КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ВИКОРИСТАННЯ ПАТЕРНУ ECS ДЛЯ СТВОРЕННЯ КОМП'ЮТЕРНИХ СИМУЛЯЦІЙ

Виконав студент 4-го курсу
Семен КЛЯЦКО _____
(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Анатолій ПАШКО _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____
(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 38 сторінок, 13 ілюстрацій, 7 джерела посилань.

СИМУЛЯЦІЯ, ПАТЕРНИ, ОТИМІЗАЦІЯ, ІГРОВИЙ РУШІЙ, UNREAL ENGINE, ENTITY – COMPONENT – SYSTEM, MASS FRAMEWORK, GAME FRAMEWORK, DATA ORIENTED PROGRAMMING, OBJECT ORIENTED PROGRAMMING

Об’єктом роботи є створення комп’ютерних симуляцій, використання патерну ENTITY – COMPONTENT – SYSTEM для створення комп’ютерних симуляцій.

Метою роботи є використання ENTITY – COMPONTENT – SYSTEM для оптимізації комп’ютерних симуляцій, розробка та програмна реалізація симуляції з графічним відображенням на основі патерну ENTITY – COMPONENT – SYSTEM для моделі попиту та пропозиції(supply and demand), а також підбір і дослідження необхідних засобів та технологій.

Інструменти розроблення: Visual Studio 2022, Rider, Unreal Engine 5.1, Mass Framework, Game Framework.

Результати роботи: створено реалізацію комп’ютерної симуляції моделі попиту та пропозиції, на основі патерну Entity – Component – System. Продемонстровані переваги та недоліки використання Entity-Component-System та Data Oriented Design для створення симуляцій.

Розроблений програмний продукт може бути використаний в навчальних цілях закладами середньої та вищою освіти для проведення наглядних демонстрацій викладеного матеріалу. На основі описаних технологій можна створити симуляції більш складних процесів для використання в науково – дослідницькій діяльності та комерційній сфері.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

OOP – Object Oriented Programming

ECS – Entity – Component – System

GC – Garbage Collector

CDO – Class Default Object

DOD – Data Oriented Design

UE – Unreal Engine

ВСТУП

Оцінка сучасного стану об'єкта розробки. З появою перших комп'ютерів почали з'являтися перші комп'ютерні симуляції. Симуляції дуже корисні для дослідження моделей різноманітного рівня об'єму та складності. На основі математичної моделі деякого процесу можна побудувати його комп'ютерну симуляцію і використовувати велику обчислювальну потужність комп'ютера для дослідження різноманітних сценаріїв поведінки системи для різних початкових параметрів цієї системи.

Одне з перших практичних застосувань комп'ютерних симуляцій – це моделювання фізичних ядерних процесів для їх аналізу і дослідження для подальшого розвитку ядерної енергетики та створення ядерної зброї. Відбувалось це під час Другою Світової та Холодної війн і дало великий поштовх для розвитку комп'ютерних симуляцій і поширило їх використання в науково – дослідницькій та інших сферах.

Наразі комп'ютерні симуляції використовуються в економічній сфері для моделювання ринкових процесів для їх дослідження і подальшого прогнозування, космічній та авіа сфері для симуляції польотів в цілях дослідження та для тренування пілотів, в біологічній та медичній сфері для дослідження поведінки нових хвороб і створення нових фармакологічних препаратів(використовувались для створення вакцини від COVID - 19), військовій сфері для навчання, стратегічного планування, проектування і тестування військового обладнання та оцінки потенційних загроз.

Також симуляції є дуже важливою складовою у розробці комп'ютерних ігор так як вони необхідні для створення реалістичного ігрового середовища. А розвиток графічних технологій, нових ігрових рушіїв та більш досконалих систем рендеру дозволив створювати графічне відображення для симуляцій. Приклад такою симуляція буде представлений в цій дипломній роботі.

Для створення симуляцій з графічним представленням будемо використовувати ігровий рушій, основний підхід для написання програм на

ігрових рушіях – об’єктно орієнтований підхід, але так як в симуляціях дуже часто моделюється поведінка дуже великої кількості об’єктів – цей підхід може бути неефективним, тому має сенс використання іншого підходу, наприклад, Data Oriented підхід. Одним з патернів Data Oriented програмування, який чудово підходить для реалізації ігрових процесів та для створення симуляцій є Entity – Component -System, який і буде використовуватись для створення симуляцій в даній дипломній роботі. Використання Data Oriented підходу і конкретно патерну ECS є наразі нетиповим для написання програмного забезпечення на основі ігрових рушіїв і знаходиться на початковому етапі свого розвитку.

Актуальність роботи. Використання Data Oriented підходу і патерну ECS для написання програмного забезпечення і симуляцій на основі ігрових рушіїв знаходиться на початковому етапі свого розвитку. Більших відповідних продуктів мають закритий код, а технології розробки мало описані. В свою чергу створення оптимізованих і ефективних симуляцій є дуже важливим, так як вони дуже корисні в багатьох сферах, і можуть грати важливу роль у знаходженні шляхів для вирішення або уникнення глобальних кризових ситуацій таких як війни, епідемії і так далі.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає у створенні на основі патерну ECS комп’ютерною симуляції з графічним відображенням моделі попиту пропозиції. Дослідження переваг та недоліків використання Data Oriented підходу і патерну ECS для написання ігрових систем та комп’ютерних симуляцій.

Для досягнення цієї мети поставлено такі завдання.

- Ознайомитись з патерном EQS
- Ознайомитись з Unreal Engine 5 та принциам роботи з ним
- Навчитись роботі з Unreal Engine Game Framework
- Ознайомитись з UnrealEngine Editor.
- Ознайомитись з математичною моделлю попит – пропозиція
- Підібрати технології для реалізації Data Oriented та патерну EQS на

основі Unreal Engine

- Ознайомитись з UE Mass Framework
- Підібрати технології для роботи з даними та їх відображення в Real Time в середовищі Unreal Engine
- Розробити симуляцію на основі патерну ECS
- Провести порівняльний аналіз двох підходів для створення симуляцій.

Об’єкт дослідження

Об’єктом дослідження є використання патерну EQS для створення ефективних та оптимізованих комп’ютерних симуляцій

Предмет дослідження

Предметом дослідження є використання та аналіз існуючих інструментів для створення програмних застосунків на основі Data Oriented підходу та патерну ECS в середовищі ігрового рушія Unreal Engine 5, який буде використовуватись для створення графічної репрезентації розробленої симуляції

Засоби реалізації

Мова програмування C++, скриптова візуальна мова програмування Blueprints, ігровий рушій Unreal Engine 5.1, бібліотеки GameFramework, MassFramework, Kantan Charts.

Можливі сфери застосування. Розроблене програмне забезпечення може використовуватись в освітній та науково – дослідницькій сфері. Дослідження методів розробки симуляцій можуть бути використанні для створення симуляцій які можуть буди корисними в інших сферах діяльності.

1 ПРЕДМЕТНА ОБЛАСТЬ

1.1 Комп’ютерні симуляції

У найвужчому розумінні комп’ютерна симуляція — це програма, яка виконується на комп’ютері та використовує покрокові методи для дослідження приблизної поведінки математичної моделі.[1] Дуже часто симуляція побудова на якійсь моделі яка може бути наявна реального світу

Комп'ютерні симуляції дуже корисні у великій кількості сфер діяльності людини, завдяки тому, що вони дозволяють моделювати, аналізувати та прогнозувати поведінку складних систем. Далі наведені приклади сфер та практичне застосування комп’ютерних симуляцій в них:

-Авіаційна та космічна промисловість: Комп'ютерні симуляції використовуються для моделювання поведінки літальних апаратів під час різних умов польоту, для проектування нових космічних апаратів та для тренування пілотів та астронавтів. Вони використовуються в розробці озброєння повітряного типу, в тому числі систем ППО.

Рисунок 1.1 – Симулятор для тренування пілотів

-Медицина: Симуляції використовуються для моделювати різних біологічні процеси, від взаємодії молекул до роботи органів та організмів. Вони також можуть бути використані для тренування медичного персоналу в навичках, як-от хірургічні операції.

-Фінанси та економіка: Комп'ютерні моделі використовуються для моделювання фінансових ринків, оцінки ризику та прогнозування економічних трендів.

-Виробництво: Симуляції використовуються для проектування та оптимізації виробничих процесів, планування ресурсів та управління ланцюгами постачання.

-Навколишнє середовище: Комп'ютерні моделі допомагають прогнозувати зміни клімату, оцінювати вплив на навколишнє середовище та розробляти стратегії управління ресурсами.

-Освіта та тренування (продовження): Комп'ютерні симуляції використовуються для навчання та тренування в різних областях, від водіння автомобіля до виконання хірургічних операцій.

-Архітектура та містобудування: Симуляції дозволяють архітекторам та містобудівникам візуалізувати свої проекти перед тим, як вони будуть побудовані, та досліджувати можливі впливи нових будівель або розвитку інфраструктури на міські середовища.

-Енергетика: Комп'ютерні симуляції допомагають в аналізі ефективності та безпеки енергетичних систем, включаючи ядерні реактори, вітрові ферми та сонячні панелі.

-Військова сфера: Симуляції використовуються для тренування військових, стратегічного планування, оцінки ризиків, а також для розробки і тестування нових технологій.

-Сфера комп’ютерних ігор: симуляції отримали велику популярність в сфері розробки комп’ютерних ігор, так як дозволяють створювати реалістичні ігрові світи.

Рисунок 1.2 – Гра - симулятор міста Cities:Skylines

Ці приклади показують наскільки комп’ютерні симуляції є ефективними та практично застосовними. Тому їх розвиток почався з появою перших комп’ютерів та продовжується і до нині.

Симуляції будуються на основі математичних моделей. Математична модель - це опис об'єкта, системи, процесу або явища, який використовує математичні змінні та формули. У математичній моделі використовуються символи, щоб представити ключові елементи системи та математичні вирази, щоб описати взаємозв'язок між цими елементами. Для створення моделі можуть використовуватися різні математичні структури, включаючи рівняння, функції, геометричні форми, статистичні розподіли, тощо.

1.2 Математична модель Попит – пропозиція

Попит – значення яке показує готовність покупців отримувати товар у даний момент часу за кожною величиною ціни запропонованою на ринці. На динаміку попиту впливають різного роду цінові і не цінові чинники.[2]

Обсяг попиту залежить від цінових чинників, ця залежність описується законом попиту. Таким чином попит можна представи Кривою попиту – кривою

з негативним нахилом. Вісь X в кривій попиту відповідає за ціну, вісь Y відповідає за готовність купувати цей товар в залежності від ціни на нього.

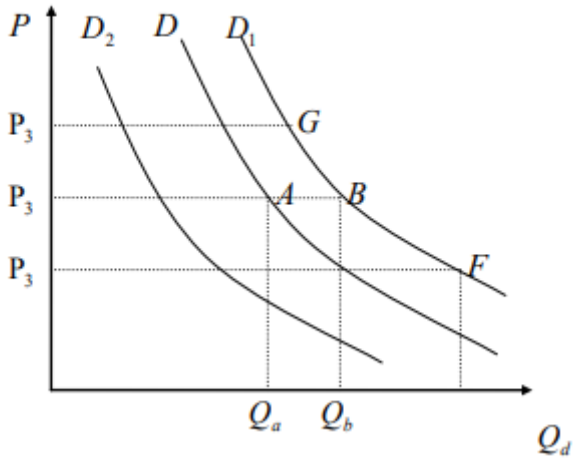


Рисунок 1.3 – Крива попиту

Взаємовідношення між попитом та ціною можуть змінюватися в залежності від різних факторів. Наприклад, за фіксованою ціною P1, споживачі можуть придбати визначену кількість товару Qa (як показано у точці A на кривій попиту D). Але якщо дохід споживачів збільшується, вони можуть дозволити собі придбати більше одиниць товару за ту саму ціну P1, скажімо, Qb. Це відображає зсув кривої попиту вправо. З іншого боку, якщо певні обставини призводять до зменшення кількості товару, яку споживачі хочуть купити (Qd), крива попиту зсувається вліво. Таким чином, весь набір можливих відношень попиту та ціни для даного товару представлений кривою попиту. Коли крива попиту зміщується вправо, це означає, що попит на товар збільшився. Навпаки, коли крива попиту зсувається вліво, це свідчить про зменшення попиту.[3]

Три ключові фактори впливають на зворотній взаємозв'язок між динамікою попиту та рівнем цін. Перш за все, коли ціни знижуються, більше людей може придбати товар. За друге, зниження цін також посилює покупну спроможність споживачів. Нарешті, насичення ринку може привести до зниження цінності додаткових одиниць продукції, тому споживачі більше схильні до покупки за нижчими цінами.

Цінові фактори можуть призвести до змін у попиті, зміщуючи його вище

постійної кривої попиту. Однак, попиту також впливають нематеріальні фактори, такі як смаки споживачів, розмір ринку, дохід (так званий "ефект доходу"), ціни на подібні товари, а також очікування споживачів.

Вплив нематеріальних факторів веде до змін у обсязі попиту, що проявляється у зсуві кривої попиту вправо (якщо попит зростає) та вліво (якщо попит знижується). "Зміна попиту", що відображається у зсуві кривої попиту вліво або вправо, відрізняється від "зміни кількості попиту", що означає переміщення вздовж кривої попиту вгору або вниз. При цьому сам попит не змінюється, а змінюється лиш е Qd, або кількість товару, на який попит за конкретною ціною.

Пропозиція відображає кількість товару, яку виробники реалізують на ринку. Пропозиція на товар може бути представлена через відповідну шкалу, яка відображає різні обсяги товару, який виробник хоче виготовити та продати за кожну ціну протягом визначеного періоду часу.

Закон пропозиції описує залежність обсягу пропозиції від ціни і представляється у вигляді кривої з позитивним нахилом. Ця крива пропозиції показує кількість товару або послуги, яку продавці намагаються продати за різними цінами впродовж визначеного періоду.

Фактори, які впливають на обсяг пропозиції товару на ринку, можуть зміщувати криву пропозиції вправо при збільшенні обсягу пропонованого товару, або вліво при його зменшенні. Крива S1 символізує збільшення пропозиції, в той час як крива S2 відображає її зменшення. Зсув кривої пропозиції вправо означає, що пропозиція змінюється при постійній ціні, якщо ж жоден з факторів, що впливають на пропозицію, не змінюється, а ціна збільшується, то відбувається зміщення вздовж кривої пропозиції вгору або вниз (при цьому сама пропозиція товару не змінюється).

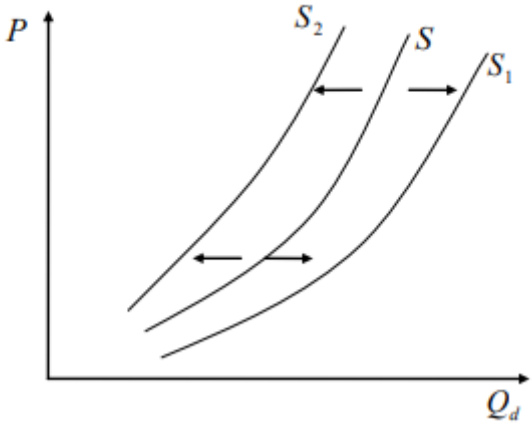


Рисунок 1.4 – Крива пропозиції

Прямий взаємозв'язок між пропозицією і рівнем цін є наслідком дії закону зменшення продуктивності виробничих факторів. Коли ціни змінюються, це впливає на величину пропозиції, і це проявляється у русі вздовж стабільної кривої пропозиції від одного набору "ціна - обсяг" до іншого.

Дія факторів, непов'язаних з ціною, призводить до зміни обсягу пропозиції, що виражається у зсуві кривої пропозиції вправо (при її зростанні) або вліво (при її зменшенні). Ці нецінові фактори, які впливають на пропозицію, включають в себе ціни на виробничі ресурси, зміни у технологіях виробництва, політику держави в області оподаткування та субсидування, а також структуру ринку.

Ринкова рівновага - це стан на ринку, в якому обсяг пропозиції дорівнює обсягу попиту. Це поняття є ключовим для вивчення економіки, оскільки відображає ситуацію, коли ринок функціонує ефективно, без втручання з боку держави чи інших зовнішніх сил.

У стані ринкової рівноваги ціна, за яку товар продається, є такою, що кількість товару, яку виробники хочуть продати, дорівнює кількості товару, яку споживачі хочуть купити. Така ціна називається ціною рівноваги, а відповідний до неї обсяг продажу - обсягом рівноваги.

Ринкова рівновага важлива для забезпечення ефективності ринку. Коли ринок не перебуває в стані рівноваги, це може призвести до витрат або неефективності. Наприклад, якщо обсяг пропозиції перевищує обсяг попиту (відбувається перевищення пропозиції), то товари можуть залишатися

непроданими, що є витратами для виробників. Якщо ж обсяг попиту перевищує обсяг пропозиції (відбувається дефіцит), споживачі можуть бути незадоволеними, що також є неефективністю.

Ринкова рівновага може змінюватися під впливом різних факторів, таких як зміна вартості сировини, зміна споживчих смаків і т.д.

Важливо зрозуміти, що ринкова рівновага - це динамічне поняття. Зміни у зовнішніх факторах, таких як вартість сировини, споживчі смаки або технології, можуть змінити пропозицію і попит, що в свою чергу призведе до зміни ціни і обсягу рівноваги.

Водночас, ринок не завжди може ефективно дійти до стану рівноваги самостійно. В таких випадках держава може втручатися, встановлюючи мінімальні або максимальні ціни, надаючи субсидії виробникам або застосовуючи інші регулюючі заходи.

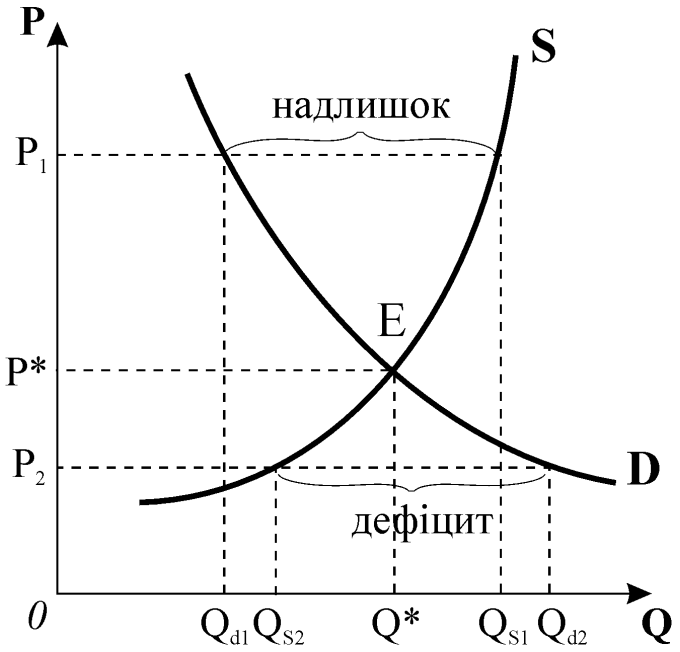


Рисунок 1.5 – Ринкова рівновага

1.3 Entity – Component - System

Entity-Component-System (ECS) є альтернативою традиційній парадигмі

об'єктно-орієнтованого програмування (ООП), де кожен об'єкт є інстанцією класу, і наслідує всі властивості та функціонал класу. В ECS, вся логіка і дані розділені, що дає розробникам більше гнучкості і контролю.[4]

Суть патерну ECS: Одна сутність охоплює кілька доменів. Щоб зберегти домени ізольованими, код для кожного розміщується у власному класі компонентів. Сутність зводиться до простого контейнера компонентів

Структурні елементи Entety – Components – System патерну:

- Об'єкти (Entities): У контексті ECS, об'єкт не містить ніяких даних і не має жодної поведінки. Це лише унікальний ідентифікатор, який служить як посилання на пов'язані з ним компоненти.

- Елементи (Components): В ECS, елементи є базовими структурами даних, які зберігають специфічну для об'єкта інформацію. За ілюстрацією, елемент "Фізика" може містити такі параметри, як маса, швидкість і сила. Важливо зазначити, що елементи є "пасивними" - вони не мають власного коду поведінки і служать лише для зберігання даних.

- Модулі (Systems): У ECS, модулі - це те, що забезпечує виконання роботи. Вони включають в себе логіку, що обробляє дані елементів. Модулі працюють над елементами, які асоціюються з певними об'єктами. Наприклад, модуль "Фізична Система" може оновлювати позиції об'єктів на основі їх швидкості і сили.

Компонентний патерн може значно ускладнити структуру класу та додати складності у розташуванні коду всередині. Кожен "об'єкт" з погляду концепції перетворюється на сукупність об'єктів, які потрібно генерувати, ініціалізувати та коректно інтегрувати. Взаємодія між відмінними компонентами стає більш складною, а керування їх використанням пам'яті стає викликом. Для обширної кодової бази, ця складність може бути виправданою, оскільки вона сприяє роз'єднанню та повторному використанню коду, але важливо переконатися, що ви не переосмислюєте "рішення" для проблеми, яка насправді не існує, перед впровадженням цього патерну. Ще одним результатом використання

компонентів є необхідність часто обходити рівень посередництва, щоб виконати дію. Коли ви розглядаєте контейнерний об'єкт, ви спочатку повинні знайти відповідний компонент, а потім виконати потрібну дію. У випадках, коли продуктивність є критичною, ця додаткова посередницька вказівка може призвести до падіння ефективності.

Компоненти зазвичай містяться в основному класі, який представляє сутності у грі, хоча вони можуть бути корисними й в інших контекстах. Цю архітектурну модель може бути ефективно використано у випадках, коли:

- Вам потрібно розділити клас, який включає в себе декілька доменів, які ви бажаєте ізолювати від інших.
- Клас стає надто великим і складним для управління та розробки.
- Ви хочете мати змогу створювати об'єкти з різними можливостями. Використання успадкування може не дозволити вам достатньо гнучко вибирати функціональність для повторного використання.

У висновку можна зазначити що Entity-Component-System (ECS) представляє собою архітектурний підхід у галузі розробки ігор, який передбачає розробку сутностей за допомогою компонентів, що містять тільки дані, а обробку логіки виконують окремі системи. При використанні такого підходу до архітектури можна стати занадто зосередженим на швидкості та ефективності. І, не те щоб це було неправильно - це дійсно важливо. Але це не повинно стати головною ціллю, особливо коли мова йде про розробку невеликих ігор. У пошуку найкращої продуктивності можна ненавмисно створити надто складну систему, що, насправді, не спрощує процес розробки.[5]

2 ВИКОРИСТОВУВАНІ ПРОГРАМНІ ЗАСОБИ ТА ТЕХНОЛОГІЇ

2.1 Мова програмування C++

Мова програмування C++ є однією з найпоширеніших та впливових мов програмування в світі. Перша версія C++ була випущена о 1985 році і її розробником був Датський вчений та програміст Бйорн Страуструп. C++ був створений як розширення мови програмування C. Назва "C++" є свого роду грою слів: в мові C, "++" - це оператор інкременту, який збільшує значення змінної. Таким чином, "C++" можна розглядати як "C збільшене" або "C + 1".

C++ є статично типізованою, компільованою мовою програмування, що підтримує багато парадигм програмування, включаючи процедурне, об'єктно-орієнтоване та загальне програмування. C++ став створений на основі мови програмування C, в C++ з’явились такі концепції як класи та об'єкти, таким чином C++ стала однією з перших мов програмування яка повністю підтримувала об’єктно орієнтовний підхід до програмування і позволяла реалізовувати такі важливі концепції і принципи ООП як наслідування, поліморфізм, абстрактні типи даних, і інкапсуляцію.

Але C++ дозволяє використовувати не лише об’єктно орієнтований підхід до програмування, але і включає такі парадигми як процедурне програмування та шаблонне програмування.

Важливою особливістю мови C++ - це її стандартизація та зворотна сумісність з попередніми стандартами. Загалом з часу створення вийшли такі стандарти: C++ 98, C++ 03, C++ 11, C++ 17, C++ 20, які покращували та адаптували мову для сучасних потреб розробників в цей же час зберігаючи зворотну сумісність з попередніми версіями стандарту.

Важливою особливістю C++ є те, що вона дозволяє програмістом мати прямий доступ до пам’яті, оперувати і працювати з нею на низькому рівні, що часто є необхідним для створення високоефективного та оптимізованого програмного забезпечення, а також позволяє розробляти специфічне програмне

забезпечення для створення якого необхідний такий функціонал мови програмування. Через цю свою особливість C++ став популярним інструментом у руках багатьох програмістів в різних сферах розробки, а особливо в сфері системного програмування для розробки операційних систем, компіляторів та іншого низькорівневого програмного забезпечення. Також ця мова використовується для розробки драйверів, вбудованих систем, високонагружених систем, важких десктопних застосунків-інструментів таких як 3д редактори, редактори зображень, інтегровані середовища розробки та інше.

Також C++ став дуже популярним інструментом у розробників комп'ютерних ігор так як є дуже ефективним для створення систем рендеру та дозволяє писати оптимізовану ігрову логіку без зайвого оверхеду накладеного мовою програмування, що є дуже важливим в цій сфері. C++ використовується в тому числі і для розробки програм на основі ігрового рушія Unreal Engine який буде використовуватись в цій дипломній роботі для створення графічної репрезентації створеної симуляції.

Одночасно з усіма плюсами і перевагами мови C++ вона має і ряд недоліків які не можна ігнорувати. Один з них – це складність написання коду, мова C++ є досить важкою для засвоєння і вивчення, тому необхідно витратити досить велику кількість часу та сил на оволодіння нею. Також написання коду на C++ часто займає більше часу ніж на інших мовах програмування, що може стати причиною того, що команда розробників буде використовувати іншу мову програмування для створення продукту, так як часто для бізнесу більш пріорітетним є зменшення часу розробки ніж оптимізації. Якщо для продукту є не критичним оверхенд накладений іншою мовою програмування який збільшить час роботи програми, то економічно вигіднішим буде використання більш ефективної з точки зору швидкості написання коду мови програмування.

2.2 Unreal Engine

Unreal Engine - це ігровий рушій розроблений компанією Epic Games на основі мови C++, який включає широкий вибір інструментів для створення ігор різного масштабу і формату, від малих ігор формату Інді до великих проектів рівня AAA. Це один з найбільш поширених та потужних інструментів в своїй сфері, який використовується великою кількістю компаній і розробників. Він дозволяє створювати ігри будь якого жанру та направленості. За допомогою UE були розроблені такі ігри:S.T.A.L.K.E.R 2, Fortnite, Sea Of Thieves, Life is Strange 2, Star Wars Jedi: Fallen Order, Valorant

Unreal Engine надає розробникам широкий спектр можливостей та засобів для роботи з комп'ютерною графікою, дозволяє налаштовувати світло, додавати матеріали і шейдери для створення візуального компоненту та стилізації комп'ютерної графіки гри, дозволяє розробникам та дизайнерам налаштовувати конфігурації рендеру під потреби проекту. Він включає в себе повний набір інструментів для розробки будь якої системи необхідної у розробці комп'ютерних ігор таких як: рафіка, звук, інтерфейс користувача, скрипти гри, штучний інтелект, мережевий коду, анімації. За допомогою Unreal Engine можна створювати ігри під різні платформи. Unreal Engine надає інструментарій для створення ігор VR та AR. Також дозволяє розробникам створювати свої інструмента та поширювати їх на маркетплейсі Epic Store.

Ігровий рушій Unreal Engine має довгу історію, протягом часу він кардинально змінювався і привносив велику кількість інновацій в технології розробки комп'ютерних ігор. Розробники цього рушія зробили величезний внесок у розвиток комп'ютерної графіки та дуже сильно вплинув на ігрову індустрію в цілому.

Історія Unreal Engine починається о 1998 році, саме тоді було представлено першу версію цього ігрового рушія Unreal Engine 1. Вона була представлена разом з грою Unreal. Цей двіжок був одним з перших, що пропонував повноцінні 3D-середовища з детальною графікою та високоякісною фізикою. Для створення

геймплею в ньому використовувалась скриптової мова програмування UnrealScript. У 2002 році був представлений Unreal Engine 2, який вніс покращення у графіку та підтримку новіших платформ, таких як Xbox. Unreal Engine 3 вийшов о 2006 році, і став основним інструментом розробки для багатьох команд програмістів на наступне десятиліття. З нововведень цієї версії рушія можна виділити підтримку шейдерів, високоякісну графіку та впровадження редактора рівнів UnrealEd. Наступним етапом розвитку рушія став випуск Unreal Engine 4, який приніс великі зміни у експлуатацію рушія та процес розробки ігор на його основі. Була представлена нова візуальна скриптова мова програмування Blueprints, яка стала чудовим інструментом для створення прототипів, експериментів в розробці, та високорівневої розробки гри. Наступним оновленням став Unreal Engine 5, остання і найбільш актуальна на даний момент часу версія цього ігрового рушія яка потрапила у відкритий доступ у 2021 році. Приніс нові технології: Nanite, що дозволяє створювати надзвичайно детальні та реалістичні сцени з мікрополігонами; та Lumen, систему динамічного глобального освітлення в реальному часі.

Unreal Engine є абсолютно безкоштовним для використання в некомерційних цілях: освітніх цілях, розробки застосунків які будуть розповсюджуватись вами безкоштовно. Для комерційних проектів треба заплатити 5% роялті від ваших валових доходів після того, як ваші доходи перевищать 1 мільйон доларів. –посилання

Для розробки використовується мова програмування C++ та візуальна скриптова мова програмування Blueprint Visual Scripting.

Тобто можна зробити висновок, що Unreal Engine це потужний інструмент, який чудово підходить як для невеликих навчальних проектів так і для розробки великих комп’ютерних ігор і створення складних симуляцій. Він надає можливість зробити графічну репрезентацію симуляції та налаштувати її під конкретні потреби. Тому Unreal Engine є чудовим вибором для реалізації цілей які поставлені в даній дипломній роботі.

2.3 Unreal Engine Gameplay Framework

Бібліотека яка включає в себе набір класів для розробки геймплею логіки ігор та інших застосунків на основі ігрового рушія Unreal Engine. Ця бібліотека – один з основних інструментів при розробці на Unreal Engine, яка дозволяє ефективно створювати гарно структуровані ігри на основі класів з розподіленою логікою для всіх аспектів гри. Ось деякі основні клас з Unreal Engine Gameplay Framework та їх взаємозв'язок:

- Actor: це клас, який визначає об'єкти як можуть бути розміщені на рівні в ігровому світі, в результаті це можуть бути об'єкти оточення, персонажі гри, вороги. Поведінку кожного Actror визначає набір компонентів які він має та його внутрішня логіка
- Pawn: це Actor який може бути ігровим агентом в світі. Pawn може управлятись PlayerController та може приймоти інпут ві ігрока. Так павни можуть управлятись AIController у випадку, коли поведінка ігрового агента має визначатись алгоритмами штучного інтелекту
- Character: це надбудова на Pawn для хуманоїдних ігрових агентів. Має CharacterMovementComponent за замовчення, який дозволяє налаштувати логіку та правила переміщення ігрового агента в світі.
- PlayerController: клас який зв'язує ігрока та ігрового агента в світі гри. Робить він це інтерпретуючи інпут ігрока, на основі якого визначається поведінка ігрового агента
- AIController: визначає і контролює поведінку яка має контролюватись алгоритмами штучного інтелекту, які можуть бути визначені за допомогою методу BehaviourTree. StateTree та інші.
- HUD: відповідальний за відображення користувацького інтерфейсу на екрані ігрока, та логіку поведінки віджетів які його утворюють
- GameMode: клас відповідальний за визначення правил ігрового світу, відслідковує та управляє загальним станом гри. Також гейм мод контролює процес приєднання ігроків до онлайн сесії у випадку

багатокористувацької гри.

- GameState: клас який відповідає за збереження у собі інформації про поточний стан гри.
- PlayerState: клас який відповідає за збереження у собі потоочного стану конкретного ігрока[6]

Наступна діаграма демонструє, як базові класи геймплею взаємодіють між собою. Кожна гра формується з двох основних компонентів - GameMode та GameState. Коли людські гравці долучаються до гри, вони пов'язані зі своїми відповідними PlayerControllers. Ці PlayerControllers надають можливість гравцям "володіти" пішаками (Pawn) у грі, давати їм фізичне відображення на сцені. PlayerControllers також відповідають за обробку введення даних від гравця, надають інтерфейс головного дисплея (HUD) та керують PlayerCameraManager, який відповідає за переключення переглядів камери.

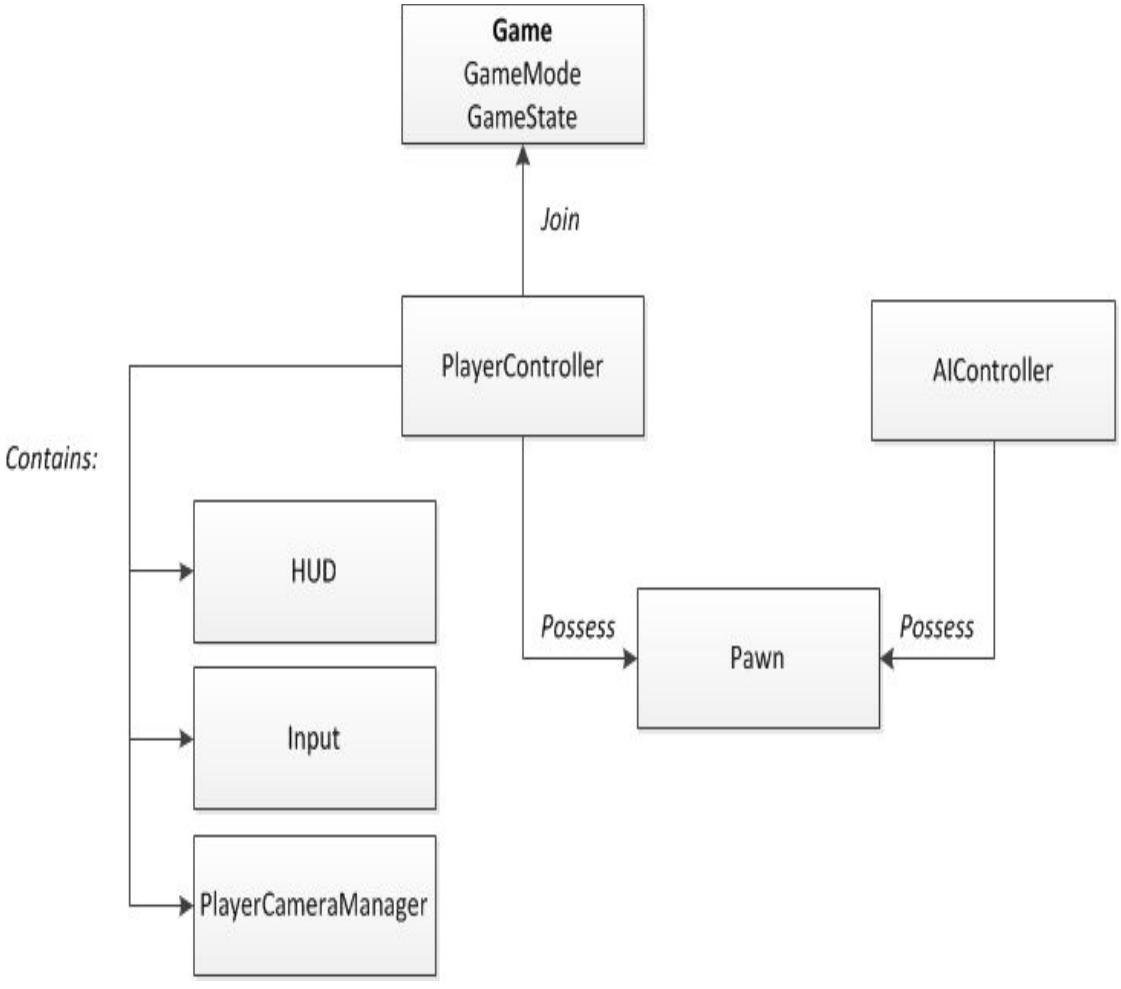


Рисунок 2.1 – Gameplay Framework

2.4 MassFramework. MassEntity

Data-Oriented Design (DOD) в програмуванні полягає в організації структури програми та її даних з огляду на ефективність використання апаратного забезпечення, зокрема пам'яті та кешу процесора.

Цей підхід змінює спосіб мислення про програмування, ставлячи дані і операції з ними в центр практик програмування. Він перестає сприймати програму як набір об'єктів, які взаємодіють між собою (як у об'єктно-орієнтованому програмуванні), і ставиться до неї як до процесу обробки даних.

В DOD більше уваги приділяється взаємодії даних із пам'яттю та кешем процесора, що може привести до значного покращення продуктивності програми. Основні принципи DOD включають організацію даних у послідовні блоки пам'яті, що сприяє більш ефективному використанню кешу процесора завдяки зменшенню кількості кешмісів; мінімізацію використання вказівників, що можуть призвести до проблем з кешуванням; і відкладання обчислень до моменту, коли це дійсно необхідно, що зменшує непотрібну роботу.

MassFramework – це фреймворк для Unreal Engine який надає можливість писати код заснований на Data Oriented підході.

MassEntity – плагін який є ядром MassFramework

Далі хотів би описати структуру даного фреймворка та навести опис деяких класів які його формують.

Почати треба з опису організації таких структурних елементів як Entity та Archetypes

- FMassFragment: легковажні фрагменти які зберігають інформацію про для кожного ентіті.
- FMassTag: типи, які перевірятимуться лише на наявність та відсутність. Підкласи ніколи не повинні містити властивості членів. Наприклад : FMassCrowdTag, FMassDebuggableTag. У майбутньому можуть бути перероблені на формат «тег для архетипу», а не для ентіті.
- FMassChunkFragmets: дані які відносяться до чанків. Наприклад:

FMassVisualizationChunkFragment. Використовуються для індикації того чи чанк має в собі видимі ентіті

- FMassSharedFragment: спільні дані, які належать усім сутностям поточного блоку Archetype. [7]

Усі вищезазначені є USTRUCT(), а не UCLASS(), оскільки вони не призначені для GC, Archetype самостійно керує пам'яттю.

Для керування сутностями існує структура FMassEntityHandle. Він використовується як полегшений хендл для сутності в Mass Framework.

```
USTRUCT()
struct FMassEntityHandle
{
    // Other functions
    UPROPERTY(VisibleAnywhere, Category = "Mass|Debug", Transient)
    int32 Index = 0;

    UPROPERTY(VisibleAnywhere, Category = "Mass|Debug", Transient)
    int32 SerialNumber = 0;
}
```

Рисунок 2.2 – FmassEntityHandle

Index тут є унікальним серед усіх сутностей у FMassEntityManager, але не послідовними та зростаючими: індекси вилучених сутностей додаються до EntityFreeIndexList і пізніше вириваються для новостворених сутностей. Цей індекс НЕ вказує на індекс сутності всередині Архетипу.

SerialNumber – це унікальний ідентифікатор, який завжди зростає. Дві сутності не можуть мати однаковий серійний номер, якщо він не переповнюється.

Однією з основних частин Archetype, яка використовується для диференціації Архетипів, є FMassArchetypeCompositionDescriptor. Структура описує склад сутності або архетипу. Він містить інформацію як про фрагменти, так і про теги без дублікатів. Порядок не має значення, буде лише один дескриптор (як і архетип) на унікальний набір типів фрагментів на підсистему менеджера об'єктів.

Усі операції з сутністю, такі як створення, видалення, зміна складу, внутрішньо керуються FMassArchetypeData. Це структура для архетипу, який

визначається набором унікальних типів фрагментів (без дублікатів)

Архетипи надають зручний API для керування сутностями, як окремими, так і пакетними: додавання, видалення, встановлення даних фрагментів, додавання та видалення фрагментів/тегів (це призводить до переміщення сутності до іншого архетипу). Для передачі архетипів функціям або іншим класам/структурам існує легка структура FMassArchetypeHandle final, яка містить лише TSharedPtr<FMassArchetypeData>. Функції архетипів не повинні мати прямого доступу в класах користувачів, уся робота з сутностями повинна виконуватися через FMassEntityManager.

Фрагменти зберігаються в пам’яті необробленої купи, яку зберігає FMassArchetypeChunk. Розмір цієї пам’яті визначається AllocSize, за замовчуванням це 128*1024, це число вибрано через розмір кешу L2, який може становити від 128 КБ до 8 МБ, тому один шматок точно поміститься в лінію кешу L2

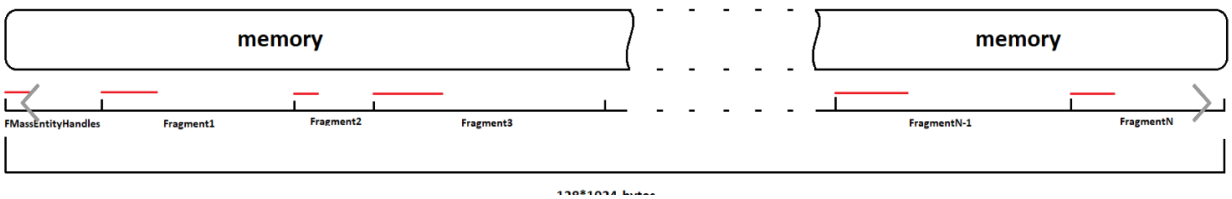


Рисунок – 2.3 Internal memory layout

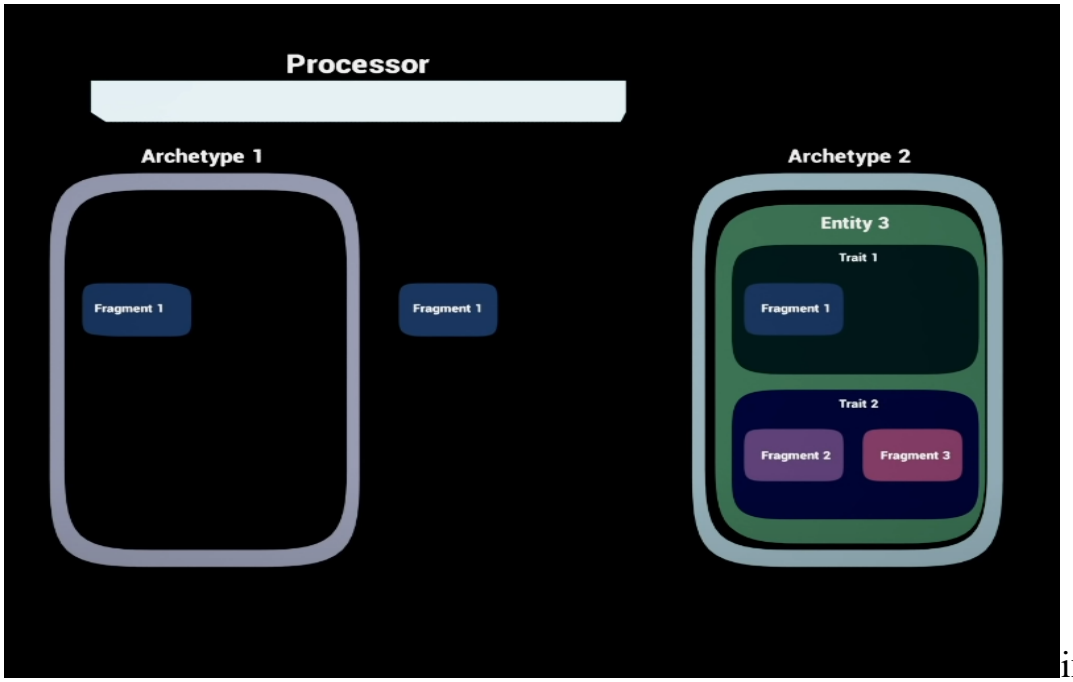
Всередині RawMemory є підмасиви фрагментів необробленої пам'яті. Кількість підмасивів дорівнює кількості фрагментів плюс один масив для хендлеів сутності. На основі розміру всіх фрагментів (зберігаються в ChunkFragmentData) і розміру дескриптора обчислюється максимальна кількість об’єктів, і кожен підмасив містить елементи MaxNumberOfEntities. Наприклад, на зображенні вище показано пам'ять блоку, який містить N фрагментів. Місце в пам'яті для кожного фрагмента визначається під час створення блоку. Червона лінія позначає пам’ять, яка зберігає дійсні фрагменти для сутностей, у випадку, коли NumInstances = 1/3 від MaxNumberOfEntities. Оскільки вся пам’ять виділяється під час створення та звільняється під час знищення, розмір цієї пам’яті не змінюватиметься. Додавання нових сутностей — це лише збільшення

NumInstances, видалення сутностей — заміна видалених сутностей останніми, а потім зменшення NumInstances.

Процесори реалізують в собі логіку ігрового процесу. UMassProcessor є базовим абстрактним класом для всіх процесорів. У MassFramework існує кілька типів процесорів

- UMassProcessor та похідні класи. Звичайний тип процесора, що відповідає за виконання корисних речей щотику.
- UMassCompositeProcessor та похідні класи. Він відповідає за створення та керування пайплайном виконання інших процесорів.
- UMassObserverProcessor та похідні класи. Він відповідає за одноразове виконання на основі певної події в MassFramework.

Процесори не зберігають жодних даних ігрового процесу або даних, специфічних для певної сутності чи системи, тому їх можна вважати такими, що не мають стану. У цьому випадку нам не потрібно самостійно створювати екземпляри UMassProcessor, усю роботу може (і справді виконує) CDO(class default object).



ів

Рисунок 2.4 – Взаємозв’язок структурних елементів MassFramework

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Алгоритм симуляції

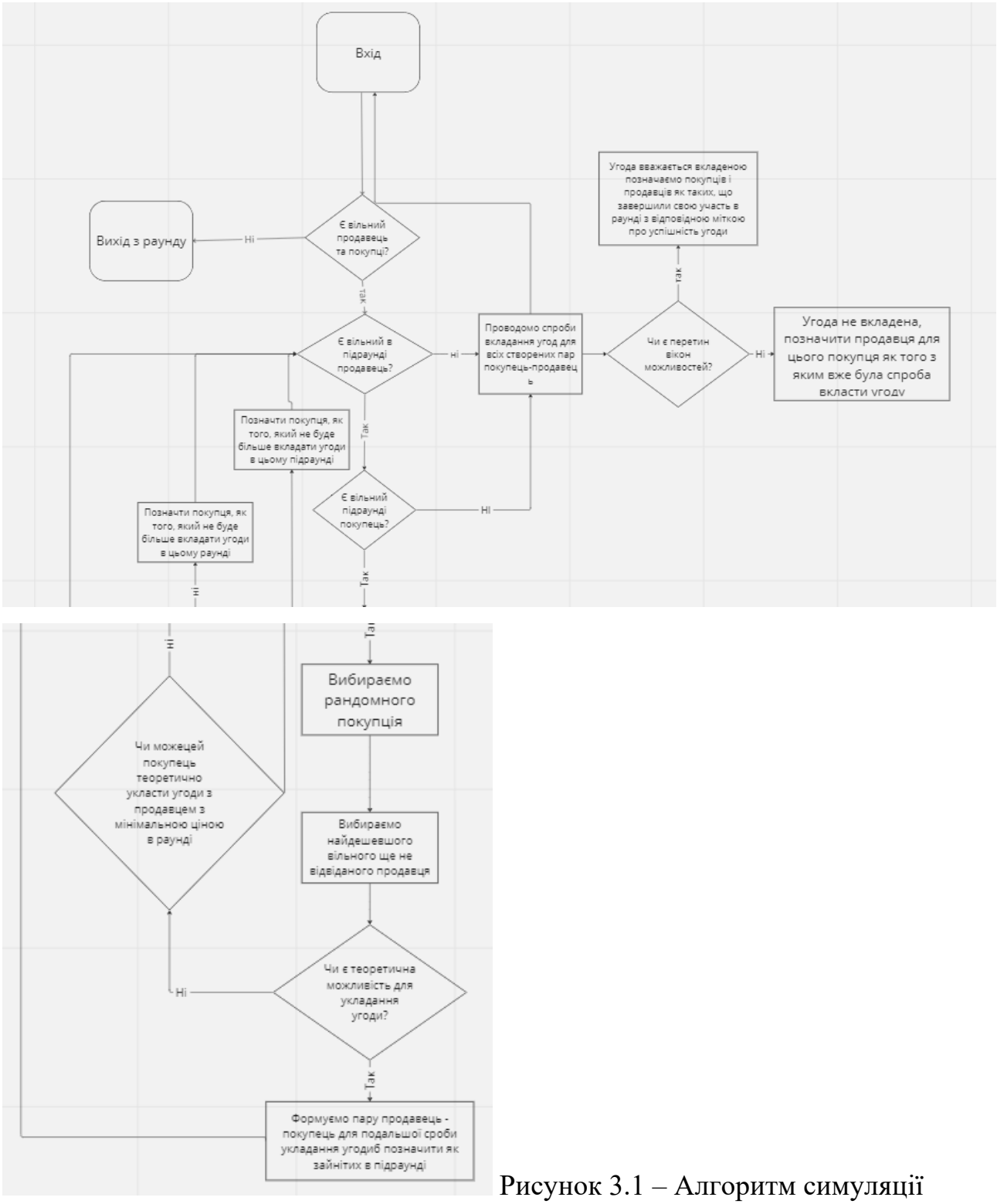


Рисунок 3.1 – Алгоритм симуляції

Алгоритм полягає в тому що в нас є два види сутностей: покупці та продавці, які намагаються купити та продати товар і корегують свої пропозиції ціни щодо покупки для отримання максимальної вигоди для себе. Покупці мають такі поля: Максимальна ціна за яку вони готові купити товар, Пропозиція в поточному раунді, вікно для торгівлі. Продавці мають такі поля: Мінімальна ціна за яку вони готові продати товар, пропозиція в поточному раунді, вікно для торгівлі.

Симуляція утворена послідовністю раундів. Кожен з раундів складається з підраундів. Протягом раунду всі покупці спробують укласти угоди з продавцями, і в результаті зроблять це успішно або угода не буде вкладена. Угода може бути успішною, якщо існує перетин вікон можливостей для торгівля як продавця так і покупця. Після завершення раунду всі продавці та покупці зроблять корегування своїх пропозицій для наступного раунду. У випадку Покупця, якщо той зміг вкласти угоду, та успішно купив товар, то на наступний раунд він зменшить свою пропозицію, щоб спробувати купити товар по більш вигідній ціні. Якщо ж він не зміг купити товар, то на наступний раунд він збільшить свою пропозицію, щоб збільшити свої шанси на покупку необхідного йому продукту. Для випадку з продавцем ситуація буде така – якщо він не зміг реалізувати свій товар, то на наступний раунд він зменшить ціну на нього, якщо ж він його продав протягом раунду, то на наступний раз він підвищить ціну і спробує отримати більший прибуток ніж в минулий раз.

Підраунди є структурною частиною раунду, під час яких відбуваються намагання утворити пари покупець-продавець.

3.2 Реалізація на основі ECS

Враховуючи особливості Data Oriented архітектури програми заснованій на патерні ECS було створено три класи – фрагменти, які зберігають дані про ігрових агентів, які були поділені між фрагментами зважаючи на частоту та необхідність їх використання.

```
USTRUCT()
struct MARKETSIMULATION_API FECS_PriceFragment : public
FMassFragment
{
    GENERATED_BODY()

    int32 CurrentPrice;
    int32 MarginalPrice;
};

USTRUCT()
struct MARKETSIMULATION_API FECS_ProposalShiftFragment :
public FMassSharedFragment
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere)
    int32 Shift;
};

USTRUCT()
struct MARKETSIMULATION_API
FECS_OpportunityWindowFragment : public
FMassSharedFragment
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere)
    int32 Window;
};
```

- FECS_PriceFragment: фрагмент який відповідає за збереження даних про поточну та «Максимальну ціну покупки» чи «мінімальну ціну продажі» для покупця та продавця відповідно.
- FECS_ProposalShiftFragment: фрагмент який відповідає за збереження даних про те наскільки буде скорегована пропозиція для наступного раунду
- FECS_OpportunityWindowFragment: фрагмент який відповідає за збереження даних про величину вікна можливостей для торгів.

```
    USTRUCT()
```

```
struct MARKETSIMULATION_API FECS_AgentAvailableInRoundTag
: public FMassTag
{
    GENERATED_BODY()
};

USTRUCT()
struct MARKETSIMULATION_API
FECS_AgentAvailableInSubroundTag : public FMassTag
{
    GENERATED_BODY()
};

USTRUCT()
struct MARKETSIMULATION_API FECS_AgentInPairTag : public
FMassTag
{
    GENERATED_BODY()
};

USTRUCT()
struct MARKETSIMULATION_API FECS_AgentTradedTag : public
FMassTag
{
    GENERATED_BODY()
};

USTRUCT()
struct MARKETSIMULATION_API
FECS_AgentNotAvailableInRoundTag : public FMassTag
```

```
{
    GENERATED_BODY()
};

USTRUCT()
struct MARKETSIMULATION_API
FECS_AgentNotAvailableInSubroundTag : public FMassTag
{
    GENERATED_BODY()
};

USTRUCT()
struct MARKETSIMULATION_API FECS_TradedLastRoundTag :
public FMassTag
{
    GENERATED_BODY()

};
USTRUCT()
struct MARKETSIMULATION_API FECS_MarketAgentTag : public
FMassTag
{
    GENERATED_BODY()

};

USTRUCT()
struct MARKETSIMULATION_API FECS_SellerTag : public
FMassTag
{
```

```
    GENERATED_BODY()
};

USTRUCT()
struct MARKETSIMULATION_API FECS_BuyerTag : public
FMassTag
{
    GENERATED_BODY()
};
```

В цьому фрагменті коду наведений список визначених тегів. Теги які ім’я яких починаються на слово Agent є взаємовиключними, тобто ентіті може мати лише один з цих тегів в один конкретний момент часу.

- FECS_AgentAvailableInRoundTag: тег який познає агента як такого, який готовий брати участь в раунді
- FECS_AgentAvailableInSubroundTag: тег який познає агента як такого, який готовий брати участь в підраунді
- FECS_AgentInPairTag: тег який познає агента як такого для якого сформована пара в якій він буде проводити спробу утворення угоди
- FECS_AgentTradedTag: тег який познає агента як такого,який успішно уклав угоду в раунді, а отже більше не буде брати участь в інших підраундах цього раунда
- FECS_AgentNotAvailableInRoundTag: тег який познає агента як такого, який більше не зможе брати участь в наступних підраундах поточного раунда
- FECS_AgentNotAvailableInSubroundTag: тег який познає агента як такого, який більше не бере участь в цьому підраунді, але буде брати участь в наступних
- FECS_TradedLastRoundTag: тег який познає агента як такого, який мав успішну спробу укладання угоди, а отже для нього треба провести процес

корегування пропозиції на новий раунд

- FECS_MarketAgentTag: тег який позначає агента як такого, який бере участь в симуляції ринкового процесу
- FECS_SellerTag: тег для конкретизації ролі агента, як учасника ринкового процесу, позначає агента як того, що має тип «Продавець»
- FECS_BuyerTag: тег для конкретизації ролі агента, як учасника ринкового процесу, позначає агента як того, що має тип «Покупець»

Для керування процесом торгівлі, початком та завершенням раундів був створений спеціальний клас UECS_MarketTradeSubsystem який наслідується від UWorldSubsystem. Він є також несе відповільність за зв’язування фреймворку UE та UE Gameplay Framework разом з Mass Framework.

За логіку проведення торгів відповідають наведені нижче класи процесорів:

- UECS_StartRoundListener: для кожного агента проводить корегування значеннння його пропозиції відповідно до успішності результату проведення торгів у минуло раунді.
- UECS_EndRoundListener: для кожного агента очизає спецефічні для раунда теги(ті, які починаються на Agent) та накладає тег на агентів які провели успішні торги в цьому раунді.
- UECS_StartSubroundListener: для кожного ентіті покупців, які не торгували в цьому раунді намагається знайти можливого продавця - кандидата в пари для подальшої спроби вкладання угоди. Якщо такого кандидата не знайдено, то перевіряється можливість покупця провести угоду в цьому раунді взагалі, якщо він не може цього зробити, то він перестає брати участь в торгівлі в поточному раунді.
- UECS_EndSubroundListener: для ентіті, які не провели торги , але мають мможливість це зробити, запускає новий підраунд, якщо таких ентіті покупців більше не має, то завершує поточний раунд.

- UECS_MoveToTargetLocationProcessor: відповідає за відслідковування переміщення ігрових агентів для того щоб в момент коли все агенти завершать своє переміщення в сабраунді передати управління процесору UECS_TradeListener
- UECS_TradeListener: відповідає безпосередньо за процес вуладання угод для всіх сформованих пар покупець-продавецью Угода вважається успішною якщо для покупця та продавця з однієї пари наявний перетин вікон можливостей. Якщо угода укладена, то в такому випадку на покупця та продавця накладається тег FECS_AgentTradedTag і ці агенти більше не будуть брати участь в процесі торгівлі в поточному раунді.

UECS_MarketTradeSubsystem може відреагувати на інпут гравця запустити процес симуляції. Використовуючи вбудовану систему передачі сигналів посилає сигнал початку раунда UE::Mass::Signals::StartRound який очікується процесором UECS_StartRoundListener, який отримавши його починає свою роботу. Після виконання попередньої логіки він посилає сигнал UE::Mass::Signals::StartSubround, який очікується процесором UECS_StartSubroundListener, що відовідає за формування пар покупець – продавець. UECS_StartSubroundListener після формування пар, для покупців з якими були утворені пари додається фрагмент FECS_MoveTargetLocationFragment, що стає причиною того, що ця ентіті автоматично почне оброблятися UMassSimpleMovementProcessor, а паралельно з цим ентіті підхоплюється процемсором UECS_MoveToTargetLocationProcessor, який відслідковує рух всіх ентіті: коли ентіті перебуватиме в допустимому радіусі до продавця, то її рух зупинятиметься і вона перейде в стан очікування завершення руху інших ентіті покупців. Коли всі покупці завершать свій рук процесор UECS_MoveToTargetLocationProcessor пошле сигнал UE::Mass::Signals::StartTrade, який очікується процесором

UECS_TradeListener. UECS_TradeListener проводить торги для утворених пар. Після цього пошле сигнал UE::Mass::Signals::EndSubround, який очікується процесором UECS_EndSubroundListener. Який в результаті перевірок поточного стану системи пошле або сигнал UE::Mass::Signals::StartSubround, або UE::Mass::Signals::EndRound. UE::Mass::Signals::EndRound очікується процесором UECS_EndRoundListener, який завершує раунд та передає керування назад до підсестеми, яка в свою чергу або продовжить симуляцію, якщо це передбачається логікою, або увійде в стан очікування нового інпута від гравця.

3.3 Демонстрація

На наступному зображені буде продемонстрований ігровий агент покупця. Жовта шкала відповідає за максимальну ціну за яку він готовий купити товар. Біла шкала – його пропозиція в поточному раунді.

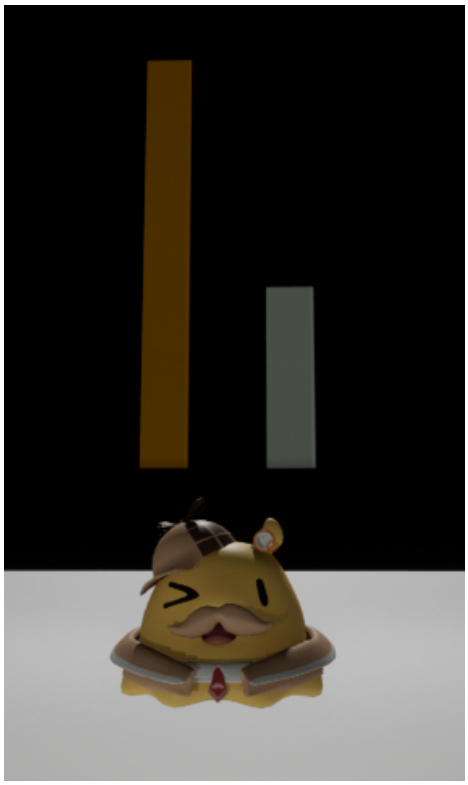


Рисунок 3.2 – ігровий агент покупця

На наступному зображені буде продемонстрований ігровий агент продавця. Синя шкала відповідає за мінімальну ціну за яку він готовий купити товар. Біла шкала – його пропозиція в поточному раунді.

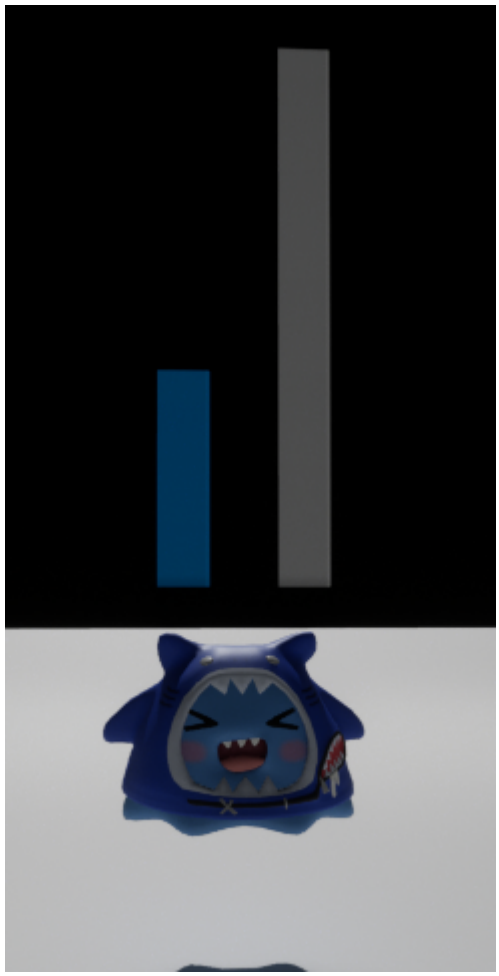


Рисунок 3.3 – Ігровий агент продавця

На наступному зображені ігрові агенти пробують укласти угоди, але невдало, так як різниці в їх пропозиція завелика, і в них не буде перетину вікон можливостей.

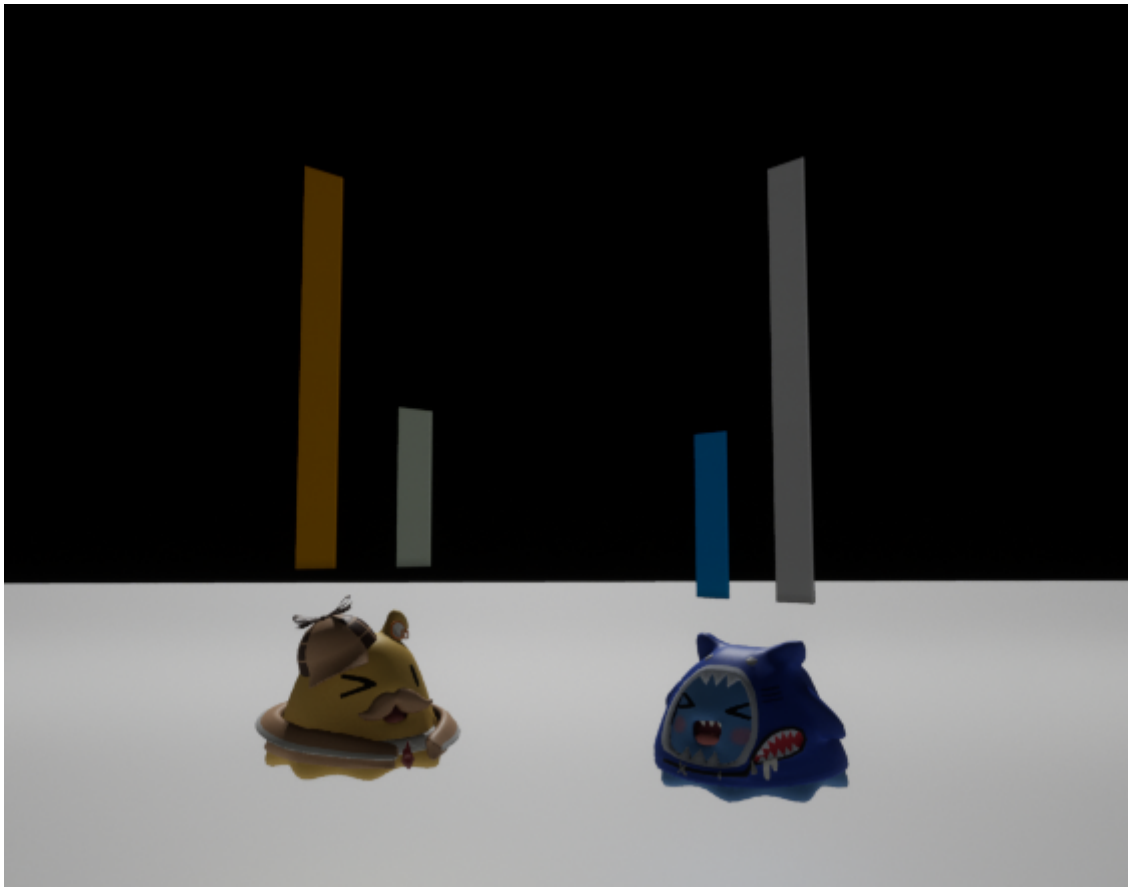


Рисунок 3.3 – Спроба заключити угоду

ВИСНОВКИ

У роботі була розглянутий Data Oriented підхід в програмуванні, та його застосування для створення комп’ютерних симуляцій. Був розглянутий патерн програмування Entity – Component – System, показана його ідея, структурні елементи, наведені переваги та недоліки його використання. Була розглянута математична модель попиту та пропозиції для подальшого написання її комп’ютерної симуляції.

Був досліджений ігровий рушій Unreal Engine 5, його структура та принципи роботи з ним. Ігрові рушії використовуються для створення ігрових програмних застосунків, які дозволяють створювати графічне відображення для комп’ютерних симуляцій симуляцій завдяки своїй системі рендеру.

Були використані інструменти та бібліотеки, які дозволяють писати програмні застосунки на основі ігрового рушія Unreal Engine 5 за принципом Data Oriented Design. До таких інструментів відноситься MassFramework та MassEntity.

За принципом Data Oriented Design та на основі патерну EQS була реалізована комп’ютерна симуляція для моделювання ринкових процесів на основі моделі Попиту та пропозиції.

Була показана актуальність теми дослідження, та важливість розвитку сфери комп’ютерних симуляцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stanford Encyclopedia of Philosophy [Електронний ресурс] – Режим доступу - https://plato.stanford.edu/entries/simulations-science/#NarDef
2. Studies. Student Library[Електронний ресурс] – Режим доступу - https://studies.in.ua/economy/17-tema-5-teoriya-sprosa-i-predlozheniya.html
3. Система електронного забезпечення навчання ЗНУ [Електронний ресурс] – Режим доступу – https://moodle.znu.edu.ua/pluginfile.php/625470/mod_resource/content/1/%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%8F%205.pdf
4. GameProgramminPattens [Електронний ресурс] – Режим доступу – http://gameprogrammingpatterns.com/component.html
5. David Colson blog. [Електронний ресурс] – Режим доступу – https://www.david-colson.com/2020/02/09/making-a-simple-ecs.html
6. Unreal Engine Official Documentation. [Електронний ресурс] -Режим доступу- https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/
7. Unreal Engine Official Documentation. [Електронний ресурс] -Режим доступу- https://docs.unrealengine.com/5.0/en-US/overview-of-mass-entity-in-unreal-engine/

i