

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 проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА VR ГРИ

В СЕРЕДОВИЩІ UNITY ENGINE

DEVELOPMENT OF A VR GAME

ON THE UNITY ENGINE PLATFORM

Виконав: студент 4 курсу, групи КНТ-120сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПАХОМОВ Р.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ТЕРЕЩЕНКО Е.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

"ПАХОМОВА Романа Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка VR гри в середовищі Unity Engine.
Development of a VR Game on the Unity Engine Platform.

керівник проєкту (роботи) к.т.н ШИТІКОВА Олена Вікторівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від "04" квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 01 червня 2023 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали 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	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ШИТІКОВА О.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 04 » квітня 2023 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)

_____ Роман ПАХОМОВ
(підпис) (Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Олена ШИТІКОВА
(підпис) (Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
121с., 44 рис., 6 табл., 3 дод., 36 джерел.

ВІРТУАЛЬНА РЕАЛЬНІСТЬ, КОМП'ЮТЕРНІ ІГРИ, ІГРОВИЙ
ДВИГУН, РОЗРОБКА, МОВА ПРОГРАМУВАННЯ, ПРОГРАМНЕ
ЗАБЕЗПЕЧЕННЯ.

Об'єкт дослідження – процес програмної реалізації VR середовища.

Предмет дослідження – програмна реалізація VR гри у середовищі
Unity Engine.

Метою роботи є розробка програмної реалізації VR гри у середовищі
UnityEngine.

Матеріали, методи та технічні засоби: ігровий двигун UnityEngine, мова програмування C#, ассет VRIF, рендер пайплайн URP, плагін MMD4Mecanim, ассет EzySlice, набір шейдерів RealToon, плагін DOTween, ассет DynamicBone, плагін TextMeshPro, програма для 3D моделювання Blender, графічний редактор Figma, засоби проектування UML.

Результати. Розроблено VR гру у середовищі Unity Engine.

Практична цінність роботи полягає у програмній реалізації VR гри у середовищі Unity Engine.

Висновки. Досліджено ринок VR застосунків та розроблено VR гру.

Галузь використання – Розважальна.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 121 pages, 44 figures, 6 tables, 3 appendixes, 36 sources.

VIRTUAL REALITY, COMPUTER GAMES, GAME ENGINE, DEVELOPMENT, PROGRAMMING LANGUAGE, SOFTWARE.

The object of research is the process of software implementation of a VR environment.

The subject of research is - process of software implementing a VR game in the Unity Engine environment.

The aim of the work is to develop a VR game with Unity Engine.

Materials, methods, and technical tools: Unity Engine, C# programming language, VRIF asset, URP render pipeline, MMD4Mecanim plugin, EzySlice asset, RealToon shader package, DOTween plugin, DynamicBone asset, TextMeshPro plugin, Blender 3D modeling software, Figma graphic editor, UML modeling tool.

Results. A VR game has been developed in the Unity Engine environment.

The practical value of the work lies in the software implementation of a VR game with Unity Engine.

Conclusions. The VR application market has been researched, and a VR game has been developed.

Field of use- Entertainment.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	8
Вступ.....	9
1 Аналіз предметної області.....	12
1.1 Становлення VR-індустрії.....	12
1.2 Unity як інструмент розробки ігор.....	14
1.3 Порівняння існуючих аналогів.....	16
1.3.1 Cooking Simulator VR.....	17
1.3.2 The Cooking Game VR.....	20
1.4 Постановка завдання.....	22
1.4.1 Функціональність системи.....	23
1.4.2 Вимоги до інтерфейсу.....	23
1.4.3 Вимоги до продуктивності.....	23
1.4.4 Експлуатаційні вимоги.....	24
1.5 Висновки за розділом 1.....	24
2 Інструменти, матеріали і методи.....	25
2.1 Ігрові двигуни.....	25
2.2 Вибір мови програмування.....	31
2.3 Вибір та обґрунтування VR SDK.....	35
2.4 Вибір та обґрунтування Render Pipeline.....	38
2.5 Інші інструменти, які були застосовані при розробці додатку.....	40
2.6 Структурна схема розробки.....	41
2.7 Висновки за розділом 2.....	42
3 Основні рішення щодо реалізації компонентів системи.....	43
3.1 Проектування моделей системи.....	43
3.2 Проектування ігрового рівня.....	45
3.3 Висновки за розділом 3.....	49
4 Експлуатація, тестування та експериментальне дослідження програми.....	50

4.1 Призначення та механіка додатку.....	50
4.2 Умови виконання додатку.....	50
4.3 Запуск додатку.....	51
4.4 Методика та результати тестування.....	52
Висновки.....	59
Перелік джерел посилання.....	60
Додаток А Технічне завдання.....	64
Додаток Б Текст програми.....	68
Додаток В Слайди презентації.....	113

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface (прикладний програмний інтерфейс);
HBAO	– Horizon Based Ambient Occlusion;
HDR	– High-Definition Range;
HDRP	– High-Definition Render Pipeline;
SDK	– Software Development Kit (набір інструментів для розробки);
SSAO	– Screen Space Ambient Occlusion;
UE	– Unreal Engine;
UML	– Unified Modeling Language (уніфікована мова моделювання);
URP	– Universal Render Pipeline;
VR	– Virtual Reality (віртуальна реальність);
VRIF	– VR Interaction Framework;
OC	– операційна система;
ПЗ	– програмне забезпечення;
ПК	– персональний комп'ютер.

ВСТУП

Ідея віртуальної реальності виникла вже у 1960-х роках, коли Іван Сазерленд і Боб Сполл провели перші експерименти зі створенням комп'ютерної графіки, яка б дозволяла користувачам взаємодіяти з віртуальним оточенням. Протягом наступних десятиліть VR технології поступово розвивалися та вдосконалювалися, але багато хто вважав цю концепцію просто науково-фантастичною [1]-[8].

Однак з появою персональних комп'ютерів та значного зростання обчислювальної потужності в 1990-х роках, ідея VR стала реалізовуватися в практичність. У 1991 році компанія SEGA випустила першу комерційну гарнітуру віртуальної реальності, відому як SEGA VR, яка залучила увагу громадськості та дослідників. Проте через обмежену функціональність та високу вартість, SEGA VR не змогла стати широко розповсюдженою.

З часом технології VR почали активно розвиватися, з'явилися нові компанії та продукти. У 2010 році компанія Oculus VR була заснована Палмером Лакі та його командою з метою створення доступних та ефективних VR-гарнітур. У 2012 році вони анонсували першу версію Oculus Rift, яка зібрала значну кількість позитивних відгуків та зацікавила розробників контенту [1]-[8].

У 2014 році Facebook оголосив про придбання компанії Oculus VR за 2 мільярди доларів, що засвідчило про зростаючий інтерес до VR. Інші великі технологічні компанії, такі як HTC, Sony, Samsung та Valve, також почали розробляти свої VR-гарнітури та платформи [1]-[8].

Актуальність та рост VR пояснюються кількома факторами. По-перше, технологічний прогрес у сфері обчислювальної потужності та графіки дозволяє створювати більш реалістичні та імерсивні віртуальні середовища. Віртуальна реальність вже не є просто звичайними комп'ютерними іграми, але й дозволяє користувачам взаємодіяти з віртуальним світом на більш глибокому рівні [1]-[8].

По-друге, VR знайшла застосування в багатьох галузях, включаючи розваги, освіту, медицину, архітектуру, авіацію та інші. Наприклад, віртуальна реальність використовується в медицині для симуляції хірургічних процедур та навчання медичних студентів. Вона також застосовується у військовій сфері для тренування військовослужбовців, та у архітектурі для візуалізації будівельних проєктів перед будівництвом [1]-[8].

Крім того, VR стала і продовжує ставати більш доступною та дешевою, завдяки чому вона стала привабливою для більшої кількості споживачів. З'явилися багато різних VR-гарнітур на ринку з різними цінами та можливостями, що надає змогу користувачам вибирати той варіант, який найкраще відповідає їх потребам, в тому числі і геймінг. Популярні ігрові двигуни Unity Engine та Unreal Engine також стали додавати та підтримувати інструменти для розробки VR додатків [1]-[8].

Відтак, актуальність та рост індустрії віртуальної реальності зумовлені комбінацією технологічних здобутків, широким спектром застосувань та зростаючою доступністю продуктів для споживачів. Ця технологія продовжує розвиватися та інтегруватися в наше повсякденне життя, надаючи нові можливості для інтерактивності та імерсії [1]-[8].

Хоча, VR-індустрія ще не знайшла тої масовості як інші галузі, пов'язані з комп'ютерною розробкою, все ж вона є перспективною, та вже застосовується для реалізації складних та багатосторонніх рішень з різних предметних областей.

Отже, актуальною задачею залишається програмна реалізація VR гри на Unity Engine.

Для виконання цієї задачі необхідно виконати наступні завдання:

- проаналізувати проблемну область VR застосунків;
- дослідити існуючі аналоги ігор зі схожою тематикою;
- з урахуванням аналогів сформулювати концепцію нового додатку;
- визначити технічні засоби, які дозволять оптимізувати процес розробки;

- спроектувати процес розробки, написання коду та тестування;
- з урахуванням попередніх пунктів розробити програмну реалізацію нового додатку;
- дослідити та протестувати нове рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Перший розділ буде присвячено дослідженню проблемної області VR ігор, в тому числі на Unity Engine, їх історії, застосунку, розробці. Також буде проведено порівняння існуючих аналогічних ігор, пов'язаних з приготуванням їжі. Дослідження ключових рис та особливостей аналогів дозволить сформулювати функціональні та візуальні вимоги до нової гри, що буде якісно відрізнятися від існуючих аналогів.

1.1 Становлення VR-індустрії

У минулому десятилітті спостерігалось значне зростання інтересу до віртуальної реальності у галузі комп'ютерних ігор. Цей технологічний прорив змінив уявлення про те, як гравці взаємодіють з віртуальними світами, пропонуючи їм неймовірно реалістичні та захоплюючі враження [3]-[8].

На початку 2023 року індустрія віртуальної реальності продовжує демонструвати стійкий ріст та інновації. Виробники VR-обладнання постійно вдосконалюють свої продукти, надаючи більш реалістичний імерсивний досвід для користувачів. Технологічні досягнення сприяють зниженню вартості VR-обладнання, що робить його більш доступним для широкої аудиторії [3]-[8].

На ринку відбувається значний попит на VR-продукти. Багато людей зацікавлені в імерсійному геймплеї, яким може похвалитись VR. Завдяки високій якості графіки та реалістичним відчуттям присутності, VR-ігри стають все більш популярними серед геймерів. Також зростає інтерес до використання VR для тренування та симуляцій у різних індустріях, оскільки вона надає безпрецедентні можливості в умовах, відносно близьких до реальних. Такими проектами можуть бути інтерактивні VR-тренажери для медичних студентів, або студентів технічних спеціальностей, за допомогою

яких можна симулювати роботу складних та дорогих пристроїв, тим самим навчаючи студентів їх експлуатувати, віртуальні екскурсії по історичних місцях, VR-симулятори для пілотів, водіїв або архітектурні візуалізації та багато всього іншого [3]-[8]. Наприклад мій друг працює в компанії, яка займається страховкою, та робить фотоскани з місць пригод, та нещодавно інтересувався, як би можна було експортувати відскановані 3D-моделі до якого-небудь движка, що би проводити більш детальні аналізи того, що відбулося у клієнта, тобто сфера застосунку обмежена майже лише фантазією.

Багато компаній і стартапів активно розробляють VR-продукти та рішення, що розширюють межі цієї технології. Зацікавленість інвесторів у VR-індустрію продовжує зростати, що стимулює подальші дослідження та розробки [1]-[8].

Віртуальна реальність в ігровій індустрії була насамперед зустрінена з захопленням інноваторами та ентузіастами, які бачили великий потенціал цієї технології для створення нових імерсивних геймплей-досвідів. Розвиток VR в ігровій індустрії можна розглядати у кілька етапів [3]-[8].

Вже в ранніх 90-х роках з'явилися спроби створити VR-ігри, які використовували прості технології того часу. Однак, технічні обмеження, висока ціна та нестабільність технологій обмежували їх популярність та доступність для широкої аудиторії [3]-[8].

У 2010-х роках з'явилися нові технологічні розробки, такі як Oculus Rift та HTC Vive, PlayStation VR, які змінили сприйняття VR. Ці нові пристрої забезпечували високу якість зображення, швидкість відгуку та точну відстеження рухів, що дозволяло створювати більш реалістичні та імерсивні ігрові враження [3]-[8].

Великі гравці, такі як Oculus, Valve та Sony, активно вкладалися в розробку VR-продуктів та розвиток платформ. Це призвело до збільшення обсягів продажів VR-пристроїв та ігор, а також залучення більшої кількості розробників у цю галузь. Крім того інструмент для розробки конкретно ігор

SteamVR допоміг об'єднати різні девайси через один інструмент розробки, тим самим посунувши границю платформ, що дозволило широкому спектру володарів гарнітури грати в одні і ті ж продукти. Oculus Quest внес значний вклад у розвиток VR-індустрії та стимулював його зростання. Oculus Quest був першим комерційним VR-пристроєм, який пропонував бездротовий та самостійний досвід. Гравцям не потрібно було підключатися до комп'ютера або використовувати провідні з'єднання, що робило його більш доступним та зручним для широкої аудиторії, і при цьому, відносно доступний у плані ціни. [3]-[8].

Усе це свідчить про перспективи та рост VR-індустрії у 2023 році та наступних роках.

1.2 Unity як інструмент розробки ігор

Unity - це крос-платформний ігровий двигун, розроблений компанією Unity Technologies, який був анонсований та випущений у червні 2005 року на конференції Apple Worldwide Developers Conference як двигун для Mac OS X. Згодом двигун поступово розширився і підтримує різноманітні платформи: PC, мобільні пристрої, консолі та VR. Він особливо популярний для розробки мобільних ігор на iOS та Android, вважається простим у використанні для розробників-початківців [1]-[3].

Цей двигун може використовуватися для створення тривимірних (3D) та двовимірних (2D) ігор, а також інтерактивних симуляцій та інших досвідів. Він був застосований в індустріях поза відеогеймінгом, таких як кіно, автомобілебудування, архітектура, інженерія, будівництво та навіть Збройними Силами США [1]-[2].

Unity Engine є одним з найпопулярніших і поширених ігрових двигунів у світі. Створення Unity бере свій початок у 2002 році, коли Девід Хелгасон (David Helgason), Ніклас Хедберг (Nicholas Francis) і Юджіро Кодзіма (Joachim Ante) заснували компанію з розробки програмного

забезпечення для мобільних пристроїв. У початковій стадії їхня робота зосереджувалась на створенні ігор для платформ J2ME та BREW [1]-[2].

У 2005 році Unity Technologies була офіційно заснована і представила Unity 1.0 на конференції Apple Worldwide Developers Conference (WWDC). Це був проривний момент для компанії, оскільки Unity Engine став доступним для розробки ігор на різних платформах, включаючи iOS, Android, Windows та MacOS. З цього часу Unity набула великої популярності в індустрії геймдеву і швидко розвивалась [1]-[2].

У наступні роки Unity Engine постійно вдосконалювався і отримував нові можливості. У 2012 році була випущена Unity 4 з підтримкою багатьох нових функцій таких як покращений рендеринг, система анімації Mecanim та підтримка Flash, платформи Linux, DirectX 11. Це значно розширило можливості розробників ігор і привело до ще більшого поширення Unity. У 2013 році Unity Technologies оголосила про початок роботи над Unity 5, яка була випущена в 2015 році. Unity 5 включала новий фізичний двигун, покращений рендеринг, підтримку розробки віртуальної реальності (VR) і багато інших поліпшень. Це дозволило розробникам створювати ще більш реалістичні ігри і досвіди [1]-[2].

У наступні роки Unity продовжувала розвиватися і адаптуватися до нових технологій. Компанія випустила ряд інструментів і послуг, таких як Unity Ads (рекламна платформа), Unity Analytics (аналітичний інструмент) і Unity Asset Store (магазин активів), що дозволило розробникам ефективніше створювати, монетизувати та оптимізувати свої ігри та багато чого іншого [1]-[2].

У 2020 році Unity Technologies випустила Unity 2020 з новими можливостями для розробки ігор віртуальної та доповненої реальності, а також з покращеною системою світла та рендерингу. Одним із важливих досягнень Unity була підтримка майже всіх головних платформ, включаючи мобільні пристрої, ПК, консолі, веб та навіть вбудовані системи [1]-[2].

Сьогодні Unity Engine залишається одним з провідних двигунів у галузі розробки ігор. Він використовується як великими студіями, такими як Ubisoft, Electronic Arts і Blizzard, так і невеликими командами та незалежними розробниками по всьому світу. Unity надає зручні інструменти, велику кількість ресурсів та активів, а також сильну спільноту, що допомагає розробникам створювати якісні ігри та досвіди для різних платформ [1]-[2].

1.3 Порівняння існуючих аналогів

Як у будь-якої технології, у VR ігор є як переваги, так і серйозні недоліки, і в цьому розділі ми розглянемо основні з них [3]-[8], [14]-[15].

Переваги VR ігор наступні:

- імерсивний досвід. VR дозволяє зануритись у віртуальний світ, що створює враження присутності та може бути дуже захоплюючим;
- реалістичність. Якби тривимірна та об'ємна картинка у VR-іграх та додатках сприяється зовсім не так, як просте зображення з монітору, що також додає досвіду іммерсивності;
- фізична активність. Деякі VR-додатки пропонують рухову активність, що дозволяє користувачеві більше рухатись та фізично взаємодіяти з оточуючим світом. Це може сприяти активному способу життя та фізичній формі;
- з точки зору розробки, нові ринки та можливості: Розробка VR може відкрити двері до нових ринків та сегментів, таких як ігрова індустрія, освіта, медицина, архітектура та інші галузі;

Недоліки VR ігор наступні.

- вартість. VR-ігри можуть бути дорогими. Потрібно мати спеціальні гарнітури та обладнання для використання VR, що може бути вартим;
- комфорт: Деякі люди можуть відчувати дискомфорт або мають схильність до захворювання від руху під час користування VR-гарнітурою;

- фізичні обмеження. VR-гарнітури можуть бути досить великими та незручними для тривалого використання. Також може виникнути проблема з взаємодією з навколишнім середовищем та обмеженням візуального полі поля зору;
- обмежена бібліотека ігор. Наразі бібліотека VR-ігор може бути менш розгалуженою порівняно з іграми на інших платформах, таких як консолі або ПК. До того ж, наразі VR індустрія більше сфокусована на різних тренажерах, та неігрових додатках, таких як E-Commerce;
- з точки зору розробки, високі вимоги до обладнання розробників: Розробка VR вимагає потужного обладнання, що може бути дорогим та обмежувати доступність для деяких розробників;
- складність розробки. Розробка VR вимагає спеціалізованих навичок та знань, а також додаткового часу для розробки та тестування;
- обмеження ринку. Наразі ринок VR-розваг та додатків ще не настільки розширений, як ринок ігор та додатків для інших платформ.

Потрібно також зазначити, що VR розробка також наслідують більшість переваг розробки цифрових ігор в цілому, і недоліки розробки будь-якого продукту, таких як початкові витрати, ризики прогоріти, і так далі [3]-[8], [14]-[15].

1.3.1 Cooking Simulator VR

Cooking Simulator - це симулятор, в якій гравець виступає у ролі кухаря. Гра була розроблена польською командою Big Cheese Studio і випущена PlayWay S.A. 6 червня 2019 року для Microsoft Windows. Версія для Nintendo Switch вийшла 14 травня 2020 року, а для Xbox One - 14 серпня 2020 року. Мобільна версія під назвою Cooking Simulator Mobile була випущена 20 жовтня 2020 року для пристроїв на базі Android. Вона також доступна на iOS в деяких країнах, а світовий реліз можна попередньо зареєструвати у PlayWay S.A. Версія для PlayStation 4 була випущена 20

травня 2021 року ексклюзивно в Європі та Австралії, а 8 червня 2021 року - у Північній Америці [11]-[12].

Cooking Simulator використовує інтегрований фізичний двигун та механіку приготування їжі для імітації реалістичного кулінарного досвіду. Гравець має можливість приготувати понад 80 рецептів, використовуючи понад 140 інгредієнтів. Гра має різноманітні режими, включаючи Кар'єрний режим, Режим пісочниці, Школу кулінарії, Піцу, Торти та Печиво, Укриття, рейтинговий, Зимові свята і SUPERHOT. У Кар'єрному режимі гравці беруть під контроль кухню і поступово заробляють славу та досвід, виконуючи замовлення та готуючи страви. Метою є досягнення п'ятої зірки слави. Режим пісочниці пропонує повністю обладнану кухню без обмежень часу. Школа кулінарії - це серія навчальних місій, які пояснюють усі механіки гри. У рейтинговому режимі гравці змагаються, намагаючись приготувати одну страву згідно рецепту на час, та правильність приготування. У режимі Зимові свята гравцям представлена нова кухня у якій вони можуть готувати вечерю для гостей, зі страв із заданими інгредієнтами [11]-[12].

У доповненні Pizza гравець працює в піцерії і має завдання готувати різні піци з різним тістаом та начинками. Доповнення Cakes and Cookies дозволяє гравцю готувати різноманітні торти, пироги, печиво та інші десерти в кондитерській [11]-[12].

Доповнення SUPERHOT Challenge було додано як безкоштовне оновлення 25 лютого 2020 року, натхненником якого була VR гра SUPERHOT, та щоб відзначити її четверту річницю в Steam. Воно змінює вигляд продуктів, посуду та кухні і вводить нову механіку, яка сповільнює час, коли гравець не рухається [11]-[12].

У доповненні Shelter, після ядерної катастрофи в 1980-х роках, гравець повинен нормувати та готувати їжу для свого бункера, шукаючи свою загублену родину. Головною особливістю режиму є можливість виходити в пустелю для обміну приготовлених страв на послуги з полювання

на мутантів задля м'яса та збирання радіоактивних рослин у якості інгредієнтів [11]-[12].

Через початкові недоліки, Cooking Simulator отримав змішані відгуки від гравців. Він отримав оцінку 64/100 на Metacritic на основі 4 оглядів. PC Games оцінила графіку і різноманітність рецептів, але вказала на складність керування, незахоплюючий кар'єрний режим та проблеми з продуктивністю. GameStar також відзначила цікаві рецепти, навчальний режим, систему огляду страв, але також невелику кількість інгредієнтів, рецептів та посуду як плюси гри. До недоліків віднесли складне керування, репетитивність, відсутність рівнів складності, проблеми з фізикою та відсутність рестарту в кар'єрному режимі. У своєму огляді Screen Rant зазначив, що гра складна через розрахунок часу та керування, але весела на короткий проміжок часу, але стає дратівливою через деякий час [11]-[12].

На рисунках 1.1 та 1.2 приведено скріншоти геймплею Cooking Simulator VR.



Рисунок 1.1 – Скріншот геймплею Cooking Simulator VR [12]



Рисунок 1.2 – Скріншот геймплею Cooking Simulator VR [12]

1.3.2 The Cooking Game VR

The Cooking Game VR це інді-гра меншого масштабу ніж попередня, але вона також дозволяє насолодитися веселим та захоплюючим процесом приготування страв швидкого харчування, змагатися з суперниками-кухарями та обслуговувати голодних клієнтів [13].

У цю гру можна грати як в режимі для одного гравця, так і в режимі PvP мультиплеєра! У режимі PvP мультиплеєра потрібне стабільне підключення до Інтернету [13].

Видання VrtheGamers відмічає, що саме мультиплеєр надає цій грі унікальності в порівнянні з аналогами, та помножує захоплюваність геймплею.

На рисунку 1.3 приведено скріншот геймплею The Cooking Game VR.



Рисунок 1.3 – Скріншот з гри The Cooking Game VR [13]

При пошуку аналогів, можна зрозуміти, що їх взагалі, особливо то і немає, що підтверджує слова з попереднього розділу про те, що VR індустрія ще досі є нішевою і специфічною. Також, судячи з кількості відгуків, можна зробити висновок, що ці ігри не отримали велике охоплення аудиторії в силу того, що більшість користувачів ПК не мають гарнітури, але все ж є великі VR ігри на ринку, наприклад Half-Life Alyx, Superhot, VRChat, та деякі інші.

У таблиці 1.1 були підсумовані основні риси, притаманні знайденим аналогам, та їх порівняння.

Таблиця 1.1 – Порівняння аналогів з по критеріям

Критерії порівняння	Cooking Simulator VR	The Cooking Game VR
Рівень графіки	Високий	Низький
Кількість рецептів	Висока	Низька
Сеттинг	Сучасна Європа/Америка Постапокаліпсис	Звичайна кухня
Кількість та різноманітність режимів	Висока	Низька
Мультиплеер	Ні	Так
Продуктивність	Середня	Висока
Кількість відгуків	1060 (85% позитивні)	40 (70% позитивні)

1.4 Постановка завдання

Мета: розробити продукт, що буде являти собою симулятор стріт фуду в Японії, в аніме стилістиці. Ціллю гри буде обслуговування клієнтів, приготування бургерів в першу чергу, та також інших замовлень на швидкість, за що гравець буде отримувати ігрові гроші, а ціллю, заробити якомога більше за одну 5и хвилинну сесію гри. Продукт буде розроблений в першу чергу для платформи Oculus Quest 2, але з можливістю розширення і на інші платформи за допомогою OpenXR.

1.4.1 Функціональність системи

Гра повинна надавати користувачу наступні можливості:

- повний VR досвід, передача зображення на шолом, функціонування та інтерактив за допомогою контролерів;
- інтерактивні об'єкти;
- приготування бургерів;
- спрощена плита для смаження котлет;
- автомат з лимонадом;
- штучний інтелект неігрових персонажів;
- анімовані неігрові персонажі в стилі аніме;
- VFX;
- SFX;
- оточення в Японському стилі;
- внесення передоплати за бронювання.

1.4.2 Вимоги до інтерфейсу

Англомовний інтерфейс, зрозумілий для користувачів з різних країн світу.

Система повинна мати інтуїтивний та зручний інтерфейс, інтуїтивні механіки та індикації, які надавали би можливість зручного і позитивного досвіду під використання додатку.

1.4.3 Вимоги до продуктивності

Гра повинна працювати без збоїв та помилок. У разі виникнення збою системи, повинна адекватно на них реагувати, виводити відповідні повідомлення користувачам та зберігати усю вже наявну інформацію про клієнтів та путівки і відновлюватися у найкоротші терміни.

1.4.4 Експлуатаційні вимоги

Гра повинна коректно запускатися та функціонувати на платформах Windows 32/64 bit, 7, 8.1, 10, 11.

Вимоги до апаратного забезпечення:

- Oculus Quest/Oculus Quest 2 наполегливо рекомендується, не дивлячись на наявність емулятора;
- відеокарта, Nvidia Geforce 1050TI / AMD Radeon RX470 або вище з підтримкою DirectX 11 або вище;
- процесор Intel Core i5-7600 / AMD Ryzen 3 2300X або вище;
- 8 GB оперативної пам'яті або більше;
- 2 GB пам'яті на жорсткому диску;
- 2 GB пам'яті на жорсткому диску;
- встановлений і заздалегідь запуснений Oculus Hub.

1.5 Висновки за розділом 1

В першому розділі було досліджено проблемну область VR ігор, в тому числі на Unity Engine, їх історії, застосування, розробці. Також було проведено порівняння існуючих аналогічних ігор, пов'язаних з приготуванням їжі. Дослідження ключових рис та особливостей аналогів дозволило сформулювати функціональні та візуальні вимоги до нової гри, що буде якісно відрізнятися від існуючих аналогів.

2 ІНСТРУМЕНТИ, МАТЕРІАЛИ І МЕТОДИ

Попередньо було досліджено проблемну область VR ігор, в тому числі на Unity Engine, їх історії, застосунку, розробці. Також було проведено порівняння існуючих аналогічних ігор, пов'язаних з приготуванням їжі. Дослідження ключових рис та особливостей аналогів дозволило сформулювати функціональні та візуальні вимоги до нової гри, що буде якісно відрізнятися від існуючих аналогів.

В другому розділі буде розв'язуватися задача вибору технології розробки. Серед завдань: вибір мови програмування, ігрового двигуна, вибір інструментів для спрощення процесу розробки. Вирішення цих завдань на основі аналізу та дослідження допоможе розв'язати поставлене завдання другого розділу.

2.1 Ігрові двигуни

Unity Engine є потужним ігровим двигуном для створення ігор і інтерактивних додатків. Він надає розробникам інструменти, ресурси та функціональність, щоб створювати графічно привабливі та захоплюючі відеоігри для різних платформ [1], [16]-[19].

Unity використовується для розробки ігор на різних платформах, таких як комп'ютери, консолі, мобільні пристрої, віртуальна реальність (VR) та доповнена реальність (AR). Його використовують як незалежні розробники, так і великі компанії в індустрії геймдеву. Unity має широкий спектр функціональності, яка допомагає розробникам створювати відкриті світи, 2D та 3D ігри, фізичну поведінку об'єктів, штучний інтелект, анімацію персонажів, управління звуком, розумний розподіл ресурсів та багато іншого [1] [16]-[19].

Основною мовою програмування в Unity є C#, але також підтримуються JavaScript та Boo. Unity надає зручний візуальний редактор,

де ви можете створювати та налаштовувати об'єкти, компоненти, шейдери, анімації та логіку гри [1], [16]-[19].

Unity також пропонує Asset Store, де розробники можуть придбати готові активи, такі як моделі персонажів, текстури, звукові ефекти та інші ресурси, що допомагають прискорити процес розробки [1], [16]-[19].

Основна перевага Unity полягає в його легкості в освоєнні, кросплатформеності, великій спільноті розробників, розширеній функціональності та зручному інтерфейсі. Він є популярним інструментом серед розробників ігор завдяки своїм можливостям і дружній навколишньому середовищу для розробки. Unity Engine надає розробникам доступ до різноманітних інструментів, які допомагають у кожному етапі розробки ігор. Від створення прототипів та дизайну до реалізації геймплею, анімації, звуків та ефектів - усе можливо в Unity [1], [16]-[19].

Завдяки вбудованому редактору, розробники можуть створювати та налаштовувати об'єкти, компоненти та логіку гри без необхідності писати весь код з нуля. Візуальні інструменти Unity дозволяють розміщувати об'єкти на сцені, редагувати їх властивості, налаштовувати фізику, створювати колізії та інтерактивність. Unity також підтримує різні формати файлів та імпортує активи зовнішніх програм. Це означає, що ви можете використовувати моделі, анімацію, текстури та звукові ефекти, створені у програмах для 3D-моделювання, таких як Blender чи Maya [1], [16]-[19].

Ще однією суттєвою перевагою Unity Engine є його здатність до кросплатформеної розробки. Ви можете створювати одну основну версію гри, а потім легко розгорнути її на різних платформах. Це дозволяє вам ефективно використовувати ресурси та зберігати єдність геймплею незалежно від пристрою, на якому гра запускається. Unity також підтримує розробку віртуальної та доповненої реальності. За допомогою додаткових плагінів та розширень, ви можете створювати ігри та додатки, які використовують VR- та AR-технології, що дозволяє зануритися у

віртуальний світ або додавати елементи реальності до реального середовища [1], [16]-[19].

Усі ці можливості роблять Unity Engine потужним інструментом для розробки ігор та інтерактивних додатків, незалежно від розмірів та обсягу проєкту. Він забезпечує розробників необхідними засобами та ресурсами для творчого втілення їх ідей та створення захоплюючих досвідів для гравців. Звісно, Unity не позбавлений недосконалостей [1], [16]-[19]:

- вимоги до ресурсів. Unity може вимагати значних обчислювальних ресурсів та пам'яті, особливо при роботі з великими та складними проєктами. Це може вплинути на продуктивність настільки, що розробка може бути повільною або вимагати потужніших комп'ютерів [1], [16]-[23];

- великий розмір файлів. Проєкти, розроблені в Unity, можуть мати великі розміри файлів, особливо якщо вони містять багато графічних, звукових та інших ресурсів. Це може призвести до складнощів з розповсюдженням ігрових файлів та збільшити час завантаження для користувачів [1], [16]-[23];

- висока конкуренція. Будучи одним з найпопулярніших двигунів розробки ігор, Unity також зазнає високої конкуренції в ігровій індустрії. Це означає, що вам доведеться боротися за увагу користувачів і виділятися серед безлічі інших ігор, що випускаються [1], [16]-[23];

- обмежена гнучкість. У порівнянні з іншими двигунами, Unity може бути менш гнучким у плані налаштування деяких аспектів гри. Деякі розробники можуть відчувати обмеження у створенні специфічних механік або функціональності [1], [16]-[23];

- залежність від Asset Store. Хоча Asset Store Unity надає великий вибір готових активів та ресурсів, деякі розробники можуть стикнутися з проблемою залежності від цього магазину. Це може призвести до витрат на покупку активів або втрати унікальності проєкту, якщо використовуються загальнодоступні активи [1], [16]-[23];

- інтерфейс користувача. Деяким розробникам може знадобитися час, щоб освоїти інтерфейс Unity, особливо для початківців. Іноді навіть досвідченим розробникам може знадобитися час для навчання та розуміння всіх можливостей та функцій, які надає Unity [1], [16]-[23];
- проблеми з оптимізацією. Оптимізація проєктів у Unity може бути складною задачею, особливо при розробці для мобільних пристроїв або старших комп'ютерів. Погана оптимізація може призвести до низької продуктивності, зависань та інших проблем, які впливають на користувацький досвід [1], [16]-[23];
- проблеми з командною розробкою. При роботі в команді над великим проєктом Unity може виникати складність з об'єднанням та керуванням версіями коду та ресурсів. Це може призвести до конфліктів інтеграції та складнощів з управлінням розробкою великих команд [1], [16]-[23];
- вартість. Хоча Unity надає безкоштовну версію для початківців та невеликих проєктів, розширені можливості та додаткові сервіси, такі як підтримка платформи та пакети розширень, можуть бути платними. Це може призвести до додаткових витрат для розробників залежно від їх потреб і масштабу проєкту [1], [16]-[23].

Варто зауважити, що багато з цих недоліків можуть бути переборені досвідченими розробниками, і Unity все ще залишається одним з найпопулярніших та потужних двигунів розробки ігор на ринку [21].

Враховуючи величезний інтерес студій і бізнесів до розробки додатків з використанням Unity або Unreal Engine, ми вирішили переглянути факти, щоб допомогти вам зробити обґрунтований вибір у розробці вашого наступного проєкту [21].

У 2015 році Unity та Unreal Engine були гарячими темами в галузі розробки ігор, викликаючи обговорення та порівняння скрізь, де вони згадувалися [21].

В останній час обидва двигуни зазнали значних змін, і враховуючи величезний інтерес студій і бізнесів до розробки додатків з використанням Unity або Unreal Engine, ми вирішили переглянути цю тему. Було проаналізовано факти, щоб прийняти обґрунтоване рішення щодо розробки проєкту, загальні риси яких наведено у таблицях 2.1 та 2.2.

Таблиця 2.1 – Порівняння розповсюдженості двигунів

Unity	Unreal Engine
Мобільні ігри, розроблені з використанням Unity, становлять 71% з 1000 найпопулярніших ігор на ринку.	Понад 2 мільйони ігор працюють на цьому двигуні.
Кількість користувачів (розробники + користувачі) перевищує 2,7 мільярда людей.	Спільнота розробників налічує понад 7 мільйонів людей.
У Asset Store представлено понад 65 000 активів.	Маркетплейс містить понад 16 000 активів.
Дохід компанії збільшився на 43% в 2020 році.	Ринкова капіталізація становить приблизно 27,8 мільярда доларів.
Ринкова капіталізація становить приблизно 27,8 мільярда доларів.	Ринкова капіталізація Epic Games (материнської компанії) становить приблизно 17 мільярдів доларів.

Таблиця 2.2 – Порівняння двигунів за використанням та деякими іншими критеріями

Критерій	Unity	Unreal Engine
Мобільні додатки	Unity був розроблений з урахуванням мобільних додатків, тому розробка для цих пристроїв є дуже легкою і зручною, як для 2D, так і для 3D проєктів. Більшість оптимізацій призначені для невеликих та інді-ігор.	Використання потужних графічних можливостей UE є надмірним для більшості мобільних додатків. А також є більш придатним для AAA-ігор і тих, що орієнтовані на високопотужні пристрої.
Імерсивні додатки	Обидва двигуни підтримують засновані на технологіях віртуальної реальності, змішаної реальності та доповненої реальності додатки. Проте вибір Unity є ідеальним для прототипування та експериментів.	Це відмінний вибір для захоплюючих додатків, розроблених з великим рівнем деталей. Крім того, завдяки системі Blueprints, він надає нижший поріг для розробки захоплюючих додатків.
Графіка	Unity надає кілька вражаючих стокових графічних можливостей проте досягнення кращого візуалу часто вимагає багато налаштувань та редагування.	У додаток до вражаючих вбудованих графічних можливостей, додатки UE часто виглядають досконало без додаткових змін завдяки широкому спектру передвстановлених налаштувань.

Продовження таблиці 2.2.

Критерій	Unity	Unreal Engine
Спільнота	Asset-Store є величезним і наповненим різноманітними 3D-асетами, а також 2D-асетами, інструментами та візуальними ефектами. На веб-сайті платформи також є велика кількість навчальних матеріалів і активний форум.	Хоча магазин асетів UE є меншим, він також пропонує достатню різноманітність асетів, в основному спрямованих на ігрові елементи. Розробники також мають безліч посібників та форумів для підтримки.
Ціна	Найпростіший план (Personal) є безкоштовним для використання, але більш розширені та бізнес-орієнтовані плани коштують \$399 на рік за кожний обліковий запис або більше.	UE є безкоштовним для використання, але включає систему роялті, яка починає діяти з моменту монетизації додатка і приносить компанії (Epic Games) 5% від доходу.

2.2 Вибір мови програмування

Не дивлячись на те, що раніше Unity підтримував UnityScript (трохи перероблений JavaScript) та Boo, починаючи з версії 2018.x в Unity залишилась тільки одна мова програмування — C# [24]-[26].

Unity припинив розвиток мови програмування Boo та UnityScript, які були альтернативними мовами для написання скриптів в Unity, з кількох причин:

Об'єднання мов: Unity прагнув спростити та зосередитись на розвитку, скорочуючи кількість мов програмування, які вони підтримували.

Припинивши розвиток Boo та UnityScript, вони могли зосередитись на поліпшенні підтримки основної мови — C# [24]-[26].

Стандартизація у галузі: C# став стандартною мовою програмування для розробки ігор, широко використовуваною в інших ігрових двигунах та програмному забезпеченні. Пристосувавшись до цього стандарту, Unity прагнув забезпечити розробникам більш знайому та потужну середу програмування [24]-[26].

Спільнота та екосистема: У C# є велика спільнота розробників та розгалужена екосистема бібліотек, фреймворків та ресурсів. Спонукаючи розробників використовувати C#, Unity прагнув скористатись цією багатою екосистемою та створити жваву та підтримувану спільноту для розробників [24]-[26].

Продуктивність та оптимізація: UnityScript та Boo були динамічно типізованими мовами, що може призводити до проблем з продуктивністю та оптимізацією. Спрямовуючись на статично типізовану мову, таку як C#, Unity міг забезпечити кращі можливості щодо продуктивності та оптимізації [24]-[26].

Припинивши розвиток Boo та UnityScript, Unity мав мету надати єдине та ефективне середовище розробки для своїх користувачів, використовуючи переваги C# як широко використовуваної та потужної мови програмування [24]-[26].

У таблиці 2.3 наведено основні критерії порівняння мов програмування, які Unity підтримує в тій чи іншій формі.

Таблиця 2.3 – Порівняння мов програмування у Unity

Критерій	C#	UnityScript	Boo
Простота вивчення та поріг входження	Середня	Низька	Низька
Типізація	Статична	Динамічна	Статична
Продуктивність	Висока	Низька	Низька
Синтаксис	Схожий на C++, Java	Схожий на JavaScript	Схожий на Python
Стан	Підтримується	Не підтримується починаючи з 2018.x	Не підтримується з виходу Unity 4

У зв'язку з тим, що UnityScript та Boo не підтримується в нових версіях двигуна, і потрібен доступ до сучасного функціонала двигуна, єдиною опцією залишається C#.

C# - універсальна і широко використовувана мова програмування, розроблена компанією Microsoft. Вона була спеціально створена для розробки різноманітних додатків, включаючи ігри, веб-додатки, програми для робочого столу та багато іншого. Ось кілька основних особливостей та характеристик мови C# [27]-[29]:

Об'єктно-орієнтоване програмування: C# ґрунтується на принципах об'єктно-орієнтованого програмування (ООП), що дозволяє розподіляти код на повторно використовувані об'єкти з властивостями, методами та подіями [27]-[29].

Безпека типів: C# - статично типізована мова, що означає, що змінні повинні мати визначений тип під час компіляції. Це допомагає виявляти помилки на ранніх етапах та забезпечує більшу надійність коду [27]-[29].

Розробка для кросплатформеного середовища: За допомогою .NET Core, C# можна використовувати для розробки додатків, що працюють на різних платформах, включаючи Windows, macOS, Linux, iOS та Android [27]-[29].

Обширний фреймворк: C# має велику екосистему, включаючи .NET фреймворк та різноманітні бібліотеки, які надають розробникам багато готових функцій та інструментів для прискорення розробки [27]-[29].

Інтеграція мов: C# може легко інтегруватися з іншими мовами в межах екосистеми .NET, що дозволяє розробникам використовувати існуючий код, написаний на різних мовах [27]-[29].

Асинхронне програмування: C# надає потужну підтримку для асинхронного програмування за допомогою функцій, таких як `async/await`, що дозволяють розробникам писати відзивчиві та масштабовані додатки [27]-[29].

Збірка сміття: C# включає автоматичне управління пам'яттю за допомогою зборки сміття, що звільняє розробників від ручного виділення та звільнення пам'яті [27]-[29].

Спільнота та ресурси: C# має велику та активну спільноту розробників, яка пропонує обширну документацію, підручники, форуми та бібліотеки третьої сторони для підтримки навчання та розробки [27]-[29].

Інтеграція з Unity: C# є рекомендованою мовою скриптування для розробки ігор на платформі Unity. Вона має відмінну інтеграцію з двигуном Unity, надаючи потужні можливості для створення інтерактивних та динамічних ігор.

Універсальність: C# може використовуватися для різноманітних завдань розробки поза галуззю ігрової розробки, включаючи веб-розробку, програми для робочого столу, мобільні додатки, хмарні сервіси та інше.

C# поєднує в собі силу сучасної мови програмування з великою підтримкою .NET фреймворку, що робить його популярним вибором для розробників у різних галузях та областях.

2.3 Вибір та обґрунтування VR SDK

VRIF (VR Interaction Framework) дозволяє легко створювати власні інтерактивні об'єкти і швидко досягати продуктивності. Цей фреймворк надає основу, необхідну для створення проєктів у віртуальній реальності [10]:

- фізичне тримання та кидання об'єктів;
- кнопки, ручки, ручки, слайдери, двері, джойстики та інші фізичні об'єкти;
- емулятор VR, щоб ви могли тестувати свій проєкт без використання гарнітури;
- плавне переміщення та телепортація з можливістю швидкої та плавної зміни напрямку;
- власні моделі та текстури рук гравця;
- повнофункціональний плагін для пози рук з автоматичним функціоналом пози і понад 40 поз рук;
- готовий контролер гравця, який використовує CharacterController Unity;
- скелелазіння;
- зони прив'язки;
- дуже модульні: можливість створювати слоти для обладнання, системи інвентаризації та інше;
- фізика лука та стріл (стріли можна забрати та знову вистрілити);
- обробка / фізика зброї. Можна вставляти та витягувати магазин з пістолета, витягати затвор і т.д.;
- дворучна зброя, дробовик, гвинтівка та бойова зброя ближнього бою;
- гарпун;
- проста система пошкоджень з руйнуванням об'єктів;
- повна підтримка FinalIK / VRIK з детальними прикладами [10].

Розглянемо деякі альтернативи VRIF.

Auto Hand - це система взаємодії з фізикою високої якості, яка розроблена з урахуванням простоти використання та високої налаштованості. Auto Hand має онлайн-документацію та майстер налаштування з параметрами якості, що дозволяють досягти найкращої якості в залежності від обмежень вашого проєкту [30].

Auto Hand включає систему автоматичної генерації пози, яка налаштовує форму руки під час захоплення. Також в комплекті йде повністю функціональний контролер руху, який включає плавний рух, телепортацію, скелелазіння, систему захисту від зіткнень голови [30].

Взаємодія з фізикою в VR включає такі ефекти, як вага, колізія, захоплення двома руками, розрив захопленого об'єкта, кидання високої якості, захоплення на відстані. У прикладах також наведені фізичні пристрої, ручки, повзуни, двері, колеса, ручки і кнопки. Auto Hand широко використовує Unity Events та документацію зі вказівками, що робить його потужним та простим у використанні інструментом [30].

Oculus Integration надає підтримку розширених можливостей рендерингу, соціальних функцій, аудіо та розробки аватарів для пристроїв віртуальної реальності Oculus та деяких пристроїв, що підтримують OpenXR.

Oculus Integration містить наступне:

- менеджер аудіо. містить скрипти для керування всіма аудіо- та звуковими ефектами у вашому додатку;
- аватар. містить скрипти та префаби для додавання аватарів Oculus у ваш додаток;
- Interaction SDK. набір модульних компонентів, що дозволяють розробникам реалізовувати широкий спектр надійних та стандартизованих взаємодій;
- LipSync. містить набір плагінів та скриптів, які можна використовувати для синхронізації рухів губ аватара зі звуками мовлення;

- Platform. містить компоненти для додавання функціональності Oculus Platform Solutions до додатку;
- Sample Framework. цей каталог містить приклади, що демонструють основні концепції Unity Integration;
- Spatializer. містить скрипти і плагіни для додавання джерел звуку та створення враження, що звук виходить з певного напрямку. Детальну інформацію знайдете в розділі Oculus Native Spatializer для Unity;
- VoiceMod. містить плагін та набір скриптів для модифікації вхідних аудіосигналів;
- Voice SDK. дозволяє додавати звукові взаємодії до AR/VR-додатків. Дозволяє гравцям використовувати свій голос для заклинань, керування кораблями або знищення зомбі;
- VR. цей каталог містить утиліти Oculus VR, які включають набір скриптів та префабів для розробки VR [30].

У таблиці 2.4 було проведене порівняння доступних SDK для роботи з VR.

Таблиця 2.4 – Порівняння SDK для роботи з VR у Unity

Критерій	VRIF	AutoHand	Oculus
Кількість готових ігрово-орієнтованих компонентів для перевикористання	Висока	Середня	Низька
Підтримка нативних функцій Окулусу	Немає	Немає	Є
Автоматична генерація поз рук	Немає	Є	Немає
Мій особистий досвід роботи	Великий	Малий	Середній

Висновок. Висновок, який ми можемо зробити — що в цілому, SDK пропонують більш-менш схожі можливості, але на фоні VRIF і AutoHand, SDK Oculus має нативний для гарнітури Oculus функціонал, такий як LipSync, VoiceMod, Voice SDK, підтримка скану положення рук, але через більшу кількість заготовок у VRIF, та моєму особистому досвіду роботи з ним, вибір був зроблений в його користь, крім того, SDK Oculus можна комбінувати з іншими SDK, щоб отримати доступ до деяких функцій, і не тільки нативних.

2.4 Вибір та обґрунтування Render Pipeline

Universal Render Pipeline (URP) є масштабованим багатоплатформним; рендер пайплайном, побудованим на основі Scriptable Render Pipeline. Завдяки своїй масштабованості, можливості налаштування та багатому набору функцій, URP надає творчу свободу у будь-якому типі проєкту, від стилізованих візуальних ефектів до фізично заснованого рендерингу. Незалежно від платформи, URP дозволяє створювати вражаючі графічні ефекти та досягати високої продуктивності. Він забезпечує підтримку широкого спектра функцій, включаючи реалістичне освітлення, тіні, відображення, пост-ефекти, а також масовий рендеринг частинок і частково прозорих об'єктів. Крім того, URP підтримує широку набір графічних API, таких як DirectX, Vulkan і Metal, що робить його сумісним з різними платформами, включаючи Windows, macOS, iOS, Android та інші. Він також надає можливості для налаштування рендерингових проходів, матеріалів, ефектів, туману та інших параметрів, щоб відповідати потребам проєкту. Загалом, Universal Rendering Pipeline є потужним і гнучким рендеринговим рішенням, яке дозволяє реалізувати творчі задуми та отримати високоякісні графічні результати у проєкті [10].

High-Defitnion Render Pipeline (HDRP) є рендер пайплайном, розробленим компанією Unity для сучасних платформ, сумісних з обчислювальними шейдерами (compute shaders) [32]-[34].

HDRP використовує фізично-основані техніки освітлення, лінійне освітлення, HDR освітлення та налаштовується гібридну архітектуру візуалізації з відкладеним/прямим освітленням на основі тайлів/кластерів. Він надає вам інструменти, необхідні для створення програм, таких як ігри, технічні демонстрації та анімації, на високому графічному рівні [32]-[34].

Гнучка архітектура візуалізації: HDRP пропонує конфігуровану гібридну архітектуру візуалізації з використанням тайлів/кластерів. Це дозволяє ефективно розподіляти обчислювальні ресурси та забезпечувати оптимальну продуктивність в різних сценах та платформах [32]-[34].

Розширена підтримка ефектів: HDRP надає розширену підтримку для різних ефектів, таких як об'єктна розмитість, водяна поверхня, туман, дим, вогонь, екранні тіні, HBAO, SSAO, та багато інших. Це дозволяє розробникам створювати більш іммерсивні та реалістичні візуальні ефекти у своїх проектах [32]-[34].

HDRP використовується для розробки ігор, технічних демонстрацій, анімацій та інших візуальних додатків, де важлива висока графічна якість. Він дозволяє розробникам створювати реалістичні та вражаючі візуальні світи, за допомогою яких користувачі можуть зануритися у незабутній досвід [32]-[34].

Взамін на продвинуті функції для візуалізації, HDRP куди більш вимогливий до апаратного забезпечення, що може вплинути як на час розробки, так і знизити охоплення аудиторії.

У таблиці 2.5 було проведено порівняння доступних для використання рендер пайплайнів.

Таблиця 2.5 – Порівняння рендер пайплайнів

Критерій	SRP	URP	HDRP
Продуктивність	Висока	Збалансована	Низька
Застосування	Реалістичний, стилізований візуал	Реалістичний, стилізований візуал	Реалістичний
Наявність продвинутих функцій для візуалізації	Мало, потребує самостійного розширення	Середня кількість	Велика кількість
Простота використання	Проста, або складна при розширенні	Середня	Середня

Висновок: використання рендер пайплайну хоч і не обов’язкове, але, вони дають доступ до деяких функцій, аналоги яких дуже складно реалізувати. Наприклад замість зручного візуального інструменту для роботи з шейдерами, Shader Graph, потрібно самостійно писати код за допомогою Shader Lab (грубо кажучи OpenGL), а у зв’язку з потребою данного функціоналу, та стилізації графіки було обрано URP.

2.5 Інші інструменти, які були застосовані при розробці додатку

При розробці додатку також були використані наступні інструменти:

- MMD4Mecanim – це невеличкий плагін, потрібний для імпорту MikuMikuDance 3D моделей в Unity;
- RealToon – ассет, потужний шейдер, який дає змогу досягти анімешного вигляду матеріалів, тобто різкі анімешні тіні, аутлайн, освітлення, і т.д. Як приклад, ігри, які використовують подібні шейдери це

The Legend of Zelda: Breath of the Wild, Genshin Impact, серія Honkai, серія Ni No Kuni;

- TextMeshPro – плагін для роботи з текстом (для UI), який насправді вже став частиною UnityEngine, на заміну давньої жахливої стандартної реалізації UI тексту;

- DynamicBone – плагін, який по більшій мірі використовується для надання фізичних властивостей окремим частинам рігу персонажів, в тому числі волосся та одягу, яка була використана для динамічної анімації хвосту та одягу НІП;

- EzySlice – невеличкий плагін, який обгортає математичні операції, в прості функції, потрібні для того, щоб різати на частини 3D моделі в рантаймі;

- DOTween – невід'ємна частина кожного мого проєкту, плагін, який дозволяє анімувати все і вся за допомогою коду, путем обгортання математичних операцій, інтерполяцій, ease-функцій, корутини, та інші. Має колбеки, та інструменти для будування черг із анімацій. Був використаний для анімації UI елементів, VFX, та локомоушену НІП;

- Recorder – плагін, який дозволяє записувати відео, та робити скріншоти у високому роздільності;

- Blender – програма для 3D моделювання, яку я використовував для ретопології деяких моделей;

- Figma – інструмент для дизайну, я використовував для малювання UI елементів.

2.6 Структурна схема розробки

Архітектура додатку базується на принципах подійно-орієнтованого програмування. Основна маса класів не пов'язана між собою, в той час як контроллер передає їм вхідні дані, і очікує результату.

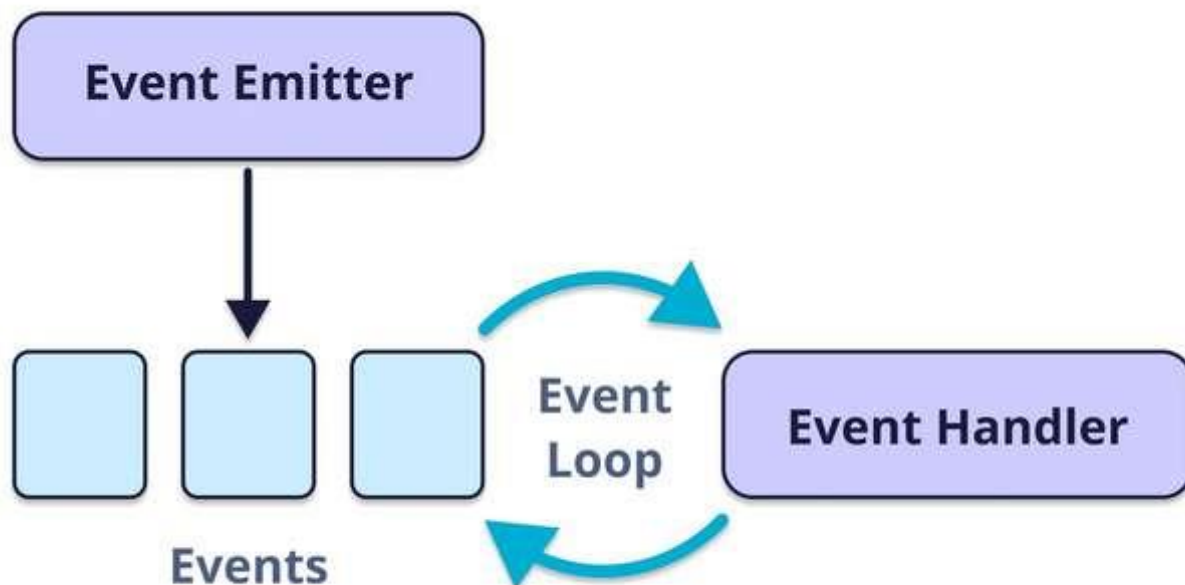


Рисунок 2.1 – Архітектура на базі принципів подійно-орієнтованого програмування

2.7 Висновки за розділом 2

На цьому етапі було успішно вирішено завдання вибору технології розробки. До них включалися: опис ігрового двигуна, мови програмування серед тих, які двигун в тій чи іншій формі підтримує, вибір необхідних інструментів, бібліотек і асетів для спрощення та прискорення процесу розробки, а також вибір шаблону проєктування. Виконання було успішно здійснене шляхом аналізу та дослідження доступних опцій, їх порівняння, та вибір найбільш підходящого для вирішення заданої проблеми оптимальним шляхом, що допомогло досягти поставленої мети другого розділу.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Проєктування моделей системи

Уніфікована мова моделювання (UML) - це мова моделювання загального призначення, яка задає набір правил для стандартизації візуалізації дизайну системи [35].

UML надає стандартну нотацію для багатьох типів діаграм, які можна приблизно розділити на 3 основні групи: діаграми поведінки, діаграми взаємодії та діаграми структури. Головна перевага використання UML - це уніфікований підхід для побудови схем структур розробляючихся систем, і не обов'язково пов'язаних саме з програмуванням, а може також включати схеми процесів у компаніях [35].

Так як представлений додаток не взаємодіє ні з якими модулями, наприклад як бази даних, сервера або API, для розрахунку архітектури нам знадобиться лише діаграма класів, для того щоб розмітити базову архітектуру взаємодії класів, фігуруючих в додатку.

Основною метою класових діаграм є побудова представлення структури додатка. Це тип діаграм, який широко використовується для проєктування та надає переваги при використуванні в об'єктно-орієнтованих мовах програмування. Він є одним з найпопулярніших різновидів діаграм UML [36]. Нижче наведено цілі класових діаграм:

- опис основних відповідальностей елементів системи;
- представлення об'єктної моделі для складних систем;
- скорочення часу фактичного написання коду, надаючи загальну картину того, як додаток має бути структурований;
- можливість виявити потенційні проблеми заздалегідь;
- вона корисна як для зацікавлених сторін та розробників можливістю оглянути заздалегідь мару розробляючогося продукту [35]-[36].

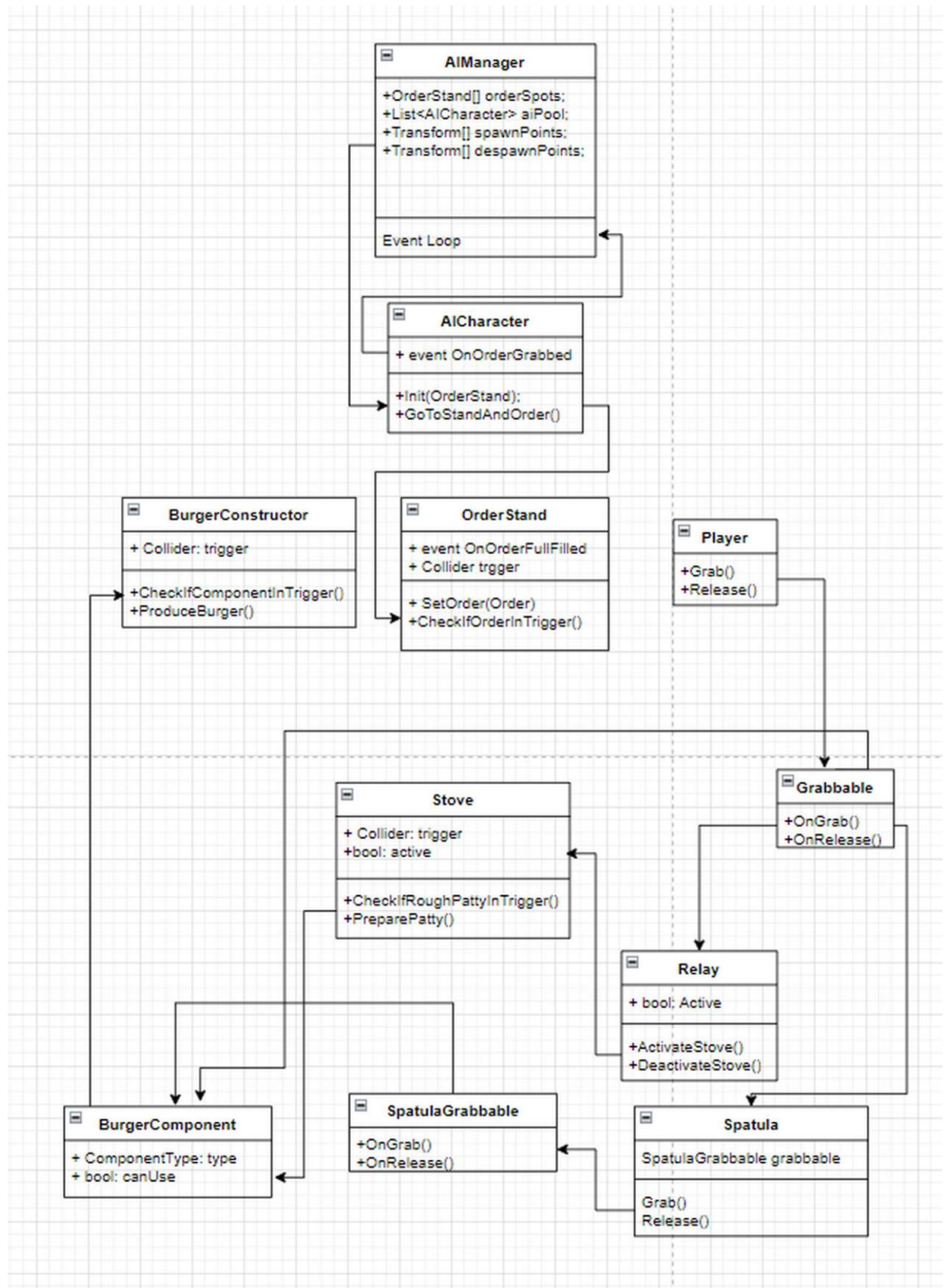


Рисунок 3.1 – Діаграма компонентів

3.2 Проектування ігрового рівня

Скріншоти рівня наведено на рисунках 3.2 – 3.9.



Рисунок 3.2 – Ігрова кімната



Рисунок 3.3 – Основна маса інтерактивних елементів гри



Рисунок 3.4 – Элементы оточення №1



Рисунок 3.5 – Элементы оточення №2



Рисунок 3.6 – Элементы оточення №3



Рисунок 3.7 – Элементы оточення №4



Рисунок 3.8 – Неігрові персонажі (Аніме дівчата)

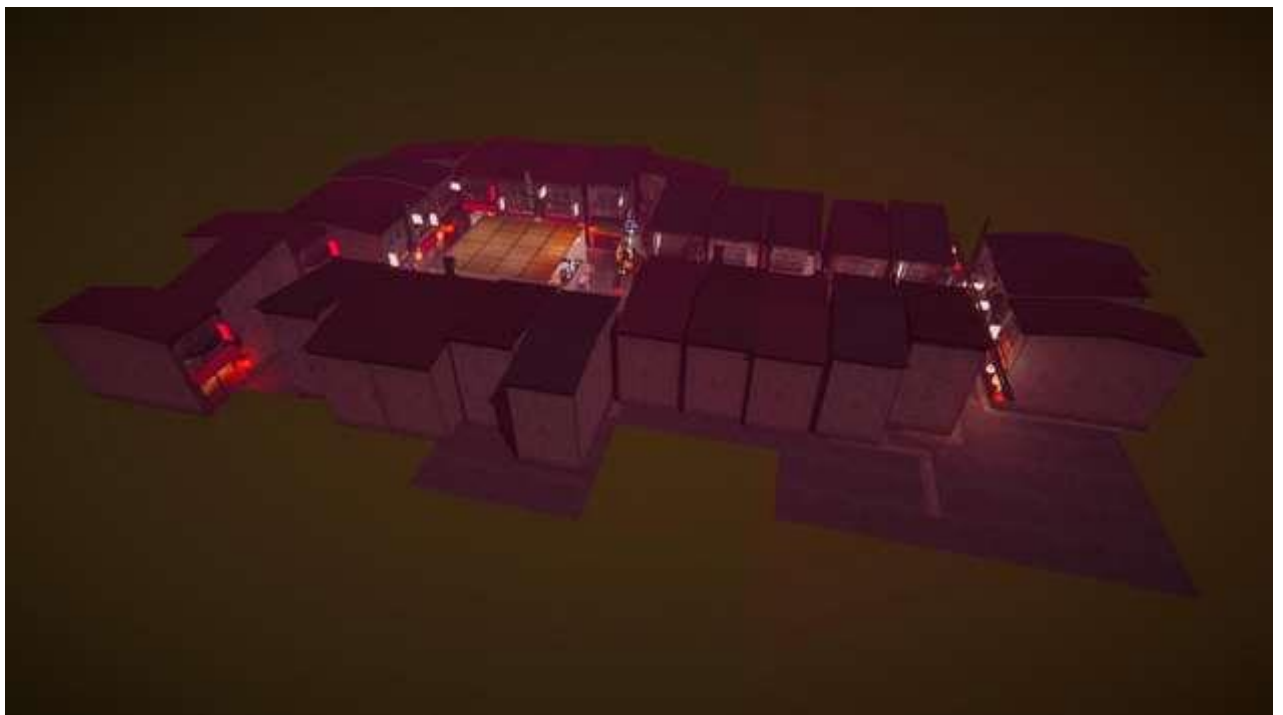


Рисунок 3.9 – Рівень загалом

3.3 Висновки за розділом 3

За допомогою UML, було проведено логічне та фізичне проєктування додатку. Спираючись на зазначені принципи розробки було прийнято рішення, щодо подальшої реалізації програмного забезпечення.

Розроблені прототипи в майбутньому значно допоможуть в розробці додатку, а модульний, та подійно-орієнтований підхід дозволить спростити та прискорити підтримку та доповнення функціоналу.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення та механіка додатку

Сутність програми заключається в передачі імерсійного досвіду буття працівником вуличного кафе в Японії та з аніме дівчатками.

Загальними можливостями є:

- відображення картинки у VR;
- взаємодія з оточенням за допомогою VR контролерів.

Також присутні інтерактивні об'єкти.

Функціями замовника є:

- підійти до стійки з замовленнями;
- зробити замовлення;
- забрати блюдо;
- піти геть.

Функціями кухні є:

- переміщати інгредієнти;
- виконувати специфічні до інгредієнтів дії;
- збирати блюдо з інгредієнтів.

Функціями пічки є:

- причепити котлету до печі;
- смажити котлету.

Функціями стійки з лимонадом є:

- причепити келих до стійки;
- налити зазначений лимонад.

4.2 Умови виконання додатку

Мінімальні системні вимоги та ПЗ, що необхідні для функціонування додатку Oculus та представленого додатку:

- підключений шолом Oculus Quest або Oculus Quest 2;
- встановлений і запущений застосунок Oculus.

4.3 Запуск додатку

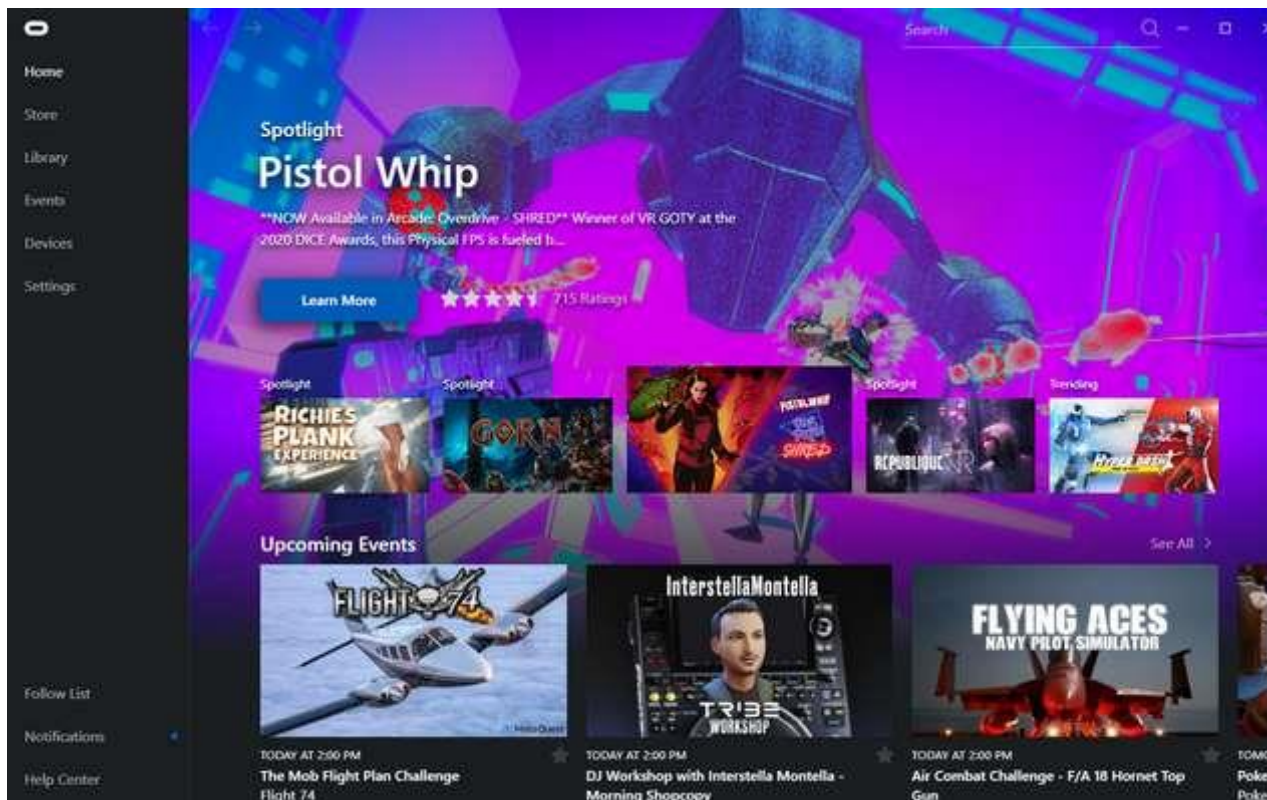


Рисунок 4.1 – Екран запущеного додатку Oculus

Для запуску застосунку потрібно запустити застосунок Oculus, і після цього можна запускати гру. Для запуску додатку потрібно запустити виконавчий файл “Japanese Street Food Cafe.exe”.

4.4 Методика та результати тестування

Для тестування було проведено білд проєкта в режимі Development Mode, який включає в білд підтримку інструментів для дебагінгу, показує консоль з логами, та автоматично пише лог файли.

Далі було запущено застосунок Oculus, який портібен для ініціалізації VR простору, рисунок 4.2.

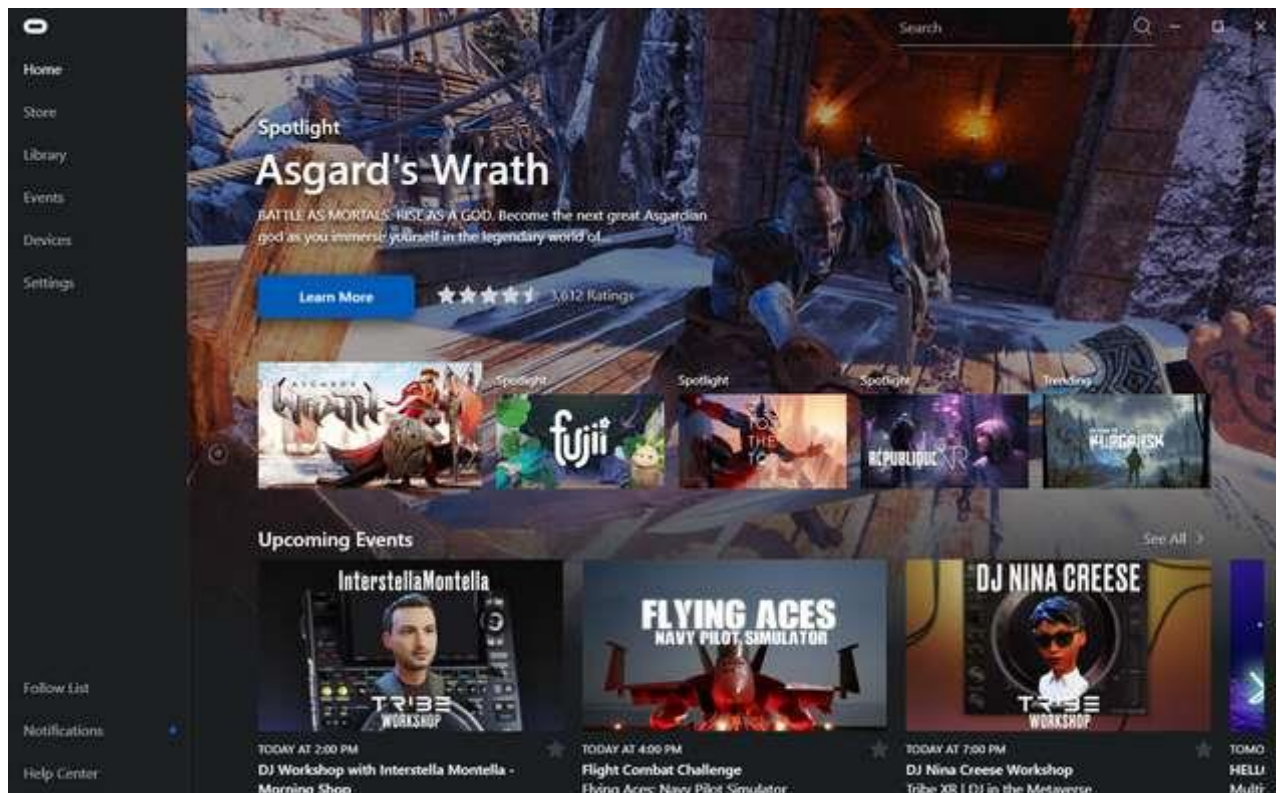


Рисунок 4.2 – Екран запущеного додатку Oculus

Після цього запуск виконавчого файла “Japanese Street Food Cafe”, рисунок 4.3.

Japanese Street Food Cafe_BurstDebugIn...	5/31/2023 5:21 AM	File folder	
Japanese Street Food Cafe_Data	5/31/2023 8:31 AM	File folder	
MonoBleedingEdge	5/31/2023 8:31 AM	File folder	
Japanese Street Food Cafe.exe	5/31/2023 8:31 AM	Application	639 KB
UnityCrashHandler64.exe	5/31/2023 8:31 AM	Application	1,098 KB
UnityPlayer.dll	5/31/2023 8:31 AM	Application exten...	28,580 KB

Рисунок 4.3 – Запуск виконавчого файла

Далі гра почалася, і клієнти вже на підході, що можна побачити на рисунку 4.4 .



Рисунок 4.4 – Функції замовників

Наступним було перевірено функціонал автомату з напоями, що можна побачити на рисунку 4.5.



Рисунок 4.5 – Процес налиття напоїв

Далі було протестовано функціонал печі, що можна побачити на рисунках 4.6-4.9.



Рисунок 4.6 – Взяття котлети із холодильника



Рисунок 4.7 – Покладання котлети на піч



Рисунок 4.8 – Запуск печі через інтерактивне реле



Рисунок 4.9 – Беремо котлету, обов'язково лопаткою, бо гаряче

Наступним було перевірено процес збору бургера і передачі замовлення, рисунки 4.10-4.14



Рисунок 4.10 – Роз'єднання булочки, взявши її двома руками



Рисунок 4.11 – Збірка бургера згідно підказки



Рисунок 4.12 – Збірка бургера згідно підказки



Рисунок 4.13 – Подача замовлення



Рисунок 4.14 – Працевдатний додаток

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було реалізовано головну мету – програмну реалізацію гри у віртуальній реальності на Unity Engine. Для цього було виконано ряд завдань:

- проаналізувати проблемну область ринку VR ігор;
- дослідити існуючі аналоги ігор на схожу тематику;
- визначити технічні засоби програмної реалізації нового рішення;
- спроектувати архітектуру розробки нового застосунку;
- на основі виконаного проєктування реалізувати новий додаток;
- дослідити та протестувати нове рішення.

Розроблена структура веб-застосунку була реалізована із використанням UnityEngine, C#, VRIF, URP, MMD4Mecanim, EzySlice, RealToon, DOTween, DynamicBone, TextMeshPro.

Було створено сценарій тестування додатку, та використовуючи його для перевірки основної логіки та функціональності було отримано результати свідчать про успішну реалізацію додатку. В результаті виконання завдання була розроблена система, яка повністю задовольняє всі вимоги, що були визначені на етапі розробки та викладені в технічному завданні. Розроблена система є зручною у використанні за допомогою VR хедсету, та виконує всі необхідні функції.

Усі завдання, які були поставлені перед дипломною кваліфікаційною роботою бакалавра, були успішно виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Unity (game engine) [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))
2. Unity Technologies [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Unity_Technologies
3. Virtual reality [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Virtual_reality
4. Tompson, S. VR Applications: 23 Industries using Virtual Reality [Electronic resource] / S. Tompson. – Access mode: <https://virtualspeech.com/blog/vr-applications>
5. Mongeon M. The Applications and History of Virtual Reality & VR Headsets [Electronic resource] / M. Mongeon. – Access mode: <https://www.mentorworks.ca/blog/business-strategy/applications-and-history-of-vr-headsets/>
6. Virtual reality headset [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Virtual_reality_headset
7. History of virtual reality: Timeline [Electronic resource]. – Access mode: <https://www.verdict.co.uk/history-virtual-reality-timeline/>
8. The rise and fall and rise of virtual reality [Electronic resource]. – Access mode: https://www.theverge.com/a/virtual-reality/oral_history
9. Universal render pipeline (URP) [Electronic resource]. – Access mode: <https://unity.com/srp/universal-render-pipeline>
10. VR Interaction Framework [Electronic resource]. – Access mode: <https://assetstore.unity.com/packages/templates/systems/vr-interaction-framework-161066>
11. Cooking Simulator [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Cooking_Simulator
12. Cooking Simulator VR [Electronic resource]. – Access mode: https://store.steampowered.com/app/1358140/Cooking_Simulator_VR/

13. The Cooking Game VR [Electronic resource]. – Access mode: https://store.steampowered.com/app/857180/The_Cooking_Game_VR/
14. Arango B. Pros and Cons of Virtual Reality [Electronic resource] / B. Arango – Access mode: <https://filmora.wondershare.com/virtual-reality/pros-cons-virtual-virtual.html>
15. Virtual Reality and Humanity [Electronic resource]. – Access mode: <https://vr.space/news/education/vr-pros-cons/>
16. Titov A. Why is the unity game engine considered the best? Pros and cons [Electronic resource] / A. Titov. – Access mode: <https://stepico.com/blog/why-is-unity-the-best-game-engine-pros-and-cons/>
17. Bullock J. Why to Use Unity: Top 8 Reasons [Electronic resource] / J. Bullock. – Access mode: <https://turbofuture.com/computers/Why-Should-I-Use-Unity-Top-8-Reasons>
18. The 5 Best Uses for Unity (Besides Game Development) [Electronic resource]. – Access mode: <https://transforminteractive.com/5-best-uses-for-unity/>
19. What Makes Unity So Popular in Game Development? [Electronic resource]. – Access mode: <https://www.arnia.com/what-makes-unity-so-popular-in-game-development/>
20. Kaur S. A. Unity vs Unreal: Which Game Engine Should You Choose? [Electronic resource] / S. A. Kaur, R. Johnes. – Access mode: <https://hackr.io/blog/unity-vs-unreal-engine>
21. Unity vs. Unreal: What to Choose for Your Project? [Electronic resource]. – Access mode: <https://program-ace.com/blog/unity-vs-unreal/>
22. Wirtz B. Unreal or Unity Engine: Fight for Supremacy [Electronic resource] / B. Wirtz. – Access mode: <https://www.gamedesigning.org/engines/unity-vs-unreal/>
23. Unity vs Unreal Engine in 2022 [Electronic resource]. – Access mode: <https://medium.com/@shakiroslann/unity-vs-unreal-engine-in-2022-9879543d420c>

24. Will Boo ever come back? [Electronic resource]. – Access mode: <https://forum.unity.com/threads/will-boo-ever-come-back.375988/#:~:text=The%20reason%20it%20was%20deprecated,shrank%20to%20almost%20non%2Dexistent.>

25. Deprecating JavaScript (UnityScript) is a good thing. [Electronic resource]. – Access mode: <https://news.ycombinator.com/item?id=22688387>

26. Unity Script deprecated?! [Electronic resource]. – Access mode: <https://forum.unity.com/threads/unity-script-deprecated.520539/>

27. A tour of the C# language [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

28. C Sharp (programming language) [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))

29. What is C# Programming? A Beginner's Guide [Electronic resource]. – Access mode: <https://www.pluralsight.com/blog/software-development/everything-you-need-to-know-about-c->

30. Auto Hand - VR Physics Interaction [Electronic resource]. – Access mode: <https://assetstore.unity.com/packages/tools/game-toolkits/auto-hand-vr-physics-interaction-165323>

31. Oculus Integration [Electronic resource]. – Access mode: <https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022>

32. Render pipelines [Electronic resource]. – Access mode: <https://docs.unity3d.com/Manual/render-pipelines.html>

33. High Definition Render Pipeline overview [Electronic resource]. – Access mode: <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@16.0/manual/index.html>

34. High definition render pipeline (HDRP) [Electronic resource]. – Access mode: <https://unity.com/srp/High-Definition-Render-Pipeline>

35. Unified Modeling Language [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Unified_Modeling_Language

36. UML Class Diagram [Electronic resource]. – Access mode:
<https://www.javatpoint.com/uml-class-diagram>

ДОДАТОК А
Технічне завдання

Вступ

Найменування: «Japanese Street Food Cafe Simulator».

Розроблювана система – це гра, симулятор, у віртуальній реальності, в якій потрібно обслуговувати клієнтів кафе, готувати замовлення на час.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка VR гри в середовищі Unity Engine», затверджене наказом Національного університету «Запорізька політехніка» № 87 від 04 квітня 2023 р.

A.2 Основні вимоги до програми, що розробляється

Це гра у віртуальній реальності ціллю якої є передати імерсивний досвід буття працівником стріт фуд кафе в Японії.

A.2.1 Вимоги до функціональних характеристик

Загальними можливостями є:

- відображення картинки у VR;
- взаємодія з оточенням за допомогою VR контролерів.

Також присутні інтерактивні об'єкти.

Функціями замовника є:

- підійти до стійки з замовленнями;
- зробити замовлення;
- забрати блюдо;
- піти геть.

Функціями кухні є:

- переміщати інгредієнти;
- виконувати специфічні до інгредієнтів дії;
- збирати блюдо з інгредієнтів.

Функціями пічки є:

- причепити котлету до печі;
- смажити котлету.

Функціями стійки з лимонадом є:

- причепити келих до стійки;
- налити зазначений лимонад.

A.2.2 Вимоги до інтерфейсу програми

Англомовний інтерфейс, зрозумілий для користувачів з різних країн світу.

Система повинна мати інтуїтивний та зручний інтерфейс, інтуїтивні механіки та індикації, які надавали би можливість зручного і позитивного досвіду під використання додатку.

A.2.3 Вимоги до надійності

У зв'язку з тим, що додаток є однокористувацьким, без взаємодії з інтернету, вимог до надійності немає.

A.2.4 Умови експлуатації

Гра повинна коректно запускатися та функціонувати на платформах Windows 32/64 bit, 7, 8.1, 10, 11.

Вимоги до апаратного забезпечення:

- Oculus Quest/Oculus Quest 2 наполегливо рекомендується, не дивлячись на наявність емулятора;
- відеокарта, Nvidia Geforce 1050TI / AMD Radeon RX470 або вище з підтримкою DirectX 11 або вище;
- процесор Intel Core i5-7600 / AMD Ryzen 3 2300X або вище;
- 8 GB оперативної пам'яті або більше;
- 2 GB пам'яті на жорсткому диску;
- 2 GB пам'яті на жорсткому диску;
- встановлений і заздалегідь запущений Oculus Hub.

ДОДАТОК Б
Текст програми

```

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

public class AIManager : MonoBehaviour
{

    [SerializeField] private OrderStand[] orderSpots;
    [SerializeField] private List<AICharacter> aiPool;

    [SerializeField] private Transform[] spawnPoints;
    [SerializeField] private Transform[] despawnPoints;

    private int _poolIndex = 0;

    private void Start()
    {
        aiPool.Shuffle();

        CreateVisitors();
    }

    private void CreateVisitors()
    {
        foreach (var orderSpot in orderSpots)
        {
            if (!orderSpot.IsBusy)

```

```

        {
            var character = aiPool[_poolIndex];
            var spawnPoint = UnityEngine.Random.Range(0,
spawnPoints.Length);

character.transform.SetPositionAndRotation(spawnPoints[spawnPoint].position,
spawnPoints[spawnPoint].rotation);

            character.gameObject.SetActive(true);

            character.OrderAt(orderSpot);
            character.OnOrderGrabbed += OnOrderGrabbed;

            _poolIndex++;

            if (_poolIndex >= orderSpots.Length)
            {
                _poolIndex = 0;

                aiPool.Shuffle();
            }
        }
    }

private void OnOrderGrabbed(AICharacter character)
{
    character.OnOrderGrabbed -= OnOrderGrabbed;
}

```

```

character.SetAgentDestination(despawnPoints[UnityEngine.Random.Range(0,
despawnPoints.Length)].position, (character) =>
    {
        character.gameObject.SetActive(false);
    });

    CreateVisitors();
}
}

```

```

using DG.Tweening;
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

```

```

public class AICharacter : MonoBehaviour
{
    public event System.Action<AICharacter> OnOrderGrabbed;

    [SerializeField] private Animator animator;
    [SerializeField] private NavMeshAgent agent;
    [SerializeField] private float walkingMagnitude = 0.01f;

    [SerializeField] private string[] randomBlendShapesAnimations;

    [SerializeField] private Vector2 blendShapeAnimDelayRange = new
Vector2(3f, 7f);

    private bool _isBusy = false;

```

```

private bool _shouldFireFinish = false;
private OrderStand _agentTargetStand;

private System.Action<AICharacter> _agentReachedCallback;

public bool IsBusy => _isBusy;

private void Start()
{
    if (randomBlendShapesAnimations.Length > 0)
        StartCoroutine(PlayRandomBlendShape());
}

private IEnumerator PlayRandomBlendShape()
{
    yield return new
    WaitForSeconds(Random.Range(blendShapeAnimDelayRange.x,
    blendShapeAnimDelayRange.y));

    animator.Play(randomBlendShapesAnimations[Random.Range(0,
    randomBlendShapesAnimations.Length)]);

    StartCoroutine(PlayRandomBlendShape());
}

public void OrderAt(OrderStand stand)
{
    _agentTargetStand = stand;

    _agentTargetStand.IsBusy = true;

```

```

        _isBusy = true;

        void OnStandReached(AICharacter _)
        {
            _agentTargetStand.GenerateRandomOrder();
            _agentTargetStand.OnOrderFulfilled += OnOrderFulfilled;
        }

        SetAgentDestination(_agentTargetStand.OrderAiSpot.position,
OnStandReached);
    }

    public void SetAgentDestination(Vector3 position,
System.Action<AICharacter> callback = null)
    {
        agent.SetDestination(position);

        _shouldFireFinish = true;

        _agentReachedCallback += callback;
    }

    private void Update()
    {
        if (agent.velocity.magnitude > walkingMagnitude)
        {
            animator.SetFloat("WalkingSpeed", 1f);
        }
        else
        {

```

```

        animator.SetFloat("WalkingSpeed", 0f);
    }

    if (_shouldFireFinish && Vector3.Distance(transform.position,
agent.destination) <= agent.stoppingDistance)
    {

transform.DORotateQuaternion(_agentTargetStand.OrderAiSpot.rotation,
1f).SetEase(Ease.Linear).OnComplete(() =>
    {
        animator.CrossFadeInFixedTime("Wave", .15f);
    });

    _shouldFireFinish = false;

    OnPointReached();
}

private void OnPointReached()
{
    _agentReachedCallback?.Invoke(this);

    _agentReachedCallback = null;
}

private void OnOrderFulfilled()
{
    _agentTargetStand.OnOrderFulfilled -= OnOrderFulfilled;
}

```

```

        _agentTargetStand.IsBusy = false;
        _isBusy = false;

        OnOrderGrabbed?.Invoke(this);
    }
}

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

public enum BurgerComponentEnum
{
    BunBot,
    Salad,
    Tomato,
    Patty,
    Cheese,
    BunTop
}

public class BurgerConstructor : MonoBehaviour
{
    [SerializeField] private GameObject burgerConstructorHint;
    [SerializeField] private Renderer bunBotHint;
    [SerializeField] private Renderer saladHint;
    [SerializeField] private Renderer tomatoHint;
    [SerializeField] private Renderer pattyHint;
    [SerializeField] private Renderer cheeseHint;
    [SerializeField] private Renderer bunTopHint;

```

```

[SerializeField] private GameObject burgerConstructorDone;
[SerializeField] private Renderer bunBotDone;
[SerializeField] private Renderer saladDone;
[SerializeField] private Renderer tomatoDone;
[SerializeField] private Renderer pattyDone;
[SerializeField] private Renderer cheeseDone;
[SerializeField] private Renderer bunTopDone;

[SerializeField] private Color highlightHintColor;
[SerializeField] private Color initialHintColor;

[SerializeField] private ParticleSystem doneParticleSystem;
[SerializeField] private AudioSource doneAudioSource;

[SerializeField] private GameObject burgerPrefab;
[SerializeField] private Transform burgerPrefabSpawnPoint;

private BurgerComponentEnum[] _burgerRecipe;
private List<BurgerComponentEnum> _currentBurgerComponents;

private bool _constructing = true;

private void Start()
{
    _burgerRecipe = new[] {
        BurgerComponentEnum.BunBot,
        BurgerComponentEnum.Salad,
        BurgerComponentEnum.Tomato,
        BurgerComponentEnum.Patty,

```

```

        BurgerComponentEnum.Cheese,
        BurgerComponentEnum.BunTop
    };

    ResetConstruction();
}

private void ResetConstruction()
{
    _currentBurgerComponents = new List<BurgerComponentEnum>();

    burgerConstructorHint.SetActive(true);
    burgerConstructorDone.SetActive(true);

    _constructing = true;

    UpdateVisuals();
}

private void StopConstructing()
{
    burgerConstructorHint.SetActive(false);
    burgerConstructorDone.SetActive(false);

    _constructing = false;
}

private void UpdateVisuals()
{
    for (int i = 0; i < _burgerRecipe.Length; i++)

```

```

        {
            if (i < _currentBurgerComponents.Count)
            {

GetComponentHint(_burgerRecipe[i]).gameObject.SetActive(false);

GetComponentDone(_burgerRecipe[i]).gameObject.SetActive(true);
            }
            else if (i == _currentBurgerComponents.Count)
            {

GetComponentHint(_burgerRecipe[i]).gameObject.SetActive(true);

GetComponentDone(_burgerRecipe[i]).gameObject.SetActive(false);
            }
            else
            {

GetComponentHint(_burgerRecipe[i]).gameObject.SetActive(false);

GetComponentDone(_burgerRecipe[i]).gameObject.SetActive(false);
            }
        }
    }

    private void Update()
    {
        if (_constructing)
        {

```

```

        burgerConstructorHint.transform.Rotate(Vector3.forward * 30f *
Time.deltaTime);
    }
}

private void OnTriggerStay(Collider other)
{
    var component = other.GetComponent<BurgerComponent>();

    if (component != null && _currentBurgerComponents.Count <=
_burgerRecipe.Length
        && _burgerRecipe[_currentBurgerComponents.Count] ==
component.BurgerComponentType)
    {
        if ((component.SpatulaGrabbable != null &&
component.SpatulaGrabbable.BeingGrabbed) || component.Grabbable.BeingHeld)
        {
            GetComponentHint(_burgerRecipe[_currentBurgerComponents.Count]).material.c
olor = highlightHintColor;
        }
        else
        {
            GetComponentHint(_burgerRecipe[_currentBurgerComponents.Count]).material.c
olor = initialHintColor;

            //component.Grabbable.DropItem(component.Grabbable.GetPrimaryGrabber(),
false, true);

```

```

_currentBurgerComponents.Add(component.BurgerComponentType);

    UpdateVisuals();

    if (_currentBurgerComponents.Count == _burgerRecipe.Length)
    {
        StopConstructing();

        doneParticleSystem.Play();
        doneAudioSource.Play();

        Instantiate(burgerPrefab, burgerPrefabSpawnPoint.position,
burgerPrefabSpawnPoint.rotation);
    }

    component.gameObject.SetActive(false);
}
}

private void OnTriggerExit(Collider other)
{
    var component = other.GetComponent<BurgerComponent>();

    if (component != null && _currentBurgerComponents.Count <=
_burgerRecipe.Length
        && _burgerRecipe[_currentBurgerComponents.Count] ==
component.BurgerComponentType)

```

```

        {
            if ((component.SpatulaGrabbable != null &&
component.SpatulaGrabbable.BeingGrabbed) || component.Grabbable.BeingHeld)
            {

GetComponentHint(_burgerRecipe[_currentBurgerComponents.Count]).material.c
olor = initialHintColor;
            }
            else
            {

GetComponentHint(_burgerRecipe[_currentBurgerComponents.Count]).material.c
olor = initialHintColor;
            }
        }
    }

private Renderer GetComponentHint(BurgerComponentEnum
component)
    {
        return component switch
        {
            BurgerComponentEnum.BunBot => bunBotHint,
            BurgerComponentEnum.Salad => saladHint,
            BurgerComponentEnum.Tomato => tomatoHint,
            BurgerComponentEnum.Patty => pattyHint,
            BurgerComponentEnum.Cheese => cheeseHint,
            BurgerComponentEnum.BunTop => bunTopHint,
            _ => null,
        };
    }

```

```
}
```

```
private Renderer GetComponentDone(BurgerComponentEnum  
component)
```

```
{
```

```
    return component switch
```

```
{
```

```
    BurgerComponentEnum.BunBot => bunBotDone,
```

```
    BurgerComponentEnum.Salad => saladDone,
```

```
    BurgerComponentEnum.Tomato => tomatoDone,
```

```
    BurgerComponentEnum.Patty => pattyDone,
```

```
    BurgerComponentEnum.Cheese => cheeseDone,
```

```
    BurgerComponentEnum.BunTop => bunTopDone,
```

```
    _ => null,
```

```
};
```

```
}
```

```
}
```

```
using BNG;
```

```
using System.Collections;
```

```
using UnityEngine;
```

```
using DG.Tweening;
```

```
public class DrinkSpot : MonoBehaviour
```

```
{
```

```
    [SerializeField] private SnapZone snapZone;
```

```
    [SerializeField] private OrderableDrinks drinkType;
```

```
    [SerializeField] private ParticleSystem fillParticleSystem;
```

```
    [SerializeField] private Color liquidColor = Color.white;
```

```

[SerializeField] private Vector3 liquidTargetScale = new
Vector3(0.1050095f, 0.0002893895f, 0.1050095f);

[SerializeField] private float liquidTargetY = 0.1491f;

private bool drinkSnapped = false;

private void Start()
{
    snapZone.OnSnapEvent.AddListener(OnDrinkSnapped);
}

private void OnDrinkSnapped(Grabbable grabbableDrink)
{
    drinkSnapped = true;

    grabbableDrink.enabled = true;

    grabbableDrink.gameObject.AddComponent<PreparedFood>().OrderableDrinkType = drinkType;
    grabbableDrink.gameObject.name = drinkType.ToString();

    StartCoroutine(FillDrink(grabbableDrink));
}

public interface ISpatulaGrabbableReceiver
{
    public string[] AllowedGrabbableNames { get; }

    public void Pass(SpatulaGrabbable grabbable);
}

```

```

        void ShowHighlight();
        void HideHighlight();
    }

    using EzySlice;
    using UnityEngine;

    public class Knife : MonoBehaviour
    {
        [SerializeField] private Transform startSlicePoint;
        [SerializeField] private Transform endSlicePoint;
        [SerializeField] private LayerMask sliceableLayer;
        [SerializeField] private VelocityEstimator velocityEstimator;

        private void FixedUpdate()
        {
            bool hasHit = Physics.Linecast(startSlicePoint.position,
            endSlicePoint.position, out RaycastHit hit, sliceableLayer);

            if (hasHit)
            {
                var target = hit.transform.gameObject;
                var sliceable = target.GetComponent<Sliceable>();

                Slice(sliceable);
            }
        }

        private void Slice(Sliceable target)
        {

```

```

        Vector3 velocity = velocityEstimator.GetVelocityEstimate();
        Vector3 planeNormal = Vector3.Cross(endSlicePoint.position -
startSlicePoint.position, velocity);
        planeNormal.Normalize();

        SlicedHull hull = target.gameObject.Slice(endSlicePoint.position,
planeNormal);

        if (hull != null)
        {
            GameObject upperHull = hull.CreateUpperHull(target.gameObject,
target.CrossSectionMaterial);
            upperHull.name = $"{target.gameObject.name} (Upper)";
            SetupHull(target, upperHull);

            GameObject lowerHull = hull.CreateLowerHull(target.gameObject,
target.CrossSectionMaterial);
            upperHull.name = $"{target.name} (Lower)";
            SetupHull(target, lowerHull);
        }

        Destroy(target.gameObject);
    }

    private void SetupHull(Sliceable target, GameObject hull)
    {
        target.SetupHull(hull.AddComponent<Sliceable>());
        hull.layer = LayerMask.NameToLayer("Sliceable");
    }
}

```

```

using System;
using System.Collections.Generic;

public static class ListExtensions
{
    private static readonly Random _rng = new();

    public static void Shuffle<T>(this IList<T> list)
    {
        int n = list.Count;
        while (n > 1)
        {
            n--;
            int k = _rng.Next(n + 1);
            T value = list[k];
            list[k] = list[n];
            list[n] = value;
        }
    }
}

```

```

using System;
using System.Collections.Generic;

public static class ListExtensions
{
    private static readonly Random _rng = new();

    public static void Shuffle<T>(this IList<T> list)
    {

```

```
int n = list.Count;
while (n > 1)
{
    n--;
    int k = _rng.Next(n + 1);
    T value = list[k];
    list[k] = list[n];
    list[n] = value;
}
}

using BNG;
using DG.Tweening;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public enum OrderableFood
{
    Burger
}

public enum OrderableDrinks
{
    Cola,
    Fanta,
    Sprite,
    Pepsi
}
```

```

public class OrderStand : MonoBehaviour
{
    public event System.Action OnOrderFulfilled;

    [SerializeField] private Transform orderAiSpot;
    [SerializeField] private CanvasGroup orderCanvasGroup;

    [SerializeField] private Image burgerSprite;

    [SerializeField] private Image colaImage;
    [SerializeField] private Image spriteImage;
    [SerializeField] private Image fantaImage;
    [SerializeField] private Image pepsiImage;

    private bool _isBusy = false;
    private bool _hasOrder = false;
    private OrderableFood[] _orderedFood;
    private OrderableDrinks[] _orderedDrinks;

    private List<PreparedFood> _foodInTrigger = new
List<PreparedFood>();

    public Transform OrderAiSpot => orderAiSpot;
    public bool HasOrder => _hasOrder;
    public bool IsBusy { get => _isBusy; set => _isBusy = value; }

    public void GenerateRandomOrder()
    {
        if (_hasOrder)

```

```

        return;

        _orderedFood = new[] { OrderableFood.Burger };
        _orderedDrinks = new[] { (OrderableDrinks)Random.Range(0, 4) };

        _hasOrder = true;

        foreach (var food in _orderedFood)
        {
            GetFoodImage(food).gameObject.SetActive(true);
        }

        foreach (var drink in _orderedDrinks)
        {
            GetDrinkImage(drink).gameObject.SetActive(true);
        }

        orderCanvasGroup.gameObject.SetActive(true);
        orderCanvasGroup.DOFade(1f, 1f);
    }

    private void FinishOrder()
    {
        _hasOrder = false;

        orderCanvasGroup.DOFade(0f, 1f).OnComplete(() =>
        {
            orderCanvasGroup.gameObject.SetActive(false);

            foreach (var food in _orderedFood)

```

```

        {
            GetFoodImage(food).gameObject.SetActive(false);
        }

        foreach (var drink in _orderedDrinks)
        {
            GetDrinkImage(drink).gameObject.SetActive(false);
        }
    });
}

private void OnTriggerEnter(Collider other)
{
    var preparedFood = other.GetComponent<PreparedFood>();

    if (preparedFood != null &&
!_foodInTrigger.Contains(preparedFood))
    {
        _foodInTrigger.Add(preparedFood);
    }
}

private void OnTriggerExit(Collider other)
{
    var preparedFood = other.GetComponent<PreparedFood>();

    if (preparedFood != null && _foodInTrigger.Contains(preparedFood))
    {
        _foodInTrigger.Add(preparedFood);
    }
}

```

```

    }

    private void Update()
    {
        if (!_hasOrder)
            return;

        bool foodFullfilled = true;
        List<PreparedFood> rightFood = new List<PreparedFood>();

        foreach (var food in _orderedFood)
        {
            var foodInTrigger = _foodInTrigger.Find(foodInTrigger =>
            foodInTrigger.IsFood && foodInTrigger.OrderableFoodType == food);

            if (foodInTrigger == null ||
            foodInTrigger.GetComponent<Grabbable>().BeingHeld)
            {
                foodFullfilled = false;
                break;
            }
            else
            {
                rightFood.Add(foodInTrigger);
            }
        }

        foreach (var drink in _orderedDrinks)
        {

```

```

        var foodInTrigger = _foodInTrigger.Find(foodInTrigger =>
foodInTrigger.IsDrink && foodInTrigger.OrderableDrinkType == drink);

```

```

        if (foodInTrigger == null ||
foodInTrigger.GetComponent<Grabbable>().BeingHeld)
        {
            foodFullfilled = false;
            break;
        }
        else
        {
            rightFood.Add(foodInTrigger);
        }
    }

```

```

    if (foodFullfilled)
    {
        foreach (var food in rightFood)
        {
            _foodInTrigger.Remove(food);
            Destroy(food.gameObject);
        }
    }

```

```

    OnOrderFulfilled?.Invoke();

```

```

    FinishOrder();

```

```

    }
}

```

```

private Image GetFoodImage(OrderableFood food)

```

```

{
    return food switch
    {
        OrderableFood.Burger => burgerSprite,
        _ => null
    };
}

```

```

private Image GetDrinkImage(OrderableDrinks food)
{
    return food switch
    {
        OrderableDrinks.Cola => colaImage,
        OrderableDrinks.Fanta => fantaImage,
        OrderableDrinks.Sprite => spriteImage,
        OrderableDrinks.Pepsi => pepsiImage,
        _ => null
    };
}
}

```

```

using BNG;
using Cysharp.Threading.Tasks;
using UnityEngine;
using DG.Tweening;

```

```

public class Patty : MonoBehaviour
{
    [SerializeField] private Grabbable grabbable;
    [SerializeField] private SpatulaGrabbable spatulaGrabbable;
}

```

```

[SerializeField] private int prepareTime = 10;
[SerializeField] private MeshRenderer gfx;
[SerializeField] private ParticleSystem bakingEffect;
[SerializeField] private Rigidbody pattyRigidbody;
[SerializeField] private BurgerComponent burgerComponent;

[SerializeField] private Color bakedColor = new Color32(121, 141, 82,
255);

```

```

private float _doneness = 0f;
private Color _gfxMaterialInitialColor;

public bool BeingBaked { get; private set; }
public bool Done;
public Grabbable Grabbable => grabbable;
public Rigidbody PattyRigidbody => pattyRigidbody;

```

```

public float Doneness
{
    get
    {
        return _doneness;
    }
    set
    {
        if (value != _doneness)
        {
            OnDonenessUpdated(_doneness, value);
        }
    }
}

```

```
        _doneness = value;
    }
}

private void Start()
{
    spatulaGrabbable.CanGrab = false;
    _gfxMaterialInitialColor = gfx.material.color;
}

public void StartBaking()
{
    BeingBaked = true;

    pattyRigidbody.isKinematic = true;

    bakingEffect.Play();
}

private void OnDonenessUpdated(float prevValue, float newValue)
{
    if (Done)
    {
        return;
    }

    var donenessValue = newValue;

    if (newValue > prepareTime)
    {
```

```
doneenessValue = prepareTime;
```

```
pattyRigidbody.isKinematic = false;
```

```
spatulaGrabbable.CanGrab = true;
```

```
burgerComponent.CanUse = true;
```

```
bakingEffect.Stop();
```

```
Done = true;
```

```
}
```

```
var color = Color.Lerp(_gfxMaterialInitialColor, bakedColor,  
doneenessValue / prepareTime);
```

```
gfx.material.DOColor(color, Time.deltaTime).SetEase(Ease.Linear);
```

```
}
```

```
}
```

```
using DG.Tweening;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using UnityEngine;
```

```
public class PattySlot : MonoBehaviour
```

```
{
```

```
    public event Action<PattySlot> OnPattyPlaced;
```

```
    public event Action<PattySlot> OnPattyRemoved;
```

```
[SerializeField] private CanvasGroup highlightCanvasGroup;
```

```

[SerializeField] private Transform snapTransform;

private List<Patty> _pattiesInTrigger = new List<Patty>();
private bool _hasPatty = false;
private Patty _patty;

public bool HasPatty => _hasPatty;
public Patty Patty => _patty;

private void OnTriggerEnter(Collider other)
{
    if (_hasPatty)
    {
        return;
    }

    var patty = other.GetComponent<Patty>();

    if (patty != null && !_pattiesInTrigger.Contains(patty) &&
!patty.Done)
    {
        _pattiesInTrigger.Add(patty);

        highlightCanvasGroup.DOFade(1f, 0.5f);
    }
}

private void OnTriggerExit(Collider other)
{
    var patty = other.GetComponent<Patty>();

```

```

    if (patty != null && _pattiesInTrigger.Contains(patty))
    {
        _pattiesInTrigger.Remove(patty);

        if (_pattiesInTrigger.Count == 0)
        {
            highlightCanvasGroup.DOFade(0f, 0.5f);
        }

        if (patty == _patty)
        {
            _patty = null;
            _hasPatty = false;
        }
    }
}

private void OnTriggerStay(Collider other)
{
    if (_hasPatty)
        return;

    var patty = other.GetComponent<Patty>();

    if (patty != null && _pattiesInTrigger.Contains(patty))
    {
        if (!patty.Grabbable.BeingHeld)
        {
            SnapPatty(patty);
        }
    }
}

```

```

        }
    }
}

private void Update()
{
    foreach (var patty in _pattiesInTrigger)
    {
        var pattyCollider = patty.GetComponent<Collider>();

        if (!pattyCollider.enabled)
            OnTriggerExit(pattyCollider);
    }
}

private void SnapPatty(Patty patty)
{
    _hasPatty = true;
    _patty = patty;

    _patty.Grabbable.enabled = false;
    _patty.PattyRigidbody.isKinematic = true;
    _patty.transform.SetParent(snapTransform);
    _patty.transform.localPosition = Vector3.zero;
    _patty.transform.localRotation = Quaternion.identity;

    highlightCanvasGroup.DOFade(0f, 0.5f);
}
}

```

```
using UnityEngine;
```

```
public class PreparedFood : MonoBehaviour
```

```
{
```

```
    [SerializeField] private bool isFood = false;
```

```
    [SerializeField] private OrderableFood orderableFoodType;
```

```
    [SerializeField] private bool isDrink = false;
```

```
    [SerializeField] private OrderableDrinks orderableDrinkType;
```

```
    public OrderableFood OrderableFoodType
```

```
    {
```

```
        get => orderableFoodType;
```

```
        set
```

```
        {
```

```
            isFood = true;
```

```
            isDrink = false;
```

```
            orderableFoodType = value;
```

```
        }
```

```
    }
```

```
    public OrderableDrinks OrderableDrinkType
```

```
    {
```

```
        get => orderableDrinkType;
```

```
        set
```

```
        {
```

```
            isFood = false;
```

```
            isDrink = true;
```

```
            orderableDrinkType = value;
```

```
        }
```

```
    }
```

```

        public bool IsFood => isFood;
        public bool IsDrink => isDrink;
    }

    using UnityEngine;

    public class Sliceable : MonoBehaviour
    {
        [SerializeField] private Material crossSectionMaterial;

        public Material CrossSectionMaterial { get => crossSectionMaterial; set
=> crossSectionMaterial = value; }

        public virtual void SetupHull(Sliceable newSliceable)
        {
            newSliceable.gameObject.AddComponent<Rigidbody>();
            newSliceable.gameObject.AddComponent<MeshCollider>().convex =
true;

            newSliceable.CrossSectionMaterial = CrossSectionMaterial;
        }
    }

    using BNG;
    using UnityEngine;

    public class BurgerComponent : MonoBehaviour
    {
        [SerializeField] private BurgerComponentEnum burgerComponentType;
        [SerializeField] private Grabbable grabbable;
    }

```

```
[SerializeField] private SpatulaGrabbable spatulaGrabbable;
```

```
[SerializeField] private bool canUse = true;
```

```
public BurgerComponentEnum BurgerComponentType =>
burgerComponentType;
public Grabbable Grabbable => grabbable;
public SpatulaGrabbable SpatulaGrabbable => spatulaGrabbable;
public bool CanUse { get => canUse; set => canUse = value; }
}
```

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

```
public enum BurgerComponentEnum
{
    BunBot,
    Salad,
    Tomato,
    Patty,
    Cheese,
    BunTop
}
```

```
using BNG;
using UnityEngine;
using UnityEngine.InputSystem;
```

```
public class Spatula : MonoBehaviour
```

```

{
    [SerializeField] private Grabbable grabbable;
    [SerializeField] private SpatulaTrigger spatulaTrigger;
    [SerializeField] private Transform receiverRaycastTransform;
    [SerializeField] private Transform snapTransform;
    [SerializeField] private InputActionReference leftGrabAction;
    [SerializeField] private InputActionReference rightGrabAction;

    private SpatulaGrabbable _grabbedGrabbable;
    private ISpatulaGrabbableReceiver _receiver;
    private bool _grabbing;

    private void Update()
    {
        if (!grabbable.BeingHeld)
        {
            if (_grabbing)
            {
                ReleaseGrabbable();
            }

            return;
        }

        var inputValue = 0f;

        if (grabbable.GetPrimaryGrabber().HandSide ==
ControllerHand.Left)
        {
            inputValue = leftGrabAction.action.ReadValue<float>();

```

```

    }
    else
    {
        inputValue = rightGrabAction.action.ReadValue<float>();
    }

    if (inputValue > .5f)
    {
        if (!_grabbing)
        {
            TryGrabGrabbable();
        }
    }
    else
    {
        if (_grabbing)
        {
            ReleaseGrabbable();
        }
    }

    if (_grabbing)
    {
        if (Physics.Raycast(receiverRaycastTransform.position,
receiverRaycastTransform.forward, out var info, 1f))
        {
            var receiver =
info.collider.GetComponent<ISpatulaGrabbableReceiver>();

            if (receiver != null)

```

```

    {
        if (_receiver != null)
        {
            if (_receiver != receiver)
            {
                _receiver.HideHighlight();
            }
        }
        else
        {
            _receiver = receiver;

            _receiver.ShowHighlight();
        }
    }
}
else
{
    if (_receiver != null)
    {
        _receiver.HideHighlight();

        _receiver = null;
    }
}
}

private void TryGrabGrabbable()
{

```

```

if (spatulaTrigger.NearestGrabbable != null)
{
    _grabbedGrabbable = spatulaTrigger.NearestGrabbable;

    _grabbedGrabbable.transform.SetParent(snapTransform);
    _grabbedGrabbable.transform.localPosition = Vector3.zero;
    _grabbedGrabbable.transform.localRotation = Quaternion.identity;

    _grabbedGrabbable.OnGrabbed();

    _grabbing = true;
}

public void ForceReleaseGrabbable()
{
    ReleaseGrabbable();
}

private void ReleaseGrabbable()
{
    _grabbedGrabbable.transform.SetParent(null);

    _grabbedGrabbable.OnRelease();

    _grabbedGrabbable = null;

    _grabbing = false;
}
}

```

```

using BNG;
using UnityEngine;

public class SpatulaGrabbable : MonoBehaviour
{
    [SerializeField] private Grabbable grabbable;
    [SerializeField] private Rigidbody spatulaRigidbody;
    [SerializeField] private Collider spatulaCollider;

    public bool BeingGrabbed { get; private set; } = false;
    public bool CanGrab { get; set; } = true;

    public Rigidbody Rigidbody => spatulaRigidbody;

    public void OnGrabbed()
    {
        spatulaCollider.enabled = false;
        spatulaRigidbody.isKinematic = true;

        BeingGrabbed = true;
    }

    public void OnRelease()
    {
        spatulaCollider.enabled = true;
        spatulaRigidbody.isKinematic = false;

        BeingGrabbed = false;
    }
}

```

```

    }

    using System.Collections.Generic;
    using UnityEngine;

    public class SpatulaTrigger : MonoBehaviour
    {
        private List<SpatulaGrabbable> _grabbablesInTrigger = new
List<SpatulaGrabbable>();
        private SpatulaGrabbable _nearestGrabbable;

        public IReadOnlyList<SpatulaGrabbable> GrabbablesInTrigger =>
        _grabbablesInTrigger;
        public SpatulaGrabbable NearestGrabbable => _nearestGrabbable;

        private void OnTriggerEnter(Collider other)
        {
            var grabbable = other.GetComponent<SpatulaGrabbable>();

            if (grabbable != null && grabbable.CanGrab &&
!_grabbablesInTrigger.Contains(grabbable))
            {
                _grabbablesInTrigger.Add(grabbable);

                UpdateNearestGrabbable();
            }
        }

        private void OnTriggerExit(Collider other)
        {

```

```

var grabbable = other.GetComponent<SpatulaGrabbable>();

if (grabbable != null && _grabbablesInTrigger.Contains(grabbable))
{
    _grabbablesInTrigger.Remove(grabbable);

    UpdateNearestGrabbable();
}
}

private void UpdateNearestGrabbable()
{
    if (_grabbablesInTrigger.Count == 0)
    {
        _nearestGrabbable = null;
    }
    else if (_grabbablesInTrigger.Count == 1)
    {
        _nearestGrabbable = _grabbablesInTrigger[0];
    }
    else
    {
        var leastDistance = Mathf.Infinity;

        foreach (var grabbableInTrigger in _grabbablesInTrigger)
        {
            var distanceToGrabbable = Vector3.Distance(transform.position,
grabbableInTrigger.transform.position);

            if (distanceToGrabbable < leastDistance)

```

```

        {
            leastDistance = distanceToGrabbable;
            _nearestGrabbable = grabbableInTrigger;
        }
    }
}

```

```

private void Update()
{
    UpdateNearestGrabbable();
}
}

```

```

using BNG;
using UnityEngine;

```

```

public class Stove : MonoBehaviour
{
    [SerializeField] private PattySlot[] pattySlots;
    [SerializeField] private HingeHelper hingeHelper;

```

```

    private float _hingeValue = 0f;

```

```

    private void Start()
    {

```

```

        hingeHelper.onHingeSnapChange.AddListener(OnHingeSnapChange);
    }

```

```
private void OnHingeSnapChange(float value)
{
    _hingeValue = value;
}

private void Update()
{
    if (_hingeValue != 0f)
    {
        foreach (var pattySlot in pattySlots)
        {
            if (pattySlot.HasPatty && !pattySlot.Patty.BeingBaked)
            {
                pattySlot.Patty.StartBaking();
            }
            else if (pattySlot.HasPatty && pattySlot.Patty.BeingBaked)
            {
                pattySlot.Patty.Doneness += Time.deltaTime;
            }
        }
    }
    else
    {
        foreach (var pattySlot in pattySlots)
        {
            if (pattySlot.HasPatty && pattySlot.Patty.BeingBaked)
            {
                pattySlot.Patty.StopBaking();
            }
        }
    }
}
```

```
    }  
  
    foreach (var pattySlot in pattySlots)  
    {  
    }  
}  
}
```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Слайд 1



Рисунок В.2 – Слайд 2

Задачі дослідження



1. Проаналізувати проблемну область VR застосунків;
2. Дослідити існуючі аналоги ігор зі схожою тематикою;
3. З урахуванням аналогів сформуванати концепцію нового додатку;
4. Визначити технічні засоби, які дозволять оптимізувати процес розробки;
5. Спроектувати процес розробки, написання коду та тестування;
6. З урахуванням попередніх пунктів розробити програмну реалізацію нового додатку;
7. Дослідити та протестувати нове рішення.

Рисунок В.3 – Слайд 3

Сфера застосування VR додатків

1. Ігрова індустрія: VR-ігри є одним з найпопулярніших сфер застосування віртуальної реальності.
2. Освіта: VR може бути використана в освітніх цілях для створення інтерактивних навчальних середовищ.
3. Архітектура і дизайн: VR дозволяє архітекторам і дизайнерам створювати віртуальні моделі будівель та інтер'єрів.



Рисунок В.4 – Слайд 4

Порівняння існуючих аналогів

Критерії порівняння	Cooking Simulator VR	The Cooking Game VR
Рівень графіки	Високий	Низький
Кількість рецептів	Висока	Низька
Сеттинг	Сучасна Європа/Америка Постапокаліпсис	Звичайна кухня
Кількість та різноманітність режимів	Висока	Низька
Мультиплеер	Ні	Так
Продуктивність	Середня	Висока
Кількість відгуків	1060 (85% позитивні)	40 (70% позитивні)

Рисунок В.5 – Слайд 5

Порівняння існуючих аналогів



Cooking Simulator VR



The Cooking Game VR

Рисунок В.6 – Слайд 6

Архітектура на базі принципів подійно-орієнтованого програмування

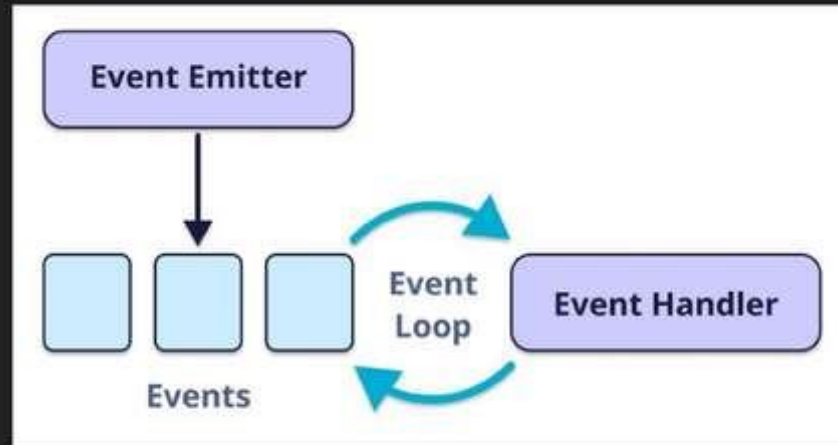


Рисунок В.7 – Слайд 7

Діаграма класів застосунку

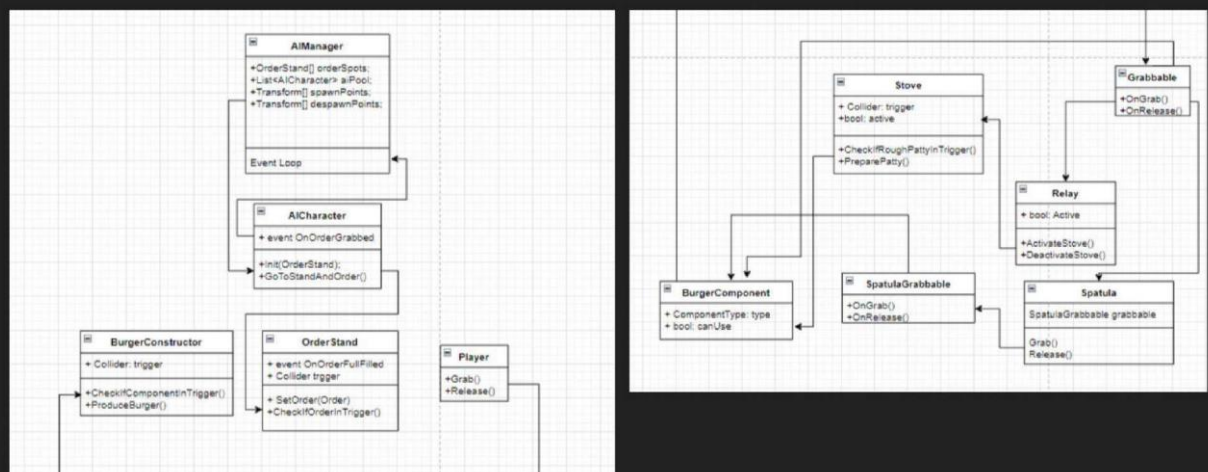


Рисунок В.8 – Слайд 8

Порівняння ігрових двигунів

Критерій	Unity	Unreal Engine
Мобільні додатки	Unity був розроблений з урахуванням мобільних додатків, тому розробка для цих пристроїв є дуже легкою і зручною, як для 2D, так і для 3D проектів. Більшість оптимізацій призначені для невеликих та indie-ігор.	Використання потужних графічних можливостей UE є надмірним для більшості мобільних додатків. А також є більш придатним для AAA-ігор і тих, що орієнтовані на високопотужні пристрої.
Імерсивні додатки	Обидва двигуни підтримують засновані на технології віртуальної реальності, змішаної реальності та доповненої реальності додатки. Проте вибір Unity є ідеальним для прототипування та експериментів.	Це відмінний вибір для захоплюючих додатків, розроблених з великим рівнем деталей. Крім того, завдяки системі Blueprints, він надає нижчий поріг для розробки захоплюючих додатків.
Графіка	Unity надає нілена вражаючих стоків графічних можливостей проте досягнення кращого візуалу часто вимагає багато налаштувань та редагування.	У додаток до вражаючих вбудовано графічних можливостей, додатки UE часто виглядають досконало без додаткових змін завдяки широкому спектру переддоставлених налаштувань.
Спільнота	Asset-Store є величезним і наповненим різноманітними 3D-асетами, а також 2D-асетами, інструментами та візуальними ефектами. На веб-сайті платформи також є велика кількість навчальних матеріалів і активний форум.	Хоча магазин ассетів UE є меншим, він також пропонує достатню різноманітність ассетів, в основному спрямованих на ігрові елементи. Розробники також мають безліч посібників та форумів для підтримки.
Ціна	Найпростіший план (Personal) є безкоштовним для використання, але більш розширені та бізнес-орієнтовані плани коштують \$399 на рік за кожний обліковий запис або більше.	UE є безкоштовним для використання, але включає систему роліти, яка починає діяти з моменту монетизації додатка і приносить компанії (Epic Games) 5% від доходу.

Рисунок В.9 – Слайд 9

Порівняння мов, які Unity в тій чи іншій формі підтримує

Критерій	C#	UnityScript	Boo
Простота вивчення та поріг входження	Середня	Низька	Низька
Типізація	Статична	Динамічна	Статична
Продуктивність	Висока	Низька	Низька
Синтаксис	Схожий на C++, Java	Схожий на JavaScript	Схожий на Python
Стан	Підтримується	Не підтримується починаючи з 2018.x	Не підтримується з виходу Unity 4

Рисунок В.10 – Слайд 10

Поставлені для реалізації функції додатку

1. Загальні можливості:

1. Відображення картинки у VR оточенні.
2. Взаємодія з оточенням за допомогою VR контролерів.

2. Інтерактивні об'єкти та їх функції:

1. Замовник:
 1. Підійти до стійки з замовленнями;
 2. Зробити замовлення;
 3. Забрати блюдо;
 4. Піти геть;

3. Кухня:

1. Переміщати інгредієнти;
2. Виконувати специфічні до інгредієнтів дії;
3. Збирати блюдо з інгредієнтів;

4. Пічка:

1. Причепити котлету до печі;
2. Смажити котлету;

5. Сійка з лимонадом:

1. Причепити келих до сійки;
2. Налити зазначений лимонад;

Рисунок В.11 – Слайд 11

Знімки з гри



Рисунок В.12 – Слайд 12

Знімки з гри



Рисунок В.13 – Слайд 13

Знімки з гри



Рисунок В.14 – Слайд 14

Висновки

В ході виконання дипломної роботи, було реалізовано головну мету - програмну реалізацію VR гри в середовищі UnityEngine. Для цього було виконано наступні завдання:

1. Проаналізовано проблемну область VR застосунків;
2. Досліджено існуючі аналоги ігор зі схожою тематикою;
3. З урахуванням аналогів сформовано концепцію нового додатку;
4. Визначено технічні засоби, які дозволять оптимізувати процес розробки;
5. Спроектовано процес розробки, написання коду та тестування;
6. З урахуванням попередніх пунктів розроблено програмну реалізацію нового додатку;
7. Досліджено та протестовано нове рішення.

В результаті виконання завдання, згідно вищезазначеного плану, було створено додаток, що відповідає вимогам, визначеним в технічному завданні, та виконує всі необхідні функції.

Рисунок В.15 – Слайд 15