МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

“До захисту допущений”
Зав. кафедри комп'ютерних наук та прикладної математики
д.т.н., професор Турбал Ю. В.
«___» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

«Розробка мобільного додатку AR-експозиції для планетаріуму»

Виконав: Мельничук Андрій Олегович
студент навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки
група ІПЗ-41

_____
(підпис)

Керівник: доц., к.т.н. Жуковський Віктор Володимирович

_____
(підпис)

Рівне – 2023

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 58 сторінок, 20 рисунків, 7 джерел

Мета роботи: Розробка спеціалізованого застосунку для планетарію для платформи Andoid з використанням технологій доповненої реальності, що надасть можливість здійснювати віртуальні подорожі по планетарію та допоможе покращити навчальний та пізнавальний процес, і зможе залучити більше відвідувачів планетарію до вивчення космосу через використання інтерактивних функцій застосунку.

Засоби розробки: двигун для розробки ігор та застосунків Unity, мова програмування C#, бібліотека .NET, технологія доповненої реальності AR Foundation, складові компоненти AR на Android та IOS (AR Kit, AR Core), бібліотека зчитування штрих-кодів ZXing.

Актуальність роботи: полягає у використанні технологій доповненої реальності та застосування засобів імерсивних механік, а саме використання засобів AR для навчання та розваг в контексті планетарію та космосу, використання поєднання технологій штрих-кодів та AR для досягнення зручності користування застосунком.

ВСТУП

У зв'язку з швидким розвитком технологій у сучасному світі, їх використання стає необхідним елементом навчального процесу та розвитку культурних і освітніх закладів. Планетарії, як надзвичайно цінні джерела знань про космос та астрономію, також усвідомлюють важливість використання передових технологій для покращення освітнього досвіду своїх відвідувачів.

У рамках цього проєкту-дослідження розробляється та реалізується спеціалізований застосунок, сприятливий для планетаріїв, який використовує передові технології з метою створення захопливого та інтерактивного середовища для навчання та розваг. Цей застосунок є інноваційним інструментом, який поєднує в собі високу якість візуалізації, покращену взаємодію та легку доступність інформації для відвідувачів різних вікових груп та рівнів освіти.

Завдяки використанню передових технологій, таких як розширена реальність з поєднанням із технологією штрих-кодів, планетарії зможуть надати своїм відвідувачам непередбачувані можливості. За допомогою цього застосунку відвідувачі зможуть побачити зірки та планети в доповненій реальності прямо у телефоні, вивчити подробиці про космічні подорожі, експонати, тощо.

Крім того, цей застосунок може бути корисним не лише для навчальних цілей, але й для розважальних заходів. Він дозволить планетаріям організовувати цікаві та захопливі шоу, де відвідувачі матимуть можливість брати участь у віртуальних космічних подорожах, досліджувати найвіддаленіші куточки Всесвіту та відчути себе часткою незвіданого космосу.

Розробка такого спеціалізованого застосунку для планетаріїв відкриває широкі перспективи для інноваційного навчання та розваг у галузі астрономії. Це дає можливість залучати більше людей до вивчення космосу, сприяє поширенню наукових знань та розвитку зацікавленості у науці серед молоді. Застосунок відображає прогресивну філософію планетаріїв, яка постійно

адаптується до вимог сучасності та використовує передові технології для надання незабутнього досвіду відвідувачам.

Мета та завдання проєкту:

1. Провести аналіз вимог і потреб планетаріїв та його відвідувачів
2. Розробити бачення спеціалізованого застосунку, визначити необхідні механіки для мінімального життєздатного продукту
3. Розробити архітектуру застосунку, що забезпечує ефективну взаємодію з відвідувачами та інтеграцію з наявними системами планетаріїв
4. Здійснити вибір технологій реалізації спеціалізованого застосунку
5. Реалізувати функціонал із використанням технологій AR
6. Забезпечити легку доступність та користування застосунком для різних категорій відвідувачів, зокрема дітей, школярів, студентів та дорослих
7. Провести тестування та оцінку ефективності застосунку, здійснити оптимізацію застосунку

Проєкт повинен мати наступний функціонал:

1. Різні меню для взаємодії користувача зі системою
2. Зручне керування
3. Використання технологій AR та штрих-кодів для створення інтерактивності
4. Серіалізація та десеріалізація локальних даних користувача

РОЗДІЛ 1. ОГЛЯД ВИКОРИСТАНИХ В ПРОЄКТІ ТЕХНОЛОГІЙ

1.1. Загальні технології

1.1.1. Unity

Для реалізації застосунку було обрано середовище Unity, що є потужною та широко використовуваною платформою для розробки ігор та інтерактивних застосунків, що надає багато можливостей для створення разючого візуального досвіду.

Unity має численні переваги для створення інтерактивних застосунків. Ось кілька з них:

1. Кросплатформеність: Unity дозволяє розробляти застосунки, які можуть працювати на різних платформах, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність. Це дозволяє досягти широкої аудиторії та забезпечити доступність застосунків на різних пристроях.
2. Графічний потенціал: Unity надає потужні інструменти для створення високоякісної графіки. Він підтримує як 2D, так і 3D графіку, різні ефекти, освітлення, анімацію, частинки та інші графічні можливості. Це дозволяє створювати разючі візуальні ефекти та привабливі ігрові світи.
3. Велика спільнота та екосистема: Unity має велику та активну спільноту розробників, що означає доступ до безлічі ресурсів, плагінів, сторонніх інструментів та допомоги. Це сприяє швидкому розвитку та розширенню функціональності вашого застосунку.
4. Легкість використання: Unity має інтуїтивний інтерфейс та просту навігацію, що полегшує розробку. Він також надає широкий набір готових компонентів, скриптів та шаблонів, які можна використовувати для швидкого створення функціональності без необхідності в повній розробці з нуля.

5. Підтримка мов програмування: Unity підтримує мову програмування C#, що є досить легкою та зручної та дає можливість використовувати бібліотеки .NET у роботі.
6. Фізична симуляція: Unity надає потужні інструменти для моделювання фізичних ефектів, таких як гравітація, зіткнення, рух та динаміка об'єктів. Це дозволяє створювати реалістичну поведінку об'єктів у вашому застосунку.
7. Навчальні ресурси: Unity пропонує багато навчальних ресурсів, включаючи документацію, онлайн-курси, відеоуроки та форуми. Це допомагає новачкам швидко освоїти Unity та розпочати розробку своїх власних інтерактивних застосунків.

Ці переваги роблять Unity потужним інструментом для створення різноманітних інтерактивних застосунків, від ігор до віртуальної реальності та симуляторів.

Принцип роботи з Unity базується на таких ключових елементах:

1. Сцена (Scene): Сцена - це простір, в якому відбувається гра або візуальний проєкт. Вона складається з об'єктів, які можуть бути 3D-моделями, 2D-спрайтами, камерами, світлом тощо. Розробник може розміщувати та налаштовувати об'єкти на сцені за допомогою інтерфейсу Unity.
2. Об'єкти (Game Objects): Об'єкти - це основні елементи сцени в Unity. Вони можуть мати різні компоненти, такі як графічні моделі, колайдери (для обробки зіткнень), скрипти (для логіки та поведінки об'єкта) тощо. Розробник може створювати нові об'єкти або використовувати готові шаблони.
3. Компоненти (Components): Компоненти надають об'єктам певну функціональність та поведінку. Розробник може додавати, видаляти та налаштовувати компоненти для кожного об'єкта. Наприклад, компонент "Рух" може забезпечувати об'єкту можливість переміщення по сцені, а

компонент "Скрипт" дозволяє прив'язати скрипт, написаний мовою програмування C#, до об'єкта.

4. Скрипти (Scripts): Скрипти - це фрагменти коду, які розробник може написати для виконання певної логіки або дій в грі. Вони можуть керувати рухом персонажа, взаємодією з об'єктами, обробкою введення користувача та багато чим іншим. Скрипти використовують мову програмування C# для опису поведінки об'єктів.
5. Фізика (Physics): Unity надає підтримку фізики для симуляції реалістичної поведінки об'єктів у грі. Розробник може встановлювати колайдери для об'єктів, які визначають їх форму та зону взаємодії з іншими об'єктами. За допомогою фізичних матеріалів та сили гравітації розробник може моделювати рух, зіткнення та інші фізичні ефекти.

Unity також має інтегровану консоль для розробки, де розробник може переглядати та налагоджувати код, перевіряти помилки та виконувати інші розробницькі задачі.

1.1.2. C# та .NET

C# (C-Sharp) є мовою програмування, яку можна використовувати в Unity для розробки ігор та інтерактивних застосунків. Вона є потужною та ефективною мовою, яка має багато переваг для розробки у середовищі Unity. Деякі особливості C# в контексті Unity включають:

1. Інтеграція з Unity: C# є мовою програмування, яка надає пряму підтримку для розробки у середовищі Unity. Це означає, що ви можете використовувати C# для створення скриптів, компонентів та логіки, що керує поведінкою об'єктів у проєкті Unity.
2. Легкість використання: C# має простий та зрозумілий синтаксис, що робить його легким у вивченні та використанні. Це особливо корисно для

початківців, які тільки починають вивчати програмування або розробку ігор.

3. Об'єктно-орієнтоване програмування: C# підтримує об'єктно-орієнтоване програмування (ООП), що дозволяє створювати класи, об'єкти, успадкування, поліморфізм та інші концепції ООП. Це сприяє організації коду, полегшує його повторне використання та підтримку.
4. Доступ до функцій Unity API: Unity API (Application Programming Interface) надає набір функцій та можливостей для взаємодії з різними аспектами двигуна Unity. З використанням C# ви можете отримати доступ до цих функцій API та використовувати їх для керування об'єктами, анімації, фізики, звуку, мережі та інших аспектів вашої гри.
5. Багатопоточність: C# має вбудовану підтримку багатопотоковості, що дозволяє вам створювати потоки (threads) та виконувати обчислення паралельно. Це особливо корисно для обробки великої кількості обчислень або виконання операцій у фоновому режимі без блокування основного потоку програми.
6. Розширені можливості розробки: C# надає доступ до багатьох розширених можливостей розробки, таких як LINQ (Language Integrated Query) для зручної роботи з колекціями даних, делегати та події для реалізації реактивної логіки, патерни проєктування та багато іншого.

Своєю чергою .NET, що підтримується Unity (та C#), надає розширені бібліотеки класів (Class Libraries), що містять готовий функціонал для широкого спектру завдань, включаючи роботу з мережевими протоколами, базами даних, графікою, шифруванням, серіалізацією даних та багато іншого. Це дозволяє розробникам ефективно використовувати готові рішення та скоротити час розробки. [2]

1.2. AR технології

1.2.1. AR Foundation

AR Foundation, яка є частиною Unity, надає потужні інструменти для розробки застосунків доповненої реальності, що дозволяє інтегрувати віртуальні об'єкти у реальний світ, використовуючи камеру мобільного пристрою. Це дозволяє користувачам спостерігати космічні об'єкти, такі як планети, зірки та експонати, в їх власному середовищі. Декілька переваг, які надає ARFoundation, включають:

1. Кросплатформеність: ARFoundation підтримує кросплатформенну розробку. Ви можете створювати AR-застосунки, які працюють на пристроях з ОС iOS та Android, використовуючи єдиний код. Це зменшує зусилля, необхідні для розробки та підтримки додатків на різних платформах.
2. Розширена функціональність AR: ARFoundation надає доступ до розширеної функціональності AR, включаючи відстеження положення та орієнтації пристрою, визначення площин, розпізнавання об'єктів, відстеження жестів та багато іншого.
3. Інтеграція з Unity: ARFoundation повністю інтегрований з Unity, що дозволяє легко поєднувати AR-елементи з іншими елементами вашої гри або застосунку. Ви можете використовувати всі можливості Unity, такі як фізика, анімація, звук та інші, для створення багатогранних AR-досвідів.
4. Підтримка різних пристроїв: ARFoundation підтримує різні пристрої AR, включаючи пристрої з підтримкою ARKit (iOS) та ARCore (Android). Він дозволяє створювати AR-застосунки, які використовують усі переваги цих технологій та пристроїв.
5. Простота використання: ARFoundation надає простий та зрозумілий API для взаємодії з AR-функціоналом. А також дозволяє швидко почати розробку AR-застосунків, навіть без попереднього досвіду роботи з AR.

6. Підтримка розширень: ARFoundation може бути легко розширений за допомогою додаткових пакетів і розширень. Адже він також надає можливість використовувати різні пакети, такі як ARKit XR Plugin та ARCore XR Plugin, для отримання додаткової функціональності та підтримки специфічних можливостей пристроїв

Основні функціональні можливості ARFoundation включають:

1. Відстеження положення та орієнтації: ARFoundation дозволяє відстежувати положення та орієнтацію пристрою в реальному часі. Це дає можливість розміщувати віртуальні об'єкти у світі AR, взаємодіяти з ними та відтворювати реалістичні AR-сцени.
2. Визначення площин: ARFoundation дозволяє виявляти плоскості у реальному світі, такі як столи, підлоги або стіни. Це дозволяє розміщувати віртуальні об'єкти на реальних поверхнях та створювати більш імерсивний досвід використання AR.
3. Розпізнавання об'єктів: ARFoundation надає можливість розпізнавати конкретні об'єкти або маркери в реальному світі. Це дозволяє створювати взаємодію з визначеними об'єктами та реалізовувати розширені AR-сценарії.
4. Відстеження руху: ARFoundation дозволяє відстежувати рухи користувача та його жестів. Це відкриває можливості для створення інтерактивних AR-додатків, які реагують на рухи користувача.
5. Розширений звук та відео: ARFoundation підтримує інтеграцію з аудіо- та відеопотоками у реальному часі. Це дозволяє створювати AR-додатки з розширеними можливостями звуку та відео, такі як звукові ефекти або відеопроєкції на фізичні об'єкти.

AR Foundation можна використовувати разом ARCore та ARKit, що надає можливість використовувати більш оптимізований функціонал спільних систем та отримати кращі результати виконання базових операцій, що стосуються AR [4].

1.2.2. ARCore

ARCore - це спеціалізований фреймворк, розроблений компанією Google, для розробки AR-додатків на платформі Android. ARCore забезпечує функціональність, таку як відстеження руху пристрою, визначення плоскостей, розпізнавання об'єктів та обробка освітлення. Він працює на рівні операційної системи та надає низькорівневий доступ до можливостей AR пристрою.

Робота ARCore разом з ARFoundation полягає в тому, що ARFoundation використовує ARCore як один з пакетів розширення для підтримки функціональності AR на пристроях Android. ARFoundation взаємодіє з ARCore, використовуючи його можливості для відстеження руху пристрою, визначення площин, розпізнавання об'єктів та іншого функціоналу. Це дозволяє розробникам Unity зосередитися на розробці AR-додатків на високорівневому рівні, використовуючи зручний API ARFoundation, тоді як ARCore відповідає за низькорівневу взаємодію з пристроєм та операційною системою Android.

1.2.3. ARKit

ARKit - це фреймворк розробки доповненої реальності (AR), розроблений компанією Apple для пристроїв iOS. ARKit дозволяє розробникам створювати інтерактивні AR-додатки, які можуть взаємодіяти з реальним світом.

ARFoundation та ARKit можуть працювати разом для розробки AR-додатків в середовищі Unity для пристроїв iOS. ARFoundation надає уніфікований інтерфейс, що поєднує різні SDK AR, включаючи ARKit, в один API. Це спрощує розробку AR-додатків, незалежно від конкретної платформи.

1.2.4. ZXing

ZXing (Zebra Crossing) - це відкрите програмне забезпечення для розпізнавання та генерації штрих-кодів, що у розроблюваному проєкті надає

можливість сканувати QR-коди для подальшого відображення необхідних 3D-моеделй в системі AR. Вона надає набір інструментів і бібліотек для роботи з різними типами штрих-кодів, включаючи QR-коди. ZXing існує для використання разом з .NET і підтримується активною спільнотою розробників.

Основні можливості бібліотеки ZXing включають:

1. Зчитування штрих-кодів: ZXing надає функції для зчитування штрих-кодів з зображень або потоків даних. Вона підтримує розпізнавання різних типів штрих-кодів, таких як QR-коди, штрих-коди EAN, UPC, Code 39, Code 128 та інші.
2. Генерація штрих-кодів: За допомогою ZXing можна створювати зображення штрих-кодів різних типів. Вона дозволяє налаштовувати параметри штрих-кодів, такі як розмір, кольори, текст та інші атрибути.
3. Підтримка різних платформ: ZXing підтримується на різних платформах, включаючи Java, Android, C#, Objective-C та інші. Це робить її універсальним інструментом для розпізнавання та генерації штрих-кодів у різних середовищах розробки.
4. Легка інтеграція: ZXing надає простий інтерфейс програмування додатків (API), що спрощує її використання та інтеграцію в проєкт. ЇЇ можна використовувати її як вбудовану бібліотеку або додати її як залежність до проєкту.

РОЗДІЛ 2. АНАЛІТИКА ДОСЛІДЖЕННЯ

2.1. Аналіз ринку спеціалізованих застосунків

На сьогодні, планетарії відіграють важливу роль у навчанні та розвитку галузей, пов'язаних з астрономією, космосом та наукою загалом. З ростом інтересу до космосу та наукових досліджень, збільшується популярність планетаріїв серед широкої аудиторії.

Спеціалізовані застосунки для планетаріїв відіграють важливу роль у забезпеченні інтерактивних та інноваційних досвідів для відвідувачів. Вони дозволяють візуалізувати небесні об'єкти, розділяти знання та сприяють участі глядачів у захопливих подорожах крізь космос.

Оскільки планетарії знаходяться по всьому світу, розмір ринку є досить значним (**рис. 2.1**). Великі міста та області зі значною кількістю населення можуть мати декілька планетаріїв або навіть мережу планетаріїв, що створює потенційний попит на спеціалізовані застосунки.

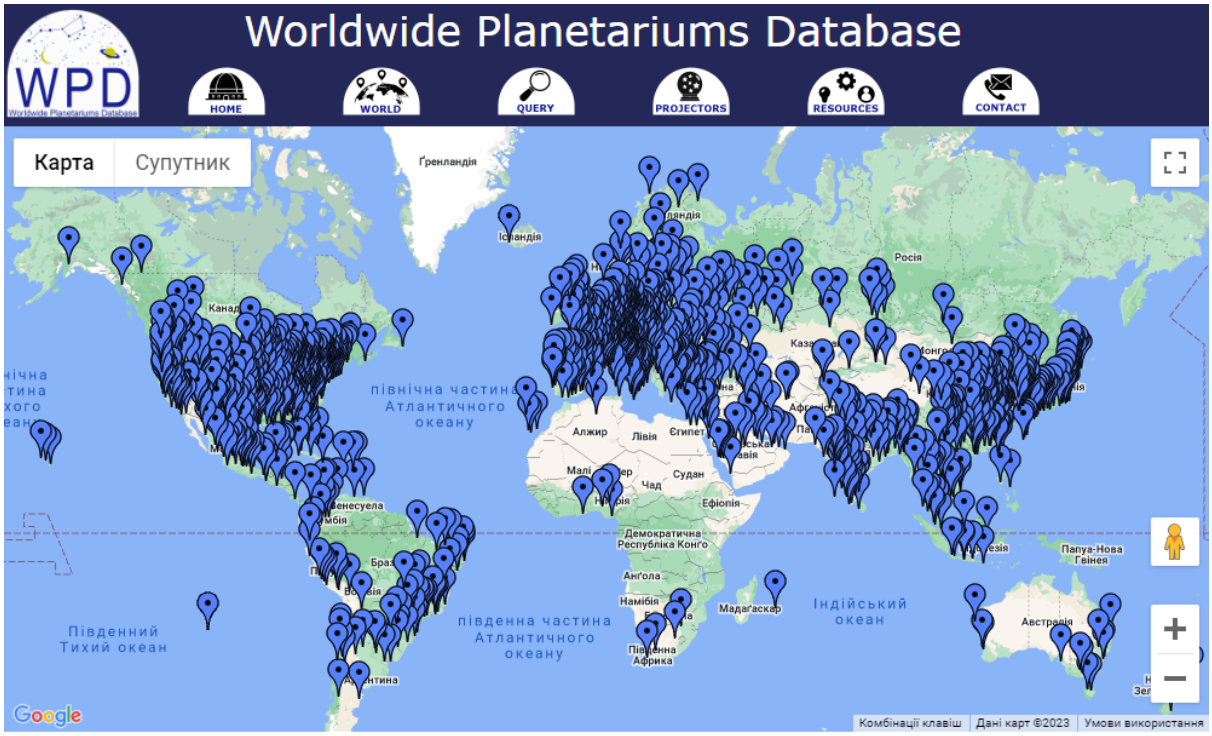


Рис. 2.1. Візуалізація розміщення планетаріїв у світі [7]

Крім того, з появою технологій віртуальної реальності та розширеної реальності, ринок спеціалізованих застосунків розширюється, надаючи нові можливості для інтерактивного та захопливого досвіду в планетарії.

На ринку спеціалізованих застосунків для планетаріїв спостерігаються деякі цікаві тренди, які варто враховувати при розробці нашого проєкту. Ось деякі з них:

1. Зростання популярності віртуальної та розширеної реальності: Використання технологій віртуальної та розширеної реальності набуває все більшої популярності в планетаріях. Це дозволяє створювати імерсивні та захопливі досвіди, де глядачі можуть взаємодіяти з космічними об'єктами та отримувати нові знання.
2. Розширення мультимедійного контенту: Застосунки для планетаріїв все більше стають мультимедійними, поєднуючи в собі не тільки візуалізацію, але й звук, музику, голосові коментарі та інші елементи. Це дозволяє створювати більш повноцінні та емоційні виставки в планетаріях.
3. Зростання інтерактивності: Користувачі все більше очікують можливості активної участі в інтерактивності під час виставок. Тому спеціалізовані застосунки все частіше включають можливості взаємодії, наприклад, використання дотикових екранів, жестів, контролерів або голосових команд.
4. Персоналізований зміст: Індивідуалізація та персоналізація також стають важливими аспектами. Користувачі хочуть мати можливість вибирати теми, досліджувати певні об'єкти або створювати власні маршрути. Тому спеціалізовані застосунки надають можливість налаштування та індивідуального підходу до кожного користувача.

На ринку спеціалізованих застосунків для планетаріїв існує кілька конкурентів, які також надають схожі продукти та послуги. Детальний аналіз конкурентів допоможе краще розуміти їх сильні та слабкі сторони й визначити унікальну пропозицію продукту. Короткий опис деяких конкурентів:

1. "Planetarium Software Inc.": Ця компанія має довгий та успішний досвід у розробці та постачанні спеціалізованого програмного забезпечення для планетаріїв. Вони пропонують високоякісний контент, візуалізацію небесних об'єктів та інтерактивні можливості для глядачів. Однак їхні продукти можуть бути відносно дорогими та менше персоналізованими.
2. "StellarTech Solutions": Ця компанія спеціалізується на інтерактивних технологіях для планетаріїв. Вони пропонують розширену реальність, віртуальну реальність та інші інноваційні можливості. Їхні застосунки дозволяють глядачам активно взаємодіяти з космічними об'єктами та отримувати персоналізований досвід. Однак їхні рішення можуть вимагати високої технічної підготовки та спеціалізованого обладнання.
3. "GalaxySim": Ця компанія спеціалізується на моделюванні та симуляції космосу для планетаріїв. Вони надають детальні та реалістичні моделі планет, зірок, галактик та інших небесних об'єктів. Їхні застосунки дозволяють відвідувачам досліджувати космос та розуміти наукові концепції. Однак їхня бібліотека може бути обмеженою в порівнянні з іншими конкурентами.

Позитивною стороною проєкту може бути унікальний підхід до персоналізації досвіду, інноваційні функції та широкий спектр контенту.

Дослідження спрямоване на задоволення потреб спеціалізованих застосунків для планетаріїв, тому важливо визначити цільову аудиторію, до якої буде спрямована наша продукція. Основні складові цільової аудиторії включають:

1. Планетарії та астрономічні центри: Застосунок може бути корисними для планетаріїв та астрономічних центрів різних розмірів та рівнів технічної оснащеності. Вони можуть використовувати наші продукти для покращення показів, інтерактивних вистав та освітніх програм.
2. Освітні заклади: Застосунки може бути корисними для освітніх закладів, таких як школи та університети. Вони можуть використовуватися для

навчання студентів астрономії, фізики та інших наукових дисциплін, розширюючи їх розуміння космосу та небесних явищ.

3. Глядачі та любителі астрономії: Застосунок може зацікавити широке коло глядачів та любителів астрономії. Вони можуть надати можливість насолоджуватися реалістичними візуалізаціями космічних об'єктів, досліджувати планети та зірки, а також поглиблювати знання про Всесвіт.

Цільова аудиторія застосунку складається з професіоналів, дослідників та ентузіастів, які цікавляться космосом та астрономією. А головна мета - розробити продукт, який задовольнить їхні потреби, забезпечуючи якісний та захопливий досвід.

На основі аналізу вище, можна викреслити основну проблематику, яку застосунок повинен вирішити, а саме покращеній імерсивності, інтерактивності та освітньої цінності виставок планетаріїв. Деякі з ключових аспектів, які може вирішити застосунок, включають:

1. Візуалізація експонатів: Застосунок має за мету створити, який надає захопливі візуальні ефекти космічних експонатів. Це дозволить глядачам максимально відчути красу та велич науки про космос.
2. Інтерактивність: Застосунок буде зосереджений на активній взаємодії з глядачами. А отже необхідно реалізувати інтерактивні функції, які дозволять користувачам самостійно вибирати об'єкти для дослідження, змінювати перспективу та отримувати додаткові інформаційні шари.
3. Освітня цінність: Застосунок має сприяти освіті та розширенню знань про космос. Тому потрібно включити навчальний контент, наукові пояснення та інформацію про космічні явища, щоб надати користувачам можливість вивчати та розуміти небесну сферу.
4. Персоналізація: Застосунок, який може буде мати інтерактивну колекцію зі зібраними експонатами та багато функцій, які за потреби користувач зможе використовувати.

Вирішення цих проблематик допоможе покращити якість виставок, забезпечити захопливий та навчальний досвід для глядачів та підвищити інтерес до астрономії та космосу.

2.2. Постановка основних вимог на основі аналізу ринку

Ідеєю розробки є створення мобільного застосунку для платформ Android та iOS, який буде пов'язаний з планетарієм та використовувати функціональні можливості доповненої реальності (AR). Застосунок буде поєднувати в собі візуалізацію космічних експонатів, які присутні в планетарії чи не присутні за різних причин, інтерактивні можливості та освітні компоненти для створення захопливого досвіду для глядачів. За допомогою камери мобільного пристрою та технології AR, користувачі зможуть спостерігати віртуальні планети, зірки та інші космічні об'єкти, розміщені у реальному оточенні. Освітні матеріали та інформаційні шари допоможуть користувачам отримувати додаткові знання про всесвіт та космос. Застосунок також надасть можливість зберігати вже попередньо досліджені експонати в планетарії у власній колекції, а також використовувати його як підтримку під час відвідування планетарію, де користувачі зможуть додатково досліджувати та вивчати космос з використанням свого мобільного пристрою.

Виділимо основні частини, що будуть присутні в застосунку:

- Модель даних. Зручне та функціональне середовище для зберігання, редагування та вивантаження локальних даних користувача.
- Тур-360. 360-градусний тур у якому користувач зможе прогулятись по планетарію.
- Перегляд експонатів у AR. Система зчитування QR-кодів для подальшого розміщення відповідного експонату в доповненій реальності.
- Колекція. Зберігання досліджених експонатів за допомогою QR-коду у власну локальну колекцію користувача із подальшою можливістю

перегляду експоната будь-де і будь-коли за допомогою вільного режиму доповненої реальності.

- Аудіогід. Можливість прослуховування навчальної інформації про експонати двома мовами (українською та англійською).
- Історія планетарію. Можливість прочитати історії планетарію та інформації про деякі його експонати двома мовами (українською та англійською).
- Зміна мови. Функція перемикання мови з української на англійську та навпаки.
- Інтерфейс. Функціональний та зручний інтерфейс для усіх секцій застосунку.

2.3. Модель роботи системи AR

AR, або доповнена реальність, це технологія, що дозволяє об'єднувати віртуальні об'єкти з реальним оточенням, створюючи враження їх взаємодії. Основна ідея AR полягає в тому, щоб додати додаткові шари інформації, графіки або 3D-елементи до реального світу, який бачить користувач.

Процес роботи AR може включати наступні етапи:

1. Відстеження: Спочатку система AR відстежує рух пристрою або камери, що дозволяє визначати їх положення і орієнтацію в просторі. Це може бути досягнуто за допомогою датчиків, які вбудовані у пристрої (такі як акселерометр, гіроскоп або компас), або за допомогою відеознімання та алгоритмів комп'ютерного зору.
2. Розпізнавання об'єктів: Після відстеження пристрій або камера можуть розпізнати реальні об'єкти або плоскості у навколишньому середовищі. Це дозволяє системі знати, де розмістити віртуальні об'єкти або як їх інтегрувати з оточенням.

3. Візуалізація: Після відстеження та розпізнавання об'єктів система AR відображає віртуальні об'єкти на реальних поверхнях або в реальному просторі. Це може включати відображення 3D-моделей, графіки, тексту, анімації та інших елементів, які взаємодіють з реальним світом.
4. Взаємодія: Користувач може взаємодіяти з віртуальними об'єктами, використовуючи різні способи вводу, такі як торкання екрана, мультидотик жестів, голосові команди або навіть жести руки. Система AR відстежує ці взаємодії та забезпечує відповідну реакцію віртуальних об'єктів.
5. Однією з ключових технологій, яка допомагає реалізувати AR, є комп'ютерне зорове сприйняття (computer vision), що дозволяє системі розпізнавати та інтерпретувати зображення з камери для розуміння довкілля. Крім того, AR може використовувати інші технології, такі як сенсори руху, геолокація, датчики освітлення та інші, для покращення точності та взаємодії з довкіллям.

2.4. Модель роботи ARFoundation в Unity

Інтеграція AR разом з Unity за допомогою ARFoundation та ARKit (IOS) і ARCore (Android) має наступні кроки:

1. Ініціалізація: У розробці AR-додатків з використанням ARFoundation спочатку потрібно ініціалізувати AR-сесію для конкретної платформи. Це включає перевірку наявності підтримки AR на пристрої, налаштування камери та інших необхідних налаштувань.
2. Відстеження руху: ARCore і ARKit відповідають за відстеження руху пристрою або камери. Вони використовують вбудовані датчики та алгоритми, щоб визначити положення та орієнтацію в просторі. Інформація про рух передається до ARFoundation для подальшої обробки.

3. Розпізнавання об'єктів: ARCore та ARKit також забезпечують функціонал розпізнавання об'єктів та площин в навколишньому середовищі. Вони використовують алгоритми комп'ютерного зору та машинного навчання для виявлення поверхонь, маркерів або інших вказівників у реальному часі.
4. Візуалізація: Після отримання інформації про рух та розпізнавання об'єктів ARFoundation використовує їх для відображення віртуальних об'єктів в доповненій реальності. Це може бути досягнуто шляхом створення 3D-моделей, накладання текстур, відображення графіки та інших візуальних ефектів на реальному оточенні.
5. Взаємодія: ARFoundation дозволяє взаємодіяти з віртуальними об'єктами, використовуючи введення, таке як торкання, жести, голосові команди або мовлення. Він надає інтерфейс для обробки вхідних подій та відповіді на них, забезпечуючи інтерактивність AR-додатків.

Усі ці етапи працюють разом, дозволяючи створювати доповнену реальність на різних платформах. ARFoundation служить мостиком між ARCore і ARKit, що спрощує розробку кросплатформенних AR-додатків і забезпечує єдиний інтерфейс програмування додатків (API) для взаємодії з функціями AR.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Структура програми

Структура програми застосунку була розроблена з використанням патерну MVC (Model-View-Controller), що дозволяє ефективно організувати роботу та управління різними компонентами застосунку.

Проєктування відбувалося через 1 (один) головний скрипт, який виступає в ролі менеджера. В Unity такий скрипт заведено називати GameManager. Цей скрипт відповідає за керування потоком даних та подій, а також взаємодію з іншими компонентами застосунку (секціями).

Кожна секція застосунку окремо реалізована за допомогою патерну MVC (Model-View-Controller) (**рис. 3.1**). Це дозволяє створити чітку та розсортовану структуру для кожної секції, полегшуючи розробку, тестування та розширення функціонала.

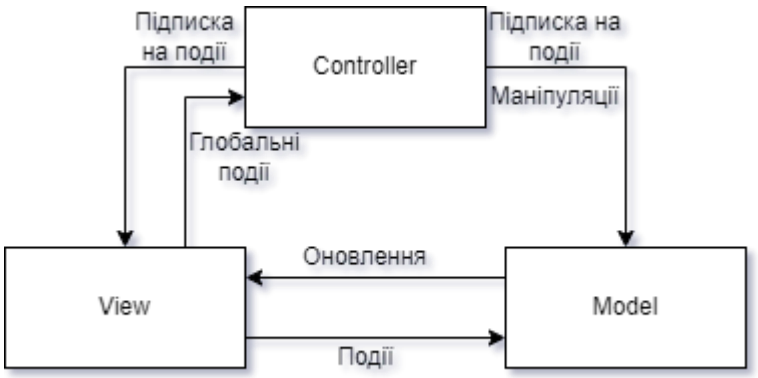


Рис. 3.1. Діаграма модифікованого патерну MVC

Для кожної секції, наприклад, візуалізації експонатів або туру-360, створено відповідні компоненти моделі (Model), представлення (View) та контролера (Controller).

Модель (Model) відповідає за управління даними, логікою та бізнес-правилами конкретної секції. Вона зберігає та оброблює необхідні дані, забезпечуючи їх доступність для інших компонентів.

Представлення (View) відображає дані користувачеві та відповідає за візуальну презентацію інтерфейсу для взаємодії з користувачем. Воно відображає дані, отримані від моделі, та передає введення користувача до контролера.

Контролер (Controller) обробляє вхідні події та взаємодіє з моделлю та представленням. Він приймає введення від користувача, виконує відповідні дії та змінює стан моделі та представлення, щоб оновити відображення даних та реагувати на події.

Таким чином, кожна секція застосунку має власну незалежну структуру MVC, що дозволяє ефективно керувати та розвивати кожну секцію окремо, забезпечуючи модульність та гнучкість розробки.

Також в додаток до головного скрипта необхідно реалізувати скрипт, що буде відповідати за локалізацію, та скрипт, який керує моделлю даних (**рис. 3.2**).

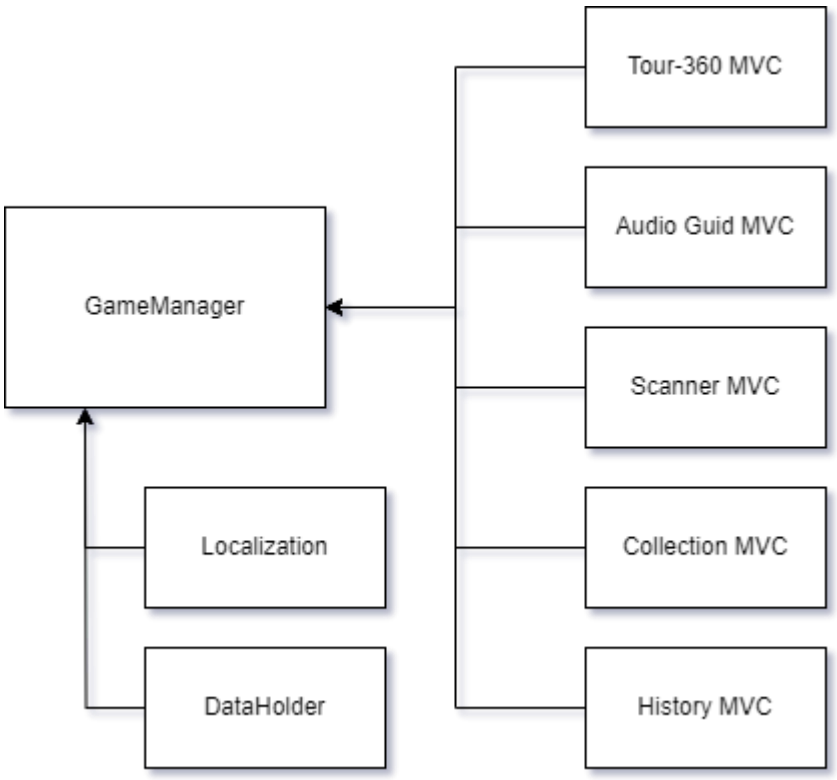


Рис. 3.2. Діаграма головної структури застосунку

3.2. Серіалізація та десеріалізація даних

У застосунку є 3 головні секції, що потребують різного роду даних, таких як:

1. Моделі експонатів
2. Аудіо файли відповідно до обраної мови
3. Текстури історії планетарію та інформації про експонати відповідно обраної мови

Для кожної із таких секцій було вирішено створити окремий скрипт, який керує цими даними:

1. Collection Models Data. Для зберігання та обробки ключів розміщення моделей експонатів.
2. Audio Guid Data. Для зберігання та обробки ключів розміщення аудіо даних.
3. Planetharium History Data. Для зберігання та обробки ключів розміщення текстур історії планетарію.

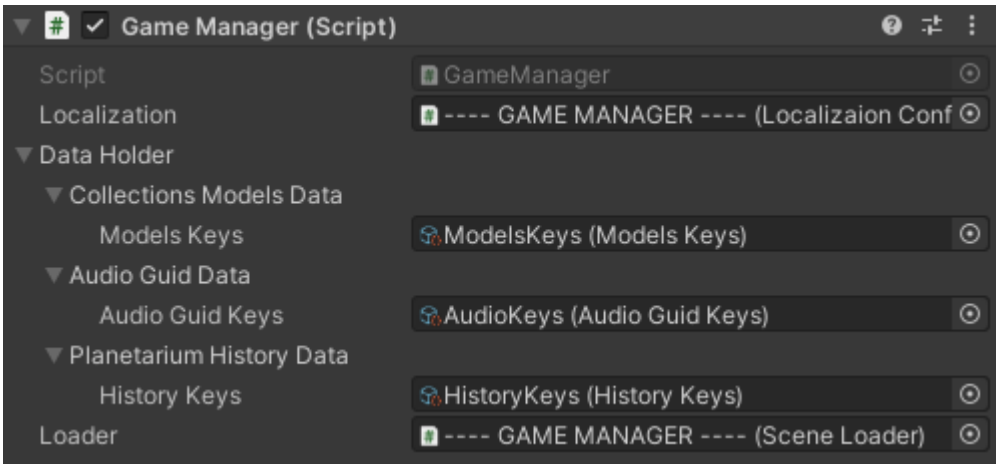


Рис. 3.3. Вигляд компонента Game Manager

Самі ж ключі можна зберігати у класах, що імплементують Scriptable Object (**рис. 3.4**). Scriptable Object (сценарійний об'єкт) є одним з потужних інструментів у Unity для зберігання та керування даними, які можуть бути використані в різних частинах застосунку. Він дозволяє створювати спеціальні об'єкти, які можуть зберігати дані, функції та налаштування, а також легко комунікувати з

іншими скриптами в проєкті. Основна перевага Scriptable Object полягає у тому, що можна створювати та настроювати екземпляри цих об'єктів без необхідності мати екземпляр якось об’єкту з таким скриптом на сцені. Також у майбутньому з ним можна буде легко працювати дизайнеру чи іншій людині, яка не займається програмуванням.

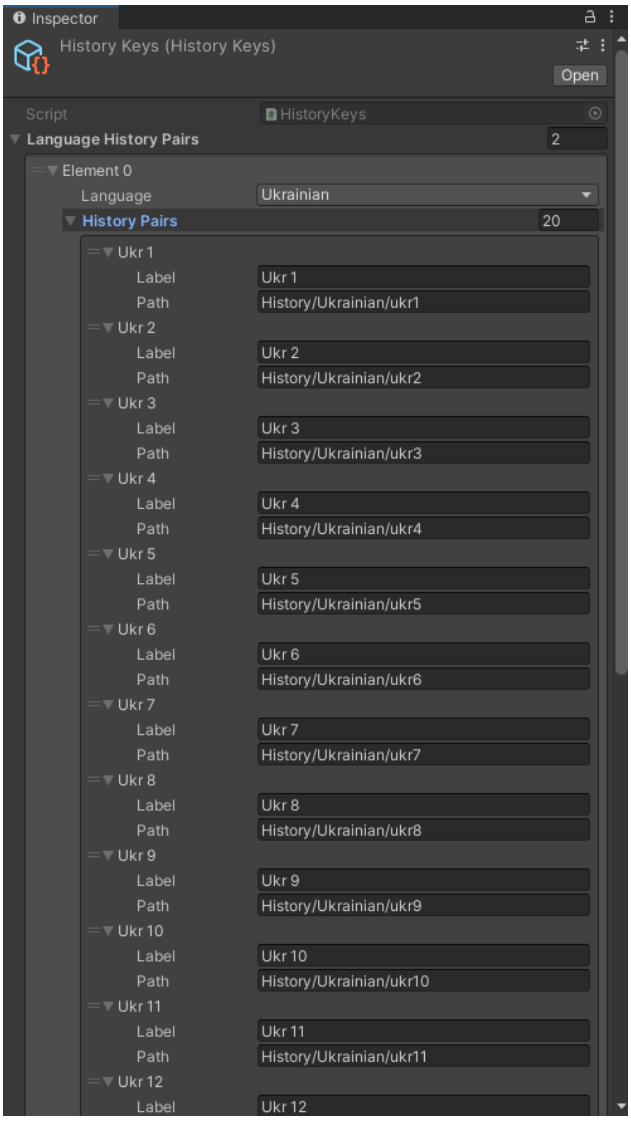


Рис. 3.4. Сценарій об’єкт з визначеними ключами

Також необхідно реалізувати збереження та вивантаження колекції користувача, тобто серіалізувати та десеріалізувати її. Це можна зробити за допомогою перетворення даних про колекцію користувача у стрічку JSON та подальшим збереженням у файл. JSON (JavaScript Object Notation) є популярним форматом обміну даними, який широко використовується для збереження та

передачі структурованих даних. В Unity можна використовувати JSON серіалізацію для збереження та завантаження даних в різних форматах, таких як конфігураційні файли, стан гри, налаштування користувача та багато іншого. Для цього можна зберігати колекцію користувача в об’єкті окремого класу, що буде містити список з назвами експонатів, які користувач уже дослідив. Кінцева структура моделі даних має наступний вигляд:

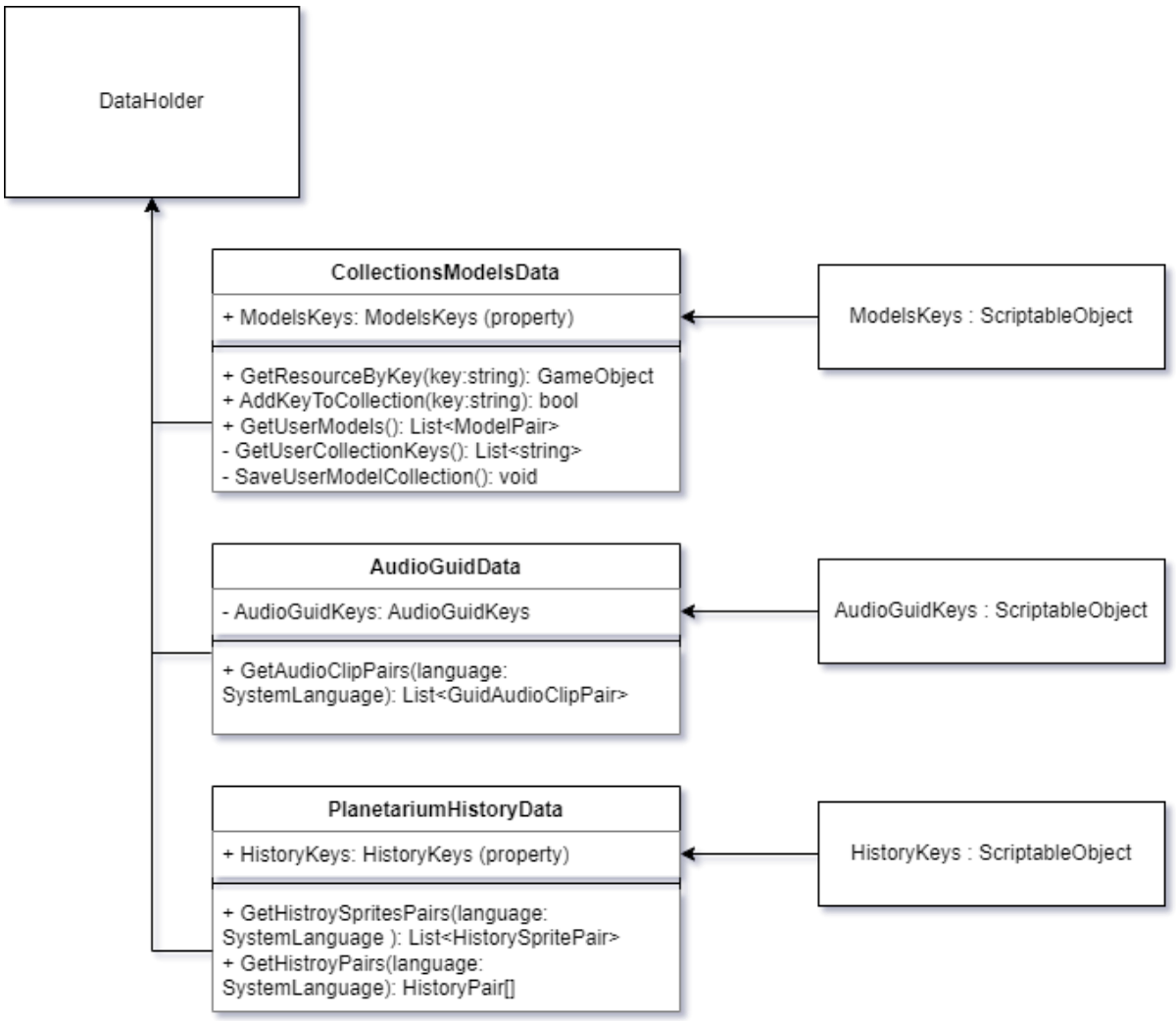


Рис. 3.5. Діаграма класів моделі даних

3.3. Тур-360

Реалізувати тур-360 по планетарію можна з використанням сфери з інвертованими полігонами та гіроскопу телефону. При такій реалізації,

користувач матиме змогу досліджувати планетарій, повертаючи телефон у просторі або взаємодіючи з ним за допомогою переміщення пальцем.

Одним з ключових елементів реалізації є використання сфери з інвертованими полігонами як основної моделі планетарію (**рис. 3.6**). Це дозволяє створити разючу 360-градусну панораму, яка оточує користувача з усіх боків. Інвертовані полігони допомагають відтворити реалістичну візуальну картину з 360-градусної камери без видимих швів або розривів у текстурі. Для цього були створені десятки відповідних зображень планетарію за допомогою 360-градусної камери. Оскільки використана у роботі камера як вихідне зображення повертала фото у форматі стереографічної проєкції для використання їх в середовищі Unity та для вдалої проєкції на сферу необхідно конвертувати їх у формат сферичної рівно прямокутної проєкції, що було зроблено за допомогою онлайн сервісів.

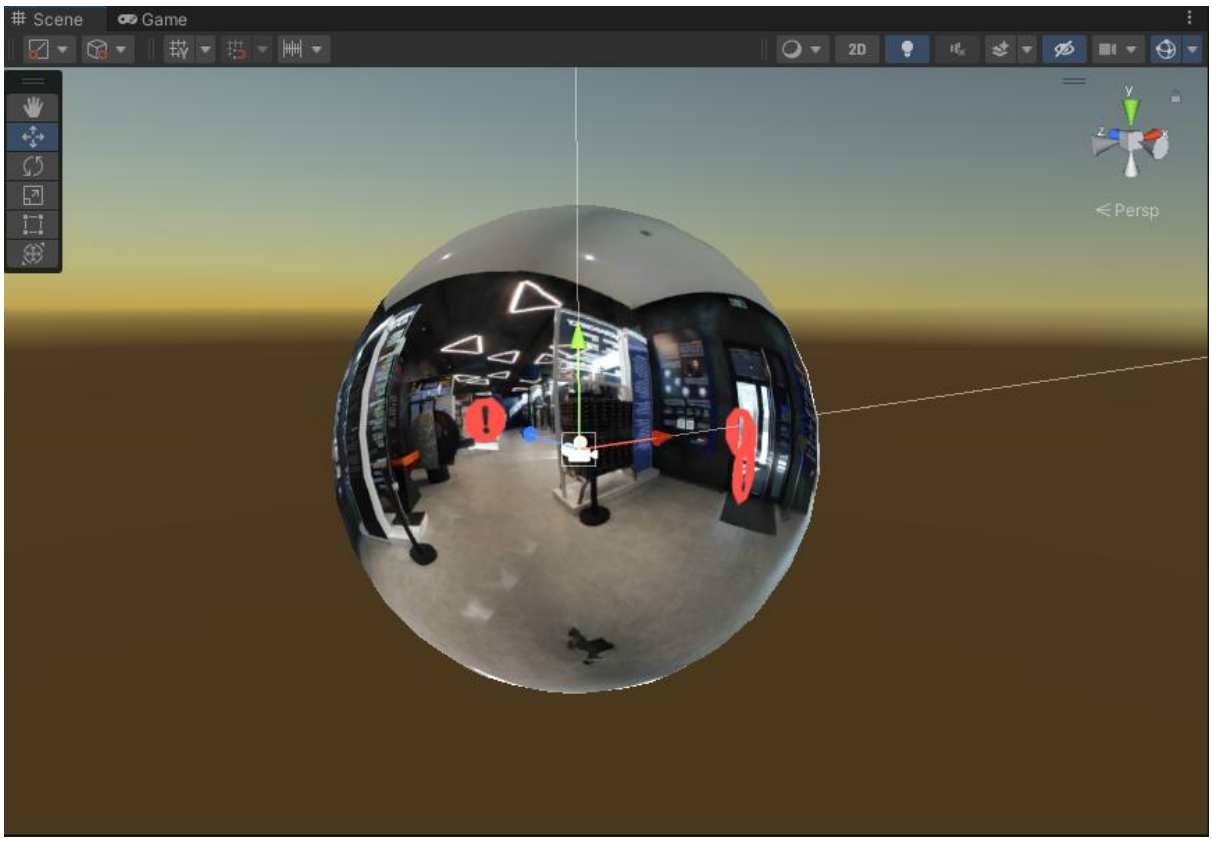


Рис. 3.6. Вигляд інвертованої сфери з текстурою в Unity

Для інтерактивності та навігації в турі-360 використовується гіроскоп телефону. За допомогою гіроскопу можна відстежувати рухи користувача телефону і відображати відповідні зміни на сфері планетарію. Коли користувач

повертає або нахиляє телефон, зображення на сфері буде відповідно рухатися, створюючи ефект погляду в різних напрямках.

Для отримання даних гіроскопу в Unity використовується API (Application Programming Interface) платформи Unity, яке надає доступ до сенсорів пристрою, включаючи гіроскоп.

Основним класом, який використовується для отримання даних гіроскопу, є клас Input. Цей клас містить статичні методи, що дозволяють отримувати дані з різних сенсорів, включаючи гіроскоп.

Паралельно з режимом гіроскопа, застосунок може також пропонувати режим переміщення з використанням пальця. В цьому режимі користувач може перетягувати пальцем по екрану, щоб змінювати напрямок перегляду на сфері. Це дозволяє більшу вільність і контроль над дослідженням планетарію.

Управління та комутація між режимами може бути реалізована через інтуїтивний інтерфейс, наприклад кнопку, що дозволяє користувачам перемикатися між режимами гіроскопа та переміщення пальцем.

Така реалізація туру-360 з використанням сфери з інвертованими полігонами та гіроскопу телефону надає захопливий спосіб дослідження планетарію, забезпечуючи імерсивний досвід для користувачів і дозволяючи їм вільно переглядати та взаємодіяти з віртуальним середовищем.

3.4. Інтерактивний перегляд експонатів

Перегляд експонатів у доповненій реальності (AR) з використанням ARFoundation після зчитування QR-коду за допомогою бібліотеки ZXing може бути захопливим досвідом для користувачів.

Принцип роботи з бібліотекою ZXing досить простий і включає наступні кроки:

1. Підключення бібліотеки: Спочатку потрібно приєднати бібліотеку ZXing до проєкту. Це можна зробити шляхом додавання залежності до проєкту

або завантаження dll-файлу, якщо як в нашому випадку використовується Unity.

2. Ініціалізація декодера: Після підключення необхідно створити екземпляр декодера штрих-кодів, який буде використовуватися для розпізнавання штрих-кодів. Для цього можна використовувати клас BarcodeReader або його похідні класи, залежно від типу штрих-коду, який планується розпізнавати. У випадку із QR-кодом можна використати стандартний клас.
3. Зчитування штрих-коду: У декодер можна передати зображення або потік даних, що містить штрих-код, і отримати результат розпізнавання (**рис. 3.7**). Як зображення можна використати вихідну текстуру камери для візуалізації, що має невеликий розмір і оптимізована для обрахунків. Результатом такого зчитування буде текстова інформація, яка міститься в штрих-коді, що надасть можливість у подальших кроках згенерувати необхідну модель експоната.

Рис. 3.7. Приклад вигляду QR-кода

Наступним кроком буде використання методу, який був отриманий при ініціалізації сканера, отримання моделі за назвою, що була зчитана з QR-коду.

Після цих дій можна повідомляти AR-системі про намір розмістити модель на сцені.

Для розміщення моделі у потрібному місці можна використати вбудовану в ARFoundation систему променів. Для цього можна обрахувати точку звідки буде кидатись AR-промінь за центром QR-кодом, що був обрахований бібліотекою ZXing.

Для використання променів можна використати метод Raycast в екземплярі класу ARRaycastManager, що окрім параметрів початкової точки та вихідного параметру для збереження даних має параметр TrackableType, що використовується для визначення типу об'єктів, з якими потрібно виконати променеве кастування. TrackableType - це перерахування, яке містить різні типи об'єктів, що можуть бути виявлені в AR-сцені [6]. Деякі з доступних значень TrackableType включають:

1. FeaturePoint: Цей тип представляє виявлені особливі точки в AR-сцені, такі як кути або краї об'єктів. Використовується, якщо достатньо взаємодіяти лише з цими особливими точками.
2. PlaneWithinPolygon: Цей тип представляє виявлені площини в межах області замкненого полігону. Використовується, коли необхідно взаємодіяти з площинами, що знаходяться всередині визначеного полігону.
3. PlaneEstimated: Цей тип представляє оцінені площини в AR-сцені. Використовується, коли необхідно взаємодіяти з оціненими площинами, такими як підлога або стіни.
4. Mesh: Цей тип представляє виявлені меші в AR-сцені. Використовується, коли потрібно взаємодіяти з виявленими мешами, такими як об'єкти або структури в реальному світі.

Для вищезазначених цілей достатньо використати тип виявлення FeaturePoint, що буде найбільш оптимізованим варіантом для цілей задачі (**рис. 3.8**).

Також для використання вищезазначених інструментів ARFoundation необхідно створити об’єкт, що буде містити компоненти AR Session та AR Session Origin.

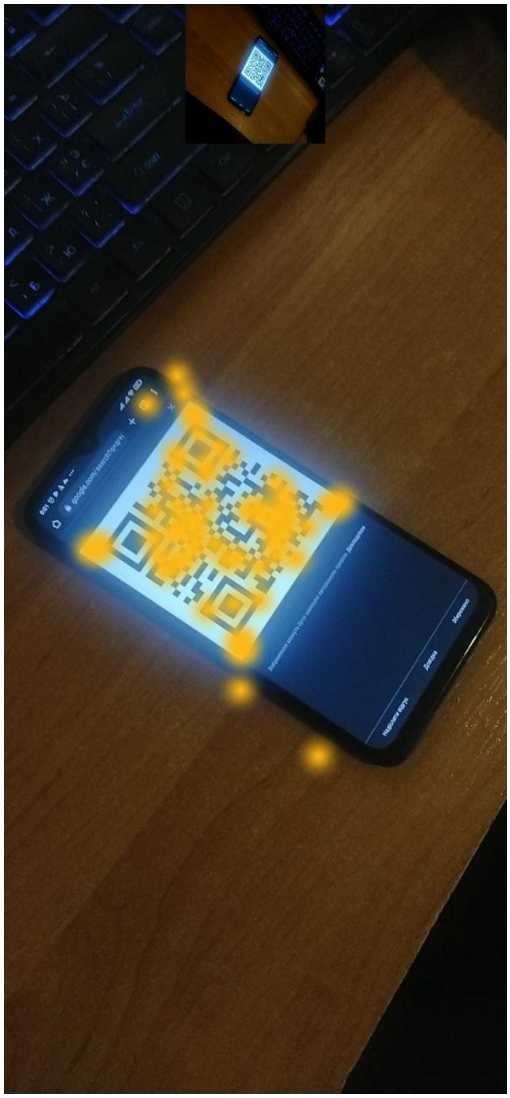


Рис. 3.8. Виявлення особливих точок (feature points) на QR-коді

AR Session є основним компонентом, який відповідає за керування AR-сесією. Він встановлює зв'язок з відповідними датчиками пристрою (такими як камера, гіроскоп, акселерометр) і службами AR платформи (такими як ARKit або ARCore), щоб отримувати дані про навколишній світ і відстежувати розташування пристрою в просторі. AR Session також дозволяє керувати життєвим циклом AR-сесії, здійснювати підключення і відключення, початок і призупинення сесії.

AR Session Origin є контейнером для об'єктів сцени, які взаємодіють з AR-сесією. Цей компонент встановлює початкове положення і орієнтацію AR-сцени, пов'язану зі світовими координатами. Він також виконує конвертацію координат AR в координати сцени Unity і навпаки. AR Session Origin дозволяє розміщувати віртуальні об'єкти в точній відповідності до навколишнього світу, забезпечуючи точне позиціювання та відстеження об'єктів у розширеній реальності. Для використання системи кастування променів необхідно додати до об'єкта AR Session Origin компоненти AR Point Cloud Manager (забезпечує розпізнавання особливих точок) та AR Raycast Manager (дозволяє здійснювати кастування променів) (**рис. 3.9**).

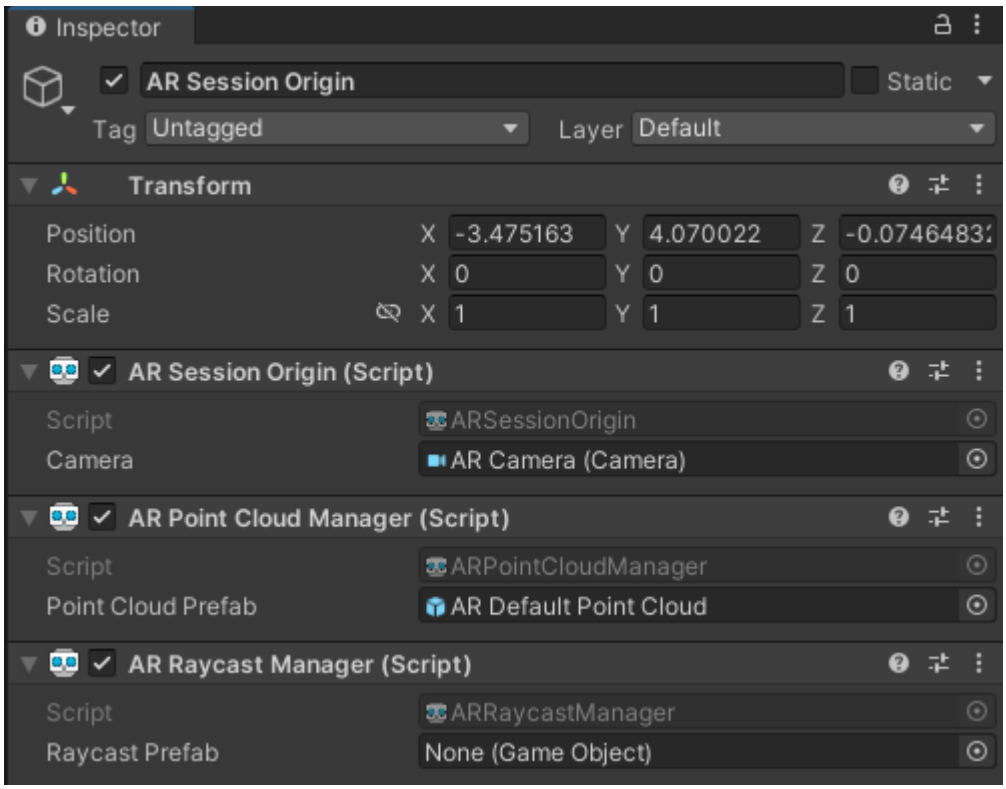


Рис. 3.9. Компоненти об’єкта AR Session Origin

Після кастування променя можна отримати дані координат, які можна використати для розміщення моделі у сцені за допомогою вбудованого в Unity методу Instantiate. Розвернути об’єкт можна відповідно до розміщення ігрової камери відносно самого об’єкта. Після цього користувачу буде відображатись модель експоната за допомогою системи доповненої реальності.

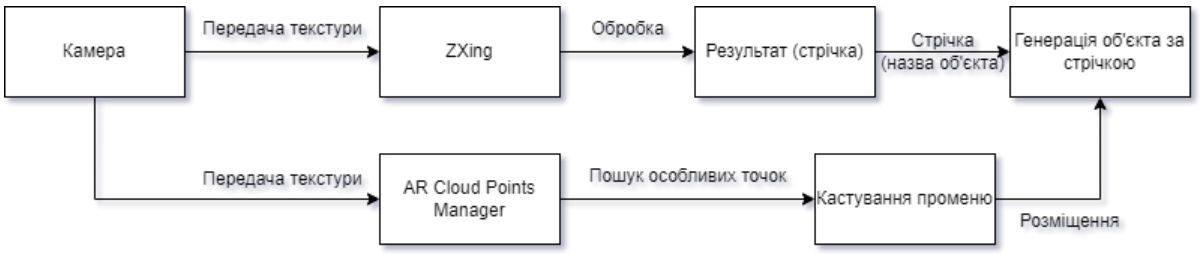


Рис. 3.10. Схема роботи секції сканування QR-коду

3.5. Колекціонування експонатів

Для реалізації колекції експонатів достатньо отримати попередньо збережені дані колекції користувача із моделі даних та розмістити ряд кнопок на інтерфейс гравця, щоб у нього була можливість обрати не обхідний йому експонат. Після вибору гравцем експоната достатньо з використанням ARFoundation віднайти площини у реальному світі за допомогою ARPlaneManager, який відповідає за виявлення та керування плоскими поверхнями у віртуальній реальності. Він надає можливість програмно взаємодіяти з плоскими об'єктами, такими як підлога, столи, стіни та інші поверхні, що розпізнаються AR-передавачем.

ARPlaneManager автоматично виявляє та відстежує плоскі поверхні, що можуть слугувати основою для розміщення віртуальних об'єктів. Він працює на основі даних, отриманих з камери пристрою, та використовує алгоритми комп'ютерного зору для розпізнавання та аналізування плоских поверхонь. Коли плоска поверхня виявлена, ARPlaneManager генерує віртуальний об'єкт, який відповідає цій поверхні та може бути використаний для розміщення AR-елементів, таких як об'єкти, ігрові елементи, або інформаційні шари. Таким чином можна періодично виконувати кастування променів з типом Planes, що будуть реагувати саме на виявлені площини, щоб в необхідний момент (наприклад, коли гравець натисне кнопку) згенерувати модель експоната, що була попередньо обрана.

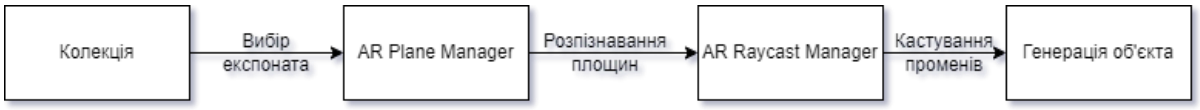


Рис. 3.11. Схема роботи секції колекції

3.6. Аудіогід

Розробку аудіо гіда можна розділити на наступні етапи:

1. Створення екземпляра AudioGuidController: В першому етапі необхідно створити екземпляр класу AudioGuidController, що відповідатиме за управління аудіогідом.
2. Ініціалізація моделі та представлення: Після створення екземпляра, необхідно ініціалізувати модель (AudioGuidModel) та представлення (AudioView). Це включає передачу списку пар GuidAudioClipPair, який містить аудіокліпи, а також UnityAction, який викликається при поверненні (виходу з секції аудіогіда).
3. Налаштування зв'язків між моделлю та представленням: На цьому етапі встановлюються зв'язки між різними подіями та методами моделі та представлення. Наприклад, методи моделі, такі як Pause, Unpause, RemoveTenSeconds, AddTenSeconds, PlayPrevious, PlayNext, ChangePlayTime, PlaySong, пов'язуються з відповідними подіями представлення (OnPauseClicked, OnUnpauseClicked, OnBackClicked, OnForwardClicked, OnPreviousClicked, OnNextClicked, OnChangeTimeClicked, OnChangeClipClicked).
4. Встановлення звукових параметрів та початок відтворення: Після налаштування зв'язків між моделлю та представленням, можна встановити звукові параметри та почати відтворення аудіо. Методи PlaySong та Pause викликаються, щоб встановити початковий аудіокліп та призупинити його відтворення.

Для програвання аудіо в моделі (AudioGuidModel) можна використати компонент AudioSource, що використовується для відтворення аудіо (**рис. 3.12**). Він дозволяє відтворювати звукові файли та керувати різними параметрами відтворення, такими як гучність, панорамування та швидкість відтворення.

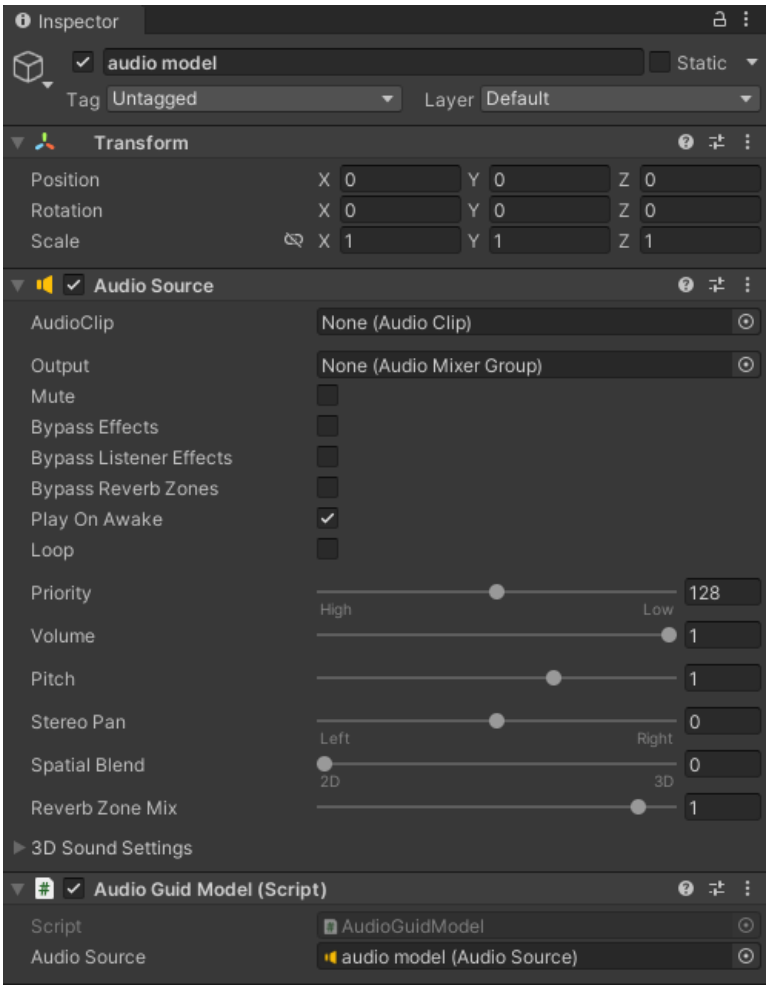


Рис. 3.12. Об’єкт моделі аудіогіда на сцені Unity

Для перемикання аудіо в AudioSource можна використовувати курутини в Unity. Курутина - це спеціальний тип функції, який може бути призупинений на певний час, а потім продовжений з того місця, де вона була призупинена. Курутини в Unity можна створити за допомогою ключового слова "yield" у спеціальних функціях, які позначені ключовим словом "IEnumerator". У такій функції можна виконувати призупинення за допомогою "yield return" і вказати, скільки часу треба призупинити виконання за допомогою "yield return new WaitForSeconds()".

Таким чином, за допомогою курутини можна очікувати та перевіряти чи закінчив програвання аудіокліпу компонент Audio Source, після чого можна перемикати аудіокліп на наступний.

Отже, розробка аудіо гіда включає створення екземпляра, ініціалізацію моделі та представлення, встановлення зв'язків між ними, налаштування параметрів звуку та початок відтворення. Своєю чергою модель для програвання аудіо може використовувати компонент Audio Source та курутини для зручного керування процесом програвання аудіо.

3.7. Історія планетарію

Для реалізації секції історії планетарію за допомогою текстур необхідно мати хорошу верстку, адже потрібно, щоб картинки із відповідними текстурами добре відображались на будь-якому пристрої. У цьому можуть допомогти налаштування Canvas (полотна) та якорів окремих взятих елементів UI (користувацького інтерфейсу).

Canvas в Unity - це компонент, який використовується для створення та керування інтерфейсом користувача (UI). Він дозволяє розміщувати та відображати кнопки, тексти, зображення та інші елементи UI на екрані, а також контролювати їх положення, розмір, взаємодію та вигляд. Canvas є основним інструментом для розробки зручних та естетичних інтерфейсів в Unity, забезпечуючи можливість адаптувати їх до різних роздільних здатностей та пристроїв.

Для початку є необхідним налаштувати параметри полотна, а саме компонента Canvas Scaler, який використовується для масштабування Canvas та його елементів у залежності від роздільної здатності екрана. Особливий режим Scale With Screen Size дозволяє автоматично змінювати розмір та положення елементів Canvas, зберігаючи пропорції та відповідність вмісту екрана. При використанні цього режиму, можна налаштувати бажану ширину та висоту, а

також вибрати спосіб масштабування (наприклад, збереження пропорцій або розтягування до всього екрана). Canvas Scaler у співпраці з режимом Scale With Screen Size допомагає створити гнучкий та адаптивний інтерфейс, який відображатиметься на різних пристроях та роздільних здатностях екрана з оптимальним співвідношенням розмірів.

Якорі в Rect Transform - це параметри, що визначають положення та розмір елемента UI відносно його батьківського контейнера (наприклад, Canvas або іншого об'єкта з RectTransform компонентом). Використовуючи якорі, можна вказати, як елемент UI повинен розтягуватись або зміщатись при зміні розмірів батьківського контейнера.

У Rect Transform є чотири якорі: верхній, нижній, лівий та правий. Кожен якір задається відсотками від розмірів батьківського контейнера. Наприклад, якщо верхній якір встановлений на значення 0.5, це означає, що верхній край елемента UI буде знаходитись посередині по вертикалі батьківського контейнера.

Завдяки якорям можна забезпечити адаптивність та еластичність елементів UI, щоб вони пропорційно змінювалися при зміні розмірів вікна або пристрою (**рис. 3.13**). Налаштовуючи якорі правильно, можна забезпечити, щоб елементи UI завжди знаходилися в потрібних місцях та мали відповідні розміри незалежно від змін вікна або пристрою, на якому вони відображаються, а також змін відповідного контейнера у якому вони знаходяться.

Таким чином можна розставити невидимі кнопки по області текстури в необхідних місцях, забезпечуючи постійний оптимальний розмір при будь-якому відношенні сторін екрана користувача.

Рис. 3.13. Вигляд верстки для головної текстури секції історії планетарію

3.8. Реалізація елементів інтерфейсу

Для кожної секції програми було пропрацьовано необхідні елементи керування (користувацький інтерфейс).

Головне меню: кнопки переходу в секцію колекції, історії планетарію, 360-градусного туру по планетарію, аудіо гіда; а також кнопка увімкнення та вимкнення камери для сканування QR-коду і кнопка для зміни мови.

Аудіогід: контейнер з вмістом усіх аудіофайлів відповідно до вмісту, кнопки взаємодії з обраним аудіо, кнопка повернення до головного меню.

Тур-360: кнопка повернення до головного меню, кнопка зміну режиму (гіроскоп - переміщення пальцем).

Колекція: контейнер з вмістом усіх експонатів користувача (кнопками), кнопка повернення в головне меню.

Історія планетарію: кнопки взаємодії для переходу між сторінками, кнопка повернення до головного меню.

3.9. Тестування

Для тестування продукту в Unity можна використовувати базові інструменти, такі як вікно статистики у вікні Game (**рис. 3.14**), що дозволяє переглядати інформацію про кількість Batches, полігонів і вершин, що бачить камера в окремо взятий момент часу та кількість затраченого часу на відрисовку одного кадру.

Batch - це процес, коли двигун Unity виконує відображення окремих елементів гри. Кожен Batch вимагає виконання додаткових операцій від двигуна, що може призводити до падіння продуктивності гри.

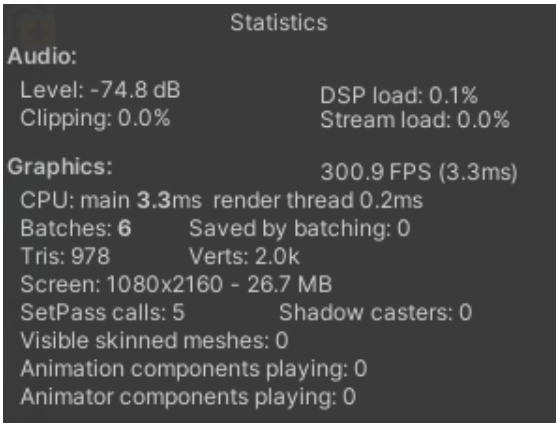


Рис. 3.14. Вікно Statistics

Саме для зменшення кількості Draw Calls (викликів відмальовки до відеокарти), необхідно об'єднувати різні елементи гри в один об'єкт, використовувати спрайтові атласи, а також зменшувати кількість елементів, що відображаються на екрані. Також можна використовувати статичний батчінг для відрисовки статичних об’єктів однією операцією та GPU Instancing для відтворення одного матеріалу на багатьох об'єктах.

Для детальної роботи із кадром та зменшення кількості Draw Calls можна використовувати Frame Debugger (**рис. 3.15**). Frame Debugger - це інструмент в Unity, який дозволяє аналізувати кадри, щоб виявити проблеми з продуктивністю гри. Він дає змогу переглядати кожен Draw Call, що здійснюється в кадрі, та оцінювати, які елементи застосунку можуть бути об'єднані для зменшення кількості Draw Calls.

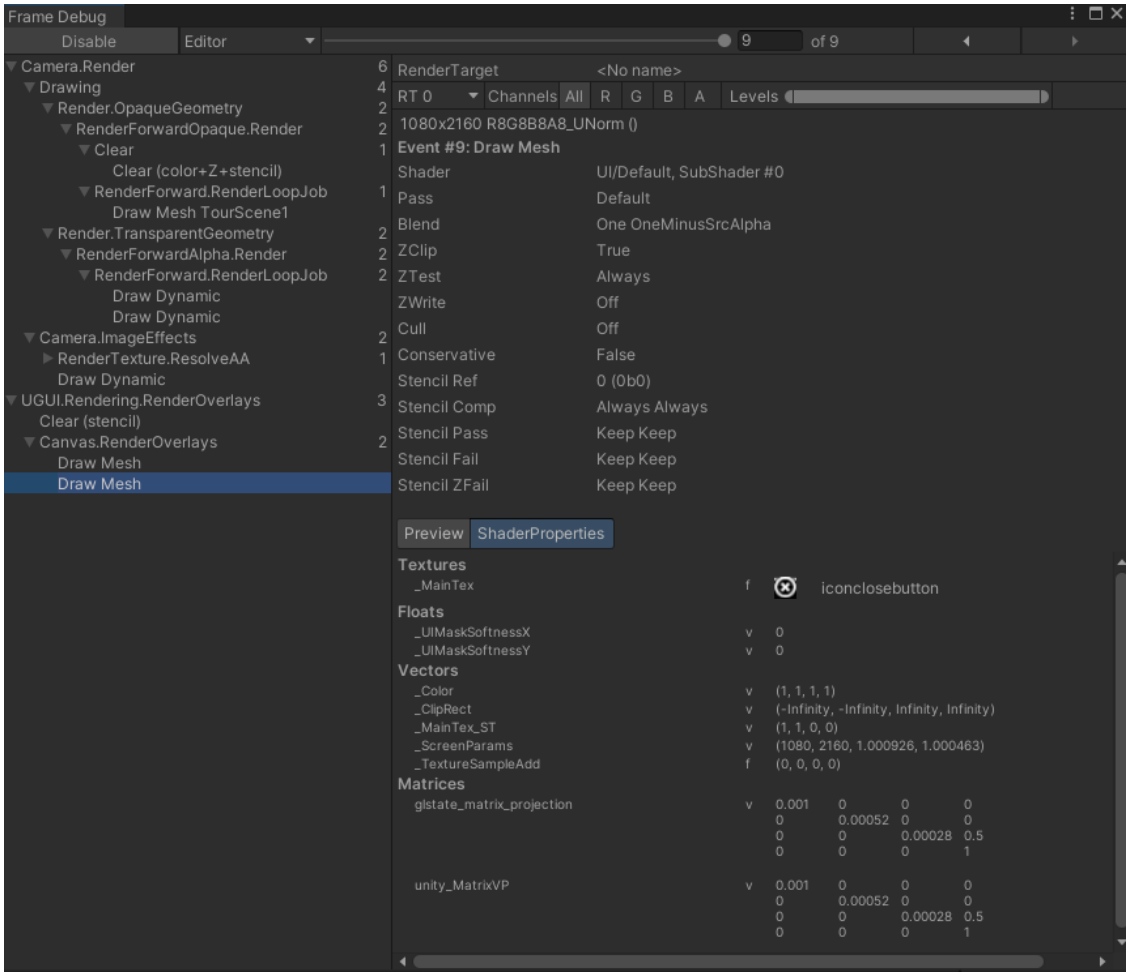


Рис. 3.15. Вікно Frame Debug

Profiler - це інструмент Unity, який дозволяє аналізувати продуктивність гри та знаходити проблеми, які можуть призводити до зниження продуктивності гри. Для використання Profiler потрібно відкрити вікно Profiler, яке містить інформацію про кількість Draw Calls, використання процесора та відеокарти, пам'ять та інші параметри гри.

Для аналізу продуктивності гри з допомогою Profiler, потрібно запустити гру та дозволити їй працювати протягом деякого часу. Після цього Profiler збирає інформацію про продуктивність гри та відображає її в графічному вигляді. За допомогою цієї інформації можна виявити проблеми, які можуть призводити до зниження продуктивності гри, та виправити їх.

Profiler має різні режими, які дозволяють аналізувати різні аспекти продуктивності гри. Режим Hierarchy в Profiler дозволяє аналізувати

використання ресурсів гри за ієрархією об'єктів гри. Для використання режиму Hierarchy, потрібно відкрити вікно Profiler та вибрати режим Hierarchy. Після цього Profiler відображає ієрархію об'єктів гри, яка включає всі об'єкти гри та ресурси, які використовуються цими об'єктами (**рис. 3.16**).

За допомогою режиму Hierarchy, можна аналізувати використання ресурсів гри за кожним об'єктом гри та виявляти проблеми, які можуть призводити до зниження продуктивності гри. Наприклад, можна виявити об'єкти, які використовують надто багато ресурсів гри, та виправити їх для забезпечення максимальної продуктивності гри.

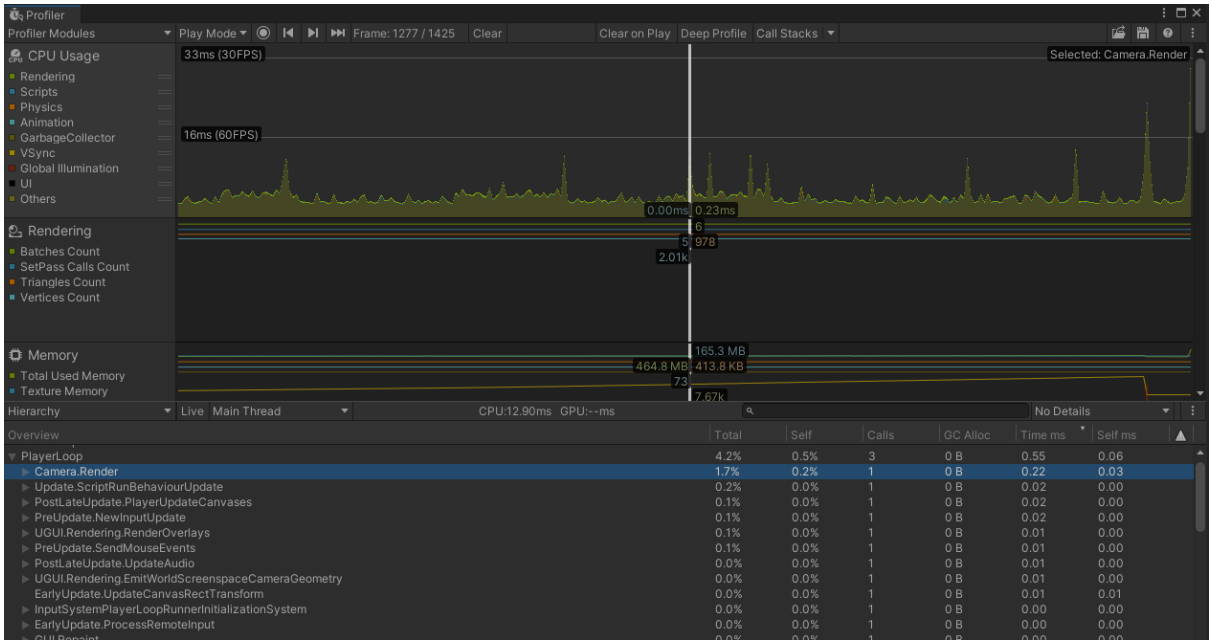


Рис. 3.16. Вікно Profiler

Development build в Unity - це версія застосунку, яка призначена для розробки та тестування. Вона має додаткові можливості та інструменти, які допомагають розробникам налагоджувати, профілювати та вдосконалювати свою роботу.

Development build включає режими налагодження, які дозволяють виводити на екран різноманітну інформацію для аналізу та налагодження, таку як повідомлення про помилки, журнали подій, дані про профілювання продуктивності та багато іншого у самому build (збірці) застосунку на будь-якому пристрої.

Одним з найважливіших аспектів розробки є профілювання, тобто аналіз продуктивності застосунку для виявлення проблемних місць та оптимізації. Unity надає інструменти профілювання у development build, які дозволяють відстежувати використання ресурсів, швидкість відтворення, витрати на CPU та GPU [1].

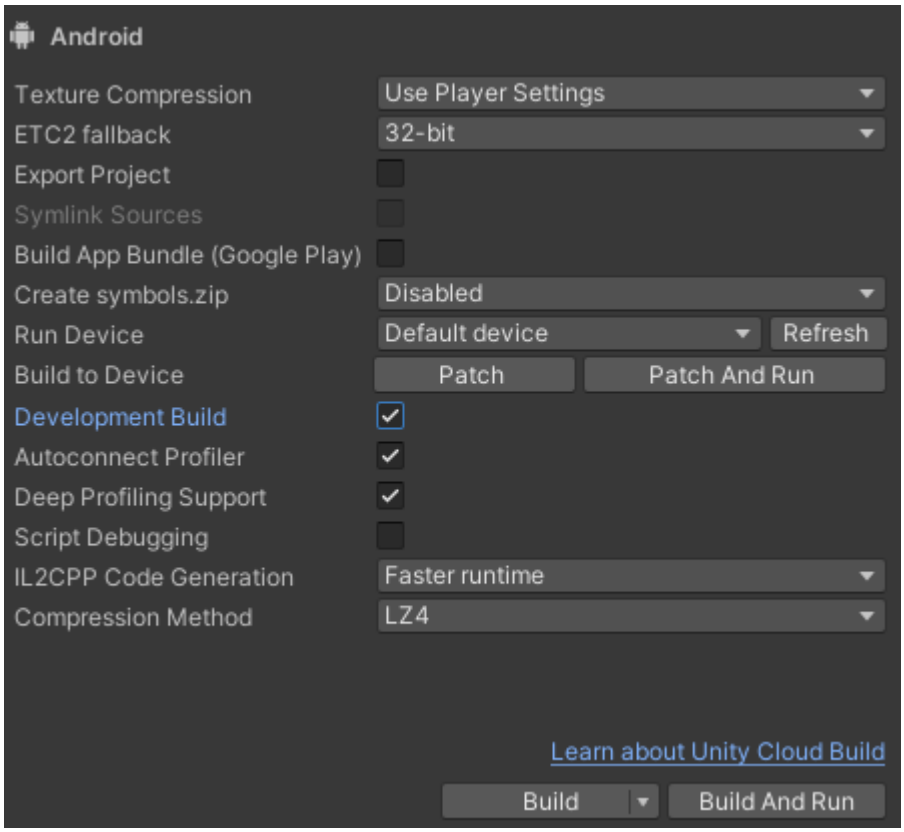


Рис. 3.17. Стандартні налаштування створення білда проєкту

3.10. Оптимізація

Оптимізація текстур в Unity є важливим кроком для забезпечення ефективної роботи вашої гри або додатку. Текстури можуть займати значну кількість пам'яті та впливати на продуктивність, особливо на мобільних пристроях [5]. Ось декілька практик оптимізації текстур:

1. Розмір текстур: Більш оптимізованим є використання розмірів текстур, що є не більші, ніж потрібно. Зайві деталі та роздільна здатність можуть вести до зайвого споживання пам'яті (**рис. 3.18**).

2. Компресія: Використання алгоритмів компресії текстур, такі як DXT, ETC або ASTC, допоможуть зменшити розмір файлу та зберегти пам'ять. Важливо збалансувати якість та розмір, враховуючи вимоги застосунку.
3. Формати текстур: Правильні формати текстур, враховуючи їх особливості та вимоги до кольорів. Наприклад, краще не використовувати формати з прозорістю, якщо вони не потрібні.
4. Використання Sprite Atlas: Створення Sprite Atlas для об'єднання та оптимізації текстур для спрайтів. Це дозволяє знизити кількість викликів до графічного процесора та покращити продуктивність.
5. Управління пам'яттю: Завантажування та вивантажування текстур залежно від потреб гри.

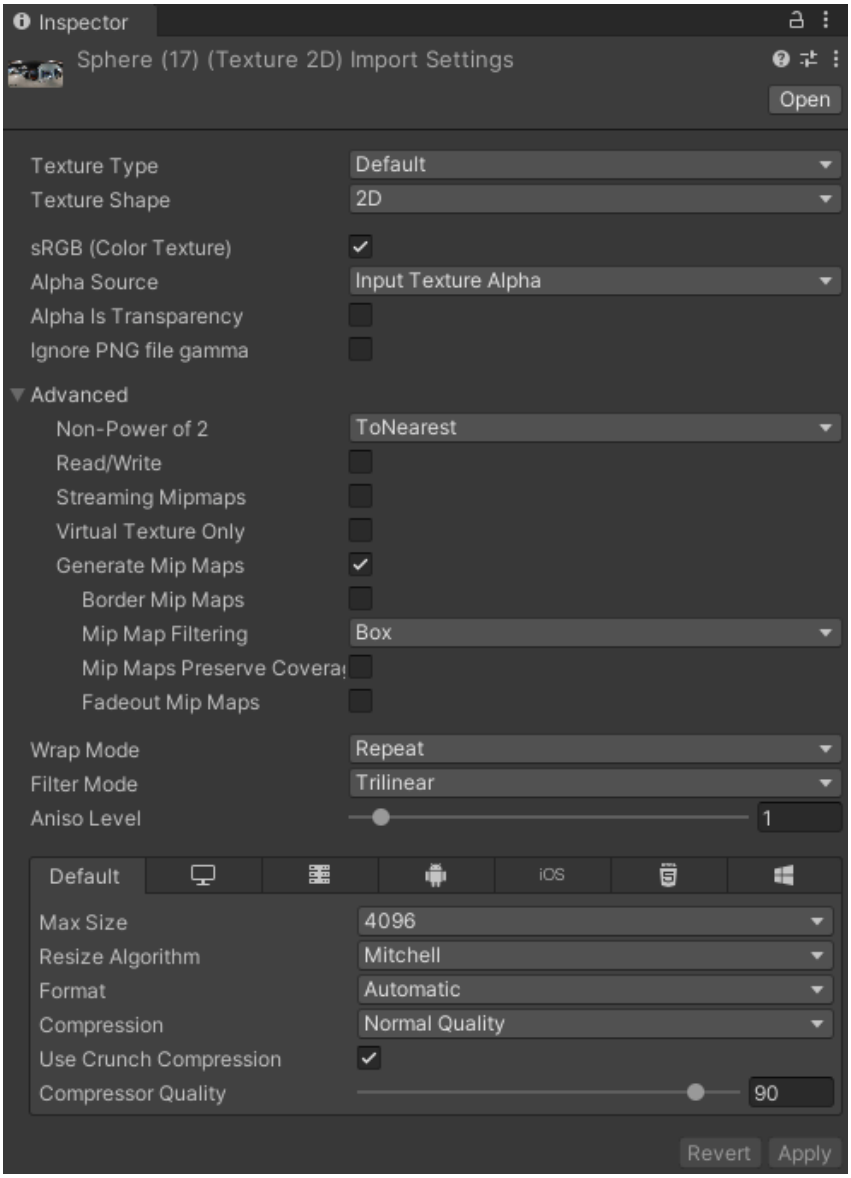


Рис. 3.18. Вигляд об’єкта текстури та її налаштування

Оптимізація аудіо також є важливою складовою частиною розробки застосунку, щоб забезпечити якісний звуковий досвід без зайвого навантаження на ресурси пристрою. Ось декілька практик оптимізації аудіо:

1. Формат аудіо: Використання оптимальних форматів аудіо, які забезпечують прийнятну якість звуку при мінімальному розмірі файлу. Наприклад, використання форматів з втратами, такі як MP3 або AAC, для компресії звуку без втрати значної якості.

2. Бітрейт: Налаштування бітрейту аудіо відповідно до потреб застосунку. Зменшення бітрейту дозволить зменшити розмір файлу, але може вплинути на якість звуку.

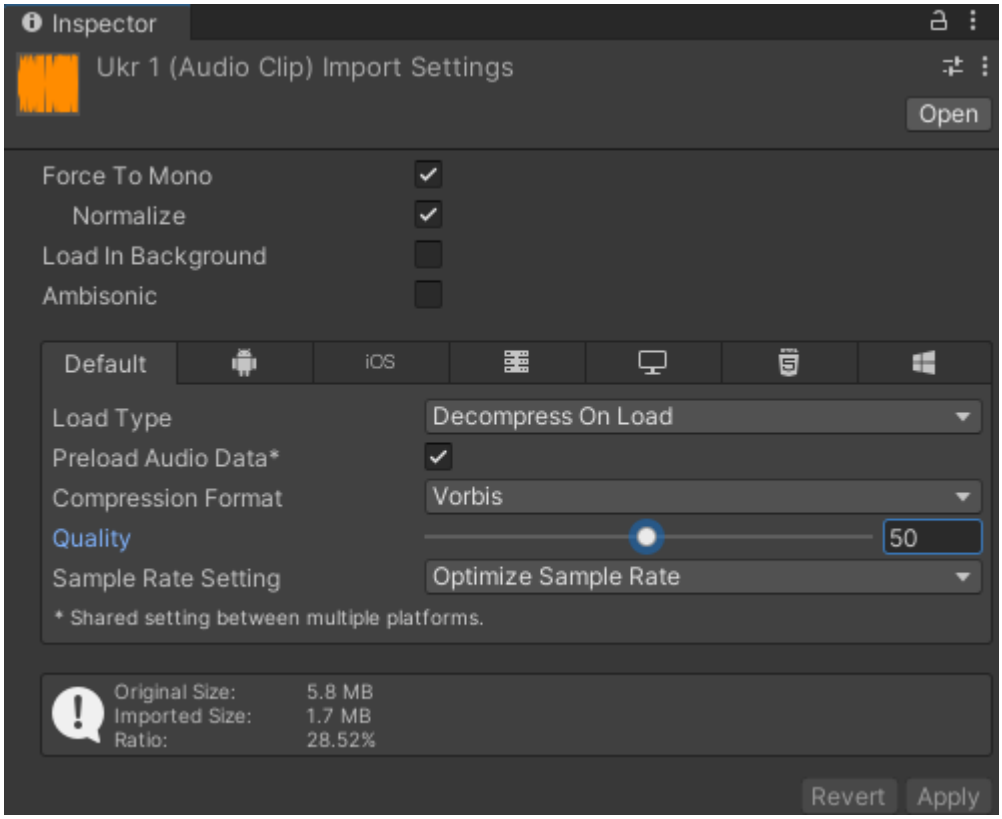


Рис. 3.19. Налаштування об’єкта аудіофайлу

Mask та RectMask2D - це компоненти Unity, які дозволяють приховувати частини елементів гри. Це може бути корисно для створення ефекту обрізання та розширення елементів застосунку, що використовується в контейнерах для колекції (а саме вмісту експонатів) та аудіогіда (вмісту аудіофайлів).

Проте використання компонента Mask може призводити до збільшення кількості Draw Calls, оскільки він створює окремий Draw Call для кожного елементу, який маскується. Щоб зменшити кількість Draw Calls, можна використовувати компонент RectMask2D, який маскує всі елементи, що містяться в межах обмежувального прямокутника, за одним Draw Call, а також використовує більш оптимізовану функцію для маскування.

Raycast Target - це компонент Unity, який дозволяє визначити, чи може бути елемент гри вибраний за допомогою події рейкастингу. Він призведе до зменшення продуктивності гри, якщо буде використовуватися для елементів, які не потребують взаємодії з користувачем.

Щоб зменшити кількість елементів, які відповідають на рейкастинг, можна використовувати компонент Raycast Target лише для елементів, які потребують взаємодії з користувачем, таких як кнопки або інтерактивні об'єкти (знаходиться в компоненті SpriteRenderer та Image). Також можна використовувати компонент Physics Raycaster замість Graphics Raycaster, якщо не потрібна взаємодія з елементами гри, які не мають фізичного тіла.

Canvas - це компонент Unity, який відповідає за відображення елементів гри на екрані. Якщо елементи гри на Canvas не змінюються, то їх відображення на екрані не потребує виконання додаткових операцій від двигуна Unity. Проте якщо елементи гри на Canvas змінюються, то це може призводити до збільшення кількості Draw Calls та зменшення продуктивності гри.

Щоб зменшити кількість Draw Calls та збільшити продуктивність гри, можна розділити елементи гри на окремі Canvas'и в залежності від того, як часто вони змінюються. Наприклад, елементи гри, які змінюються лише один раз на екрані, можна розмістити на окремому Canvas, щоб зменшити кількість Draw Calls для інших елементів гри.

ІНСТРУКЦІЯ З ВИКОРИСТАННЯ

Можливості інтерфейсу:

Головне меню:

Кнопки по черзі (зверху вниз):

1. Тимчасова для майбутньої функції новин,
2. Аудіогід,
3. Зміна мови,
4. Історія планетарію,
5. Тур-360 по планетарію,
6. Колекція користувача,
7. Згорнути/розгорнути меню.

Центральна кнопка знизу - увімкнути/вимкнути камеру для сканування QR-коду.

Рис. 4.1. Головне меню

Аудіогід:

Кожна секція контейнера - окремий аудіофайл, який можна програти.

Центральна кнопка знизу - повернення до головного меню.

Рис. 4.2. Секція “Аудіогід”

Історія планетарію:

Кнопки керування (відрізняються в залежності від поточної текстури).

Центральна кнопка знизу - повернення до головного меню.

Рис. 4.3. Секція “Історія планетарію”

Тур-360:

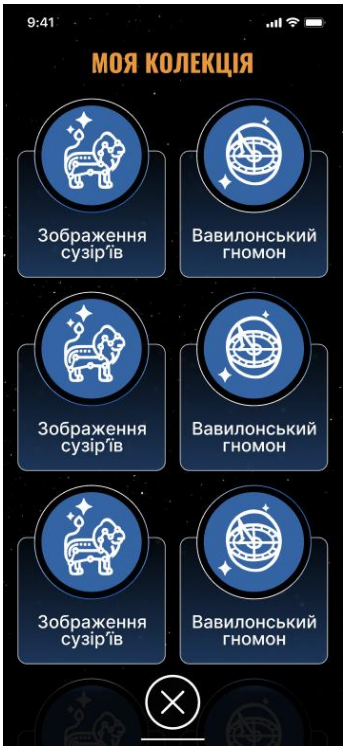
Кнопки переходу на іншу сцену (кнопки керування сценами).

Справа зверху - кнопка зміни режиму (гіроскоп - керування жестами).

Справа знизу - повернення до головного меню.

Рис. 4.4. Секція “Тур-360”

Колекція користувача:



Кожна секція контейнера - окремий експонат, який можна розмістити в AR.

Центральна кнопка знизу - повернення до головного меню.

Рис. 4.5. Секція “Колекція”

Можливості функціоналу:

Дана система дозволяє користувачу здійснювати 360-градусні прогулянки по планетарію, читати історію планетарію двома мовами (українською та англійською), прослуховувати інформацію про більш ніж 20-и експонатів двома мовами (українською та англійською), зчитувати QR-коди всередині планетарію для огляду більш ніж 30-и експонатів у режимі AR та зберігати їх у свою локальну колекцію із подальшою можливістю відтворення їх в режимі AR будь-де та будь-коли.

ВИСНОВКИ

У цьому проєкті було розроблено спеціалізований застосунок для планетарію, що базується на використанні технології доповненої реальності (AR). Головною метою проєкту було створення інноваційного досвіду для глядачів, які відвідують планетарій.

Застосунок було розроблено для платформ Android та iOS з використанням Unity та AR Foundation. Він пропонує унікальну можливість досліджувати планетарій у віртуальному просторі, де глядачі можуть переглядати експонати, долучатися до інтерактивних активностей та отримувати розширену інформацію про кожен об'єкт.

Проєкт використовує різноманітні технології, такі як AR, гіроскоп, QR-коди та інші, для створення захопливого та освітнього досвіду для користувачів. Він надає зручний інтерфейс та функціональність, що дозволяють глядачам насолоджуватися планетарієм в новому форматі.

У процесі розробки були враховані основні вимоги до оптимізації графіки та аудіо, щоб забезпечити плавний та ефективний досвід використання застосунку.

Проєкт успішно втілює ідею створення інноваційного досвіду для відвідувачів планетарію, використовуючи технології AR та мультимедіа. Він надає можливість поглибитися у світ космосу, досліджувати планети та галактики, а також отримувати цікаву та освітню інформацію. Цей проєкт відкриває нові горизонти для інтерактивного навчання та розваги у сфері планетаріїв.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Joe Hocking. - “Unity in Action: Multiplatform Game Development in C#” - Персональна інтернет-публікація, 2022. - 386с.
2. Mark Reed. - “C#: 3 Books in 1” - Персональна інтернет-публікація, 2022. - 344с.
3. Nicolas Alejandro Borromeo. - “Hands-On Unity 2022 Game Development” - Packt Publishing, 2022. - 712с.
4. Unity Documentation [Електронний ресурс] : [Веб-сайт]. - Електронні дані. - Режим доступу: https://docs.unity3d.com/Manual/index.html (дата звернення 23.03.2023) - Назва з екрана.
5. Google. Free Fonts [Електронний ресурс] : [Веб-сайт]. - Електронні дані. - Режим доступу: https://fonts.google.com/ (дата звернення 13.06.2023) - Назва з екрана.
6. AR Foundation Documentation [Електронний ресурс] : [Веб-сайт]. - Електронні дані. - Режим доступу: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/index.html (дата звернення 21.04.2023) - Назва з екрана.
7. Planetarium Database [Електронний ресурс] : [Веб-сайт]. - Електронні дані. - Режим доступу: planetariums-database.org (дата звернення 27.06.2023) - Назва з екрана.

ДОДАТКИ

Додаток А

Код модулю аудіо гіда

```
using NooPlanetAR.Structures;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace NooPlanetAR.AudioGuid
{
   public class AudioGuidModel : MonoBehaviour
   {
      public Action<GuidAudioClipPair> OnClipStarted;
      public Action<GuidAudioClipPair, float> OnClipTimeChanged;
      public Action<bool> OnClipPaused;

      [SerializeField] private AudioSource audioSource;

      private List<GuidAudioClipPair> clips;
      private bool isPaused = true;
      private int currentClipIndex;

      private Coroutine controlRoutine;

      public void Initialize(List<GuidAudioClipPair> clips)
      {
```

```
    this.clips = clips;
}

public bool IsPlaying()
    => audioSource.isPlaying;

public void PlaySong(int index)
{
    print(index);
    audioSource.time = 0f;
    audioSource.clip = clips[index].Clip;
    audioSource.Play();
    OnClipStarted?.Invoke(clips[index]);
    currentClipIndex = index;
    isPaused = false;
    print($"play song: {clips[currentClipIndex].Label}");

    if (controlRoutine == null)
        StartCoroutine(ControlAudioAsync());
}

public void Pause()
{
    if (isPaused == false)
    {
        audioSource.Pause();
        OnClipPaused?.Invoke(true);
        isPaused = true;
        print("paused");
```

```
    }
}

public void Unpause()
{
    if (isPaused)
    {
        audioSource.Play();
        OnClipPaused?.Invoke(false);
        isPaused = false;
        print("unpaused");
    }
}

public void PlayNext()
{
    if (currentClipIndex >= clips.Count - 1)
        currentClipIndex = -1;
    currentClipIndex++;
    print($"play next at index: {currentClipIndex}");
    PlaySong(currentClipIndex);
}

public void PlayPrevious()
{
    if (currentClipIndex == 0)
        currentClipIndex = clips.Count;
    currentClipIndex--;
    print($"play previous at index: {currentClipIndex}");
```

```
            PlaySong(currentClipIndex);
        }

        public void ChangePlayTime(float time)
        {
            if (time < 0f || time > 1f)
            {
                Debug.LogError("play time isn't clamped between 0 and 1");
                return;
            }
            audioSource.time = audioSource.clip.length * time;
        }

        public void AddTenSeconds()
        {
            print("add ten seconds");
            const float addTime = 10f;
            if (audioSource.time + addTime > audioSource.clip.length)
            {
                PlayNext();
                return;
            }
            audioSource.time += addTime;
            OnClipTimeChanged?.Invoke(clips[currentClipIndex], audioSource.time
/ audioSource.clip.length);
        }

        public void RemoveTenSeconds()
        {
```

```
            print("remove ten seconds");
            const float removeTime = 10f;
            if (audioSource.time - removeTime < 0f)
            {
                if (audioSource.time < 1f)
                    PlayPrevious();
                else
                    audioSource.time = 0f;
                return;
            }
            audioSource.time -= removeTime;
            OnClipTimeChanged?.Invoke(clips[currentClipIndex], audioSource.time
/ audioSource.clip.length);
        }

        private IEnumerator ControlAudioAsync()
        {
            while (true)
            {
                if (audioSource.isPlaying == false)
                {
                    if (isPaused == false)
                    {
                        //end of track, play next
                        if (currentClipIndex >= clips.Count - 1)
                            currentClipIndex = -1; //repeat playlist
                        currentClipIndex++;
                        print("start to play next song");
                        PlaySong(currentClipIndex);
```

```
                }
            }
            else
            {
                OnClipTimeChanged?.Invoke(clips[currentClipIndex],
audioSource.time / audioSource.clip.length);
            }
            yield return null;
        }
    }
}
```

Код моделі даних колекції користувача:

```
using NooPlanetAR.Data.ModelsCollection.Scriptable;
using UnityEngine;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using NooPlanetAR.Structures;

namespace NooPlanetAR.Data.ModelsCollection
{
    [Serializable]
    public class CollectionsModelsData
    {
        public ModelsKeys ModelsKeys { get => modelsKeys; }
```

```
        [SerializeField] private ModelsKeys modelsKeys;

        private UserModelCollection userCollection;
        private bool userCollectionIsLoaded = false;

        private string FullCollectionPath => Application.persistentDataPath +
collectionPath;
        private const string collectionPath = "/UserModelCollection.txt";

        public bool HasKey(string key)
            => modelsKeys.HasKey(key);

        public GameObject GetResourceByKey(string key)
        {
            if (HasKey(key) == false)
            {
                Debug.LogError($"cann't find model-key pair with key {key}");
                return null;
            }
            var path = modelsKeys.GetPathByKey(key);
            var resource = Resources.Load<GameObject>(path);
            return resource;
        }

        public bool AddKeyToCollection(string key)
        {
            var collectionKeys = GetUserCollectionKeys();

            if (collectionKeys.Any(k => k == key))
```

```
            return false;

        collectionKeys.Add(key);
        SaveUserModelCollection();
        return true;
    }

    public List<ModelPair> GetUserModels()
    {
        var userCollectionKeys = GetUserCollectionKeys();

        var userCollectionModelPairs = new List<ModelPair>();
        foreach (var key in userCollectionKeys)
            if (modelsKeys.HasKey(key))
                userCollectionModelPairs.Add(modelsKeys.GetPairByKey(key));

        return userCollectionModelPairs;
    }

    private List<string> GetUserCollectionKeys()
    {
        if (userCollectionIsLoaded == false)
        {
            if (File.Exists(FullCollectionPath))
            {
                var json = File.ReadAllText(FullCollectionPath);
                userCollection                                          =
JsonUtility.FromJson<UserModelCollection>(json);
            }
```

```
            else
            {
                userCollection = new UserModelCollection();
            }
            userCollectionIsLoaded = true;
        }
        return userCollection.Keys;
    }

    private void SaveUserModelCollection()
    {
        var json = JsonUtility.ToJson(userCollection);

        Debug.Log($"save collection: {json}");
        File.WriteAllText(FullCollectionPath, json);
    }
  }
}
```