МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КАПЛЯ ГЕОРГІЙ ОЛЕКСАНДРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

ІГРОВИЙ ДОДАТОК НА БАЗІ ТЕХНОЛОГІЙ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ BLUEPRINT

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Р. М. Бабаков, професор кафедри
інформаційних технологій,
д. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця - 2024

АНОТАЦІЯ

Капля Г.О. Ігровий додаток на базі технологій візуального програмування Blueprint. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2024.

У кваліфікаційній (бакалаврській) роботі досліджена та проаналізована робота із візуальним середовищем програмування Blueprint ігрового рушія unreal engine, за допомогою якого був створений ігровий додаток із використанням мережевого з'єднання онлайн платформи steam.

Ключові слова: Unreal Engine, Blueprint, візуальне програмування, ООП.

56 ст. 16 рис., 2 табл., 1 дод., 42 джерел.

ABSTRACT

Kaplia H. Game application based on Blueprint visual programming technologies. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2024.

In the qualification (bachelor's) thesis, the work with the visual programming environment of the unreal engine blueprint game engine, which was used to create a game application using the network connection of the steam online platform, was researched and analyzed.

Keywords: Unreal Engine, Blueprint, visual programming, OOP.

ЗМІСТ

ВСТУП

Популярність відеоігор невпинно зростає, що призводить до постійно зростаючого попиту на нові ігрові проекти. Однак, створення відеогри є складним і трудомістким процесом, який вимагає креативності та інноваційних ідей для розробки унікальних ігрових механік, світів та сюжетних ліній.

Технологія візуального програмування, така як Blueprint в Unreal Engine 5, надає розробникам потужні інструменти для реалізації своїх ідей з меншими технічними бар'єрами та витратами часу. Це робить процес розробки більш доступним як для досвідчених розробників, так і для початківців, які тільки роблять перші кроки у відеоігровій індустрії.

Метою даної бакалаврської роботи є дослідження можливостей застосування технології візуального програмування Blueprint та створення за його допомогою повноцінного відеоігрового додатку.

Завдання дослідження:

1. Аналіз відеоігрового ринку для визначення концепції та жанру відеоігрового застосунку.
2. Дослідження особливостей візуального програмування Blueprint.
3. Визначення основних вимог до кінцевого продукту.
4. Розробка повноцінного відеоігрового додатку з Unreal Engine та Blueprint.
5. Тестування кінцевого продукту на комп'ютерах з різними специфікаціями.

Об’єктом дослідження є повний процес розробки відеоігрового додатку із використанням технологій візуального програмування.

Предметом дослідження є можливості та особливості візуального програмування у Blueprint в Unreal Engine 5.3.

Проміжні результати дослідження пройшли апробацію на V Всеукраїнській науково-практичній конференції здобувачів вищої освіти та

молодих вчених (2024 р.) із доповіддю “Використання технології Blueprint в розробці ігрових додатків” та І Міжнародній науково-практичній конференції (2022 р.) із доповіддю “Моделювання польоту балістичної ракети у віртуальному просторі”.

Основна частина бакалаврської роботи складається з трьох розділів, що відображають 3 етапи розробки відеоігрового продукту.

У першому розділі буде проведено дослідження відеоігрового ринку, вибір ігрового рушія та огляд можливостей технології візуального програмування Blueprint. За результатами цього етапу будуть сформовані вимоги до кінцевого продукту.

Другий розділ буде присвячено безпосередній розробці основних ігрових механік, які є фундаментом кінцевого продукту.

У третьому розділі відбудеться фінальна стадія розробки, що включатиме дизайн ігрових рівнів з використанням усіх розроблених механік, збірку проекту та тестування фінального продукту на комп'ютерах з різними специфікаціями.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ВІДЕОІГРОВОГО РИНКУ ТА АНАЛІЗ ІГРОВИХ РУШІЇВ

1.1 Аналіз відеоігрового ринку та вибір жанру гри

Індустрія відеоігор пережила вибухове зростання протягом останніх двох десятиліть, перетворившись з нішевої субкультури на мейнстримний сегмент розваг, що приносить мільярди доларів щороку. Ця галузь, яка колись була обмежена простими пікселізованими іграми, наразі пропонує неймовірно реалістичні та захопливі ігрові світи, здатні занурити гравців у безмежні віртуальні виміри.

Відеоігрова індустрія на даний момент – найприбутковіша індустрія у сфері розваг. За даними Statista на 2024 рік прибуток відеоіргової індустрії становить 211.4 мільярди доларів США, тоді як телевізійна індустрія заробляє лише 94 мільярди доларів, що свідчить про неабияку популярність відеоігор у світі [41][42]. Це пояснюється технологічними досягненнями у сфері графічних процесорів, поширенням стаціонарних комп’ютерів, мобільних пристоїв, а також доступностю високошвидкісного інтернету. Що робить її найперспективнішою індустрією у сфері розвах у сучансому світі [4][5].

Зі зростанням кількості гравців, росте також попит на нові відеоігрові проекти різних жанрів та концепцій. Кожен рік приносить унікальні та інноваційні ігри, які вражають своїми можливостями та створюють нові стандарти в галузі. Багато з цих ігор створюють невеликі розробники, які вже давно стали невід’ємною частиною відеоігрової індустрії [6].

Серед найбільш популярних інді-ігор останіх років можна виділити:

- Deep Rock Galactic – кооперативний шутер у стилі roguelite, де космічні дворфи виконують ризиковані завдання та борються з ворожими істотами.

- Lethal Company – кооперативна відеогра у жанрі roguelike виживання, хоррор, де четверо гравців досліджують небезпечні планети у пошуках ресурсів.
- Stardew Walley – 2D симулятор сільского господарства з елементами рольової гри, який дозволяє кооперативне проходження до восьми гравців.
- Phasmophobia – кооперативна відеогра у жанрі жахів, у якому команда із 4-ох гравців досліджує паранормальні локації із привидами.
- Content Warning – кооперативна відеогра у жанрі жахів, виживання, де четверо гравців намагаються зняти максимально страшне відео в локаціях, населених монстрами.
- Palworld – гра у жанрі виживання, де гравці можуть взаємодіяти з істотами, званими “Палами”, ловити їх у стилі “покемонів” та використовувати для різних цілей. Гра також підтримує кооперативний режим.
- Barotrauma – кооперативна рольова гра у жанрі хоррор виживання, де гравці вивчають на підводному човні дно супутника Юпітера Європи.

Аналізуючи тенденції інді-ігор, можна помітити, що особливу популярність здобувають ігри з кооперативним елементом. Це легко пояснити: спільне проходження ігор з друзями приносить безліч веселих та цікавих моментів, роблячи ігровий процес значно приємнішим і згуртовуючим. Тому жанр гри пов’язаний із активним залученням гравців до командної взаємомідії стане ключовим елементом для майбутньої гри.

Однак розробка гри у жанрі рольової гри, виживання чи шутера із кооперативним елементом є дуже складним і часомістким завданням. Натомість слід обрати більш простіший жанр, такий як головоломка, який буде

оптимальним рішенням, оскільки він вже довів свою ефективність у таких популярних кооперативних проектах як Portal 2, We were here together та Trine.

1.2 Вибір ігрового рушія

Ігровий рушій є критично важливим програмним забезпеченням, що становить серцевину будь-якої відеогри та відповідає за її технічну реалізацію. Він значно спрощує процес розробки ігор, надаючи розробникам комплексний набір інструментів та компонентів для створення, оптимізації та систематизації внутрішньої структури гри. Типовий ігровий рушій включає в себе рушій рендерингу для візуалізації та обробки графіки, фізичний рушій для реалістичної симуляції фізичних взаємодій, системи для обробки звуку, скриптів, анімацій, штучного інтелекту, мережевого коду, керування пам'яттю, багатопотоковості та ієрархічної структури сцен. Завдяки цьому ігрові рушії дозволяють створювати різноманітні ігри, які можуть працювати на широкому діапазоні платформ, від персональних комп'ютерів до мобільних пристроїв та ігрових консолей..

На ринку існує велика кількість різноманітних ігрових рушіїв, таких як Urho3D, Godot, Amazon Lumberyard, LibGDX, RPG Maker, CryEngine, GameMaker та AppGameKit [24][25][26][27][28][29][30]. Проте, два рушії, які виділяються своєю популярністю, універсальністю та гнучкістю, - це Unreal Engine та Unity. Їхня потужність, багатофункціональність, підтримка різних платформ та вигідна цінова політика забезпечили їм лідируючі позиції у галузі розробки ігор [12].

На відміну від більш спеціалізованих рушіїв, таких як RPG Maker, який призначений виключно для створення рольових ігор [8], Unreal Engine та Unity не мають чіткої прив'язки до певного жанру [13][14]. Їхня цінова політика дозволяє початківцям та малим студіям безкоштовно розробляти комерційні проекти до того моменту, поки річний дохід не перевищить певний поріг. Ця

універсальність та доступність роблять їх ідеальними інструментами для створення різноманітних ігрових проектів, включаючи кооперативні головоломки.

Розглянемо детальніше ці два ігрові рушії:

Unreal Engine, розроблений компанією Epic Games, використовує мову програмування C++ для своїх проектів, що збільшує складність вивчення рушія порівняно з Unity, який використовує більш доступну мову C#[15][16]. Однак обидва рушії мають системи візуального скриптингу (Blueprints в Unreal Engine та Visual Scripting в Unity), які можуть значно полегшити процес вивчення та використання рушіїв для початківців [7][11][19].

Аналізуючи інформацію з онлайн-спільноти Unity, можна знайти експеримент, у якому порівнювалася продуктивність систем візуального скриптингу Unity Visual Scripting та Unreal Blueprints див. табл. 1.1. Результати показують, що за більшістю операцій, таких як цикли, арифметичні обчислення, генерація шуму Перлінга, робота зі змінними та списками, Unreal Blueprints демонструє кращу продуктивність порівняно з Unity Visual Scripting.

Таблиця 1.1 Різниця швидкодії Unity Visual Scripting та Unreal Blueprint

Action	Unity Visual Scripting	Unreal Blueprint
For Loop	0.14236450	0.02829439
Addition	0.43872830	0.05233720
Multiplication	0.43047710	0.05263600
Perlin Noise	1.54048500	0.07378839
Random	1.31377400	0.05797909
Set Graph Var	0.33867260	0.04620750
Set Object Var	0.46269230	…
Add List Item	0.37500760	0.05534089
Set List Item	0.40145110	…

Set Array Item	…	0.05658640
If Statement	0.46377180	0.03870380

Безсумнівно, обидва рушії - Unreal Engine та Unity - є потужними та широко використовуваними інструментами для розробки ігор. Однак, Unreal Engine виділяється своїми інноваційними технологіями та постійними оновленням. [32]

Також немало важним фактором є те, що Unreal Engine був розроблений компанією Epic Games, що має значний досвід у розробці відеоігор. Це дає їм глибше розуміння потреб та вимог індустрії, що відображається у функціональності та зручності використання рушія. У той час, як Unity Technologies зосереджена переважно на розвитку самого движка Unity.

Тому, з огляду на ці фактори, для розробки відеогри буде використаний саме рушій Unreal Engine.

1.3 Огляд Blueprint

У рушії Unreal Engine, як зазначено вище, є технологія візуального програмування Blueprint. Хоча Blueprint часто асоціюють лише з візуальним скриптингом, що складається з різноманітних блок-схем, насправді це набагато потужніший інструмент, який виступає у ролі спеціального ресурсу в середовищі Unreal Engine. Завдяки інтуїтивному інтерфейсу Blueprint, розробники можуть створювати та редагувати будь-які класи, визначаючи їхні властивості, компоненти та поведінку. [17][20][21][22]

Інтерфейс Blueprint складається з кількох ключових секцій, які допомагають розробникам візуалізувати та маніпулювати об'єктами класу. Давайте детально розглянемо інтерфейс на прикладі класу BP_ThirdPersonCharacter як показано на рис 1.1. [31]

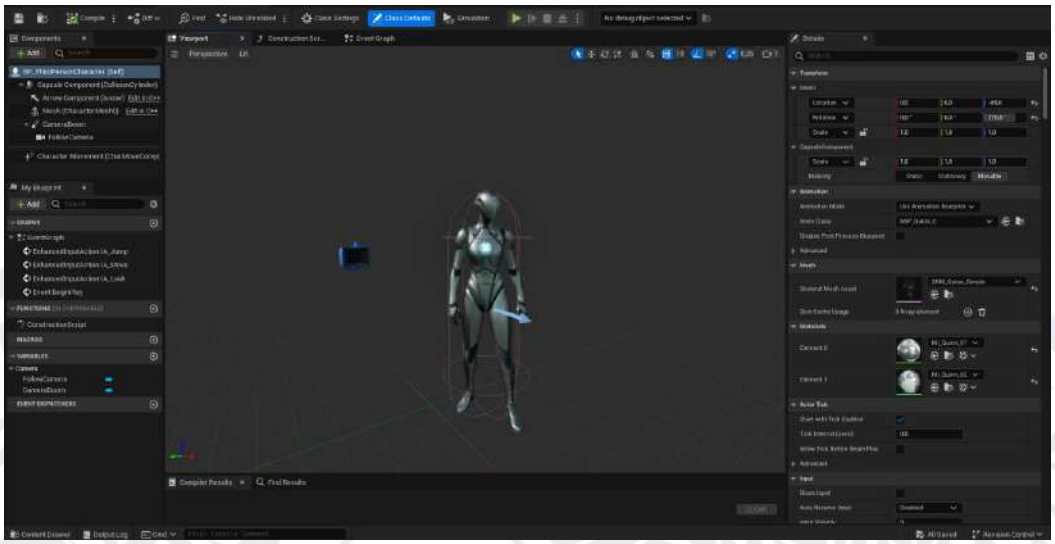

Рисунок 1.1 – Інтерфейс Blueprint класу BP_ThirdPersonCharacter

- Viewport - центральне вікно, у якому відображається зовнішній вигляд об'єкта класу. Це вікно доступне лише для тих класів, які є похідними від класу Actor та можуть бути відображені в ігровому світі. [37]
- Details - вікно з правого боку, де відображаються та редагуються всі властивості обраного об'єкта, включаючи компоненти, змінні, функції, події та налаштування самого класу.[35]
- Components - вікно зліва, що дозволяє додавати, видаляти, обирати та змінювати ієрархію компонентів актора.[38]
- My Blueprint - секція під вікном Components, яка стосується програмної частини Blueprint. Тут відображаються всі події, функції, макроси та змінні, визначені для цього Blueprint класу.
- Construction Script та Event Graph - дві ключові комірки біля вікна Viewport, де безпосередньо відбувається процес візуального програмування. [40]

Візуальне програмування в Blueprint полягає у додаванні та з'єднанні нодів (вузлів), які представляють різноманітні дії, умови, обчислення та інші елементи програмної логіки. Цей процес відбувається у двох основних секціях:[34]

Construction Script - спеціальна секція, яка використовується для налаштування властивостей та компонентів об'єкта під час його створення або переміщення в ігровий світ. Її код виконується лише один раз під час ініціалізації об'єкта, дозволяючи встановлювати початкові значення, створювати та додавати компоненти, задавати трансформації тощо.

Event Graph - основна секція, де визначається поведінка об'єкта під час роботи гри. Код в Event Graph виконується постійно, реагуючи на різноманітні події, такі як натиснення клавіш, зіткнення об'єктів, виклики функцій тощо. Тут можна створювати складну логіку за допомогою вузлів умов, циклів, математичних операцій та інших інструментів візуального програмування, взаємодіяти з компонентами об'єкта та виконувати різноманітні дії, рис 1.2.

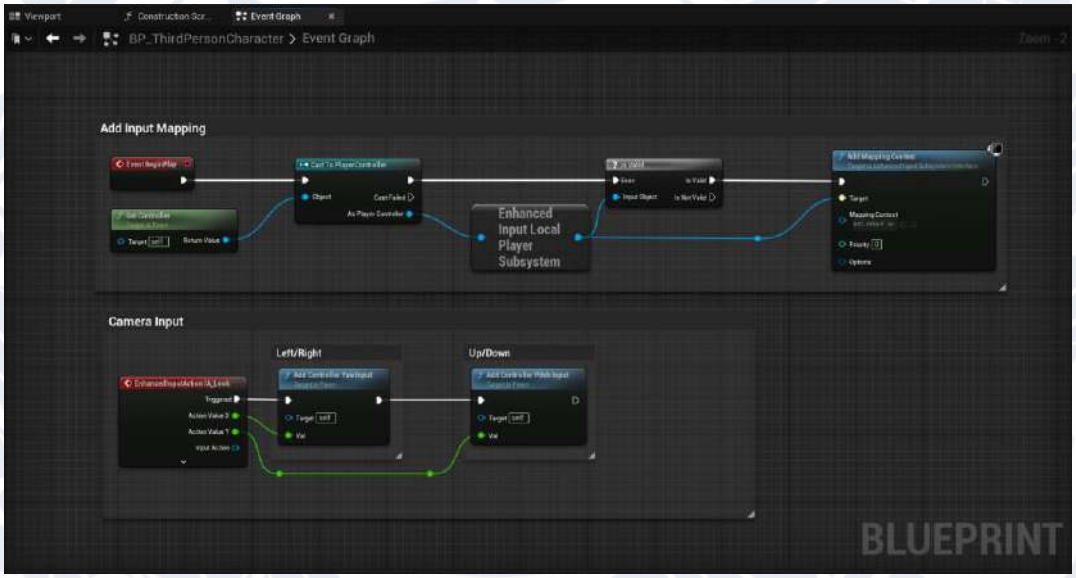

Рисунок 1.2 – приклад програми Blueprint, що представляє собою переміщення камери

Хоча Blueprint має багато переваг, таких як простота вивчення, інтуїтивний інтерфейс та швидка компіляція, що прискорює розробку відеоігрових додатків, він також має певні обмеження. Незважаючи на свою гнучкість, Blueprint досить обмежений у своїх можливостях порівняно з мовою програмування C++. Також, хоча Blueprint швидший від свого головного конкурента Unity Visual Scripting приблизно в 5 разів, він все ж повільніший від класичного програмування в сотні разів. Тому Blueprint зазвичай

використовують для створення прототипів ігрових механік, які потім будуть переписані на C++, та для виконання простих алгоритмів.

Проте, незважаючи на певну повільність та обмеження, Blueprint залишається потужним інструментом, який допоможе не лише швидко розробити відеоігровий додаток, а й полегшить вивчення Unreal Engine для новачків, що стане у нагоді в майбутньому.

1.4 Онлайнова складова рушія

Онлайн-функціональність в Unreal Engine є стандартною і доступною "з коробки". Кожна гра, створена на цьому рушії, може підтримувати мережеве підключення у формі прослуховування сервера (Listen Server), де один з гравців функціонує як власник, одночасно бути і клієнтом, і сервером. Проте просте підключення не забезпечує повноцінної гри, оскільки багато ігрових механік не синхронізуються на клієнтському та серверному рівнях. Наприклад, об'єкт, такий як куб, може бути переміщений і фізика якого буде порізному прорахована між клієнтом і сервером, що призводить до різниці у швидкості та напрямку руху. [2][36]

У Unreal Engine, для вирішення цієї проблеми, існують налаштування реплікації в кожному класі. Реплікація автоматично синхронізує дані та події між сервером і клієнтами, забезпечуючи однаковий результат. Це можна застосовувати до об'єктів, їхніх змін, компонентів та подій, і навіть викликати події лише для сервера або для всіх клієнтів з репліцированими об'єктами

Однак, деякі важливі геймплейні класи в Unreal Engine мають реплікацію за замовчуванням, що потребує ретельного розуміння під час розробки онлайн-ігор рис 1.3.

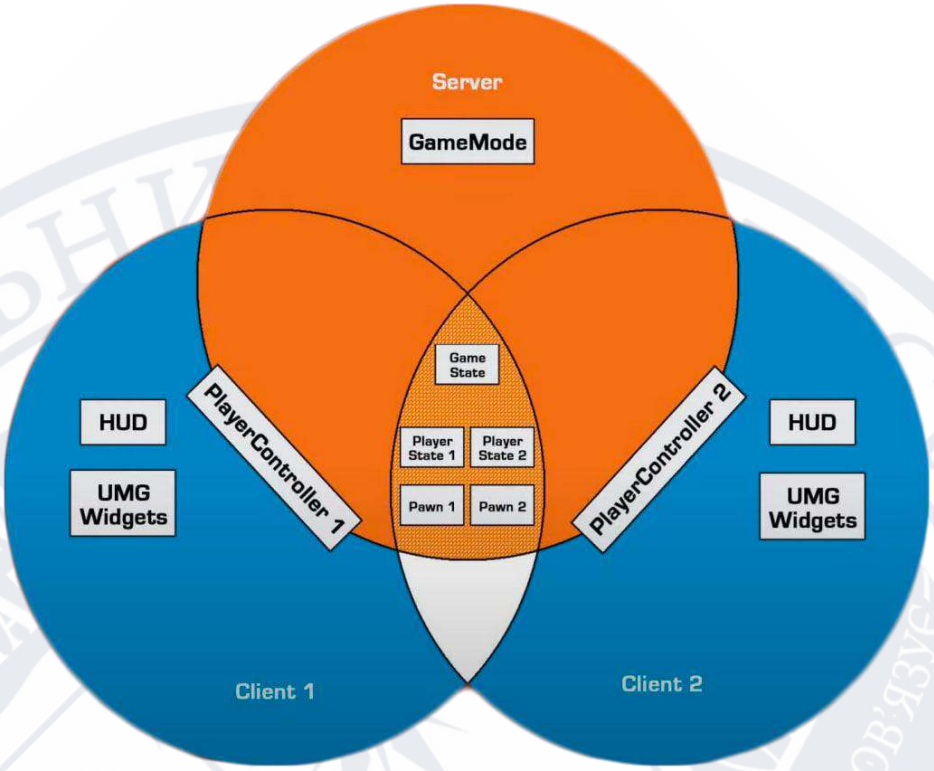

Рисунок 1.3 – реплікація класів елементів гри

На прикладі створення текстового чату у відео грі, що може бути реалізований різними способами, через клас GameMode, або через GameState. У першому випадку, використання GameMode для передачі повідомлень потребує встановлення зв'язку через PlayerController, оскільки UI елементи та логіка чату створюються окремо для кожного клієнта, а GameMode працює тільки на сервері. Це може потребувати додаткових кроків для обміну даними між клієнтами через сервер.

У другому випадку, використання GameState може спростити реалізацію, оскільки він доступний і на сервері, і на клієнтах. Це дозволяє передавати дані про чат безпосередньо між UI елементами і GameState без потреби в посередниках.

Таким чином Unreal Engine має чудовий інструментарій для розробки онлайнової відеогри

1.5 Постановка задачі

Основним завдання цієї роботи є створення відеоігрового додатку. Проаналізувавши можливості ігрового двинуга Unreal Engine можемо вигадати головну ідею гри: Нехай кожен гравець матиме свій унікальний колір. Гравці можуть заволодіти об’єктами того ж кольору та управляти ними. Таким чином пару гравців мають спільними зусиллями брати під контроль потрібні об’єкти та вирішувати різні головоломки, щоб дібратись до наступного чекпоінту.

Таким чином можемо зробити такий план проекту:

1. Створена гра буде кооперативною головоломкою із доволі простою графікою, що є достатньо нескладною у реалізації задачею та доволі цікавою на виході.
2. Проект буде розроблений з використанням технології Blueprint ігрового рушія Unreal Engine 5.3.
3. У грі має бути реалізовано:
   - Онлайнове підключення та створення listen server.
   - Список серверів.
   - Механіка кольорів.
   - Механіка заволодіння об’єктами та повернення.
   - Пару підконтрольний об’єкта.
   - Щонайменш 4 ігрових об’єкта
   - Головне меню та меню під час гри.
   - Налаштування графіки.
   - Внутрішньоігровий чат.
   - Щонайменш 3 ігрових рівня.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ ОСНОВНИХ МЕХАНІК

2.1 Створення та налаштування проекту

У Unreal Engine доступні вбудовані шаблони проектів із попередньо налаштованими елементами, такими як First Person, Third Person, Virtual Reality, Vehicle, Top Down та HandHeld AR. У цьому випадку буде використано шаблон Third Person (третя особа), який автоматично створює 3D персонажа з анімаціями, рухом та камерою з виглядом від третьої особи.

Оскільки планується створити онлайнову відеогру, необхідно обрати один з варіантів мережевого підключення:

- Підключення через локальну мережу - гравці повинні перебувати в одній локальній мережі, при цьому один з гравців виступає в ролі власника сервера, а інші вводять IP-адресу та порт сервера.
- Онлайнове підключення з використанням VPN для віддаленого доступу - схоже на локальне підключення, але гравці повинні завантажити спеціальну VPN-утиліту, наприклад Hamachi, для створення віртуальної приватної мережі, що дозволяє підключатися через глобальну мережу.
- Підключення з використанням сервісів Epic Games - створюється мережева сесія з використанням додаткових плагінів, що дозволяє гравцям створювати та знаходити сервери за допомогою цього сервісу. Гравці повинні мати облікові записи та запустити Epic Games Launcher.
- Підключення з використанням сервісів Steam - спосіб, подібний до попереднього, але з використанням сервісу Steam.

З усіх варіантів найкращим вибором є підключення через сервіси Steam, оскільки Steam є найпопулярнішою онлайн-платформою серед геймерів, а підключення натисканням однієї кнопки замість введення IP-адреси є зручнішим у використанні.

Для підключення онлайн сервісів Steam до гри необхідно виконати наступні дії:

- Конвертувати проект з використанням Blueprint на проект з використанням C++, для цього достатньо створити будь-який C++ клас у проекті, після чого рушій автоматично створить додаткові файли.
- Завантажити плагіни AdvancedSessions та AdvancedSteamSessions з GitHub[3].
- Створити папку Plugins у папці проекту та розпакувати туди завантажені плагіни.
- Перезібрати проект. Для цього треба видалити папки: .vs, DerivedDataCache, Intermediate, saved та файли: .vsconfig, [навза проекту].sln. Після чого за допогою правої кнопки миші натиснути на .uproject файл, обираємо функцію створити файли Visual Studio. Відкриваємо створений .vsl файл та за допомогою Visual Studio перезбираємо проект. Після даних маніпуляцій у проекті мають з’явитись завантажені плагіни
- Відкрити конфігураційний файл \Config\DefaultEngine.ini, де потрібно додати наступні рядки: [1]

```
[/Script/Engine.GameEngine]
+NetDriverDefinitions=(DefName="GameNetDriver",DriverClassName="
OnlineSubsystemSteam.SteamNetDriver",DriverClassNameFallback="On
lineSubsystemUtils.IpNetDriver")
[OnlineSubsystem]
DefaultPlatformService=Steam

[OnlineSubsystemSteam]
bEnabled=true
SteamDevAppId=480
bInitServerOnClient=true
[/Script/OnlineSubsystemSteam.SteamNetDriver]
NetConnectionClassName="OnlineSubsystemSteam.SteamNetConnection"
```

У налаштуваннях конфігураційного файлу DefaultEngine.ini більшість рядків прописані у довіднику Unreal Engine і не потребують змін. Однак, два параметри потребують окремого пояснення:

SteamDevAppId - вказує на ідентифікатор додатку в Steam. Для розробників ігор створено спеціальний додаток з ID 480 "Spacewar", який дозволяє тестувати онлайн-підключення Steam у своїх проектах. Якщо завантажити розроблену гру на платформу Steam, їй буде присвоєно унікальний ідентифікатор проекту, який необхідно буде замінити замість значення 480.

bInitServerOnClient - цей параметр вказує, чи буде серверний функціонал ініціалізовано на клієнтському комп'ютері. У нашому випадку, коли один гравець виступає як клієнт та сервер одночасно, цей параметр має бути встановлений як true.

Після цих всіх маніпуляцій при запуску гри у редакторі Unreal engine в режимі Standalone game, має з’явитись SteamOverlay, що свідчить про успішне підключення гри до сервісу Steam, рис 2.1.

Рисунок 2.1 – Запуск гри із SteamOverlay

2.2 Створення списку серверів, підключення та створення сесії

Створимо новий рівень під назвою "Меню". Цей рівень буде порожнім, без освітлення та об'єктів. Режим гри буде стандартним, без особливих

можливостей, оскільки цей рівень буде використовуватися як головне меню. Тут будуть викликатися та створюватися початкові віджети.

Для створення списку серверів розроблено 3 віджета:

- Віджет списку серверів - містить список (scrollbox) та 3 кнопки: кнопка "Назад" для повернення до попереднього віджета, кнопка "Створити" для переходу до віджета зі створенням сервера та кнопка "Оновити список серверів", натиснувши яку за допомогою плагіну AdvancedSession відбудеться пошук сесій, створених іншими гравцями.
- Віджет знайденого сервера - відображає інформацію про сервер та кнопку для підключення до сесії. Цей віджет створюється під час пошуку серверів і приймає результати пошуку, виводячи інформацію про сервер та дозволяючи підключитися до сесії рис 2.2.

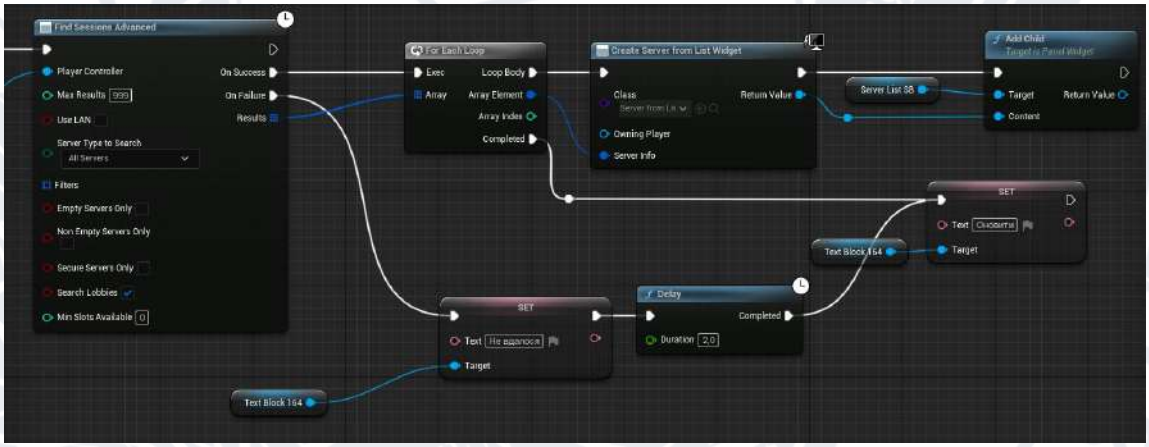

Рисунок 2.2 – Пошук серверів та створення віджетів знайдених серверів

- Віджет для створення сервера - містить елементи combobox для вибору рівня та кількості гравців, які можуть приєднатися до сесії. Також є дві кнопки: одна для запуску алгоритму створення сервера, а інша для повернення до попереднього віджета (списку серверів).

У Blueprint рівня записується створення віджета списку серверів, показується курсор миші та встановлюється режим введення тільки для інтерфейсу.

2.3 Механіка кольорів

У даній грі, персонажі та об’єкти з якими вони можуть взаємодіяти, мають містити один з кольорів із заздалегіть визначеного списку кольорів. Тому для цього був створений Enumerator кольорів, що містить на даний момент 3 кольори, синій червоний та білий. За цією системою гравці із синім кольором зможуть управляти синіми та білими об’єктами, гравці із червоним кольором зможуть управляти червоними та білими об’єктами, гравці що мають білий колір, можуть управляти лише білими об’єктами.

Кольори гравців будуть зберігатись у двох об’єктах класів: у PlayerState (клас, що зберігає стан гравця) та у самому персонажі гравця.

У даних зміних буде налаштована реплікація RepNotify, що не тільки сінхронізує зміну між клієнтами та сервером, а і додає функцію OnRep_(назва зміної), що автоматично викликається при кожній змінній.

Таким чином коли у гравця буде змінювати колір сервером (класом Gamemode), у класі PlayerState – то автоматично викличиться функція, що змінить колір персонажа, а у персонажа виклиться функція, що змінить деякі його властивості, що відображатимуть його колір, наприклад зміна кольору імені гравця або зміна UI елементів.

2.4 Механіка заволодіння об’єкта

Механіка заволодіння об’єкта буде виглядати наступним чином: Гравець цілиться на об'єкт та натискає клавішу (за замовчуванням ліву кнопку миші). Якщо відбулося попадання на ігровий об'єкт, він звіряє свій колір із кольором гравця. У випадку успішної перевірки, гравець втілюється в об'єкт що дає доступ до контролювання його рухів. Також, коли гравець втілений в об'єкт, натискання клавіші повернення (за замовчуванням, R) призводить до того, що гравець знову заволодіває своїм попереднім персонажем.

Щоб втілити механіку захоплення об'єктів, спочатку необхідно створити події, які спрацьовуватимуть при натисканні відповідних клавіш. У даному випадку були створені дві дії введення (input actions), що являють собою логічні значення. Ці дві дії були записані, а клавіші для них визначені у карті введення (Input Mapping Context), яка вже була створена ігровим рушієм. Потім ця карта введення додається до ініціалізується у Blueprint графі персонажа, надаючи доступ до подій, викликаних зазначеними діями введення.

Після виклику події заволодіння персонажем, функція, яку обробляє сервер, створює промінь певної довжини, початкова точка якого йде від камери до центру екрана гравця (де знаходитиметься приціл гравця). У разі попадання променя по об'єкту, сервер викликає функцію Blueprint інтерфейсу PossessTarget, яка приймає три параметри:

- Контролер гравця – щоб гравець зміг отримати контроль над ігровим об’єктом
- Колір гравця – для звіряння кольорів персонажа та об’єкта
- Посилання на персонажа, звідки йде виклик – щоб гравець зміг повернутись до свого персонажа.

Створемо батьківський клас ігрових об'єктів, якими можуть заволодівати гравці, що є дочірним класом pawn (пішак). Даний клас може

управлятись гравцем як і клас Character (персонаж), але має меньше різноманітних функцій.

У даному класі, зберігаються змінні кольорів: поточного та початкового (у разі заволодіння білим об'єктом синім або червоним гравцем), логічне значення чи є об'єкт вже підконтрольним, та посилання на персонажа гравця для повернення гравця назад до свого персонажа.

До цього класу під'єднаний Blueprint інтерфейс Possess, у якому є функція PossessTarget. При виклику цієї функції даний клас робить ряд перевірок, і у разі успіху, ним заволодіває гравець.

Також додана подія повернення гравця до свого основного персонажа, яка викликається натисканням клавіші повернення персонажа, що викликає на сервері функцію цьогож інтерфейсу із посиланням на головного персонажа гравця. Таким чином, персонаж також робить ряд перевірок, і гравець отримує контроль над своїм персонажем назад.

Варто зазначити, що коли у персонажа гравця зникає контролер, персонаж може вести себе дивно, наприклад, застигнути в повітрі або продовжити рух на деяких клієнтах. Для вирішення цієї проблеми, при зникненні контролера гравця, персонажем володіє контролер штучного інтелекту, який нічого не робить [18].

2.5 Створення ігрових об’єктів

2.5.1 Ігровий об’єкт - куб

Раніше вже був створений батьківський клас ігрового об’єкта з реалізованою логікою заволодіння об'єкта гравцем. У цьому випадку достатньо створити дочірній клас цього об'єкта та описати його унікальні властивості та рухи.

Першим ігровим об'єктом буде звичайний куб. Цей об'єкт виконуватиме дві функції: як об'єкт, на який можна забратися, та як об'єкт, що активуватиме натисну плиту відповідного кольору.

Даний об’єкт має змінювати свій колір залежно від поточного кольору, а також рухатися, отримуючи фізичну силу по горизонталі, залежно від натиснутих клавіш гравцем.

У класі куба, налаштований компонент статичної сітки (StaticMesh), що представляє собою куб, таким чином, щоб об'єкт міг симулювати фізику та реплікуватися на інші клієнти. Також додана камера від третьої особи.

У Blueprint графі, а саме у функції що викликається після зміни поточного кольору, створена блоксхема, що автоматично змінює матеріал сітки куба в залежності від поточного кольору. Таким чином з боку гравця виглядатиме так, що куб просто змінив колір на відповідний.

Також у Blueprint графі створені події переміщення камери та переміщення персонажа. Переміщення камери змінюється лише на клієнті і залежить від зміни положення комп'ютерної миші гравця. Натомість переміщення самого куба викликається на сервері, якому передається обрахований вектор сили на клієнті (оскільки він залежить від повороту камери). Далі сервер зрівнює поточну швидкість куба та максимальну допустиму швидкість, і у разі успіху надає силу, що рухає куб.

Оскільки персонаж, має активовувати натисну плиту відповідного кольору, був створений для цього Blueprint інтерфейс getcolor, що просто виводить поточний колір, а плита у майбутньому, просто буде викликати цю функцію і робити влану перевірку. Дана функція інтерфейсу буде реалізована у батьківському класі об’єктів

2.5.2 Ігровий об’єкт - Платформа

Платформа - це ігровий об'єкт, який рухається по заздалегідь визначеній траєкторії, заданій сплайном (spline) рис 2.3. Мета цього об'єкта - переносити персонажів з точки А в точку В. Гравець, який заволодіє цією платформою, зможе лише плавно змінювати її швидкість і рухатися тільки вперед або назад.

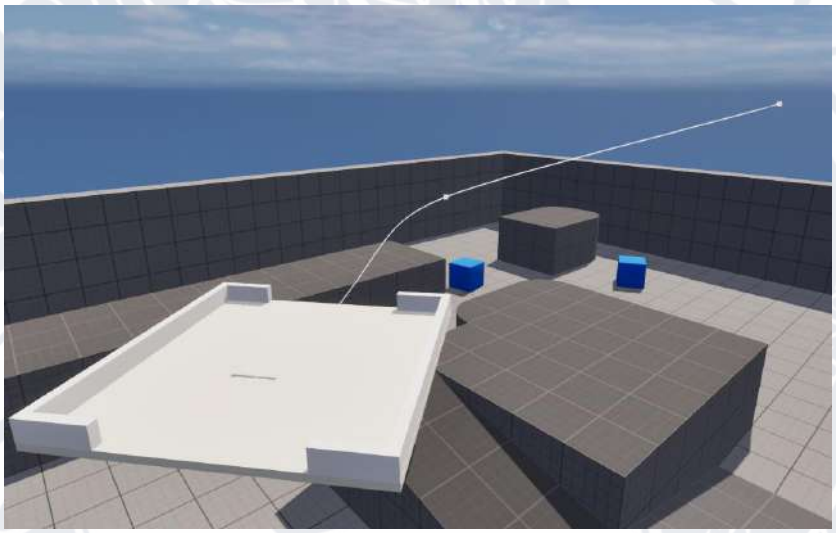

Рисунок 2.3 – рух платформи за сплайном

Платформа як і куб, буде змінювати свій колір в залежності від поточного кольроку, а також матиме рухому камеру від третьої особи. Але на відміну від куба, платформа не симулюватиме фізику і матиме іншу форму з додатковими статичними сітками.

Платформа матиме додаткові змінні, такі як: посилання на сплайн, вздовж якого вона рухатиметься, пройдена відстань, поточна швидкість, максимальна швидкість, зменшення швидкості та логічне значення повернення платформи на старт сплайну.

Переміщення платформи відбуватиметься таким чином: гравець може контролювати лише швидкість, натискаючи клавіші "вперед" або "назад". Це викликає функцію на сервері, яка змінює швидкість платформи, якщо вона не досягла максимуму. Якщо платформа не стоїть на місці або має повернутися назад, у події Tick (подія, що викликається кожен кадр) викликається інша серверна функція, яка переміщує об'єкт вздовж сплайну на відстань, записану

в його змінних. Після цього відбувається низка математичних обчислень, які змінюють відстань та поступово зменшують швидкість платформи. Таким чином, об'єкт зможе продовжити рух, навіть коли гравець повернувся до свого персонажа, а також дозволяє керувати цією платформою автономно, що може бути використано в інших ігрових механіках.

2.6 Створення акторів

2.6.1 Натисні плити

Актори (Actors) - це клас об'єктів, якими не можна керувати безпосередньо гравцям, але вони можуть бути використані на сцені та виконувати різні алгоритми.

Одними з важливих об’єктів даної відеогри – є натисна плита персонажа та натисна для куба. Їхнє призначення - проаналізувати об'єкт, який на них наступив, і в разі успішного проходження низки перевірок, відтворити відповідну анімацію натискання та активувати прив'язану до них подію.

Ці плити мають подібні компоненти та змінні: статичні сітки, що описують їхнє тло та саму плиту (кругла форма для плити персонажа, прямокутна - для плити куба, рис 2.4), прямокутну колізію, перетин якої викличе подію перевірки об'єкта на відповідні властивості, а також логічну змінну з реплікацією RepNotify, яка відображатиме стан плити.

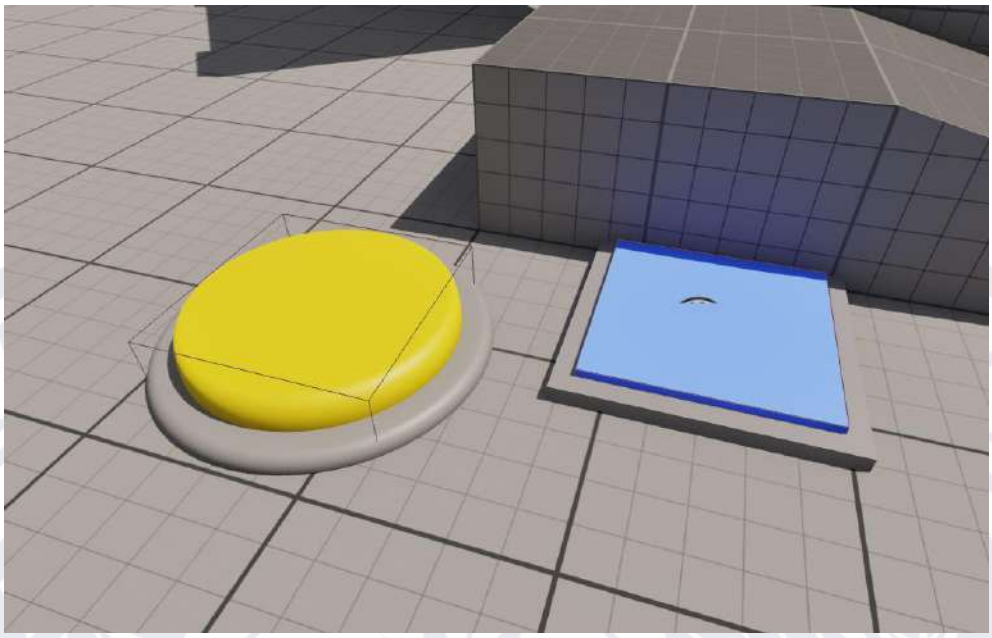

Рисунок 2.4 – зовнішній вигляд плит

При перетині колізії плити персонажа, вона перевірятиме, чи належить клас перетнутого об'єкта до класу Character, і в разі успіху змінить свій стан на true (натиснуто) та відтворить анімацію натискання.

При зміні стану плити викличеться функція, яка запустить створений диспетчер подій (Event Dispatcher). За його допомогою викличеться подія в іншому класі, зокрема на рівні, який був заздалегідь пов'язаний з цією подією.

Логіка плити куба буде схожою, за винятком того, що ця плита також перевірятиме не лише клас об'єкта, а й його колір за допомогою виклику функції Blueprint інтерфейсу GetColor, створеної раніше для батьківського класу підконтрольних об'єктів.

2.6.2 Брама

Брама - це актор, мета якого перешкоджати руху гравців до того моменту, поки вони не виконають певну дію, наприклад, не натиснуть на плиту.

Клас актора “Брама” матиме два компоненти - дві статичні сітки, які при відкритті брами будуть віддалятися одна від одної. Також цей клас міститиме

лише одну логічну змінну з реплікацією RepNotify, що відображатиме стан брами (відкритий або закритий).

У Blueprint графі подій будуть лише дві події, які відтворюватимуть анімацію відкриття та закриття брам. Ці події викликатимуться лише після зміни логічної змінної і викликатимуться залежно від її стану. Таким чином, якщо стан брами змінюється на true (відкритий), це викличе подію відкриття дверей, і відповідна анімація почне програватися. У разі зміни стану на false (закритий), викличеться подія закриття дверей, і анімація програватиметься у зворотному порядку.

Отже, брама - це простий актор з двома станами (відкриті/закриті), що керуються логічною змінною. Зміна цієї змінної, ймовірно, буде ініційована натисканням плит або виконанням інших умов у грі, що дозволить гравцям рухатись далі.

2.6.3 Плафторма-актор

Платформа-актор - це ігровий об'єкт, який не може керуватися гравцем безпосередньо. Його мета - переносити об'єкти з точки А в точку В. На відміну від підконтрольної платформи, ця може прив'язати до себе об'єкт і рухатися по горизонталі, керуючись різними чиниками, такими як натисні плити.

Платформа-актор матиме 3 компоненти: статичну сітку, прямокутну колізію та компонент PhysicsConstraint, який дозволяє фізично прив'язати кілька об'єктів.

У Blueprint графі подій всі події викликатимуться ззовні (наприклад, диспетчером подій натисної плити) і матиме такі події::

- Прив’язка об’єкта.
- Переміститись вперед.
- Переміститись назад.
- Переміститись вліво.

- Переміститись право.

При запуску події прив'язки об'єкта платформа-актор перевірятиме всі об'єкти, що знаходяться в її прямокутній колізії, на відповідність певним класам, таким як куб. У разі успішної перевірки, вона з'єднає себе та об'єкт за допомогою PhysicsConstraint. Це з'єднання можна розірвати двома способами: або викликати знову функцію прив'язки об'єкта, що розірве з'єднання, або на прив'язаний об'єкт буде діяти достатньо потужна сила, що також розірве з'єднання.

Події переміщення відрізняються лише вектором переміщення об'єкта. Вони, як зазначено в назві, переміщують платформу-актора на певну відстань разом із прив'язаним об'єктом по горизонталі: вліво, вправо, вперед або назад. Актор не буде переміщатися, якщо щось перешкоджатиме його руху.

2.6.4 Damage floor: актор що завдає шкоди

Мета даного актора - змусити гравців використовувати ігрові об'єкти-платформи та не злазити з них.

У завданнях, де потрібно перемістити персонажа або підконтрольний об'єкт з точки А до точки Б за допомогою платформ, часто можуть виникати ситуації, коли об'єкт або персонаж впали з цих платформ. У такому випадку ці об'єкти мають повернутися на своє початкове місце або “відродитися” у зазначених точках рівня..

Актор DamageFloor буде складатись з двох компонентів, статичної сітки, яка відображатиме даний об’єкт та прямокутної колізії.

При потраплянні об'єкта до цієї колізії буде викликана функція Blueprint інтерфейсу Destroy, яка викличе функцію знищення будь-якого об'єкта, що потрапить до нього.

Механіка знищення персонажа гравця відбуватиметься таким чином: якщо функція знищення персонажа викликалась під час того, як гравець керує

іншим ігровим об'єктом, то його контролер повертається. Після цього деактивується невидима ігрова колізія персонажа, вимикається контроль персонажа, і після 3-секундного очікування надсилається запит за допомогою інтерфейсу на відродження персонажа до класу режиму гри (GameMode). З GameMode надсилається подія до Blueprint рівня, у якому рівень звіряє, чи є даний персонаж або об'єкт у масиві структури, що зберігає посилання на об'єкт та точку відродження. Після цього створюється персонаж на рівні у відповідній точці, що бере відповідні змінні знищеного персонажа. І у випадку з персонажем гравця, його контролер переноситься до нового створеного персонажа. Після усіх цих дій попередній персонаж, що був знищений, видаляється.

Механіка знищення куба буде досить схожою: якщо гравець контролює куб, то куб повертає контролер гравця до його персонажа, решта алгоритму подібна до алгоритму знищення персонажа.

2.7 UI-елементи

2.7.1 Внутрішньоігрове меню

У кожній повноцінній відеогрі має бути внутрішньоігрове меню, яке надає гравцеві доступ до різних налаштувань під час гри, можливість вийти з гри або повернутися до головного меню.

Зазвичай елементи користувацького інтерфейсу (UI) у рушії Unreal Engine створюються у класі Hud, який створюється класом GameMode, тоді як події для створення різних віджетів викликаються з класу PlayerController [39]. Це дозволяє викликати UI елементи з будь-якого персонажа або ігрового об'єкта, а також не перевантажує клас контролера гравця.

Було створено віджет внутрішньоігрового меню, який містить чотири кнопки: "Продовжити гру", "Налаштування", "Вийти до меню" та "Вийти з гри". Ці кнопки виконують прості функції, зазначені їхніми назвами:

- Продовжити гру – приховує даний віджет, приховує курсор та повертає керування персонажем.
- Налаштування – відкриває вікно налаштувань.
- Вийти до меню – знищує поточну ігрову сесію та повертає до рівня головного меню.
- Вийти з гри – вимикає відеоігровий додаток.

Далі була створена подія DrawMenu у класі Hud, у якій створюється та відображається віджет внутрішньоігрового меню, з'являється курсор, вимикається керування над ігровим персонажем та активується режим введення для UI елементів. Якщо даний віджет вже створений і промальований, то він використовує готову функцію кнопки "Продовжити гру".

Ця подія викликається через клас PlayerController при натисканні клавіші Escape.

Таким чином, у користувача з'явилася можливість відкривати та закривати внутрішньоігрове меню за допомогою кнопок віджета та клавіш на клавіатурі.

2.7.2 Меню налаштувань графіки

У кожній повноцінній відеогрі також має бути меню налаштування графіки, оскільки у кожного користувача персональний комп'ютер має різну потужність, і у кожного можуть бути різні налаштування параметрів якості графіки та продуктивності.

Віджет налаштувань графіки буде складатися з таких елементів рис 2.5:

- Основні кнопки: “Повернутися назад”, “Застосувати налаштування”, “Налаштування за замовчуванням”.
- Стовпець віджетів-слайдерів, які дозволять налаштовувати параметри графіки від 0 до 4.

- Віджет-комбобокс (combobox), що представляє налаштування розширення екрану.
- Ряд кнопок для вибору режиму вікна, що допоможе змінювати режим вікна додатку.
- Чекбокс “Показати FPS”, який у разі істинного значення викликає віджет показу FPS.

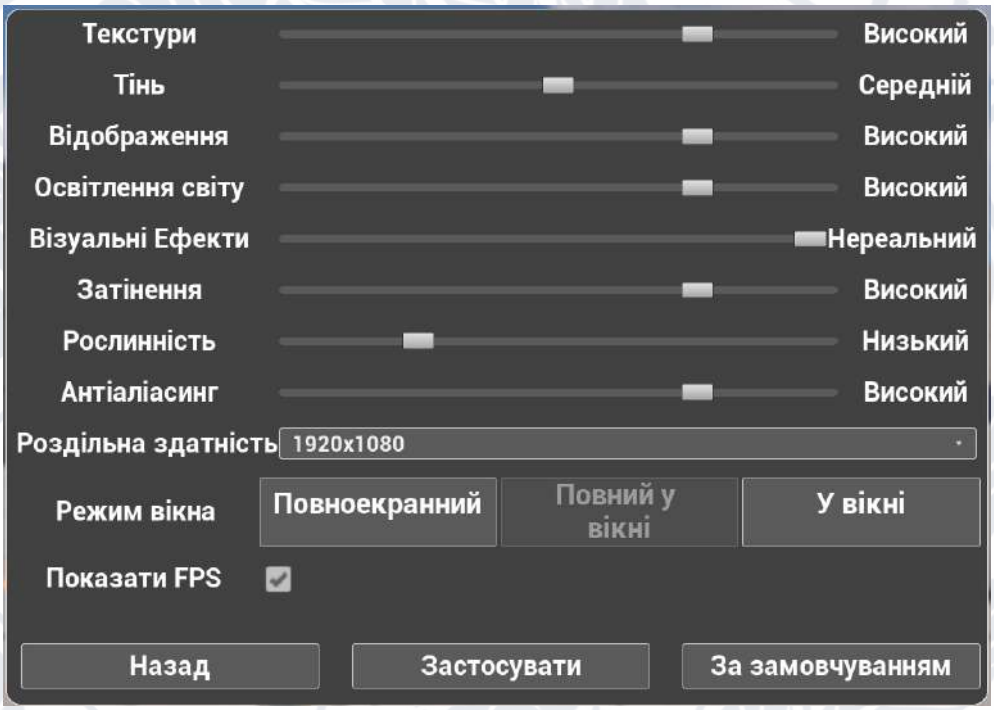

Рисунок 2.5 – Інтерфейс віджету меню налаштувань

Розглянемо детальніше віджети-слайдери:

Кожен такий віджет має 3 ключові елементи: назву опції, слайдер та текстове поле значення опції.

Слайдер налаштований на переміщення повзунка від значення 0.0 до 4.0 з кроком 1, що відповідає певному значенню опції. При кожному переміщенні цього повзунка буде змінюватися текстове поле значення опції, відображаючи, наприклад, “високий”, “низький”, “середній”. При зміні положення повзунка слайдера буде викликатись диспетчер подій, пов'язаний з віджетом налаштувань. Також даний віджет при створенні буде приймати назву опції та записувати її у свій текстовий компонент.

Віджет ComboBox буде налаштований подібним чином, за винятком того, що замість слайдера використовується випадаюче меню (ComboBox), і немає окремого компонента для текстового значення опції.

Ряд кнопок налаштування режиму вікна, та чекбокс показу “FPS” не мають окремих класів віджетів і створені лише у класі меню налаштувань.

Перейдемо до логіки Blueprint графу меню налаштувань.

При створенні цього віджета викликається нода GetGameUserSettings, що надає доступ до більшості параметрів налаштувань графіки. У даному випадку меню налаштувань отримує та записує у віджети-слайдери значення таких налаштувань, як: текстури, тіні, відображення, освітлення світу, візуальні ефекти, затінення, рослинність та антіаліасинг.

Після цього відбувається алгоритм запису налаштувань режиму вікна та роздільної здатності. Для режиму вікна створена спеціальна функція, яка залежно від налаштувань вимикає та вмикає кнопки. Для випадаючого меню роздільної здатності створена функція, що конвертує розширення екрану у рядок (string). За допомогою цієї функції та ноди GetSupportedFullScreenResolutions комбобокс автоматично заповнюється доступними значеннями розширення екрану.

Після кожної зміни значення слайдерів, комбобоксу чи при натисканні кнопок зміни режиму вікна – викликаються відповідні події, що записують відповідні значення налаштувань у GameUserSettings, але не змінюють їх, доки не буде натиснута кнопка “Застосувати”.

З налаштуванням показу FPS ситуація трохи складніша, оскільки в Blueprint немає вбудованого налаштування для цього. Тому для вирішення цієї проблеми необхідно створити два класи: SaveGame та GameInstance.

Клас SaveGame дозволяє зберігати дані. У цьому класі потрібно додати логічну змінну ShowFPS, яка представляє налаштування показу FPS.

Клас GameInstance відповідає за управління глобальними даними. Він зберігає дані від запуску додатку до його вимкнення. Таким чином, навіть при зміні рівнів чи під'єднанні до інших сесій, дані залишаються незмінними. У

цьому класі в графі подій потрібно створити об'єкт класу SaveGame і зберегти його як змінну.

Після цього у графі подій меню налаштувань отримується з ноди GetGameInstance об'єкт створеного класу GameInstance. З нього завантажується SaveGame, а з нього береться значення налаштування ShowFPS і записується це значення у чекбокс (Checkbox) цього віджета. Так само, як і з іншими віджетами, при зміні параметрів чекбокса перезаписуються параметри у SaveGame з GameInstance.

Далі у класі Hud, в залежності від значення змінної ShowFPS, відображається або не відображається додатковий віджет, який обчислює кількість FPS.

2.7.3 Відображення імен гравців

Дана гра розрахована переважно на проходження двома гравцями, але у майбутньому можуть з'явитися рівні, які вимагатимуть проходження трьома або більшою кількістю гравців. Оскільки всі персонажі гравців мають однакові моделі, знайти і дізнатися, хто є хто, буде доволі важкою задачею. Тому для вирішення цієї проблеми можна додати функції отримання та відображення імен гравців із сервісу Steam.

Алгоритм отримання імені гравця буде реалізований у PlayerState, оскільки він створюється разом із новим гравцем і слугує менеджером зберігання його даних у сесії. У графі подій Blueprint створимо подію, яка викликається при створенні об'єкта PlayerState. У цій події викликається алгоритм для локального користувача, в якому береться його ім'я за допомогою ноди GetPlayerName з плагіна AdvancedSessions, для цього достатньо вказати контролер гравця. Після цього це ім'я зберігається на серверній частині алгоритму та реплікується.

Таким чином реалізовано отримання та зберігання імені гравця у PlayerState, доступ до якого можна легко отримати з інших класів.

Відображення імені гравця буде реалізовано за допомогою віджет-компонента, який добавимо до всіх підконтрольних класів (таких як персонаж гравця, куб, платформа). Даний віджет складається з одного текстового поля та двох функцій, які дозволять змінювати значення цього поля (записати ім'я гравця), а також змінювати колір тексту в залежності від кольору гравця.

Даний віджет додамо як компонент для головного персонажа. На початку гри персонаж звернеться до PlayerState для отримання імені гравця, після чого запише це ім'я у щойно створений віджет, який зможуть побачити інші гравці. У події onRep_Color (подія, яка викликається при зміні кольору гравця) викликається подія, що змінює колір тексту у віджеті залежно від кольору гравця. Таким чином, гравці не тільки побачать ім'я гравця, але й будуть знати його колір.

Однак, на відміну від інших гравців, користувачу не дуже потрібно бачити своє ім'я, тому при ініціалізації об'єкта персонажа гравця віджет імені буде прихований для його власника.

Також існує проблема, що віджет дивиться в той самий бік, куди дивиться персонаж гравця, що не є дуже зручним рішенням. Тому на подію Tick (яка виконується кожен кадр) у кожного клієнта буде викликатись алгоритм, який буде переміщувати віджет імені таким чином, щоб він дивився прямо у камеру гравця. Цей алгоритм не потрібно реплікувати, оскільки у кожного клієнта віджети інших гравців мають бути повернуті індивідуально.

У даній грі гравець не завжди буде грати лише у ролі свого персонажа, тому слід додати цю функцію і до інших підконтрольних об'єктів, таких як куб та платформа. Тому дану функцію слід додати до батьківського класу підконтрольних об'єктів, що автоматично додасть її до куба та платформи.[33]

Алгоритм додавання досить схожий із додаванням імені гравця до персонажа - створюється віджет-компонент, який змінює свій колір залежно від кольору об'єкта та повертається до кожного гравця окремо. Але на відміну

від алгоритму для персонажа, даний об'єкт під час ініціалізації буде приховувати свій віджет від усіх, і під час захоплення об’єкта гравцем він бере ім'я з PlayerState, яке записується до віджету, та стає видимим для всіх гравців, крім власника.

2.7.4 Внутріншьоігровий чат

Однією з ключових складових у жанрі кооперативних головоломок є комунікація між гравцями, що надає більше можливостей для взаємодії та дозволяє легше проходити різні перешкоди. Тому внутрішньоігровий чат у даній грі є одним з основних елементів користувацького інтерфейсу.

Сам чат складатиметься з 4 віджетів та буде приймати та надсилати повідомлення за допомогою класу GameState (стану гри), який реплікується як на сервері, так і на всіх клієнтах.

Першим віджетом буде “повідомлення”. Воно складається з двох текстових частин, що позначають того, хто відправив повідомлення, та сам текст повідомлення. При створенні даний віджет запитуватиме 3 аргументи: текст повідомлення, відправника та числову змінну, яка буде змінювати колір повідомлення і допоможе відрізнити повідомлення від гравців від повідомлення самого сервера.

Другим віджетом є прокручуваний блок (scrollbox), який очікуватиме подію від GameState та у разі її виклику створюватиме віджет-повідомлення з вхідними параметрами, що містять текст, відправника та значення кольору.

Третій віджет міститиме widgetswitcher, який являє собою змінну віджетів і слугуватиме для переключення чату між активним станом (у якому буде писатись повідомлення) та пасивним станом (у якому буде виводитись останні повідомлення). Тому у даному віджеті буде два чати з прокручуваними блоками (Scrollbox), у одного з яких буде ще доданий textbox та кнопка для відправлення повідомлення.

Четвертий віджет містить у собі лише третій віджет і створений для того, щоб було легше редагувати положення та масштаб чату в інтерфейсі користувача.

Алгоритм роботи цих віджетів буде такий: при створенні, чат отримує від PlayerState дані про ім'я гравця, які записуються у змінну Sender (відправник). Після цього при активації чату та написанні й відправці повідомлення, надсилається подія з віджету у PlayerController, а звідти у GameState з аргументами того, хто відправив і яке повідомлення відправив. Ця подія викликає на кожному клієнті іншу подію за допомогою менеджера подій (Event Dispatcher) з тими самими аргументами, мета якої - створити новий віджет-повідомлення та помістити його у scrollbox. Таким чином, усі гравці побачать дане повідомлення.

Також у даному чаті можна зробити функцію надсилання повідомлення з сервера під час заходу та виходу гравців з гри. Це реалізовано наступним чином: після заходу в гру, коли гравець відправляє дані свого імені до PlayerState, викликається подія у класі режиму гри (GameMode), яка бере дане ім'я та відправляє повідомлення від свого імені у GameState про те, що даний гравець зайшов у гру, а той у свою чергу – створює дане повідомлення до всіх гравців. На відміну від відправки повідомлення від гравців, GameMode надсилає ще один додатковий аргумент, який змінює колір повідомлення сервера.

При виході гравця з гри у тому ж GameMode автоматично викликається інша подія, яка робить ту ж саму функцію, що і попередня функція – відправляє повідомлення клієнтам, але про те, що гравець вийшов з гри. У даній події він бере ім'я гравця з його PlayerState та змінює колір тексту на червоний.

Тепер залишилося створити логіку для створення віджету чату та зміни його стану у класі Hud. Ця логіка буде викликатись з класу PlayerController при натисканні клавіші "T". Також при зміні стану чату буде відображатись або

приховуватись курсор миші, а режим введення буде перемикатись між UI елементами та ігровим режимом.

Таким чином, реалізовано повноцінний внутрішньоігровий чат з можливістю відправляти/отримувати повідомлення від інших гравців та сервера рис 2.6.

Рисунок 2.6 – робота внутрішньоігрового чату

2.7.5 Інші UI елементи

У даній грі ще не було реалізовано декілька доволі важливих елементів користувацького інтерфейсу: головне меню, приціл та вивід поточного кольору гравця.

Головне меню - це UI, який має створюватися під час запуску рівня "Меню". У даному віджеті буде створено лише 3 кнопки: “Вихід з гри”, "Відкрити налаштування графіки" та “Відкрити список серверів”. Оскільки більша частина Blueprint коду готова, достатньо просто створити даний віджет у Blueprint графі рівня. На кнопку “Список серверів” потрібно створити та перейти до вже готового віджету списку серверів. На кнопку “Налаштування” потрібно зробити ті ж самі дії, що і зі списком серверів, але з віджетом

налаштувань графіки. А на кнопку “Вихід з гри” треба просто вимкнути відеоігровий застосунок.

Приціл - це дуже простий віджет, що складається з маленької картинки посередині екрану, на якій зображене коло. Він викликається у класі Hud під час його створення.

Вивід кольору гравця - це віджет, що складається з текстової комірки. У цьому віджеті присутня лише одна функція, яка приймає енумератор кольору і в залежності від його значення змінює текст і колір цього тексту.

Даний віджет створюється у класі Hud і бере колір під час початкового створення з класу PlayerState. Натомість, коли колір гравця зміниться у PlayerState, автоматично викличеться подія у контролері гравця, що надасть команду класу Hud, на зміну кольору цього віджета.

РОЗДІЛ 3
ФІНАЛЬНА СТАДІЯ РОЗРОБКИ ВІДЕОІГРОВОГО ЗАСТОСУНКУ

3.1 Геймдизайн рівня

3.1.1. Створення та налаштування рівня

Створено новий ігровий рівень, у якому буде реалізовано всі попередньо розроблені ігрові механіки.

На початку новий ігровий рівень не містить нічого: ні логіки, ні об'єктів, ні навіть освітлення. Почнемо з налаштування освітлення.

Для масштабного освітлення всієї сцени були додані такі компоненти сцени з незмінними параметрами за замовчуванням:

- DirectionalLight (Напрямне світло) - створює візуальний ефект освітлення від джерела світла, яке є великим та віддаленим, наприклад, як сонце.
- ExponentialHeightFog (Експоненційний туман висоти) - створює ефект туману, щільність якого зростає з висотою.
- PostProcessVolume (Об'єм післяобробки) - дозволяє налаштовувати різні параметри післяобробки, такі як колір, контраст, розмиття, освітлення тощо.
- SkyAtmosphere (Атмосфера неба) - створює візуальний ефект неба та атмосферних явищ, таких як розсіяне світло, колір неба.
- SkyLight (Небесне світло) - симулює вплив неба на освітлення сцени, включаючи кольорове освітлення від неба.
- VolumetricCloud (Об'ємні хмари) - створює об'ємні хмари.

Після налаштування освітлення рівня наступним важливим кроком буде додавання створеного режиму гри (GameMode) до рівня. Це, у свою чергу,

під'єднає персонажа гравця, його Hud, PlayerController, GameState та PlayerState.

3.1.2. Хаб гравців

Наступним кроком буде створення хаб-зони для гравців - невеликого блоку, у якому гравці, що зайшли на сервер, будуть очікувати інших гравців, перш ніж розпочати гру.

Цей блок буде створений з таких компонентів сцени:

- Статичні сітки - представляють собою стіни та підлогу блоку. Всі ці сітки будуть стилізовані у вигляді лабораторії, використовуючи вже створений матеріал, який безкоштовно надає рушій Unreal Engine.
- Текстові актори - актори сцени, які представляють собою текст у 3D просторі. Для них був створений та відредагований шрифт та матеріал для коректного виводу української мови. Вони створені з метою допомогти новачкам орієнтуватись у грі і додають підказки про те, що треба робити далі.
- Стартові точки гравця (PlayerStart) - цей актор представляє позицію, у якій створюватиметься персонаж гравця після заходу до гри.
- Натисні плити гравців - раніше створений клас, який активується при натисканні персонажем гравця на плиту. У даному блоці буде створено дві плити, одночасне натискання яких свідчитиме про готовність гравців починати гру.

За планом цього блоку, гравець, який є власником сервера, буде очікувати підключення іншого гравця-клієнта. Після того, як усі гравці увійдуть, їм надасться можливість вибрати, за який колір вони будуть грати - червоний або синій. Після цього гравцям потрібно буде стати на дві натисні

плити, що активують скрипт, який присвоїть їм потрібний колір та перенесе на наступний блок.

Цей алгоритм буде створений у Blueprint графі рівня. У даному алгоритмі рівень отримає посилання на натисні плити та зв'яжеться з подією натискання плит. Він перевірить, чи натиснуті обидві плити, і у разі успіху змінить кольори гравців, збереже дані про те, який гравець до якого класу (кольору) належить, а також перенесе їх до точкових акторів (PointActor), які є простими акторами без зовнішнього вигляду і будуть переміщені до іншого блоку, виступаючи в ролі точок появи персонажів.

Після написання логіки було додано декілька надписів, які слугуватимуть підказками для гравців рис 3.1.

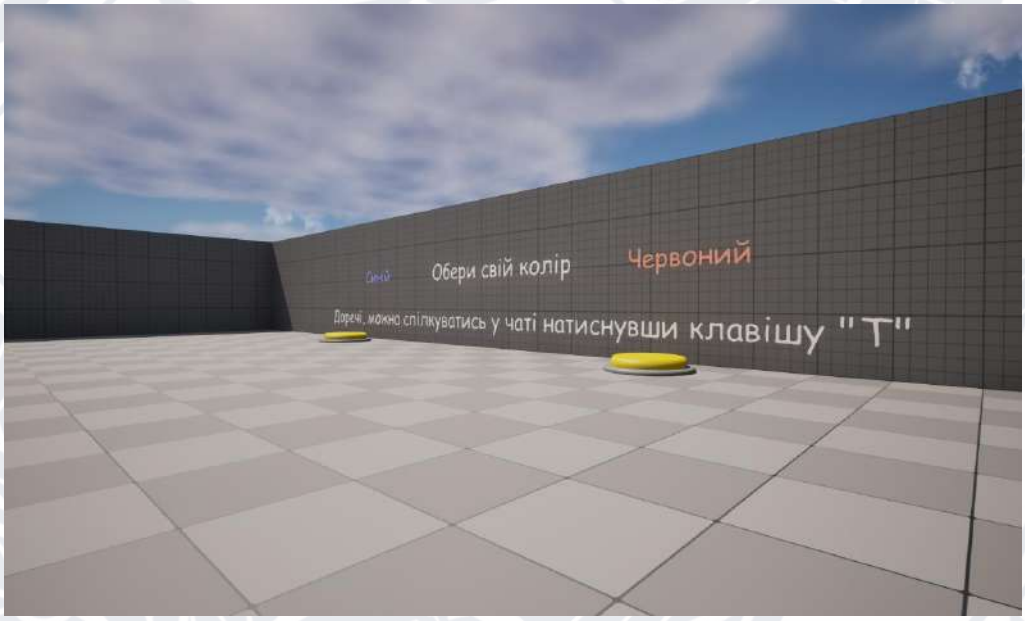

Рисунок 3.1 – зовнішній вигляд хабу гравців

3.1.3. Перший блок

Перші два ігрові блоки створені з метою навчити гравців правилам гри.

У першому блоці гравці повинні навчитися брати під контроль ігрові об'єкти, використовувати їх та повертатися до свого персонажа. Важливо, що об'єкти мають бути розташовані так, щоб гравці не могли пересувати їх за

допомогою колізії персонажа, змушуючи їх використовувати механіку отримання контролю над об'єктами.

Дизайн блоку такий: гравці повинні використати куб, розташований за межами досяжності персонажа, щоб піднятися на уступ і натиснути плиту, яка перенесе їх на наступну локацію. Також будуть додані навчальні текстові підказки.

Таким чином, даний блок складатиметься з точкових акторів, що переносять гравців на цей блок, стін та підлог, ігрового куба, що розташований над уступом та натисної плити рис 3.2.

Рисунок 3.2 – зовнішній вигляд першого блоку

Натисна плита активуватиме подію у Blueprint графі рівня, що телепортує гравців на інші точкові актори, розташовані у другому блоці.

3.1.4. Другий блок

У даному блоці гравці мають зрозуміти механіку кольорів, тому їм потрібно використати усі доступні типи кольорів.

Структура блоку буде схожа із структурою першого блоку, із деякими відмінностями. Уступ буде збільшений на 50%, переступити який можна буде за допомогою 3-ох кубів, розташованих сходинкою. Будуть 3 куба 3-ох кольорів, червоний, синій та білий, кожен з яких має бути використаний за для

подолання перешкоди. Для побудови сходів, знадобляться статичні об’єкти, що будуть представляти себе у ролі “пандусів” рис 3.3.

Рисунок 3.3 – зовнішній вигляд другого блоку

Дана натисна плита буде запускати складніший алгоритм ніж попередня. Окрім звичайного переміщення персонажів до наступної локації, рівень обробляє необхідні дані, що дозволять режиму гри (Gamemode), запустити функцію відновлення ігрових об’єктів та персонажів гравців, оскільки у наступному рівні буде використаний актор DamageFloor, що знищує об’єкти.

3.1.5. Третій блок

У третьому блоці будуть використані всі інші ігрові елементи: DamageFloor, платформа, платформа-актор, брама та натисні плити для кубів..

Цей блок починається з того, що гравці можуть подорожувати по станціях уздовж DamageFloor за допомогою підконтрольної платформи, рис 3.4.

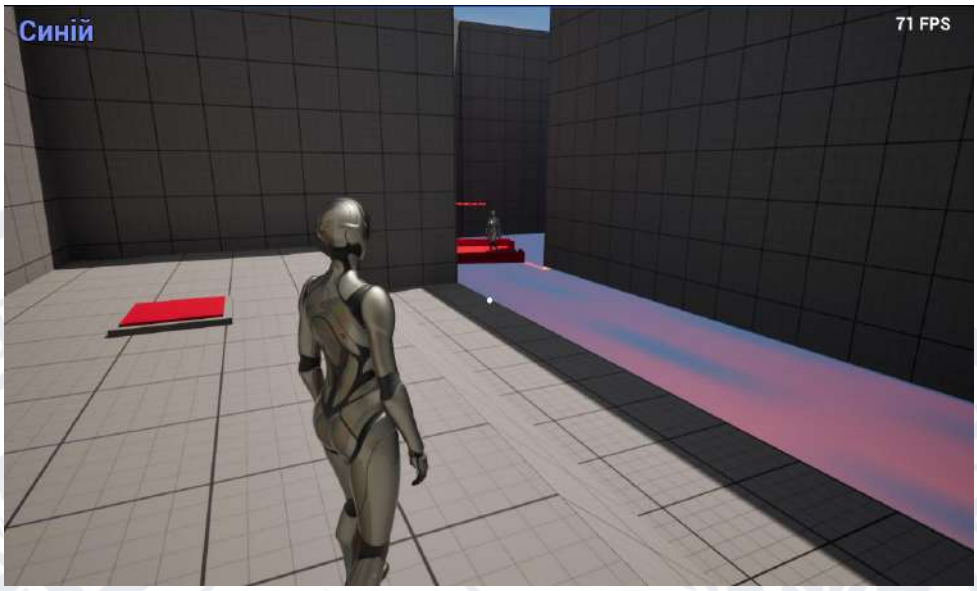

Рисунок 3.4 – Приклад подорожу за допомогою платформи

Всього створено 5 станцій, розглянемо кожну окрему:

Станція 1 – початкова. Тут з'являються персонажі гравців, яких зустрічає підконтрольна платформа. Ця станція – відправна точка гравців. У разі відродження після потрапляння у DamageFloor, платформа автоматично повернеться, запобігаючи застряганню гравців на рівні.

Станція 2 - тут є червона натисна плита куба рис. 3.4. Активація цієї плити нічого не зробить, якщо не активувати іншу плиту на станції 5. Якщо ці дві плити натиснуті – відкриються фінальні брами у кінцевій станції. На цій станції також захований синій куб, що потрібний для плити на 5-ій станції

Станція 3 – дана станція обсипана десятьма натисними плитами рис 3.5, кожна з яких виконує певну функцію:

- 1 плита відповідає за відкриття брами між 3-ою та 4-ою станцією, тому першому гравцеві потрібно залишитись на даній станції в той час я другий має продовжити путь і вигадати як забрати першого до 4 та 5 станції.
- 3 кнопки – відповідають за віддалене керування підконтрольною платформою, таким чином гравці можуть рухати платформу, вперед, назад, або взагалі перемістити її до першої станції не використовуючи механіку заволодіння об’єктом.

- 5 кнопок відповідають за віддалене керування платформою-актором. 4 з них відповідають за переміщення платформи по горизонталі, а 1 відповідає за прив’язку об’єкта на платформі, до самої платформи.
- Остання кнопка відтворює певний звуковий сигнал. Ця кнопка не надає жодної переваги гравцям, проте через відсутність позначень на кнопках, її можуть випадково натиснути. Такий випадковий натиск сповістить інших гравців, що гравець, який залишився на цій станції, ще не зрозумів призначення кожної кнопки.

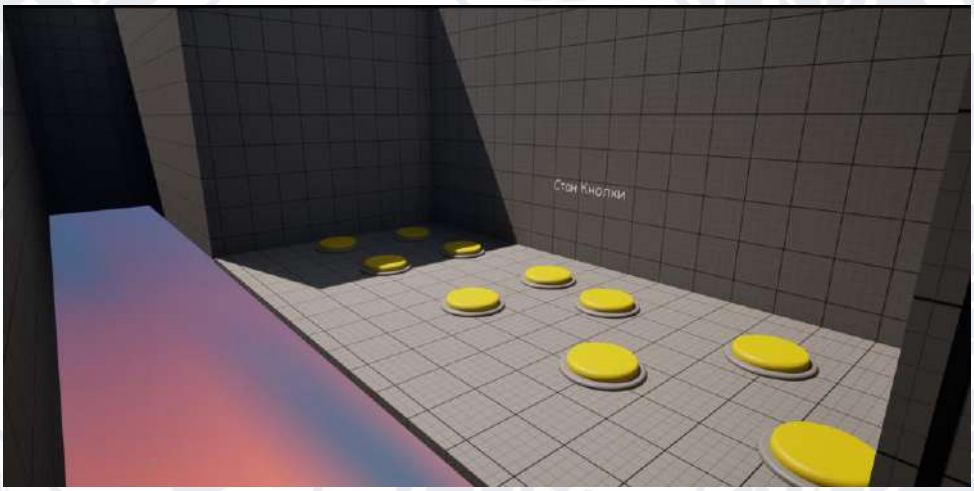

Рисунок 3.5 – Станція 3

Станція 4 – поділена на три сегменти. Перший сегмент має дві плити: перша відчиняє браму між 3 та 4 станціями, друга відповідає за віддалене взяття контролю над червоним кубом, що знаходиться у третьому сегменті. У другому сегменті є actor-платформа, якою віддалено керують плитами з 3 станції та невеликий лабіринт з напівпрозорих стін. Завдання на цій станції - доправити червоний куб через другий сегмент за допомогою actor-платформи та командної взаємодії.

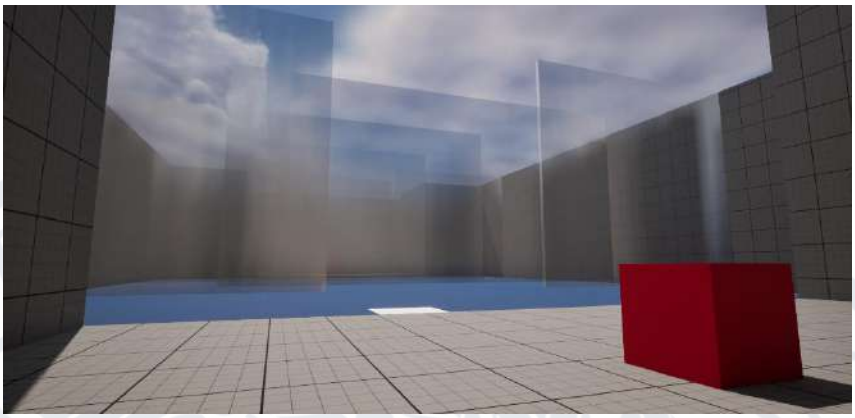

Рисунок 3.6 – Станція 4

Станція 5 слугує кінцевою станцією на рівні. Вона містить лише синю натисну плиту та браму, що відчиняється за допомогою натискання червоної плити 2-ої станції та синьої на 5-ій рис.3.7.

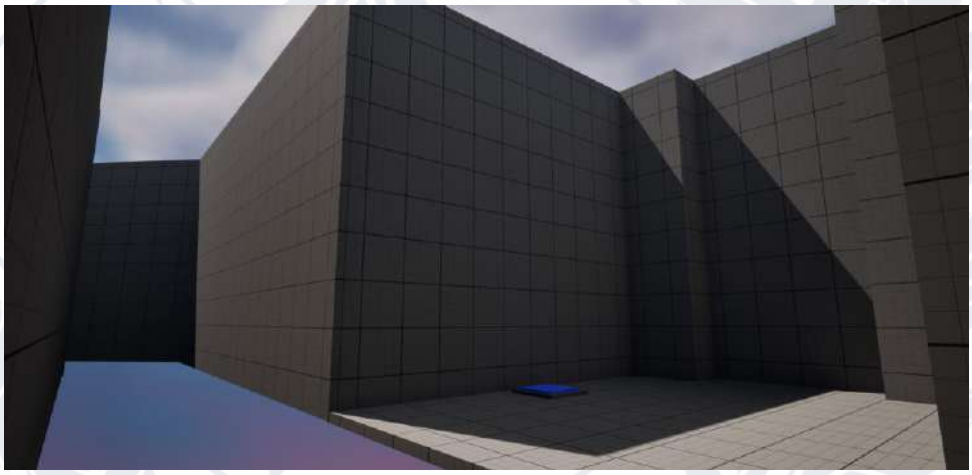

Рисунок 3.7 – Станція 5

Отже, головне завдання для гравців на цьому блоці - відкрити фінальні брами, для чого потрібно розгадати невелику головоломку та діяти узгоджено, командою.

3.2. Збірка проекту

В головному редакторі Unreal Engine 5.3 є спеціальна вкладка Platforms, де розробники можуть зібрати свій проект для різних платформ: Android, iOS,

tvOS, Linux, LinuxARM64 та Windows. Остання платформа - Windows - є тією, для якої було створено цю гру..

Після вибору платформи, рушій надає три варіанти збірки: DebugGame, Development та Shipping. Перші два містять інструменти для відладки та тестування, тоді як Shipping призначений для створення фінальної збірки для кінцевих користувачів.

Коли розробник натискає "Package Project", рушій розпочинає процес збірки проекту. Якщо збірка пройшла успішно без помилок, фінальний збірник гри буде створено у попередньо вказаній папці. [10]

Однак, для гри, що використовує сесії Steam, для забезпечення коректної роботи з підключенням сервісів Steam, необхідно додатково створити текстовий файл steam_appid.txt у папці Binaries\Win64. У цьому файлі потрібно вказати ідентифікатор проекту Steam. Для цієї гри Spacewar використовується ідентифікатор 480, спеціально створений для розробників.

3.3. Тестування

Тестування цієї гри здійснювалось безпосередньо в процесі розробки в редакторі Unreal Engine, що дозволяє навіть одному розробнику перевіряти розроблені онлайн механіки. Усі тести проводились поетапно з метою виявлення технічних недоліків, наприклад, незвичної поведінки об'єктів або некоректно налаштованої реплікації. Виявлені помилки виправлялися на ходу під час розробки. Тому на цьому етапі суттєвих технічних проблем не спостерігалося.

Однак, було проведено дослідження залежності кількості кадрів за секунду від продуктивності комп'ютера та рівня налаштувань графіки. Це дозволило краще зрозуміти системні вимоги гри до комп'ютера див. табл. 3.1.

Таблиця 3.1 Залежність кількості кадрів за секунду від специфікації комп’ютера та рівня графіки

Специфікація комп’ютера	Рівень графіки				
	Дуже низький	Низький	Середній	Високий	Ультра
Intel i5-12450H ОЗП 16 GB RTX 3060 Mobile	373 к/с	244 к/с	162 к/с	126 к/с	57 к/с
Intel i3-12100F ОЗП 32 GB RX 6600 XT	293 к/с	229 к/с	165 к/с	113 к/с	55 к/с
Intel i5-12400F ОЗП 16 GB RTX 3080 ti	470 к/с	340 к/с	230 к/с	187 к/с	80 к/с
Ryzen 5 4500U ОЗП 8GB	36 к/с	26 к/с	19 к/с	11 к/с	8 к/с

У таблиці наведено результати експериментів у кадрах за секунду, що проводилися за однакових умов: на одній і тій самій локації з роздільною здатністю екрану 1920x1080 пікселів.

ВИСНОВКИ

У даній бакалаврській роботі був проаналізований весь процес розробки відеоігрового додатка. Під час дослідження було вивчено відеоігровий ринок, на якому виявилася зростаюча популярність кооперативних ігор, розроблених незалежними розробниками. Також проводився аналіз можливостей технології візуального програмування Blueprint, яка є частиною ігрового движка Unreal Engine. На основі цих даних був розроблений план створення відеоігри, включаючи її концепцію, жанр і вимоги до кінцевого продукту.

Використовуючи технологію візуального програмування Blueprint, було створено повноцінний відеоігровий додаток, що відповідає всім поставленим вимогам, і дозволяє гравцям грати онлайн за допомогою сервісу Steam.

Отже, на основі результатів цієї роботи можна зробити висновок, що технологія візуального програмування Blueprint є потужним і доступним інструментом, який дозволяє розробникам ефективно та швидко створювати відеогрові застосунки для великого ринку розваг.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Unreal Engine 5.3 documentation. Online Subsystem Steam. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/online-subsystem-steam-interface-in-unreal-engine?application_version=5.3 (дата звернення: 19.01.2024).
2. Marco Secchi. Multiplayer Game Development with Unreal Engine 5: Create compelling multiplayer games with C++, Blueprints, and Unreal Engine's networking features, Packt Publishing, 2023, 394c.
3. Advanced Sessions Plugin. URL: https://forums.unrealengine.com/t/advanced-sessions-plugin/300204. (дата звернення: 19.01.2024).
4. Ігор Пипилів. Третина населення планети грає у відеоігри. Коли ринок геймінгу захопить світ?. URL: https://www.epravda.com.ua/publications/2023/05/18/700238/ (дата звернення: 12.03.2024).
5. How Many Gamers Are There? (New 2024 Statistics). https://explodingtopics.com/blog/number-of-gamers#number-of-gamers (дата звернення: 12.03.2024).
6. 70+ Video Game Statistics You Need to Know in 2024: Market Growth, Emerging Trends, and More. URL: https://www.techopedia.com/video-game-statistics (дата звернення: 12.03.2024).
7. Unity Visual Scripting. URL: https://unity.com/features/unity-visual-scripting (дата звернення: 06.11.2023).
8. Офіційний сайт RPG Maker. URL: https://www.rpgmakerweb.com (дата звернення: 24.03.2024).
9. Game Development Patterns with Unreal Engine 5
10. Satheesh Pv, Unreal Engine 4 Game Development Essentials: Master the Basics of Unreal Engine 4 to Build Stunning Video Games Paperback, Packt Publishing, 2016, 241c.

11. Lucas Bertolini. Hands-On Game Development Without Coding : Create 2D and 3D Games with Visual Scripting in Unity, Packt Publishing Ltd, Birmingham, 2018, 419 с.
12. 10 найкращих ігрових рушіїв. URL: https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiiv (дата звернення: 24.03.2024)
13. Sargey Rose. Unreal Engine 5 for Beginners: Dive into the world of game development with Unreal Engine 5 to build amazing 3D games, Packt Publishing, 2024, 440 с.
14. Jon Manning, Tim Nugent, Unity Game Development Cookbook: Essentials for Every Game Paperback, O'Reilly Media, 2019, 391 с.
15. Jiadong Chen. Game Development with Unity for .NET Developers: The ultimate guide to creating games with Unity and Microsoft Game Stack, Packt Publishing, 2022, 584 с.
16. Joseph Hocking. Unity in Action: Multiplatform Game Development in C#, Manning Publications, 2018, 370 с.
17. Kassandra Arevalo, Matthew Tovar, Jingtian Li. Creating Games with Unreal Engine, Substance Painter, & Maya, CRC Press, 2020, 826 с.
18. Peter L Newton, Jie Feng. Unreal Engine 4 AI Programming Essentials, Packt Publishing, 2016, 188 с.
19. Macros Romeo, Brenden Sewell. Blueprints Visual Scripting for Unreal Engine: The faster way to build games using UE4 Blueprints, Second Edition
20. Marcos Romero, Brenden Sewell, Luis Cataldi. Blueprints Visual Scripting for Unreal Engine 5: Unleash the true power of Blueprints to create impressive games and applications in UE5, Packt Publishing, 2022, 568 с.
21. Joanna Lee, Learning Unreal Engine Game Development, Packt Publishing, 2015, 249 с.

22. Brenden Sewell, Blueprints Visual Scripting for Unreal Engine: Build professional 3D games with Unreal Engine 4's Visual Scripting system, 2015, 190 с.
23. Nicola Valcasara, Unreal Engine Game Development Blueprints: Discover all the secrets of Unreal Engine and create seven fully functional games with the help of step-by-step instructions, 2015, 535 с.
24. Офіційний сайт Unro3D. URL: https://urho3d.io/ (дата звернення: 24.03.2024)
25. Офіційний сайт libGDX. URL: https://libgdx.com/ (дата звернення: 24.03.2024)
26. Офіційний сайт CryEngine. URL: https://www.cryengine.com/ (дата звернення: 24.03.2024)
27. Офіційний сайт GameMaker. URL: https://gamemaker.io/en (дата звернення: 24.03.2024)
28. Офіційний сайт AppGameKit. URL: https://www.appgamekit.com/ (дата звернення: 24.03.2024)
29. Офіційний сайт Godot. URL: https://godotengine.org/ (дата звернення: 24.03.2024)
30. Офіційний сайт Amazon Lumberyard. URL: https://aws.amazon.com/lumberyard/ (дата звернення: 24.03.2024)
31. Patrick Felicia. Unreal Engine from Zero to Proficiency (Foundations): A Step-by-step guide to your first game with Unreal Engine, 2022, 221 с.
32. Patrick Felicia. Unreal Engine from Zero to Proficiency (Beginner): A stet-by-step guide to coding your first game with unreal engine, 2023, 255 с.
33. Stuart Butler, Tom Oliver. Game Development Patterns with Unreal Engine 5: Build maintainable and scalable systems with C++ and Blueprint, Packt Publishing, 2024, 254 с.
34. Greg Keast. Learn Unreal Engine By Making A Basic Game, 2023, 118 с.

35. Mitchell Lynn, Cliff Sharif, Game Development with Unreal Engine 5: Learn the Basics of Game Development in Unreal Engine 5, BPB Publications, 2022, 336 с.
36. Bob Roy. First person shooter with procedural gererated level made in Unreal Engine: Procedural generation Multiplayer Computer game Unreal Engine 4, 2022, 74 с.
37. Ryan Shah. Master the Art of Unreal Engine 4 - Blueprints - Double Pack #1: Book #1 and Extra Credits: HUD, Blueprint Basics, Variables, Unreal Motion Graphics and more! 2014, 170 с.
38. Rachel Cordone. Unreal Engine 4 Game Development Quick Start Guide: Programming professional 3D games with Unreal Engine 4, Packt Publishing, 2019, 204 с.
39. David Nixon. Beginning Unreal Game Development: Foundation for Simple to Complex Games Using Unreal Engine 4, Apress, 2020, 418 с.
40. Aram Cookson, Ryan DowlingSoka, Tim Johnson, Clinton Crumpler. Unreal Engine 4 Game Development in 24 Hours, Sams Teach Yourself, 2016, 496 с.
41. Television industry statistics & facts. URL: https://www.statista.com/topics/4999/television-industry-worldwide/#topicOverview (дата звернення: 13.05.2024)
42. Video game market revenue worldwide from 2019 to 2029. URL: https://www.statista.com/statistics/1344668/revenue-video-game-worldwide/ (дата звернення: 13.05.2024)

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____
_____
____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)