Кафедра СПЕЦІАЛІЗОВАНІ КОМП’ЮТЕРНІ СИСТЕМИ

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

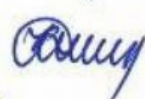
**до бакалаврської кваліфікаційної роботи на тему:**

Система керування світлодіодним освітленням /

Control system of LED lighting **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Студент групи КІ-48, Лещенко Данило Ігорович

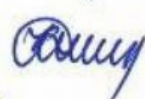
(шифр, П. І. Б. )



**Керівник роботи** \_\_\_\_\_\_\_\_\_\_ ( Клим Г.І. )



**Консультанти** \_\_\_\_\_\_\_\_\_\_ ( Войцеховська Ю.В. )



**Нормоконтроль** \_\_\_\_\_\_\_\_\_ ( Клим Г.І. )

****

**Попередній захист \_\_\_\_\_\_\_\_** ( Кочан Р.В. )

**Завідувач кафедри СКС** **Кочан Р.В.**

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 р.**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Національний університет «Львівська політехніка»** | | | | | | | | |
| (назва вищого навчального закладу) | | | | | | | | |
| Інститут **ІКТА**  Кафедра **СКС** | | | | | | | | |
| Спеціальність **Комп’ютерна інженерія**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | |
|  | | | «ЗАТВЕРДЖУЮ»  Завідувач кафедри СКС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кочан Р.В.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 р. | | | | | |
| **ЗАВДАННЯ** | | | | | | | | |
| **на кваліфікаційну роботу (проект) студента групи** КІ-48 **ОКР 6.050102** | | | | | | | | |
| Лещенка Данила Ігоровича  **аа**  (прізвище, ім’я, по батькові) | | | | | | | | |
| 1. Тема роботи Cистема керування світлодіодним освітленням / Control system of LED lighting | | | | | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | |
| затверджена наказом по університету від «30» \_\_\_\_\_травня\_\_\_\_\_ 2022 р. № \_\_\_\_1399-4-08\_\_\_\_ | | | | | | | | |
| 2. Термін подання студентом закінченої роботи \_\_17 червня 2022 р.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | |
| 3. Вихідні дані до роботи \_\_ Розробити систему керування світлодіодним освітленням. Система повинна мати декілька режимів роботи, можливості налаштування кольорів, яскравості, температури світіння як автаматично, так і вручну. | | | | | | | | |
| 4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити) | | | | | | | | |
| Вступ – доцільність теми та мета роботи; Аналітичний огляд існуючих практичних реалізацій; Обґрунтування обраних методів і засобів виконання проекту; Структура системи, що проектується; Реалізація проектних рішень; Розрахунок економічної ефективності проектних рішень. | | | | | | | | |
| 5. Перелік графічного матеріалу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Аркуш 1: Апаратний модуль. Схема електрична функціональна. | | | | | | | | |
| Аркуш 2: Алгоритм роботи програмного модуля 1 (апаратний модуль). Граф-схема. | | | | | | | | |
| Аркуш 3: Алгоритм роботи програмного модуля 2 (сервер). Граф-схема. | | | | | | | | |
| Аркуш 4: Алгоритм роботи програмного модуля 3 (клієнт). Граф-схема. | | | | | | | | |
| 6. Перелік програмних продуктів, які належить використати в процесі розроблення проекту  Середовище Arduino IDE, середовище Processing IDE | | | | | | | | |
|  | | | | | | | |
| 7. Консультанти у роботі, із зазначенням розділів роботи кожного | | | | | | | |
| Розділ | | Консультант | | | Підпис, дата | | |
| Завдання видав | Завдання прийняв | |
| Розділ №4. Економічна частина | | Войцеховська Ю.В. | | | Sign_войц  20.05.22 | 13.06.22 | |
|  | |  | | |  |  | |
|  | |  | | |  |  | |
| 8. Дата, коли видано завдання: **09 травня 2022 року**  Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) | | | | | | | |
| Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) | | | | | | | |
| КАЛЕНДАРНИЙ ПЛАН | | | | | | | |
| № з/п | Назва етапів кваліфікаційної роботи | | | Термін виконання етапів роботи | | | Примітка |
| 1 | Пошук і опрацювання літературних джерел | | | 09.05.22 – 22.05.22 | | |  |
| 2 | Аналіз літературних джерел | | | 23.05.22 – 26.05.22 | | |  |
| 3 | Розробка структури системи | | | 27.05.22 – 30.05.22 | | |  |
| 4 | Вибір засобів реалізації проекту | | | 31.05.22 – 04.06.22 | | |  |
| 5 | Розробка елементів системи | | | 05.06.22 – 06.06.22 | | |  |
| 6 | Підготовка аналітичного розділу | | | 07.06.22 – 08.06.22 | | |  |
| 7 | Підготовка розділу обґрунтування проектних рішень | | | 09.06.22 –10.06.22 | | |  |
| 8 | Підготовка проектного розділу | | | 11.06.22 – 12.06.22 | | |  |
| 9 | Виконання розрахунків з економіки | | | 13.06.22 – 14.06.22 | | |  |
| 10 | Підготовка графічного матеріалу | | | 14.06.22 – 15.06.22 | | |  |
| 11 | Оформлення пояснювальної записки | | | 14.05.22 – 17.06.22 | | |  |
| Студент \_\_\_\_\_\_\_\_\_\_\_\_  (підпис) | | | | | | | |
| Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_  (підпис) | | | | | | | |
|  | | | | | | | |

**АНОТАЦІЯ**

Лещенко Д.І, Клим Г.І.(керівник). Система керування світлодіодним освітленням. Бакалаврська кваліфікаційна робота. - Національний університет “Львівська політехніка”, Львів, 2022.

Розширена анотація

Цей проект передбачає розробку системи керування світлодіодним освітленням, який використовує можливості плати Arduino та адресної світлодіодної стрічки WS2812B. У прототипі реалізовані: різні режими роботи у залежності від команди користувача, ручна та автоматична зміна параметрів яскравості та температури світіння, швидкість анімацій(не для всіх режимів), передавання інформації користувачу про поточні час, температуру та вологість навколо пристрою.

Дана система може використовуватися для налаштування атмосфери у приміщенні, додаткового або нічного освітлення, створення ефектів для світломузики.

Програмні частини реалізовані за допомогою мов програмування С++ та Java. Програма апаратної частини використовує алгоритм роботи, що дозволяють за дуже малі проміжки часу виконати декілька команд, через що одноядерна плата вважається здатною до паралельного виконання задач. Апаратна частина виконана за допомогою Arduino, до якої підключені компоненти стеження за навколишніми умовами (яскравість, температура, вологість) з використанням макетної плати.

Реалізований проект повинен реагувати на вказівки користувача щодо зміну режимів світіння, мати достатню кількість цих режимів, самостійно або під впливом команд змінювати яскравість та температуру світіння і швидкість анімацій, коректно відправляти та отримувати дані, мати зручний інтерфейс, у якому користувач зможе легко обирати та надсилати параметри.

Об’єкт дослідження – світлодіодні стрічки та пристрої керування ними.

Предмет дослідження – робота пристрою керування світлодіодною стрічкою.

Мета дослідження – реалізація системи, яка матиме можливість до широкої зміни кольорів, яскравості та режимів світіння, самостійно керувати температурою та яскравістю, обмінюватися даними з користувачем.

Результатом є розроблена система керування світлодіодним освітленням на базі плати Arduino, яка отримує дані від датчиків та користувача.

Ключові слова: система керування, адресна світлодіодна стрічка, освітлення, світлодіод.

Перелік використаних літературних джерел:

1. [Brock Craft](https://www.amazon.com/s/ref=dp_byline_sr_book_1?ie=UTF8&field-author=Brock+Craft&text=Brock+Craft&sort=relevancerank&search-alias=books), Arduino Projects For Dummies, 2013, 416 p.
2. Andrew Koenig, Accelerated C++: Practical Programming by Example (Addison-Wesley C++ In-Depth), 2000, 352 p.

**ABSTRACT**

Leshchenko D., Klim H. (supervisor). Control system of LED lighting. Bachelor's thesis. - Lviv Polytechnic National University, Lviv, 2022.

Extended annotation

This project involves the development of an LED lighting control system which uses the capabilities of the Arduino board and WS2812B address LED strip. The prototype implements: different operating modes depending on the user's commands, manual and automatic change of brightness and temperature parameters, animation speed (not for all modes), transmission of information to the user about the current time, temperature and humidity around the device.

This system can be used to adjust the atmosphere in the room, additional or night lighting, creating effects for light music.

Software parts are implemented with C ++ and Java programming languages using. The hardware program uses an algorithm that allows you to execute several commands in very short periods of time, so it seems that a single-core board is considered capable of parallel tasks. The hardware is made with Arduino, which is connected to the components of environmental monitoring (brightness, temperature, humidity) using a mock-up board.

The implemented project must respond to user instructions to change the glow modes, have a sufficient number of these modes, to change the brightness and temperature and speed of animations independently or under the influence of commands, send and receive data correctly, have a user-friendly interface where users can select and send parameters easily.

The object of research is LED strips and their control devices.

The subject of research - the operation of the control device of the LED strip.

The purpose of the research is to implement a system that will be able to change colors, brightness and glow modes widely, control the temperature and brightness independently, exchange data with the user.

The result is an Arduino-based control system of LED lighting that receives data from sensors and the user.

Keywords: control system, address LED strip, lighting, LED.

List of used literature sources:

1. [Brock Craft](https://www.amazon.com/s/ref=dp_byline_sr_book_1?ie=UTF8&field-author=Brock+Craft&text=Brock+Craft&sort=relevancerank&search-alias=books), Arduino Projects For Dummies, 2013, 416 p.
2. Andrew Koenig, Accelerated C++: Practical Programming by Example (Addison-Wesley C++ In-Depth), 2000, 352 p.

**ЗМІСТ**

[ВСТУП 9](#_Toc13018)

[РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД СИСТЕМ КЕРУВАННЯ СВІТЛОДІОДНИМ ОСВІТЛЕННЯМ ТА СВІТЛОДІОДНИХ СТРІЧОК 14](#_Toc20263)

[1.1. Аналітичний огляд існуючих пристроїв - аналогів 14](#_Toc12951)

[1.2. Аналіз технічного завдання, розбиття роботи на підпункти, розробка алгоритму функціювання пристрою та структурної схеми 18](#_Toc26867)

[РОЗДІЛ 2. ВИБІР НАПРЯМКУ ПРОЕКТУВАННЯ, ЕЛЕМЕНТНОЇ БАЗИ І ЗАСОБІВ РЕАЛІЗАЦІЇ 21](#_Toc18122)

[2.1. Вибір апаратних засобів реалізації 21](#_Toc433)

[2.2. Вибір програмних засобів реалізації 31](#_Toc3625)

[РОЗДІЛ 3. Проектування системи керування світлодіодний освітленням 34](#_Toc16875)

[3.1 Розробка електричної функційної схеми, підбір необхідних компонентів та перевірка працездатності 34](#_Toc10518)

[3.2 Створення апаратного модуля 39](#_Toc628)

[3.3 Створення програмного модуля 1 41](#_Toc18217)

[3.4 Створення програмного модуля 2 45](#_Toc20047)

[3.5 Створення програмного модуля 3 46](#_Toc8701)

[3.6 Результат роботи пристрою керування 49](#_Toc28868)

[РОЗДІЛ 4. Економічна частина 52](#_Toc22379)

[4.1. Розрахунок витрат на розробку програмного забезпечення 52](#_Toc22833)

[4.2 Визначення експлуатаційних витрат 56](#_Toc26267)

[4.3 Розрахунок ціни споживання проектного рішення 59](#_Toc10845)

[4.4 Визначення показників економічної ефективності 60](#_Toc28300)

[ВИСНОВКИ 62](#_Toc14669)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63](#_Toc524)

[ДОДАТОК А 65](#_Toc8183)

[ДОДАТОК Б 74](#_Toc17458)

[ДОДАТОК В 83](#_Toc27197)

**ВСТУП**

Ігровий движок — це її ядро, базове програмне забезпечення, на якому будуються всі інші компоненти гри. Програмний код, який можна використовувати для створення варіацій гри, доповнень до неї або навіть цілого нового ігрового світу.

Вперше визначення з'явилося в середині 90-х, коли почали з'являтися ігри, схожі на головний шутер того часу - Doom. Тоді ж у вільному доступі почали з'являтися ігрові движки, на основі яких як сторонні розробники, так і звичайні користувачі могли спробувати написати власні ігри. Відтоді ігрові движки ставали все більш і більш технічно складнішими, довшими та багатшими за програмним кодом. Але при цьому, як і на початку свого існування, вони містять жорстко фіксовані дані: ігрова логіка, фізика об'єктів, правила відтворення об’єктів, геймплей в цілому. Всі інші компоненти гри написані «поверх» движка, і їх дуже багато. Таким чином, навіть при використанні одного і того ж двигуна, отримані віртуальні світи абсолютно різні. У той же час, все ж є деякі обмеження. Наприклад, один і той же движок не можна використовувати для стратегії та дії, RPG і тактики. Зазвичай движок заточують під ігри одного або кількох споріднених жанрів.

Unreal Engine, також відомий як UE, — це інструмент розробки відеоігор від компанії Epic Games, що займається розробкою відеоігор і програмного забезпечення. За допомогою цього інструменту розробники мають можливість створювати симуляцію, редагувати відео чи аудіо та відтворювати анімацію. Розробники використовували його для створення деяких із найпопулярніших ігор на ринку сьогодні. Однією з головних переваг є те, що він безкоштовний і зручний для нових користувачів. Epic Games прагне просувати своє програмне забезпечення в усьому світі за допомогою правильних стратегічних союзників, щоб гарантувати, що більше людей зможуть наважитися на розробку відеоігор. Як компанія з понад 25-річним досвідом, Epic Games розробила програмне забезпечення, яке вважається стандартом AAA для розробки відеоігор як частини ринку, що розвивається найшвидше у світі. Написаний на C++, Unreal Engine дуже портативний, і користувачі мають можливість використовувати його на багатьох платформах, включаючи iOS, Android, Windows, PlayStation і Xbox.

Хоча існують попередні ітерації Unreal Engine, остання версія цього ігрового двигуна є важливою віхою, оскільки дозволяє розробникам створювати та додавати власні карти до своїх ігор.

**Актуальність теми** обґрунтовується тим, що якісні освітлення та система керування нею є важливими з точки зору налаштування робочої середи. Швидке та зручне управління якісним світлом допоможе створити комфортні умови для роботи.

**Метою цієї роботи** є розробка системи керування світлодіодним освітленням. Основною особливістю є можливість зручно керувати системою самостійно або налаштувати автоматичне функціювання згідно з умов навколишнього середовища.

**РОЗДІЛ 1. АНАЛІТИЧНИЙ**  **ОГЛЯД UNREAL ENGINE**

Даний розділ призначений для огляду Unreal Engine, засоби програмування в даному середовищі, виділення переваг та недоліків, розробки алгоритму роботи штучного інтелекту класу ворога, розробка ієрархії класів для класу керованого користувачем персонажа.

**1.1. Аналітичний огляд існуючих пристроїв - аналогів**

Заклятим “суперником” UE довгий час вважається Unity. Від Hellblade: Senua's Sacrifice та Sea of Thieves до Rainbow Six Siege, Fortnite та The Stanley Parable, Unreal та Unity забезпечують багато ігор AAA.

Обидва двигуни мають свої переваги, але також мають обмеження. Наприклад, Unity Engine ідеально підходить для початківців у розробці 3D-додатків, оскільки використовує просту для розуміння мову програмування C#. Unreal Engine, з іншого боку, є потужним професійним інструментом, який кидає виклик досвідченим 3D-розробникам завдяки використанню складної мови програмування C++. Однак він справді здатний з легкістю відображати оптимізоване 3D-середовище. Переваги та недоліки кожного двигуна наступні:

Unreal Engine – повністю безкоштовний для початківців. Користувачам достатньо завантажити движок із веб-сайту, запустити його, і вони готові до роботи. Через величезну кількість розробників, які також використовують цей інструмент, розробники мають багато онлайн-підручників і підтримки, доступних через онлайн-спільноту. Крім того, Unreal Engine неймовірно універсальний і дозволяє користувачам створювати дуже складну гру майже з нічого. Однак ця можливість є більшою проблемою для багатьох початківців через складність мови.

Unity також є одним із найважливіших і використовуваних двигунів. Що робить Unity таким унікальним, так це його велика онлайн-спільнота, як-от у Unreal, але платформа полегшує доступ до основної функціональності для початківців. Немає значення, чи користувач тільки починає роботу, чи він справді хоче створити інтенсивний і важкий проект: Unity легко зрозуміти та все ще має всі основні функції, необхідні користувачам.

Через нещодавні зміни в ціновій політиці Unity, UE, разом зі своїми перевагами став набагато привабливим вибором.

**1.2. Аналіз програмних можливостей Unreal Engine для реалізації проекту**

Unreal Engine використовує C++ як мову програмування для розробки ігор. C++ забезпечує загальну стабільність і чудовий розподіл пам’яті, тому розробники Unreal Engine використовують його. Проте це видозмінена С++, так як використовує розроблені Epic Games фреймоврки, контейнери, засоби реплікацїї, подання даних в едіторі тощо. Вона підлаштована для зручного користування з движком.

До особливостей програмування в UE є:

* У Epic є кілька простих стандартів кодування та угод. Умовні позначення коду покращують читабельність програмного забезпечення, дозволяючи інженерам швидко і досконало розуміти новий код. Це необхідно для сумісності між компіляторами. Стандарти кодування Epic орієнтовані на C++; однак очікується, що стандарт буде дотримуватись незалежно від мови, яка використовується.
* Система Unreal Engine Reflection інкапсулює ваші класи різними макросами, які забезпечують функціональність механізму та редактора.
* Під час програмування за допомогою Unreal Engine(UE) ви можете мати стандартні класи, функції та змінні C++. Під час програмування за допомогою Unreal Engine у вас можуть бути стандартні класи, функції та змінні C++. Їх можна визначити за допомогою стандартного синтаксису C++. Однак макроси UCLASS(), UFUNCTION() і UPROPERTY() можна використовувати, щоб повідомити Unreal Engine про нові класи, функції та змінні. Наприклад, змінна з оголошенням, перед яким стоїть макрос UPROPERTY(), може бути сміттям, яке збирає механізм і відображається та редагується в Unreal Editor. Існують також макроси UINTERFACE() і USTRUCT() і ключові слова для кожного макросу, які можна використовувати для визначення поведінки класу, функції, властивості, інтерфейсу або структури в Unreal Engine і Unreal Editor. На додаток до наведених вище макросів існує макрос UPARAM(), який в основному використовується під час надання коду C++ Blueprints. Приклади використання UPARAM() дивіться в документації Exposing Game Elements to Blueprints.
* Бібліотека Unreal Smart Pointer — це спеціальна реалізація інтелектуальних вказівників C++11, призначена для полегшення завдання розподілу пам’яті та відстеження. Ця реалізація включає спільні, слабкі та унікальні вказівники. Він також додає спільні посилання, що не допускають значення null. Ці класи не можна використовувати з системою UObject, оскільки Unreal Objects використовує окрему систему відстеження пам’яті, яка краще налаштована для коду гри.
* Unreal Engine має власні контейнери-аналоги stl-контейнерів:

1. TArray - відповідає за володіння та організацію послідовності інших об’єктів (так звані «елементи») того самого типу.
2. TMap і TMultiMap. Різниця між цими двома полягає в тому, що ключі TMap унікальні, тоді як TMultiMap підтримує зберігання кількох ідентичних ключів.
3. TSet схожий на TMap і TMultiMap, але з однією важливою відмінністю: замість того, щоб пов’язувати значення даних із незалежними ключами, TSet використовує саме значення даних як ключ за допомогою функції, що перекривається, яка обчислює елемент. TSet дуже швидко (з постійним часом) додає, знаходить і видаляє елементи.

* Архітектура ігрового процесу: кожен клас визначає шаблон для нового актора або об’єкта. У файлі заголовка класу оголошується клас і будь-які функції та властивості класу. Класи також можуть містити структури, структури даних, які допомагають організовувати та керувати пов’язаними властивостями. Структури також можна визначити окремо. Інтерфейси дозволяють реалізовувати додаткову ігрову поведінку різними класами.
* Делегати можуть викликати функції-члени об’єктів C++ у загальний, безпечний спосіб. Делегат може бути динамічно прив’язаний до функції-члена довільного об’єкта, викликаючи функцію об’єкта в майбутньому, навіть якщо абонент не знає типу об’єкта. Делегатів можна безпечно копіювати. Ви також можете передати їх за значенням, але це не рекомендується, оскільки цей процес виділить пам’ять у купі; коли це можливо, передавайте делегатів за посиланням. Механізм підтримує три типи делегатів:Singur, Multicast, Dinamic.

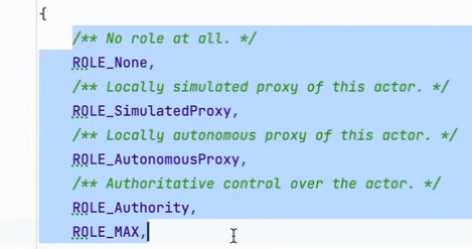
При необхідності створення онлайн гри UE має для цього вбудовані засоби реплікації. Це процес синхронізації даних і викликів процедур між клієнтами та серверами. Система реплікації забезпечує абстракцію вищого рівня разом із налаштуваннями низького рівня, щоб полегшити роботу з усіма різноманітними ситуаціями, з якими ви можете зіткнутися під час створення проекту, призначеного для кількох одночасних користувачів. Онлайн в UE побудований на базі архітектури “клієнт-сервер”. Кожен з учасників має свою роль при підключенні (Рисунок ). Авторитет виступає в

Рисунок 8. Перелік онлайн ролей

ролі сервера, решта в ролі клієнтів. Запити на виконання дій подаються від клієнта до сервера, і сервер вирішує і має можливість відображати ці дії усім користувачам, або , навпроти, відмінити їх. У UE є можливість створити різні типи серверів — звичайний сервер як окремий невізуальний персонаж, що містить інформацію про всіх інших, або клієнт, що виступає у ролі сервера, і навіть не знає про це. Перший підхід є безпечнішим і біль затратним. Другий дає можливість не витрачатись на створення власного великого сервера і віддати клієнту ресурсну роль для цього. Засоби реплікації дозволяють обмінюватися даними між сервером та клієнтом.

Nanite — це система віртуалізованої геометрії Unreal Engine 5, яка використовує новий формат внутрішньої сітки та технологію візуалізації для рендерингу піксельних деталей і великої кількості об’єктів. Він працює розумно лише над деталями, які можна сприйняти, і не більше. Формат даних Nanite також сильно стиснутий і підтримує дрібну потокову передачу з автоматичним рівнем деталізації.

Переваги Nanite:

На кілька порядків збільшується складність геометрії, збільшується кількість трикутників і об’єктів, ніж це було можливо раніше, у реальному часі;

Бюджети фреймів більше не обмежуються полірахунками, викликами малювання та використанням сітчастої пам’яті;

Використовуйте високополігональні деталі замість запікання деталей у звичайні текстури карти;

Рівень деталізації (LOD) обробляється автоматично і більше не потребує ручного налаштування окремих LOD сітки;

Втрата якості трапляється рідко або взагалі відсутня, особливо з переходами LOD.

Nanite максимально легко інтегрується в існуючі робочі процеси двигуна, одночасно використовуючи новий підхід до зберігання та відтворення сітчастих даних. Під час імпорту — сітки аналізуються та розбиваються на ієрархічні групи груп трикутників. Під час відтворення — кластери змінюються на льоту з різними рівнями деталізації залежно від кута зору камери та плавно з’єднуються без тріщин із сусідніми кластерами в одному об’єкті. Дані передаються за запитом, тому в пам’яті повинні залишатися лише видимі деталі. Nanite працює у власному проході візуалізації, який повністю обходить традиційні виклики малювання.

Lumen — це повністю динамічна система глобального освітлення та відображень Unreal Engine 5, яка розроблена для консолей наступного покоління та є системою глобального освітлення та відображень за замовчуванням. Lumen відтворює нескінченне дифузне взаємовідбиття та непрямі дзеркальні відбиття у великих деталізованих середовищах у масштабах від міліметрів до кілометрів. Основні принципи роботи та переваги системи "Люмен" в Unreal Engine:

Реалістична глобальна ілюмінація: Система "Люмен" дає змогу отримати більш реалістичну обстановку освітлення в іграх. Вона дозволяє промінцям світла поширюватися та відбиватися від поверхонь, створюючи природні відтінки та тіні в реальному часі.

Динамічне освітлення: За допомогою "Люмен" можна досягти динамічного освітлення, що означає, що світло може змінювати своє положення та інтенсивність під час гри. Це дозволяє створювати реалістичні ефекти освітлення, такі як сходи та заходи сонця.

Покращена якість зображення: Система "Люмен" допомагає покращити якість зображення в графічно вимогливих сценах. Вона робить відображення більш природним та деталізованим, що робить ігровий світ більш іммерсивним.

Швидкодія: Unreal Engine 5 створює ілюмінацію у реальному часі без значних затрат на обчислення. Це робить її ефективною для використання в іграх та додатках. Простота розробки: У порівнянні з попередніми методами вимірювання освітленості, система "Люмен" в Unreal Engine дозволяє розробникам створювати складні ефекти освітлення без значних зусиль.

Гарним рішенням для створення онлайн проекту є онлайн-підсистема. Вона забезпечує загальний спосіб доступу до функцій онлайн-сервісів. Під час роботи над грою, яка постачається на кількох платформах або підтримує кілька онлайн-сервісів, онлайн-підсистема гарантує, що єдині зміни, які розробники повинні внести, це коригування конфігурації для кожної підтримуваної служби. Оскільки швидкість мережевого з’єднання, затримки сервера та час виконання серверної служби невідомі локальній машині, взаємодія з цими системами займає непередбачуваний проміжок часу. Щоб упоратися з цим, онлайн-підсистема використовує делегати для всіх віддалених операцій і гарантує, що ці делегати будуть викликані щоразу, коли використовується підтримувана асинхронна функція. На додаток до надання можливості відповідати на запити в міру їх виконання та надсилати запити на льоту. Представники також забезпечують єдиний шлях до коду, який усуває розробникам потребу писати спеціальний код для визначення різних умов успіху чи невдачі. Модульні сервісні інтерфейси об’єднують підтримувані функції. Наприклад, інтерфейс друзів обробляє все, що стосується списків друзів, інтерфейс досягнень обробляє списки, перевірку та нагородження досягнень тощо. Існують інтерфейси для кожної групи функцій у кожній онлайн-службі, яка її підтримує, хоча певні функції, які служба не підтримує, можуть просто повертати false. Цей дизайн гарантує, що розробники можуть писати однаковий код для всіх онлайн-сервісів. На вищому рівні складніші операції використовують онлайн-диспетчер асинхронних завдань для підтримки послідовних завдань або завдань, що виконуються в різних потоках. Асинхронні завдання можуть описувати свої залежності, дозволяючи непов’язаним завданням виконуватися незалежно та паралельно, тоді як послідовні завдання виконуються послідовно. Усі інтерфейси в онлайн-підсистемі планують завдання таким чином, щоб підтримувати узгодженість роботи.

**1.3. Аналіз технічного завдання, розбиття роботи на підпункти, розробка алгоритму штучного інтелекту та ієрархії класів**

Згідно з аналізу, метою даної роботи . Дана робота описуватиме створення проекту, що складається з таких частин:

* Розробка рівня;
* Розробка локомаціних анімацій;
* Розробка анімацій реагування на перешкоди;
* Створення управління;
* Створення ієрархія класів;
* Розробка механік вистрілу снаряду;
* Створення поведінки ворогів, використовуючи штучний інтелект.

Ієрархія класів необхідна для організації коду, розмежування окремих компонент, які відповідають за однотипні операції, привального використання лише потрібних частин окрим екземпляром класу, наявним на сцені, використання віртуальних методівющо змінюватимуть поведінку виконання в залежності від поточного тип, який виконує операцію. Так як класи персонажу та ворога повинні містити компоненти колізій, меш-скелет для відображення анімацій, то базово обидва наслідуватимуться від наявного в двіжку класу персонажа. Проте обидва використовуватимуть систему відображення ефектів та динамічної зміни матеріалу, який міститиме клас NiagaraCharacter, визначення відстані до поверхні від кісток ніг меша-скелету, зміни швидкості пересування, який міститиме клас SHCharacter, отже наслідуватимуться вони обидва від класу NiagaraCharacter, який є наслідником класів персонажу та SHCharacter.

Клас персонажу має лінійну ієрархію, так як в двіжку неможливий випадок ромбовидного віртаульного наслідування, яке наявне в С++. Ієрархія поділена чином розподілу імплементацій відповідальних за свої модулі виконання:

1. EnhаncedInputChаrаcter — клас, що налаштовує ввід користувача та відповідні методи, що мають бути викликані, та віртуальні методи реагування на ввід, які буду реалізовані в нащадках.
2. ClimbingSystemChаrаcter — клас, що містить методи переміщення вертикальними поверхнями та переміщення через перешкоди.
3. WithCameraChаrаcter — клас, що містить налаштування камери.
4. ThrowChаrаcter — клас, що реалізує як атаку постріл снаряду, його повернення, прицілювання та переміщення до снаряду.
5. CrouchChаrаcter — клас, що реалізовує перехід між звичайним станом та станом присідання.
6. HideChаrаcter — клас, що реалізовує приховування персонажа за валідним об’єктом.
7. MotionWarpingChаrаcter — клас, що реалізовую ближню атаку.

Ієрархія наведена в .

**РОЗДІЛ 2. ВИБІР НАПРЯМКУ ПРОЕКТУВАННЯ, ЕЛЕМЕНТНОЇ БАЗИ І ЗАСОБІВ РЕАЛІЗАЦІЇ**

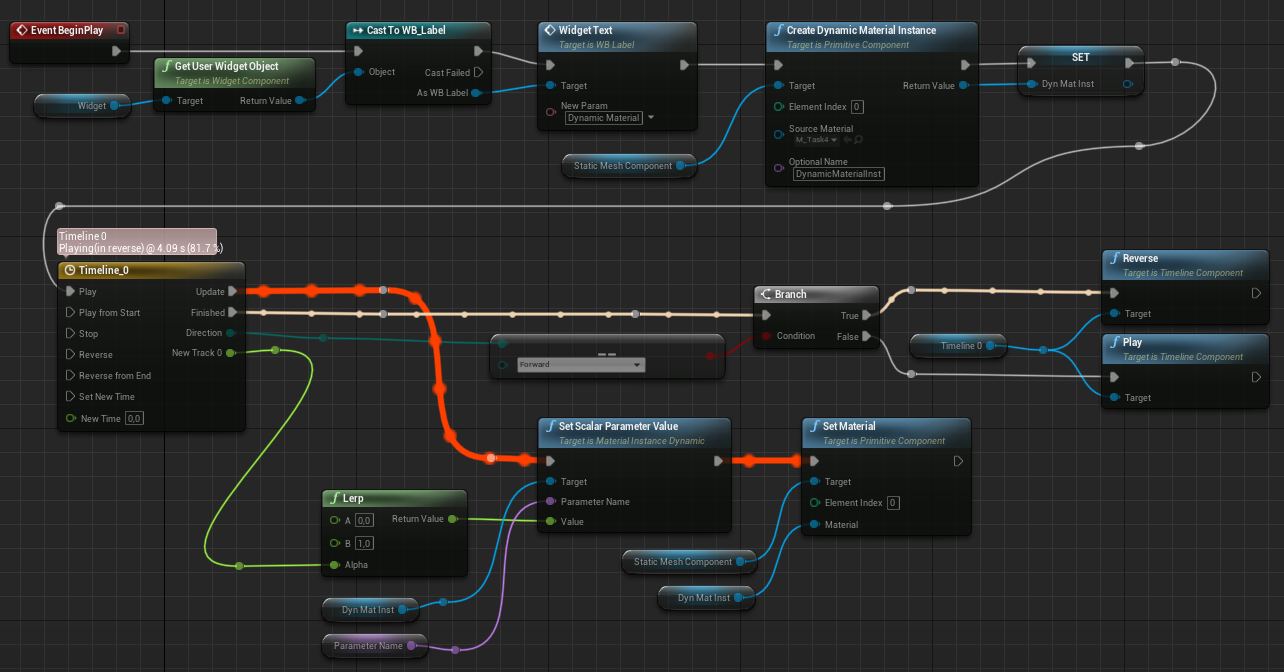
Даний розділ призначений для вибору оптимальних програмних засобів серед наявних для реалізації проекту, опису їх характеристик, пояснення програмних засобів, які використовуватимуться.

**2.1. Опис основних програмних засобів реалізації**

Існує два варіанта використання Unreal Engine в розробці: клонування репозиторію движка або використовувати готовий. Перший варіант дає змогу переписувати сам Unreal Engine в залежності від потреб, як мінус необхідно кожен раз будувати проект движка. Другий варіант дає доступ до стандартної імплементації UE, що є простішим і швидшим рішенням для розробки. Для роботи з UE також необхідно обрати IDE, уважно встановлюючи необхідну її версію під обрану збірку движка. Найпопулярніші середовища: Visual Studio, Xcode, Rider. Rider є найрзучнішим і найкращим рішенням, бо був створенний якраз для UE, Visual Studio є універсальним рішенням з наявними перевагами в можливостях дебагу проекту.

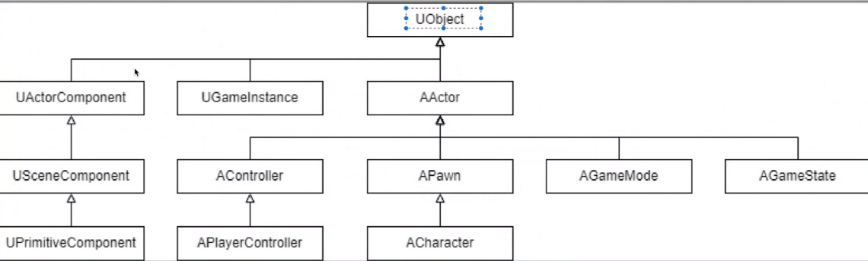
Кожен проект UE має три основні директорії. Config - містить файли конфігурацій (такі як стандартний левел для запуску, налаштування вводів користувача). Content - містить усі ассети проекту (текстури, меши, анімації, віджети тощо). Source - містить файли з написаним кодом.

Система візуальних сценаріїв Unreal Engine Blueprint (Рисунок 1) — це повна система ігрових сценаріїв, заснована на концепції використання інтерфейсу на основі вузлів для створення елементів гри з Unreal Editor. Як і в багатьох поширених мовах сценаріїв, він використовується для визначення об’єктно-орієнтованих класів або об’єктів у механізмі. Ця система є надзвичайно гнучкою та потужною, оскільки дає дизайнерам можливість використовувати практично весь спектр концепцій та інструментів, які зазвичай доступні лише програмістам. Крім того, спеціальна розмітка Blueprint, доступна в C++ реалізації Unreal Engine, дозволяє програмістам створювати основні системи, які можуть бути розширені дизайнерами.

Рисунок 1. Приклад використання Blueprint для створення динамічного матеріалу

Движок має власну ієрархію класів (Рисунок 2), яка створена для імплементації наявних можливостей UE та потреб розробника.

Рисунок 2. Діаграма наслідування основних класів



UObject — базовий клас, від якого наслідуються всі наявні класи, та який є видимий в для UE.

GameInstance - сінглтон гри, один в грі, ініціалізується на початку гри, видаляється в кінці, місце, де я може ініціалізувати глобальні параметри.

Actor — клас, який може бути розміщений на сцені, має компоненти положення (локація, поворот, розмір), має компоненту, що оновлюється кожного кадру.

GameMode — клас для поточного левела, необхідний для опису усіх ігрових “правил”: можливість приєднання до гри, переходи на різні секції, завантаження місій тощо. Важливі методи: InitGame (Викликається після конструктора, левел стартується звідси, можна задати її розмір, погоду, кількість гравців тощо); PreLogin (Перевірки гравця перед підключенням), PostLogin (Оновлення гри після додавання нового гравця).

Pawn — клас, наслідується від Actor та додатково має можливість виконувати логіку переміщення від гравця або ШІ.

Controller — це не фізичні актори, які можуть володіти Pawn (або похідним від нього), щоб контролювати його дії. PlayerController використовується гравцями для керування Pawn , а AIController реалізує штучний інтелект для керування.

Компоненти: ActorComponent(нефізичний, має логіку), SceneComponent - дочірній від попереднього, має ще і візуалізацію (колізії, Меши, камера), PrimitiveComponent - дочірній компонент від сцени (має графічне представлення).

Character — дочірній клас від Pawn, додатково має компоненту, що може відображати анімації (SkeletalMesh).

Необхідним інструментом взаємодії Акторів між собою в світі є колізії. Відповіді на зіткнення та відповіді відстеження складають основу того, як Unreal Engine обробляє зіткнення та трасування променів під час виконання. Кожному об’єкту, який може зіткнутися, надається тип об’єкта та набір відповідей, які визначають, як він взаємодіє з усіма іншими типами об’єктів. Коли відбувається зіткнення або накладання, обидва (або всі) задіяні об’єкти можуть бути налаштовані на вплив або на них впливають блокування, накладення або ігнорування один одного. Відповіді на трасування працюють так само, за винятком того, що сам промінь можна визначити як один із типів відповідей на трасування, що дозволяє акторам блокувати або ігнорувати його на основі їхніх відповідей на трасування.

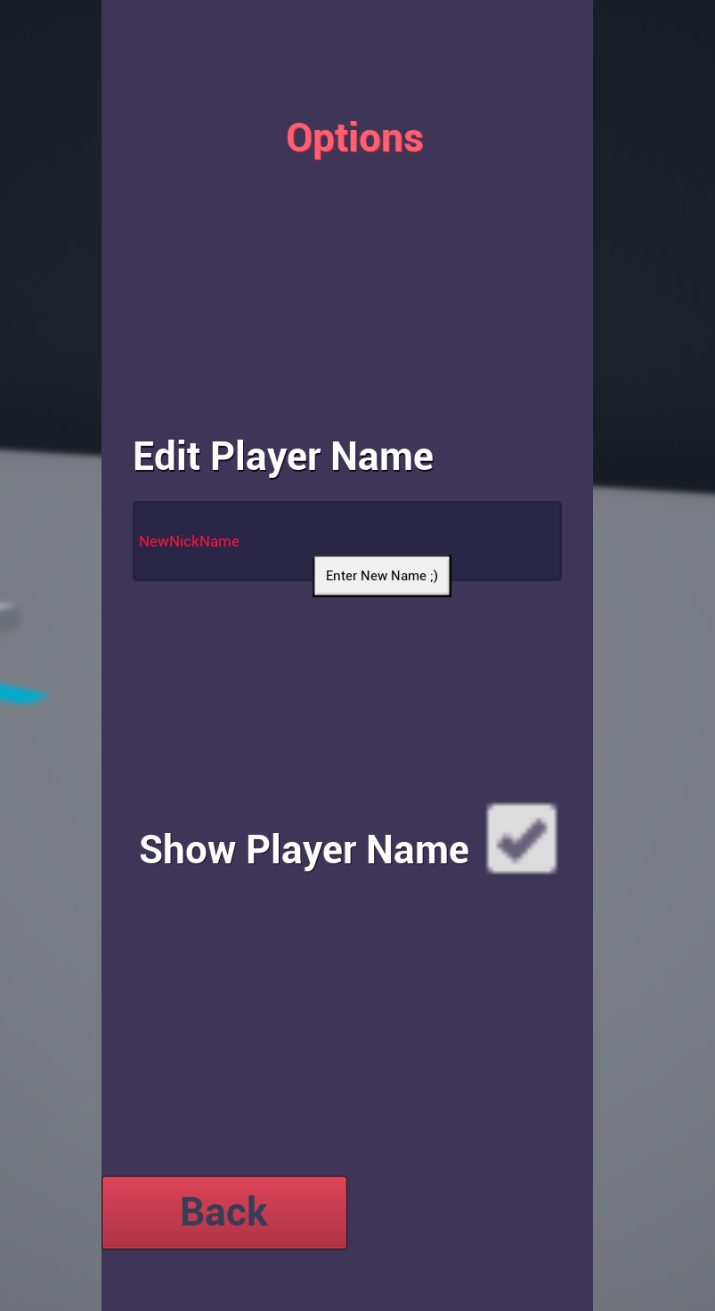
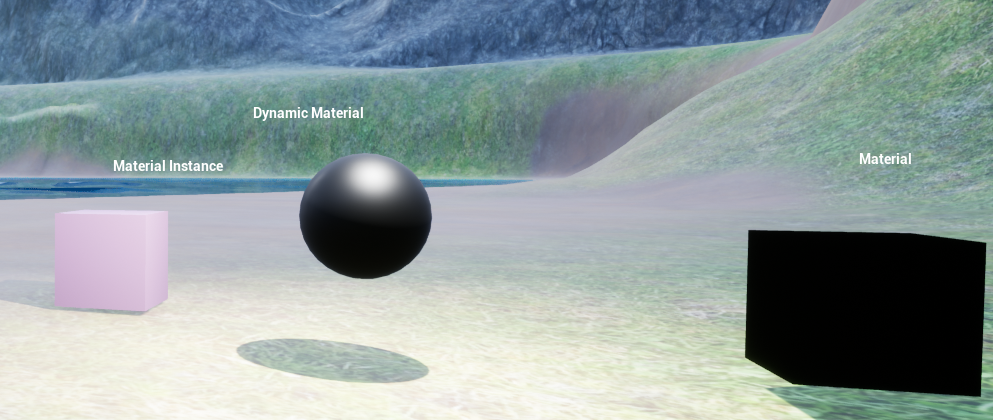
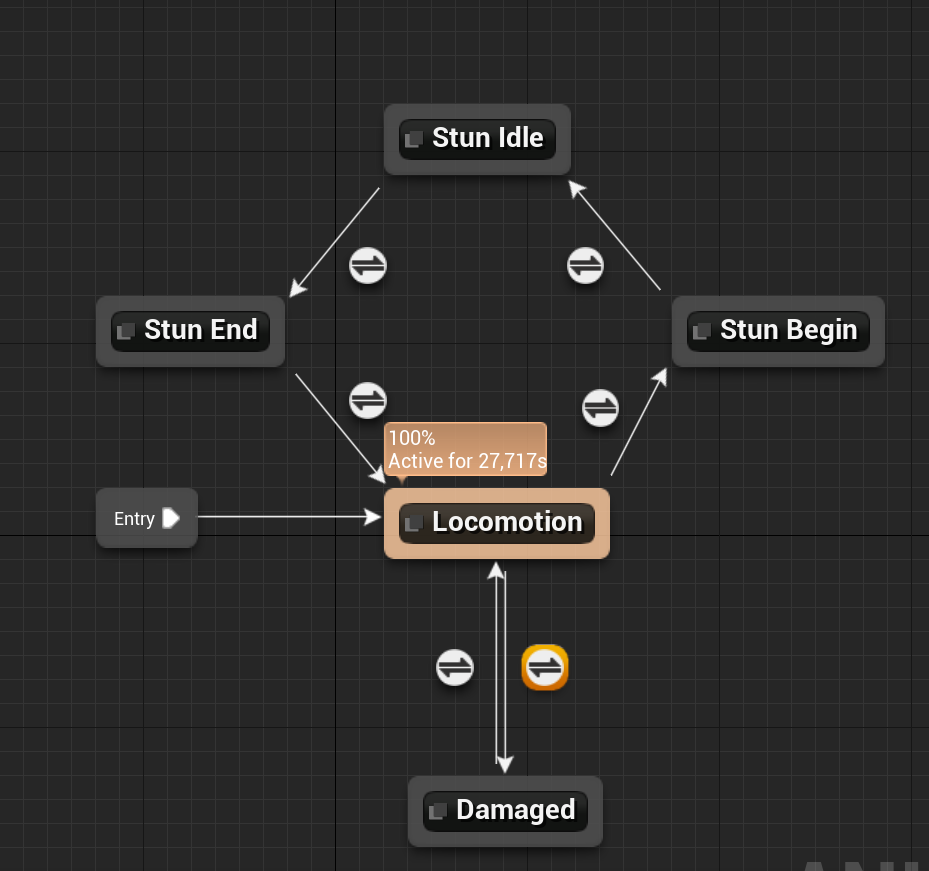
Якщо необхідно обмінюватися даними не через колізії, для цього існують інші види комунікації через код. По-перше, можливо отримувати вказівник на необхідного Актора та завдяки засобам касту взаємодіяти з ним я з іншим, необхідним для комунікування, типом. По-друге, UE має клас Interface, наслідуючись від якого є повна гарантія, що необхідний метод обов’язково буде в Актора, після чого необхідно знайти всіх Акторів, що мають даний інтерфейс та взаємодіяти з ним без прив’язки до його справжнього типу. По-трете, делегати, для їх використання потрібно все рівно проводити операцію касту, але це можна зробити всього один раз, що є

Рисунок 3. Віджет-меню

порівняно краще за перший варіант взаємодії. Делегат має функцію, яку він буде виконувати, та можливість її викликати, коли необхідно.

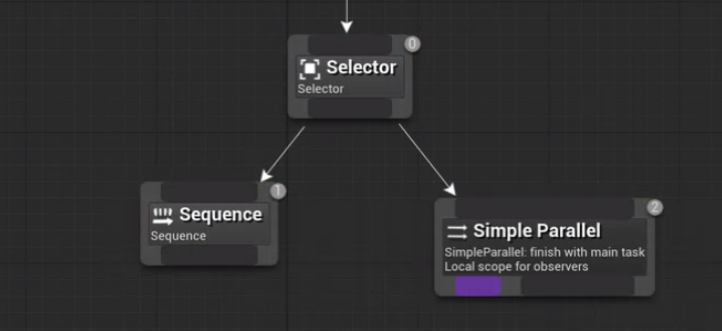
Важливою компонентою для відображення даних на екрані є віджети. Віджети це інтерфейс користувача, який надає можливість відображати інформацію та керувати ігровими механіками. Прикладом цього можуть бути: головне меню, де користувач може налаштовувати опції, запустити основну гру або вийти з неї; міні-мапа, що відображає поточну локацію гравця та повідомляє про найближчі активності тощо. Структура віджету є строго визначеною с точки зору рендеру - порядок елементів згори-донизу означає, що перший елемент буде відображатись на екрані в першу чергу, останній —, відповідно, останнім. Також важливою особливістю є дві можливості використання. Віджет може відображатися як двовимірний об'єкт на екрані користувача незалежно від положення ігрового світу (Рисунок 3), або як той Рисунок 4 Віджети в ігровому світі

самий двовимірний об’єкт але вже в ігровому світі як елемент Актора (Рисунок 4). Ці два підходи є корисними з точки зору відображення даних в певний інформації користувачу. Віджети повинні керуватися класом Controller, отже, розміщуватися повинні так само в ньому. Код створеного класу Віджет, що використовується в роботі, описаний в ДОДАТКУ Б.

Рисунок 5 Приклад автомату станів анімації

Для створення анімації движок має дуже потужні інструменти. Анімація персонажів у Unreal Engine створена на основі Skeletal Mesh, керованої сітки, якою можна керувати для створення анімації. Крім того, площини анімації можна масштабувати до скелетних сіток, щоб застосувати логіку, яка керує поведінкою анімації та взаємодією на рівнях. Unreal Engine надає різноманітні інструменти анімації для роботи зі скелетними сітками для подальшого покращення вашої анімації. Також блюпринт анімації — це потужна та універсальна система, яка дає вам можливість візуально створювати сценарії поведінки анімації. За допомогою Animation Blueprints ви можете керувати змішуванням анімації, взаємодією сценаріїв і створювати інші процедурні поведінки. Приклад побудови станів анімації персонажа показана на Рисунку 5.

Штучний інтелект в UE дає змогу задати поведінку для Акторів без участі для неї користувача. Так, наприклад, NPC (non-player character — неігровий персонаж) мають поведінку, яка керується штучним інтелектом. Для налаштування системи керування Актора Штучним інтелектом необхідно створити та налаштувати три речі. По-перше, створити клас AIController, який безпосередньо контролює поведінку Актора. По-друге, створити Behaviour Tree, далі BH, (дерево поведінки), логіка роботи якого відбувається в обході дерева, створеного користувачем, зліва-направо. Це

Рисунок 6 Основні елементи дерева поведінки

дерево містить алгоритм дій, які має робити AIController. Наприклад, постійне патрулювання території, яке переривається, як тільки в зоні роботи AIController з’являється ціль; далі, якщо ціль валідна і належить зоні дії AIController, виконується знаходження майбутньої точки перебування цілі в залежності від поточного напрямку та швидкості; наступним AIController повинен виконати операцію прицілювання та пострілу в визначену точку. Всі ці алгоритми дій налаштовуються в BH. BH має важливі компоненти, які виконують поставлені задачі по-різному (Рисунок 6).

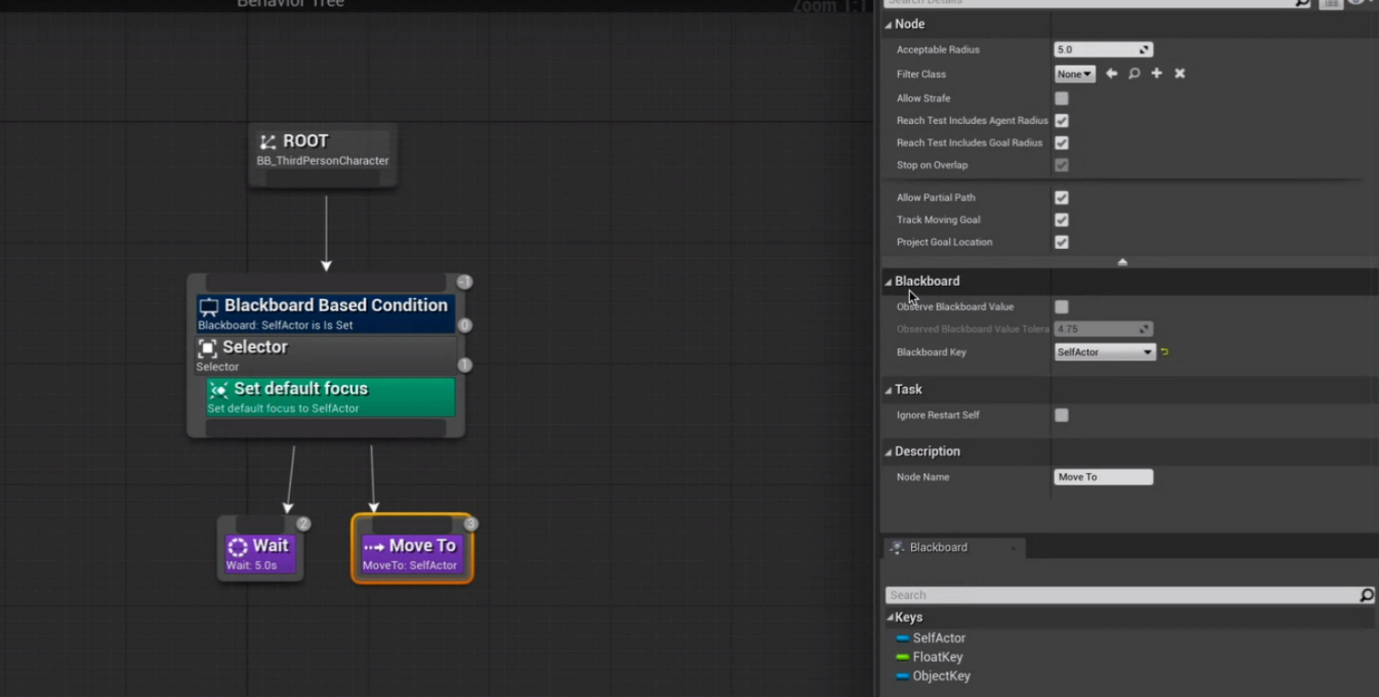
Селектор (Selector) виконує завдання по порядку зліва направо. Вихід з селектора відбувається при успішному виконанні завдання. Сіквенс — подібний до селектора, за винятком того, що вихід відбувається, коли хоча б одне завдання виконалось неуспішно. Проста паралель (Simple Parallel) — композит, що має два вихода: головне завдання і розгалуження, що може вести до іншого завдання або іншої гілки дерева, і виконання обох виходів з сімпл паралел виконуються паралельно. Завдання це листки дерева, вонии виконують певну логіку. Приклад робіт наведено на Рисунку .

Рисунок 7. Приклад реалізації дії поведінки

По-третє, налаштувати Blackboard. Blackboard — це контейнер “ключ-значення” з різними змінними. Приклад його використання — збереження всіх цілей, які знаходяться в зоні роботи AIController. Налаштування цих трьох компонент дозволяють отримати автономну поведінку для Актора. Останній необхідний компонент для роботи всієї системи поведінки — Nav Mesh Bounds Volumes — використовуються для керування тим, де на рівні буде побудовано навігаційну сітку. Він розраховує шляхи навігації в областях рівня. У межах об’єму сітка цього меша створена на всіх поверхнях під відповідним кутом, на який можна наступати і дозволяє в своїх межах виконувати логіку Актору, який керується штучним інтелектом.

**2.2. Вибір програмних засобів реалізації**

*Програмне забезпечення*

Програмне забезпечення UE5 дозволяє легко писати код, оновлювати проект згідно змін, налагоджувати його роботу і створювати готовий проект. Він складається з безлічі бібліотек та плагінів. Програмне забезпечення, сумісне з багатьма операційними системами (Windows, Linux, Mac OS X). Середа розробки - Visual Studio 2022 - сумісна з Unreal Engine та дозволяє відлагоджувати програму наявними в ній засобами.

Об’єкт PlayerInput відповідає за перетворення вхідних даних від гравця в дані, які актори (такі як PlayerController або Pawn) можуть розуміти та використовувати. Це частина потоку обробки вхідних даних, яка перетворює апаратні дані від гравців у події гри та руху за допомогою відображення вхідних даних гравців і компонентів введення. Ви можете почати з огляду запису або вибрати будь-яку з наведених нижче тем.

При створені анімацій корисними є Animation Blueprint, Blendspace та Animation montage. Animation Blueprint дозволяє описати всі стани анімації та переходи між ними (переходи із стану спокою до стану руху, далі стан підняття предмета, стан стрибка тощо). Blendspace дозволяє поєднати кілька анімацій, що плавно змінюють одна дну в залежності від вхідних даних. Так, наприклад, маючи вхідний параметр кількості отриманого пошкодження, анімації спокою та анімацію отримання пошкодження можна отримати різні варіацію анімації отримання пошкодження від 0 до максимального значення отримання пошкодження. Animation montage дозволяє отримати анімацію, яка може запускатися в вищим пріоритетом з усіма іншими анімаціями. Наприклад, анімація перезарядження зброї повинна викликатися після нажатя відповідної кнопки незалежно від того, яка поточна анімація виконується.

*Плагіни*

Під час використання Enhanced Input ви можете додавати та видаляти контексти зіставлення під час виконання програвача. Це полегшує керування великою кількістю дій. Ви можете змінити поведінку певних входів залежно від поточного стану Актора. Наприклад, якщо у вас є гравець, який може ходити, спринтувати та лежати. Для кожного з цих типів руху символів ви можете змінити контекст відображення, щоб клавіша CTRL виконувала різні дії. Під час ходьби і натискання клавіші CTRL ви повинні присісти. Під час спринту та натискання CTRL ви повинні сковзати. Лежачи і натиснувши CTRL, ви повинні встати.

Для проектів, які вимагають більш розширених функцій введення, таких як складне керування введенням або переналаштування контролю під час виконання, експериментальний плагін Enhanced Input надає розробникам простий шлях оновлення та сумісність із системою введення за замовчуванням. Цей плагін реалізує такі функції, як радіальні мертві зони, хордові дії, контекстне введення та встановлення пріоритетів, а також можливість розширити власну фільтрацію та обробку необроблених вхідних даних у середовищі на основі активів. Отже, Enhanced Input є більш гнучким та надає більше можливостей щодо динамічної зміни вводів користувача. Приклад налаштування звичайного двовимірного руху наведено в ДОДАТКУ А.

**РОЗДІЛ 3. Проектування системи керування світлодіодний освітленням**

Даний розділ описує створення функційної схеми, її перевірку та реалізацію; описує створення програмного забезпечення для програмних модулів 1, 2 та 3, їх основні функції та принципи роботи; описує результат створення та роботи пристрої керування.

* 1. **Розробка ієрархії класів керованого персонажу та ворогів**

Даний підрозділ описує за що відповідальний окремий клас та їх основні методи.

EnhаncedInputChаrаcter - клас, що налаштовує ввід користувача. Назви подій вводу користувача та відповідних методів виклику є:

* Підйом, що прив’язаний до кнопки “Space”, викликає метод OnClimbActionStarted();
* Переміщення, що прив’язаний до кнопок “W” “S” “D” “A”, викликає метод Move();
* Огляд, що прив’язаний до миші, викликає метод Look();
* Прискорення, що прив’язаний до кнопки “Shift”, спрацьовує при натисканні, зупиняє роботи при відпусканні, викликає метод WalkJogSwitcher\_Implеmеntаtіon();
* Присідання, що прив’язаний до кнопки “С”, викликає метод CrouchSwitcher\_Implеmеntаtіon();
* Ховання, що прив’язаний до кнопки “Q”, викликає метод HideSwitcher\_Implеmеntаtіon();
* Атака, що прив’язаний до лівої кнопки миші, викликає метод Throw();
* Прицілювання, що прив’язаний до правої кнопки миші, працює, поки кнопка натиснута, викликає метод ShowParticlePath(), при відпусканні кнопки викликається метод RemoveParticlePath();
* Повернення снаряду, що прив’язаний до кнопки “R”, викликає метод SpawnKnifeBack();

Деякі методи містять в назві *\_Implеmеntаtіon*, так як ці методи наслідуються від інтерфейсів зміни швидкості, приховування та присідання. Інтерфейси необхідні для взаємодії з класом анімацій.

SHChаrаcter - клас, що відповідальний за зміну швидкості переміщення. Містить змінну стану швидкості персонажа та медоти отримання цієї змінної. Має:

* віртуальний метод визначення напрямку руху ReturnDirection(), що реалізований визначати напрямок в XY координатах для переміщення вперед-назад та праворуч-ліворуч, цей метод переписаний в класі ;
* віртуальний метод WalkJogSwitcher\_Implеmеntаtіon() призначений для переключання прискорення в режим “бігу”;
* віртуальний метод GetPlayerMovementInfo(), призначений для повернення даних швидкості переміщення для стану ходьби, та стану бігу, цей метод переписаний в класі CrouchChаrаcter.

ClimbingSystemChаrаcter — клас, що містить методи переміщення вертикальними поверхнями та переміщення через перешкоди. Містить змінену компонету переміщення — CustomMovementComponent. Його основні методи:

* віртуальний метод OnClimbActionStarted(), що запускає метод руху чеез перешкоду в CustomMovementComponent;
* віртуальний метод Move(), що змушує персонажа рухатися вгору-вниз та ліворуч-праворуч, якщо поточний стан персонажа - піднімання, та викликає базовий метод в іншому випадку;
* метод HandlеClіmbMovemеntInput(), що реалізовує рух по вертикальним поверхням в залежності від положення персонажа;

CustomMovementComponent - клас користувацьої компоненти переміщення, призначений для встановлення швидкості підйому, встановлення типу перешкоди, реалізації переміщення через низькі першкоди та активації режиму підйому горизонтальними поверхнями. Його основні методи:

* OnMоvementMоdeChanged() — змінює стан переміщення на стан “підйому” або попередній стан;
* PhysCustоm() — оновлює дані швидкості та анімацій підйому;
* ConstrainAnimRооtMotionVelocity() — повертає вектор швидкості анімації;
* ProcessClimableSurfaceInfo() — записує поточний вектор нормалі вертикальної поверхні;
* ChеckShоuldStopClimbіng() — визначає чи персонаж досяг верхнього краю вертикальної поверхні;
* ChеckHasReachedFloor() — визначає чи персонаж досяг горизонтальної поверхні;
* GеtClimbRotation() — визначає завдяки вектору нормалі поверхні поворот персонажу;
* TrаceClimbableSurfaces() — визначає, чи є перед песонажем вертикальна поверхня, вища за персонажа;
* CheckHаsReachedLedge() — визначає, чи персонаж поряд з верхнім краєм вертикальної поверхні;
* TryStаrtVаulting() — перевіряє, чи є поряд перешкода, нижча за персонажа, якщо є, то запускає анімацію переміщення з заданими конкордатами пересування через перешкоду, якщо ні, запускає анімацію стрибка;
* PlаyClimbMontage() — запускає, анімації початку, кінця підйому, підйому з вертикальної поверхні;
* OnClimbMоntageEnded() — перемикає параметри персонажа після закінчення програвання анімації;
* DоLineTraceSingleByObject() — повертає результат влучання проміня в об'єкт перед персонажем, визначає чи є перед персонажем перешкода;
* DоCаpsuleTraceMultіByObject() — визначає, повертає результат влучання капсули з розміром персонажа в об'єкт перед персонажем, визначає чи є перед очима персонажем перешкода;;
* CаnStаrtClimbing() — визначає, чи є наявні елементи у персонажа для початку пійдому;
* CanClimbDоwnLеdge() — визначає, чи можливий рух вниз;
* TogglеClimbing() — керує переходом до стану підйому та зі стану підйому, якщо підйом почату, в залежності від перевірок запускає рух вертикальної поверхнею, спуск вниз, пересування через перешкоду, стрибок;
* IsClіmbіng() — повертає поточний стан підйому персонажа;
* PlаyTurnMontаge() - програє анімацію повороту на 180 градусів праворуч або ліворуч.

WithCamerаChаrаcter — клас, що містить камеру та компоненту, що намагається утримувати дочірні елементи на фіксованій відстані від батьківського. Має методи доступу до цих компонент: GetCamera() та GetSpringArm().

ThrowChаrаcter клас, що реалізує як атаку постріл снаряду, його повернення, прицілювання та переміщення до снаряду. Містить компоненти: візуальних ефектів, траєкторію руху снаряду; лінії часу, за якою відіграється візуальний ефект та зникнення або поява персонажа через динамічну зміну матеріалу скелет-меша. Містить методи: зміни матеріалу скелета-меша; активації візуальних ефектів, встановлення траєкторію руху снаряду. Основні методи:

* віртуальний метод Throw(), що запускає снаряд у напрямку направлення камери. Повторне використання встановлює нову локацію персонажа в на місце кинутого снаряду;
* віртуальний метод ShowParticlePath(), що встановлює та відображає траєкторію руху снаряду, основаної на початковій локації запуску, швидкості руху персонажа та снаряду, нарпямку, у якому направлена камера персонажа;
* віртуальний метод RemoveParticlePath(), що видаляє траєкторію руху.
* віртуальний метод SpawnKnifeBack(), що видаляє кинутий снаряд та відображає його в руці персонажа;
* делегат OnAddSplineMeshAtIndex, на який підписується метод в Блюпринті, що всановлює візуалізацію траєкторії руху снаряду.

CrouchChаrаcter — клас, що реалізовує перехід між звичайним станом та станом присідання. Містить змінну стану положення тіла персонажа та медоти отримання цієї змінної. Має:

* віртуальний метод GetPlayerMovementInfo(), що виклкає базовий метод, якщо персонаж не в стані присідання, і повертає дані швидкості переміщення для стану присідання в іншому випадку;
* віртуальний метод CrouchSwitcher\_Implеmеntаtіon(), що переводить персонажа в стан присідання.

HideChаrаcter — клас, що реалізовує приховування персонажа за валідним об’єктом. Він має визначати об’єкти біля персонажа та перевіряти на наявність можливості ховання за ними. Містить змінну стану ховання персонажа та медоти отримання цієї змінної; методи обробки колізій з об’єктами біля персонажа, їх перевірки на валідність укриття, визначення меж руху. Має:

* віртуальний метод визначення напрямку руху ReturnDirection(), якщо поточний стан персонажа є “схований”, то повертає проекцію вектора руху на вектор, колінеарний до напрямного вектора персонажа та перпендикулярний до вектора нормалі поверхні, викликає базовий метод в іншому випадку.
* віртуальний метод переходу в стан “ховання” HideSwitcher\_Implеmеntаtіon(), який викликається при наявності валідного укриття біля персонажа;
* віртуальний метод руху Move(), викликається, якщо поточний стан персонажа — “ховання”, викликає базовий метод в іншому випадку. Про спробі переміщення за межі укриття зупиняє рух персонажа;
* метод SеtUpHideBorders() визначає межі укриття, якщо його ширина менша за ширину персонажа, в іншому випадку встановлює межі на відстані ширини персонажа;
* метод UpdateBorderLocation() змінює межу укриття, якщо під час руху біля персонажа наявна поверхня;
* методи ToHide() та FromHide() виконуєть зміни налаштувань персонажа під час переходу до та з режиму “ховання”;

MotionWarpingChаrаcter — клас, що реалізовую ближню атаку. Має:

* віртуальний метод Throw(), що перевіряє наявність валідної цілі атаки біля персонажа та викликає метод Attack() при наявності, в іншому випадку, викликає базовий метод Throw();
* метод Attack() відключає можливість руху, встановлює Поворот та локацію персонажа, запускає анімацію атаки та викликає метод реакції на атаку AfterAttacked у цілі;

Клас ворога міститиме методи різних режимів патрулювання та метод знищення:

* RecivePatrolData\_Implementation() — метод отримання поточного стану патрулювання;
* GoForwardBackwardPath\_Implementation() — слідування шляхом вперед-назад;
* GoCyclePath\_Implementation() — слідування шляхом циклічно;
* AfterAttacked() — метод знищення, який програє відповідну анімацію;

Алгоритм роботи персонажа на ввід користувача працює наступним чином:

1. Перед початком роботи персонаж має характеристик: положення — *стояння*, швидкість — *ходьба*, видимість — *видимий. У* цьому стані рух персонажа горизонтальний.
2. Переключення швидкості: якщо користувач тримає кнопку “Shift”, то відбувається переключення стану швидкість — *біг.*
3. Переключення положення: якщо користувач натискає кнопку “С”, то відбувається переключення стану положення у залежності від попереднього стану — *стояння/присід.*
4. Переключення видимості: якщо користувач має стан положення - *присід*, натискає кнопку “Q”, то, при наявності поблизу укриття, відбувається переключення стану видимість — *прихований.* Укриття Рух персонажа можливий лише уздовж укриття. Якщо під час руху положення камери дивиться у зворотню від персонажа сторону, запускається анімація повороту на 180 градусів ліворуч або праворуч, залежно від поточного положення персонажа
5. Атака: якщо користувач натискає ліву кнопку миші, то персонаж кидає снаряд у напрямку повороту камери; повторне нажаття кнопки переміщує персонажа до снаряду. Якщо персонаж перебуває в стані видимість — *прихований*, то персонаж атакує ціль біля себе, якщо вона існує.
6. Прицілювання: якщо користувач тримає праву кнопку миші, то відображається траєкторія руху, по якій снаряд буде летіти.
7. Вертикальний рух: при натисканні пробілу персонаж:

* при відсутності перешкод стрибає;
* при наявності перешкоди, нижчої за персонажа, перестрибує перешкоду;
* при наявності перешкоди, вищої за персонажа, починає вертикальний рух.

1. Обробка вертикального руху: Вихід з режиму підйому відбувається при натисканні пробілу; коли персонаж досяг нижнього або верхнього краю вертикальної поверхні. Рух відбувається вгору-вниз та праворуч-ліворуч.

*Зауваження:* Дана система не реалізовує перевірку використання компонет в нелогічних умовах. Наприклад, поки персонаж перебуває у вертикальному русі, він може запускати снаряд. *Результат:* система працездатна, коректно обробляє ввід користувача.

**3.2 Створення системи анімацій руху**

Створення системи анімацій руху персонажа повинна реалізувати режими руху, показати спосіб, яким можливо об'єднати кілька анімацій разом, процедурно їх змінювати, щоб створити більш обґрунтований результат щодо взаємодії персонажа зі світом та з швидкодії обробки. Є є два загальні підходи до реалізації руху персонажа: один, де введення використовується для управління логікою анімації, а рухи персонажа керуються даними анімації; інший, де вхідні дані обробляються кодом, який математично переміщує персонажа ігровим світом, а анімація якомога найкраще відповідає цьому руху. Обидва ці підходи мають свої переваги та недоліки. Рух, керований анімацією, який називається основним рухом, зазвичай виглядає краще реалістичний, але менш реагує на вхід гравця та вимагає більше типів і варіацій анімації руху, щоб функціонувати на базовому рівні. З іншого боку, керовані кодом рухи можуть миттєво реагувати на ваш вхід і керуються напряму за допомогою математики та логіки, що обґрунтовує рух персонажа. Прикладом є тертя між персонажем і землею, що впливає на швидкість руху персонажа, використовуючи цю систему.

Реалізована система відноситься до другого типу. Отже, вона поділяється на етапи: *отримання даних руху персонажа кожного кадру. переходи між анімаціями.*

*Отримання даних руху персонажа*

Продуктивність анімаційної системи як і ефективність кожного кадру залежить від того, скільки часу потрібно вашому потоку гри та робочому потоку для обробки анімаційної системи кожного кадру. Анімації додаються до плану анімації, де вони будуть оцінені та відтворені на персонажі під час виконання. Додаткові процеси, такі як змішування анімації, фізичне моделювання тощо, будуть відніматися від продуктивності проекту. Деякі процеси є простими та не вимагають багато часу для обробки, інші процеси виконують складніші операції, що призводять до кращої анімації, але може потребувати значної частини бюджету продуктивності. Усі функції анімаційної системи мають відповідну вартість продуктивності. Як правило, найдорожчою операцією в анімації є логіка в Event Graph тому необхідно максимально скоротити логіку в цьому графі, щоб отримати найкращу продуктивність. Кожен анімаційний кадр містить кілька розрахунків, і з кожним розрахунком оцінюється графік подій. Оцінки Event Graph зазвичай є найбільшою операцією, що виконується кожного кадру, вони є послідовними, тобто оцінка кожного наступного розрахунку займає більше часу. Логіка обрахункіів повинна бути перенесена, до потокобезпечних функцій, які виконується одночасно на доступних робочих потоках. Коли ви переміщуєте всі операції Event Graph у функції Thread Safe, операції можна виконувати одночасно, що значно скорочує час, необхідний для оцінки кожного розрахунку, що покращує продуктивність. Для цього необхідно створювати функції Thread Safe — це функції Blueprint, які можна використовувати для виконання логіки для встановлення змінних і властивостей, які можуть використовуватися вашою анімаційною системою, на додаток до виконання інших операцій, які зазвичай виконуються в графі подій.

*Переходи між анімаціями*

овавава

**3.3 Створення поведінки ворогів**

**3.4 Створення програмного модуля 2**

Програмний модуль 2 (сервер) - шлюз між користувачем та апаратною частиною, що забезпечує з користувачем бездротовий зв’язок та дротовий з контролером. Його задача: приймати дані через протокол TCP від користувача, аналізувати дані, надсилати відповідь на запит користувача або надсилати дані до апаратної частини, приймати дані від апаратної частини через USB та надсилати їх користувачеві. Алгоритм роботи програми наведено на Аркуші 3. Працює програмний модуль наступним чином:

1. відбувається ініціалізація сервера та очікування підключення клієнта;
2. після приєднання сервер перевіряє чи є дані від клієнта та зчитує їх при наявності;
3. після отримання сервер аналізує дані: якщо для сервера, виконує запит або відправляє вказівку для апаратної частини;
4. далі , якщо дані для апаратного модуля в наявності, сервер відправляє ці дані;
5. при наявності даних від апаратного модуля зчитує їх та відправляє до клієнта;
6. далі програма повертається до пункту 2 до виключення клієнта або програми;
7. при відключенні клієнта програма повертається до пункту 1.

Опис роботи основних функцій:

* void ServerSide(int port) - налаштування сервера через протокол TCP;
* void RX\_procc(), void TX\_procc() - функції читання та запису даних до та від апаратного модуля;
* void ReceiveDataFromClient(), void SendDataToClient(String RXcommand) - функції, що отримують дані від клієнта та відправляють їх до клієнта;
* void Switcher(String com) - функція, яка аналізує отримані дані від клієнта, відповідає на запит, якщо дані призначені для сервера, та відправляє команди на апаратну частину в іншому випадку.

Код програмного модуля 2 наведений у **Додатку Б**. Результат роботи програми показаний у Рис. 3.8.

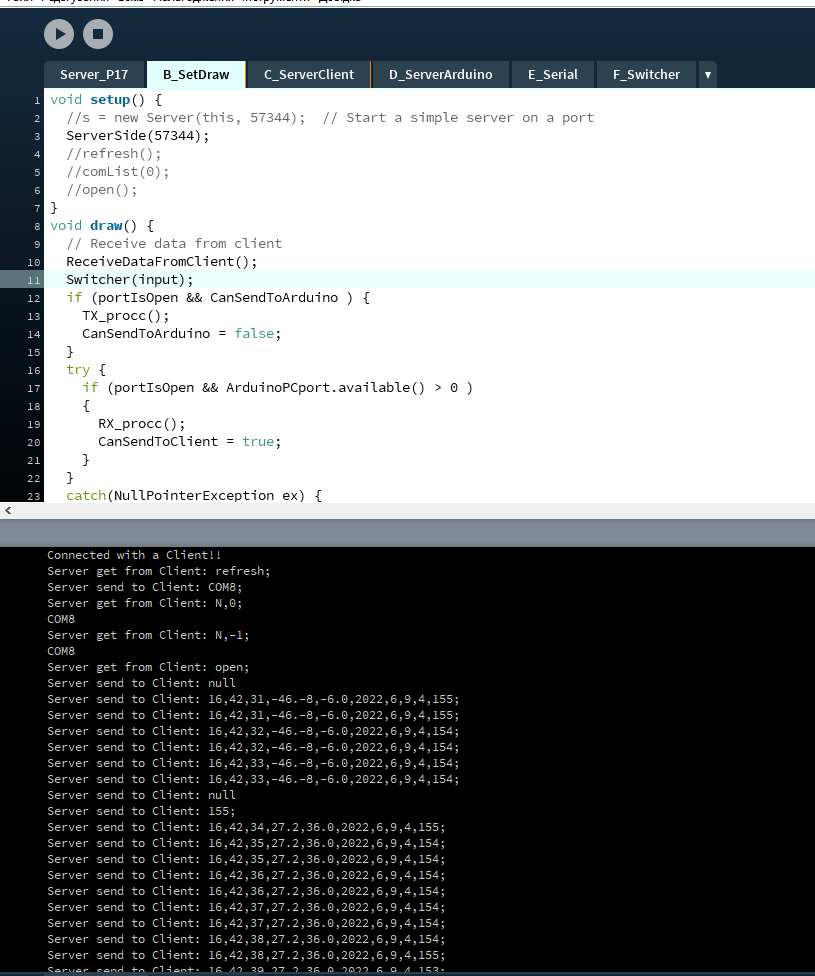


Рисунок 3.8 Результат роботи серверної частини

**3.5 Створення програмного модуля 3**

Програмний модуль 3 (клієнт) - програма, завдяки якій користувач може зручно обирати режими роботи, кольори та параметри стрічки, підключається до сервера по локальній мережі, формує рядок команду та надсилає її до сервера; отримує дані від сервера та показує їх користувачеві.Алгоритм роботи програми наведено на Аркуші 4. Код програмного модуля 3 наведений у **Додатку В.** Працює програмний модуль наступним чином:

1. Спочатку ініціалізуються елементи меню, деякі, у залежності від обраної секції, приховуються та блокуються.
2. Далі користувачу доступна кнопки “Serverconnect” та “Serverdisconnect”, після натискання який програма підключається до або відключається від сервера.
3. Після підключення користувач може підключитися до порту,через який сервер з’єднаний з апаратним модулем. Після підключення запускається апаратний модуль.
4. Після стають доступними три вікна для користувача на вибір: у першому користувач обирає кольори та надсилає їх на сервер; у другому користувач обирає режим та надсилає його до сервера; у третьому користувач може обрати яскравість, швидкість, температуру та встановити авто налаштування;
5. Інформація від сервера виводиться вгорі. Виводяться час, дата, температура, вологість яскравість.

Опис роботи основних функцій:

* void ClientSide(String address, int port) - функція, що відповідає за підключення до сервера;
* gui(), ForGroup1(), ForGroup2(), ForGroup3(), ForGroup4(), Clock() - функції, що налаштовують елементи меню.
* void ReceiveDataFromServer(), void SendDataToServer(String dataToServer) - функції, що отримують та надсилають дані до сервера.
* void PrepareDateFromArd(String info) - розбиває дані від сервера на підстроки для зручного зображення;
* ShowClock() - функція, що відображає інформацію від сервера.

Вигляд програми показаний у Рис. 3.9**.**

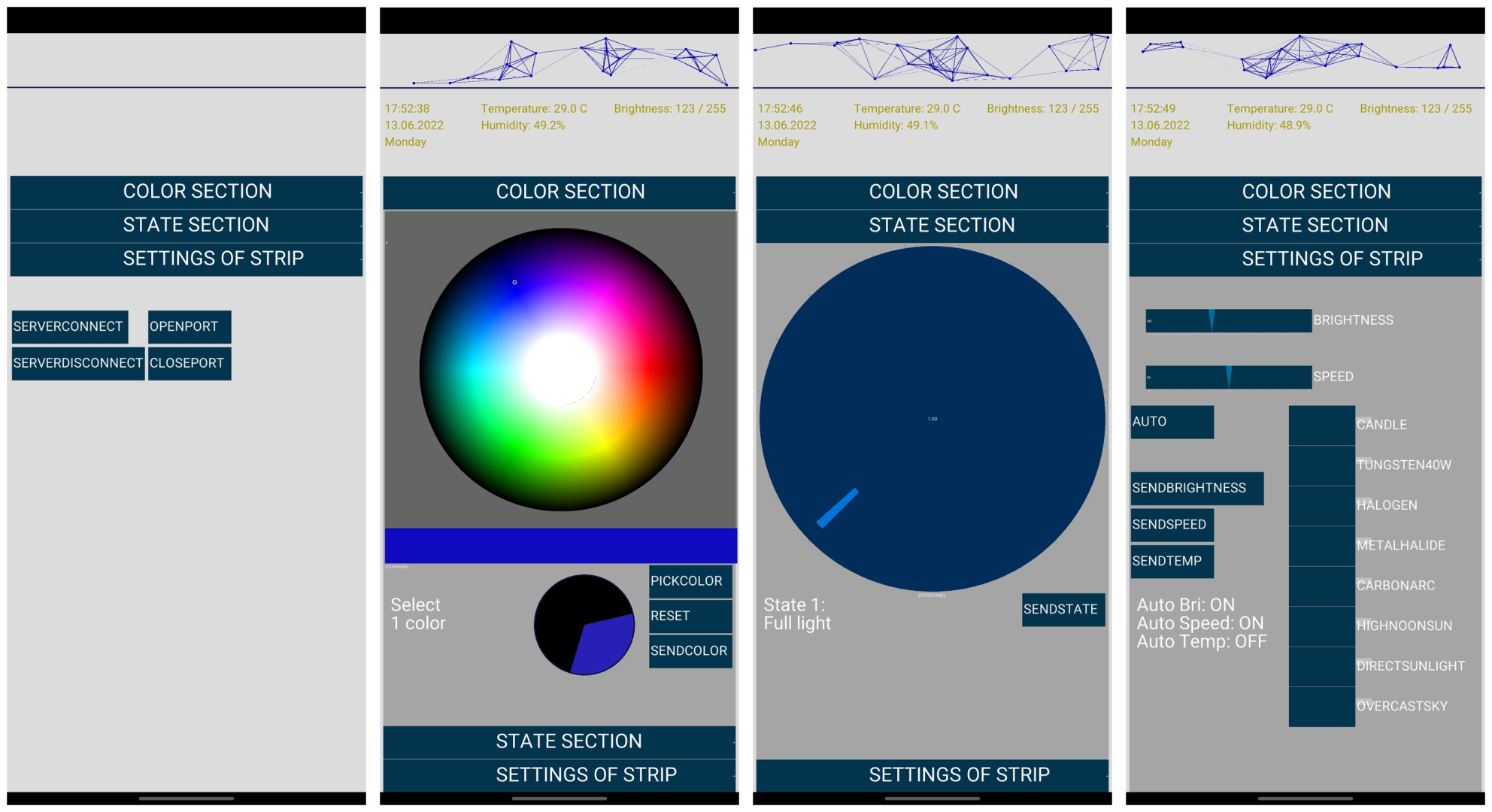


Рисунок 3.9 Вигляд меню програмного модуля 3

**3.6 Результат роботи пристрою керування**

Перед запуском апаратний модуль має бути приєднаний до ПК, на якому запущений програмний модуль 2 (сервер). Програмний модуль 3 запущено на телефоні, який знаходиться у одній локальній мережі з ПК (через Wi-Fi). Користувач запускає програмний модуль 3, підключається до сервера, потім до апаратного модуля (Рис. 3.10).

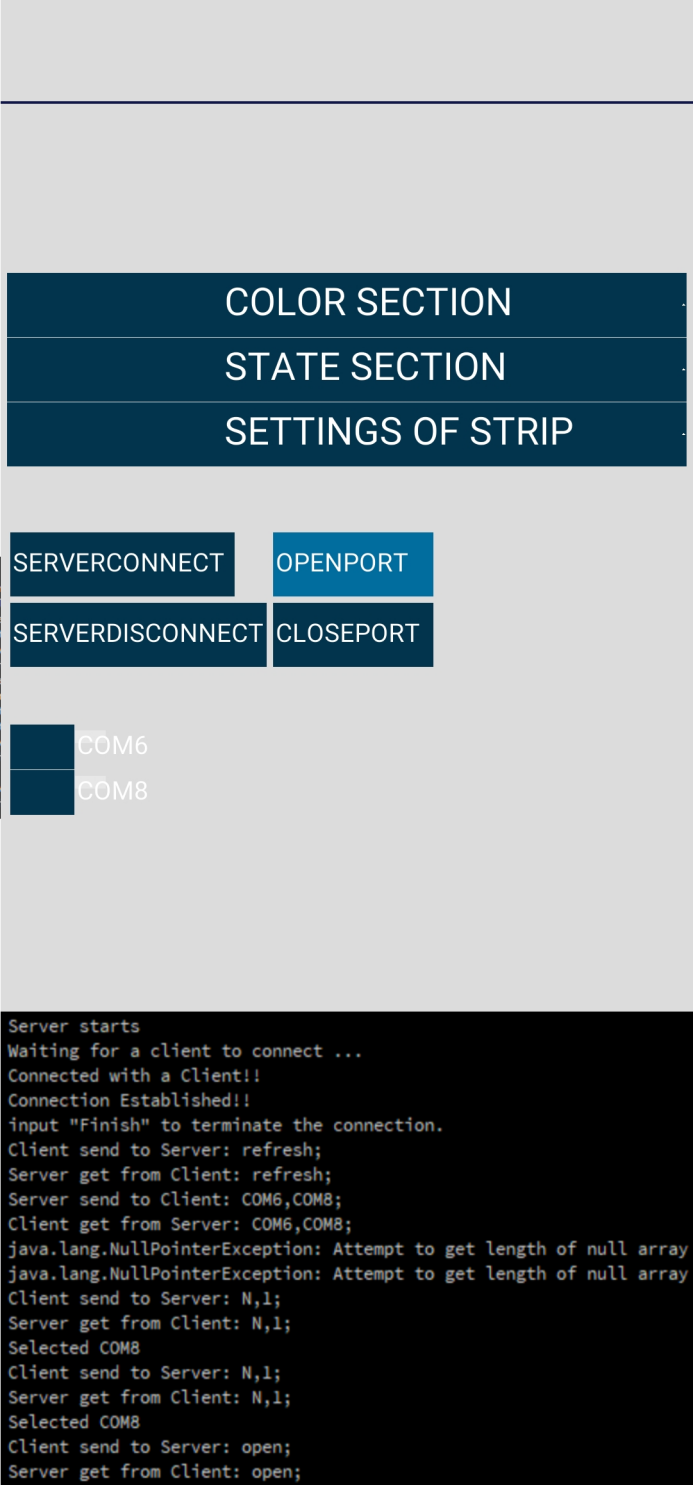


Рисунок 3.10 Клієнтська програма - підключення (вгорі) та реакція серверної програми (внизу)

Далі налаштовує кольори, обирає режим та налаштування, які надсилає до сервера по черзі.(Рис 3.11). Після підтвердження команди, програмний модуль 3 формує рядок з символу команди та числових даних-параметрів та надсилає їх де сервера.

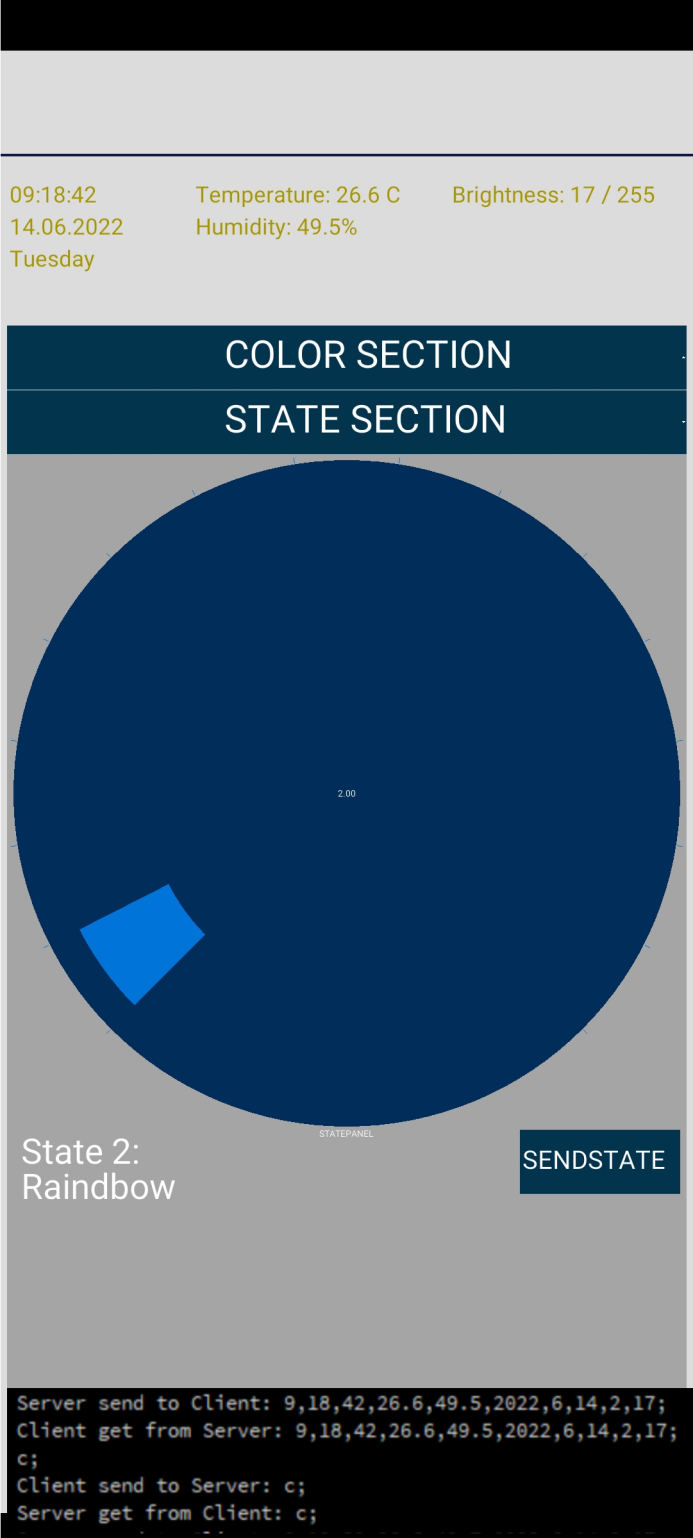


Рисунок 3.11 Клієнтська програма - надсилання команди (вгорі) та реакція серверної програми (внизу)

Сервер приймає дані, які надсилає до апаратного модуля, який аналізує та виконує їх (Рис. 3.12).



Рисунок 3.12 Реакція апаратної частини на команду з сервера

# **РОЗДІЛ 4.**

**РОЗДІЛ 5. Економічна частина**

У цій роботі розроблялась система керування світлодіодним осві

**ВИСНОВКИ**

Під час виконання бакалаврської кваліфікаційної роботи були: розглянуто використання Unreal Engine в розробці ігор і візуалізації; розглянули базові класи, які лягають в основу будь-якого проекту на Unreal Engine, а також звернули увагу на важливий аспект реплікації, який дозволяє створювати багатокористувацькі ігри. Ми також розглянули можливості онлайн-інтеграції та використання віджетів для створення інтерактивного інтерфейсу. Крім того, ми зазначили важливість системи штучного інтелекту в Unreal Engine, яка відкриває безмежні можливості для створення ігор та симуляцій з реалістичною поведінкою персонажів та об'єктів. Застосування Unreal Engine стає все більш важливим у галузі розробки ігор та інтерактивних додатків, і вона надає розробникам широкий спектр інструментів для втілення їхніх творчих ідей.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Посібник по основним можливостям Unreal Engine [Інтернет-ресурс]. – Режим доступу: <https://github.com/MrRobinOfficial/Guide-UnrealEngine>
2. [Документація Unreal Engine [Інтернет-ресурс]. – Режим доступу: https://docs.unrealengine.com/](https://github.com/MrRobinOfficial/Guide-UnrealEngine)
3. [Курс Unreal Engine 5 C++ Developer: Learn C++ & Make Video Games [Інтернет-ресурс]. – Режим доступу:  
   https://www.udemy.com/course/unrealcourse/](https://github.com/MrRobinOfficial/Guide-UnrealEngine)

**ДОДАТОК А**

**ДОДАТОК Б**

**ДОДАТОК В**