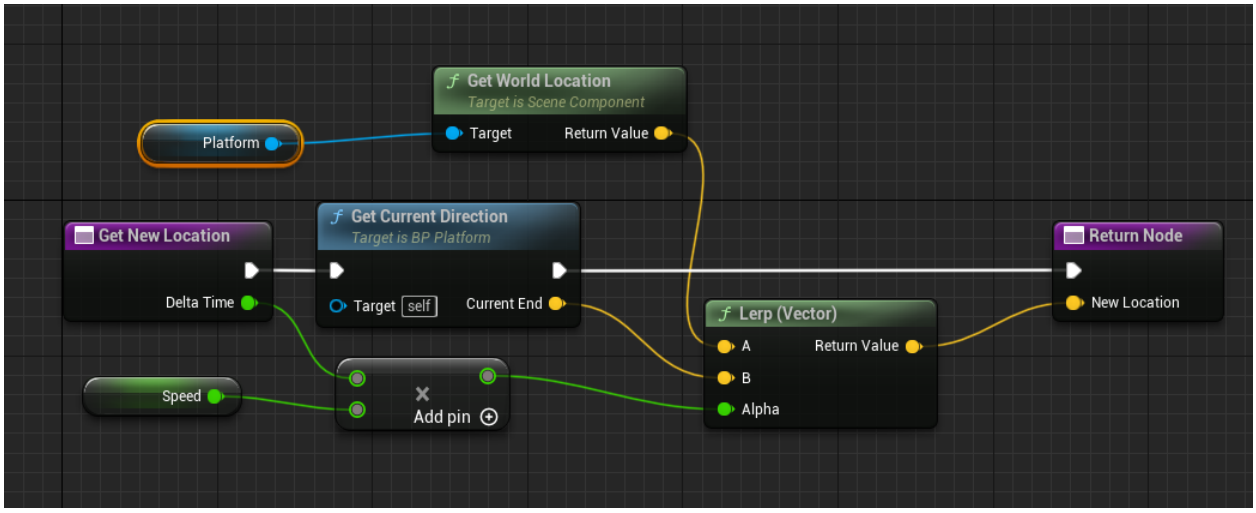


Рис. 3.4.4 – Функція GetNewLocation

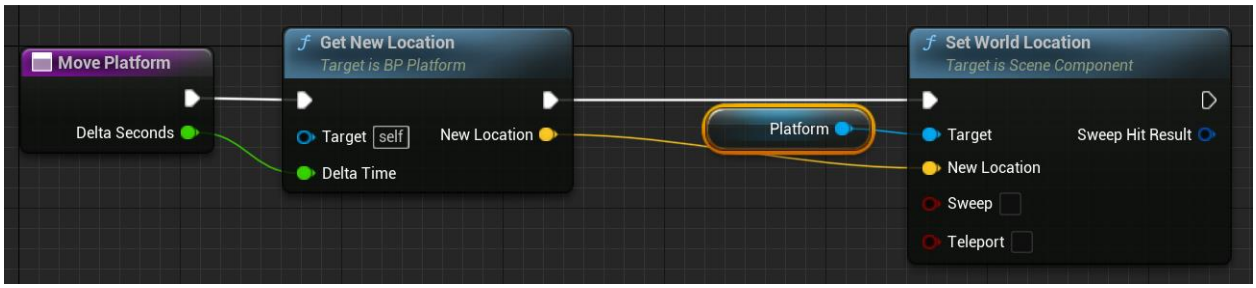


Рис. 3.4.5 – Функція MovePlatform

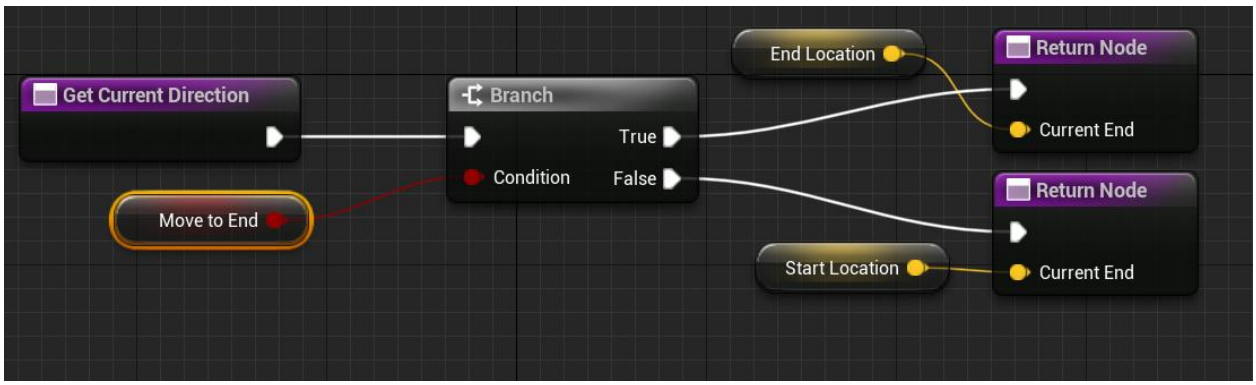


Рис. 3.4.6 – Функція GetCurrentDirection

3.5. Розробка інтерфейсу користувача

Unreal Engine має вбудовані інструменти для створення графічних інтерфейсів за допомогою різних об’єктів, таких як текст, кнопки, картинки, шкала виконання та багато інших.

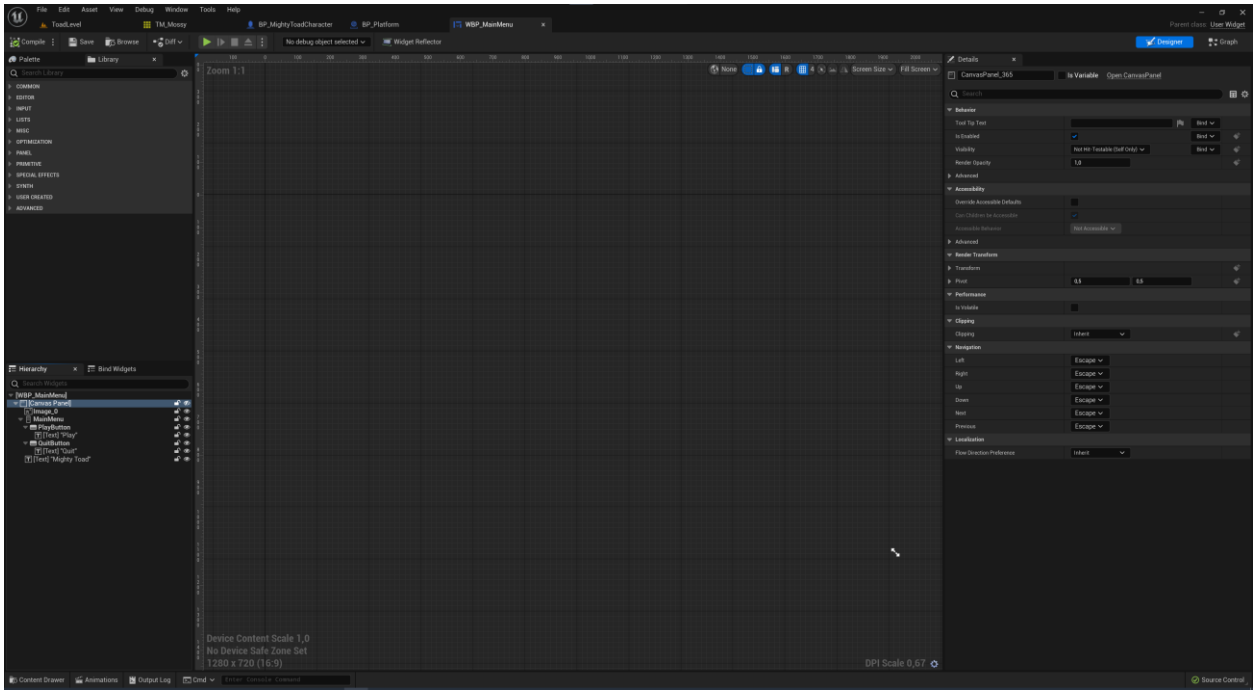


Рис. 3.5.1 – Вікно розробки дизайну інтерфесу

Рис. 3.5.2 – Готовий дизайн головного меню гри MightyToad

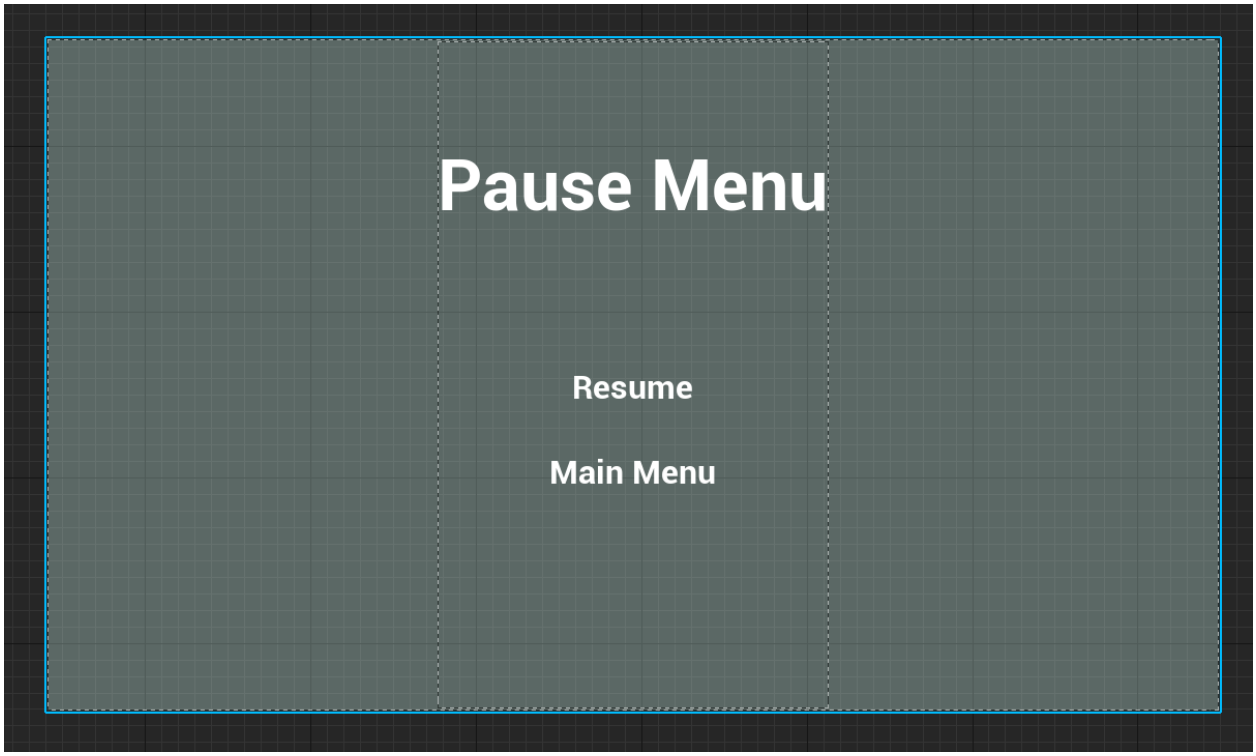


Рис. 3.5.3 – Готовий дизайн меню паузи

Дизайнер інтерфейсів дозволяє додати логіку до створеного дизайну за допомогою системи сумісної з Blueprint.

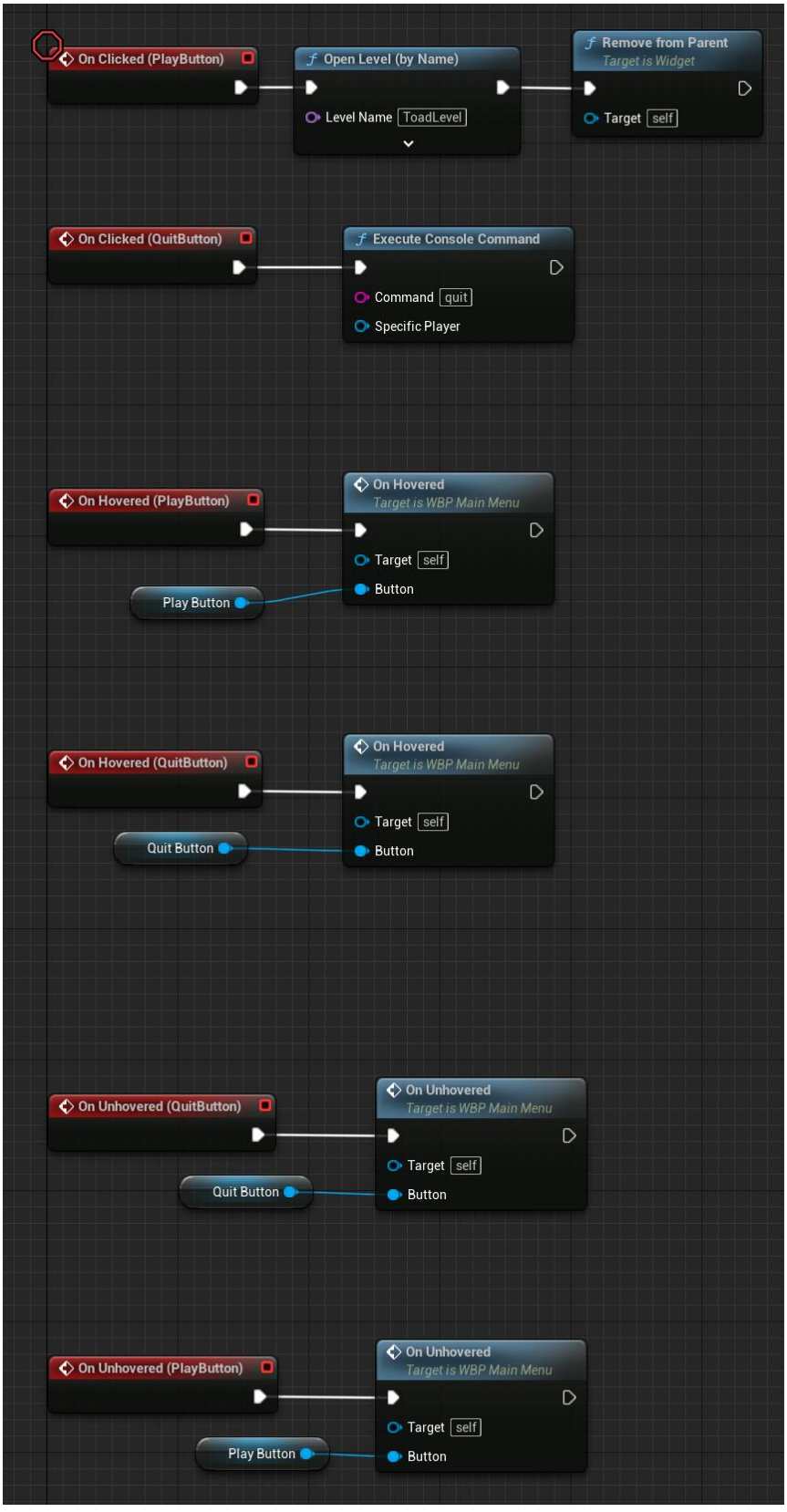


Рис. 3.5.4 – Логіка роботи головного меню

Враховуючи, що персонаж має здоров’я і на рівні є монети, які можна збирати, буде доцільно створити візуальний інтерфейс для відображення данної інформації. Кількість життів буде відображено за допомогою зображень серця, а монети – зображення монети та текст, що буде показувати кількість зібраних монет.

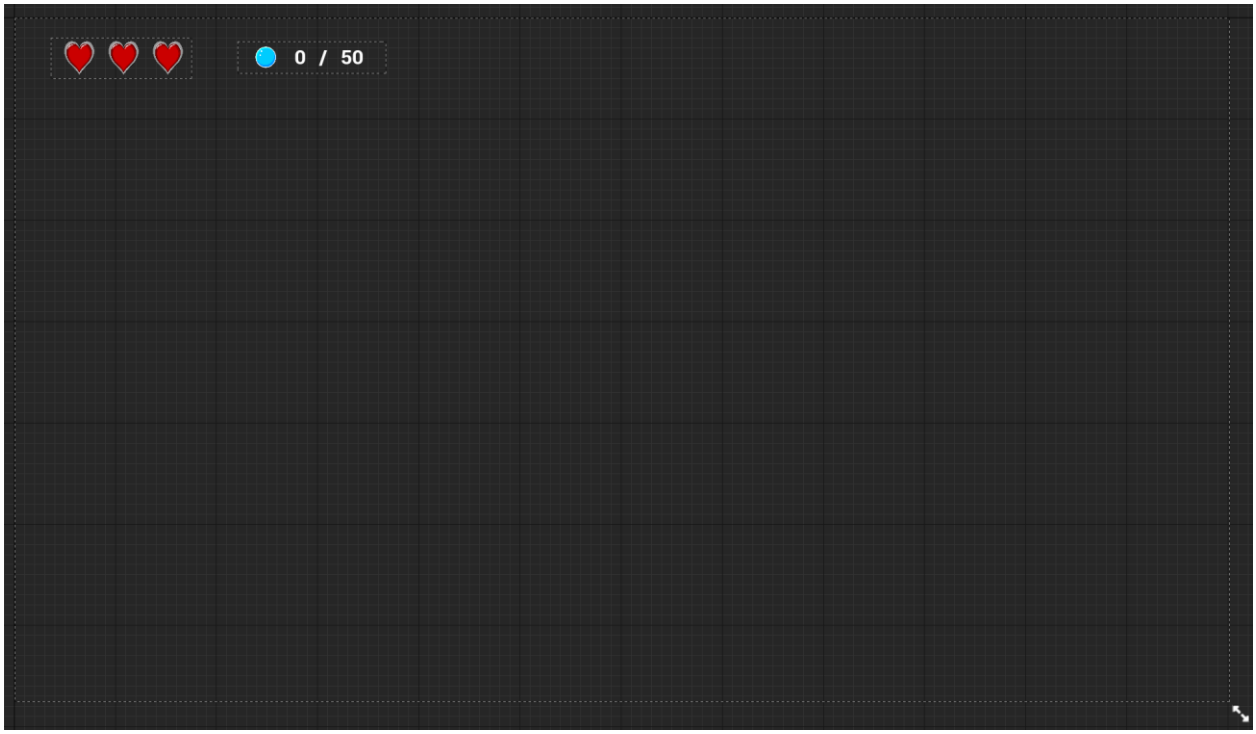


Рис. 3.5.5 – Дизайн HUD

3.6. Тестування програмного продукту

Тестування проекту відбувалось по ходу його розробки. Після додавання нових об’єтів проводилась перевірка їхньої роботи у різних ситуаціях та вплив на загальну атмосферу гри.

Реалізована гра була перевірена за такими критеріями:

- всі функції були перевірені на роботоспроможність
- рівні перевірено на наявність невидимих стін
- кожний об’єкт чітко видно і можна відрізнити рухомі частини від декорацій
- керування персонажем працює без збоїв і є стандартним для ігор данного жанру

У процесі перевірки якості були виявлені та виправлені недоліки:

- здоров’я, що випадає з ворогів не можна підняти, якщо у персонажа вже максимальна кількість;
- система запускання магічних куль змушувало гру вилітати
- персонаж міг випасти за рівень

Також у процесі розробки довелось відмовитись від системи секретних кімнат на рівні, бо об’єкт який використовався для даного функціоналу ще знаходиться у розробці і працював некоректно.

ВИСНОВКИ

У дипломній роботі розроблену гру у жанрі 2D платформер під назвою Mighty Toad. У процесі створення проекту проаналізовано принципи розробки ігор, взаємодії з рушієм Unreal Engine та мовою програмування C++.

Під час аналізу було знайдено потрібну інформацію на тематичних інтернет-ресурсах та були розібрані можливості популярних ігрових рушіїв. Досліджено ігрові аналоги, які були популярні у свій час та проаналізовано їхні сильні і слабкі сторони.

Було розроблено концепт майбутньої відеогри та визначені основні характеристики проекту.

У процесі проектування відеогри були розроблені об’єкти які можуть бути використаними повторно у наступних рівнях чи навіть інших проектах. Окрім цього було доповнено один з базових класів для персонажа, що може допомогти при розробці наступних ігор.

Для досягнення поставленої мети було проаналізовано актуальність даної теми, реалізовано необхідні функціональні вимоги до гри, досліджено методи взаємодії з ігровим рушієм, побудовано можель та архітектуру розроблюваного програмного забезпечення.

Робота пройшла апробацію на Науково-технічній конференції «Застосування програмного забезпечення в ІКТ», м. Київ, ДУТ, 20 квітня 2022 року.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Video Game History [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.history.com/topics/inventions/history-of-video-games
2. The History Of Gaming: An Evolving Community [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/
3. Games industry experts weigh in on the global game industry [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://venturebeat.com/2021/05/12/games-industry-experts-weigh-in-on-the-global-game-industry/
4. 'Spacewar!' The story of the world's first digital video game [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.theverge.com/2013/2/4/3949524/the-story-of-the-worlds-first-digital-video-game
5. The First Video Game? [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.bnl.gov/about/history/firstvideo.php
6. Here's who won each console war [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://venturebeat.com/2014/08/20/heres-who-won-each-console-war/
7. Angry Birds Maker Rovio Reports $200 Million In Revenue, $71 Million In Profit For 2012 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.businessinsider.com/angry-birds-made-200-million-in-2012-2013-4
8. Video Game History Timeline [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.museumofplay.org/video_games/
9. The Brown Box, 1967–68 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://americanhistory.si.edu/collections/search/object/nmah_1301997
10. Platform game [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://en.wikipedia.org/wiki/Platform_game
11. How video games are made: the game development process [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.cgspectrum.com/blog/game-development-process
12. History of C++ [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.cplusplus.com/info/history/
13. Bjarne Stroustrup A HISTORY OF C++: 1979-1991, 1996, - 769 с.

14. C++ Origin and philosophy - History of C++ [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.scaler.com/topics/cpp/origin-and-philosophy-of-cpp/
15. McDonald, T. Liam The 3D Engine Wars. Maximum PC. Vol. 3, no. November 1998. p. 43. ISSN 1522-4279. Archived from the original on October 16, 2019. Retrieved September 17, 2017.
16. Sweeney, Tim (2005). "GPU Gems 2 – Foreword". Nvidia Developer. Archived from the original on October 3, 2017. Retrieved October 3, 2017.
17. Epic's Tim Sweeney lays out the case for Unreal Engine 4 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.gamedeveloper.com/programming/epic-s-tim-sweeney-lays-out-the-case-for-unreal-engine-4
18. From The Past To The Future: Tim Sweeney Talks [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://web.archive.org/web/20170809214002/http://www.gamasutra.com/view/feature/4035/from_the_past_to_the_future_tim_.php
19. Classic Tools Retrospective: Tim Sweeney on the first version of the Unreal Editor [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://web.archive.org/web/20180823012812/https://www.gamasutra.com/blogs/DavidLightbown/20180109/309414/Classic_Tools_Retrospective_Tim_Sweeney_on_the_first_version_of_the_Unreal_Editor.php
20. Talkin' nasty with Epic's code-p1mp, Tim Sweeney [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://web.archive.org/web/19990501170629/http://www.voodooextreme.com/articles/interview_ts.html
21. Brandon, Alexander (2004). Audio for Games: Planning, Process, and Production. New Riders. p. 70. ISBN 9780735714137.
22. Unreal: Epic's would-be Doom... er... Quake killer Sweeney [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.theregister.com/2013/07/16/antique_code_show_unreal/
23. Next gen engines [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://web.archive.org/web/20011121083307/http://www.tolstiy.ag.ru/what/sweeney.htm
24. Doom to Dunia: A Visual History of 3D Game Engines [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://web.archive.org/web/20090724065546/http://www.maximumpc.com/article/features/3d_game_engines?page=0,8

25. Learn to Let Go: How Success Killed Duke Nukem [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.wired.com/2009/12/fail-duke-nukem/
26. Tim Sweeney Looks Ahead [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.ign.com/articles/1998/10/24/tim-sweeney-looks-ahead
27. Improving "America's Army" [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.army.mil/article/11935/improving_americas_army\
28. McDonald, Thomas L. "Gentlemen, Start Your Engines!". Maximum PC. No. January 2001. p. 44
29. Ubisoft: 3DS Can Handle Unreal Engine 2 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.tomsguide.com/us/Unreal-Engine-2-Splinter-Cell-3DS-Nitendo-3DS,news-10601.html
30. Edge Australia. "The unreal thing". Edge Australia. No. 1. p. 68. Retrieved August 26, 2017.
31. Unreal Engine 3 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.eurogamer.net/i-epicgames-june04
32. Maximum PC. "Game Engines – Exposed!". Maximum PC. No. Fall 2004 (Special ed.). Future US. pp. 59, 62–64. Archived from the original on October 16, 2019. Retrieved August 11, 2017.
33. A Top-Grossing iOS Game Like Epic's Infinity Blade II Can Earn More Than $5 Million a Month [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.adweek.com/performance-marketing/a-top-grossing-ios-game-like-epic%E2%80%99s-infinity-blade-ii-can-earn-more-than-5-million-a-month/
34. See Epic's Unreal Engine 3 running in HTML5 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.gamedeveloper.com/programming/see-epic-s-unreal-engine-3-running-in-html5
35. Geomerics Announces New Enlighten Integration with Unreal Engine 3 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://web.archive.org/web/20120605210049/http://www.unrealengine.com/news/geomerics_announces_new_enlighten_integration_with_unreal_engine_3/
36. Epic Games Founder Tim Sweeney Pushes Unreal Engine 3 Technology Forward [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.forbes.com/sites/johngaudiosi/2011/09/21/epic-games-founder-tim-sweeney-pushes-unreal-engine-3-technology-forward/?sh=2d84f13548b4

37. Steamworks Integration Now Available to Unreal Engine 3 Licensees [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://web.archive.org/web/20100517122844/http://epicgames.com/press_releases/steamworks.html
38. Epic launches Unreal Engine 5 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.gamesindustry.biz/articles/2022-04-05-epic-launches-unreal-engine-5
39. Epic Games: Unreal Engine 5 will bring a generational change to graphics [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://venturebeat.com/2020/05/13/how-epic-games-is-tailoring-unreal-engine-5-to-make-next-gen-graphics-shine/
40. The Matrix Awakens imagines the future of storytelling in Unreal Engine 5 [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.polygon.com/interviews/22827126/the-matrix-awakens-unreal-engine-5-ps5-xbox-series-x
41. Blueprint Overview [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО- НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ГРИ «MIGHTY TOAD» ЖАНРУ ПЛАТФОРМЕР НА ОСНОВІ РУШІЯ UNREAL ENGINE

Виконав студент 5 курсу
Групи ППЗ-51
Єрмак Б.С.
Керівникроботи
Негоденко ОВ.

Київ – 2022

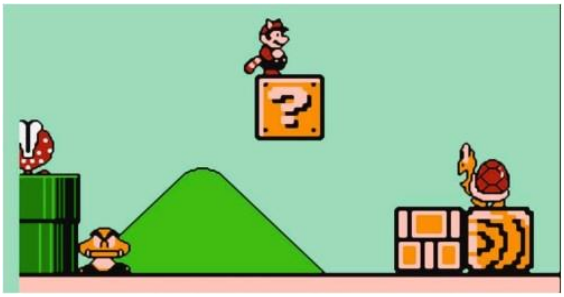
Актуальність теми

Внаслідок карантинних обмежень спричинених глобальною пандемією 2019-2022 рр. люди стали проводити вдома набагато більше часу. Перебування вдома означало, що кількість людей, які грають у ігри, різко зросла. З 2019 по 2021 роки кількість завантажень ігор у світі зросла на 75%, а час перегляду стрімів ігор збільшився на 45%. У 2020 році світовий ринок ігор досяг 162,32 мільярда доларів. Очікується, що до 2026 року ця цифра зросте до 295,63 мільярда доларів. Іншими словами, галузь процвітає і неймовірно конкурентоспроможна .

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи -** розробити та вдосконалити комп’ютерну гру, з можливістю її використання на платформі Windows.
- **Об’єкт дослідження -** процес розробки двовимірної комп’ютерної гри у жанрі платформер.
- **Предмет дослідження -** методи, прийоми та інформаційні технології розробки ігрового програмного забезпечення.

АНАЛОГИ



Super Mario Bros. 3

Limbo

ТЕХНІЧНІ ЗАВДАННЯ

Гра Mighty Toad повинна бути реалізована для платформи Windows та мати наступні характеристики:

- двовимірний світ;
- можливість контролювати персонажа за допомогою клавіатури або геймпаду;
- рівень має складатися з статичних та динамічних платформ;
- персонаж має обмежену кількість спроб на подолання рівня;
- персонаж має засоби для знешкодження ворогів та перешкод;

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

UML діаграма базових класівUnreal Engine

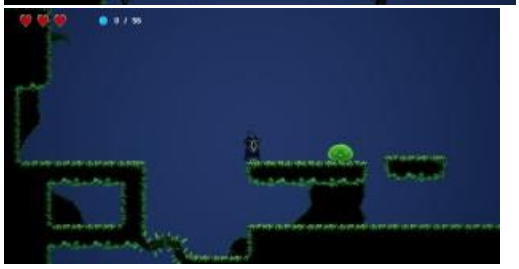
МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Блок схема алгоритмупідбору анімаціїперсонажа

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Блок схема алгоритму підбору анімації персонажа

Ігровий процес



Ігровий процес

ВИСНОВКИ

1. Для досягнення поставленої мети було проаналізовано актуальність даної теми, реалізовано необхідні функціональні вимоги до гри, досліджено методи взаємодії з ігровим рушієм, побудовано можель та архітектуру розроблюваного програмного забезпечення.

2. Було розроблено концепт майбутньої відеогри та визначені основні характеристики проекту.

3. У процесі проектування відеогри були розроблені об’єкти які можуть бути використаними повторно у наступних рівнях чи навіть інших проектах. Окрім цього було доповнено один з базових класів для персонажа, що може допомогти при розробці наступних ігор.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Робота пройшла апробацію на Науково-технічній конференції «Застосування програмного забезпечення в ІКТ», м. Київ, ДУТ, 20 квітня 2022 року.

ДЯКУЮ ЗА УВАГУ!

КОД ПРОГРАМИ

Файл *MightyToadCharacter.h*

```
#pragma once

#include "CoreMinimal.h"
#include "PaperCharacter.h"
#include "AProjectile.h"
#include "MightyToadCharacter.generated.h"

class UTextRenderComponent;

/**
 * This class is the default character for MightyToad, and it is
responsible for all
 * physical interaction between the player and the world.
 *
 * The capsule component (inherited from ACharacter) handles collision
with the world
 * The CharacterMovementComponent (inherited from ACharacter) handles
movement of the collision capsule
 * The Sprite component (inherited from APaperCharacter) handles the
visuals
 */
UCLASS(config=Game)
class AMightyToadCharacter : public APaperCharacter
{
	GENERATED_BODY()

	/** Side view camera */
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category=Camera,
meta=(AllowPrivateAccess="true"))
	class UCameraComponent* SideViewCameraComponent;

	/** Camera boom positioning the camera beside the character */
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera,
meta = (AllowPrivateAccess = "true"))
	class USpringArmComponent* CameraBoom;

	UTextRenderComponent* TextComponent;
	virtual void Tick(float DeltaSeconds) override;
protected:
	// The animation to play while running around
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category=Animations)
	class UPaperFlipbook* RunningAnimation;

	// The animation to play while idle (standing still)
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Animations)
	class UPaperFlipbook* IdleAnimation;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Animations)
	class UPaperFlipbook* JumpAnimation;
```

```
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Gameplay)
	FVector MuzzleOffset;

	// Projectile class to spawn.
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Projectile)
	TSubclassOf<class AAProjectile> ProjectileClass;

	/** Called to choose the correct animation to play based on the
character's movement state */
	void UpdateAnimation();

	/** Called for side to side input */
	void MoveRight(float Value);

	void UpdateCharacter();

	/** Handle touch inputs. */
	void TouchStarted(const ETouchIndex::Type FingerIndex, const
FVector Location);

	/** Handle touch stop event. */
	void TouchStopped(const ETouchIndex::Type FingerIndex, const
FVector Location);

	// APawn interface
	virtual void SetupPlayerInputComponent(class UInputComponent*
InputComponent) override;
	// End of APawn interface

public:
	AMightyToadCharacter();

	/** Returns SideViewCameraComponent subobject **/
	FORCEINLINE class UCameraComponent* GetSideViewCameraComponent()
const { return SideViewCameraComponent; }
	/** Returns CameraBoom subobject **/
	FORCEINLINE class USpringArmComponent* GetCameraBoom() const {
return CameraBoom; }

	// Function that handles firing projectiles.
	UFUNCTION()
	void Fire();


private:
	//int JumpAnimPlayed = 0;
	int JumpButtonPressed;
	bool JumpAnimPlaying = false;

	void StartJumpAnimation();

	UFUNCTION(BlueprintCallable, category = "Animation")
	void OnAnimationStopped();
};
```

Файл *MightyToadCharacter.cpp*

```
#include "MightyToadCharacter.h"
#include "PaperFlipbookComponent.h"
#include "Components/TextRenderComponent.h"
#include "Components/CapsuleComponent.h"
#include "Components/InputComponent.h"
#include "GameFramework/SpringArmComponent.h"
#include "GameFramework/CharacterMovementComponent.h"
#include "GameFramework/Controller.h"
#include "Camera/CameraComponent.h"

DEFINE_LOG_CATEGORY_STATIC(SideScrollerCharacter, Log, All);

//////////////////////////////////////////////////////////////////////////
//
// AMightyToadCharacter

AMightyToadCharacter::AMightyToadCharacter()
{
	// Use only Yaw from the controller and ignore the rest of the
rotation.
	bUseControllerRotationPitch = false;
	bUseControllerRotationYaw = true;
	bUseControllerRotationRoll = false;

	// Set the size of our collision capsule.
	GetCapsuleComponent()->SetCapsuleHalfHeight(96.0f);
	GetCapsuleComponent()->SetCapsuleRadius(40.0f);

	// Create a camera boom attached to the root (capsule)
	CameraBoom =
CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));
	CameraBoom->SetupAttachment(RootComponent);
	CameraBoom->TargetArmLength = 500.0f;
	CameraBoom->SocketOffset = FVector(0.0f, 0.0f, 75.0f);
	CameraBoom->SetUsingAbsoluteRotation(true);
	CameraBoom->bDoCollisionTest = false;
	CameraBoom->SetRelativeRotation(FRotator(0.0f, -90.0f, 0.0f));


	// Create an orthographic camera (no perspective) and attach it to
the boom
	SideViewCameraComponent =
CreateDefaultSubobject<UCameraComponent>(TEXT("SideViewCamera"));
	SideViewCameraComponent->ProjectionMode =
ECameraProjectionMode::Orthographic;
	SideViewCameraComponent->OrthoWidth = 2048.0f;
	SideViewCameraComponent->SetupAttachment(CameraBoom,
USpringArmComponent::SocketName);

	// Prevent all automatic rotation behavior on the camera,
character, and camera component
	CameraBoom->SetUsingAbsoluteRotation(true);
```

```
	SideViewCameraComponent->bUsePawnControlRotation = false;
	SideViewCameraComponent->bAutoActivate = true;
	GetCharacterMovement()->bOrientRotationToMovement = false;

	// Configure character movement
	GetCharacterMovement()->GravityScale = 2.0f;
	GetCharacterMovement()->AirControl = 0.80f;
	GetCharacterMovement()->JumpZVelocity = 1000.f;
	GetCharacterMovement()->GroundFriction = 3.0f;
	GetCharacterMovement()->MaxWalkSpeed = 600.0f;
	GetCharacterMovement()->MaxFlySpeed = 600.0f;

	// Lock character motion onto the XZ plane, so the character can't
move in or out of the screen
	GetCharacterMovement()->bConstrainToPlane = true;
	GetCharacterMovement()->SetPlaneConstraintNormal(FVector(0.0f, -
1.0f, 0.0f));

	// Behave like a traditional 2D platformer character, with a flat
bottom instead of a curved capsule bottom
	// Note: This can cause a little floating when going up inclines;
you can choose the tradeoff between better
	// behavior on the edge of a ledge versus inclines by setting this
to true or false
	GetCharacterMovement()->bUseFlatBaseForFloorChecks = true;

    //     TextComponent =
CreateDefaultSubobject<UTextRenderComponent>(TEXT("IncarGear"));
    //     TextComponent->SetRelativeScale3D(FVector(3.0f, 3.0f, 3.0f));
    //     TextComponent->SetRelativeLocation(FVector(35.0f, 5.0f,
20.0f));
    //     TextComponent->SetRelativeRotation(FRotator(0.0f, 90.0f,
0.0f));
    //     TextComponent->SetupAttachment(RootComponent);

	// Enable replication on the Sprite component so animations show up
when networked
	GetSprite()->SetIsReplicated(true);
	bReplicates = true;
}

//////////////////////////////////////////////////////////////////////////
//
// Animation

void AMightyToadCharacter::StartJumpAnimation()
{
	GetSprite()->SetFlipbook(JumpAnimation);
	GetSprite()->SetLooping(false);
	JumpAnimPlaying = true;
}

void AMightyToadCharacter::OnAnimationStopped()
{
	GetSprite()->SetFlipbook(IdleAnimation);
	GetSprite()->SetLooping(true);
```

```
	GetSprite()->PlayFromStart();
	JumpAnimPlaying = false;
}

void AMightyToadCharacter::UpdateAnimation()
{
	UPaperFlipbook* DesiredAnimation = nullptr;

	if (GetCharacterMovement()->IsFalling())
	{
		if (JumpCurrentCount > JumpButtonPressed)
		{
			++JumpButtonPressed;
			StartJumpAnimation();
			return;
		}
		else if (JumpAnimPlaying)
		{
			return;
		}
		else
		{
			DesiredAnimation = IdleAnimation;
		}
	}
	else
	{
		const FVector PlayerVelocity = GetVelocity();
		const float PlayerSpeedSqr = PlayerVelocity.SizeSquared();
		DesiredAnimation = (PlayerSpeedSqr > 0.0f) ? RunningAnimation
: IdleAnimation;
		JumpButtonPressed = 0;
	}

	if (DesiredAnimation && GetSprite()->GetFlipbook() !=
DesiredAnimation)
	{
		GetSprite()->SetFlipbook(DesiredAnimation);
	}
}

void AMightyToadCharacter::Tick(float DeltaSeconds)
{
	Super::Tick(DeltaSeconds);

	UpdateCharacter();
}


//////////////////////////////////////////////////////////////////////////
//
// Input

void AMightyToadCharacter::SetupPlayerInputComponent(class
UInputComponent* PlayerInputComponent)
{
```

```
	// Note: the 'Jump' action and the 'MoveRight' axis are bound to actual keys/buttons/sticks in DefaultInput.ini (editable from Project Settings..Input)
	PlayerInputComponent->BindAction("Jump", IE_Pressed, this, &ACharacter::Jump);
	PlayerInputComponent->BindAction("Jump", IE_Released, this, &ACharacter::StopJumping);
	PlayerInputComponent->BindAxis("MoveRight", this, &AMightyToadCharacter::MoveRight);

	PlayerInputComponent->BindTouch(IE_Pressed, this, &AMightyToadCharacter::TouchStarted);
	PlayerInputComponent->BindTouch(IE_Released, this, &AMightyToadCharacter::TouchStopped);

	PlayerInputComponent->BindAction("Fire", IE_Pressed, this, &AMightyToadCharacter::Fire);
}

void AMightyToadCharacter::MoveRight(float Value)
{
	/*UpdateChar();*/

	// Apply the input to the character motion
	AddMovementInput(FVector(1.0f, 0.0f, 0.0f), Value);
}

void AMightyToadCharacter::TouchStarted(const ETouchIndex::Type FingerIndex, const FVector Location)
{
	// Jump on any touch
	Jump();
}

void AMightyToadCharacter::TouchStopped(const ETouchIndex::Type FingerIndex, const FVector Location)
{
	// Cease jumping once touch stopped
	StopJumping();
}

void AMightyToadCharacter::UpdateCharacter()
{
	// Update animation to match the motion
	UpdateAnimation();

	// Now setup the rotation of the controller based on the direction we are travelling
	const FVector PlayerVelocity = GetVelocity();
	float TravelDirection = PlayerVelocity.X;
	// Set the rotation so that the character faces his direction of travel.
	if (Controller != nullptr)
	{
		if (TravelDirection < 0.0f)
		{
```

```
                Controller->SetControlRotation(FRotator(0.0, 180.0f,
0.0f));
            }
            else if (TravelDirection > 0.0f)
            {
                Controller->SetControlRotation(FRotator(0.0f, 0.0f,
0.0f));
            }
      }
}

void AMightyToadCharacter::Fire()
{
      // Attempt to fire a projectile.
      if (ProjectileClass)
      {
            // Get the camera transform.
            FVector CameraLocation;
            FRotator CameraRotation;
            GetActorEyesViewPoint(CameraLocation, CameraRotation);

            // Set MuzzleOffset to spawn projectiles slightly in front of
the camera.
            MuzzleOffset.Set(100.0f, 0.0f, -50.0f);

            // Transform MuzzleOffset from camera space to world space.
            FVector MuzzleLocation = CameraLocation +
FTransform(CameraRotation).TransformVector(MuzzleOffset);

            // Skew the aim to be slightly upwards.
            FRotator MuzzleRotation = CameraRotation;
            //MuzzleRotation.Pitch += -90.0f;


            UWorld* World = GetWorld();
            if (World)
            {
                FActorSpawnParameters SpawnParams;
                SpawnParams.Owner = this;
                SpawnParams.Instigator = GetInstigator();

                // Spawn the projectile at the muzzle.
                AAProjectile* Projectile = World-
>SpawnActor<AAProjectile>(ProjectileClass, MuzzleLocation,
MuzzleRotation, SpawnParams);
                if (Projectile)
                {
                    // Set the projectile's initial trajectory.
                    FVector LaunchDirection = MuzzleRotation.Vector();
                    Projectile->FireInDirection(LaunchDirection);
                }
            }
      }
}
```

Файл *AProjectile.h*

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "GameFramework/ProjectileMovementComponent.h"
#include "Components/SphereComponent.h"
#include "AProjectile.generated.h"

UCLASS()
class MIGHTYTOAD_API AAProjectile : public AActor
{
	GENERATED_BODY()
public:
	// Sets default values for this actor's properties
	AAProjectile();

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

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

	// Projectile movement component.
	UPROPERTY(VisibleAnywhere, Category = Projectile)
	UProjectileMovementComponent* ProjectileMovementComponent;

	// Sphere collision component.
	UPROPERTY(VisibleDefaultsOnly, Category = Projectile)
	USphereComponent* CollisionComponent;

	// Function that initializes the projectile's velocity in the shoot
direction.
	void FireInDirection(const FVector& ShootDirection);

	// Function that is called when the projectile hits something.
	UFUNCTION(BlueprintCallable)
	void OnHit(UPrimitiveComponent* HitComponent, AActor* OtherActor,
UPrimitiveComponent* OtherComponent, FVector NormalImpulse, const
FHitResult& Hit);

};
```

Файл *AProjectile.cpp*

```
#include "AProjectile.h"
#include "AEnemy.h"
```

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

	if (!RootComponent)
	{
		RootComponent =
CreateDefaultSubobject<USceneComponent>(TEXT("ProjectileSceneComponent")
);
	}

	if (!CollisionComponent)
	{
		// Use a sphere as a simple collision representation.
		CollisionComponent =
CreateDefaultSubobject<USphereComponent>(TEXT("SphereComponent"));

		// Set the sphere's collision profile name to "Projectile".
		CollisionComponent-
>BodyInstance.SetCollisionProfileName(TEXT("Projectile"));

		// Event called when component hits something.
		CollisionComponent->OnComponentHit.AddDynamic(this,
&AAProjectile::OnHit);

		// Set the sphere's collision radius.
		CollisionComponent->InitSphereRadius(15.0f);

		// Set the root component to be the collision component.
		RootComponent = CollisionComponent;
	}

	if (!ProjectileMovementComponent)
	{
		// Use this component to drive this projectile's movement.
		ProjectileMovementComponent =
CreateDefaultSubobject<UProjectileMovementComponent>(TEXT("ProjectileMov
ementComponent"));
		ProjectileMovementComponent-
>SetUpdatedComponent(CollisionComponent);
		ProjectileMovementComponent->InitialSpeed = 3000.0f;
		ProjectileMovementComponent->MaxSpeed = 3000.0f;
		ProjectileMovementComponent->bRotationFollowsVelocity = true;
		ProjectileMovementComponent->bShouldBounce = true;
		ProjectileMovementComponent->Bounciness = 0.3f;
		ProjectileMovementComponent->ProjectileGravityScale = 0.0f;
	}

	// Delete the projectile after 3 seconds.
	InitialLifeSpan = 3.0f;
}

// Called when the game starts or when spawned
```

```
void AAProjectile::BeginPlay()
{
	Super::BeginPlay();

}

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

}

// Function that initializes the projectile's velocity in the shoot
direction.
void AAProjectile::FireInDirection(const FVector& ShootDirection)
{
	ProjectileMovementComponent->Velocity = ShootDirection *
ProjectileMovementComponent->InitialSpeed;
}

// Function that is called when the projectile hits something.
void AAProjectile::OnHit(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComponent, FVector NormalImpulse,
const FHitResult& Hit)
{
	/*AAEnemy* Enemy = Cast<AAEnemy>(OtherActor);
	if (OtherActor != this && OtherComponent->IsSimulatingPhysics() &&
Enemy)
	{
		Enemy->TakeDamage(1.0f, FDamageEvent(),
GetInstigatorController(), this);
		UE_LOG(LogTemp, Warning, TEXT("Slime TookDamage"))
	}*/

	if (OtherActor != this && OtherComponent->IsSimulatingPhysics())
	{
		OtherComponent-
>AddImpulseAtLocation(ProjectileMovementComponent->Velocity * 100.0f,
Hit.ImpactPoint);
	}

	Destroy();
}
```

Файл *AEnemy.h*

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Components/CapsuleComponent.h"
#include "AEnemy.generated.h"
```

```
UCLASS()
class MIGHTYTOAD_API AAEnemy : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AAEnemy();

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

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


	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Projectile)
	UCapsuleComponent* CollisionComponent;

};
```

Файл *AEnemy.cpp*

```
#include "AEnemy.h"

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

	if (!CollisionComponent)
	{
		CollisionComponent =
CreateDefaultSubobject<UCapsuleComponent>(TEXT("CapsuleComponent"));

		CollisionComponent-
>BodyInstance.SetCollisionProfileName(TEXT("Enemy"));

		// Set the root component to be the collision component.
		RootComponent = CollisionComponent;
	}

}

// Called when the game starts or when spawned
void AAEnemy::BeginPlay()
{
```

```
	Super::BeginPlay();

}

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

}
```