

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ІНТЕРАКТИВНА ПЛАТФОРМА ВІРТУАЛЬНОЇ
РЕАЛЬНОСТІ ДЛЯ ВІЗУАЛІЗАЦІЇ, НАВЧАННЯ ТА ВЗАЄМОДІЇ З
АВТОМОБІЛЬНИМИ СИСТЕМАМИ
INTERACTIVE VIRTUAL REALITY PLATFORM FOR VISUALIZATION,
TRAINING AND INTERACTION WITH AUTOMOTIVE SYSTEMS

Виконав: студент 4 курсу, групи КНТ-221
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ЄНАЛЬСВ Б.Р

(ПРИЗВИЩЕ та ініціали)

Керівник ОЛІЙНИК А.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СНАЛЬЄВА Богдана Романовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Інтерактивна платформа віртуальної реальності для візуалізації, навчання та взаємодії з автомобільними системами
Interactive Virtual Reality Platform for Visualization, Training and Interaction with Automotive Systems

керівник проєкту (роботи) д.т.н., професор, ОЛІЙНИК Андрій Олександрович

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) _____

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) _____

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ОЛІЙНИК А.О., професор		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “28” квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Богдан ЄНАЛЬСВ
 (підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Андрій ОЛІЙНИК
 (підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
101 с., 27 рис., 3 табл., 3 дод., 18 джерел.

ВІРТУАЛЬНА РЕАЛЬНІСТЬ, ІМЕРСИВНИЙ ДОСВІД,
ІНТЕРАКТИВНЕ НАВЧАННЯ, МЕХАНІК, ФІЗИЧНА СИМУЛЯЦІЯ, OPEN XR

Об'єкт дослідження – процес розробки програмних засобів віртуальної реальності.

Предмет дослідження – програмні засоби реалізації VR-середовища для навчання, тренування та візуалізації процесу розбору та збирання автомобіля.

Мета роботи – розробка інтерактивного VR-середовища, яке забезпечує реалістичний і зручний процес дослідження будь яких інженерних проектів. Ефективний і дешевий спосіб ознайомлення з інноваціями та складними приладами.

Матеріали, методи та технічні засоби: програмне забезпечення – Unity 2022.3.51f1, OpenXR, Unity XR Interaction Toolkit; обладнання – Oculus Meta Quest 2, VR-контролери.

Практична цінність полягає у створенні можливості використання VR для навчання механіків без потреби у фізичних автомобілях.

Галузь використання – механіки та інженери, автосервіси, університети.

ABSTRACT

Explanatory note to the bachelor's qualification thesis: 101 pages, 27 figures, 3 tables, 3 appendices, 18 references.

VIRTUAL REALITY, IMMERSIVE EXPERIENCE, INTERACTIVE TRAINING, MECHANIC, PHYSICAL SIMULATION, OPEN XR

Object of the research – the process of developing virtual reality software tools.

Subject of the research – software implementation of a VR environment for training, education, and visualization of the disassembly and assembly processes of a car, as well as optimization of user interaction with car parts in a virtual space.

Purpose of the work – to develop an interactive VR environment that provides a realistic and convenient process for exploring any engineering projects. An effective and low-cost method for familiarization with innovations and complex devices.

Materials, methods, and technical tools: software – Unity 2022.3.51f1, OpenXR, Unity XR Interaction Toolkit; hardware – Oculus Meta Quest 2, VR controllers.

Practical value lies in the ability to use VR for mechanic training without the need for physical vehicles.

Field of application – mechanics and engineers, auto repair services, universities.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
ВСТУП	9
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Віртуальна реальність у технічному навчанні та симуляції	10
1.2 Огляд існуючих рішень	11
1.3 Висновок за розділом	14
2. МАТЕРІАЛИ І МЕТОДИ	16
2.1 Вибір рушія	16
2.2 Вибір мови програмування	18
2.3 Вибір середовища розробки	21
2.4 Висновок за розділом 2	22
3. ОПИС ПРОГРАМИ	24
3.1 Структура програмного забезпечення	24
3.2 Особливості реалізації програмного забезпечення	29
3.3 Висновок до розділу 3	33
4. ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	35
4.1 Призначення	35
4.2 Умови застосування	35
4.3 Можливості та обмеження	36
4.4 Приклади роботи користувача з системою	36
4.5 Тестування програмного забезпечення	42
4.6 Висновок до розділу 4	43
ВИСНОВОК	45
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ	46
ДОДАТОК А Технічне завдання	48
ДОДАТОК Б Фрагмент тексту програми	51
ДОДАТОК В Слайди презентації	108

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE	– інтегроване середовище розробки;
UI	– інтерфейс користувача;
UML	– Unified Modeling Language;
VR	– віртуальна реальність;
ОС	– операційна система;
ПЗ	– програмне забезпечення;
Префаб	– заготовлений і налаштований завчасно об'єкт.

ВСТУП

Розвиток технологій віртуальної реальності (VR) значно вплинув на різні сфери, включаючи інженерію, виробництво, навчання та розробку програмного забезпечення. Особливо VR отримала широке застосування в моделюванні складних технічних об'єктів, навчанні фахівців та оптимізації виробничих процесів[1]-[8].

Одним із перспективних напрямків використання VR є створення інтерактивних навчальних симуляторів для вивчення будови і роботи технічних систем. Зокрема, у сфері автомобільної інженерії VR може забезпечити безпечне, гнучке та економічно вигідне середовище для навчання механіків, інженерів і технічного персоналу без потреби у фізичних транспортних засобах[1]-[8].

Результати цього дослідження можуть бути корисними для навчальних закладів, автомобільних майстерень, виробничих підприємств та інших організацій, які займаються технічним обслуговуванням і ремонтом транспортних засобів. Використання VR-технологій у таких сферах дозволить значно покращити якість підготовки спеціалістів, зменшити витрати на навчання та підвищити безпеку виконання технічних робіт[1]-[8].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Віртуальна реальність у технічному навчанні та симуляції

Віртуальна реальність (VR) вже давно перестала бути просто іграшкою – сьогодні вона активно використовується в освіті, особливо в технічних напрямках, як-от інженерія. Завдяки VR можна створити такі умови, в яких студенти мають можливість навчатися, моделювати складні процеси або тренуватися без ризику, витрат і складної логістики[1]-[8].

Однією з головних переваг VR у навчальному процесі є безпечність. Наприклад, під час роботи з небезпечним обладнанням або матеріалами у віртуальному просторі можна зробити помилку, не боячись наслідків – ніхто не постраждає і нічого не зламається. Це дуже корисно для навчання майбутніх інженерів, бо багато задач у цій сфері пов'язані з потенційно небезпечними ситуаціями[1]-[8].

Крім того, віртуальні симуляції значно знижують витрати. Не потрібно купувати дороге обладнання чи облаштовувати спеціалізовані лабораторії. Усе, що потрібно – це VR-система і відповідне програмне забезпечення. Наприклад, можна створити тренажер для роботи з двигуном, і всі студенти зможуть попрактикуватися, не торкаючись реальної техніки[1]-[8].

Ще один плюс – це гнучкість. Віртуальні середовища легко налаштовуються під конкретні навчальні теми. Наприклад, якщо треба показати поломку автомобіля, то її можна швидко змодельовати, змінити параметри або ситуацію і не потрібно щоразу готувати нове обладнання, як у справжній майстерні[1]-[8].

Окрема тема – це реалістичність сучасних VR-рішень. У таких симуляціях можна побачити, як ведуть себе різні матеріали, як взаємодіють механізми, що відбувається в електросистемах. Це дозволяє глибше зрозуміти принципи роботи інженерних систем, бо все видно наживо, а не просто на картинках чи схемах[1]-[8].

Дуже важливо, що VR дозволяє створювати різні навчальні сценарії, які складно або навіть неможливо відтворити в реальності. Наприклад, аварійні ситуації на виробництві або рідкісні поломки. Такі ситуації можна симулювати у VR і навчитися правильно реагувати без ризику[1]-[8].

І останнє – доступність. Оскільки VR-системи не прив'язані до конкретного місця, то навчатися можна будь-де. Це особливо корисно для студентів із регіонів, де немає потрібної інфраструктури, або для компаній, які не можуть собі дозволити побудувати справжню лабораторію. А ще – це зручно для практикуючих фахівців, які хочуть підвищити кваліфікацію, не відриваючись надовго від роботи[1]-[8].

1.2 Огляд існуючих рішень

У світі віртуальної реальності для інженерії та навчання з'являється все більше платформ, що дозволяють розробляти, тестувати та вивчати автомобільні системи. У цьому розділі ми розглянемо кілька відомих рішень у VR-сфері, таких як Ford Immersive Vehicle Environment (FiVE), Siemens NX VR Design Review та різні VR-платформи для навчання механіків, порівнюючи їх з можливостями мого проєкту[6-8].

Ford Immersive Vehicle Environment (FiVE) є VR-платформою, розробленою компанією Ford для моделювання автомобілів на етапі проєктування та тестування. Вона дозволяє інженерам взаємодіяти з віртуальними моделями автомобілів і оцінювати їх дизайн на різних етапах розробки[6].

Інженери можуть взаємодіяти з 3D-моделями автомобілів у віртуальному середовищі, що дозволяє більш детально оцінити елементи дизайну, їх пропорції та ергономіку[6].

Віртуальне середовище дає змогу виявити потенційні проблеми в дизайні автомобіля, які можуть бути складними для виявлення у традиційних методах проєктування[6].

Хоча платформа дозволяє переглядати моделі та виявляти дефекти, вона не підтримує інтерактивні дії, такі як розбір автомобіля по деталях, а також не дозволяє взаємодії з компонентами(Знімати та встановлювати)[6].

Siemens NX VR Design Review – це VR-рішення, що застосовується для аналізу 3D-моделей промислового обладнання та транспортних засобів. Воно орієнтоване на інженерів, які займаються перевіркою конструкцій та їх візуалізацією в реальному часі[7].

Платформа підтримує відображення великих і складних моделей промислових об'єктів і транспортних засобів у VR, що дозволяє інженерам аналізувати їх компонування і зовнішні характеристики[7].

Користувачі можуть робити коментарі та модифікації безпосередньо в системі, що дозволяє оптимізувати конструкції і враховувати зауваження команди[7].

Хоча система надає можливість візуалізувати та модифікувати моделі, вона не підтримує реалістичну фізику або симуляцію взаємодії, що значно обмежує її функціональність для практичних тренувань[7].

Різні VR-платформи, такі як Car Mechanic Simulator VR, орієнтований на навчання механіків в умовах віртуальної реальності. Дозволяє користувачам вивчати механіку автомобілів, проводити ремонт та розбір автомобілів у віртуальних середовищах[8].

Інтерактивний розбір автомобіля: Користувачі можуть демонтувати компоненти автомобіля, проводити ремонтні роботи, використовуючи різні інструменти[8].

Має інтегровані бази даних, де користувачі можуть дізнаватися більше про деталі автомобіля і способи їх обслуговування[8].

Платформа надає віртуальні інструменти, за допомогою яких можна виконувати різні дії з автомобілем, такі як викручування болтів, заміна коліс і інші більш складні функції[8].

У таблиці 1.1 наведено порівняння інтерактивних платформ віртуальної реальності для візуалізації, навчання та взаємодії з автомобільними системами.

Таблиця 1.1 – Порівняння інтерактивних платформ віртуальної реальності для візуалізації, навчання та взаємодії з автомобільними системами.

Критерій порівняння	Ford FiVE	Siemens NX VR	Car Mechanic Simulator VR
Візуалізація складних моделей	+	+	+
Інтерактивність із компонентами	–	–	+
Реалістична фізика	–	–	±
Можливість тренування механіків	–	–	+
Коментування, редагування моделей	–	+	–
Робота з базою даних деталей	–	–	+
Інтеграція з інженерними інструментами	+	+	–
Орієнтація на навчання/освіту	–	±	+

Аналіз предметної області та огляд існуючих рішень показав, що VR-технології мають значний потенціал у сфері технічного навчання, зокрема в автомобільній інженерії. Проте більшість наявних рішень орієнтовані або на візуалізацію та аналіз 3D-моделей, або на навчання за готовими сценаріями, які обмежують гнучкість взаємодії з деталями[6-8].

При розробці інтерактивних платформ віртуальної реальності для візуалізації, навчання та взаємодії з автомобільними системами, необхідно забезпечити: реалістичне відображення автомобільних систем та окремих компонентів у віртуальному середовищі, користувач має мати змогу демонтувати, замінювати та досліджувати окремі деталі автомобіля у VR за допомогою контролерів або жестів, необхідно реалізувати базову фізику (гравітація, зіткнення, кріплення деталей), яка відповідає реальним умовам при роботі з компонентами автомобіля, платформа повинна підтримувати навчальні модулі з поетапним виконанням завдань (наприклад: зняття акумулятора, коліс)[1]-[8].

1.3 Висновок за розділом

Віртуальна реальність зараз показує себе як дуже перспективний інструмент у сфері технічного навчання, особливо коли йдеться про автомобільну інженерію. Завдяки тому, що VR створює безпечне середовище, дозволяє економити ресурси, легко налаштовується під різні завдання і виглядає максимально реалістично, вона стає реально корисною для підготовки майбутніх фахівців.

Великий плюс VR у тому, що в ньому можна відтворити ситуації, які в реальному житті або дуже рідкісні, або просто небезпечні. Наприклад, серйозні поломки чи аварії. У VR можна спокійно тренуватись, не ризикуючи ні технікою, ні життям. До того ж навчання можна проходити будь-де, потрібен

тільки VR-шолом і програмне забезпечення. Це відкриває нові можливості для університетів, навчальних центрів або компаній, які хочуть навчати працівників.

Якщо глянути на вже існуючі рішення, такі як Ford Immersive Vehicle Environment, Siemens NX VR Design Review чи Car Mechanic Simulator VR, то видно, що більшість із них більше фокусуються або на красивій картинці, або на конкретних моментах взаємодії. Це, на мою думку, залишає простір для створення нової VR-платформи, яка б поєднувала все разом: реалістичну симуляцію технічних процесів, повну інтерактивність та можливість підлаштовуватись під різні навчальні цілі.

Якщо таку систему зробити правильно, вона зможе реально покращити підготовку спеціалістів у складних галузях, де важлива як точність, так і практичний досвід – таких, як, наприклад, автомобільна інженерія.

2. МАТЕРІАЛИ І МЕТОДИ

Другий розділ буде присвячено обґрунтуванню вибору технічних засобів розробки: мови програмування, середовища розробки, програмних надбудов тощо. Рішення, прийняті в цьому розділі, будуть підтверджені та аргументовані з урахуванням специфіки проекту.

2.1 Вибір рушія

При виборі рушія для розробки VR-симулятора механіків важливо враховувати кілька ключових факторів, які визначатимуть ефективність та зручність роботи, а також якість кінцевого продукту. Розглянемо такі критерії, як, простота і швидкість розробки, гнучкість рушія, підтримка VR, та наявність великої спільноти, що може допомогти у розв'язанні проблем під час розробки (табл. 2.1)[16]-[18].

Таблиця 2.1 – Порівняння рушіїв

Критерій	Unity	Unreal Engine	Godot
Складність входу	Середня	Висока	Низька
Простота розробки	Висока	Низька	Висока
Графіка	Хороша(Але поступається Unreal)	Найкраща	Хороша для 2d погана для 3d
Підтримка VR	Чудова(OpenXR, UnityXR, SteamVR)	Хороша, але складніша в налаштуванні	Мінімальна (слабка підтримка OpenXR)
Підходить для VR-симулятора механіків?	Так	Можна, але складніше та вимагає більше ресурсів	Ні (недостатньо підтримки)

Як показує таблиця 2.1, Unity є найкращим вибором для VR-симулятора механіків завдяки його гнучкості, простоті розробки та підтримці VR-технологій. Unreal Engine є чудовим рішенням для фото реалістичної графіки та великих AAA-проектів, однак його складність і ресурсоємність роблять його менш привабливим для швидкого прототипування. Godot, у свою чергу, відмінно підходить для 2D-ігор, але має слабку підтримку VR. Тому Unity є оптимальним вибором для реалізації поставленої задачі[16]-[18].

Unity має зручний інтерфейс і потужні інструменти, що дозволяють швидко створювати прототипи та готові проекти. Завдяки великій кількості шаблонів та інструментів для оптимізації процесу розробки, час на створення базових функціональностей значно зменшується[16]-[18].

Unity є хорошим вибором для інді-проектів та середнього рівня розробки, оскільки він підтримує широкий спектр типів ігор і додатків. Водночас, його гнучкість дозволяє реалізувати складні механіки, зокрема для навчальних VR-симуляторів[16]-[18].

Unity має чудову інтеграцію з OpenXR та підтримує всі основні сучасні VR-платформи, такі як Meta Quest, SteamVR, Windows Mixed Reality та інші. Це дозволяє реалізувати повноцінний VR-досвід, який буде підтримувати різноманітні пристрої та платформи[16]-[18].

Unity має одну з найбільших спільнот серед рушіїв, що забезпечує доступ до величезної кількості ресурсів, документації, туторіалів, а також можливість звернутись за допомогою до інших розробників через форуми та онлайн-спільноти[16]-[18].

На противагу цьому, Unreal Engine є відмінним для створення фотореалістичних графічних рішень та великих AAA-проектів, але його складність та великі вимоги до ресурсів роблять його менш привабливим для швидкої розробки VR-симуляторів механіків. Хоча Unreal Engine має потужні

інструменти для графіки та рендерингу, він вимагає більше часу на налаштування і більше ресурсів для оптимізації[16]-[18].

Godot, хоча і має сильні позиції в розробці 2D-ігор, має обмежену підтримку VR, що робить його невідповідним для розробки VR-додатків на сьогоднішній день. Для вашого проекту з інтерактивним навчанням механіки автомобілів важливо, щоб рушій мав підтримку VR, тому Godot є менш підходящим вибором[16]-[18].

Отже, Unity є оптимальним вибором для розробки VR-симулятора механіків, оскільки він забезпечує необхідну гнучкість, підтримку VR-платформ та спільноту для швидкої розробки та вирішення проблем. Це дозволить зосередитись на розробці і використовувати вже наявні технології та поєднувати їх у новий застосунок[16]-[18].

2.2 Вибір мови програмування

У моєму проєкті було прийнято рішення використовувати мову програмування C#, що є найкращим вибором для роботи з Unity. Ось детальніші причини та переваги цього вибору (табл 2.2)[14].

Як можемо бачити з таблиці 2.2 на розгляді було 3 мови програмування. Після аналізу сильних і слабких сторін кожної з них, я зупинився саме на C#, оскільки ця мова забезпечує оптимальний баланс між зручністю, продуктивністю та інтеграцією з Unity[14]-[15].

C# був єдиним правильним вибором з кількох причин. Насамперед, Unity офіційно підтримує цю мову, що значно спрощує процес розробки. Усі інструменти, документація та приклади побудовані навколо C#, а отже, не доводиться витрачати додатковий час на обхідні рішення. Крім того, мова є досить простою у вивченні, має чітку структуру та сучасні можливості, зокрема асинхронне програмування через `async/await`, інтерфейси й делегати, що дозволяє ефективно реалізовувати логіку поведінки в застосунку[14]-[15].

Таблиця 2.2 – Порівняння мов програмування

Критерій	C#	Rust	Python
Складність мови	Помірна, легка у вивченні	Висока, особливо через систему запозичень	Низька, дуже проста для швидкого старту
Контроль над пам'яттю	Автоматична (Garbage collector)	Безпечне ручне керування, без збирача сміття	Автоматичне керування
Продуктивність	Висока, але нижча за C++	Дуже висока, близька до C++	Низька
Розробка UI логіки	Зручна, багаті можливості	Можлива, але потребує інтеграції через C API або плагіни	Зручна для логіки, слабка підтримка UI
Паралелізм потоки	Простий у використанні async/await	Гнучкий, але складний	Обмежений через GIL
Масштабованість проєкту	Добре підходить для проєктів різної величини	Ідеально, але вимагає суворої структури та досвіду	Підходить для малих і середніх проєктів
Час розробки	Оптимальний баланс	Довший через складність та налаштування	Дуже швидкий
Бібліотеки для 3D VR	Багато розширень	Можна інтегрувати з Unity, але через FFI	Мало можливостей для повноцінного 3D

Щодо Rust, ця мова справляє враження дуже потужної і високопродуктивної, проте має значно вищий поріг входження. Особливості управління пам'яттю, система запозичень і строгий контроль за життєвим циклом змінних, створюють додаткову складність на етапі розробки. До того ж,

Rust не має прямої підтримки Unity, і для його інтеграції доводиться використовувати FFI, що передбачає створення окремих бібліотек з наступним доступом до них із C#. Це підвищує складність проєкту, зменшує гнучкість при швидких змінах і уповільнює розробку, особливо на ранніх етапах, до того ж все зводиться до використання C# з кількома додатковими кроками, що є не доцільними зі сторо[14]-[15].

Python, зі свого боку, є дуже привабливою мовою завдяки своїй простоті та високій швидкості розробки. Проте він не підходить для повноцінної розробки додатків у середовищі Unity через низьку продуктивність і обмежені можливості роботи з графікою та VR[14]-[15].

Крім того, наявність Global Interpreter Lock(GIL) значно ускладнює ефективне використання потоків, що критично важливо для складніших ігрових систем. У випадках, де потрібна взаємодія з графічними рушіями чи побудова масштабних 3D-сцен, Python поступається більш спеціалізованим мовам[14]-[15].

C# є основною мовою програмування для Unity. C# дозволяє користуватися всіма можливостями рушія на повну, включаючи обробку анімацій, фізичних ефектів, управління штучним інтелектом, а також оптимізацію процесів. Це забезпечує гладку розробку без необхідності в додаткових налаштуваннях, що є важливим для прискорення створення прототипів та додатків в цілому. Використання C# дозволяє максимально ефективно використовувати екосистему Unity для реалізації складних механік та високоякісної графіки[14]-[15].

C# є об'єктно-орієнтованою мовою програмування, що надає величезну кількість інструментів для створення складних інтерактивних систем. Підтримка класів, інтерфейсів, подій і делегатів, що дозволяє будувати чітко структуровані та масштабовані ігрові системи. Простота створення скриптів для керування ігровими об'єктами, а також для взаємодії з UI та іншими компонентами Unity. Можливості асинхронного програмування через

async/await, що дозволяє ефективно працювати з паралельними обчисленнями. Ці особливості роблять C# ідеальним інструментом для створення складних застосунків, які потребують динамічної взаємодії між різними елементами додатку[14]-[15].

C# є дуже добре оптимізованим для роботи в середовищі Unity, що дозволяє створювати високопродуктивні застосунки. Unity автоматично компілює C# код у проміжний IL-код (Intermediate Language), який потім оптимізується для конкретної платформи (ПК, мобільні пристрої, консолі, VR/AR системи). Це дозволяє досягти високої продуктивності без потреби в додаткових налаштуваннях і значно полегшує створення кросплатформених продуктів[14]-[15].

У підсумку, хоча Rust і Python мають свої переваги у вузьких сферах, саме C# забезпечує стабільну, зручну й ефективну основу для розробки в Unity, дозволяючи зосередитись на архітектурі застосунку, а не на вирішенні технічних обмежень.

2.3 Вибір середовища розробки

У рамках мого проекту я обрав Visual Studio як основне середовище розробки для роботи з Unity та C#. Це рішення ґрунтується на кількох важливих факторах. Visual Studio є рекомендованим середовищем розробки від Unity. Під час встановлення Unity користувачі можуть безпосередньо інтегрувати Visual Studio як стандартний редактор для написання C# скриптів. Visual Studio надає потужні інструменти для дебагінгу, що робить процес пошуку і виправлення помилок значно ефективнішим. Підключення до Unity Editor для запуску дебаг-сесій безпосередньо в середовищі Unity. Брейкпоінти, що дозволяють зупиняти виконання програми на конкретних рядках коду, щоб детальніше вивчити поточний стан змінних. Спостереження за значеннями змінних у реальному часі. Відстеження викликів методів та крокування по коду, що

дозволяє гнучко переходити по кроках виконання коду і аналізувати його деталі. Visual Studio підтримує всі новітні функції мови C#, що дозволяє застосовувати сучасні методи програмування, такі як: асинхронне програмування за допомогою `async/await`, LINQ для роботи з колекціями даних, шаблони проектування для побудови масштабованих та підтримуваних архітектур. Visual Studio надає багатий набір інструментів для роботи з великими проєктами: підтримка великих кодових баз та зручна організація проєкту, покращене автодоповнення та рефакторинг коду, що значно знижує час на розробку та підвищує якість коду, інтеграція з системами контролю версій, такими як Git, для легшого управління версіями коду та командної роботи[14]-[18].

Отже, вибір Visual Studio як основного середовища розробки для роботи з Unity є раціональним і практичним кроком. Завдяки тісній інтеграції з обраним рушієм Unity, розширеній підтримці обраної мови C# та наявності зручних інструментів для налагодження і структурування коду, ця IDE дозволяє ефективно реалізовувати функції проєкту, так як використовуються поєднані між собою рушій, мова та середовище розробки[14]-[18].

2.4 Висновок за розділом 2

У цьому розділі було здійснено аналіз сучасних інструментів та технологій, що застосовуються для розробки VR-додатків. Проведене порівняння рушіїв підтвердило доцільність використання Unity, з огляду на його широку підтримку платформ віртуальної реальності, інтуїтивно зрозумілий інтерфейс, активну спільноту розробників, а також гнучку архітектуру, що дозволяє швидко створювати, тестувати й масштабувати інтерактивні сцени.

Мовою програмування було обрано C# як найбільш сумісну з Unity. Її об'єктно-орієнтована структура, підтримка подій, делегатів, асинхронності та гнучкі засоби організації проєкту забезпечують високу якість коду і легкість у підтримці великої системи.

Вибрані технології дозволяють реалізувати ключові вимоги до VR-симулятора: взаємодію користувача з віртуальними об'єктами; зміну станів компонентів; гнучке управління інтерфейсом та адаптивність до сценаріїв навчання. У подальших розділах буде представлено архітектуру системи, реалізацію основних модулів та результати практичної реалізації, що спираються саме на ці методи та засоби.

3. ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення

Програма складається з 3 основних категорій класів. Класів автомобіля, сцен та рівнів. На рисунку 3.1 позначена структура класів автомобіля.

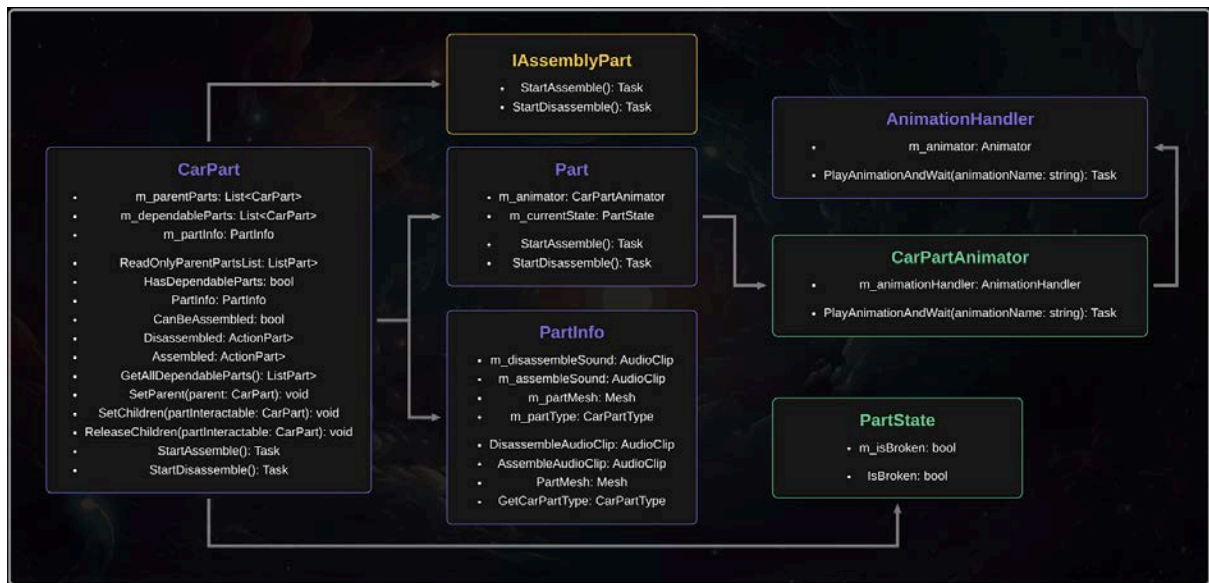


Рисунок 3.1 – Структура класів автомобіля

IAssemblyPart.cs – визначає інтерфейс для класів, які представляють частини над якими дозволено виконувати операції а також визначає основні методи.

CarPart.cs – абстрактна основа для будь-якого компонента у складі автомобіля. Вона інкапсулює взаємозв'язки, поведінку та стани, необхідні для визначення взаємодії між окремими частинами та загальною системою автомобіля.

CarPartType.cs – визначає різні види деталей автомобіля.

Part.cs – конкретна реалізація класу CarPart, яка керує даними, специфічними для автомобільного компонента, через PartInfo. Надає функціональність для складання та розбирання деталей, зберігаючи ієрархічні зв'язки між залежними компонентами. Клас використовує анімації для

візуального зворотного зв'язку та забезпечує правильний порядок і умови для складання або розбирання.

`PartInfo.cs` – `ScriptableObject`, розроблений для зберігання та використання основних даних про автомобільну деталь, включаючи 3D-модель (`PartMesh`), аудіокліпи складання та розбирання, а також тип деталі (`CarPartType`). Цей об'єкт є багаторазовим і спрощує управління даними для різних компонентів.

`CarPartInteractable.cs` – керує взаємодією користувача з автомобільною деталлю у XR-середовищі, включаючи поведінку наведення та вибору. Змінює візуальний вигляд деталі(матеріал) відповідно до станів взаємодії.

`Car.cs` – керує автомобільними компонентами у віртуальній симуляції, включаючи їх стани, взаємодію та візуальне представлення. Підтримує різні режими (наприклад, огляд, інспекція, розбирання, складання) і динамічно коригує поведінку та зовнішній вигляд деталей.

`CarStateManager` – керує поведінкою у залежності від режиму,

`CarPartManager` – відстежує стан деталей, зламані компоненти та залежності.

`CarViewModeManager.cs` – клас, який керує різними режимами перегляду автомобіля. Він дозволяє змінювати режими перегляду, застосовуючи новий режим і виходячи з попереднього.

`ICarViewMode.cs` – інтерфейс для режимів перегляду автомобіля.

`InspectionMode.cs`, `AssemblyMode.cs`, `DisassemblyMode.cs`, `OverviewMode.cs` – реалізації інтерфейсу `ICarViewMode` для режимів інспекції(пошуку несправностей), розбирання, збору та огляду автомобіля.

`ModeSwitcher.cs` – клас, що дозволяє перемикати режими перегляду автомобіля через `CarViewModeManager`.

`AnimationHandler.cs` – спрощує відтворення анімацій за допомогою `Animator` у `Unity`, дозволяючи очікувати їх завершення асинхронно у коді.

Одними з найважливіших для застосунку є класи сцен, так як ефективно і правильне завантаження і вивантаження сцен дозволяє розширювати проєкт зберігаючи при цьому легкість та оптимізованість застосунку. На рисунку 3.2 наведено структуру класів сцен.

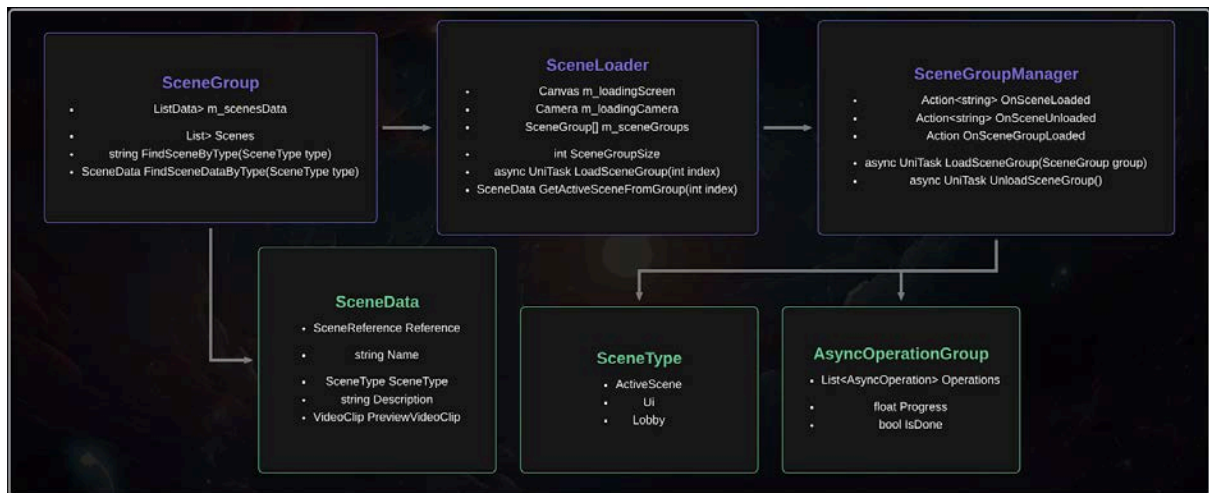


Рисунок 3.2 – Структура класів сцен

`SceneGroup.cs` – представляє дані сцени як окремо, так і у складі групи, дозволяючи отримувати сцени за типом та надаючи їх детальну інформацію.

`SceneLoader.cs` – керує завантаженням і відвантаженням груп сцен, включаючи перемикання активних сцен, відображення завантажувального екрана та повідомлення про завершення завантаження чи вивантаження.

`SceneGroupManager.cs` – відповідає за управління групами сцен, забезпечуючи їх асинхронне завантаження та вивантаження. Гарантує, що всі сцени у групі завантажуються або відвантажуються разом та керує плавними переходами між сценами.

`SceneManipulator.cs` – керує навігацією по сценах та дозволяє вибирати та завантажувати сцени в межах групи, використовуючи `SceneLoader` для завантаження обраної сцени.

`ScenePresenter.cs` – оновлює інтерфейс користувача (UI) та відтворює прев'ю-відео, коли вибрана сцена змінюється.

Не менш важливі для застосунку класи – класи рівнів, так як вони надають змогу змінювати деталі рівнів не впливаючи на інші компоненти програми. На рисунку 3.3 наведено структуру класів рівнів.

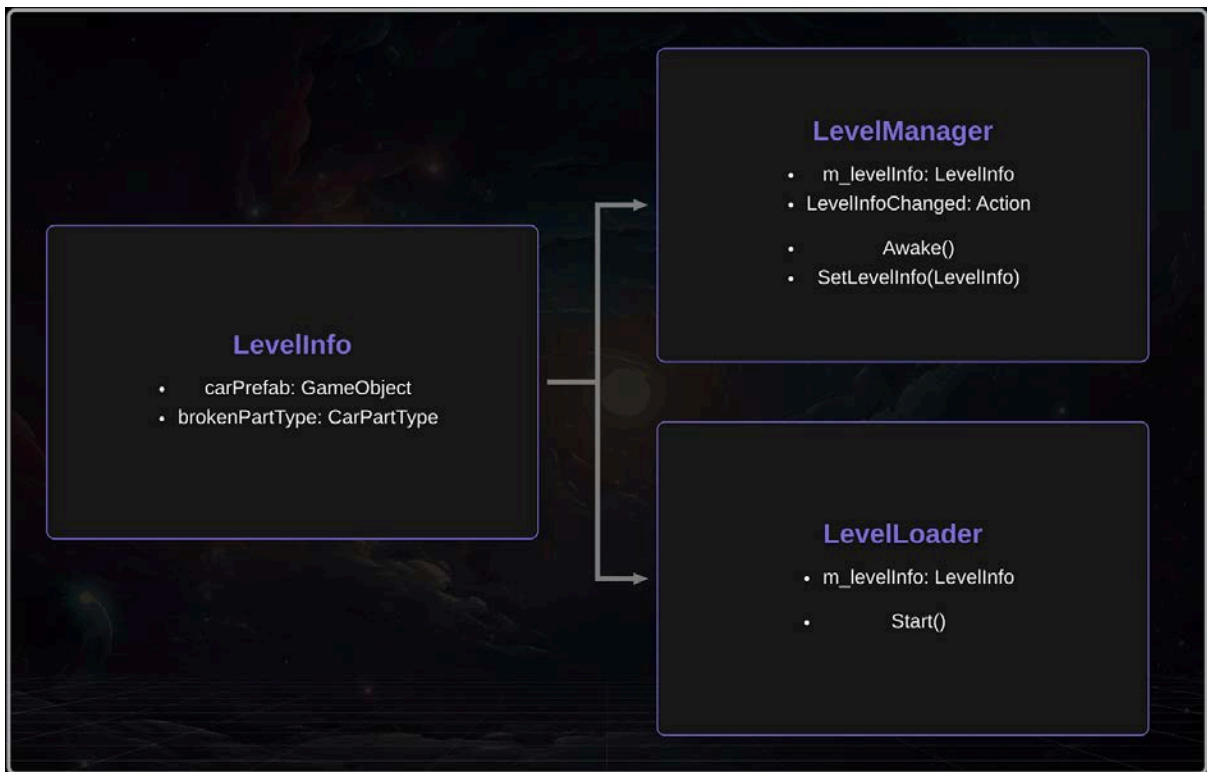


Рисунок 3.3 – Структура класів рівнів

LevelInfo.cs – `ScriptableObject`, який містить ключову інформацію про рівень гри, включаючи префаб автомобіля та тип пошкодженої деталі, яку потрібно відремонтувати.

LevelLoader.cs – керує процесом передачі інформації про рівень до **LevelManager** на початку рівня.

LevelManager.cs – керує інформацією, специфічною для рівня, та повідомляє інші об'єкти про зміни в даних рівня.

Також у проекті присутні групи класів що не є критично важливими, але вони утворюють собою основу взаємодії користувача з середовищем.

`UIInstance.cs` – визначає контракт для UI об'єктів, щоб підтримувати оновлення та коригування позиції.

`UIInstance.cs` – гнучке представлення UI об'єкта з основними функціональностями для управління його життєвим циклом, видимістю, оновленнями та позицією.

`UiManager.cs` – відповідає за управління видимістю та життєвим циклом UI елементів в різних сценах.

`UIButtonAudio.cs` – клас, що відтворює звук при натисканні на кнопку UI. Коли кнопка натискається, викликається метод, який відтворює визначений звук через `AudioManager`.

`AudioManager.cs` – клас для відтворення звуків в грі. Він має статичний метод `PlaySound`, який створює тимчасовий об'єкт, додає до нього `AudioSource`, відтворює звук і видаляє об'єкт після завершення відтворення. Якщо вже існує екземпляр класу, новий не створюється.

`MenuUi.cs` – управляє UI елементами меню та пов'язує функціональність кнопок з обробкою команд, являє собою головне меню.

`Billboard.cs` – забезпечує, щоб об'єкт завжди дивився на ціль, коригуючи своє обертання для постійного спрямування на визначену позицію.

`Container.cs` – управляє реєстрацією, зняттям реєстрації та зберіганням залежностей, дозволяючи вирішення та отримання зареєстрованих залежностей.

`HoverMaterial.cs` – `ScriptableObject` для управління спільними посиланнями на матеріали для станів деталі.

`ToolInteractor.cs` – клас, що дозволяє інструменту взаємодіяти з частинами машини в середовищі розширеної реальності. Коли інструмент захоплюється, він активується і може взаємодіяти з об'єктами, якщо вони

відповідають певним умовам. Після відпускання інструмента, взаємодія припиняється. Клас управляє логікою взаємодії.

3.2 Особливості реалізації програмного забезпечення

Розробка інтерактивної VR-платформи для візуалізації, навчання та взаємодії з автомобільними системами потребує високої інтеграції між віртуальними об'єктами, їх фізичними властивостями та механіками взаємодії з користувачем. Нижче описано основні етапи та особливості реалізації програмного забезпечення для такої платформи, а також використання різних компонентів Unity та специфічних скриптів для забезпечення ефективної роботи VR-системи[1]-[13].

У VR-платформі важливо забезпечити зручне і інтуїтивно зрозуміле управління різними системами автомобіля, такими як двигун, трансмісія, електричні системи, гальмівна система тощо[1]-[13].

Для взаємодії користувача з автомобільними компонентами використовуються спеціалізовані grab interactors (наприклад, при виборі інструменту чи деталі автомобіля). Взаємодія з цими компонентами має бути максимально реалістичною і відображати фізичні властивості об'єктів, наприклад, можливість розібрати частини машини, використовувати інструменти або змінювати деталі. На рисунку 3.4 наведено приклад взаємодії з інструментом[1]-[13].

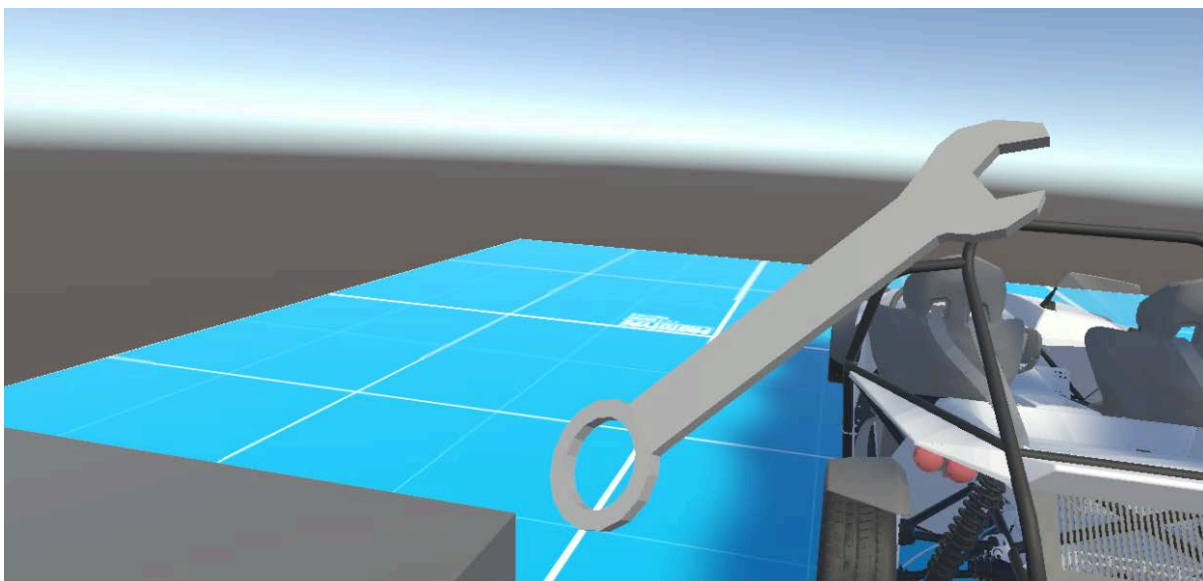


Рисунок 3.4 – Приклад взаємодії з інструментом

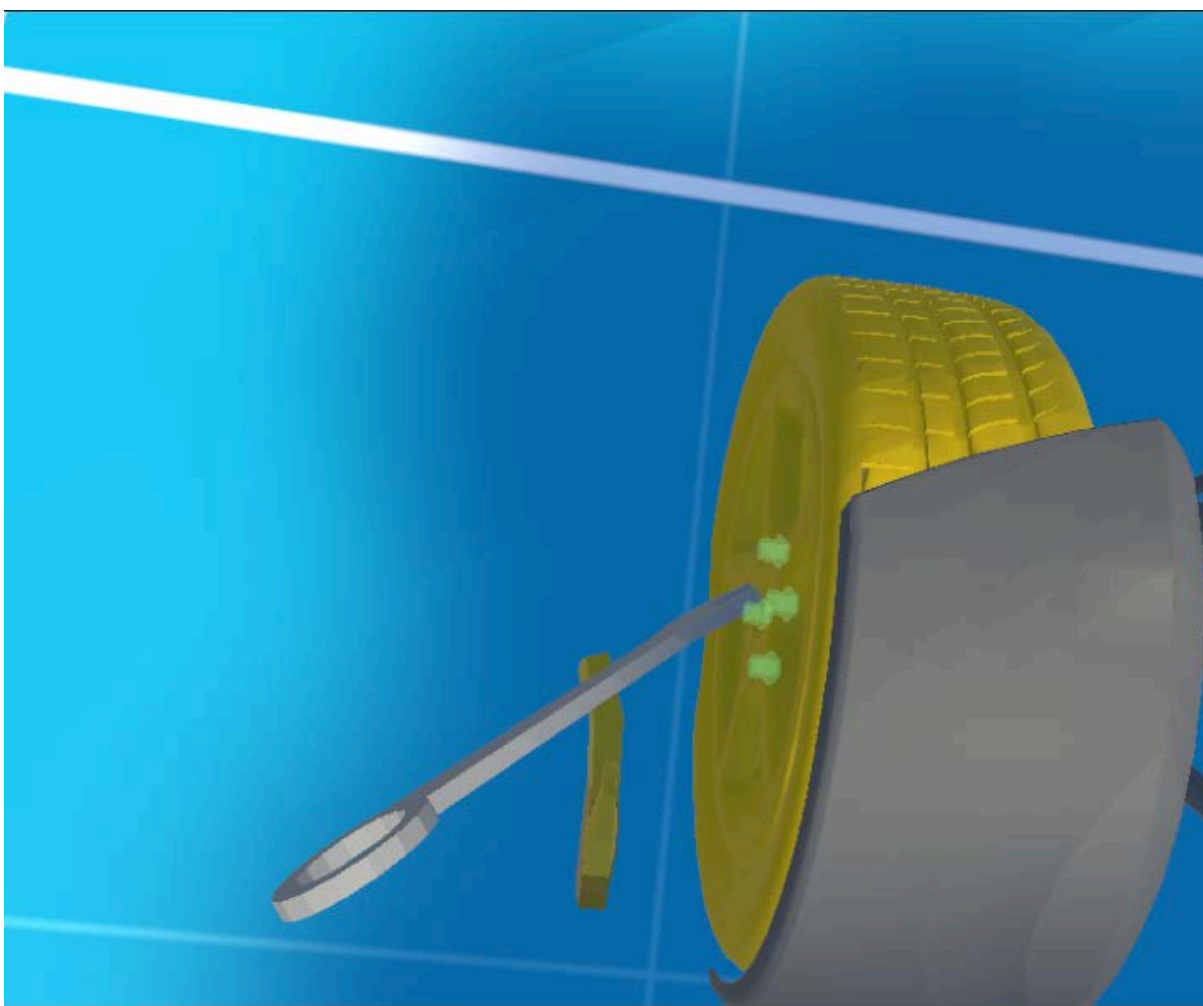


Рисунок 3.5 – Взаємодія з автомобільними запчастинами

На рисунку 3.4 – 3.5 можемо бачити взаємодію інструментів з гравцем а також інструментів з деталями, для зручності деталі підсвічуються, жовтий колір каже цей інструмент не підходить для взаємодії з обраною деталлю, зелений показує що інструмент правильний і деталь можна зняти, червоний – що деталь може бути знята цим інструментом, але деталь не можливо зняти через інші деталі.

Для забезпечення роботи з різними VR-платформами, такими як Meta Quest або SteamVR, використовується Unity XR для інтеграції віртуальної реальності. Unity XR підтримує широкий спектр VR-пристроїв та забезпечує оптимальну продуктивність для візуалізації складних тривимірних сцен і відображення реалістичних механічних та електричних систем автомобіля[1]-[13].

Інтерфейс користувача (UI): Для забезпечення навчання користувачам і надання доступу до необхідної інформації використовується UI-менеджер. Цей компонент відповідає за відображення інтерфейсу віртуального автомобіля, де користувач може вибирати системи для взаємодії, отримувати відомості про стан системи, а також виконувати інші дії через VR-контролери[1]-[13].



Рисунок 3.6 – Головне меню користувача

Рисунок 3.6 являє собою просте меню для користувача, яке дозволяє повернутися в лобі, вийти з застосунку, змінити режим машини, а також поміняти рівень - навчальний сценарій.

Віртуальний автомобіль: Автомобіль складається з окремих модулів, таких як двигун, трансмісія, підвіска, кузов тощо, що кожен має своє унікальне поведінкове дерево для обробки стану та взаємодії. Наприклад, скрипт CarPart обробляє дані про кожну деталь автомобіля, зберігаючи їх стан, який може змінюватися під впливом дій користувача (заміна частини, виправлення пошкоджень, зміна налаштувань)[1]-[13].

Щоб забезпечити реалістичну та ефективну взаємодію з автомобільними системами, у системі повинна бути реалізована динамічна взаємодія між користувачем і об'єктами[1]-[13].

Скрипт CarPartInteractable відповідає за обробку взаємодії з окремими частинами автомобіля. Ці скрипти можуть реагувати на події, такі як натискання кнопок, використання інструментів або активація системи за допомогою VR-контролера[1]-[13].

Для забезпечення високої якості візуалізації та продуктивності в VR важливо застосовувати оптимізаційні техніки. Користувачі повинні мати можливість отримувати підказки або навчальні матеріали, наприклад, під час взаємодії з новими компонентами автомобіля. Цю задачу вирішує UIManager та TutorialManager що відображають навчальні етапи[1]-[13].

Для реалізації функціоналу навчання віртуального симулятора використовуються скрипти, що відповідають за різні частини системи[1]-[13].

Менеджер туторіалів TutorialManager забезпечує демонстрацію крок за кроком навчання користувача. Скрипт має можливість запускати різні етапи навчання, перевіряти умови виконання завдання та адаптувати взаємодію з користувачем[1]-[13].

Кожен крок туторіалу реалізується через окремий клас TutorialStep, який визначає умови, коли користувач завершив етап і може перейти до наступного[1]-[13].

Скрипт TutorialCondition відповідає за перевірку виконання умов для переходу до наступного етапу навчання або для завершення конкретної задачі (наприклад, якщо користувач відремонтував певну частину автомобіля)[1]-[13].

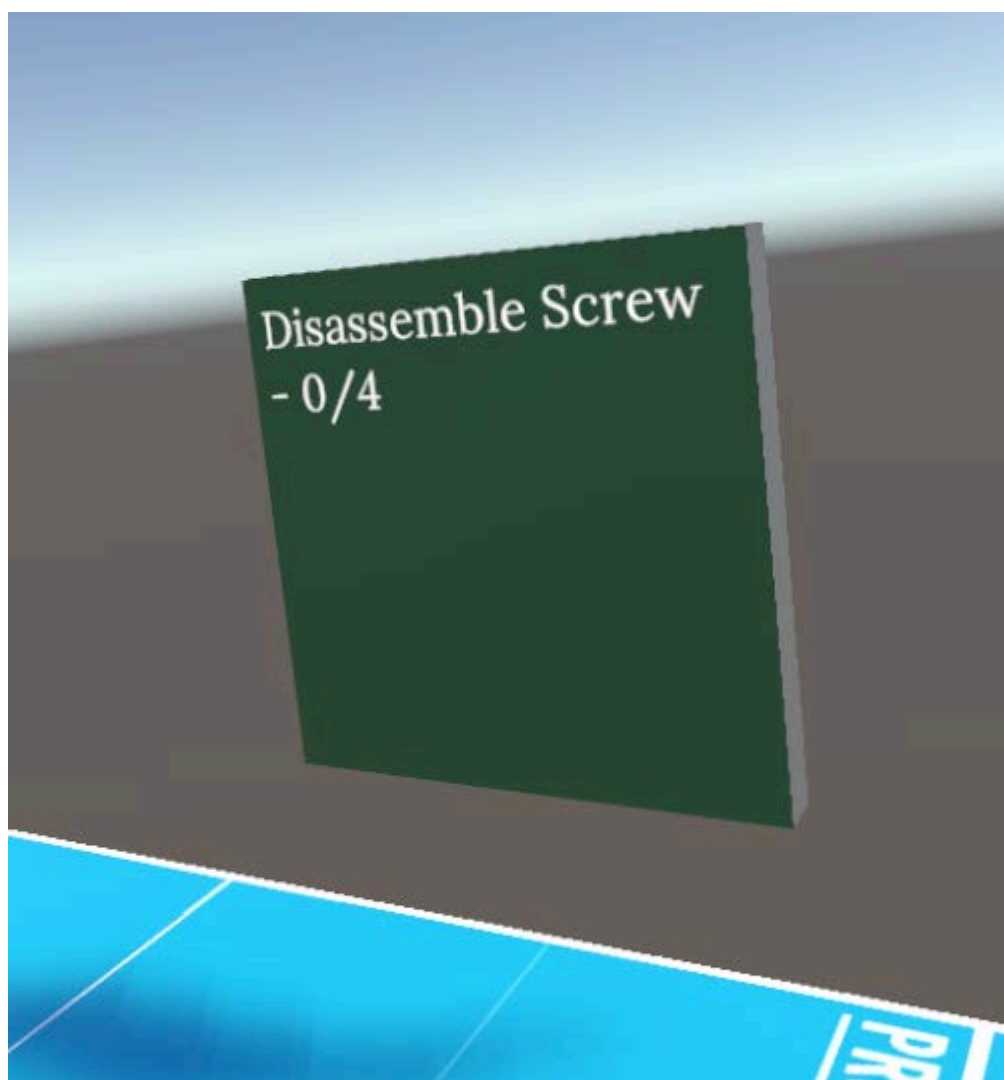


Рисунок 3.7 – Приклад дошки з виконаним завданням

Рисунок 3.7 відображає поточний навчальний крок, та оновлюється в залежності від того як ми виконуємо завдання.

3.3 Висновок до розділу 3

У цьому розділі було детально розглянуто структуру програмного забезпечення VR-платформи, що включає основні категорії класів – автомобіля, сцен та рівнів. Продумана архітектура коду дозволяє ефективно керувати станами деталей, сценами та режимами взаємодії, а також забезпечує гнучкість для розширення функціональності без шкоди для цілісності системи. Особлива увага приділена зручності взаємодії у VR-середовищі, що досягається завдяки використанню `grab interactor`'ів, `ScriptableObject`'ів та модульної структури. Такий підхід дозволяє досягнути високого рівня реалістичності, гнучкості та масштабованості системи, що є ключовим для розробки сучасного навчального інструменту з інтерактивною взаємодією.

4. ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення

Інтерактивна VR-платформа для візуалізації, навчання та взаємодії з автомобільними системами розроблена для надання користувачам можливості здобути практичні знання та навички з ремонту, обслуговування та налаштування сучасних автомобілів за допомогою віртуальної реальності.

Використання віртуальної реальності для створення інтерактивних навчальних сценаріїв, які дозволяють користувачам знайомитися з будовою автомобіля, основними системами та компонентами, а також отримати практичні навички з ремонту та діагностики.

Створення високоякісних візуалізацій автомобільних систем для детального ознайомлення з принципами їх роботи. Програма дозволяє користувачам поглянути на автомобіль з усіх боків, розібрати компоненти, що дозволяє краще зрозуміти внутрішню структуру.

Надання можливості користувачам безпосередньо взаємодіяти з різними компонентами автомобіля за допомогою VR-контролерів, а також виконувати різноманітні операції, такі як заміна деталей, налаштування систем, або використання інструментів для ремонту.

4.2 Умови застосування

Програма вимагає наявності сумісного VR-шолома, такого як Meta Quest, HTC Vive, Oculus Rift або аналогічних пристроїв, що підтримують Unity XR.

Мінімальними вимогами до ПК є:

- процесор: Intel Core i5 або аналогічний;
- оперативна пам'ять: 8 GB або більше;
- відеокарта: NVIDIA GeForce GTX 1060 або вище.

Для ефективного використання програми користувачам рекомендовано мати базові знання про принципи роботи автомобільних систем.

Користувачі повинні забезпечити достатньо простору для рухів у VR-середовищі, оскільки програма передбачає використання фізичних рухів для взаємодії з автомобільними компонентами.

Рекомендується робити перерви під час використання програми для зменшення навантаження на очі та нервову систему.

4.3 Можливості та обмеження

Інтерактивне навчання з можливістю розбирання і збирання автомобільних компонентів. Використання VR-технологій для забезпечення максимального занурення та реалістичності процесу навчання. Програма не замінює реальні практичні навички, а є доповненням до теоретичного навчання. Потрібна спеціальна апаратура для коректної роботи (VR-шолом та сумісний комп'ютер). Програма не підтримує всіх типів автомобілів, обмежуючись лише моделями, створеними в межах проєкту. Застосунок ще на стадії прототипу і не має всього необхідного функціоналу.

4.4 Приклади роботи користувача з системою

Після запуску системи користувач може ходити по стартовій кімнаті і вибрати локацію та навчальний сценарій що показано на рисунку 4.1



Рисунок 4.1 – Екран для вибору рівня та локації

Рисунок 4.1 показує реалізований в застосунку спосіб керування сценаріями які користувач хоче змодельовати. Всього на екрані 2 пари стрілок, перші(верхні) дозволяють змінювати локації, другі(нижні) обрати навчальний сценарій, тобто що в автомобілі зламане і потребує ремонту.

Після того як користувач вибирає він завантажується на іншу сцену, залежно від того що було обрано і бачить перед собою сцену як зображено на рисунку 4.2.

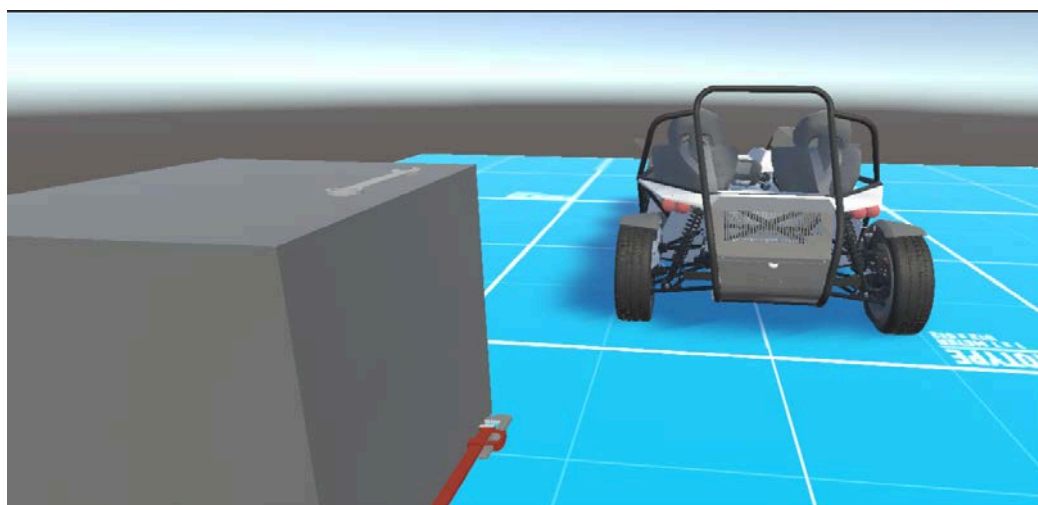


Рисунок 4.2 – Приклад завантаженої сцени

Далі користувачу відкриті можливості зміни взаємодії з автомобілем, він може розбирати і збирати назад частини автомобіля за заздалегідь прописаним сценарієм як показано на рисунках 4.3 – 4.5.

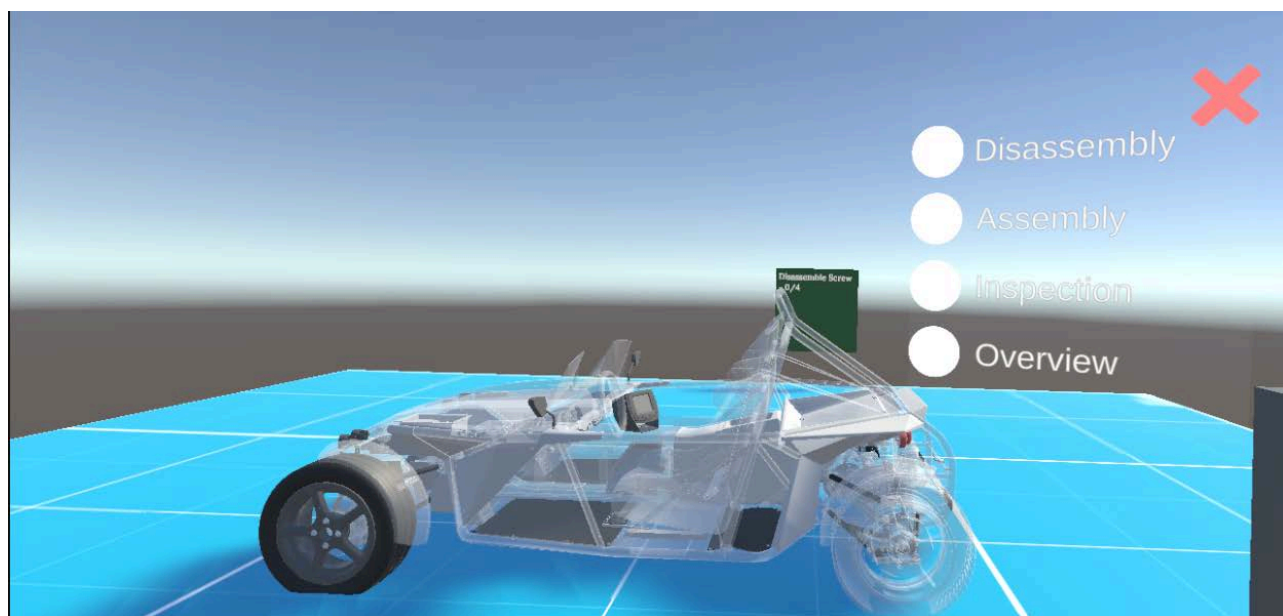


Рисунок 4.3 – Користувач перейшов в режим огляду(Inspection mode)

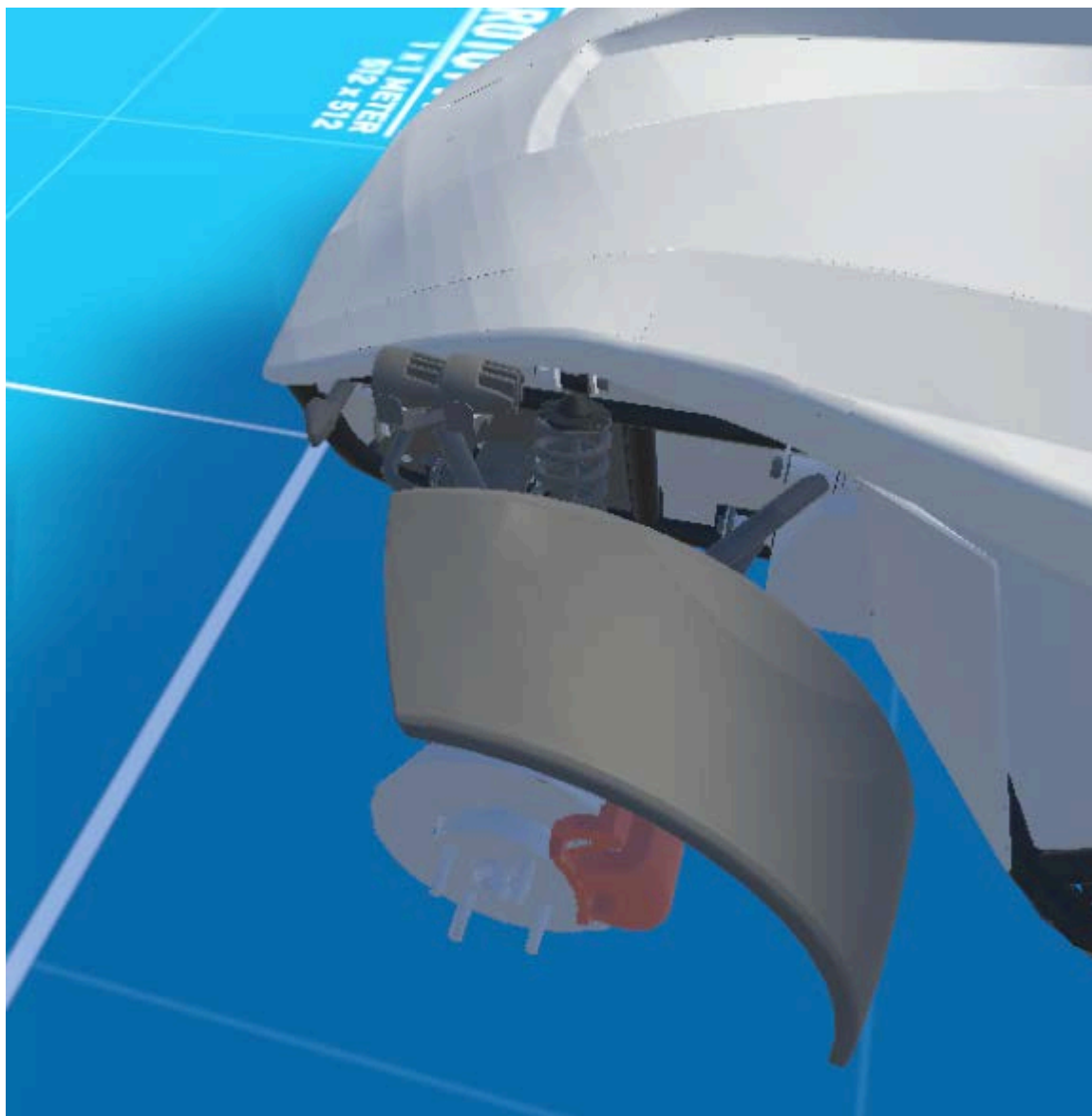


Рисунок 4.4 – Користувач розібрав колесо

Як можемо бачити на рисунку 4.4 слідуючи візуальним підказкам користувач може розібрати колесо автомобіля.



Рисунок 4.5 – Користувач зібрав колесо

Залежно від бажання користувач може спробувати піти у зворотному шляху і поставити деталь назад, як зображено на рисунку 4.5.

Коли користувач закінчить з обраним рівнем, він може або повернутися в лоббі через головне меню(рис. 4.6) натиснувши іконку будинку, або у головному меню перейти у розділ Application та на поточній сцені поміняти сценарій(рис 4.7), або ж вийти з застосунку натиснувши кнопку Exit на головному меню(рис 4.6)

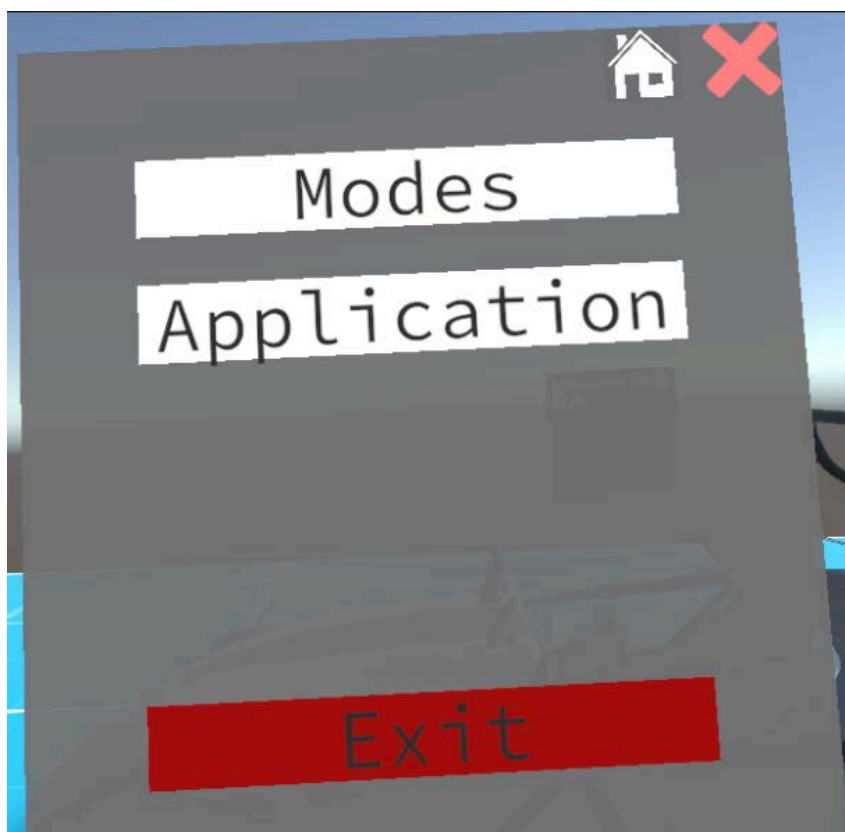


Рисунок 4.6 – Головне меню

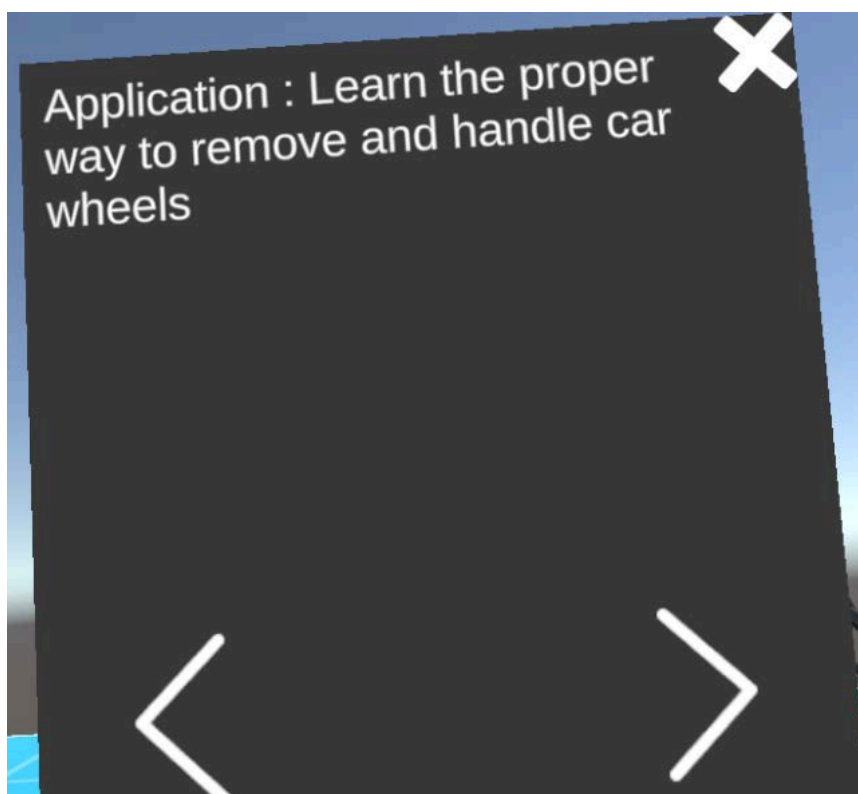


Рисунок 4.7 – Розділ Application головного меню

4.5 Тестування програмного забезпечення

Тестування програмного забезпечення є критично важливим етапом розробки, оскільки дозволяє виявити помилки, недоліки та несправності у програмі ще на етапі розробки, що забезпечує стабільність і функціональність кінцевого продукту. Для розробки Інтерактивної VR-платформи для візуалізації, навчання та взаємодії з автомобільними системами було розроблено низку тестів для перевірки основних аспектів програми, таких як інтеграція з апаратним забезпеченням, коректність взаємодії користувача з інтерфейсами та функціонування віртуальної реальності.

Для забезпечення всебічного перевіряння програмного забезпечення використовуються різні методи тестування.

Метою функціонального тестування є перевірка, чи виконуються всі функції програми відповідно до специфікацій. Тестуються основні сценарії користувацької взаємодії, такі як: завантаження програми, взаємодія з компонентами автомобіля (наприклад, заміна деталей, взаємодія з інструментами), перевірка правильності роботи анімацій та звуків, навігація по інтерфейсу програми.

Нефункціональне тестування – це тестування параметрів та ефективності програми, до прикладу, продуктивність – перевірка швидкості завантаження та рендерингу сцен, плавності анімацій та стабільності роботи.

Метою інтеграційного тестування є перевірка коректності взаємодії між різними компонентами програми, системою керування віртуальним інтерфейсом, взаємодія між користувачем і об'єктами на сцені, тестування сумісності з VR-пристроями, тестування користувацького інтерфейсу (UI) – перевірка правильності відображення елементів інтерфейсу в VR, а також зручність їх взаємодії з користувачем, відображення інформаційних панелей, кнопок та інших елементів інтерфейсу, перевірка зручності взаємодії з інтерфейсом через VR-контролери, перевірка роботи програми на різних

операційних системах та з різними VR-пристроями, перевірка сумісності з різними версіями операційних систем, підтримка різних VR-пристроїв, таких як Oculus Rift, Steam VR, Meta Quest.

Ручне тестування є необхідним для перевірки специфічних сценаріїв, які складно автоматизувати, зокрема для тестування взаємодії з VR-пристроями та перевірки коректності відображення віртуального середовища. Ручне тестування проводиться тестувальниками та користувачами з різним досвідом роботи з VR.

Для того, щоб переконатися, що програма працює коректно на всіх цільових платформах, здійснюється тестування на різних пристроях і операційних системах, включаючи різні моделі VR-шоломів.

Для тестування взаємодії з VR-пристроями були використані емулятори VR, щоб перевірити, чи коректно працює інтеграція з фізичними пристроями.

Для забезпечення високої продуктивності програми, важливо провести тестування в реальних умовах, щоб переконатися, що програма не зависає та не демонструє погіршення продуктивності при виконанні складних операцій.

Перевіряється плавність роботи програми, вимірюється середній фреймрейт в процесі взаємодії з різними об'єктами та використання інструментів. Вимірюється також використання пам'яті програмою під час роботи з великими сценами.

Після проведення тестування кожен виявлений баг або помилка фіксується та оцінюється за критичністю, критичні помилки виправляються в першу чергу, оскільки вони можуть суттєво впливати на стабільність програми або її функціональність, незначні помилки фіксуються для подальшого виправлення. Це можуть бути незначні проблеми в UI або незначні баги, які не впливають на основну функціональність.

4.6 Висновок до розділу 4

Розділ охоплює ключові аспекти експлуатації, тестування та експериментального дослідження інтерактивної VR-платформи для навчання та взаємодії з автомобільними системами. Було визначено цільове призначення програми, технічні та експлуатаційні вимоги, а також її можливості й обмеження. Значну увагу приділено тестуванню програмного забезпечення, що охоплює функціональне, нефункціональне, інтеграційне та ручне тестування для забезпечення стабільної роботи системи. Ретельна перевірка на сумісність із різними VR-пристроями, операційними системами та сценаріями взаємодії дозволила виявити та усунути критичні помилки на ранньому етапі розробки. Таким чином, виконане тестування підтверджує працездатність і ефективність розробленої платформи, що є важливим кроком до подальшого вдосконалення та впровадження системи в навчальний процес.

ВИСНОВКИ

У результаті виконання роботи було створено інтерактивну VR-платформу для навчання та тестування автомобільних систем, яка дозволяє здійснювати віртуальну взаємодію з різними компонентами автомобіля. Метою роботи було створення прототипу, який би спрощував процес навчання та обслуговування автомобільних систем, використовуючи технології віртуальної реальності.

Розроблено VR-прототип застосунку, в якому реалізовані базові сценарії для розбору автомобіля, що дозволяють користувачам взаємодіяти з окремими частинами автомобіля, вивчаючи їх будову та принципи роботи. Для розробки платформи використовувалися сучасні технології, зокрема OpenXR та Meta Quest 2, що забезпечують ефективне тестування та розробку інтерфейсу, а також взаємодію з компонентами VR-системи.

Перспективи подальшого розвитку системи включають можливість розширення платформи для підтримки нових автомобільних моделей та сценаріїв взаємодії, зокрема для роботи з великими і важливими деталями, такими як двигун, коробка передач або електрична система. Крім того, у майбутньому можливо впровадження мультиплеєрного режиму для виконання спільних віртуальних завдань, що ще більше підвищить ефективність навчання та тестування.

Розробка цієї платформи є важливим кроком у напрямку інтерактивного навчання в сфері автомобільних технологій і може стати основою для подальших інновацій у галузі віртуального навчання та симуляцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. OpenXR Specifications [Electronic resource]. – Access mode : <https://www.khronos.org/registry/OpenXR/>.
2. Meta Quest Developer Documentation [Electronic resource]. – Access mode : <https://developer.oculus.com/>.
3. Unity XR Documentation [Electronic resource]. – Access mode : <https://docs.unity3d.com/Manual/XR.html>.
4. Unity Input System [Electronic resource]. – Access mode : <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/index.html>.
5. Unity Asset Store (VR Assets) [Electronic resource]. – Access mode : <https://assetstore.unity.com/>
6. Ford Immersive Vehicle Environment (FiVE) [Electronic resource]. – Access mode : <https://newatlas.com/ford-immersive-vehicle-environment-five-lab-vr/33952/>.
7. Siemens NX VR Design Review [Electronic resource]. – Access mode : <https://blogs.sw.siemens.com/nx-design/immersive-design-review-in-virtual-reality-with-nx/>.
8. Car Mechanic Simulator VR [Electronic resource]. – Access mode : <https://gameformatic.com/guidelines/>
9. Virtual Reality: Immersive Design Review [Electronic resource]. – Access mode : <https://community.sw.siemens.com/s/article/virtual-reality-immersive-design-review>.
10. NVIDIA VRWorks SDK [Electronic resource]. – Access mode : <https://developer.nvidia.com/vrworks>.
11. Pico VR SDK (for VR Development) [Electronic resource]. – Access mode : <https://developer.pico-interactive.com/>.
12. Google Cardboard SDK [Electronic resource]. – Access mode : <https://developers.google.com/cardboard>.

13. VRTK (Virtual Reality Toolkit) [Electronic resource]. – Access mode : <https://github.com/ExtendRealityLtd/VRTK>.
14. C# Documentation [Electronic resource]. – Access mode : <https://learn.microsoft.com/en-us/dotnet/csharp/>
15. Rust Documentation [Electronic resource]. – Access mode : <https://doc.rust-lang.org/std/index.html>
16. Unity Documentation [Electronic resource]. – Access mode : <https://docs.unity.com/>
17. Unreal Engine Documentation [Electronic resource]. – Access mode : <https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation>
18. Godot Documentation [Electronic resource]. – Access mode : <https://docs.godotengine.org/en/stable/>

ДОДАТОК А
Технічне завдання

Сучасні технології віртуальної реальності (VR) значно змінюють підходи до навчання та тестування, зокрема в таких галузях, як автомобільна інженерія та механічне обслуговування. Використання VR дозволяє створювати інтерактивні платформи, що забезпечують глибоке занурення у віртуальне середовище, де користувач може навчатися взаємодії з автомобільними системами, аналізувати складні механізми та проводити діагностику без необхідності фізичного контакту з реальними об'єктами.

Цей застосунок описує розробку інтерактивної VR-платформи для навчання та тестування автомобільних систем. Проєкт передбачає створення прототипу VR-застосунку для розбору автомобіля, який використовує технології OpenXR та Meta Quest 2, забезпечуючи реалістичне навчальне середовище для майбутніх інженерів і механіків.

A.1 Підстава для розробки

Необхідність розробки інтерактивної VR-платформи для навчання та тестування автомобільних систем обумовлена швидким розвитком технологій віртуальної реальності та потребою в нових підходах до навчання в галузі автомобільної інженерії. Традиційні методи навчання, що включають фізичне обслуговування реальних автомобільних компонентів, мають обмеження щодо безпеки, доступу до необхідних матеріалів і засобів, а також високих витрат на реалістичне обладнання та середовище для навчання.

Розробка VR-платформи дозволяє мінімізувати ці проблеми, надаючи користувачам можливість працювати з реалістичними віртуальними моделями автомобілів, взаємодіяти з деталями, проводити діагностику та ремонти в умовах, що наближені до реальних, без потреби в реальному обладнанні.

А.2 Призначення розробки

Призначенням цієї розробки є створення інтерактивної VR-платформи для навчання та тестування автомобільних систем.

Платформа надає користувачам можливість:

- ознайомитися з основними автомобільними системами та їх складовими;
- виконувати операції з розбору та складання автомобіля в реалістичному віртуальному середовищі;
- проводити навчання та тестування з ремонту, діагностики та обслуговування автомобільних систем;
- розвивати навички взаємодії з автомобільними компонентами через віртуальні сценарії.

Ця платформа також може бути використана для створення нових навчальних матеріалів, зокрема для автомобільних техніків і студентів інженерних спеціальностей, а також для тестування нових технологій в автомобільній індустрії.

А.3 Основні вимоги до програми, що розробляється

Користувач повинен мати можливість взаємодіяти з віртуальними моделями автомобільних компонентів за допомогою VR-контролерів.

Платформа має забезпечувати високу якість візуалізації та фізичної симуляції, що наближає навчання до реальних умов.

Можливість додавання нових сценаріїв та автомобільних моделей для подальшого розширення функціональності.

Платформа повинна бути зручна в користуванні, з чітким та зрозумілим інтерфейсом для користувачів різного рівня підготовки.

Структура системи має бути розроблена таким чином, щоб її можна було легко адаптувати для додавання нових можливостей, таких як мультиплеер чи додаткові сценарії.

Платформа повинна підтримувати різні VR-пристрої, зокрема Meta Quest 2, забезпечуючи плавний досвід користування.

Система має бути спроектована з урахуванням фізичного комфорту та безпеки користувача, з урахуванням потенційних ризиків та стресу при тривалому використанні.

A.4 Вхідні дані до роботи

Для розробки віртуальних автомобільних систем використовуються детальні 3D-моделі автомобілів та їх компонентів, які повинні бути точно відтворені для навчання.

Розробка інтерфейсів для взаємодії з автомобільними деталями вимагає створення або імплементації інструментів (наприклад, гайкових ключів, викруток, спеціалізованих ремонтних пристроїв).

Для створення навчальних сценаріїв та інструкцій використовуються реальні технічні дані та інструкції по ремонту автомобільних систем.

Сценарії для навчання повинні включати покрокові інструкції, а також симуляцію поширених поломок і технічних проблем, з якими користувач може зіштовхнутися під час ремонту автомобіля.

Для використання застосунку використовуються VR-пристрої з відповідними контролерами для перевірки інтерфейсу та досвіду взаємодії.

ДОДАТОК Б
Фрагмент тексту програми

AnimationHandler.cs

```

using System.Linq;
using System.Threading.Tasks;
using UnityEngine;

public class AnimationHandler : MonoBehaviour {
    [SerializeField]
    private Animator m_animator;
    public Animator Animator => m_animator;

    private void Start() {
        m_animator = GetComponent<Animator>();
    }

    public async Task PlayAnimationAndWait(string animationName) {
        m_animator.SetTrigger(animationName);

        await WaitForAnimation(m_animator, animationName);
    }

    private async Task WaitForAnimation(Animator animator, string animationName) {
        var animationState = animator.GetCurrentAnimatorStateInfo(0);

        while(!animationState.IsName(animationName)) {
            await Task.Yield();
            animationState = animator.GetCurrentAnimatorStateInfo(0);
        }

        float animationLength = animationState.length;

        await Task.Delay((int)(animationLength * 1000));
    }
}

```

AssignPartMesh.cs

```

using UnityEditor;
using UnityEngine;

```

```

public class AssignPartMesh : MonoBehaviour
{
    [MenuItem("Tools/Assign Part Mesh")]
    public static void GenerateAndAssignPartInfo()
    {
        Object[] selectedPrefabs = Selection.objects;

        foreach (Object obj in selectedPrefabs)
        {
            if (obj is GameObject prefab)
            {
                var part = prefab.GetComponent<Part>();
                var partMeshFilter = prefab.GetComponentInChildren<MeshFilter>();

                if (part.PartInfo.PartMesh == null)
                {
                    Debug.LogWarning($"Skipping {prefab.name}: No valid PartInfo
reference found.");
                    continue;
                }

                if (partMeshFilter == null)
                {
                    Debug.LogWarning($"Skipping {prefab.name}: No MeshFilter
component found.");
                    continue;
                }

                if (part.PartInfo.PartMesh == null)
                {
                    Debug.LogWarning($"Skipping {prefab.name}: PartInfo does not
contain a valid mesh.");
                    continue;
                }

                // Modify prefab correctly
                string path = AssetDatabase.GetAssetPath(prefab);
                GameObject prefabInstance =
PrefabUtility.LoadPrefabContents(path);

```

```

        if (prefabInstance != null)
        {
            MeshFilter instanceMeshFilter =
prefabInstance.GetComponentInChildren<MeshFilter>();
            if (instanceMeshFilter != null)
            {
                instanceMeshFilter.sharedMesh = part.PartInfo.PartMesh;
                PrefabUtility.SaveAsPrefabAsset(prefabInstance, path);
                Debug.Log($"Assigned mesh to {prefab.name} and saved
changes.");
            }
            PrefabUtility.UnloadPrefabContents(prefabInstance);
        }
    }
}

AssetDatabase.Refresh();
Debug.Log("Finished assigning meshes to selected objects.");
}
}

```

AudioManager.cs

```

using Cysharp.Threading.Tasks;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AudioManager : MonoBehaviour
{
    public static AudioManager instance;

    void Start()
    {
        if(instance != null)
        {
            Destroy(this);
        }
        instance = this;
    }
}

```

```

public static async UniTaskVoid PlaySound(Transform position, AudioClip sound)
{
    GameObject obj = new GameObject("TempAudio");
    AudioSource source = obj.AddComponent<AudioSource>();
    obj.transform.position = position.position;

    source.PlayOneShot(sound);
    await UniTask.Delay((int)(sound.length * 1000)); // Wait for the clip
duration
    Destroy(obj); // Clean up after sound finishes
}
}

```

Billboard.cs

```

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

public class Billboard : MonoBehaviour
{
    [SerializeField] private Transform m_target;

    public void SetTarget(Transform target)
    {
        m_target = target;
    }

    private void LateUpdate()
    {
        if (!m_target)
        {
            return;
        }
        else
        {
            Vector3 targetPosition = Camera.main.transform.position;

```

```

        // Calculate the direction to the camera
        Vector3 directionToCamera = transform.position - targetPosition;

        // Make sure the UI faces the camera directly
        transform.rotation = Quaternion.LookRotation(directionToCamera);
    }
}
}

```

CameraValidation.cs

```

using Unity.XR.CoreUtils;
using UnityEngine;

public class CameraValidation : MonoBehaviour
{
    private CharacterController m_characterController;
    private XROrigin m_xrOrigin;

    void Start()
    {
        m_characterController = GetComponent<CharacterController>();
        m_xrOrigin = GetComponent<XROrigin>();
    }

    private void FixedUpdate()
    {
        m_characterController.height = m_xrOrigin.CameraInOriginSpaceHeight +
0.15f;

        var centerPoint =
transform.InverseTransformPoint(m_xrOrigin.Camera.transform.position);

        m_characterController.center = new Vector3(
            centerPoint.x,
            m_characterController.height / 2 + m_characterController.skinWidth,
            centerPoint.z);
    }
}

```

```

        //Done to make sure we don't allow the headset to pass through physical
objects, validate right physic interaction
        m_characterController.Move(new Vector3(0.001f, -0.001f, 0.01f));
        m_characterController.Move(new Vector3(-0.001f, 0.001f, -0.01f));
    }
}

```

Car.cs

```

using Cysharp.Threading.Tasks;
using System;
using System.Collections.Generic;
using System.Linq;
using Tutorials;
using UnityEngine;

public class Car : MonoBehaviour {
    [SerializeField] private CarStateManager m_stateManager;
    [SerializeField] private Material m_inactiveMaterial;

    private LevelInfoHolder m_levelInfoHolder;
    private TutorialManager m_tutorialManager;

    private CarPartManager m_partManager;
    public CarStateManager StateManager => m_stateManager;

    private void Awake() {
        if(!Container.TryGetInstance<LevelInfoHolder>(out m_levelInfoHolder)) {
            Debug.LogError($"Unable to instance of type
{m_levelInfoHolder.GetType()} in {this.GetType()}");
            return;
        }
        if(!Container.TryGetInstance<TutorialManager>(out m_tutorialManager)) {
            Debug.LogError($"Unable to instance of type
{m_tutorialManager.GetType()} in {this.GetType()}");
            return;
        }
        m_levelInfoHolder.LevelInfoChanged += OnLevelInfoChanged;
        OnLevelInfoChanged(m_levelInfoHolder.CurrentLevelInfo);
    }
}

```

```

private async void OnLevelInfoChanged(LevelInfo info) {
    await SetCar(info);
    SetTutorial();
}

private List<CarPart> GetAllCarParts() {
    return GetComponentsInChildren<CarPart>().ToList();
}

private async UniTask SetCar(LevelInfo info) {
    List<CarPart> tmp = new();
    if(info == null) {
        Debug.LogError($"{this.name} was unable to set the car due to lack of
LevelInfo");
        return;
    }
    if(m_partManager != null) {
        await m_partManager.RestoreToDefault();
        tmp = m_partManager.AllParts;
    }
    if(tmp.Count == 0) {
        tmp = GetAllCarParts();
    }
    m_partManager = new(tmp, info.brokenPartType);
    m_stateManager = new(m_partManager, m_inactiveMaterial);
}

private void SetTutorial() {
    var providers = m_partManager.BrokenParts.ToList<ITutorialProvider>();
    providers.Reverse();
    m_tutorialManager.SetTutorial(providers);
}

private void OnDestroy() {
    m_levelInfoHolder.LevelInfoChanged -= OnLevelInfoChanged;
}
}

public class CarStateManager {
    private Material m_inactiveMaterial;
    private CarPartManager m_partManager;

```

```

public CarStateManager(CarPartManager partManager, Material inactive) {
    m_partManager = partManager;
    m_partManager.PartAssembled += UpdateAssemblyMode;
    m_inactiveMaterial = inactive;
}

public void EnterOverviewMode() {
    Debug.Log("Entered Overview Mode");
    foreach(var part in m_partManager.AllParts) {
        if(!part.gameObject.TryGetComponent<CarPartInteractable>(out
CarPartInteractable interactable)) {
            continue;
        }
        interactable.ResetRendererMaterialsToDefault();
        interactable.SetInteraction();
    }
}

public void EnterInspectionMode() {
    foreach(var part in m_partManager.AllParts) {
        if(!part.TryGetComponent<CarPartInteractable>(out CarPartInteractable
interactable)) {
            continue;
        }
        if(!m_partManager.BrokenParts.Contains(part)) {
            interactable.SetRendererMaterialsTo(m_inactiveMaterial);
        }
    }
}

public void ExitInspectionMode() {
    foreach(var part in m_partManager.AllParts) {
        if(!part.TryGetComponent<CarPartInteractable>(out CarPartInteractable
interactable)) {
            continue;
        }
        interactable.ResetRendererMaterialsToDefault();
    }
}

public void EnterDisassemblyMode() {
    foreach(var part in m_partManager.AllParts) {

```

```

        if(!part.gameObject.TryGetComponent<CarPartInteractable>(out
CarPartInteractable interactable)) {
            continue;
        }

        if(m_partManager.BrokenParts.Count <= 0) {
            interactable.SetInteraction(true);
        }
        else if(m_partManager.BrokenParts.Contains(part)) {
            interactable.SetInteraction(true);
        }
    }
}

public void ExitDisassemblyMode() {
    foreach(var part in m_partManager.AllParts) {
        if(!part.gameObject.TryGetComponent<CarPartInteractable>(out
CarPartInteractable interactable)) {
            continue;
        }

        if(m_partManager.BrokenParts.Count <= 0) {
            interactable.SetInteraction();
        }
        else if(m_partManager.BrokenParts.Contains(part)) {
            interactable.SetInteraction();
        }
    }
}

public void EnterAssemblyMode() {
    foreach(var part in m_partManager.RemovedParts) {
        if(!part.TryGetComponent<CarPartInteractable>(out CarPartInteractable
interactable)) {
            continue;
        }
        if(part.CanBeAssembled) {
            part.gameObject.SetActive(true);
        }
        interactable.SetRendererMaterialsTo(m_inactiveMaterial);
        interactable.SetInteraction(canAssemble: true);
    }
}

```

```

    }
}

public void ExitAssemblyMode() {
    foreach(var part in m_partManager.RemovedParts) {
        if(!part.TryGetComponent<CarPartInteractable>(out CarPartInteractable
interactable)) {
            continue;
        }
        if(part.CanBeAssembled) {
            part.gameObject.SetActive(false);
        }
        interactable.ResetRendererMaterialsToDefault();
        interactable.SetInteraction();
    }
}

public void UpdateAssemblyMode() {
    foreach(var part in m_partManager.RemovedParts) {
        if(!part.TryGetComponent<CarPartInteractable>(out CarPartInteractable
interactable)) {
            continue;
        }
        if(part.CanBeAssembled) {
            part.gameObject.SetActive(true);
        }
        interactable.SetRendererMaterialsTo(m_inactiveMaterial);
        interactable.SetInteraction(canAssemble: true);
    }
}

private void OnDestroy() {
    m_partManager.PartAssembled -= UpdateAssemblyMode;
}

}

public class CarPartManager {
    private CarPartType m_brokenPartsType;
    private List<CarPart> m_parts = new();
    private List<CarPart> m_brokenParts = new();
    private List<CarPart> m_removedCarParts = new();

```

```

public Action PartAssembled;
public List<CarPart> AllParts => m_parts;
public List<CarPart> BrokenParts => m_brokenParts;
public List<CarPart> RemovedParts => m_removedCarParts;

// private Action<CarPart> DisassembledPart;

public CarPartManager(List<CarPart> parts, CarPartType brokenPartType =
CarPartType.None) {
    m_parts = parts;
    Debug.Log($"Number of collected parts = {m_parts.Count}");
    foreach(var part in m_parts) {
        part.Disassembled += OnDisassembled;
        part.Assembled += OnAssembled;
    }
    SetBrokenPartsType(brokenPartType);
    m_brokenParts = GetBrokenPartsByType(m_brokenPartsType, m_parts);
}

public async UniTask RestoreToDefault() {
    List<UniTask> tasks = new();
    Debug.Log("Restore to default started");
    for(int i = m_removedCarParts.Count - 1; i >= 0; i--) {
        m_removedCarParts[i].gameObject.SetActive(true);
        tasks.Add(m_removedCarParts[i].Assemble());
    }

    await UniTask.WhenAll(tasks);

    m_brokenParts.Clear();
    m_removedCarParts.Clear();
    Debug.Log("Restore to default finished work");
}

private void OnAssembled(CarPart part) {
    if(m_removedCarParts.Contains(part)) {
        m_removedCarParts.Remove(part);
    }
    var interactable = part.GetComponent<CarPartInteractable>();
    interactable.ResetRendererMaterialsToDefault();
}

```

```

        interactable.SetInteraction();
        PartAssembled?.Invoke();
    }

    private void OnDisassembled(CarPart part) {
        if(m_removedCarParts.Contains(part)) {
            return;
        }
        m_removedCarParts.Add(part);
    }

    public void SetBrokenPartsType(CarPartType type) { m_brokenPartsType = type; }

    public List<CarPart> GetBrokenPartsByType(CarPartType brokenPartsType,
List<CarPart> carParts) {
        List<CarPart> parts = new();
        foreach(var part in carParts) {
            if(part.PartInfo && part.PartInfo.GetCarPartType == m_brokenPartsType)
        {
                parts.AddRange(part.GetAllDependableParts());
                //TODO: changed it to get only one broken detail of a type;
                return parts;
            }
        }
        return parts;
    }

    private void OnDestroy() {
        foreach(var part in m_parts) {
            part.Disassembled -= OnDisassembled;
            part.Assembled -= OnAssembled;
        }
    }
}

```

CarFactory.cs

```

using Cysharp.Threading.Tasks;

```

```

using UnityEngine;
using SceneManagement;
using UnityEngine.UIElements;
public class CarFactory : MonoBehaviour
{
    //[SerializeField] private Transform m_spawnPosition;
    private LevelInfo m_levelInfo;
    private GameObject m_currentCar;

    private LevelInfoHolder m_holder;

    private void Awake()
    {
        if(Container.TryGetInstance<LevelInfoHolder>(out var holder))
        {
            m_holder = holder;
            TakeLevelInfo(m_holder.CurrentLevelInfo);
            CreateInstance();
            /*m_holder.LevelInfoChanged += OnLevelInfoChanged;
            OnLevelInfoChanged(m_holder.CurrentLevelInfo);*/
            return;
        }
        Debug.LogError($"{this.name} was unable to create a anything because
there is no LevelInfoHolder in Container");
    }

    private void TakeLevelInfo(LevelInfo info)
    {
        m_levelInfo = info;
    }

    private void OnLevelInfoChanged(LevelInfo info)
    {
        TakeLevelInfo(info);
        CreateInstance();
    }

    /*    public GameObject CreateInstance(GameObject carPrefab, Vector3 position,
Quaternion rotation)
    {
        m_currentCar = Instantiate(carPrefab, position, rotation);
    }

```

```

        return m_currentCar;
    }*/
    public void CreateInstance()
    {
        if (m_currentCar)
        {
            Destroy(m_currentCar);
        }
        if (m_holder.CurrentLevelInfo != null)
        {
            m_currentCar = Instantiate(m_levelInfo.carPrefab, transform.position,
transform.rotation);
        }
    }
    public void OnDestroy()
    {
        if (m_holder != null)
        {
            m_holder.LevelInfoChanged -= OnLevelInfoChanged;
        }
    }
}

```

CarPart.cs

```

using Cysharp.Threading.Tasks;
using System;
using System.Collections.Generic;
using System.Linq;
using Tutorials;
using UnityEngine;

public abstract class CarPart : MonoBehaviour, IAssemblyPart, ITutorialProvider {
    [SerializeField] private List<CarPart> m_parentParts;
    [SerializeField] private List<CarPart> m_dependableParts;
    [SerializeField] private PartInfo m_partInfo;

    //private CarPartAnimator m_animator;
    public CarPartAnimator CarPartAnimator { get; private set; }
}

```

```

public List<CarPart> ReadOnlyParentPartsList => m_parentParts;
public bool HasDependableParts => m_dependableParts.Count > 0;
public bool disassembled = false;
public PartInfo PartInfo {
    get => m_partInfo;
    set => m_partInfo = value;
}

public bool CanBeAssembled {
    get {
        return m_parentParts != null && m_parentParts.All(part =>
part.gameObject.activeSelf);
    }
}

public Action<CarPart> Disassembled;
public Action<CarPart> Assembled;

protected virtual void Awake() {
    CarPartAnimator = new(GetComponentInChildren<AnimationHandler>());
    if(m_partInfo != null) {
        GetComponentInChildren<MeshFilter>().mesh = m_partInfo.PartMesh;
    }
    if(!HasDependableParts) {
        return;
    }
    foreach(var part in m_dependableParts) {
        part.SetParent(this);
    }
}

public List<CarPart> GetAllDependableParts() {
    List<CarPart> parts = new();
    parts.Add(this);
    CollectAllDependablePartsRecursive(this, parts);
    return parts;
}

private void CollectAllDependablePartsRecursive(CarPart part, List<CarPart>
parts) {
    foreach(var dependablePart in part.m_dependableParts) {

```

```

        if(!parts.Contains(dependablePart)) // Avoid duplicates
        {
            parts.Add(dependablePart);
            CollectAllDependablePartsRecursive(dependablePart, parts);
        }
    }
}

public virtual void SetParent(CarPart parent) {
    m_parentParts.Add(parent);
    Disassembled += parent.ReleaseChildren;
    Assembled += parent.SetChildren;
}

public virtual void SetChildren(CarPart partInteractable) {
    m_dependableParts.Add(partInteractable);
}

public virtual void ReleaseChildren(CarPart partInteractable) {
    m_dependableParts.Remove(partInteractable);
}

public virtual async UniTask StartAssemble() {
    if(!CanBeAssembled) {
        return;
    }
    await Assemble();
}

public virtual async UniTask Assemble() {
    //if(m_animator.AnimationHandler) {
    //    await
m_animator.AnimationHandler.PlayAnimationAndWait(m_animator.AssembleAnimationName)
;
    //}
    disassembled = false;
    Assembled?.Invoke(this);
}

public virtual async UniTask StartDisassemble() {
    if(HasDependableParts) {

```

```

        return;
    }
    await Disassemble();
}

public async UniTask Disassemble() {
    /*if(m_animator.AnimationHandler) {
        await
m_animator.AnimationHandler.PlayAnimationAndWait(m_animator.DisassembleAnimationNa
me);
    }*/

    Disassembled?.Invoke(this);
    disassembled = true;
    gameObject.SetActive(false);
    /*      var command = new HideCommand(gameObject);
        CommandHandler.ExecuteCommand(command);*/
}

public virtual TutorialStep GetTutorialStep() {
    if(disassembled) {
        return new TutorialStep($"Assemble {gameObject.name}", new
AssembledCondition(this));
    }
    return new TutorialStep($"Disassemble {gameObject.name}", new
DisassembledCondition(this));
}
}

public class CarPartAnimator
{
    [SerializeField] protected AnimationHandler m_animationHandler;

    public AnimationHandler AnimationHandler => m_animationHandler;
    public CarPartAnimator(AnimationHandler animationHandler)
    {
        m_animationHandler = animationHandler;
    }

    private const string DISASSEMBLE_ANIMATION_NAME = "Disassemble";
    private const string ASSEMBLE_ANIMATION_NAME = "Assemble";

```

```

    public string AssembleAnimationName => ASSEMBLE_ANIMATION_NAME;
    public string DisassembleAnimationName => DISASSEMBLE_ANIMATION_NAME;
}

```

CarPartInteractable.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class CarPartInteractable : XRSimpleInteractable {
    [SerializeField] private CarPart m_part;

    [SerializeField] private List<Renderer> m_renderers = new List<Renderer>();
    private List<Material> m_defaultMaterials = new List<Material>();

    public Action<HoverEnterEventArgs> HoverEntered;
    public Action<HoverExitEventArgs> HoverExited;

    [SerializeField]
    private bool isFixed = false;

    public CarPart CarPart => m_part;
    public bool CanBeDisassembled { get; set; }
    public bool CanBeAssembled { get; set; }

    public CarPartType CarPartType => m_part.PartInfo.GetCarPartType;

    protected override void Awake() {
        base.Awake();
        Initialize();
    }

    private void Initialize() {
        m_part = GetComponent<CarPart>();
        m_renderers = GetComponentsInChildren<Renderer>().ToList();
        m_defaultMaterials = GetMaterialsFromRenderers(m_renderers);
    }
}

```

```

    }

    protected void Start() {
        SubscribeToParentEvents();
    }

    private List<Material> GetMaterialsFromRenderers(List<Renderer> renderers) {
        foreach(var renderer in renderers) {
            m_defaultMaterials.Add(renderer.material);
        }
        return m_defaultMaterials;
    }

    private void SubscribeToParentEvents() {
        foreach(var part in m_part.ReadOnlyParentPartsList) {
            part.GetComponent<CarPartInteractable>().HoverEntered +=
OnHoverEntering;
            part.GetComponent<CarPartInteractable>().HoverExited +=
OnHoverExiting;
        }
    }

    protected override void OnHoverEntering(HoverEnterEventArgs args) {
        var interactor = args.interactorObject as ToolInteractor;
        var info = interactor.GetComponentInParent<InteractionInfo>();
        if(info == null) {
            return;
        }
        if(isFixed) {
            return;
        }
        if(!CanBeDisassembled) {
            return;
        }
        if(interactor.SupportedTypes.Contains(m_part.PartInfo.GetCarPartType)) {
            Material targetMaterial = m_part.HasDependableParts ?
info.GetHoverMaterials.ReadOnlyWrongMaterial :
info.GetHoverMaterials.ReadOnlyRightMaterial;
            ChangeMaterial(targetMaterial);
        }
        else {

```

```

        Material targetMaterial =
info.GetHoverMaterials.ReadOnlyWrongToolMaterial;
        ChangeMaterial(targetMaterial); ;
    }

    HoverEntered?.Invoke(args);
    //base.OnHoverEntering(args);
}

protected override void OnHoverExiting(HoverExitEventArgs args) {
    if(!CanBeDisassembled) {
        Debug.Log("Hover exiting BLOCKED as we are not in Disassembly mode");
        return;
    }
    Debug.Log("Hover exiting!! ");
    ResetMaterials();

    HoverExited?.Invoke(args);
    //base.OnHoverExiting(args);
}

public void OnActivate(ActivateEventArgs args) {
    if(CanBeDisassembled && !m_part.HasDependableParts && !isFixed) {
        ResetMaterials();
        //SetInteraction();
        //m_part.StartDisassemble();
    }
    else if(CanBeAssembled && m_part.CanBeAssembled && !isFixed) {
        ResetMaterials();
        //SetInteraction();
        //m_part.StartAssemble();
    }
}

private void ResetMaterials() {
    if(m_renderers == null) {
        Debug.LogError($"There is no Renderers in {this.name}");
        return;
    }
    ResetRendererMaterialsToDefault();
}

```

```

public void ChangeMaterial(Material material) {
    SetRendererMaterialsTo(material);
}

public void ResetRendererMaterialsToDefault() {
    int defaultMaterialsIndex = 0;
    foreach(var renderer in m_renderers) {
        renderer.material = m_defaultMaterials[defaultMaterialsIndex];
        ++defaultMaterialsIndex;
    }
}

public void SetRendererMaterialsTo(Material material) {
    foreach(var renderer in m_renderers) {
        renderer.material = material;
    }
}

public void SetInteraction(bool canDisassemble = false, bool canAssemble =
false) {
    CanBeDisassembled = canDisassemble;
    CanBeAssembled = canAssemble;
}

public void UpdateProgress(float value) {
    if(m_part.CarPartAnimator.AnimationHandler.Animator == null)
        return;

    string animStateName;
    if(m_part.disassembled) {
        animStateName = m_part.CarPartAnimator.AssembleAnimationName;
    }
    else {
        animStateName = m_part.CarPartAnimator.DisassembleAnimationName;
    }

    m_part.CarPartAnimator.AnimationHandler.Animator.Play(animStateName, 0,
value);

    m_part.CarPartAnimator.AnimationHandler.Animator.speed = 0f;
}

```

```
    }
}
```

CarPartType.cs

```
using System;
using System.Linq;
using Unity.VisualScripting;
using UnityEngine;

namespace Assets.Scripts
{
    public enum CarPartType
    {
        None,
        Wheel,
        Brake,
        Battery,
        Bolt,
        Nut
    }
}
```

Container.cs

```
using System;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public static class Container
{
    private static readonly Dictionary<Type, object> DependenciesDictionary =
new();

    public static void Register<T>(T instance) where T : class
    {
        Type type = typeof(T);
```

```

        if (DependenciesDictionary.ContainsKey(type))
        {
            Debug.LogWarning($"Type {type} is already registered. Overwriting the
instance.");
        }
        DependenciesDictionary[type] = instance;
        Debug.Log($"Registered instance of type {type}");
    }

    public static void Unregister<T>() where T : class
    {
        Type type = typeof(T);
        if (DependenciesDictionary.TryGetValue(type, out var instance))
        {
            Debug.Log($"Unregistered instance of type {type}");
            DependenciesDictionary.Remove(type);
        }
    }

    public static T GetInstance<T>() where T : class
    {
        Type type = typeof(T);
        if (DependenciesDictionary.TryGetValue(type, out var instance))
        {
            Debug.Log($"Retrieved instance of type {type}");
            return instance as T;
        }

        Debug.LogWarning($"Instance of type {type} not found in container.");
        return null;
    }

    public static bool TryGetInstance<T>(out T instance) where T : class
    {
        Type type = typeof(T);
        if (DependenciesDictionary.TryGetValue(type, out var instance1))
        {
            instance = instance1 as T;
            Debug.Log($"Retrieved instance of type {type}");
            return true;
        }
    }

```

```

        instance = null;
        return false;
    }
}

```

HoverMaterials.cs

```

using System;
using UnityEngine;

namespace Assets.Scripts
{
    [CreateAssetMenu(menuName = "ScriptableObject/HoverMaterials", fileName =
    "NewHoverMaterials", order = 1)]
    [Serializable]
    public class HoverMaterials : ScriptableObject
    {
        [SerializeField] private Material m_rightMaterial;
        [SerializeField] private Material m_wrongMaterial;
        [SerializeField] private Material m_wrongToolMaterial;

        public Material ReadonlyRightMaterial => m_rightMaterial;
        public Material ReadonlyWrongMaterial => m_wrongMaterial;
        public Material ReadonlyWrongToolMaterial => m_wrongToolMaterial;
    }
}

```

InteractionInfo.cs

```

using System.Collections;
using System.Collections.Generic;
using Assets.Scripts;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class InteractionInfo : MonoBehaviour
{
    [SerializeField] private HoverMaterials m_materials;
}

```

```

        public HoverMaterials GetHoverMaterials => m_materials;
    }

```

IUiInstance.cs

```

using Cysharp.Threading.Tasks;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.Accessibility;
using UnityEngine.InputSystem;

namespace Assets.Scripts
{
    public interface IUiInstance
    {
        public GameObject PrefabUiObject { get; }

        public abstract void UpdateUi();

        public abstract void ShowUi();

        public abstract void CloseUi();

        public abstract void SetPosition(Vector3 position);

        public abstract void SetTarget(Transform objectTransform);

        public abstract void SetPrefab(GameObject prefab);

        public abstract UniTask FadeIn(float duration = 0.5f);

        public abstract UniTask FadeOut(float duration = 0.5f);

        public abstract UniTask FadeCanvasGroup(CanvasGroup canvasGroup, float
startAlpha, float targetAlpha, float duration);
    }
}

```

LevelInfo.cs

```
using Assets.Scripts;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(fileName = "New Level Info", menuName =
"ScriptableObject/LevelInfo", order = 1)]
public class LevelInfo : ScriptableObject
{
    public GameObject carPrefab;
    public CarPartType brokenPartType;
    public string levelDescription;
}
```

LevelInfoHolder.cs

```
using ModestTree;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Management;

public class LevelInfoHolder : MonoBehaviour
{
    [SerializeField] private LevelInfo[] m_levelInfo;
    [SerializeField]
    private LevelInfo m_currentLevelInfo;

    public int LevelCount => m_levelInfo.Length;
    public Action<LevelInfo> LevelInfoChanged;
    public LevelInfo CurrentLevelInfo
    {
        get { return m_currentLevelInfo; }
        set { m_currentLevelInfo = value; }
    }
}
```

```

/*    private LevelInfo this[int index]
    {
        get { return m_levelInfo[index]; }
    }*/

private void Start()
{
    Container.Register(this);
    m_currentLevelInfo = m_levelInfo[0];
}

public LevelInfo GetLevelInfo(int index)
{
    return m_levelInfo[index];
}

public int GetLevelInfoIndex(LevelInfo levelInfo)
{
    return m_levelInfo.IndexOf(levelInfo);
}

public void LoadNewLevelInfo(LevelInfo info)
{
    m_currentLevelInfo = info;
    LevelInfoChanged?.Invoke(m_currentLevelInfo);
}
}

```

LevelPresenter.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using TMPPro;
using UnityEngine;
using UnityEngine.UI;

public class LevelPresenter : MonoBehaviour
{
    [SerializeField] private LevelInfoHolder m_levelInfoHolder;

```

```

[SerializeField] private TextMeshProUGUI m_levelDescription;
[SerializeField] private Button m_previousIndexButton;
[SerializeField] private Button m_nextIndexButton;

private int m_index = 0;

private void Start()
{
    if (Container.TryGetInstance<LevelInfoHolder>(out LevelInfoHolder holder))
    {
        m_levelInfoHolder = holder;
        m_index =
m_levelInfoHolder.GetLevelInfoIndex(m_levelInfoHolder.CurrentLevelInfo);
        m_previousIndexButton.onClick.AddListener(PreviousIndex);
        m_nextIndexButton.onClick.AddListener(NextIndex);
        m_levelInfoHolder.LevelInfoChanged += OnLevelInfoChanged;
        UpdateUi();
    }
}

public void NextIndex()
{
    m_index++;
    if (m_index > m_levelInfoHolder.LevelCount - 1)
    {
        m_index = m_levelInfoHolder.LevelCount - 1;
    }

m_levelInfoHolder.LoadNewLevelInfo(m_levelInfoHolder.GetLevelInfo(m_index));
    UpdateUi();
}

public void PreviousIndex()
{
    m_index--;
    if (m_index < 0)
    {
        m_index = 0;
    }
}

```

```

m_levelInfoHolder.LoadNewLevelInfo(m_levelInfoHolder.GetLevelInfo(m_index));
    UpdateUi();
}

private void OnLevelInfoChanged(LevelInfo info)
{
    UpdateUi();
}

private void UpdateUi()
{
    m_levelDescription.text = "Application : " +
m_levelInfoHolder.CurrentLevelInfo.levelDescription;
}
}

```

MenuOpener.cs

```

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

public class MenuOpener : MonoBehaviour
{
    [SerializeField] private InputActionProperty m_menuOpenActionProperty;
    [SerializeField] private GameObject m_mainMenu;

    private UiManager m_uiManager;

    private void Start()
    {
        m_uiManager = Container.GetInstance<UiManager>();
    }

    private void Update()
    {
        if (m_menuOpenActionProperty.action.WasPerformedThisFrame())
        {

```

```

        m_uiManager.OpenUi(m_mainMenu);
    }
}

```

MenuUi.cs

```

using UnityEngine;
using UnityEngine.UI;
using SceneManagement;
using UnityEngine.Events;

public class MenuUi : UiInstance
{
    [SerializeField] private GameObject m_modesUiPrefab;
    [SerializeField] private GameObject m_levelChangeUiPrefab;
    [SerializeField] private Button m_modesButton;
    [SerializeField] private Button m_levelButton;
    [SerializeField] private Button m_closeUiButton;
    [SerializeField] private Button m_returnToLobbyUiButton;
    [SerializeField] private Button m_exitGameButton;

    private UnityAction SceneLoadMethod = () => { };

    private void Start()
    {
        //base.Start();
        if (Container.TryGetInstance<SceneLoader>(out SceneLoader loader))
        {
            if (loader.SceneGroupManager.ActiveSceneGroup != loader[0])
            {
                SceneLoadMethod = async () => await loader.LoadSceneGroup(0);
                m_returnToLobbyUiButton.onClick.AddListener(SceneLoadMethod);
            }
        }
        m_modesButton.onClick.AddListener(ModeButtonPressed);
        m_closeUiButton.onClick.AddListener(this.CloseUi);
        m_exitGameButton.onClick.AddListener(Application.Quit);
        m_levelButton.onClick.AddListener(LevelButtonPressed);
    }
}

```

```

public void ModeButtonPressed()
{
    OpenUi(m_modesUiPrefab);
}
public void LevelButtonPressed()
{
    OpenUi(m_levelChangeUiPrefab);
}

public void OpenUi(GameObject uiPrefab)
{
    Container.GetInstance<UiManager>().OpenUi(uiPrefab);
}
private new void OnDestroy()
{
    base.OnDestroy();
    m_returnToLobbyUiButton.onClick?.RemoveListener(SceneLoadMethod);
    m_modesButton.onClick.RemoveListener(ModeButtonPressed);
    m_closeUiButton.onClick.RemoveListener(this.CloseUi);
    m_exitGameButton.onClick.RemoveListener(Application.Quit);
    m_levelButton.onClick.RemoveListener(LevelButtonPressed);
}
}

```

Part.cs

```

public class Part : CarPart
{
}

```

PartInfo.cs

```

using UnityEngine;

[CreateAssetMenu(menuName = "ScriptableObject/PartInfo", fileName = "New Part
Info", order = 1)]
public class PartInfo : ScriptableObject {

```

```

[SerializeField] private AudioClip m_disassembleSound;
[SerializeField] private AudioClip m_assembleSound;
[SerializeField] private Mesh m_partMesh;
[SerializeField] CarPartType m_partType;

public AudioClip DisassembleAudioClip {
    get => m_disassembleSound;
    set => m_disassembleSound = value;
}
public AudioClip AssembleAudioClip {
    get => m_assembleSound;
    set => m_assembleSound = value;
}
public Mesh PartMesh {
    get => m_partMesh;
    set => m_partMesh = value;
}
public CarPartType GetCarPartType => m_partType;
}

```

PartInfoGenerator.cs

```

using Assets.Scripts;
using UnityEditor;
using UnityEngine;

public class PartInfoGenerator : Editor
{
    [MenuItem("Tools/Generate and Assign Part Info")]
    public static void GenerateAndAssignPartInfo()
    {
        string savePath = "Assets/PartInfo/Nut/";

        if (!AssetDatabase.IsValidFolder(savePath))
        {
            AssetDatabase.CreateFolder("Assets/PartInfo", "Nut");
        }

        Object[] selectedPrefabs = Selection.objects;
    }
}

```

```

foreach (Object prefab in selectedPrefabs)
{
    if (prefab is GameObject prefabGO)
    {
        // Create ScriptableObject
        var partInfo = ScriptableObject.CreateInstance<PartInfo>();
        var part = prefabGO.GetComponent<CarPart>();

        partInfo.AssembleAudioClip = null;
        partInfo.DisassembleAudioClip = null;
        partInfo.PartMesh =
prefabGO.GetComponentInChildren<MeshFilter>().sharedMesh;

        string assetName = prefab.name + "_PartInfo.asset";
        string assetPath = savePath + assetName;

        AssetDatabase.CreateAsset(partInfo, assetPath);
        AssetDatabase.SaveAssets();

        // Assign ScriptableObject to prefab

        if (part != null)
        {
            part.PartInfo = partInfo;

            PrefabUtility.SavePrefabAsset(prefabGO);
        }

        Debug.Log($"Created and assigned PartInfo for {prefab.name}.");
    }
}

AssetDatabase.Refresh();
Debug.Log("Finished generating and assigning PartInfo.");
}
}

```

PlayerFactory.cs

```

using System.Collections;

```

```

using System.Collections.Generic;
using UnityEngine;

public class PlayerFactory : MonoBehaviour
{
    [SerializeField] GameObject m_playerPrefab;
    [SerializeField] Transform m_playerPosition;

    private void Awake()
    {
        CreateInstance(m_playerPosition.position, m_playerPosition.rotation);
    }

    public GameObject CreateInstance(Vector3 position, Quaternion rotation)
    {
        var car = Instantiate(m_playerPrefab, position, rotation);
        return car;
    }
}

```

SceneGroup.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using Eflatun.SceneReference;
using UnityEngine;
using UnityEngine.Video;

namespace SceneManagement
{
    [Serializable]
    public class SceneGroup
    {
        [SerializeField] private List<SceneData> m_scenesData;

        public List<SceneData> Scenes => m_scenesData;

        public string FindSceneByType(SceneType type)
        {

```

```

        return m_scenesData.FirstOrDefault(scene => scene.SceneType ==
type)?.Reference.Name;
    }

    public SceneData FindSceneDataByType(SceneType type)
    {
        return m_scenesData.FirstOrDefault(scene => scene.SceneType == type);
    }
}

[Serializable]
public class SceneData
{
    public SceneReference Reference;
    public string Name => Reference.Name;
    public string LocationName;
    public SceneType SceneType;
    public VideoClip PreviewVideoClip;
}

public enum SceneType
{
    ActiveScene,
    Ui,
    Lobby
}
}

```

SceneGroupManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Cysharp.Threading.Tasks;
using UnityEngine;
using UnityEngine.SceneManagement;

```

```

namespace SceneManagement
{
    public class SceneGroupManager
    {
        public Action<string> OnSceneLoaded;
        public Action<string> OnSceneUnloaded;
        public Action OnSceneGroupLoaded;
        private SceneGroup m_activeSceneGroup;

        public SceneGroup ActiveSceneGroup => m_activeSceneGroup;
        public async UniTask LoadSceneGroup(SceneGroup group)
        {
            m_activeSceneGroup = group;
            var loadedScenes = new List<string>();

            await UnloadSceneGroup();

            int sceneCount = SceneManager.sceneCount;

            for (int i = 0; i < sceneCount; ++i)
            {
                loadedScenes.Add(SceneManager.GetSceneAt(i).name);
            }

            var totalScenesToLoad = m_activeSceneGroup.Scenes.Count;

            var operationGroup = new AsyncOperationGroup(totalScenesToLoad);

            for (int i = 0; i < totalScenesToLoad; ++i)
            {
                var sceneData = m_activeSceneGroup.Scenes[i];

                var operation =
SceneManager.LoadSceneAsync(sceneData.Reference.Path, LoadSceneMode.Additive);
                operationGroup.Operations.Add(operation);

                OnSceneLoaded?.Invoke(sceneData.Name);
            }

            while (!operationGroup.IsDone)
            {

```

```

        await UniTask.Delay(100);
    }

    Scene activeScene =
SceneManager.GetSceneByName(m_activeSceneGroup.FindSceneByType(SceneType.ActiveScene));

    if (activeScene.IsValid())
    {
        SceneManager.SetActiveScene(activeScene);
    }

    OnSceneGroupLoaded?.Invoke();
}

private async UniTask UnloadSceneGroup()
{
    var scenes = new List<string>();

    int sceneCount = SceneManager.sceneCount;
    for (int i = 0; i < sceneCount; i++)
    {
        var sceneAt = SceneManager.GetSceneAt(i);
        if (!sceneAt.isLoaded)
        {
            Debug.Log($"Scene {sceneAt.name} is not loaded fully");
            continue;
        }

        var sceneName = sceneAt.name;
        if (sceneName == "Bootstrapper")
        {
            continue;
        }
        scenes.Add(sceneName);
    }

    var operationGroup = new AsyncOperationGroup(scenes.Count);
    foreach (var scene in scenes)
    {
        var operation = SceneManager.UnloadSceneAsync(scene);
        if (operation == null)

```

```

        {
            continue;
        }
        operationGroup.Operations.Add(operation);
        OnSceneUnloaded?.Invoke(scene);
    }

    while (!operationGroup.IsDone)
    {
        await UniTask.Delay(100);
    }
}

public readonly struct AsyncOperationGroup
{
    public readonly List<AsyncOperation> Operations;
    public float Progress => Operations.Count == 0 ? 0 : Operations.Average(o
=> o.progress);
    public bool IsDone => Operations.All(o => o.isDone);

    public AsyncOperationGroup(int initialCapacity)
    {
        Operations = new List<AsyncOperation>(initialCapacity);
    }
}
}

```

SceneLoader.cs

```

using System.Collections.Generic;
using UnityEngine;
using Cysharp.Threading.Tasks;
using UnityEngine.SceneManagement;

namespace SceneManager
{
    public class SceneLoader : MonoBehaviour

```

```

{
    [SerializeField] private Canvas m_loadingScreen;
    [SerializeField] private Camera m_loadingCamera;
    [SerializeField] private SceneGroup[] m_sceneGroups;

    public readonly SceneGroupManager SceneGroupManager = new();

    public SceneGroup this[int index]
    {
        get { return m_sceneGroups[index]; }
    }

    public int SceneGroupSize => m_sceneGroups.Length;

    private void Awake()
    {
        Container.Register(this);
        SceneGroupManager.OnSceneLoaded += sceneName => Debug.Log($"Scene
{sceneName} is finished loading");
        SceneGroupManager.OnSceneUnloaded += sceneName => Debug.Log($"Scene
{sceneName} is finished unloading");
        SceneGroupManager.OnSceneGroupLoaded += () => Debug.Log("All scenes
are finished loading");
    }

    async void Start()
    {
        await LoadSceneGroup(0);
    }

    public async UniTask LoadSceneGroup(int index)
    {
        EnableLoadingScreen();
        await SceneGroupManager.LoadSceneGroup(m_sceneGroups[index]);
        EnableLoadingScreen(false);
    }

    private void EnableLoadingScreen(bool enable = true)
    {
        m_loadingCamera.gameObject.SetActive(enable);
        m_loadingScreen.gameObject.SetActive(enable);
    }
}

```

```

    }

    public SceneData GetActiveSceneFromGroup(int index)
    {
        return
m_sceneGroups[index].FindSceneDataByType(SceneType.ActiveScene);
    }
}
}

```

SceneManipulator.cs

```

using System;
using Cysharp.Threading.Tasks;
using SceneManagement;
using TMPPro;
using UnityEngine;

public class SceneManipulator : MonoBehaviour
{
    [SerializeField] private SceneLoader m_sceneLoader;
    private SceneData m_chosenScene;
    private int m_index = 1;

    public Action<SceneData> SceneChanged;

    private void Start()
    {
        m_sceneLoader = Container.GetInstance<SceneLoader>();
        ChangeChosenScene(m_index);
    }

    public void NextIndex()
    {
        m_index++;
        if (m_index > m_sceneLoader.SceneGroupSize - 1)
        {
            m_index = m_sceneLoader.SceneGroupSize - 1;
        }
    }
}

```

```

        ChangeChosenScene(m_index);
    }

    public void PreviousIndex()
    {
        m_index--;
        if (m_index < 1)
        {
            m_index = 1;
        }

        ChangeChosenScene(m_index);
    }

    public async void Select()
    {
        await m_sceneLoader.LoadSceneGroup(m_index);
    }

    private void ChangeChosenScene(int index)
    {
        if (!m_sceneLoader)
        {
            return;
        }

        m_chosenScene = m_sceneLoader.GetActiveSceneFromGroup(index);
        SceneChanged?.Invoke(m_chosenScene);
    }
}

```

ScenePresenter.cs

```

using SceneManagement;
using TMPPro;
using UnityEngine;
using UnityEngine.Video;

public class ScenePresenter : MonoBehaviour
{
    [SerializeField] private SceneManipulator m_sceneManipulator;
    [SerializeField] private TextMeshProUGUI m_locationName;

```

```

[SerializeField] private VideoPlayer m_videoPlayer;
private SceneData m_currentScene;
void Awake()
{
    m_sceneManipulator.SceneChanged += OnSceneChanged;
}
private void OnSceneChanged(SceneData data)
{
    m_currentScene = data;
    UpdateUi();
    UpdateAndPlayVideoClip();
}
private void UpdateUi()
{
    m_locationName.text = "Location :" + m_currentScene.LocationName;
}
private void UpdateAndPlayVideoClip()
{
    if (!m_currentScene.PreviewVideoClip)
    {
        return;
    }
    m_videoPlayer.clip = m_currentScene.PreviewVideoClip;
    PlayVideoClip();
}
private void PlayVideoClip()
{
    m_videoPlayer.Play();
}
}

```

ДОДАТОК В
Слайди презентації

Національний університет “Запорізька Політехніка”
Кафедра програмних засобів

**ІНТЕРАКТИВНА ПЛАТФОРМА ВІРТУАЛЬНОЇ
РЕАЛЬНОСТІ ДЛЯ ВІЗУАЛІЗАЦІЇ,
НАВЧАННЯ ТА ВЗАЄМОДІЇ З
АВТОМОБІЛЬНИМИ СИСТЕМАМИ**

Студент: гр. КНТ-221, Богдан ЄНАЛЬЄВ

Керівник: д.т.н, професор, Андрій ОЛІЙНИК

Рисунок В.1 – Слайд 1

Об'єкт, предмет та мета роботи

Об'єкт дослідження – процес розробки програмних засобів віртуальної реальності.

Предмет дослідження – програмні засоби реалізації VR-середовища для навчання, тренування та візуалізації процесу розбору та збирання автомобіля.

Мета роботи – розробка інтерактивного VR-середовища, яке забезпечує реалістичний і зручний процес дослідження будь яких інженерних проєктів. Ефективний і дешевий спосіб ознайомлення з інноваціями та складними приладами.

Рисунок В.2 – Слайд 2

Завдання роботи

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для підтримки процесу управління проєктами;
- здійснити проєктування інтерактивної платформи віртуальної реальності для візуалізації, навчання та взаємодії з автомобільними системами;
- створити програмне забезпечення для підтримки процесу управління проєктами;
- виконати тестування розробленого програмного забезпечення для підтримки процесу управління проєктами.

3

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів

Критерій порівняння	Ford FiVE	Siemens NX VR	Car Mechanic Simulator VR
Візуалізація складних моделей	+	+	+
Інтерактивність із компонентами	–	–	+
Реалістична фізика	–	–	±
Можливість тренування механіків	–	–	+
Коментування, редагування моделей	–	+	–

4

Рисунок В.4 – Слайд 4

Вибір рушія

Критерій	Unity	Unreal Engine	Godot
Складність входу	Середня	Висока	Низька
Простота розробки	Висока	Низька	Висока
Графіка	Хороша(Але поступається Unreal)	Найкраща	Хороша для 2d погана для 3d
Підтримка VR	Чудова(OpenXR, UnityXR, SteamVR)	Хороша, але складніша в налаштуванні	Мінімальна (слабка підтримка OpenXR)
Підходить для VR-симулятора механіків?	Так	Можна, але складніше та вимагає більше ресурсів	Ні (недостатньо підтримки)

5

Рисунок В.5 – Слайд 5

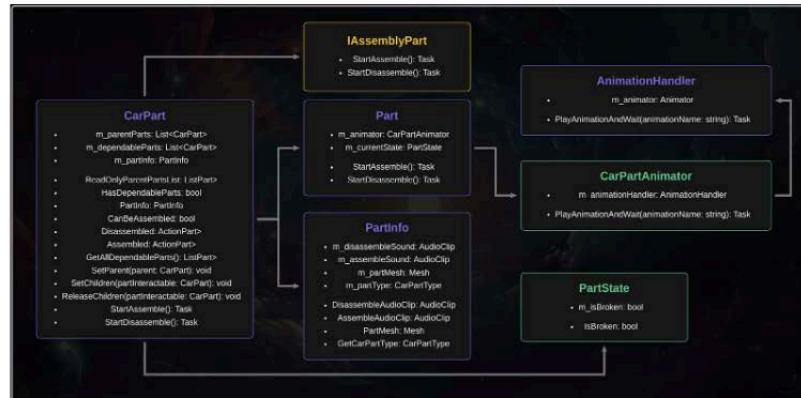
Вибір мови програмування

Критерій	C#	Rust	Python
Складність мови	Помірна, легка у вивченні	Висока, особливо через систему запозичень	Низька, дуже проста для швидкого старту
Контроль над пам'яттю	Автоматична (Garbage collector)	Безпечне ручне керування, без збирача сміття	Автоматичне керування
Продуктивність	Висока, але нижча за C++	Дуже висока, близька до C++	Низька
Розробка UI / логіки	Зручна, багаті можливості	Можлива, але потребує інтеграції через C API або плагіни	Зручна для логіки, слабка підтримка UI
Паралелізм / потоки	Простий у використанні async/await	Гнучкий, але складний	Обмежений через GIL

6

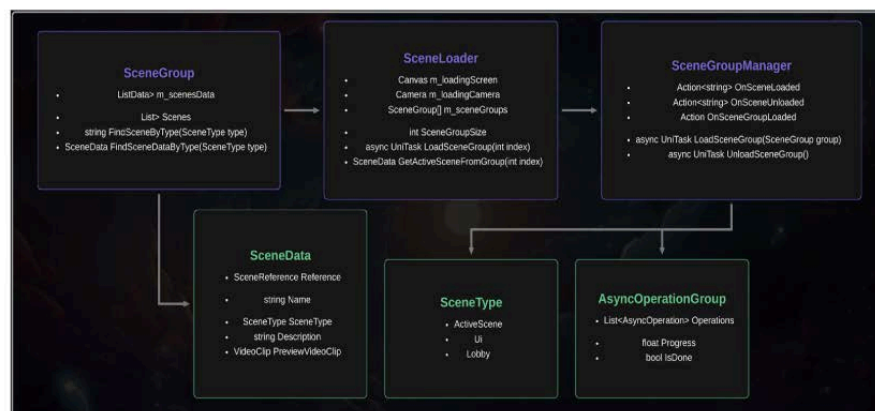
Рисунок В.6 – Слайд 6

Діаграми класів програми



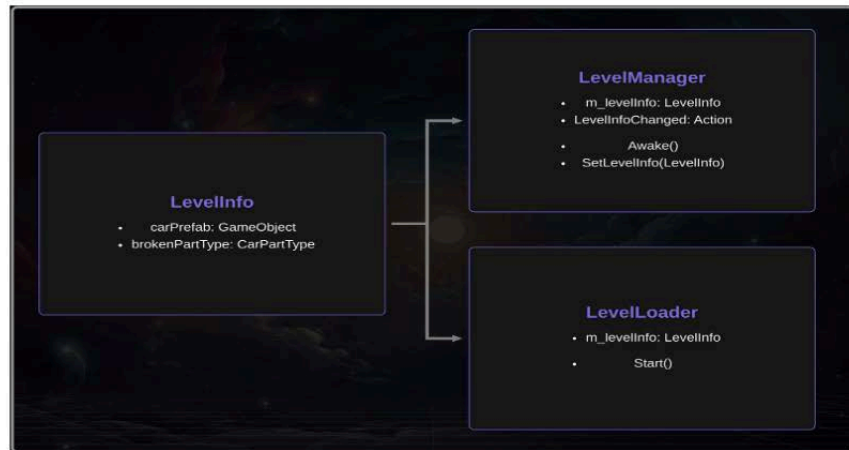
7

Рисунок В.7 – Слайд 7



8

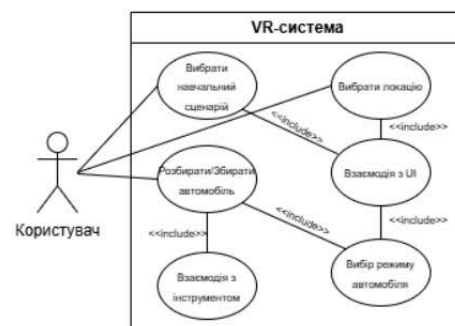
Рисунок В.8 – Слайд 8



9

Рисунок В.9 – Слайд 9

Діаграма прецедентів програмного забезпечення



10

Рисунок В.10 – Слайд 10

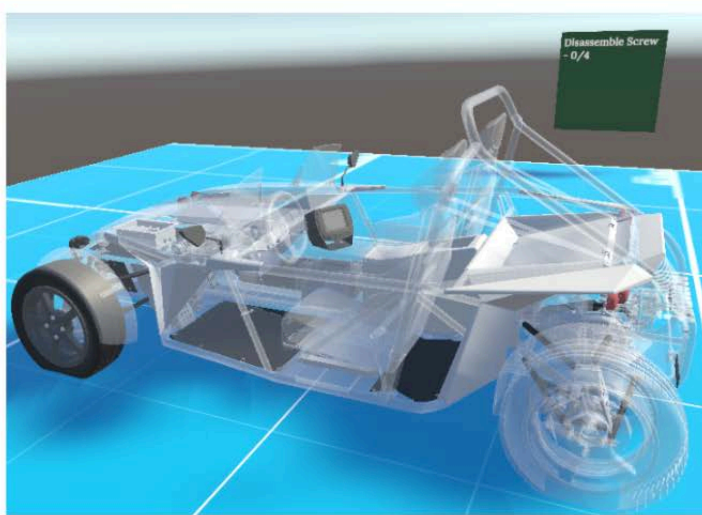
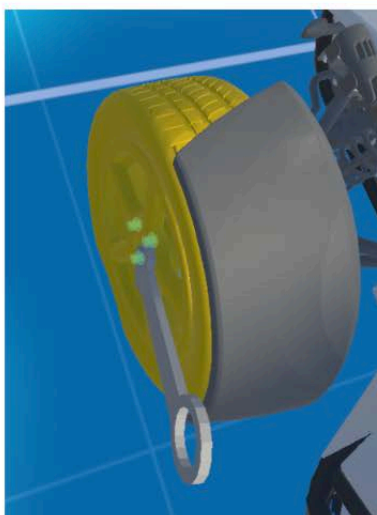
Робота з програмою. Головне лоббі



11

Рисунок В.11 – Слайд 11

Робота з програмою. Взаємодія з деталями



12

Рисунок В.12 – Слайд 12

Висновок

У результаті виконання роботи було створено інтерактивну VR-платформу для навчання та тестування автомобільних систем, яка дозволяє здійснювати віртуальну взаємодію з різними компонентами автомобіля. Метою роботи було створення прототипу, який би спрощував процес навчання та обслуговування автомобільних систем, використовуючи технології віртуальної реальності.

Розроблено VR-прототип застосунку, в якому реалізовані базові сценарії для розбору автомобіля, що дозволяють користувачам взаємодіяти з окремими частинами автомобіля, вивчаючи їх будову та принципи роботи.

Перспективи подальшого розвитку системи включають можливість розширення платформи для підтримки нових автомобільних моделей та сценаріїв взаємодії.

13

Рисунок В.13 – Слайд 13