

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ РОЗРАХУНКУ ТА
ПРЕДСТАВЛЕННЯ РУХУ ТРАНСПОРТНИХ ЗАСОБІВ У ДОПОВНЕНІЙ
РЕАЛЬНОСТІ ЗА ДОПОМОГОЮ КРИВИХ БЕЗЬЄ. APPLICATION
DEVELOPMENT FOR CALCULATING AND REPRESENTING THE
MOVEMENT OF VEHICLES IN AR USING BEZIER CURVES

Виконав: студент(ка) 2 курсу, групи КНТ-110м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Свістельнік В.Ю.

(прізвище та ініціали)

Керівник Гладкова О.М.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

Свістельніка Владислава Юрійовича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного застосунку розрахунку та представлення руху транспортних засобів у доповненій реальності за допомогою кривих Безьє. Application Development for Calculating and Representing the Movement of Vehicles in AR Using Bézier Curves
 керівник проєкту (роботи) Гладкова Ольга Миколаївна, к.т.н.,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “5” листопада 2021 року № 435
2. Строк подання студентом проєкту (роботи) 01 грудня 2021 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали та методи. 3. Розробка програмного рішення. 4. Експлуатація та експериментальне дослідження програми.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Гладкова О.М., доцент		
Нормоконтроль	Бєлова А.В., асистент		

7. Дата видачі завдання 06 вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1-2 тижні	Завдання, ТЗ
2	Аналіз предметної області.	3-5 тижні	Розділ 1
3	Матеріали та методи.	6-7 тижні	Розділ 2
4	Розробка програмного рішення.	8–11 тижні	Розділ 3
5	Експлуатація та експериментальне дослідження програми.	12-13 тижні	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	14-15 тижні	Додатки
7	Захист роботи.	16 тиждень	

Студент (ка) _____ Свістельнік В.Ю. _____
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Гладкова О.М. _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 91 с., 3 табл., 52 рис., 2 дод., 17 джерел.

ДОПОВНЕНА РЕАЛЬНІСТЬ, РУХ ТРАНСПОРТНИХ ЗАСОБІВ, МОБІЛЬНИЙ ПРИСТРІЙ, UNITY.

Об'єкт дослідження – візуалізація руху транспортних засобів.

Предмет дослідження – програмне забезпечення для візуалізації руху транспортних засобів на мобільному пристрої.

Мета роботи – створення програмного застосунку на мобільний пристрій, що візуалізує рух транспортних засобів за допомогою кривих Безьє у доповненій реальності.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мова програмування C#, кросплатформний ігровий рушій Unity.

Результати. Створено програмний застосунок, керований мобільним пристроєм, який у доповненій реальності візуалізує переміщення транспортних засобів по спеціальним маркерам, які розпізнаються камерою того самого мобільного пристрою.

Наукова новизна отриманих результатів. Отримано подальший розвиток алгоритм побудови кривих Безьє для прокладання оптимальної траєкторії руху транспортного засобу у доповненій реальності.

Практична значимість результатів. Розроблено програмний продукт для візуалізації переміщення об'єктів з одного місця в інше за траєкторією кривої Безьє в обхід встановлених позицій.

Галузь використання – інтерактивне представлення переміщення військової техніки або інших транспортних засобів у докладах щодо деяких дій, на уроках у школі та парах в університетах.

ABSTRACT

Explanatory note to the master's qualification work: 91 pages, 52 figures, 3 tables, 2 appendixes, 17 sources.

AUGMENTED REALITY, THE MOVEMENT OF VEHICLES, MOBILE PHONE, UNITY.

Object of study is a representing the movement of vehicles.

Subject of study is application for calculating and representing the movement of vehicles on the mobile device.

The purpose of the work is to create a software application for a mobile device that visualizes the movement of vehicles in augmented reality.

Materials, methods, and tools: structured and object-oriented programming, , mobile device running any operating system, programming languages C#, cross-platform game engine Unity.

Results. Created a software application controlled by a mobile device, which in augmented reality visualizes the movement of vehicles on special markers that are recognized by the camera in this mobile device.

Scientific novelty of the obtained results. The algorithm of construction of Bezier curves for paving of an optimum trajectory of movement of the vehicle in addition to a reality is further developed.

Practical significance of the results. Developed a software product to automate the visualization of the movement of objects from one place to another along the trajectory by the Bézier curve bypassing the established positions.

The field of use is a interactive presentation of the movement of military equipment or other vehicles in reports on certain actions, in school lessons and at universities.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Технології віртуальної реальності	10
1.2 Технології доповненої реальності	13
1.2.1 Огляд пристроїв доповненої реальності.....	15
1.2.2 Види доповненої реальності.....	17
1.3 Методи логістики транспортних засобів	20
1.4 Висновок до розділу	22
2 МАТЕРІАЛИ ТА МЕТОДИ.....	23
2.1 Криві Безьє	23
2.2 Вибір ігрового рушія	24
2.3 Огляд мови програмування	26
2.4 Огляд середовищ розробки.....	27
2.5 Висновок до розділу	31
3 РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ.....	33
3.1 Структура програмного продукту	33
3.2 Проєктування інтерфейсу	35
3.3 Розробка логіки.....	38
3.4 Висновок до розділу	42
4 ЕКСПЛУАТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ.....	43
4.1 Призначення й умови застосування програми	43
4.2 Початкові та вихідні дані.....	43
4.3 Звертання до програми.....	43
4.4 Робота з застосунком.....	44
4.5 Опис методики проведення тестування	47
4.6 Висновок до розділу	50

ВИСНОВКИ	51
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	52
ДОДАТОК А. Текст програми	54
ДОДАТОК Б. Слайди презентації.....	84

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AR	– augmented reality;
IDE	– integrated development environment;
SDK	– software development kit;
VCS	– version control system;
VR	– virtual reality;
ПЗ	– програмне забезпечення;
ТЗ	– транспортні засоби.

ВСТУП

На сьогоднішній день будь-який життєвий процес відбувається через комунікацію людей. Ми спілкуємось, жестикулюємо, якщо потрібно щось представити або пояснити, ми можемо виконати схематичні рисунки, надати графічне зображення. У новинах, на уроках, частіше історії, також використовують відео: доповідь іншої людини, мальовані або, знову ж таки, схематичні анімації, архівні зйомки і інше. Інформація в такому вигляді краще засвоюється та запам'ятовується, аніж просто текстова чи голосова. Тож, цей дипломний проєкт може використовуватись у будь-якій сфері, де потрібно показати переміщення якогось об'єкту з однієї точки в іншу в обхід третьої, а щоб це було ще цікавіше, - об'єкт, що переміщується, відображується у доповненій реальності. Наприклад, школярам, це б було точно цікаво. І при цьому не потрібно щось малювати чи монтувати. Для взаємодії з програмою потрібен лише пристрій з камерою та мітки, які відповідають тим, що запрограмовані у системі.

Мета дипломної кваліфікаційної роботи магістра є розробка програмного застосунку на мобільний пристрій, що візуалізує рух транспортних засобів у доповненій реальності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Технології віртуальної реальності

Віртуальна реальність — це чудова можливість миттєво перенестися в інше середовище, отримати яскраві враження та набути незвичайного досвіду. І хоча технологія віртуальної реальності лише нещодавно увійшла в наше життя, сама вона далеко не така нова, як можна було б очікувати [1]. Ще у 1838 році, до появи перших комп'ютерів, Чарльз Уїтстоун винайшов перші форми стереоскопу (рис. 1.1), який використовував можливості здібностей мозку поєднувати два слабких та різних зображення в одне, ізолюючи змішану картинку над кожним оком, створюючи трьохвимірний ефект. Цікаво, що такі продукти як Google Cardboard, VR мобільних телефонів використовують приблизно ідентичні з таким стереоскопом методи для створення глибинного сенсу зображень [2].



Рисунок 1.1 – Стереоскоп [2]

З розвитком технологій почали з'являтися речі, які саме передували сьогоднішнім, зрозумілим для нас, пристроям віртуальної реальності. Так, наприклад, у 1929 році Едвін Лінк винайшов перший симулятор польотів. Він був призначений для безпечного та ефективного навчання пілотів, завдяки чому ним скористалися понад 500 тисяч американських льотчиків під час Другої світової війни, а також пілотів практично усіх інших націй, що були залучені у війні. Тренажер Лінка змінював тангаж та крен у відповідь на зміну пілотом позицій елементів керування. Попри відсутність повноцінної візуальної перспективи, Link Trainer все ж таки демонстрував певне занурення у віртуальне середовище та дозволяв досягнути певних результатів швидким та економічним способом [2].

На рисунку 1.2 зображено перший симулятор польотів.



Рисунок 1.2 – Тренажер Лінка [2]

Після того було ще багато різних варіантів, прикладів, розробок, але перші споживчі пристрої, які стали схожі і виглядом і технологією на те, що ми маємо сьогодні, почали з'являтися в 1990-х роках. У 1994 році, гігант відеоігор Sega представила гарнітуру Sega VR для аркадних ігор, разом з шоломом віртуальної реальності, стерео навушниками, і простим інерційним датчиком

для примітивного відстеження рухів головою. Але цей аксесуар так і залишився прототипом (рис. 1.3) [2].



Рисунок 1.3 – Sega VR[2]

Це був перший великий приклад бажання великих ігрових компаній працювати з віртуальною реальністю, але не останній. Наприклад, після того Nintendo у 1995 році анонсувала такий же пристрій Virtual Boy, який також провалився як проєкт.

На сьогодні найвідомішими такими продуктами є, наприклад, Oculus, Valve, HTC.

На рисунку 1.4 зображено шолом з контролерами Oculus Quest.



Рисунок 1.4 – Сучасний шолом з контролерами Oculus Quest [2]

Сфери використання таких продуктів дуже різноманітні. Маємо безліч ігор різних жанрів, віртуальні екскурсії: приміщення установ, замки, вулиці, відтворенні історичні локації. Також таким чином будують тренажери: знову ж таки, по управлінню літаком, проведення медичних операцій, або тренування технологічним процесам на підприємствах. Такий підхід зменшує витрати компаній, знижує потенційні ризики від невдач, полегшує навчання людей у відповідній галузі.

Вже зараз відточують технології, за допомогою яких можна віддалено керувати спеціальними роботами, наприклад знешкодження мін, дещо. Тому перспективи та користь значна.

1.2 Технології доповненої реальності

Як і у випадку з віртуальною реальністю, доповнена починає розвиватись ще задовго до сьогодення. Перша згадка про доповнену реальність сходить до 1950-х років, коли кінематографіст Мортон Хейліг думав про розвиток кіно і залученості глядача за рахунок використання всіх почуттів людини. І у 1968 році Сазерленд створив першу таку систему з використанням оптичного прозорого дисплея з головним кріпленням [3].

Перша система доповненої реальності зображена на рисунку 1.5.



Рисунок 1.5 – Перша система доповненої реальності

Наступним експериментатором у сфері доповненої реальності став Стів Манн, який винайшов в 1980 році перший AR-пристрій, який можна було носити на голові. Він не був зручним, але дозволяв накладати на реальну картинку текст із зображення, що зберігалося в портативному комп'ютері захованому в портфелі випробувача [3].

Перші запущені у виробництво пристрої AR призначалися зовсім не для розваги. Так, з моменту винаходу її використовували у військовому секторі, де вона отримала найбільш стрімкий розвиток. Super Cockpit представляв собою продуману систему доповненої реальності для пілотів. Спеціальний шолом виводив в поле зору льотчика текст, що містив всю необхідну для швидкого прийняття рішень інформацію. Надалі були розроблені також AR-інтерфейси для управління польотами [3].

Напівпрозорий дисплей з виведеною інформацією зображено на рисунку 1.6.



Рисунок 1.6 – Напівпрозорий дисплей з виведеною інформацією

AR стала поступово набувати популярності, проникаючи в нові сфери життя. Наприклад, з'явилися перші ігри з застосуванням доповненої реальності

і суміжних технологій. Під час запису спортивних матчів стали використовувати обладнання, яке демонструє траєкторію руху м'яча [3].

Термін «доповнена реальність», ймовірно, був запропонований дослідниками корпорації Boeing Томом Коделом (Tom Caudell) у 1990 році. І трактували її як результат введення у поле сприйняття будь-який сенсорних даних з метою доповнення даних про оточення і поліпшення сприйняття інформації [4].

Існує кілька інших визначень доповненої реальності. Зокрема, дослідник Рональд Азума (Ronald Azuma) у 1997 році визначив її як систему, яка:

- зміщує віртуальне і реальне;
- взаємодіє у реальному часі;
- працює з 3D.

Технологія AR зручна і досить багатофункціональна. З її допомогою можна швидко передавати інформацію, демонструвати роботу механізмів, влаштовувати креативні перформанси і створювати оригінальні онлайн-ігри. Ще одна важлива функція доповненої реальності – збереження бюджету. AR дозволяє економити на різних ресурсах, починаючи з паперу і закінчуючи кадрами. Медицина, військова промисловість, ритейл і ігрова індустрія: у всіх цих сферах в доповненої реальності маса переваг [3]. Наприклад, деякі компанії виготовлення одягу, носимих аксесуарів, меблів використовували цю технологію надаючи своїм клієнтам застосунок на мобільний телефон, за допомогою якого вони могли приміряти на собі речі або розташовувати шафи, крісла тощо у своїх приміщеннях за допомогою свого мобільного телефону та наявної камери в ньому, відповідно.

1.2.1 Огляд пристроїв доповненої реальності

Як згадувалося раніше, дана технологія використовується на різних системах, пристроях, для різних цілей. По пристроям можна виділити три загальні групи: мобільні пристрої, стаціонарні пристрої та спеціальні засоби.

До мобільних пристроїв відносять планшети, смартфони, окуляри доповненої реальності, лінзи доповненої реальності.

На планшети і смартфони має бути встановлено спеціалізоване програмне забезпечення.

Окуляри доповненої реальності — це окремий повноцінний пристрій, розроблений безпосередньо для роботи з AR. Вони вміють проєктувати голограми та інформацію у реальний простір, але не прив'язуються до їх фізичних об'єктів. Фактично, це просто екран перед очима. Найбільш відомі окуляри Google Glass, Vuzix Blade, Epson Moverio, Sony SmartEyeglass. Ще є такі окуляри, як Microsoft HoloLens, Magic Leap One и Meta 2, але це вже окуляри змішаної реальності, тобто вони дозволяють працювати з віртуальними об'єктами, прив'язаними до реального світу [4].

Лінзи для доповненої реальності поки ще лишаються технологією майбутнього. Розробники прагнуть перетворити лінзи у прозорий екран, що містить систему управління, мініатюрну камеру, антену, світлодіоди та інші оптоелектронні компоненти. Зокрема, компанія Samsung вже подала патент на «розумні» контактні лінзи, роботи у цьому напрямку веде і компанія Google. Але на ринок подібні пристрої вийдуть не раніше, ніж 5-10 років [4].

Стаціонарними пристроями можуть бути телевізор, екран комп'ютера, ігровий комп'ютер типу Kinect. На екран телевізора виводиться вже доповнене зображення (особливо часто це буває під час трансляції футбольних і хокейних матчів), приклад для комп'ютера — карти Google в режимі «Satellite», коли на супутниковий знімок накладаються назви вулиць і визначні місця. Іноді використовуються широкоформатні екрани, а також проєкційні системи, здатні накладати зображення не лише на екрани, але і на будь-які поверхні [4].

До спеціальних засобів відносять, наприклад, спеціалізовані шоломи військових пілотів. На скло шолома виводиться необхідна пілоту важлива інформація і він може сприймати її, не переводячи погляд на панель приладів, тим самим економлячи дорогоцінні секунди. Багато з подібних систем дозволяє

здійснювати вказання цілі шляхом повороту голови чи рухом очних яблук пілота.

На захисному склі «розумного шолома» може бути відображена швидкість мотоциклу, маршрут, текстові повідомлення і багато іншого. Схожу технологію використовують і для відображення інформації на лобовому склі автомобіля [4].

1.2.2 Види доповненої реальності

Доповнена реальність буває чотирьох типів: на основі маркерів, без маркерів, на основі проекції та AR на основі накладання. Давайте розглянемо їх по одному в деталях.

У випадку маркерного типу, маркер є особливим візуальним об'єктом, як спеціальний знак або щось інше. Камера використовуються для ініціювання 3D-цифрових анімацій. Система буде обчислювати орієнтацію та позицію пристрою, щоб ефективно позиціонувати вміст [5]. Прикладом може бути, згаданий вище, застосунок для розміщення меблів, який зображено на рисунку 1.7.

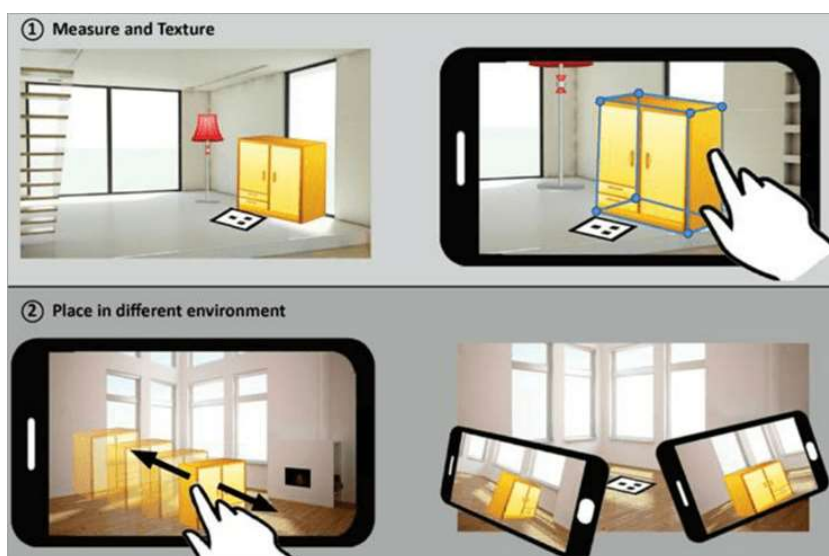


Рисунок 1.7 – Застосунок на мобільний пристрій для розташування меблів [5]

Доповнена реальність без маркерів використовується в програмах для подій, бізнесу та навігації, наприклад, технологія використовує інформацію на основі місцезнаходження для визначення того, який вміст користувач отримує або знаходить у певній області. Він може використовувати GPS, компаси, гіроскопи та акселерометри, як це можна використовувати на мобільних телефонах [5]. Прикладом є популярна, не так давно, мобільна гра – Pokemon Go (рис. 1.8).



Рисунок 1.8 – Pokemon Go

ДР на основі проєкту використовує синтетичне світло, що проєктується на фізичні поверхні, щоб виявити взаємодію користувача з поверхнями. Він використовується на голограмах, таких як у 'Зоряних війнах' та інших науково-фантастичних фільмах (рис. 1.9).



Рисунок 1.9 – Проекція меча в AR-гарнітурі на основі проєкту

У випадку накладання оригінальний предмет замінюється збільшенням, повністю або частково. Наведений нижче приклад дозволяє користувачам розміщувати віртуальні предмети меблів над зображенням кімнати зі шкалою в застосунку IKEA Catalog (рис. 1.10).



Рисунок 1.10 – Приклад доповненої реальності на основі накладання

1.3 Методи логістики транспортних засобів

Транспортна логістика – це система по організації доставки, а саме переміщення будь-яких матеріальних предметів або речовин з однієї точки в іншу за оптимальним маршрутом. Один із основоположних напрямків науки про управління інформаційними і матеріальними потоками у процесі руху товарів. Оптимальним вважається той маршрут, по якому можливо доставити логістичний об'єкт в найкоротші терміни (або передбачені терміни) з мінімальними витратами, а також з мінімальною шкодою для об'єкта доставки.

Предметом вивчення транспортної логістики є: питання, що стосуються транспорту як ланки логістичної системи, інформаційного та транспортно-експедиційного забезпечення логістики, методів розв'язання транспортних та розподільних задач, вибору перевізника вантажа і постачальника, а також міжнародні логістичні інтеграційні процеси [6].

Будь-який вид логістики повинен являти собою сукупність алгоритмів і технологій, що дозволяють реалізувати в економіці логістичний підхід. Суть реалізації логістичної концепції полягає в розробленні та впровадженні логістичних систем управління матеріальними і відповідними інформаційними потоками, котрі ґрунтуються на логістичних принципах і методах.

Висока ефективність використання методів і моделей в логістиці досяжна, якщо буде забезпечено виконання низки умов серед яких:

- системний підхід до вирішення даної проблеми;
- наукова обґрунтованість самих методів і моделей;
- адекватність моделі реальній системі;
- об'єктивний облік взаємозв'язку підсистем;
- узгодження матеріальних, транспортних, інформаційних та інших потоків [7].

Оптимізація руху транспорту у будь-якому місті із розвинутою інфраструктурою залишається однією із головних урбаністичних проблем, а для її вирішення все більше значення відіграють методи математичного

моделювання. Математичні моделі, що застосовуються сьогодні для опису та аналізу руху транспортних потоків по транспортній мережі міста доволі різноманітні, однак вони нерідко вносять у процес моделювання ряд неточностей принципового характеру [8], [9].

Транспортну мережу міста можна інтерпретувати як сукупність транспортних зв'язків, по яких здійснюються пасажирські або вантажні перевезення. Слід зауважити, що транспортна мережа, будь-якого міста ніколи не відповідає дорожній мережі. Залежно від габаритів і маси вантажу, параметрів використовуваних транспортних засобів, транспортна мережа буде тільки тим чи іншим фрагментом дорожньої мережі [10], [11].

Математичний опис транспортних потоків і пов'язаних із ними явищ, давно привертає увагу науковців, проте і на сьогодні не існує універсального математичного підходу до адекватного моделювання складних процесів.

Більшість відомих моделей динаміки транспортних потоків можуть бути умовно розділені на три групи:

- макроскопічні;
- мікроскопічні;
- мезоскопічні моделі.

Макроскопічні моделі описують рух транспортних засобів в усереднених показниках і термінах, таких як щільність, середня швидкість, інтенсивність транспортного потоку та ін. [12].

Мікроскопічні моделі - у яких детально моделюється рух кожного транспортного засобу за допомогою звичайних диференціальних рівнянь. Такий підхід дозволяє теоретично досягти точнішого опису руху транспорту у порівнянні із макроскопічним моделюванням, проте у разі практичного застосування цей підхід вимагає від науковців величезних затрат часу та обчислювальних ресурсів. Особливе місце у групі мікро- моделей займають моделі типу «кліткових автоматів», що отримали широкий розвиток останніми роками. У них прийнятий надзвичайно спрощений дискретний у часі та

просторі опис руху транспортних засобів, за рахунок чого досягається висока обчислювальна ефективність моделей даного виду [13].

Мезоскопічні моделі є проміжною ланкою між двома попередніми моделями. У них потік описується щільністю розподілу автомобілів у просторі координат і швидкостей ТЗ. Динаміка змін щільності описується кінетичними рівняннями, які ґрунтуються на усередненому описі взаємодії окремих автомобілів. Важливе теоретичне значення мезоскопічних моделей полягає в тому, що на їх основі можна, у заданому часовому інтервалі, застосовувати макроскопічні моделі [12].

1.4 Висновок до розділу

У цьому розділі було розглянуто історію розвитку доповненої та віртуальної реальності, галузі застосування та стан розвитку на сьогодні. Були проаналізовані більш детально технології пристрої та види доповненої реальності. Крім того, було виконано аналіз транспортної логістики, а саме моделі динаміки транспортних засобів.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Криві Безьє

Вперше криві були представлені широкій публіці в 1962 році французьким інженером П'єром Безьє, який, розробивши незалежно від де Кастельє, використовував їх для комп'ютерного проєктування автомобільних кузовів. Криві були названі ім'ям Безьє, а ім'ям де Кастельє названо розроблений ним рекурсивний спосіб визначення кривих – алгоритм де Кастельє [14].

Крива Безьє – основа векторної графіки. Застосування кривої Безьє є досить широким: спосіб управління рухом, засіб створення анімаційних рисунків, метод опису художніх шрифтів, спосіб передачі виділених об'єктів, інтерактивний інструмент дизайну форми різноманітних об'єктів [14].

На початку крива проходила через всі задані точки, тобто ті методи були методами інтерполяції. Але вони часто мають ряд недоліків, тому Безьє розробив інший метод зображення кривої. В новому методі крива визначається вершинами багатокутника, який задає форму кривої. Кривій належить перша та остання вершина, в той час як інші вершини характеризують похідні, порядок та вид кривої (рис. 2.1) [15]. Порядок точок масиву має важливе значення, так як його зміна повністю змінить вигляд кривої.

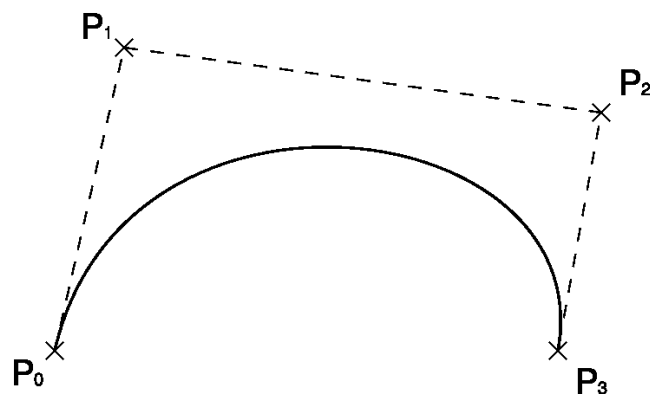


Рисунок 2.1 – Крива Безьє по точкам ламаної лінії

Представимо записану формулу кривої Безьє (2.1) – (2.3).

$$B(t) = \sum_{i=0}^n b_{i,n}(t) P_i, t \in [0,1], \quad (2.1)$$

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad (2.2)$$

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \quad (2.3)$$

де n – степінь полінома, який характеризується $n+1$ вершинами;

i – порядковий номер вершини;

t – час;

P_i – вершини.

Формула (2.2) – це базисні функції кривої Безьє, названі також поліномами Бернштейна.

2.2 Вибір ігрового рушія

Для доповненої або віртуальної реальності можна використовувати як просто можливості мов програмування скомпільовані в застосунок, так і ігрові рушії. Було обрано ігровий рушій, так як для візуалізації руху транспортних засобів ігрові рушії мають механізми, з якими досить просто це все зробити.

Ігровий рушій – це ядро усіх сучасних ігор, в якому є багато методів для взаємодії з об'єктами гри, запрограмована фізика, наприклад, гравітація, яку можна змінювати під свої цілі, рендерінг кадрів, готові вбудовані скрипти та інше. Також вони дозволяють кросплатформенність, тобто не потрібно під кожен новий пристрій, або операційну систему окремо створювати свій застосунок, достатньо один раз розробити і побудувати готове рішення під цільові платформи. Найвідоміші сьогодні ігрові рушії: Unity, Unreal Engine 4, GameMaker, Godot, AppGameKit, CryEngine та інші. Всі вони різні і кожен

зможе знайти свої переваги в будь-якому рушії для себе. Наприклад GameMaker, не вимагає від користувача навиків програмування, але якщо потрібно, то можна і програмувати скрипти під себе.

Unreal Engine від Epic Games. Оригінальна версія була випущена в 1998 році, а через 21 рік вона все ще продовжує використовуватися для великої кількості ігор щороку. Найвідомішими іграми, створеним разом з Unreal Engine, є серія Gears of War, серія Mass Effect, серія Bioshock, серія Batman: Arkham та інші [16]. Як бачимо, часто використовується у складних, великих комп'ютерних іграх. В Unreal вбудовано практично все, що може знадобитися, включаючи 3D-моделювання та роботу з ландшафтом. Використовується мова програмування C++. Цей ігровий рушій досить складний, тому для новачків він не дуже підходить, але є велика кількість інформації, форумів, відео-курсів, які допоможуть освоїти його.

Unity – найкраще з існуючих середовище розробки простих комп'ютерних ігор, яке дозволяє вам робити тривимірні та двовимірні ігри для будь-якої платформи [17]. Часто обирають цей рушій для створення мобільних ігор. Мова програмування, що використовується – C#.

GameMaker. На відміну від більшості інших ігрових рушіїв, GameMaker: Studio набув більш широкого поширення, оскільки для його використання не потрібні знання з програмування. Натомість користувачі можуть користуватися цим рушієм, як конструктором, створюючи ігри набагато простіше та швидше, ніж кодувати рідними мовами. Зрозуміло, що таким чином можна створювати лише прості і невеличкі ігри. Доступна можливість писати і свої скрипти на власній мові програмування Game Maker Language, який синтаксисом дещо схожий на відомі Pascal або C++, але є спрощеним.

Інші рушії розглядати не будемо, так як їх, дійсно, велика кількість і для вирішення поставленої задачі було обрано Unity.

Приведемо коротке порівняння рушіїв у таблиці 2.1.

Таблиця 2.1 – Огляд ігрових рушіїв

Ігровий рушій	Unreal Engine	Unity	GameMaker
Рівень складності	5/5	3/5	1/5
Доступність	Вимагає підписки	Є безкоштовна версія	Якщо не експортувати на платформи, безкоштовна
Кросплатформність	Так	Так	Так
Легка відладка застосунку	Так	Так	Ні
Інструментарій	5/5	4/5	2/5

Ще раз зазначимо, що з описаного, GameMaker орієнтований більше на новачків. З цим рушієм можна спочатку навчитися створювати ігри, і вже потім вчитися програмувати.

В результаті аналізу було обрано Unity, так як він надає нам усі необхідні інструменти для розв'язання поставленого завдання і має велику кількість інформації по роботі з ним, до того ж є легшим для розуміння, якщо ніколи не працював з подібними технологіями.

2.3 Огляд мови програмування

Так як рушієм обрали Unity, мова, з якою будемо працювати далі, – C#. Це об'єктно-орієнтована, C-подібна мова програмування з безпечною системою типізації для платформи .NET, тобто має строгу статичну типізацію. Належить корпорації Microsoft і є наслідником мови C++. Синтаксис C# близький до C++ і Java. Переїнявши багато від своїх попередників, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів, проте

передбачає множинне успадкування інтерфейсів. Також, вважається конкурентом мові Java. Ці дві мови дуже схожі, також через те, що C# і створювався компанією для того, щоб мати власний інструмент такий, як Java. Приведемо порівняльну таблицю 2.2 деяких характеристик цих мов.

Таблиця 2.2 – Порівняння C# та Java

Мова	C#	Java
Строга типізація	+	+
Збірник мусора	+	+
Одиночне наслідування	+	+
Делегати	+	-
Поліморфізм за замовчуванням	-	+

Ці дві мови дуже схожі, але Java більш орієнтована на серверну роботу, і, до того ж, ми працюємо з рушієм Unity, тому зупиняємось на мові програмування C#.

2.4 Огляд середовищ розробки

Для написання коду існує багато редакторів та середовищ розробки. Деякі з них безкоштовні, деякі на основі підписки або разової купівлі. На сьогоднішній редактори від середовищ відрізняються наявністю в IDE компілятора або інтерпретатора. Інші функції, такі як система керування версіями, схема ієрархії класів, графічний інтерфейс користувача, вбудовані як в редактори, так і в середовища розробки. І все ж таки, кожний з них має свої переваги або індивідуальну зручність для користувача.

Розглянемо два середовища розробки та текстовий редактор для роботи з C#, а також, з якими зручно працювати і в контексті Unity.

Visual Studio Code — засіб для створення, редагування та зневадження сучасних вебзастосунків і програм для хмарних систем. Розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X (рис. 2.2). Підтримує велику кількість мов програмування, що, згідно з відповідним синтаксисом, підсвічує команди, пропонує варіанти і так далі. Включає в себе також зручний інтерфейс по роботі з контролем версій (рис. 2.3), в якому вікно ділиться на дві частини: в лівій відображена стара версія, в правій — нова, і, при чому, ті місця, що відрізняються виділені червоним та зеленим кольором, відповідно. Крім того, дозволяє зручно вирішувати конфлікти версій, які виникають тоді, коли дві гілки системи контролю версій з різними змінами в одному і тому ж файлі або об'єкті, намагаються об'єднати. Саме конфлікт виникає через те, що автоматично система не знає, яку саме зміну прийняти у результат.

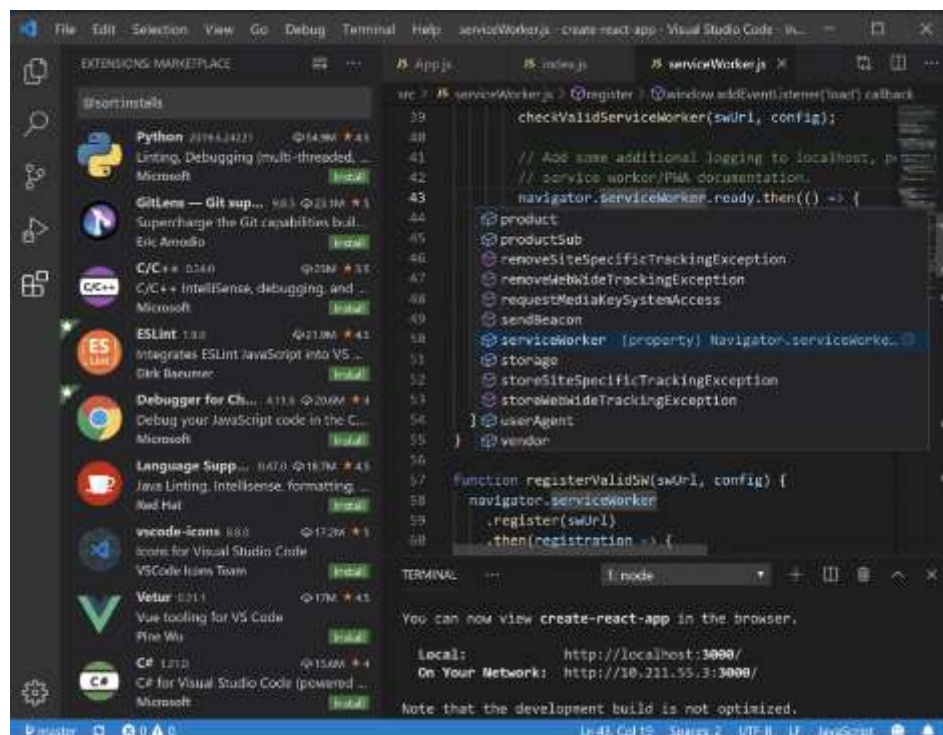


Рисунок 2.2 – Вікно редактору Visual Studio Code

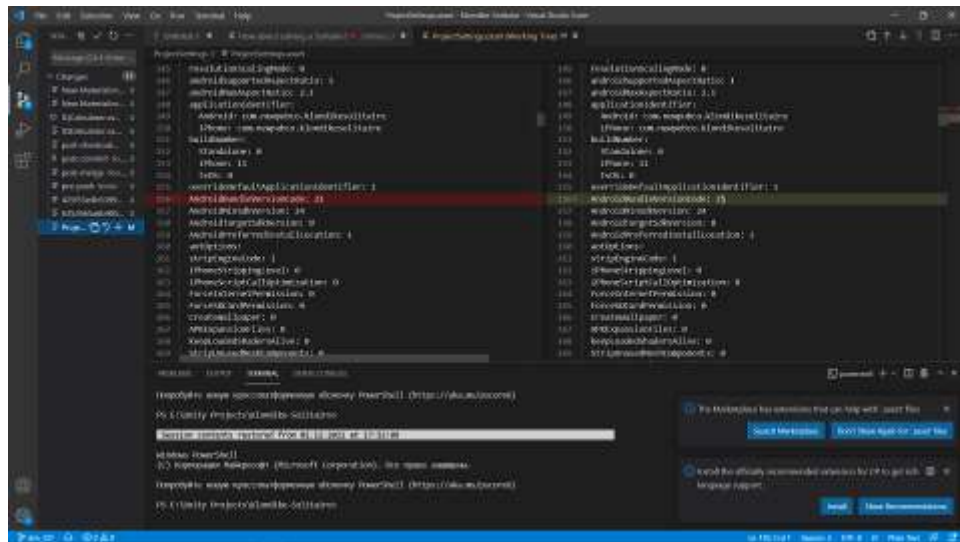


Рисунок 2.3 – Приклад підсвічування змін у файлі

Microsoft Visual Studio — продукт фірми Майкрософт, середовище розробки програмного забезпечення. Надає змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також вебсайти, вебзастосунки, вебслужби як у рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight (рис. 2.4).

Visual Studio є досить функціональним середовищем, має безліч горячих клавіш, та ж підсвітка коду, інструменти і таке інше. Серед недоліків виокремлю ту ж роботу із системами контролю версій. Не усім вона зручна, хоча і є графічний інтерфейс для цієї роботи, але часто програмісти звертаються до інших застосунків для такого питання. Наприклад, згаданий вище Visual Studio Code.

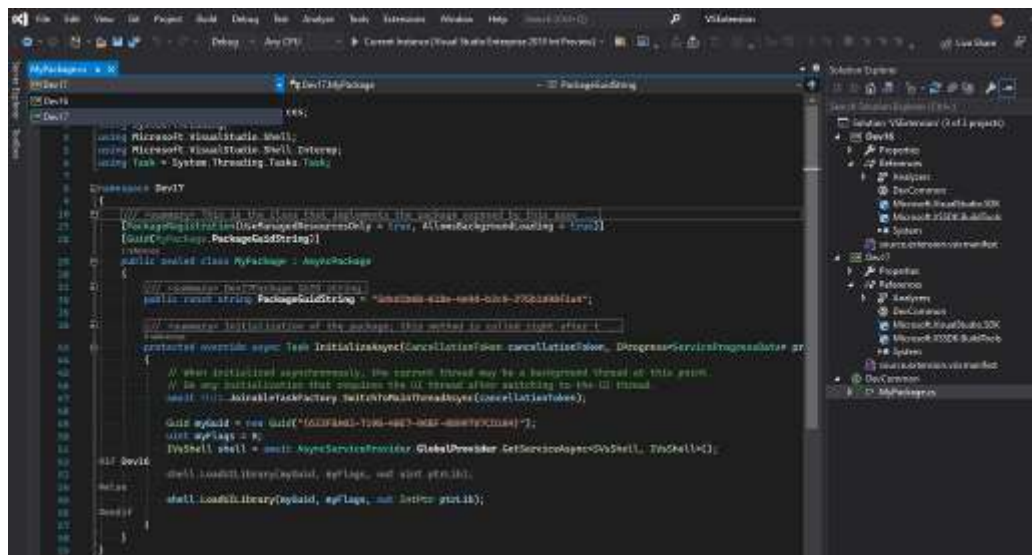


Рисунок 2.4 – Вікно середовища розробки Visual Studio

Rider від JetBrains – середовище розробки для Unity. Надає потужну підтримку, зокрема: швидкі виправлення, дії й перевірки контексту, завершення коду та підсвічування синтаксису у файлах шейдерів, налагодження скриптів, запуск та налагодження тестів Unity, профілювання ігор Unity, оновлення ресурсів та багато іншого (рис. 2.5). Має багато зручних функцій, які відрізняють його від конкурентів, наприклад, та ж робота з VCS.

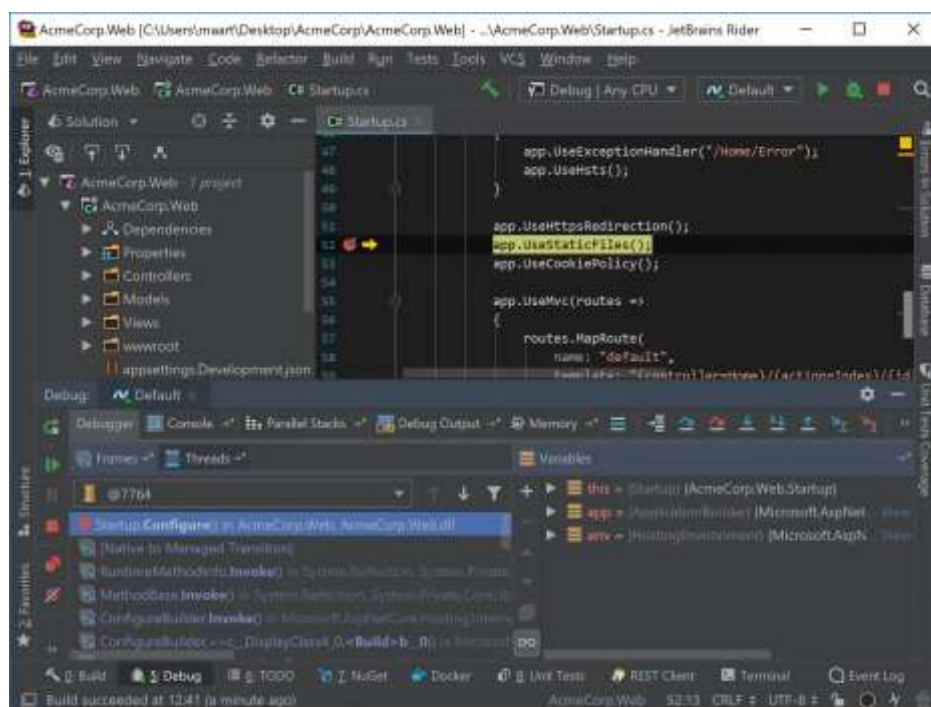


Рисунок 2.5 – Вікно середовища розробки Rider

Приведемо коротку таблицю 2.3 по цим трьом продуктам.

Таблиця 2.3 – Порівняння інструментарію розробки

Інструментарій	Microsoft Visual Studio	Visual Studio Code	Rider
Підсвічування коду	+	+	+
Автодоповнення	+	+	+
Горячі клавіші	+	+	+
Компілятор	+	-	+
Робота з VCS	-	+	+
Unit-тести	+	-	+
Робота з базами даних	+	-	+
Плагіни	+	+	+

Звичайно, представлено не всі можливості цих інструментарієв, але багато в чому вони схожі, особливо Microsoft Visual Studio з Rider. Проблема останнього лише в тому, що він потребує підписки у розмірі 15 доларів на місяць.

Було обрано середовище розробки Microsoft Visual Studio, так як він надає нам усі необхідні функції і, до того ж, ми розроблюємо не комерційний продукт, тому немає сенсу працювати з Rider, хоча він і більш зручний. Visual Code більше б підійшов до розробки сайтів або роботи з Arduino.

2.5 Висновок до розділу

У цьому розділі проаналізовано алгоритм Безьє. Було розглянуто ігрові рушії на прикладі Unreal Engine, Unity, GameMaker, виконано порівняння і обгрунтовано вибір ігрового рушія Unity. Виконано порівняння та вибір мови

програмування. Також, розглянули такі середовища розробки як Microsoft Visual Studio та Rider і текстовий редактор Visual Code, з якими можна працювати в Unity, і, відповідно, обґрунтовано вибір Microfostr Visual Studio.

3 РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ

Необхідно розробити застосунок доповненої реальності за допомогою ігрового рушія Unity. З цього випливає, що інтерфейсом виступає камера телефону та об'єкти, що візуалізуються. Для полегшення реалізації в проєкті використовуються бібліотека Vuforia.

Vuforia – це платформа доповненої реальності та інструментарій розробника програмного забезпечення для мобільних пристроїв, розроблена компанією Qualcomm. Vuforia використовує технології комп'ютерного зору, а також відстеження плоских зображень та простих об'ємних реальних об'єктів у реальному часі.

3.1 Структура програмного продукту

Спочатку приведемо структурну схему розроблюваного продукту (рис. 3.1).



Рисунок 3.1 – Структурна схема розробленого застосунку доповненої реальності

Проект містить різні файли, необхідні для роботи. Велику кількість генерує Unity, деякі необхідні для збірки проекту на платформи. Так, наприклад, для того, щоб побудувати проект для Android, потрібно налаштувати такі файли, як MainManifest та GradleTemplate.

Для того, щоб мати у проекті об'ємні об'єкти, їх потрібно також додати. Unity підтримує такі формати:

- fbx;
- dae;
- dxf;
- obj.

Ми використовуємо fbx. Тому наш об'єкт, що візуалізується має назву T90LP.fbx (рис. 3.2).

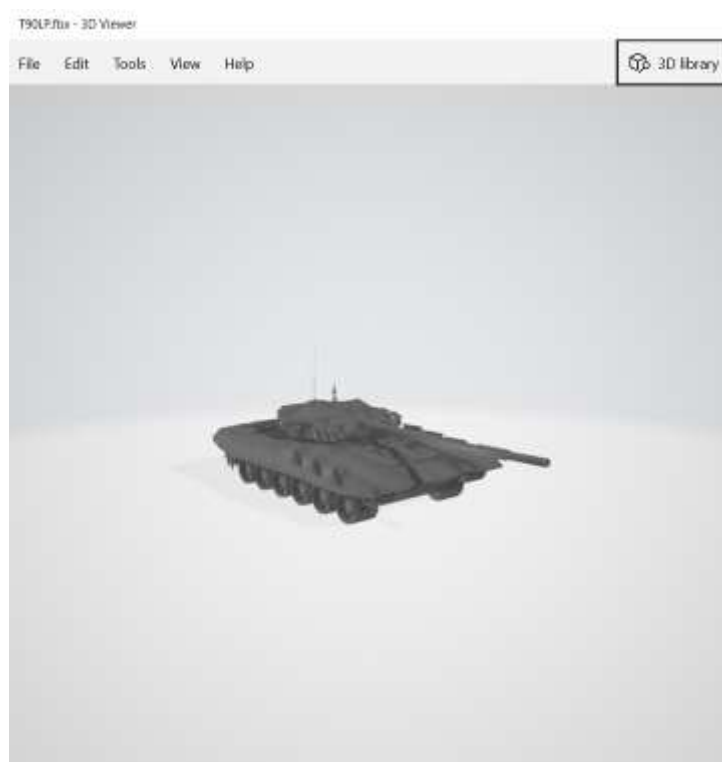


Рисунок 3.2 – 3D-модель нашого об'єкту

Якщо додати до нього матеріали та конвертувати в об'єкт, з яким непосредньо працює Unity, отримаємо такий вигляд, як показано на рисунку 3.3.



Рисунок 3.3 – Об'єкт в Unity

Основна сцена, яку бачить користувач у результаті, має розширення .unity.

Інші файли, що нас цікавлять, – розроблені власноруч скрипти із розширенням .cs. Найголовніші з них по темі – це Controller, який працює з нашим об'єктом, та BezierPath, який розраховує траєкторію руху.

Controller.cs має такі важливі функції:

- IsCanMove, що перевіряє чи всі дані є необхідні для побудови кривої та руху об'єкту;
- Move, яка саме змінює актуальну позицію об'єкту;
- Rotate, що повертає наш об'єкт у бік руху;
- SetSpeed – задає швидкість руху.

3.2 Проєктування інтерфейсу

Перед початком роботи з vuforia потрібно зареєструватись на сайті, обрати тип роботи з нею: звичайний або купити преміум, який розширить можливості та видалить водяний знак з розроблюваного застосунку, і отримати ліцензійний ключ для роботи з цією бібліотекою (рис. 3.4).

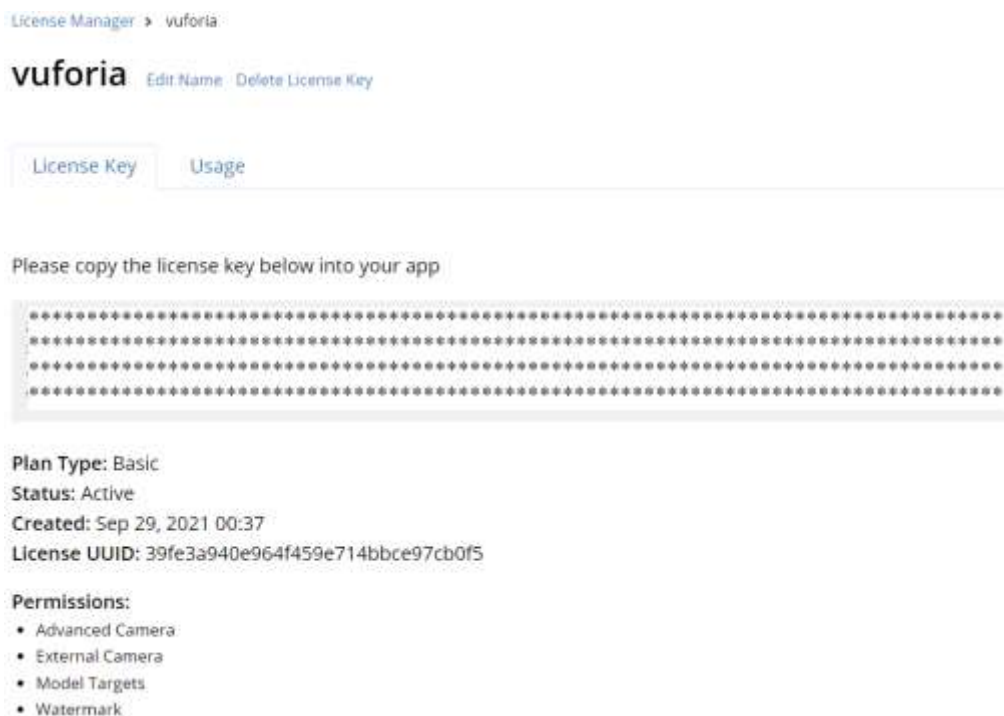


Рисунок 3.4 – Отримання ліцензії Vuforia

Отриманий ключ додаємо в проєкт через інспектор Unity у об'єкта конфігуратора Vuforia (рис. 3.5).

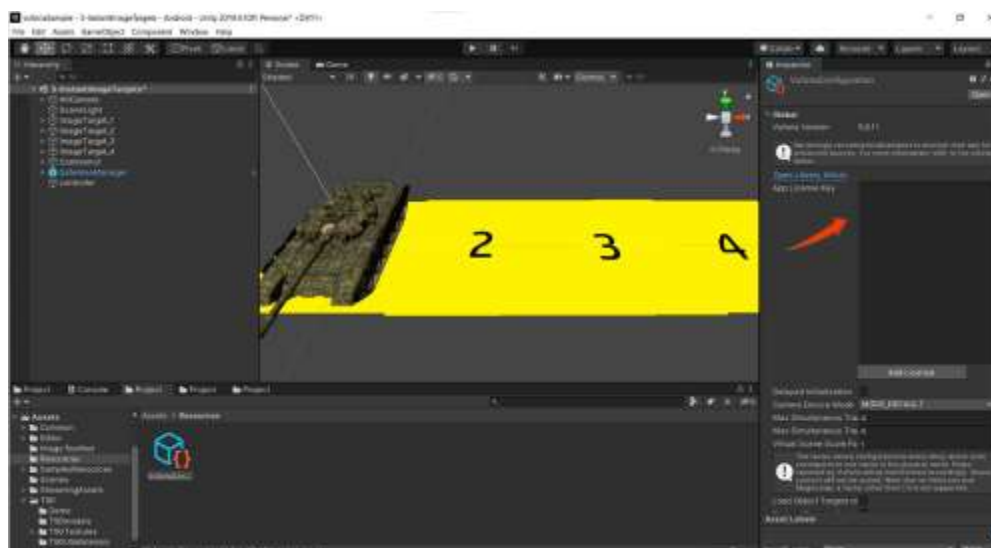


Рисунок 3.5 – Конфігуратор в інспекторі Vuforia

Для роботи з об'єктами Unity надає три основні вікна: ієрархія, сцена та вищезгаданий інспектор.

У вікні ієрархії ми створюємо об'єкти, які можуть мати будь-який рівень вкладеності (рис. 3.6). Вкладені об'єкти мають залежність від своїх «батьків» і, наприклад, переміщуючи основний його дочірні також змінюють свої глобальні координати, але не змінюють локальні. Локальні координати – це і є ті координати, які розраховуються по відношенню до батьківського елемента.

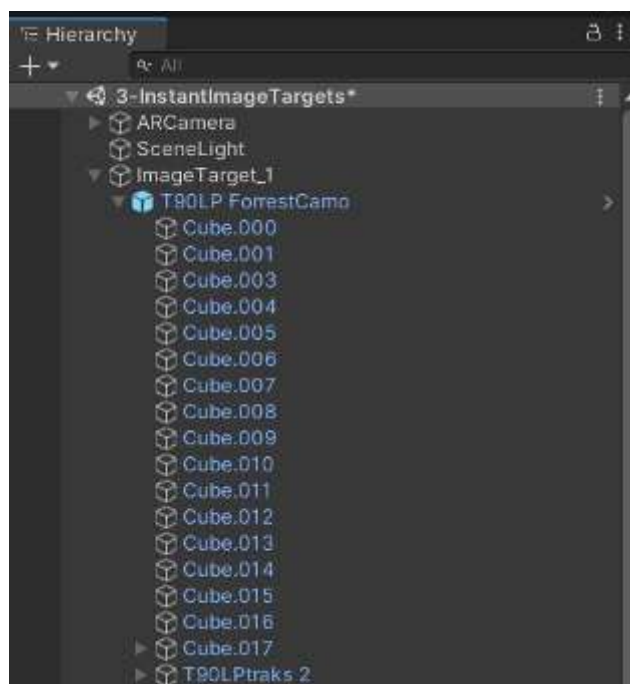


Рисунок 3.6 – Ієрархія об'єктів

На сцені ми можемо також маніпулятором «миша» розміщати об'єкти і наочно бачити вихідний результат. При чому, є багато режимів відображення: плоскі або об'ємні, у локальних координатах або у глобальних і таке інше.

У інспекторі ми налаштовуємо кожний елемент. Змінюємо його характеристики, такі як положення, масштаб, поворот, та інше. Додаємо необхідні компоненти, які можуть бути як стандартні властивості з Unity або з інших встановлених бібліотек до проєкту, так і власні написані скрипти (рис. 3.7).

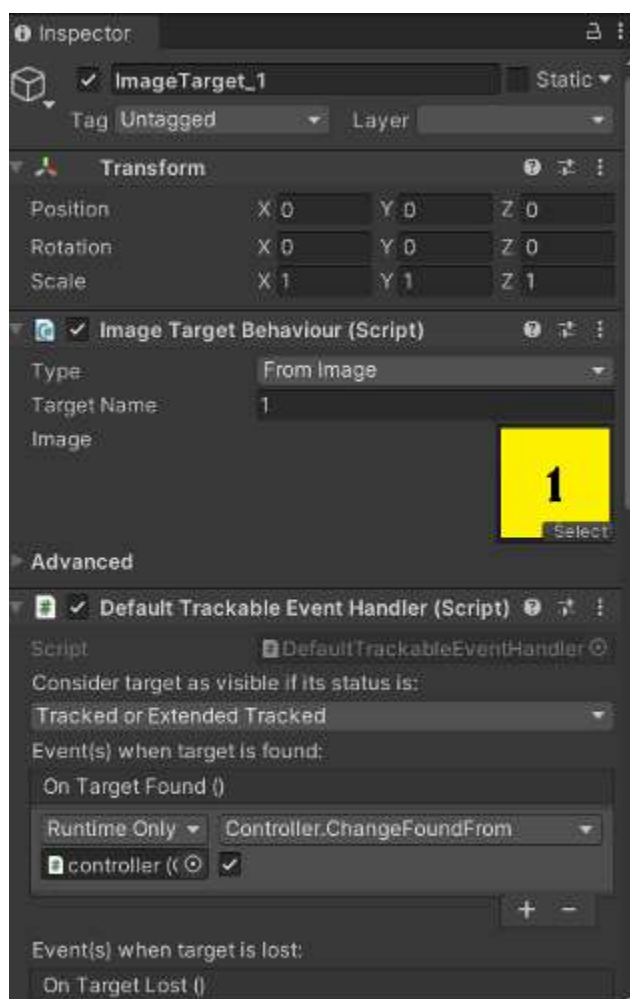


Рисунок 3.7 – Частина компонентів цільового зображення для розпізнавання

Як вже згадувалося, інтерфейсом є камера пристрою та візуалізовані об'єкти. Тому розробка інтерфейсу закінчується тим, що ми додали на сцену цільові зображення для розпізнавання, що слугують нам в якості точок, які необхідні для побудови кривої Безьє.

Для того щоб ми в результаті просто побачили бажані об'єкти по маркеру, потрібно встановити його, як дочірній до цього маркеру, як ми бачили на рисунку 3.3 у ієрархії та на рисунку 3.2 на сцені.

3.3 Розробка логіки

Для керування об'єктами створимо власний скрипт з назвою Controller. Є декілька способів, які дозволять оперувати об'єктами на сцені. Один із – це

серіалізувати їх визначення в коді, як результат – поля під визначені об'єкти відображатимуться в інспекторі скрипта (рис. 3.8, рис. 3.9).

```
// маркери
[SerializeField]
Transform[] targets;

// наш об'єкт, що переміщується
[SerializeField]
Transform obj;

// швидкість переміщення
[SerializeField]
private float speedModifier;
```

Рисунок 3.8 – Визначення серіалізованих полів

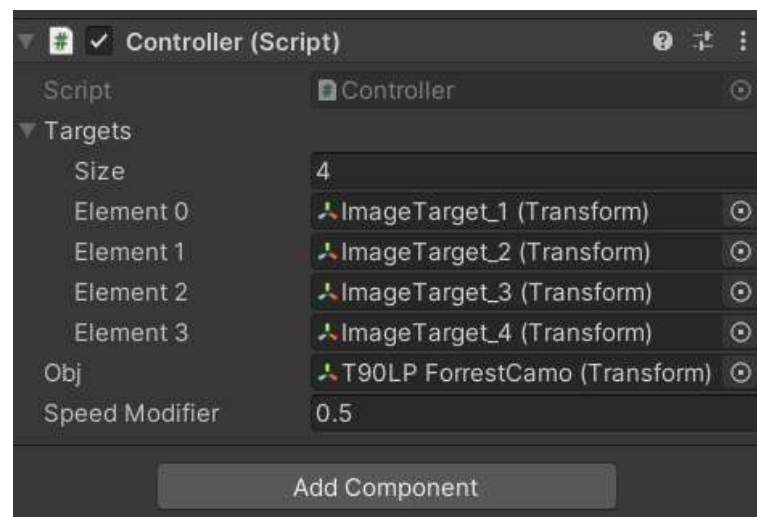


Рисунок 3.9 – Серіалізовані поля в інспекторі

Основний алгоритм програми працює так, як представлено у блок-схемі (рис. 3.10).

Алгоритм роботи кривої Безьє розробляємо згідно формул, що були представлені у другому розділі. Приведемо блок-схеми цього алгоритму (рис. 3.11 – рис. 3.13).



Рисунок 3.10 – Основний алгоритм

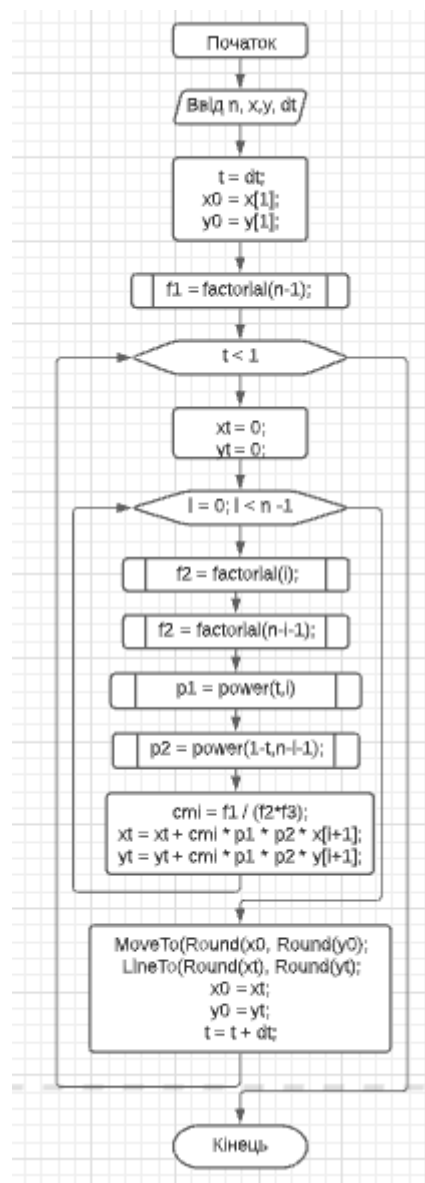


Рисунок 3.11 – Блок-схема алгоритму Безьє

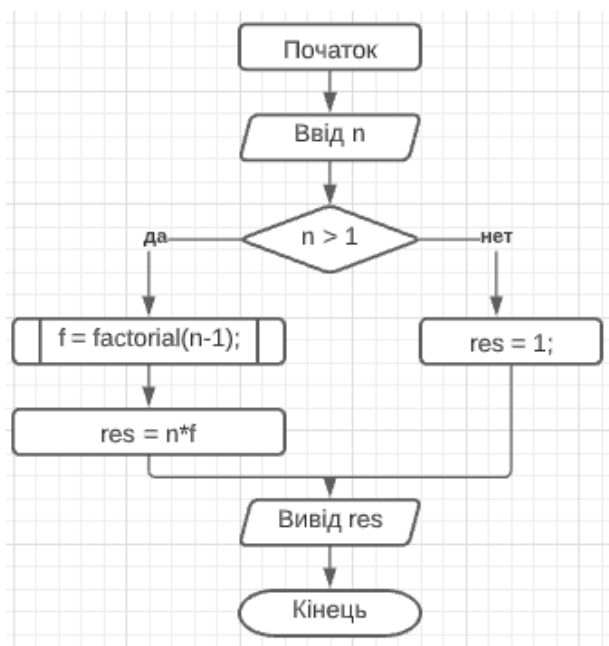


Рисунок 3.12 – Функція factorial з основного алгоритму

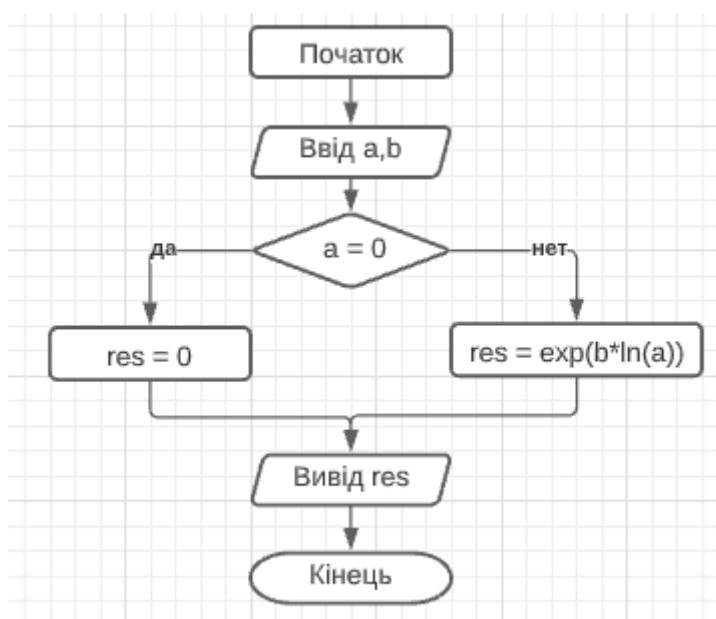


Рисунок 3.13 – Функція power з основного алгоритму

Представлений алгоритм працює в корутині. Це така функція, що працює начебто паралельно іншим процедурам, не зупиняючи іншу логіку. Їх зручно використовувати, коли потрібно працювати з анімаціями, переміщеннями, тощо. Але слід розуміти, що відбувається вона в тому ж потоці, що і уся програма, якщо не було реалізовано інші потоки. Просто корутина виконується між іншими процесами, надаючи їм пріоритет стільки,

скільки визначив програміст. Наприклад, можна задати, щоб вона чекала відновлення кадру, можна задати очікування в секундах, або визначити спеціальну умови виконання або очікування функції.

В нашому випадку використовується два типи. Спочатку ми чекаємо до тих пір, поки система не розпізнає усі маркери (рис. 3.14). І далі, на відтворення переміщення, ми чекаємо одне відновлення кадру екрану (рис. 3.15).

```
yield return new WaitUntil(() => isCanBuildCurve == false);
```

Рисунок 3.14 – Очікування корутини по умові

```
yield return new WaitForEndOfFrame();
```

Рисунок 3.15 – Очікування корутини відновлення одного кадру

При розробці застосунку ми дотримувалися парадигми об'єктно-орієнтованого програмування, так як уся Unity побудована на об'єктах з різними властивостями та методами. Неможливо не дотриматись принципів цієї парадигми у ігрових рушіях працюючи виключно із об'єктами цього рушія.

3.4 Висновок до розділу

В даному розділі описано процес розробки графічних та логічних компонентів застосунку. Розроблено блок-схеми основного алгоритму та алгоритму функцій керувати об'єктами у рушії, що необхідно розробити в Unity.

4 ЕКСПЛУАТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для візуалізації руху транспортних засобів у доповненій реальності. Для допрацювання розробленого програмного продукту, необхідно мати таку мінімальну апаратну платформу: персональний комп'ютері з тактовою частотою мікропроцесора не менше 2 ГГц та об'ємом оперативної пам'яті не менше 4 Гб. На персональному комп'ютері повинна бути встановлена операційна система Windows 10 з підтримкою .NET Framework 4.0. Для функціонування цього продукту необхідно мати будь-який пристрій, що має камеру та можливість встановлювання сторонніх застосунків, а також повинні бути підготовлені спеціальні графічні маркери, по яким орієнтується система.

4.2 Початкові та вихідні дані

Початковими або вхідними даними є саме графічні маркери, які необхідні системі для роботи взагалі.

Вихідними є візуалізація руху по побудованій кривій.

4.3 Звертання до програми

Для доступу до програмного забезпечення потрібно запуснути встановлений заздалегідь застосунок, розмістити фізичні карточки-маркери так, як ми хочемо будувати криву. Після цих дій ми побачимо результат роботи застосунку.

4.4 Робота з застосунком

Для демонстрації роботи застосунку проведемо діаграму послідовності (рис. 4.1).

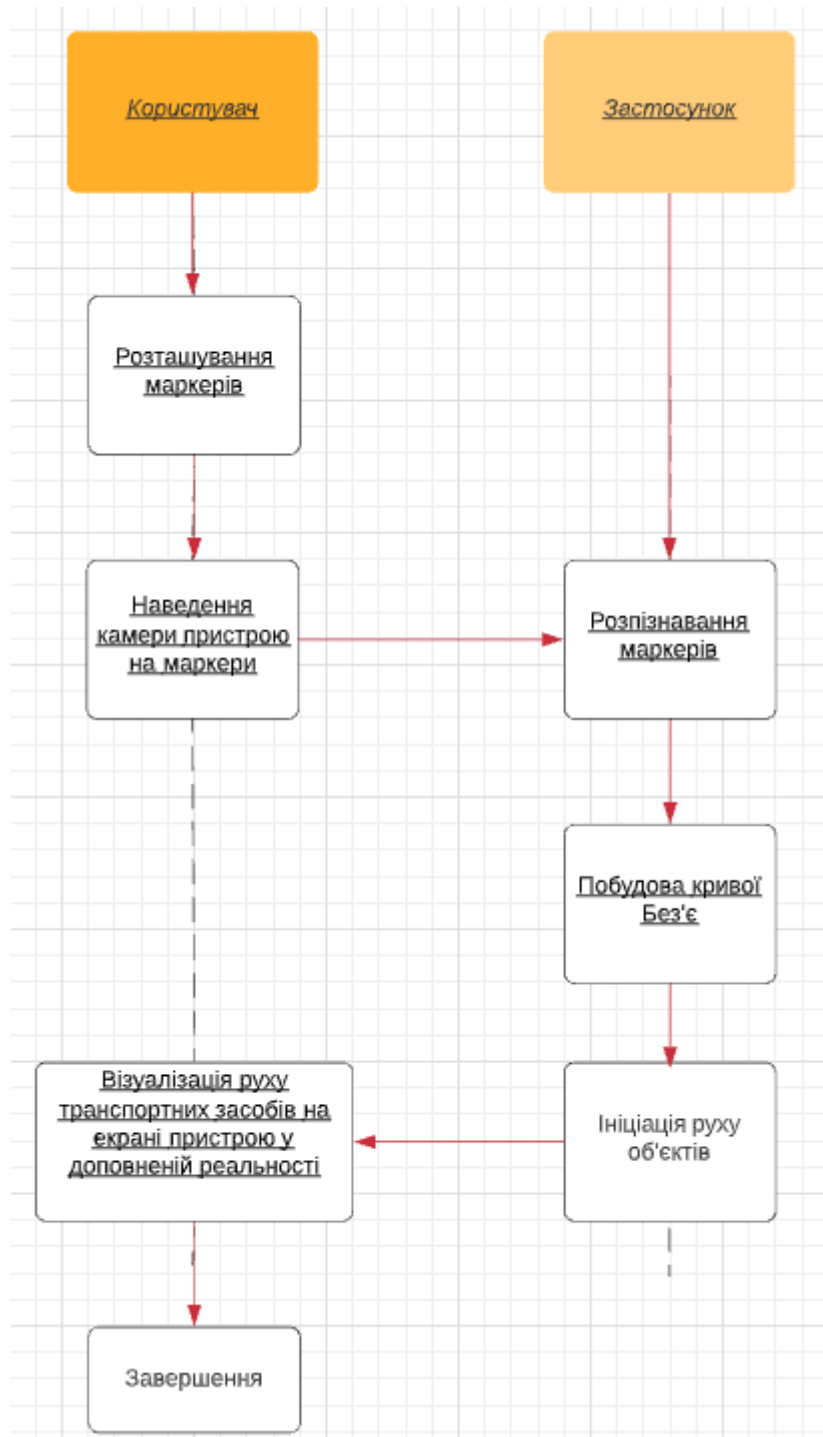


Рисунок 4.1 – Діаграма послідовності робіт застосунку

Для того, щоб відтворити цей застосунок нам потрібно:

- скачати та встановити Unity;
- відкрити в ньому цей проєкт;
- побудувати рішення під обрану операційну систему, при чому пам'ятати, що якщо ми бажаємо побудувати його під iOS, то ми маємо це робити виключно на macOS;
- встановити вихідний файл на бажаний пристрій;
- підготувати та розташувати відповідні маркери;
- навести камеру пристрою так, щоб у області її бачення було видно усі 4 маркери;
- переконатися, що все працює.

Коли усі ці пункти виконані, ми можемо бачити таку поведінку (рис. 4.2 – рис. 4.4) при одному розположенні маркерів, та іншу (рис. 4.5, рис. 4.6) при іншому.

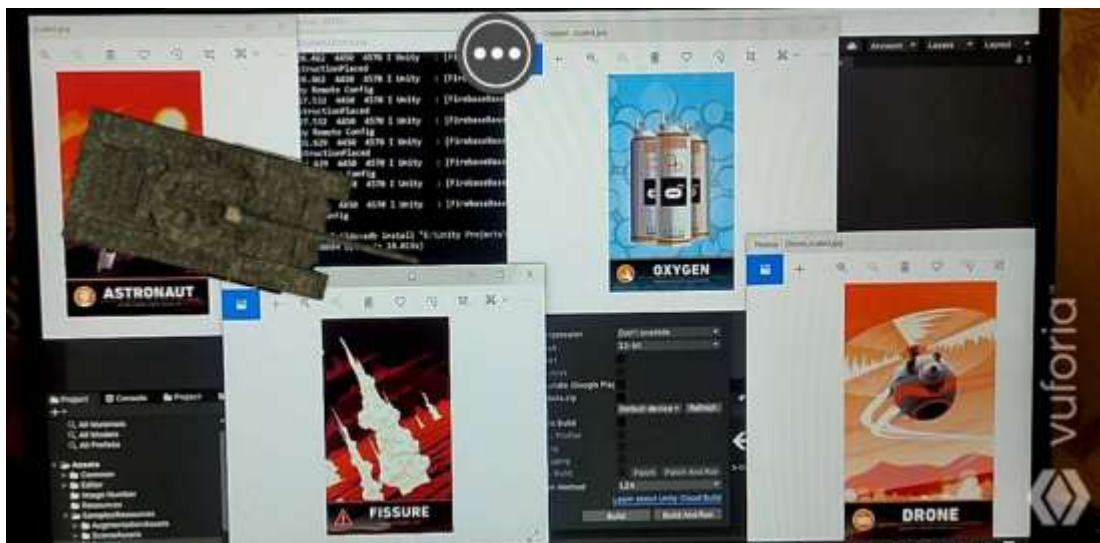


Рисунок 4.2 – Початкова позиція об'єкту



Рисунок 4.3 – Об'єкт у русі



Рисунок 4.4 – Об'єкт на останній позиції

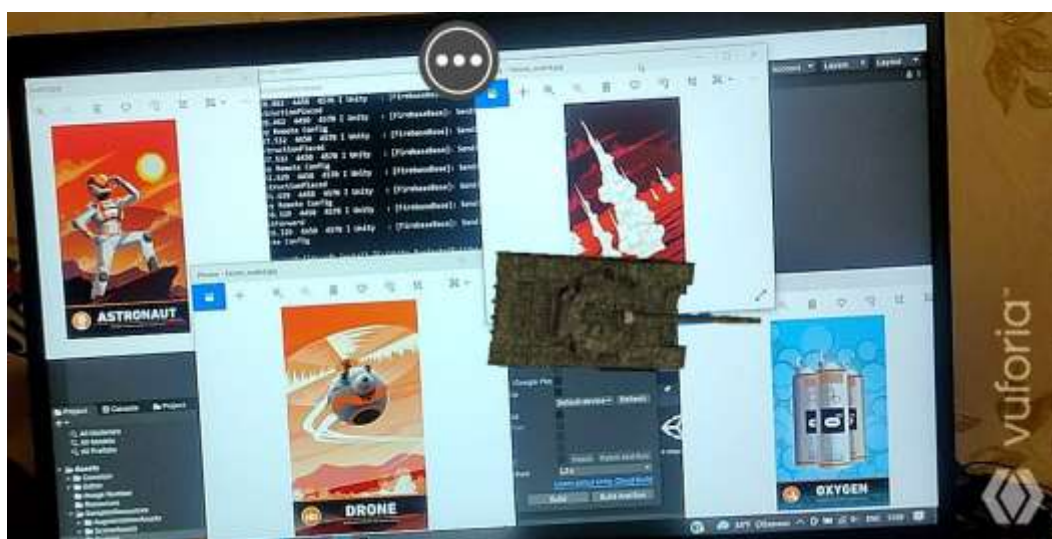


Рисунок 4.5 – Об'єкт у русі при іншому розположенні маркерів

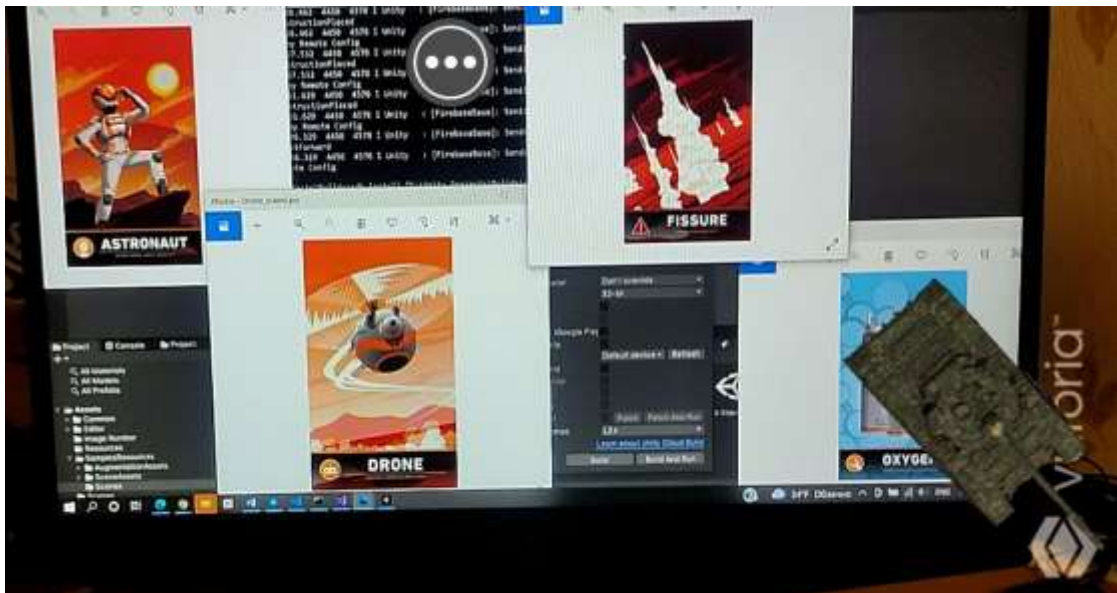


Рисунок 4.6 – Об’єкт на останній позиції

Як бачимо, об’єкт завжди проходить через першу та останню точку, але не торкаючись проміжних, а лише змінює траєкторію.

4.5 Опис методики проведення тестування

При тестуванні були виконані такі дії:

- змінювали маркерні зображення;
- змінювали позиції маркерів;
- перевіряли систему з недостатнім освітленням;
- дивилися на роботу алгоритму, якщо не усі маркери знаходяться в зоні камери.

Приведемо декілька прикладів роботи.

На рисунку 4.7 система наче бачить маркери, але процент коректної роботи досить низкий, через те, що маркери дуже схожі між собою.

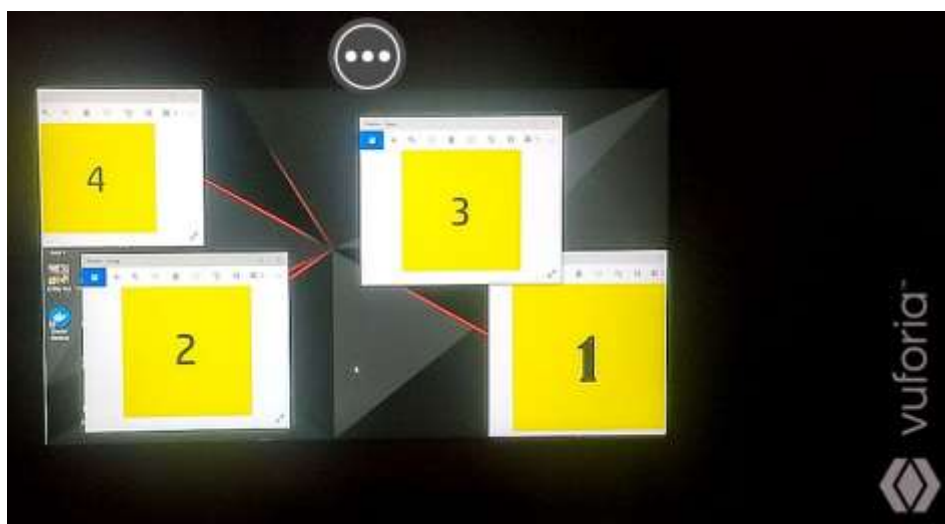


Рисунок 4.7 – Подібні маркери, які важко розпізнаються

На рисунку 4.8 бачимо, що взагалі нічого не відбувається, так як система «не бачить» визначених маркерів.

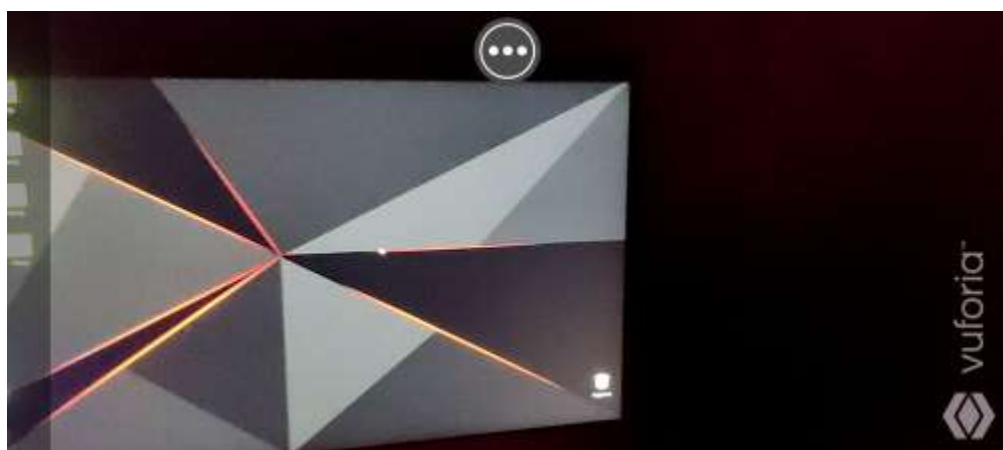


Рисунок 4.8 – Відсутність маркерів

На рисунку 4.9 система розпізнала один маркер, відобразила об'єкт і більше нічого не робить, так як не має ключових точок, за якими потрібно будувати криву Безьє.

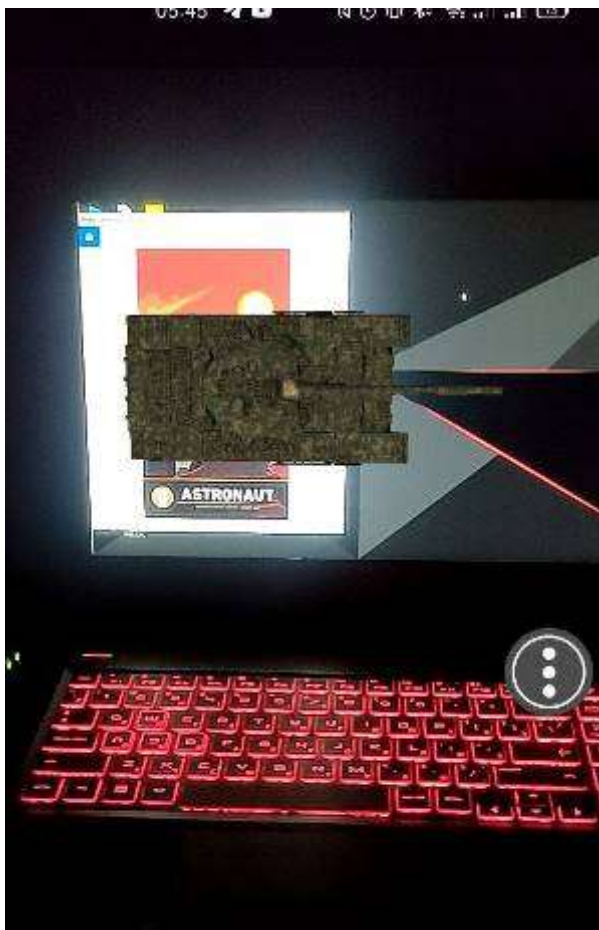


Рисунок 4.9 – Один розпізнаний маркер

В результаті тестування прийшли висновку, що для того, щоб система працювала більш коректно, потрібно, щоб маркери сильно відрізнялись один від одного. Можемо спостерігати їх заміну від частини, де ми будували інтерфейс, до того, що бачимо в попередньому розділі.

Від зміни позицій об'єкт коректно змінює свою траєкторію.

При не усіх доступних маркерів у камері алгоритм продовжує працювати, але через обмеженість камери та орієнтування системи на положення пристрою, ми не маємо можливості бачити де там рухається наш об'єкт.

Отже можна ще працювати та удосконалювати роботу усієї системи.

4.6 Висновок до розділу

В даному розділі розглянуто призначення та умови застосування програми. Розроблена діаграма послідовності робот застосунку. Та наведено послідовність дії, які потрібно зробити, щоб відтворити роботу з розробленим застосунком. Крім того, наведено приклад тестування розробленого програмного забезпечення.

ВИСНОВКИ

У ході виконання дипломної кваліфікаційної роботи магістра проведено детальне ознайомлення із предметною областю. Дослідження проводилося на основі системно-функціонального, динамічного, комплексного наукових підходів, за допомогою наукових методів: аналізу і синтезу, індукції і дедукції, моделювання, історико-логічного методу, методів статистичного аналізу.

В першому розділі було розглянуто доповнену та віртуальну реальності, їх коротку історію і важливість. Також торкнулися питання логістики транспортів. Було сформульовано основні функціональні вимоги до розроблюваної програми.

В другому розділі був проведений аналіз та обґрунтування вибору середовища розробки, ігрового рушія, мови програмування, для розробки застосунку і опис саме алгоритму побудови кривих Безьє. В якості мови розробки програми було обрано C#. Для створення програмного забезпечення використовувалось середовище розробки Microsoft Visual Studio. Ігровим рушієм було обрано Unity. Для доповненої реальності користувалися бібліотекою Vuforia.

В третьому розділі було розглянуті основні рішення щодо реалізації компонентів системи. Було реалізовано основні функції для роботи з доповненою реальністю.

Як результат, було розроблено програмний застосунок який реалізовував усі поставлені задачі та список функціональних вимог. Тестування довело працездатність програми та стабільність у роботі, але й показало слабкі місця системи. В подальшому застосунок можливо покращувати, додавати нові функції та можливості.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Історія віртуальної реальності с 19-го століття по наші дні [Електронний ресурс]. – Режим доступу: <https://www.imena.ua/blog/the-history-of-virtual-reality/>
2. Історія віртуальної реальності [Електронний ресурс]. – Режим доступу: <https://medium.com/@gwaramedia/історія-віртуальної-реальности-7a8b71edddfa>.
3. Доповнена реальність (Augmented Reality, AR) [Електронний ресурс]. – Режим доступу: <https://lookinar.com/uk/rozyasnennya/dopovnena-realnistaugmented-reality-ar/>
4. Augmented Reality, AR [Electronic source]. – Access mode: <https://www.it.ua/knowledge-base/technology-innovation/dopolnennaja-realnost-ar>
5. Що таке доповнена реальність – технології, приклади та історія [Електронний ресурс]. – Режим доступу: https://uk.myservername.com/what-is-augmented-reality-technology#3_Project-based_AR
6. Методи транспортної логістики [Електронний ресурс]. – Режим доступу: http://lib.kart.edu.ua/bitstream/Силабус_МТЛ.pdf
7. Методи і моделі вирішення завдань транспортної логістики [Електронний ресурс]. – Режим доступу: <http://repository.vsau.org/getfile.php/4193.pdf>
8. Забуранна Л. Оптимізаційні методи та моделі / Л. Забуранна, Н. Попрозман, Н. Клименко, О. Попрозман // К И Ї В, 2014. – 372с.
9. Оптимізація маршрутів громадського транспорту – Режим доступу: <https://apluss.pro/upravlinnya-hromadskym-transportom/optimizacija-marshrutiv-gromadskogo-transportu/>
10. Цветов Ю. Залізничний транспорт України на порозі реформування / [Ю. Цветов, М. Макаренко, А Лашко. та ін.]. – Київ : ДЕТУТ, 2008. – 189 с.
11. Котлубая О. Конкурентоспроможність та сталий розвиток морегосподарського комплексу України / О. Котлубая. – Одеса : ППРЕД, 2011.

– 427 с.

12. Моделювання транспортного потоку на перетинах в різних рівнях [Електронний ресурс]. – Режим доступу: <https://bespalov.me/2014/01/08/modelyuvannya-transportnogo-potoku-na-peretynah-v-riznyh-rivnyah/>

13. Пустюльга С. Багатовимірна геометрична модель функціонування транспортного потоку на перевантаженній ділянці дорожньої мережі / С. Пустюльга, В. Придюк, В. Самостян. – Луцький національний технічний університет, 2019. – 11с.

14. Крива Безьє [Електронний ресурс]. – Режим доступу: <https://studfile.net/preview/5199908/page:24/>

15. Криві Безьє. Основні поняття та властивості кривих Безьє [Електронний ресурс]. – Режим доступу: <https://www.mathros.net.ua/kryvi-bezje.html>

16. 10 найкращих ігрових рушіїв [Електронний ресурс]. – Режим доступу: <https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiiv>

17. 10 кращих ігрових рушіїв [Електронний ресурс]. – Режим доступу: <https://senfil.net/index.php?newsid=321>

ДОДАТОК А
Текст програми

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Controller : MonoBehaviour
{
    // маркери
    [SerializeField]
    Transform[] targets;

    // наш об'єкт, що переміщується
    [SerializeField]
    Transform obj;

    //швидкість переміщення
    [SerializeField]
    private float speed;

    // кількість розпізнаних маркерів. Для перевірки чи можемо ми побудувати
    криву
    private List<string> foundTargets = new List<string>();

    private bool allowBuild;

    private float time;

    private bool isCanBuildCurve
    {
        get
        {
            return foundTargets.Count == 4;
        }
    }

    void Start()
    {
        time = 0f;
        allowBuild = true;
    }

    public void FoundTarget(string name)

```

```

{
    if (!foundTargets.Contains(name))
        foundTargets.Add(name);
}
public void LostTarget(string name)
{
    foundTargets.Remove(name);
}

IEnumerator Move()
{
    allowBuild = false;

    var p0 = targets[0].position;
    var p1 = targets[1].position;
    var p2 = targets[2].position;
    var p3 = targets[3].position;

    yield return new WaitUntil(() => isCanBuildCurve == false);
    while (time < 1)
    {
        time += Time.deltaTime * speed;

        var pos = Mathf.Pow(1 - time, 3) * p0 + 3 * Mathf.Pow(1 - time, 2) * time *
p1 + 3 * (1 - time) * Mathf.Pow(time, 2) * p2 + Mathf.Pow(time, 3) * p3;
        obj.forward = pos - obj.position;
        obj.position = pos;

        yield return new WaitForEndOfFrame();
    }

    time = 0f;

    allowBuild = true;

    while (true)
    {
        yield return new WaitForEndOfFrame();

        float step = Time.deltaTime; // calculate distance to move
    }
}

```

```

    }
    void Update()
    {
        if (allowBuild)
        {
            StartCoroutine(Move());
        }
    }
}

```

```

public static class MathUtility {

```

```

    static PosRotScale LockTransformToSpace (Transform t, PathSpace space) {
        var original = new PosRotScale (t);
        if (space == PathSpace.xy) {
            t.eulerAngles = new Vector3 (0, 0, t.eulerAngles.z);
            t.position = new Vector3 (t.position.x, t.position.y, 0);
        } else if (space == PathSpace.xz) {
            t.eulerAngles = new Vector3 (0, t.eulerAngles.y, 0);
            t.position = new Vector3 (t.position.x, 0, t.position.z);
        }
    }

```

```

        //float maxScale = Mathf.Max (t.localScale.x * t.parent.localScale.x,
t.localScale.y * t.parent.localScale.y, t.localScale.z * t.parent.localScale.z);
        float maxScale = Mathf.Max (t.lossyScale.x, t.lossyScale.y, t.lossyScale.z);

        t.localScale = Vector3.one * maxScale;

        return original;
    }

```

```

    public static bool LineSegmentsIntersect (Vector2 a1, Vector2 a2, Vector2 b1,
Vector2 b2) {
        float d = (b2.x - b1.x) * (a1.y - a2.y) - (a1.x - a2.x) * (b2.y - b1.y);
        if (d == 0)
            return false;
        float t = ((b1.y - b2.y) * (a1.x - b1.x) + (b2.x - b1.x) * (a1.y - b1.y)) / d;
        float u = ((a1.y - a2.y) * (a1.x - b1.x) + (a2.x - a1.x) * (a1.y - b1.y)) / d;

        return t >= 0 && t <= 1 && u >= 0 && u <= 1;
    }

```

```

    public static bool LinesIntersect (Vector2 a1, Vector2 a2, Vector2 a3, Vector2
a4) {

```

```

    return (a1.x - a2.x) * (a3.y - a4.y) - (a1.y - a2.y) * (a3.x - a4.x) != 0;
}

public static Vector2 PointOfLineLineIntersection (Vector2 a1, Vector2 a2,
Vector2 a3, Vector2 a4) {
    float d = (a1.x - a2.x) * (a3.y - a4.y) - (a1.y - a2.y) * (a3.x - a4.x);
    if (d == 0) {
        Debug.LogError ("Lines are parallel, please check that this is not the case
before calling line intersection method");
        return Vector2.zero;
    } else {
        float n = (a1.x - a3.x) * (a3.y - a4.y) - (a1.y - a3.y) * (a3.x - a4.x);
        float t = n / d;
        return a1 + (a2 - a1) * t;
    }
}

public static Vector2 ClosestPointOnLineSegment (Vector2 p, Vector2 a,
Vector2 b) {
    Vector2 aB = b - a;
    Vector2 aP = p - a;
    float sqrLenAB = aB.sqrMagnitude;

    if (sqrLenAB == 0)
        return a;

    float t = Mathf.Clamp01 (Vector2.Dot (aP, aB) / sqrLenAB);
    return a + aB * t;
}

public static Vector3 ClosestPointOnLineSegment (Vector3 p, Vector3 a,
Vector3 b) {
    Vector3 aB = b - a;
    Vector3 aP = p - a;
    float sqrLenAB = aB.sqrMagnitude;

    if (sqrLenAB == 0)
        return a;

    float t = Mathf.Clamp01 (Vector3.Dot (aP, aB) / sqrLenAB);
    return a + aB * t;
}

public static int SideOfLine (Vector2 a, Vector2 b, Vector2 c) {

```

```

    return (int) Mathf.Sign ((c.x - a.x) * (-b.y + a.y) + (c.y - a.y) * (b.x - a.x));
}

/// returns the smallest angle between ABC. Never greater than 180
public static float MinAngle (Vector3 a, Vector3 b, Vector3 c) {
    return Vector3.Angle ((a - b), (c - b));
}

public static bool PointInTriangle (Vector2 a, Vector2 b, Vector2 c, Vector2 p) {
    float area = 0.5f * (-b.y * c.x + a.y * (-b.x + c.x) + a.x * (b.y - c.y) + b.x *
c.y);
    float s = 1 / (2 * area) * (a.y * c.x - a.x * c.y + (c.y - a.y) * p.x + (a.x - c.x) *
p.y);
    float t = 1 / (2 * area) * (a.x * b.y - a.y * b.x + (a.y - b.y) * p.x + (b.x - a.x) *
p.y);
    return s >= 0 && t >= 0 && (s + t) <= 1;
}

public static bool PointsAreClockwise (Vector2[] points) {
    float signedArea = 0;
    for (int i = 0; i < points.Length; i++) {
        int nextIndex = (i + 1) % points.Length;
        signedArea += (points[nextIndex].x - points[i].x) * (points[nextIndex].y +
points[i].y);
    }

    return signedArea >= 0;
}

class PosRotScale {
    public readonly Vector3 position;
    public readonly Quaternion rotation;
    public readonly Vector3 scale;

    public PosRotScale (Transform t) {
        this.position = t.position;
        this.rotation = t.rotation;
        this.scale = t.localScale;
    }

    public void SetTransform (Transform t) {
        t.position = position;
        t.rotation = rotation;
        t.localScale = scale;
    }
}

```

```

    }
    }
}

public static class CubicBezierUtility {

    /// Returns point at time 't' (between 0 and 1) along bezier curve defined by 4
    /// points (anchor_1, control_1, control_2, anchor_2)
    public static Vector3 EvaluateCurve (Vector3[] points, float t) {
        return EvaluateCurve (points[0], points[1], points[2], points[3], t);
    }

    /// Returns point at time 't' (between 0 and 1) along bezier curve defined by 4
    /// points (anchor_1, control_1, control_2, anchor_2)
    public static Vector3 EvaluateCurve (Vector3 a1, Vector3 c1, Vector3 c2,
    Vector3 a2, float t) {
        t = Mathf.Clamp01 (t);
        return (1 - t) * (1 - t) * (1 - t) * a1 + 3 * (1 - t) * (1 - t) * t * c1 + 3 * (1 - t) * t
        * t * c2 + t * t * t * a2;
    }

    /// Returns a vector tangent to the point at time 't'
    /// This is the vector tangent to the curve at that point
    public static Vector3 EvaluateCurveDerivative (Vector3[] points, float t) {
        return EvaluateCurveDerivative (points[0], points[1], points[2], points[3], t);
    }

    /// Calculates the derivative of the curve at time 't'
    /// This is the vector tangent to the curve at that point
    public static Vector3 EvaluateCurveDerivative (Vector3 a1, Vector3 c1, Vector3
    c2, Vector3 a2, float t) {
        t = Mathf.Clamp01 (t);
        return 3 * (1 - t) * (1 - t) * (c1 - a1) + 6 * (1 - t) * t * (c2 - c1) + 3 * t * t * (a2
        - c2);
    }

    /// Returns the second derivative of the curve at time 't'
    public static Vector3 EvaluateCurveSecondDerivative (Vector3[] points, float t)
    {
        return EvaluateCurveSecondDerivative (points[0], points[1], points[2],
        points[3], t);
    }

    ///Returns the second derivative of the curve at time 't'

```

```

    public static Vector3 EvaluateCurveSecondDerivative (Vector3 a1, Vector3 c1,
Vector3 c2, Vector3 a2, float t) {
    t = Mathf.Clamp01 (t);
    return 6 * (1 - t) * (c2 - 2 * c1 + a1) + 6 * t * (a2 - 2 * c2 + c1);
}

```

/// Calculates the normal vector (vector perpendicular to the curve) at specified time

```

public static Vector3 Normal (Vector3[] points, float t) {
    return Normal (points[0], points[1], points[2], points[3], t);
}

```

/// Calculates the normal vector (vector perpendicular to the curve) at specified time

```

public static Vector3 Normal (Vector3 a1, Vector3 c1, Vector3 c2, Vector3 a2,
float t) {
    Vector3 tangent = EvaluateCurveDerivative (a1, c1, c2, a2, t);
    Vector3 nextTangent = EvaluateCurveSecondDerivative (a1, c1, c2, a2, t);
    Vector3 c = Vector3.Cross (nextTangent, tangent);
    return Vector3.Cross (c, tangent).normalized;
}

```

```

    public static Bounds CalculateSegmentBounds (Vector3 p0, Vector3 p1,
Vector3 p2, Vector3 p3) {

```

```

    MinMax3D minMax = new MinMax3D ();
    minMax.AddValue (p0);
    minMax.AddValue (p3);

```

```

    List<float> extremePointTimes = ExtremePointTimes (p0,p1,p2,p3);
    foreach (float t in extremePointTimes) {
        minMax.AddValue (CubicBezierUtility.EvaluateCurve (p0, p1, p2, p3, t));
    }

```

```

    return new Bounds ((minMax.Min + minMax.Max) / 2, minMax.Max -
minMax.Min);
}

```

/// Splits curve into two curves at time t. Returns 2 arrays of 4 points.

```

public static Vector3[][] SplitCurve (Vector3[] points, float t) {
    Vector3 a1 = Vector3.Lerp (points[0], points[1], t);
    Vector3 a2 = Vector3.Lerp (points[1], points[2], t);
    Vector3 a3 = Vector3.Lerp (points[2], points[3], t);
    Vector3 b1 = Vector3.Lerp (a1, a2, t);
    Vector3 b2 = Vector3.Lerp (a2, a3, t);

```

```

Vector3 pointOnCurve = Vector3.Lerp (b1, b2, t);

return new Vector3[][] {
    new Vector3[] { points[0], a1, b1, pointOnCurve },
    new Vector3[] { pointOnCurve, b2, a3, points[3] }
};
}

// Crude, but fast estimation of curve length.
public static float EstimateCurveLength (Vector3 p0, Vector3 p1, Vector3 p2,
Vector3 p3) {
    float controlNetLength = (p0 - p1).magnitude + (p1 - p2).magnitude + (p2 -
p3).magnitude;
    float estimatedCurveLength = (p0 - p3).magnitude + controlNetLength / 2f;
    return estimatedCurveLength;
}

/// Times of stationary points on curve (points where derivative is zero on any
axis)
public static List<float> ExtremePointTimes (Vector3 p0, Vector3 p1, Vector3
p2, Vector3 p3) {
    // coefficients of derivative function
    Vector3 a = 3 * (-p0 + 3 * p1 - 3 * p2 + p3);
    Vector3 b = 6 * (p0 - 2 * p1 + p2);
    Vector3 c = 3 * (p1 - p0);

    List<float> times = new List<float> ();
    times.AddRange (StationaryPointTimes (a.x, b.x, c.x));
    times.AddRange (StationaryPointTimes (a.y, b.y, c.y));
    times.AddRange (StationaryPointTimes (a.z, b.z, c.z));
    return times;
}

// Finds times of stationary points on curve defined by  $ax^2 + bx + c$ .
// Only times between 0 and 1 are considered as Bezier only uses values in that
range
static IEnumerable<float> StationaryPointTimes (float a, float b, float c) {
    List<float> times = new List<float> ();

    // from quadratic equation:  $y = [-b \pm \sqrt{b^2 - 4ac}]/2a$ 
    if (a != 0) {
        float discriminant = b * b - 4 * a * c;
        if (discriminant >= 0) {
            float s = Mathf.Sqrt (discriminant);

```

```

        float t1 = (-b + s) / (2 * a);
        if (t1 >= 0 && t1 <= 1) {
            times.Add (t1);
        }

        if (discriminant != 0) {
            float t2 = (-b - s) / (2 * a);

            if (t2 >= 0 && t2 <= 1) {
                times.Add (t2);
            }
        }
    }
    return times;
}

}

public static class VertexPathUtility
{
    public static PathSplitData SplitBezierPathByAngleError(BezierPath
bezierPath, float maxAngleError, float minVertexDst, float accuracy)
    {
        PathSplitData splitData = new PathSplitData();

        splitData.vertices.Add(bezierPath[0]);

        splitData.tangents.Add(CubicBezierUtility.EvaluateCurveDerivative(bezierPath.GetP
ointsInSegment(0), 0).normalized);
        splitData.cumulativeLength.Add(0);
        splitData.anchorVertexMap.Add(0);
        splitData.minMax.AddValue(bezierPath[0]);

        Vector3 prevPointOnPath = bezierPath[0];
        Vector3 lastAddedPoint = bezierPath[0];

        float currentPathLength = 0;
        float dstSinceLastVertex = 0;

        // Go through all segments and split up into vertices
        for (int segmentIndex = 0; segmentIndex < bezierPath.NumSegments;
segmentIndex++)
        {
            Vector3[] segmentPoints = bezierPath.GetPointsInSegment(segmentIndex);

```

```

        float estimatedSegmentLength =
CubicBezierUtility.EstimateCurveLength(segmentPoints[0], segmentPoints[1],
segmentPoints[2], segmentPoints[3]);
        int divisions = Mathf.CeilToInt(estimatedSegmentLength * accuracy);
        float increment = 1f / divisions;

        for (float t = increment; t <= 1; t += increment)
        {
            bool isLastPointOnPath = (t + increment > 1 && segmentIndex ==
bezierPath.NumSegments - 1);
            if (isLastPointOnPath)
            {
                t = 1;
            }
            Vector3 pointOnPath =
CubicBezierUtility.EvaluateCurve(segmentPoints, t);
            Vector3 nextPointOnPath =
CubicBezierUtility.EvaluateCurve(segmentPoints, t + increment);

            // angle at current point on path
            float localAngle = 180 - MathUtility.MinAngle(prevPointOnPath,
pointOnPath, nextPointOnPath);
            // angle between the last added vertex, the current point on the path, and
the next point on the path
            float angleFromPrevVertex = 180 -
MathUtility.MinAngle(lastAddedPoint, pointOnPath, nextPointOnPath);
            float angleError = Mathf.Max(localAngle, angleFromPrevVertex);

            if ((angleError > maxAngleError && dstSinceLastVertex >=
minVertexDst) || isLastPointOnPath)
            {
                currentPathLength += (lastAddedPoint - pointOnPath).magnitude;
                splitData.cumulativeLength.Add(currentPathLength);
                splitData.vertices.Add(pointOnPath);

                splitData.tangents.Add(CubicBezierUtility.EvaluateCurveDerivative(segmentPoints,
t).normalized);
                splitData.minMax.AddValue(pointOnPath);
                dstSinceLastVertex = 0;
                lastAddedPoint = pointOnPath;
            }
            else

```

```

        {
            dstSinceLastVertex += (pointOnPath - prevPointOnPath).magnitude;
        }
        prevPointOnPath = pointOnPath;
    }
    splitData.anchorVertexMap.Add(splitData.vertices.Count - 1);
}

    return splitData;
}

    public static PathSplitData SplitBezierPathEvenly(BezierPath
bezierPath, float spacing, float accuracy)
    {
        PathSplitData splitData = new PathSplitData();

        splitData.vertices.Add(bezierPath[0]);

        splitData.tangents.Add(CubicBezierUtility.EvaluateCurveDerivative(bezierPath.GetP
ointsInSegment(0), 0).normalized);
        splitData.cumulativeLength.Add(0);
        splitData.anchorVertexMap.Add(0);
        splitData.minMax.AddValue(bezierPath[0]);

        Vector3 prevPointOnPath = bezierPath[0];
        Vector3 lastAddedPoint = bezierPath[0];

        float currentPathLength = 0;
        float dstSinceLastVertex = 0;

        // Go through all segments and split up into vertices
        for (int segmentIndex = 0; segmentIndex < bezierPath.NumSegments;
segmentIndex++)
        {
            Vector3[] segmentPoints = bezierPath.GetPointsInSegment(segmentIndex);
            float estimatedSegmentLength =
CubicBezierUtility.EstimateCurveLength(segmentPoints[0], segmentPoints[1],
segmentPoints[2], segmentPoints[3]);
            int divisions = Mathf.CeilToInt(estimatedSegmentLength * accuracy);
            float increment = 1f / divisions;

            for (float t = increment; t <= 1; t += increment)
            {
                bool isLastPointOnPath = (t + increment > 1 && segmentIndex ==
bezierPath.NumSegments - 1);

```

```

        if (isLastPointOnPath)
        {
            t = 1;
        }
        Vector3 pointOnPath =
CubicBezierUtility.EvaluateCurve(segmentPoints, t);
        dstSinceLastVertex += (pointOnPath -
prevPointOnPath).magnitude;

        // If vertices are now too far apart, go back by amount
we overshoot by
        if (dstSinceLastVertex > spacing) {
            float overshootDst = dstSinceLastVertex -
spacing;
            pointOnPath += (prevPointOnPath -
pointOnPath).normalized * overshootDst;
            t -= increment;
        }

        if (dstSinceLastVertex >= spacing || isLastPointOnPath)
        {
            currentPathLength += (lastAddedPoint - pointOnPath).magnitude;
            splitData.cumulativeLength.Add(currentPathLength);
            splitData.vertices.Add(pointOnPath);

splitData.tangents.Add(CubicBezierUtility.EvaluateCurveDerivative(segmentPoints,
t).normalized);
            splitData.minMax.AddValue(pointOnPath);
            dstSinceLastVertex = 0;
            lastAddedPoint = pointOnPath;
        }
        prevPointOnPath = pointOnPath;
    }
    splitData.anchorVertexMap.Add(splitData.vertices.Count - 1);
}

    return splitData;
}

```

```

public class PathSplitData {
    public List<Vector3> vertices = new List<Vector3>();
    public List<Vector3> tangents = new List<Vector3>();
    public List<float> cumulativeLength = new List<float>();
    public List<int> anchorVertexMap = new List<int>();
}

```

```

        public MinMax3D minMax = new MinMax3D();
    }
}
public class MovementEffectHandler : EffectHandler
{
    public MovementHandler MovementHandler { get; set; }

    public override float CurrentTime {
        get => base.CurrentTime;
        set {
            base.CurrentTime = value;
            MovementHandler.SetCurrentPositionByPercent(CurrentTime / (EndTime -
StartTime));
        }
    }
    //public MovementData Data { get; set; }
    public Coroutine coroutine;
    public override void Remove()
    {
        StopCoroutine(coroutine);
        Destroy(gameObject);
    }
    public override void Play(bool instantPlay)
    {
        coroutine = instantPlay==true? StartCoroutine(InstantPlay()) :
StartCoroutine(PlayByTime());
    }
    public override void Stop()
    {
        StopCoroutine(coroutine);
    }
    public override void Hide()
    {
        MovementHandler.EnableMovementShapeObject(false);
        MovementHandler.EnableSatellite(false);
    }

    public override void Show()
    {
        MovementHandler.EnableMovementShapeObject(true);
        MovementHandler.EnableSatellite(true);
    }
}
public class MovementEffectsManager : MonoBehaviour

```

```

{
    [SerializeField]
    private ModelsCollection satelliteModelsCollections;

    [SerializeField]
    private GameObject centerPoint;
    [SerializeField]
    private GameObject orbitPrefab;
    [SerializeField]
    private GameObject parabolaPrefab;
    [SerializeField]
    private GameObject curvePrefab;
    [SerializeField]
    private GameObject pointPrefab;

    [SerializeField]
    private Vector3 satelliteScale = new Vector3(0.09f, 0.09f, 0.09f);

    private void CreateSatelliteObject(ModelSourceData modelSourceData,
    Action<GameObject> OnCreateGO)
    {
        switch (modelSourceData.Type)
        {
            case ModelSource.File:
                ObjectsLoaderManager.Instance.LoadObject(modelSourceData.Value,
                null, delegate (AssetLoaderContext context)
                {
                    var goFromFile = context.RootGameObject;
                    goFromFile.transform.localScale = satelliteScale;
                    OnCreateGO?.Invoke(goFromFile);
                }, null, null);
                break;
            case ModelSource.Preset:
                var prefab =
satelliteModelsCollections.GetValueByKey(modelSourceData.Value);
                if (prefab == null)
                {
                    throw new Exception($"There is no model {modelSourceData.Value} in
presets");
                }
                var goFromPreset = Instantiate(prefab, Vector3.zero, Quaternion.identity);
                goFromPreset.transform.localScale = satelliteScale;
                OnCreateGO?.Invoke(goFromPreset);
                break;
        }
    }
}

```

```

        default:
            break;
    }
}

private T CreateMovementShape<T,K>(float startTime,float endTime,GameObject
prefab, Transform parent, GameObject satellite, K data) where
T:MovementHandler<K> where K : MovementData
{
    var movementGo = Instantiate(prefab, Vector3.zero, Quaternion.identity,
parent);
    var controller = movementGo.GetComponent<T>();
    controller.Set(startTime,endTime, data, satellite);
    return controller;
}

private void EditMovementShape<T, K>(T handler, float startTime, float
endTime, GameObject satellite, K data) where T : MovementHandler<K> where K :
MovementData
{
    handler.Set(startTime,endTime, data, satellite);
}

public void Create(float startTime,float endTime,MovementEffectData
movementEffect,Action<MovementEffectHandler> OnMovementEffectCreated)
{
    var handlerGo = new GameObject("Movement Effect Handler");
    handlerGo.transform.parent = centerPoint.transform;

    handlerGo.transform.localPosition = Vector3.zero;
    var movementEffectHandler =
handlerGo.AddComponent<MovementEffectHandler>();

    CreateSatelliteObject(movementEffect.MovingObject.ModelSource, (satellite)
=> {

        MovementHandler movementHandler;
        switch (movementEffect.ShapeMovementData)
        {
            case BoomerangMovementData boomerang:
                movementHandler=CreateMovementShape<ParabolicTrajectory,
BoomerangMovementData>(startTime,endTime,
parabolaPrefab,handlerGo.transform,satellite,
(BoomerangMovementData)movementEffect.ShapeMovementData);
                break;
            case CelestialMovementData celestial:

```

```

        movementHandler = CreateMovementShape<CelestialOrbit,
CelestialMovementData>(startTime, endTime, orbitPrefab, handlerGo.transform,
satellite, (CelestialMovementData)movementEffect.ShapeMovementData);
        break;
    case PathMovementData path:
        movementHandler = CreateMovementShape<Curve,
PathMovementData>(startTime, endTime, curvePrefab, handlerGo.transform,
satellite, (PathMovementData)movementEffect.ShapeMovementData);
        break;
    case PointMovementData point:
        movementHandler = CreateMovementShape<PointMovement,
PointMovementData>(startTime, endTime, pointPrefab, handlerGo.transform,
satellite, (PointMovementData)movementEffect.ShapeMovementData);
        break;
    default:
        // Do something else...
        movementHandler = null;
        break;
    }
    handlerGo.layer = centerPoint.layer;
    foreach (var child in handlerGo.GetComponentsInChildren<Transform>())
    {
        child.gameObject.layer = centerPoint.layer;
    }

    movementEffectHandler.MovementHandler = movementHandler;

movementEffectHandler.MovementHandler.ChangeSateliteColor(movementEffect.M
ovingObject.Color);

    handlerGo.transform.localScale = Vector3.one;

movementEffectHandler.MovementHandler.EnableMovementShapeObject(false);
    movementEffectHandler.MovementHandler.EnableSatellite(false);

    OnMovementEffectCreated?.Invoke(movementEffectHandler);
});
}
public void Edit(float startTime,float endTime, MovementEffectHandler
movementEffectHandler, MovementEffectData movementEffect, Action
OnMovementEffectEdited)
{

```

```

        //CreateSatelliteObject(movementEffect.MovingObject.ModelSource, (satellite)
=>
        //{
        //});

        switch (movementEffect.ShapeMovementData)
        {
            case BoomerangMovementData boomerang:

                EditMovementShape((ParabolicTrajectory)movementEffectHandler.MovementHandler,
                startime, endime, movementEffectHandler.MovementHandler.Satellite,
                (BoomerangMovementData)movementEffect.ShapeMovementData);
                break;
            case CelestialMovementData celestial:

                EditMovementShape((CelestialOrbit)movementEffectHandler.MovementHandler,
                startime, endime, movementEffectHandler.MovementHandler.Satellite,
                (CelestialMovementData)movementEffect.ShapeMovementData);
                break;
            case PathMovementData path:
                EditMovementShape((Curve)movementEffectHandler.MovementHandler,
                startime, endime, movementEffectHandler.MovementHandler.Satellite,
                (PathMovementData)movementEffect.ShapeMovementData);
                break;
            case PointMovementData point:

                EditMovementShape((PointMovement)movementEffectHandler.MovementHandler,
                startime, endime, movementEffectHandler.MovementHandler.Satellite,
                (PointMovementData)movementEffect.ShapeMovementData);
                break;
            default:
                // Do something else...
                break;
        }

        movementEffectHandler.MovementHandler.ChangeSateliteColor(movementEffect.M
        ovingObject.Color);
        OnMovementEffectEdited?.Invoke();
    }
}
public abstract class MovementHandler : MonoBehaviour
{
    private const float Pi2=Mathf.PI*2;
    public float StartTime { get; set; }

```

```

public float EndTime { get; set; }
public int NumPoints { get {
    if ((Length * 360 / Pi2) > (TimeLength / 5 * 360))
    {
        return (int)TimeLength / 5 * 360;
    }
    else
    {
        return (int)(Length * 360 / Pi2);
    }
} }
public abstract float Length { get; }
public float TimeLength { get { return EndTime - StartTime; } }
public GameObject Satellite { get; set; }

public void EnableMovementShapeObject(bool enable)
{
    gameObject.SetActive(enable);
}

public void EnableSatellite(bool enable)
{
    if (Satellite != null)
    {
        Satellite.SetActive(enable);
    }
}

public void RemoveSatellite()
{
    Destroy(Satellite);
}

public void ChangeSateliteColor(Color32? color)
{
    if (Satellite == null || color == null)
        return;
    var colorChangers = Satellite.GetComponentsInChildren<ColorChanger>();
    if (colorChangers.Length == 0)
    {
        colorChangers = new ColorChanger[1];
        colorChangers[0] = Satellite.AddComponent<MaterialsColorChanger>();
    }
    foreach (var colorChanger in colorChangers)
    {
        colorChanger.SetColor(color.Value);
    }
}

```

```

    }
}

public abstract Vector3[] GetPoints();
public abstract void SetCurrentPositionByPercent(float percent);
}
public abstract class MovementHandler<T> : MovementHandler where T :
MovementData
{
    public void Set(float startTime, float endTime, T data, GameObject satellite)
    {
        StartTime = startTime;
        EndTime = endTime;
        Set(data, satellite);
    }
    protected virtual void Set(T data, GameObject satellite)
    {
        Satellite = satellite;
        satellite.transform.parent = transform;
        satellite.transform.localPosition = Vector3.zero;
    }
}
public class MovementDataConverter : JsonConverter
{
    public override bool CanConvert(Type objectType)
    {
        return objectType.BaseType == typeof(MovementData);
    }

    public override object ReadJson(JsonReader reader, Type objectType, object
existingValue, JsonSerializer serializer)
    {
        if (reader.TokenType == JsonToken.Null) return null;

        JObject jsonObject = JObject.Load(reader);
        switch (jsonObject.ToObject<MovementData>().Type)
        {
            case MovementType.Boomerang:
                return jsonObject.ToObject<BoomerangMovementData>();
            case MovementType.Celestial:
                return jsonObject.ToObject<CelestialMovementData>();
            case MovementType.Path:
                return jsonObject.ToObject<PathMovementData>();
            case MovementType.Point:

```

```

        return jsonObject.ToObject<PointMovementData>();
    default:
        throw new Exception("No such movement type");
    }
}

public override void WriteJson(JsonWriter writer, object value, JsonSerializer
serializer)
{
    throw new NotImplementedException();
}
}

public class CelestialRotation : MonoBehaviour, ISimulation {
    const double Deg2Rad = Math.PI/180.0;

    #region Fields
    //input fields
    [SerializeField] float _rightAscension = 0.0f; //[deg]
    public float rightAscension {
        get {return _rightAscension;}
        set {
            _rightAscension = value;
            axis = Kepler.ComputeAxis(_rightAscension, _declination);
        }
    }

    [SerializeField] float _declination = 0.0f; //[deg]
    public float declination {
        get {return _declination;}
        set {
            _declination = value;
            axis = Kepler.ComputeAxis(_rightAscension, _declination);
        }
    }

    //control fields
    public bool simulate = true;

    public float meanAngle = 0.0f; //[deg]

    //time fields
    [SerializeField] double _period = 10.0; //[s]
    public double period {
        get {return _period;}
    }
}

```

```

        set {
            _period = value;
            rate = Kepler.ComputeRate(_period, -Math.PI, Math.PI);
        }
    }

    public double timeScale = 1.0;

    public double startEpoch = 0.0; //[s]
    #endregion

    #region Properties
    //static properties
    public Quaternion axis {get; private set;}
    public double rate {get; private set;} //[rad/s]

    //dynamic properties
    public double angle {get; private set;} //[rad]
    public Quaternion rotation {get; private set;}
    #endregion

    #region Messages
    void Reset() {
        _rightAscension = 0.0f;
        _declination = 0.0f;

        simulate = true;
        meanAngle = 0.0f;

        _period = 10.0;
        timeScale = 1.0;
        startEpoch = 0.0;
    }

    void Start() {
        ResetSimulation();
    }

    void OnEnable() {
        ComputeStaticProperties();
        ComputeDynamicProperties(angle);
        transform.localRotation = rotation;
    }

```

```

void Update() {
    if (simulate) UpdateSimulation();
}

void OnValidate() {
    Start();
    OnEnable();
}
#endregion

#region Computation
/// <summary>Computes static properties for rotational axis and
speed</summary>
public void ComputeStaticProperties() {
    axis = Kepler.ComputeAxis(_rightAscension, _declination);
    rate = Kepler.ComputeRate(_period, -Math.PI, Math.PI);
}

/// <summary>Compute dynamic properties that change over the
angle</summary>
/// <param name="M">The angle to evaluate properties at</param>
public void ComputeDynamicProperties(double angle) {
    rotation = Kepler.ComputeRotation(axis, angle/Deg2Rad);
}
#endregion

#region Simulation
public void StartSimulation() {simulate = true;}
public void StopSimulation() {simulate = false;}

public void ResetSimulation() {
    angle = Kepler.WrapAngle(meanAngle*Deg2Rad + startEpoch *
rate, 0, 2*Math.PI);
}

public void UpdateSimulation() {
    angle = Kepler.WrapAngle(angle + Time.deltaTime * rate *
timeScale, 0, 2*Math.PI);
    ComputeDynamicProperties(angle);
    transform.localRotation = rotation;
}
#endregion
}
[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]

```

```

public class CelestialRings : MonoBehaviour {
    [SerializeField] float innerRadius = 1f;
    [SerializeField] float outerRadius = 1.5f;
    [Range(3,50), SerializeField] int segments = 10;

    void Reset() {
        innerRadius = 1f;
        outerRadius = 1.5f;
        segments = 10;
    }

    void OnValidate() {
        if (innerRadius < 0) innerRadius = 0f;
        if (outerRadius < 0) outerRadius = 0f;
        Generate();
    }

    public void Generate() {
        MeshFilter filter = GetComponent<MeshFilter>();
        DestroyImmediate(filter.sharedMesh);
        filter.sharedMesh = GenerateRingMesh(innerRadius, outerRadius,
segments);
    }

    public static Mesh GenerateRingMesh(float ri, float ro, int segments) {
        if (segments <= 0)
            throw new
System.ArgumentOutOfRangeException("Number of segments cannot be less than or
equal to 0");

        Mesh mesh = new Mesh();
        mesh.name = "Ring Mesh";

        Vector3[] vertices = new Vector3[segments*4];
        Vector2[] uvs = new Vector2[segments*4];
        int[] triangles = new int[segments*6];

        /*
        *      n+1-----n
        *      |\      |
        *  |\      |
        *   |\      |
        *   | \      |
        *   |  \      |
        *   |   \      |
        */

```

```

*      |   \ |
*      |   \ |
*      n+3-----n+2
*
*  n  = ro < i      ->   uv(1,0)
*  n+1 = ro < i+1  ->   uv(1,1)
*  n+2 = ri < i     ->   uv(0,0)
*  n+3 = ri < i+1  ->   uv(0,1)
*/

for (int i = 0; i < segments; i++) {
    float angle = i*2f*Mathf.PI/segments;
    float angle2 = (i+1)*2f*Mathf.PI/segments;

    vertices[4*i+0] = new Vector3(ro*Mathf.Cos(angle), 0f,
ro*Mathf.Sin(angle));
    vertices[4*i+1] = new Vector3(ro*Mathf.Cos(angle2), 0f,
ro*Mathf.Sin(angle2));
    vertices[4*i+2] = new Vector3(ri*Mathf.Cos(angle), 0f,
ri*Mathf.Sin(angle));
    vertices[4*i+3] = new Vector3(ri*Mathf.Cos(angle2), 0f,
ri*Mathf.Sin(angle2));

    uvs[4*i+0] = new Vector2(1f,0f);
    uvs[4*i+1] = new Vector2(1f,1f);
    uvs[4*i+2] = new Vector2(0f,0f);
    uvs[4*i+3] = new Vector2(0f,1f);

    //tri 1
    triangles[6*i+0] = 4*i+2;
    triangles[6*i+1] = 4*i+1;
    triangles[6*i+2] = 4*i+0;
    //tri 2
    triangles[6*i+3] = 4*i+1;
    triangles[6*i+4] = 4*i+2;
    triangles[6*i+5] = 4*i+3;
}

mesh.vertices = vertices;
mesh.uv = uvs;
mesh.triangles = triangles;

;
mesh.RecalculateNormals();

```

```

        mesh.RecalculateBounds();

        return mesh;
    }
}

public static class Kepler {
    static int maxIterations = 200;
    static double tolerance = 0.0001;

    #region Computation
    /// <summary>Wraps an angle value between lower and
upper.</summary>
    /// <returns>Wrapped angle</returns>
    /// <param name="angle">Angle</param>
    /// <param name="lower">Lower</param>
    /// <param name="upper">Upper</param>
    public static double WrapAngle(double angle, double lower, double
upper) {
        if (angle > lower && angle < upper)
            return angle;
        else if (angle < lower)
            return upper - (lower - angle) % (upper - lower);
        else
            return lower + (angle - lower) % (upper - lower);
    }

    /// <summary>Wraps an angle value between lower and
upper.</summary>
    /// <returns>Wrapped angle</returns>
    /// <param name="angle">Angle</param>
    /// <param name="lower">Lower</param>
    /// <param name="upper">Upper</param>
    public static float WrapAngle(float angle, float lower, float upper) {
        if (angle > lower && angle < upper)
            return angle;
        else if (angle < lower)
            return upper - (lower - angle) % (upper - lower);
        else
            return lower + (angle - lower) % (upper - lower);
    }

    /// <summary>Compute the rotation defining the orbital plane coordinate
system</summary>

```

```

        /// <param name="o">longitude of the ascending node
[degrees]</param>
        /// <param name="w">argument of the periapsis [degrees]</param>
        /// <param name="i">inclination [degrees]</param>
        /// <returns>Orbital Plane Orientation</returns>
        public static Quaternion ComputeOrientation(float o, float w, float i) {
            return Quaternion.Euler(i, -w, 0f)*Quaternion.Euler(0f, -o, 0f);
        }

        /// <summary>Compute the semi-latus rectum of the conic
section</summary>
        /// <param name="a">periapsis [m]</param>
        /// <param name="e">eccentricity [1]</param>
        /// <returns>Semi-Latus Rectum [m]</returns>
        public static double ComputeSemiLatusRectum(double p, double e) {
            if (e == 0.0) {                //circular
                return p;
            } else if (e == 1.0) {        //parabolic
                return p*2.0;
            } else if (e < 1.0) { //elliptical
                return p*(1.0-e*e)/(1.0-e);
            } else {                    //hyperbolic
                return p*(e*e-1.0)/(e-1.0);
            }
        }

        /// <summary>
        /// Computes the semi major axis of the conic section
        /// </summary>
        /// <param name="p">periapsis [m]</param>
        /// <param name="e">eccentricity [1]</param>
        /// <returns>Semi-major axis [m]</returns>
        public static double ComputeSemiMajorAxis(double p, double e) {
            if (e == 0.0) {                //circular
                return p;
            } else if (e == 1.0) {        //parabolic
                return double.PositiveInfinity;
            } else if (e < 1.0) { //elliptical
                return p/(1-e);
            } else {                    //hyperbolic
                return -p/(e-1);
            }
        }
}

```

```

/// <summary>Compute the rate of orbit</summary>
/// <param name="T">Period [seconds/orbit]</param>
/// <param name="from">From Angle [rad]</param>
/// <param name="to">To Angle [rad]</param>
/// <returns>Rate [rad/second]</returns>
public static double ComputeRate(double T, double from, double to) {
    if (T == 0) {
        Debug.LogWarning("Period cannot be 0");
        return 0;
    }
    return Math.Abs((from-to)/T);
}

/// <summary>Compute the eccentric anomaly of the conic
section</summary>
/// <param name="M">anomaly [rad]</param>
/// <param name="e">eccentricity [1]</param>
/// <returns>Eccentric Anomaly [rad]</returns>
public static double ComputeEccentricAnomaly(double M, double e) {
    if (e < 0.0)
        throw new ArgumentOutOfRangeException("eccentricity",
"cannot be negative");

    if (e == 0.0) { //circular
        return M;
    } else if (e == 1.0) { //parabolic
        return NewtonParabolic(M, e);
    } else if (e < 1.0) { //elliptical
        return NewtonElliptical(M, e);
    } else { //hyperbolic
        return NewtonHyperbolic(M, e);
    }
}

/// <summary>Compute the true anomaly of the conic section.
/// This is the actual angle between the periapsis and the position of
/// the orbital body.
/// </summary>
/// <param name="E">eccentric anomaly [rad]</param>
/// <param name="e">eccentricity [1]</param>
/// <returns>True Anomaly [rad]</returns>
public static double ComputeTrueAnomaly(double E, double e) {
    if (e < 0.0)
        throw new ArgumentOutOfRangeException("eccentricity",
"cannot be negative");

```

```

        if (e == 0.0) {                                //circular
            return E;
        } else if (e == 1.0) {                        //parabolic
            return 2.0*Math.Atan(E);
        } else if (e < 1.0) { //elliptical
            return 2.0*Math.Atan2(Math.Sqrt(1.0+e)*Math.Sin(E/2.0),
Math.Sqrt(1.0-e)*Math.Cos(E/2.0));
        } else {                                     //hyperbolic
            return 2.0*Math.Atan2(Math.Sqrt(e+1.0)*Math.Sinh(E/2.0),
Math.Sqrt(e-1.0)*Math.Cosh(E/2.0));
        }
    }

    /// <summary>Compute the radius at the given anomaly</summary>
    /// <param name="l">semi latus rectum [m]</param>
    /// <param name="e">eccentricity [1]</param>
    /// <param name="v">>true anomaly [rad]</param>
    /// <returns>Radius</returns>
    public static double ComputeRadius(double l, double e, double v) {
        return l/(1.0 + e*Math.Cos(v));
    }

    /// <summary>Compute the position of the body within the orbital
plane</summary>
    /// <param name="r">radius [m]</param>
    /// <param name="v">>true anomaly [rad]</param>
    /// <returns>Position Vector</returns>
    public static Vector3 ComputePosition(double r, double v) {
        return new Vector3((float)(r*Math.Cos(v)),
            0f,
            (float)(r*Math.Sin(v)));
    }

    /// <summary>Compute the velocity of the body within the orbital
plane</summary>
    /// <param name="a">semi-major axis [m]</param>
    /// <param name="r">radius [m]</param>
    /// <param name="n">rate [rad/second]</param>
    /// <param name="E">eccentric anomaly [rad]</param>
    /// <param name="e">eccentricity [1]</param>
    /// <returns>Velocity Vector</returns>
    // TODO: support hyperbolic and parabolic orbits

```

```

    public static Vector3 ComputeVelocity(double a, double r, double n,
double E, double v, double e) {
        if (e == 0.0) { //circular velocity
            Vector3 vel = new Vector3((float)(-Math.Sin(E)),
                0f,
                (float)(Math.Cos(E)));
            return (float)(a * n) * vel;

        } else if (e == 1.0) { //parabolic velocity
            Vector3 vel = new Vector3((float)(-
Math.Sin(v)/(Math.Cos(v)+1)),
                0f,
                1f);
            return (float)Math.Sqrt(n*n*a*a*a*(2/r)) * vel.normalized;

        } else if (e < 1.0) { //elliptical velocity
            Vector3 vel = new Vector3((float)(-Math.Sin(E)),
                0f,
                (float)(Math.Sqrt(1-e*e)*Math.Cos(E)));
            return (float)((a*a*n)/r) * vel;

        } else { //hyperbolic velocity
            Vector3 vel = new Vector3((float)(-Math.Sinh(E)),
                0,
                (float)(Math.Sqrt(e*e-1)*Math.Cosh(E)));
            return (float)((a*a*n)/r) * vel;
        }
    }
}

```

ДОДАТОК Б
Слайди презентації

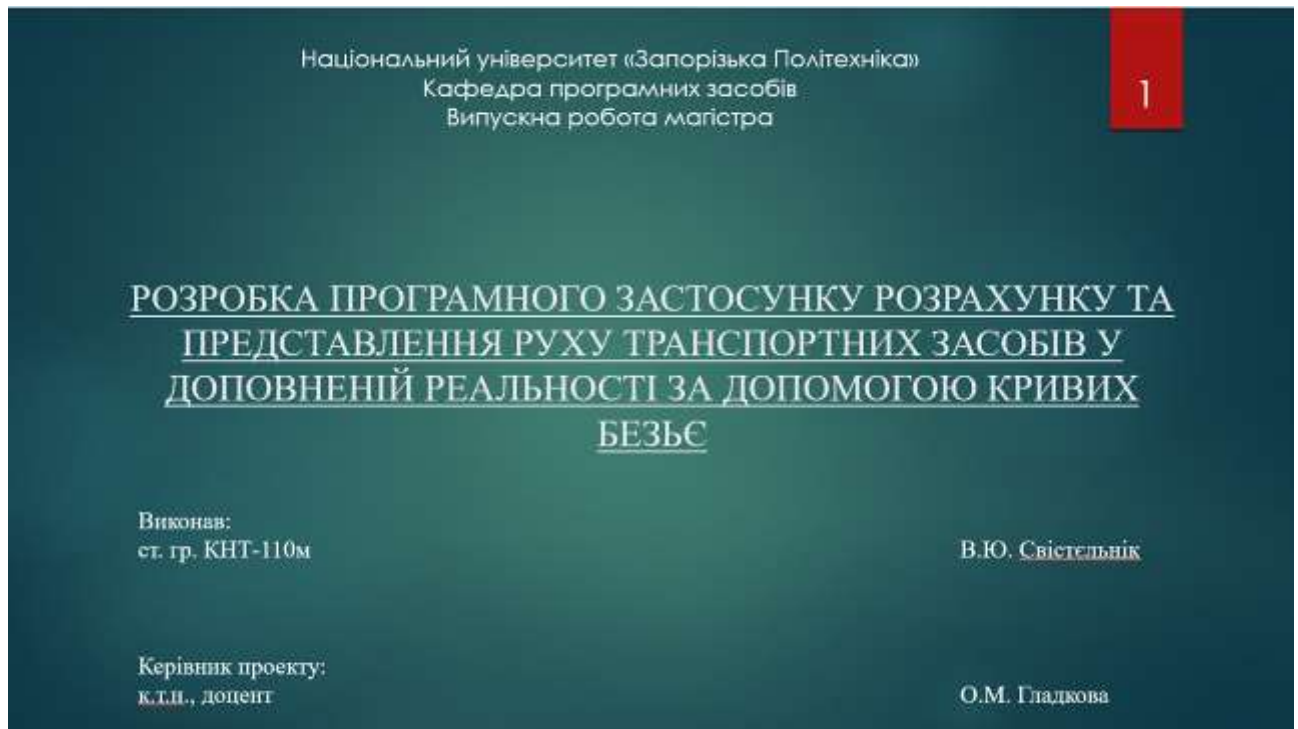


Рисунок Б.1 – Слайд №1

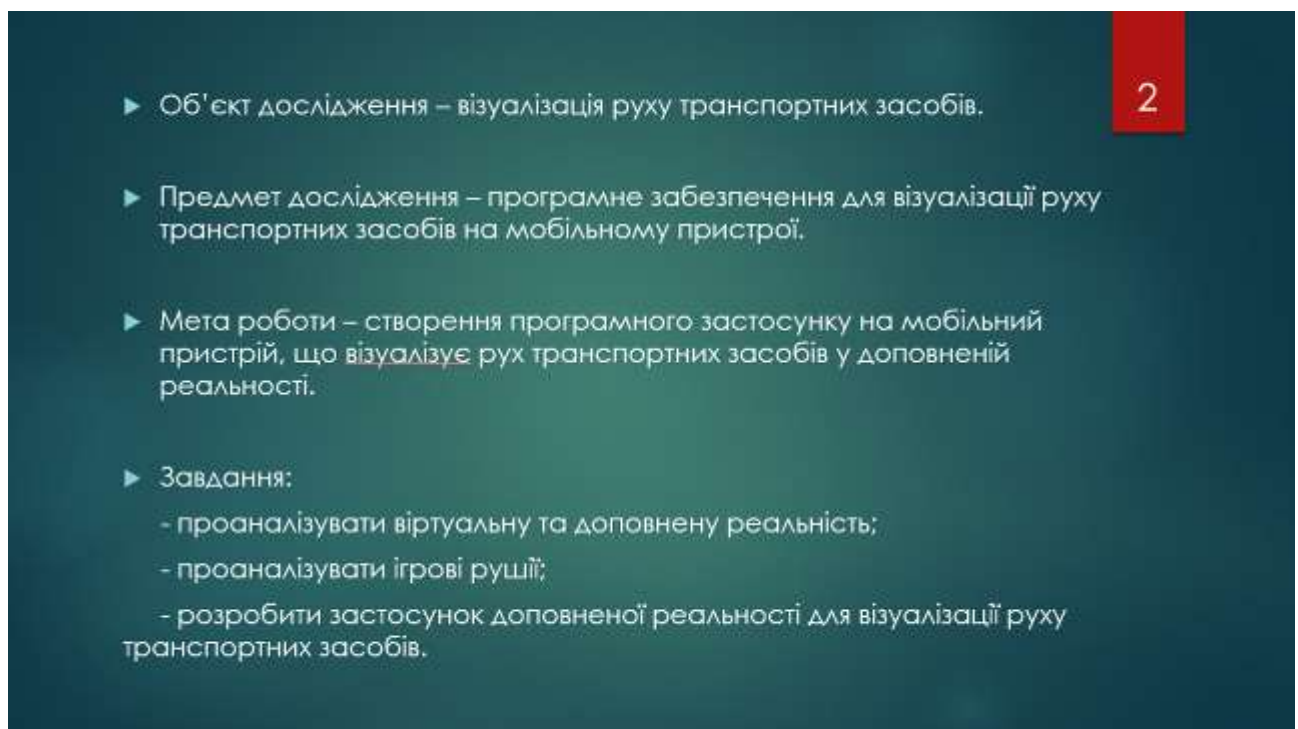


Рисунок Б.2 – Слайд №2



Рисунок Б.3 – Слайд №3

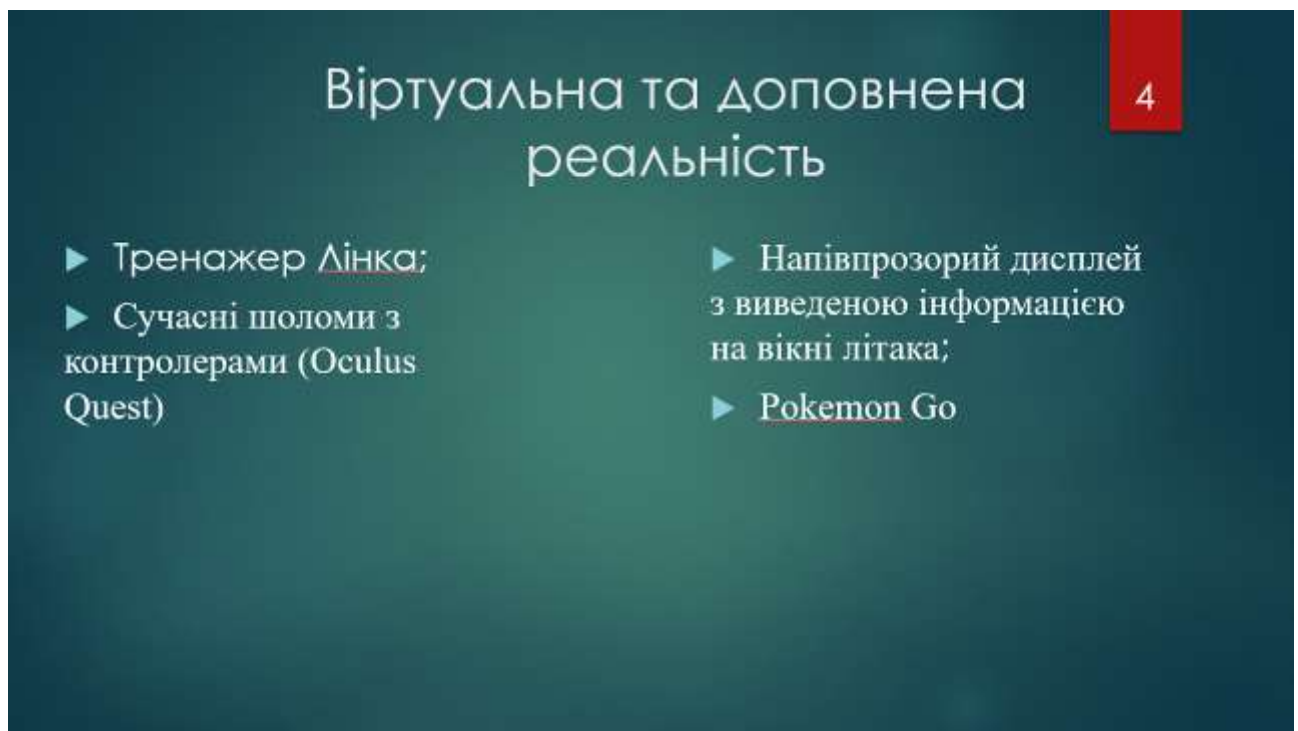


Рисунок Б.4 – Слайд №4

Ігрові рушії

5

Ігровий рушій	Unreal Engine	Unity	GameMaker
Рівень складності	5/5	3/5	1/5
Доступність	Вимагає підписки	Є безкоштовна версія	Якщо не експортувати платформи, безкоштовна
Кросплатформенність	Так	Так	Так
Легка відладка додатку	Так	Так	Ні
Інструментарій	5/5	4/5	2/5

Рисунок Б.5 – Слайд №5

Середовища розробки

6

Інструментарій	Microsoft Visual Studio	Visual Studio Code	Rider
Підсвічування коду	+	+	+
Автодоповнення	+	+	+
Горячі клавіші	+	+	+
Компілятор	+	-	+
Робота з VCS	-	+	+
Unit-тести	+	-	+
Робота з базами даних	+	-	+
Плагіни	+	+	+

Рисунок Б.6 – Слайд №6

Структура програмного продукту 7



Рисунок Б.7 – Слайд №7

Основний алгоритм роботи 8



Рисунок Б.8 – Слайд №8



Рисунок Б.9 – Слайд №9



Рисунок Б.10 – Слайд №10

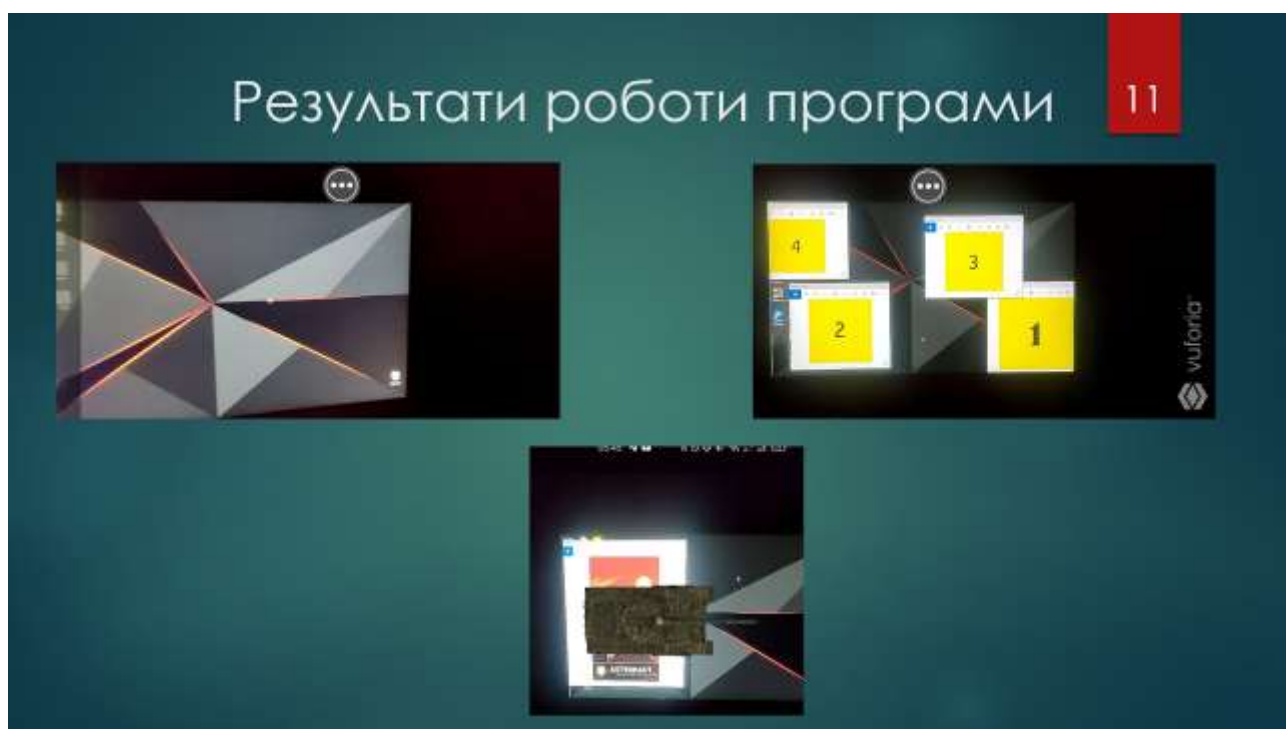


Рисунок Б.11 – Слайд №11



Рисунок Б.12 – Слайд №12

Висновки

13

За результатами випускної роботи магістра:

- ▶ проаналізовано віртуальну та доповнену реальності;
- ▶ проаналізовано ігрові рушії;
- ▶ розроблено застосунок доповненої реальності для візуалізації руху транспортних засобів.

Наукова новизна отриманих результатів. Отримано подальший розвиток алгоритму побудови кривих Без'є для прокладання оптимальної траєкторії руху транспортного засобу у доповненій реальності.

Практична значимість результатів. Розроблено програмний продукт для візуалізації переміщення об'єктів з одного місця в інше за траєкторією кривої Без'є в обхід встановлених позицій.

Рисунок Б.13 – Слайд №13