

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультет)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ВПРОВАДЖЕННЯ AR-ТЕХНОЛОГІЙ В КОРИСТУВАЦЬКИХ
МОБІЛЬНИХ ЗАСТОСУНКАХ НА ПРИКЛАДІ ARKIT У IOS
IMPLEMENTATION OF AR TECHNOLOGY IN CUSTOMER MOBILE
APPLICATIONS ON THE CASE OF ARKIT IN IOS

Виконав: студент 2 курсу, групи КНТ-210м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

Овчаренко Д.Е.

(прізвище та ініціали)

Керівник Колпакова Т.О.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

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

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(КОД І НАЙМЕНУВАННЯ)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Овчаренка Даніла Едуардовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Впровадження AR-технологій в користувацьких мобільних застосунках на прикладі ARKit у iOS. Implementation of AR Technology in Customer Mobile Applications on the case of ARKit in iOS

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “5” листопада 2021 року
№ 435

2. Строк подання студентом проєкту (роботи) 13 грудня 2021 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблем та постановка завдань дослідження. 2. Вибір та обґрунтування структури програмного продукту 3. Розробка архітектури програми. 4. Розробка програми. 5. Експлуатація та тестування програми. 6. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Колпакова Т.О., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання “1” вересня 2021 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.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9-10 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	11-12 тиждень	Додатки
8	Нормоконтроль та рецензування.	13-15 тижні	
9	Захист роботи.	16 тиждень	

Студент(ка)

(підпис) Овчаренко Д.Е.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Колпакова Т.О.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 93 с., 52 рис., 3 табл., 3 дод., 12 джерел.

ФРЕЙМВОРК, ДОПОВНЕНА РЕАЛЬНІСТЬ, КОРИСТУВАЧ, ОБ'ЄКТ, ЗАСТОСУНОК, ПРОСТІР, ІНТЕРФЕЙС, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, SWIFT, XCODE, КЛАСИ.

Об'єкт дослідження: технології доповненої реальності у розробці мобільних застосунків.

Предмет дослідження: програмний застосунок для вивчення космічних тіл та космосу на основі фреймворку ARKit.

Мета роботи: створення мобільного застосунку для вивчення космічних тіл та космосу за допомогою доповненої реальності.

Матеріали та методи: мова програмування Swift, фреймворки ARKit, UIKit та SwiftUI.

Технічні засоби: для розробки використано персональний комп'ютер під управлінням операційної системи MacOS, для тестування – пристрої під управлінням операційної системи iOS.

Результати: Створено мобільний застосунок, який надає користувачеві можливість комфортного вивчення астрономії з дружнім інтерфейсом. Підвищено точність визначення місця розташування AR-елементів за рахунок методу touchesBegan.

Висновки: Розроблено мобільний застосунок для вивчення астрономії, з можливістю додавання зменшених копій космічних тіл у реальний простір через камеру смартфона користувача.

Галузь використання: вивчення астрономії та ознайомлення з небесними тілами сонячної системи.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 93 pages, 52 figures, 3 tables, 3 appendixes, 12 sources.

FRAMEWORK, AUGMENTED REALITY, USER, OBJECT, APPLICATION, SPACE, INTERFACE, SOFTWARE, SWIFT, XCODE, CLASSES.

Object of research is technologies of augmented reality in the development of mobile applications.

Subject of study is a software application for the study of cosmic bodies and space on the basis an ARKit framework.

The purpose is to create a mobile application for the study of cosmic bodies and space with the help of augmented reality.

Materials and methods: Swift programming language, ARKit, UIKit and SwiftUI frameworks.

Hardware: A personal computer running the MacOS operating system was used for development, and devices running the iOS operating system were used for testing.

Results: A mobile application has been created that allows the user to comfortably study astronomy with a friendly interface. Improved the accuracy of locating AR elements due to the method of touchesBegan.

Conclusions. Developed a mobile application for the study of astronomy, with the ability to add reduced copies of space bodies in real space through the camera of the user's smartphone.

Field of use is a study of astronomy and acquaintance with the celestial bodies of the solar system.

ЗМІСТ

	С.
Перелік умовних позначень і скорочень.....	8
Вступ.....	9
1 Аналіз проблем та постановка завдань дослідження	10
1.1 Огляд існуючих методів вирішення.....	10
1.2 Типи доповненої реальності.....	10
1.3 Сфери використання доповненої реальності.....	12
1.4 Існуючі мобільні застосунки, засновані на AR-технологіях	14
1.5 Огляд AR-бібліотек.....	18
1.6 Стандартні методи позиціонування в доповненій реальності.....	23
1.7 Висновки за розділом 1 та постановка завдання роботи	24
2 Вибір та обґрунтування структури програмного продукту	25
2.1 Аналіз технологій розробки.....	25
2.2 Розробка структури програми	26
2.3 Проектування системи	29
2.4 Висновки за розділом 2	32
3 Розробка архітектури програми	34
3.1 Впровадження технологій ARKit	34
3.2 Реалізація основних функцій системи.....	34
3.3 Метод touchesBegan.....	34
3.4 Висновки за розділом 3	45
4 Розробка програми	46
4.1 Опис основних етапів розробки.....	46
4.2 Початок створення та налаштування додатку	46
4.3 Створення файлової системи та наповнення додатку	47
4.4 Налаштування елементів проектування	47
4.5 Додавання іконки додатку та SwiftUI	50
4.6 Висновки за розділом 4	51
5 Експлуатація та тестування програми.....	52

5.1 Характеристика, призначення та умови використання програми	52
5.2 Звертання до програми.....	52
5.3 Опис основних функцій	52
5.4 Призначення та умови виконання програми.....	53
5.5 Виконання програми	54
5.6 Повідомлення	61
5.7 Тестування програми	61
Висновки.....	63
Перелік джерел і посилання	64
Додаток А Технічне завдання.....	65
Додаток Б Текст розробленої програми.....	68
Додаток В Слайди презентації	86

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

App	– Application;
AR	– Augmented reality, доповнена реальність;
ARKit	– AR бібліотека;
ARV	– AR View;
ARScene	– AR елемент проєктування;
IDE	– Integrated Development Environment;
GDB	– відладчик;
GNU	– Unix-подібна операційна система;
-Kit	– закінчення назв бібліотек та фреймворків;
LLVM	– програмна інфраструктура для створення компіляторів;
SceneKit	– SceneKit бібліотека;
SDK	– Software Development Kit;
UIKit	– UI бібліотека;
UI	– User Interface;
VIO	– Visual Inertial Odometry;
WV	– Web View (елемент проєктування);
ООП	– об'єктно-орієнтоване програмування;
ОП	– оперативна пам'ять;
ОС	– операційна система;
ПЗ	– програмне забезпечення;
ТЗ	– технічне завдання.

ВСТУП

Згідно із щорічними звітами компаній, які займаються створенням мобільних пристроїв, у 2021 році майже кожна людина у світі є власником смартфона. Кожен з цих користувачів має певні вподобання та теми, що їх цікавлять. Потік інформації, що щоденно проходить через смартфони користувачів, постійно зростає, і вже немає фізичної можливості прослідкувати за усіма змінами, що відбуваються у світі мобільної розробки, незалежно від операційної системи чи об'єму постійної пам'яті вашого мобільного пристрою. Серед великої кількості мобільних застосунків, які використовують доповнену реальність, майже всі є іграми.

У App Store небагато освітніх мобільних застосунків, а ще менше з них використовують AR технології.

Тож актуальність використання AR технологій у користувацьких мобільних застосунках пов'язана з потребою в зацікавленні людей в навчанні. Для вирішення даної проблеми необхідно створити програму, яка б надавала можливість користувачеві навчатися певної теми з можливістю розважитись за допомогою AR технологій.

Програмний продукт матиме архітектуру MVC, а також має працювати на iOS пристроях починаючи з iPhone7.

1 АНАЛІЗ ПРОБЛЕМ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд існуючих методів вирішення

Доповнена реальність (в перекладі з англійської augmented reality) – це доповнення фізичного світу за допомогою цифрових даних, яке забезпечується комп'ютерними пристроями (смартфонами, планшетами та окулярами AR) в режимі реального часу.

Доповнена реальність на перший погляд може здатися не такою захопливою, як віртуальна реальність, проте може привнести неабияку користь в повсякденне життя. Вона має в собі величезний потенціал, оскільки переносить елементи з віртуального світу в реальний, доповнюючи речі, які ми здатні бачити, чути, чи навіть відчувати [1].

Говорячи простими словами, на відміну від VR (Virtual Reality), яка вимагає повного занурення у віртуальне середовище, AR використовує середовище навколо нас та просто накладає поверх нього певну частинку віртуальної інформації, наприклад графіку. Оскільки віртуальний та реальний світи гармонійно співіснують, користувачі з досвідом доповненої реальності мають змогу спробувати цілком новий, покращений світ, де віртуальна інформація використовується як додатковий корисний інструмент, що забезпечує допомогу в повсякденній діяльності [1]. Розрізняють декілька типів доповненої реальності.

1.2 Типи доповненої реальності

1.2.1 Доповнена реальність, що базується на маркерах

Деколи доповнену реальність, що базується на маркерах, також називають розпізнаванням зображень. Цей тип технології використовує камеру та спеціальний пасивний візуальний маркер, наприклад QR-код (quick response code – код швидкого відгуку), який показує запрограмований

результат лише тоді, коли сенсор його зчитує. Таким чином, вдається вирізнити віртуальні об'єкти з реального світу [2].

1.2.2 Безмаркерна доповнена реальність

Також безмаркерну доповнену реальність ще називають координатно-, або GPS-орієнтованою. Щоб надати дані про ваше місцеперебування, вона може використовувати систему глобального позиціювання (GPS – Global Positioning System), цифровий компас, датчик швидкості або акселерометр, якими оснащено ваш пристрій. Завдяки масовому розповсюдженню смартфонів та планшетів ця технологія використовується найчастіше на даний момент. Найпоширеніші випадки використання – це позначення напрямків, пошук потрібних місць, таких як кафе чи офіс, або ж у додатках, що орієнтовані на місце розташування [2].

1.2.3 Доповнена реальність, що базується на проєкції

Доповнена реальність, що базується на проєкції, працює шляхом проєктування світлових проєкцій на фізичні поверхні. Спеціальні додатки допомагають здійснювати взаємодію між людиною та проєкцією, визначаючи моменти дотику людини до світла, яке проєктується. Це досягається за допомогою порівняння очікуваної проєкції та зміненої певними перешкодами, наприклад через дотик рукою. Ще один цікавий спосіб – застосування плазмової технології, завдяки якій можна створювати тривимірні проєкції в просторі [2].

1.2.4 Доповнена реальність, що базується на VIO

Візуальна інерціальна одометрія (Visual Inertial Odometry) – це технологія, яка допомагає відслідковувати позицію та орієнтуватися в

просторі за допомогою сенсорів та камери. Завдяки цьому можливо створити точну 3D-модель простору навколо пристрою, оновлювати її в реальному часі, визначати в ній положення, передавати ці дані всім додаткам та накладати поверх неї додаткові шари. Можливості цієї технології насправді унікальні: можна вимірювати відстані, вставляти різноманітні об'єкти в інтер'єр та взаємодіяти з ними. VIO обіцяє стати найперспективнішою технологією в AR, на даний момент її використовують такі гіганти, як компанія Google в своєму Project Tango та компанія Apple в ARKit [2].

1.3 Сфери використання доповненої реальності

Можливості використання AR технологій практично безмежні та можуть бути застосовані майже в усіх аспектах нашого життя. Вони якісно змінюють способи нашого спілкування, споживання інформації та ведення бізнесу [1].

Нижче розглянуто декілька прикладів використання AR в реальному житті.

1.3.1 Освіта

Додаток Google Expeditions може зробити навчання більш інтерактивним за допомогою спеціальних турів, в яких юні дослідники можуть розглядати різноманітні об'єкти, поки викладач про них розповідає.

1.3.2 Медицина

Компанія Viraar запропонувала рішення, яке поєднує телемедицину та AR. За допомогою окулярів Google Glass та додатку Viraar, хірурги на відстані можуть асистувати своїм колегам, проєктуючи свої руки на окуляри хірурга, який проводить операцію [1].

1.3.3Авіація

AR досить давно використовується військовими пілотами. Спеціальні дисплеї та шоломи виводять інформацію про системи винищувача та допомагають наведенню на ціль. AR починає проникати також в цивільну авіацію. Наприклад, компанія Aero Glass розробила спеціальні окуляри доповненої реальності, які допомагають пілоту орієнтуватися в просторі, дотримуватися маршруту та отримувати додаткову інформацію під час польоту [1].

1.3.4 Маркетинг

За допомогою AR технологій бренди можуть проводити більш креативні інтерактивні рекламні компанії, тим самим привертаючи додаткову увагу до своїх продуктів. Наприклад, для реклами п'ятого сезону серіалу "Ходячі Мерці", на автобусній зупинці встановили AR систему, яка "приводила" зомбі в реальний світ [2].

1.3.5 Туризм

Для нових поколінь притаманно пізнавати світ за допомогою прямої взаємодії. Наприклад, Каталонський національний музей мистецтва почав активно застосовувати AR для кращої навігації заплутаними коридорами та інтерактивного ознайомлення з експонатами [2].

1.3.6 Дизайн

Облаштувати квартиру стало набагато простіше, оскільки замість того щоб уявляти, чи підійде вам певний елемент, меблі можна просто підставити у свій інтер'єр віртуально – за допомогою AR додатку Furniture Dropping.

1.3.7 Шопінг

Під час того, як ви будете в супермаркеті, додатки на основі AR допомагатимуть орієнтуватися з поміж великої кількості рядів та знаходити оптимальний шлях до потрібного товару. Крім того буде можливість отримати додаткову інформацію стосовно знижок та вигідних пропозицій.

1.3.8 Ігри

Напевне, немає людини, яка б не стикнулася з божевіллям Pokemon Go тим чи іншим способом. Дехто сам бігав містом вишукуючи покемонів, а дехто дратувався оминаючи людей, які втупившись в смартфон плуталися під ногами [2].

1.4 Існуючі мобільні застосунки, засновані на AR-технологіях

Наразі на ринку мобільних застосунків вже існують продукти, які спроектовані для вирішення певних навчальних і розважальних проблем, і які широко використовують технології доповненої реальності. Далі наведено декілька прикладів подібних продуктів [2].

1.4.1 Програмний продукт Pokemon Go

Pokemon Go – це ігровий проєкт у так званій «доповненій реальності» для смартфонів та планшетів на базі операційних систем Android та iOS, що набув шаленої популярності одразу після виходу. У 2016 році це був один із найбільш використовуваних і прибуткових мобільних додатків, до кінця року його завантажили понад 500 мільйонів разів у всьому світі, а у 2019 році кількість завантажень сягнула 1 мільярду.

В Pokemon Go гравцям належить шукати віртуальних покемонів, що ховаються в реальних локаціях, наводячи камеру смартфона на цю локацію, викликати їх на поєдинки і ловити у разі перемоги.

Переваги програмного продукту Pokemon Go:

- адаптованість на смартфони усіх ОС;
- зручний інтерфейс.

Недолік даного продукту – це те, що єдиною функцією цього застосунку є розвага.

1.4.2 Програмний продукт Solar Walk

Solar Walk – це 3D модель Сонячної системи для наявного вивчення планет, супутників, зірок, комет та інших небесних об'єктів, цифровий гід просторами нашого Всесвіту, що дозволяє вам здійснити прогулянку Сонячною системою і відчути себе справжнім дослідником космосу (рис.1.1).

Переваги програмного продукту Solar Walk:

- можливість навчання для користувача;
- достатньо зручний інтерфейс;
- наявність моделей космічних тіл.

Недоліки даного продукту наступні:

- відсутність української мови;
- висока академічність подання інформації.

1.4.3 Програмний продукт Cool Astronomy

Cool Astronomy – це мобільний додаток орієнтований на навчання користувача астрономії та космічних фактів, з моделями небесних об'єктів, схемами та описами сузір'їв.



Рисунок 1.1 – Інтерфейс програми Solar Walk

Переваги програмного продукту Cool Astronomy:

- навчальна орієнтованість;
- зрозумілий та зручний інтерфейс;
- наявність моделей космічних тіл;
- наявність схем та описів сузір'їв.

Недоліки даного продукту наступні:

- відсутність української мови;
- платний мобільний додаток;
- висока серйозність подання інформації (рис.1.2).

1.4.4 Програмний продукт Star Chart

Star Chart – це мобільний додаток орієнтований на навчання користувача астрономії та космічних фактів, з моделями небесних об'єктів, схемами та описами сузір'їв.

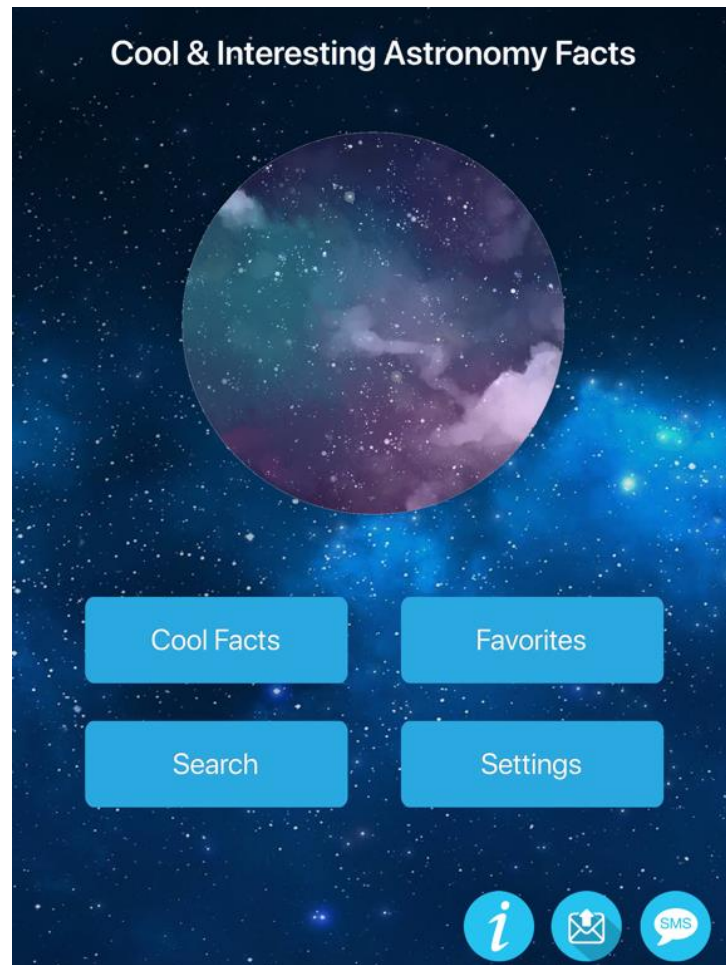


Рисунок 1.2 – Інтерфейс програми Cool Astronomy

Переваги програмного продукту Star Chart:

- можливість навчання для користувача;
- привабливий інтерфейс;
- наявність схем та описів сузір'їв.

Недоліки даного продукту наступні:

- відсутність української мови;
- відсутність моделей космічних тіл (рис.1.3).

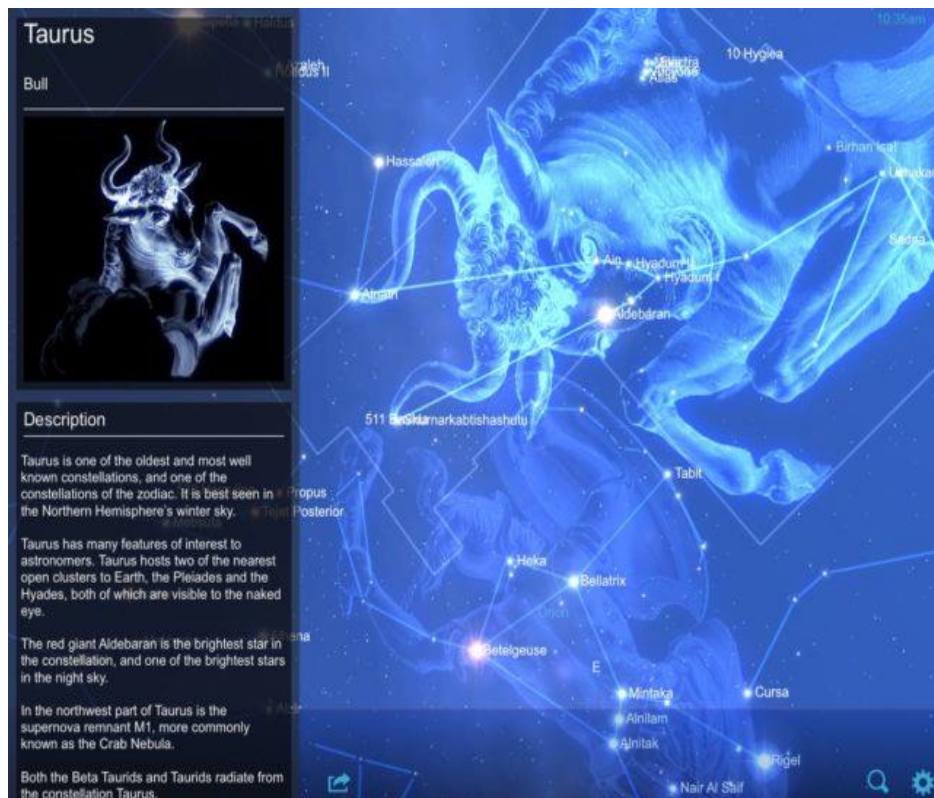


Рисунок 1.3 – Інтерфейс програми Star Chart

1.5 Огляд AR-бібліотек

Створюючи програми доповненої реальності, розробники можуть спростити написання програми за допомогою спеціальних AR-бібліотек з відкритим API. За функціоналом AR-бібліотеки знаходяться приблизно на одному рівні, разом з тим, кожна з них має свої характерні риси. Нижче наведено плюси та мінуси бібліотек доповненої реальності.

1.5.1 Vuforia

Будучи повноцінною SDK для розробників, Vuforia є набором інструментів для створення додатків доповненої реальності.

Vuforia підтримує:

- розпізнавання кількох цілей одночасно (включаючи об'єкти, зображення та текст);

- відстеження цілей;
- розпізнавання 2D та 3D форматів;
- сканування реального об'єкта для подальшого розпізнавання;
- віртуальні кнопки;
- Smart Terrain™ – можливість реконструювати навколишній ландшафт, створюючи його 3D-карту;
- Extended Tracking – можливість продовжити відображення мети на мобільному пристрої, навіть коли вона знаходиться поза увагою.

Зокрема, при розпізнаванні зображень Vuforia дозволяє мобільним програмам використовувати дані, що знаходяться або на пристрої, або у хмарі. До основних переваг бібліотеки можна віднести підтримку пристроїв віртуальної реальності, а також тестовий додаток з коментарями, що супроводжують, в якому показано можливості бібліотеки. Однак відсутність повноцінного посібника з використання бібліотеки ускладнює перший досвід роботи з Vuforia. Окремі інструкції та стислі поради представлені у великій кількості, але не впорядковані і тому не замінюють розробнику необхідну документацію. У безкоштовному варіанті бібліотеки є обмеження щодо використання cloud recognition, а також один раз на день з'являється водяний знак компанії [2].

1.5.2 ARToolKit

ARToolKit – це набір програмних бібліотек, які можуть використовуватися в AR додатках. Головна перевага бібліотеки – відкритий вихідний код. Тобто, бібліотека розповсюджується абсолютно безкоштовно.

До основних можливостей ARToolKit відносяться:

- розпізнавання 2D формату;
- відображає додатки через OpenGL.

Бібліотека призначена для відстеження у кадрі камери мобільного пристрою заздалегідь відомих квадратних маркерів об'єктів та відтворення на

екрані розташування в просторі. За допомогою цих даних створюється інтерфейс доповненої дійсності. ARToolKit для роботи на різних платформах: Android, iOS, Windows, Linux, Mac OS X, SGI. Для кожної конкретної операційної системи потрібне своє середовище розробки. Безкоштовні середовища доступні на всіх платформах. Незважаючи на безкоштовний доступ до бібліотеки, документація для розробників обмежена. Є тестові програми, але не всі з них вдається зібрати. Код прикладів представлений погано і немає інформації щодо подальшого розвитку бібліотеки.

1.5.3 Wikitude

Бібліотека Wikitude підтримує:

- розпізнавання 2D та 3D форматів (зображення, текст, відео);
- можливість сканування реального об'єкта для подальшого розпізнавання;
- рендеринг та анімацію 3D-моделей;
- відстеження розташування об'єктів.

За допомогою Wikitude можна створювати програми для відображення місць навколо користувача на віртуальній карті або у вигляді списку, для пошукових запитів про події, твіти, статті з Вікіпедії, для отримання рекомендацій від інших користувачів. Крім цього, програма на базі WikiTude дозволить отримувати мобільні купони, інформацію про вигідні пропозиції та знижки в магазинах навколо, а також грати в AR-ігри. Wikitude може використовуватися для платформ Android, iOS, як плагін для PhoneGap, модульний елемент для Titanium та компонент для Xamarin. Цей фреймворк підходить для розумних окулярів Google Glass, Epson Moverio, Vuzix M-100 та ODG R-7. Включає SLAM і підтримує Unity. Для розробників є безкоштовна пробна версія, в іншому використанні бібліотеки потребує фінансових вливань. Плюсом є представлена документація [2].

1.5.4 LayAR

Як випливає з назви фреймворку, за його допомогою можна дивитися на навколишнє оточення через «шари», що відображаються на екрані мобільного пристрою.

LayAR підтримує:

- розпізнавання зображень;
- відображення доповнень на основі положення користувача та розпізнаних зображень.

Кожен із шарів LayAR може містити інформацію про розташування окремих географічних місць або користувачів соціальних мереж. Окрім цього, використання функціоналу цієї бібліотеки у додатку розширює можливості друкованої продукції. Наприклад, можна оформити покупку в каталозі або прослухати музичну композицію, яку написали в журналі. Вся робота з бібліотекою відбувається на сервері через JSON, у тому числі логіка відображення додаткових елементів під час розпізнавання. Через те, що вся конфігурація відбувається на сервері, робота з фреймворком здійснюється не дуже гнучко. Великим плюсом LayAR є докладна документація. Однак, керівництво доступне лише в онлайн-режимі.

1.5.5 Kudan AR

Функціонал бібліотеки Kudan включає:

- розпізнавання зображень;
- відображення доповнень на основі положення користувача та розпізнаних зображень;
- безмаркерне відстеження об'єктів (без реперних точок, лише з урахуванням реальних характеристик об'єкта – кутів, згинів чи текстур);
- відображення доповнень через окремий компонент-обгортку над OpenGL.

Kudan відрізняється від інших фреймворків швидкістю дії, дозволяє програмам відображати мультиполігональні моделі в реальному світі та імпортувати 3D моделі безпосередньо з популярних програм з моделювання та анімації. До додаткових плюсів можна віднести відсутність обмежень на кількість зображень, що розпізнаються, і невеликий обсяг пам'яті, необхідний для зберігання файлів на девайсі. Розробники можуть скористатися базовою документацією при зверненні до бібліотеки, проте посібник із використання не надто докладний і вимагає пошуку додаткової інформації. Також є ризик, що при створенні програми може вистачити вбудованих можливостей бібліотеки, а доступ безпосередньо до OpenGL відсутній. Порівняння AR-фреймворків наведено у (табл. 1.1).

Загалом, перераховані бібліотеки доповненої реальності надають широкий спектр можливостей розробнику – від різних операційних систем до розгорнутого набору інструментів для розпізнавання та відстеження об'єктів. Проте, зупинившись перед вибором конкретного фреймворку, розробнику важливо розуміти, що він отримає своє розпорядження. Частина інструментів можна використовувати безкоштовно, просто зайшовши на сайт або завантаживши невелику програму. Інші вимагають укладання партнерських відносин та регулярної плати, але при цьому забезпечують більш розгорнутий та якісний функціонал. Роблячи вибір на користь певної AR-бібліотеки, насамперед, відштовхуйтеся від завдань свого проєкту, від запланованих результатів, та порівнюйте їх із можливостями обраних рішень [2].

Таблиця 1.1 – Порівняння AR-фреймворків

	Vuforia	ARToolKit	Wikitude	Lay AR	Kudan AR	ARKit
Розпізнавання кількох цілей одночасно	+	-	+	-	+	+
Відстеження цілей	+	+	+	+	+	+

Продовження таблиці 1.1

	Vuforia	ARToolKit	Wikitude	Lay AR	Kudan AR	ARKit
Розпізнавання 2D форматів	+	+	+	+	+	+
OpenGL	-	+	-	-	-	+
Сканування реального об'єкта	+	+	+	+	+	+
Розпізнавання 3D форматів	+	-	+	+	-	+
Cloud recognition	+	-	-	-	+	+
Швидкодія	повільна	швидка	швидка	повільна	повільна	швидка

1.6 Стандартні методи позиціонування в доповненій реальності

Очевидно, що однією з найважливіших проблем впровадження доповненої реальності є «зчеплення» віртуальних об'єктів з фізичними, які користувач бачить крізь камеру смартфона. Розглянемо способи позиціонування AR-об'єктів на прикладі стандартних методів фреймворку ARKit.

У фреймворку ARKit передбачено два основні методи позиціонування: hitTest та worldTransform.

hitTest шукає реальні об'єкти або поверхні, виявлені при обробці зображення камери під час сеансу AR. Двовимірна точка в координатах зображення може посилатися на будь-яку точку вздовж 3D-лінії, яка починається від камери пристрою і тягнеться в напрямку, визначеному орієнтацією пристрою та проекцією камери. Цей метод здійснює пошук уздовж цієї лінії, повертаючи всі об'єкти, які перетинають її, у порядку відстані від камери. Схема роботи hitTest (рис.1.4).

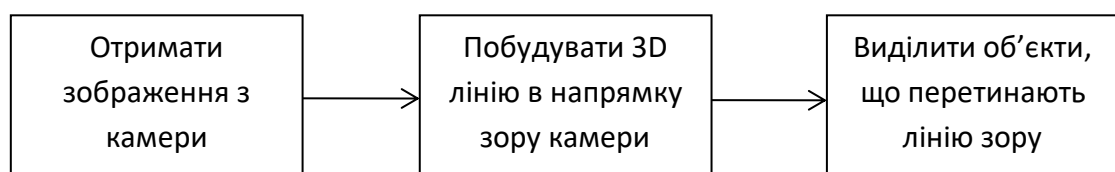


Рисунок 1.4 – Схема роботи hitTest

worldTransform вказує точку перетину між виявленою поверхнею та променем, який створив результат hitTest. hitTest створює двовимірну точку на зображенні або в системі координат перегляду вздовж променя в тривимірний простір світу і повідомляє про результати, де ця лінія перетинає виявлені поверхні. Схема роботи worldTransform (рис.1.5).

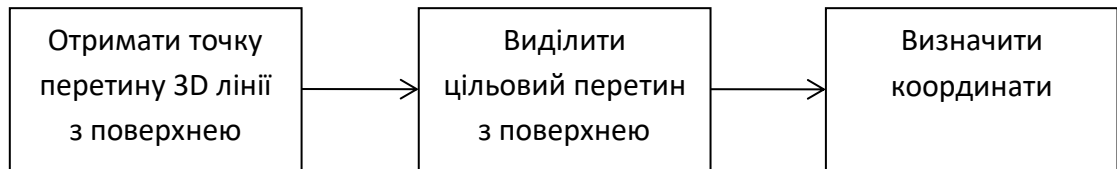


Рисунок 1.5 – Схема роботи worldTransform

Ці методи є широко вживаними, оскільки вони вже включені в склад бібліотеки, але є певні недоліки їх використання, особливо при застосуванні кінцевого продукту в локаціях зі складним рельєфом.

1.7 Висновки за розділом 1 та постановка завдання роботи

В першому розділі було розглянуто найпопулярніші застосунки, які базуються на доповненій реальності, і які є націленими на навчання. Було розглянуто їх переваги та недоліки, а також було доведено актуальність цієї теми.

У зв'язку з масовим інтересом до мобільних застосунків з елементами AR, що з'явився після виходу культової гри Pokemon Go, розробка навчального проєкту з використанням технологій доповненої реальності є актуальною задачею. Також прийняте рішення при розробці дослідити можливість удосконалення методів позиціонування AR-об'єктів.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз технологій розробки

Доповнена реальність (AR) описує досвід користувача, який додає 2D- або 3D-елементи до живого перегляду з камери пристрою таким чином, щоб ці елементи виглядали як реальний світ. ARKit поєднує в собі відстеження руху пристрою, зйомку сцени камерою, розширену обробку сцени та зручності відображення, щоб спростити завдання створення доповненої реальності. Ви можете створювати багато видів доповненої реальності за допомогою цих технологій за допомогою передньої або задньої камери пристрою iOS [2].

Фреймворк ARKit був обраний для створення мобільного за стосунку, тому що він є новим, достатньо функціональним та надає можливість збільшити інтерес користувача та покращити підхід до вивчення астрономії.

Мова програмування Swift – багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta [3].

Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас, відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C [4].

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і

словникові типи, швидкі операції над колекціями, елементи функційного програмування. Основним застосуванням Swift є розробка користувацьких додатків для macOS, iOS, tvOS, watchOS з використанням класів Cocoa і Cocoa Touch.

Мова програмування Swift обрана для створення мобільного додатку для iOS пристрою, тому що вона найновіша серед можливих для використання, і має набагато більше можливостей ніж стара Objective-C.

Xcode – інтегроване середовище розробки (IDE) виробництва Apple.

Xcode включає в себе більшу частину документації розробника від Apple та Interface Builder – застосунок, який використовується для створення графічних інтерфейсів [6].

Xcode містить мови C, C++, Swift, Java, AppleScript, Python і Ruby з різними моделями програмування, включаючи Cocoa, Carbon і Java. Сторонніми розробниками реалізована підтримка GNU Pascal, Free Pascal, Ada, C #, Perl, Haskell і D. Пакет Xcode використовує GDB як back-end для свого налагоджувача.

Середовище розробки Xcode було обране для створення додатку, тому що воно є найбільш актуальним і комфортним.

2.2 Розробка структури програми

Логічна структура системи була розроблена відповідно до поставленого завдання і являє собою багаторівневу структуру, що складається з вікон додатку (рис.2.1).

Модульний підхід до проєктування програмного забезпечення дозволяє розробнику краще орієнтуватися в своїй системі, а також зробити більш ефективним процес її написання та налагодження.

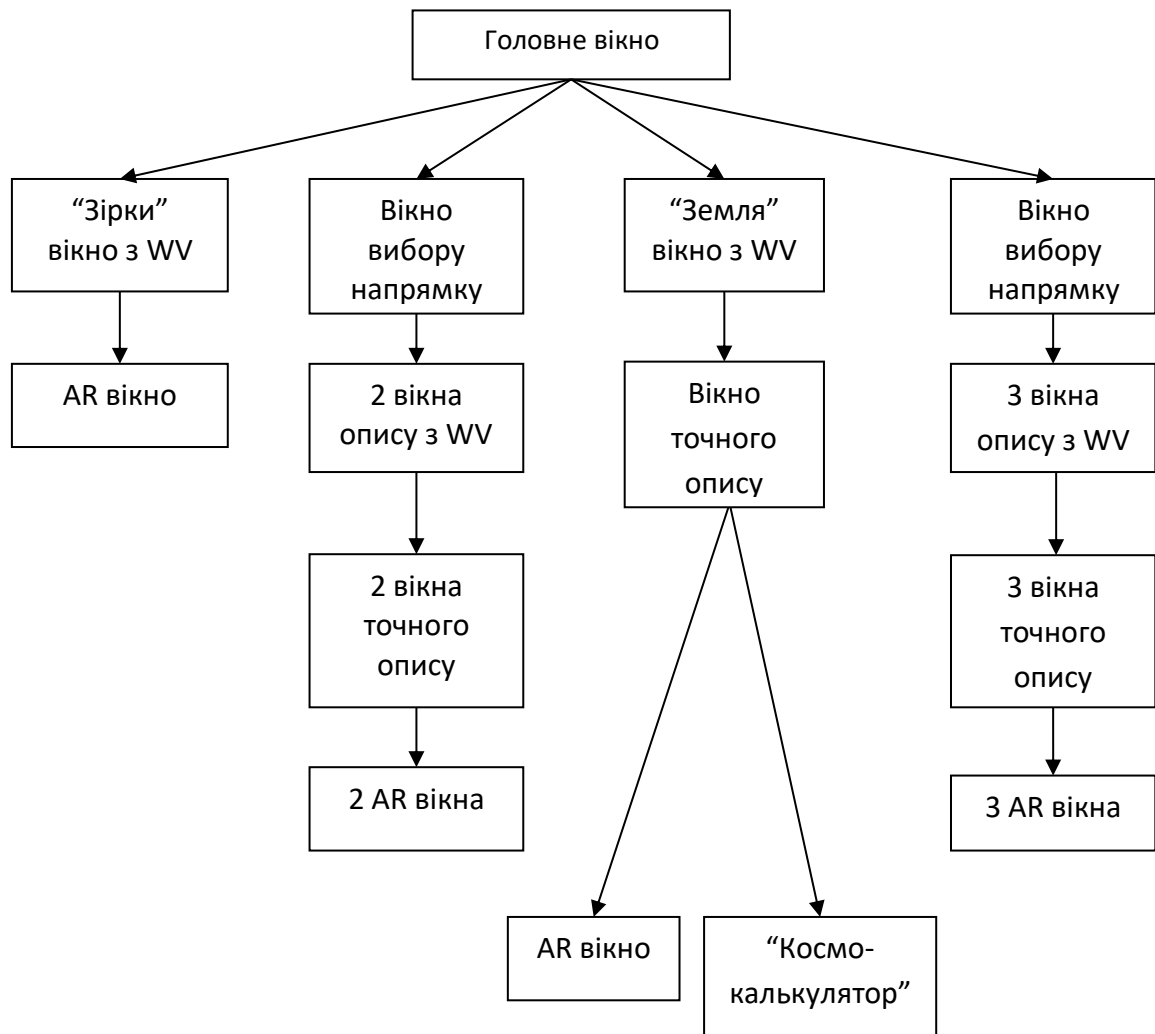


Рисунок 2.1 – Логічна структурна схема програми

Головним модулем (об'єктом) проєктованої системи є модуль головного вікна, який грає роль диспетчера, тобто визначає залежно від обраного користувачем пункту меню, яку процедуру необхідно виконати, або на яке вікно перейти [7].

До підлеглих модулів відносяться:

- Зірки();
- Сонце/місяць();
- Сонце();
- Місяць();
- Земля();
- Планети();

- Марс();
- Венера();
- Юпітер();
- елементТочногоОпису1();
- елементТочногоОпису2();
- елементТочногоОпису3();
- елементТочногоОпису4();
- елементТочногоОпису5();
- елементТочногоОпису6();
- елементAR1();
- елементAR2();
- елементAR3();
- елементAR4();
- елементAR5();
- елементAR6();
- елементSwiftUI().

У додатку необхідно передбачити наявність навчальних відео та інформації, призначених для користувача, оскільки користувач може дізнаватися нове про космічні тіла виконання саме з інформативного розділу додатку.

До цього розділу можна віднести, наприклад, інформацію та відео по певному небесному тілу.

Інформація додається зі стороннього джерела, а підпис до відео (інструкція) – програмістом, і розділяється за такими напрямками:

- Зірки (загальна інформація);
- Сонце;
- Місяць;
- Земля;
- Марс;
- Венера;

– Юпітер.

Вихідна інформація представляється у вигляді відео з підписом, що виводяться на екран.

Структурна схема програми інформативного вікна наведена на (рис. 2.2).

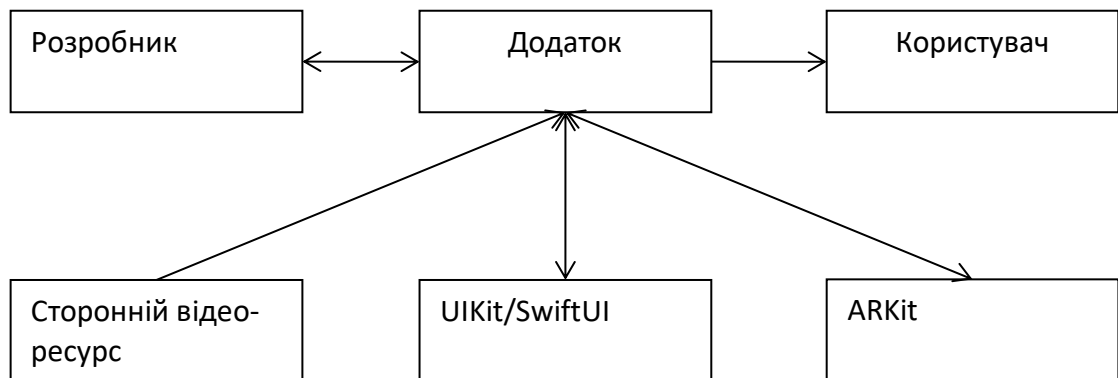


Рисунок 2.2 – Структурна схема інформативного вікна

У додатку, окрім інформативних розділів, є вікно, що дозволяє користувачеві розрахувати, за яку кількість днів можна дістатись до обраних небесних тіл з вказаною швидкістю (кастомний Космо-калькулятор). Він працює таким чином, що людина вказує швидкість, обирає небесне тіло і калькулятор показує точний час польоту (у днях).

2.3 Проєктування системи

Проєктування системи включає кілька етапів:

- технічне проєктування;
- художнє проєктування;
- ергономічне проєктування.

Технічне проєктування полягає в машинній розробці системи. Більш детальний опис модулів, які розроблені на цьому етапі представлено нижче:

- Mian.storyboard – інтерфейс головного вікна програми;

- ViewController.swift – зберігає логіку роботи модулю Mian.storyboard, функціонал головного вікна програми;
- LaunchScreen.storyboard – інтерфейс вікна запуску;
- StarScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- MoonSunScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- SunScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- MoonScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- EarthScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- PlanetsScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- MarsScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- VenusScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- JupiterScreen.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- SunMore.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- MoonMore.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- EarthMore.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- MarsMore.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;

- VenusMore.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- JupiterMore.swift – зберігає логіку роботи фрагменту модулю Mian.storyboard, функціонал вікна програми;
- StarAR.swift – реалізує підключення фреймворку ARKit та виконання його функцій в проєкті;
- MoonAR.swift – реалізує підключення фреймворку ARKit та виконання його функцій у проєкті;
- SunAR.swift – реалізує підключення фреймворку ARKit та виконання його функцій в проєкті;
- EarthAR.swift – реалізує підключення фреймворку ARKit та виконання його функцій в проєкті;
- MarsAR.swift – реалізує підключення фреймворку ARKit та виконання його функцій в проєкті;
- VenusAR.swift – реалізує підключення фреймворку ARKit та виконання його функцій в проєкті;
- JupiterAR.swift – реалізує підключення фреймворку ARKit та виконання його функцій в проєкті;
- SwiftUIKit.swift – реалізує під’єднання модулю SwiftUIView.swift;
- SwiftUIView.swift – зберігає логіку роботи фрагменту модулю Main.storyboard, реалізує підключення фреймворку SwiftUI;

Суть вікна Mian.storyboard полягає в управлінні переходами між вікнами програми (табл. 2.1).

Таблиця 2.1 – Структура вікна Mian.storyboard

Назва вікна	Напрямок	Кількість вікон
Main.storyboard	навігація	4
Stars.storyboard	напрямок	2

Продовження таблиці 2.1

Назва вікна	Напрямок	Кількість вікон
Earth.storyboard	напрямок	4
SunMoon.storyboard	напрямок	7
Planets.storyboard	напрямок	10
SwiftUIView.swift	калькулятор	1

Суть вікна Stars.storyboard полягає в наданні інформації і порад користувачеві. Вікно StarScreen.swift відповідає за надання інформації по напрямку.

Суть вікна Earth.storyboard полягає в наданні інформації і порад користувачеві. Вікно EarthScreen.swift відповідає за надання інформації по напрямку.

Суть вікна SunMoon.storyboard полягає в наданні інформації і порад користувачеві. Вікна MoonScreen.swift та SunScreen.swift відповідає за надання інформації по напрямках.

Суть вікна Planets.storyboard полягає в наданні інформації і порад користувачеві. Вікна MarsScreen.swift, VenusScreen.swift та JupiterScreen.swift відповідає за надання інформації по напрямках.

Структура основної частини додатку, зробленого у середовищі розробки Xcode, представлена на (рис. 2.3 – 2.4).

2.4 Висновки за розділом 2

У другому розділі запропоновано рішення щодо структури програмного забезпечення вивчення астрономії.

Розроблено структурну схему програми. Створено зв'язок зі стороннім відео-ресурсом. Додані описи і детальні описи певних небесних тіл. Створено кастомний Космо-калькулятор за допомогою фреймворку SwiftUI. Створені моделі космічних тіл та підключено фреймворк ARKit.

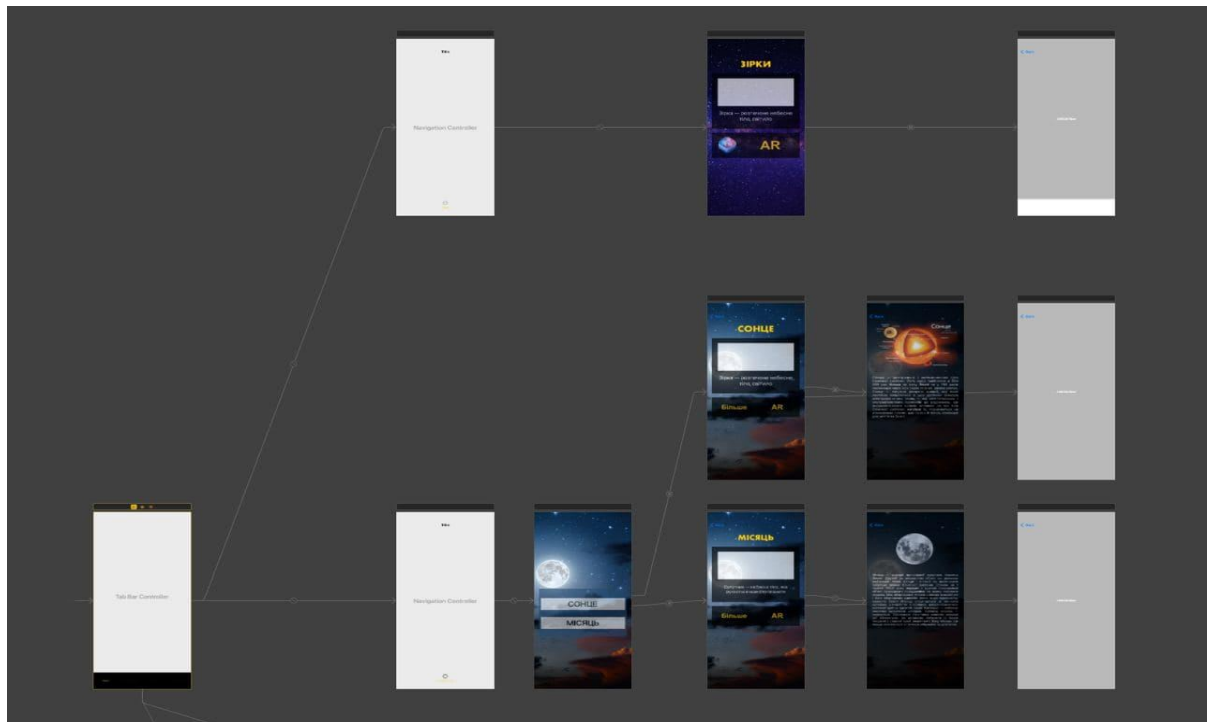


Рисунок 2.3 – Структура основної частини додатку

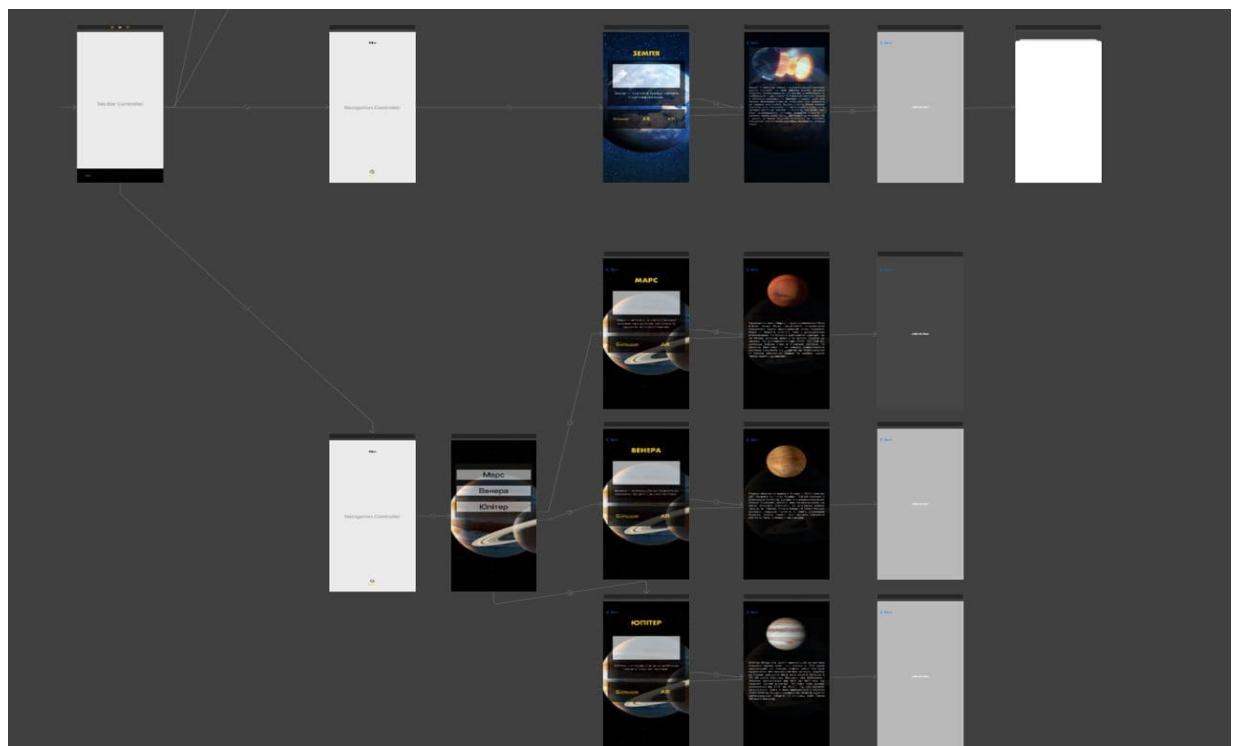


Рисунок 2.4 – Структура основної частини додатку

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Впровадження технологій ARKit

ARKit має зрозумілий та комфортний інтерфейс, працює з пристроями на iOS 11 та новіше, з процесором A9 та новіше [1].

Шари ARKit: Tracking, Scene Understanding, Rendering:

- Tracking – можливість відстежувати зміни, які відбуваються з самим пристроєм;
- Scene Understanding – розуміння того, що відображується через камеру (відстань до об'єкту, освітлення...);
- Rendering – робота з даними та підготовка їх до відображення.

Більш детально про шар Tracking. Він складається з 3 частин:

- World Tracking – дозволяє відстежувати положення пристрою у реальному світі із реальним часом. (не тільки орієнтацію пристрою);
- Visual Inertial Odometry – метод оцінки положення та орієнтації пристрою за допомогою аналізу послідовних зображень, знятих встановленою (звичайною) камерою;
- No External setup – дозволяє користуватись доповненою реальністю без встановлення стороннього ПЗ.

Більш детально про шар Scene Understanding. Він складається з 3 частин:

- Plane Detection – визначає поверхню для розміщення моделі;
- Hit-testing – порівнює координати моделі та координати реального світу. (проводить тестування);
- Light Estimation – використовується для оцінки освітлення.

Більш детально про шар Rendering. Він складається з 3 частин:

- Easy Integration – забезпечує просту інтеграцію у візуалізатор;
- AR Views – забезпечує змалювання об'єкту всередині себе заздалегідь;

– Custom rendering – надає можливість працювати з кастомним рендерингом прямо в середовищі розробки Xcode.

ARKit поєднує роботу двох фреймворків: AVFoundation, CoreMotion [1].

Вони забезпечують захват зображення та отримання даних з сенсорів пристрою.

У Xcode можливість працювати з ARKit реалізована за допомогою елемента проєктування SceneView (рис. 3.1).

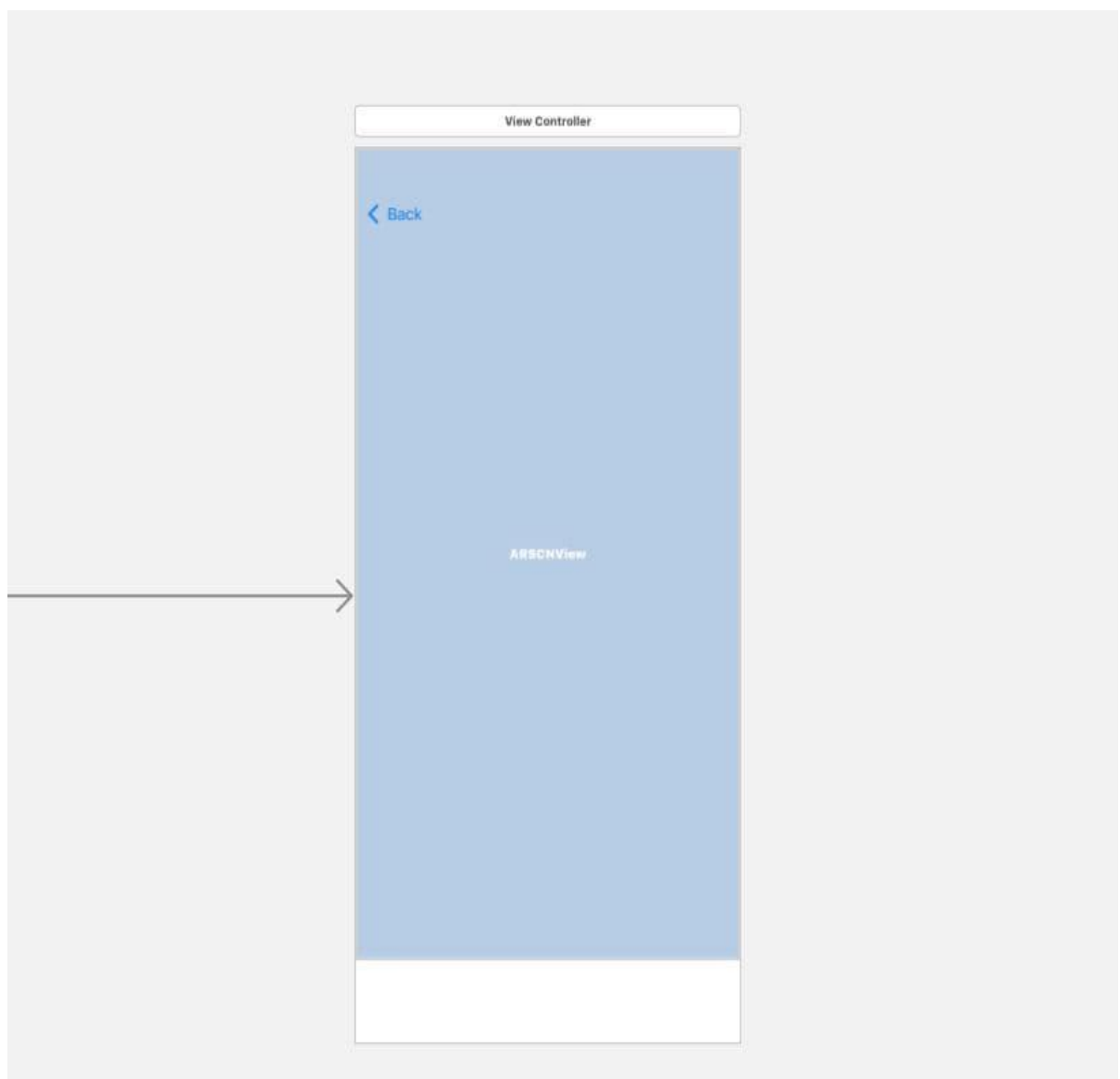


Рисунок 3.1 – ARSCNView

У ARKit є можливість відобразити стан пристрою під час роботи з ARSession за допомогою такої строки коду (рис. 3.2).

```
// Show statistics such as fps and timing information  
sceneView.showsStatistics = false
```

Рисунок 3.2 – відображення стану пристрою

У користувацькому додатку така функція не потрібна, тому вона вимкнена.

Також створення кастомної сцени реалізоване за допомогою наступних рядків коду (рис. 3.3).

```
// Create a new scene  
let scene = SCNScene()  
  
// Set the scene to the view  
sceneView.scene = scene
```

Рисунок 3.3 – створення кастомної сцени

ARWorldTrackingConfiguration забезпечує три вісі обертання та три вісі переміщення, саме завдяки цьому, віртуальні об'єкти залишаються ніби зафіксованими в реальному світі. (рис. 3.4).

```
// Create a session configuration  
let configuration = ARWorldTrackingConfiguration()  
  
// Run the view's session  
sceneView.session.run(configuration)
```

Рисунок 3.4 – ARWorldTrackingConfiguration

Нанесення текстур на наш віртуальний об'єкт виконується таким чином (рис. 3.5).

```
func createEarth(position: SCNVector3) {
    let earthShape = SCNSphere(radius: 0.13)
    let earthNode = SCNNode.init(geometry: earthShape)
    earthNode.position = position
    earthNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named: "8k_earth_daymap")
    earthNode.geometry?.firstMaterial?.specular.contents = UIImage(named: "8k_earth_specular_map")
    earthNode.geometry?.firstMaterial?.emission.contents = UIImage(named: "8k_earth_clouds")
    earthNode.geometry?.firstMaterial?.normal.contents = UIImage(named: "8k_earth_normal_map")
    EarthAR.scene.rootNode.addChildNode(earthNode)
}
```

Рисунок 3.5 – Нанесення текстур на об'єкт

3.2 Реалізація основних функцій системи

Розроблений проєкт складається з 23 частин:

- головний екран – необхідний для того, щоб реалізувати можливість навігації у додатку (4 напрямки);
- екран напрямку зорі – частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш інформативно-наповненого в'ю та до ARSCNView;
- екран напрямку сонце/місяць – частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш інформативно-наповненого в'ю та до ARSCNView;
- екран напрямку земля – частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш інформативно-наповненого в'ю, до ARSCNView та до “Космо-калькулятору”;
- екран напрямку планети – частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш інформативно-наповненого в'ю та до ARSCNView.

Основна частина навігації була виконана за допомогою таких елементів як Tab Bar Controller (рис. 3.6) та Navigation Controller, але в програмному продукті були використані і такі функції навігації:

- Relationship Segue – надає можливість зв'язування екранів для структуризації програми;
- Storyboard Reference – надає можливість роз'єднання екранів розробки;
- Navigation Item – одиниця навігації [8];
- Button – кнопка;
- Label – текстове поле [9];
- View – надає можливість додавати зображення та відео;
- Safe Area – надає можливість використовувати цілий екран [10].



Рисунок 3.6 – Tab Bar Controller

Елемент Tab Bar Controller використовується для створення 4 різних вікон з одним спільним навігаційним контролером (рис. 3.7).

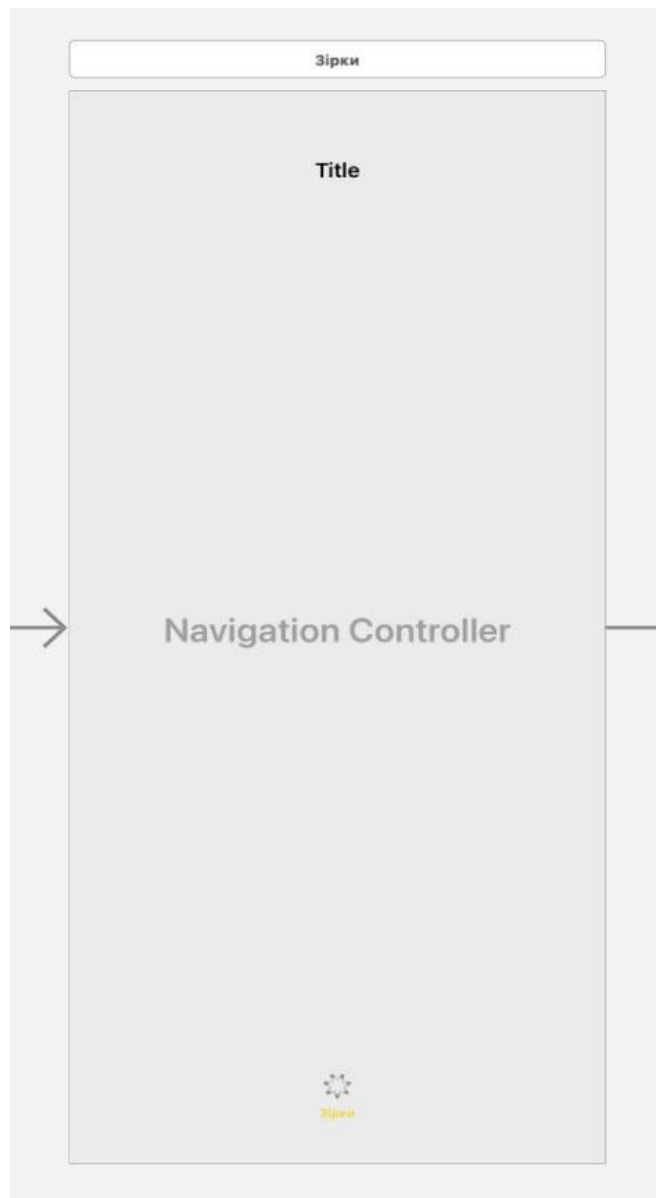


Рисунок 3.7 – Navigation Controller

У додатку було задіяно один елемент Tab Bar Controller для головного вікна. Задіяно 4 елементи Navigation Controller, 25 елементів Relationship Segue та створено 19 кнопок, для переходів між вікнами застосунку. За допомогою елементу Relationship Segue створено зворотні переходи у додатку.

Однією з переваг використання елементу Tab Bar Controller є запам'ятовування додатком переходів за різними напрямками, тобто переходячи в інший напрямок у додатку, ОП зберігає те, що було досягнуто у попередньому екрані [11].

Для створення інформативних екранів, окрім кнопок та текстових полів, були використані елементи View (рис. 3.8) та Webkit бібліотека (в коді) [12].

Було створено функцію типу viewDidLoad (виконання під час відкриття екрану), в якій за допомогою відео-коду буде прив'язане до View посилання на відео зі стороннього відео-ресурсу.

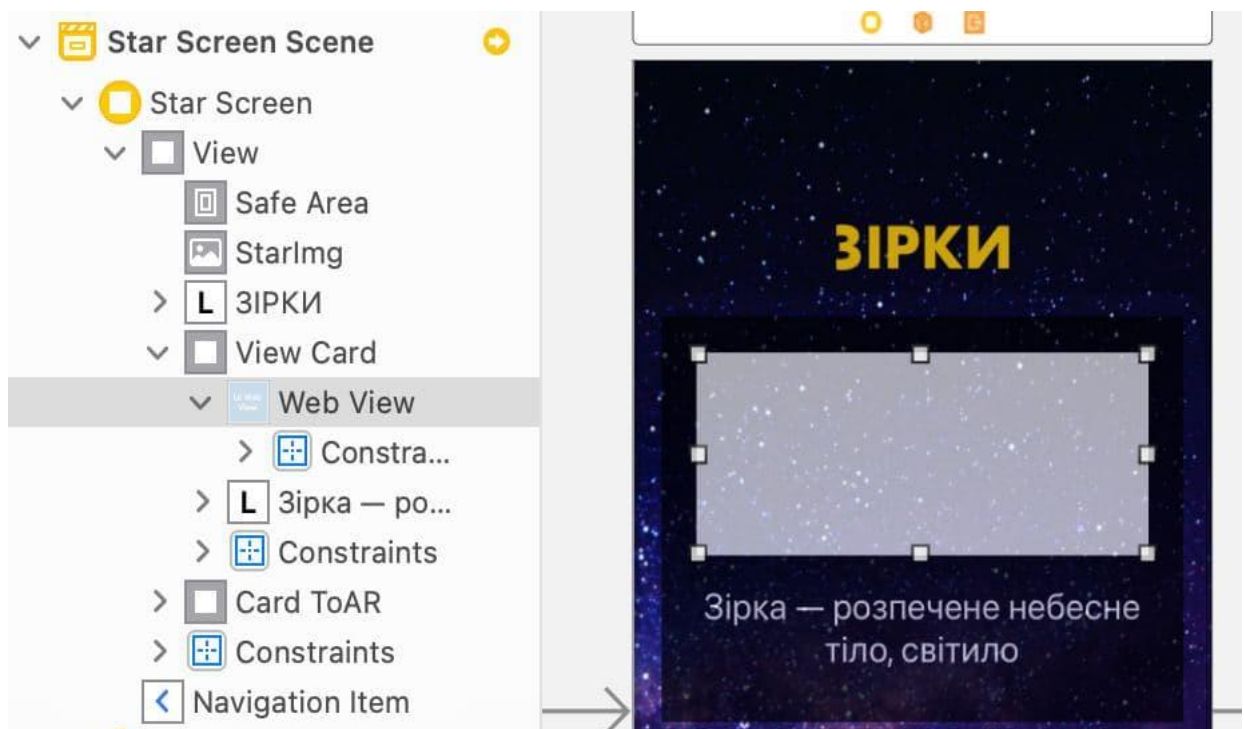


Рисунок 3.8 – View

Реалізація вікна Космо-калькулятор була виконана в коді за допомогою фреймворку SwiftUI. Логікою роботи кастомного калькулятора є виконання розрахунку кількості днів, котрі треба подолати до обраного небесного тіла зі швидкістю, вказаною користувачем.

Для роботи з великою кількістю екранів було створено багато класів типу Cocoa Touch Class, після чого файлова структура додатку стала виглядати таким чином (рис. 3.9).



Рисунок 3.9 – Файлова структура

Завдяки комфортності та функціоналу середовища вдалося додати актуальний рисунок на екран завантаження (рис. 3.10).

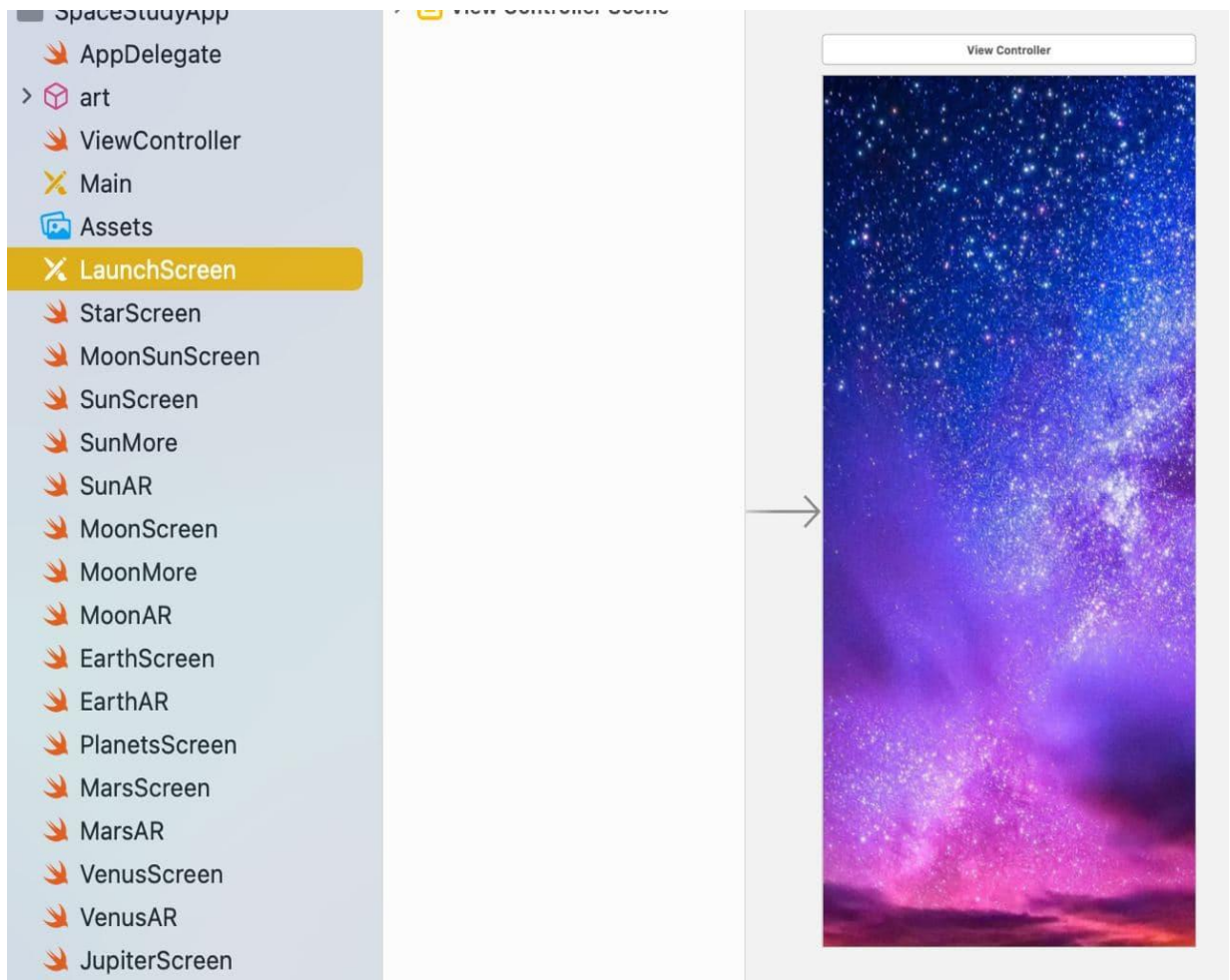


Рисунок 3.10 – Вікно завантаження

3.3 Метод touchesBegan

У першому розділі були описані стандартні методи `hitTest` та `worldTransform`, які використовуються для позиціювання AR-моделей. Дані методи не забезпечують достатньої точності у розміщенні та відображенні моделі. Для покращення роботи цих методів було вирішено комбінувати їх та доповнити тривимірною матрицею у кастомному методі. Було створено метод `touchesBegan`, який у собі використовує стандартні методи: `hitTest` – для створення точки на екрані смартфона у місці дотику користувача, `worldTransform` – для аналізу поверхні навколишнього середовища. За допомогою використання тривимірної матриці для позиціонування, вдалось встановити більш точне визначення місця розташування AR-моделі, під час її розміщення [1].

При додаванні моделі відбувається аналіз поверхні і завдяки методу touchesBegan елемент розташовується не на самій поверхні, а перед нею на невеликій відстані, пропорційній розміру об'єкту.

Порівняємо роботу програми без кастомного методу touchesBegan та з ним (табл. 3.1).

Таблиця 3.1 – Порівняння методів позиціонування

Показники	touchesBegan	Стандартні методи
Точність розташування моделі	Покращена 7-11%	Достатня
Вплив на модель	++ визначаються 3D координати та розраховується додаткове позиціонування	— тільки визначаються 3D координати
Візуальне сприйняття	++ Візуально AR-елемент розташовується перед зазначеною поверхнею	— Візуально AR-елемент розташовується на зазначеній поверхні

Для переміщення об'єкту на сцену потрібно визначити метод touchesBegan. Спочатку слід створити масив дотиків. Далі за допомогою стандартного методу hitTest (рис. 3.11) програма визначає точку на площині віртуального світу. Таким чином, ARKit знаходить місце розташування віртуального об'єкту за допомогою променя, який виходить із крапки дотику на екрані пристрою і направлений до віртуальної площини. На перетині променя з віртуальною площиною і створюється об'єкт (рис.3.12).

Створений метод підвищує точність позиціювання та відображення моделей із зазначеними в програмі розмірами на 7-11 відсотків, залежно від якості освітлення.

```

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let touch = touches.first else { return }
    let result = sceneView.hitTest(touch.location(in: sceneView), types: [.featurePoint])
    guard let hitResult = result.last else { return }
    let hitMatrix = hitResult.worldTransform
    let matrix = SCNMatrix4(hitMatrix)
    let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
    createBall(position: hitVector)
}

func createBall(position: SCNVector3) {
    let ballShape = SCNSphere(radius: 0.01)
    let ballNode = SCNNode.init(geometry: ballShape)
    ballNode.position = position
    sceneView.scene.rootNode.addChildNode(ballNode)
}

```

Рисунок 3.11 – Створення віртуального об'єкту

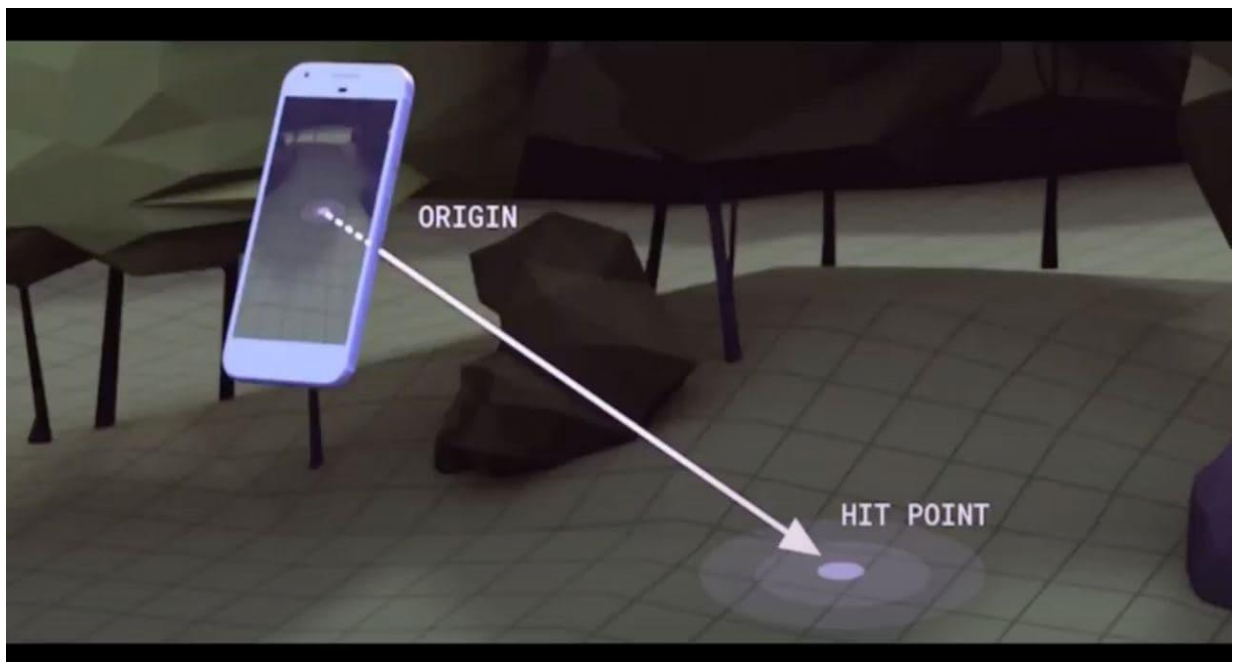


Рисунок 3.12 – Промінь hitTestу

Метод `tochesBegan` дозволяє покращити точність визначення координат під час розташування AR-об'єкту. Схема алгоритму визначення координат наведена на (рис. 3.13).

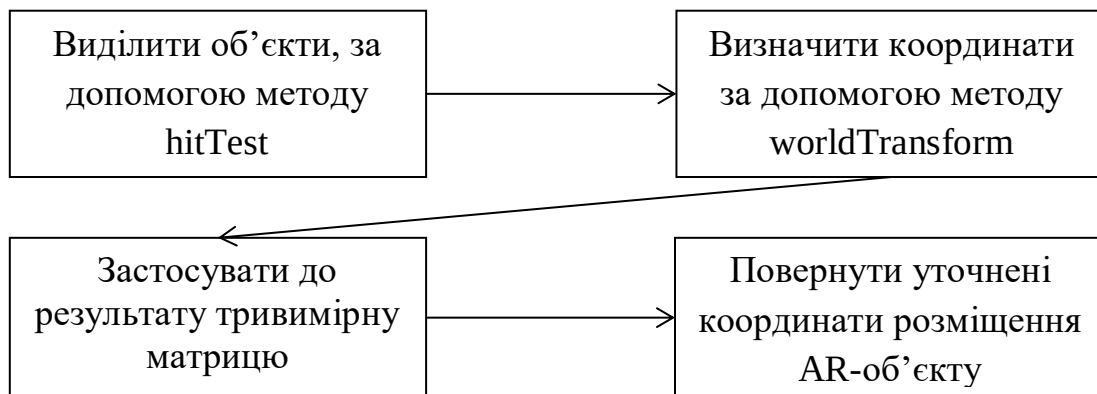


Рисунок 3.13 – Схема алгоритму визначення координат

3.4 Висновки за розділом 3

В ході проєктування архітектури системи було проаналізовано, описано та обґрунтовано основні рішення з реалізації компонентів мобільного застосунку. Окрім того, було описано основні модулі та функції розробленого програмного забезпечення. Було запропоновано метод `touchesBegan` для поліпшення процесу позиціювання моделей.

4 РОЗРОБКА ПРОГРАМИ

4.1 Опис основних етапів розробки

Мобільний застосунок або додаток – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях [5].

Для правильної роботи застосунку потрібно, щоб логіка була коректно заданою, архітектура – зрозуміло побудованою, фреймворки – правильно імпортованими та, звісно, код – безпомилковим. Мобільний додаток повинен створюватися правильною послідовністю дій, приклад такої послідовності представлений в цьому розділі.

4.2 Початок створення та налаштування додатку

Після відкриття середовища розробки Xcode ми обираємо тип застосунку App (рис. 4.1).

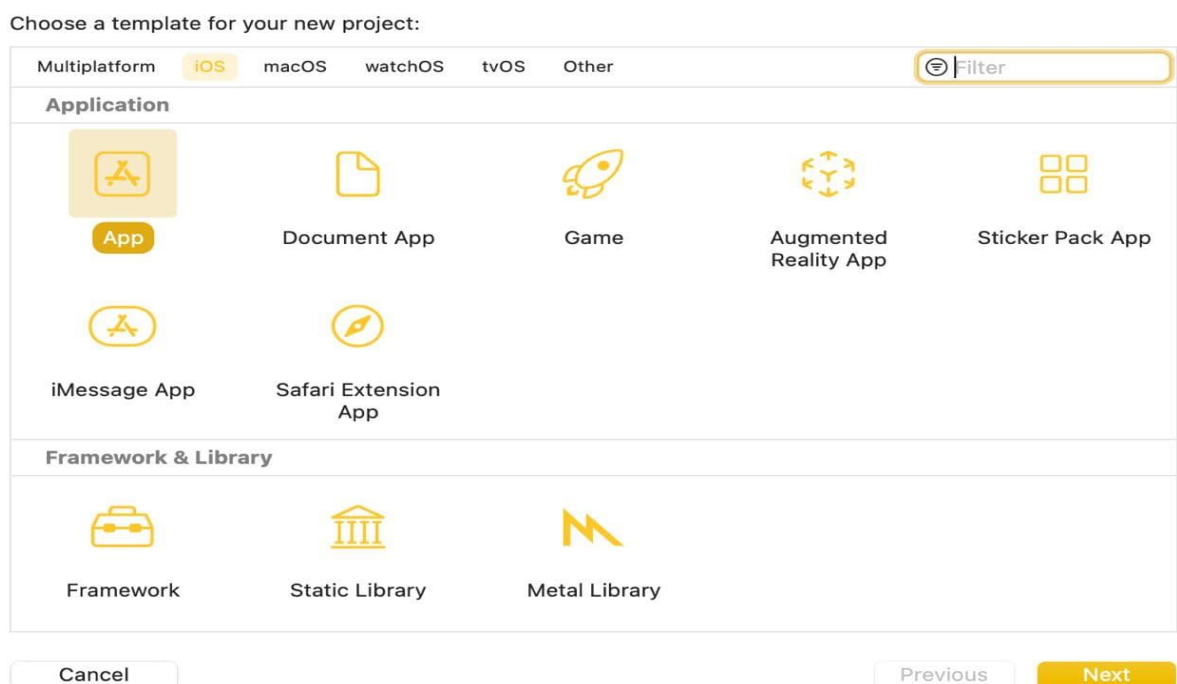


Рисунок 4.1 – Вікно вибору типу додатку

Другим кроком є надання імені та основних показників застосунку, таких як мова програмування, розробник, та тип UI (рис. 4.2).

Choose options for your new project:

Product Name:

Team: Данил Овчаренко (Personal Team)

Organization Identifier: com.OvcharenkoDan

Bundle Identifier: com.OvcharenkoDan.ProductName

Interface: SwiftUI

Language: Swift

☐ Use Core Data

☐ Host in CloudKit

☐ Include Tests

Cancel Previous Next

Рисунок 4.2 – Вікно створення додатку

4.3 Створення файлової системи та наповнення додатку

Створення файлової системи було висвітлено в 3 розділі. Вона складається з папок, фреймворків, класів, екранів проєктування, де створену архітектуру застосунку та папки Assets (рис. 4.3) у якій зберігаються елементи інтерфейсу.

4.4 Налаштування елементів проєктування

Щодо елементів проєктування, у середовищі Xcode можна додавати

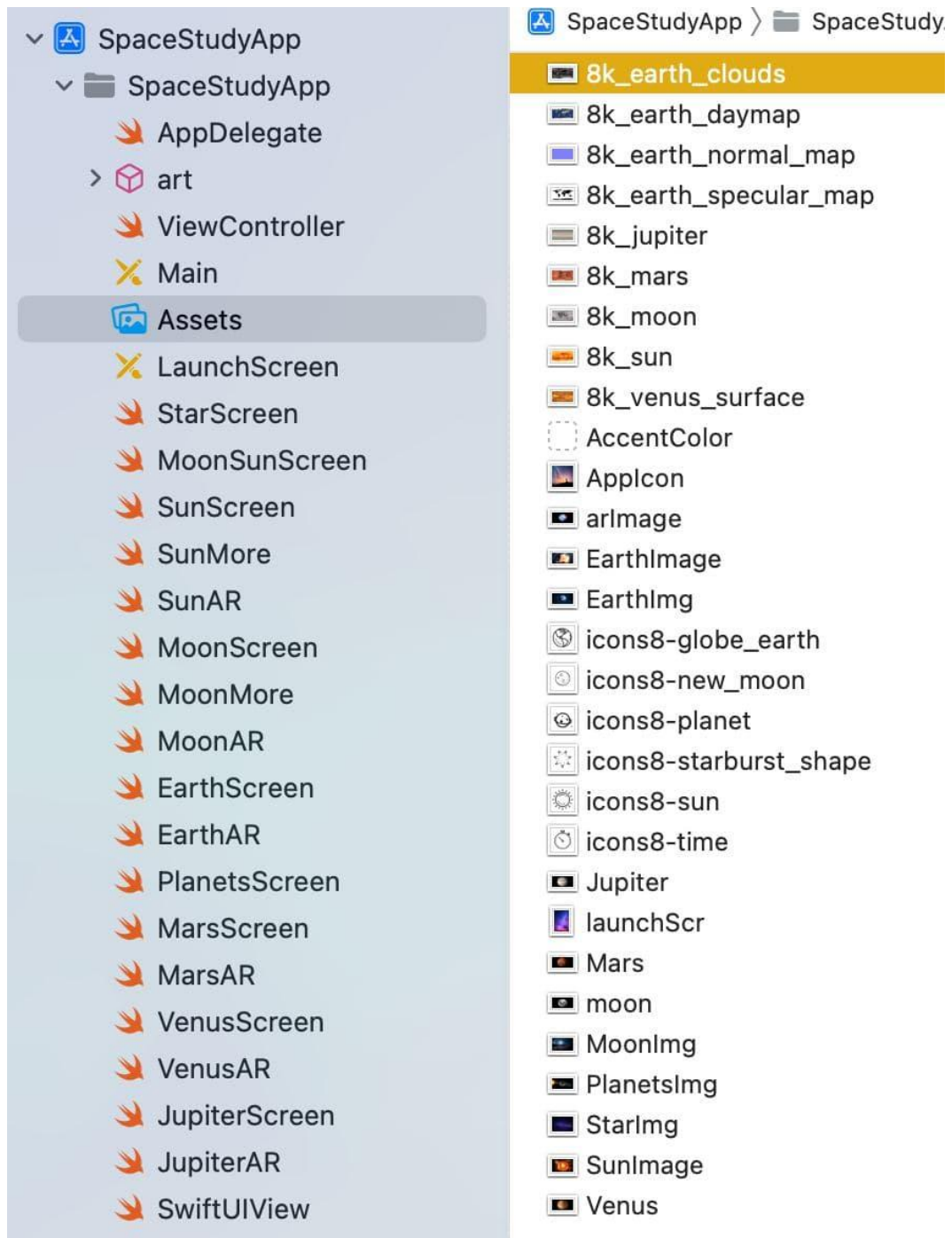


Рисунок 4.3 – Папка Assets з елементами UI

елементи, перетягуючи їх прямо на робочу область (рис.4.4), а також налаштовувати їх з коду та зі спеціальної вкладки в Xcode (рис.4.5).

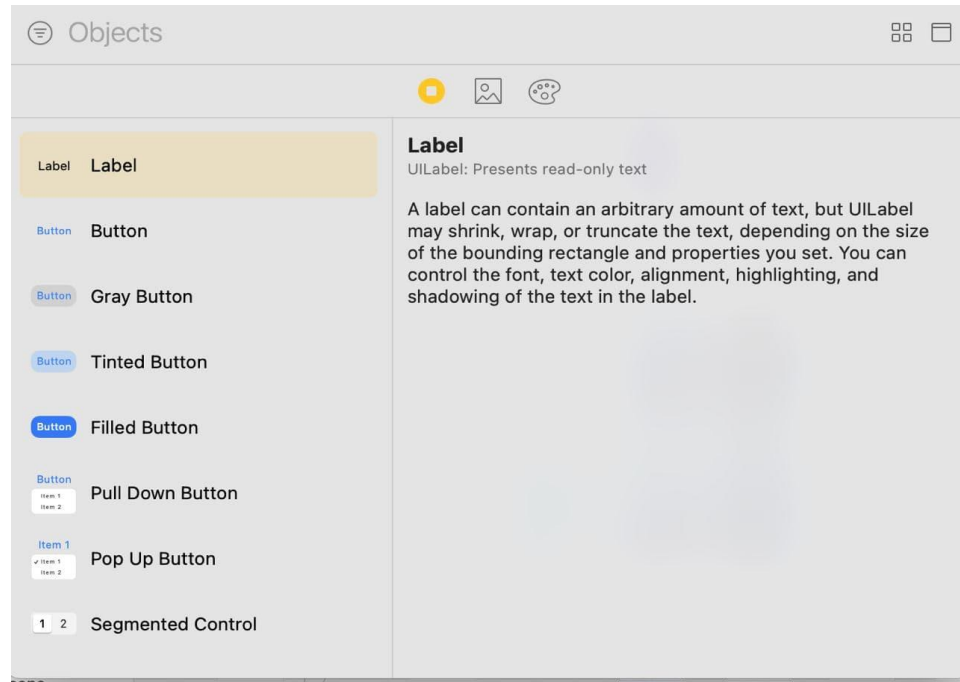


Рисунок 4.4 – Вибір додаваних елементів проєктування

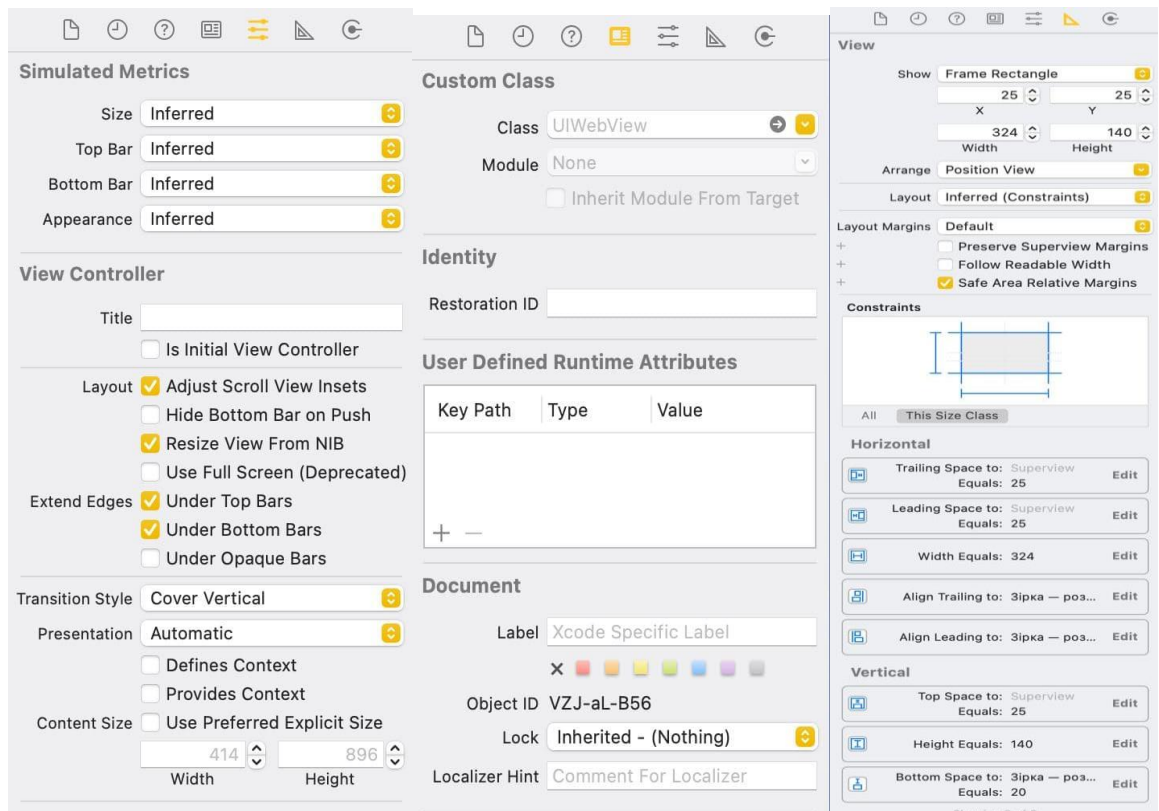


Рисунок 4.5 – Вкладки налаштувань елементів проєктування

Щодо ARKit, він підключається до проєкту методом `import ARKit`.

4.5 Додавання іконки додатку та SwiftUI

Додавання іконки застосунку відбувається в папці Assets (рис.4.6).

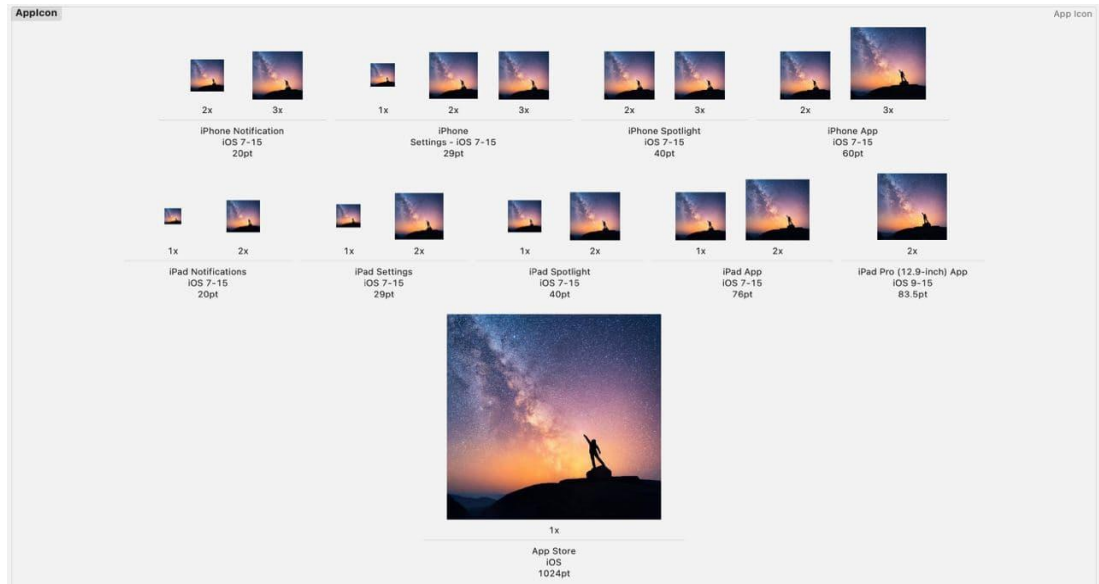


Рисунок 4.6 – Іконка застосунку

Також в проєкт був доданий фреймворк SwiftUI одним рядком коду (`import SwiftUI`). Завдяки ньому вдалося розробити користувач-орієнтований калькулятор космічних шляхів, який працює з використанням постійно-обчислюваних змінних завдяки property wrapper `@State` (рис. 4.7).

```

1  import SwiftUI
2
3  struct SwiftUIView: View {
4
5      @State private var showSheet = false
6
7      @State private var checkFast = ""
8      @State private var skybody = 0
9      @State private var planet = 0
10
11     let planets: [String] = ["Сонце", "Місяць", "Марс", "Венера", "Юпітер"]
12     let waySizes: [Double] = [150000000, 384400, 55760000, 38000000, 588000000]
13
14     var TotalDays: Double {
15         let planetWayLong = Double(waySizes[planet])
16         let orderFast = Double(checkFast) ?? 0
17
18         let tipValue = (planetWayLong / orderFast) / 24
19
20         return tipValue
21     }
22
23     var body: some View {

```

Рисунок 4.7 – Фрагмент коду з property wrapper `@State`

4.6 Висновки за розділом 4

Програмний продукт було створено на основі мови програмування Swift а також бібліотек ARKit, SwiftUI та UIKit. В якості функцій ARKit було взято можливість розташовувати об'єкти в реальному просторі. Завдяки цьому у користувача є можливість роздивитися небесні тіла з усіх сторін де б він не знаходився. Окрім цього було створено цікавий космо-калькулятор для учбових потреб користувача.

5 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

5.1 Характеристика, призначення та умови використання програми

Програма призначена для покращення підходу до вивчення астрономії.

Програма створена в середовищі програмування Xcode, на основі мови програмування Swift. Встановлення додатку на смартфон виконано за допомогою прийняття програм розробника у налаштуваннях телефону. Додаток створений з використанням принципів ООП.

5.2 Звертання до програми

Завантажувальний файл додатку має назву SpaceStudyApp. Додаток має назву SpaceStudy.app.

Програміст може змінити програму відкривши файл SpaceStudyApp за допомогою Xcode.

5.3 Опис основних функцій

Розроблений проєкт складається з 23 частин:

- головний екран — необхідний для того, щоб реалізувати можливість навігації у додатку за чотирма напрямками;
- екран напрямку Зірки — частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш вузького напрямку;
- екран напрямку Сонце/Місяць — частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш вузького напрямку;
- екран напрямку Земля — частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш вузького напрямку;

- екран напрямку Планети – частина, в якій реалізовано навігацію за даним напрямком, відображає кнопки переходів до більш вузького напрямку;
- екрани вузького (конкретного) напрямку – для надання інформації клієнтові (13 екранів);
- екрани з ARKit – модулі, необхідні для надання певних переваг у вивченні астрономії;

Основна частина навігації була виконана за допомогою таких елементів, як Tab Bar Controller та Navigation Controller, але в програмному продукті були використані і такі функції навігації:

- relationship Segue – надає можливість зв'язування екранів для структуризації програми;
- storyboard Reference – надає можливість роз'єднання екранів розробки;
- navigation Item – одиниця навігації;
- button – кнопка;
- label – текстове поле;
- view – надає можливість додавати зображення та відео;
- safe Area – надає можливість використовувати цілий екран.

У додатку розроблена функція перегляду відео зі стороннього ресурсу за допомогою елементу Webkit. Створені AR вікна для можливості розглянути небесні тіла детально через камеру смартфона. Створено кастомний “Космо-калькулятор” за допомогою фреймворку SwiftUI.

5.4 Призначення та умови виконання програми

Програма призначена для покращення якості вивчення астрономії. Для роботи додатку необхідно мати iOS пристрій (смартфон) з операційною системою iOS 13 і пізніше. Для функціонування елементу програми WebView

необхідна наявність доступу до інтернету. Усі інші частини застосунку працюють без доступу до інтернету.

5.5 Виконання програми

Головне вікно (рис. 5.1) має навігаційне меню і кнопку переходу до напрямку “AR”.

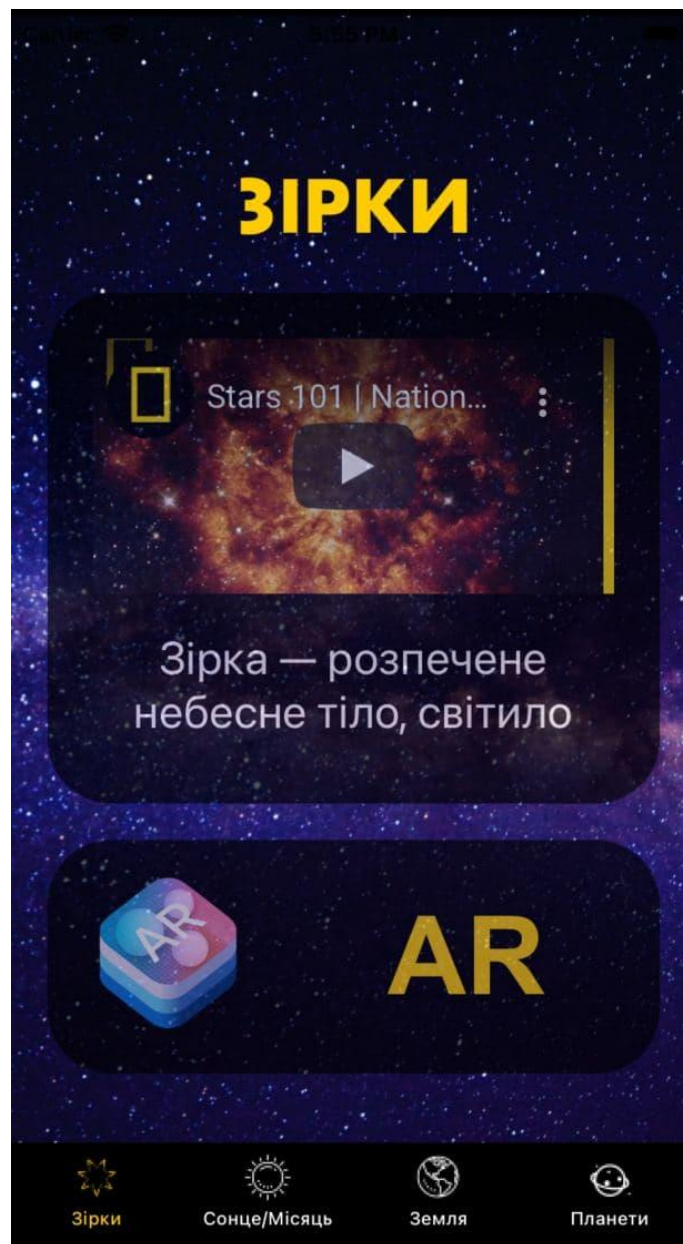


Рисунок 5.1 – Вигляд основного вікна програми

У навігаційному меню знаходяться пункти: «Сонце/Місяць», «Земля» (рис. 5.2 – 5.3).

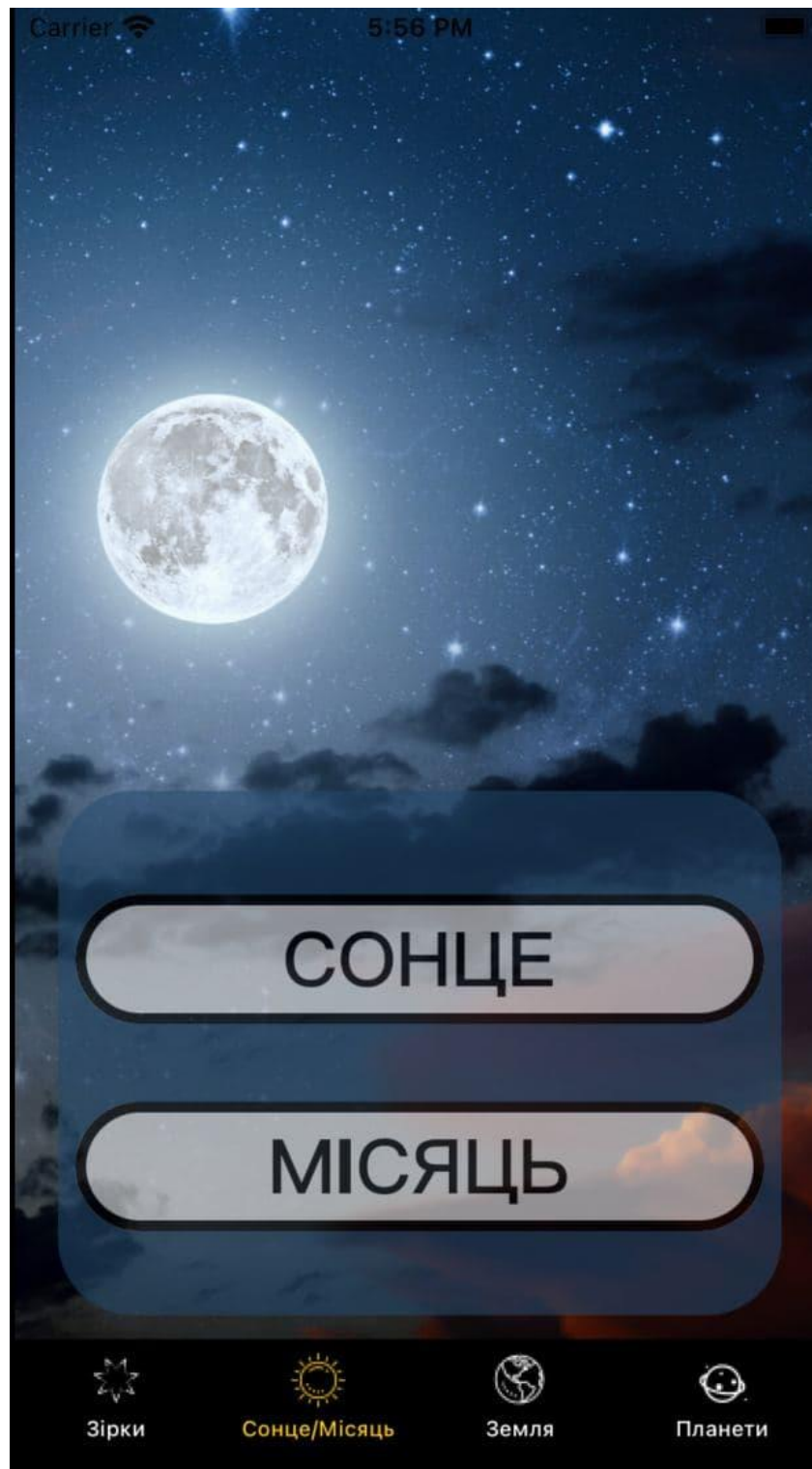


Рисунок 5.2 – Вигляд вікна після натиснення кнопки «Сонце/Місяць»

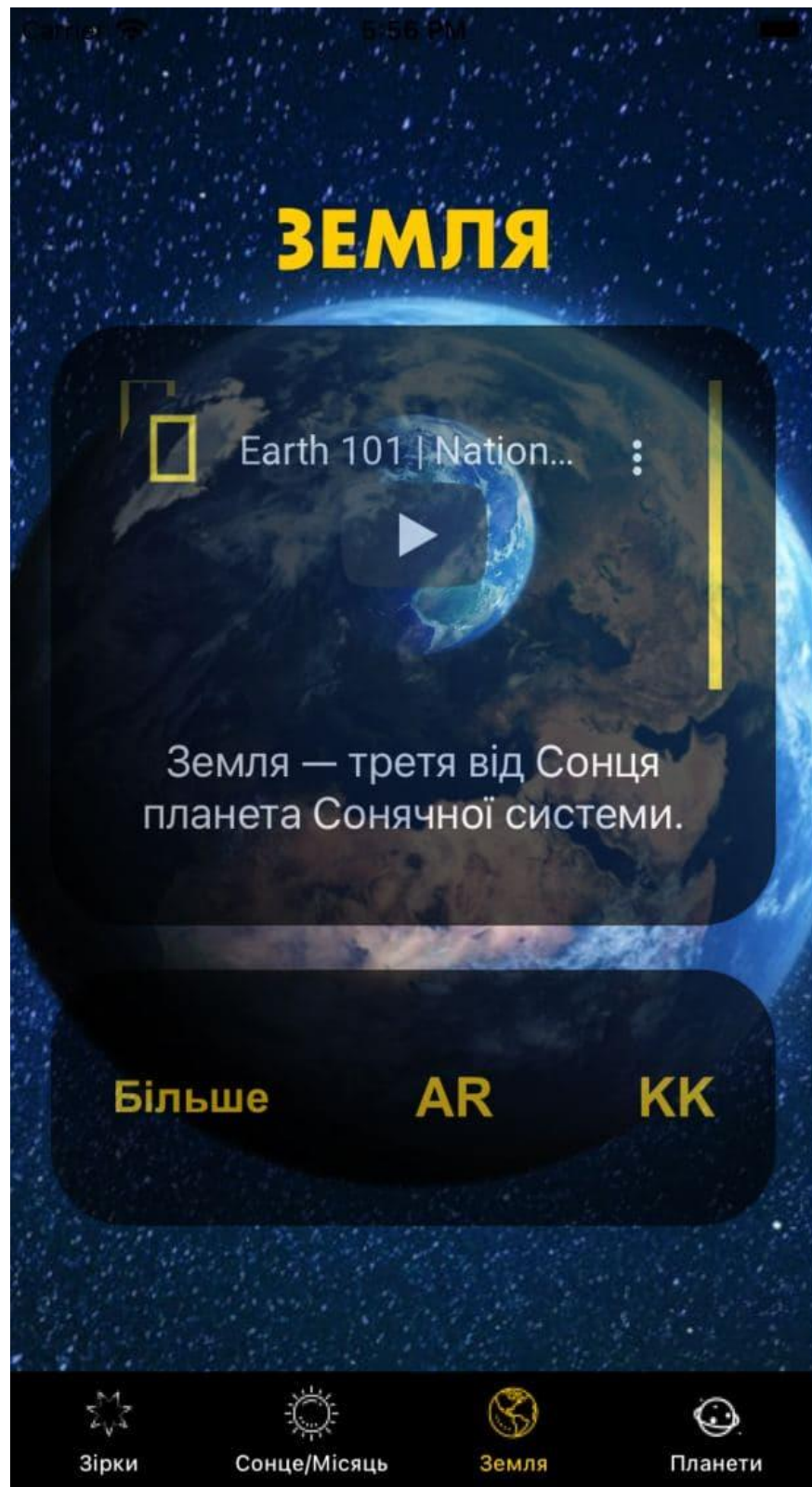


Рисунок 5.3 – Вигляд вікна після натиснення кнопки «Земля»

Також у навігаційному меню є пункти: «Зірки», «Планети» (рис. 5.4).

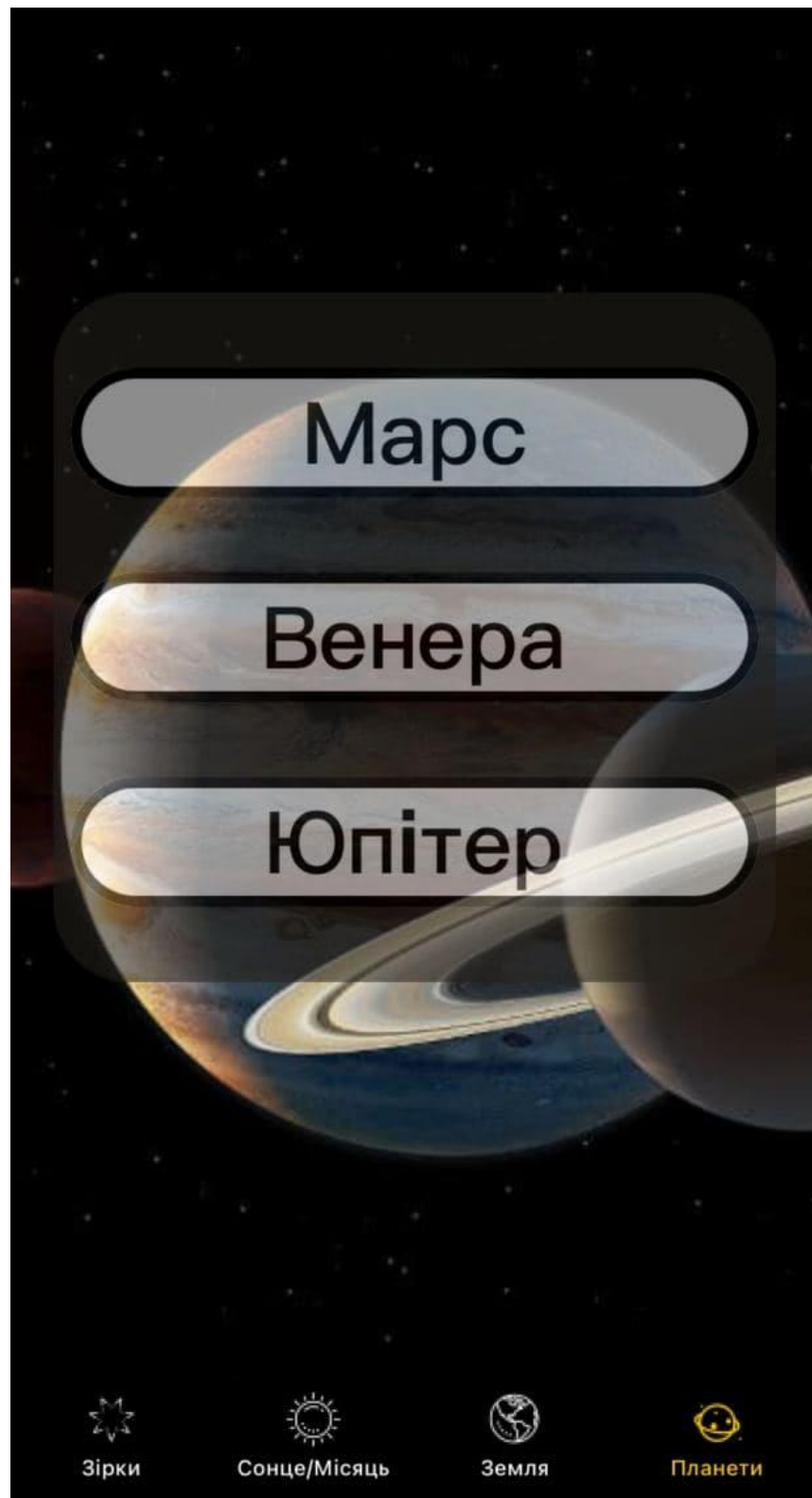


Рисунок 5.4 – Вигляд вікна після натиснення кнопки «Планети»

Після натискання кнопки “Сонце” в розділі “Сонце/Місяць” ми переходимо до екрану напрямку “Сонце” (рис. 5.5).

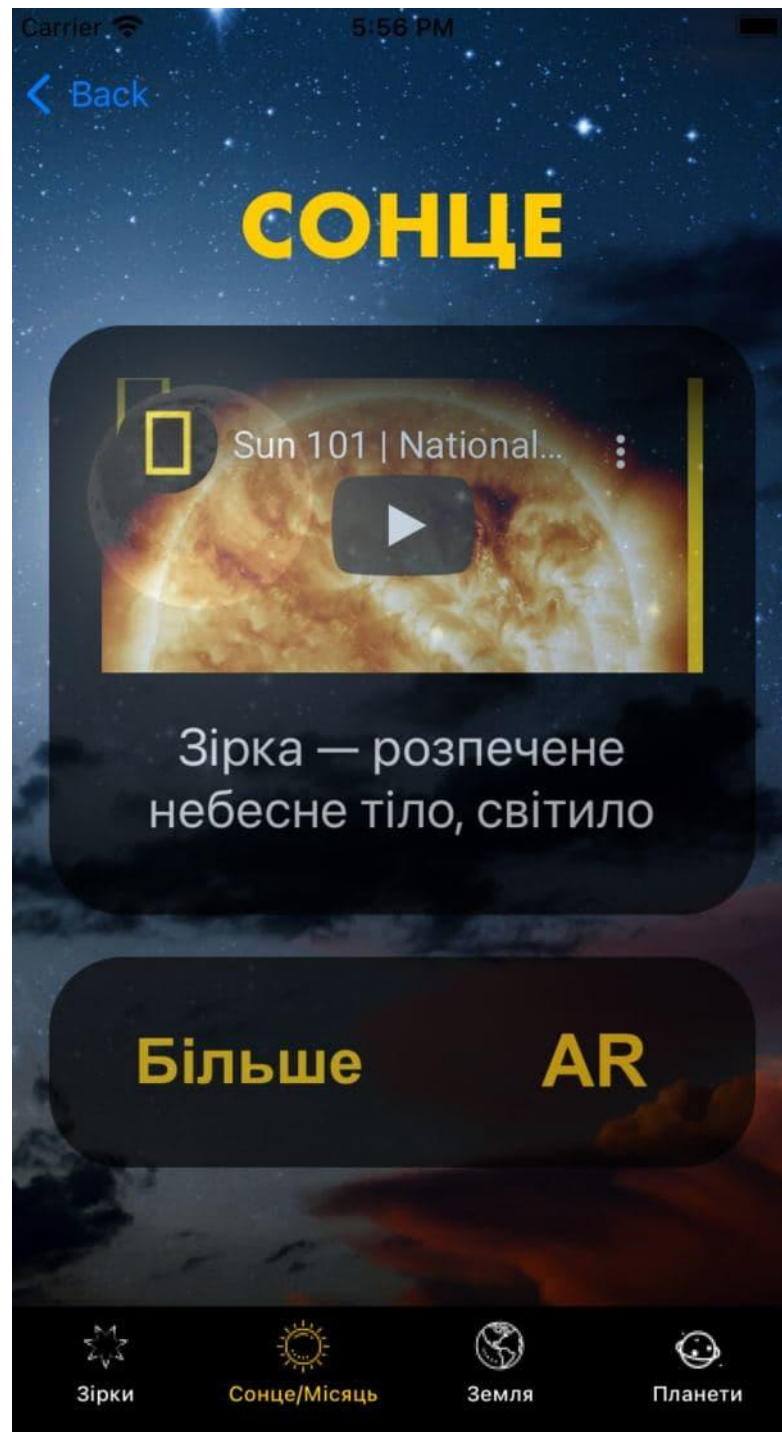


Рисунок 5.5 – Вигляд вікна після натиснення кнопки «Сонце»

Після натискання кнопки «Play» відео розгортається на весь екран та починається (рис. 5.6). За допомогою елементу Webkit View у додатку реалізовані можливості відключити звук, розгорнути відео на весь екран, додати субтитри, перемотування кнопками на 15 секунд вперед/назад і перемотування скролом.

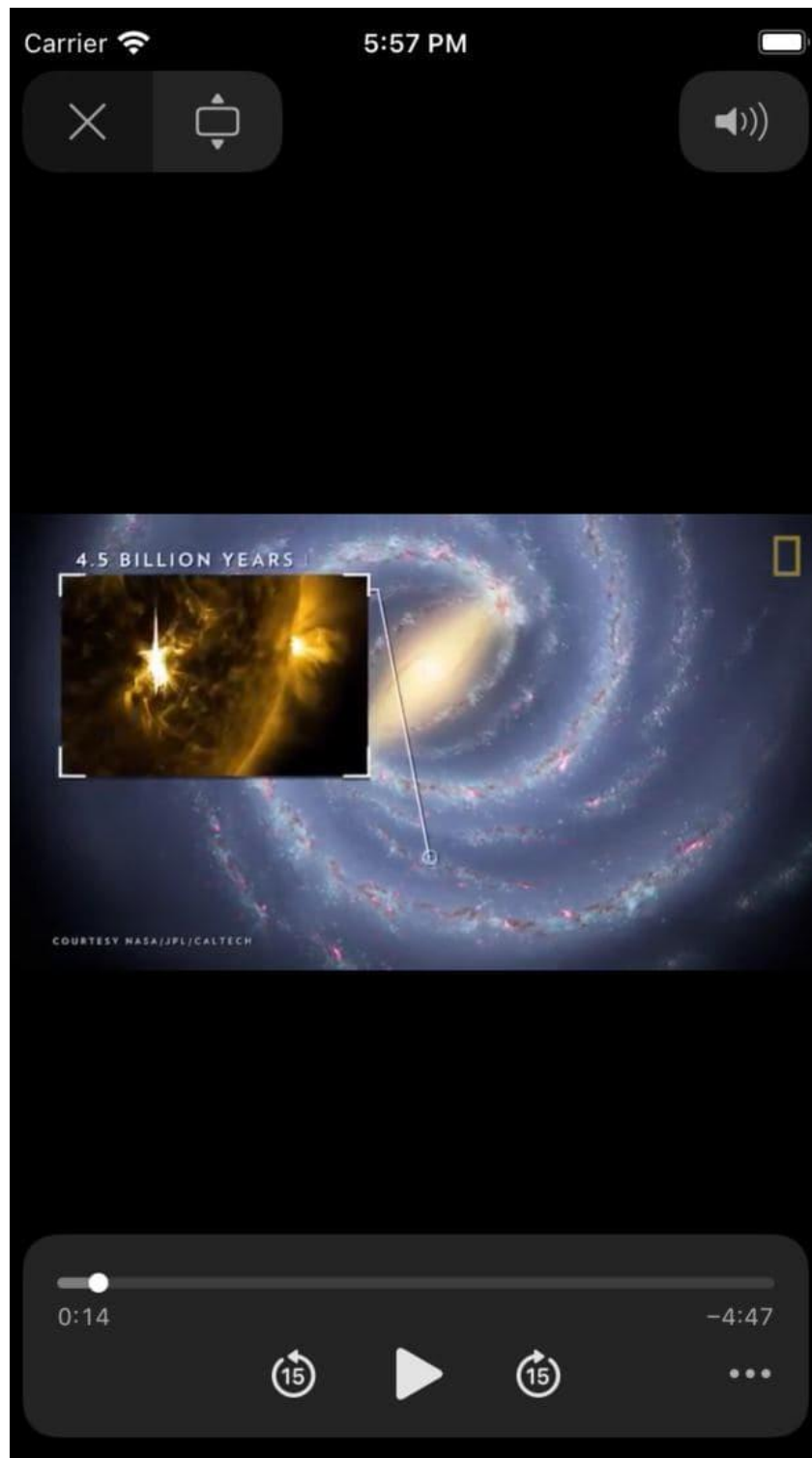


Рисунок 5.6 – Вигляд вікна після натиснення кнопки «Play»

Після натискання кнопки “Більше” у напрямку “Сонце” відкривається екран з детальним описом по обраному небесному тілу (рис. 5.7).



Рисунок 5.7 – Вигляд вікна після натиснення кнопки «Play»

При натисканні кнопки “AR” у додатку відкривається камера смартфона і у людини з’являється можливість розмістити Модель небесного тіла (рис. 5.8).

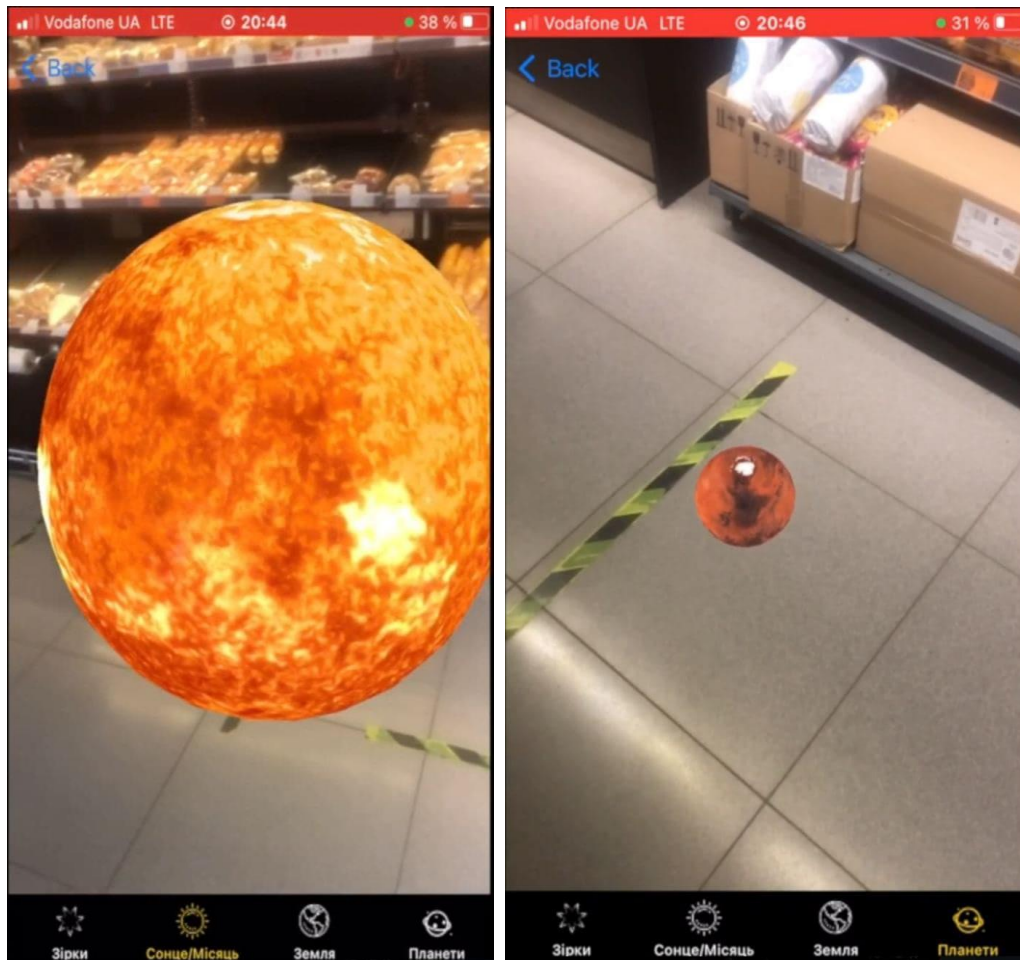


Рисунок 5.8 – Вигляд вікна після натиснення кнопки «AR»

При натисканні кнопки «КК» на напрямку “Земля”, ми переходимо до вікна “Космо-калькулятору” (рис. 5.9).

5.6 Повідомлення

Програма забезпечена різними поясненнями і повідомленнями користувачеві. При спробі перегляду інформації без доступу до інтернету на місці відео будуть знаходитись тільки текстові інструкції, вікна “AR” та “КК” будуть працювати.

5.7 Тестування програми

Для тестування програми були використані ПК з macOS, Смартфон з iOS13 та кабель для з'єднання. У смартфоні потрібно перейти у налаштування, пункт «Основні», далі пункт «Профіль», далі дозволити роботу додатку «стороннього розробника».

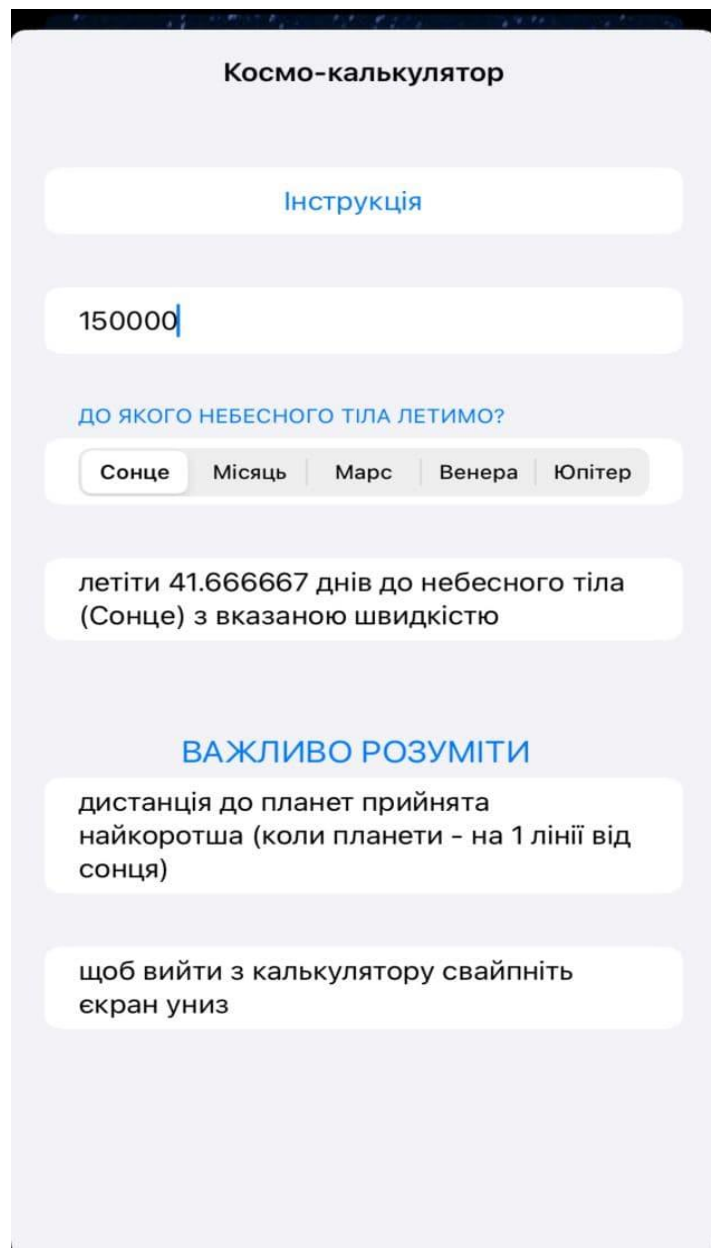


Рисунок 5.9 – Вигляд вікна «Космо-калькулятор»

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було вирішено такі задачі:

- проведено аналіз предметної області;
- проаналізовано існуючі аналоги;
- проаналізовано існуючі технології доповненої реальності;
- оглянуто методи позиціювання об'єктів в доповненій реальності;
- виконано проєктування додатку;
- створено метод `touchesBegan` для удосконалення позиціювання об'єктів в доповненій реальності;
- розроблено програмний продукт;
- проведено тестування.

При розробці мобільного застосунку було використано фреймворк ARKit для реалізації можливості розміщувати віртуальні об'єкти вивчення поруч із собою і вивчати їх з усіх боків через камеру смартфона.

Було запропоновано використовувати метод `touchesBegan`, заснований на комбінації методів `hitTest` та `worldTransform` та тривимірної матриці. Цей метод дозволяє покращити точність розташування моделі над реальною поверхнею.

Для реалізації програмного продукту була використана мова програмування Swift, бібліотеки ARKit, SwiftUI та UIKit, окрім цього, були описані особливості обраної мови програмування.

В результаті виконання дипломної роботи з використанням методів об'єктно-орієнтованого програмування було спроектовано мобільний застосунок, який повністю реалізує поставлену задачу.

ПЕРЕЛІК ДЖЕРЕЛ І ПОСИЛАННЯ

1. Apple Developer Augmented Reality [Electronic resource]. – Access mode: <https://developer.apple.com/augmented-reality/>.
2. Доповнена реальність, або AR-технології. Як це працює? [Електрон. ресурс]. – Режим доступу : <http://thefuture.news/page1837780.html>.
3. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: «Пітер», 2001. – 367 с.
4. Маскри, М. Swift 3. Разработка приложений в среде Xcode для iPhone и iPad с использованием iOS SDK. [Текст] / М. Маскри. – «Apress», 2017. – 900 с.
5. Усов, В. Swift. Основы разработки приложений под iOS и macOS. [Текст] / В. Усов. – СПб.: «Пітер», 2019. – 444 с.
6. Босуэлл, Д. Читаемый код, или Программирование как искусство. [Текст] / Д. Босуэлл, Т. Фаучер. – СПб.: «Пітер», 2012. – 208 с.
7. Фримен, Э. Паттерны проектирования [Текст] / Э. Фримен, К. Сьерра, Б. Бейтс. – СПб.: «Пітер», 2018. – 657 с.
8. Dipperry M. Professional Swift [Текст] / М. Dipperry. – «Apress», 2016. – 520 с.
9. Hoffman, J. Mastering swift 5: deep dive into the latest edition of the swift programming language [Текст] / J. Hoffman. – «Packt», 2017. – 731 с.
10. Manning, J. Learning Swift: Building Apps for macOS, iOS, and Beyond [Текст] / J. Manning. – «Oreilly», 2017. – 820 с.
11. Mathias, M. Swift Programming: The Big Nerd Ranch Guide (2nd Edition) [Текст] / М. Mathias. – «Big Nerd Ranch Guides», 2016. – 340 с.
12. Martin, M. Swift: The Complete Guide for Beginners, Intermediate and Advanced Detailed Strategies To Master Swift Programming [Текст] / М. Martin. – «Apress», 2016. – 705 с.

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.1 Назва розроблюваного програмного забезпечення

Назва розроблюваного програмного забезпечення: «SpaceStudyApp».

A.1.2 Призначення та область застосування

Система призначена для надавання доступу користувачеві до цікавого та комфортного вивчення астрономії, покращення підходу до вивчення саме астрономії та небесних тіл.

A.2 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему: «Впровадження AR-технологій в користувацьких мобільних застосунках на прикладі ARKit у iOS».

A.3 Призначення розробки

Розробка призначена для створення та покращення умов вивчення астрономії.

A.4 Вимоги до програмного виробу

A.4.1 Вимоги до функціональних характеристик

Алгоритм має виконувати такі функції:

- доступ до відео та інформації з астрономії;
- можливість додавання моделей небесних тіл у реальний простір через камеру смартфона.

Інтерфейс користувача повинен забезпечити просту й ефективну взаємодію користувача з програмним продуктом.

A.4.2 Вимоги до надійності

Програма повинна розпізнавати ситуації взаємного нерозуміння між програмою і користувачем, надавати підказки користувачу. У випадку критичних помилок коректна робота системи повинна відновлюватися після перезавантаження системи.

A.4.3 Умови експлуатації

Користувач застосунку повинен мати базові навички володіння смартфоном з операційною системою iOS.

A.4.4 Вимоги до складу та параметрів технічних засобів

Для відсутності проблем функціонування додатку на смартфоні користувача повинні виконуватися такі умови до апаратної та програмної частини:

- смартфон з iOS (iPhone);
- iPhone8 і новіше;
- операційна система iOS12 і новіше.

A.4.5 Вимоги до зберігання

Програмний засіб зберігатиметься у вигляді папки з необхідними для роботи програми файлами на ПК, а на смартфоні користувача – мобільний застосунок з іконкою і назвою.

ДОДАТОК Б**Текст розробленої програми**

Б.1 Лістинг AppDelegate.swift

```
import UIKit

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        UITabBar.appearance().unselectedItemTintColor = .white
        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can
occur for certain types of temporary interruptions (such as an incoming phone call or SMS
message) or when the user quits the application and it begins the transition to the background
state.

        // Use this method to pause ongoing tasks, disable timers, and invalidate graphics
rendering callbacks. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers,
and store enough application state information to restore your application to its current state in
case it is terminated later.
    }

    func applicationWillEnterForeground(_ application: UIApplication) {
        // Called as part of the transition from the background to the active state; here you
can undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(_ application: UIApplication) {
```

// Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.

```
    }
}
```

Б.2 Лістинг ViewController.swift

```
import UIKit
import SceneKit
import ARKit

class ViewController: UIViewController, ARSCNViewDelegate {
    @IBOutlet var sceneView: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Set the view's delegate
        sceneView.delegate = self
        // Show statistics such as fps and timing information
        sceneView.showsStatistics = false
        // Create a new scene
        let scene = SCNScene()
        // Set the scene to the view
        sceneView.scene = scene
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        sceneView.session.run(configuration)
    }
}
```

```

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    // Pause the view's session
    sceneView.session.pause()
}

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let touch = touches.first else { return }
    let result = sceneView.hitTest(touch.location(in: sceneView), types:
[.featurePoint])
    guard let hitResult = result.last else { return }
    let hitMatrix = hitResult.worldTransform
    let matrix = SCNMatrix4(hitMatrix)
    let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
    createBall(position: hitVector)
}

func createBall(position: SCNVector3) {
    let ballShape = SCNSphere(radius: 0.01)
    let ballNode = SCNNode.init(geometry: ballShape)
    ballNode.position = position
    sceneView.scene.rootNode.addChildNode(ballNode)
}

func session(_ session: ARSession, didFailWithError error: Error) {
}

func sessionWasInterrupted(_ session: ARSession) {
}

func sessionInterruptionEnded(_ session: ARSession) {
}
}

```

Б.3 Лістинг StarScreen.swift

```

import UIKit
import WebKit

```

```

class StarScreen: UIViewController {
    @IBOutlet weak var ViewCard: UIView!
    @IBOutlet weak var webView: WKWebView!
    @IBOutlet weak var CardToAR: UIView!
    @IBOutlet weak var img: UIImageView!
    var link = "https://www.youtube.com/embed/EFO_bsg1sw8"
    override func viewDidLoad() {
        super.viewDidLoad()
        ViewCard.layer.cornerRadius = 35
        CardToAR.layer.cornerRadius = 35
        img.layer.cornerRadius = 35
        webView.load(URLRequest(url: URL(string: link)!))
        // Do any additional setup after loading the view.
    }
}

```

Б.4 Лістинг MoonSunScreen.swift

```

import UIKit
class MoonSunScreen: UIViewController {
    @IBOutlet weak var BView: UIView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()
        BView.layer.cornerRadius = 35
        Button1.layer.borderWidth = 5
        Button1.layer.cornerRadius = 30
        Button2.layer.cornerRadius = 30
        Button2.layer.borderWidth = 5
    }
}

```

Б.5 Лістинг SunScreen.swift

```
import UIKit
import WebKit

class SunScreen: UIViewController {
    @IBOutlet weak var View1: UIView!
    @IBOutlet weak var View2: UIView!
    @IBOutlet weak var webView: WKWebView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!

    var link = "https://www.youtube.com/embed/2HoTK_Gqi2Q"

    override func viewDidLoad() {
        super.viewDidLoad()

        View1.layer.cornerRadius = 35
        View2.layer.cornerRadius = 35

        Button2.layer.cornerRadius = 35
        Button1.layer.cornerRadius = 35

        webView.load(URLRequest(url: URL(string: link)!))
    }
}
```

Б.6 Лістинг SunAR.swift

```
import UIKit
import SceneKit
import ARKit

class SunAR: UIViewController, ARSCNViewDelegate {

    @IBOutlet weak var SunAR: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Set the view's delegate
        SunAR.delegate = self
        // Show statistics such as fps and timing information
        SunAR.showsStatistics = false
        // Create a new scene
        let scene = SCNScene()
        SunAR.scene = scene
    }
}
```

```

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    // Create a session configuration
    let configuration = ARWorldTrackingConfiguration()
    SunAR.session.run(configuration)
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    SunAR.session.pause()
}

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let touch = touches.first else { return }
    let result = SunAR.hitTest(touch.location(in: SunAR), types: [.featurePoint])
    guard let hitResult = result.last else { return }
    let hitMatrix = hitResult.worldTransform
    let matrix = SCNMatrix4(hitMatrix)
    let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
    createEarth(position: hitVector)
}

func createEarth(position: SCNVector3) {
    let sunShape = SCNSphere(radius: 0.13)
    let sunNode = SCNNode.init(geometry: sunShape)
    sunNode.position = position
    sunNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named: "8k_sun")
    SunAR.scene.rootNode.addChildNode(sunNode)
}
}

```

Б.7 Лістинг MoonScreen.swift

```

import UIKit
import WebKit

class MoonScreen: UIViewController {
    @IBOutlet weak var View1: UIView!
    @IBOutlet weak var View2: UIView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!
    @IBOutlet weak var webView: WKWebView!

    var link = "https://www.youtube.com/embed/6AviDjR9mmo"
    override func viewDidLoad() {
        super.viewDidLoad()
        View1.layer.cornerRadius = 35
        View2.layer.cornerRadius = 35
        Button2.layer.cornerRadius = 35
        Button1.layer.cornerRadius = 35
        webView.load(URLRequest(url: URL(string: link)!))
    }
}

```

```

    }
}

```

Б.8 Лістинг MoonAR.swift

```

import UIKit
import SceneKit
import ARKit

class MoonAR: UIViewController, ARSCNViewDelegate {

    @IBOutlet weak var MoonAR: ARSCNView!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Set the view's delegate
        MoonAR.delegate = self
        // Show statistics such as fps and timing information
        MoonAR.showsStatistics = false
        // Create a new scene
        let scene = SCNScene()
        MoonAR.scene = scene
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        MoonAR.session.run(configuration)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        // Pause the view's session
        MoonAR.session.pause()
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        guard let touch = touches.first else { return }
        let result = MoonAR.hitTest(touch.location(in: MoonAR), types: [.featurePoint])
        guard let hitResult = result.last else { return }
        let hitMatrix = hitResult.worldTransform
        let matrix = SCNMatrix4(hitMatrix)
        let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
        createEarth(position: hitVector)
    }

    func createEarth(position: SCNVector3) {
        let moonShape = SCNSphere(radius: 0.13)
    }
}

```

```

        let moonNode = SCNNode.init(geometry: moonShape)
        moonNode.position = position
        moonNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named:
"8k_moon")
        MoonAR.scene.rootNode.addChildNode(moonNode)
    }
}

```

Б.9 Лістинг EarthScreen.swift

```

import UIKit
import WebKit

class EarthScreen: UIViewController {
    @IBOutlet weak var View1: UIView!
    @IBOutlet weak var View2: UIView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!
    @IBOutlet weak var webView: WKWebView!
    var link = "https://www.youtube.com/embed/HCDVN7DCzYE"
    override func viewDidLoad() {
        super.viewDidLoad()
        View1.layer.cornerRadius = 35
        View2.layer.cornerRadius = 35
        Button2.layer.cornerRadius = 35
        Button1.layer.cornerRadius = 35
        webView.load(URLRequest(url: URL(string: link)!))
    }
}

```

Б.10 Лістинг EarthAR.swift

```

import UIKit
import SceneKit
import ARKit

class EarthAR: UIViewController, ARSCNViewDelegate {

    @IBOutlet weak var EarthAR: ARSCNView!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Set the view's delegate
        EarthAR.delegate = self
        // Show statistics such as fps and timing information
        EarthAR.showsStatistics = false
        // Create a new scene
        let scene = SCNScene()
        // Set the scene to the view
        EarthAR.scene = scene
    }
}

```

```

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    let configuration = ARWorldTrackingConfiguration()
    // Run the view's session
    EarthAR.session.run(configuration)
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    // Pause the view's session
    EarthAR.session.pause()
}

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let touch = touches.first else { return }
    let result = EarthAR.hitTest(touch.location(in: EarthAR), types: [.featurePoint])
    guard let hitResult = result.last else { return }
    let hitMatrix = hitResult.worldTransform
    let matrix = SCNMatrix4(hitMatrix)
    let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
    createEarth(position: hitVector)
}

func createEarth(position: SCNVector3) {
    let earthShape = SCNSphere(radius: 0.13)
    let earthNode = SCNNode.init(geometry: earthShape)
    earthNode.position = position
    earthNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named:
"8k_earth_daymap")
    earthNode.geometry?.firstMaterial?.specular.contents = UIImage(named:
"8k_earth_specular_map")
    earthNode.geometry?.firstMaterial?.emission.contents = UIImage(named:
"8k_earth_clouds")
    earthNode.geometry?.firstMaterial?.normal.contents = UIImage(named:
"8k_earth_normal_map")
    EarthAR.scene.rootNode.addChildNode(earthNode)
}
}

```

Б.11 Лістинг PlanetsScreen.swift

```

import UIKit

class PlanetsScreen: UIViewController {

    @IBOutlet weak var View1: UIView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!
    @IBOutlet weak var Button3: UIButton!

```

```

override func viewDidLoad() {
    super.viewDidLoad()

    View1.layer.cornerRadius = 35
    Button1.layer.borderWidth = 5
    Button1.layer.cornerRadius = 30
    Button2.layer.cornerRadius = 30
    Button2.layer.borderWidth = 5
    Button3.layer.cornerRadius = 30
    Button3.layer.borderWidth = 5
}
}

```

Б.12 Лістинг MarsScreen.swift

```

import UIKit
import WebKit

class MarsScreen: UIViewController {

    @IBOutlet weak var View1: UIView!
    @IBOutlet weak var View2: UIView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!
    @IBOutlet weak var webView: WKWebView!

    var link = "https://www.youtube.com/embed/D8pnmwOXhoY"

    override func viewDidLoad() {
        super.viewDidLoad()
        View1.layer.cornerRadius = 35
        View2.layer.cornerRadius = 35
        Button2.layer.cornerRadius = 35
        Button1.layer.cornerRadius = 35
        webView.load(URLRequest(url: URL(string: link)!))
    }

}

```

Б.13 Лістинг MarsScreen.swift

```

import UIKit
import SceneKit
import ARKit

class MarsAR: UIViewController, ARSCNViewDelegate {

    @IBOutlet var MarsAR: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}

```

```

        // Set the view's delegate
        MarsAR.delegate = self

        // Show statistics such as fps and timing information
        MarsAR.showsStatistics = false

        // Create a new scene
        let scene = SCNScene()

        // Set the scene to the view
        MarsAR.scene = scene
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        MarsAR.session.run(configuration)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        // Pause the view's session
        MarsAR.session.pause()
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        guard let touch = touches.first else { return }
        let result = MarsAR.hitTest(touch.location(in: MarsAR), types: [.featurePoint])
        guard let hitResult = result.last else { return }
        let hitMatrix = hitResult.worldTransform
        let matrix = SCNMatrix4(hitMatrix)
        let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
        createEarth(position: hitVector)
    }

    func createEarth(position: SCNVector3) {
        let marsShape = SCNSphere(radius: 0.13)
        let marsNode = SCNNode.init(geometry: marsShape)
        marsNode.position = position
        marsNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named:
"8k_mars")
        MarsAR.scene.rootNode.addChildNode(marsNode)
    }
}

```

Б.14 Лістинг VenusScreen.swift

```
import UIKit
import WebKit

class VenusScreen: UIViewController {

    @IBOutlet weak var View1: UIView!
    @IBOutlet weak var View2: UIView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!
    @IBOutlet weak var webView: WKWebView!

    var link = "https://www.youtube.com/embed/BvXa1n9fjow"

    override func viewDidLoad() {
        super.viewDidLoad()

        View1.layer.cornerRadius = 35
        View2.layer.cornerRadius = 35

        Button2.layer.cornerRadius = 35
        Button1.layer.cornerRadius = 35

        webView.load(URLRequest(url: URL(string: link)!))
    }
}
```

Б.15 Лістинг VenusAR.swift

```
import UIKit
import SceneKit
import ARKit

class VenusAR: UIViewController, ARSCNViewDelegate {

    @IBOutlet weak var VenusAR: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Set the view's delegate
        VenusAR.delegate = self

        // Show statistics such as fps and timing information
        VenusAR.showsStatistics = false

        // Create a new scene
        let scene = SCNScene()

        // Set the scene to the view
        VenusAR.scene = scene
    }
}
```

```

    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        VenusAR.session.run(configuration)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        // Pause the view's session
        VenusAR.session.pause()
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        guard let touch = touches.first else { return }
        let result = VenusAR.hitTest(touch.location(in: VenusAR), types: [.featurePoint])
        guard let hitResult = result.last else { return }
        let hitMatrix = hitResult.worldTransform
        let matrix = SCNMatrix4(hitMatrix)
        let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
        createEarth(position: hitVector)
    }

    func createEarth(position: SCNVector3) {
        let venusShape = SCNSphere(radius: 0.13)
        let venusNode = SCNNode.init(geometry: venusShape)
        venusNode.position = position
        venusNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named:
"8k_venus_surface")
        VenusAR.scene.rootNode.addChildNode(venusNode)
    }
}

```

Б.16 Лістинг JupiterScreen.swift

```

import UIKit
import WebKit

class JupiterScreen: UIViewController {

    @IBOutlet weak var View1: UIView!
    @IBOutlet weak var View2: UIView!
    @IBOutlet weak var Button1: UIButton!
    @IBOutlet weak var Button2: UIButton!
    @IBOutlet weak var webView: WKWebView!

```

```

var link = "https://www.youtube.com/embed/PtkqwsIbLY8"

override func viewDidLoad() {
    super.viewDidLoad()

    View1.layer.cornerRadius = 35
    View2.layer.cornerRadius = 35

    Button2.layer.cornerRadius = 35
    Button1.layer.cornerRadius = 35

    webView.load(URLRequest(url: URL(string: link)!))
}
}

```

Б.17 Лістинг JupiterAR.swift

```

import UIKit
import SceneKit
import ARKit

class JupiterAR: UIViewController, ARSCNViewDelegate {

    @IBOutlet weak var JupiterAR: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Set the view's delegate
        JupiterAR.delegate = self

        // Show statistics such as fps and timing information
        JupiterAR.showsStatistics = false

        // Create a new scene
        let scene = SCNScene()

        // Set the scene to the view
        JupiterAR.scene = scene
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        JupiterAR.session.run(configuration)
    }

    override func viewWillDisappear(_ animated: Bool) {

```

```

super.viewWillDisappear(animated)

// Pause the view's session
JupiterAR.session.pause()
}

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let touch = touches.first else { return }
    let result = JupiterAR.hitTest(touch.location(in: JupiterAR), types: [.featurePoint])
    guard let hitResult = result.last else { return }
    let hitMatrix = hitResult.worldTransform
    let matrix = SCNMatrix4(hitMatrix)
    let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
    createEarth(position: hitVector)
}

func createEarth(position: SCNVector3) {
    let jupiterShape = SCNSphere(radius: 0.13)
    let jupiterNode = SCNNode.init(geometry: jupiterShape)
    jupiterNode.position = position
    jupiterNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named:
"8k_jupiter")
    JupiterAR.scene.rootNode.addChildNode(jupiterNode)
}
}

```

Б.18 Лістинг SwiftUIKit.swift

```

import UIKit
import SwiftUI

class SwiftUIKit: UIViewController {
    @IBOutlet weak var theContainer : UIView!

    override func viewDidLoad() {
        super.viewDidLoad()
        let childView = UIHostingController(rootView: SwiftUIView())
        addChild(childView)
        childView.view.frame = theContainer.bounds
        theContainer.addSubview(childView.view)
    }
}

```

Б.19 Лістинг SwiftUIView.swift

```

import SwiftUI

struct SwiftUIView: View {

    @State private var showSheet = false

    @State private var checkFast = ""

```

```

@State private var skybody = 0
@State private var planet = 0

let planets: [String] = ["Сонце", "Місяць", "Марс", "Венера", "Юпітер"]
let waySizes: [Double] = [150000000, 384400, 55760000, 38000000, 588000000]

var TotalDays: Double {
    let planetWayLong = Double(waySizes[planet])
    let orderFast = Double(checkFast) ?? 0

    let tipValue = (planetWayLong / orderFast) / 24

    return tipValue
}

var body: some View {
    NavigationView{
        Form{
            Button(action: {
                self.showSheet = true
            }) {
                Text("Інструкція")
                .padding(.horizontal, 120)
            }
            .actionSheet(isPresented: $showSheet) {
                ActionSheet(title: Text("Космо-Калькулятор"),
                    message: Text("Спочатку вкажіть швидкість 'вашого космічного транспорту' у кілометрах на годину -> Далі оберіть небесне тіло до якого летіти -> Отримайте кількість днів, за які ви зможете дістатися до обраного вами небесного тіла, з вказаною швидкістю"),
                    buttons: [.default(Text("Зрозумів. Готовий користуватись"))])
            }

            Section{
                TextField("Ваша швидкість (км/г)", text: $checkFast)
            }

            Section(header: Text("До якого небесного тіла летимо?").foregroundColor(.blue)){
                Picker("Planet", selection: $planet){
                    ForEach(0..

```

```

        Section(header:
            Text("Важливо розуміти")
            .font(.title3)
            .padding(.top)
            .padding(.horizontal, 60)
            .foregroundColor(.blue)
        ){
            Text("дистанція до планет прийнята найкоротша (коли планети - на 1
лінії від сонця)")
        }

        Section{
            Text("щоб вийти з калькулятора свайпніть екран униз")
        }
    }
    .navigationBarTitle("Космо-калькулятор", displayMode: .inline)
}

struct SwiftUIView_Previews: PreviewProvider {
    static var previews: some View {
        SwiftUIView()
    }
}

```

ДОДАТОК В
Слайди презентації

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

Дипломна кваліфікаційна робота
магістра на тему: “Впровадження AR-
технологій в користувацьких мобільних
застосунках на прикладі ARKIT у IOS”

Виконав

ст. гр. КНТ-210м

Д.Е. Овчаренко

Керівник

доцент, к.т.н.

Т.О. Колпакова

1

Рисунок В.1 – Слайд 1

Об’єкт дослідження: технології доповненої реальності у розробці мобільних застосунків.

Предмет дослідження: програмний застосунок для вивчення космічних тіл та космосу на основі фреймворку ARKit.

Мета роботи: створення мобільного застосунку для вивчення космічних тіл та космосу за допомогою доповненої реальності.

2

Рисунок В.2 – Слайд 2

ЗАДАЧІ РОБОТИ

- провести аналіз предметної області;
- проаналізувати існуючі аналоги;
- проаналізувати існуючі технології доповненої реальності;
- оглянути методи позиціонування об'єктів в доповненій реальності;
- виконати проектування додатку;
- створити метод для удосконалення позиціонування об'єктів в доповненій реальності;
- розробити програмний продукт;
- провести тестування.

3

Рисунок В.3 – Слайд 3

Типи доповненої реальності

- Доповнена реальність, що базується на маркерах;
- Безмаркерна доповнена реальність;
- Доповнена реальність, що базується на проєкції;
- Доповнена реальність, що базується на VIO.

4

Рисунок В.4 – Слайд 4

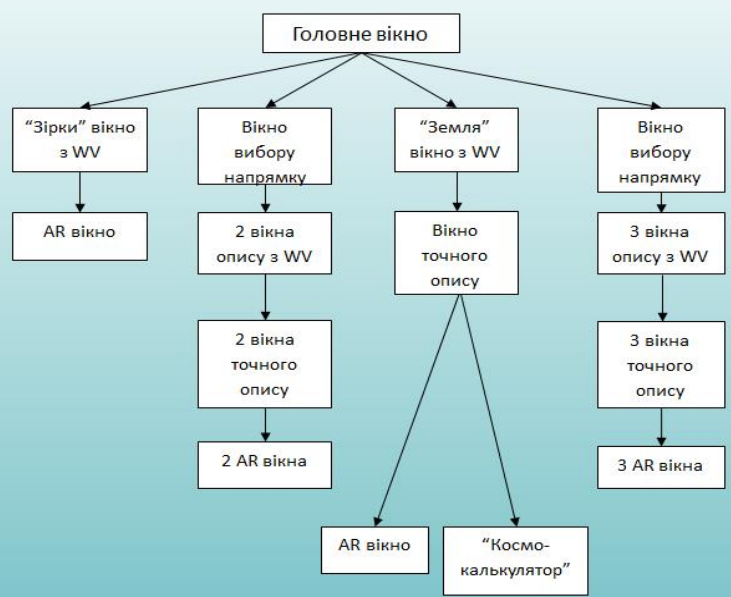
Порівняння AR-фреймворків

	Vuforia	ARToolKit	Wikitude	Lay AR	Kudan AR	ARKit
Розпізнавання кількох цілей одночасно	+	-	+	-	+	+
Відстеження цілей	+	+	+	+	+	+
Розпізнавання 2D форматів	+	+	+	+	+	+
OpenGL	-	+	-	-	-	+
Сканування реального об'єкта	+	+	+	+	+	+
Розпізнавання 3D форматів	+	-	+	+	-	+
Cloud recognition	+	-	-	-	+	+
Швидкодія	повільна	швидка	швидка	повільна	повільна	швидка

5

Рисунок В.5 – Слайд 5

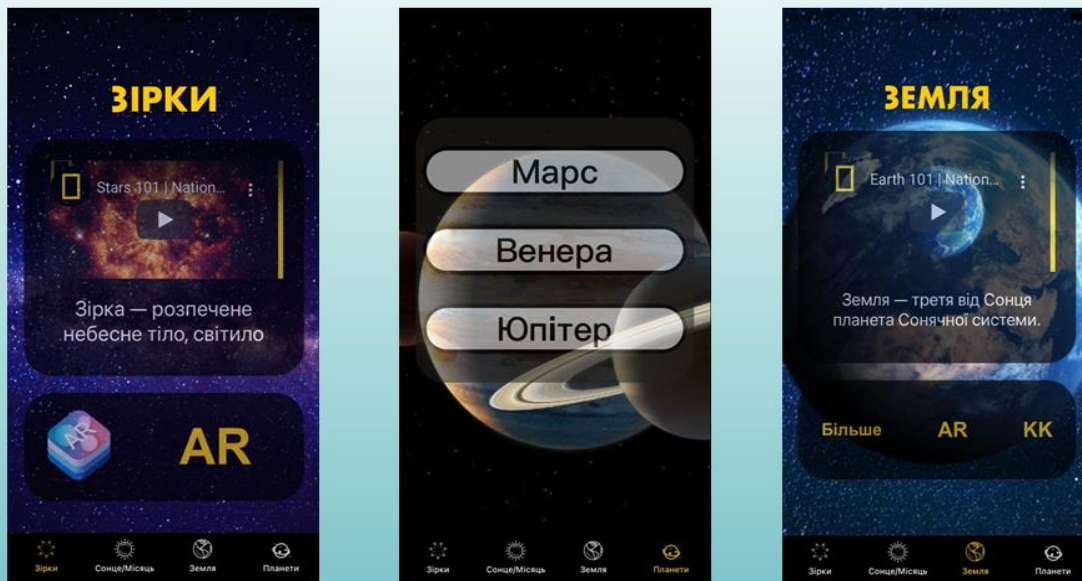
СТРУКТУРНА СХЕМА ПРОГРАМИ



6

Рисунок В.6 – Слайд 6

ПРИКЛАД РОБОТИ З ПРОГРАМОЮ. ГОЛОВНЕ ВІКНО



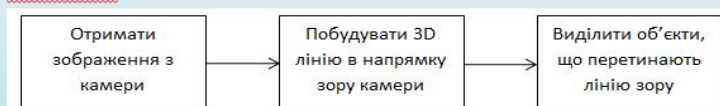
7

Рисунок В.7 – Слайд 7

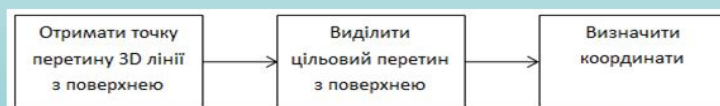
Стандартні методи позиціонування

Методи фреймворку ARKit:

- hitTest



- worldTransform



8

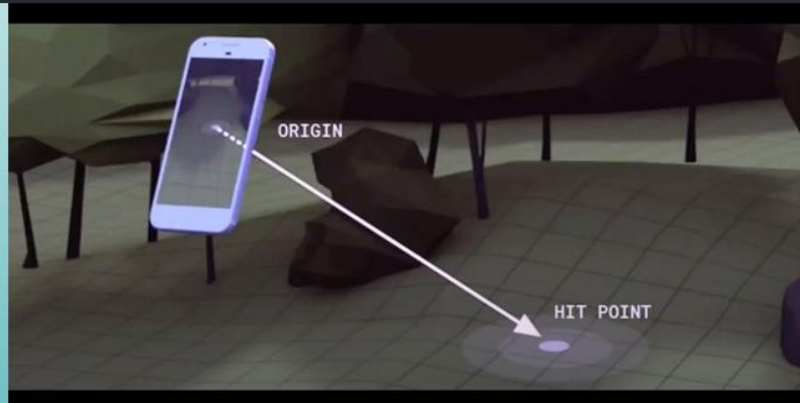
Рисунок В.8 – Слайд 8

Удосконалений методі позиціонування AR-моделей

```

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let touch = touches.first else { return }
    let result = sceneView.hitTest(touch.location(in: sceneView), types: [.featurePoint])
    guard let hitResult = result.last else { return }
    let hitMatrix = hitResult.worldTransform
    let matrix = SCNMatrix4(hitMatrix)
    let hitVector = SCNVector3.init(matrix.m41, matrix.m42, matrix.m43)
    createBall(position: hitVector)
}

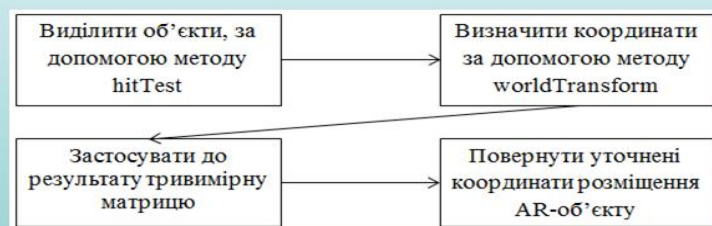
```



9

Рисунок В.9 – Слайд 9

Удосконалений метод позиціонування AR-моделей



10

Рисунок В.10 – Слайд 10

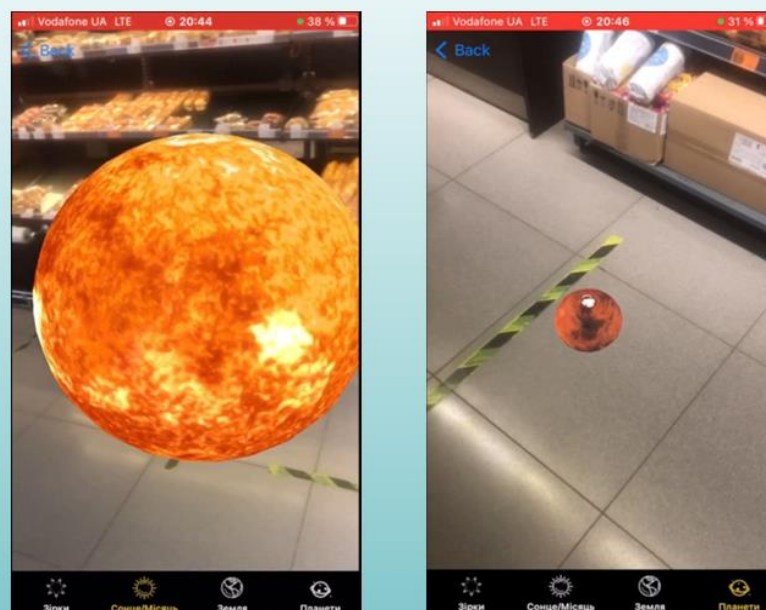
Порівняння методів позиціонування

Показники	touchesBegan	Стандартні методи
Точність розташування моделі	Покращена 7-11%	Достатня
Вплив на модель	++ визначаються 3D координати та розраховується додаткове позиціонування	– тільки визначаються 3D координати
Візуальне сприйняття	++ Візуально AR- елемент розташовується перед зазначеною поверхнею	– Візуально AR-елемент розташовується на зазначеній поверхні

11

Рисунок В.11 – Слайд 11

ПРИКЛАД РОБОТИ З ПРОГРАМОЮ. AR-ВІКНА



12

Рисунок В.12 – Слайд 12

ВИСНОВКИ

У ході виконання роботи було створено мобільний застосунок призначений для вивчення астрономії користувачем.

Вирішено такі завдання:

- проведено аналіз предметної області;
- проаналізовано існуючі аналоги;
- проаналізовано існуючі технології доповненої реальності;
- оглянуто методи позиціювання об'єктів в доповненій реальності;
- виконано проєктування додатку;
- створено метод `touchesBegan` для удосконалення позиціювання об'єктів в доповненій реальності;
- розроблено програмний продукт;
- проведено тестування.

13

Рисунок В.13 – Слайд 13

ВИСНОВКИ

При розробці мобільного застосунку було використано фреймворк ARKit для реалізації можливості розміщувати віртуальні об'єкти вивчення поруч із собою і вивчати їх з усіх боків через камеру смартфона.

Було запропоновано використовувати метод `touchesBegan`, заснований на комбінації методів `hitTest` та `worldTransform` та тривимірної матриці. Цей метод дозволяє покращити точність розташування моделі над реальною поверхнею.

Для реалізації програмного продукту була використана мова програмування Swift, бібліотеки ARKit, SwiftUI та UIKit, окрім цього, були описані особливості обраної мови програмування.

В результаті виконання дипломної роботи з використанням методів об'єктно-орієнтованого програмування було спроектовано мобільний застосунок, який повністю реалізує поставлену задачу.

14

Рисунок В.14 – Слайд 14