Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ СТРАТЕГІЙ РУХУ ТА СИМУЛЯЦІЯ ОБ'ЄКТІВ У ВІРТУАЛЬНИХ СЕРЕДОВИЩАХ
(тема)

Виконав:
здобувач 2 року навчання,
групи ІНФм-24-1
Кузьменко П. М.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Науковий керівник проф. Гороховатський В. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кузьменку Пилипу Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження стратегій руху та симуляція об'єктів у віртуальних середовищах

затверджена наказом університету від 14 листопада 2025 року № 1045Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 21 листопада 2025 р.
3. Вихідні дані до роботи перелік використовуваних програмних засобів, теоретичні відомості про існуючі методи руху та навігації, графіки розподілу та таблиці відношення часу та швидкості проходження пар маршрутних точок, графіки розподілу та таблиці часу обчислення, науково-методична та науково-технічна літератури, дані інтернет-мережі.

4. Перелік питань, що потрібно опрацювати в роботі
1. Проведення аналізу літературних джерел.
2. Формування покрокового алгоритму для кожної із запропонованих стратегій поведінки.
3. Візуалізація покрокового алгоритму для кожної із запропонованих стратегій поведінки.
4. Створення програмної реалізації інтерактивної симуляції, стратегій поведінки, збору та аналізу даних.
5. Візуалізація та проведення аналізу отриманих результатів у вигляді графіків розподілу та таблиць.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) ілюстрація систем координат різних програм, схематичне зображення агента та навігаційної сітки, схеми алгоритмів стратегій поведінки об'єктів дослідження, шейдерні графи візуалізації оптичних ефектів, демонстрації ефектів оптики, приклад візуалізації ядрової оцінки щільності розподілу, демонстрація інтерактивності віртуального середовища, зображення, пов'язані із моделюванням та нанесенням текстур на тривимірні моделі, графіки розподілу та таблиці часу та швидкості проходження маршрутних точок, графіки розподілу та таблиці часу обчислення, схема виклику функцій оновлень Update та FixedUpdate.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	29.09.2025	
2	Аналіз завдання, підбір літератури	30.09.25-07.10.25	
3	Аналіз літератури з досліджуваної проблеми	08.10.25-13.11.25	
4	Програмна реалізація	29.09.25-25.10.25	
5	Обґрунтування отриманих результатів	21.10.2025	
6	Оформлення пояснювальної записки	29.09.25-18.11.25	
7	Перевірка на нормоконтроль	19.11.25-10.12.25	
8	Перевірка на плагіат	20.11.25-10.12.25	
9	Рецензування	21.11.25-10.12.25	
10	Підготовка презентації та доповіді	11.11.25-22.12.25	
11	Занесення роботи в електронний архів	21.11.25-22.12.25	
12	Попередній захист кваліфікаційної роботи	01.12.25-22.12.25	

Дата видачі завдання 29 вересня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____проф. Гороховатський В. О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 86 с., 6 табл., 29 рис., 46 джерел.

БІБЛІОТЕКА SCOTTPLOT, ВІРТУАЛЬНА РЕАЛЬНІСТЬ, ІНТЕРАКТИВНІ СИМУЛЯЦІЇ, КВАДРОКОПТЕРИ, МОДЕЛЮВАННЯ, МУЛЬТИКОПТЕРИ, СИМУЛЯЦІЯ, СТРАТЕГІЇ ПОВЕДІНКИ, UNITY.

Об’єктом дослідження є симуляція руху фізичних об’єктів у віртуальному середовищі на прикладі квадрокоптерів та мультикоптерів.

Метою дослідження є вивчення моделей навігації та стратегій поведінки квадрокоптерів та мультикоптерів, розробки програмних засобів збору та аналізу телеметрії, створення інтерактивної симуляції віртуального полігону із застосуванням технологій віртуальної реальності.

Використано методи програмного моделювання віртуальних середовищ та розробки шейдерів. Проведено аналіз сучасних методів доставки дронами та відповідних літературних джерел. Сформовано та візуалізовано алгоритми методів блок-схемами, ілюстраціями та іншими методами візуалізації.

Наукова новизна роботи полягає у вивченні і порівнянні стратегій поведінки квадрокоптерів та мультикоптерів для впровадження їх у практиці моделювання та доставки вантажів.

Взаємозв’язок з іншими роботами полягає у перетині та можливості використання цієї роботи в рамках доставки вантажів дронами, їх навігації тощо.

Рекомендації щодо використання результатів роботи полягають у можливості застосування результатів роботи у контексті автономної навігації.

У результаті дослідження розроблено програмні засоби збору та аналізу телеметрії, створено VR симуляцію віртуального полігону, проведено аналіз отриманих результатів на різних типах місцевості.

ABSTRACT

Explanatory note to the qualification work: 86 pages, 6 tables, 29 figures, 46 sources.

BEHAVIOUR STRATEGIES, INTERACTIVE SIMULATIONS, MODELING, MULTICOPTERS, QUADCOPTERS, SCOTTPLOT LIBRARY, SIMULATION, UNITY, VIRTUAL REALITY.

The object of the research is the simulation of the movement of physical objects in a virtual environment using the example of quadcopters and multicopters.

The aim of the research is to study navigation models and behaviour strategies of quadcopters and multicopters, develop software tools for collecting and conducting analysis of telemetry, and create an interactive simulation of a virtual polygon using virtual reality technologies.

Methods of software modeling of virtual environments and shader development were used. An analysis of modern methods of drone delivery and relevant literature sources was conducted. Algorithms of methods were formed and visualized using flowcharts, illustrations, and other visualization methods.

Scientific novelty of the work lies in the study and comparison of behavioral strategies of quadcopters and multicopters for their implementation in the practice of modeling and delivering cargo.

Interconnection with other works lies in the intersection and possibility of using this work within the framework of drone cargo delivery, their navigation, etc.

Recommendations for using the results of the work are the possibility of applying the results of the work in the context of autonomous navigation.

As a result of the research, software tools for collecting and analyzing telemetry were developed, a VR simulation of a virtual polygon was created, and the results were analyzed on different types of terrain.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕОМ – електронно-обчислювальна машина

ШІ – штучний інтелект

BCR – Battery consumption rate (споживання енергії батареї)

DOTS – Data-Oriented Technology Stack

FSM – Finite-State Machine (машина кінцевих станів або скінченний автомат)

GIMP – GNU Image Manipulation Program

ICPO – Improved Crown Porcupine Optimizer

KDE – Kernel Density Estimation (ядрова оцінка щільності розподілу)

PBR – Physically Based Rendering (фізично коректне відображення)

NavMesh – Navigation Mesh (навігаційна сітка)

PNG – Portable Network Graphics

TARGA – Truevision Advanced Raster Graphics Adapter

TGA – Truevision Graphics Adapter

VR – Virtual Reality (віртуальна реальність)

ВСТУП

Сучасний світ характеризується стрімким розвитком у сфері безпілотних авіаційних систем, а саме квадрокоптерів та мультикоптерів. Ці пристрої можуть бути застосовані у широкому спектрі задач: доставка вантажів, моніторингу інфраструктури, застосування в сільському господарстві та агрикультурі, проведення пошуково-рятувальних операцій, веденні розвідки місцевості тощо.

Особливу увагу в контексті навігації автономних чи безпілотних літальних апаратів привертаються стратегії поведінки, які визначають те, як система взаємодіє та здійснює навігацію по навколишньому середовищу. Існуючі підходи часто фокусуються на мінімізації відстані, енергоспоживання чи максимізації швидкості, проте ці підходи не завжди оптимальні в складних, неструктурованих умовах, особливо в умовах щільної забудови, нерівної місцевості та вимог до проведення обчислень у реальному часі.

Комп’ютерна симуляція або цифрове моделювання – це дослідження об’єктів, а саме явищ, процесів, пристроїв, систем тощо; за допомогою математичних моделей, що виконуються та обчислюються на електронно-обчислювальних машинах (ЕОМ).

Дана кваліфікаційна робота присвячена дослідженню та порівняльному аналізу двох стратегій поведінки квадрокоптерів та мультикоптерів: «пішохідної» (Pedestrian) та «висотної» (Overfly). Pedestrian стратегія поведінки передбачає рух на низькій висоті, зазвичай три метри від рівня землі, з орієнтацією на «обхід» перешкод по горизонтальні площині, подібно до того, як це робить людина, йдучи вулицею, або транспорт, переміщуючись дорогами чи бездоріжжям. Натомість стратегія Overfly передбачає обліт перешкод через верх, використовуючи більший запас висоти для забезпечення безпечного та ефективного пересування.

Окрім цього в даній роботі підлягають розгляду аспекти застсоування технологій віртуальної реальності, забезпечення інтерактивності та взаємодії із такими об’єктами як зброя, квадрокоптери, портали тощо, а також моделювання та нанесення текстур на тривимірні моделі.

Актуальність даного дослідження полягає у необхідності розробки адаптивних та надійних систем керування безпілотними авіаційними системами, здатних ефективно функціонувати в різноманітних умовах. Вибір оптимальної стратегії поведінки залежить від багатьох факторів, включно з типом місцевості, щільністю забудови, наявністю динамічних перешкод та вимогами до швидкості та енергоспоживання тощо.

В майбутніх ітераціях роботи можна додати, покращити чи змінити наступні елементи:

– перенести програму, що відповідає за симуляцію об’єктів дослідження, на більш відкритий ігровий рушій, такий, як рушій Godot;

– зробити симуляцію мультиагентної, що збільшить об’єм та точність зібраних даних;

– поглибити рівень симуляції шляхом використання більш повноцінної фізичної моделі, що діє на симульовані об’єкти: симуляція сил, утворених лопатями квадрокоптера чи мультикоптера тощо;

– поглибити рівень симуляції шляхом моделювання вітру та погодних ефектів, таких, як снігопад, дощ чи злива;

– збирати більше даних для аналізу, таких як рівень шуму, створеного об’єктами дослідження, використання батареї, швидкість руху тощо, у віртуальному середовищі;

– поглибити та розширити рівень аналізу, здійснюваного допоміжною програмою.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ ОБРАНОЇ ТЕМАТИКИ ТА ЗАСОБІВ СИМУЛЯЦІЇ ОБ'ЄКТІВ

1.1 Аналіз вибраної предметної області та літературних джерел

Предметна область даної кваліфікаційної роботи включає моделювання руху та дослідження стратегій поведінки об'єктів у віртуальних середовищах, окрім цього також підлягають вивченню використання засобів технології віртуальної реальності. Зазначимо, що об'єктом дослідження в даному випадку є дрони, а точніше квадрокоптери чи, потенційно, будь-які безпілотні літальні апарати із довільною кількістю несучих гвинтів, розміщених в одній площині (мультикоптери).

Інтерактивність середовища полягає у можливості людини (гравця) змінювати стан середовища таким чином, що це впливатиме на поведінку та взаємодію інших об'єктів, що належать до цього віртуального середовища, при цьому зміни середовища відбуваються у режимі реального часу, а самі зміни неможливо передбачити у повній мірі. З такої точки зору є цікавим вивчення предметної області ігрових рушіїв, їх можливостей та обмежень, про що йтиметься детальніше у наступних підрозділах роботи.

В загальному випадку для вирішення даної проблеми необхідно проробити три важливих питання: моделювання руху об'єктів, здійснення ними навігації та їх стратегія пересування. Окрім цього, проробці підлягає вивчення питання застосування технології віртуальної реальності у віртуальних середовищах.

Дана робота може бути корисною у прикладній сфері автономної адресної доставки їжі та інших предметів. Сучасні вантажні дрони здатні долати дистанції від 5 до 20 кілометрів, несучи при цьому від 5 до декількох десятків кілограмів корисного навантаження, таких характеристик може бути

цілком достатньо для пересування середнім за розміром міста чи сукупності невеликих населених пунктів. Часто дрони оснащуються знімними акумуляторами, що дає їм змогу миттєво поповнювати свій заряд батареї у визначених для цього місцях – це дає їм змогу виконувати роботу цілодобово та не стомлюючись. Окрім цього є можливим транспортування вантажу у важкодоступні місця чи місця із низьким, але постійним, попитом на доставку; вантаж не обов’язково має бути важким – це може бути книжки чи пошта. Індустрія безпілотних літальних апаратів є відносно новою та активно розвивається, тож вирішувана проблема є цілком актуальною. Новизна роботи полягає у дослідженні стратегій пересування літальних апаратів з практичної точки зору в умовах симуляції у реальному часі.

Далі було проведено аналіз літературних джерел.

У джерелі [1] йдеться про системи доставки на основі безпілотних літальних апаратів. Так розгляду підлягають питання етики; сталого розвитку; здоров’я, зокрема аспект доставки ліків та пандемій; фізики, зокрема енергетичні моделі та погодна фактори; зв’язок; доставки, зокрема тільки дронами, громадським транспортом та дронами, вантажівками та дронами; розглядаються проблеми, такі як безпека та конфіденційність, та безпека, ризик та надійність; також розглядаються впровадження, що імплементовані або реалізовані на практиці.

Так в дослідженні йдеться про енергетичні моделі, що дозволяють робити оцінку витрат енергії на основі таких факторів як вага корисного навантаження, напрямок вітру та швидкість, належна оцінка рівню витрат енергії в свою чергу призводить до меншої кількості неочікуваних посадок дронів, тим самим підвищуючи ефективність та надійність процесу доставки. Втім, також було виявлено значні відмінності в показниках споживання енергії дронами в залежності від області застосування моделі, конструкції дрона та передбачуваних операціях, які має виконувати дрон. Окрім цього, в дослідженні наведено інші роботи, що торкаються теми розробки алгоритму планування доставки дронами, який враховує рівень заряду акумулятора – це

є корисною відправною точкою для подальших досліджень. Йдеться також про те, що використання дронів у вантажній галузі має потенціал зменшити викиди вуглекислого газу CO_2 разом із споживанням енергії.

Так в дослідженні, проведеному у джерелі [2], пропонується модель споживання енергії для дронів для опису потреби в енергії для доставки дронами залежно від умов навколишнього середовища та шаблону польоту. Модель використовується для моделювання потреби в енергії стаціонарної системи доставки посилок, яка обслуговує певний набір клієнтів з одного лепо. Енергія, спожита дронами, порівнюється з потребами в енергії дизельних та електричних вантажівок, що обслуговують тих самих клієнтів з того самого депо.

Зокрема у висновках йдеться про те, що в міській місцевості стаціонарна система доставки, заснована на використанні дронів, використовує більше енергії у порівнянні з системою, що заснована на використанні вантажівок, але водночас у сільській місцевості використання дронів для доставки потребує обсяг енергії, подібний до того, що вимагає використання систем заснованих на використанні електричних вантажівок. Дане твердження є цікавим з тої точки зору, що його можна в певній мірі перевірити шляхом комп’ютерної симуляції, яка проводитиметься в даній кваліфікаційній роботі.

У джерелі [3] представлено алгоритм планування доставки, що враховує рівень заряду батареї дрона, що підвищує ефективність та надійність доставки. Основною метою роботи є продемонструвати нову модель планування доставки, що враховує рівень заряду та інші параметри батареї дрона, яка є більш ефективною за інші, традиційні, моделі доставки в аспекті кількості доставлених посилок за умови однакової ємності батареї дронів. Окрім цього, за висновками дослідження, модель, що враховує аспект батареї виявилася на 17% більш точною за традиційну модель доставки з такою ж схемою доставки, що в свою чергу запобігає неочікуваним посадками дрона.

Наступне джерело [4] розглядає проєктування системи доставки посилок за допомогою використання дронів, яка включає стратегічне планування системи та операційне планування для певного регіону. Метою дослідження є розробка надійного алгоритму планування доставки посилок, що використовує дрони та розглядає показник споживання енергії батареї (Battery consumption rate, BCR) як функцію ваги корисного навантаження. Так за результатами дослідження у наведеному джерелі було розроблено модель планування доставки дронами, що враховує споживання заряду батареї, що запобігає неможливим маршрутам польоту.

У джерелі [5] представлено покращений оптимізатор коронних дикобразів (Improved Crown Porcupine Optimizer, ICPO). Метою дослідження є представити, продемонструвати, а також порівняти даний алгоритм із іншими у аспекті планування маршрутів дронами, що дозволяє ефективно здійснювати навігацію складними та динамічними середовищами, уникати перешкод, оптимізувати маршрут польоту та споживання батареї. Виходячи з самої роботи та, додатково, джерела [6] – цей алгоритм надихається захисними механізмами дикобразів та мотивований поведінкою хижака-жертви, він має покращену ранню конвергенцію завдяки механізмам балансування візуального та слухового захисту та може оптимізовувати задачі лінійної, нелінійної, неперервної та дискретної природи. Наведене сімейство алгоритмів є новим – так перші згадки цього алгоритму датуються 2024 роком.

В результатах дослідження також наведено знайдені алгоритмами оптимальні маршрути, з яких видно, що в гірській та, в меншій мірі, міській місцевості зміна висоти під час руху є значною компонентою оптимальності маршруту.

У наступному джерелі [7] розглядається така тема як шаблони програмування ігор, зокрема в контексті продуктивності кінцевого продукту. В даному джерелі розглядаються наступні шаблони проєктування: команда, Flyweight, спостерігач, прототип, Singleton, стан – шаблони послідовності:

подвійний буфер, ігровий цикл, метод оновлення – шаблони поведінки: байт-код, пісочниця підкласу, об’єкт типу – шаблони відокремлення (Decoupling Patterns) – компонент, черга подій, локатор сервісів – а також шаблони оптимізації: локальність даних, прапорець брудних даних, пул об’єктів, просторове розбиття. Інформація, що міститься в даному джерелі, є корисною та може знадобитися на етапах проєктування та розробки програмного продукту.

У джерелі [8] розглядається тема шаблонів програмування ігор. Так в даному джерелі підлягають вивченню та розгляду принцип KISS, принципи SOLID: принцип єдиної відповідальності (Single responsibility principle), принцип відкритості/закритості (Open/closed principle), принцип підстановки Ліскoв (Liskov substitution principle), принцип розділення інтерфейсу (Interface segregation principle), принцип інверсії залежностей (Dependency inversion principle) – шаблони проєктування фабрика, пул об’єктів, Singleton, команда, стан, спостерігач, модель-вид-контролер (Model-View-Controller, MVC) та модель-представлення-представник (Model-View-Presenter, MVP). Дане також джерело згадує та звертається до книги, що відома під назвою «Банда чотирьох» [9].

У джерелі [10] охоплено напрямок галузі знань з фізики. Так розгляду підлягають знання з елементарної механіки: закони Ньютона, робота, енергія, системи частинок, момент, імпульс, зіткнення, момент сили та обертання в одному вимірі, векторний момент сили, момент імпульсу, статика тощо – а також застосування механіки: рідини, коливання, рівняння хвиль, звук та гравітація. Дане друковане джерело є корисним для вирішення проблеми моделювання рівноприскореного прямолінійного руху, а також фізичного моделювання інтерактивних об’єктів, такі як пружини.

1.2 Огляд існуючих ігрових рушіїв

Ігровий рушій є центральною програмною частиною будь-якої відеогри та слугує фундаментом для її технічної складової. Рушії допомагають полегшити розробку відеогри шляхом уніфікації та систематизації її внутрішньої структури. Зазвичай ігрові рушії складаються з наступних компонент: рушій рендерингу, фізичний рушій, звук, анімації, керування пам’яттю та, часто, ігровий штучний інтелект, мережевий код тощо.

В контексті даної кваліфікаційної роботи ігрові рушії є цікавими через те, що вони забезпечують проведення симуляції у реальному часі, що водночас накладає і певні обмеження, системи навігації, візуалізацію тривимірної графіки, та інтерактивність, а саме наявність гравця, що здатен непередбачувано змінювати стан віртуального середовища у реальному часі. Таким чином, проєкт можна розширити, поєднавши аспект симуляції з інтерактивністю.

З добре відомих та досі підтримуваних рушіїв можна навести наступні: Construct, CryEngine, GameMaker Studio, Godot, Unity та Unreal Engine. З них націленими на візуалізацію тривимірної графіки є лише рушії CryEngine, Godot, Unity та Unreal Engine.

Зупинимося детальніше на рушіях Unity та Godot.

Так цікавою особливістю рушія Godot, що значно вирізняє його на фоні інших існуючих ігрових рушіїв на ринку є його ліцензія. Так рушій використовує MIT ліцензію, що дозволяє використовувати рушій в некомерційних, комерційних та освітніх цілях абсолютно безкоштовно, а також модифікувати його початковий (вихідний) код; єдиною вимогою є включення тексту ліцензії рушія в довільному місці гри або похідний проєкт [11].

Окрім цього рушій Godot є новішим за інші рушії, перша його версія була опублікована 14 січня 2014 року і з тих пір він активно розвивається, що

в свою чергу означає те, що він спирається на більш новий досвід в індустрії розробки відеоігор та інтерактивних симуляцій.

Цікавою особливістю рушія Unity, яка вигідно вирізняє його на фоні інших – це наявність технологій Job System, Burst Compiler та DOTS (Data-Oriented Technology Stack). Вони дозволяють значно пришвидшувати швидкість виконання коду шляхом багатопотокового чи паралельного виконання інструкцій, ціною обмежень щодо технічних рішень у проєкті.

Так, з точки зору ієрархії віртуального середовища, у рушії Unity існують сцени, ігрові об’єкти та префаби. Сцени за своєю суттю є набором ігрових об’єктів та екземплярів префабів, вони також можуть мати додаткові пов’язані дані, такі, як текстури запеченого світла, навігаційна сітка тощо. Ігрові об’єкти є основою гри, вони можуть мати дочірні та батьківський об’єкти в деревовидній ієрархії, кожен об’єкт містить у собі певний набір компонентів, який задається користувачем редактора рушії, так об’єкт може містити компоненти та їх дані, такі, як Transform, який є обов’язковим для будь-якого ігрового об’єкта, Mesh FIlter, Mesh Renderer, Rigidbody чи будь-який інший компонент, вбудований у рушій Unity чи створений користувачем редактора рушія. Префаби за своєю суттю є ігровими об’єктами, що мають спільний об’єкт в корені ієрархії; слугують у якості шаблона, на основі якого можна створювати нові об’єкти у сцені; зміна даних компонентів чи ієрархії об’єктів префаба відображається на усіх існуючих його екземплярах; а кожен з префабів зберігається у якості окремого файлу в директорії проєкту, над яким ведеться робота у редакторі рушія. Префаби також можуть успадковувати інші префаби (Prefab Variant) та містити у собі один та більше префабів у якості дочірніх об’єктів (Prefab Nesting).

В рушії Godot з точки зору ієрархії існують лише сцени та вузли. Вузли є фундаментальним будівельним блоком будь-якого проєкту, створеного у рушії Godot. Так існують різні вузли, задачею яких може бути відображення зображення, програвання звуку, відображення зображення з камери, створення та врахування геометрії зіткнень, фізична взаємодія тощо. Як і

ігрові об’єкти у поєднанні з їх компонентами в рушії Unity, усі вузли мають наступні властивості: назва, дані, отримання зворотних викликів на оновлення кожного кадру, можливість їх розширення новими даними та функціями та можливість додавання іншого вузла у якості дочірнього до даного. Разом вузли формують деревовидну ієрархію; вузол також може містити скрипти, але не більше одного на вузол. Сцени у рушії Godot містять у собі набір вузлів, що дотримуються деревовидної ієрархії, та слугують коренем для такого набору вузлів. Сцени в рушії можна успадковувати, що дозволяє змінювати лише певні характеристики окремих вузлів, в інших же випадках дотримуючись властивостей оригінальної сцени. Окрім цього сцени, коли збережені, можуть грати роль нового типу вузлів, що дозволяє сценам містити у собі інші дочірні сцени [12]. Сцени, розміщені всередині іншої сцени, також можна повністю редагувати шляхом натискання правої кнопки миші по дочірній сцені та вибору параметра Editable Children. Таким чином, сцени у рушії Godot еквіваленті поєднанню сцен та префабів у рушії Unity.

Різні програми використовують різні системи координат. Так рушії Unity та Godot використовують метричну систему координат, де одна одиниця виміру відповідає одному метру. Система фізики та інші системи рушіїв підлаштовані саме на цю шкалу. Так напрямок осей можна визначати за правилами лівої та правої руки – так можна зімкнути усі пальці обраної руки, повністю розімкнути великий палець таким чином, що його положення еквівалентне положенню при жесті «палець вгору», вказівний палець повністю розімкнути, а середній – розімкнути наполовину, тобто на 90 градусів. Виконавши ці дії, великий палець вказуватиме на напрямок осі X+, середній вказуватиме на напрямок осі Y+, а середній – на напрямок осі Z+. Так в рушії Unity напрямок осі X+ відповідає за напрямок вправо, Y+ – напрямок вверх, Z+ – уперед, в рушії Godot напрямок осі X+ відповідає за напрямок вправо, Y+ – напрямок вверх, а Z- – уперед, натомість в рушії Unreal Engine напрямок осі X+ відповідає за напрямок уперед, Y+ відповідає

за напрямок вправо, а Z+ – вгору. На рисунку 1.1 [13] зображено напрямки системи координат у різних ігрових рушіях та інших програмах, що працюють із тривимірною графікою.

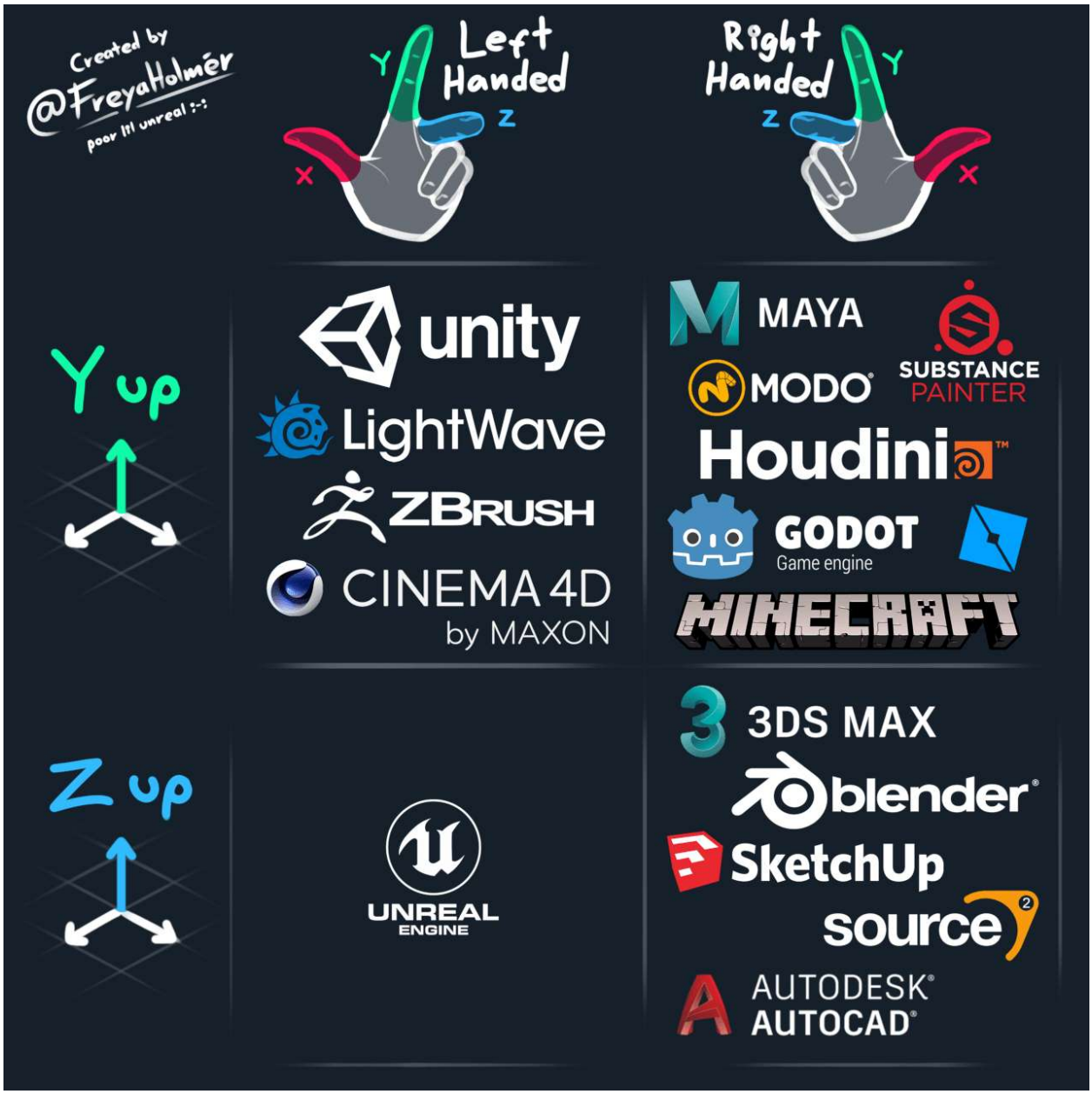

Рисунок 1.1 – Системи координат різних програм

Далі розглянуто аспект навігації в рушіях Unity та Godot. Агентом в даному контексті називається самостійна одиниця, що здійснює рух та навігацію.

Так в рушії Unity задача навігації розглядається шляхом розділення на дві підзадачі: як міркувати про рівень, для того, щоб знайти пункт призначення; та як туди переміститися. Проблема міркування про рівень є більш глобальною та статичною, оскільки враховує всю сцену; натомість рух до пункту призначення є більш локальним та динамічним завданням, оскільки враховує лише напрямок руху та як запобігти зіткненням із іншими агентами, що рухаються.

В Unity агенти описуються як циліндри – в них є власні параметри радіусу та висоти. Зона для пересування створюється автоматично з геометрії сцени, шляхом перевірки місць, де агент може стояти. Потім ці місця з'єднуються до поверхні, що лежить поверх геометрі сцени – ця поверхня називається навігаційною сіткою (Navigation Mesh, NavMesh). Навігаційна сітка зберігає цю поверхню як опуклі полігони, таке представлення є корисним, оскільки воно дозволяє точно знати, що між будь-якими двома точками всередині полігону немає перешкод. Окрім меж полігону, збереженню підлягає також і інформація про те, які полігони є сусідами один з одним – це дозволяє міркувати про всю зону для переміщення.

Для того, щоб знайти шлях між двома локаціями у сцені, спочатку необхідно зіставити початкову та кінцеву локації з їхніми найближчими полігонами. Потім розпочинається пошук з початкової локації з відвідуванням усіх сусідів, доки кінцевий полігон не буде досягнуто. Для пошуку шляху між двома локаціями, рушій Unity використовує алгоритм A*. На рисунку 1.2 наведено схематичне зображення агента та навігаційної сітки [14].

Реалізація навігації в рушії Godot є подібної до тої, що представлена в рушії Unity, вона також використовує алгоритм пошуку шляху під назвою A*. Окрім цього, обидва рушія дозволяють використовувати динамічні навігаційні перешкоди.

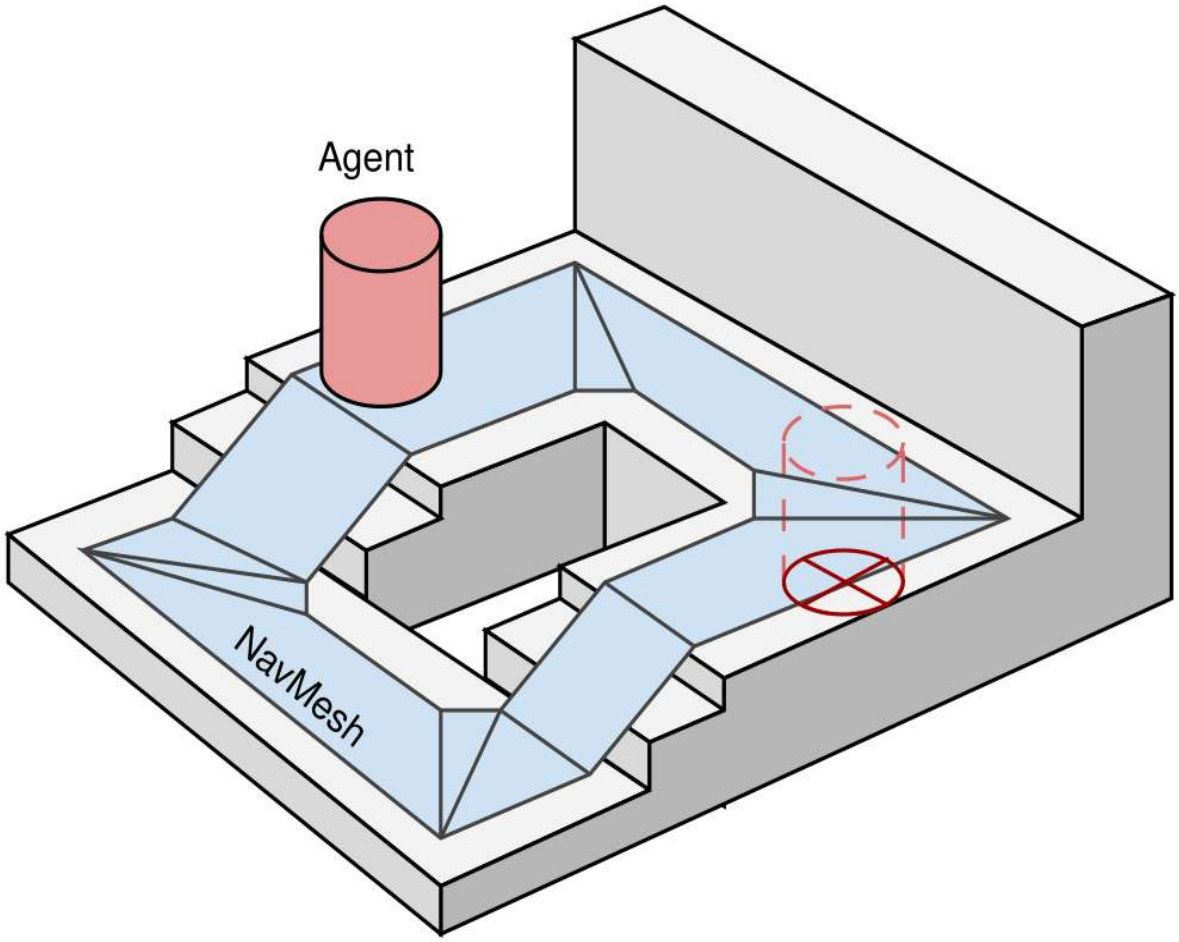

Рисунок 1.2 – Схематичне зображення агента та навігаційної сітки

Окрім цього, обидва рушія підтримують розробку під віртуальну реальність, зокрема стандарт OpenXR. Втім варто зазначити, що рушій Unity не підтримує збірку під платформу Linux разом з використанням OpenXR на момент написання даної кваліфікаційної роботи.

1.3 Постановка задачі дослідження

Таким чином, порівняння стратегій руху та створення інтерактивної симуляції є актуальним завданням. Прийнято рішення щодо розроблення програмного засобу моделювання об’єктів з різними стратегіями руху, а саме

з «пішохідною» (Pedestrian) та «висотною» (Overfly) стратегіями поведінки, даний програмний засіб також здійснює збір телеметрії та забезпечує інтерактивну симуляцію віртуального полігону з використанням технологій віртуальної реальності. Окрім цього прийнято рішення про створення допоміжного програмного засобу, що забезпечує візуалізацію та статистичний аналіз зібраних даних.

Об’єктом дослідження є симуляція руху фізичних об’єктів у віртуальному середовищі на прикладі квадрокоптерів та мультикоптерів

Метою дослідження є вивчення моделей навігації та стратегій поведінки квадрокоптерів та мультикоптерів, розробки програмних засобів збору та аналізу телеметрії, створення інтерактивної симуляції віртуального полігону із застосуванням технологій віртуальної реальності.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз літературних джерел щодо використання квадрокоптерів чи мультикоптерів у сфері транспортування, доставки вантажів та їх стратегії поведінки;

– сформувати покроковий алгоритм для кожної із запропонованих стратегій поведінки квадрокоптерів та мультикоптерів;

– візуалізувати покроковий алгоритм для кожної із запропонованих стратегій поведінки квадрокоптерів та мультикоптерів блок-схемою чи іншим методом візуалізації;

– розробити програмний засіб, що дозволить проводити моделювання об’єктів з різними стратегіями руху, збирати телеметрію та забезпечить інтерактивну симуляцію віртуального полігону із застосуванням технологій віртуальної реальності;

– розробити програмний засіб, що дозволить проводити аналіз зібраної телеметрії;

– візуалізувати отримані результати у вигляді графіків та таблиць;

– провести аналіз отриманих результатів та зробити висновки.

2 ОСНОВНІ АСПЕКТИ ТЕОРІЇ СИМУЛЯЦІЇ ОБ'ЄКТІВ

2.1 Математична модель пружини

Розглянемо математичну модель компоненту симуляції, що дозволяє імітувати об'єкт на пружині, він має параметри базового видовження, максимальної дистанції видовження, маси та коефіцієнта пружності. Нижче наведено формули, за якими розраховується сила, прискорення та позиція такого об'єкту у просторі [10].

$$F = kx, \tag{2.1}$$

де F – сила, що діє на об'єкт на пружині;

k – коефіцієнт пружності;

x – видовження пружини.

$$\vec{a} = \vec{d}\frac{F}{m}, \tag{2.2}$$

де $\vec{a}$ – прискорення, що діє на об'єкт на пружині;

$\vec{d}$ – вектор напрямку сили, зазвичай дорівнює (0; 0; -1) у локальному просторі координат;

m – маса об'єкту.

$$\vec{p} = \vec{p_0} + \vec{V}\Delta t + \frac{1}{2}\vec{a}\Delta t^2, \tag{2.3}$$

де $\vec{p}$ – позиція об’єкту в кінці кроку симуляції;

$\vec{p_0}$ – позиція об’єкту на початку кроку симуляції;

$\vec{V}$ – вектор поточної швидкості об’єкту;

Δt – часовий крок симуляції в секундах.

Математична модель пружини застосовується у проєкті кваліфікаційної роботи задля забезпечення інтерактивності з такими об’єктами як, наприклад, затвор зброї тощо.

2.2 Модель поведінки мультикоптерів

Одним із важливих питань в даній кваліфікаційній роботі є питання руху модельованих об’єктів. Так, для моделювання руху об’єктів застосовується модель рівноприскореного руху.

Загальна формула рівноприскореного руху має наступний вигляд [10, 15]:

$$\vec{a} = \frac{\vec{V} - \vec{V}_0}{t}, \tag{2.4}$$

де $\vec{a}$ – прискорення фізичного тіла (м/с^2);

$\vec{V}$ – кінцева швидкість;

$\vec{V}_0$ – початкова швидкість;

t – час.

Формула зміни швидкості під час рівноприскореного руху має наступний вигляд [15]:

$$\vec{V} = \vec{V}_0 + \vec{a}t. \tag{2.5}$$

Формула координати тіла має наступний вигляд [15]:

$$x = x_0 + V_{0x}t + \frac{1}{2}a_x t^2. \tag{2.6}$$

Формула проєкції переміщення має наступний вигляд [15]:

$$m_x = V_{0x}t + \frac{1}{2}a_x t^2. \tag{2.7}$$

Виходячи з наведених формул, для обчислення того, чи варто для симульованого об’єкта почати сповільнюватись, або чи варто продовжувати прискорення, можна навести наступний хід рішення:

$$m = Vt - \frac{1}{2}at^2;$$

$$2m = 2Vt - at^2; \quad -2m + 2Vt - at^2 = 0;$$

$$2m - 2Vt + at^2 = 0;$$

$$[D = b^2 - 4ac].$$

Розглянемо випадок 1. $D < 0$:

$$4 - 8am < 0; \quad 1 - 2am < 0; \quad 1 < 2am; \quad 2am > 1;$$

$$m > \frac{1}{2a};$$

Немає розв ' язків $\Rightarrow$ *сповільнюватись зарано*.

Розглянемо випадок 2. $D > 0$:

$$m < \frac{1}{2a};$$

$$t_1=[\frac{-b+\sqrt{D}}{2a}]=\frac{2V+2\sqrt{1-2am}}{2a}=\frac{V+\sqrt{1-2am}}{a};$$

$$t_2=[\frac{-b-\sqrt{D}}{2a}]=\frac{V-\sqrt{1-2am}}{a}.$$

Розглянемо випадок 3. $D=0$:

$$m=\frac{1}{2a};$$

$$t=[\frac{-b}{2a}]=\frac{2V}{2a}=\frac{V}{a}.$$

Спробуємо знайти більш лаконічний та простий розв'язок проблеми. Так існують наступні твердження:

$$V-at=0;$$

$$m=Vt-\frac{1}{2}at^2.$$

З цього випливає наступне:

$$t=\frac{V}{a};$$

$$m=\frac{V^2}{a}-\frac{V^2}{2a}=\frac{V^2}{2a};$$

$$m<m_{target}\Rightarrow \textit{раннє сповільнення};$$
$$m\geq m_{target}\Rightarrow \textit{сповільнення із запасом швидкості};$$

де m_{target} – відстань до цільової точки руху.

Таким чином тепер відомо переміщення симульованого об’єкта, якщо він почне сповільнюватись в даний момент часу, маючи певну швидкість та стале прискорення руху. З цього також виведено умови, за яких симульований об’єкт має починати своє сповільнення так, щоб він перетнув цільову точку руху.

Нижче наведено модель визначення орієнтації симульованого об’єкта у просторі, маючи на вході швидкість об’єкта. Така модель є корисною для потреб візуалізації та жодним чином не впливає на рух чи інші властивості симульованого об’єкта.

$$\vec{V}_H = (V_x;\ 0;\ V_z);$$

$$t = Clamp\left(\frac{\|\vec{V}_H\|}{V_T},\ 0,\ t_{max}\right);$$

$$\vec{D}_{TF} = Lerp\left(\hat{V}_H,\ \vec{D}_{down},\ t\right);$$

$$\vec{D}_{HTF} = (D_{TFx};\ 0;\ D_{TFz});$$

$$y_{max} = \tan\left(Tilt_{max}\right) \cdot \|\vec{D}_{HTF}\|;$$

$$D_{TFy} = Clamp\left(D_{TFy},\ -y_{max},\ y_{max}\right);$$

$$\vec{T}_F = \begin{cases} \vec{D}_{TF}, & якщо \|\vec{D}_{TF}\| \neq 0 \\ \vec{T}_F, & якщо \|\vec{D}_{TF}\| > 0 \end{cases};$$

де $\vec{V}$ – швидкість симульованого об’єкта (м/с);

$\vec{V}_H$ – швидкість симульованого об’єкта у горизонтальній площині;

V_T – цільова швидкість симульованого об’єкта, за якої його нахил стає найбільшим;

t_{max} – константа, значення якої дорівнює 0,99;

$Clamp(value, min, max)$ – функція приведення вхідного значення *value* до діапазону значень *[min; max]*;

$\vec{D_{TF}}$ – цільовий напрямок симульованого об’єкта;

$\hat{V_H}$ – нормована швидкість симульованого об’єкта у горизонтальній площині;

$\vec{D_{down}}$ – напрямок вниз, дорівнює (0; -1; 0);

$Lerp(a, b, t)$ – функція лінійної інтерполяції між значеннями *a* та *b*;

$\vec{D_{HTF}}$ – цільовий напрямок симульованого об’єкта у горизонтальній площині;

$Tilt_{max}$ – максимальний нахил симульованого об’єкта (рад);

$\vec{T_F}$ – напрямок вперед для симульованого об’єкта.

Зазначимо, що після обчислення значення $\vec{T_F}$ ми можемо визначити кватерніон цього напрямку і поступово обертатись від поточного кватерніону положення симульованого об’єкту у просторі до щойно обчисленого кватерніона за швидкістю не більшою за певну кількість градусів на секунду.

Під час прямого керування квадрокоптером чи мультикоптером гравцем, на об’єкт діють фізичні сили. Визначимо вектори $a = (a_x; a_y)$ та $b = (b_x; b_y)$, що позначають введення гравця на стіках для великого пальця на лівому та правому контролерах відповідно, $i = (i_x; i_y; i_z)$ – позиція мотора відносно дрона, m – множник сили, а $f(a)$ та $g(b)$ – криві, що виражають ступінь відклику на дію користувача. Визначимо силу F, що створена кожним з моторів симульованого об’єкта окремо, вона обчислюється за наступним принципом:

$$F = m(f(a_y \cdot 0.5 + 0.5) + x + y), \tag{2.8}$$

$$x = \begin{cases} \frac{1}{2} g(|b_x|), & якщо\ i_x b_x < 0 \\ -\frac{1}{2} g(|b_x|), & якщо\ i_x b_x \geq 0 \end{cases}, \tag{2.9}$$

$$y = \begin{cases} \frac{1}{2} g(|b_y|), & якщо\ i_z b_y < 0 \\ -\frac{1}{2} g(|b_y|), & якщо\ i_z b_y \geq 0 \end{cases}. \tag{2.10}$$

Визначимо момент сили, що діє на симульований об’єкт під час прямого керування гравцем.

$$x = Clamp(a_x, -1, 1), \tag{2.11}$$

$$\vec{M} = \vec{D}_{up} M_{yaw} \frac{x}{|x|} f(|x|), \tag{2.12}$$

де $\vec{a}$ – введення гравця на стіках для великого пальця на лівому контролері;

$Clamp(value, min, max)$ – функція приведення вхідного значення *value* до діапазону значень *[min; max]*;

$\vec{M}$ – момент сили, що діє на симульований об’єкт;

$\vec{D}_{up}$ – напрямок вгору, дорівнює (0; 1; 0);

M_{yaw} – максимальний момент сили рискання, що може діяти на симульований об’єкт;

$f(input)$ – крива, що виражає ступінь відклику на дію користувача.

2.2.1 Модель поведінки «пішохідного» мультикоптера

В цьому та наступному підрозділах розгляду підлягають моделі поведінки, тобто руху, мультикоптерів. Вони є основою для дослідження та порівняння, результати якого буде наведено у наступному, останньому, розділі роботи.

Компонент, що відповідає за Pedestrian, тобто «пішохідну», стратегію поведінки за своєю суттю є достатньо простою машиною кінцевих станів (Finite-State Machine) з трьома станами: режим польоту, приземлення та зльоту. На рисунку 2.1 проілюстровано стани та умови переходу із одного стану до іншого. Так симульований об’єкт починає зі стану зльоту та злітає доти, доки точку злету не буде досягнуто, після чого перемикається в режим польоту та летить по усім точкам призначення, допоки точка призначення не стане маршрутною точкою та її не буде досягнуто, після чого симульований об’єкт перемикається до стану приземлення та приземлюється, а після приземлення робить запит на нову маршрутну точку та очікує на призначення нової маршрутної точки.

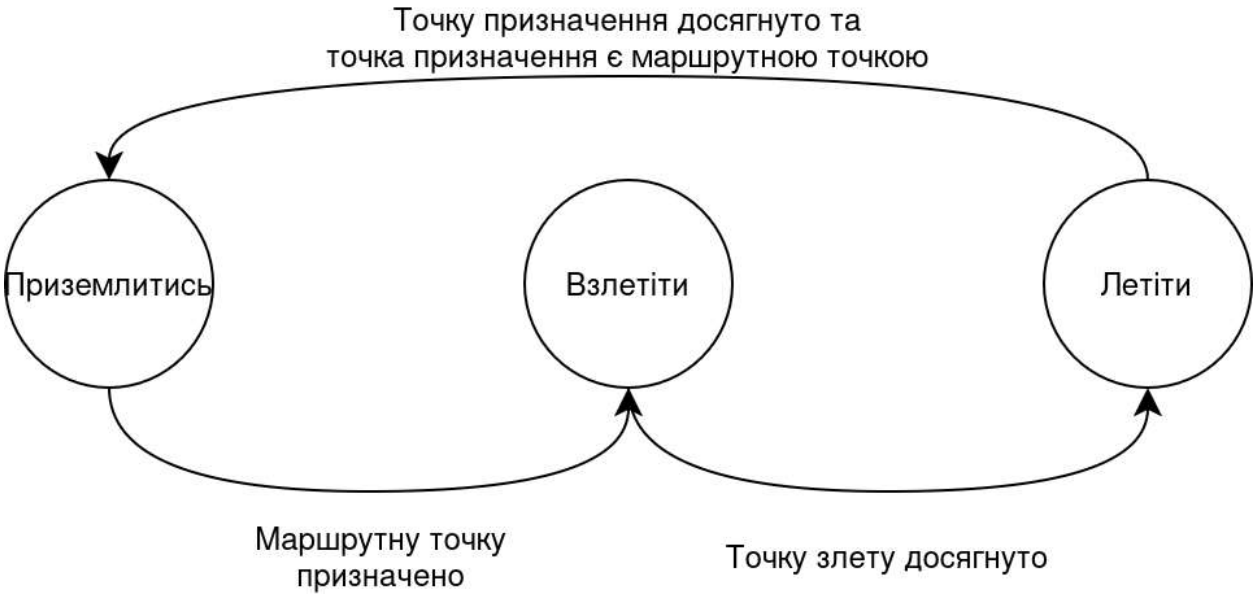

Рисунок 2.1 – Скінченний автомат станів симульованого об’єкта з Pedestrian стратегією поведінки

Таким чином, після призначення нової маршрутної точки, симульований об’єкт перемикається в режим польоту та будує маршрут до призначеної маршрутної точки, використовуючи для цього компонент NavMesh Agent. Так для кожного типу агента у віртуальному середовищі будується своя навігаційна сітка, за якою будуть будуватись маршрути від однієї точки до іншої. В кожного типу навігаційного агента є свої параметри, такі як радіус, висота, висота кроку, максимальний нахил тощо. На рисунку 2.2 наведено приклад запеченої навігаційної сітки з параметрами Radius = 1, Height = 3, Step Height = 0,9 та Max Slope = 45. Слово «запечена» в даному контексті позначає те, що навігаційна сітка була попередньо обчислена та збережена як частина даних гри – це дозволяє оптимізувати продуктивність гри ціною можливості до динамічної адаптації, виключенням є компонент NavMesh Obstacle, який дозволяє створювати перешкоди, в тому числі динамічно.

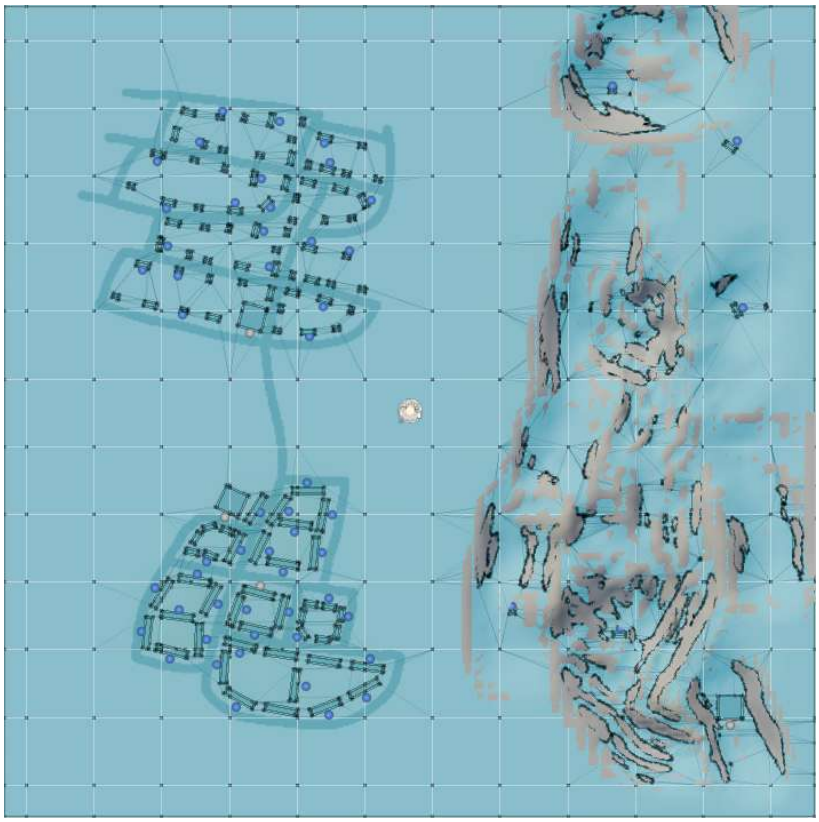

Рисунок 2.2 – Приклад навігаційної сітки

Наведемо детальний покроковий опис Pedestrian стратегії поведінки:

Крок 1. Ініціалізація. Симульований об’єкт переходить у стан зльоту.

Крок 2. Точка зльоту. Визначається точка приземлення, виходячи з поточної позиції симульованого об’єкта, до якої застосовується зміщення.

Крок 3. Переміщення до точки зльоту. Здійснюється переміщення до точки зльоту із заданим часовим кроком. Якщо точку зльоту досягнуто – переходимо до Кроку 4, інакше – Крок 3.

Крок 4. Перехід до стану польоту. Відбувається перехід до стану польоту та скид лічильника пройдених кутів маршруту, окрім цього здійснюється побудова маршруту пересування до цільової маршрутної точки за допомогою компоненту NavMesh Agent.

Крок 5. Переміщення до маршрутної точки. Здійснюється переміщення до наступного кута маршруту, або, якщо усі кути пройдено, безпосередньо до маршрутної точки із заданим часовим кроком. Якщо точку переміщення досягнуто і існує наступний кут маршруту – переходимо до наступного кута, шляхом збільшення лічильника пройдених кутів на одиницю, та повторюємо Крок 5; якщо ж точку переміщення досягнуто і точкою переміщення є маршрутна точка – переходимо до Кроку 6, інакше – повторюємо Крок 5.

Крок 6. Перехід до стану приземлення. Відбувається перехід до стану приземлення.

Крок 7. Приземлення на маршрутну точку. Здійснюється переміщення до точки приземлення, яка визначається розташуванням поточної маршрутної точки, із заданим часовим кроком. Якщо точку приземлення досягнуто – завершуємо маршрутну точку та переходимо до Кроку 8, інакше – повторюємо Крок 7.

Крок 8. Очікування призначення нової маршрутної точки. Симульований об’єкт очікує призначення нової маршрутної точки, а у випадку, коли нову маршрутну точку призначено, відбувається перехід до стану зльоту та до Кроку 2.

2.2.2 Модель поведінки «висотного» мультикоптера

На рисунках 2.3 та 2.4 наведено можливі випадки, які можуть відбутись із симульованим об’єктом, що використовує стратегію поведінки Overfly.

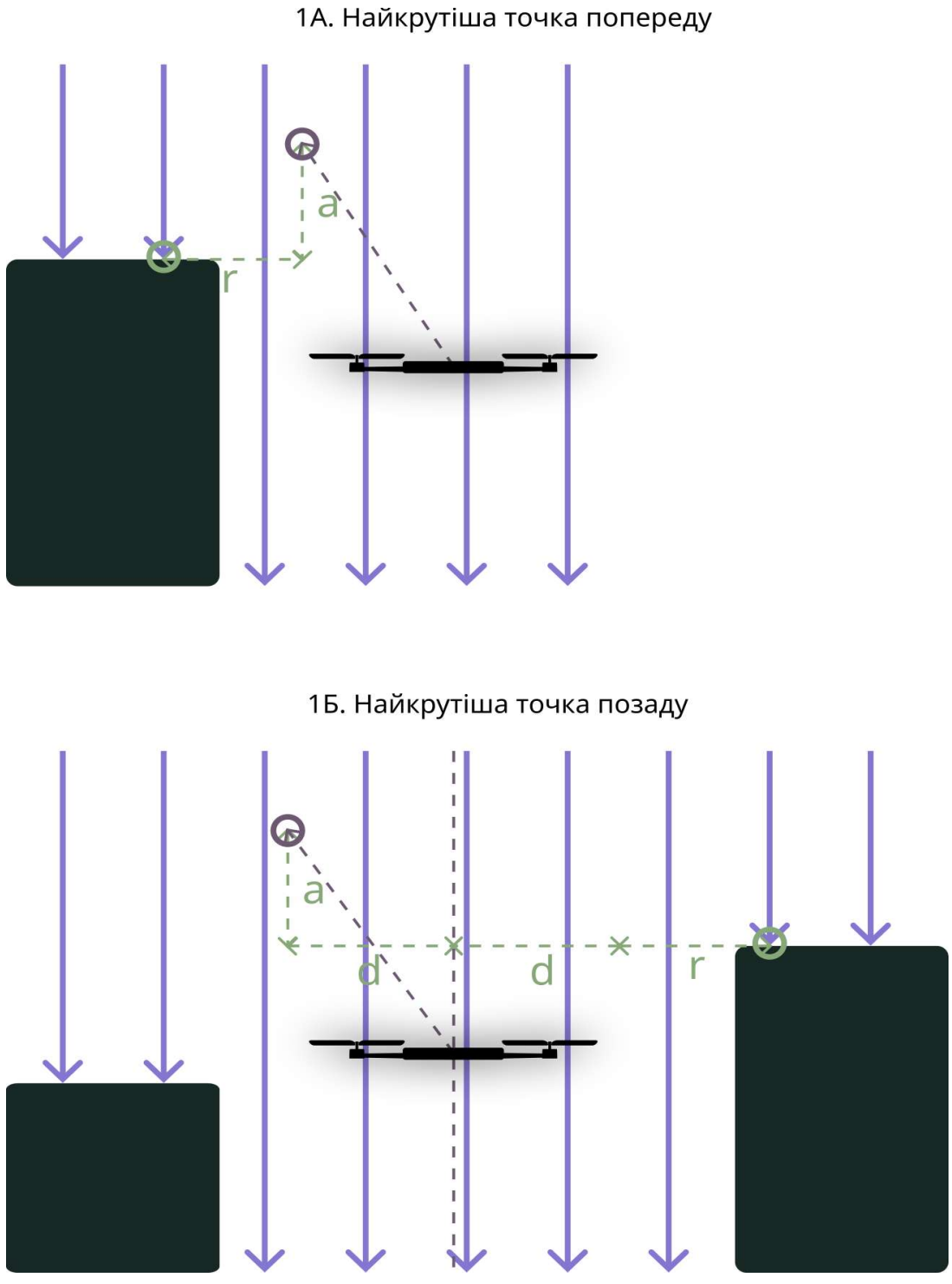

Рисунок 2.3 – Випадки 1А та 1Б стратегії поведінки Overfly

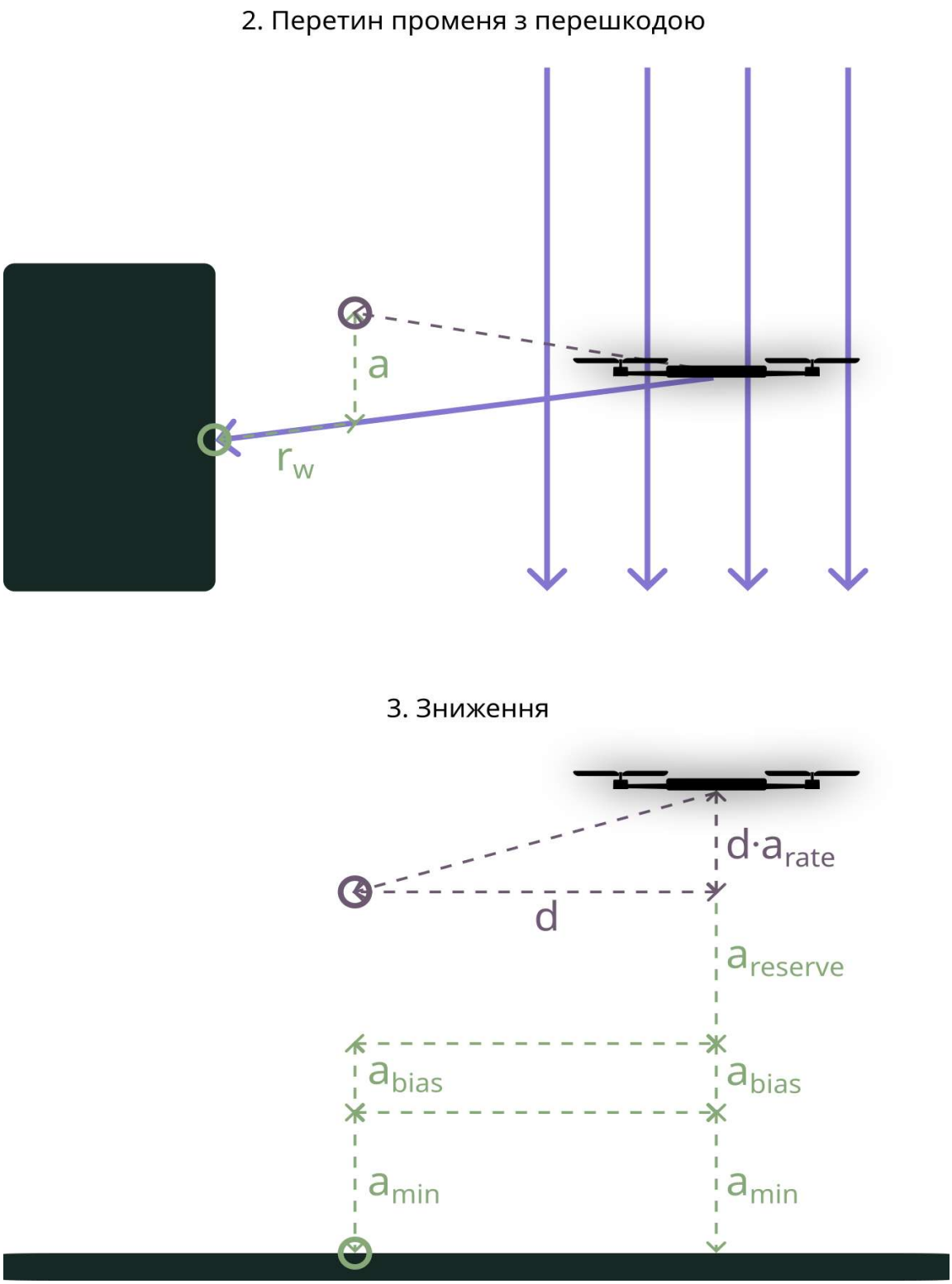

Рисунок 2.4 – Випадки 2 та 3 стратегії поведінки Overfly

Випадки 1А та 1Б виникають тоді, коли висота між симульованим об’єктом та найкрутішою точкою в межах певного радіусу є вищою за нуль. Під найкрутішою точкою розуміється така точка, в якої кут між напрямком від симульованого об’єкта до даної точки та напрямком вертикально вгору є мінімальним. В таких випадках симульований об’єкт здійснює рух до найкрутішої точки, до якої застосовано зміщення вертикально вгору на відстань бажаної висоти польоту та зміщення проти напрямку до маршрутної точки, але не більше ніж горизонтальна компонента вектору, до якого застосовується зміщення.

Якщо в межах певного радіусу не існує такої точки, яка є вищою за симульований об’єкт – то в такому випадку випускається промінь по напрямку до маршрутної точки та на зниження так, що кожен пройдений метр шляху в горизонтальній площині промінь знижується на a_{rate} метрів, а довжина горизонтальної компоненти променю не перевищує відстані між симульованим об’єктом та маршрутною точкою у горизонтальній площині.

Випадок 2 виникає тоді, коли промінь перетинається із іншим об’єктом. В такому випадку цільова точка переміщення визначається за наступною формулою:

$$\vec{P_{target}} = \vec{P_{hit}} + \vec{D_{up}} \cdot a_{preferred} - \frac{\vec{V}}{\|\vec{V}\|} \cdot min\left(R_{scan}, \|\vec{V}\| \cdot R_{factor}\right), \qquad (2.13)$$

де $\vec{P_{target}}$ – цільова точка переміщення;

$\vec{P_{hit}}$ – точка перетину променя з іншим об’єктом;

$\vec{D_{up}}$ – напрямок вгору, дорівнює (0; 1; 0);

$a_{preferred}$ – бажана висота польоту;

$\vec{V}$ – продукт різниці вектора $\vec{P_{hit}}$ та поточної позиції симульованого об’єкта;

R_{scan} – радіус сканування висот;

R_{factor} – фактор досяжності, що має значення в діапазоні [0; 1]. Значення 0 виражає намір сповільнюватися аж до безпосередньо точки перетину променя з іншим об’єктом, а одиниця – намір сповільнюватися якомога раніше.

Випадок 3 виникає тоді, коли промінь не перетинається з жодним іншим об’єктом. В такому випадку симульований об’єкт здійснюватиме рух до точки, що має позицію поточної маршрутної точки в горизонтальній площині, а висота якої знижується на a_{rate} метрів кожен метр шляху в горизонтальній площині. Висота точки, до якої здійснюється рух, не може бути нижчою висоти маршрутної точки та точки приземлення за поточної позиції симульованого об’єкта, до якої застосовано зміщення на бажану висоту польоту вертикально вгору.

У випадках, коли поточну маршрутну точку досягнуто у горизонтальній площині, симульований об’єкт здійснює посадку та завершує маршрутну точку. Якщо ж висота симульованого об’єкту стає меншою за мінімальну – він здійснює рух вертикально вгору до набуття бажаної висоти польоту; оскільки ці дві висоти різні, стає можливим уникнення ситуації здійснення циклу зниження та злету на коротких проміжках часу.

Таким чином, компонент, що відповідає за Overfly, тобто «висотну», стратегію поведінки за своєю суттю є дещо складнішим за Pedestrian стратегію поведінки. Він не використовує машину кінцевих станів, натомість реалізуючи систему пріоритетності цільових точок. На рисунку 2.5 проілюстровано блок-схему алгоритму. Таким чином, стратегія поведінки, реалізована в цьому компоненті, є основою для дослідження та порівняння, результати якого буде наведено у наступному, тобто останньому, розділі роботи.

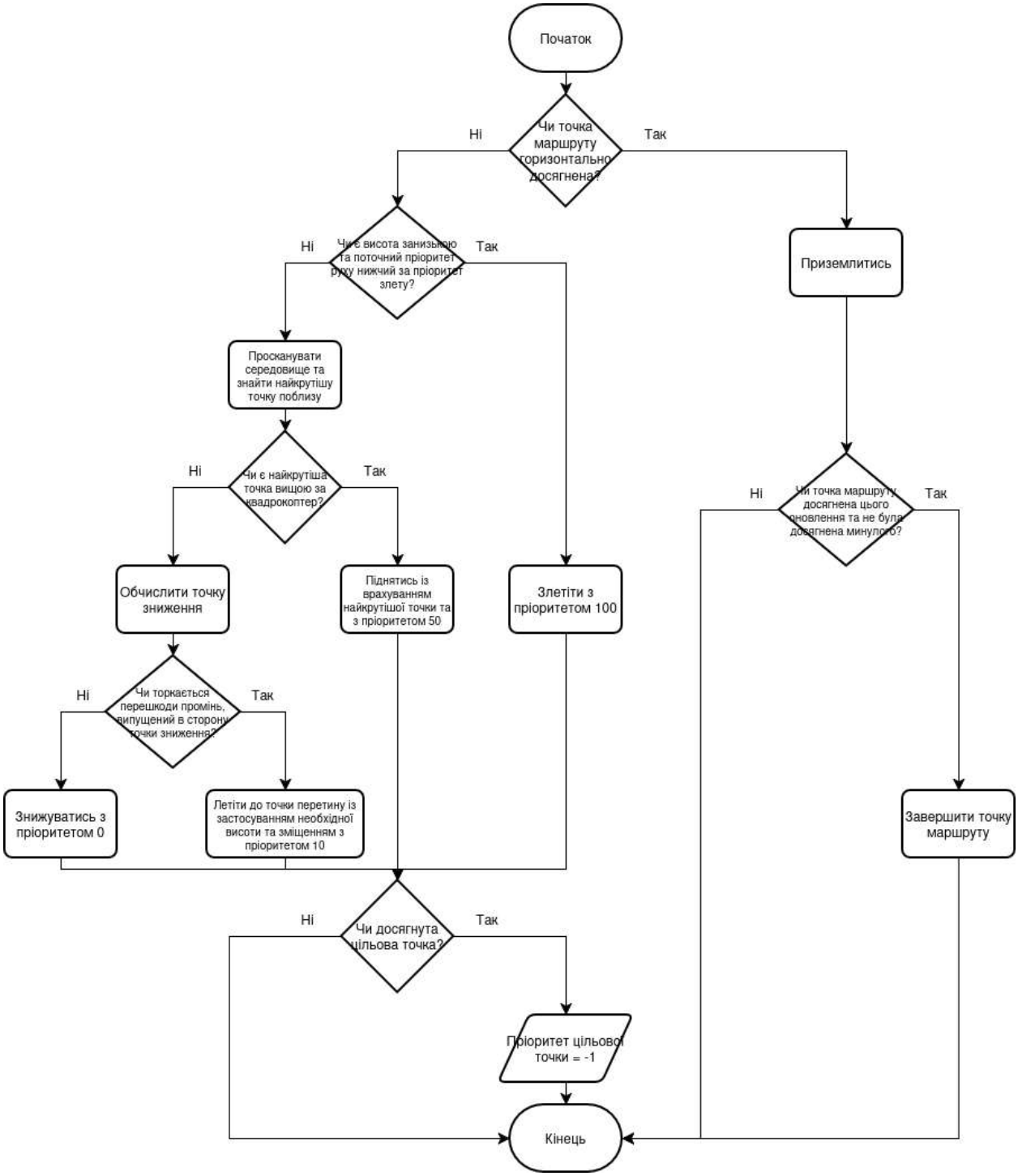

Рисунок 2.5 – Блок-схема стратегії поведінки Overfly

Система пріоритетності цільових точок полягає в тому, що цільову точку неможливо перевизначити у випадках, коли пріоритетність нової цільової точки є нижчою за поточну. У випадках, коли цільова точка досягнута – її пріоритетність набуває значення -1.

2.3 Створення ефекту оптики за допомогою інструменту Shader Graph

Для відтворення оптичних ефектів в даній кваліфікаційній роботі було використано інструменти засобу Shader Graph. Так для створення моделі було використано вузли Position, Camera, Distance, Subtract, Transform, Multiply, Divide, UV, Tiling And Offset, Polar Coordinates, Split, Power, Sample Texture 2D, Lerp.

На рисунках 2.6 та 2.7 наведено граф та підграф, що відповідають за візуалізацію оптичних ефектів.

Підграф, що наведений на рисунку 2.7, відповідає за оптичне зміщення та використовується як частина обчислень шейдерного графу, який наведено на рисунку 2.6.

Після наведення шейдерних графів у форматі рисунків, їх можна навести у більш лаконічному формульному вигляді.

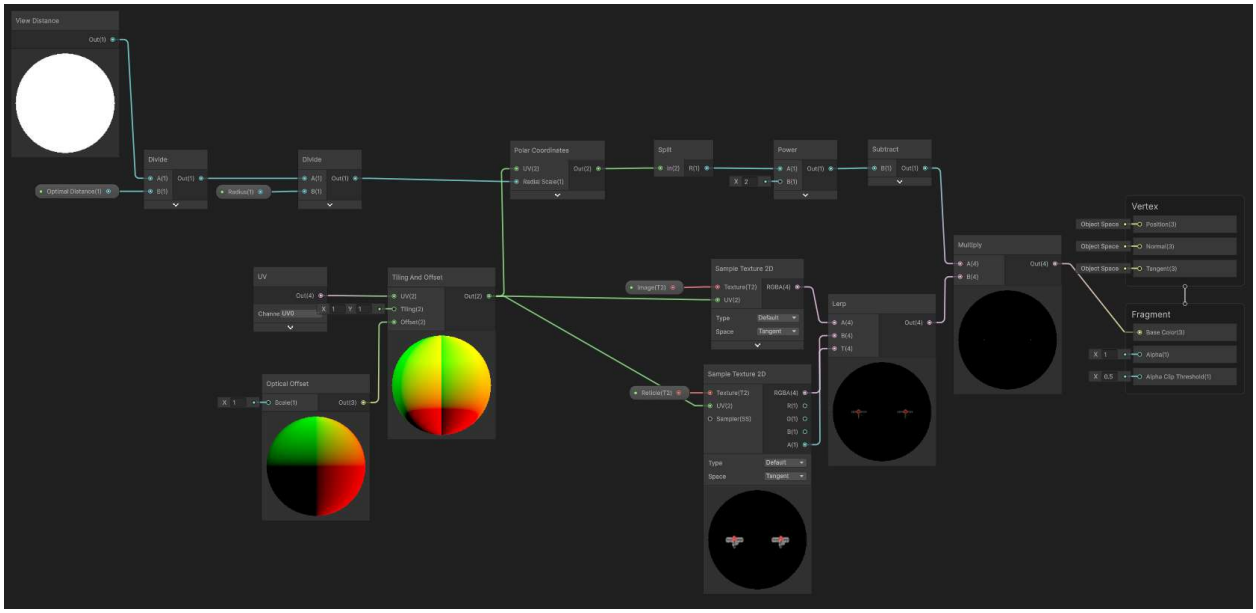

Рисунок 2.6 – Граф, що відповідає за візуалізацію оптичних ефектів

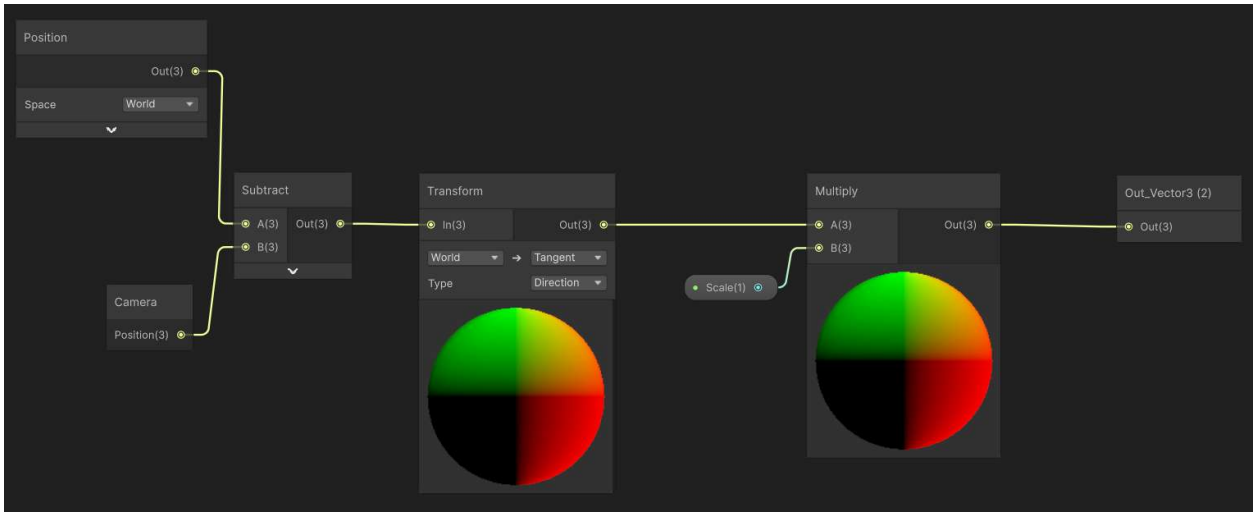

Рисунок 2.7 – Підграф, що відповідає за оптичне зміщення

Далі описано ефект оптики у вигляді формул. Зазначимо, що формули не є фізично коректними та є лише імітацією ефектів оптики.

$$\vec{P} = \vec{P_{UV}} + WTT\left(\vec{P_{world}} - \vec{P_{camera}}\right) \cdot O_{scale}, \tag{2.14}$$

$$\left(1 - PC_R^{Sharpness}\left(\vec{P}, \frac{\left\|\vec{P_{camera}} - \vec{P_{world}}\right\|}{D_{optimal} R}\right)\right) \cdot Lerp\left(SIT\left(\vec{P}\right), SRT\left(\vec{P}\right), SRT_A\left(\vec{P}\right)\right), \tag{2.15}$$

де $\vec{P_{UV}}$ – UV координати оброблюваного фрагменту;

$\vec{P_{world}}$ – позиція оброблюваного фрагменту у глобальних координатах;

$\vec{P_{camera}}$ – позиція камери у глобальних координатах;

$WTT\left(direction\right)$ – функція перетворення напрямку з глобального до дотичного простору координат;

O_{scale} – сила зміщення, дорівнює одиниці за нормальних умов;

$D_{optimal}$ – оптимальна дистанція споглядання на зображення, що утворене лінзами;

R – радіус затінення;

$PC_R^{Sharpness}(UV, Radial\,Scale)$ – функція перетворення з декартової до полярної системи координат з масштабом $Radial\,Scale$, де R – радіус, тобто перша компонента вектору, а $Sharpness$ – степінь, до якого підноситься отримане значення;

$SIT(UV)$ – функція вибірки текстури, у випадку практичного використання – Render Texture, отриманої з камери;

$SRT(UV)$ – функція вибірки текстури сітки оптичного прицілу;

$SRT_A(UV)$ – альфа-канал вектору кольору, отриманого з функції вибірки текстури сітки оптичного прицілу;

$Lerp(a, b, t)$ – функція лінійної інтерполяції між значеннями a та b.

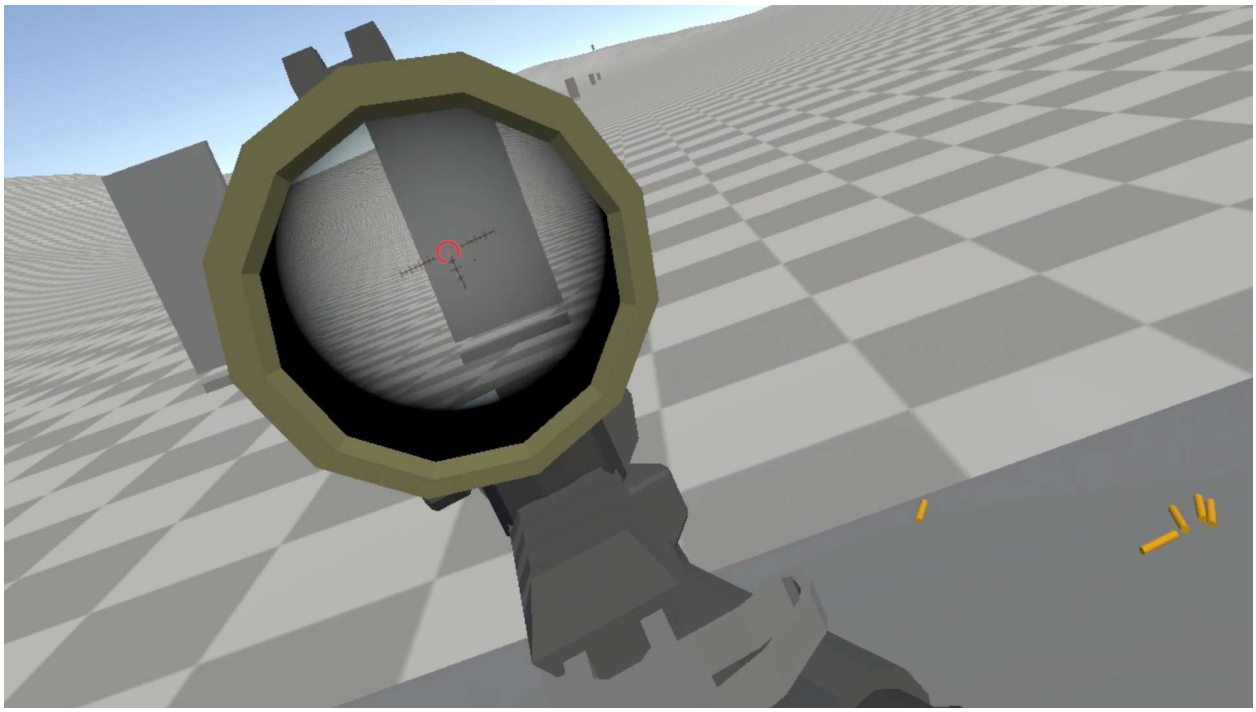

Рисунок 2.8 – Демонстрація ефекту оптики

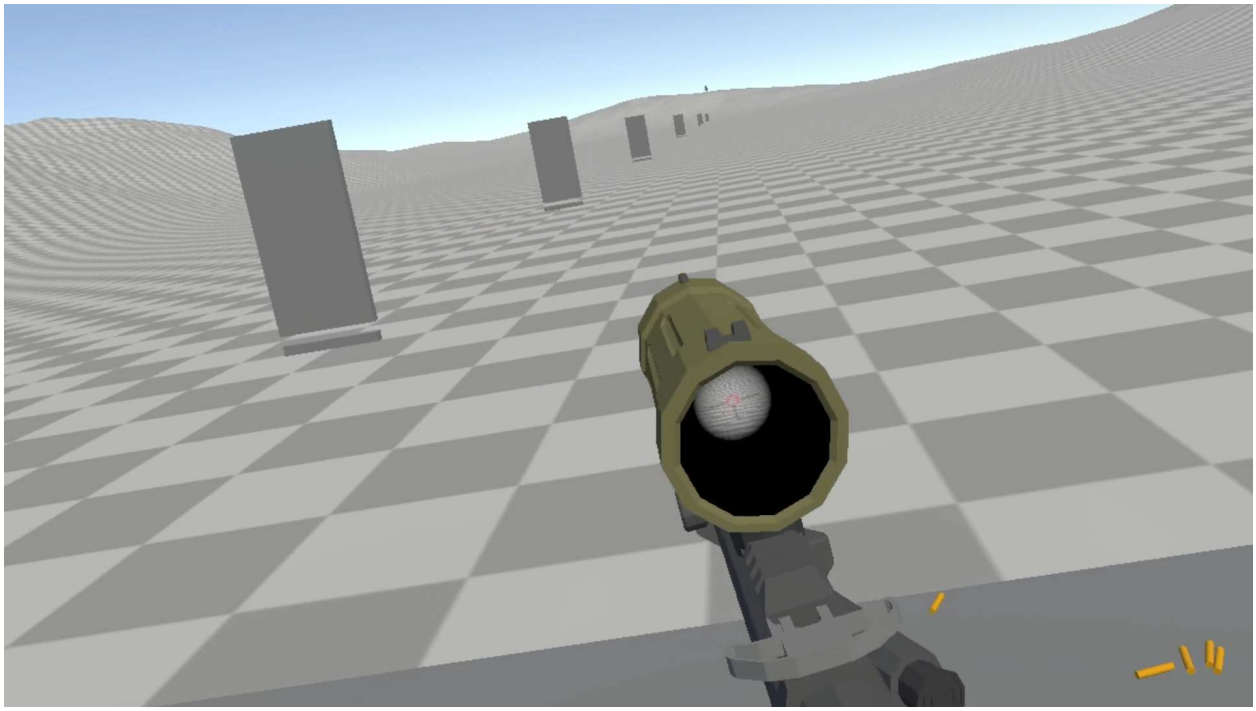

Рисунок 2.9 – Демонстрація ефекту оптики

2.4 Візуалізація результатів дослідження

Основою презентації (відображення) результатів є двовимірні графіки розподілу значень та таблиці. У випадках відображення розподілу значень обидві осі є числовими, де горизонтальна вісь позначає значення величини, а вертикальна – відносну частоту її виникнення. Вертикальна лінія в таких графіках позначає середнє значення відображуваної величини.

Графік розподілу величин відображається за допомогою методу оцінки функції щільності розподілу випадкової величини під назвою Kernel Density Estimation [16, 17]; в даній роботі для оцінки використовується нормальне ядро.

Існує наступне визначення ядрової оцінки щільності розподілу:

$$\hat{f}_h(x)=\frac{1}{n}\sum_{i=1}^{n}K_h(x-x_i)=\frac{1}{nh}\sum_{i=1}^{n}K\left(\frac{x-x_i}{h}\right), \tag{2.16}$$

де $f_h(x)$ – функція ядрової оцінки щільності розподілу;

h – параметр згладжування (пропускна здатність);

K – статистичне ядро;

K_h – масштабоване ядро, $K_h=\frac{1}{h}K\left(\frac{x}{h}\right)$;

$(x_1, x_2, \ldots, x_n)$ – вибірка з незалежних і однаково розподілених величин.

Для нормального ядра формула оцінка щільності розподілу набуває наступної форми:

$$\hat{f}_h(x)=\frac{1}{nh\sigma}\frac{1}{\sqrt{2\pi}}\sum_{i=1}^{n}\exp\left(\frac{-(x-x_i)^2}{2h^2\sigma^2}\right), \tag{2.17}$$

де σ – стандартне відхилення вибірки $\vec{x}$.

В цьому методі можна задавати параметр сили згладжування, який треба задавати в залежності від потреб точності та задачі. В деяких випадках, для пришвидшення обчислень, можна скорочувати розмірність вхідних даних.

На рисунку 2.10 наведена ілюстрація роботи цього методу [18]. На наведеній ілюстрації варто звернути увагу саме на чорну криву лінію, яка і є результатом відображення кривої ядрової оцінки щільності розподілу. Водночас елементи стовпчастої діаграми на наведеній ілюстрації відображають більш класичний підхід до візуалізації розподілу величин.

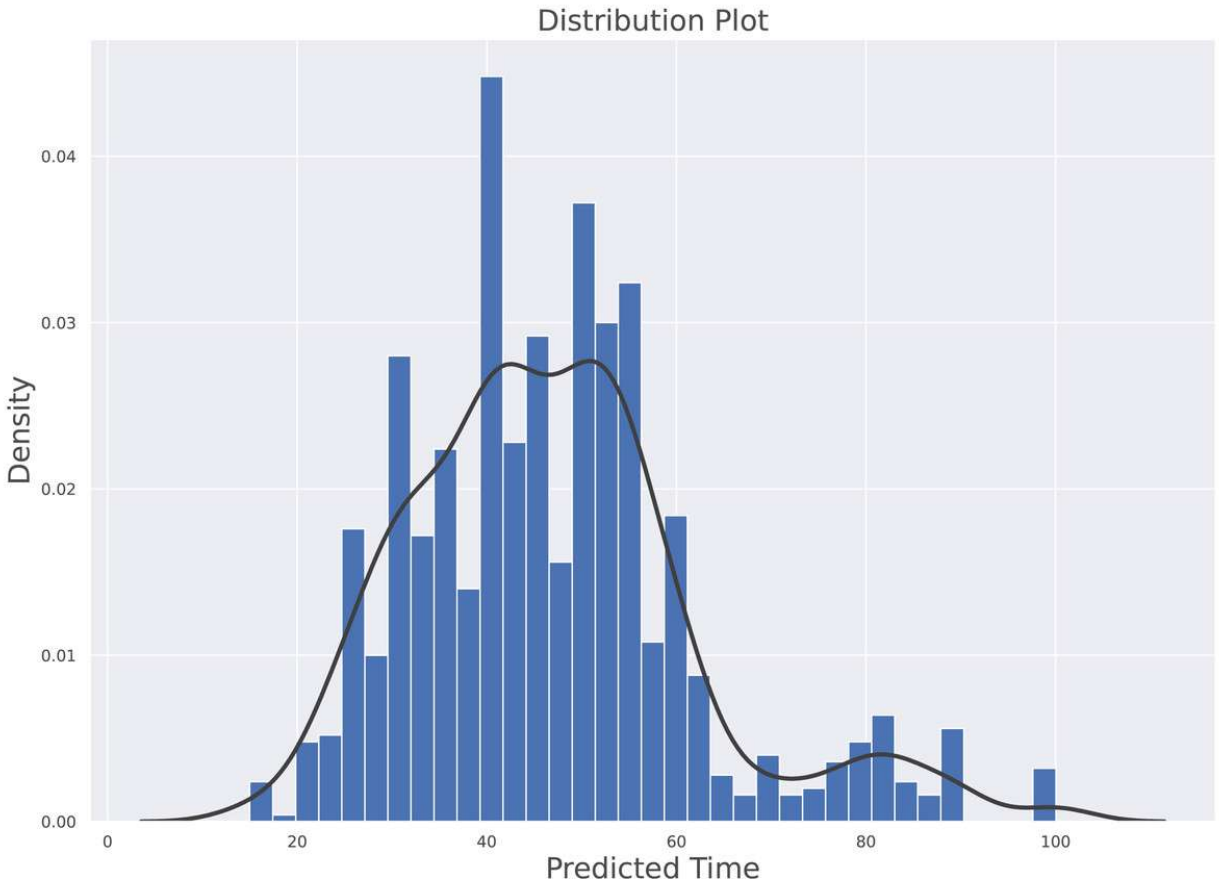

Рисунок 2.10 – Приклад візуалізації ядрової оцінки щільності розподілу

3 ДОСЛІДЖЕННЯ ТА ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ

3.1 Вибір програмного забезпечення

Основою для розробки програмного засобу, що відповідає за задачі проведення симуляції та збору даних для порівняння було обрано ігровий рушій Unity 6 [19], а саме версії 6000.0.58f2. Такий вибір пояснюється наявністю досвіду роботи з цим ігровим рушієм. Зазначимо, що молодші версії рушія, а саме Unity 6000.0.58f1 чи 2022.3.67f1 та нижче, мають вразливість високого рівня тяжкості (CVSS Score: 8,4), яка дозволяє локально виконувати сторонній програмний код та викрадати конфіденційну інформацію з пристрою, на якому запущено цю програму [20, 21] – таким чином, використання молодших версій програми може бути неприпустимо. Ігровий рушій Unity версії, що використовується у проєкті, працює у тандемі з мовою програмування C# на платформі .NET Framework 4.6.

Основою для розробки допоміжного програмного засобу, який має здійснювати порівняння стратегій поведінки на основі вже зібраних даних, обрано мову програмування C# на платформі .NET 8.0. Такий вибір пояснюється особистими уподобаннями та наявністю досвіду роботи з цією мовою програмування: код написаний цією мовою програмування відносно легко читається, добре підходить для написання програм з об’єктно-орієнтованим підходом до розробки, добре документована та підтримує усі основні платформи. В проєкті не використовуються сторонні математичні бібліотеки, натомість для малювання графіків використовується бібліотека ScottPlot. Альтернативою цій бібліотеці є бібліотека OxyPlot, втім перевагами ScottPlot є те, що ця бібліотека добре документована та має так звану книгу рецептів [22], вона також є гнучкою, добре підтримує векторну графіку, тобто підтримує збереження файлів у формат .svg, і не має проблем із збереженням

в такий формат, окрім цього графіки можна масштабувати разом зі збільшенням роздільної здатності: так зображення розміром 4096×4096 пікселів не буде відрізнятись від зображення 256×256 пікселів нічим іншим аніж сама його роздільна здатність.

З інших, сторонніх, програмних засобів було використано наступні програми: VSCodium – редактор коду, який по своїй суті є редактором Visual Studio Code, але без деяких його частин, таких як телеметрія та частин, що не є відкритими; Blender – програмний пакет для створення тривимірних моделей та комп'ютерної графіки; GIMP або GNU Image Manipulation Program – растровий графічний редактор; Inkscape – редактор векторної графіки з можливостями, подібними до можливостей CorelDraw чи Adobe Illustrator; Git та GitGUI – розподілена система керування версіями файлів та її графічний інтерфейс відповідно; LibreOffice Writer – програма для створення документів; draw.io [23] – вебзастосунок для створення схем тощо; reveal.js, про який йтиметься пізніше, – усі з наведених програмних засобів є вільними та відкритими. Розробка велась на операційній системі на основі ядра Linux 6.14, а саме дистрибутиві Ubuntu 25.04, графічній оболонці GNOME 48 та віконній системі X11. Втім усі наведені програмні засоби мають також запуститись і на операційних системах сімейства Windows.

Для створення слайдів презентації було використано засіб reveal.js, ключовою особливістю якого є дотримання парадигми для редагування структурованих документів WYSIWYM (What You See Is What You Mean) замість більш часто використовуваної WYSIWYG (What You See Is What You Get). Відділення вмісту від презентації дозволяє, наприклад, швидко змінювати візуальний стиль без потреби вносити зміни у вміст презентації, а також допомагає стандартизувати візуальне оформлення як в самій презентації, так і поза нею. Засіб підтримує мову розмітки гіпертексту HTML та, з деякими обмеженнями, Markdown розмітку.

3.2 Застосування технологій віртуальної реальності у проєкті

На початкових стадіях розробки проводилося дослідження використання засобів віртуальної реальності в грі, що імітує полігон з доступними різними видами зброї, керованими дронами тощо. Цей досвід є корисним з точки зору дослідження засобів технології віртуальної реальності у рушії Unity.

Розглянемо аспект взаємодії зі зброєю, так на початкових ітераціях проєкту було імплементовано систему віддачі зброї, засновану на рушії фізики PhysX, він використовується в Unity за замовчуванням. Система полягає у розділенні об’єкта зброї на два підоб’єкти: основний об’єкт, який є фізичним тілом та доступний для взаємодії із руками гравця, він містить дочірні набір колайдерів та фізичні зв’язки (наприклад компонент ConfigurableJoint), зв’язки з’єднані із дочірнім фізичним тілом другого підоб’єкту; та другий об’єкт який відображає віддачу пострілу, він копіює позицію та обертання основного об’єкту, є фізичним об’єктом та містить дочірній об’єкт, який також є фізичним тілом, на який застосовуються сили під час пострілу, в свою чергу він містить візуальне відображення зброї, візуальні ефекти, сокети магазинів тощо. На рисунку 3.1 наведено описану ієрархію об’єктів.

Завдяки такій структурі вдалося уникнути проблем фізичного рушія, що виникають у підході без розділення на два підоб’єкти, окрім цього сама віддача є фізично коректною, або як мінімум в достатній мірі наближена до такої. Втім, такий підхід має й проблеми: фізичність віддачі дещо обмежує налаштовуваність такої системи, окрім цього виникають і нові проблеми – так фізичне тіло може зупинити рух так і не повернувшись у початкове положення, окрім цього з’являється непередбачувана поведінка по типу хаотичного руху зброї. Такі явища можна пояснити тим, що фізичний рушій PhysX або його імплементація не призначені під подібні конфігурації

фізичних об’єктів, помилкою в них або помилкою в конфігурації компонентів у редакторі рушія Unity.

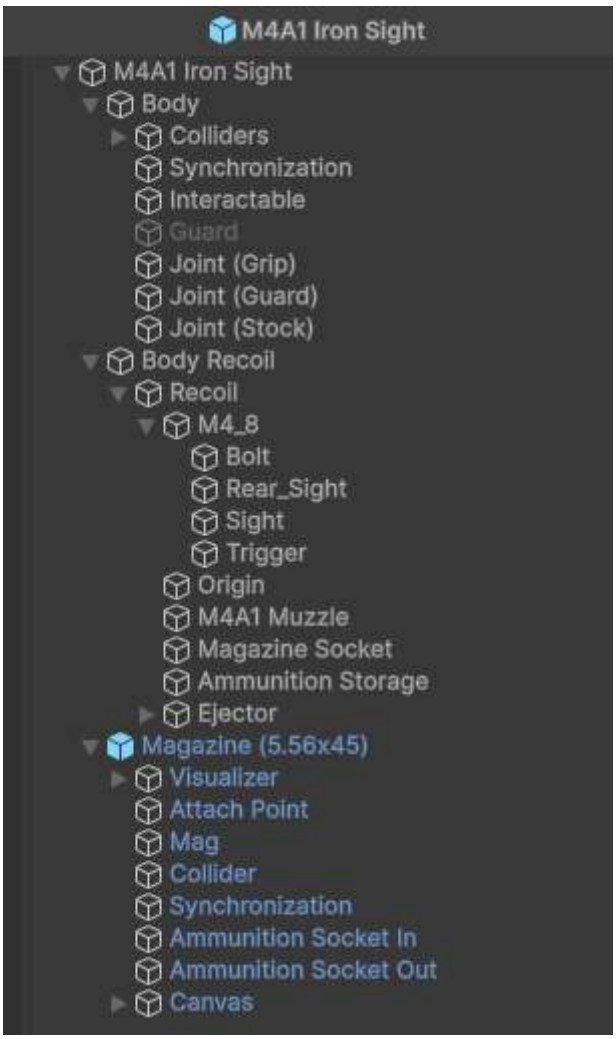

Рисунок 3.1– Ієрархія об’єкта зброї із системою віддачі
(не використовується у фінальній версії проєкту)

Зважаючи на це, було вирішено відмовитись від такого підходу, що в свою чергу дозволило значно знизити ієрархічну складність зброї, знизило непередбачуваність її поведінки та в достатній мірі полегшило хід розробки. Втім, систему віддачі зброї можна реалізувати й іншим чином. Є можливим реалізація шляхом дерев анімації та скриптова, тобто процедурна, поведінка.

При використанні компоненту XR Socket Interactor було виявлено наступну проблему: при зчеплюванні інтерактивного об’єкту із сокетом вони

починають хаотично переміщуватись (тобто «трястись») за умови, коли їх колайдери, тобто геометрія зіткнень, натрапляють одна на одну. Такого роду проблема вирішується за рахунок ігнорування колайдерів зчіплюваного об’єкта (наприклад, магазин) по відношенню до цільового об’єкта (наприклад, зброї), з яким йде зчеплення, та, за необхідності, створенням таких самих колайдерів в цільовому об’єкті та синхронізацією їх позицій кожного кадру, втім подібний підхід ускладнюється, наприклад, при наявності можливості додавати та виймати патрони з магазину – виникає потреба робити це рекурсивно. Окрім того, колайдери на зчіплювальному об’єкті можуть змінюватися, додаватися та видалятись – це також слід враховувати.

В проєкті на рушії Unity було створено компонент SocketItem, що використовує теги, які представлені у вигляді набору даних ScriptableObject, він працює у тандемі з компонентом SocketFilter. В такий спосіб досягається фільтрація об’єктів, що можуть взаємодіяти один з одним, тобто таким чином магазин може бути зчеплений лише зі зброєю, що під нього підходить, а патрон лише з магазином, що сумісний з ним. Окрім цього було прописано логіку, що відповідає за ігнорування предметів, що вже знаходяться в сокеті, таким чином в магазин, що знаходиться в зброї, неможливо додати ще один патрон.

Окрім цього, було створено компонент InteractableJoint, що дозволяє імітувати об’єкт на пружині, він має параметри базового видовження, максимальної дистанції видовження, маси та коефіцієнта пружності. Компонент також оброблює події введення та виведення із стану спокою, а також події видовження чи скорочення від певної дистанції. У другому розділі цієї роботи наведено формули, за якими розраховується сила, прискорення та позиція такого об’єкту у просторі.

Рисунок 3.2 – Магазин під час зчеплення зі зброєю

В проєкті також було реалізовано внутрішні механізми зброї. Так було створено компонент AmmunitionStorage який застосовується до магазинів та патроннику, він містить та відслідковує події під’єднання та від’єднання батьківського та дочірнього компоненту AmmunitionStorage, події викиду патрону, зміни стану та події настання стану заповнення чи пустоти сховища. У випадку з патронником ми маємо компонент AmmunitionStorage з місткістю у один патрон, де викид патрону з нього підіймає подію, що включає інформацію про те, яким чином було викинуто патрон (шляхом пострілу чи простим відкиданням цілого патрону під час пересмикування затвору), таким чином компонент AmmunitionEjector створює або нову пусту оболонку патрону, або цілий патрон в залежності від інформації, що міститься в події. Компонент AmmunitionLoader заряджає патрон кожного разу, коли відбувається подія викидання патрону, та кожного разу, коли компонент InteractableJoint об’єкту ручки затвору виконує подію повернення до звичайного стану RestEntered.

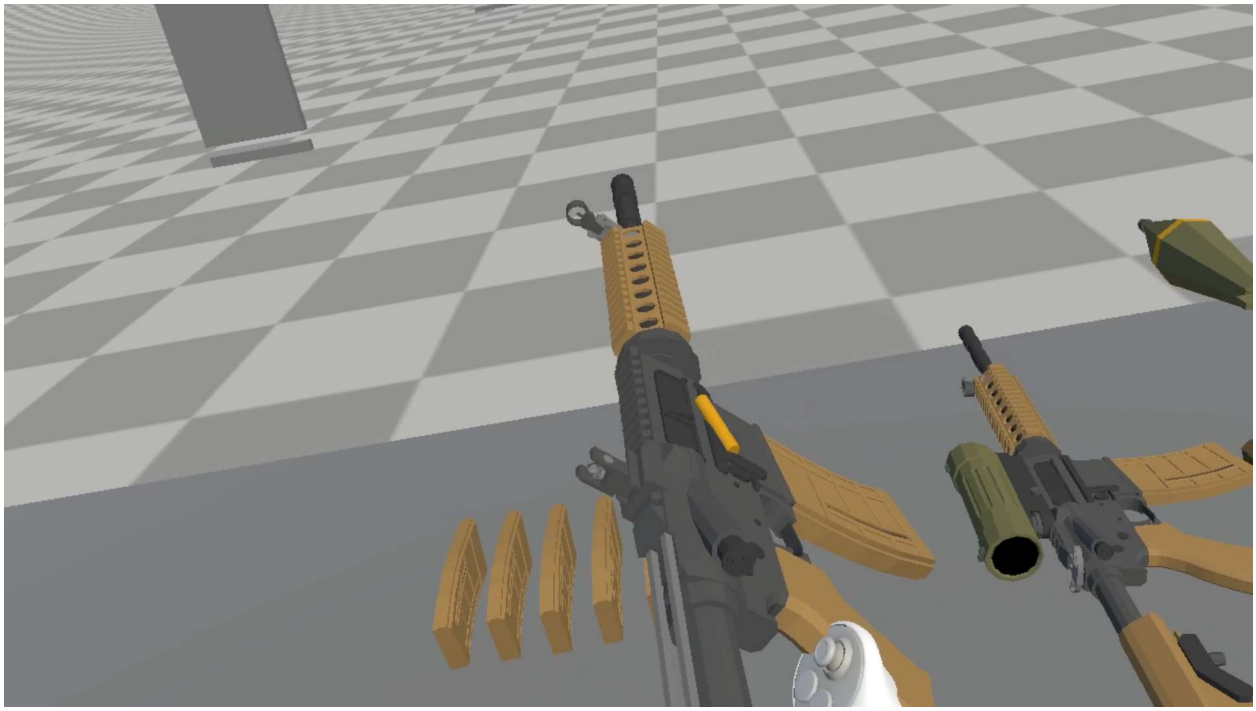

Рисунок 3.3 – Відкидання патрону шляхом пересмикування затвору

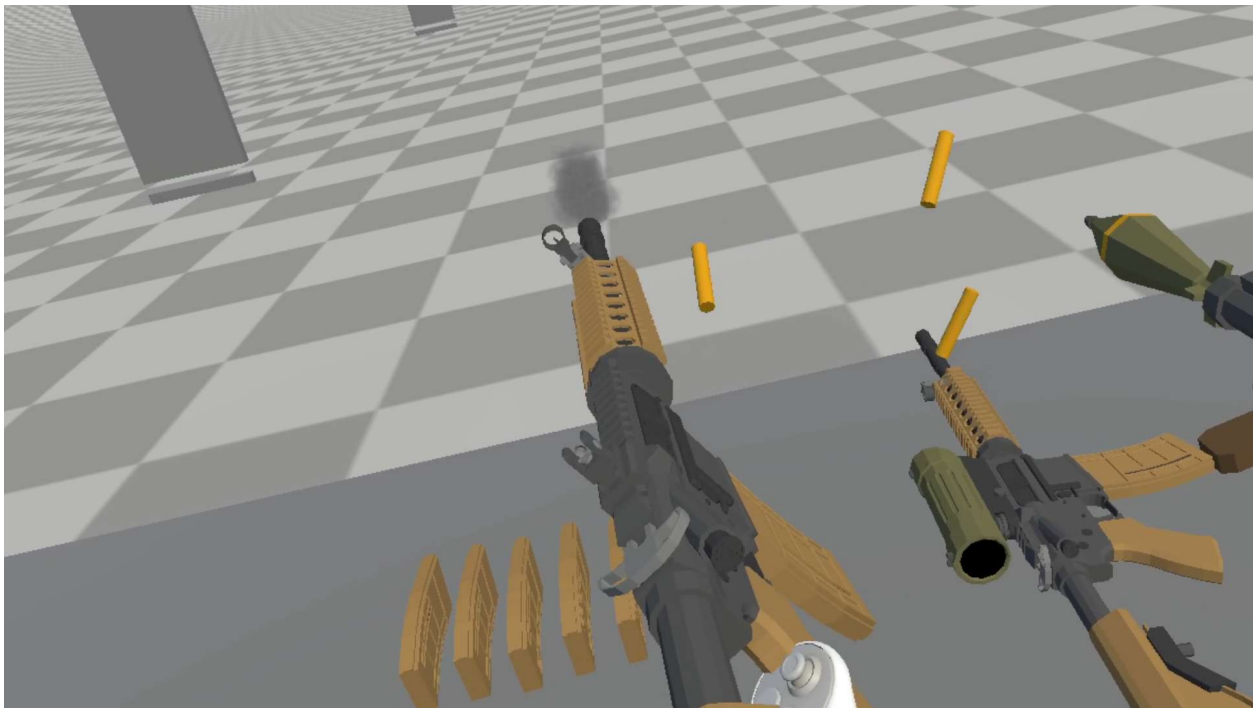

Рисунок 3.4 – Відкидання пустих оболонок патронів під час стрільби

Гравець має можливість додавати та виймати патрони з магазину. Перший патрон сконфігурований скриптом в такий спосіб, що з ним можна взаємодіяти руками гравця, в той же час інші візуалізовані патрони

сконфігуровані скриптом не бути фізичними та інтерактивними. Існує два сокети (компонент XRSocketInteractor) – першим є вихідний сокет (Socket Out), який має задачу візуалізувати перший патрон в магазині, а також надати можливість взаємодіяти із ним лише задля досягнення мети мати можливість виймати цей патрон з магазину, в той самий час другий сокет (Socket In) не містить в собі нічого і кожного разу, коли новий та сумісний патрон додається до магазину, сокет видаляє цей патрон та візуалізує новий поточний стан магазину. Кожного разу, коли магазин зчіплюється чи відчеплюється від іншого об’єкту, тобто зброї, він блокує чи, відповідно, розблоковує другий сокет (Socket In); а також перевізуалізовує поточний стан магазину через компонент AmmunitionVisualizer, таким чином запобігаючи можливості виймати патрон із зчепленого магазину, а також спрощуючи завдання синхронізації та обробки зіткнення з екземплярами патронів магазину. Такий підхід є в певній мірі важким з точки зору оптимізації гри, візуалізацію можна спростити шляхом перевикористання візуалізованих патронів замість їх видалення кожного оновлення стану магазину.

В грі також присутні портали та можливість прямого керування квадрокоптерами та мультикоптерами. Так компонент PortalHandler підписується на події вводу гравця через компонент PortalProvider, а компонент Portal реєструється чи відписується від компонента PortalHandler щоразу, коли гравець входить чи виходить з порталу відповідно. Водночас компонент DroneProvider відповідає за взаємодію та керування квадрокоптерами та мультикоптерами, підписується на події вводу та, під час керування, передає ввід гравця до компоненту DroneController.

Взаємодія зі зброєю, а саме такі дії як натиснення спускового гачка, перемикання режимів вогню та звільнення магазину, здійснюється за допомогою компонентів WeaponInteractor та WeaponInteractable. Так компонент WeaponInteractable реєструється чи відписується від компоненту WeaponInteractor коли їхня геометрія перетинається чи перестає перетинатися відповідно. Компонент WeaponInteractor містить об’єкти InputActionProperty,

що призначені для обробки вводу гравця, а компонент WeaponInteractable містить флаги, тобто булеві значення, того, чи є цей інтерактивний об’єкт спусковим гачком, чи дозволяє він перемикати режим вогню та чи дозволяє він звільняти магазин.

Таким чином було успішно застосовано технології віртуальної реальності у проєкті даної кваліфікаційної роботи, окрім цього були забезпечені інтерактивність та взаємодія з об’єктами даного віртуального середовища.

3.3 Візуалізація та графічне представлення

Задля потреб візуалізації було вирішено створити тривимірну модель квадрокоптера – ця задача включає створення геометрії, розділення геометрії на окремі об’єкти, створення UV розгортки, текстур та матеріалів. Розмах моделі має становити приблизно 1 метр, така вимога полегшує масштабування та використання моделі в самому рушії. В якості взірця для натхнення було використано зображення квадрокоптера, що наведено на рисунку 3.5 [24].

Рисунок 3.5 – Взірець моделі квадрокоптера

Для потреб моделювання було використано засоби програми Blender версії 4.3.2 – ця програма є безкоштовною, має відкритий вихідний код та працює на операційних системах сімейства Windows, macOS, Linux, BSD, Haiki та IRIX. Вона використовується для створення анімаційних фільмів, візуальних ефектів, мистецтва, тривимірних друкованих моделей, анімаційної графіки, інтерактивних 3D-додатків, віртуальної реальності, а також для створення моделей, зокрема для відеоігор Під час створення моделі було враховано систему координат, що використовується у програмі, тож таким чином використовується система координат, утворена за правилом правої руки, де вісь Y+ вказує по напрямку вперед, а вісь Z+ вказує вгору. Така система координат відрізняється від тієї, що використовується у рушіях Unity та Godot.

Під час етапу створення геометрії, використовувалася досить проста техніка моделювання, яка використовувала у якості початкової геометрії примітиви. Таким чином корпус, рама та лопаті квадрокоптера брали свій початок з геометрії прямокутного паралелепіпеда, мотор – з циліндра, а кріплення лопатей з об’єднання циліндра з прямокутним паралелепіпедом. В режимі редагування сітки (Edit Mode) застосовувалися інструменти (Edge ▸ Bevel Edges) для згладжування граней, (Face ▸ Inset Faces) для виокремлення матеріалів конструкції на лопатях, (Edge ▸ Subdivide Edge-Ring) та (Edge ▸ Edge Slide) для виокремлення матеріалу індикаторів на рамі. Кінцева геометрія складалася з наступних об’єктів: корпус, дві рами («крила»), чотири мотори, чотири кріплення лопатей та вісім лопатей – такий підхід дозволив модифікувати позицію, обертання та розмір кожного об’єкту окремо, разом із врахуванням ієрархії цих об’єктів. На рисунку 3.6 зображено геометрію та сітку модельованого об’єкту.

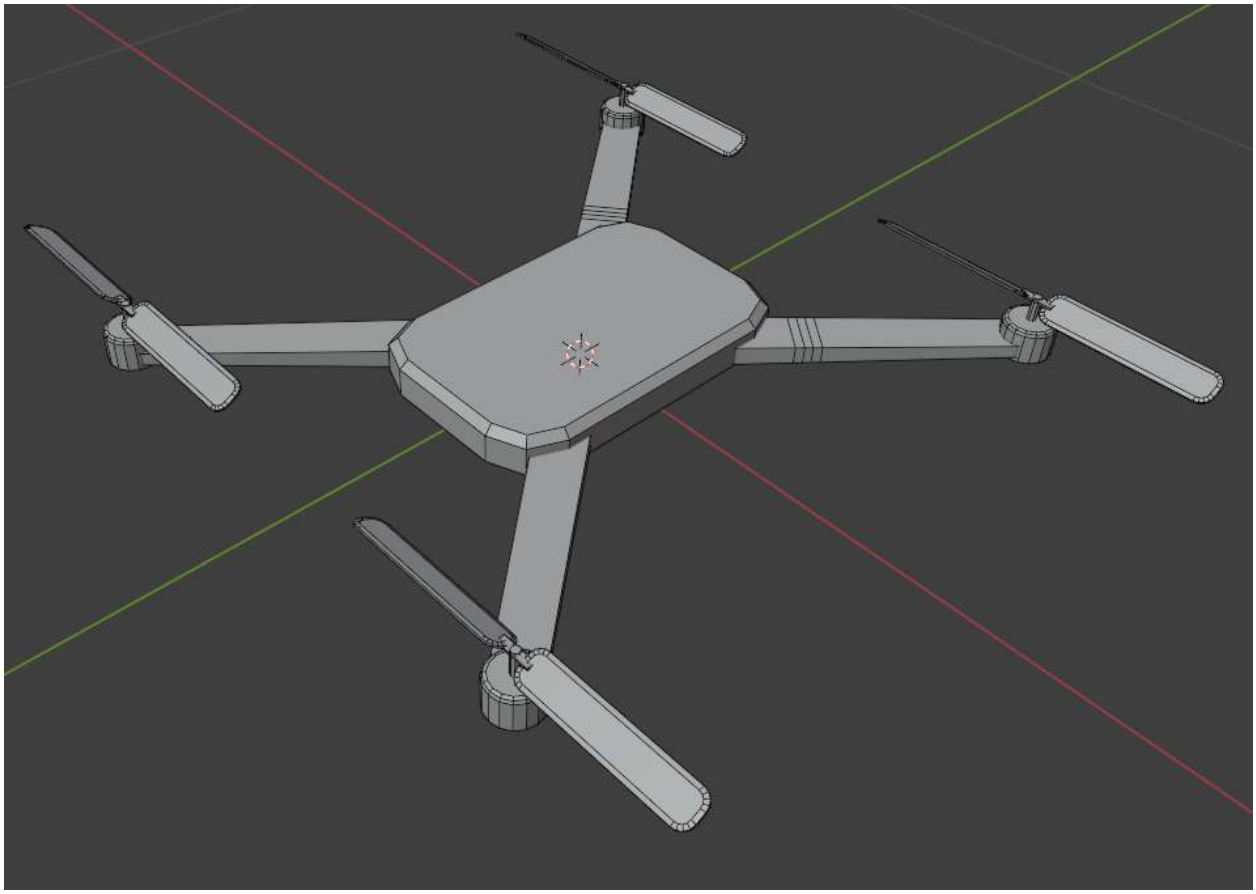

Рисунок 3.6 – Геометрія та сітка моделі квадрокоптера

Наступним етапом після цього стало створення UV розгортки моделі квадрокоптера. UV розгортка являє собою двовимірну мапу, яка по своїй суті є проєкцією сітки тривимірної моделі на плоске двовимірне зображення. Літери U і V позначають осі координат площини розгортки, оскільки літери X, Y і Z вже застосовуються для позначення просторових координат [25]. Для створення розгортки застосовувалася вже існуюча розгортка базових примітивів, з яких було створено об’єкт, якщо вона добре підходила під ті трансформації геометрії, які були застосовані під час етапу моделювання, в інших випадках використовувалися засоби інструментів програми Blender (Edge ▸ Mark Seam) та (UV ▸ Unwrap ▸ Minimum Stretch). Для виділення, переміщення та масштабування окремих островів геометрії на розгортці було використано інструменти (Select ▸ Select Linked ▸ Linked), Move та Scale відповідно у вікні UV Editing.

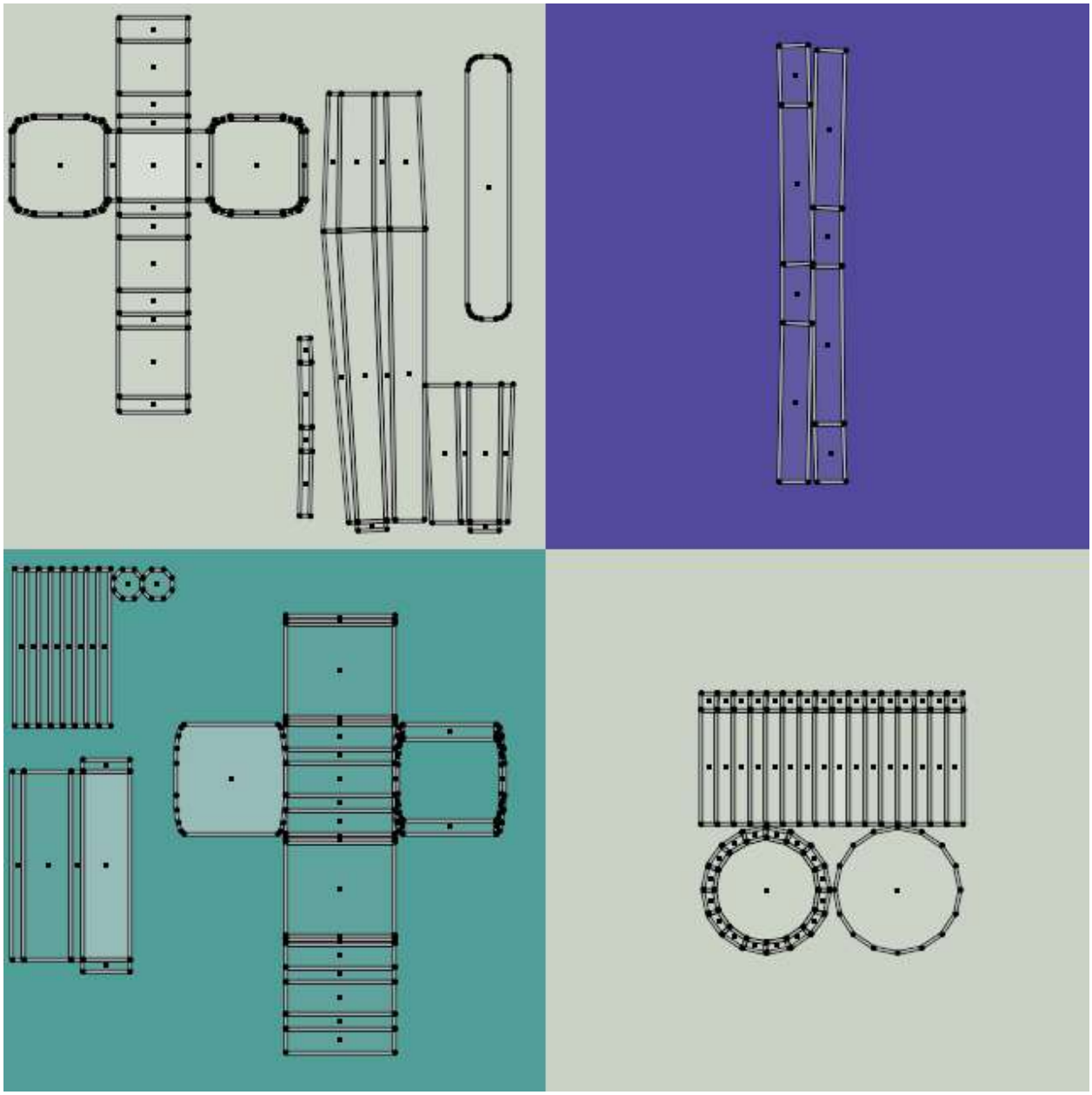

Рисунок 3.7 – UV розгортка моделі квадрокоптера

Наступним етапом після цього стало створення текстур та матеріалів 3D моделі. В даному проєкті використовується PBR (Physically Based Rendering) підхід до відображення об’єктів – тобто іншими словами застосовується фізично коректна модель відображення матеріалів. Особливістю такої моделі моделі є принцип «блищить все», дотримання закону збереження енергії, можливість візуалізації як поверхностей, так і об’ємів (лінз, каустика, розсіювання частинок, атмосферні ефекти) та, як можна побачити з назви, її фізична коректність.

При створенні моделі квадрокоптера було використано так звані текстурні атласи (Texture Atlas) – за своєю суттю вони є зображеннями, що містять у собі декілька менших зображень. В даному випадку зображення є достатньо малим та містить у собі окремі пікселі, кожен з яких позначає свій окремий матеріал поверхні. Таким чином ми маємо три текстури: Albedo, Metallic, яка одночасно слугує і мапою Roughness у випадку застосування рушія Godot, та Emmision – розмірами 8×8 пікселів кожна. Зазначимо, що кінцева модель квадрокоптера використовує лише один блок цього зображення розміром 2×2 пікселі, таким чином використовується 4 різних матеріали для корпусу та пофарбованої частини лопатей, індикаторів, металевої частини лопатей, моторів відповідно. Таким чином у випадку, коли виникне потреба використати такий самий квадрокоптер, але з іншим забарвленням, є можливим скопіювати цю модель та змістити UV розгортку в інше, невикористане місце текстури – це дозволить використовувати один і той самий матеріал для, в даному випадку, 16 забарвлень моделі. Таким чином використовується лише один виклик на відмальовку (Drawcall) для одночасно декількох забарвлень моделі, що в купі з малим розміром текстури є вигідним з точки зору оптимізації. В такий спосіб можна використати набагато більший за розмірами текстурний атлас, який в деякому сенсі можна вважати палітрою матеріалів, та використати його для усіх або більшості тривимірних моделей у проєкті, втім зазначимо, що для сучасних комп’ютерів така техніка не дасть помітного приросту частоти кадрів. На рисунку 3.8 наведено текстурний атлас для слоту матеріалу Albedo квадрокоптеру.

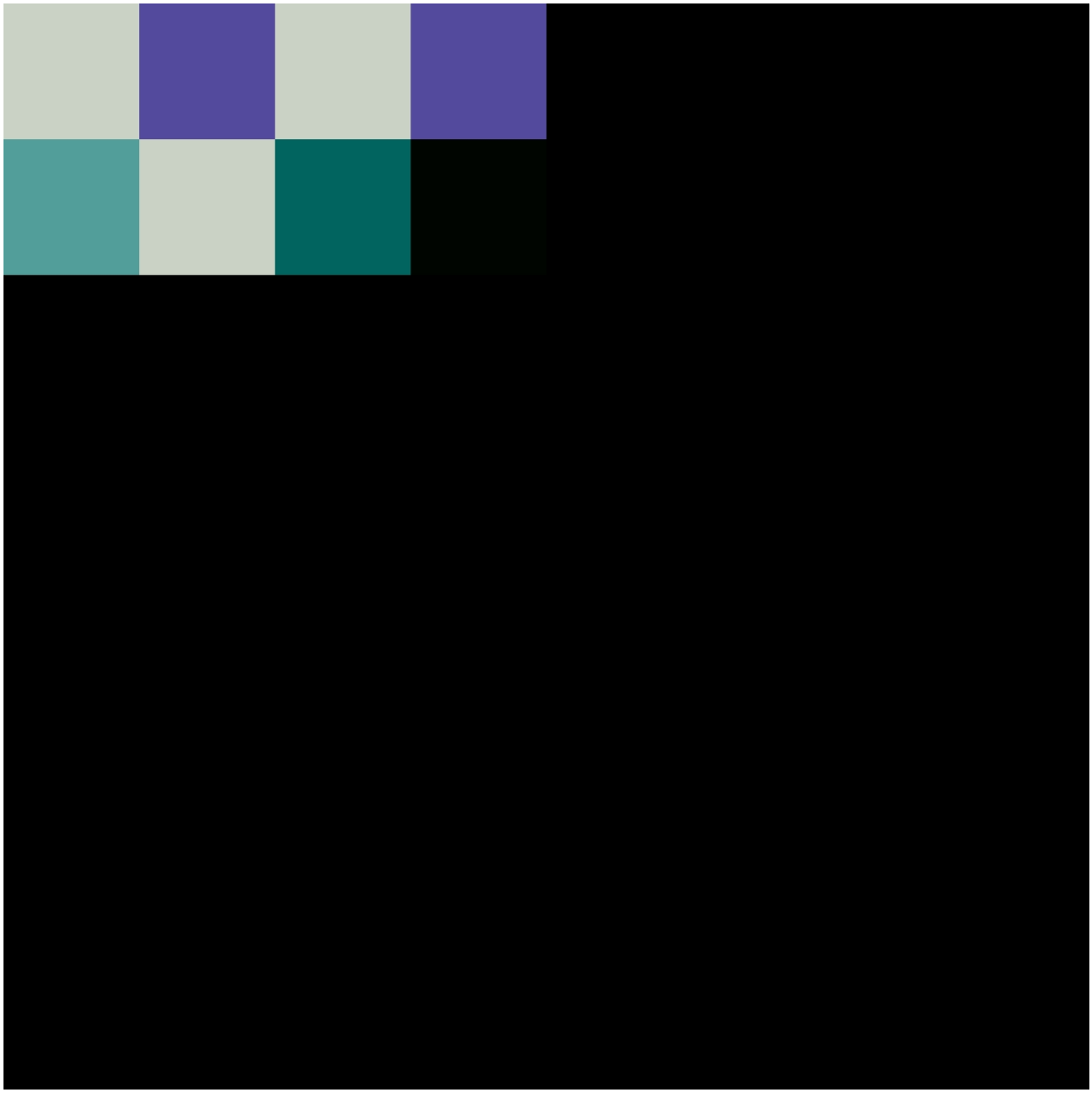

Рисунок 3.8 – Текстура Albedo матеріалу моделі квадрокоптера

На верхньому лівому куті рисунка можна побачити два блоки розміром 2×2 пікселі – вони є двома окремими забарвленнями моделі, інші ж блоки текстури Albedo є невикористаними і, відповідно, мають чорний колір.

Матеріал пластику корпуса та моторів має значення Metallic 0% та Roughness 50%, індикаторів – 0% та 20%, металевої частини лопатей – 95% та 10%. Відповідні значення містяться у каналі Red та Green у RGB просторі, така мапа добре підійде до застосування у рушії Godot. Значення кольору індикаторів в мапі Emission дорівнюють (82; 72; 156) у RGB просторі.

Для того, щоб перевести Metal текстуру, призначену для рушія Godot, у формат, що підходить під рушій Unity, необхідно відкрити його в редакторі зображень GIMP. В ньому робимо дублікат основного шару зображення (вікно Layers ▸ Create duplicates of selected layers and add them to the image), обираємо новостворений шар та витягуємо канал Green цього RGB зображення (Colors ▸ Components ▸ Extract Components), де обираємо RGB Green та Invert component, далі додаємо до основного шару зображення маску (вікно Layers ▸ Add Layer Masks... ▸ White (full opacity)), далі копіюємо новостворений шар (CTRL + C), обираємо створену маску, вставляємо (CTRL + V) та закріплюємо його (вікно Layers ▸ Anchor the floating mask), вимикаємо усі шари крім основного та експортуємо зображення, наприклад, у формат TGA (.tga). Тепер ми можемо використовувати таку текстуру у якості Metallic Map шейдеру «Universal Render Pipeline/Lit». Зазначимо, що зазначені перетворення зображення залишають по собі канал Green без змін, що не є оптимальним з точки зору компресії та використання ресурсів.

Після проходження наведених етапів, модель можна експортувати у формати glTF 2.0 (.glb) чи FBX (.fbx), які добре підходять для рушіїв Godot чи Unity відповідно. Далі необхідно створити матеріали, в слоти текстур яких занести відповідні зображення у форматі Truevision Advanced Raster Graphics Adapter (.tga) чи Portable Network Graphics (.png). Наступним етапом є налаштування об’єкту у редакторі відповідно рушія, застосування скриптів тощо. На рисунку 3.9 наведено кінцевий вигляд об’єкту у рушії Godot. Відзначимо, що цікавою особливістю рушія Godot, яка вигідно вирізняє його на фоні конкурентів, є можливість генерування довільного тривимірного тексту з ефектом глибини прямо у редакторі цього рушія.

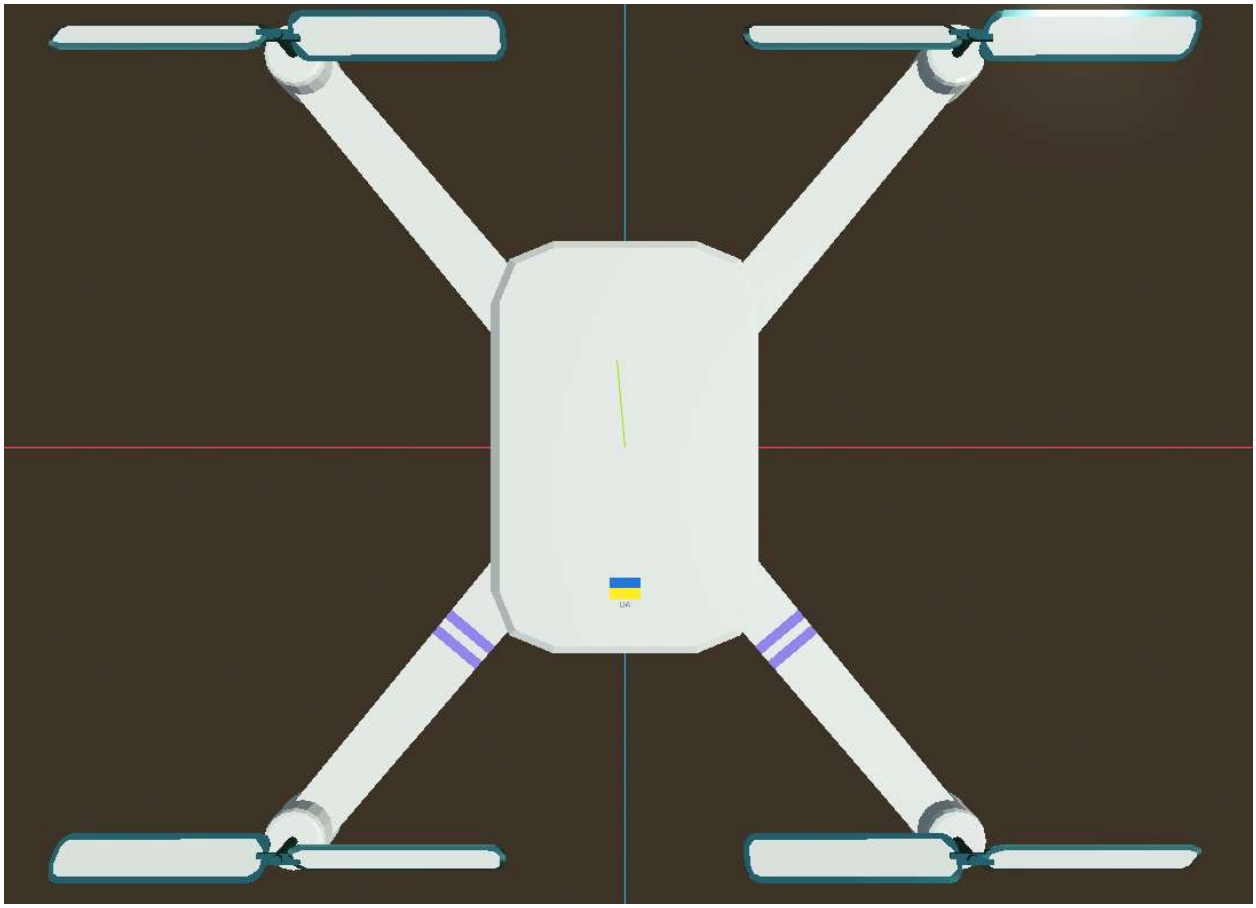

Рисунок 3.9 – Вигляд квадрокоптера з емблемою у рушії Godot

3.4 Інструкція користувача

Для запуску проєкту у рушії Unity, спершу необхідно завантажити та встановити програму під назвою Unity Hub, після чого запустити її та перейти до розділу (Installs ▸ Install Editor), де вибрати для встановлення Unity 6.0. Після цього перед встановленням можна вибрати компоненти підтримки цільових платформ, на які здійснюватиметься збірка проєкту, це також можна зробити із вже встановленим редактором, натиснувши на пункт меню (Manage ▸ Add Modules).

Після встановлення редактору переходимо до вкладки (Projects ▸ Add ▸ Add project from disk) та обираємо місцерозташування проєкту, обираємо встановлену версію рушія.

Для того, щоб запустити проєкт у VR режимі необхідно відкрити редактор рушія Unity, натиснути (File ▸ Build Profiles) обрати необхідну цільову платформу, в даному випадку Windows, натиснути на кнопку Switch Platform за її наявності у випадку, якщо поточна цільова платформа не відповідає обраній, натиснути кнопку Build та обрати цільову директорію. Збірка під Linux не є можливою через те, що рушій Unity, станом на час написання цієї роботи, не підтримує стандарт OpenXR під цю платформу – це також є одним з факторів, що вигідно вирізняє рушій Godot від Unity для даного проєкту.

Далі завантажуємо та встановлюємо програму та сервіс цифрової дистрибуції Steam. В ньому завантажуємо SteamVR та, для платформи Linux, обираємо (Steam ▸ Settings ▸ Compatibility ▸ Default compatibility tool ▸ Proton 10.0 або вище). Наступним кроком є додавання зібраного проєкту до бібліотеки: тиснемо (Games ▸ Add a Non-Steam Game to My Library... ▸ Browse...) та обираємо відповідний файл із розширенням .exe. Далі тиснемо по доданій грі у вкладці Library правою кнопкою миші, обираємо Properties та вмикаємо опцію (Shortcut ▸ Include in VR Library) та, для Linux, (Compatibility ▸ Force the use of a specific Steam Play compatibility tool) та обираємо Proton 10.0 або вище.

Наступним кроком, у випадку використання VR шоломів сімейства Meta Quest та операційної системи Windows, є встановлення програми Steam Link безпосередньо у VR шолом, після чого можна запускати проєкт у VR режимі. У випадку використання VR шоломів сімейства Meta Quest та операційної системи сімейства Linux, необхідно завантажити та встановити програму ALVR як на шоломі, так і на пристрої, з якого ведеться розробка, та налаштувати її.

У випадку використання рушія Godot та операційної системи сімейства Linux, зв’язка Steam + ALVR не є обов’язковою – так для стрімінгу достатньо мати лише встановлену та налаштовану програму WiVRn на VR шоломі та пристрої, з якого ведеться розробка.

Зазначимо, що збірка та запуск безпосередньо на самому VR шоломі також є можливою. Так для Meta Quest необхідно зробити збірку під платформу Android, отримати права розробника у самому VR шоломі та встановити отриманий .apk файл шляхом підключення VR шолома до комп'ютера, що містить необхідний .apk файл, та виконання інших додаткових маніпуляцій.

У VR режимі керування здійснюється наступними кнопками контролерів: вторинна кнопка лівого контролера (кнопка Y) – перехід у режим керування квадрокоптером та телепортація через портал; тригер лівого та правого контролера – постріл; первинні джойстики – переміщення та обертання; натиснення первинного джойстика – переключити режим вогню; первинна кнопка лівого та правого контролерів (кнопки X та A відповідно) – скид магазину.

Для того, щоб провести комп'ютерну симуляцію та зібрати дані з модельованих об'єктів, достатньо запустити проєкт в редакторі Unity, а у вікні Project перейти до директорії Scenes та відкрити сцену QuadcopterBenchmark. Після цього відключити об'єкти, що не беруть участі в симуляції, наприклад, об'єкт під назвою «Quadcopter_01 (Pedestrian)» та включити ті, що мають брати в ній участь, наприклад «Quadcopter_01 (Overfly)» – це дозволяє вимірювати продуктивність програми окремо для кожного з об'єктів. Якщо натиснути кнопку Play, компонент Drone Benchmark забезпечить генерацію завдань за допомогою менеджерів завдань в тій послідовності, що занесена до списку Task Managers цього компоненту, для усіх квадрокоптерів та мультикоптерів, що занесені до нього.

Компонент Drone Telemetry забезпечує збір та збереження телеметрії під час проведення симуляції, він прив'язується до кожного компонента Drone Agent окремо. Після збереження в директорії $MyDocuments/DroneWorld, тобто «C:\Documents and Settings*username*\My Documents\DroneWorld» на платформі Windows та «/home/*username*/DroneWorld» на платформі Linux, має з'явитись файл під

назвою «*Назва_завдання_Назва_симульованого_об'єкта*.json». Далі отримані файли можна порівнювати між собою у допоміжному програмному засобі під робочою назвою Grapher.

Якщо під час проведення симуляції перейти до вікна Scene та увімкнути опцію Draw Gizmos (остання на верхній панелі) – то можна побачити каркас сфери зеленого кольору та сіру лінію від симульованого об'єкта до неї – таким чином позначається точка, до якої намагається долетіти симульований об'єкт на поточний момент часу.

Коли симульований об'єкт повністю виконує завдання на певній місцевості, що включає обліт певної заданої кількості маршрутних точок, у вікні Console виводиться повідомлення у наступному форматі: «Drone Task (*Назва_завдання*) Complete». В той самий час повідомлення «Drone (*Назва_симульованого_об'єкта*) completed whole benchmark» позначає кінець симуляції для цього об'єкта, оскільки він виконав усі завдання обльоту заданої кількості маршрутних точок.

Для того, щоб додати нову потенційну точку маршруту, що буде використовуватися під час симуляції необхідно створити новий пустий ігровий об'єкт (GameObject ▸ Create Empty) та додати до нього компонент, що походить від класу Waypoint, наприклад компоненти Cargo Load Waypoint та Cargo Unload Waypoint. Такий об'єкт має бути включений до відповідного списку до довільного об'єкта, що містить компонент Waypoint Collection, на який в свою чергу має посилатися об'єкт з компонентом Drone Task Manager, на який в свою чергу посилається об'єкт з компонентом Drone Benchmark.

3.5 Проведення та результати практичних тестів

Проведення порівняння зібраних даних здійснюється за допомогою допоміжного програмного засобу під робочою назвою Grapher. Згаданий програмний засіб був створений в рамках написання даної кваліфікаційної

роботи, він генерує графіки та таблиці, які допомагають порівняти стратегії поведінки симульованих об’єктів між собою, отримуючи на вхід файли телеметрії. Такий програмний засіб має наступний синтаксис: dotnet run --
-p PATH1 PATH2 [... -p PATH1 PATH2]. Таким чином, команда dotnet run --
-p "Data/Droneville_City_Overfly.json" "Data/Droneville_City_Pedestrian.json"
-p "Data/Droneville_Village_Overfly.json" "Data/Droneville_Village_Pedestrian.js
on"
-p "Data/Droneville_Mountains_Overfly.json" "Data/Droneville_Mountains_Pedes
trian.json" є допустимою та виконає порівняння стратегій поведінки Overfly та Pedestrian на локаціях City, Village та Mountains.

На рисунку 3.10 та таблиці 3.1 наведено відношення швидкості проходження маршруту від одних однакових пар маршрутних точок до інших однакових пар маршрутних точок для стратегій поведінки Overfly та Pedestrian на міській місцевості. Зазначимо, що значення в таблиці сортуються за відношеннями швидкості проходження маршрутних точок, а не самими значеннями часу проходження, тобто сортування в таблиці відбувається за значеннями в дужках, а не поза ними. Окрім цього, значення a_{rate} стратегії поведінки Overfly становить 0,05 для усіх тестових випадків.

Криві розподілу позначають показник того, у скільки разів швидшим є симульований об’єкт, що використовує стратегію поведінки Overfly, у порівнянні з тим об’єктом, що використовує Pedestrian стратегію поведінки. Чим більшим є значення, тим більше стратегія поведінки Overfly є швидшою у досягненні маршрутних точок у порівнянні зі стратегією Pedestrian для об’єктів дослідження. Пунктирними вертикальними лініями позначається середнє значення в розподілі величини.

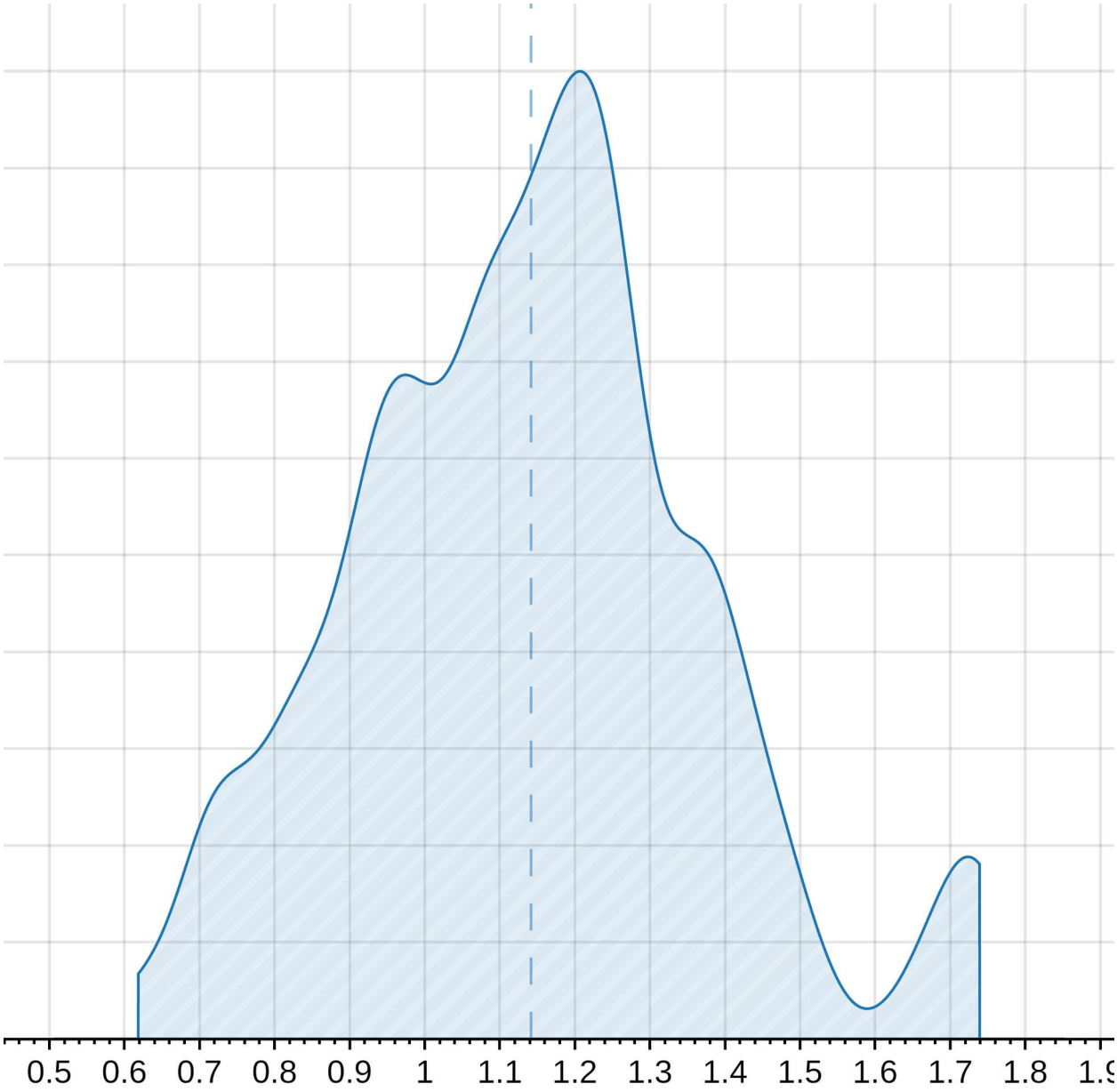

Рисунок 3.10 – Розподіл відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на міській місцевості

З рисунку розподілу відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на міській місцевості можна побачити, що стратегія Overfly виявилася у приблизно від 0,62 до 1,75 разів швидшою, з середнім значенням у приблизно 1,14. Рисунок 3.10 доповнюється даними на таблиці 3.1.

Таблиця 3.1 – Час та відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на міській місцевості

Значення	**Overfly**	**Pedestrain**
Середнє	29,22 (1,16x)	33,92
Медіана	30,91 (1,16x)	35,73
Мінімум	22,8	14,1 (1,62x)
1-й	22,8	14,1 (1,62x)
5-й	18,57	13,54 (1,37x)
10-й	32,72	26,91 (1,22x)
25-й	36,17	35,35 (1,02x)
75-й	36,38 (1,27x)	46,09
90-й	41,23 (1,41x)	58,25
95-й	37,47 (1,5x)	56,32
99-й	38,09 (1,74x)	66,2
Максимум	38,09 (1,74x)	66,25

Бачимо, що в міській місцевості стратегія поведінки Overfly має незначну перевагу над стратегією Pedestrian, хоч і не завжди. Так у 69,39% випадків стратегія Overfly виявилася швидшою. Звернемо увагу на одну цікаву деталь зі стратегії поведінки Overfly – так, симульовані об’єкти з цією стратегією, в даному випадку та за умов запасу висоти, кожні 100 метрів шляху знижуються на 5 метрів; враховуючи, що відстань між високоповерхівками зазвичай є відносно великою, симульовані об’єкти спочатку спускаються нижче, гальмують перед будинком на своєму шляху та знову набирають висоту – на це може йти значна частка часу при такій стратегії поведінки.

Зазначимо, що лише 98 з 100 значень пройшли фільтр, який відсікає часи проходження пар маршрутних точок для кожної зі стратегій поведінки менші за одну секунду; часи проходження, що не пройшли фільтр, імовірно мають однакові точки старту та призначення.

Далі на рисунках 3.11 та 3.12 наведено час на обчислення кроку симуляції для компонентів, що відповідають за обчислення, пов’язані зі стратегіями поведінки Overfly та Pedestrian, на міській місцевості. Зазначимо, що для усіх значень, пов’язаних з часом обчислень, застосовується виявлення та видалення аномалій – так усі значення менші за (5 процентиль) / 1,5 та більші за (95 процентиль) · 1,5 підлягають видаленню.

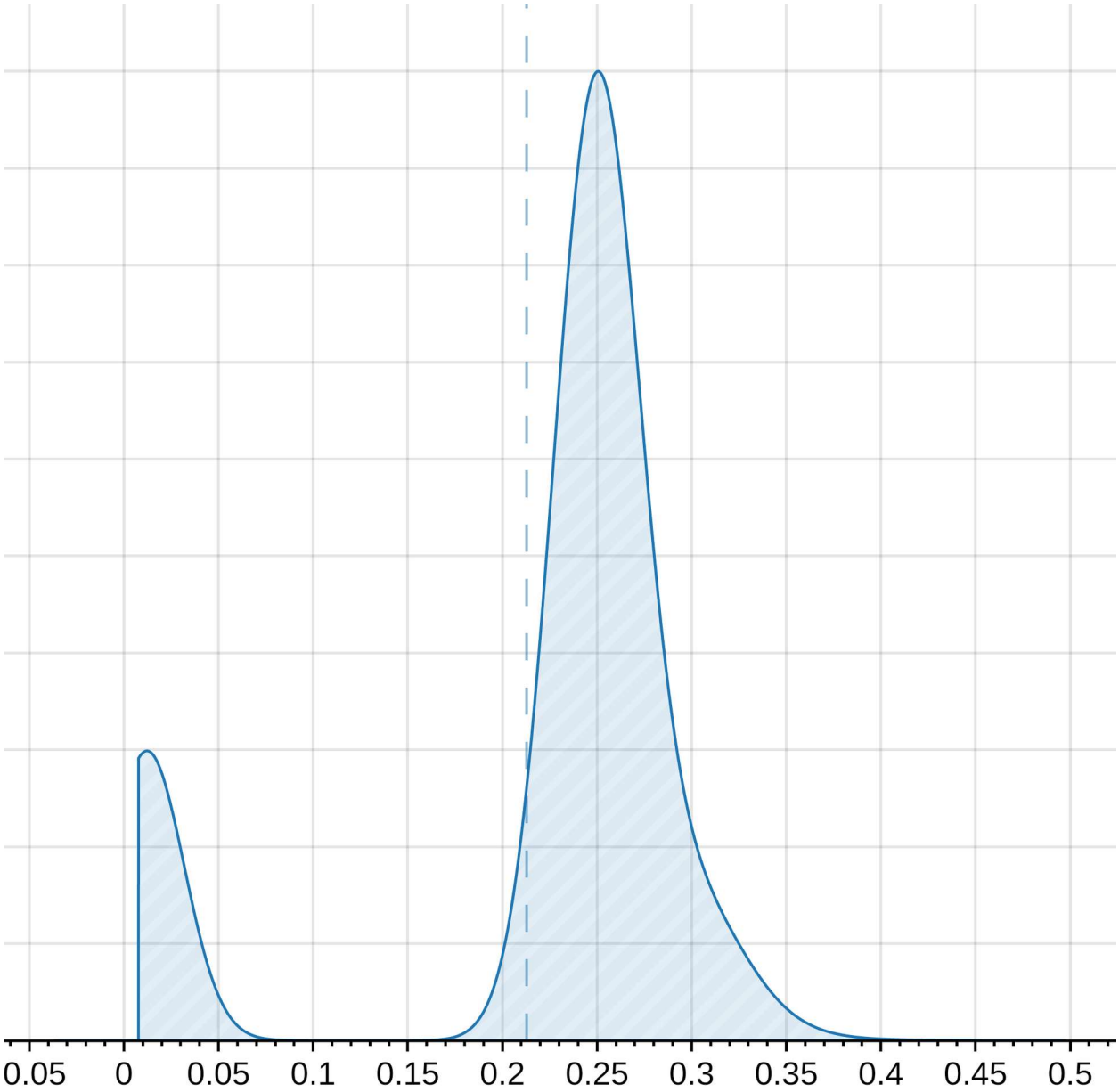

Рисунок 3.11 – Розподіл часу обчислення кроку оновлення компоненту, що відповідає за стратегію поведінки Overfly

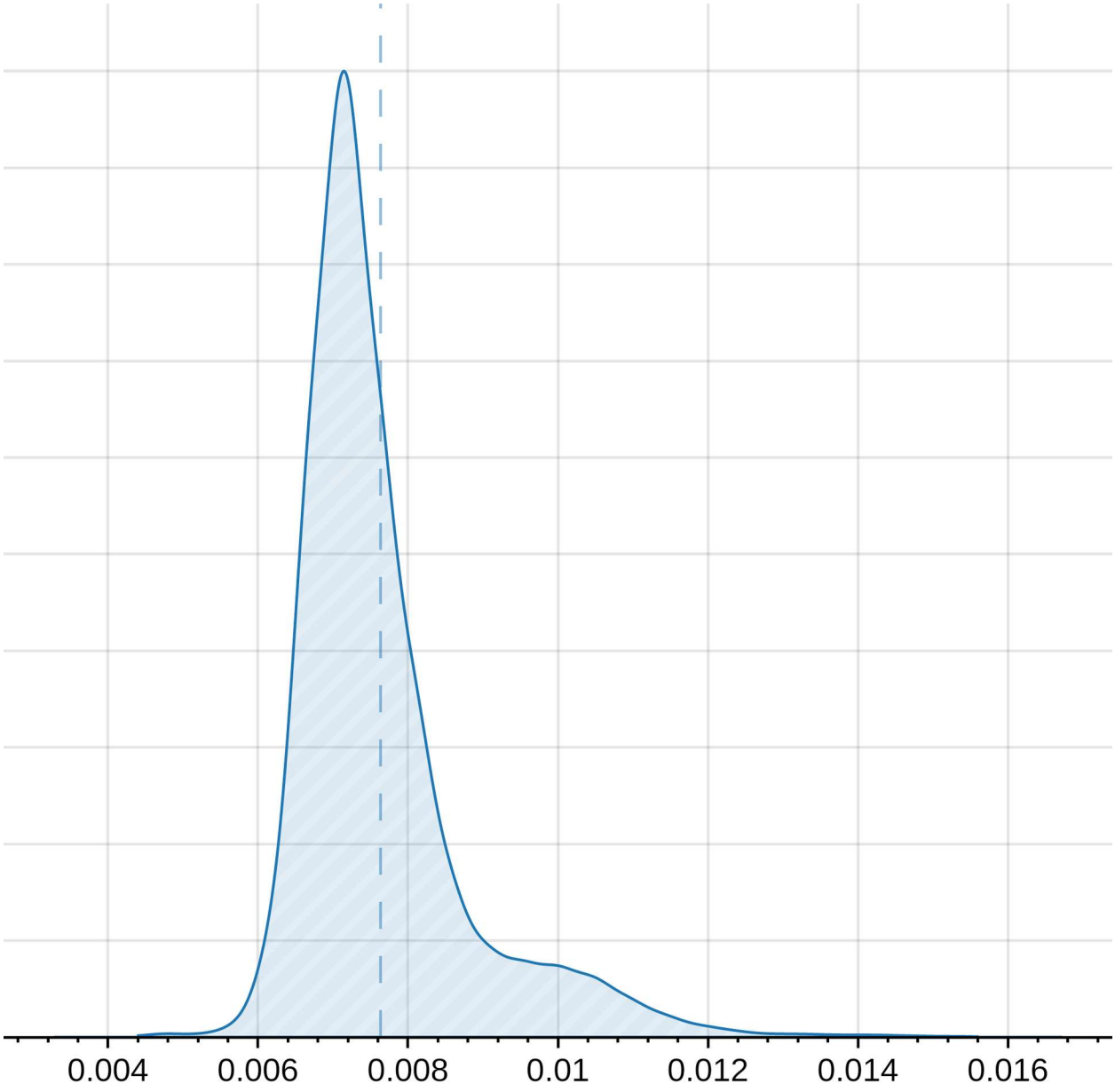

Рисунок 3.12 – Розподіл часу обчислення кроку оновлення компоненту, що відповідає за стратегію поведінки Pedestrian

Далі в таблиці 3.2 наведено час на обчислення кроку симуляції для компонентів, що відповідають за обчислення, пов’язані з обома стратегіями поведінки на міській місцевості.

Таблиця 3.2 – Час обчислення кроку оновлення для компонентів, що відповідають за стратегії поведінки Overfly та Pedestrian

Значення	**Overfly**	**Pedestrian**
Середнє	0,21	0,01 (27,87x)
Медіана	0,25	0,01 (34,05x)
Мінімум	0,01	0 (1,75x)
1-й	0,01	0,01 (1,79x)
5-й	0,01	0,01 (1,74x)
10-й	0,01	0,01 (1,79x)
25-й	0,23	0,01 (33,4x)
75-й	0,26	0,01 (32,51x)
90-й	0,28	0,01 (31,1x)
95-й	0,31	0,01 (30,41x)
99-й	0,33	0,01 (28,75x)
Максимум	0,45	0,02 (29,04x)

Бачимо, що в середньому час обчислення кроку симуляції для компонента, що обчислює стратегію поведінки Overfly, складає 0,21 мілісекунди, водночас для стратегії поведінки Pedestrian цей час складає лише 0,01 мілісекунди. Така сильна різниця в часі на обчислення пояснюється тим, що компонент, який реалізує стратегію поведінки Pedestrian, за задумом обчислює маршрут до наступної точки лише один раз – безпосередньо під час призначення наступної маршрутної точки, а усі інші обчислення, що виконуються під час обчислення кроку симуляції, є більш легкими та пов’язані з рухом та візуалізацією симульованого об’єкта.

Водночас, для компоненту, що відповідає за стратегію поведінки Overfly, кожного кроку симуляції необхідно обчислити висоти в, для даного випадку, радіусі 10 метрів з роздільною здатністю променів в один промінь на один метр. Такий підхід використовує значно більше обчислень, чим і

пояснюється час обчислення кроку симуляції для даного компонента у 0,21 мілісекунди.

На рисунку 3.13 та таблиці 3.3 наведено оброблений час обчислення кадру симуляції. В рушії Unity цей показник береться зі значення Time.deltaTime кожного кадру.

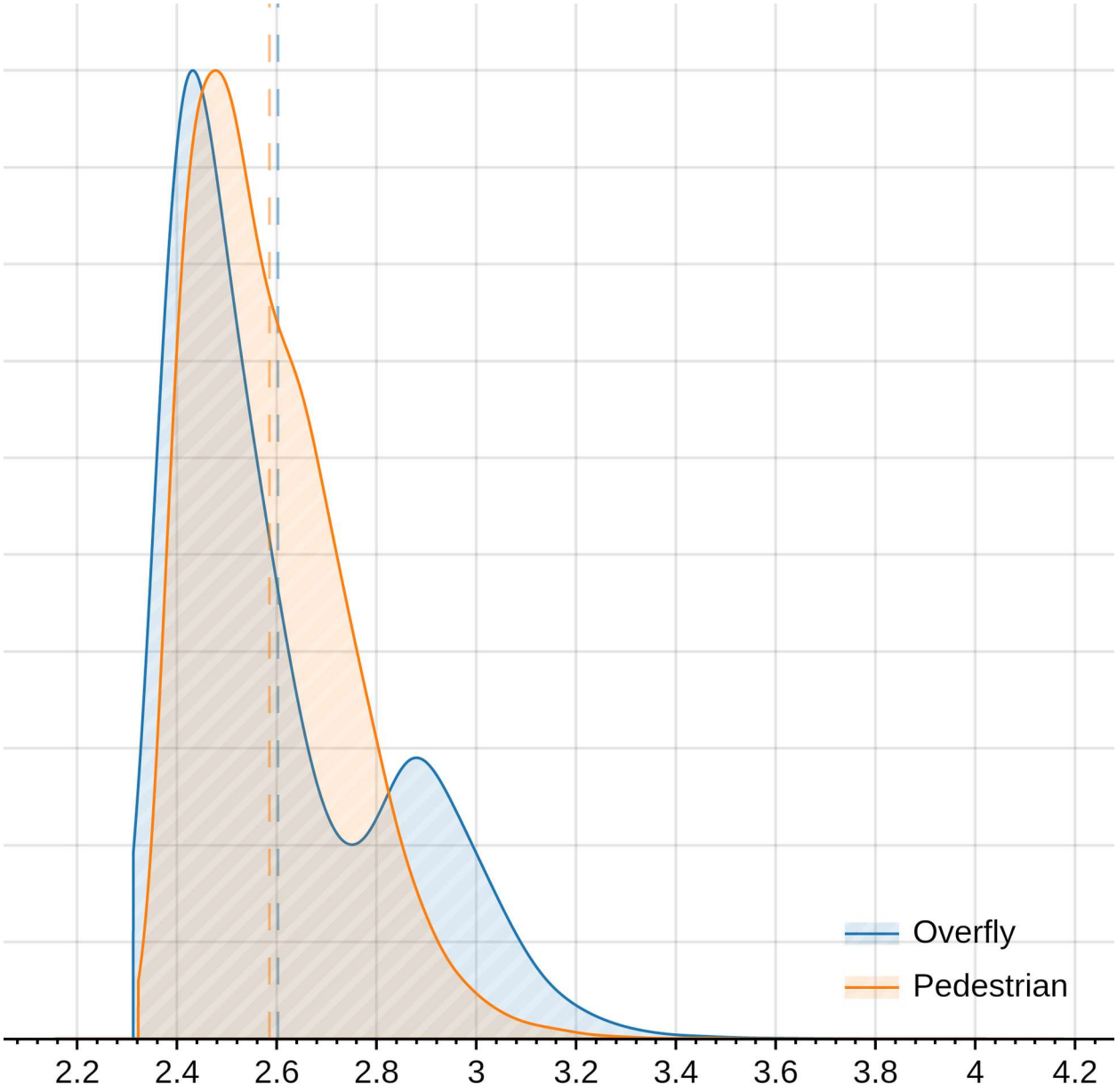

Рисунок 3.13 – Розподіл обробленого часу обчислення кадру симуляції

Таблиця 3.3 – Оброблений час обчислення кадру симуляції

Значення	**Overfly**	**Pedestrian**
Середнє	2,6	2,59 (1,01x)
Медіана	2,52 (1,01x)	2,56
Мінімум	2,31 (1x)	2,32
1-й	2,35 (1,01x)	2,37
5-й	2,37 (1,01x)	2,39
10-й	2,39 (1,01x)	2,41
25-й	2,43 (1,01x)	2,47
75-й	2,75	2,68 (1,03x)
90-й	2,94	2,79 (1,05x)
95-й	3,03	2,86 (1,06x)
99-й	3,19	3,02 (1,05x)
Максимум	3,87 (1,04x)	4,01

На графіку розподілу обробленого часу обчислення кадру симуляції видно, що для стратегії Overfly часто зустрічаються значення часу у приблизно 2,4 та 2,9 мілісекунд – таке явище пояснюється тим, що компоненти, що відповідають за стратегії поведінки Overfly та Pedestrian, мають фіксований крок оновлення, в той час як кадри симуляції не мають фіксованого кроку оновлення та змінюють одне одного якомога швидше. На рисунку 3.14 зображена схема виклику функцій оновлень Update та FixedUpdate в ігровому рушії Unity [26].

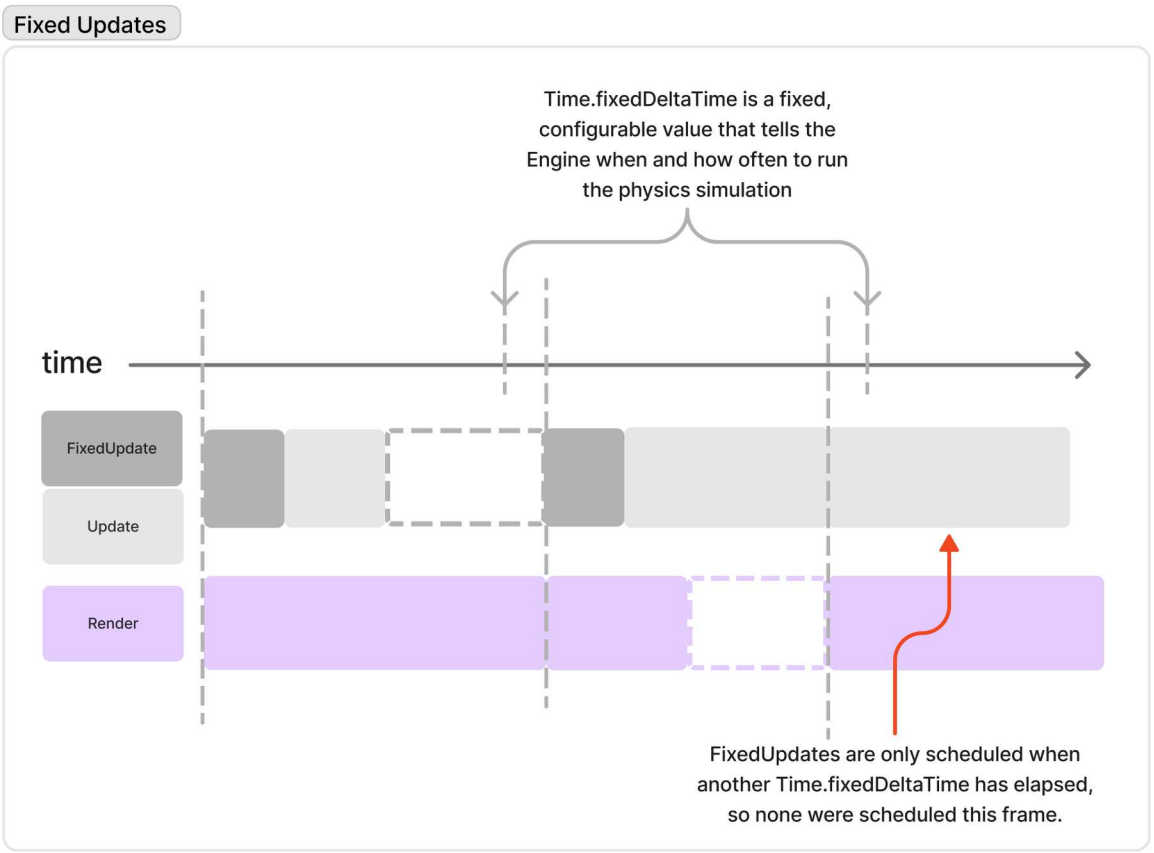

Рисунок 3.14 – Схема виклику функцій оновлень Update та FixedUpdate

Таким чином, можна сказати що в даному випадку на кожні декілька обчислених та відображених кадрів припадає одне оновлення компонентів, що відповідають за стратегії поведінки Overfly та Pedestrian, з фіксованим кроком обчислення. Якщо припустити, що другий локальний максимум функції щільності розподілу для стратегії Overfly відповідає кроку симуляції з фіксованим періодом виклику – то стратегія поведінки Overfly є на 300 - 400 мікросекунд повільнішою за Pedestrian з точки зору обчислень, що є в незначній мірі більшим значенням, ніж очікувалося, виходячи з попередніх графіків та таблиць.

Далі на рисунку 3.15 наведено розподіл необробленого часу обчислення кадру симуляції, виміри проводилися за допомогою класу Stopwatch.

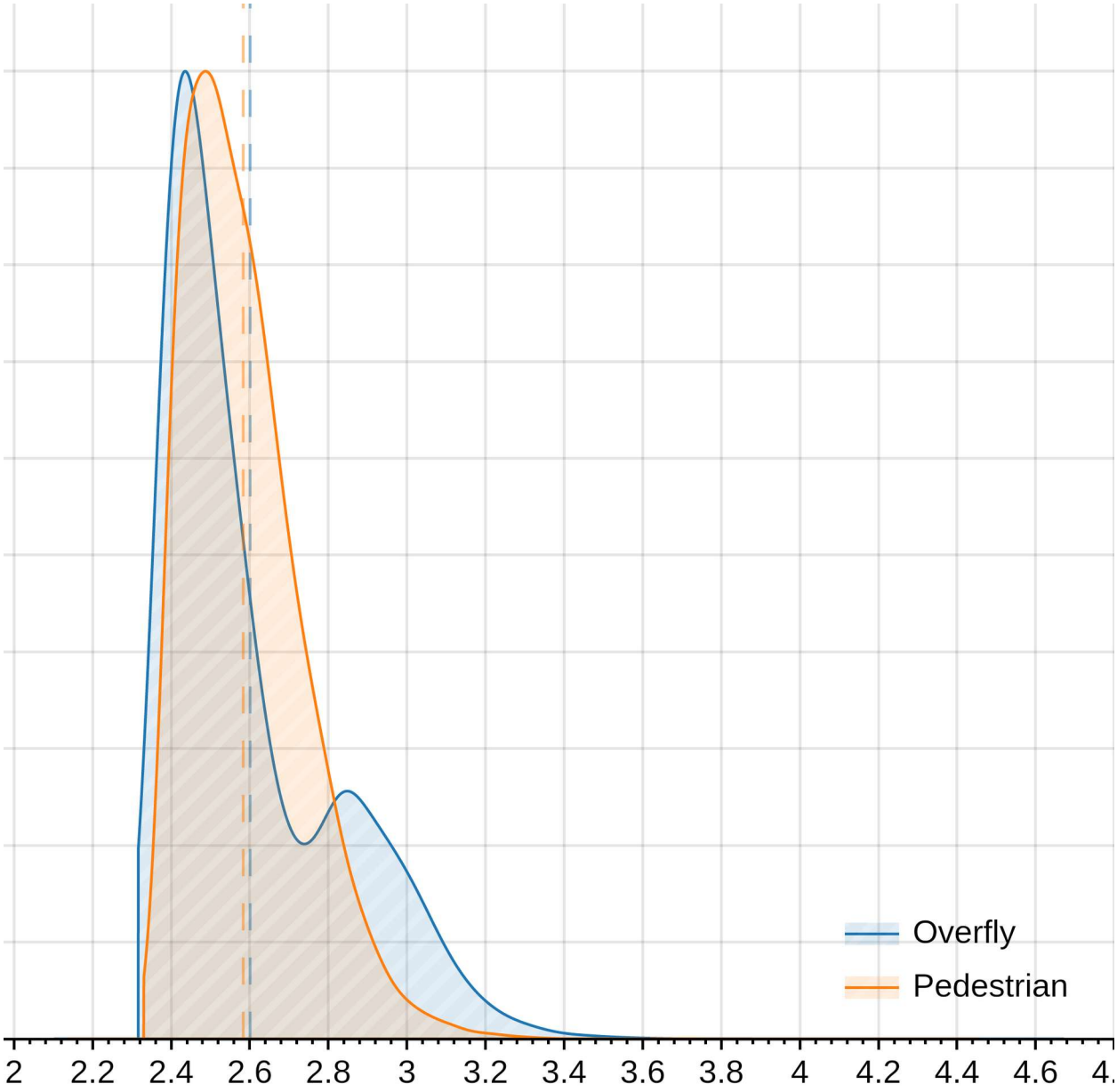

Рисунок 3.15 – Розподіл необробленого часу обчислення кадру симуляції

Далі в таблиці 3.4 наведено значення, пов’язані з необробленим часом обчислення кадру симуляції для обох стратегій поведінки на міській місцевості.

Таблиця 3.4 – Необроблений час обчислення кадру симуляції

Значення	**Overfly**	**Pedestrian**
Середнє	2,6	2,58 (1,01x)
Медіана	2,52 (1,01x)	2,56
Мінімум	2,32 (1,01x)	2,33
1-й	2,35 (1,01x)	2,37
5-й	2,37 (1,01x)	2,4
10-й	2,39 (1,01x)	2,42
25-й	2,43 (1,02x)	2,47
75-й	2,74	2,67 (1,03x)
90-й	2,95	2,78 (1,06x)
95-й	3,04	2,85 (1,07x)
99-й	3,22	3,02 (1,07x)
Максимум	4,46	4,26 (1,05x)

Суттєвих відмінностей між обробленим та необробленим часом обчислення кадру симуляції не спостерігається.

На рисунку 3.16 та таблиці 3.5 наведено відношення швидкості проходження маршруту від одних однакових пар маршрутних точок до інших однакових пар маршрутних точок для стратегій поведінки Overfly та Pedestrian на сільській місцевості. Зазначимо, що значення в таблиці сортуються за відношеннями швидкості проходження маршрутних точок, а не самими значеннями часу проходження, тобто сортування в таблиці відбувається за значеннями в дужках, а не поза ними.

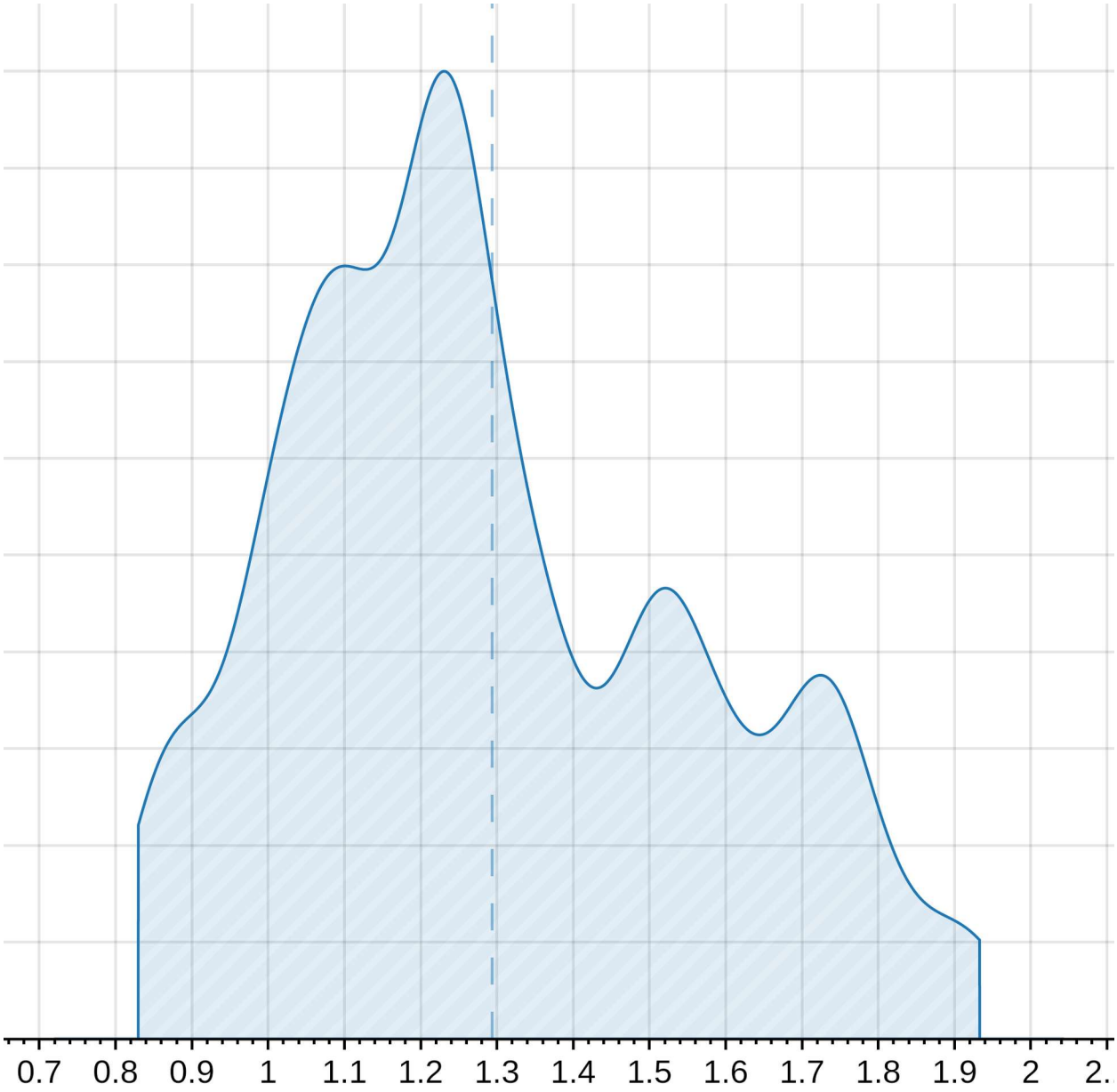

Рисунок 3.16 – Розподіл відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на сільській місцевості

З рисунку розподілу відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на сільській місцевості можна побачити, що стратегія Overfly виявилася у приблизно від 0,83 до 1,93 разів швидшою, з середнім значенням у приблизно 1,3. Рисунок 3.16 доповнюється даними на таблиці 3.5.

Таблиця 3.5 – Час та відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на сільській місцевості

Значення	**Overfly**	**Pedestrian**
Середнє	25,26 (1,3x)	32,9
Медіана	25,73 (1,24x)	31,86
Мінімум	18,43	15,29 (1,21x)
1-й	18,43	15,29 (1,21x)
5-й	25,87	22,71 (1,14x)
10-й	16,52	16,33 (1,01x)
25-й	20,93 (1,09x)	22,74
75-й	21 (1,5x)	31,44
90-й	21,54 (1,72x)	36,97
95-й	27,08 (1,76x)	47,62
99-й	24,29 (1,91x)	46,51
Максимум	26,71 (1,93x)	51,63

Бачимо, що в сільській місцевості стратегія поведінки Overfly має перевагу над стратегією Pedestrian, так у 86,46% випадків стратегія Overfly виявилася швидшою. Зазначимо, що лише 96 з 100 значень пройшли фільтр, який відсікає часи проходження пар маршрутних точок для кожної зі стратегій поведінки менші за одну секунду; часи проходження, що не пройшли фільтр, імовірно мають однакові точки старту та призначення.

Час на обчислення кроку симуляції для компонентів, що відповідають за обчислення, пов’язані зі стратегіями поведінки Overfly та Pedestrian, оброблений та необроблений час обчислення кадру симуляції на сільській місцевості великою мірою є однаковими та не мають суттєвих відмінностей від тих, що є актуальними на міській місцевості. Таким чином відповідні рисунки та таблиці не буде наведено в цій роботі.

На рисунку 3.17 та таблиці 3.6 наведено відношення швидкості проходження маршруту від одних однакових пар маршрутних точок до інших однакових пар маршрутних точок для стратегій поведінки Overfly та Pedestrian на гірській місцевості. Зазначимо, що значення в таблиці сортуються за відношеннями швидкості проходження маршрутних точок, а не самими значеннями часу проходження, тобто сортування в таблиці відбувається за значеннями в дужках, а не поза ними.

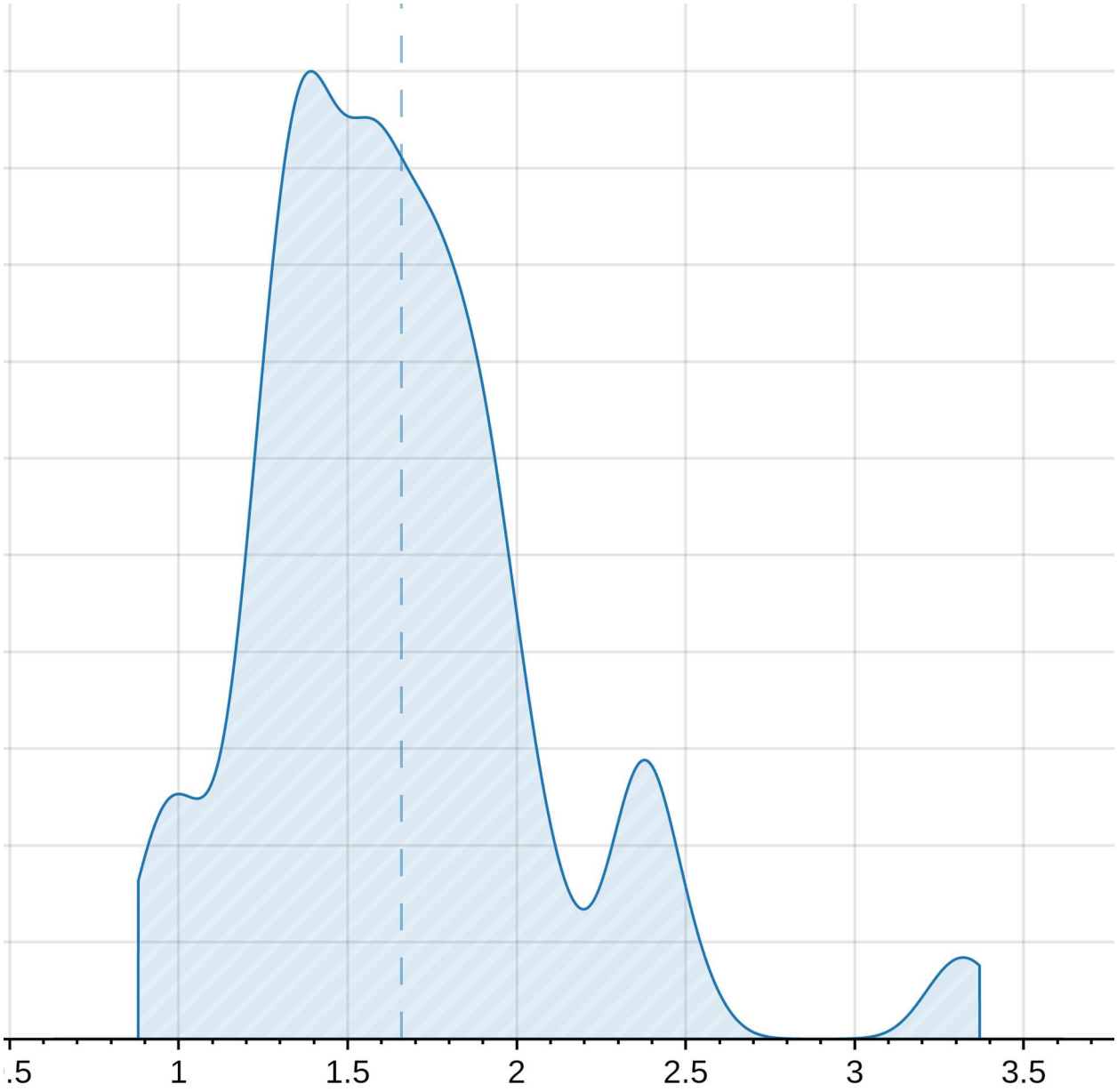

Рисунок 3.17 – Розподіл відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на гірській місцевості

З рисунку розподілу відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на гірській місцевості можна побачити, що стратегія Overfly виявилася у приблизно від 0,9 до 3,4 разів швидшою, з середнім значенням у приблизно 1,65. Рисунок 3.17 доповнюється даними на таблиці 3.6.

Таблиця 3.6 – Час та відношення швидкості проходження маршрутних точок між стратегіями поведінки Overfly та Pedestrian на гірській місцевості

Значення	**Overfly**	**Pedestrian**
Середнє	45,55 (1,63x)	74,31
Медіана	49,41 (1,59x)	78,49
Мінімум	28,94	25,48 (1,14x)
1-й	28,94	25,48 (1,14x)
5-й	27,83 (1,03x)	28,78
10-й	56,67 (1,24x)	70,43
25-й	27,68 (1,36x)	37,76
75-й	60,73 (1,86x)	113,13
90-й	31,72 (2,15x)	68,21
95-й	31,72 (2,39x)	75,87
99-й	18,73 (3,27x)	61,27
Максимум	27,83 (3,37x)	93,78

Бачимо, що в сільській місцевості стратегія поведінки Overfly має значну перевагу над стратегією Pedestrian, так у 95,24% випадків стратегія Overfly виявилася швидшою. Зазначимо, що лише 84 з 100 значень пройшли фільтр, який відсікає часи проходження пар маршрутних точок для кожної зі стратегій поведінки менші за одну секунду; часи проходження, що не пройшли фільтр, імовірно мають однакові точки старту та призначення.

Час на обчислення кроку симуляції для компонентів, що відповідають за обчислення, пов’язані зі стратегіями поведінки Overfly та Pedestrian, оброблений та необроблений час обчислення кадру симуляції на гірській

місцевості великою мірою є однаковими та не мають суттєвих відмінностей від тих, що є актуальними на міській місцевості. Таким чином відповідні рисунки та таблиці не буде наведено в цій роботі.

Для зниження обсягу обчислень компоненту, що відповідає за стратегію поведінки Overfly, можна знизити радіус обчислення висот, або знизити щільність променів. Також можна обчислити висоти лише один раз та запам’ятати усю карту висот на симульованій місцевості – це має значно знизити об’єм обчислень ціною більших витрат пам’яті та додатковими обмеженнями щодо площі та статичності симульованої місцевості.

В майбутніх версіях проєкту можна поглибити ідею порівняння моделей та стратегій поведінки об’єктів дослідження за рахунок застосування методів обробки зображень та вилучення необхідної допоміжної інформації [31, 33-40].

ВИСНОВКИ

Таким чином, у кваліфікаційній роботі досліджено стратегії руху таких об’єктів як квадрокоптери та мультикоптери, та вирішено такі завдання:

– проведено аналіз літературних джерел щодо використання квадрокоптерів чи мультикоптерів у сфері транспортування, доставки вантажів та їх стратегії поведінки, що дало можливість виявити сучасний стан дослідженої проблематики;

– сформовано покроковий алгоритм для кожної із запропонованих стратегій поведінки квадрокоптерів та мультикоптерів, що дало можливість запрограмувати кожну із вибраних стратегій;

– візуалізовано покроковий алгоритм для кожної із запропонованих стратегій поведінки квадрокоптерів та мультикоптерів блок-схемою чи іншим методом візуалізації, що дозволило наочно зобразити кожний крок та, за можливості, оптимізувати або удосконалити окремі кроки;

– розроблено програмний засіб, що дозволяє проводити моделювання об’єктів з різними стратегіями руху, збирати телеметрію та забезпечує інтерактивну симуляцію віртуального полігону із застосуванням технологій віртуальної реальності, це дозволило провести необхідне дослідження та задовільнити у цілому мету кваліфікаційної роботи;

– розроблено програмний засіб, що дозволяє проводити аналіз зібраної телеметрії, що дозволило провести необхідне дослідження та задовільнити у цілому мету кваліфікаційної роботи;

– візуалізовано отримані результати у вигляді графіків та таблиць, що дозволило провести аналіз отриманих результатів та зробити висновки.

У рамках кваліфікаційної роботи було проведено дослідження стратегій руху квадрокоптерів та мультикоптерів, а саме Pedestrain та Overfly стратегій поведінки, шляхом програмної реалізації засобу, що надає можливість проводити симуляцію об’єктів та збирати дані телеметрії.

Змодельовано тривимірну модель квадрокоптера, що використовує технологію текстурних атласів та матеріали якої використовують PBR (Physically Based Rendering) модель шейдингу.

У згаданому програмному засобі реалізовано можливість проведення інтерактивної симуляції віртуального полігону із застосуванням засобів технологій віртуальної реальності.

Реалізовано та прорoблено схему керування, системи зброї, порталів, квадрокоптерів, балістики снарядів та інтерактивних об’єктів із використанням відомих законів фізики.

Побудовано покроковий алгоритм для кожної зі стратегій поведінки та візуалізовано чи описано алгоритми за допомогою блок-схем або тексту відповідно.

Створено три унікальні типи місцевості: міська місцевість, приватний сектор та гори – для кожної з яких окремо проводиться автоматичне тестування обраних стратегій поведінки на однакових парах маршрутних точок.

Розроблено додатковий програмний засіб, який допомагає автоматично проводити аналіз зібраних даних телеметрії, візуалізовувати аналізовані дані у вигляді графіків, генерувати звіти з використанням таблиць та у форматі Markdown.

Наукова новизна роботи полягає у порівнянні результатів спрацювання двох стратегій руху об’єктів дослідження, що дозволило зробити висновки стосовно їх ефективності в умовах реальних завдань. Такий підхід сприяє глибшому розумінню можливостей стратегій руху досліджуваних об’єктів та їхньому впровадженню на практиці.

Якщо вважати маршрут та модель руху, утворену Pedestrian стратегією поведінки, подібними до того, що демонструють реальні наземні транспортні засоби, то те спостереження, що в приватному секторі та горах Overfly стратегія поведінки показує себе значно краще у порівнянні з Pedestrian стратегією поведінки в міській місцевості, перекликається із результатами

дослідження в джерелі [2]. Окрім цього, підтверджується те спостереження, що в гірській, та інших місцевостях, зміна висоти є значною компонентою оптимальності маршруту, як це видно з джерела [5].

В майбутніх ітераціях роботи можна додати, покращити чи змінити наступні елементи:

– перенести програму, що відповідає за симуляцію об’єктів дослідження, на більш відкритий ігровий рушій, такий, як рушій Godot;

– зробити симуляцію мультиагентною, що збільшить об’єм та точність зібраних даних;

– застосувати контролери руху та поведінки, засновані на штучних нейронних мережах, навчених за допомогою методів навчання з підкріпленням або еволюційних алгоритмів;

– поглибити рівень симуляції шляхом використання більш повноцінної фізичної моделі, що діє на симульовані об’єкти: симуляція сил, утворених лопатями квадрокоптера чи мультикоптера тощо;

– поглибити рівень симуляції шляхом моделювання вітру, зокрема за допомогою Perlin чи Simplex шумів, що забезпечить баланс необхідного рівня продуктивності програми та точності симуляції, та погодних ефектів, таких, як снігопад, дощ чи злива;

– збирати більше даних для аналізу, таких як рівень шуму, створеного об’єктами дослідження, використання батареї, швидкість руху тощо, у віртуальному середовищі;

– поглибити та розширити рівень аналізу, здійснюваного допоміжною програмою;

– поєднати досліджувану предметну область із предметною областю великих мовних моделей, що в свою чергу дозволить симулювати обробку замовлень, написаних людською мовою; в даному контексті корисними можуть виявитись роботи, наведені в джерелах [27, 28];

– поєднати досліджувану предметну область із предметною областю комп’ютерного зору, обробки та розпізнавання зображень тощо, що в свою

чергу дозволить поглибити рівень симуляції та моделювати більш реальну поведінку; в даному контексті корисними можуть виявитись роботи, наведені в джерелах [32, 41-46].

Результати роботи апробовано у вигляді 2 тез доповідей під час XXIX Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [29], III Міжнародної науково-практичної конференції «Сучасні досягнення та перспективи науки та освіти» [30].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Betti Sorbelli, F. (2024). UAV-based delivery systems: A systematic review, current trends, and research challenges. *Journal on Autonomous Transportation Systems*, 1(3), 1-40.

2. Kirschstein, T. (2020). Comparison of energy demands of drone-based and ground-based parcel delivery services. *Transportation Research Part D: Transport and Environment*, 78, 102209.

3. Baek, D., Chen, Y., Bocca, A., Macii, A., Macii, E., & Poncino, M. (2018, July). Battery-aware energy model of drone delivery tasks. *In Proceedings of the international symposium on low power electronics and design* (pp. 1-6).

4. Torabbeigi, M., Lim, G. J., & Kim, S. J. (2020). Drone delivery scheduling optimization considering payload-induced battery consumption rates. *Journal of Intelligent & Robotic Systems*, 97(3), 471-487.

5. Liu, S., Jin, Z., Lin, H., & Lu, H. (2024). An improve crested porcupine algorithm for UAV delivery path planning in challenging environments. *Scientific Reports*, 14(1), 20445.

6. Adalja, D., Patel, P., Mashru, N., Jangir, P., Arpita, Jangid, R., ... & Khishe, M. (2025). A new multi objective crested porcupines optimization algorithm for solving optimization problems. *Scientific Reports*, 15(1), 14380.

7. Nystrom R. Game Programming Patterns. Genever Benning, 2014. 354 p. URL: https://gameprogrammingpatterns.com/contents.html (дата звернення 07.11.2025).

8. Level up your code with game programming patterns. 2021 LTS Edition. URL: https://unity.com/resources/level-up-your-code-with-game-programming-patterns (дата звернення 07.11.2025).

9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994. 416 p.

10.Brown R. G. Introductory Physics I. Duke University Physics Department, 2013. 661 p. URL: https://webhome.phy.duke.edu/~rgb/Class/intro_physics_1/intro_physics_1.pdf (дата звернення 05.11.2025).

11.Godot Engine Documentation. Complying with licenses. URL: https://docs.godotengine.org/en/stable/about/complying_with_licenses.html (дата звернення 12.11.2025).

12.Godot Engine Documentation. Nodes and Scenes. URL: https://docs.godotengine.org/en/stable/getting_started/step_by_step/nodes_and_scenes.html (дата звернення 12.11.2025).

13.Freya Holmér. Godot Engine Documentation. Introduction to 3D. URL: https://docs.godotengine.org/en/stable/tutorials/3d/introduction_to_3d.html (дата звернення 09.10.2025).

14.Navigation System in Unity. Inner Workings of the Navigation System. URL: https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/NavInnerWorkings.html (дата звернення 12.11.2025).

15.Рівноприскорений прямолінійний рух. Прискорення. https://physics-online.com.ua/10-klas/rivnopryskorenyy-rukh.html (дата звернення 05.11.2025).

16.Davis R. A., Lii K. S., Politis D. N. (2011). Remarks on Some Nonparametric Estimates of a Density Function.

17.Parzen E. (1962). On Estimation of a Probability Density Function and Mode. The Annals of Mathematical Statistics, 33(3), 1065–1076.

18.Pritul M. D., et al. (2021). An amalgamation of YOLOv4 and XGBoost for next-gen smart traffic management system.

19.Start Your Creative Projects and Download the Unity Hub | Unity. URL: https://unity.com/download (дата звернення 22.10.2025).

20.Security Update Advisory (September 2025). URL: https://unity.com/security/sept-2025-01 (дата звернення 22.10.2025).

21.CVE Record: CVE-2025-59489. URL: https://www.cve.org/CVERecord?id=CVE-2025-59489 (дата звернення 22.10.2025).

22.ScottPlot 5 Cookbook. URL: https://scottplot.net/cookbook/5/ (дата звернення 22.10.2025).

23.Flowchart Maker & Online Diagram Software. URL: https://app.diagrams.net/ (дата звернення 22.10.2025).

24.[1DAY_1CAD] DRONE (Tinkercad : Know-how / Style / Education). 1DAY_1CAD : Tinkercad 3D Design. URL: https://www.youtube.com/watch?v=9CXnxZg1J1w (дата звернення 09.10.2025).

25.UV mapping. URL: https://uk.wikipedia.org/wiki/UV_mapping (дата звернення 27.10.2025).

26.Unity Simulation. Clock Management. FixedUpdate (Physics) Scheduling. URL: https://docs.unity3d.com/Simulation/manual/author/clock-management/background.html (дата звернення 27.10.2025).

27.Bohdan N., Tvoroshenko I., Gorokhovatskyi V., and Kobylin O. (2025) Development of a hybrid method to enhance context memory for a chatbot application based on large language models, *International Journal of Academic Information Systems Research*, 9(10), pp. 7-18.

28.Suprun A., Tvoroshenko I., Gorokhovatskyi V., and Yakovleva O. (2025) Development and research of a method for the combined use of large language models for text generation, *International Journal of Academic and Applied Research*, 9(10), pp. 249-263.

29.Кузьменко П. М. Засоби віртуальної реальності у комп’ютерній грі. Радіолектроніка та молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16-19 квітня 2025 р.). Харків: ХНУРЕ, 2025. Т. 7. С. 70-72.

30.Кузьменко П. М. Дослідження стратегій поведінки квадрокоптерів та мультикоптерів у віртуальному середовищі. Сучасні досягнення та перспективи науки та освіти : матеріали III Міжнародної науково-практичної конференції / Міжнародний гуманітарний дослідницький центр (Житомир, 26 жовтня 2025 р.). С. 63-66. URL: https://doi.org/10.64076/ihrc251026 (дата звернення 14.11.2025).

31.Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., and Hudáková M. (2025) Image description compression in classification structural methods, *IEEE Access*, vol. 13, pp. 43631-43641, doi: 10.1109/ACCESS.2025.3548910.

32.Gadetska, S.V., Gorokhovatskyi, V. O., Stiahlyk, N. I., Vlasenko, N.V. Statistical data analysis tools in image classification methods based on the description as a set of binary descriptors of key points. *Radio Electronics, Computer Science, Control*, 2021, №4, 58-68.

33.Gorokhovatskyi, V., Vlasenko, N. (2021). Редукція опису зображення у складі множини дескрипторів на основі метричного критерію інформативності. *Advanced Information Systems*, 5(4), 10-16.

34.Gorokhovatskyi, O., Peredrii, O., Gorokhovatskyi, V., Vlasenko, N. (2023) Explanation of CNN Image Classifiers with Hiding Parts. In: J. Benois-Pineau, R. Bourqui, D. Petkovic, G. Quenot (eds), *Explainable Deep Learning Artificial Intelligence*, pp. 125-146, Academic Press, 346 p.

35.Gorokhovatskyi V., Tvoroshenko I., Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, *Indonesian Journal of Electrical Engineering and Computer Science*, 33 (1), 113-125.

36.Gorokhovatskyi, V., Gadetska, S., & Stiahlyk, N. (2023). Accelerating Image Classification based on a Model for Estimating Descriptor-to-Class Distance. *International Journal of Computing*, 22(4), 485-492.

37.Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, *IEEE Access*, 11, 126938-126949.

38.Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., Hudáková M., and Gorokhovatskyi O. (2024) Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set, *IEEE Access*, vol. 12, 73376-73385.

39.Gorokhovatskyi V., Gadetska S., Stiahlyk N. (2020) Image structural classification technologies based on statistical analysis of descriptions in the form

of bit descriptor set. In CEUR Workshop Proceedings: *Computer Modeling and Intelligent Systems (CMIS-2020)*, 2608, 1027-1039.

40.Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, *Computers, Materials & Continua*, vol. 80, no. 2, 3085-3106.

41.Tvoroshenko I., Pomazan V., Gorokhovatskyi V., and Kobylin O. (2023) Application of video data classification models using convolutional neural networks, *International Journal of Academic and Applied Research*, 7(11), pp. 134-145.

42.Gadetska S., Gorokhovatskyi V., Stiahlyk N., Vlasenko N. (2022) Aggregate Parametric Representation of Image Structural Description in Statistical Classification Methods. In CEUR Workshop Proceedings: *Computer Modeling and Intelligent Systems (CMIS-2022)*, 3137, pp. 68-77.

43.Гороховатський В.О., Гадецька С.В., Стяглик Н.І. (2019) Вивчення статистичних властивостей моделі блочного подання для множини дескрипторів ключових точок зображень. *Радіоелектроніка, інформатика, управління*, №2, 100–107.

44.Gorokhovatskyi, V., Stiahlyk, N., Mazur, Y., Vechirska, A. (2024) Способи метричної грануляції для опису зображень у задачі класифікації. *Системи управління, навігації та зв'язку. Збірник наукових праць*, 3(77), 106-112.

45.Gorokhovatskyi, V. A. (2003). Recognition of images in conditions of incomplete information. *KNURE, Kharkov.*

46.Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image classification methods. *Advanced Information Systems*, 9(1), 5–12. URL: https://doi.org/10.20998/2522-9052.2025.1.01 (дата звернення 17.11.2025).