НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
_____ / Голуб Б.Л. /
(підпис) (ПІБ)
"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему
Програмне забезпечення для 3D малювання у віртуальній реальності
Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н, доцент _____ Вайганг Г.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
асистент _____ Баранова Т.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Консультант бакалаврської кваліфікаційної роботи
к.т.н, доцент _____ Даков С.Ю.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Кузьменко Ярослав Олександрович
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
_____ / Голуб Б.Л. /
(підпис) (ПІБ)
"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту

Кузьменку Ярославу Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення для 3D малювання у віртуальній реальності» затверджена наказом ректора НУБіП України від 16.12.2024 р. №2248 "С"
Термін подання завершеної роботи на кафедру 2025.06.____
(рік, місяць, число)
Вихідні дані до бакалаврської кваліфікаційної роботи
_____
Перелік питань, які розглядаються:

1. Системний аналіз предметної області
2. Реалізація ключових ігрових механік
3. Оптимізація та тестування продуктивності
4. Висновки

Дата видачі завдання "___16___"___12_____2024 р.
Керівник бакалаврської кваліфікаційної роботи
асистент _____ Баранова Т.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)
Консультант бакалаврської кваліфікаційної роботи
к.т.н, доцент _____ Даков С.Ю.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання
_____ Кузьменко Ярослав Олександрович
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

VArt - назва розроблюваного програмного забезпечення

VR — віртуальна реальність (Virtual Reality)

JSON — JavaScript Object Notation

FPS — кадрів за секунду (Frames Per Second)

UML — єдина мова моделювання (Unified Modeling Language)

WMR — Windows Mixed Reality

API — програмний інтерфейс додатків (Application Programming Interface)

SDK — комплект засобів розробки програмного забезпечення (Software Development Kit)

Unity — двигун, що спрощує розробку проєктів в яких використовується 2D або 3D графіка

XRIT — бібліотека для зв’язування управління фізичних контролерів з Unity (XR Interaction Toolkit)

OpenXR — відкритий стандарт для розширеної та віртуальної реальності (OpenXR)

GPU — графічний процесор (Graphics Processing Unit)

UI — користувацький інтерфейс (User Interface)

UX — користувацький досвід (User Experience)

ВСТУП

Актуальність завдання

У сучасному світі VR стає дедалі більш популярним інструментом не лише в розважальній індустрії, а й у сфері освіти, медицини, архітектури, дизайну та мистецтва. Одним із перспективних напрямів застосування VR є цифрова творчість, зокрема 3D-малювання у віртуальному просторі. На відміну від традиційних методів створення тривимірної графіки за допомогою миші та клавіатури, VR дозволяє художнику працювати в тривимірному середовищі з використанням природних жестів і рухів, що робить процес інтуїтивно зрозумілим і значно ближчим до реального малювання.

Попри наявність деяких відомих продуктів на ринку, таких як Tilt Brush, Gravity Sketch чи Adobe Medium, значна частина з них має обмеження — закритий вихідний код, обмежений функціонал, або ж орієнтацію лише на конкретні пристрої та платформи. Крім того, з розвитком апаратного забезпечення відкриваються нові можливості для створення більш оптимізованих, доступних і гнучких рішень.

Розробка власного програмного забезпечення для 3D-малювання у VR дозволяє не лише поглибити розуміння принципів побудови інтерфейсів віртуальної реальності, але й створити інноваційний інструмент, що відповідає сучасним вимогам креативних індустрій. Така система може бути адаптована під потреби конкретного користувача та доповнена новими функціями.

Отже, розробка VR-додатку для 3D-малювання є актуальним завданням, яке поєднує інженерну, художню та дослідницьку складові й сприяє подальшому розвитку інтерактивних цифрових середовищ.

Мета розробки

Метою даної дипломної роботи є створення програмного забезпечення для 3D-малювання у середовищі віртуальної реальності, яке забезпечує зручний, інтуїтивно зрозумілий та функціонально гнучкий інструмент для художньої та дизайнерської діяльності.

Основна увага приділяється реалізації можливостей просторового малювання в режимі реального часу, оптимізації взаємодії з користувачем за допомогою VR-контролерів.

Досягнення поставленої мети передбачає:

- аналіз існуючих рішень у сфері 3D-малювання у VR;
- розробку архітектури VR-додатку на основі сучасного рушія Unity;
- реалізацію основного функціоналу для створення, редагування та збереження тривимірних малюнків;
- створення інтуїтивного інтерфейсу для взаємодії користувача з віртуальним середовищем;
- тестування додатку з урахуванням зручності, стабільності та якості візуалізації.

Таким чином, кінцевим результатом розробки є прототип VR-додатку, який може використовуватись як самостійний інструмент для творчості або як основа для подальшого вдосконалення.

Методи та технології

Для реалізації програмного забезпечення 3D-малювання у віртуальній реальності в рамках даної роботи було використано низку сучасних методів і технологій, що забезпечують ефективну розробку, зручність користування та високу продуктивність системи.

1. Методологія розробки:

Об'єктно-орієнтоване програмування (ООП) — як основний підхід до побудови програмної архітектури, що дозволяє реалізувати модульність, повторне використання коду та зручну масштабованість;

Ітеративна модель розробки — з поетапним впровадженням і тестуванням функціоналу, що дозволило поступово удосконалювати застосунок відповідно до поставлених завдань.

2. Технології програмування:

Unity 3D (версія 2017 або вище) — кросплатформовий рушій для розробки інтерактивних 3D-додатків з підтримкою VR;

C# — мова програмування для створення логіки взаємодії в Unity;

XRIT — набір інструментів для розробки VR-інтерфейсів з використанням контролерів віртуальної реальності;

3. Інструменти для моделювання та тестування:

VR-гарнітура WMR — використана як основна платформа для тестування та взаємодії з додатком;

4. Методи обробки введення та рендерингу:

Використання Line Renderer або Mesh Generation для створення об’ємних ліній малювання в реальному часі;

Таким чином, обрані методи та технології дозволяють забезпечити якісну реалізацію функціоналу VR-додатку, гнучкість розширення і підтримку сучасних апаратних засобів.

Апробація

Тези - Програмне забезпечення для 3D малювання у віртуальній реальності.

"Теоретичні та прикладні аспекти розробки комп’ютерних систем". Науково-практична конференція студентів і аспірантів.

Структура записки

Кількість сторінок - 101

Кількість використаних джерел - 8

Кількість додатків - 9

Короткий опис розділів:

1 Вступ

Визначає актуальність теми, формулює мету і завдання дослідження, описує об’єкт і предмет роботи та очікувані результати.

2 Огляд існуючих рішень у сфері VR-малювання

Аналізує популярні інструменти (Tilt Brush, Medium тощо), порівнює їх функціональні можливості й UX-підходи, обґрунтовує вибір платформи Unity і XRIT.

3 Проєктування архітектури системи

Описує модульну структуру додатку, наведено різні UML-діаграми для ключових сценаріїв взаємодії, розподіл відповідальностей між компонентами.

4 Алгоритмізація та програмування програмних модулів

Деталізує реалізацію:

- Обробка вводів — перетворення сигналів контролерів у команди малювання.

- 3D-моделі контролерів — створення віртуальної репрезентації пристроїв користувача.
- Інтерфейс — menus і панелі налаштувань з raycast-керуванням.
- Візуальні підказки — підсвічування та графічні підказки для полегшення роботи.
- Інструменти — модулі Brush, Eraser і Selector із параметризацією кривих.
- Збереження даних — JSON-серіалізація BrushStroke-об’єктів і механізм автозбереження.
- Реалізація в Unity — інтеграція усіх компонентів у середовище Unity із оптимізацією під VR.

5 Тестування системи

Описує методику інтеграційного тестування, процедуру відкритого тестування з користувачами, результати продуктивності (FPS, стійкість) і відповідність технічним вимогам.

6 Висновки

Узагальнює виконані завдання, оцінює досягнення мети роботи, підкреслює ефективність обраної архітектури та технологій, дає рекомендації щодо подальшого використання та розвитку прототипу.

7 Список використаних джерел

Перелік нормативних документів, наукових статей, офіційних посібників і електронних ресурсів, які були використані при написанні та реалізації проєкту.

8 Додатки

Містять повний вихідний код, схему управління для контролерів і README файл з описом програми для кінцевого користувача.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область цієї роботи охоплює перетин двох сучасних галузей: віртуальної реальності та тривимірної графіки, зокрема — 3D-малювання в інтерактивному середовищі.

3D-малювання — це процес створення тривимірних візуальних об’єктів, які можуть мати об’єм, форму та положення у просторі. Традиційно для цього використовуються професійні програми моделювання (Blender, Autodesk Maya, ZBrush тощо), що передбачають використання миші, клавіатури або графічного планшета. Ці інструменти хоч і потужні, однак вимагають складного навчання та не завжди забезпечують природну взаємодію з простором.

Віртуальна реальність — це технологія занурення користувача у штучне тривимірне середовище з повним ефектом присутності. У VR-контенті користувач має можливість взаємодіяти з об’єктами та середовищем за допомогою жестів, рухів голови й контролерів. У випадку 3D-малювання VR дає змогу "малювати в повітрі", створюючи об'єкти інтуїтивно — так як художник малює у просторі пензлем чи рукою.

Основні характеристики предметної області:

- Просторова орієнтація. Художник може вільно переміщуватись у віртуальному середовищі, змінювати ракурси і напрямки малювання;
- Інтерактивність. Кожна дія користувача (натискання кнопки, рух) миттєво викликає відповідну реакцію віртуального середовища;
- Гнучкість інтерфейсу. Потрібен спеціально адаптований користувацький інтерфейс, який буде зручним в умовах VR;
- Креативна спрямованість. Система орієнтована на дизайнерів, художників та ентузіастів цифрової творчості.

Таким чином, предметна область включає в себе не тільки технічні аспекти створення віртуального середовища, а й питання взаємодії з користувачем, ергономіку, UX-дизайн, продуктивність графіки, підтримку просторового введення та збереження результатів творчості.

1.2 Аналіз вимог до програмної системи

Проєктування програмного забезпечення для 3D-малювання у віртуальній реальності потребує ретельного аналізу функціональних та нефункціональних вимог. Це дозволяє визначити ключові цілі, які повинна реалізовувати система, а також встановити технічні обмеження й умови експлуатації.

1.2.1 Функціональні вимоги. Визначені наступні функціональні вимоги до програми:

- Малювання у 3D-просторі

Користувач повинен мати можливість створювати тривимірні криві, використовуючи VR-контролери.

- Навігація у віртуальному середовищі

Програма повинна дозволяти переміщення та огляд з різних ракурсів.

- Інструменти малювання

Необхідно реалізувати базові інструменти: пензель(створення кривих з параметрами: кольору, товщини лінії, форми тощо), гумка(видалення намальованих кривих), селектор(зміна параметрів намальованих кривих).

- Збереження результатів

Система повинна підтримувати збереження створених 3D-малюнків у файл.

- Завантаження сцен

Можливість відкривати раніше збережені роботи для редагування або перегляду.

- Інтерфейс

Інтерфейс користувача має бути адаптованим до VR, з підтримкою контролерів і просторових меню.

- Сумісність із VR-пристроями

Підтримка гарнітури віртуальної реальності WMR.

1.2.2 Нефункціональні вимоги. Визначені наступні нефункціональні вимоги до програми:

- Продуктивність

Система повинна працювати з високою частотою кадрів (не менше 40 FPS), щоб забезпечити плавність і комфорт у VR.

- Надійність і стабільність

Додаток не повинен аварійно завершуватись при звичайному використанні або при помилках введення користувача.

- Можливість розширення

Архітектура має передбачати додавання нових функцій без необхідності переписувати вже реалізовану частину системи.

- Юзабіліті

Інтерфейс повинен бути зрозумілим навіть для користувачів без досвіду роботи з професійними графічними програмами.

1.2.3 Обмеження.

Необхідність наявності сумісної VR-гарнітури та контролерів.

Рекомендована конфігурація комп’ютера: процесор не нижче Intel Core i5, не менше 8 ГБ оперативної пам’яті, відеокарта з підтримкою VR (NVIDIA GTX 1060 / AMD RX 580 або вище).

1.3 Моделювання предметної області

Для побудови програмного забезпечення важливо сформувати уявлення про основні поняття предметної області. Це дозволяє краще зрозуміти задачу.

Основні поняття предметної області

- Художник — людина, яка має набір інструментів для малювання у 3D просторі.
- Сцена — простір, у якому відбувається малювання.
- Інструмент малювання(пензель) — об'єкт, який дозволяє малювати в 3D просторі.
- Об’єкт малювання(мальована крива) — результат використання пензля художником.
- Інструмент виділення(селектор) — об'єкт, який дозволяє виділяти криву та змінювати те як вона виглядає.
- Інструмент видалення(гумка) — об'єкт, який дозволяє стирати намальовані криві.

1.3.1 Визначення прецедентів. На етапі моделювання функціональної поведінки системи було побудовано діаграму прецедентів(рис. 1), яка описує взаємодію людини(актора) з предметною областю. Ця діаграма дозволяє візуально представити дії які може виконувати актор, а також визначити ролі акторів, які виконують ці дії.

Діаграма прецедентів в даному випадку, показує які дії може виконувати художник, в предметній області 3D малювання у віртуальній реальності.

Основні прецеденти, зазначені на діаграмі:

- Вибір інструмента — художник обирає пензель, гумку або селектор для малювання, видалення чи редагування намальованих кривих.

- Зміна параметрів інструментів — можливість змінювати параметри інструментів(розмір, колір, форма). Параметри залежать від вибраного інструмента.
- Малювання кривої — основна дія, під час якої створюється крива за траєкторією руху руки художника.
- Видалення кривої — інструмент гумки дозволяє стерти криві.
- Виділення/редагування кривої — художник виділяє криву для подальших змін положення, обертання, масштабу або інших параметрів (колір, товщина, тощо).
- Зберігати/відновлювати намальоване — збереження та відновлення створених малюнків.
- Переміщення — можливість художнику переміщуватися у просторі.
- Створення фотографій малюнків — для подальшого використання в якості референсів.

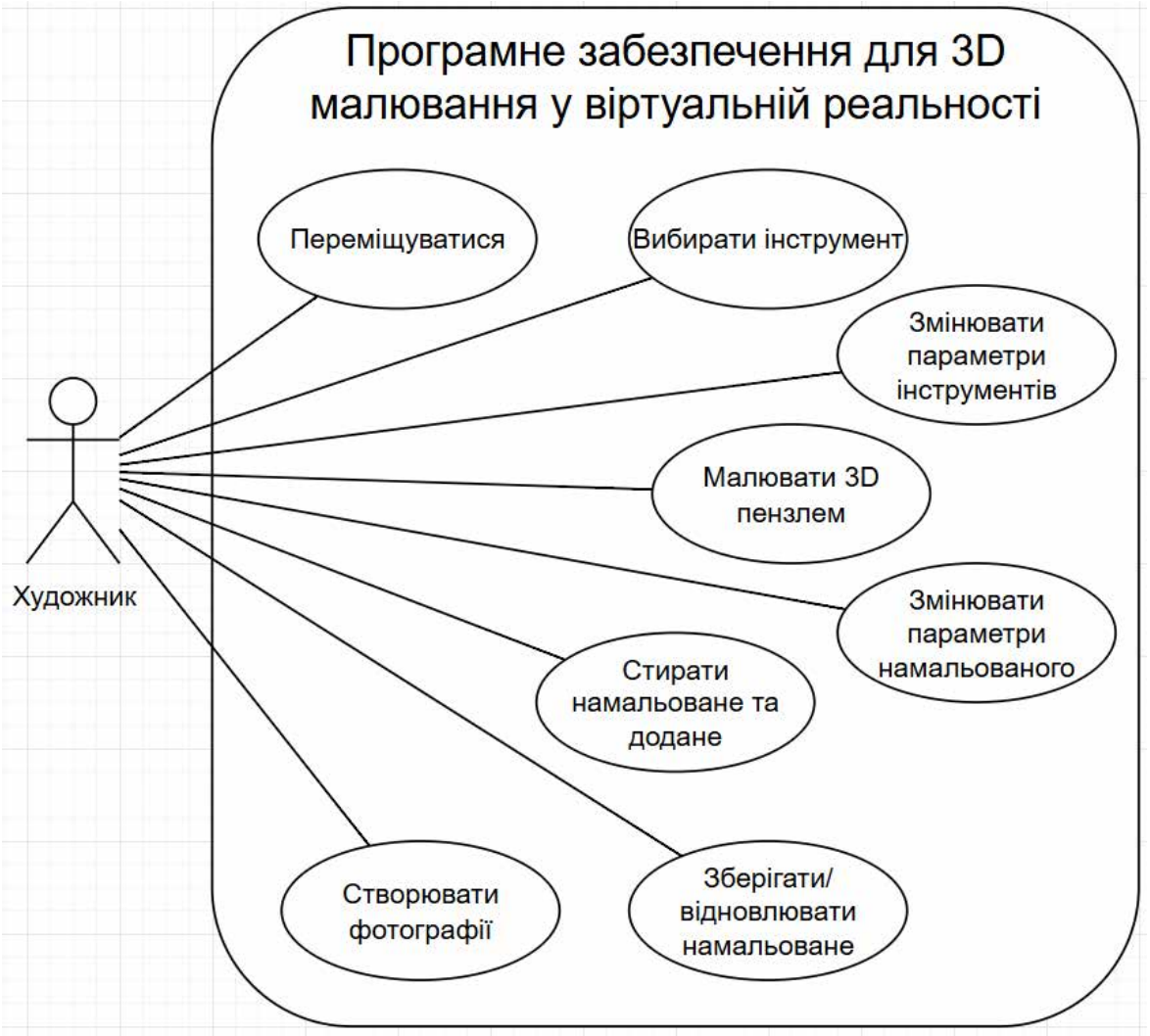

Рис.1 - Діаграма прецедентів

Ця діаграма є основою для побудови подальших діаграм активності, послідовності тощо.

1.3.2 Абстракції. У межах моделювання предметної області наступним кроком після визначення прецедентів є виділення абстракцій — ключових понять, що існують у межах предметної області. Для цього побудовано діаграму абстракцій(рис. 2), яка візуально демонструє основні об'єкти, їхні властивості та дії які вони повинні виконувати.

1.3.2.1 Художник. Має такі властивості, як:

- Положення у просторі та поворот голови
- Положення у просторі та поворот обох рук
- Інструменти та їх налаштовані параметри(розмір, колір тощо)

Та має виконувати наступні дії:

- Переміщуватися
- Обирати інструменти та їх налаштування
- Створювати криві
- Редагувати створені криві
- Видаляти створені криві
- Зберігати/відновлювати результат своєї праці
- Робити фотографії створених малюнків

1.3.2.1 Мальована крива. Має такі властивості, як:

- Форма
- Положення
- Поворот
- Розмір
- Колір
- Точки за якими крива малюється

Та має виконувати наступні дії:

- Створити криву
- Змінити параметри кривої
- Видалити криву

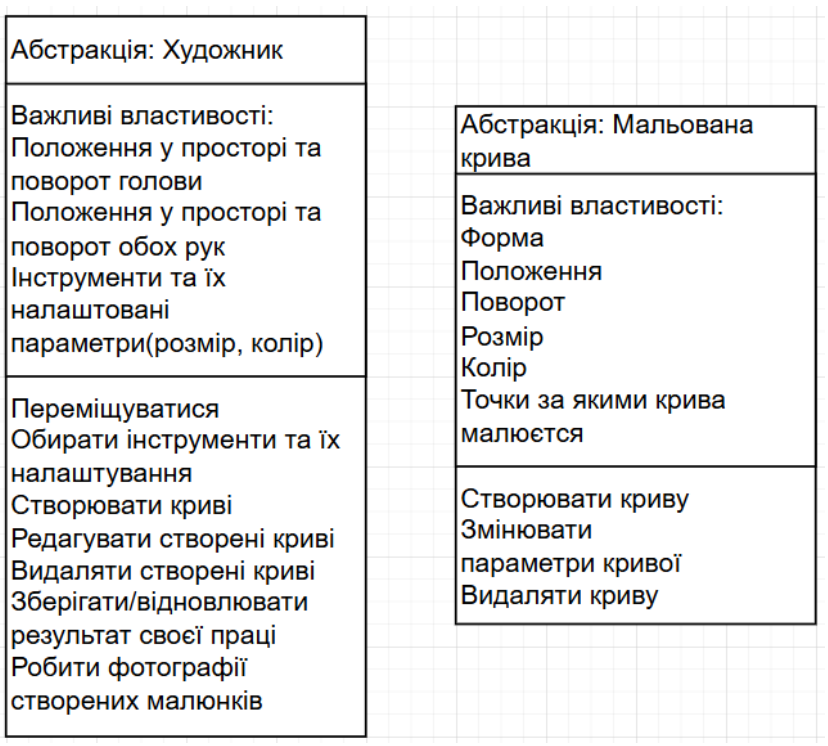

Рис. 2 - Абстракції

Призначення цієї діаграми — сформувати абстракції для предметної області, для подальшого використання на етапі проєктування системи

1.3.3 Діаграма активності. Діаграма активності відображає послідовність дій та рішень які робить художник під час малювання, від початку до кінця. На відміну від прецедентів та абстракцій, які фокусуються тільки на діях, діаграма активності моделює послідовність виконання цих дій.

Послідовність дій, зображена на діаграмі активності(рис. 3), показує, що після початку роботи над малюнком художник може одночасно переміщуватись, обирати різні інструменти і використовувати їх, робити фотографії своїх малюнків та зберігати результати для продовження малювання в майбутньому.

Друга частина діаграми(рис. 4), більш детально показує як саме художник використовує інструменти, а саме, при виборі одного з них:

Пензель — для нього художник обирає колір та форму, і малює в повітрі.

Селектор — виділяє намальовані криві і може їх змінювати.

Гумка — стирає намальовані криві.

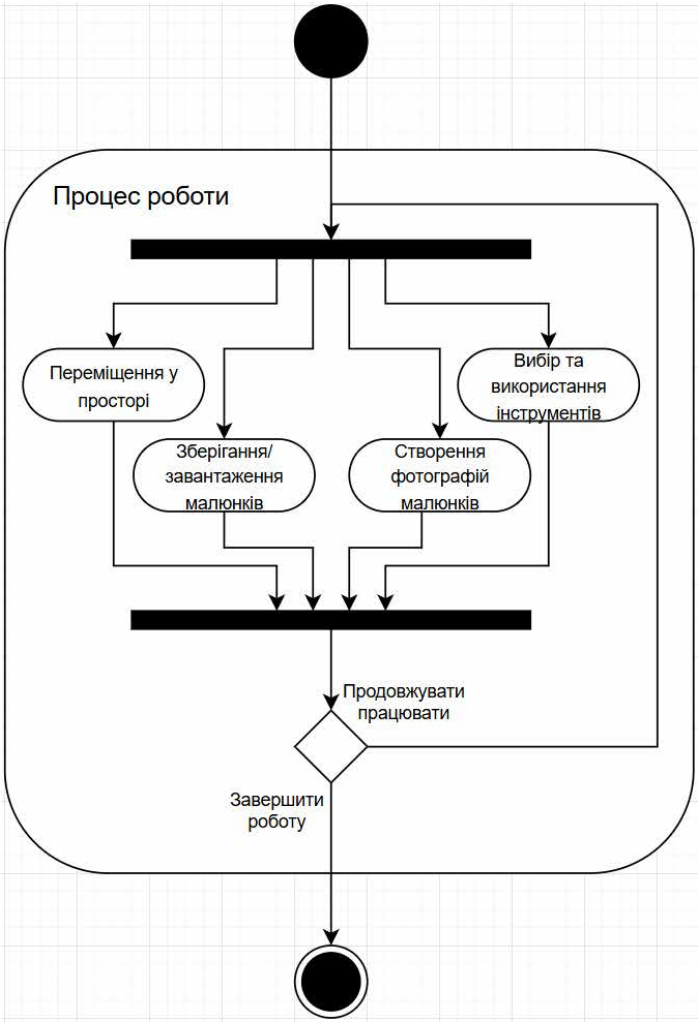

Рис. 3 - Перша частина діаграми активності

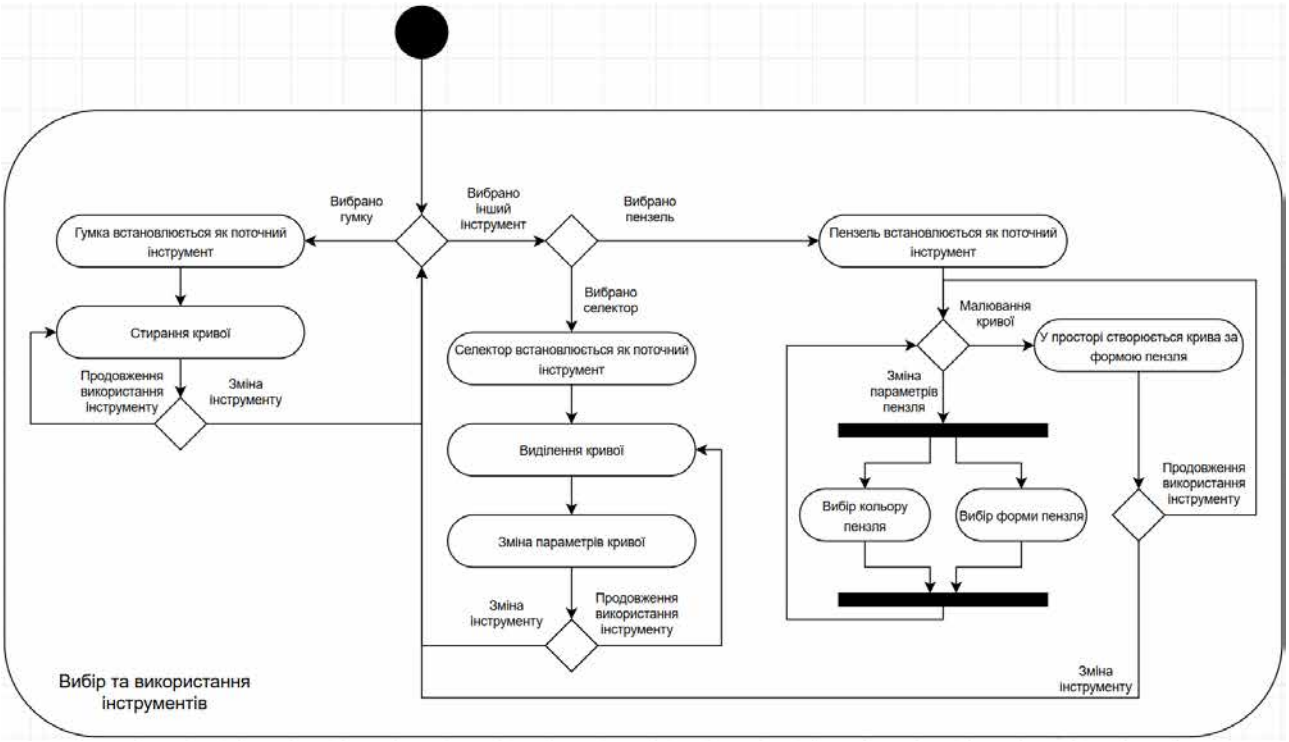

Рис. 4 - Друга частина діаграми активності

1.3.4 Діаграма послідовності. Діаграма послідовності моделює часову послідовність виконання певних дій в більш детальному масштабі у рамках певного сценарію. У контексті предметної області для 3D-малювання вона дозволяє відобразити, як саме відбувається виконання тієї чи іншої дії.

На представленій діаграмі(рис. 5) зображена одна з головних дій — малювання кривої.

Послідовність дій в цьому сценарії наступна, художник бере пензель, вибирає для нього розмір та колір, починає малювати. Після чого, поки художник малює, крива з’являється там де проходить пензель. В результаті художник отримує намальовану криву.

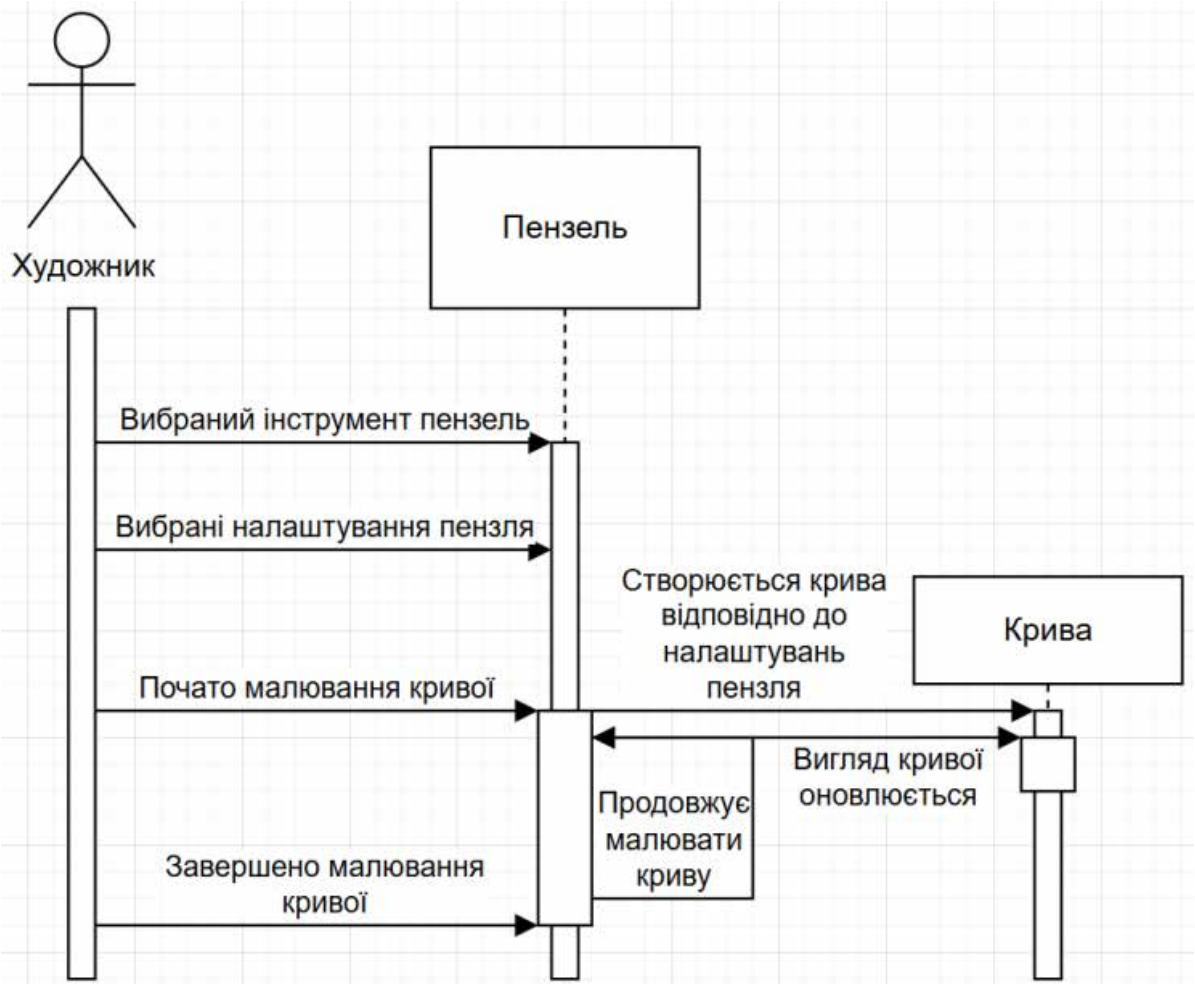

Рис. 5 - Малювання кривої зображене у вигляді діаграми послідовності

Ця діаграма завершує логічний ланцюг моделювання предметної області, демонструючи, як абстракції функціонують у динамічному контексті, а прецеденти реалізуються через взаємодії цих абстракцій.

1.4 Огляд інформаційних джерел та існуючих рішень

Розробка програмного забезпечення для 3D-малювання у віртуальній реальності потребує аналізу вже наявних рішень, а також вивчення теоретичних та практичних джерел, що стосуються VR-технологій, тривимірної графіки та UI/UX-дизайну у віртуальному середовищі.

1.4.1 Огляд літературних та наукових джерел. У процесі підготовки було опрацьовано наукові публікації, статті та підручники з таких напрямків:

- Основи побудови VR-середовищ та використання технології OpenXR;
- Принципи створення 3D-графіки, в тому числі — трасування променів, побудова мешів;
- Інтерфейси у віртуальній реальності, UX-патерни взаємодії у 3D-просторі (дослідження компаній Unity, Meta, Valve).

Також було враховано технічну документацію:

- Unity Documentation;
- XR Interaction Toolkit Manual;
- OpenXR API Reference;
- Рекомендації з розробки VR-додатків від Microsoft, Meta, SteamVR.

1.4.2 Аналіз існуючих рішень. На сучасному ринку є кілька популярних VR-додатків, що реалізують функціонал 3D-малювання:

Tilt Brush (розробка Google, пізніше — відкритий проєкт Open Brush): дозволяє створювати просторові мазки, застосовувати анімовані ефекти, переміщатися у просторі. Основний недолік — обмежена підтримка формату експорту та деяка складність інтерфейсу для новачків.

Quill (від Meta): орієнтований на художників-аніматорів. Підтримує створення складних сцен з ключовими кадрами. Висока якість, але потребує великого обсягу ресурсів і глибокого вивчення.

Gravity Sketch: використовується для концепт-дизайну, промислового проєктування та UX-моделювання. Підтримує моделювання за допомогою примітивів, але менш адаптований під вільне художнє малювання.

Таблиця 1.1

Порівняння вже існуючих додатків

Назва	Платформи	Основна функціональність	Особливості
Tilt Brush / Open Brush	SteamVR, Quest	Вільне малювання 3D-мазками	Ефекти, обмежений експорт
Quill by Meta	Oculus	Анімація в VR	Високий поріг входу
Gravity Sketch	SteamVR, Quest	Промислове 3D-моделювання	Технічний фокус, менше художніх функцій

1.5 Постановка завдання

Метою даної дипломної роботи є розробка програмного забезпечення, що дозволяє користувачеві виконувати 3D-малювання у віртуальному середовищі з використанням VR-гарнітури. Для цього необхідно реалізувати систему, яка забезпечить інтуїтивно зрозумілий інтерфейс, високий рівень інтерактивності, гнучкий вибір інструментів малювання та можливість збереження результатів.

Основні задачі, які необхідно вирішити в межах проєкту:

- Проаналізувати предметну область, виділити ключові поняття та процеси, пов’язані з 3D-малюванням у VR.
- Розробити технічне завдання та архітектуру програмної системи, що охоплює основні функціональні модулі, зокрема: модуль обробки введення користувача через VR-контролери, інтерфейс користувача,

модуль збереження/завантаження даних, система управління інструментами малювання.

- Підготувати візуальні UML-діаграми, які описують структуру системи, взаємодію класів, основні сценарії використання, тощо.
- Реалізувати програмне забезпечення у середовищі Unity із застосуванням OpenXR, забезпечивши сумісність з WMR.
- Провести тестування системи та оцінити її зручність у реальному середовищі.

Очікуваний результат

Результатом виконання поставленого завдання має стати функціональний VR-додаток, що дозволяє користувачам створювати 3D-малюнки в реальному часі у віртуальному просторі з використанням інтуїтивно зрозумілих інструментів малювання, а також переглядати, редагувати та зберігати результати своєї творчості.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

У процесі розробки програмного забезпечення для 3D-малювання у віртуальній реальності було створено логічну модель даних у вигляді ER-діаграми(рис. 7), яка відображає основні сутності системи, їх атрибути та зв’язки між ними. Модель охоплює всі ключові компоненти, необхідні для управління збереженими сценами, інструментами, кривими малювання та взаємодією користувача з VR-середовищем.

Основні сутності та зв’язки:

- SaveFile – збереження сцен користувача з назвою, даними кривих, місцем розташування та датою;
- ObjectHandler – посередник між збереженим файлом і об’єктами на сцені;
- DrawnCurve – елемент, створений користувачем за допомогою інструменту малювання. Має параметри кривої такі як форма(яка йде з об’єкта primitive), позиція, поворот, колір та точки за якими крива малюється;
- 2D_Primitive – базовий примітив (форма, поворот, розмір);
- VRController – контролер, за допомогою якого здійснюється малювання;
- VRHelmet – VR-шолом користувача, що відповідає за просторову орієнтацію;
- DrawerEntity – узагальнений об’єкт, що відповідає за дії користувача;
- Brush, Selector, Eraser – спеціалізовані типи інструментів;
- Instrument – інструмент малювання, пов’язаний із контролером.

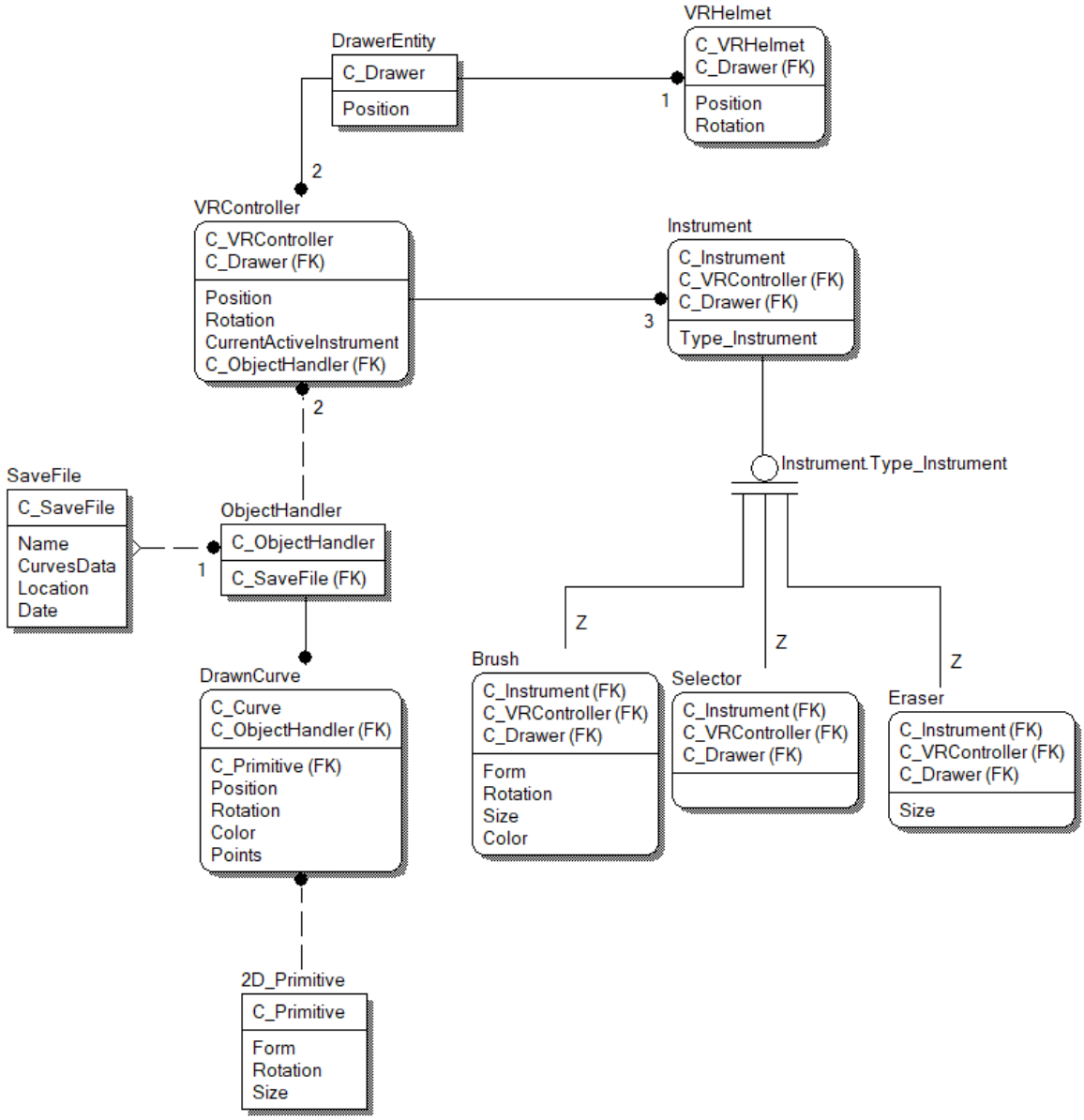

Рис. 7 - ER-діаграма

Коментар до діаграми:

- Зв’язки типу 1, 2, 3 вказують на кількість екземплярів (один до одного, один до багатьох тощо);
- Всі інструменти наслідуються від базового типу Instrument, що реалізує поліморфну взаємодію з різними типами інструментів;
- Вся структура підтримує повну трасування від збереженого файлу до кожного створеного об’єкта.

2.2 Діаграма класів та кооперацій

Для детального структурування компонентів програмного забезпечення було створено UML-діаграми класів та кооперацій. Вони відображають логіку побудови об'єктів, їх атрибути, методи, а також взаємозв’язки та сценарії використання системи у VR-середовищі.

2.2.1 Діаграма класів. UML-діаграма класів(рис. 8) демонструє структуру наступних об'єктів:

Клас Художник агрегує систему збереження, контролери та шолом;

Клас Контролер включає список інструментів, а також методи маніпуляції з інструментами й положенням;

Клас Шолом позицію та поворот, а також методи переміщення;

Система збереження має методи для збереження/завантаження сцени та збереження фотографій;

Інструменти (Пензель, Селектор, Гумка) реалізують стандартний інтерфейс, але мають власні унікальні поля;

Примітив та Мальована крива моделюють об’єкти, що створюються у віртуальному просторі. Мальована крива має примітив та містить свій набір параметрів.

Композиційні зв’язки (has-a) вказують на володіння об’єктами: наприклад, фігура має криву, а контролер — інструменти.

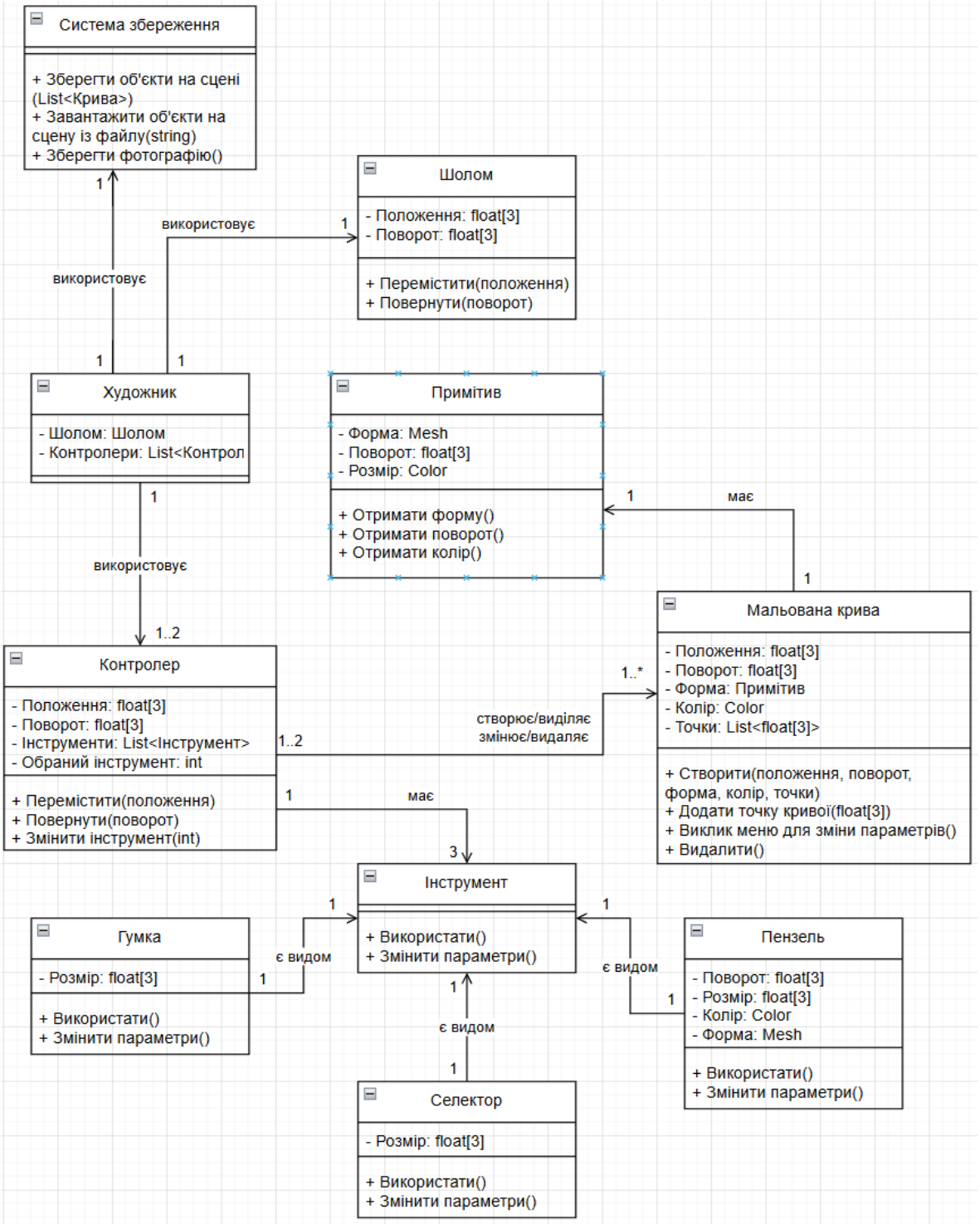

Рис. 8 - Діаграма класів

Ця діаграма наочно показує, як компоненти системи пов’язані між собою на рівні об'єктно-орієнтованої моделі, що дозволяє ефективно реалізувати VR-додаток.

2.2.2 Діаграма кооперацій. Поділяється на 3 частини, кожна з яких моделює відповідні взаємозв’язки між компонентами.

2.2.2.1 Кооперація VR-гарнітура. У цьому фрагменті описано співпрацю між користувачем системи (Художником), шоломом та контролерами(рис. 9). Вказані основні параметри:

- Шолом: має атрибути положення та повороту в просторі; забезпечує функції переміщення та обертання;
- Контролер: працює з інструментами, відстежує координати, забезпечує маніпуляцію інструментами;
- Художник: агрегує шолом і контролери як єдину точку входу взаємодії користувача з VR, а також використовує систему збереження.
- Система збереження: дозволяє зберігати/завантажувати сцену, а також зберігати фотографії.

Зв’язки типу 1..2 демонструють, що користувач працює щонайменше з одним і максимум з двома контролерами.

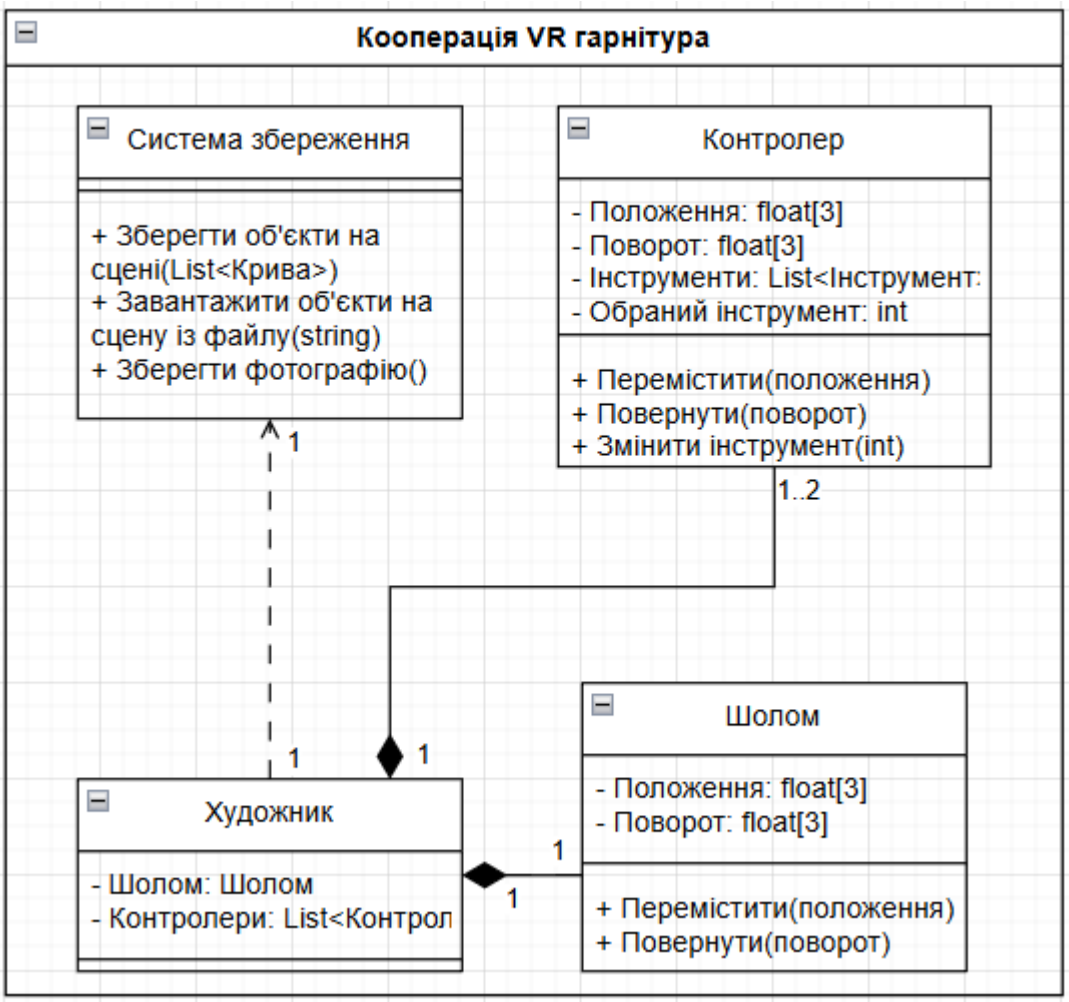

Рис. 9 - Кооперація VR-гарнітура

2.2.2.2 Кооперація об’єкти. Тут представлена логіка створення та редагування примітивів і кривих у віртуальному просторі(рис. 10):

Примітив: має параметри форми, повороту та розміру. Забезпечує методи створення і редагування.

Мальована крива: має базовий примітив, і має набір точок за якими малюється — що дозволяє створювати складні контури.

Контролер: взаємодіє з кривими, створює або змінює їх.

Ця частина моделі показує, як у результаті дій користувача створюються складові сцени.

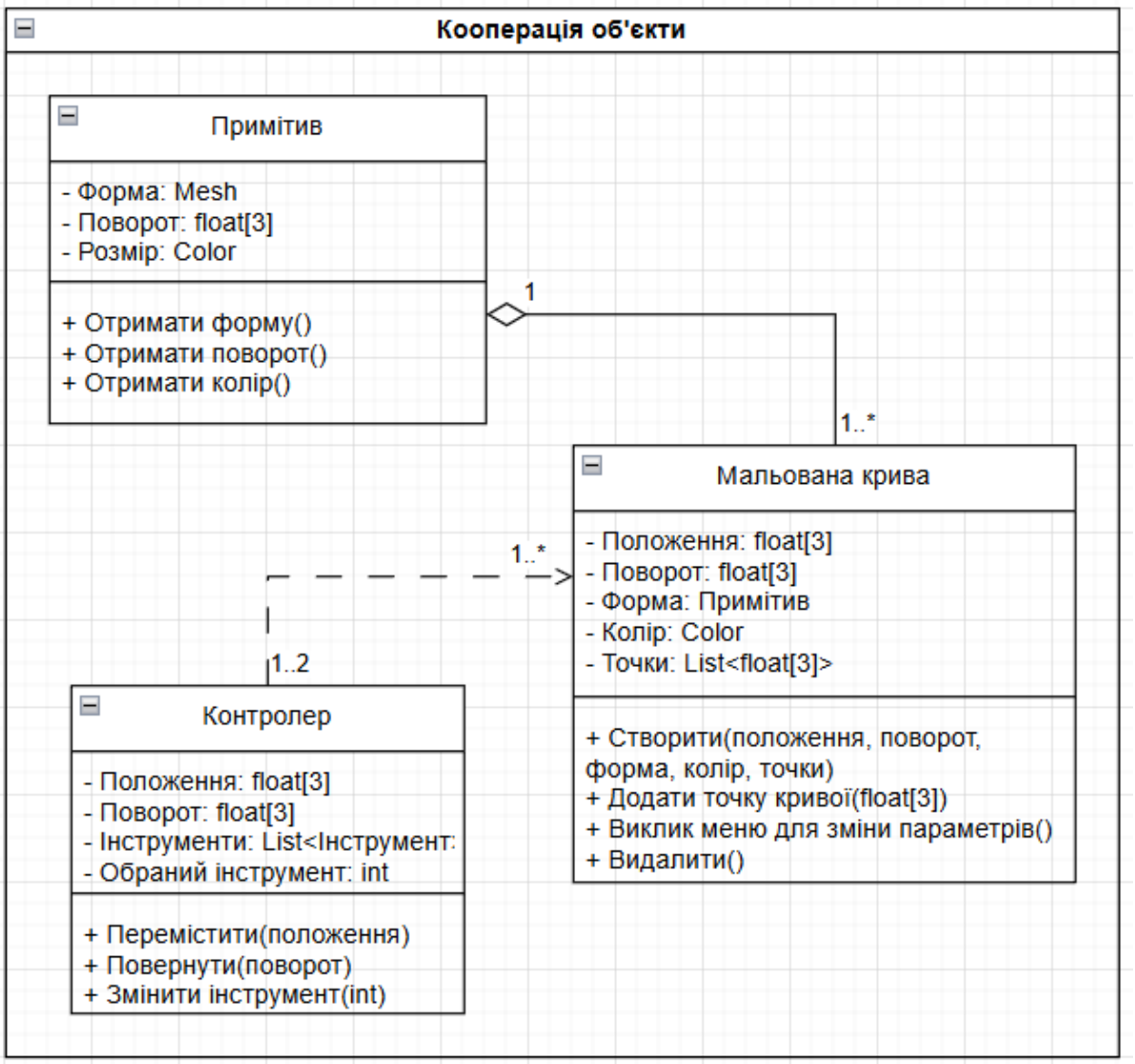

Рис. 10 - Кооперація об’єкти

2.2.2.3 Кооперація інструменти. Розглядається структура класів інструментів, які використовуються для створення або редагування об’єктів(рис. 11):

Інструмент (базовий клас): містить загальні методи Використати() та Змінити параметри(). Наслідування:

- Селектор – працює з параметрами кривих;
- Пензель – має атрибути повороту, розміру, кольору. Дозволяє малювати криві;
- Гумка - дозволяє видаляти намальовані криві.

Контролер: має 3 інструменти, по одному екземпляру кожного.

Цей фрагмент демонструє гнучкість системи щодо додавання нових інструментів через наслідування базового класу.

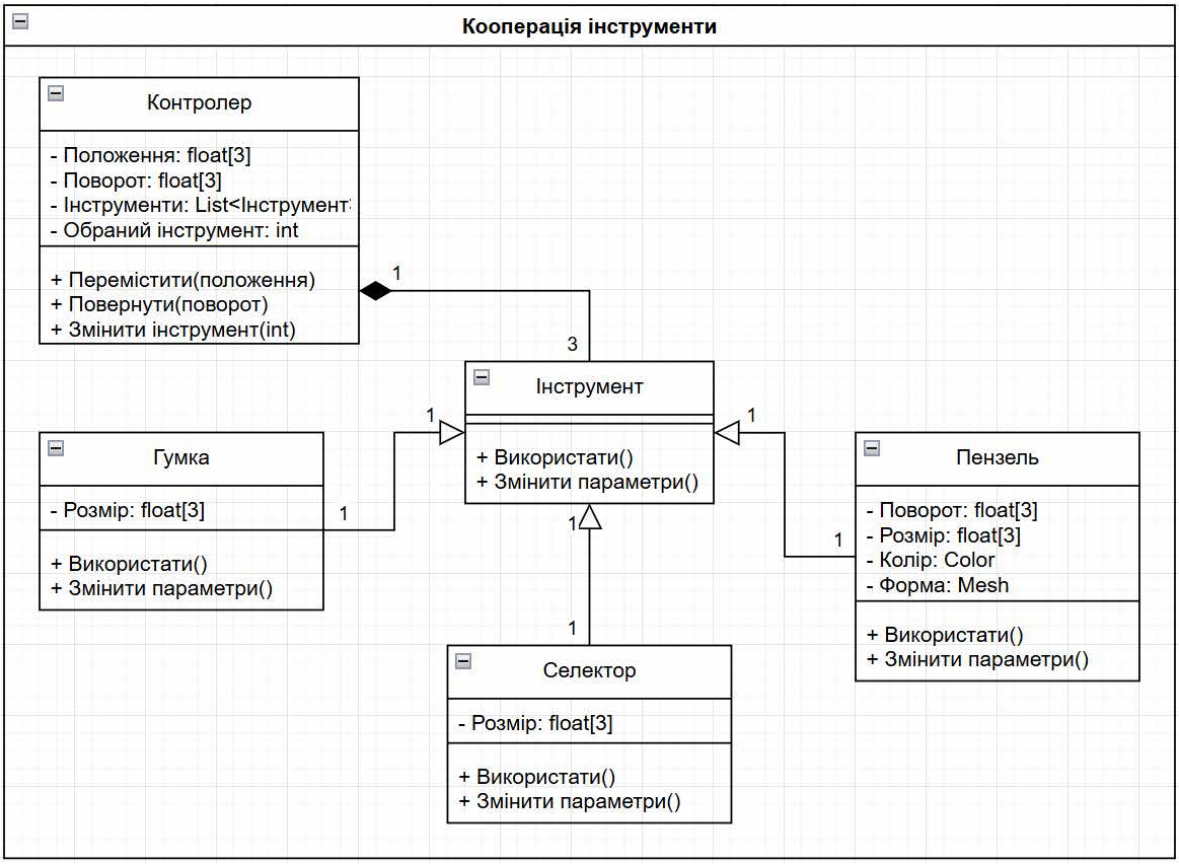

Рис. 11 - Кооперація інструменти

2.3 Діаграма пакетів

Для забезпечення модульності, масштабованості та зрозумілої структури проєкту було спроєктовано діаграму пакетів(рис. 12), яка відображає основні логічні компоненти системи та їхні взаємозв’язки.

2.3.1 Пакети.

Drawer — основний пакет, що містить головні елементи взаємодії користувача з VR-додатком:

- Шолом — відповідає за положення та орієнтацію користувача;
- Контролери — керують вибором та використанням інструментів;
- Художник — центральний клас, що координує взаємодію користувача з програмою.

Instruments — пакет інструментів для взаємодії з об'єктами. Містить класи:

- Пензель — реалізує функціонал створення кривих;
- Селектор — дозволяє редагувати об'єкти;
- Гумка — інструмент для знищення об'єктів;
- Інструмент — базовий клас для всіх інструментів.

Objects data — пакет, що містить структури даних об'єктів, які створюються у середовищі:

- Примітив — основний об’єкт із параметрами (позиція, колір, форма);
- Мальована лінія — наслідує примітив, що представляє криву, створену інструментом малювання.

Save system — пакет, що зберігає дані об’єктів:

- Система збереження — має функції для збереження даних;

2.3.2 Зв’язки між пакетами.

Drawer <<import>> Instruments — пакет Drawer імпортує інструменти, щоб використовувати їх для створення або редагування об’єктів.

Drawer <<access>> Objects data — пакет Drawer звертається до даних об’єктів для зчитування або модифікації їхніх властивостей під час взаємодії з користувачем.

Drawer <<access>> Save system — пакет Drawer звертається до системи збереження для виклику необхідних функцій.

Save system <<access>> Objects data — пакет Save system звертається до даних об’єктів для їх збереження.

Ця діаграма демонструє модульний підхід до побудови VR-додатку, що дозволяє легко розширювати функціонал, підтримувати розділення відповідальності та повторно використовувати компоненти системи.

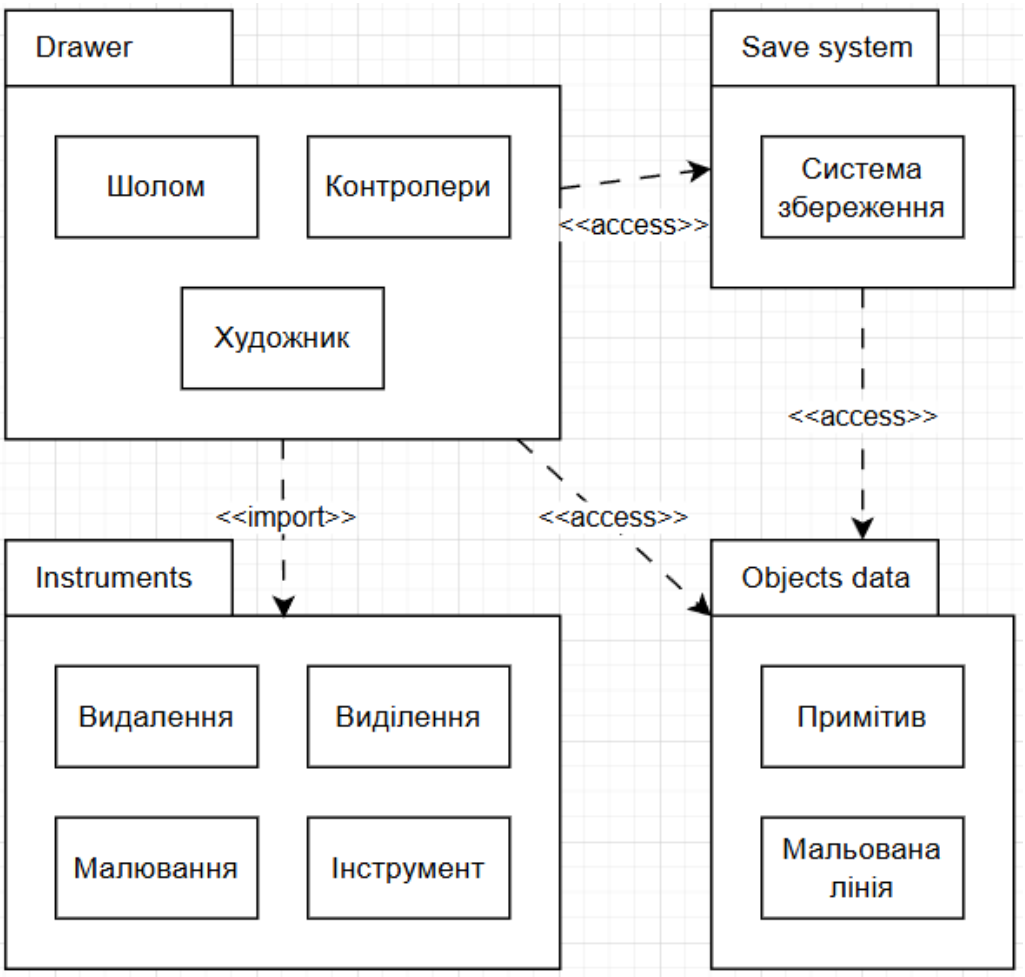

Рис. 12 - Діаграма пакетів

2.4 Діаграма компонентів

На діаграмі компонентів(рис. 13) зображено основну структуру програмного забезпечення VArt.exe — застосунку для VR-малювання. Центральний компонент взаємодіє з декількома підсистемами, що виконують конкретні функції. Нижче подано опис кожного з компонентів та їх зв’язків.

2.4.1 Компоненти.

VArt.exe — основний виконуваний файл програми, який інкапсулює інші компоненти, включаючи менеджери, інструменти та збереження.

Компоненти всередині VArt.exe:

- ToolManager — менеджер, який керує активними інструментами для малювання.
- Tool (abstract) — абстрактний інтерфейс інструменту.
- Brush — відповідає за малювання кривих та їх параметри про малюванні.
- Eraser — видаляє криві.
- Selector — виділяє та дозволяє змінювати параметри кривих.
- SaveManager — відповідальний за збереження даних. Працює з SaveFile.json — файл, до якого зберігаються об’єкти на сцені.
- BrushStrokeManager — менеджер мазків, який передає дані для збереження.

2.4.2 Зв’язки між компонентами.

ToolManager використовує абстрактний Tool, а той, у свою чергу, реалізується як Brush, Eraser, Selector.

SaveManager має відношення до:

- SaveFile.json
- BrushStrokeManager

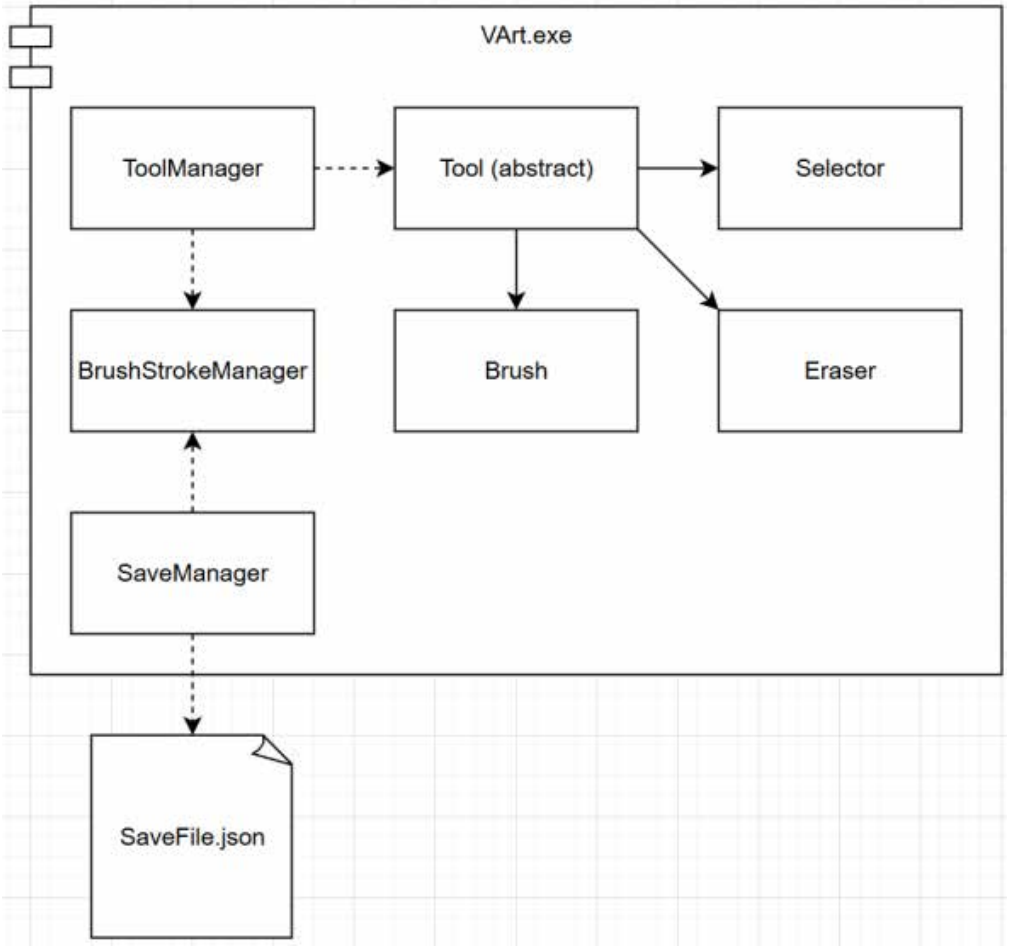

Рис. 13 - Діаграма компонентів

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

У розробленому VR-додатку VArt інформаційна база формується з даних, що зберігають дані всіх об’єктів на сцені. Основним компонентом системи управління інформаційною базою є SaveManager, який забезпечує збереження та завантаження стану сеансу малювання.

3.1.1 Структура інформаційної бази. Інформаційна база представлена у форматі JSON-файлів. Це обумовлено такими перевагами формату JSON:

- Легка читабельність і простота структурування;
- Підтримка вкладених об’єктів, що дозволяє зручно зберігати дані;
- Сумісність з більшістю мов програмування та платформ.

У структурі збереженого файлу (SaveFile.json) зберігаються дані масиву кривих, а саме їх:

- Позиція
- Поворот
- Форма
- Колір
- Набір точок

3.1.2 Операції над інформаційною базою. Передбачені наступні операції:

- Створення нової сцени – ініціалізується новий об’єкт інформаційної бази;
- Збереження сцени – здійснюється серіалізація поточного стану в JSON-файл;
- Завантаження сцени – десеріалізація файлу у внутрішні структури програми;
- Резервне копіювання – періодичне створення копій у випадку критичних помилок або падінь програми.

3.2 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка VR-додатку для тривимірного малювання потребує ретельного підбору інструментарію, який забезпечить високу продуктивність, сумісність з VR-пристроями та гнучкість у візуалізації 3D-сцен. Нижче наведено обґрунтування вибору основних засобів, використаних у процесі створення програмного продукту.

3.2.1 Engine: Unity. Основною платформою для реалізації програмного забезпечення обрано Unity, що є потужним середовищем розробки 2D та 3D застосунків, зокрема для віртуальної та доповненої реальності.

Переваги Unity:

- Вбудована підтримка VR-пристроїв через XRIT;
- Потужна система рендерингу (URP, HDRP);
- Велика спільнота та доступ до Asset Store;
- Гнучка робота з 3D-об'єктами та сценами;
- Зручна система подій, анімацій та обробки вводу.

Unity також дозволяє інтегрувати C#-скрипти, які реалізують логіку інструментів, малювання, взаємодії з користувачем та обробки інформації.

3.2.2 Мова програмування: C#. Unity працює на основі мови C#, яка є сучасною, об'єктно-орієнтованою мовою з широкими можливостями для модульного проєктування.

Переваги C#:

- Чітка структура класів і типів;
- Інтеграція з Unity API;
- Потужні засоби налагодження;
- Підтримка шаблонів, делегатів та подій.

3.2.3 Бібліотеки та SDK. Обрані бібліотеки, забезпечать зручний інструментарій для написання програмного коду та функціонування відповідних модулів.

XRIT — офіційна бібліотека для підтримки взаємодії в XR-середовищах (включно з контролерами, жестами, взаємодією з об’єктами).

Mixed Reality Toolkit (MRTK) — набір компонентів для покращеної інтеграції з WMR.

Json.NET (Newtonsoft.Json) — бібліотека для серіалізації та десеріалізації даних, що використовується для збереження інформаційної бази (формат JSON).

3.2.4 Редактори. Завдяки наступним редакторам забезпечується створення необхідних компонентів програми.

Visual Studio 2022 — основне середовище розробки з підтримкою Unity-інтеграції, зручним редактором коду, IntelliSense та засобами налагодження.

Blender — використовується для створення та редагування 3D-моделей, які потім імпортуються до сцени Unity.

3.2.5 Цільова платформа. Основною цільовою платформою обрано Windows 10/11 з VR-підтримкою через WMR. Це обумовлено наявністю відповідного обладнання для тестування та широкою підтримкою драйверів.

3.3 Алгоритмізація та програмування програмних модулів

Весь код, що буде продемонстрировано в цьому розділі, доступний у додатоку Б.

3.3.1 Обробка вводів. У середовищі віртуальної реальності введення з боку користувача є основою будь-якої взаємодії з програмою. У даному додатку обробка вводів реалізована через інтеграцію WMR та XRIT в Unity. Це дозволяє в повній мірі використовувати можливості VR-контролерів, включаючи позиційне відстеження, натискання кнопок, роботу з аналоговими тригерами, джойстиками та тачпадом.

3.3.1.1 Основні типи введення.

Кожен кадр програма оновлює положення контролерів у просторі, що дозволяє точно визначати, де саме користувач тримає руку, куди вона спрямована, і як вона обертається. Це особливо важливо для коректної візуалізації інструментів малювання, які прив’язуються до позиції контролера. Ця технологія є в Unity за замовчуванням і реалізована за допомогою XRIT.

У програмі задіяні основні кнопки контролерів, включаючи:

- тригер (Trigger) — для малювання, видалення або виділення об’єктів;
- грип-кнопка (Grip) — допоміжна кнопка для перемикання режиму введення;
- меню (Menu) — для відкриття/закриття головного меню;
- джойстик/тачпад — для переміщення в просторі або зміни певних налаштувань інструментів.

3.3.1.2 Архітектура обробки. Кожен контролер у системі має окремий обробник подій, що реалізований як компонент Unity на відповідному об'єкті і містить логіку, що відповідає за:

- відстеження натискань;
- виклик методів відповідних дій (початок малювання, виклик меню, виділення об’єкта тощо);
- перемикання режимів інструментів.

Події введення обробляються у методах InteractionSourcePressed, InteractionSourceUpdated та InteractionSourceReleased(рис. 14), де перевіряється стан кожної кнопки на кожному контролері. Обробник цих подій вбудований в XRIT, завдяки чому потрібно реалізувати тільки дії, які відбуваються при активації подій(рис. 15). При виявленні відповідних подій викликаються функції. Наприклад:

- Тригер натиснуто → почати малювання кривої.
- Джойстик зміщено → переміщення моделі користувача в просторі.
- Меню натиснуто → переключити видимість головного меню.

Переваги реалізованої системи:

Гнучкість: логіку кожної кнопки можна легко змінювати або адаптувати під нові інструменти.

Модульність: окремі скрипти відповідають за конкретні види взаємодії, що спрощує тестування та налагодження.

```
Ссылок: 0
private void InteractionSourcePressed(InteractionSourcePressedEventArgs obj)...

Ссылок: 0
private void InteractionSourceUpdated(InteractionSourceReleasedEventArgs obj)...

Ссылок: 0
private void InteractionSourceReleased(InteractionSourceReleasedEventArgs obj)...
```

Рис. 14 - Структура функій обробника подій

```
else if (obj.state.source.handedness == InteractionSourceHandedness.Right
    && obj.pressType == InteractionSourcePressType.Touchpad
    && gripPressed)
{
    Vector2 touchpadPosition = obj.state.touchpadPosition;
    if (touchpadPosition.magnitude > 0.5f)
    {
        ToolType selected = currentTool.Type;

        if (touchpadPosition.y > 0.5f)
            selected = ToolType.Brush;
        else if (touchpadPosition.x < -0.5f)
            selected = ToolType.Eraser;
        else if (touchpadPosition.x > 0.5f)
            selected = ToolType.Selector;

        if (selected != currentTool.Type)
            SetTool(selected);
    }
}
```

Рис. 15 - Реалізація обробки подій в InteractionSourcePressed для зміни інструменту

3.3.2 3D моделі контролерів. Для забезпечення максимальної зручності взаємодії та візуальної ідентифікації контролерів у віртуальному середовищі було використано реалістичні 3D-моделі(рис. 16), які повністю відповідають зовнішньому вигляду справжніх VR-контролерів WMR.

Моделі було взято з офіційної сторінки Microsoft з розробки VR-додатків на базі HoloLens і WMR, де наявні готові ресурси, що включають:

- 3D-моделі лівого та правого контролера,
- текстури,
- анімації натискання кнопок,
- колесо вибору кольору та його анімація.

Моделі було імпортовано у середовище Unity, після чого вони були зв’язані з фізичними пристроями через XR Rig. У результаті:

- 3D-модель контролера автоматично слідує за рухами руки користувача;
- всі натискання кнопок анімовано – наприклад, при натисканні на тригер він віртуально "вгинається";
- контролери використовуються також як базова точка для прив’язки інструментів малювання та інтерфейсних елементів.

Рис. 16 - 3D моделі контролерів

3.3.3 Інтерфейс. Інтерфейс користувача у VR-додатку для 3D-малювання виконує роль центральної системи навігації та налаштувань, яка дозволяє керувати основними функціями програми без виходу з віртуального середовища. Основна мета при створенні інтерфейсу – забезпечення інтуїтивного керування, яке не відволікає від творчого процесу.

Інтерфейс умовно поділено на два основних компоненти:

- Головне меню
- Міні-меню параметрів

3.3.3.1 Головне меню. Головне меню(рис. 17) з’являється перед користувачем при запуску додатку або після натискання відповідної кнопки(додаток А). Його реалізовано у вигляді плаваючої панелі, яка фіксується у просторі перед користувачем і містить основні опції:

- Створити нову сцену
- Зберегти сцену
- Завантажити сцену
- Вийти з додатку

Рис. 17 - Вигляд головного меню

Взаємодія з меню здійснюється за допомогою променя (raycast), який виходить з контролера. Кнопки мають візуальний відгук при наведенні та натисканні (зміна кольору, анімація).

3.3.3.2 Міні-меню. Міні-меню(рис. 18) активується натисканням на відповідну кнопку(додаток А), та з’являється перед користувачем. Це меню використовується для зміни параметрів:

- інструментів (параметри залежать від вибраного інструмента);
- об'єктів на сцені.

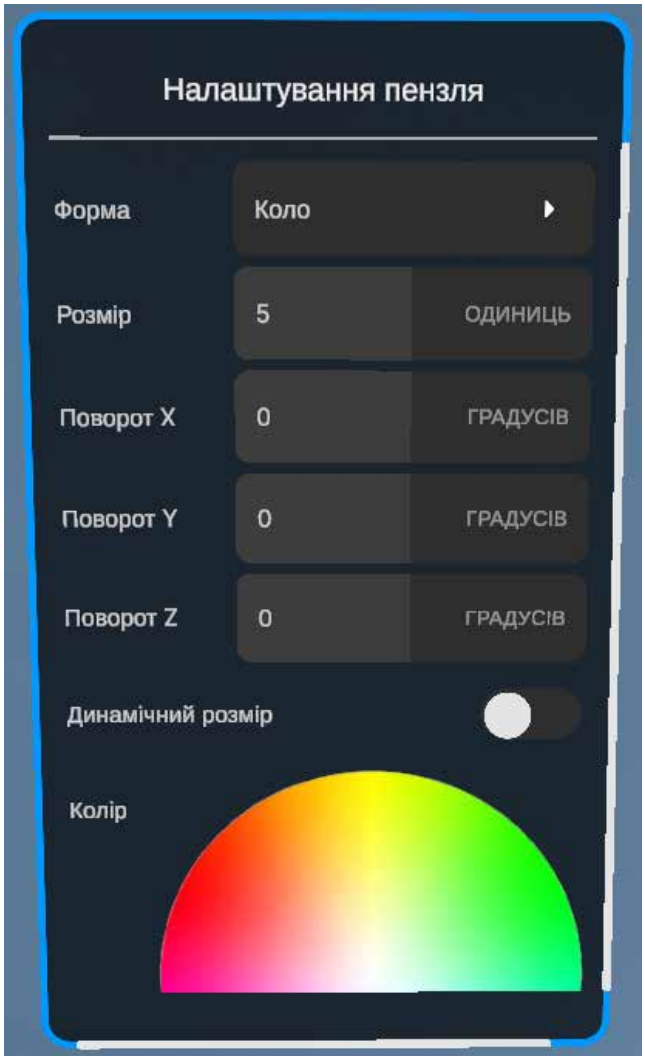

Рис. 18 - Вигляд міні-меню

Основні особливості:

- Напівпрозорий фон панелей, щоб не перекривати простір
- Збалансоване розташування елементів для легкої доступності

Таке компонування інтерфейсу дозволяє користувачу швидко змінювати параметри та виконувати дії без потреби знімати гарнітуру або перемикатися на зовнішній інтерфейс.

3.3.4 Візуальні підказки. Для підвищення зручності використання додатку у VR-середовищі було реалізовано систему візуальних підказок на контролерах, яка дозволяє користувачеві швидко орієнтуватися в доступних функціях та кнопках керування без потреби запам’ятовувати всі дії.

Система підказок активується автоматично, коли користувач певний час (1-2 секунди) фіксує погляд на контролер. Це реалізовано за допомогою відстеження напрямку погляду (gaze tracking). Якщо кут між вектором погляду та контролером менший за певне значення протягом обраного інтервалу часу, підказки з’являються.

Після активації на віртуальному зображенні контролера з’являються невеликі UI-мітки(рис. 19), що відповідають фізичним кнопкам. Біля кожної кнопки вказано її дію (наприклад: Малювати, Гумка, Меню).

Мітки мають прозорий фон, заокруглені краї та легко читаються на різному фоні. Вони зникають при відведенні погляду.

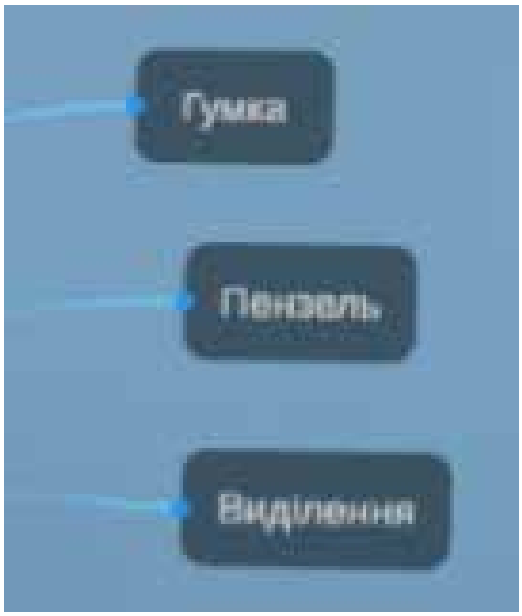

Рис. 19 - Вигляд візуальних підказок

В залежності від вибраного інструменту (пензель, селектор, гумка) або натисканні кнопки для зміни режиму вводу(додаток А), зміст підказок може змінюватись. Наприклад, якщо активовано інструмент пензель(див. рис. 19), кнопки будуть мати підписи типу Малювати, Зміна кольору, тоді як для гумки — Видалити, Зміна розміру.

Ця система значно знижує поріг входження для нових користувачів, допомагає краще орієнтуватися в інтерфейсі й пришвидшує роботу зі складними функціями, особливо під час перших сеансів у VR.

3.3.5 Інструменти. У VR-додатку реалізовано кілька базових інструментів для 3D-малювання, що дозволяють створювати, редагувати й видаляти художній контент у тривимірному просторі. Кожен інструмент має власну поведінку, унікальні параметри та відповідні методи взаємодії з об'єктами.

Пензель (Brush). Основний інструмент для створення кривих. Крива це складний об’єкт, який складається з набору примітивів, що дозволяє робити лінії будь-якої форми(рис. 20). Під час натискання кнопки на контролері відбувається реєстрація положення кисті у просторі, створюючи послідовність точок, які формують криву(рис. 23). Параметри кривої наслідуються від налаштувань пензля.

Рис. 20 - Вигляд намальованої кривої

Гумка (Eraser). Служить для видалення кривих. Працює за принципом перевірки на перетин із сферою довкола кінчика інструменту(рис. 21). Якщо крива знаходиться в зоні дії, вона видаляється.

```
[Header("Eraser Settings")]
public float eraseRadius = 0.1f;
public Transform eraseOrigin;

Ссылок: 1
private void TryErase()
{
    Collider[] hits = Physics.OverlapSphere(eraseOrigin.position, eraseRadius);

    foreach (Collider hit in hits)
        if (hit.CompareTag("BrushStroke"))
            Destroy(hit.gameObject);
}


Ссылок: 3
public override void TriggerPressed()
{
    TryErase();
}

Ссылок: 3
public override void TouchPadPressed(Vector2 input)
{
    if (input.magnitude > 0.5f)
    {
        if (input.y > 0.5f)
            eraseRadius += 0.05f;
        else if (input.y < -0.5f)
            eraseRadius -= 0.05f;
    }
    else if (input.magnitude <= 0.5f)
        MiniMenuManager.Open(this);
}
```

Рис. 21 - Реалізація видалення кривої

Виділення (Selector). Дає змогу вибрати криву для подальшого редагування. Реалізована за тим же принципом, що і видалення, тільки замість видалення відкриває меню, в якому можна міняти параметри кривих. Після виділення можна переміщати криву, змінювати її властивості або видалити(рис. 22). Виділені об’єкти підсвічуються для зручності.

```
private void TrySelect()
{
    Collider[] hits = Physics.OverlapSphere(selectOrigin.position, selectRadius);

    foreach (Collider hit in hits)
    {
        strokes.Add(hit.GetComponent<VRBrushStroke>());
        renderers.Add(hit.GetComponent<Renderer>());
    }

    if (hits.Length > 0)
        MiniMenuManager.Open(hits);
}
Ссылок: 3
public override void TriggerPressed()
{
    TrySelect();
}

Ссылок: 4
public override void TouchPad(Vector2 input)
{
    if (renderers == null) return;
    if (input.magnitude != 0)
    {
        touchPadTimer += Time.deltaTime;

        if (touchPadTimer >= holdDuration)
        {
            float angle = Mathf.Atan2(input.y, input.x);
            float hue = (angle / (2 * Mathf.PI) + 1f) % 1f;
            float saturation = Mathf.Clamp01(input.magnitude);
            float value = 1f;

            foreach (Renderer render in renderers)
                if (render != null)
                    render.material.color = Color.HSVToRGB(hue, saturation, value);
        }

    }
    else
        touchPadTimer = 0;
}
Ссылок: 3
public override void TouchPadPressed(Vector2 input)
{
    if (strokes == null) return;
    if (input.magnitude > 0.5f)
    {
        foreach (VRBrushStroke stroke in strokes)
            if (input.y > 0.5f)
                stroke.radius += 0.05f;
            else if (input.y < -0.5f)
                stroke.radius -= 0.05f;
            else if (input.x > 0.5f)
                stroke.form = PrimitiveShape.GetNextPrimitiveType(stroke.form);
            else if (input.x < -0.5f)
                stroke.form = PrimitiveShape.GetPreviousPrimitiveType(stroke.form);
    }
    else if (input.magnitude <= 0.5f)
        MiniMenuManager.Open(this);

    touchPadTimer = 0;
}
```

Рис. 22 - Реалізація виділення та зміни параметрів кривої

Вибір інструменту здійснюється через натискання на певну зону тачпаду(додаток А).

Криві в системі представлені як об'єкти типу BrushStroke, які містять:

- масив точок
- колір
- товщину
- унікальний ідентифікатор
- форму

```
public void UpdateMesh()
{
    if (points.Count < 2) return;

    int ringCount = points.Count;
    int vertsPerRing = radialSegments;
    List<Vector3> vertices = new List<Vector3>();
    List<int> triangles = new List<int>();

    for (int i = 0; i < ringCount; i++)
    {
        Vector3 center = points[i];
        Vector3 forward;
        if (i < ringCount - 1)
            forward = points[i + 1] - points[i];
        else
            forward = points[i] - points[i - 1];

        Vector3 up = Vector3.up;
        if (Vector3.Cross(forward, up).sqrMagnitude < 0.001f)
            up = Vector3.forward;

        Vector3 right = Vector3.Cross(forward, up).normalized * radius;
        up = Vector3.Cross(right, forward).normalized * radius;

        for (int j = 0; j < vertsPerRing; j++)
        {
            float angle = (2 * Mathf.PI * j) / vertsPerRing;
            Vector3 offset = Mathf.Cos(angle) * right + Mathf.Sin(angle) * up;
            vertices.Add(center + offset);
        }
    }

    for (int i = 0; i < ringCount - 1; i++)
    {
        for (int j = 0; j < vertsPerRing; j++)
        {
            int current = i * vertsPerRing + j;
            int next = (i + 1) * vertsPerRing + j;
            int currentNext = i * vertsPerRing + (j + 1) % vertsPerRing;
            int nextNext = (i + 1) * vertsPerRing + (j + 1) % vertsPerRing;

            triangles.Add(current);
            triangles.Add(next);
            triangles.Add(currentNext);

            triangles.Add(currentNext);
            triangles.Add(next);
            triangles.Add(nextNext);
        }
    }

    mesh.Clear();
    mesh.SetVertices(vertices);
    mesh.SetTriangles(triangles, 0);
    mesh.RecalculateNormals();
    mesh.RecalculateBounds();

    MeshCollider collider = GetComponent<MeshCollider>();
    if (collider == null)
        collider = gameObject.AddComponent<MeshCollider>();

    collider.convex = true;
    collider.sharedMesh = null;
    collider.sharedMesh = mesh;
}
```

Рис. 23 - Реалізація створення кривої з точок у просторі

Усі операції (додавання, редагування, видалення) здійснюються через вбудований менеджер об’єктів Unity, що забезпечує зручний доступ до структури даних.

Система інструментів побудована так, щоб бути інтуїтивно зрозумілою та гнучкою для користувача, незалежно від його досвіду у VR. Вона дозволяє легко поєднувати творчість із технологічними можливостями середовища, відкриваючи широкий простір для художнього самовираження.

3.3.6 Збереження даних. Збереження даних у VR-додатку є важливим етапом, що дає змогу користувачеві зберегти свою роботу для подальшого перегляду чи редагування. У розробленому програмному забезпеченні реалізовано функції збереження та завантаження даних сцени до зовнішніх файлів.

3.3.6.1 Формат збереження. Для збереження було обрано текстовий формат JSON(рис. 24), який забезпечує:

- простоту реалізації та читання;
- підтримку структурованих об'єктів;
- можливість легкої серіалізації та десеріалізації у Unity.

Структура одного збереженого об'єкта (кривої) включає всі параметри, що були описані в попередньому пункті.

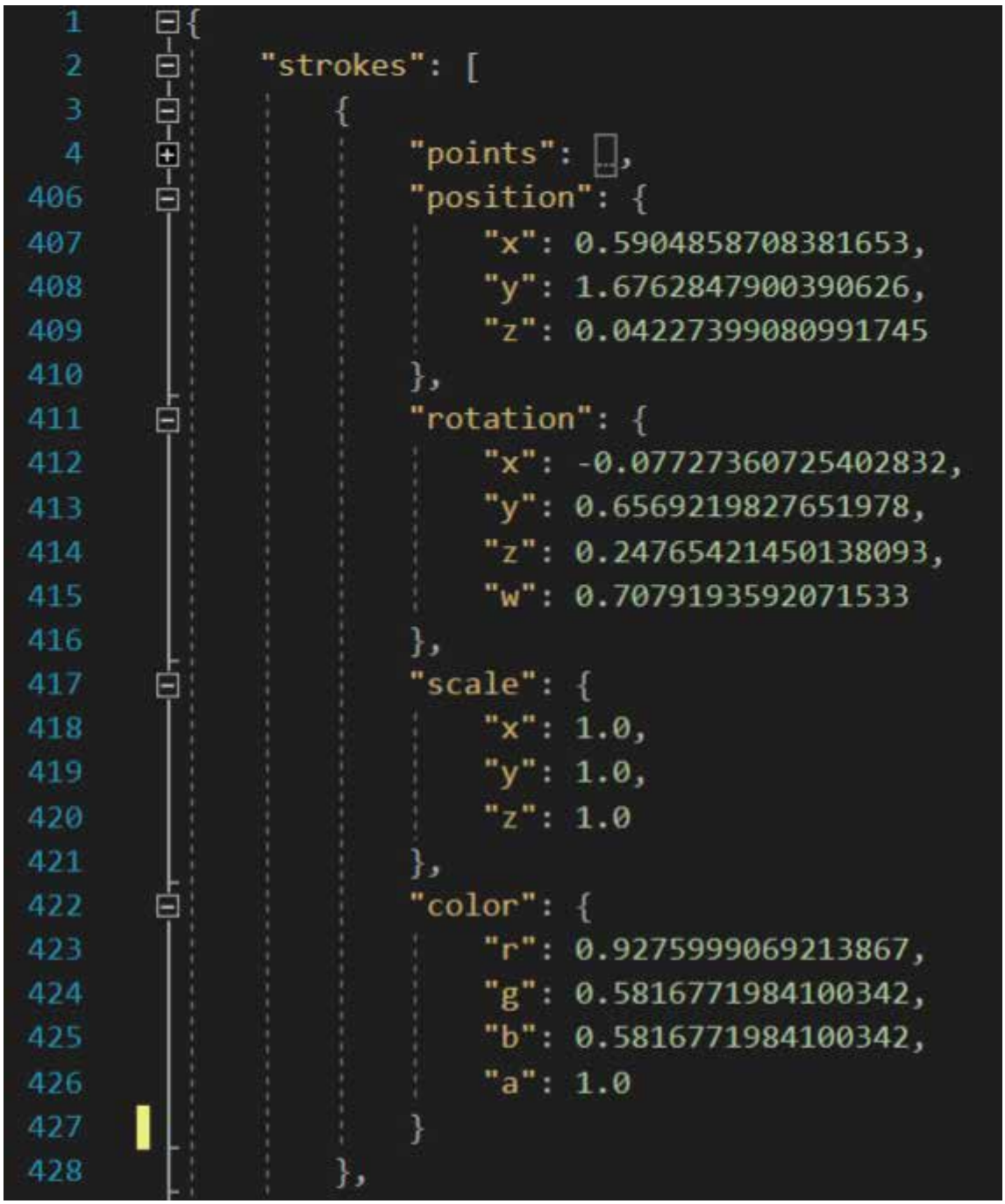

Рис. 24 - Структура JSON-файлу

3.3.6.2 Реалізація збереження. Процес збереження виконується через функцію менеджера збереження(рис. 25), який отримує доступ до всіх об'єктів типу BrushStroke, серіалізує їх у JSON-рядок та записує у файл.

Використовується клас System.IO.StreamWriter для створення файлу, а серіалізація відбувається через бібліотеку Newtonsoft.Json.

```
public void SaveAll()
{
    var allStrokes = GameObject.FindGameObjectsWithTag("BrushStroke");
    var saveFile = new SaveFile();

    foreach (var obj in allStrokes)
    {
        var stroke = obj.GetComponent<VRBrushStroke>();
        if (stroke != null)
            saveFile.strokes.Add(stroke.ToData());
    }

    string json = JsonUtility.ToJson(saveFile, true);
    File.WriteAllText(savePath, json);
    Debug.Log("Saved to: " + savePath);
}
```

Рис. 25 - Реалізація збереження даних в JSON-файл

3.3.6.3 Реалізація завантаження. При завантаженні сцени збережений JSON-файл зчитується(рис. 26), десеріалізується, і на основі отриманих даних динамічно створюються відповідні 3D-криві у просторі.

Криві додаються до сцени в тій самій позиції, з відповідними параметрами. Завантаження відбувається за викликом користувача через меню.

```
public void LoadAll(GameObject strokePrefab)
{
    if (!File.Exists(savePath))
    {
        Debug.LogWarning("Save file not found");
        return;
    }

    string json = File.ReadAllText(savePath);
    var saveFile = JsonUtility.FromJson<SaveFile>(json);

    foreach (var data in saveFile.strokes)
    {
        GameObject obj = Instantiate(strokePrefab);
        obj.tag = "BrushStroke";

        obj.transform.position = data.position;
        obj.transform.rotation = data.rotation;
        obj.transform.localScale = data.scale;

        var stroke = obj.GetComponent<VRBrushStroke>();
        if (stroke != null)
        {
            stroke.SetPoints(data.points);
            stroke.UpdateMesh();
            obj.GetComponent<MeshRenderer>().material.color = data.color;
        }
    }

    Debug.Log("Loaded from: " + savePath);
}
```

Рис. 26 - Реалізація завантаження даних із JSON-файлу

3.3.7 Реалізація в Unity. Усі програмні модулі, описані вище, були об’єднані та реалізовані в середовищі Unity.

3.3.7.1 Проєктна структура. Проєкт було організовано за такими основними компонентами, що вбудовані в Unity:

Scripts — програмна логіка інструментів, взаємодії, збереження тощо.

Prefabs — готові об’єкти (криві, контролери, підказки), які можна повторно використовувати.

Scenes — головна сцена малювання.

Materials & Models — 3D-моделі контролерів, матеріали для кривих та підказок.

3.3.7.2 Підключення XR та контролерів. Як попередньо було сказано, зчитування введень було реалізовано за допомогою XRIT. Це відбувається за допомогою обробки подій. При натисканні на будь яку кнопку викликається подія(рис. 27), що активує певну функцію. Знову ж таки, функціонал вже наданий XRIT.

Рис. 27 - Елемент, що відповідає за обробку

3.3.7.3 Компонування сцени. Головна сцена містить такі ключові об’єкти(рис. 28):

- XR Origin — скрипт, що управляє положенням контроллерів та їх параметрами ;
- ToolManager — головний скрипт, що управляє інструментами та об’єктами на сцені;
- Brush, Eraser, Selector — скрипти для управління відповідними інструментами;
- MainMenu — скрипт головного меню;

- MiniMenu — скрипт для міні меню параметрів. Працює для обох контролерів;
- HintManager — система відображення підказок на контролерах;
- SaveManager — окрема сутність для роботи з файлами.

Усі ці компоненти взаємодіють між собою через публічні методи.

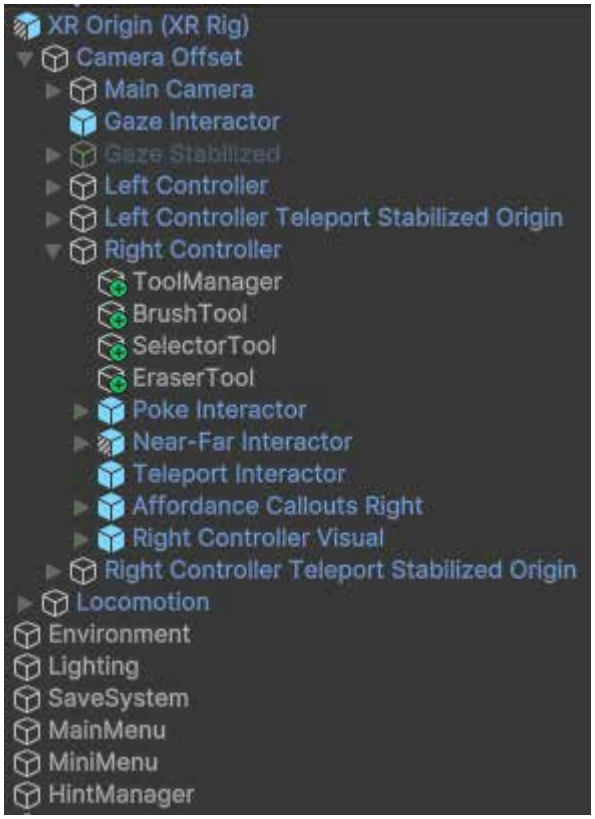

Рис. 28 - Фінальна структура проєкту в юніті

Усі вищеназвані об’єкти мають свої скрипти, завдяки яким функціонують. Інші об’єкти які можна побачити на зображенні(див. рис. 29) надані середовищем Unity за замовчуванням і не були окремо реалізовані.

3.3.7.4 Інтерфейс. Інтерфейс реалізовано через Canvas у режимі World Space, щоб він міг бути прикріплений до сцени(рис. 29). В окремих випадках використано XR UI Interaction, щоб забезпечити натискання кнопок меню лазером контролера.

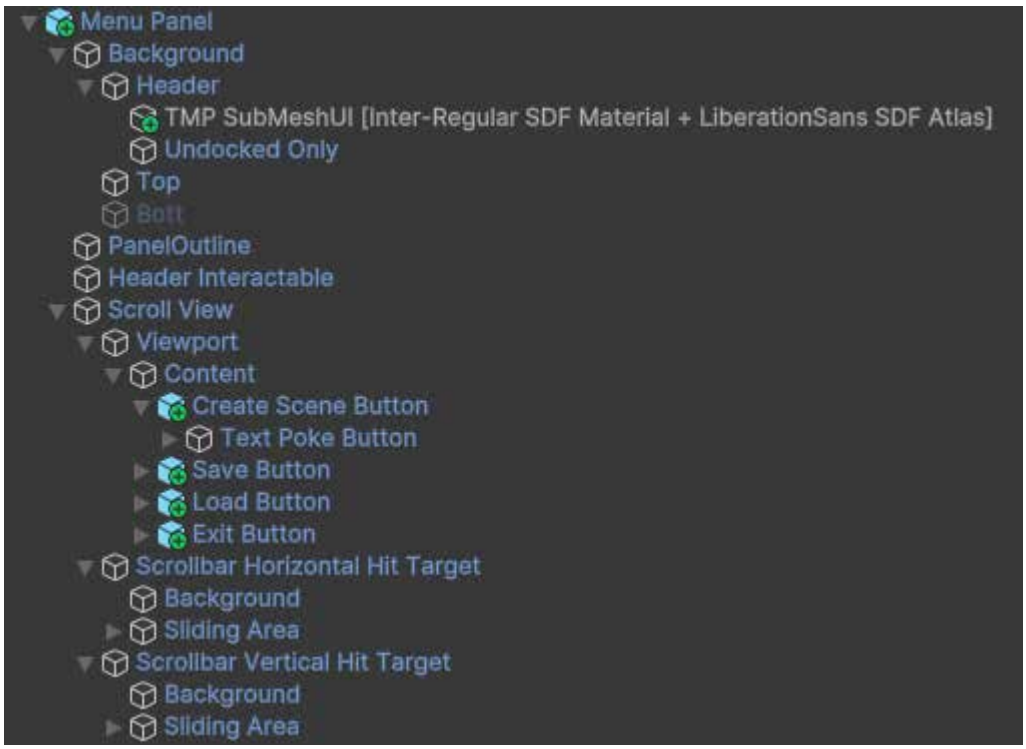

Рис. 29 - Елементи меню в менеджері

3.3.7.5 Білд і запуск.

Unity дозволяє створювати збірки для Windows x86_64 з підтримкою UWP (Universal Windows Platform) або стандартної десктопної версії. Для тестування застосунку використовувався режим Play з підключеною гарнітурою, що дозволяло швидко перевіряти всі функції без повного білду.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

4.1.1 Визначення користувачів. Для оцінки ефективності та зручності створеного прикладного програмного забезпечення було проведено аналіз потенційних категорій користувачів. На основі цього аналізу були сформульовані основні групи користувачів, їх потреби, очікування та сценарії використання програми.

4.1.1.1 Професійні дизайнери та художники. Ця категорія користувачів включає спеціалістів у сфері 3D-графіки, концепт-арту та прототипування. Вони мають досвід роботи з такими програмами, як Blender, ZBrush, Maya та вже знайомі з технологіями віртуальної реальності.

Основні цілі:

- Створення концепт-артів та 3D-моделей для подальшої обробки.
- Швидке створення прототипів у VR-середовищі.

Очікування:

- Простота та швидкість малювання.
- Можливість збереження моделей для подальшого перегляду та редагування.

4.1.1.2 Студенти творчих спеціальностей. До цієї групи належать користувачі, які навчаються в художніх школах, університетах та інших закладах

освіти, пов’язаних з дизайном, мистецтвом та 3D-моделюванням. Їхні знання можуть варіюватися від базового до середнього рівня.

Основні цілі:

- Освоєння основ 3D-малювання у віртуальному середовищі.
- Виконання навчальних або експериментальних завдань.
- Застосування VR як інструменту для вивчення нових методів роботи.

Очікування:

- Інтуїтивно зрозумілий інтерфейс.
- Візуальні підказки та допомога в роботі з інструментами.
- Простота у встановленні та запуску програми.

4.1.1.3 Любителі та ентузіасти VR-мистецтва. Це користувачі, які займаються VR-творчістю на аматорському рівні. Вони використовують програмне забезпечення переважно як хобі або інструмент для самовираження.

Основні цілі:

- Розвага та творчі експерименти у VR.
- Створення простих 3D-малюнків та візуалізацій.
- Покращення особистих навичок у VR.

Очікування:

- Зручний та доступний інтерфейс.
- Можливість самостійного встановлення та використання без потреби у професійних знаннях.

4.1.1.4 Узагальнення вимог користувачів.

Таблиця 4.1

Критерій	Професіонали	Студенти	Любителі
Рівень підготовки	Високий	Початковий – середній	Базовий
Основна мета	Концепт-арт, прототип	Навчання, експерименти	Розвага, хобі
Очікування	Швидкість, функціонал	Простота, підказки	Зручність, доступність
Інструменти	Потужний ПК, VR	Середній ПК, VR	Домашній ПК, VR

Таким чином, розроблене програмне забезпечення було орієнтоване на покриття потреб усіх зазначених категорій користувачів. Під час тестування враховувалися можливі варіанти використання, рівень підготовки та очікування щодо роботи програми.

4.1.2 Тестування окремих частин програми. Для забезпечення надійної та стабільної роботи програмного забезпечення було проведено модульне тестування основних функціональних компонентів. Це дозволило виявити та усунути потенційні помилки на ранніх етапах розробки. Нижче наведено короткий опис протестованих частин системи.

4.1.2.1 Обробка вводів з VR-контролерів. Було перевірено стабільність розпізнавання натискань кнопок та рухів, що надходять з контролерів. Система коректно інтерпретує всі основні дії: вибір інструментів, малювання тощо.

4.1.2.2 Інтерфейс інструментів. Перевірялась робота панелі інструментів у VR-просторі, зокрема: коректне відображення панелей меню, робота кнопок

меню, активація відпонідних подій при взаємодії з меню. А також коректне відображення підказок. Тестування показало коректну зміну стану інтерфейсу залежно від обраного інструмента та активацію відповідних подій.

4.1.2.3 Система збереження проєкту. Було протестовано збереження і завантаження створених 3D-малюнків у файлову систему. Моделі коректно зберігаються у вигляді локальних файлів, які потім можна повторно завантажити для редагування та перегляду.

4.1.2.4 Візуалізація у VR. Окремо перевірено якість відображення 3D-малюнків у VR-просторі. Оцінювалася плавність переміщення віртуальної камери, коректність відображення об’єктів при великій кількості елементів на сцені.

4.1.3 Відкрите тестування. На завершальному етапі розробки було проведено відкрите тестування системи за участю трьох користувачів: автора проєкту та двох незалежних тестувальників. Метою тестування було перевірити стабільність роботи, зручність використання, а також загальне враження від взаємодії з VR-додатком.

4.1.3.1 Умови тестування. Тестування проводилось у реальних умовах, кожен учасник працював зі своєю гарнітурою та комп’ютером. Користувачі отримали фінальну збірку програми, а також короткий файл README з інструкцією щодо встановлення та використання(додаток В). Жоден з тестувальників, окрім автора, не мав досвіду роботи з цим програмним забезпеченням раніше.

Кожному тестувальнику було запропоновано виконати наступні дії:

- самостійно встановити програму;
- обрати інструменти для малювання та створити просту 3D-композицію;
- скористатись функцією збереження та перегляду створеного малюнка;
- оцінити інтерфейс, зручність використання та стабільність програми.

Усі тестувальники використовували VR-гарнітури з контролерами, а саме WMR.

4.1.3.2 Результати тестування. Результати показали, що всі користувачі змогли пройти повний сценарій без серйозних труднощів. Програма стабільно працювала на всіх тестових конфігураціях. Основні дії — вибір інструментів, створення об’єктів, збереження та завантаження сцени — виконувалися коректно.

Перший тестувальник відзначив інтуїтивність інтерфейсу, а також зручність використання базових інструментів для швидкого створення концептів. Його 3D-малюнок(рис. 30) був готовий менш ніж за 10 хвилин, без потреби звертатись до документації.

Рис. 30 - Скріншот сцени першого тестувальника

Другий тестувальник(рис. 31) підкреслив простоту у використанні для новачків. Він також скористався візуальними підказками, що значно полегшило процес знайомства з інструментами. Збереження та завантаження сцени пройшло без помилок, VR-перегляд працював плавно.

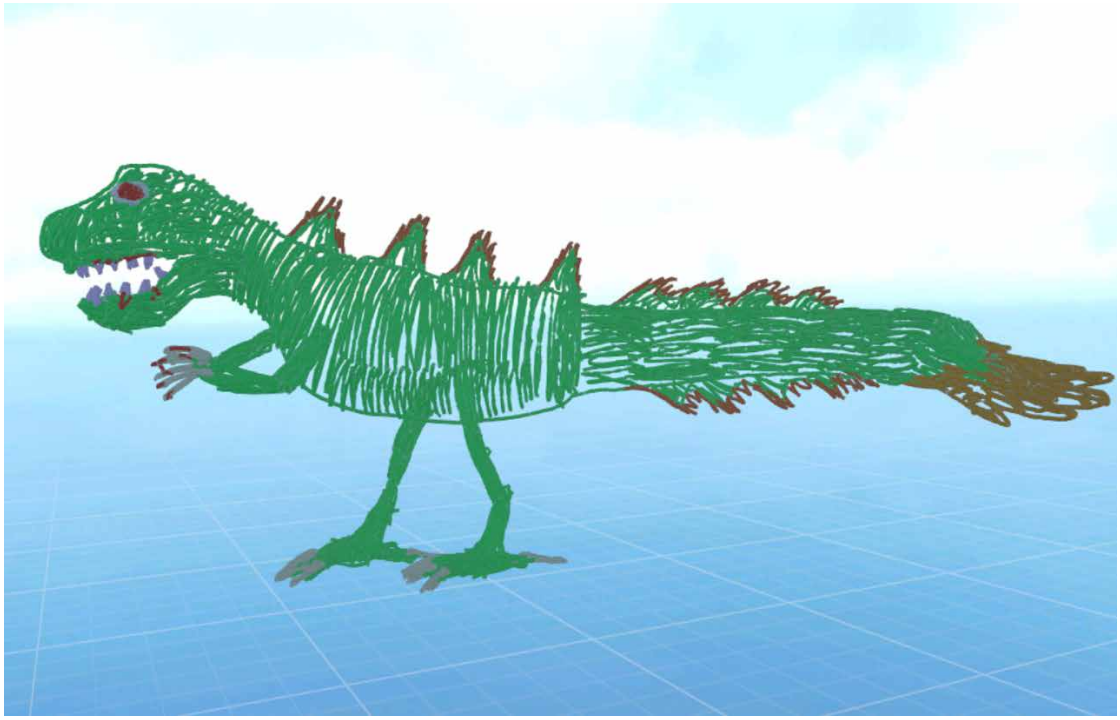

Рис. 31 - Скріншот сцени другого тестувальника

Автор системи провів серію стрес-тестів, додавши до сцени велику кількість об’єктів(рис. 32). Зображена сцена була продубльована 100 разів(накладалася одна на одну). Програма зберігала стабільність навіть у навантаженому стані, ніяких зависань не було помічено. Сучасні комп’ютери достатньо потужні, щоб обробити навіть мільйони об’єктів. Тому проблеми можуть з’явитися тільки у випадку, якщо користувач буде завжди малювати тільки на одній сцені, що мало вірогідно.

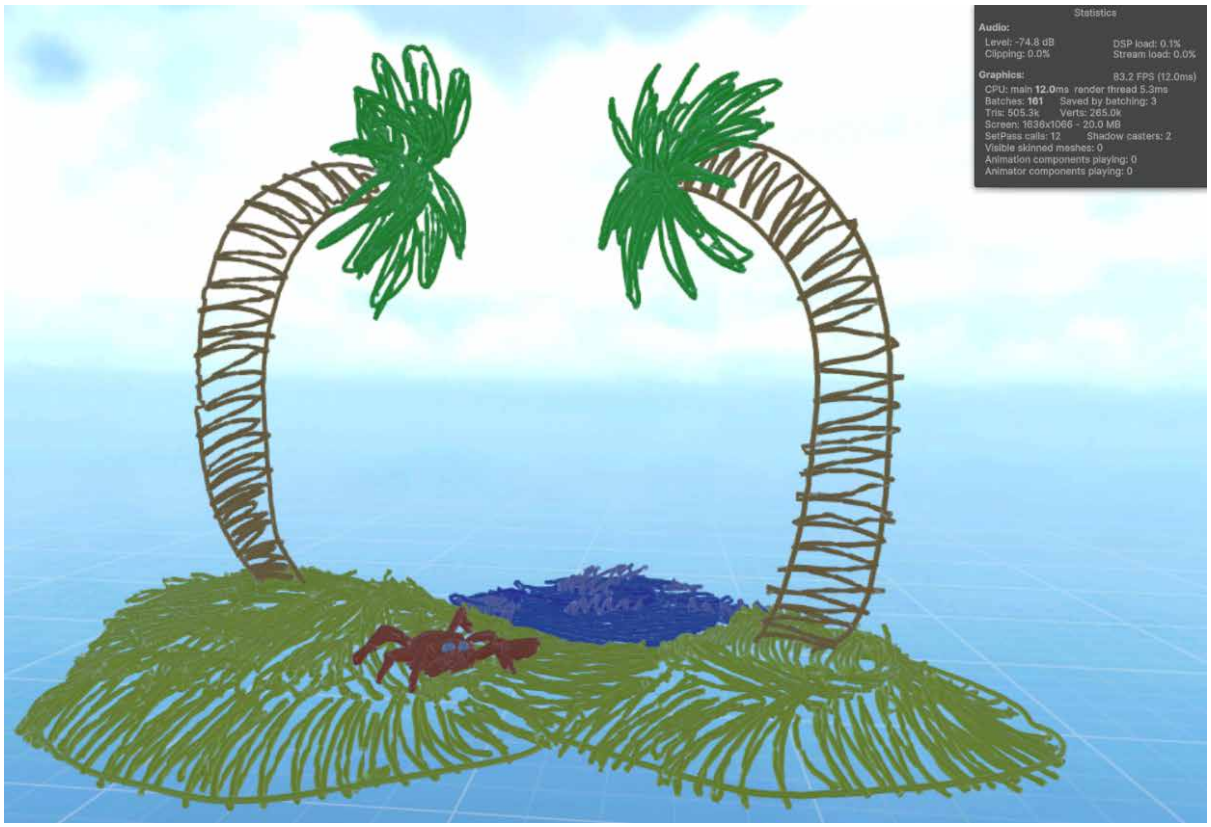

Рис. 32 - Скріншот сцени та кількості кадрів в секунду програми автора

Жодних фатальних помилок або аварійного завершення роботи програми зафіксовано не було. Усі створені об’єкти успішно зберігалися та відкривалися повторно. Програма працювала плавно у всіх тестувальників.

4.1.3.3 Характеристики комп’ютерів тестувальників. Результати показали, що всі користувачі змогли пройти повний сценарій без серйозних труднощів. Програма стабільно працювала на всіх тестових конфігураціях.

Основні дії — вибір інструментів, створення об’єктів, збереження та завантаження сцени — виконувалися коректно.

Таблиця 4.2

Порівняння характеристик комп’ютерів

№	Користувач	ОС	Процесор	Відеокарта	RAM
1	незалежний тестувальник	Windows 11	Ryzen 5 9600	NVIDIA RTX 4070	32 ГБ
2	незалежний тестувальник	Windows 11	Intel Core i5 7600	NVIDIA RTX 1060	16 ГБ
3	Автор	Windows 10	Ryzen 5 3600	AMD RX 5600 XT	32 ГБ

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної роботи VR-застосунку для 3D-малювання користувачу необхідно мати відповідне апаратне та програмне забезпечення.

1. Мінімальні вимоги до апаратного забезпечення

- Процесор: Intel Core i5-5500 / AMD Ryzen 5 1600;
- Відеокарта: NVIDIA RTX 1060 / AMD RX 580
- Операційна пам’ять: 8 GB;
- Постійна пам’ять: 5 GB;
- VR гарнітура: Windows Mixed Reality шолом та контролери;
- USB: USB 3.0 порт для VR шолома;
- Display: HDMI 1.4;
- Bluethooth: версія 4.1.

2. Вимоги до програмного забезпечення

- Операційна система: Windows 10 або 11 (64-bit);
- Платформа запуску: Windows Mixed Reality Portal;
- GPU драйвери: рекомендовані останні NVIDIA / AMD драйвери відеокарт.

Забезпечення відповідності апаратних та програмних вимог є критично важливим для коректної роботи VR-застосунку. Наявність сумісної VR-гарнітури та сучасного графічного адаптера дозволяє гарантувати плавне відображення сцен, стабільну роботу інструментів малювання та комфортну взаємодію користувача з тривимірним простором.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення для 3D-малювання у віртуальній реальності призначений для забезпечення швидкого та зручного розгортання VR-додатку на комп’ютері кінцевого користувача. У пакет включено всі необхідні компоненти для запуску програми без додаткових налаштувань.

Основні компоненти інсталяційного пакету:

1. Виконуваний файл програми

- VArt.exe — основний файл запуску VR-додатку;
- Створений за допомогою Unity, зібраний у форматі Standalone Build (Windows x86_64).

2. Папка з бібліотеками

- /VArt_Data/ — службові дані Unity (ресурси, сцени, шейдери);
- /Plugins/ — зовнішні DLL (наприклад, XR SDK, OpenXR підтримка);
- /MonoBleedingEdge/ — бібліотеки .NET середовища, необхідні для роботи скриптів.

3. Конфігураційні файли

- config.json — збереження налаштувань користувача (вибрані інструменти, колір, остання сцена).

4. Файли ресурсів

- /Assets/BrushPresets/ — збережені параметри пензлів та інструментів;
- /Textures/ — текстури або ефекти мазків.

5. Файли збереження

- /Saves/ — папка, де автоматично зберігаються результати малювання у форматі .json.

6. Інсталяційний скрипт

- setup.exe (згенерований через Inno Setup / NSIS / Unity Build Installer);

7. Файл документації

- README.txt — короткий посібник користувача;
- Інструкція зі встановлення, запуску, підтримуваних гарнітур та клавіш керування.

ВИСНОВКИ

У результаті виконання дипломної роботи було створено функціональний прототип VR-додатку VArt для 3D-малювання у віртуальному просторі, який реалізує весь заявлений у вступі функціонал. Під час роботи були виконані наступні етапи: аналіз існуючих рішень, проєктування архітектури, розробку інтерфейсу та основних модулів, комплексне тестування.

Основні виконані завдання та досягнуті результати:

1 Аналіз існуючих рішень і вибір технологій

- Проведено огляд популярних VR-додатків для 3D-малювання (Tilt Brush, Medium тощо).
- Обґрунтовано вибір Unity та XRIT як основи проєкту, що забезпечило гнучкість у роботі з контролерами Windows Mixed Reality.

2 Проєктування архітектури системи

- Розроблено модульну архітектуру із чітким розділенням на компоненти: обробка вводів, рендеринг кривих, інтерфейсні елементи та збереження даних.
- Створено UML-діаграми класів і послідовностей для всіх ключових сценаріїв використання.

3 Реалізація інструментів малювання

- Модулі Brush, Eraser та Selector забезпечують малювання об’ємних кривих із динамічними налаштуваннями параметрів (колір, товщина) і видалення та редагування вже створених елементів у режимі реального часу.
- Впроваджено систему генерації кривих для плавного відображення з оптимізацією під VR.

4 Інтуїтивний VR-інтерфейс

- Розроблено головне меню у World Space із raycast-керуванням та контекстні міні-панелі для налаштування параметрів інструментів.
- Додано систему візуальних підказок і підсвічування активних елементів, що знижує поріг входження новачків.

5 Збереження та відновлення сеансів малювання

- Реалізовано JSON-серіалізацію об’єктів BrushStroke та механізм автозбереження.
- Забезпечено резервне копіювання та відновлення даних із локального сховища, що гарантує безперервність робочого процесу.

6 Тестування та верифікація

- Модульне тестування кожного компонента.
- Інтеграційне тестування основних сценаріїв (створення, редагування, збереження/завантаження).
- Відкрите тестування за участі трьох користувачів із різними ПК: підтверджено стабільну роботу понад 40 FPS при навантаженні великою кількістю об’єктів, без артефактів та падінь.

Всі поставлені задачі виконані в повному обсязі: від аналітичної частини до результатів експлуатаційного тестування. Розроблений програмний продукт демонструє обрану архітектуру та технології, як ефективні та достатньо продуктивні для подальшої еволюції продукту.

Таким чином, робота підтвердила доцільність використання Unity та XRIT для створення VR-додатків у сфері 3D-малювання, а також надала готовий базовий функціонал для подальшої інтеграції нових можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google LLC. Tilt Brush [Електронний ресурс]. – Посилання для доступу: https://www.tiltbrush.com – Дата звернення: 20.05.2025.

2. Oculus VR, LLC. Medium [Електронний ресурс]. – Посилання для доступу: https://www.oculus.com/medium/ – Дата звернення: 20.05.2025.

3. Unity Technologies. Unity User Manual [Електронний ресурс]. – Посилання для доступу: https://docs.unity3d.com/Manual/index.html – Дата звернення: 20.05.2025.

4. Unity Technologies. XR Interaction Toolkit Documentation [Електронний ресурс]. – Посилання для доступу: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/index.html – Дата звернення: 20.05.2025.

5. Newtonsoft. Json.NET – High-performance JSON framework for .NET [Електронний ресурс]. – Посилання для доступу: https://www.newtonsoft.com/json – Дата звернення: 20.05.2025.

6. Microsoft Corporation. Mixed Reality Input 213: Motion controllers [Електронний ресурс]. – Посилання для доступу: https://learn.microsoft.com/windows/mixed-reality/deprecated/mixed-reality-213 – Дата звернення: 20.05.2025.

7. Андерсон К. "Розробка VR застосунків на платформі Unity", 2023. Дата звернення: 20.05.2025.

8. Pham, S., Nguyen, T., & Lee, J. Real-Time 3D Sketching in Virtual Reality: Techniques and Applications // Proceedings of the 2021 ACM Symposium on Virtual Reality Software and Technology. – New York: ACM, 2021. – P. 45–54.

ДОДАТОК А

Управління за допомогою контролера

Тригер - активує основну дію вибраного інструмента(малювання, виділення, видалення)

Грип - використовується в комбінації з іншими кнопками

Меню - відкриває/закриває головне меню

Джойстик - переміщення у віртуальному просторі. При натисканні робить знімок екрану

Тачпад - при натисканні в центрі викликає меню інструмента(для селектора відкриває/закриває параметри виділених об’єктів). Інші дії різняться в залежності від обраного інструмента:

- Пензель. При дотику по всій поверхні, через 1 секунду контакту змінює колір на обраний за допомогою колеса кольору. При натисканні вверх/вниз змінює розмір більше/менше. При натисканні вліво/вправо змінює форму.
- Селектор. При дотику по всій поверхні, через 1 секунду контакту змінює колір виділеного об’єкту на обраний за допомогою колеса кольору. При натисканні вверх/вниз змінює розмір виділеного об’єкта більше/менше. При натисканні вліво/вправо змінює форму виділеного об’єкта.
- Гумка. При натисканні вверх/вниз змінює розмір сфери видалення.

При затиснутому Грипі - натискання на тачпад будуть змінювати інструмент.

- Ліво - гумка
- Право - селектор
- Низ - пензель

ДОДАТОК Б

Код скриптів, що використовуються в програмі

ToolManager.cs

```
public class ToolManager : MonoBehaviour
{
    public Tool[] tools;

    private Tool currentTool;

    public GameObject handMenuCanvas;

    private bool gripPressedLeft = false;
    private bool gripPressedRight = false;

    public void SetTool(ToolType type)
    {
        foreach (Tool tool in tools)
        {
            if (tool.Type == type)
            {
                if (currentTool != null) currentTool.Deactivate();
                currentTool = tool;
                currentTool.Activate();

                Debug.Log("Switched to: " + type);
                break;
            }
        }
    }
```

```
    void Start()
    {
        SetTool(ToolType.Brush);
    }

    private void InteractionSourcePressed(InteractionSourcePressedEventArgs obj)
    {
        if (obj.pressType == InteractionSourcePressType.Menu)
            MenuManager.menuVisible = !MenuManager.menuVisible;
        if (MenuManager.menuVisible) return;

if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.pressType == InteractionSourcePressType.Select)
            currentTool.TriggerPressed();
else if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.pressType == InteractionSourcePressType.Grasp)
            gripPressedRight = true;
else if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.state.touchpadTouched && !gripPressedRight)
            currentTool.TouchPad(obj.state.touchpadPosition);
else if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.pressType == InteractionSourcePressType.Touchpad && gripPressedRight)
        {
            Vector2 touchpadPosition = obj.state.touchpadPosition;
            if (touchpadPosition.magnitude > 0.5f)
            {
                ToolType selected = currentTool.Type;

                if (touchpadPosition.y > 0.5f)
                    selected = ToolType.Brush;
```

```
            else if (touchpadPosition.x < -0.5f)
                selected = ToolType.Eraser;
            else if (touchpadPosition.x > 0.5f)
                selected = ToolType.Selector;

            if (selected != currentTool.Type)
                SetTool(selected);
        }
    }
else if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.pressType == InteractionSourcePressType.Touchpad)
        currentTool.TouchPadPressed(obj.state.touchpadPosition);

if (obj.state.source.handedness == InteractionSourceHandedness.Left &&
obj.pressType == InteractionSourcePressType.Select)
        currentTool.TriggerPressed();
else if (obj.state.source.handedness == InteractionSourceHandedness.Left &&
obj.pressType == InteractionSourcePressType.Grasp)
        gripPressedLeft = true;
else if (obj.state.source.handedness == InteractionSourceHandedness.Left &&
obj.state.touchpadTouched && !gripPressedLeft)
        currentTool.TouchPad(obj.state.touchpadPosition);
else if (obj.state.source.handedness == InteractionSourceHandedness.Left &&
obj.pressType == InteractionSourcePressType.Touchpad && gripPressedLeft)
    {
        Vector2 touchpadPosition = obj.state.touchpadPosition;
        if (touchpadPosition.magnitude > 0.5f)
        {
            ToolType selected = currentTool.Type;
```

```
            if (touchpadPosition.y > 0.5f)
                selected = ToolType.Brush;
            else if (touchpadPosition.x < -0.5f)
                selected = ToolType.Eraser;
            else if (touchpadPosition.x > 0.5f)
                selected = ToolType.Selector;

            if (selected != currentTool.Type)
                SetTool(selected);
        }
    }
else if (obj.state.source.handedness == InteractionSourceHandedness.Left &&
obj.pressType == InteractionSourcePressType.Touchpad)
        currentTool.TouchPadPressed(obj.state.touchpadPosition);
  }

  private void InteractionSourceUpdated(InteractionSourceReleasedEventArgs obj)
  {
if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.state.touchpadTouched)
        currentTool.TouchPad(obj.state.touchpadPosition);
  }

  private void InteractionSourceReleased(InteractionSourceReleasedEventArgs obj)
  {
if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.pressType == InteractionSourcePressType.Select)
        currentTool.TriggerReleased();
else if (obj.state.source.handedness == InteractionSourceHandedness.Right &&
obj.pressType == InteractionSourcePressType.Grasp)
```

```
            gripPressedRight = false;

if (obj.state.source.handedness == InteractionSourceHandedness.Left &&
obj.pressType == InteractionSourcePressType.Select)
            currentTool.TriggerReleased();
else if (obj.state.source.handedness == InteractionSourceHandedness.Left &&
obj.pressType == InteractionSourcePressType.Grasp)
            gripPressedLeft = false;
    }
}
```

Tool.cs

```
public abstract class Tool : MonoBehaviour
{
    public abstract ToolType Type { get; }

    public virtual void Activate()
    {
        gameObject.SetActive(true);
    }

    public virtual void Deactivate()
    {
        gameObject.SetActive(false);
    }

    public virtual void TriggerPressed() { }
    public virtual void TriggerReleased() { }
    public virtual void TouchPad(Vector2 input) { }
    public virtual void TouchPadPressed(Vector2 input) { }
```

```
}
```

Eraser.cs

```
public class Eraser : Tool
{
    public override ToolType Type
    {
        get { return ToolType.Eraser; }
    }

    [Header("Eraser Settings")]
    public float eraseRadius = 0.1f;
    public Transform eraseOrigin;

    private bool isErasing = false;

    private void TryErase()
    {
        Collider[] hits = Physics.OverlapSphere(eraseOrigin.position, eraseRadius);

        foreach (Collider hit in hits)
            if (hit.CompareTag("BrushStroke"))
                Destroy(hit.gameObject);
    }


    public override void TriggerPressed()
    {
        TryErase();
    }
```

```
    public override void TouchPadPressed(Vector2 input)
    {
        if (input.magnitude > 0.5f)
        {
            if (input.y > 0.5f)
                eraseRadius += 0.05f;
            else if (input.y < -0.5f)
                eraseRadius -= 0.05f;
        }
        else if (input.magnitude <= 0.5f)
            MiniMenuManager.Open(this);
    }
}
```

Selector.cs

```
public class Selector : Tool
{
    public override ToolType Type
    {
        get { return ToolType.Selector; }
    }

    List<VRBrushStroke> strokes = new List<VRBrushStroke>();
    List<Renderer> renderers = new List<Renderer>();

    public float selectRadius = 0.1f;
    public Transform selectOrigin;

    private float touchPadTimer = 0f;
```

```
private float holdDuration = 1f;

private void TrySelect()
{
    Collider[] hits = Physics.OverlapSphere(selectOrigin.position, selectRadius);

    foreach (Collider hit in hits)
    {
        strokes.Add(hit.GetComponent<VRBrushStroke>());
        renderers.Add(hit.GetComponent<Renderer>());
    }

    if (hits.Length > 0)
        MiniMenuManager.Open(hits);
}
public override void TriggerPressed()
{
    TrySelect();
}

public override void TouchPad(Vector2 input)
{
    if (renderers == null) return;
    if (input.magnitude != 0)
    {
        touchPadTimer += Time.deltaTime;

        if (touchPadTimer >= holdDuration)
        {
            float angle = Mathf.Atan2(input.y, input.x);
```

```
            float hue = (angle / (2 * Mathf.PI) + 1f) % 1f;
            float saturation = Mathf.Clamp01(input.magnitude);
            float value = 1f;

            foreach (Renderer render in renderers)
                if (render != null)
                    render.material.color = Color.HSVToRGB(hue, saturation, value);
        }

    }
    else
        touchPadTimer = 0;
}
public override void TouchPadPressed(Vector2 input)
{
    if (strokes == null) return;
    if (input.magnitude > 0.5f)
    {
        foreach (VRBrushStroke stroke in strokes)
            if (input.y > 0.5f)
                stroke.radius += 0.05f;
            else if (input.y < -0.5f)
                stroke.radius -= 0.05f;
            else if (input.x > 0.5f)
                stroke.form = PrimitiveShape.GetNextPrimitiveType(stroke.form);
            else if (input.x < -0.5f)
                stroke.form = PrimitiveShape.GetPreviousPrimitiveType(stroke.form);
    }
    else if (input.magnitude <= 0.5f)
        MiniMenuManager.Open(this);
```

```
        touchPadTimer = 0;
    }
}
```

Brush.cs

```
public class Brush : Tool
{
    public override ToolType Type
    {
        get { return ToolType.Brush; }
    }

    [Header("Brush Settings")]
    public PrimitiveType form;

    [Tooltip("Size (diameter) of the brush stroke")]
    public float size = 0.05f;

    [Tooltip("Color of the brush stroke")]
    public Color color = Color.white;

    public GameObject strokePrefab;
    public Transform controllerTransform;

    private VRBrushStroke currentStroke;
    private bool isDrawing = false;

    private float touchPadTimer = 0f;
    private float holdDuration = 1f;
```

```
public override void TriggerPressed()
{
    if (!isDrawing)
    {
        StartStroke();
        isDrawing = true;
    }
}
public override void TriggerReleased()
{
    if (isDrawing)
    {
        currentStroke = null;
        isDrawing = false;
    }
}
public override void TouchPad(Vector2 input)
{
    if (input.magnitude != 0)
    {
        touchPadTimer += Time.deltaTime;

        if (touchPadTimer >= holdDuration)
        {
            float angle = Mathf.Atan2(input.y, input.x);
            float hue = (angle / (2 * Mathf.PI) + 1f) % 1f;
            float saturation = Mathf.Clamp01(input.magnitude);
            float value = 1f;
```

```
            color = Color.HSVToRGB(hue, saturation, value);
        }

    }
    else
        touchPadTimer = 0;
}
public override void TouchPadPressed(Vector2 input)
{
    if (input.magnitude > 0.5f)
    {
        if (input.y > 0.5f)
            size += 0.05f;
        else if (input.y < -0.5f)
            size -= 0.05f;
        else if (input.x > 0.5f)
            form = PrimitiveShape.GetNextPrimitiveType(form);
        else if (input.x < -0.5f)
            form = PrimitiveShape.GetPreviousPrimitiveType(form);
    }
    else if (input.magnitude <= 0.5f)
        MiniMenuManager.Open(this);

    touchPadTimer = 0;
}

private void StartStroke()
{
GameObject strokeObj = Instantiate(strokePrefab, Vector3.zero,
Quaternion.identity);
```

```
        currentStroke = strokeObj.GetComponent<VRBrushStroke>();
        currentStroke.form = form;
        currentStroke.radius = size;
        strokeObj.transform.position = controllerTransform.position;
        strokeObj.transform.rotation = controllerTransform.rotation;
        strokeObj.tag = "BrushStroke";
        Renderer rend = strokeObj.GetComponent<Renderer>();
        if (rend != null)
            rend.material.color = color;

    }

    private void Update()
    {
        if (isDrawing && currentStroke != null)
            currentStroke.AddPoint(controllerTransform.position);
    }

    public override void Deactivate()
    {
        base.Deactivate();
        currentStroke = null;
    }
}
```

VRBrushStroke.cs

```
[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class VRBrushStroke : MonoBehaviour
{
    public float radius = 0.01f;
```

```
    public int radialSegments = 6;
    public PrimitiveType form;

    private List<Vector3> points = new List<Vector3>();
    private Mesh mesh;

    private void Awake()
    {
        mesh = new Mesh();
        GetComponent<MeshFilter>().mesh = mesh;
    }

    public void AddPoint(Vector3 position)
    {
if (points.Count == 0 || Vector3.Distance(points[points.Count - 1], position) > 0.01f)
        {
            points.Add(transform.InverseTransformPoint(position));
            UpdateMesh();
        }
    }

    public void UpdateMesh()
    {
        if (points.Count < 2) return;

        int ringCount = points.Count;
        int vertsPerRing = radialSegments;
        List<Vector3> vertices = new List<Vector3>();
        List<int> triangles = new List<int>();
```

```
        for (int i = 0; i < ringCount; i++)
        {
            Vector3 center = points[i];
            Vector3 forward;
            if (i < ringCount - 1)
                forward = points[i + 1] - points[i];
            else
                forward = points[i] - points[i - 1];

            Vector3 up = Vector3.up;
            if (Vector3.Cross(forward, up).sqrMagnitude < 0.001f)
                up = Vector3.forward;

            Vector3 right = Vector3.Cross(forward, up).normalized;
            up = Vector3.Cross(right, forward).normalized;

List<Vector3> section = PrimitiveShape.GenerateSection(form, vertsPerRing, radius, right, up);
            foreach (var offset in section)
            {
                vertices.Add(center + offset);
            }
        }

        for (int i = 0; i < ringCount - 1; i++)
        {
            for (int j = 0; j < vertsPerRing; j++)
            {
                int current = i * vertsPerRing + j;
```

```
            int next = (i + 1) * vertsPerRing + j;
            int currentNext = i * vertsPerRing + (j + 1) % vertsPerRing;
            int nextNext = (i + 1) * vertsPerRing + (j + 1) % vertsPerRing;

            triangles.Add(current);
            triangles.Add(next);
            triangles.Add(currentNext);

            triangles.Add(currentNext);
            triangles.Add(next);
            triangles.Add(nextNext);
        }
    }

    mesh.Clear();
    mesh.SetVertices(vertices);
    mesh.SetTriangles(triangles, 0);
    mesh.RecalculateNormals();
    mesh.RecalculateBounds();

    MeshCollider collider = GetComponent<MeshCollider>();
    if (collider == null)
        collider = gameObject.AddComponent<MeshCollider>();

    collider.convex = true;
    collider.sharedMesh = null;
    collider.sharedMesh = mesh;
}

public BrushStrokeData ToData()
```

```
    {
        var data = new BrushStrokeData();
        data.points = new List<Vector3>(points);
        data.position = transform.position;
        data.rotation = transform.rotation;
        data.scale = transform.localScale;
        data.color = GetComponent<MeshRenderer>().material.color; return data;
    }

    public void SetPoints(List<Vector3> newPoints)
    {
        points = new List<Vector3>(newPoints);
    }

}
```

Primitive.cs

```
public enum PrimitiveType
{
    Circle,
    Square,
    Triangle,
    Star
}

public static class PrimitiveShape
{
public static List<Vector3> GenerateSection(PrimitiveType type, int segments, float radius, Vector3 right, Vector3 up)
    {
```

```
List<Vector3> section = new List<Vector3>();

switch (type)
{
    case PrimitiveType.Circle:
        for (int i = 0; i < segments; i++)
        {
            float angle = 2 * Mathf.PI * i / segments;
            Vector3 offset = Mathf.Cos(angle) * right + Mathf.Sin(angle) * up;
            section.Add(offset.normalized * radius);
        }
        break;

    case PrimitiveType.Square:
        for (int i = 0; i < segments; i++)
        {
            float t = (float)i / segments;
            float x = 0f, y = 0f;
            if (t < 0.25f)
            {
                x = Mathf.Lerp(-1, 1, t / 0.25f);
                y = -1;
            }
            else if (t < 0.5f)
            {
                x = 1;
                y = Mathf.Lerp(-1, 1, (t - 0.25f) / 0.25f);
            }
            else if (t < 0.75f)
            {
```

```
            x = Mathf.Lerp(1, -1, (t - 0.5f) / 0.25f);
            y = 1;
        }
        else
        {
            x = -1;
            y = Mathf.Lerp(1, -1, (t - 0.75f) / 0.25f);
        }
        section.Add((x * right + y * up).normalized * radius);
    }
    break;

case PrimitiveType.Triangle:
    for (int i = 0; i < segments; i++)
    {
        float t = (float)i / segments;
        float angle = 2 * Mathf.PI * t;
        float adjusted = Mathf.Floor(t * 3) / 3;
        angle = 2 * Mathf.PI * adjusted;
        Vector3 offset = Mathf.Cos(angle) * right + Mathf.Sin(angle) * up;
        section.Add(offset.normalized * radius);
    }
    break;

case PrimitiveType.Star:
    for (int i = 0; i < segments; i++)
    {
        float angle = 2 * Mathf.PI * i / segments;
        float r = (i % 2 == 0) ? radius : radius * 0.5f;
        Vector3 offset = Mathf.Cos(angle) * right + Mathf.Sin(angle) * up;
```

```
                section.Add(offset.normalized * r);
            }
            break;
        }

        return section;
    }
public static PrimitiveType GetNextPrimitiveType(PrimitiveType current)
{
    PrimitiveType[] types = (PrimitiveType[])Enum.GetValues(typeof(PrimitiveType));
    int index = Array.IndexOf(types, current);
    index = (index + 1) % types.Length;
    return types[index];
}
public static PrimitiveType GetPreviousPrimitiveType(PrimitiveType current)
{
    PrimitiveType[] types = (PrimitiveType[])Enum.GetValues(typeof(PrimitiveType));
    int index = Array.IndexOf(types, current);
    index = (index - 1 + types.Length) % types.Length;
    return types[index];
}

}
```

ToolType.cs

```
public enum ToolType
{
    Brush,
    Eraser,
    Selector
```

```
}
```

MenuManager.cs

```
public class MenuManager : MonoBehaviour
{
    public static GameObject menuCanvas;

    public static bool menuVisible = false;

    private void Update()
    {
menuCanvas.transform.position = Camera.main.transform.position +
Camera.main.transform.forward * 1.5f;
        menuCanvas.SetActive(menuVisible);
    }
}
```

SaveManager.cs

```
public class SaveManager : MonoBehaviour
{
    private string savePath;

    private void Awake()
    {
        savePath = Application.persistentDataPath + "/drawing.json";
    }

    public void SaveAll()
    {
        var allStrokes = GameObject.FindGameObjectsWithTag("BrushStroke");
```

```
    var saveFile = new SaveFile();

    foreach (var obj in allStrokes)
    {
        var stroke = obj.GetComponent<VRBrushStroke>();
        if (stroke != null)
            saveFile.strokes.Add(stroke.ToData());
    }

    string json = JsonUtility.ToJson(saveFile, true);
    File.WriteAllText(savePath, json);
    Debug.Log("Saved to: " + savePath);
}

public void LoadAll(GameObject strokePrefab)
{
    if (!File.Exists(savePath))
    {
        Debug.LogWarning("Save file not found");
        return;
    }

    string json = File.ReadAllText(savePath);
    var saveFile = JsonUtility.FromJson<SaveFile>(json);

    foreach (var data in saveFile.strokes)
    {
        GameObject obj = Instantiate(strokePrefab);
        obj.tag = "BrushStroke";
```

```
            obj.transform.position = data.position;
            obj.transform.rotation = data.rotation;
            obj.transform.localScale = data.scale;

            var stroke = obj.GetComponent<VRBrushStroke>();
            if (stroke != null)
            {
                stroke.SetPoints(data.points);
                stroke.UpdateMesh();
                obj.GetComponent<MeshRenderer>().material.color = data.color;
            }
        }

        Debug.Log("Loaded from: " + savePath);
    }
}
```

SaveFile.cs

```
[Serializable]
public class SaveFile
{
    public List<BrushStrokeData> strokes = new List<BrushStrokeData>();
}
```

ДОДАТОК В

Текст записки README, що додається до програми

VArt — це програмне забезпечення для тривимірного малювання у середовищі віртуальної реальності, розроблений у рамках дипломної роботи. Програма дозволяє користувачеві малювати об’ємні криві за допомогою контролерів Windows Mixed Reality, змінювати їх параметри (колір, товщину), зберігати результат і взаємодіяти з інтерфейсом у VR-середовищі.

Основні функції:

- Малювання 3D-кривих у віртуальному просторі
- Інструменти: пензель (Brush), гумка (Eraser), селектор (Selector)
- Інтуїтивний інтерфейс користувача з raycast-навігацією
- Візуальні підказки для зручного керування
- Збереження і завантаження проєктів у форматі JSON
- Підтримка Windows Mixed Reality гарнітур

Вимоги до системи:

- Windows 10/11
- Гарнітура WMR
- Підключені контролери руху
- GPU з підтримкою VR

Установка і запуск:

- Завантажити установник.
- Встановити програму слідуючи вказівкам в установнику.
- Увімкнути гарнітуру WMR та переконатися, що вона підключена.
- Запустити файл VArt.exe.

Файли проєктів зберігаються автоматично у папці з програмою.
/VArt/Saves/[your_save_files]