МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДОДАТКУ ДОПОВНЕНОЇ РЕАЛЬНОСТІ

Здобувач освіти гр. ІН – 82 Арсеній БОНДАР

Науковий керівник,
асистент кафедри комп'ютерних наук Олександр ВЛАСЕНКО

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-82 спеціальності «122 – Комп'ютерні науки» денної форми навчання Бондаря Арсенія Родіоновича.

Тема: «ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДОДАТКУ ДОПОВНЕНОЇ РЕАЛЬНОСТІ»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) вибір оптимальних інструментів для розробки мобільного додатку; 3) практична реалізація

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Олександр ВЛАСЕНКО
Завдання прийняв до виконання _____ Арсеній БОНДАР

РЕФЕРАТ

Записка: 36 стор., 27 рис., 1 додаток, 10 джерел.

Об’єкт дослідження – додаток для iOS, архітектура системи

Мета роботи – розробка додатку доповненої реальності, який може нести розважальну функцію, візуалізувати об'єкти або допомагати в навчанні.

Методи дослідження – технології створення AR-додатків.

Результати – розроблено додаток доповненої реальності. Він має інтуїтивний та зручний інтерфейс, гарну продуктивність і енергоефективність. Розробка проводилась на базі мови програмування Swift за допомогою фреймворку ARKit 5.

iOS, SWIFT, ARKIT 5, AR-APPLICATION, XCODE.

ЗМІСТ

ВСТУП

Важливість мобільних телефонів у нашому повсякденному житті та діяльності, безперечно, безмежна. Її важко переоцінити, оскільки наразі відбувається величезна трансформація у галузі застосування мобільних телефонів: вони відтепер не є лише пристроєм зв'язку, яким вони були ще не так давно. Завдяки різноманітним неймовірним функціям та можливостям, які пропонують мобільні телефони, вони стали колосальним об'єктом уваги як для окремих осіб, так і для компаній. Сукупний прогрес мобільних технологій, доступність, можливість використання високошвидкісного інтернету та чудовий комунікативний інтерфейс у цих пристроях виводять мобільні обчислення на інноваційний рівень. Це стало можливим завдяки розробці мобільних додатків.

Щодня технології невпинно розвиваються, та роблять це так швидко, що ми навіть не помічаємо. Поки ви спите в одному куточку Землі, інша людина вже розробляє щось нове. Це реальність сучасного світу. Технології, які начебто вирішують одну проблему або задумувались для вирішення певної задачі, насправді мають безліч корисних секретів та фішок. AR - на перший погляд, це технологія для дозвілля. Насправді її використовують і в наукових/освітніх цілях. Наприклад, Microsoft Hololens – ці окуляри знайшли застосування у бізнесі, як військове спорядження та для демонстрації автомобілів.

2007 року компанія Apple презентувала свій перший iPhone. Він міг знімати лише фото. В якості процесора Джобс обрав Samsung S3C6400. Лише 1 ядро на 600 мгц.

Поступово, завдяки постійному вдосконаленню продукту 3-тє покоління набуло можливості знімати VGA-відео. Того часу складно було уявити, що за 12 років смартфони стануть потужнішими за переважну кількість персональних комп'ютерів. Але наразі це так, тому ми маємо, без перебільшення, безмежні можливості у створенні AR-додатків.

Метою роботи є розробка додатку доповненої реальності, який може нести розважальну функцію або допомагати в навчанні.

1. ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ

1.1. Огляд останніх досліджень і публікацій

Markets&Markets прогнозують зростання обсягу ринку мобільних AR-додатків у чотири рази – з $7,6 млрд до $29,5 млрд у 2025 році [1]. Станом на 2017 рік, коли відбувся запуск платформи ARKit, Apple мала близько 300 програм у розділі доповненої реальності. На початок 2022 року їх кількість зросла до кількох десятків тисяч - від гри з драконами та монстрами, віртуальних домашніх тварин, Lego до навчальних 3D моделей людини. Також користувачам доступні різноманітні редактори селфі з масками, примірка тату до нанесення.

Додаток IKEA Place, який допоможе "приміряти" меблі у своєму помешканні створено за допомогою фреймворку ARKit. Компанія IKEA неодноразово впроваджувала інновації, щоб залучити нових клієнтів. А саме: розробки в галузі «розумного будинку», постачання товарів за допомогою електромобілів. Компанію можна віднести до першопрохідників у дизайні інтер'єрів за допомогою доповненої реальності. Для використання програми користувач повинен просканувати квартиру камерою смартфона. Після цього ви можете вільно переміщати та масштабувати всі меблі з каталогу. Тому можна бути впевненим, що стіл або стілець з невимовною назвою гарно стане у крихітний куточок на кухні чи в кімнаті.

Vuforia Chalk – це техпідтримка, яка допоможе у будь-якій ситуації. Припустимо що є необхідність звернутися до клієнтської підтримки за допомогою. Цей додаток створено для візуальної демонстрації функцій та налаштувань. У користувача є можливість навести камеру на потрібний предмет, а технічний фахівець зробить необхідні позначки та виділить деталі.

Augment дозволяє презентувати продукт замовнику продукт онлайн. Цей AR-додаток – мрія маркетолога. Завдяки йому кожен може надіслати свої 3D чи 2D моделі через зручну форму. Макети товарів у масштабі один до одного можна розміщувати у квартирі, на полицях – реальних чи віртуальних. Покупці

інтернет-магазинів мають можливість не тільки покласти цікавий товар на свій стіл, але навіть “потримати” його у своїх руках.

Пандемія COVID-19 серйозно вплинула світові галузі. Це призвело до обмеження поїздок, соціального дистанціювання, меншої кількості персоналу на робочих місцях, відсутності навчальних інструментів та багато іншого. Однак, технологія доповненої реальності може запропонувати величезну підтримку під час кризи і збільшити потенціал ринку. Відповідно до звіту IBM про індекс роздрібної торгівлі США за 2020 рік, підприємства та їхні клієнти перейшли на цифрові покупки під час пандемії. IKEA, Home Depot, Louis Vuitton, Gucci та багато інших впровадили AR, щоб запропонувати віртуальний досвід «спробуй, перш ніж купити». Згідно з Shopify, продукти із взаємодією з контентом AR отримали 94% конверсії порівняно з продуктами без технологічної підтримки під час кризи. У промисловості робітники можуть отримати досвід від інженера, який працює вдома, щоб полагодити несправні машини та багато іншого. Наприклад, щоб допомогти працівникам під час пандемії, PTC Inc. безкоштовно надала рішення віддаленої допомоги із доповненою реальністю під назвою Vuforia Chalk. Програмні додатки забезпечують безпеку під час виробництва та технічного обслуговування. Британські лікарні впровадили платформу Virti на основі доповненої та віртуальної реальності для вирішення проблем психічного здоров'я та підтримки у навчанні. Таким чином, під час пандемії попит на програмне забезпечення для доповненої реальності значно зріс [2].

1.2. Аналіз відомих рішень

Перш ніж розробити власний AR додаток, варто ознайомитися з аналогами.

iScape: Landscape Design – це сервіс, який допомагає в ландшафтному дизайні завдяки AR технологіям [3]. Додаток дозволяє створювати неперевершені проекти для облагородження території навколо будинку,

допомагає підібрати необхідні матеріали. Також ви можете одразу побачити і відкоригувати результат, не чекаючи та не витрачаючи кошти.

Переваги додатку:

- величезна кількість ассетів;
- зручний інтерфейс;
- можливість редагувати проект з кількома користувачами;
- зберігання проекту у хмарному сховищі;
- зрозумілі підказки у додатку.

Рисунок 1.1 – Скріншот додатку iScape

До негативних моментів можна віднести доволі значну суму за річну підписку та незручну реєстрацію. Та найголовніше – при запуску режиму AR додаток просто зависає. Для демонстрації роботи додатку зроблено 5 холодних перезапусків, але результату отримано не було. Пристрій запуску – iPhone 11, версія iOS – 15.5 (останнє оновлення). Саме тому приклад екрану додатку завантажено з офіційного сайту Apple.

Наступним додатком в огляді буде мобільна AR-гра Hot Lava [4]. Додаток використовує нові можливості iPhone 13 Pro та iPad Pro сканер LiDAR. Дана програма з легкістю дає можливість перетворити кімнату на майданчик для ігор. Основна мета – довести до кінцевої точки головного героя, який стрибатиме з реальних предметів в кімнаті на ігрові.

Рисунок 1.2 – Екран додатку Hot Lava

Переваги даного додатку:

- чудова оптимізація;
- приємний інтерфейс;
- красиві анімації та неперевершена інтеграція реального світу у гру.

До недоліків додатку можна віднести лише величезний розмір розмір, майже 3 гігабайта.

1.3. Постановка задачі

У результаті аналізу та огляду існуючих аналогів визначено мету даної роботи – проектування та розробка додатку доповненої реальності, який допомагатиме у навчанні, візуалізує об'єкти або має розважальну функцію. Додаток повинен мати інтуїтивний та зручний інтерфейс, гарну продуктивність і енергоефективність. Для досягнення мети були поставлено наступні завдання:

- Огляд та вибір мови програмування для додатку.
- Аналіз IDE xCode.
- Вибір фреймворку для реалізації.
- Розробка AR-додатку.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ ДЛЯ РЕАЛІЗАЦІЇ IOS AR APPLICATION

2.1. Інструменти та теоретична частина розробки

Apple одна з найбільших компаній з виробництва смартфонів, ноутбуків та іншої електроніки, але на відміну від більшості інших компаній Apple самостійно створює програмне забезпечення для своїх пристроїв; надає інфраструктуру для простого та зручного написання швидких та ефективних додатків під iOS або MacOS. До цієї інфраструктури входить середовище розробки Xcode, мови Swift та Objective-C, а також різні бібліотеки та фреймворки. Також щороку в Каліфорнії проводиться Worldwide Developers Conference (WWDC) – всесвітня конференція для розробників на платформах Apple. На цих конференціях анонсують нові продукти, розробки та оновлення платформ Apple.

Основним інструментом розробки для macOS та iOS – є Xcode. Він включає у собе інструменти для створення додатків під усі актуальні платформи компаніі Apple. А саме: Mac, iPad, Apple Watch, iPhone та Apple TV. В IDE xCode користувач знайде конструктор інтерфейсу, редактор коду, емулятори всіх пристроїв, графічні інструменти для аналізу продуктивності додатку. Є можливість побачити навантаження на відеокарту, центральний процесор та оперативну пам'ять.

Поточна версія включає: мову програмування Swift 5, нову мовну версію, інструмент для навчання Swift Playground (ексклюзивно для iPad), більш швидку версію Interface Builder, в якій можна переглядати макет, зменшений розмір програми за рахунок відсутності стандартних динамічних бібліотек. У IDE Xcode 13 можна розробляти програми для tvOS 15, iOS 15, macOS Monterey та watchOS 8.

2.2. Вибір мови реалізації

Історично склалося так, що розробка програмного забезпечення для iOS та MacOS виконувалася на Objective-C. Нещодавно випущена мова Swift може замінити Objective-C. Хоча остання активно використовується, вона застаріла і не має майбутнього. Swift відмінно підходить для швидкого навчання та розробки. Ставлячи завдання розробити альтернативу Objective-C, компанія Apple висунула лише дві основні вимоги: мова повинна бути простою в освоєнні і максимально прискорювати цикл розробки додатків.

Таким чином, Swift має всі характеристики сучасної мови програмування і, безсумнівно, перевершує Objective-C у всіх відносинах. Основні функції безпеки:

- явна обробка значень nil (null);
- немає помилок із розміром масивів;
- автоматичне керування пам'яттю;
- немає помилок переповнення;
- немає невизначених або неініціалізованих змінних.

У результаті користувач може витрачати більше часу на реалізацію ідей, не турбуючись про можливі помилки, збої або конфлікти в коді. Також Swift усуває багатослівність синтаксису в Objective-C, спрощуючи написання коду та його читання надалі. За даними Apple мова Swift до 7 разів швидше за Python і в 2,2 рази швидше за Objective-C [5].

Важливо, що Swift не лише швидкий, але й сповнений сучасних можливостей, що дозволяють писати функціональний код. До них відносяться:

- ітератори;
- замикання;
- множинні повернення;
- дженерики;
- вбудовані шаблони функціонального програмування;
- кортежі.

Введення цих функцій, поряд з покращенням синтаксису, робить Swift безпечнішим, ніж Objective-C. Покращена обробка пам'яті означає меншу

можливість несанкціонованого доступу до даних. Переміщення в неправильну частину пам'яті, неправильні зміни даних також утруднені. Ефективна обробка помилок значно знижує кількість збоїв та критичних сценаріїв.

Apple зробила Swift безкоштовною мовою з відкритим кодом через рік після його запуску. Компанії такого рівня нечасто вирішуються на подібний крок, хоча у сучасному світі це не безпрецедентний випадок. Користувачі постійно пропонують способи виправлення помилок та покращення функціональності.

Згідно зі звітом GitHub Octoverse 2021 [6], Swift є 13-ою за популярністю мовою для проектів з відкритим вихідним кодом. В опитуванні StackOverflow 2021 [7] Swift займає впевнену позицію в улюблених мовах серед активних розробників. Дивлячись крізь призму 8 років розробки та покращень мови Swift, можна сказати, що її популярність тільки зростатиме, як і попит на розробників.

2.3. Фреймворк ARKit

Технології доповненої реальності (AR) наразі руйнують бар'єр між реальним і технологічно створеним світами. ARKit відкриває нові перспективи для ігор, роботи, навчання, та взаємодії з оточенням. Тепер можна побачити те, що насправді неможливо показати або надто складно реалізувати.

На конференції для розробників WWDC 2017 року компанія Apple презентувала платформу для доповненої реальності, в основі якої мільйони пристроїв з підтримкою AR, а також тисячі додатків у магазині App Store, розроблених спеціально для доповненої реальності [8].

Оскільки пристрої і програмне забезпечення Apple відразу проектуються з урахуванням технологій AR, усі вони просто ідеально підходять для занурення у доповнену реальність.

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 містить у собі трекінг руху

пристрою, зйомку сцени камерою, розширену обробку сцени та зручність відображення, щоб спростити завдання створення доповненої реальності. Застосування цих технологій з використанням передньої або задньої камери пристрою iOS робить можливим створення багатьох видів AR.

Завдяки ARKit 2 на iOS 12 програми AR тепер можуть використовуватися кількома користувачами одночасно і відновлюватися пізніше в тому ж стані. Також допускається включення реальних об’єктів у власний досвід доповненої реальності, що надає користувачам ще більші можливості для занурення.

ARKit 3 додав можливість природньо показувати AR-контент перед або позаду людей (використовуючи People Occlusion), відстежувати до трьох обличь одночасно, підтримує спільні сеанси тощо. Також в ARKit 3 з’явилася функція одночасного використання відстеження обличь і навколишнього світу на фронтальній і головній камерах, що відкриває нові можливості. Наприклад, користувач може взаємодіяти з контентом доповненої реальності на задній камері, використовуючи лише своє обличчя.

ARKit 4 на iPadOS містить абсолютно новий API Depth, який надає легкий доступ до детальної інформації про глибину навколишнього середовища, зібрану сканером LiDAR на iPad Pro. Це дозволяє використовувати попіксельну інформацію про глибину довкілля. У поєднанні з даними 3D-мережі, створеними геометрією сцени, інформація про глибину робить перекриття віртуальних об'єктів ще більш реалістичним, що дозволяє миттєво розміщувати віртуальні об'єкти та плавно змішувати їх із фізичним оточенням.

ARKit 5 має вдосконалене відстеження руху та підтримку відстеження обличчя в надширококутній камері на iPad Pro, Motion Capture в режимі реального часу за допомогою однієї камери. Розуміючи положення тіла як серію суглобів і кісток, ви можете використовувати рух і пози як вхідні дані для AR. Мінімальні технічні вимоги для ARKit це iOS/iPadOS 11 та мобільний чіп Apple A9.

Рисунок 2.1 – Пошарова структура фреймворку

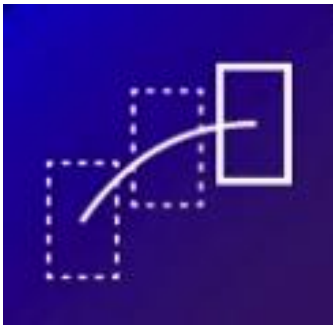

Рисунок 2.2 – Tracking

Tracking – відстежування змін положення телефона завдяки сенсорам. Це основна (базова) функціональність ARKit.

Рисунок 2.3 – SceneUnderstanding

Scene Understanding – розуміння сцени (того, що відображається через камеру), а саме відстань до об’єкта, освітлення. У свою чергу, Scene Understanding містить у собі такі складові: Plane Detection, Hit-testing, Light estimation.

Рисунок 2.4 – Rendering

Rendering – коли програма отримала всі необхідні дані, вона готова рендерити потібний кадр. Rendering складається з таких компонентів: Easy integration, AR views, Custom rendering.

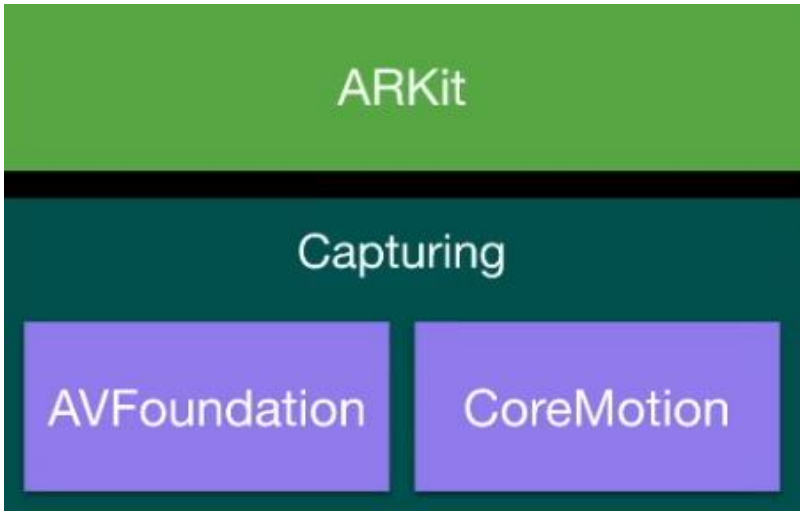

Рисунок 2.5 – Складові ARKit

У основі ARKit є 2 фреймворка. Це AVFoundation, який відповідає за захоплення зображення, та CoreMotion для отримання даних з сенсорів пристрою.

2.4. Архітектура операційної системи iOS

iOS – це мобільна операційна система, розроблена компанією Apple Inc. для iPhone, iPad та інших мобільних пристроїв Apple. iOS є другою за

популярністю та найбільш використовуваною мобільною операційною системою після Android.

Архітектура операційної системи iOS базується на кількох рівнях. Обмін інформацією між ними не відбувається безпосередньо. Нижній рівень надає базові послуги, на які покладаються всі програми, а вищі рівні надають надають графіку та послуги, пов’язані з інтерфейсом. Більшість системних інтерфейсів постачаються зі спеціальним пакетом, який називається фреймворком [9].

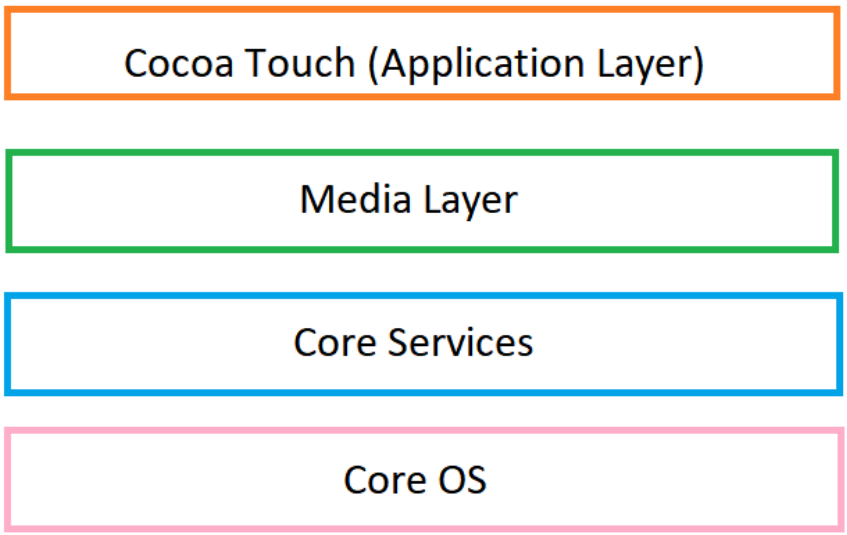

Рисунок 2.6 – Архітектура iOS

Фреймворк – це каталог, який містить динамічні спільні бібліотеки, такі як файли, файли заголовків, зображення та допоміжні програми, які підтримують бібліотеку. Кожен рівень має набір фреймворків, корисних для розробників [10].

Розглянемо більш детально кожен шар:

CORE OS Layer:

Усі технології iOS побудовані на найнижчому рівні, тобто на CoreOS. Ці технології включають:

1. Core Bluetooth Framework
2. External Accessories Framework
3. Accelerate Framework
4. Security Services Framework

5. Local Authorization Framework

CORE SERVICES Layer:

Деякі важливі фреймворки присутні на рівні Core SERVICES, який допомагає операційній системі iOS забезпечувати кращу функціональність. Це 2-й шар в архітектурі, як показано на рисунку вище. Далі наведено кілька важливих фреймворків, наявних у цьому шарі:

6. Address Book Framework
7. Cloud Kit Framework
8. Core Data Framework
9. Core Foundation Framework
10. Core Location Framework
11. Core Motion Framework
12. Foundation Framework
13. HealthKit Framework
14. HomeKit Framework
15. Social Framework
16. StoreKit Framework

MEDIA Layer:

За допомогою медіарівня ми маємо можливість підключити всі відео- та аудіосистеми. Основні фреймфорки:

17. ULKit Graphics
18. Core Graphics Framework
19. Core Animation
20. Media Player Framework
21. AV Kit
22. Open AL
23. Core Images
24. GL Kit

COCOA TOUCH:

Цей рівень діє як інтерфейс для роботи користувача з операційною системою iOS. Він підтримує події дотику до екрана, анімації та багато інших функцій. Шар COCOA TOUCH забезпечує наступні фреймворки:

1. EvenKit Framework
2. GameKit Framework
3. MapKit Framework
4. PushKit Framework

Розглянемо переваги операційної системи iOS:

1. Безпечніша за інші операційні системи.
2. Захоплюючий UX та інтерфейс користувача, швидке реагування.
3. Найкраще підходить для бізнес-користувачів та професіоналів.
4. Незрівнянна екосистема серед мобільних систем.
5. Має дуже потужні API, зокрема і камери пристрою.

Недоліки операційної системи iOS:

1. Якщо ви вирішили приєднатися до користувачів, це потребує чимало коштів. Телефони команії Apple – лише флагмани.

2. Менш гнучка стосовно налаштувань порівняно з операційною системою Android.

3. Закритість системи, підтримує лише телефони від компанії Apple.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ AR-ДОДАТКУ ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ ARKIT 5

Для початку розробки додатку потрібно встановити IDE xCode останньої версії. Після створення проекту буде запропоновано обрати один з наданих шаблонів додатку.

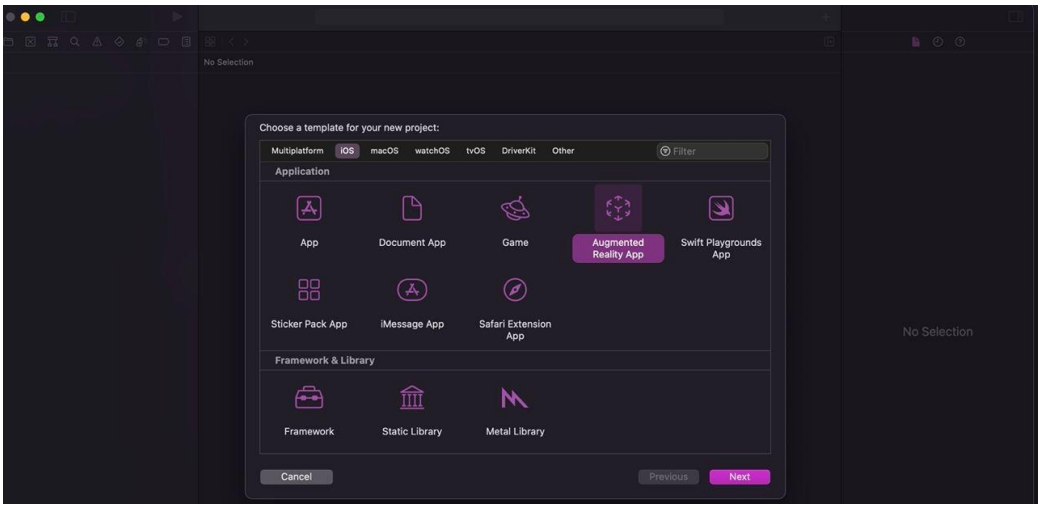

Рисунок 3.1 – Вибір шаблону

На наступному кроці обираємо ім’я проєкту та команду розробки. Це дуже важливий крок, бо без нього неможливо встановити додаток на iPhone для проведення тестування. В IDE xCode наявна функція емуляції пристроїв, але це не дозволить коректно зробити усі перевірки AR-додатку, бо потрібна камера.

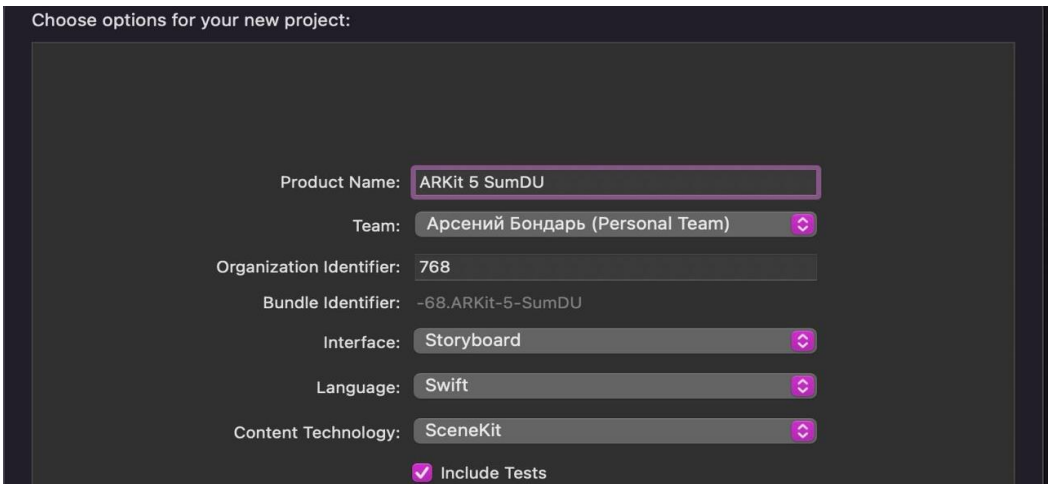

Рисунок 3.2 – Назва додатку, вибір мови програмування

Далі потрібно імпортувати важливі бібліотеки, а саме: UIKit, SceneKit, ARKit. Для спрощення отримання діагностичних даних під час тестування додаю у клас ViewController властивість sceneView.showsStatistics з параметром true.

```
7
8  import UIKit
9  import SceneKit
10 import ARKit
11
12 class ViewController: UIViewController, ARSCNViewDelegate {
13
14     @IBOutlet var sceneView: ARSCNView!
15
16     override func viewDidLoad() {
17         super.viewDidLoad()
18
19         // Set the view's delegate
20         sceneView.delegate = self
21
22
23         sceneView.showsStatistics = true
24
```

Рисунок 3.3 – Імпорт бібліотек

Створюємо функцію для віртуального об’єкта та додаємо паузу при згортанні додатку. Це необхідно для збереження батареї та коректного використання ресурсів пристрою.

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


    let configuration = ARWorldTrackingConfiguration()


    sceneView.session.run(configuration)
}

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


    sceneView.session.pause()
}
```

Рисунок 3.4 – Функція для віртуального об’єкта

Для розміщення об’єкта на поверхні потрібно додати деяку площину. Тому наступним кроком буде створення файлу Plane.swift. Обов’язково імпортуємо необхідні бібліотеки SceneKit та ARKit та створюю клас Plane, який, у свою чергу, наслідує клас SCNNode. Властивість Anchor дозволяє розміщувати об’єкт з прив’язкою до певних координат. planeGeometry передає розмір поверхні. Також створюємо метод для ініціалізації.

```
1  import SceneKit
2  import ARKit
3
4  class Plane: SCNNode {
5
6      var anchor: ARPlaneAnchor!
7      var planeGeometry: SCNPlane!
8
9      init(anchor: ARPlaneAnchor) {
10         self.anchor = anchor
11         super.init()
12         configure()
13     }
```

Рисунок 3.5 – Створення площини

Коли на екрані багато об’єктів, не обов’язково вони повинні взаємодіяти між собою. Наприклад, літак може без перешкод пролетіти крізь хмару, не відскочивши від неї. Для визначення взаємодії потрібна бітова маска. Створюємо файл BitMaskCategory, а у ньому структуру з таким же ім’ям.

```
struct BitMaskCategory {

    static let none  = 0 << 0
    static let box   = 1 << 0
    static let plane = 1 << 1
}
```

Рисунок 3.6 – Бітова маска

Далі потрібна функція створення віртуального об’єкта createVirualObject, ставтмо параметри розташування. Для полегшення розробки додамо перевірку на наявність об’єкта. Якщо він не доступний – у консолі буде відповідне повідомлення.

```
func createVirtualObject(hitResult: ARHitTestResult) {

    let position = SCNVector3(hitResult.worldTransform.columns.3.x,
                              hitResult.worldTransform.columns.3.y,
                              hitResult.worldTransform.columns.3.z)

    guard let virtualObject = VirtualObject.availableObjects.first else { fatalError("No
        virtual object available") }

    virtualObject.load()
    virtualObject.position = position

    if let particleSystem = SCNParticleSystem(named: "Smoke.scnp", inDirectory: nil), let
        smokeNode = virtualObject.childNode(withName: "SmokeNode", recursively: true) {

        smokeNode.addParticleSystem(particleSystem)
    }

    sceneView.scene.rootNode.addChildNode(virtualObject)
}
```

Рисунок 3.7 – Функція створення віртуального об’єкта

func placeVirtualObject розміщує об’єкт та прив’язує його до навколишнього світу.

```
@objc func placeVirtualObject(tapGesture: UITapGestureRecognizer) {

    let sceneView = tapGesture.view as! ARSCNView
    let location = tapGesture.location(in: sceneView)

    let hitTestResult = sceneView.hitTest(location, types: .existingPlaneUsingExtent)
    guard let hitResult = hitTestResult.first else { return }

    createVirtualObject(hitResult: hitResult)
}
```

Рисунок 3.8 – Функція розміщення віртуального об’єкта

На наступному кроці додаємо фунцію setupGestures. Вона дозволяє зчитувати жести, дотики до екрану пристрою. Параметр tapGestureRecognizer.numberOfTapsRequired = 1 говорить про те, що для певної

дії потрібен 1 дотик. Функція placeVirtualObject розміщає об’єкт за заданими вище координатами.

```
func setupGestures() {

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
        #selector(placeVirtualObject(tapGesture:)))
    tapGestureRecognizer.numberOfTapsRequired = 1
    self.sceneView.addGestureRecognizer(tapGestureRecognizer)
}
```

Рисунок 3.9 – Функція для зчитування жестів, дотиків

На цьому етапі завантажуємо з сайту Free3d будь-яку модель у потрібному форматі (.dae), додаємо її до проєкту та робимо деякі корективи. А саме: виставляємо розмір моделі за всіма осями 0.02 та повертаю на 90° за віссю *x*.

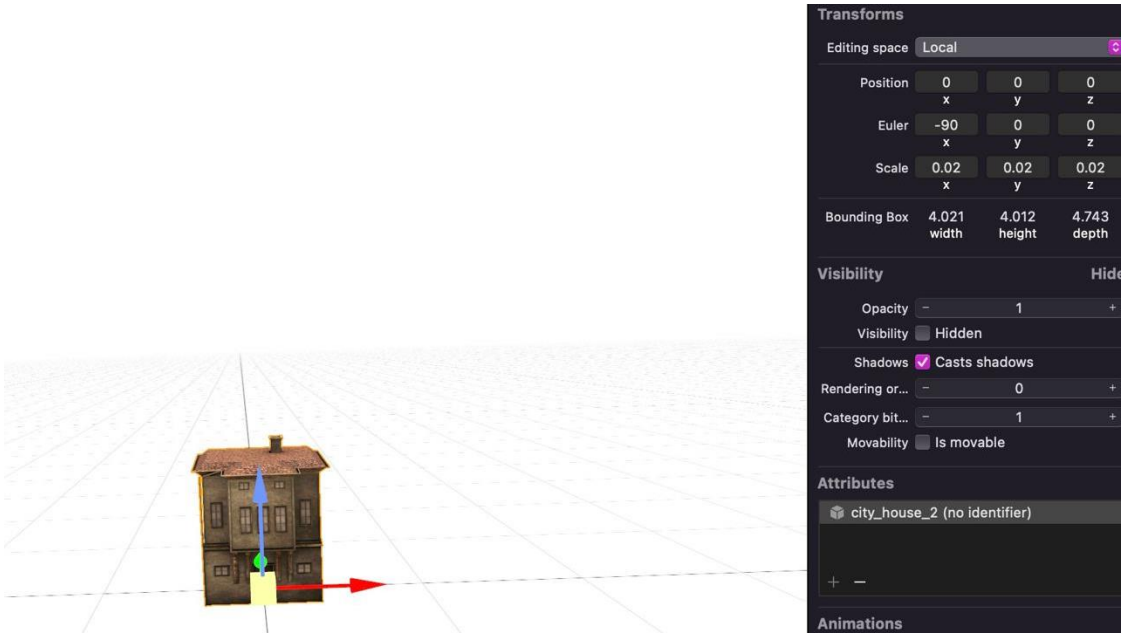

Рисунок 3.10 – 3D модель будинку

Для реалізму необхідно додати ефект диму з труби. Для цього створюємо файл для подальшої роботи з частинками (particles).

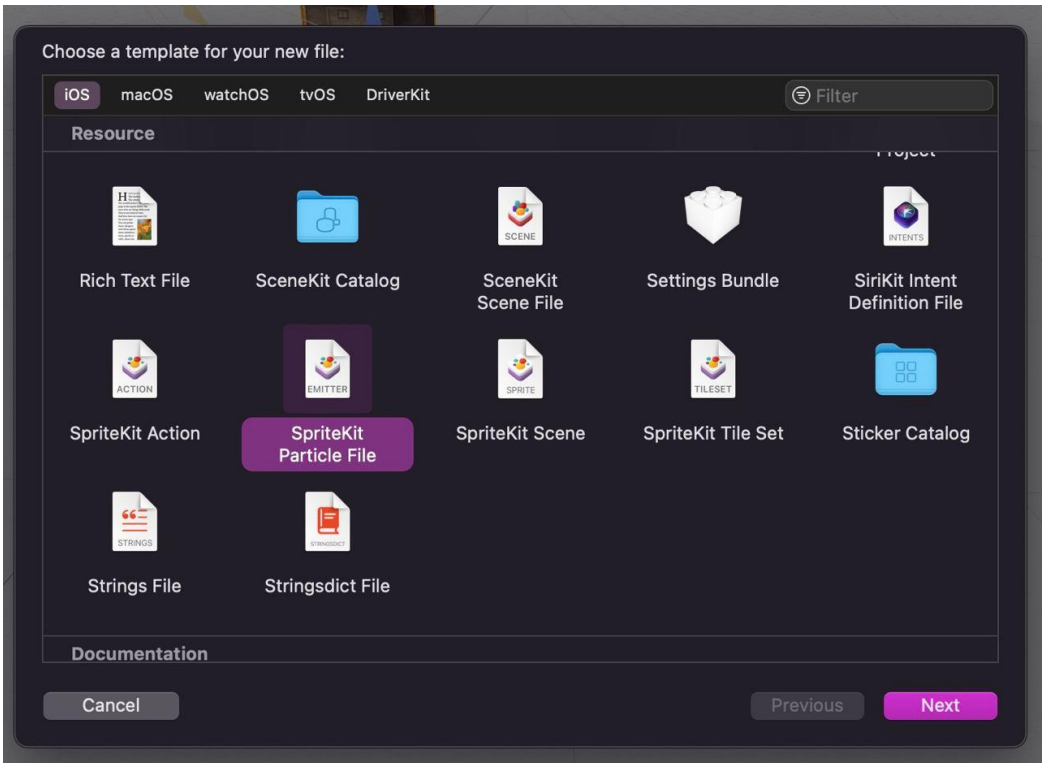

Рисунок 3.11 – Створення файлу

В xCode наявні деякі шаблони, обираю Smoke.

Рисунок 3.12 – Вибір шаблону частинок

Ефект диму за замовчуванням буде від початку координат. У нашому випадку це низ будинку. Для прив'язки його до труби створюємо додаткову ноду та додаємо її до коду проєкту.

```
if let particleSystem = SCNParticleSystem(named: "Smoke.scnp", inDirectory: nil), let
    smokeNode = virtualObject.childNode(withName: "SmokeNode", recursively: true) {

    smokeNode.addParticleSystem(particleSystem)
}
```

Рисунок 3.13 – Створення додаткової ноди

Далі потрібно налаштувати розмір, колір, швидкість та напрям для досягнення чудового результату.

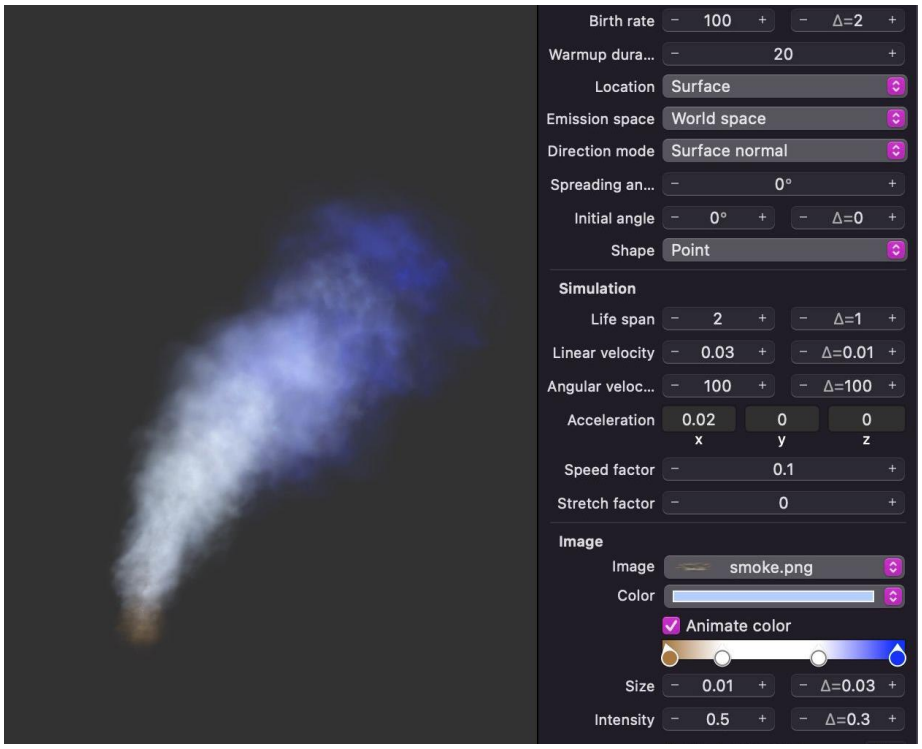

Рисунок 3.14 – Налаштування ефекту диму

Під час дотику до екрану будинок займає нове положення. Але у ході тестування було виявлено деяку проблему при цьому. ParticleSystems не тільки переміщується разом з об’єктом, але й дублюється. Тому було додано до функції placeVirtualObject код, який видаляє усі ParticleSystems, і тільки потім створюється об’єкт.

```
@objc func placeVirtualObject(tapGesture: UITapGestureRecognizer) {

    self.sceneView.scene.removeAllParticleSystems()

    let sceneView = tapGesture.view as! ARSCNView
    let location = tapGesture.location(in: sceneView)

    let hitTestResult = sceneView.hitTest(location, types: .existingPlaneUsingExtent)
    guard let hitResult = hitTestResult.first else { return }

    createVirtualObject(hitResult: hitResult)
}
```

Рисунок 3.15 – Видалення клонів частинок

Для покращення зображення було додано Spot Light (спрямоване світло) та Ambient Light (навколишнє світло) та переміщено джерела світла у потрібні місця.

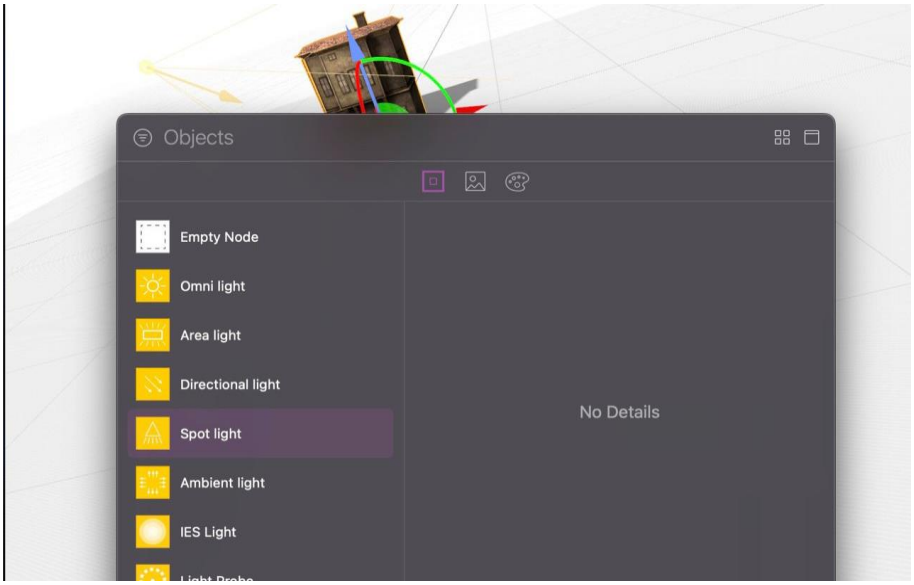

Рисунок 3.16 – Додавання освітлення сцени

На додаванні джерел світла робота не закінчилася, тому що тінь повинна падати на певну поверхню. Наступним кроком є створення прозорої поверхні.

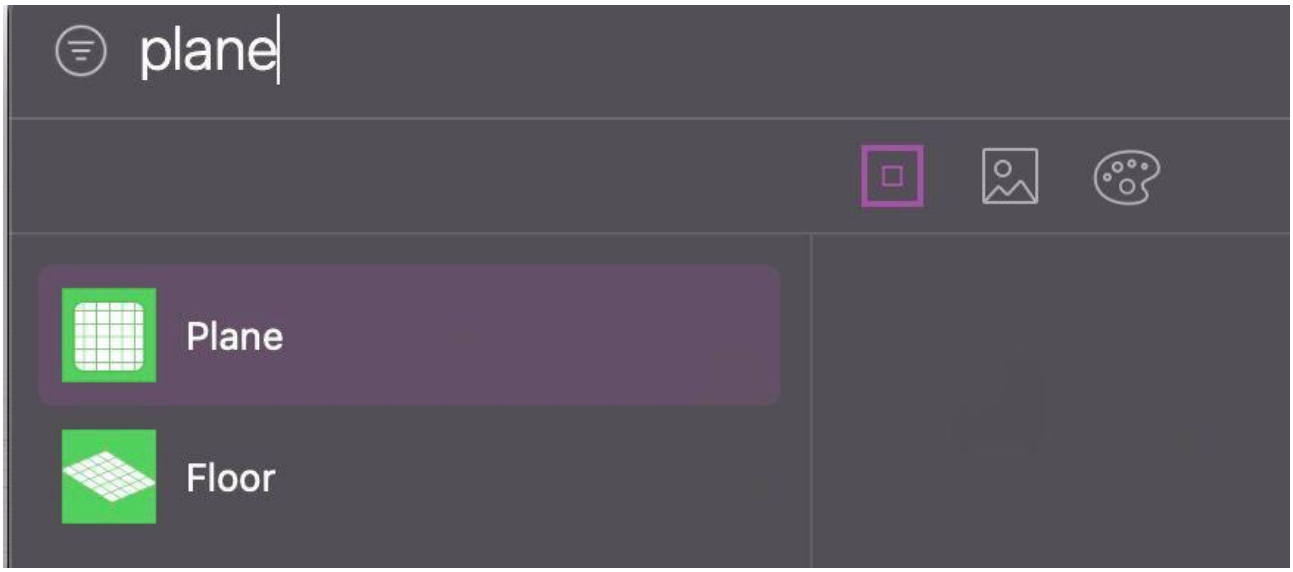

Рисунок 3.17 – Додавання поверхні

На рисунку показана тінь від об’єкта та розташування джерела світла.

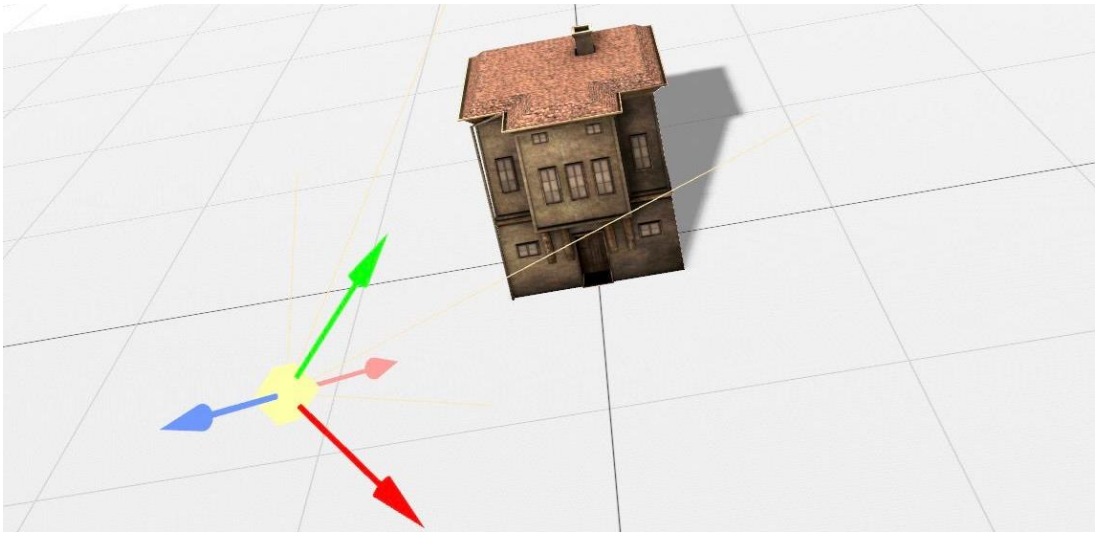

Рисунок 3.18 – Фінальний результат сцени

Результат роботи розробленого додатку показано на рисунку 3.19:

Рисунок 3.19 – Приклад роботи розробленого AR-додатку

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було:

- розглянуто та проаналізовано дві сучасні мови програмування (Swift та Objective-C), які використовуються для створення додатків для iOS, iPadOS та MacOS;
- проведено аналіз тематичної літератури;
- вибір програмних засобів для реалізації проекту;
- розроблено додаток доповненої реальності.

Розроблений додаток віртуальної реальності має інтуїтивний та зручний інтерфейс, гарну продуктивність і енергоефективність. Розробка проводилась на базі мови програмування Swift за допомогою фреймворку ARKit 5.

СПИСОК ЛІТЕРАТУРИ

1. Flexreality [Електронний ресурс] – Режим доступу: https://flexreality.pro/

2. Fortune Business Insights [Електронний ресурс] – Режим доступу: https://www.fortunebusinessinsights.com/augmented-reality-ar-market-102553

3. iScape [Електронний ресурс] – Режим доступу: https://apps.apple.com/ua/app/iscape-outdoor-home-designer/id439688430?platform=iphone

4. HotLava [Електронний ресурс] – Режим доступу: https://apps.apple.com/ua/app/hot-lava/id1469252166

5. SwiftBook [Електронний ресурс] – Режим доступу: https://docs.swift.org/swift-book/#

6. GitHub Octoverse 2021 [Електронний ресурс] – Режим доступу: https://octoverse.github.com/static/octoverse-report-2021.pdf

7. StackOverflow Survey 2021 [Електронний ресурс] – Режим доступу: https://insights.stackoverflow.com/survey/2021#technology

8. ARKit DOC [Електронний ресурс] – Режим доступу: https://developer.apple.com/documentation/arkit/#overview

9. iOS architecture [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/architecture-of-ios-operating-system/

10. iOS architecture [Електронний ресурс] – Режим доступу: https://www.foxrichcode.com/page/52/ios-operating-system-architecture

ДОДАТОК А

Вихідний код

AppDelegate.swift

```
import UIKit
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
   var window: UIWindow?
   func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey:
Any]?) -> Bool {
      return true
   }
   func applicationWillResignActive(_ application: UIApplication) {
   }
   func applicationDidEnterBackground(_ application: UIApplication) {
   }
   func applicationWillEnterForeground(_ application: UIApplication) {
   }
   func applicationDidBecomeActive(_ application: UIApplication) {
   }
   func applicationWillTerminate(_ application: UIApplication) {
   }
}
```

ViewController.swift

```
import UIKit
import SceneKit
import ARKit
```

```
class ViewController: UIViewController {
    @IBOutlet var sceneView: ARSCNView!
    var planes = [Plane]()
    override func viewDidLoad() {
        super.viewDidLoad()
        sceneView.delegate = self
        sceneView.showsStatistics = true
        sceneView.autoenablesDefaultLighting = true
        let scene = SCNScene()
        sceneView.scene = scene
        setupGestures()
    }
    func setupGestures() {
        let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(placeVirtualObject(tapGesture:)))
        tapGestureRecognizer.numberOfTapsRequired = 1
        self.sceneView.addGestureRecognizer(tapGestureRecognizer)
    }
    @objc func placeVirtualObject(tapGesture: UITapGestureRecognizer) {
        let sceneView = tapGesture.view as! ARSCNView
        let location = tapGesture.location(in: sceneView)

        let hitTestResult = sceneView.hitTest(location, types:
.existingPlaneUsingExtent)
        guard let hitResult = hitTestResult.first else { return }
        createVirtualObject(hitResult: hitResult)
    }
    func createVirtualObject(hitResult: ARHitTestResult) {
        let position = SCNVector3(hitResult.worldTransform.columns.3.x,
                                  hitResult.worldTransform.columns.3.y,
```

```
                                hitResult.worldTransform.columns.3.z)
            guard let virtualObject = VirtualObject.availableObjects.first else {
fatalError("There is no virtual object available") }
            virtualObject.load()
            virtualObject.position = position
            if let particleSystem = SCNParticleSystem(named: "Smoke.scnp",
inDirectory: nil), let smokeNode = virtualObject.childNode(withName:
"SmokeNode", recursively: true) {
                smokeNode.addParticleSystem(particleSystem)
            }
            sceneView.scene.rootNode.addChildNode(virtualObject)
        }
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            let configuration = ARWorldTrackingConfiguration()
            configuration.planeDetection = .horizontal
            sceneView.session.run(configuration)
        }
        override func viewWillDisappear(_ animated: Bool) {
            super.viewWillDisappear(animated)
            sceneView.session.pause()
        }
    }
    extension ViewController: ARSCNViewDelegate {
        func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for
anchor: ARAnchor) {
            guard anchor is ARPlaneAnchor else { return }
            let plane = Plane(anchor: anchor as! ARPlaneAnchor)
            print("Plane is detected and created-->")
            self.planes.append(plane)
```

```
            node.addChildNode(plane)
        }
        func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode,
for anchor: ARAnchor) {
            let plane = self.planes.filter { plane in
                return plane.anchor.identifier == anchor.identifier
                }.first
            guard plane != nil else { return }
            plane?.update(anchor: anchor as! ARPlaneAnchor)
        }
    }
```

BitMaskCategory.swift

```
    struct BitMaskCategory {
        static let none  = 0 << 0
        static let box   = 1 << 0
        static let plane = 1 << 1
    }
```

Plane.swift

```
    import SceneKit
    import ARKit
    class Plane: SCNNode {
        var anchor: ARPlaneAnchor!
        var planeGeometry: SCNPlane!
        init(anchor: ARPlaneAnchor) {
            self.anchor = anchor
            super.init()
```

```
            configure()
        }
        private func configure() {
            self.planeGeometry = SCNPlane(width: CGFloat(anchor.extent.x), height:
CGFloat(anchor.extent.z))
            let material = SCNMaterial()
            material.diffuse.contents = UIColor.clear // UIImage(named:
"pinkWeb.png")
            self.planeGeometry.materials = [material]
            self.geometry = planeGeometry
            let physicsShape = SCNPhysicsShape(geometry: self.geometry!, options:
nil)
            self.physicsBody = SCNPhysicsBody(type: .static, shape: physicsShape)
            self.physicsBody?.categoryBitMask = BitMaskCategory.plane
            self.physicsBody?.collisionBitMask = BitMaskCategory.box
            self.physicsBody?.contactTestBitMask = BitMaskCategory.box
            self.position = SCNVector3(anchor.center.x, 0, anchor.center.z)
            self.transform = SCNMatrix4MakeRotation(Float(-Double.pi / 2), 1.0,
0.0, 0.0)
        }
        func update(anchor: ARPlaneAnchor) {
            self.planeGeometry.width = CGFloat(anchor.extent.x)
            self.planeGeometry.height = CGFloat(anchor.extent.z)
            self.position = SCNVector3(anchor.center.x, 0, anchor.center.z)
        }
        required init?(coder aDecoder: NSCoder) {
            fatalError("init(coder:) has not been implemented")
        }
    }
```

VirtualObject.swift

```
import SceneKit
class VirtualObject: SCNReferenceNode {
   static let availableObjects: [SCNReferenceNode] = {
      guard let modelsURLs = Bundle.main.url(forResource: "art.scnassets",
withExtension: nil) else { return [] }
      let fileEnumirator = FileManager().enumerator(at: modelsURLs,
includingPropertiesForKeys: nil)!
      return fileEnumirator.flatMap{ element in
         let url = element as! URL
         guard url.pathExtension == "scn" else { return nil }
         return VirtualObject(url: url)
      }
   }()
}
```