Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

О. С. Шкурат

XR-ЗАСТОСУНКИ

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»
спеціальності 121 Інженерія програмного забезпечення

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2026

УДК 004.9 (004.4'27)

Автор: *Шкурат Оксана Сергіївна*, канд. техн. наук

Рецензент *Дичка Іван Андрійович,* д-р техн. наук, проф.,
декан факультету програмних систем та прикладної математики

Відповідальний
редактор *Заболотня Тетяна Миколаївна*, канд. техн. наук, доц.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 05.03.2026 р.)
за поданням Вченої ради факультету програмних систем та прикладної математики
(протокол № 9 від 23.02.2026 р.)

Шкурат О. С.

XR-застосунки [Електронний ресурс] : лаб. практикум : навчальний посібник для здобувачів ступеня бакалавр за освітньою програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем» спеціальності 121 Інженерія програмного забезпечення / О. С. Шкурат ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 19,9 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2026. – 160 с. – Назва з екрана.

Навчальний посібник надає покрокові інструкції для розроблення застосунків доповненої, змішаної та віртуальної реальностей в ігровому рушії Unity для виконання чотирьох лабораторних робіт. Навчальний посібник містить контрольні запитання, які дозволяють здобувачу освіти перевірити засвоєння практичних навичок та рекомендовану літературу, яка дозволить поглибити практичні вміння для розроблення систем розширеної реальності різного призначення. Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення» факультету програмних систем та прикладної математики НТУУ «КПІ ім. Ігоря Сікорського».

УДК 004.9 (004.4'27)

Реєстр. № НП 25/26-266. Обсяг 4 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

API (*Application Programming Interface*) – прикладний програмний інтерфейс.

AR (*Augmented Reality*) – доповнена реальність.

MR (*Mixed Reality*) – змішана реальність.

SDK (*Software Development Kit*) – набір інструментів для розроблення програмного забезпечення.

UI (*User Interface*) – інтерфейс користувача.

UX (*User Experience*) – досвід користувача.

VR (*Virtual Reality*) – віртуальна реальність.

XR (*Extended Reality*) – розширена реальність.

ВСТУП

Розширена реальність (*Extended Reality*, *XR*) є узагальнювальним терміном, що охоплює сукупність імерсивних технологій, зокрема доповнена реальність, змішана реальність та віртуальна реальність (рис. 1). Ключовою особливістю *XR* є забезпечення інтерактивної взаємодії користувача з цифровими об’єктами в реальному часі.

Доповнена реальність (*Augmented Reality*, *AR*) є імерсивною технологією, що забезпечує накладання мультимедійних цифрових об’єктів – зображення, *3D*-моделі та анімації, текст, звук, відео – на об’єкти фізичного середовища. *AR*-технологія робить цифрові об’єкти частиною фізичного середовища, створюючи інтерактивний досвід для користувача.

Змішана реальність (*Mixed Reality*, *MR*) є імерсивною технологією, що забезпечує інтегрування мультимедійних цифрових об’єктів та взаємодію з об’єктами фізичного середовища. *MR*-технологія поєднує цифрові об’єкти з фізичними об’єктами та забезпечує їхню взаємодію відповідно до дій користувача в реальному часі.

Віртуальна реальність (*Virtual Reality*, *VR*) є імерсивною технологією, що забезпечує генерування інтерактивного цифрового середовища, відокремленого від фізичного середовища. *VR*-технологія робить користувача частиною цифрового середовища, забезпечуючи інтерактивну взаємодію з цифровими об’єктами та самим середовищем у реальному часі.

Розширена реальність

Extended Reality, XR

Розширена реальність є узагальнювальним терміном, що охоплює сукупність імерсивних технологій, зокрема доповнена реальність, змішана реальність та віртуальна реальність.

XR-технології спрямовані на створення інтерактивних середовищ, у яких відбувається поєднання фізичного та цифрового середовищ або повна заміна фізичного середовища цифровим.

XR-технології ґрунтується на використанні різних обчислювальних та сенсорних пристроїв, зокрема смартфонів, планшетів, смарт-окулярів, шоломів змішаної та віртуальної реальності тощо. Такі пристрої виконують функції відображення цифрового середовища, збору початкових вхідних даних користувача, а також безперервного отримання зворотного зв'язку під час взаємодії з цифровим середовищем. До таких даних належать положення користувача в просторі, рухи голови та рук, жести, напрямок погляду, а також сенсорні або контролерні дії.

01

Доповнена реальність

Augmented Reality, AR

Доповнена реальність — це імерсивна технологія, яка накладає мультимедійні цифрові об'єкти — зображення, 3D-моделі, анімації, текст, звук і відео — на фізичне середовище, доповнюючи його та забезпечуючи інтерактивну взаємодію користувача з цифровими об'єктами.

02

Змішана реальність

Mixed Reality, MR

Змішана реальність — це імерсивна технологія, яка інтегрує мультимедійні цифрові об'єкти — зображення, 3D-моделі, анімації, текст, звук і відео — у фізичне середовище та забезпечує інтерактивну взаємодію користувача з цифровими й фізичними об'єктами в реальному часі.

03

Віртуальна реальність

Virtual Reality, VR

Віртуальна реальність — це імерсивна технологія, яка генерує цифрове середовище — зображення, 3D-моделі, анімації, текст, звук і відео — та забезпечує інтерактивну взаємодію користувача з об'єктами цифрового середовища в реальному часі.

Рис. 1 – *XR*-технології

XR ґрунтується на технологіях відстеження фізичного середовища, реєстрації, рендерингу та взаємодії на основі *3D*-інтерфейсів. Апаратні *XR*-пристрої безперервно збирають та передають дані щодо положення та орієнтації користувача, рухів голови та рук, жести, погляд, а також візуалізують цифрове середовище. Спеціалізовані програмні фреймворки (*Unity*, *Unreal Engine*, *WebXR* та інші) та інструменти (*ARKit*, *ARCore*, *Vuforia*, *Oculus / Meta SDK*, *Windows Mixed Reality*, *SteamVR SDK*, *Vive Wave SDK* та інші) аналізують ці дані, обробляють їх та генерують поведінку цифрових об'єктів відповідно до дій користувача. Завдяки цьому досягається імерсивний досвід, що забезпечує відчуття присутності та фізичної взаємодії з цифровими об'єктами.

XR-технології знаходять широке застосування у сферах освіти, медицини, промислового моделювання, архітектури, розваг та комп'ютерних ігор, де вони дозволяють створювати наочні, інтерактивні та інтуїтивно зрозумілі користувацькі середовища. Завдяки поєднанню апаратних і програмних компонентів *XR* забезпечує новий рівень взаємодії людини з цифровою інформацією, розширюючи можливості традиційних способів візуалізації та керування даними.

Навчальний посібник «XR-застосунки» надає покрокові інструкції для виконання чотирьох лабораторних практикумів та розроблення застосунків доповненої, змішаної та віртуальної реальностей в ігровому рушії *Unity*.

Навчальний посібник призначений для здобувачів першого (бакалаврського) рівня освіти, які навчаються за освітньою програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем» спеціальності 121 Інженерія програмного забезпечення факультету програмних систем та прикладної математики НТУУ «КПІ ім. Ігоря Сікорського».

ЛАБОРАТОРНИЙ ПРАКТИКУМ №1. ПРОЄКТУВАННЯ АНІМОВАНОЇ *3D*-СЦЕНИ В *BLENDER*

Лабораторний практикум присвячений моделюванню, анімуванню та рендерингу цифрового *3D*-об'єкта(ів) в середовищі розроблення *Blender* для використання в *XR*-застосунках.

В розділі «Порядок виконання» розглянутий процес створення та анімування *3D*-об'єкта в середовищі розроблення *Blender*.

ЗАВДАННЯ

Обрати тему для створення *3D*-сцени. Увага! Результати даного лабораторного практикуму використовуватимуться у наступних практикумах. За допомогою інструментів середовища розроблення *Blender* необхідно створити, анімувати та виконати рендеринг *3D*-сцени.

Для створення *3D*-сцени необхідно:

1. Створити *3D*-об'єкт(и) в середовищі розроблення *Blender*. *3D*-сцена обов'язково має містити не менше п'яти *3D*-об'єктів. Якщо *3D*-об'єкт містить багато складових елементів (п'ять та більше складових об'єктів), то такий «складний» об'єкт може бути на сцені один.

2. Додати текстури до *3D*-об'єкта(ів).

3. Анімувати *3D*-об'єкт(и). Тривалість анімації повинна бути не менше 20 секунд.

Звіт з лабораторного практикуму повинен містити:

1. Титульний аркуш, формулювання завдання, текстовий опис процесу створення, анімування та рендерингу *3D*-сцени, зокрема інформація щодо використаних інструментів, скриншоти проміжних та ключових результатів (*.pdf* файл).

2. Демонстрацію результату рендерингу анімованої *3D*-сцени (*.mp4* файл).

3. Файл проєкту в *Blender* (*.blend* файл).

1.1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Моделювання *3D*-сцени є процесом створення тривимірного цифрового представлення об’єктів і середовища, що включає формування їх геометрії, матеріалів, текстур, освітлення та просторових відносин між елементами сцени з метою подальшого анімування та рендерингу.

Анімування *3D*-сцени є процесом створення та керування послідовністю змін властивостей об’єктів і середовища в часі.

Рендеринг *3D*-сцени є процесом створення візуального представлення графічних даних у формі *2D*-зображення, відеопотоку або графічного інтерфейсу користувача на основі застосування обчислювальних алгоритмів проєктування *3D*-сцени на двовимірну поверхню (екран, лінзи тощо), оброблення освітлення, текстурування, видалення невидимих поверхонь, а також растеризації або трасування променів.

Реалізація процесів моделювання, анімування та рендерингу *3D*-сцен може здійснюватися за допомогою процедурного підходу, при якому всі етапи виконуються через алгоритмічні обчислення та програмні бібліотеки або ігрові рушії (напр., *OpenGL*, *DirectX*, *Vulkan*, *Unity*, *Unreal Engine* та інші), або із використанням спеціалізованого програмного забезпечення, що надає готові інструменти для створення геометрії, текстурування, анімації та рендерингу сцени (напр., *Blender*, *Maya*, *3ds Max*, *Cinema 4D* та інші).

Blender є безкоштовним та відкритим середовищем розроблення тривимірної графіки, яке підтримує повний цикл створення *3D*-сцени, від моделювання й текстурування до анімування та рендерингу з використанням сучасних алгоритмів. Робота в *Blender* поділена на так звані робочі простори, серед яких *Layout* призначений для розміщення та моделювання *3D*-об’єктів, *Modeling* – для моделювання полігональної сітки (*mesh*) *3D*-об’єктів, *Sculpting* – для моделювання природних форм та аватарів, *UV Editing* – для текстурування *3D*-об’єктів, *Texture Paint* – для створення текстур, *Shading* – для створення матеріалів та налаштування шейдерів, *Animation* – для створення анімацій, *Rendering* – для налаштування рендерингу, *Compositing* – для постоброблення

зображень та відео, *Scripting* – для написання програмного коду мовою програмування *Python*, що розширює функції *Blender*, а також *Geometry Nodes* – для процедурного моделювання *3D*-об’єктів.

При початковому запуску *Blender* за замовчуванням завантажується стандартна сцена, що включає базові об’єкти – куб, камеру та джерело освітлення (рис. 1.1). Робочий простір *Layout* є основним простором, призначеним для моделювання *3D*-об’єктів, в якому підтримуються базові дії з об’єктами:

1. Для виділення об’єкта у сцені необхідно навести курсор на об’єкт і натиснути ліву клавішу миші (ЛКМ).

2. Для зміни положення об’єкта використовується інструмент *Transform* (рис. 1.1) або необхідно натиснути клавішу *G* (*Grab*) та рухати об’єкт мишею. Щоб підтвердити дію переміщення, необхідно натиснути ЛКМ, щоб скасувати – праву клавішу миші (ПКМ) Рух можна обмежити по осях, натиснувши клавіші *G*+*X* (або *Y,* або *Z*). Рух можна обмежити по площині, натиснувши клавіші *G*+*Shift*+*X* (або *Y,* або *Z*).

3. Для зміни орієнтації об’єкта використовується інструмент *Transform* або необхідно натиснути клавішу *R* (*Rotate*) та рухати об’єкт мишею. Щоб підтвердити дію обертання, необхідно натиснути ЛКМ, щоб скасувати – ПКМ. Рух можна обмежити по осях, натиснувши клавіші *R*+*X* (або *Y,* або *Z*). Рух можна обмежити по площині, натиснувши клавіші *R*+*Shift*+*X* (або *Y,* або *Z*).

4. Для масштабування об’єкта використовується інструмент *Transform* або необхідно натиснути клавішу *S* (*Scale*) та рухати об’єкт мишею. Щоб підтвердити дію масштабування, необхідно натиснути ЛКМ, щоб скасувати – ПКМ. Рух можна обмежити по осях, натиснувши клавіші *S*+*X* (або *Y,* або *Z*). Рух можна обмежити по площині, натиснувши клавіші *S*+*Shift*+*X* (або *Y,* або *Z*).

5. Для видалення об’єкта необхідно виділити об’єкт та натиснути *X* та *Delete*.

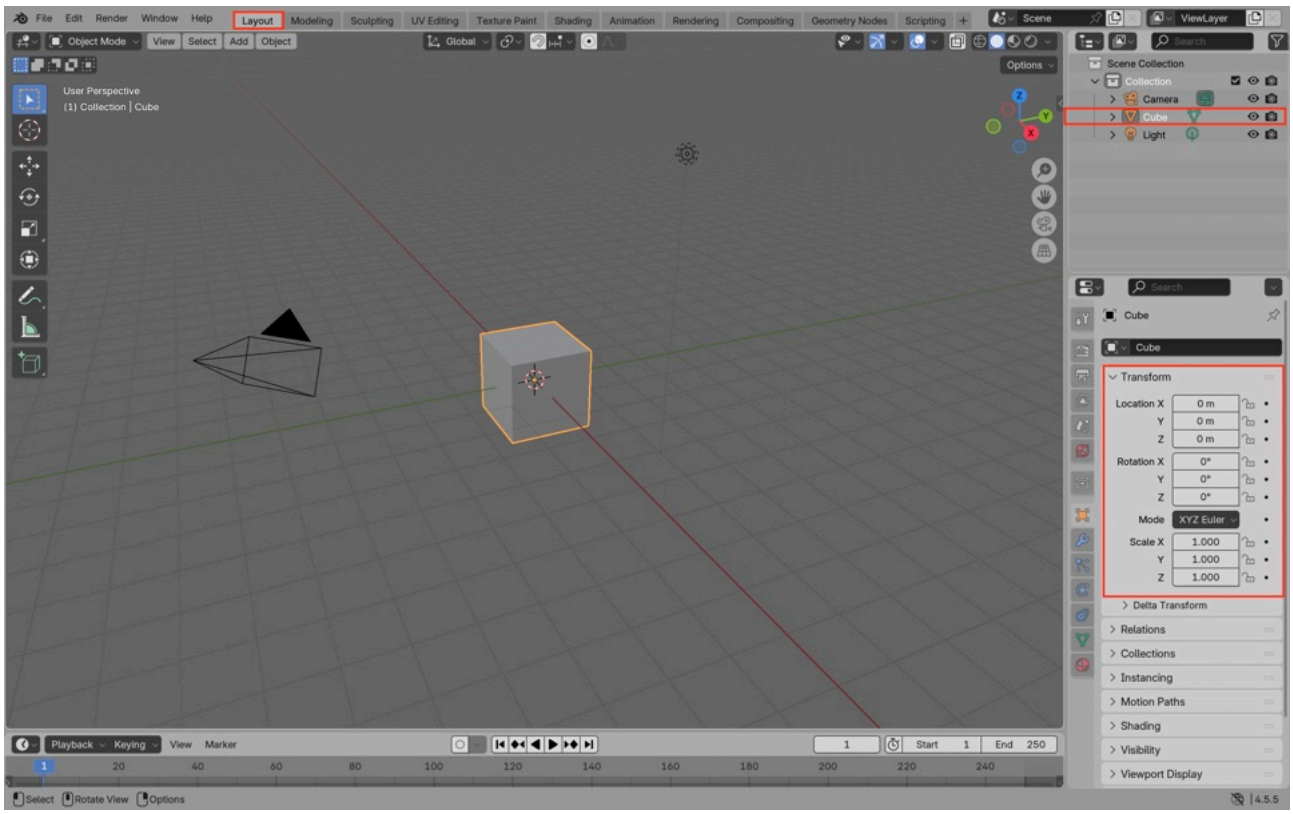


Рис. 1.1 – Робочий простір *Layout*

3D-об’єкти в *Blender* є полігональними моделями, геометрія яких формується з множини полігонів, об’єднаних у полігональну сітку. Така сітка складається з вершин, ребер і граней. Для перетворення полігонів *3D*-об’єкта необхідно ввійти в режим редагування простору *Layout* (рис. 1.2) або натиснути клавішу *Tab*. В режимі редагування можна перетворювати вершини полігонів, ребра та грані. Для цього необхідно обрати елемент (напр., вершину) або декілька елементів (натиснути клавішу *Shift*) та переміщувати, обертати, масштабувати елементи полігонів.

6. Інструмент редагування *Extrude* (рис. 1.2) призначений для витіснення обраних елементів полігонів відповідно до заданого напрямку (клавіша *E* в режимі редагування простору *Layout*).

7. Інструмент редагування *Loop Cut* (рис. 1.2) призначений для замкненого розрізання уздовж граней *3D*-об’єкта, що дозволяє деталізувати полігональну сітку, керувати її топологією (клавіші *Command+R*).

8. Інструмент *Knife* (клавіша *K*) призначений для довільного розрізання полігональної сітки *3D*-об’єкта (рис. 1.2). Для завершення розрізання необхідно натиснути клавішу *Enter*.

9. Інструмент *Poly Build* (рис. 1.2) призначений для інтерактивного створення та редагування полігональної сітки шляхом додавання нових вершин і граней, а також зміни положення існуючих елементів.

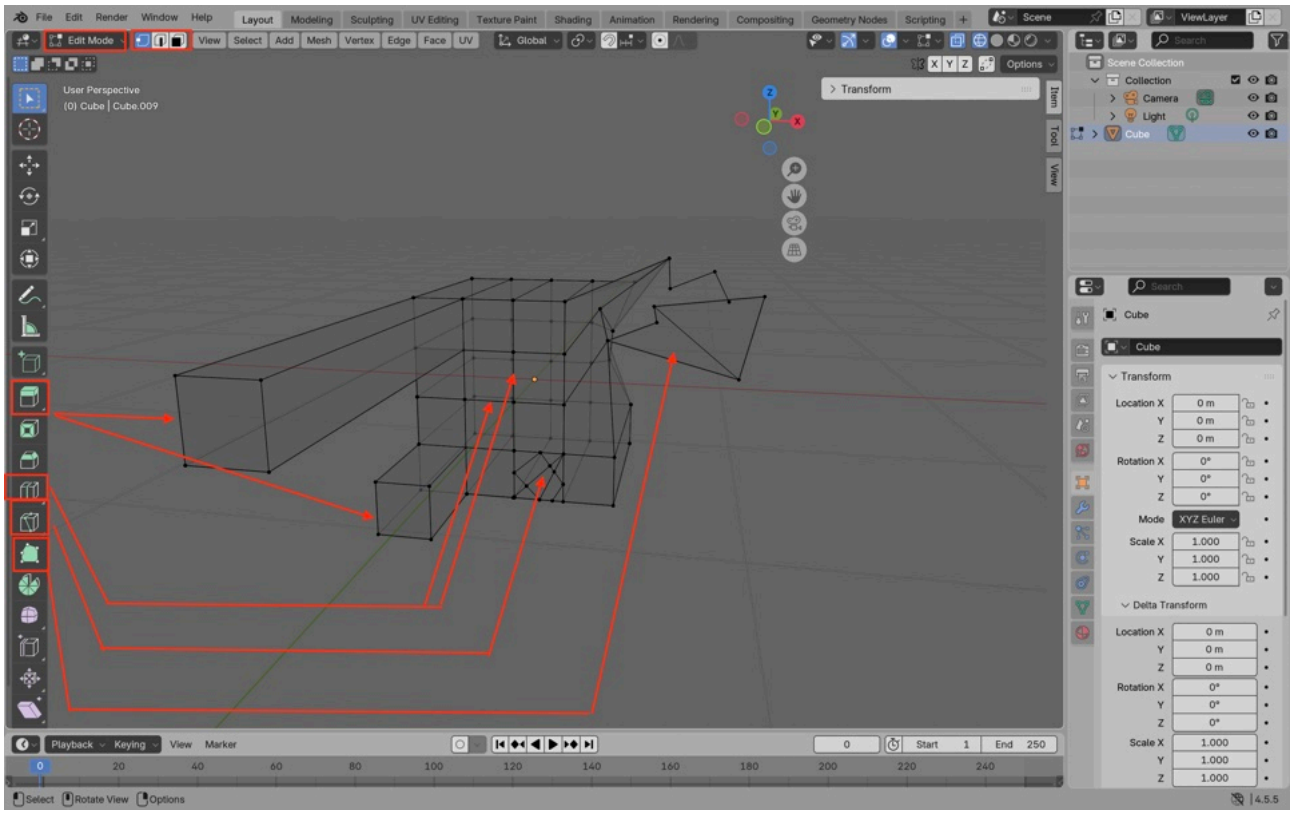


Рис. 1.2 – Режим редагування робочого простору *Layout*

10. Операція *Subdivide* використовується для рівномірного поділу вершин, ребер або граней та деталізації полігональної сітки. В режимі редагування необхідно виділити елемент полігональної сітки, натиснути ПКМ та в контекстному меню обрати *Subdivide* (рис. 1.3). Параметрами операції *Subdivide* є кількість розрізів (*Number of Cuts*), які дозволяють керувати ступенем деталізації сітки, ступінь згладжування та інші (рис. 1.4).

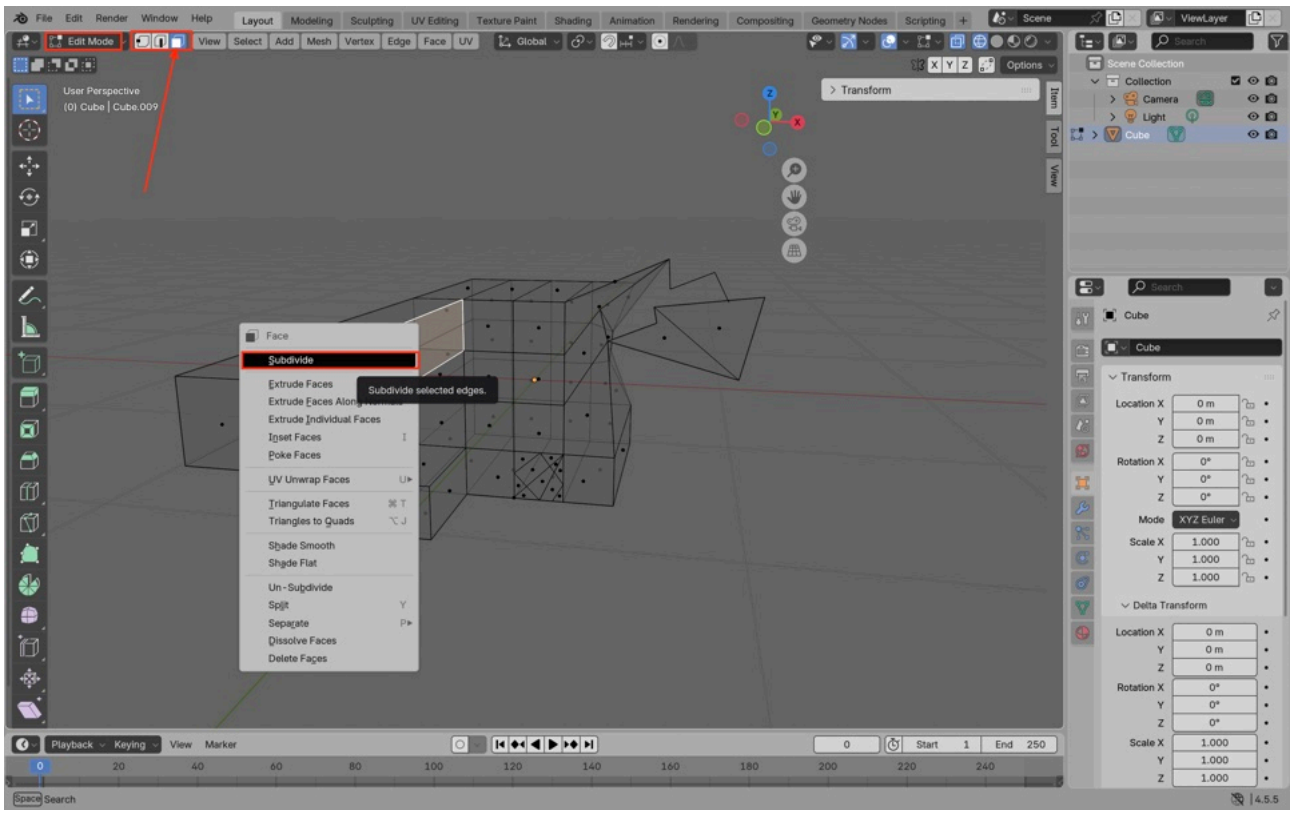


Рис. 1.3 – Операція *Subdivide*

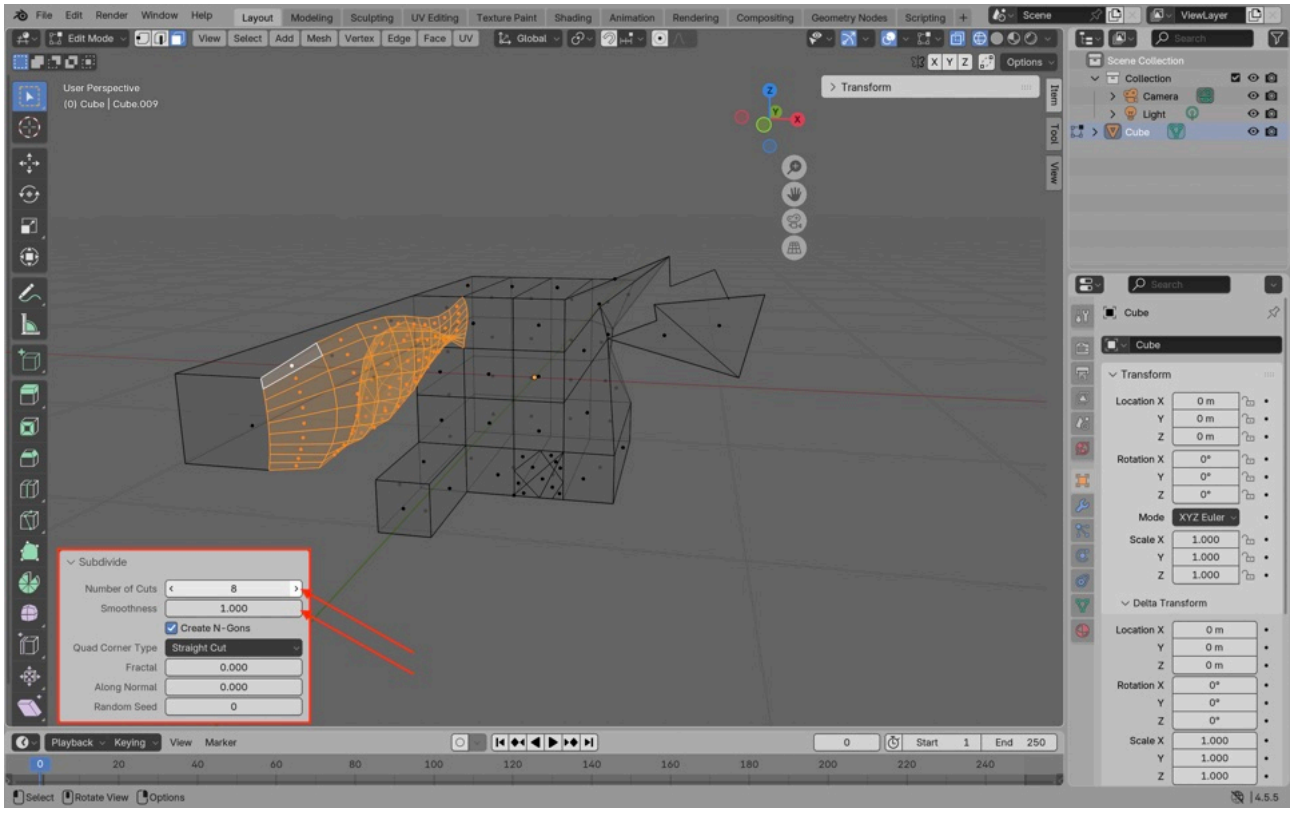


Рис. 1.4 – Параметри операції *Subdivide*

11. Для згладжування геометричних форм *3D*-об’єкта використовується інструмент *Smooth Vertices* (рис. 1.5), який визначає нові координати вибраних вершин шляхом їх усереднення відносно сусідніх вершин полігональної сітки. У режимі редагування необхідно виділити вершини, натиснути ПКМ-> *Smooth Vertices*. Параметрами інструмента (рис. 1.6) є ступінь згладжування (*Factor*) та кількість повторів застосування операції (*Repeat*).

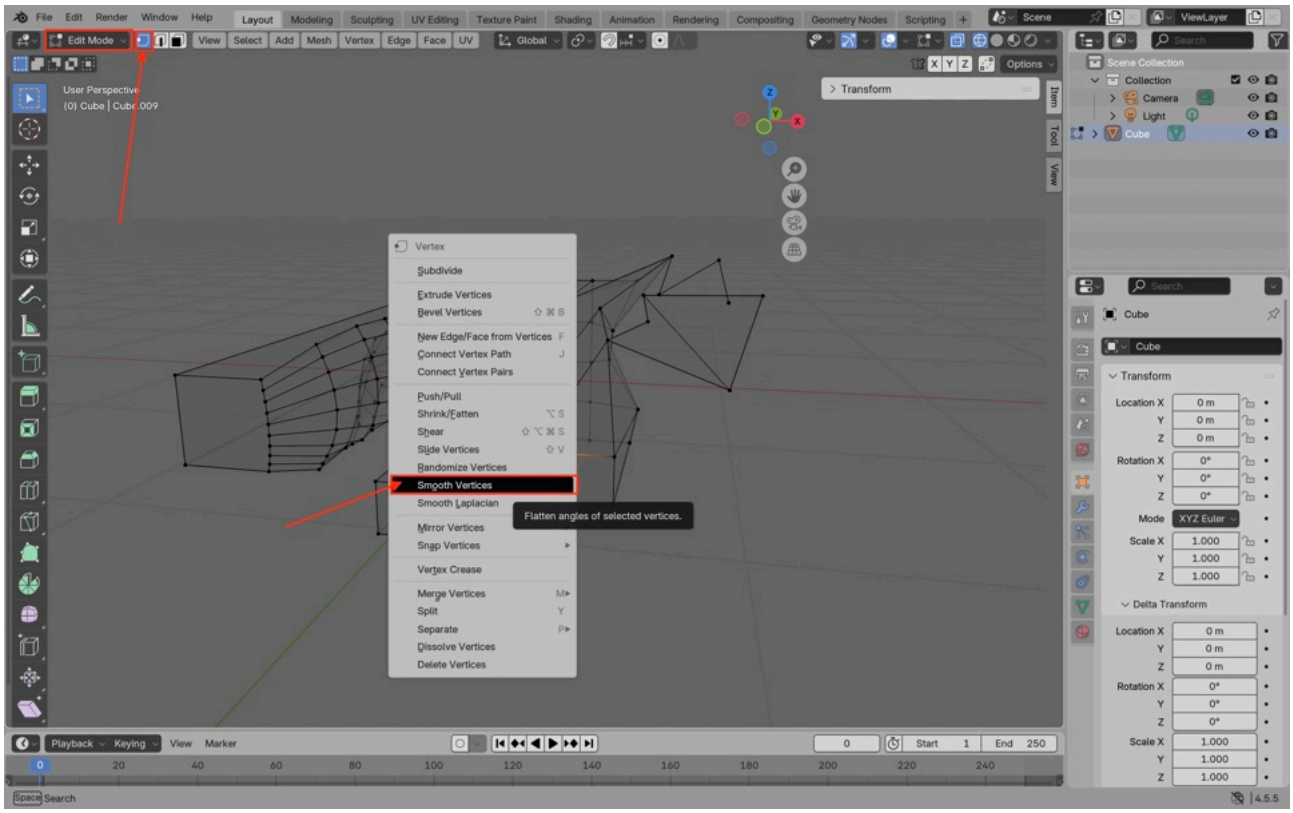


Рис. 1.5 – Інструмент *Smooth Vertices*

12. Для об’єднання вершин полігональної сітки в одну вершину використовується інструмент *Merge* (рис. 1.7). У режимі редагування необхідно виділити вершини, натиснути ПКМ- > *Merge Vertices* (або клавіша *M*) та обрати спосіб об’єднання (у центрі, за курсором, за першою або останньою вершиною, за відстанню між вершинами).

13. Додавання нових *3D*-об’єктів в *Blender*-сцену здійснюється через меню *Add* (рис. 1.8) або клавішами *Shift* + *A*.

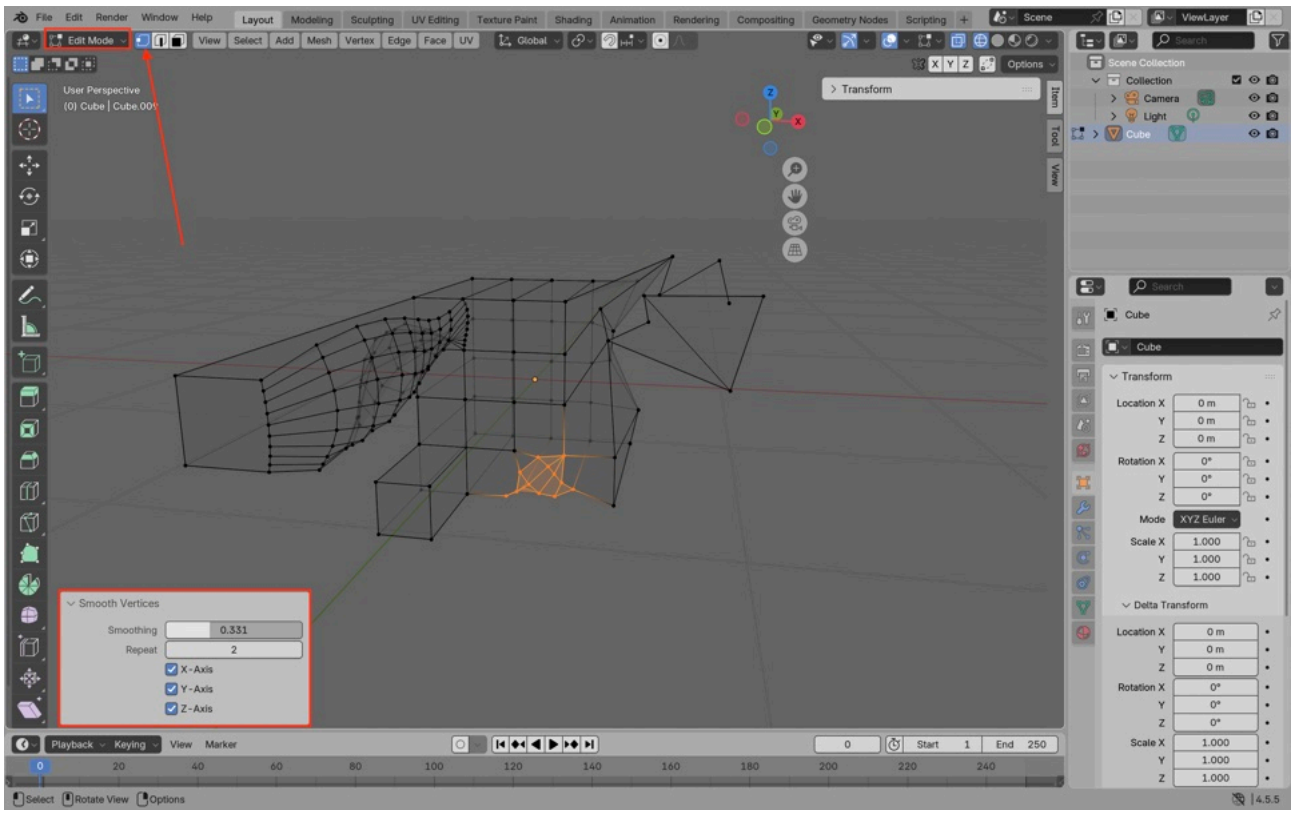


Рис. 1.6 – Параметри інструмента *Smooth Vertices*

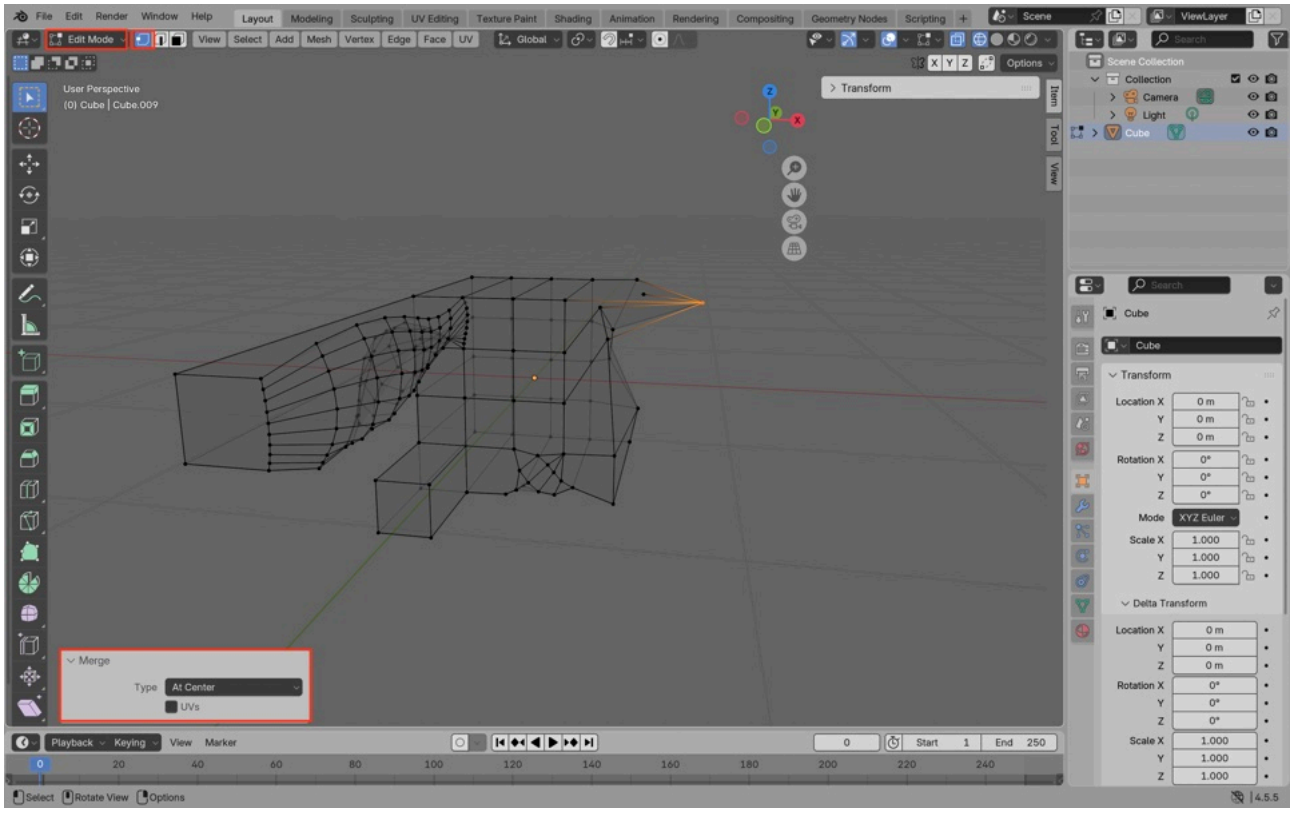


Рис. 1.7 – Інструмент *Merge*

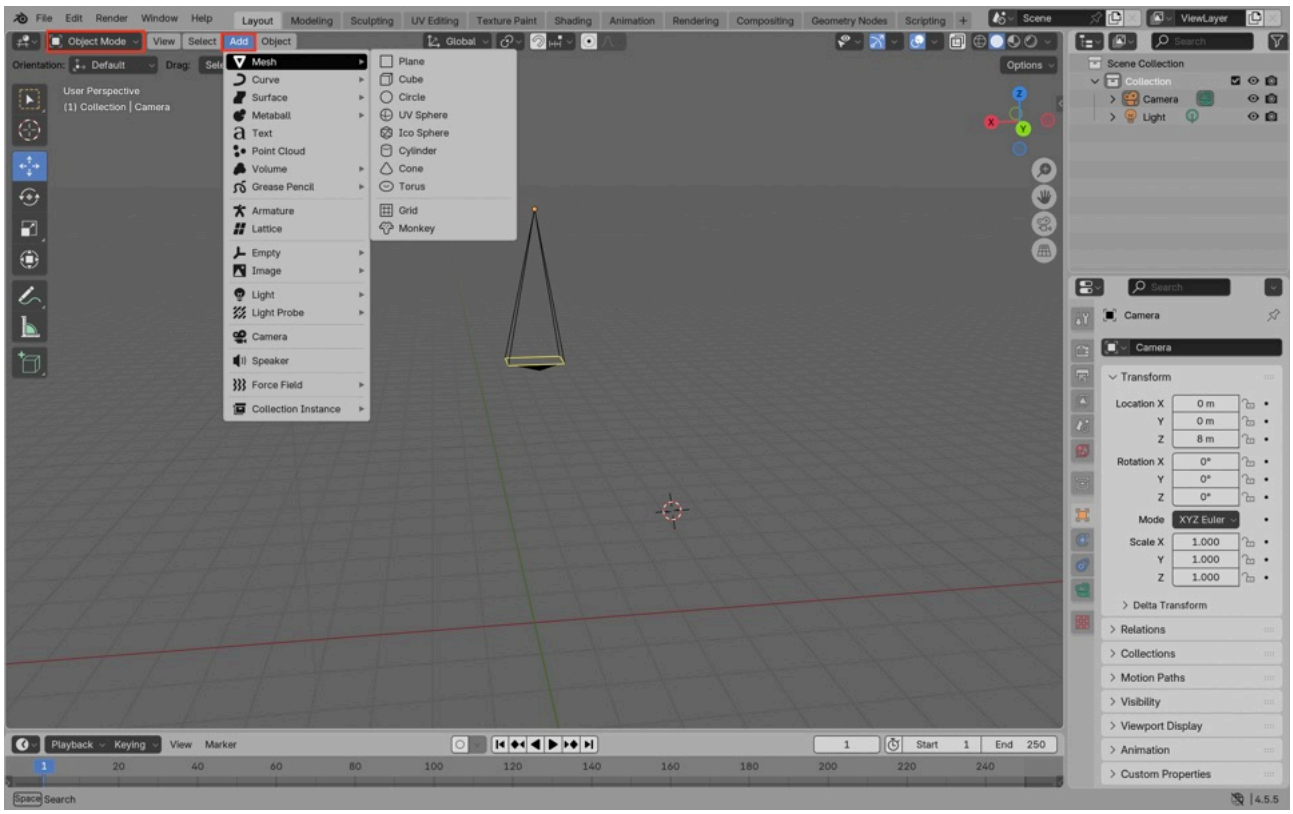


Рис. 1.8 – Додавання *3D*-об’єктів в *Blender*-сцену

В середовищі *Blender* *3D*-об’єкти, камери та джерела світла організовуються у так звані колекції. Кожна колекція може містити будь-яку кількість об’єктів, камер, світла та допоміжних елементів, а також інші колекції, створюючи ієрархічну структуру. Колекції використовуються для організації об’єктів сцени, керування їх видимістю та рендерингом.

В середовищі *Blender* *3D*-об’єкти можна об’єднати в один за допомогою операції *Join*. В режимі об’єктів (*Object Mode*) необхідно виділити об’єкти для об’єднання та обрати *Object– > Join* (рис. 1.9) або натиснути клавіші *Ctrl+J*. Після об’єднання утворюється єдиний *3D*-об’єкт із сумарною геометрією, що зберігає топологію та властивості матеріалів, але втрачається незалежність окремих складових *3D*-об’єкта.

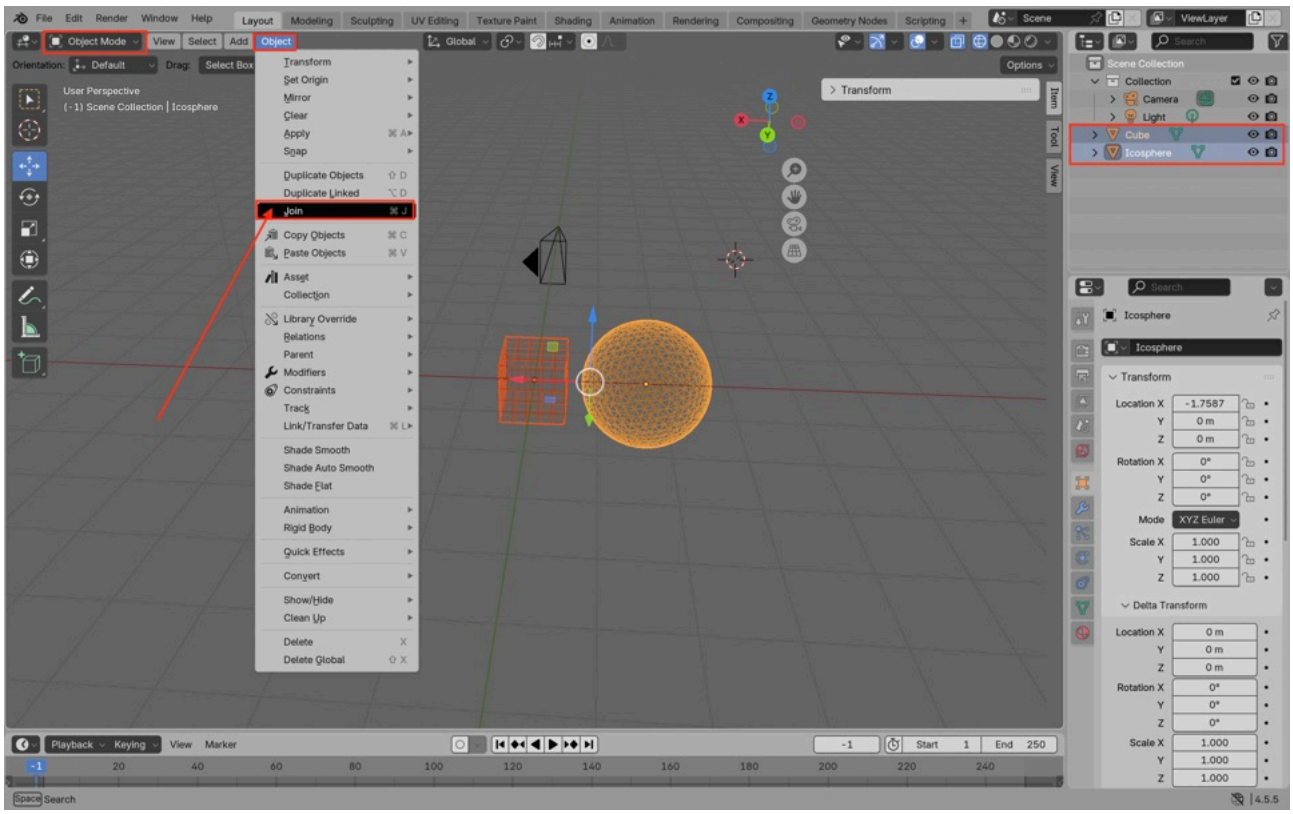


Рис. 1.9 – Операція об’єднання об’єктів (*Join Objects*)

14. Для зміни точки спостереження сцени в *Blender* використовується кругове меню навігації *Viewpoint Pie Menu* (рис. 1.10), яке викликається натисканням клавіші ~ (тильда). У цьому меню можна обрати необхідний напрямок перегляду сцени. За умови попереднього виділення об’єкта та вибору пункту *View Selected* точка спостереження у *3D Viewport* вирівнюється відносно вибраного об’єкта. У разі виділення об’єкта типу «світло» перегляд сцени здійснюється з позиції джерела освітлення. За умови вибору пункту *View Camera* перегляд сцени здійснюється з позиції активної камери (рис. 1.11).

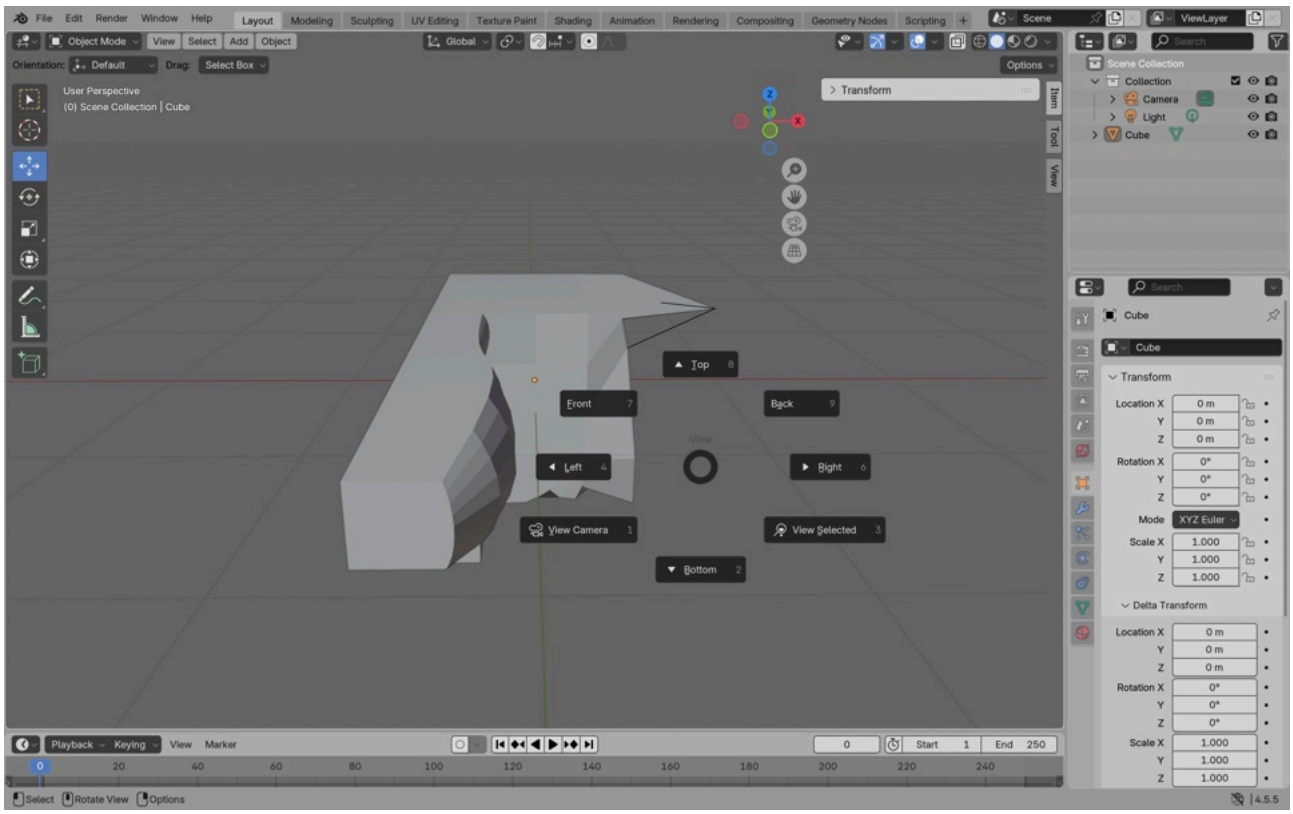


Рис. 1.10 – Кругове меню навігації для зміни точки спостереження

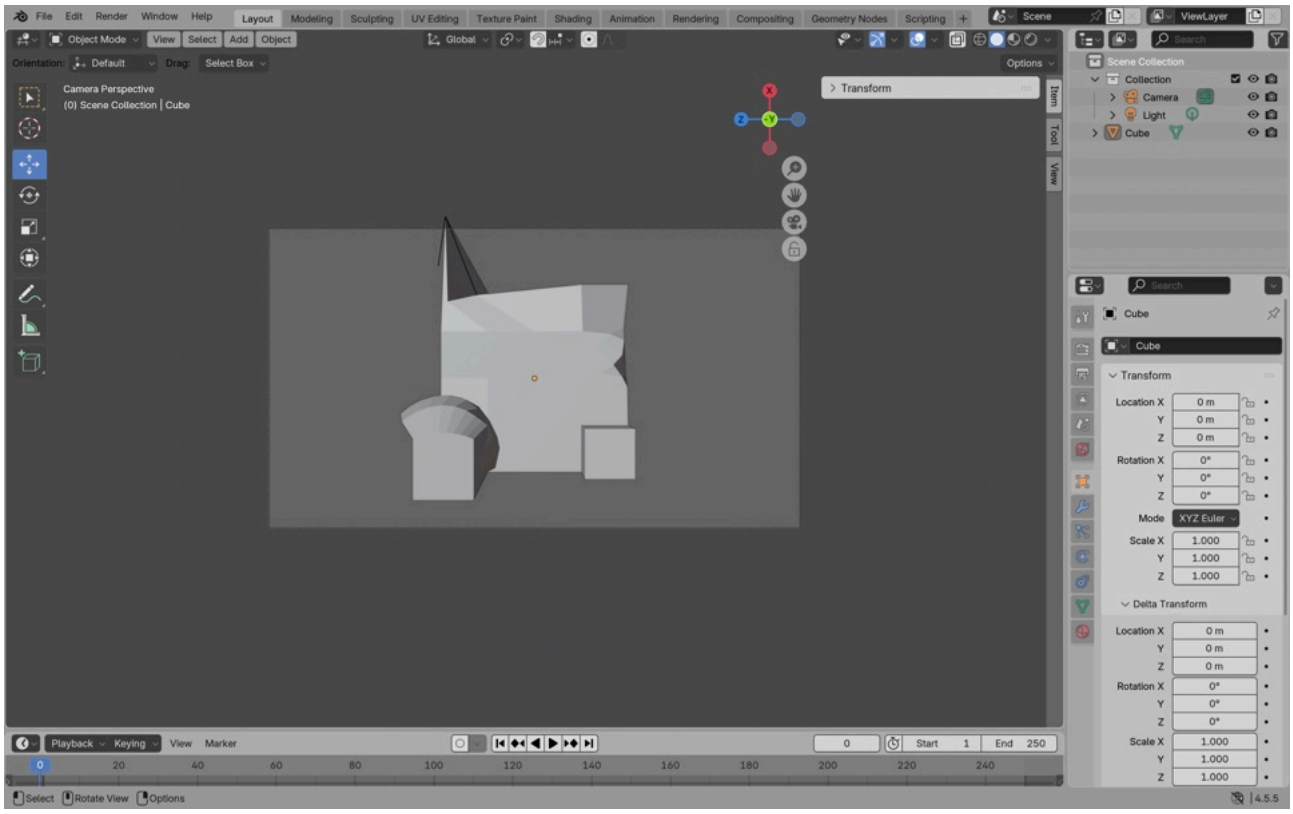
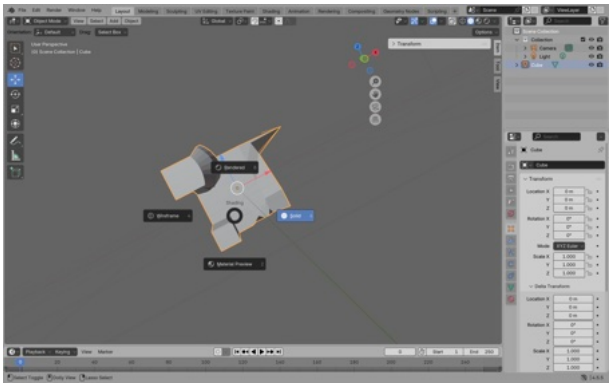
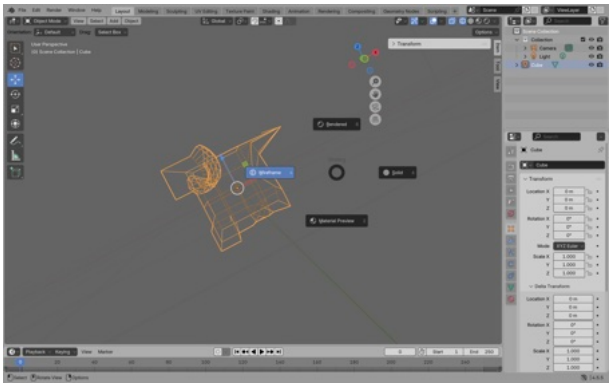


Рис. 1.11 – Спостереження *Blender*-сцени з позиції камери

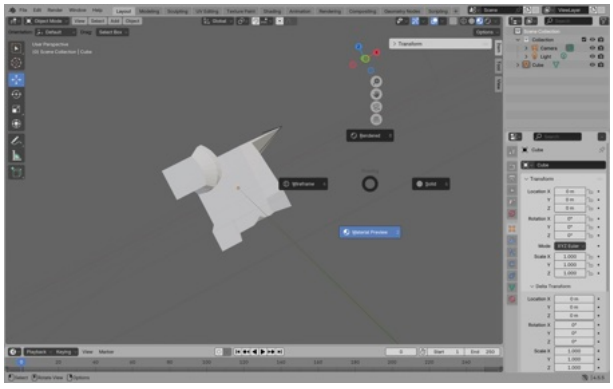
15. Для зміни режиму відображення сцени в *Blender* використовується кругове меню візуалізації *Shading Pie Menu* (рис. 1.12), яке викликається натисканням клавіші *Z*. Пункт *Solid* відображає *3D*-об’єкт без урахування матеріалів – суцільний режим. Пункт *Wireframe* відображає *3D*-об’єкт у вигляді ребер полігональної сітки – каркасний режим. Пункт *Material Preview* відображає *3D*-об’єкт з урахуванням матеріалів, текстур та базових параметрів освітлення – режим матеріалу. Пункт *Rendered* відображає *3D*-об’єкт з урахуванням матеріалів, текстур, параметрів освітлення та налаштувань рендерингу – режим рендерингу.



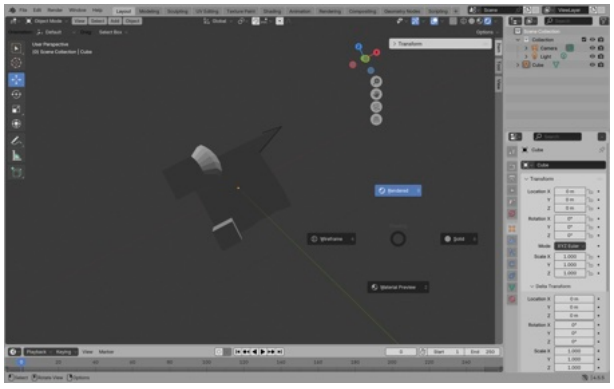
а)



б)



в)



г)

Рис. 1.12 – Кругове меню візуалізації: а) режим *Solid*, б) режим *Wireframe*, в) режим *Material Preview*, г) режим *Rendered*

Для керування налаштуваннями *3D*-об’єктів, сцени, рендерингу та інших складових *Blender*-проєкту використовується редактор параметрів (*Properties Editor*). Основними вкладками редактора є наступні:

– параметри рендерингу (*Render Properties*);

– параметри керування результатом рендерингу у вигляді зображень або відео (*Output Properties*);

– параметри керування шарами відображення сцени (*View Layer Properties*);

– параметри сцени (*Scene Properties*);

– параметри глобального оточення сцени (*World Properties*);

– просторові параметри об’єктів (*Object Properties*);

– параметри застосування модифікаторів до об’єктів (*Modifier Properties*);

– параметри керування системою частинок (*Particle Properties*);

– параметри фізичних симуляцій (*Physics Properties*);

– параметри керування обмеженнями поведінки об’єктів (*Constraints Properties*);

– параметри матеріалів об’єктів (*Material Properties*);

– параметри текстур об’єктів (*Texture Properties*).

1.2 ПОРЯДОК ВИКОНАННЯ

Оскільки процес моделювання, налаштування матеріалів та текстур, освітлення, а також анімування та рендеринг *3D*-об’єктів є індивідуальним і творчим, у цьому розділі наведено загальний підхід та ключові етапи проєктування анімованої *3D*-сцени в середовищі *Blender* без детального відтворення всіх можливих варіантів виконання. Основну увагу зосереджено на базових інструментах моделювання, анімування, налаштування матеріалів та рендерингу.

1.2.1 Створення *3D*-аватара засобами полігонального та скульптурного моделювання

3D-аватар є цифровим тривимірним графічним представленням персонажа (людини або вигаданої істоти), яке створене за допомогою засобів *3D*-графіки та відтворює зовнішній вигляд, міміку й рухи об’єкта. *3D*-аватар використовується для візуалізації користувача або персонажа у цифрових середовищах, іграх, анімації, *XR*-застосунках та інтерактивних системах.

Для створення геометрії *3D*-аватара застосовуються засоби полігонального моделювання на основі еталонних зображень. Полігональне моделювання *3D*-об’єктів у середовищі *Blender* полягає у поетапному формуванні просторової геометрії *3D*-моделі на основі двовимірних візуальних орієнтирів (зображень), що застосовуються як проєкційні шаблони для забезпечення коректних пропорцій, анатомічної достовірності та структурної узгодженості полігональної сітки, яка надалі використовується для текстурування, анімування та рендерингу тривимірної сцени.

Для завантаження зображення необхідно натиснути на меню *Add* (або клавіші *Shift* + *A*), обрати *Image-> Reference* та у вікні файлового менеджера обрати зображення, та підтвердити його завантаження. Завантажене зображення буде додано до сцени як неполігональний допоміжний об’єкт (рис. 1.13.), за потреби параметри відображення еталонного зображення (прозорість, положення, масштаб тощо) можна змінити у редакторі параметрів (*Object Data Properties*).

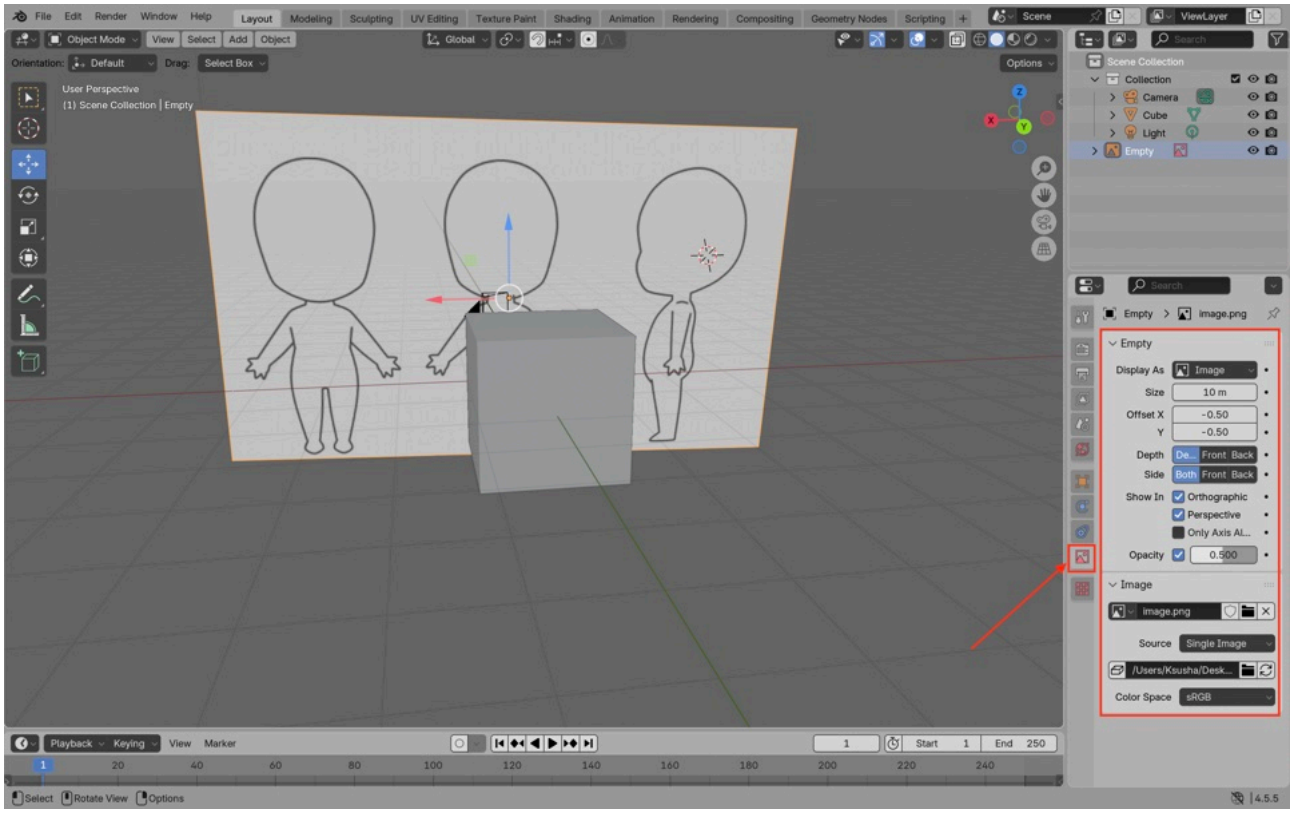


Рис. 1.13 – Додавання еталонного зображення для моделювання

Для полігонального моделювання просторової геометрії *3D*-об’єкта застосовується деталізація полігональної сітки за допомогою операції *Subdivide* (рис. 1.14).

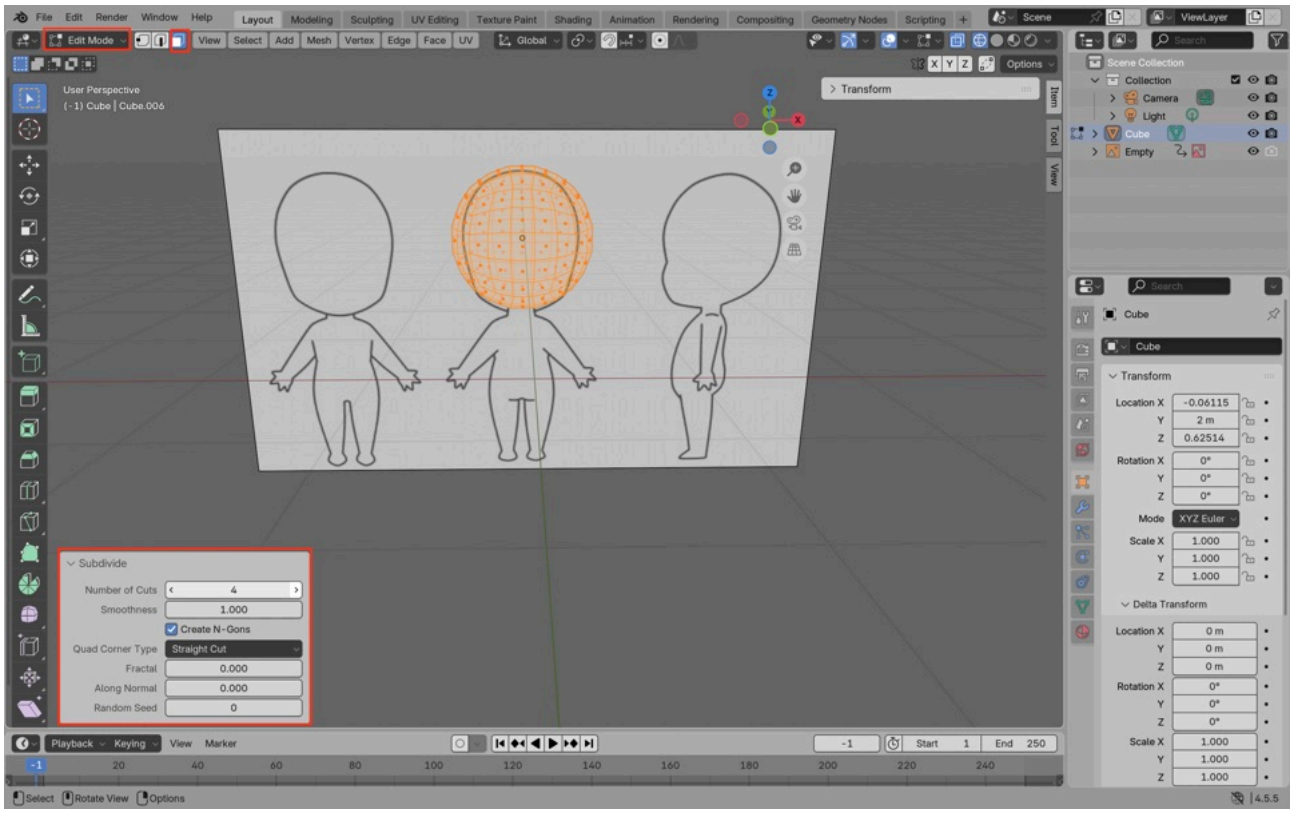


Рис. 1.14 – Результат операції *Subdivide*

В режимі редагування за допомогою інструментів полігонального моделювання *Poly Build*, *Extrude* (клавіша *E*), *Knife* (клавіша *K*), *Loop Cut* (клавіші *Command+R*), переміщення (клавіша *G*), обертання (клавіша *R*) та масштабування (клавіша *S*) поетапно створюються полігональні *3D*-об’єкти (рис. 1.15). Для згладжування геометричних форм *3D*-об’єктів використовується інструмент *Smooth Vertices*, для об’єднання вершин полігональної сітки – інструмент *Merge*. Для створення симетричної геометричної форми у редакторі параметрів використовується модифікатор *Mirror*. *3D*-об’єкти було об’єднано в єдиний за допомогою операції *Join Objects*.

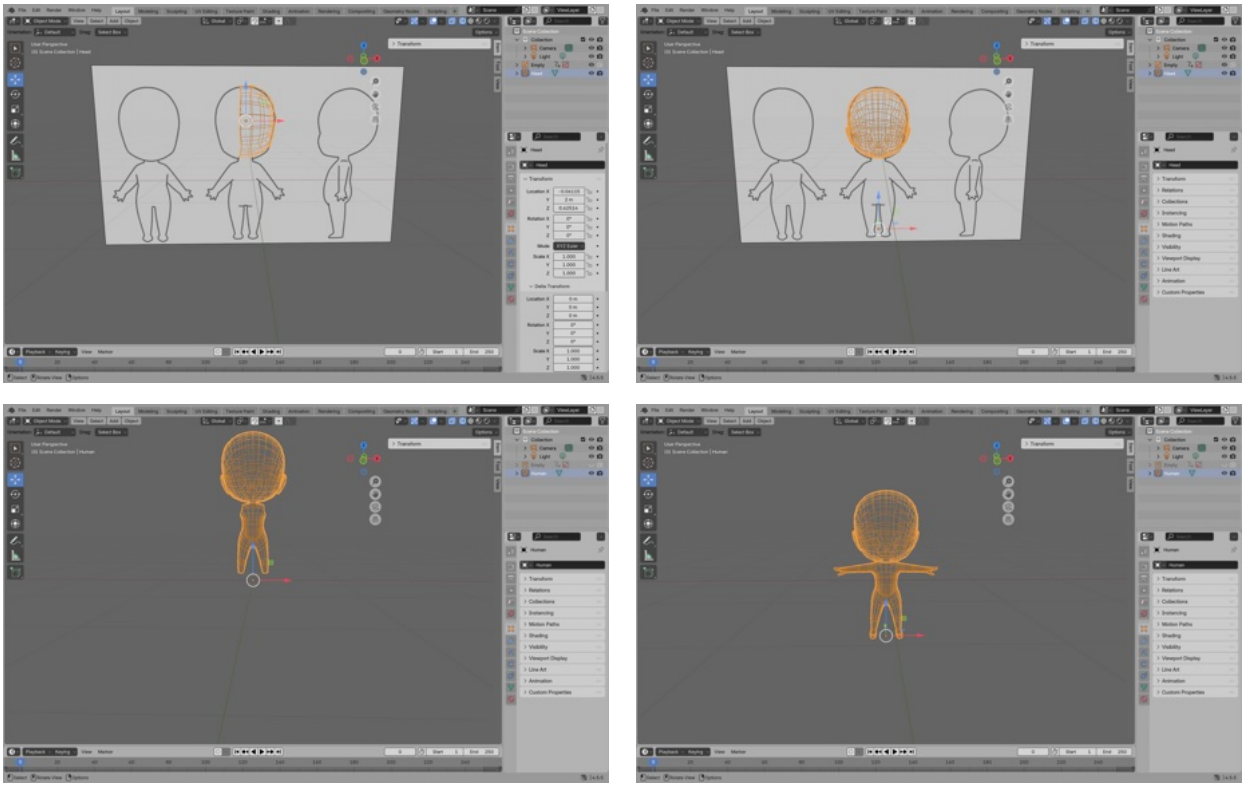


Рис. 1.15 – Полігональне моделювання *3D*-аватара

Для персоналізації цифрового *3D*-аватара доцільно також завантажити еталонне зображення, як візуальний орієнтир під час моделювання деталей об’єкта.

Для моделювання очей *3D*-аватара використовуються примітиви сфери. При створенні примітивів (напр., сфери, циліндра) задаються початкові параметри, зокрема кількість сегментів, що визначають деталізацію та кількість граней полігональної сітки. Якщо початкові налаштування не були змінені, кількість граней можна збільшити у процесі моделювання за допомогою інструментів *Loop Cut*, *Knife*, операції *Subdivide* або модифікатора *Subdivision Surface*. Для згладжування форми примітива необхідно в режимі об’єкта обрати примітив та натиснути *Object- > Shade Smooth* або натиснути ПКМ- > *Shade Smooth* (рис. 1.16).

Для моделювання брів *3D*-аватара використовуються примітиви типу циліндр. За допомогою інструментів *Extrude*, *Knife*, *Poly Build*, переміщення, обертання та масштабування поетапно створюються полігональні *3D*-об’єкти (рис. 1.17).

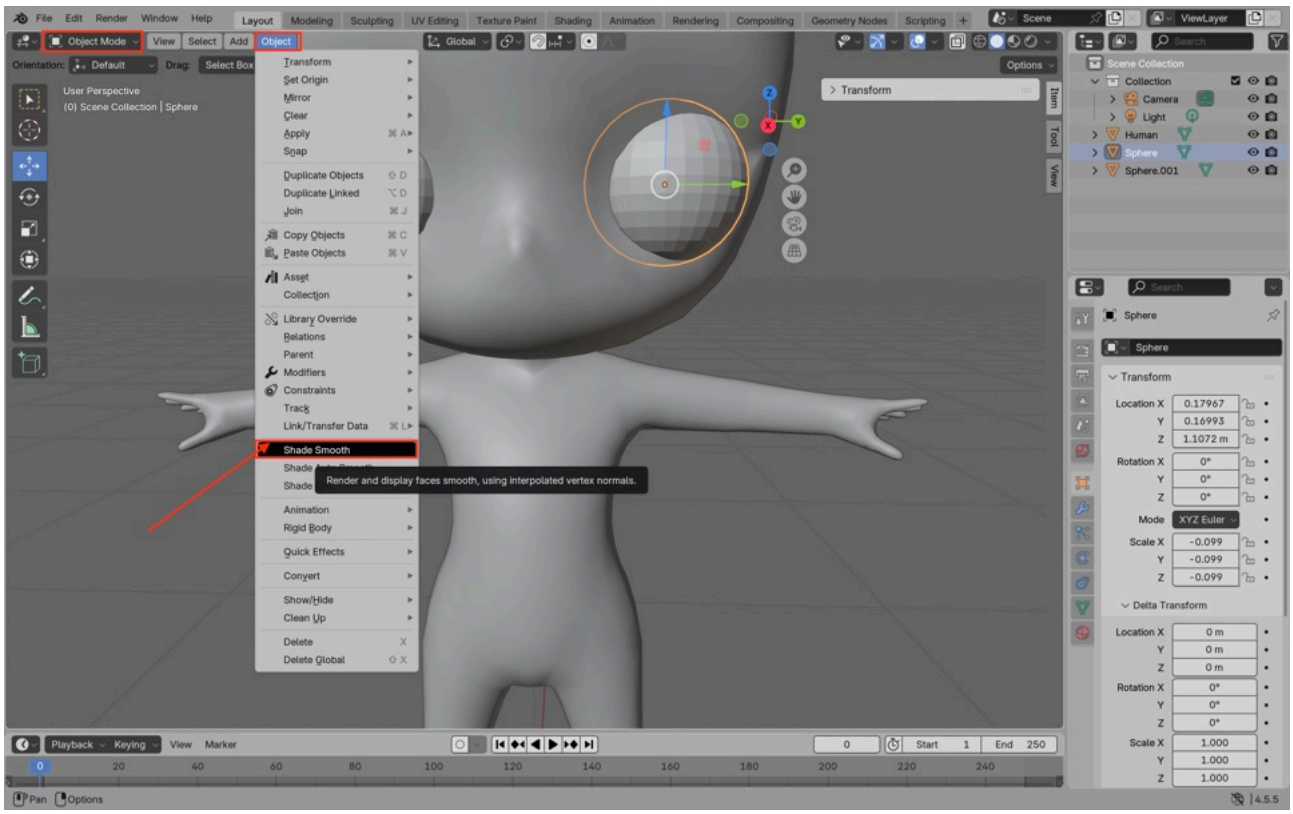


Рис. 1.16 – Метод затінення *Shade Smooth*

Рис. 1.17 – Полігональне моделювання брів *3D*-аватара

Для моделювання вус *3D*-аватара використовуються примітиви типу циліндр. За допомогою інструментів *Extrude*, *Poly Build*, переміщення, обертання та масштабування поетапно створюються полігональні *3D*-об’єкти (рис. 1.18).

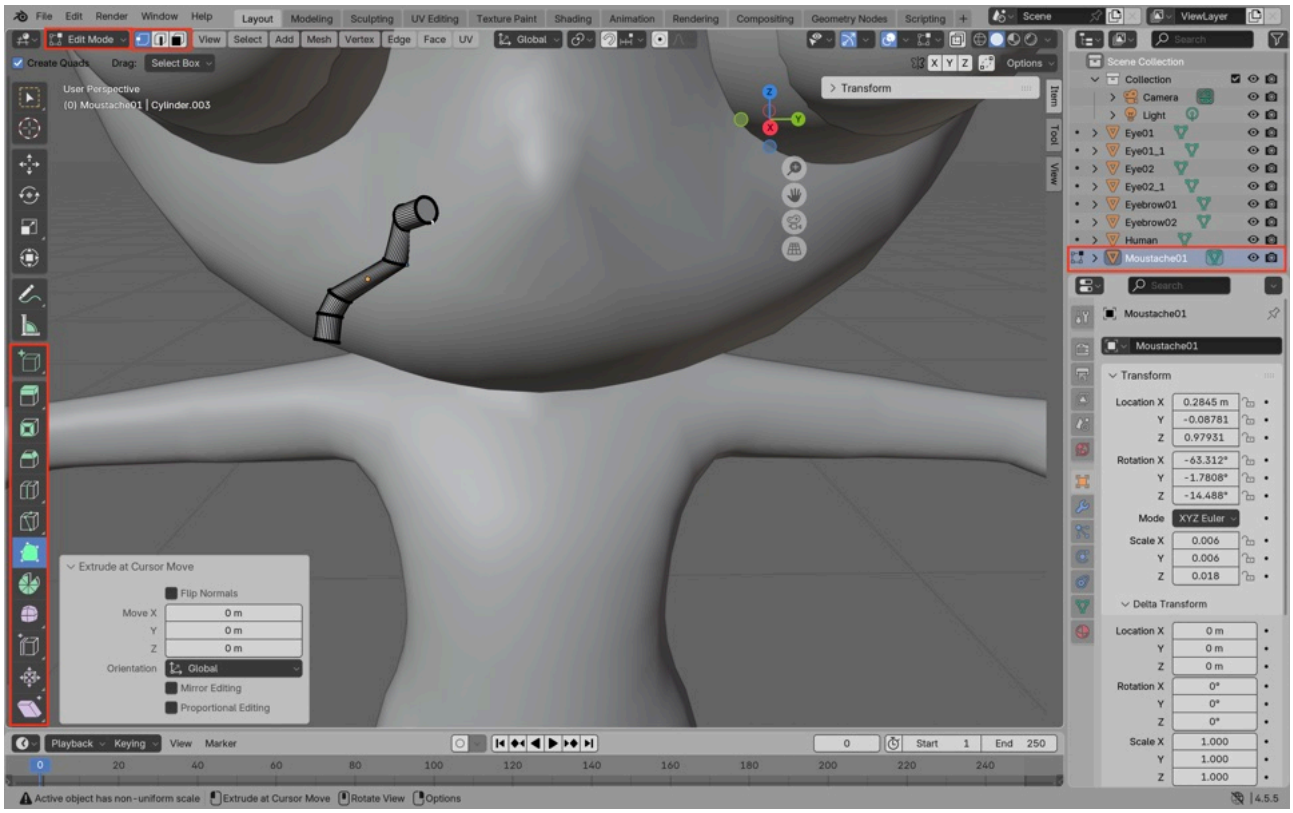


Рис. 1.18 – Полігональне моделювання вус *3D*-аватара

Для згладжування геометричної форми *3D*-об’єкта застосовується модифікатор *Subdivision Surface*. В редакторі параметрів необхідно обрати вкладку модифікаторів та натиснути *Add Modifier- > Generate- > Subdivision Surface* (рис. 1.19).

Основними параметрами модифікатора є рівень розбиття полігональної сітки (рис. 1.20). Модифікатор *Subdivision Surface* додає нові вершини, що підвищує деталізацію та гладкість *3D*-об’єкта, а також ускладнює його топологію.

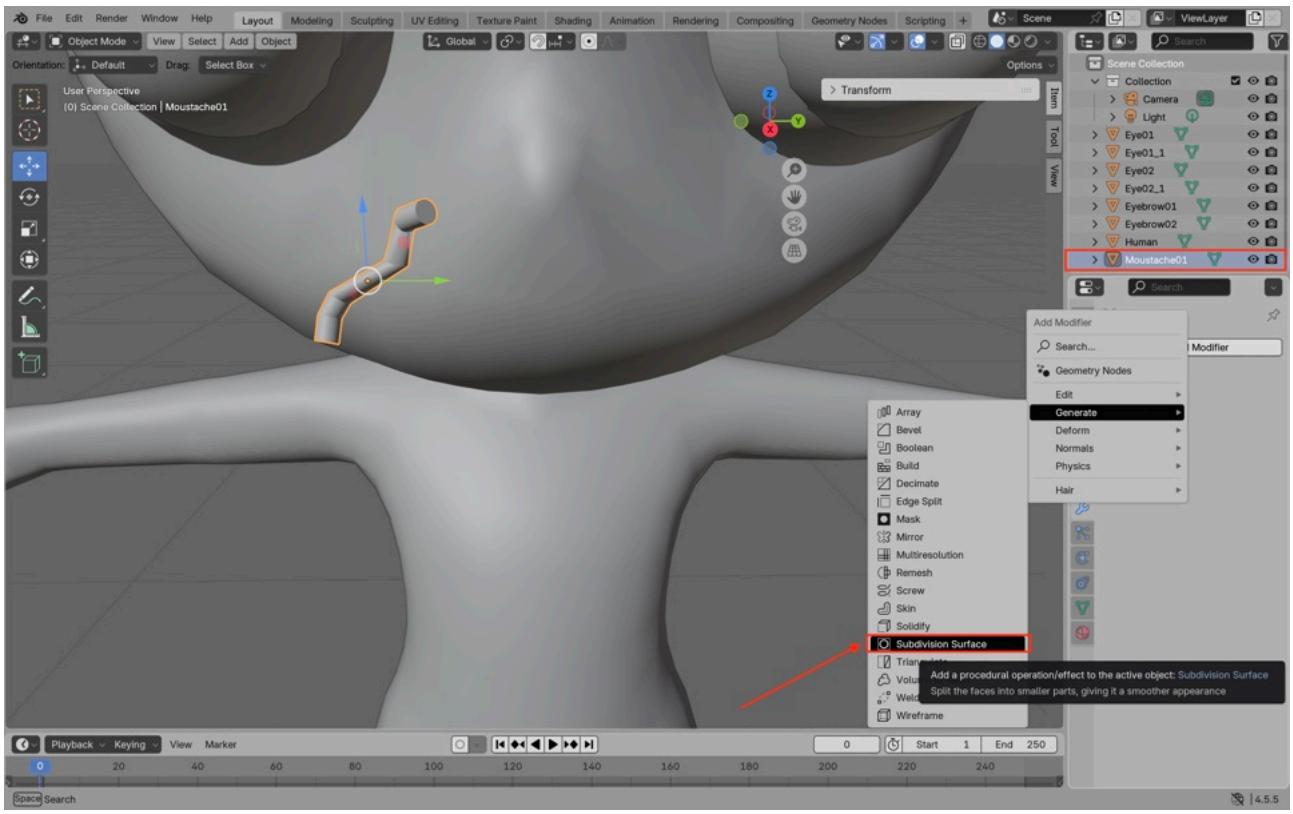


Рис. 1.19 – Модифікатор *Subdivision Surface*

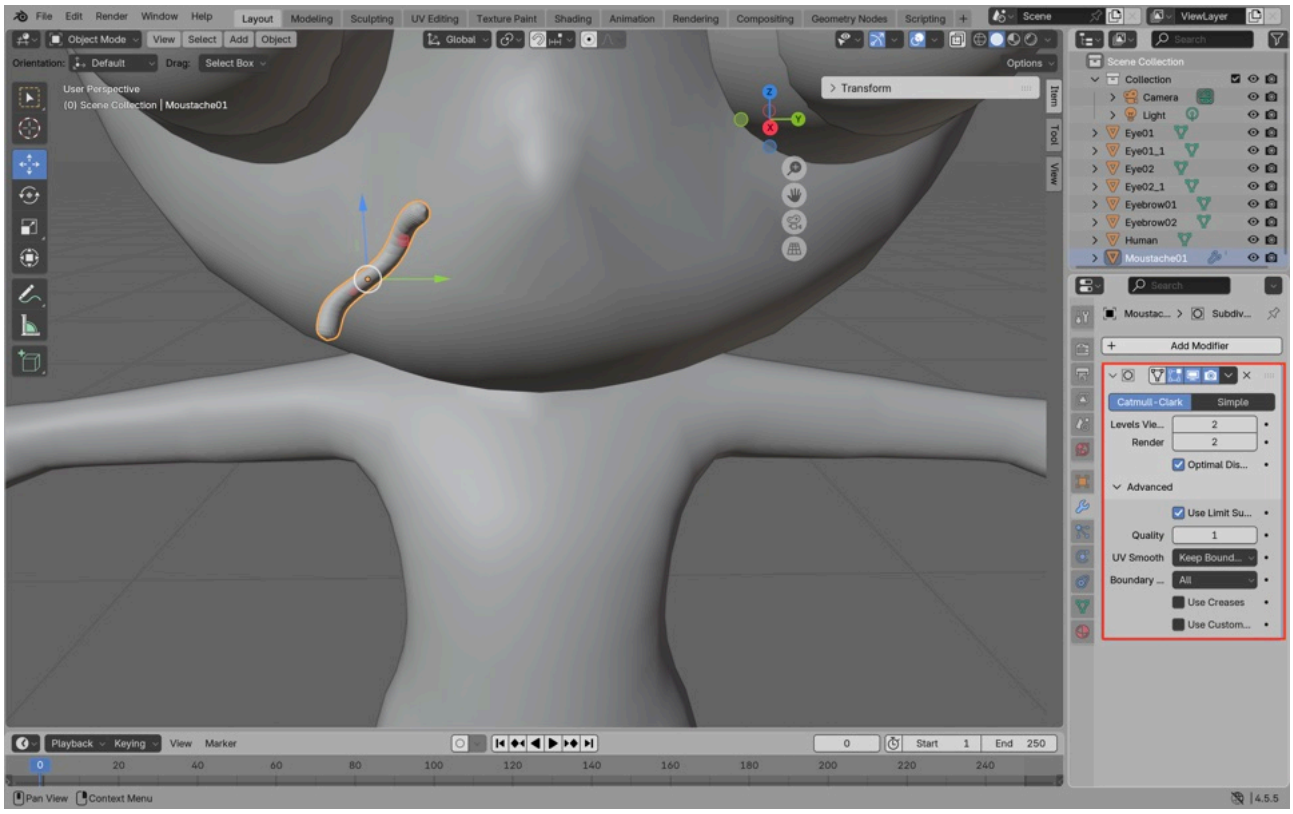


Рис. 1.20 – Параметри модифікатора *Subdivision Surface*

Для зміни форми полігональної сітки без прямого редагування вершин застосовуються інструменти скульптурування. У режимі *Sculpt Mode* робочого простору *Layout* (рис. 1.21) або у спеціалізованому робочому просторі *Sculpting* доступні інструменти інтерактивного моделювання різноманітних природних форм.

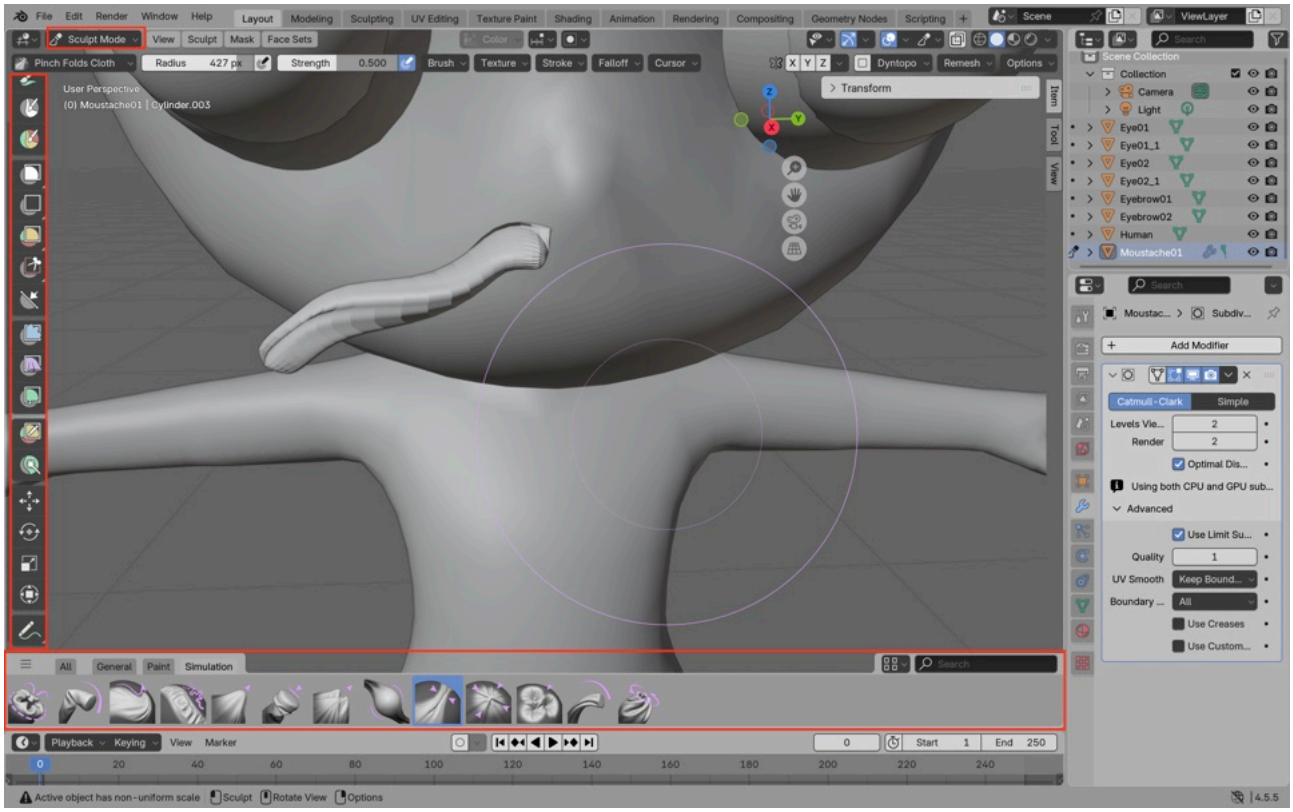


Рис. 1.21 – Інструменти скульптурування

Моделювання інших полігональних *3D*-об’єктів (напр., борода та волосся) було виконано за описаним вище підходом (рис. 1.22). По-перше, застосовувались інструменти полігонального моделювання, зокрема *Extrude* для витягування полігонів і створення об’ємних форм, *Loop Cut* для додавання додаткових ребер і контролю топології, *Knife* для точного розрізання сітки, *Merge* для об’єднання вершин та *Poly Build* для додавання нових та редагування наявних елементів сітки. Для зміни положення та форми *3D*-об’єктів застосовувались інструменти переміщення, обертання та масштабування, які дозволяють модифікувати вершини, ребра, грані або весь *3D*-об’єкт. По-друге, для підвищення рівня деталізації та плавності поверхні застосовувався

модифікатор *Subdivision Surface*, який автоматично додає нові вершини та ребра, розбиваючи полігони на дрібніші площини. По-третє, для досягнення природних, органічних форм і тонких деталей поверхні використовувались інструменти скульптурування (*Sculpt Mode*), такі як *Draw*, *Clay*, *Inflate*, *Smooth* та *Grab*, що дозволяють змінювати форму об’єкта інтерактивно без прямого редагування вершин та забезпечують більш реалістичний вигляд *3D*-об’єктів.

Рис. 1.22 – Результат моделювання полігональних об’єктів для *3D*-аватара

Для створення моделей одягу *3D*-аватара можна або моделювати нові графічні примітиви або до існуючих *3D*-об’єктів застосувати модифікатор *Solidify* в редакторі параметрів. Для реалізації другого підходу спочатку необхідно обрати всі грані, які буде покривати модель одягу (рис. 1.23). Для вибору декількох граней одночасно у *Blender* використовується клавіша *Shift* – утримуючи її, можна поєднувати виділення нових граней із вже вибраними.

Рис. 1.23 – Виділення граней *3D*-об’єкта для дублювання

Наступним необхідно дублювати виділені грані. В режимі редагування обрати *Mesh- > Duplicate* (або клавіші *Shift+D*). Для відокремлення виділених граней в окремий самостійний *3D*-об’єкт останнім необхідно натиснути *ПКМ-> Separate- > Selection* (рис. 1.24).

Для надання товщини плоскому *3D*-об’єкту застосовується модифікатор *Solidify*. В редакторі параметрів необхідно обрати панель модифікаторів та натиснути *Add Modifier- > Generate- > Solidify* (рис. 1.25). Останнім необхідно визначити величину товщину, напрямок додавання товщини, рівномірність та інші параметри (рис. 1.26).

Моделювання інших *3D*-об’єктів (напр., штани та взуття) було виконано за описаним вище підходом (рис. 1.27). *3D*-об’єкти було об’єднано в єдиний за допомогою операції *Join Objects*.

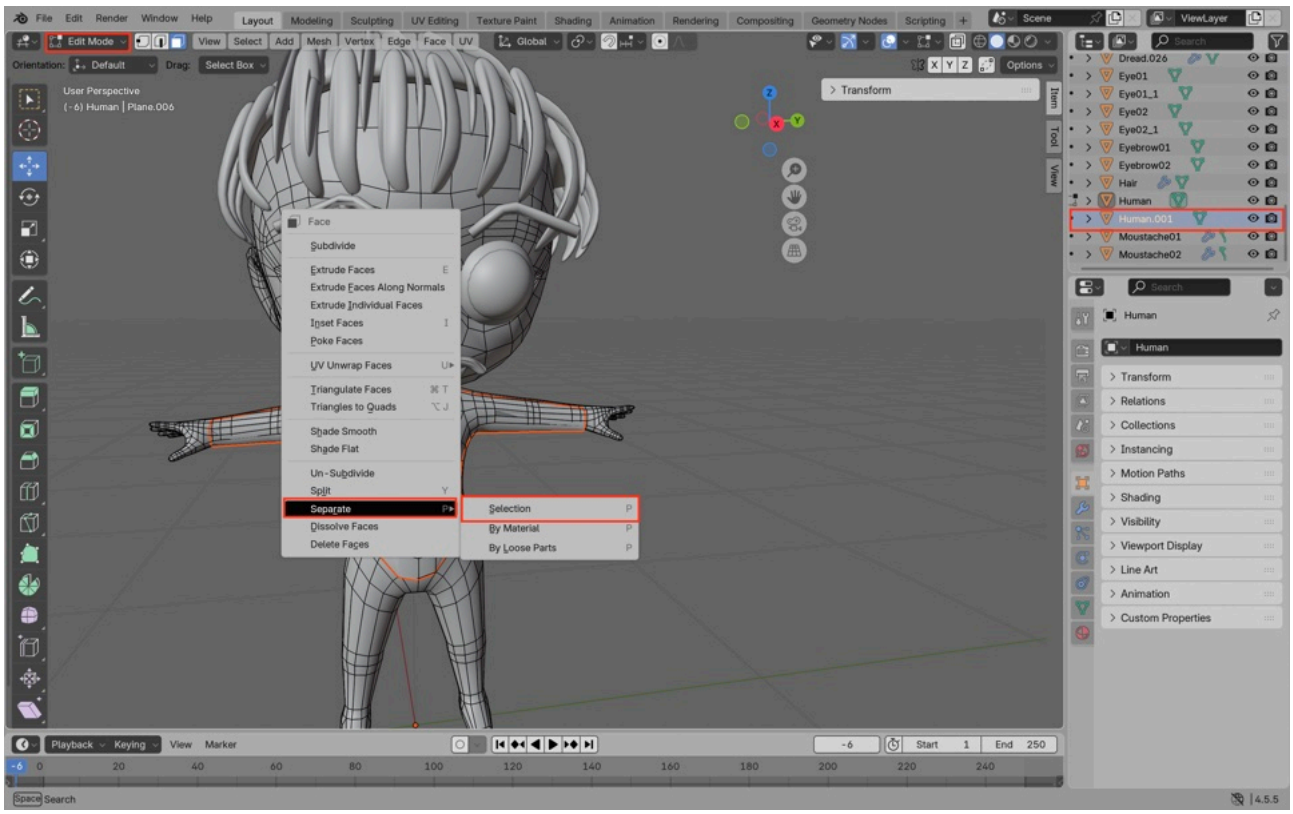


Рис. 1.24 – Відокремлення граней в самостійний *3D*-об’єкт

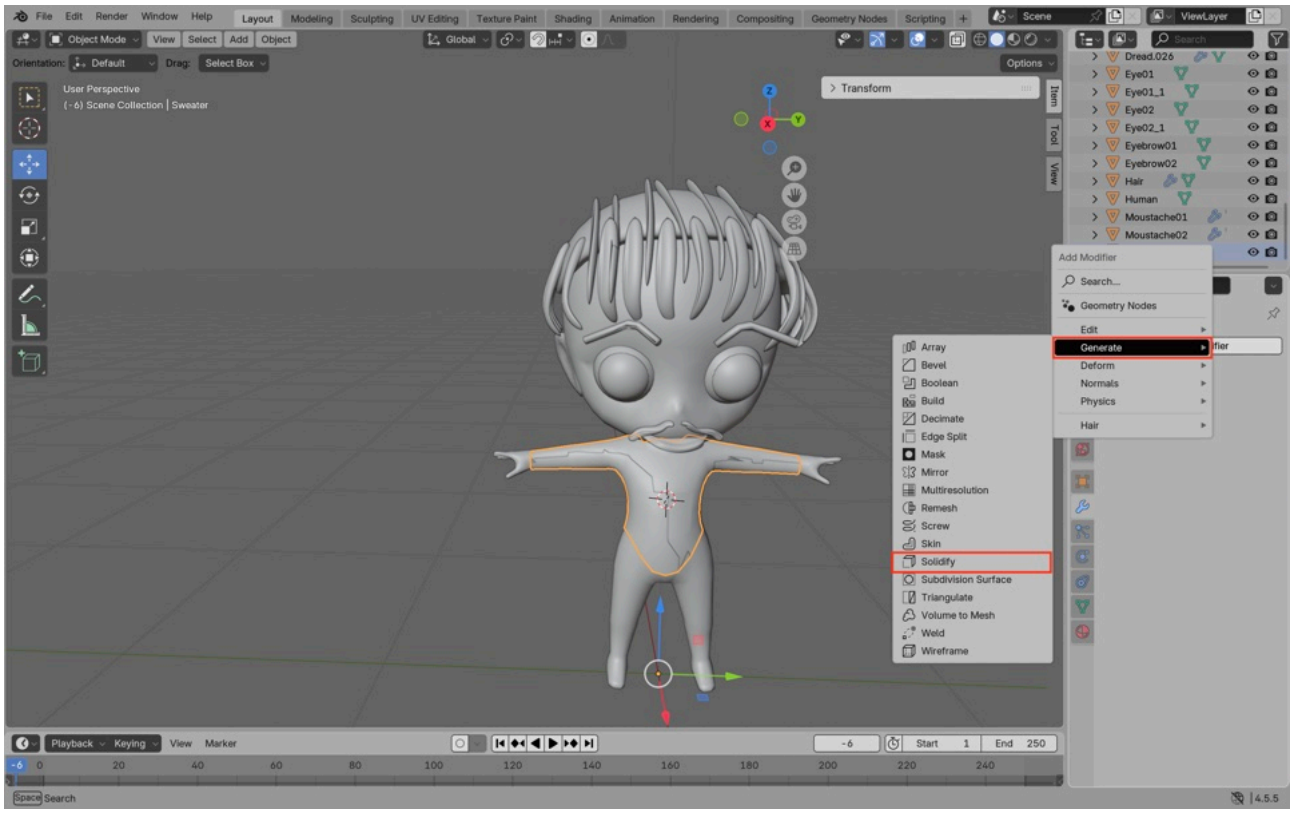


Рис. 1.25 – Додавання модифікатора *Solidify*

Рис. 1.26 – Налаштування модифікатора *Solidify*

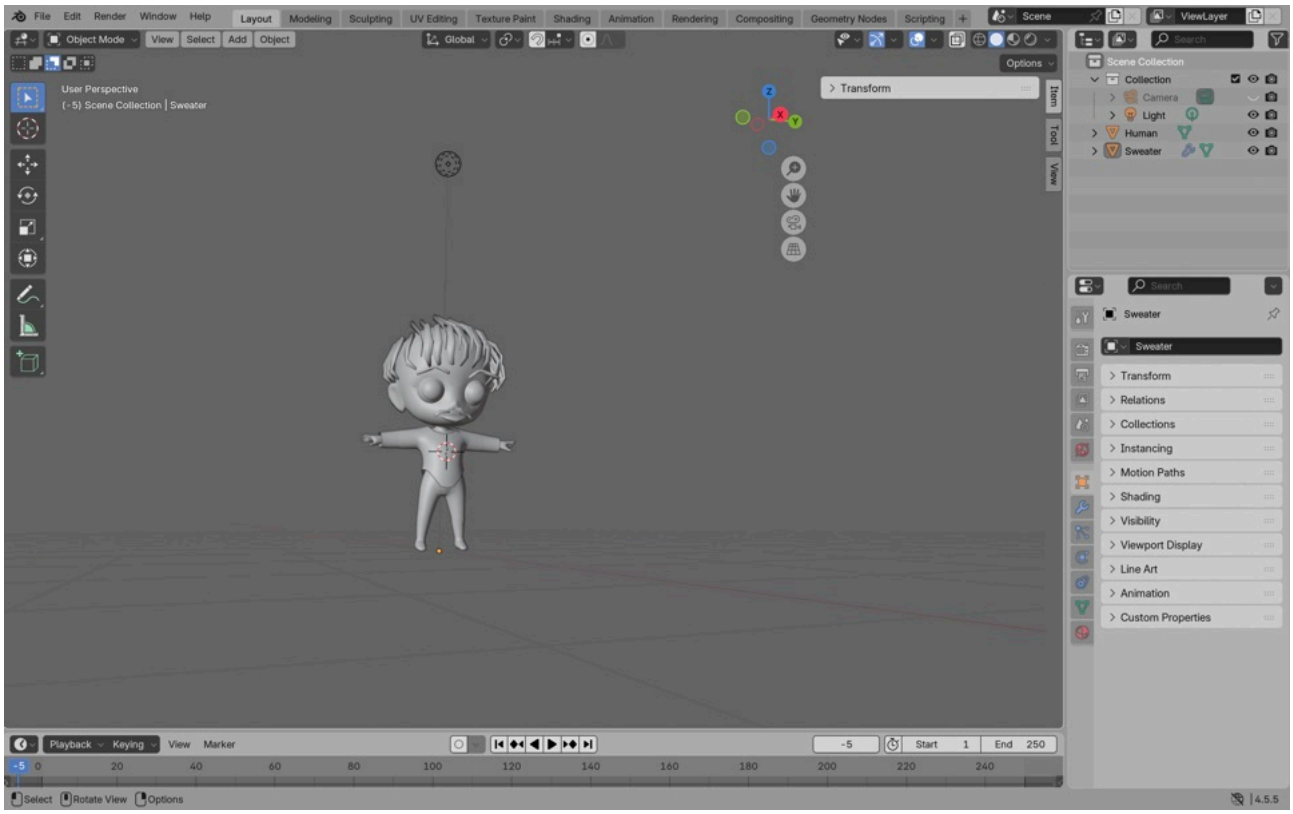


Рис. 1.27 – Результат моделювання *3D*-аватара

1.2.2 Налаштування матеріалів *3D*-аватара

Для додавання *3D*-об’єктам візуальних характеристик використовується система матеріалів *Blender*. У редакторі параметрів необхідно обрати панель параметри матеріалів (*Material Properties*) та за допомогою кнопки *New* створити (рис. 1.28) та налаштувати параметри матеріалу (рис. 1.29), зокрема колір, прозорість, відбивна здатність та текстури. Для забезпечення коректних візуальних характеристик *3D*-об’єкта при створенні матеріалу можна використовувати еталонне зображення та за допомогою інструмента *Eyedropper* («Піпетка») задавати необхідні відтінки безпосередньо із зображення.

Для створення матеріалу зі складними властивостями (наприклад, градієнтом кольору) використовується робочий простір *Shading* та редактор вузлів (*Shader Editor*). Робочий простір *Shading* також містить редактор параметрів та вікно перегляду. Основним універсальним шейдером є *Principled BSDF*, який дозволяє керувати кольором, блиском, металевістю, прозорістю та іншими фізичними властивостями матеріалу (рис. 1.30).

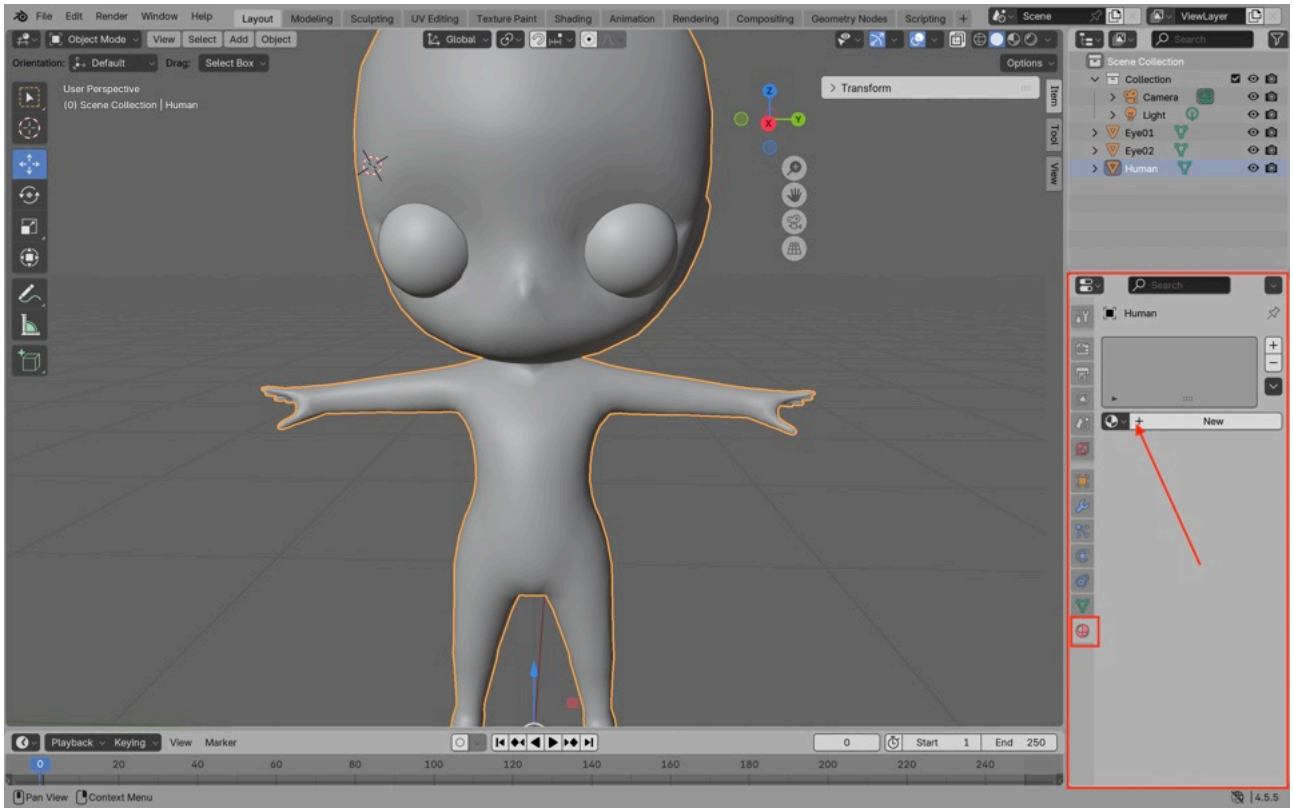


Рис. 1.28 – Створення нового матеріалу для *3D*-об’єкта

Рис. 1.29 – Налаштування матеріалу для *3D*-об’єкта

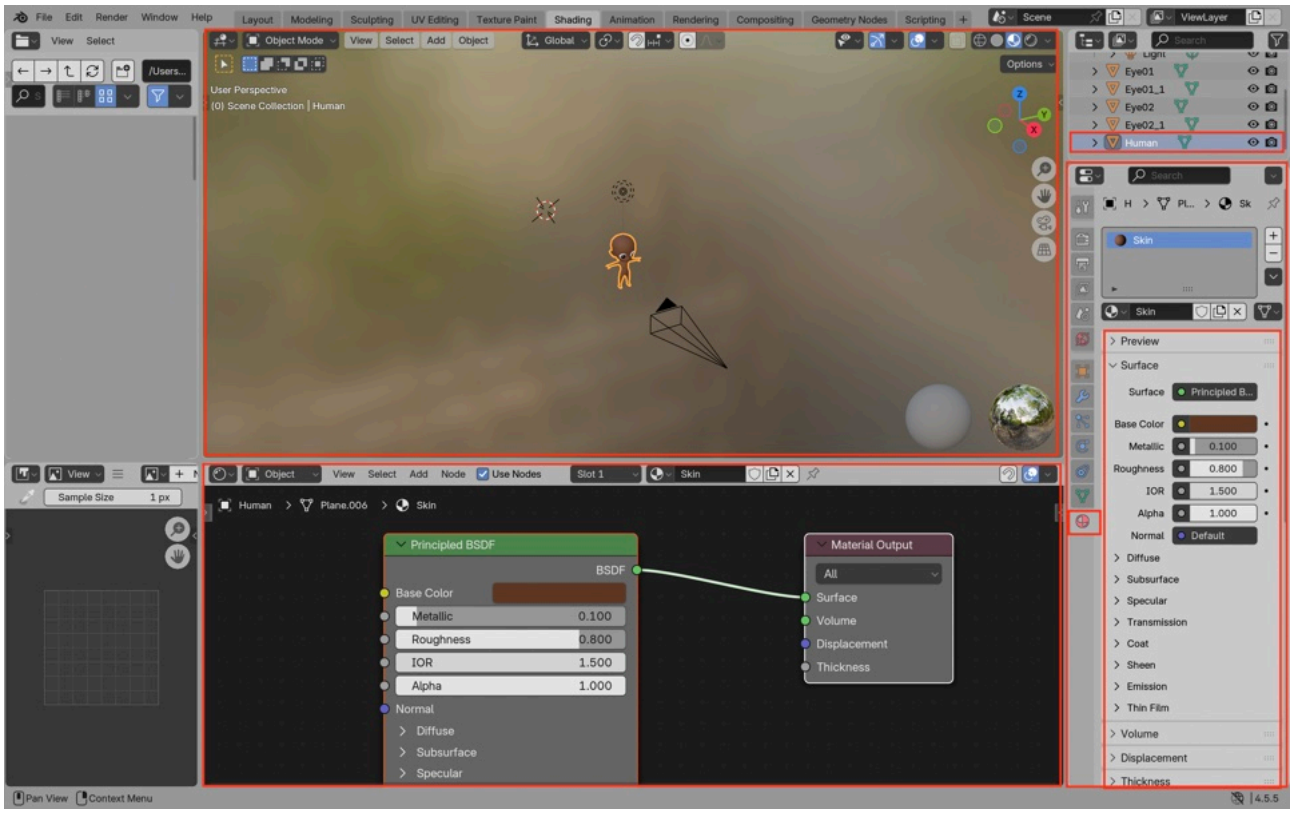


Рис. 1.30 – Налаштування матеріалів *3D*-об’єктів у робочому просторі *Shading*

Для створення матеріалу з колірним градієнтом, спочатку необхідно створити новий матеріал для 3*D*-об’єкта (волосся) в редакторі параметрів (рис. 1.31).

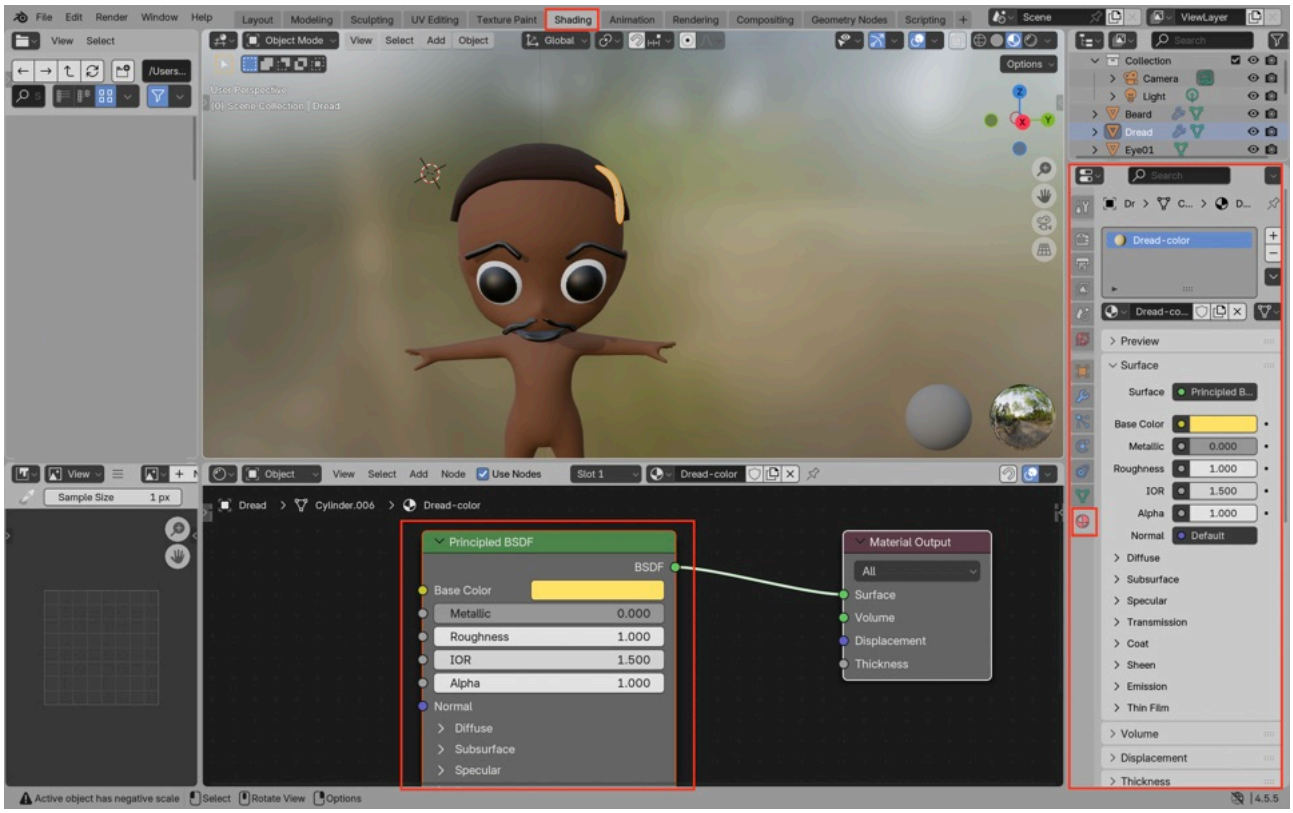


Рис. 1.31 – Створення та налаштування матеріалу *3D*-об’єкта за допомогою редактора *Shader Editor*

Наступним необхідно додати вузол колірної шкали, натиснути *Add-> Converter-> Color Ramp* (рис. 1.32) та з’єднати з базовим кольором вузла *Principled BSDF*. Параметри вузла колірної шкали необхідно налаштувати – додати ключові точки (при необхідності) та обрати основні кольори для градієнтного переходу (рис. 1.33).

Рис. 1.32 – Створення вузла колірної шкали (*Color Ramp*)

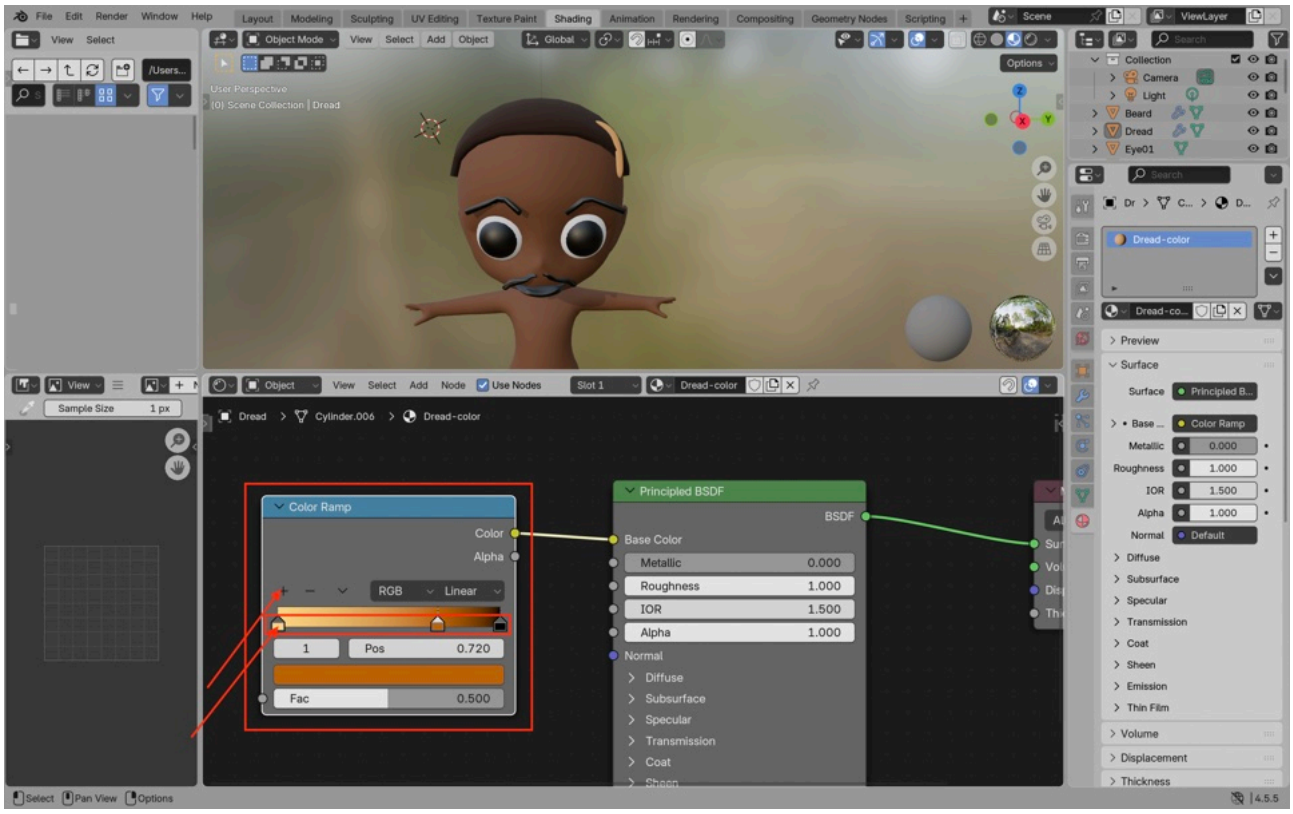


Рис. 1.33 – Налаштування вузла колірної шкали

Наступним необхідно створити та з’єднати вузли картографування та координат текстури, натиснути *Add-> Vector-> Mapping* та *Add->Input-> Texture Coordinate* (рис. 1.34). Вузли *Texture Coordinate* та *Mapping* використовуються для керування способом накладання текстури на поверхню *3D*-об’єкта шляхом вибору типу координат та їх трансформації. *Texture Coordinate* задає джерело координат, а *Mapping* дозволяє перетворювати текстуру (зміщувати, обертати, масштабувати) перед її застосуванням до матеріалу.

Рис. 1.34 – Налаштування вузлів картографування (*Mapping*) та координат текстури (*Texture Coordinate*)

За допомогою редактора параметрів та шейдерів у *Blender* визначаються фізичні та візуальні властивості всіх *3D*-об’єктів (рис. 1.35).

Рис. 1.35 – Результат налаштування матеріалів *3D*-аватара

1.2.3 Анімування та рендеринг *3D*-аватара

Анімування *3D*-аватара в *Blender* здійснюється за допомогою об’єкта Арматура (*Armature*), який є внутрішнім скелетом моделі та складається з системи кісток (*Bones*). Кістки Арматури прив’язуються до полігональної сітки *3D*-об’єкта, що дозволяє задавати різні рухи для компонентів *3D*-об’єкта. Після прив’язання *3D*-об’єкта до кісток Арматури створюються ключові кадри на часовій шкалі шляхом переміщення, обертання та масштабування кістки. Цей підхід до анімування забезпечує реалістичну анімацію *3D*-аватарів, персонажів або інших моделей складної геометричної форми.

Для додавання Арматури необхідно натиснути *Add-> Armature* (рис. 1.36). Для відображення арматури в режимах об’єкта, редагування та інших, необхідно в редакторі просторових параметрів об’єкта в пункті *Viewport Display* ввімкнути *In Front*.

Рис. 1.36 – Додавання та налаштування відображення об’єкта *Armature*

Наступним необхідно в режимі редагування створити та розташувати кістки Арматури на *3D*-аватарі за допомогою інструментів *Extrude*, переміщення, масштабування та обертання (рис. 1.37). Кількість, точність положення та орієнтації кісток Арматури визначають рівень реалістичності деформації *3D*-об’єкта під час анімації, але також впливають на складність сцени та обчислювальні витрати під час її рендерингу.

Для прив’язання *3D*-об’єкта до кісток Арматури необхідно в режимі об’єкта виділити спочатку *3D*-об’єкт та Арматуру. Далі необхідно натиснути *Object-> Parent-> With Automatic Weights* (рис. 1.38).

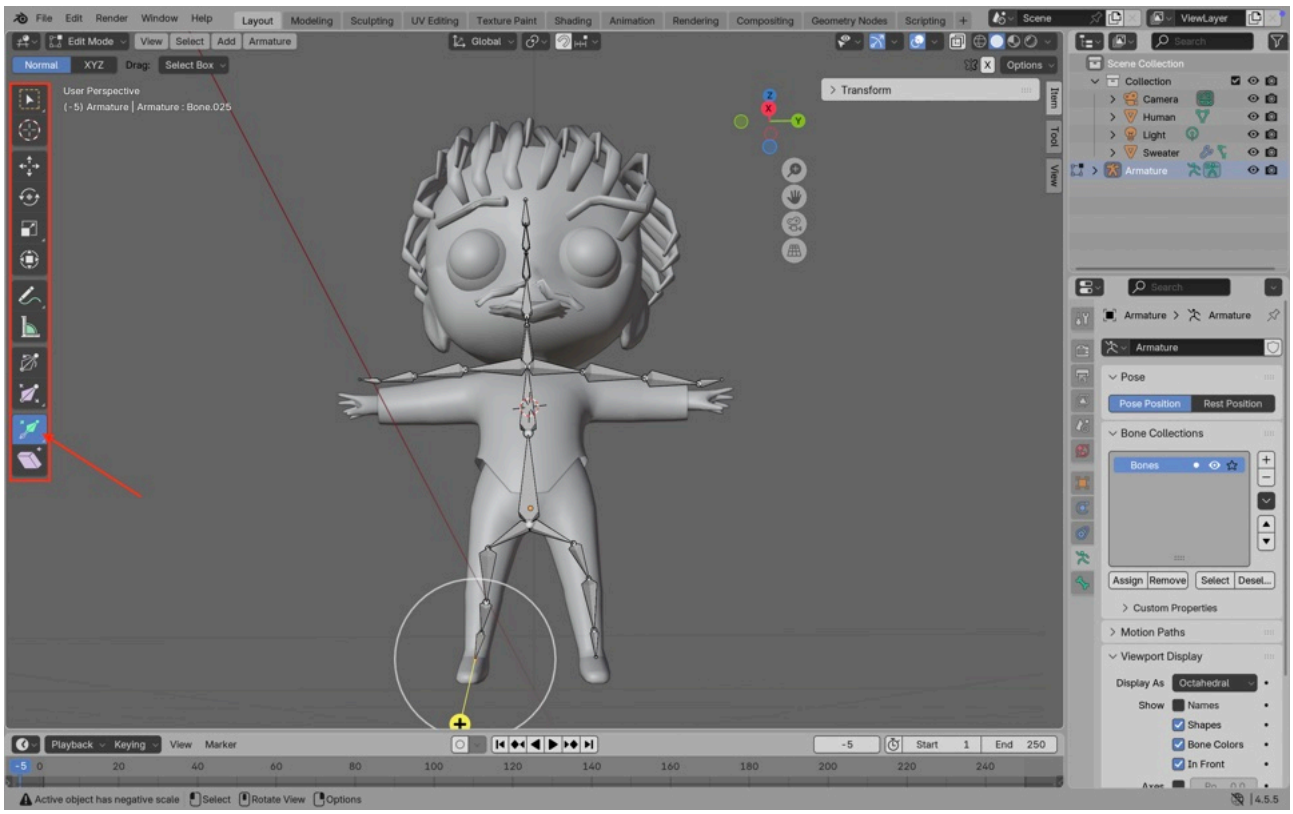


Рис. 1.37 – Створення кісток Арматури

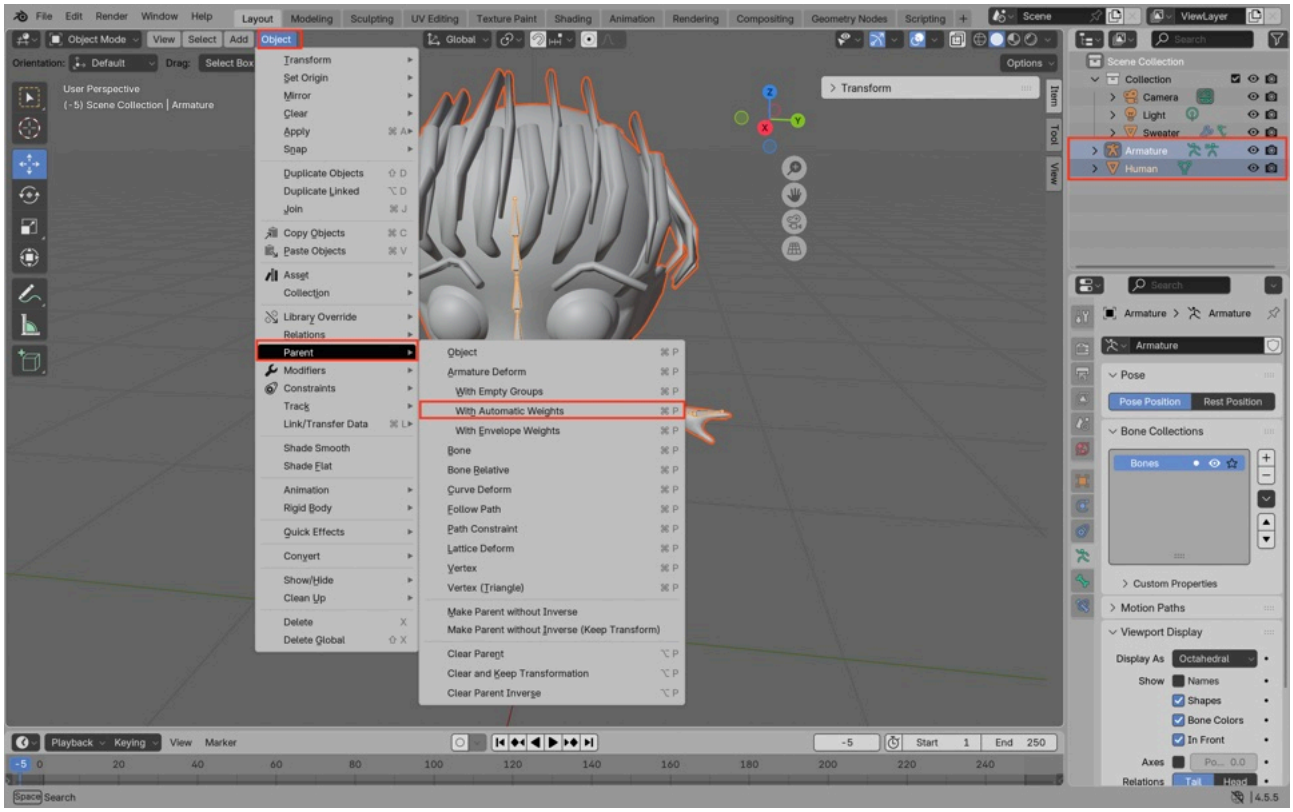


Рис. 1.38 – Прив’язання *3D*-об’єкта до кісток Арматури

Для інтерактивного налаштування ступеня впливу кісток Арматури на полігональну сітку *3D*-об’єкта під час анімації необхідно відкрити режим вагового фарбування (*Weight Paint*) та застосувати пензель Вага (*Weight*). Вага є числовим значенням від 0 до 1, що визначає ступінь деформації під час руху відповідної кістки: синій колір означає відсутність впливу, червоний – максимальний вплив, а проміжні кольори (напр., зелений, жовтий) показують часткову деформацію. Спочатку необхідно виділити Арматуру та *3D*-об’єкт, обрати режим вагового фарбування, налаштувати параметри пензля *Weight* (зокрема радіус і значення *Weight*) та визначити, які частини *3D*-об’єкта та з якою інтенсивністю реагують на рух кісток Арматури (рис. 1.39).

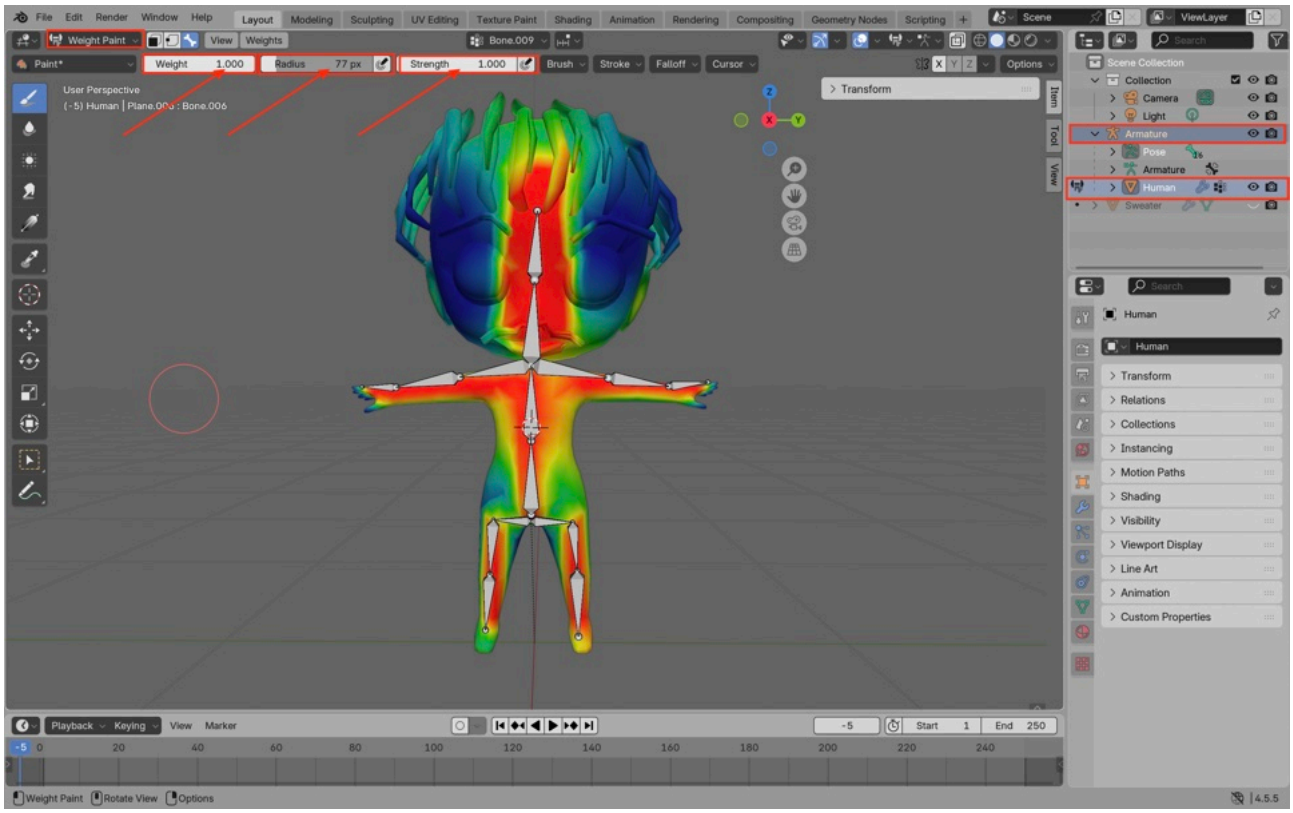


Рис. 1.39 – Режим вагового фарбування для налаштування *3D*-об’єкта

Анімація *3D*-об’єкта, на якого накладена Арматура створюється шляхом керування кістками, які деформують прив’язані до них грані полігональної сітки *3D*-об’єкта. Керування кістками виконується в режимі *Pose Mode*. На початковому кадрі часової шкали (*Timeline*) задається перша позиція кісток,

після чого створюється ключовий кадр – необхідно натиснути *ПКС-> Insert Keyframe* (або клавіша *I*) (рис. 1.40).

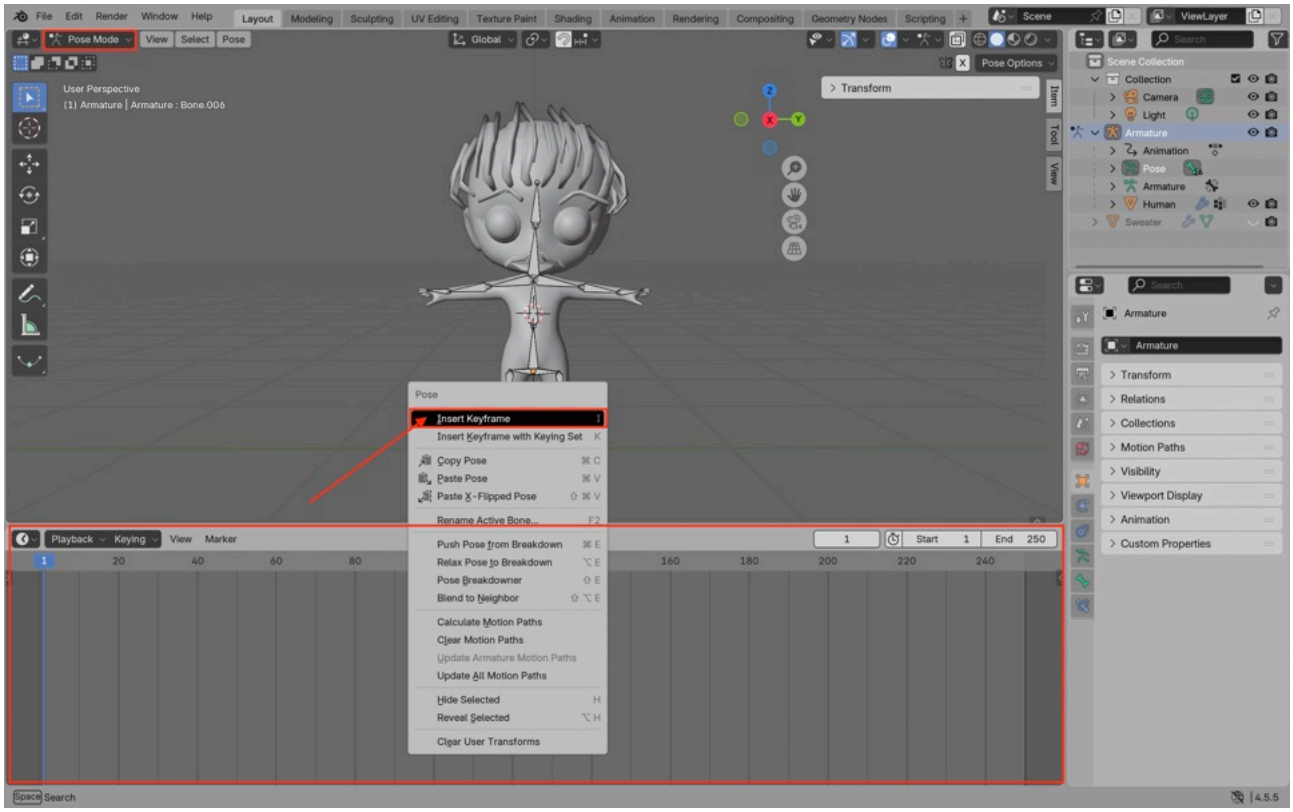


Рис. 1.40 – Створення ключового кадра на часовій шкалі (*Timeline*)

Створення інших ключових кадрів супроводжується зміною положення та орієнтації кісток. *Blender* автоматично інтерполює рух між заданими ключовими кадрами, створюючи плавну анімацію. Налаштування та редагування анімації також можна виконувати у робочому просторі *Animation* (рис. 1.41). У робочому просторі поєднані інструменти *Timeline* для перегляду й відтворення анімації, *Dope Sheet* для редагування ключових кадрів та *Graph Editor* для точного налаштування кривих інтерполяції руху.

Перед виконанням рендерингу необхідно налаштувати положення та орієнтацію камери в редакторі параметрів (рис. 1.42). Для зміни точки спостереження сцени в *Blender* з позиції активної камери необхідно натиснути клавішу ~ (тильда) та обрати пункт *View Camera*.

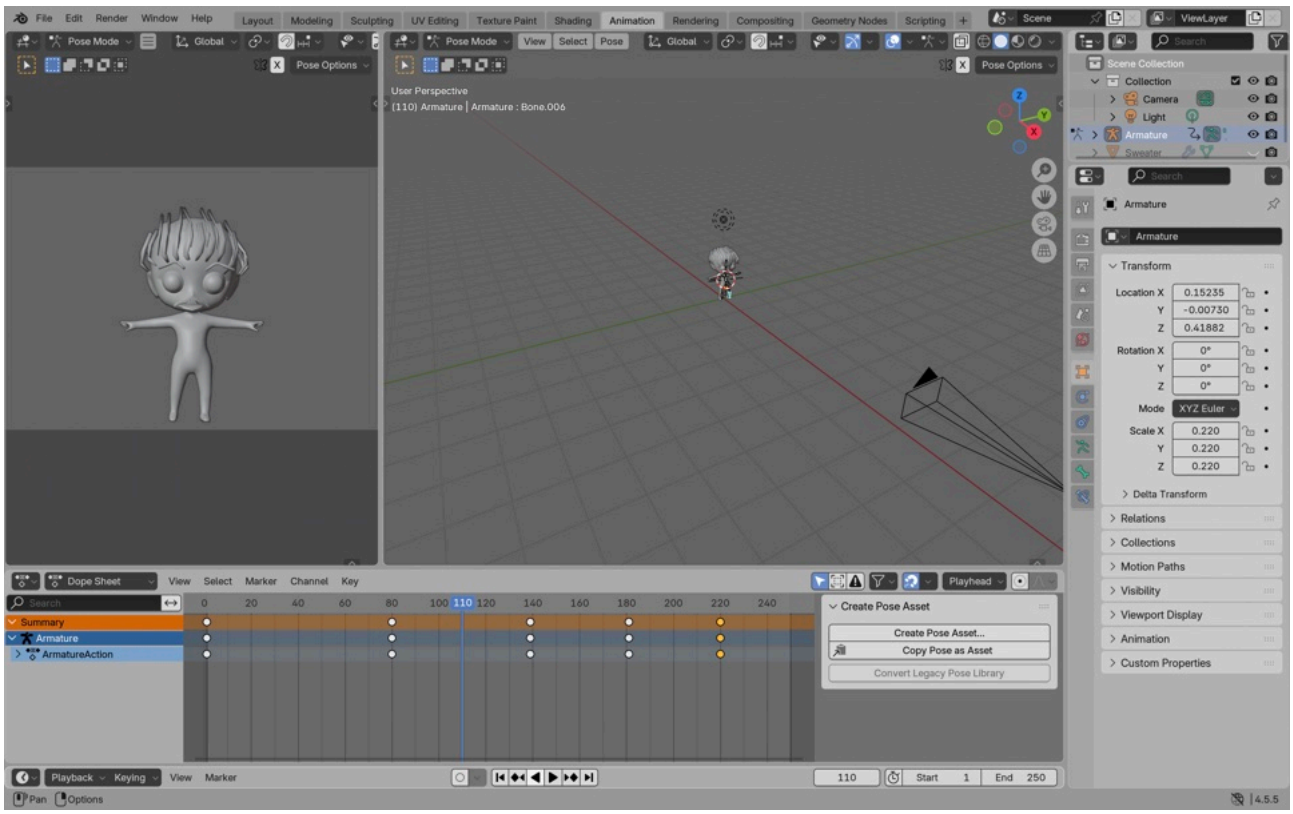


Рис. 1.41 – Створення анімації для *3D*-аватара у робочому просторі *Animation*

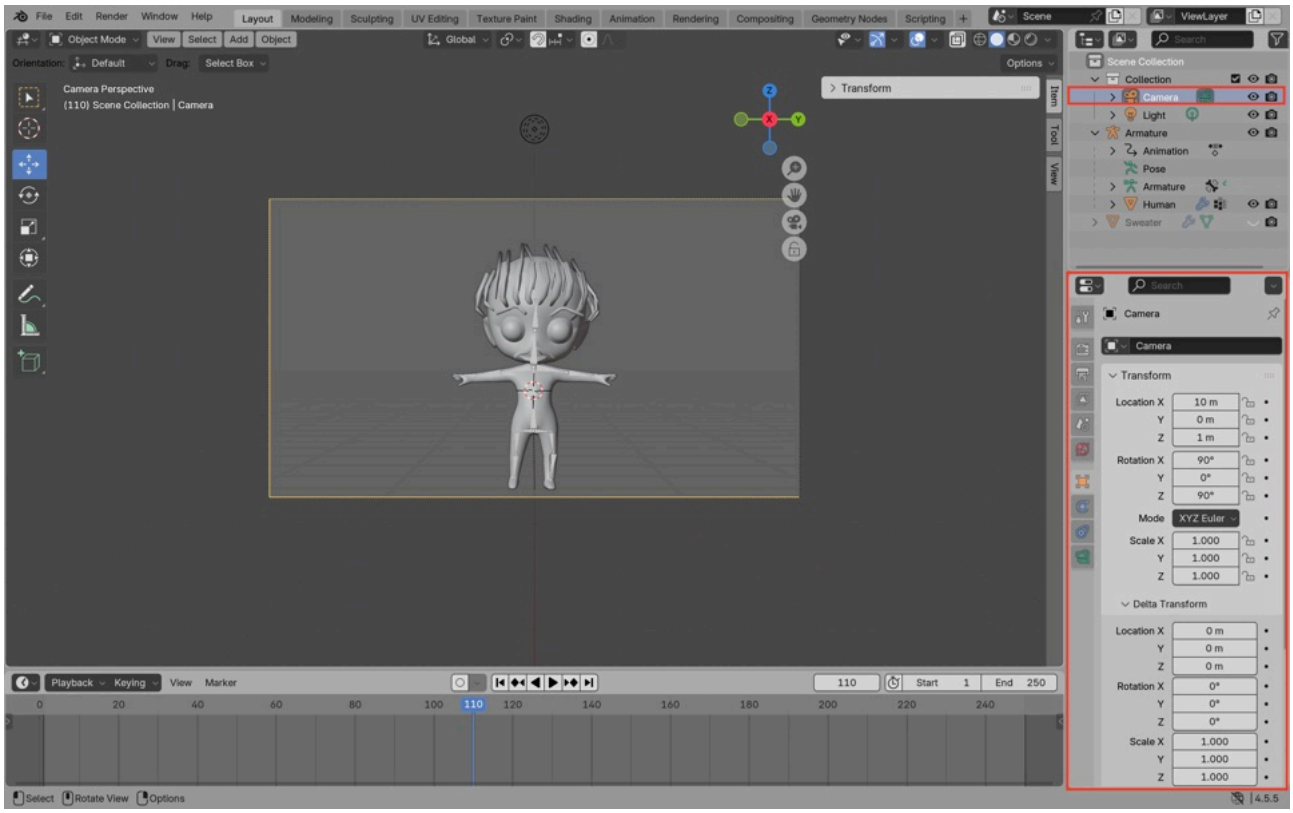


Рис. 1.42 – Налаштування параметрів об’єкта камера

Наступним необхідно налаштувати параметри джерел освітлення. Для зміни режиму відображення сцени в *Blender* необхідно натиснути клавішу *Z* та обрати пункт *Rendered*. До сцени додається декілька джерел освітлення, зокрема типу *Area*, який випромінює світло з певної площини. В редакторі параметрів задаються параметри позиції, орієнтації, кольору та інтенсивності світла, радіус та форма джерела світла (рис. 1.43).

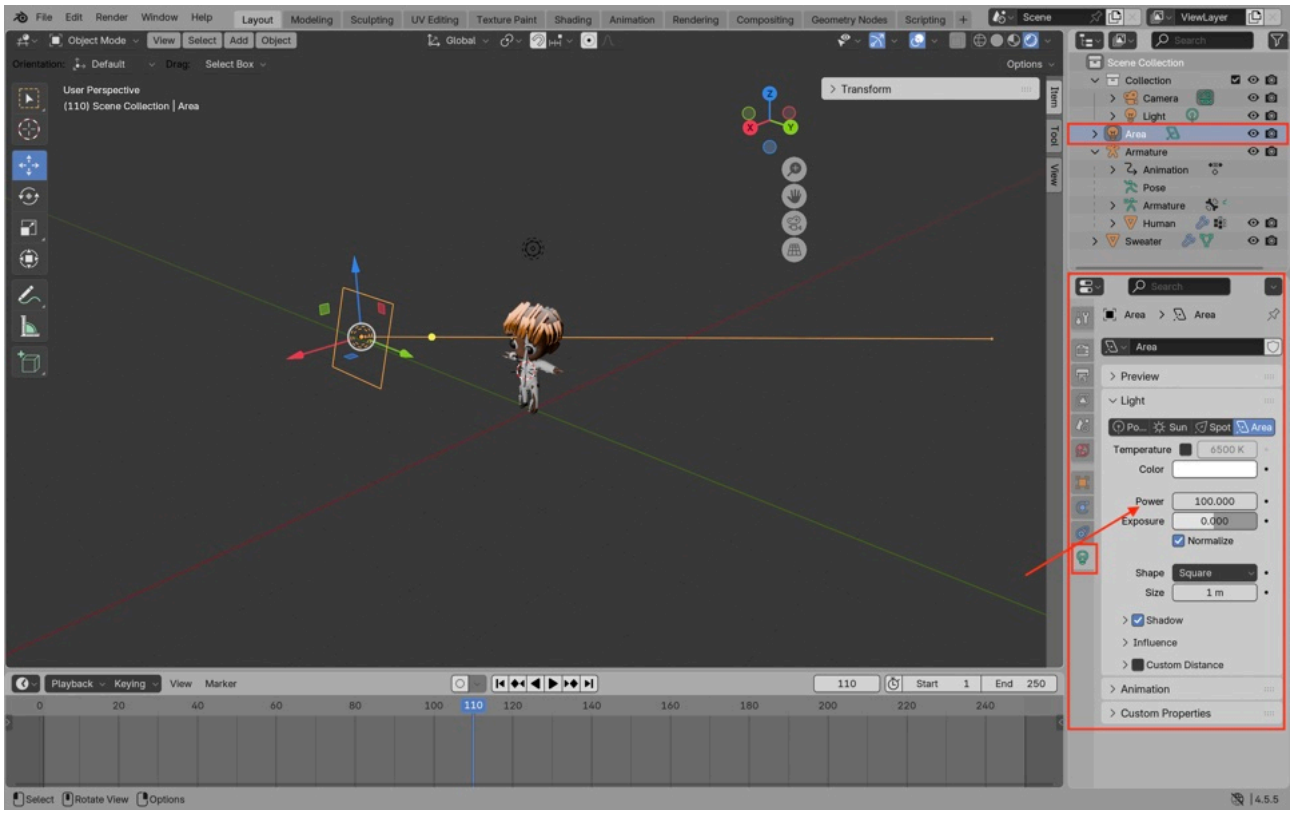


Рис. 1.43 – Налаштування параметрів об’єкта світло

Для рендерингу анімованої *3D*-сцени в редакторі параметрів *Render Properties* використовуються налаштування:

– *Render Engine*: *Cycles*;

Device: *GPU Compute.*

– *Sampling*

Render

Noise Threshold: 0.1;

Max Samples: 150;

Denoise: ввімкнено.

Інші параметри залишаються за замовчуванням.

В редакторі параметрів *Output Properties* використовуються налаштування:

– *Format:*

Resolution X: 800 *px*;

Resolution Y: 800 *px*;

Frame Rate: 24 *fpx.*

– *Frame Range:*

Frame Start: 1*;*

End: 40;

Step: 1.

Інші параметри залишаються за замовчуванням. В *Output Properties* також визначається формат файлу для збереження результату рендерингу та шлях його збереження.

Для запуску рендерингу після налаштування сцени, камери, світла та матеріалів необхідно натиснути *Render-> Render Animation*.

1.2.4 Демонстрація роботи

Результат рендерингу анімованої *3D*-сцени представлено на рис. 1.44.

Рис. 1.44 – Фрагмент анімації *3D*-сцени: рух правої ноги персонажа

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке моделювання *3D*-сцени і які його основні складові?
2. Які властивості *3D*-об’єктів і середовища змінюються під час анімації?
3. Які етапи включає процес рендерингу *3D*-сцени?
4. Що таке процедурний підхід до реалізації процесів *3D*-графіки та які програмні інструменти для цього використовуються?
5. Назвіть робочі простори в *Blender* та опишіть їх призначення у процесі створення *3D*-сцен.
6. Для чого призначений модифікатор *Mirror* в *Blender* та яким чином він використовується?
7. Для чого призначений модифікатор *Subdivision Surface* в *Blender* та яким чином він використовується?
8. Для чого призначений модифікатор *Solidify* в *Blender* та яким чином він використовується?
9. Для чого призначений інструмент *Extrude* в *Blender* та яким чином він використовується?
10. Для чого призначений інструмент *Poly Build* і чим він відрізняється від інших інструментів редагування полігональної сітки?
11. У чому полягає різниця між інструментами *Loop Cut* та *Knife* у полігональному моделюванні *Blender*?
12. У яких випадках доцільно застосовувати інструмент *Smooth Vertices*, операцію *Subdivide* та модифікатор *Subdivision Surface*, визначте їх функціональні відмінності?
13. Призначення скульптурування в *Blender*. Чим відрізняється скульптурування від полігонального моделювання?
14. Для чого призначений об’єкт Арматура (*Armature*) та яким чином він використовується?
15. Для чого призначений інструмент *Weight* у роботі з Арматурою та яким чином він впливає на анімацію моделі?

16. Які основні етапи створення анімації об’єкта в *Blender* за допомогою ключових кадрів?

17. Для чого використовується *Graph Editor* й яку інформацію можна там редагувати?

18. Які типи інтерполяції *keyframe* існують у *Blender* та як вони впливають на рух *3D*-об’єкта?

19. Для чого використовується *Pose Mode* у скелетній анімації?

20. Які типи систем частинок існують у *Blender* та для чого кожен тип зазвичай використовується?

ЛАБОРАТОРНИЙ ПРАКТИКУМ №2. РОЗРОБЛЕННЯ ЗАСТОСУНКУ ДОПОВНЕНОЇ РЕАЛЬНОСТІ В ІГРОВОМУ РУШІЇ *UNITY*

Лабораторний практикум присвячений розробленню *AR*-застосунку в ігровому рушії *Unity*. Для створення доповненої реальності на основі *iOS*-платформи використовується фреймворк *ARKit*, який доступний в *Unity* через плагін *Apple ARKit XR Plugin*. Для створення доповненої реальності на основі *Android*-платформи використовується *SDK ARCore*, який доступний в *Unity* через плагін *Google ARCore XR Plugin*. Для реалізації відстеження фізичного середовища та позиціювання цифрових об'єктів у ньому застосовується *Unity*-фреймворк *AR Foundation*.

В розділі «Порядок виконання» розглянутий процес розроблення двох видів *AR*-застосунків, зокрема на основі розпізнавання зображення та розпізнавання поверхонь.

ЗАВДАННЯ

Розробити *AR*-застосунок в ігровому рушії *Unity*. Тип доповненої реальності – маркерна або просторова (на вибір). *AR*-застосунок повинен містити *3D*-об'єкт(и) з лабораторного практикуму №1 (обов'язково), а також мультимедійні дані – *3D*-об'єкти з відкритих сторонніх джерел, аудіо, відео, текст (за бажанням). Для розробленого *3D*-об'єкта(ів) повинна бути реалізована анімація, а також застосовані відповідні текстури та матеріали.

Звіт з лабораторного практикуму повинен містити:

1. Титульний аркуш, формулювання завдання, текстовий опис процесу розроблення *AR*-застосунку, зокрема інформація щодо використаних інструментів та технологій, фрагменти програмного коду, скриншоти проміжних та ключових результатів (*.pdf* файл).

2. Демонстрацію роботи *AR*-застосунку (*.mp4* файл).

3. Архів програми (*.zip* файл).

2.1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Доповнена реальність є *XR*-технологією, що формує гібридне середовище, у якому цифрові об’єкти накладаються на об’єкти фізичного світу в реальному часі.

За способом накладання цифрових об’єктів на фізичне середовище доповнену реальність класифікують на чотири основні типи: маркерну (з використанням візуальних міток), просторову (інтеграція з фізичним середовищем), проєкційну (відображення на поверхнях об’єктів) та локаційно-орієнтовану (прив’язка до географічного положення користувача).

Розроблення застосунків розширеної реальності здійснюється зокрема за допомогою ігрових рушіїв (*Unity*, *Unreal Engine*, *Godot* та інші), які надають інструменти для створення тривимірних сцен, оброблення взаємодії користувача та інтеграції *XR*-технологій.

Unity є кросплатформним ігровим рушієм, що забезпечує розроблення *XR*-застосунків для різних платформ і пристроїв з використанням єдиного проєкту та спільного програмного коду. В *Unity* вбудовані інструменти для роботи з графікою, анімацією та взаємодією користувача у цифровому середовищі. *Unity* функціонує як інтегроване середовище, яке об’єднує проєктування сцен, програмування логіки та тестування *XR*-досвіду користувача.

Для отримання останньої версії ігрового рушія *Unity* необхідно перейти на офіційний сайт (https://unity.com/) та натиснути *Download*. Далі необхідно завантажити та встановити *Unity Hub* (рис. 2.1) відповідно до інструкцій на екрані. Після відкриття *Unity Hub* буде запропоновано встановити останню версію *Unity*. Зараз це версія *Unity 6.3 LTS (6000.3.3.f1)*. Зазвичай, остання версія *Unity* має додаткові функції, які попередня функція не підтримувала.

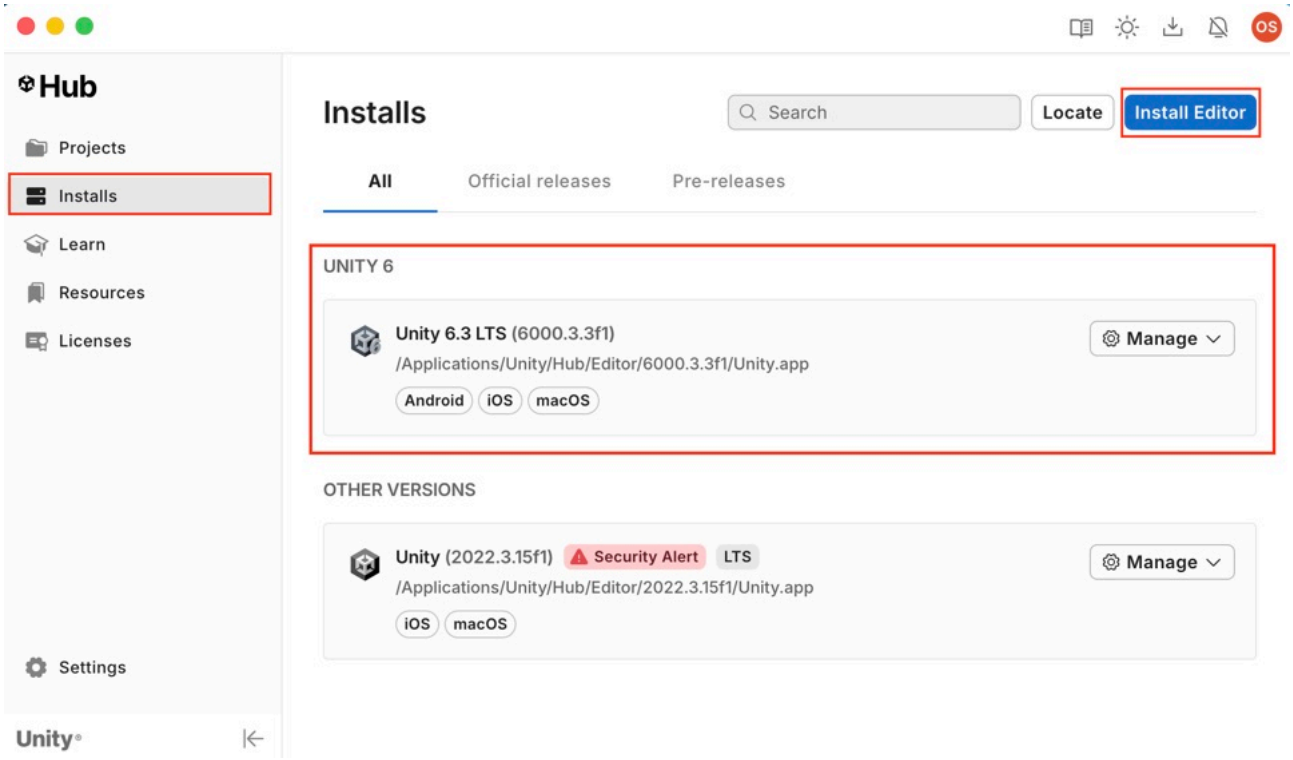


Рис. 2.1 – Версія *Unity 6.3 LTS (6000.3.3.f1)*

Існують випадки, коли необхідний *SDK* є несумісний з новою версією *Unity.* Тоді необхідно встановити попередню версію *Unity*, яка підтримує *SDK*. Для цього в *Unity Hub* необхідно обрати *Installs* та натиснути *Install Editor* (рис. 2.1). Далі у вікні *Install Unity Editor* необхідно обрати *Archive* та натиснути *download archive* (рис. 2.2). На відкритій сторінці необхідно обрати та завантажити необхідну версію *Unity* (рис. 2.3).

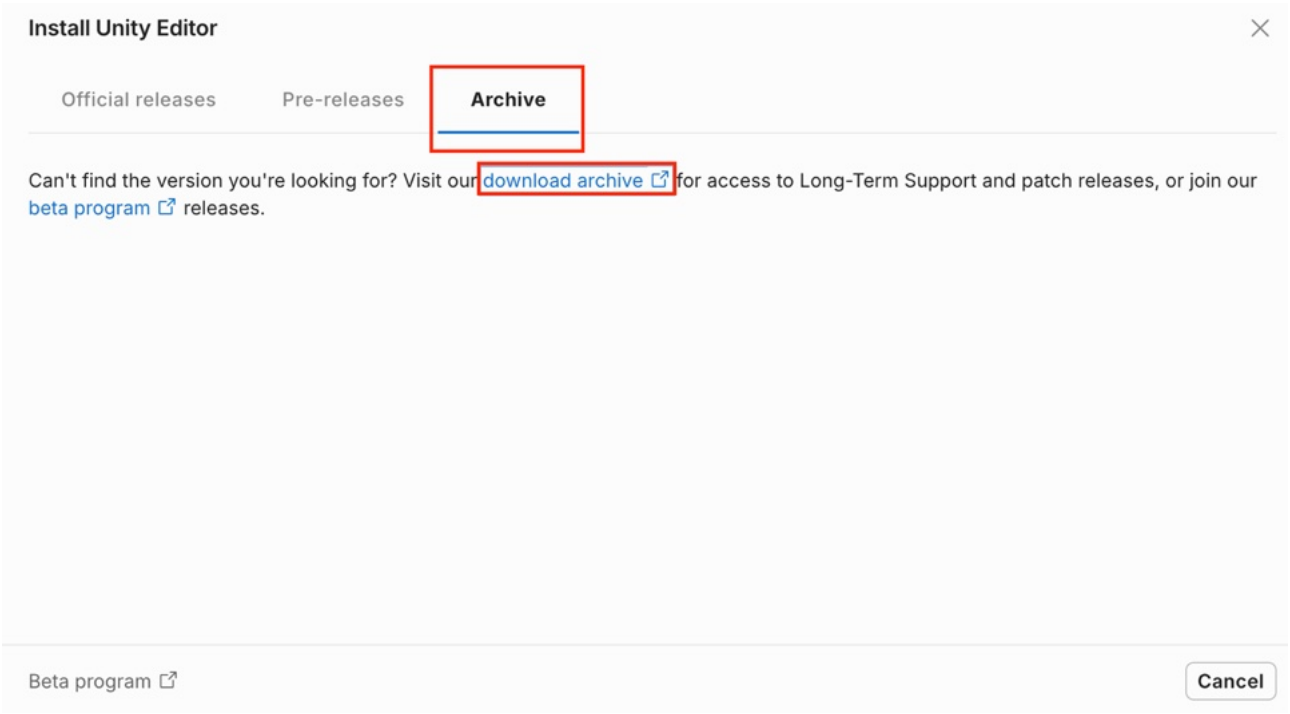


Рис. 2.2 – Вікно *Install Unity Editor*

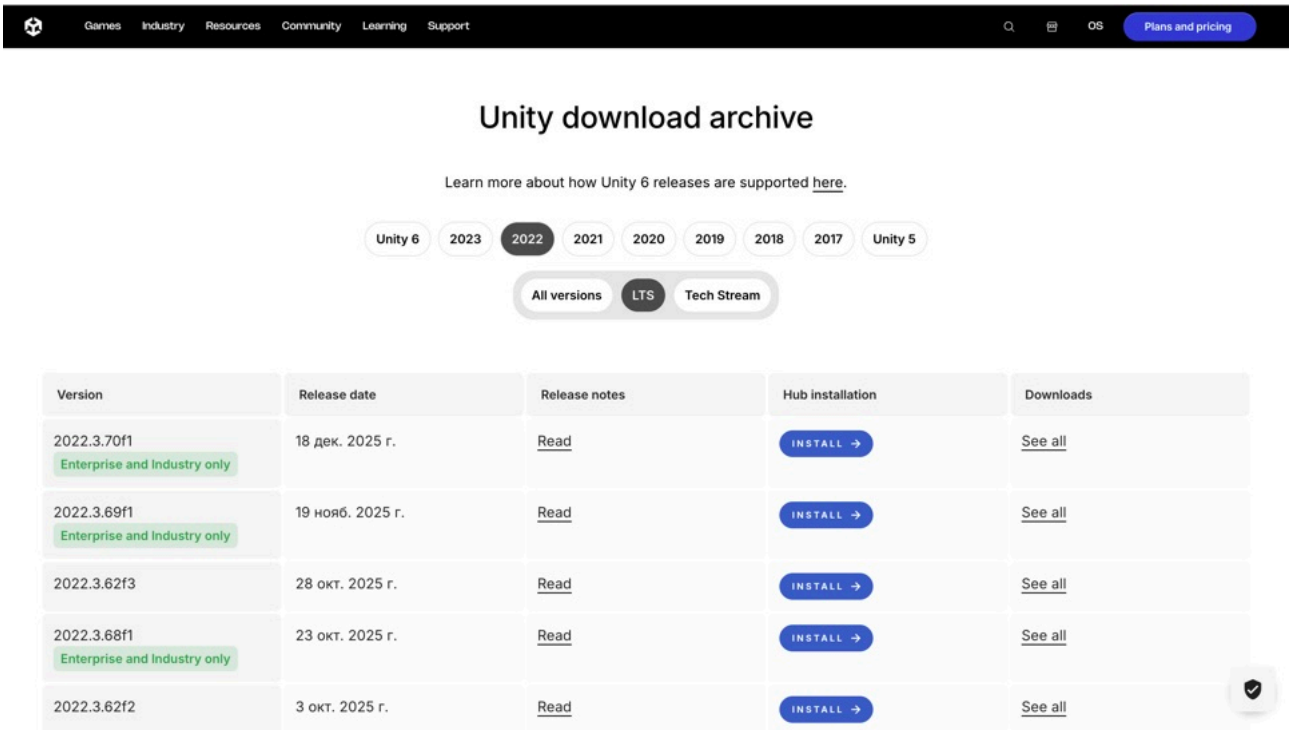


Рис. 2.3 – Архівні версії *Unity*

Для встановлення *Unity* необхідно в *Unity Hub* натиснути *Install Editor-> Install* та обрати модулі, які будуть використовуватись для створення *Unity*-проєктів (рис. 2.4). Головним інструментом розроблення є *Visual Studio*, для

розроблення програми на основі *Android*-платформи необхідно встановити модуль *Android Build Support,* на основі *iOS*-платформи – модуль *iOS Build Support* та натиснути *Continue*. Останнім необхідно погодитись на запропоновані умови на екрані та натиснути *Install* (рис. 2.5). Завантаження та встановлення *Unity* виконується протягом певного часу.

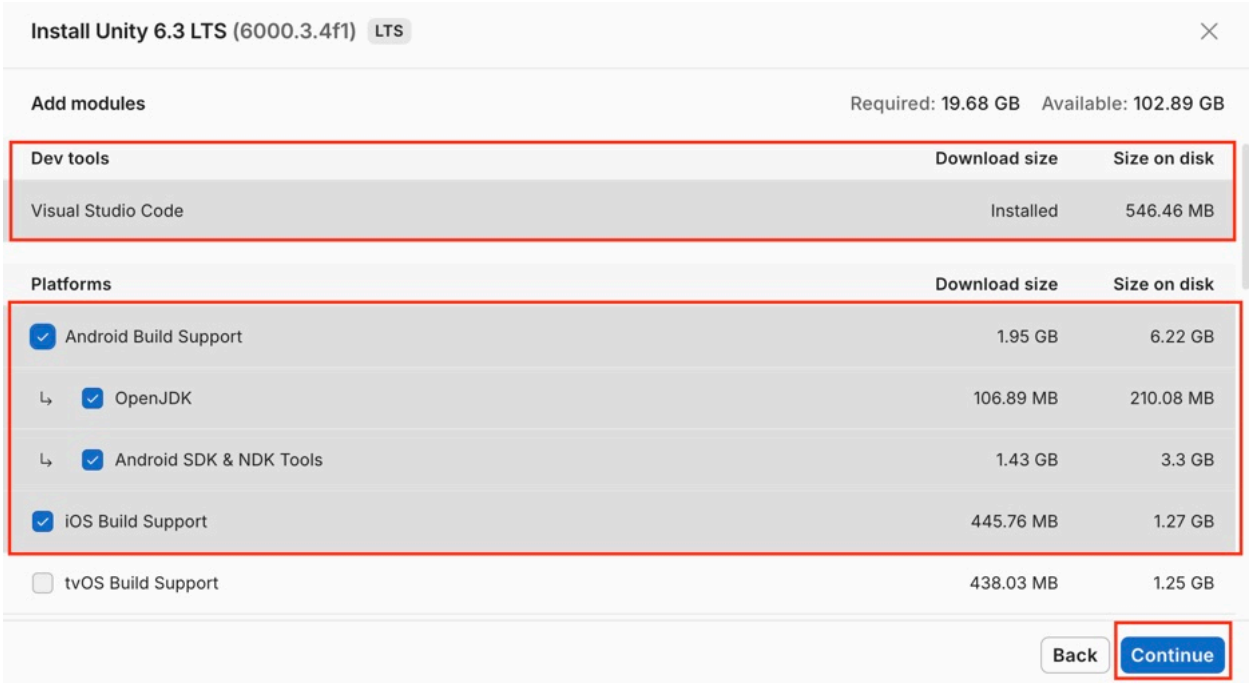


Рис. 2.4 – Модулі *Unity*

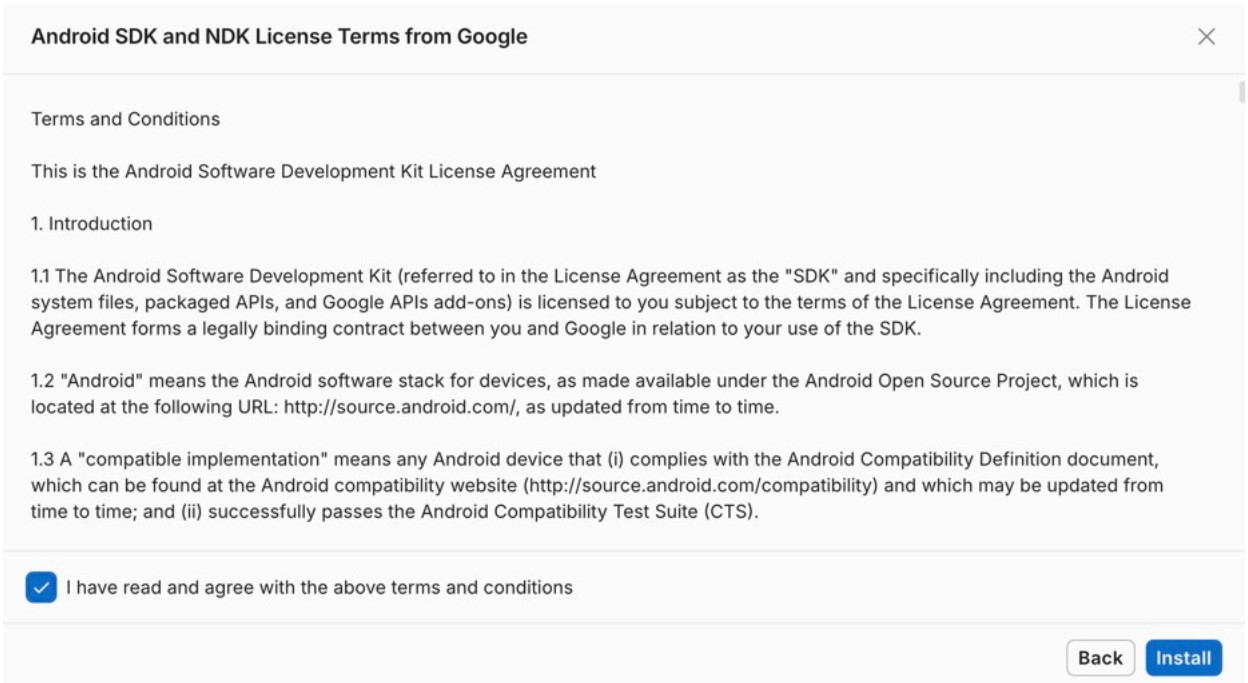


Рис. 2.5 – Умови використання *Unity*

Для створення нового *Unity*-проєкту в *Unity Hub* необхідно обрати *Projects* та натиснути *New Project*. Далі необхідно обрати версію *Unity*, шаблон проєкту (напр., *Universal 3D*), дати назву *Unity*-проєкту та натиснути *Create project* (рис. 2.6). Завантаження нового *Unity*-проєкту виконується протягом певного часу.

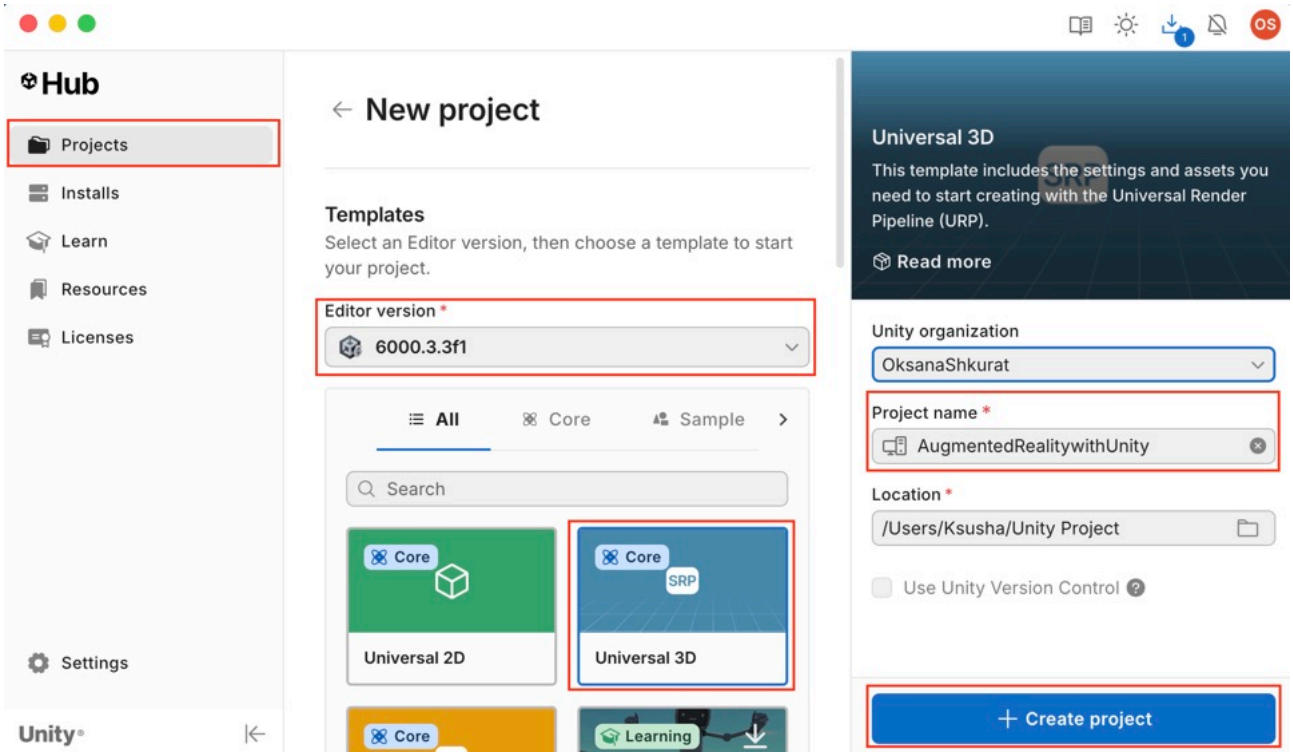


Рис. 2.6 – Створення нового *Unity*-проєкту

Редактор *Unity* містить декілька основних вікон (рис. 2.7) для керування ігровими об’єктами, проєктування сцени, налаштування параметрів ігрових об’єктів і контроль помилок під час розроблення застосунків. Вікно *Hierarchy* призначене для керування (додавання нових ігрових об’єктів (*Game Object*), дублювання, видалення, створення зв’язків тощо) ієрархією всіх ігрових об’єктів *Unity*-проєкту. Вікно *Scene* призначене для візуального проєктування ігрових об’єктів (визначення позиції, орієнтації, масштабу ігрових об’єктів). Вікно *Inspector* призначене для налаштування параметрів ігрових об’єктів через компоненти. Компоненти в *Unity* є функціональними модулями, які додаються до ігрових об’єктів та визначають їх поведінку. Поведінка ігрового об’єкта визначається доданою компонентою або групою компонент, які реалізують

механізми взаємодії, оброблення введення користувача, фізичні властивості, візуальне відображення та інше. Вікно *Console* призначене для контролю помилок, які виникають під час розроблення та виконання *Unity*-проєкту. Вікно *Project* призначене для зберігання всіх ресурсів *Unity*-проєкту (напр., скрипти, *3D*-моделі, матеріали, текстури, сцени тощо). Вікно *Game* призначене для відображення поточного варіанту виконання *Unity*-проєкту, яке буде бачити користувач застосунку.

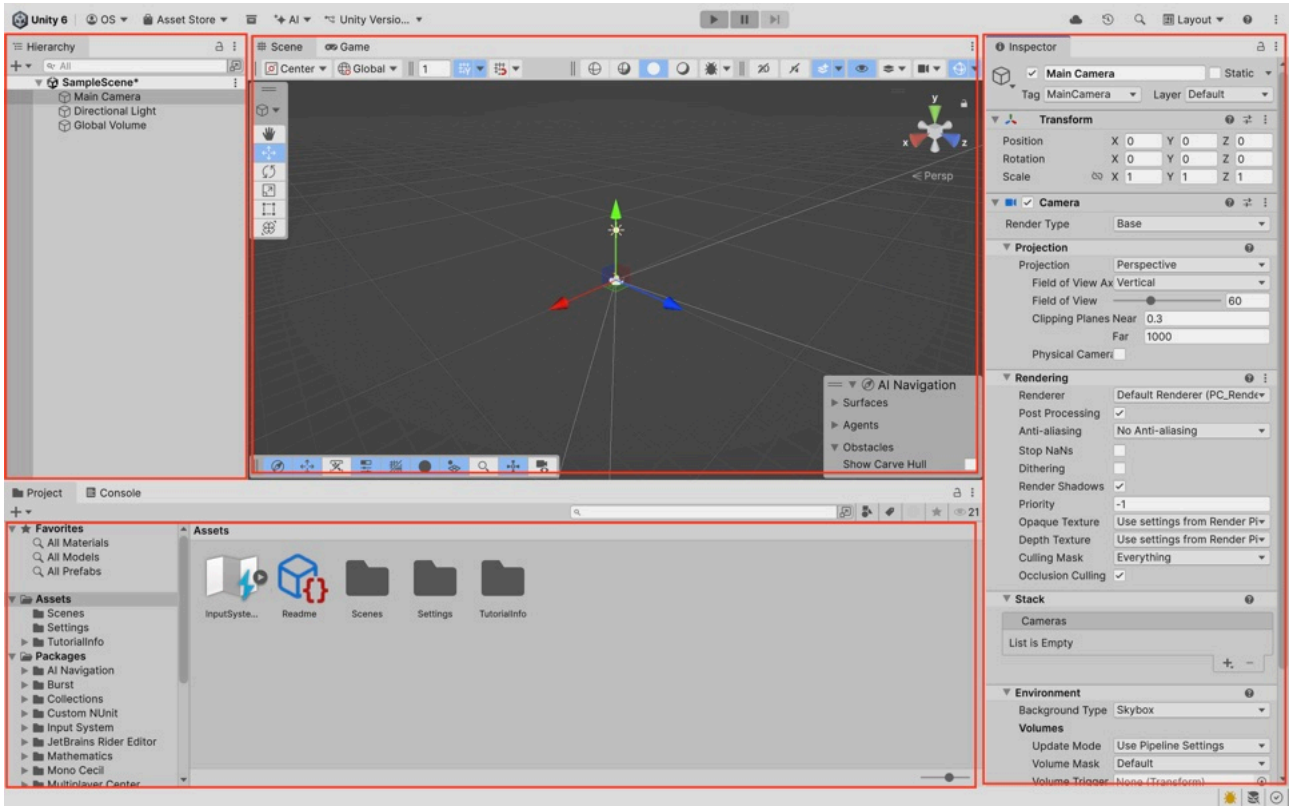


Рис. 2.7 – Вікно редактора *Unity*

1) Спочатку необхідно обрати платформу для якої буде розроблений *AR*-застосунок. Для цього необхідно обрати *File->Build Profiles*. Далі необхідно обрати платформу та натиснути *Switch Platform* (рис. 2.8).

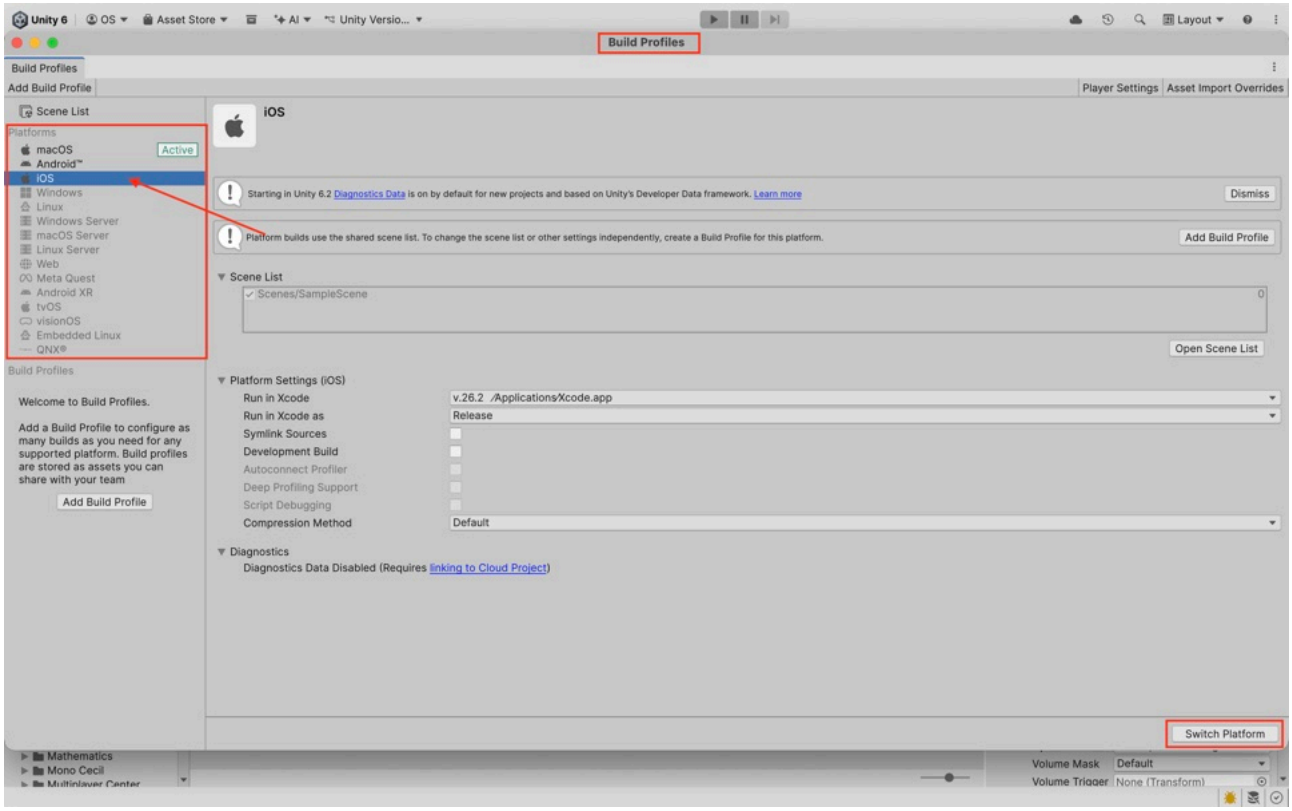


Рис. 2.8 – Вибір платформи *AR*-застосунку

2) *Unity* підтримує сучасні технології розроблення *AR*-застосунків через *AR Foundation, XR Plug-in Management* та інші.

AR Foundation є інтерфейсом *Unity* для роботи з різними *SDK* доповненої реальності, зокрема *ARKit* та *ARCore*. *AR Foundation* не реалізує *AR*-функції самостійно, а забезпечує доступ до них, зокрема відстеження положення та орієнтації *AR*-пристрою, фізичних горизонтальних та вертикальних поверхонь, зображень, об'єктів, визначення освітлення фізичного світу, розміщення цифрових об'єктів та взаємодія з ними. Основними компонентами *AR Foundation* є *AR Session* (керування життєвим циклом *AR*-досвіду), *AR Session Origin* (система координат *AR*-сцени), *AR Camera* (камера для візуалізації фізичного світу), *AR Plane Manager* (керування відстеженням фізичних горизонтальних та вертикальних поверхонь), *AR Tracked Image Manager* (керування відстеженням зображень), *AR Raycast Manager* (керування взаємодією з фізичними горизонтальними та вертикальними поверхнями).

AR Foundation доцільно використовувати для розроблення застосунків маркерної та просторової доповненої реальності.

Для завантаження *AR Foundation* в *Unity*-проєкт спочатку необхідно обрати *Window-> Package Management-> Package Manager*. Далі необхідно обрати реєстр пакетів *Unity-> AR Foundation* та натиснути *Install* (рис. 2.9). Встановлення та налаштування *AR Foundation* в *Unity*-проєкті виконується протягом певного часу.

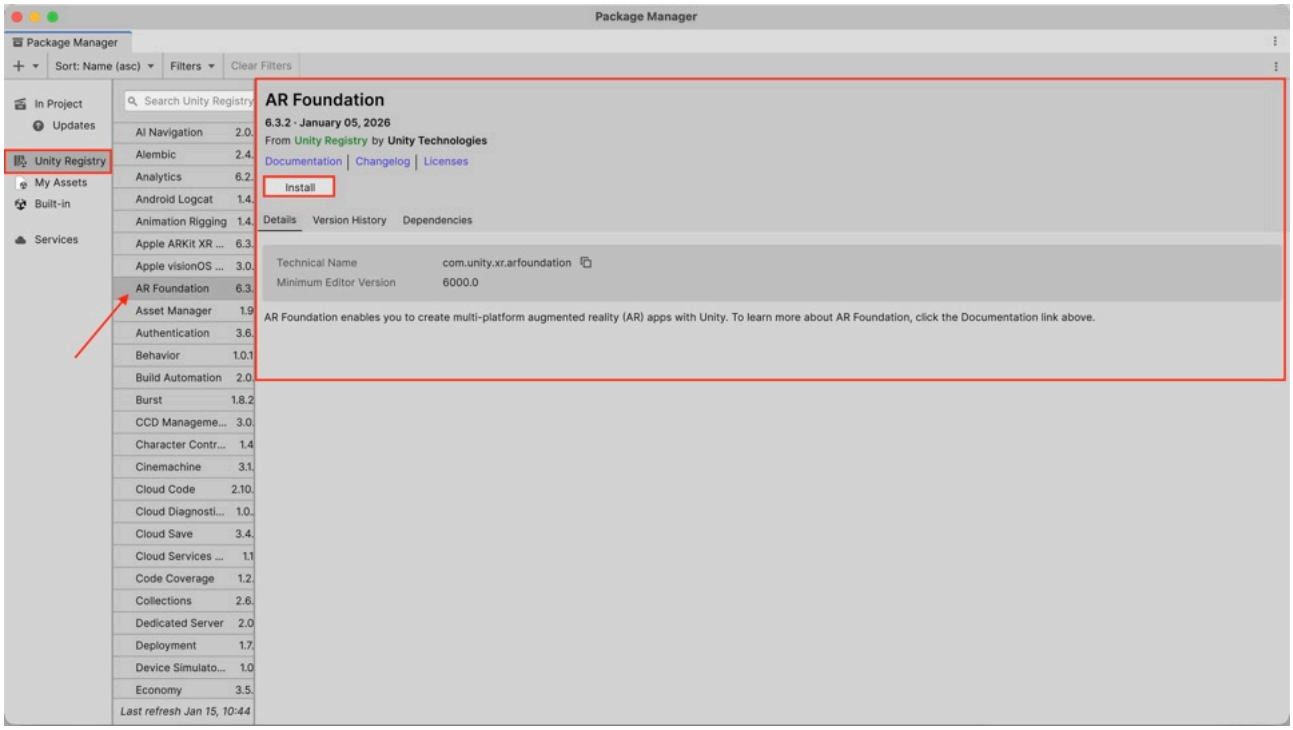


Рис. 2.9 – Встановлення *AR Foundation* в *Unity*-проєкт

Наступним необхідно встановити *SDK* доповненої реальності на основі обраної платформи. Для *iOS*-платформи необхідно обрати та встановити плагін *Apple ARKit XR Plugin* (рис. 2.10). Для *Android*-платформи необхідно обрати та встановити плагін *Google ARCore XR Plugin* (рис. 2.11).

Рис. 2.10 – Встановлення *Apple ARKit XR Plugin* в *Unity*-проєкт

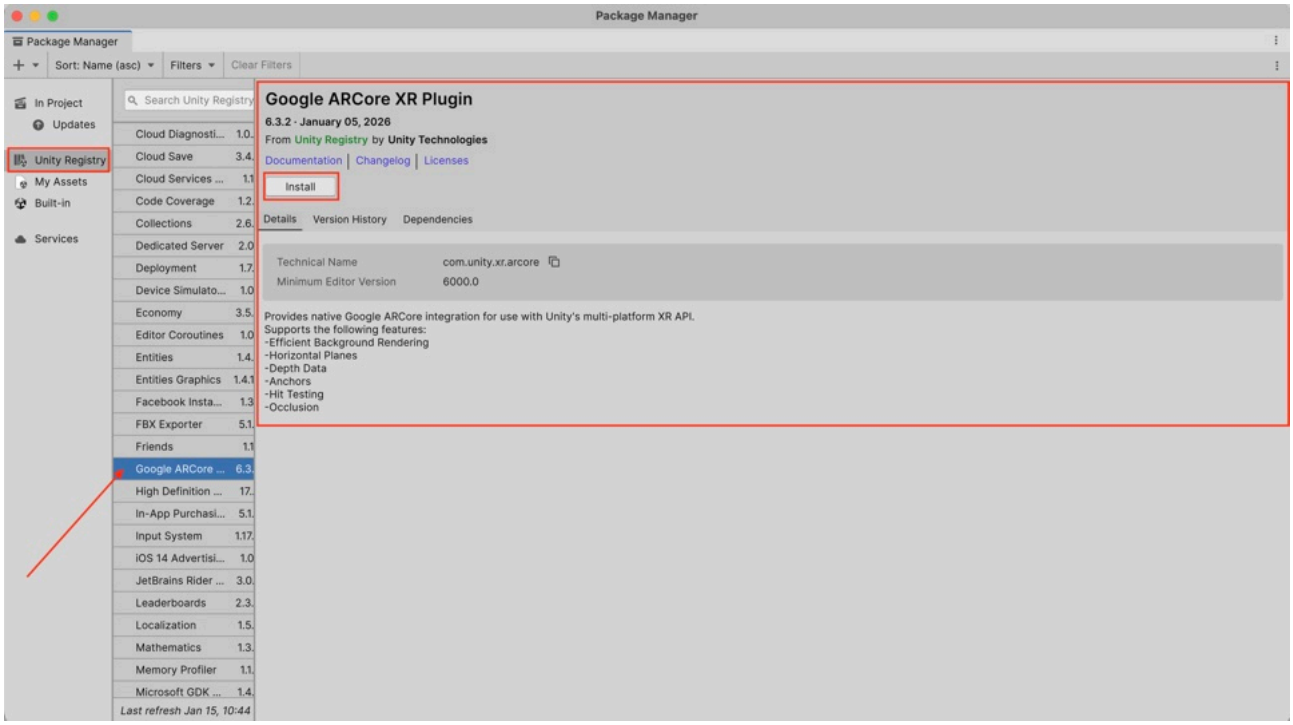


Рис. 2.11 – Встановлення *Google ARCore XR Plugin* в *Unity*-проєкт

Останнім необхідно налаштувати *SDK* доповненої реальності. *XR Plug-in Management* є системою керування *XR*-плагінами (*ARKit*, *ARCore*, *OpenXR* тощо) в *Unity*. *XR Plug-in Management* дозволяє керувати підключенням *XR*-

технологій в *Unity*-проєкті, налаштовувати параметри плагінів через *Project Settings* та запобігати помилкам на етапі збірки *XR*-застосунку.

Для налаштування *SDK* доповненої реальності в *Unity*-проєкті спочатку необхідно обрати *Edit-> Project Settings-> XR Plug-in Management* та обрати *ARKit* (рис. 2.12) або *ARCore* (рис. 2.13) на основі обраної платформи.

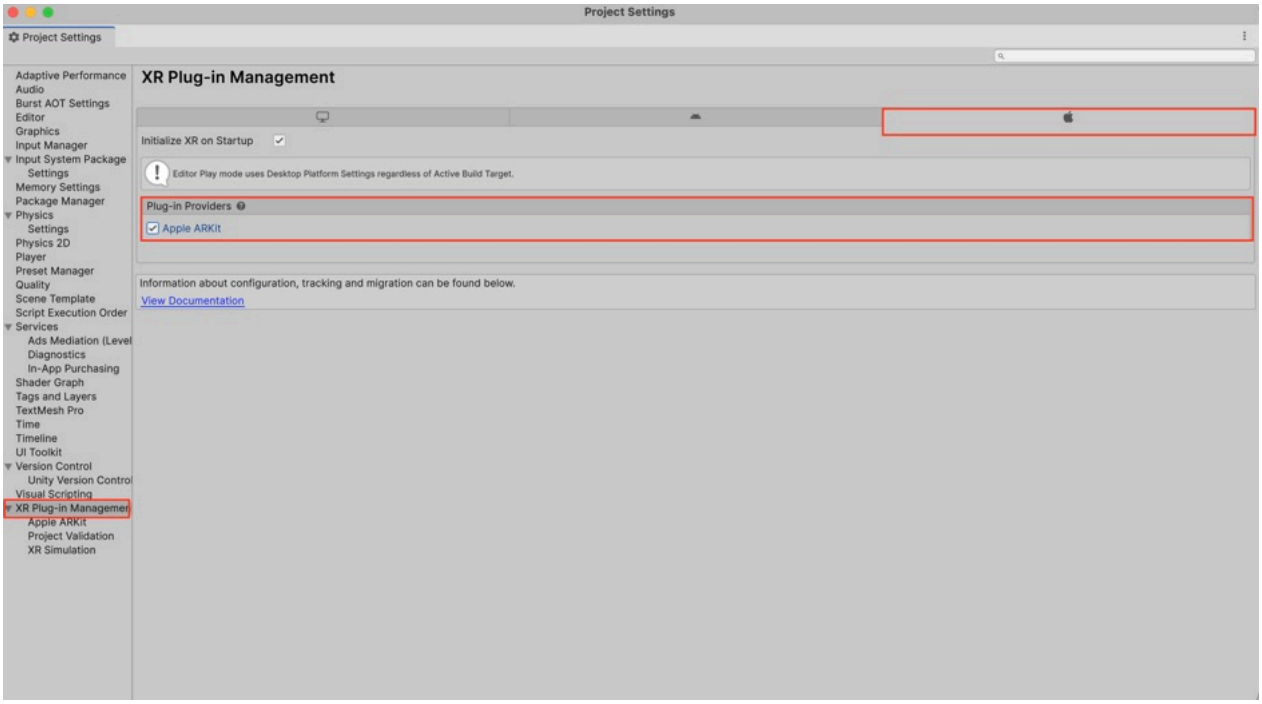


Рис. 2.12 – Налаштування *ARKit* в *Unity*-проєкт

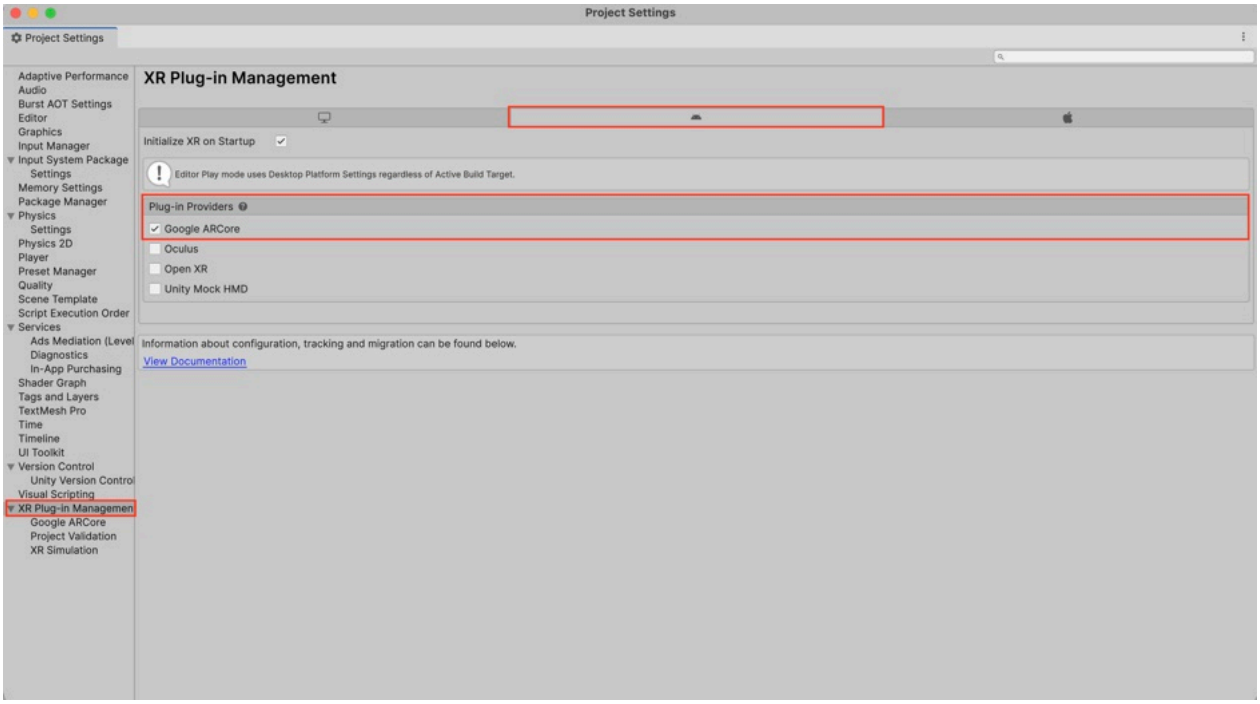


Рис. 2.13 – Налаштування *ARCore* в *Unity*-проєкт

В *Unity*-проєкті *XR Plug-in Management* встановлюється автоматично після встановлення та налаштування *AR Foundation.* Для самостійного встановлення *XR Plug-in Management* необхідно обрати *Edit-> Project Settings-> XR Plugin Management* та натиснути *Install XR Plugin Management* (рис. 2.14).

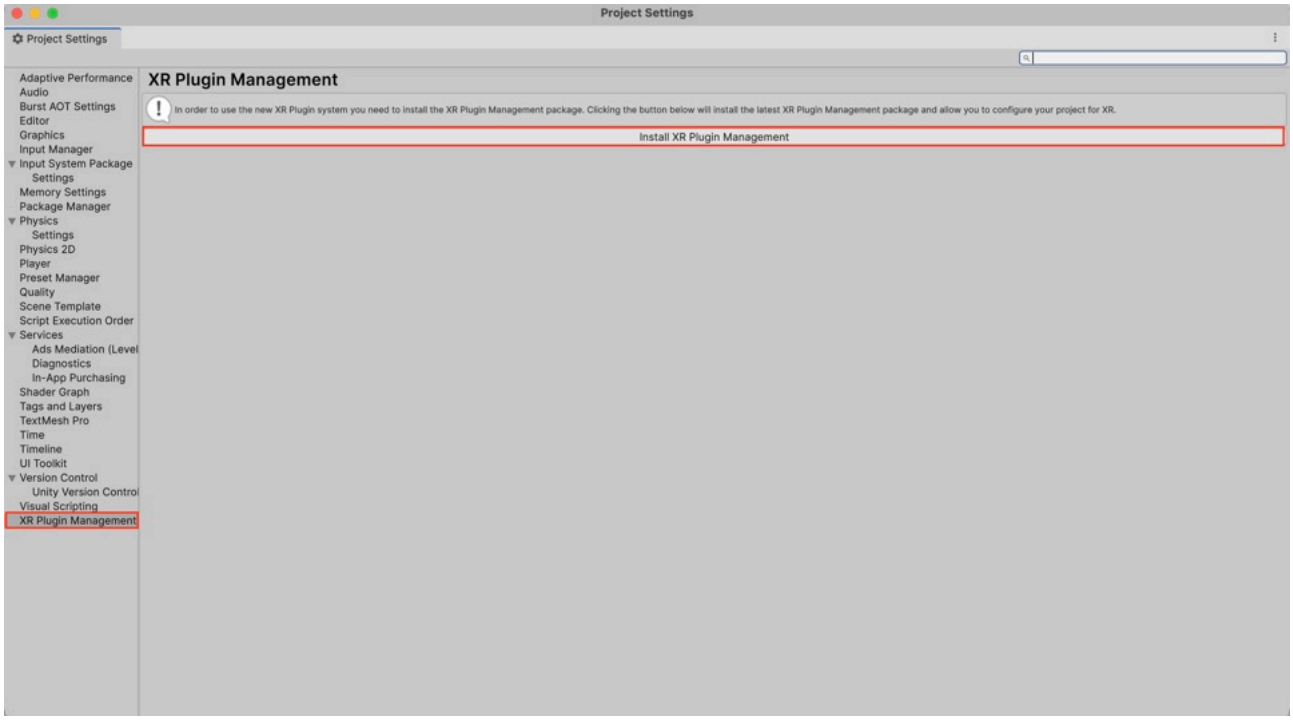


Рис. 2.14 – Встановлення *XR Plug-in Management* в *Unity*-проєкт

3) Головними ігровими об’єктами доповненої реальності в *Unity* є *ARSession* та *XR Origin*. Перед додаванням та налаштуванням *ARSession* та *XR Origin*, необхідно видалити ігровий об’єкт *Main Camera* з *Unity*-проєкту. У вікні *Hierarchy* необхідно натиснути на ігровий об’єкт *Main Camera* правою кнопкою миші та обрати *Delete*.

Для додавання ігрового об’єкта *ARSession* у вікні *Hierarchy* необхідно натиснути правою кнопкою миші та обрати *XR-> ARSession*. В результаті в *Unity*-проєкт буде додано ігровий об’єкт *ARSession* (рис. 2.15) з компонентою *ARSession*, яка відповідає за керування життєвим циклом *AR*-досвіду шляхом ввімкнення/вимкнення доповненої реальності на цільовому пристрої та компонентою *ARInputManager*, яка відповідає за відстежування та доступ до фізичного світу. Для однієї *AR*-сцени необхідно використовувати тільки одну

компоненту *ARSession* та одну компоненту *ARInputManager*. Використання декількох компонент може призвести до конфліктів.

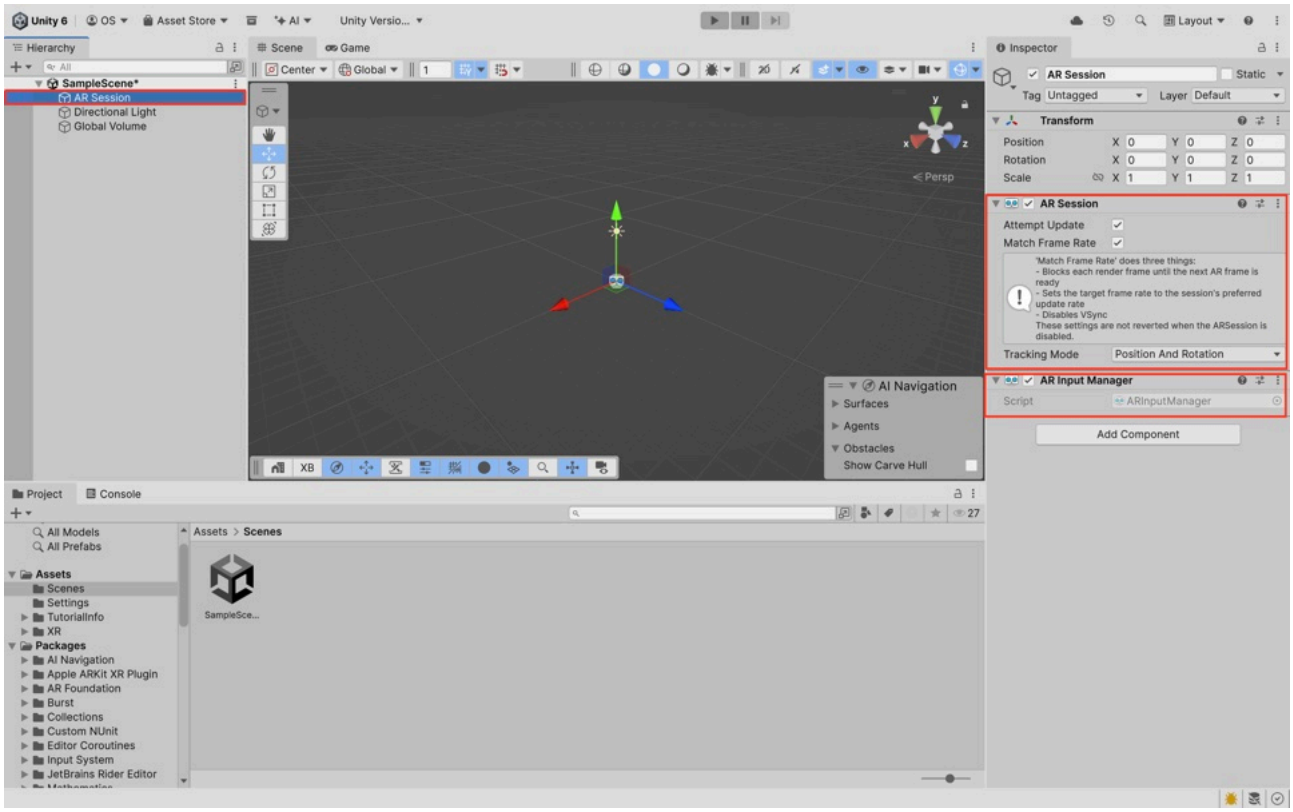


Рис. 2.15 – Ігровий об’єкт *ARSession*

Для додавання ігрового об’єкта *XR Origin* у вікні *Hierarchy* необхідно натиснути правою кнопкою миші та обрати *XR-> XR Origin (Mobile AR)*. В результаті в *Unity*-проєкт буде додано ігровий об’єкт *XR Origin* (рис. 2.16) з компонентою *XR Origin*, яка відповідає за визначення позиції, орієнтації та масштабу відстежуваних фізичних поверхонь та точок об’єктів в сцені *Unity*.

Ігровий об’єкт *XR Origin* має дочірній ігровий об’єкт *Main Camera* з компонентою *AR Camera Manager*, яка відповідає за налаштування різних параметрів, зокрема автофокус, оцінка освітлення, напрямок камери та режим рендерингу (рис. 2.17).

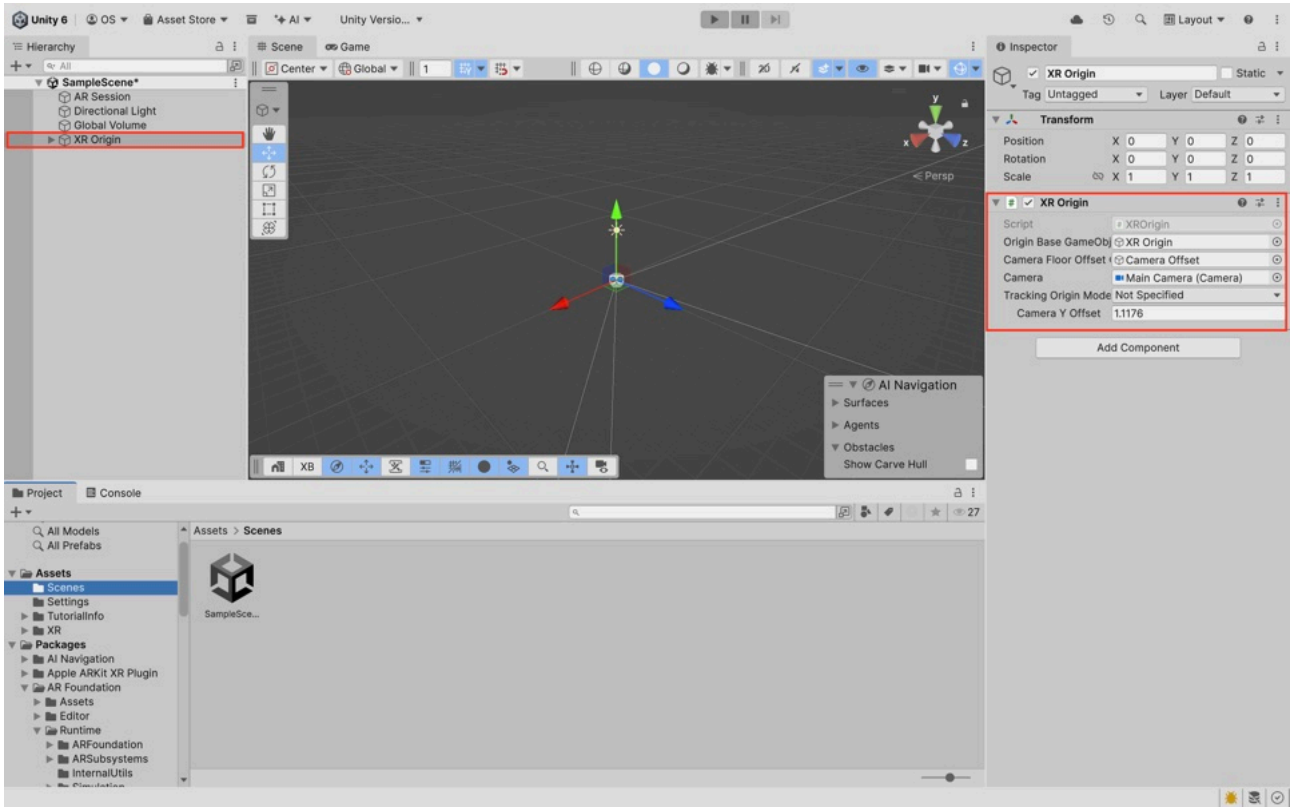


Рис. 2.16 – Ігровий об’єкт *XR Origin*

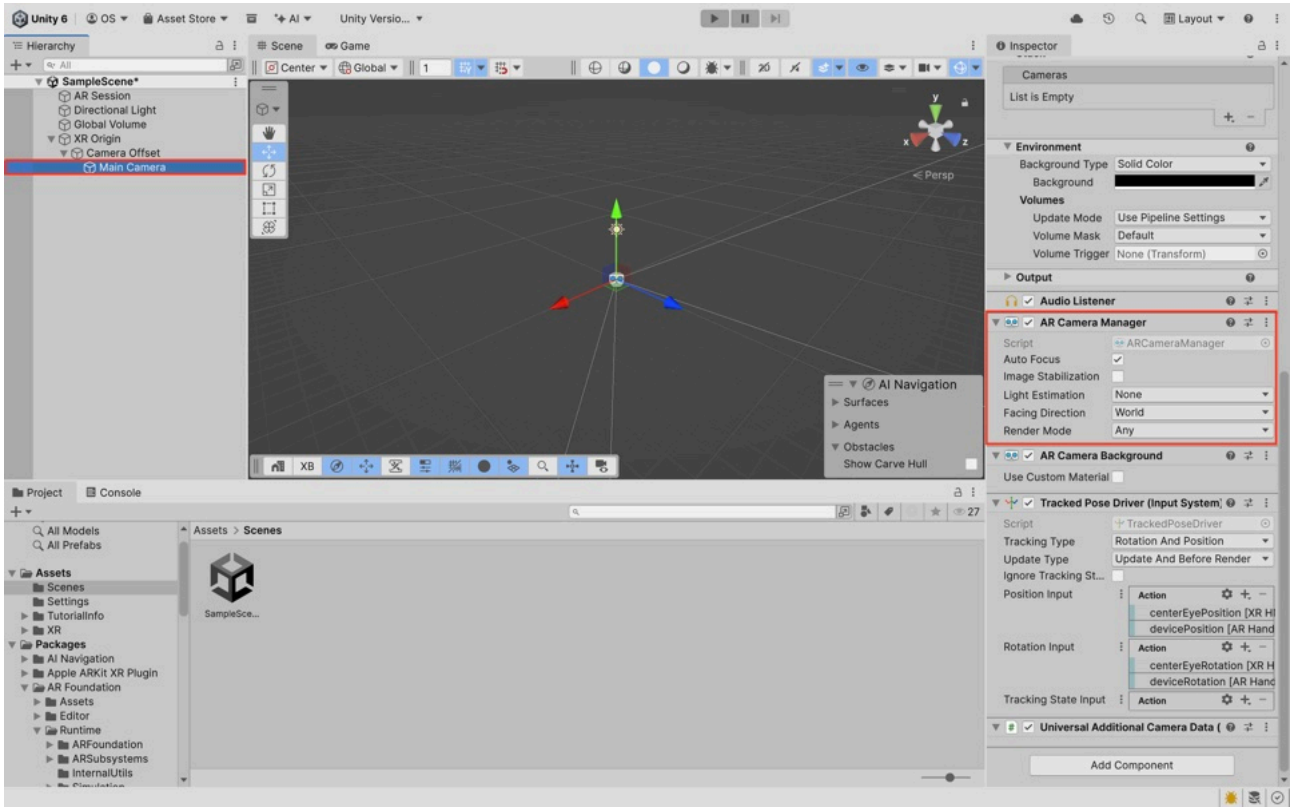


Рис. 2.17 – Ігровий об’єкт *Main Camera*

2.2 ПОРЯДОК ВИКОНАННЯ

2.2.1 Створення та налаштування *Unity*-проєкту

В *Unity Hub* необхідно обрати *Projects*, натиснути *New Project*, обрати шаблон проєкту *Universal 3D*, дати назву проєкту (напр., *AugmentedRealitywithUnity*) та натиснути *Create project* (рис. 2.6).

Наступним необхідно обрати платформу для якої буде розроблений *AR*-застосунок (рис. 2.8), встановити та налаштувати *AR Foundation* (рис. 2.9)*, SDK* доповненої реальності (рис. 2.10-2.11) та *XR Plug-in Management* (рис. 2.12-2.13). Пояснення та покрокові інструкції для цього етапу наведено у попередньому розділі 2.1.

1) Налаштування *AR*-застосунку на основі *iOS*-платформи.

Для налаштування *AR*-застосунку спочатку необхідно обрати *Edit-> Project Settings->Player*. Далі необхідно налаштувати параметри *Unity*-проєкту відповідно до Таблиці 2.1.

Таблиця 2.1. Параметри налаштування *Unity*-проєкту для розроблення *AR*-застосунку на основі *iOS*-платформи

Назва параметра	Значення
Company Name	Поле призначене для зазначення розробника *AR*-застосунку та використовується під час формування ідентифікаторів і параметрів збірки *Unity*-проєкту. Необхідно ввести назву, напр., *MyImmersiveCompany.* Не рекомендується використовувати значення за замовчуванням.
Bundle Identifier	Поле призначене для ідентифікації *AR*-застосунку. Необхідно ввести значення відповідно до шаблону *com.CompanyName.ProductName.*

	Напр., *com.MyImmersiveCompany.AugmentedRealitywithUnity*.
Camera Usage Description	Поле призначене для зазначення причини використання камери *AR*-застосунком, яка відображається користувачу під час запиту дозволу на доступ до камери *AR*-пристрою. Необхідно ввести значення, напр., *Augmented Reality requires the camera.*
Active Input Handing	Поле призначене для визначення системи введення в *Unity*-проєкті для оброблення дій користувача *AR*-застосунку (дотик, миша, клавіатура, контролери). Необхідно обрати значення *Both* (обидві системи введення одночасно). Після зміни цього параметра, *Unity* потребує перезапуску.
Requires ARKit support	Поле призначене для зазначення обов'язкової підтримки *ARKit* на пристрої. Необхідно активувати параметр.

2) Налаштування *AR*-застосунку на основі *Android*-платформи.

Для налаштування *AR*-застосунку спочатку необхідно обрати *Edit-> Project Settings->Player*. Далі необхідно налаштувати параметри *Unity*-проєкту відповідно до Таблиці 2.2.

Таблиця 2.2. Параметри налаштування *Unity*-проєкту для розроблення *AR*-застосунку на основі *Android*-платформи

Назва параметра	Значення
Company Name	Поле містить інформацію про розробника *AR*-застосунку та застосовується при формуванні системних ідентифікаторів і параметрів збірки *Unity*-проєкту. Необхідно ввести назву, напр., *MyImmersiveCompany.*
Graphics	Поле призначене для визначення інтерфейсу, через який *AR*-

API	застосунок взаємодіє з графічним процесором для відображення графіки. Можна визначити декілька *Graphics API* та їх порядок застосування для *Unity*-проєкту. Необхідно обрати *Vulkan* першим *Graphics API*, *OpenGLES3* – другим.
Package Name	Поле призначене для ідентифікації *AR*-застосунку. Необхідно ввести значення відповідно до шаблону *com.CompanyName.ProductName.* Напр., *com.MyImmersiveCompany.AugmentedRealitywithUnity*.
Minimum API Level	Поле призначене для визначення мінімального рівня *Android API*, необхідного для встановлення та роботи *AR*-застосунку. Мінімальний рівень *Android API* обирається відповідно до вимог *Google ARCore*. Неправильно встановлений рівень може спричинити помилки збірки або запуску. Необхідно обрати значення *Android 7.1 'Nougat'*(*API level 25)* або вище.
Scripting Backend	Поле призначене для визначення механізму компіляції та виконання *C#*-коду застосунку на цільовій платформі. Необхідно обрати *IL2CPP*.
Active Input Handing	Поле призначене для визначення системи введення в *Unity*-проєкті, яка забезпечує оброблення дотиків, натискань і роботи з контролерами в *AR*-застосунку. Необхідно обрати значення *Both*.
Target Architectures	Поле призначене для визначення процесорних архітектур, для яких виконується збірка *AR*-застосунку. Необхідно обрати *ARM64*.

3) Останнім необхідно налаштувати рендеринг *AR*-сцени (це необхідно для шаблону проєкту *Unity – Universal 3D*). Для цього необхідно обрати *Edit-> Project Settings-> Graphics*. Для параметра *Default Render Pipeline* обрати

значення *Mobile_RPAsset* (рис. 2.18) та підтвердити вибір. В теці *Assets* для *Unity*-проєкту обрати *Settings-> Mobile_Renderer*, натиснути *Add Renderer Feature* та обрати *AR Background Renderer Feature* (рис. 2.19).

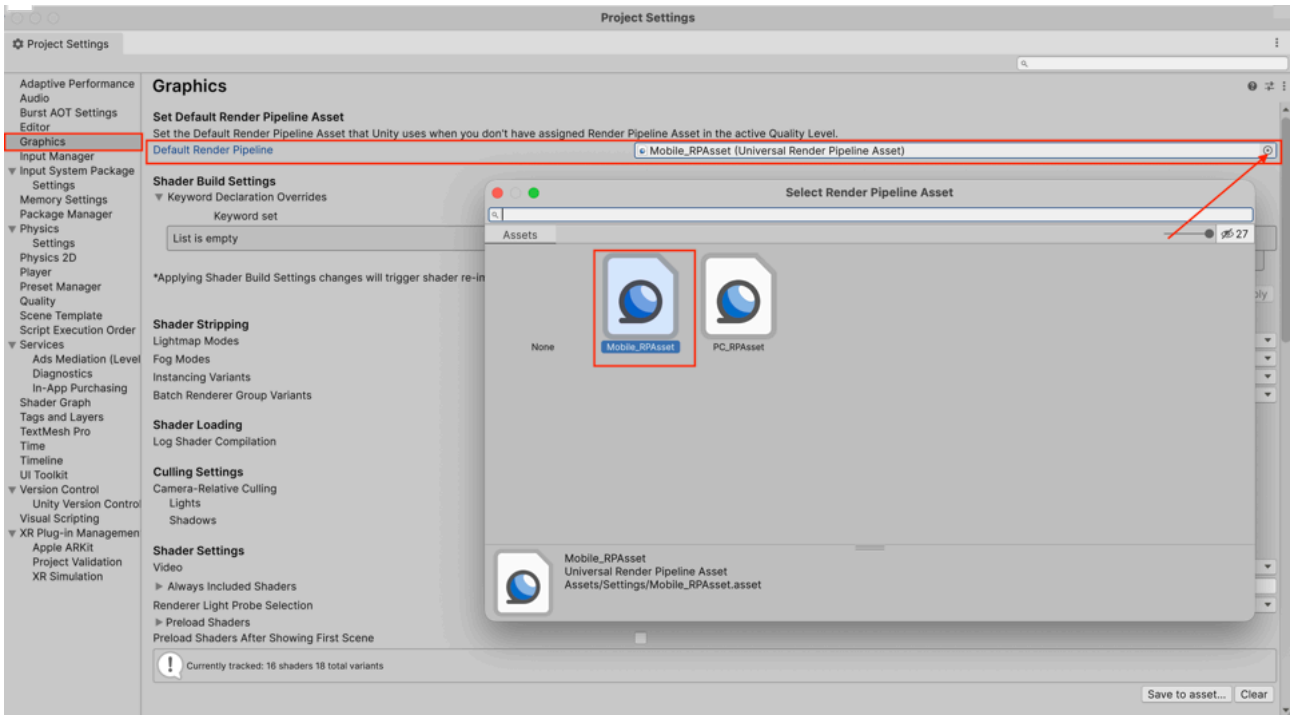


Рис. 2.18 – Вибір конфігурації рендеринг-конвеєра *AR*-сцени

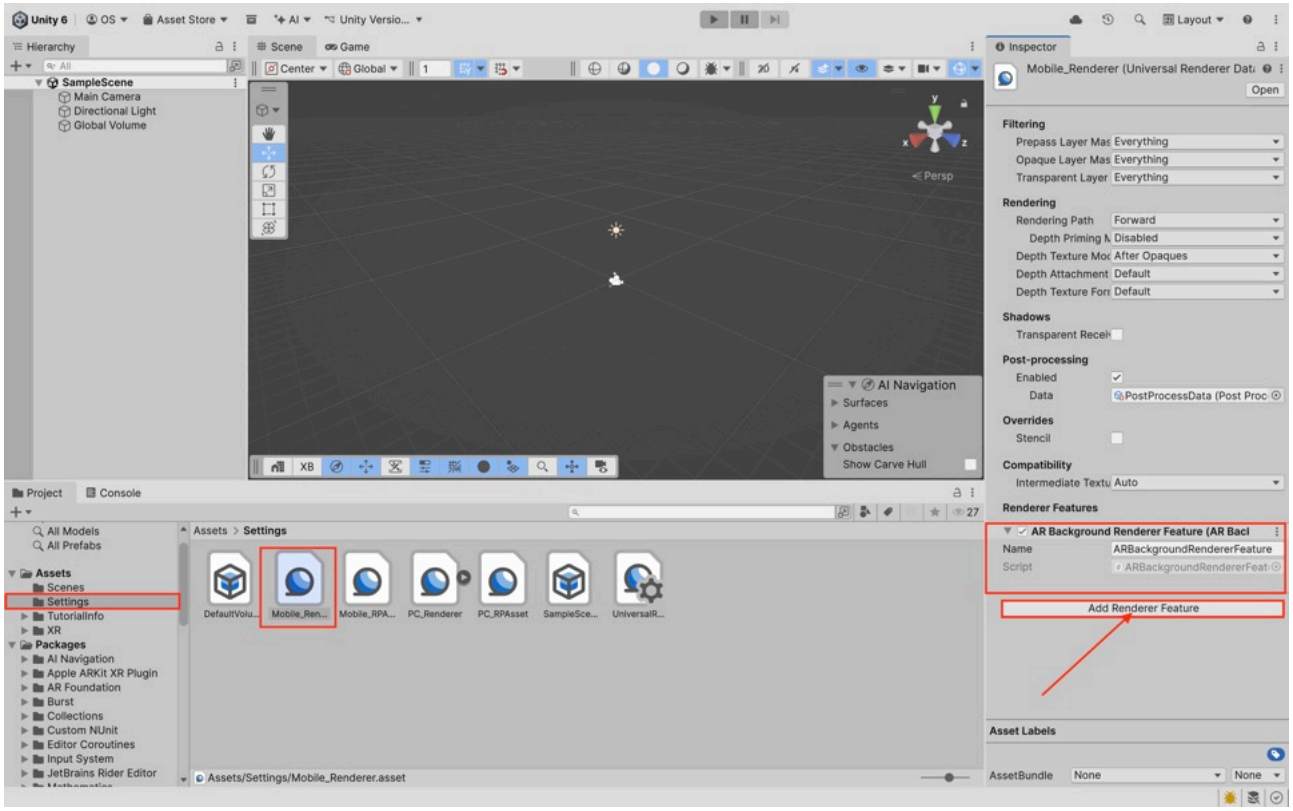


Рис. 2.19 – Налаштування *AR*-фону на основі зображення з камери пристрою

2.2.2 Імпортування та налаштування *AR*-об’єктів

Спочатку необхідно додати ігровий об’єкт *ARSession* (рис. 2.15). Пояснення та покрокові інструкції для цього етапу наведено у попередньому розділі 2.1. Ігровий об’єкт *Main Camera* необхідно видалити з *Unity*-проєкту. Наступним необхідно додати ігровий об’єкт *XR Origin* (рис. 2.16). Пояснення та покрокові інструкції для цього етапу наведено у попередньому розділі 2.1.

3D-об’єкт(и) з лабораторного практикуму №1 необхідно використати для розроблення сцени доповненої реальності. Для цього в *Blender*-проєкті необхідно обрати *File-> Export* та обрати формат зберігання *3D*-об’єкта, зокрема *.fbx*. Останнім необхідно вибрати каталог зберігання, дати назву файлу та натиснути *Export FBX*.

Для імпортування файлів в *Unity*-проєкт, зокрема *3D*-об’єктів необхідно обрати *Assets-> Import New Asset…* . Далі необхідно обрати файл, який необхідно імпортувати та натиснути *Import*. В результаті в теці *Assets Unity*-проєкту буде додано префаб (рис. 2.20). Префаб (*Prefab*) в *Unity* є шаблоном ігрового об’єкта, який зберігає його структуру, компоненти та налаштування.

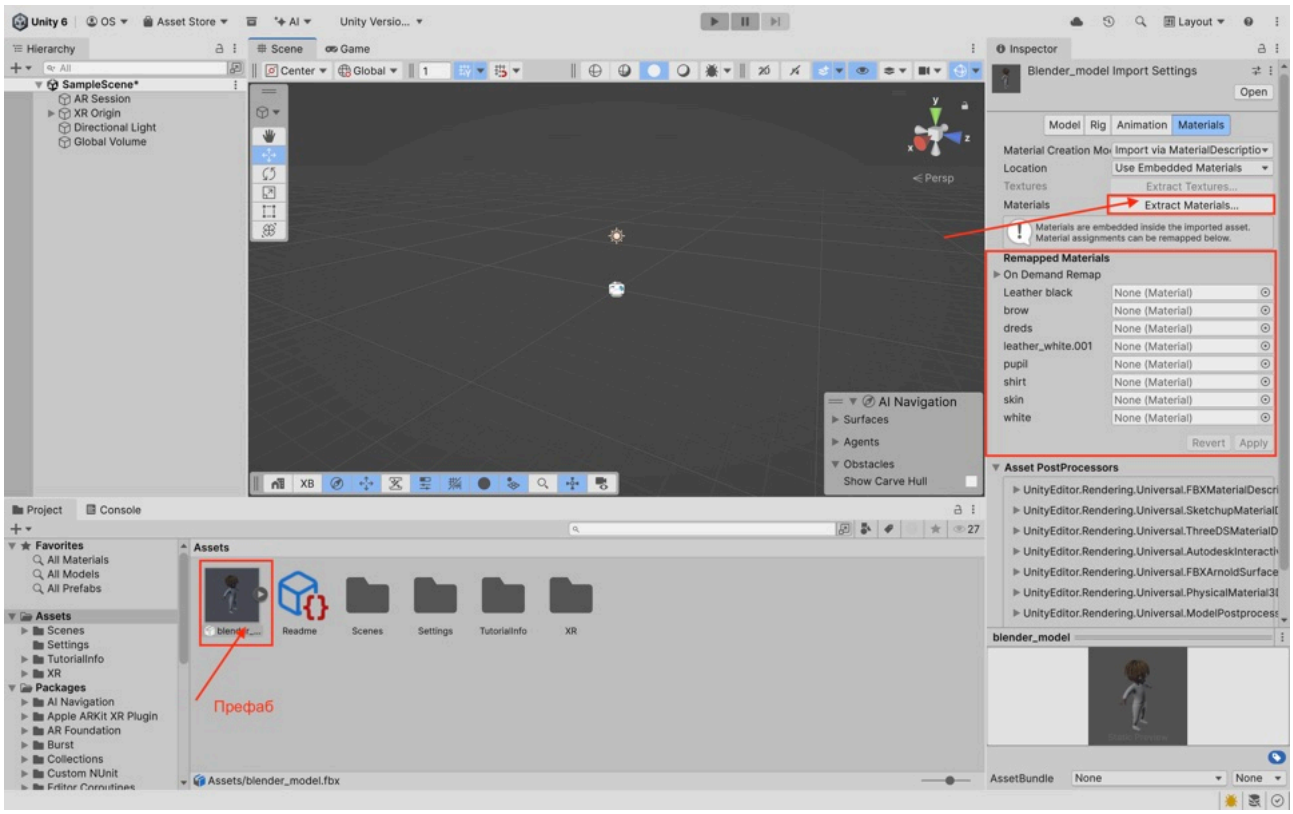


Рис. 2.20 – Імпортування анімованого *3D*-об’єкта

Для налаштування імпортованого *3D*-об’єкта з вбудованими матеріалами необхідно у вікні *Inspector* натиснути *Extract Materials… -> New Folder*, дати назву теці, напр., *Materials* та натиснути *Create*, а потім *Choose* (рис. 2.21). Вбудовані матеріали розміщуються в окремій теці (*Materials*) та призначаються для відповідних частин *3D*-об’єкта. Вилучені матеріали при необхідності можна редагувати в *Unity*.

Для створення нових матеріалів в *Unity* необхідно у вікні *Assets* натиснути правою кнопкою миші та обрати *Create-> Material* (рис. 2.22). Далі необхідно дати назву матеріалу та визначити параметри *Surface Option*, *Surface Inputs*, *Detail Inputs*, *Advanced Options*.

Група параметрів *Surface Options* визначають спосіб відображення поверхні об’єкта в *AR*-сцені та її взаємодію зі світлом і фоном. Параметр *Workflow Mode* визначає спосіб задання властивостей матеріалу поверхні об’єкта та характер відбивання світла від поверхні. Параметр *Surface Type* визначає тип матеріалу поверхні (непрозора/прозора). Для прозорих матеріалів поверхні налаштовується параметр *Blend Mode*, який визначає спосіб змішування з фоном. Параметр *Render Face* визначає сторони полігонів поверхонь об’єкта для відображення (зовнішня/внутрішня/обидві сторони). Параметр *Alpha Clipping* визначає застосування механізму видалення пікселів поверхні об’єкта з низьким значенням прозорості (значення альфа-каналу). Параметр *Receive Shadows* визначає застосування механізму відображення тіней від інших об’єктів у сцені.

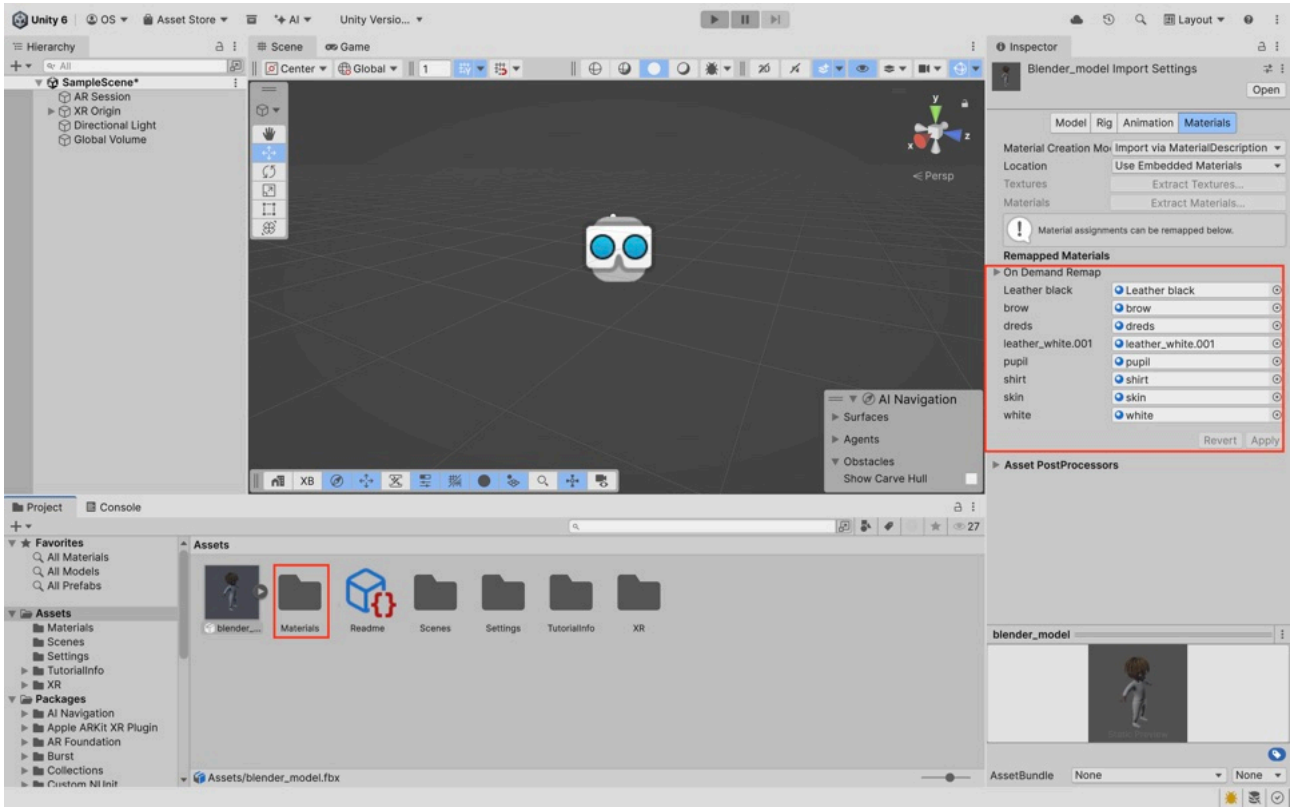


Рис. 2.21 – Налаштування *3D*-об’єкта з вбудованими матеріалами

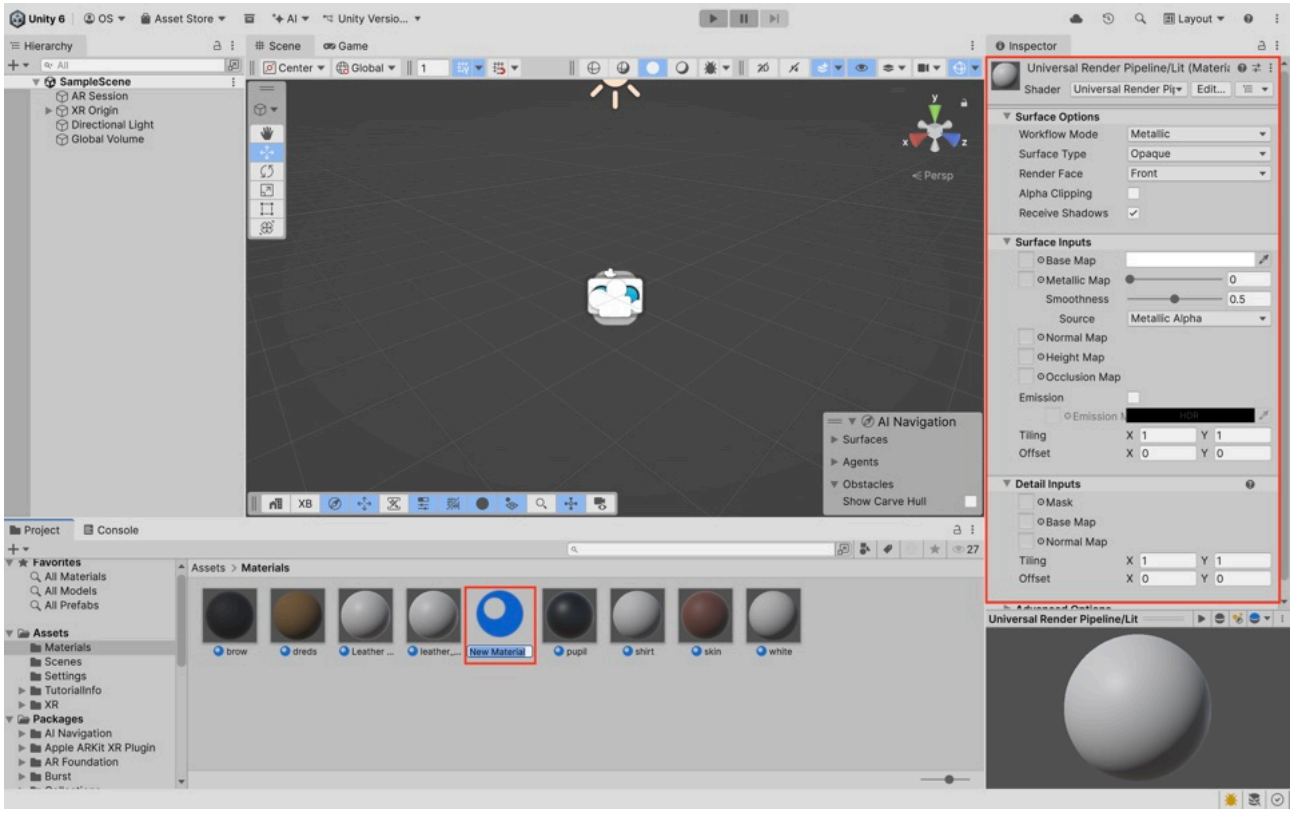


Рис. 2.22 – Створення нового матеріалу для *3D*-об’єкта в *Unity*

Група параметрів *Surface Inputs* визначають параметри матеріалу поверхні об’єкта. Параметр *Base Map* (або *Albedo*) визначає текстуру та колір поверхні. Параметр *Metallic Map* визначає відбиття світла від поверхні (якщо вона металева/ неметалева). Параметр *Smoothness* визначає ступінь згладжування поверхні. Параметр *Normal Map* визначає текстуру для оптичного рельєфу на поверхні без зміни геометрії об’єкта. Параметр *Height Map* визначає текстуру для оптичного рельєфу на поверхні зі змінами геометрії об’єкта. Параметр *Occlusion Map* визначає області поверхні об’єкта, де світло поглинається або не відбивається (темні області поверхні). Параметр *Emission* визначає властивість поверхні випромінювати світло. Параметр *Tiling* визначає повторюваність текстури по осях, параметр *Offset* – зсув текстури.

Група параметрів *Detail Inputs* визначають параметри матеріалу поверхні об’єкта, що накладаються на *Surface Inputs*. Параметр *Mask* визначає області поверхні об’єкта, де застосовуються дрібні деталі *Base Map* (група *Detail Inputs*). Параметр *Base Map* визначає текстуру дрібних деталей, яка накладається на *Base Map* (або *Albedo*). Параметр *Normal Map* визначає текстуру дрібних деталей для додаткового оптичного рельєфу, який накладається на основний *Normal Map* (група *Surface Inputs*). Параметри *Tiling* та *Offset* визначають повторюваність та зсув текстури дрібних деталей по осях.

Імпортування текстури *3D*-об’єкта для налаштування створеного матеріалу в *Unity* (рис. 2.23) виконується за принципом імпортування будь-якого іншого файлу в *Unity*-проєкт, зокрема необхідно обрати *Assets-> Import New Asset…* . Далі необхідно обрати файл, який необхідно імпортувати та натиснути *Import*.

Наступним необхідно розмістити імпортований *3D*-об’єкт на *Unity*-сцену, зокрема перетягнути файл *3D*-об’єкта у вікно *Hierarchy*. Далі необхідно вилучити ігровий об’єкт з префабу для налаштування в *Unity*. Для цього натиснути правою кнопкою миші на *3D*-об’єкт у вікні *Hierarchy* та обрати *Prefab-> Unpack Completely* (2.24). Для налаштування ігрового об’єкта

необхідно визначити його позицію, орієнтацію та масштаб за допомогою компоненти *Transform* у вікні *Inspector* (2.24).

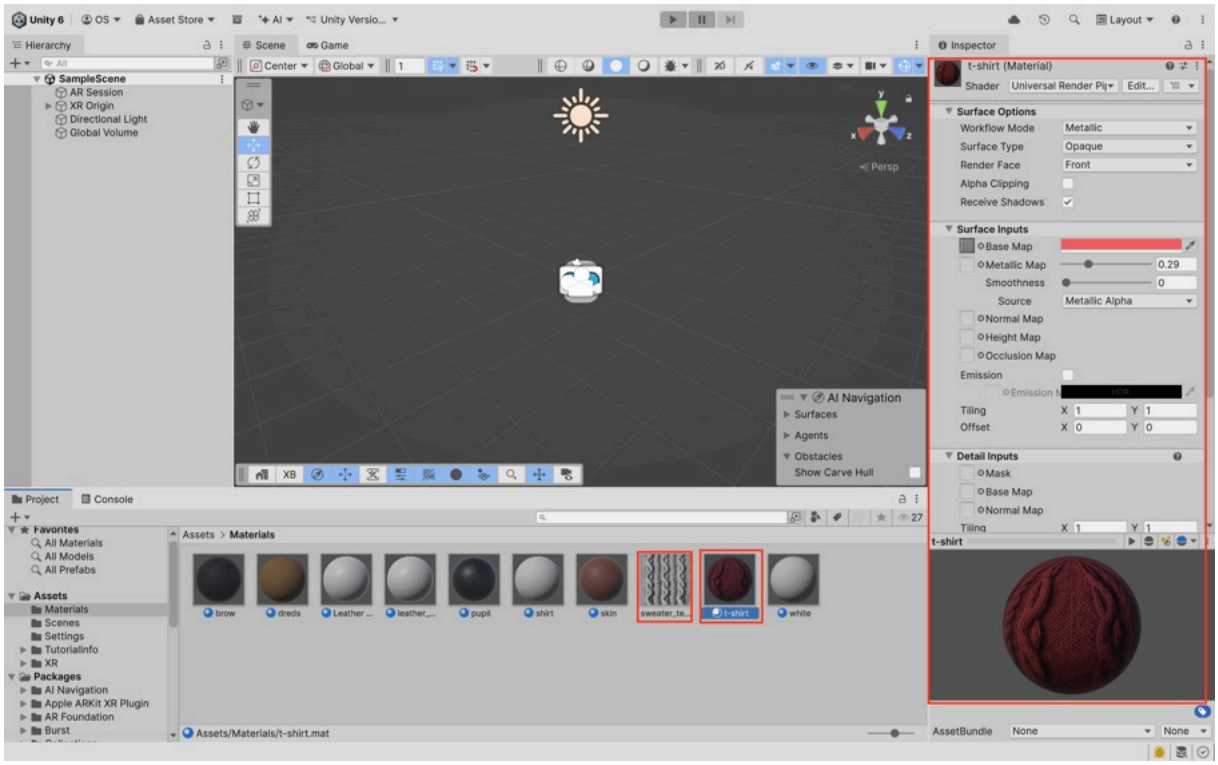


Рис. 2.23 – Налаштування нового матеріалу для *3D*-об’єкта в *Unity*

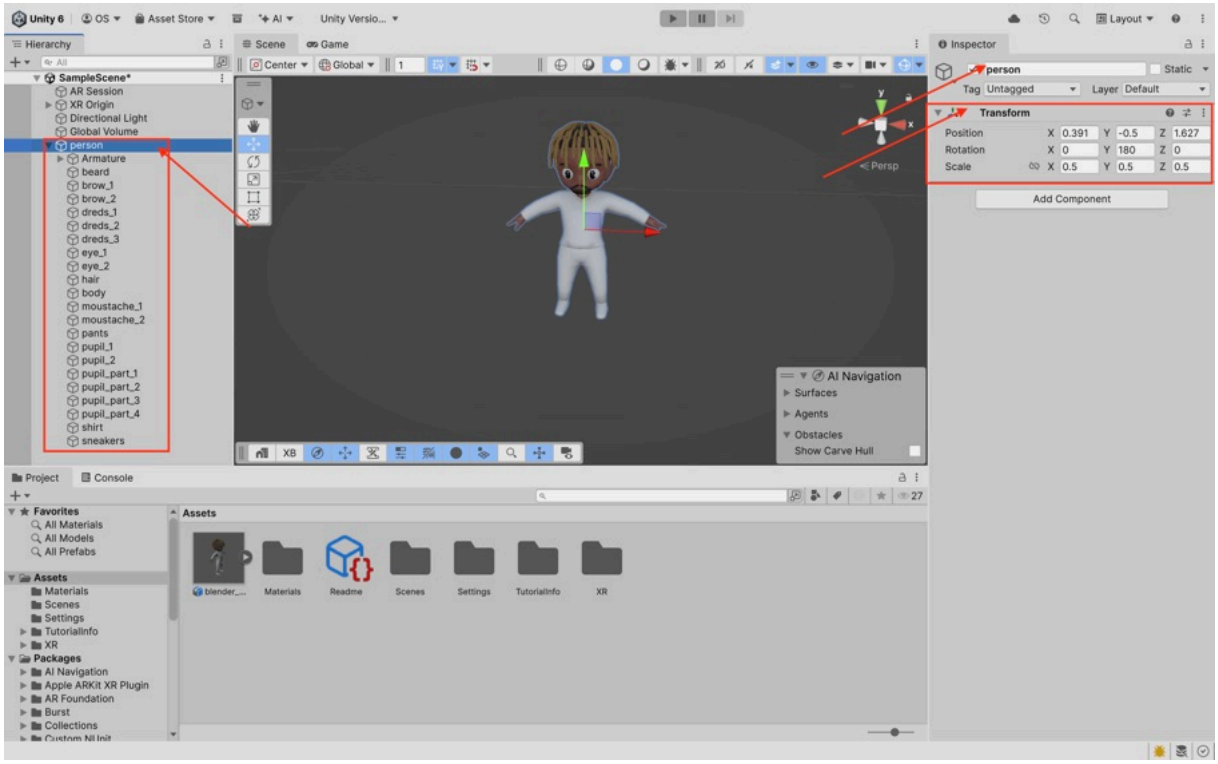


Рис. 2.24 – Вилучення з префабу ігрового об’єкта та налаштування компоненти *Transform*

Для застосування створених матеріалів до ігрового об’єкта необхідно перетягнути файл матеріалу на сам об’єкт або його частину у вікні *Hierarchy*, або безпосередньо у вікні сцени (рис. 2.25). При необхідності створені матеріали потрібно застосувати до кожної частини об’єкта окремо.

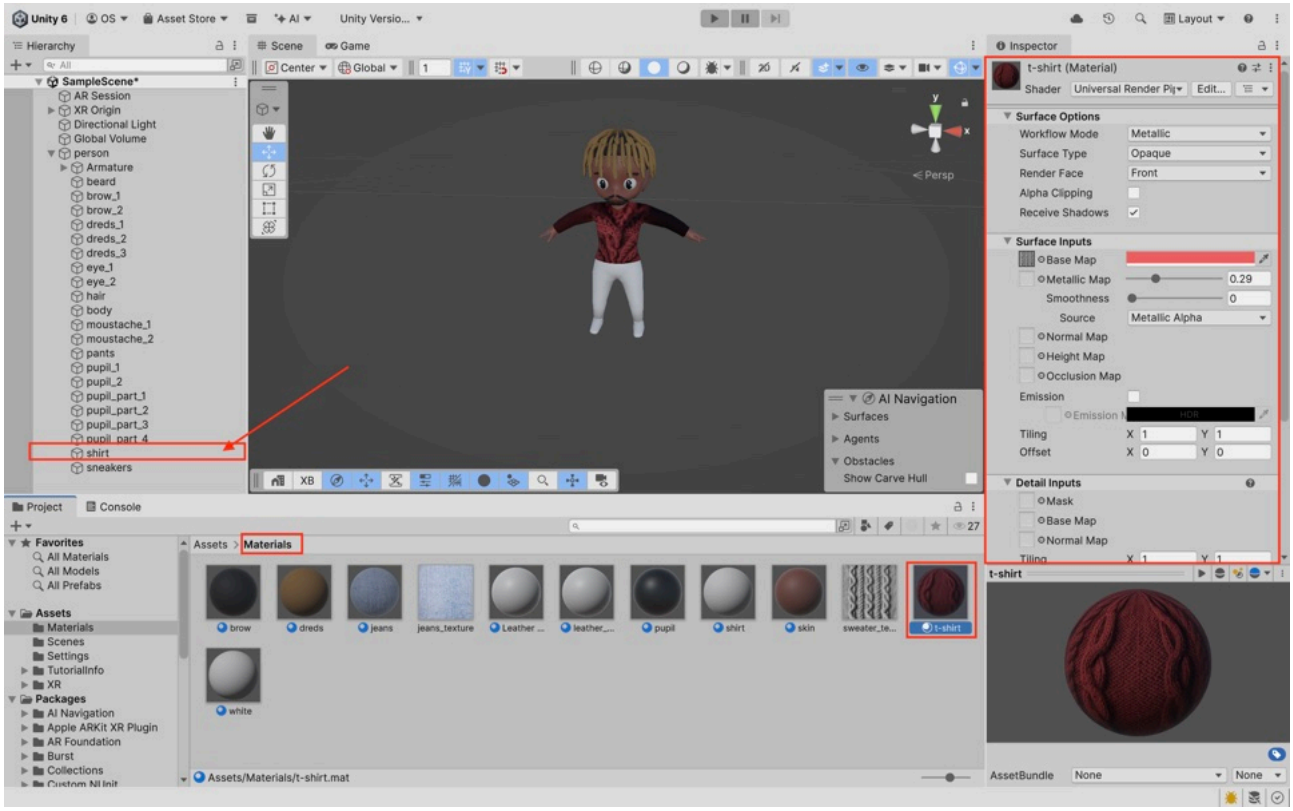


Рис. 2.25 – Застосування створеного матеріалу до ігрового об’єкта

Останнім необхідно анімувати *3D*-об’єкт. Анімацію з лабораторного практикуму №1 необхідно використати для розроблення сцени доповненої реальності. Для цього файл анімаційного кліпу необхідно скопіювати з префабу (*Ctrl+C*) та вставити як окремий файл у теку *Assets* (*Ctrl+V*) для налаштування в *Unity*.

Для налаштування анімації в *Unity* спочатку необхідно додати до ігрового об’єкта компоненту *Animator* (рис. 2.26). Для цього у вікні *Inspector* натиснути *Add Component* та обрати *Animation-> Animator*.

Наступним необхідно створити файл *Animator Controller*. Для цього необхідно у вікні *Assets* натиснути правою кнопкою миші, обрати *Create-> Animation-> Animator Controller* та дати назву створеному файлу. Створений

файл необхідно вибрати для параметра *Controller* компоненти *Animator 3D*-об’єкта (рис. 2.27).

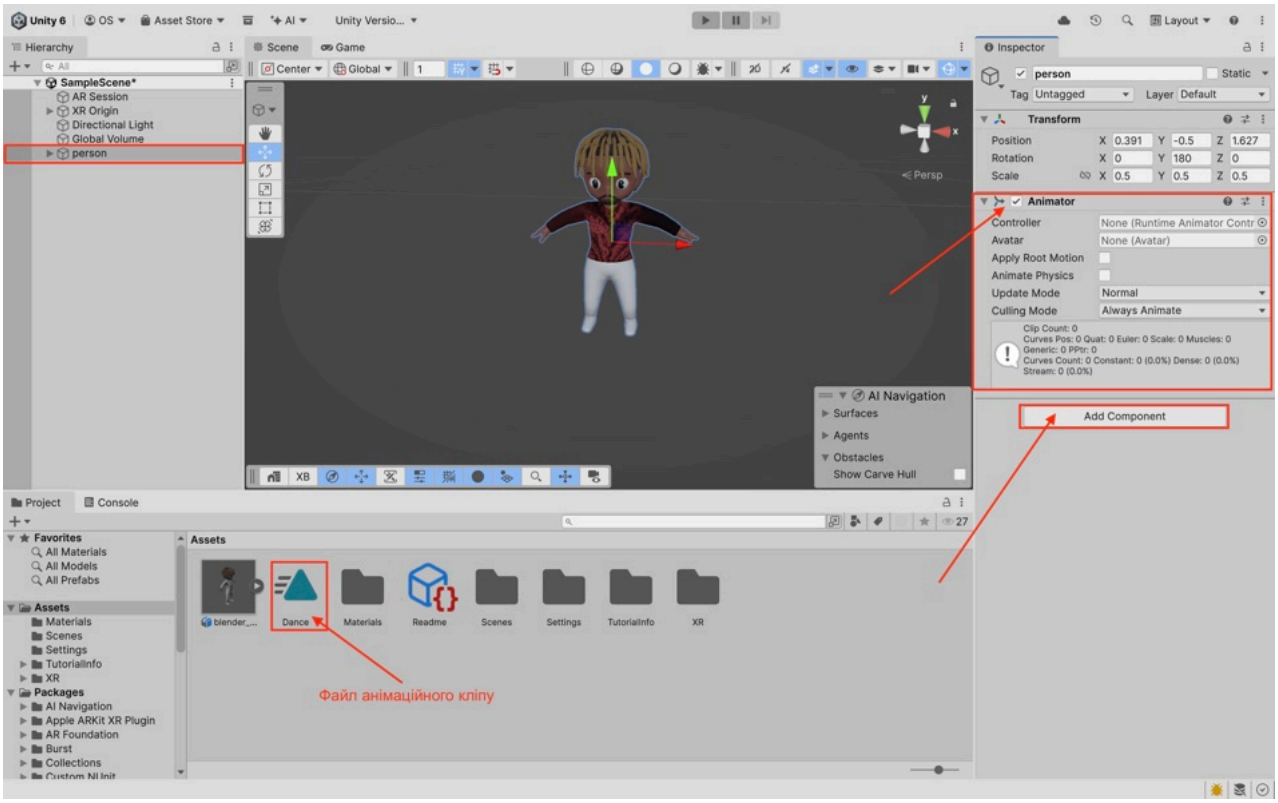


Рис. 2.26 – Створення компоненти *Animator*

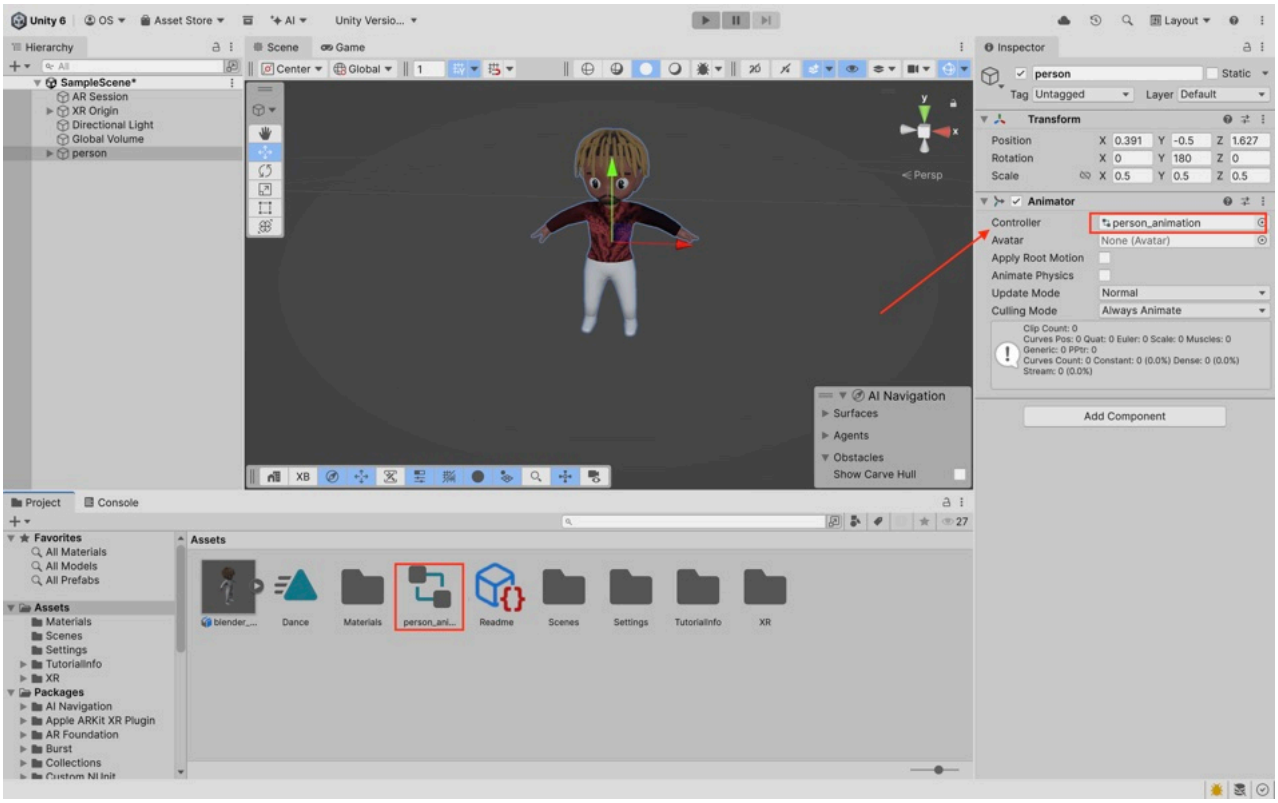


Рис. 2.27 – Налаштування компоненти *Animator*

Останнім необхідно відкрити вікно *Animator*, зокрема обрати *Window-> Animation-> Animator* та відкрити (натиснути двічі) файл *Animator Controller* (рис. 2.28). Файл(и) анімаційного кліпу необхідно перетягнути у вікно *Animator*, налаштувати за допомогою функції *Make Transition* та інших параметри (швидкість відтворення, повторюваність тощо).

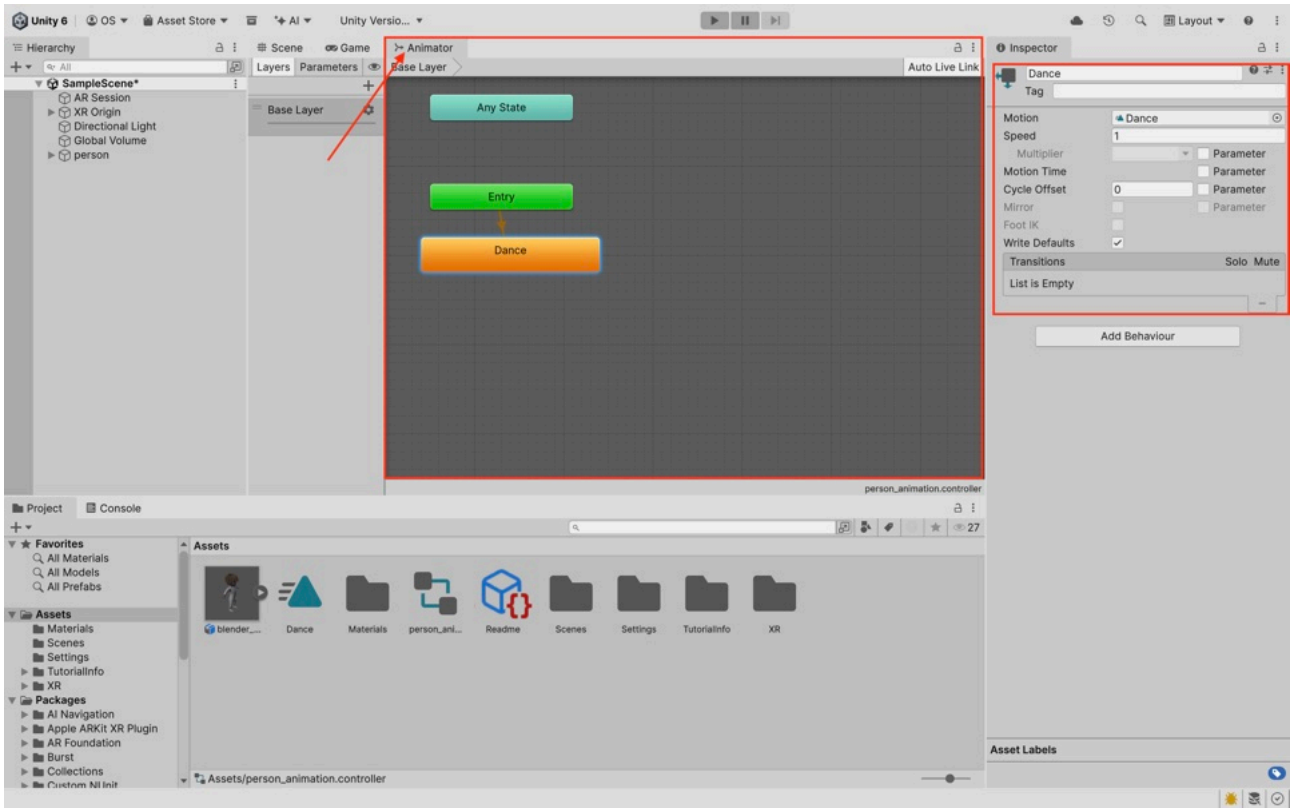


Рис. 2.28 – Налаштування анімації *3D*-об’єкта у вікні *Animator*

Для налаштованого *3D*-об’єкта буде створено префаб на цьому етапі, щоб використовувати його у досвіді просторової доповненої реальності (розділ 2.2.4).

2.2.3 Проєктування маркерної доповненої реальності

У цьому розділі розглядається проєктування маркерної доповненої реальності, що ґрунтується на технології розпізнавання зображення (маркера) у фізичному середовищі та подальшому накладанні цифрового об’єкта на зображення (маркер). Налаштування *3D*-об’єкта розглядається у попередньому

розділі 2.2.2. Для *3D*-об’єкта реалізується звуковий супровід під час відтворення анімації.

Для реалізації звукового супроводу необхідно для *3D*-об’єкта створити скрипт-компоненту. Спочатку у вікні *Inspector* необхідно натиснути *Add Component-> New script*, назвати файл та натиснути *Create and Add*. У створеному файлі необхідно написати програмний код (мова *C#*):

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

// На об’єкті, до якого додається скрипт, обов’язково є компонента AudioSource. Якщо ні,
Unity автоматично додасть його при додаванні скрипта
[RequireComponent(typeof(AudioSource))]

public class EventSound : MonoBehaviour
{

// Змінна для збереження компонента AudioSource об’єкта
    AudioSource mySource;
// Аудіокліп, який можна призначити у вікні Inspector
    public AudioClip myClip;


    void Start()
    {
// Отримуємо компонент AudioSource з об'єкта і зберігаємо в mySourse
        mySourse = GetComponent<AudioSource>();
    }

// Метод, який можна викликати через кнопки, тригери, скрипти або анімаційні події
    public void VoidEventSound()
    {
// Відтворює аудіокліп один раз, не зупиняючи інші звуки
        mySourse.PlayOneShot(myClip);
    }
}
```

Наступним необхідно імпортувати аудіо файл, який буде відтворюватись під час анімації та додати його до компонент *Event Sound* та *Audio Source* у вікні *Inspector* (рис. 2.29).

Останнім необхідно створити точки *Event* (рис. 2.30), які будуть реалізовувати метод *EventSound* та відтворювати звук під час анімації *3D*-об’єкта. Для цього необхідно обрати *Window-> Animation-> Animation*, на *Timeline* створити точки *Event* та обрати створений метод (рис. 2.30).

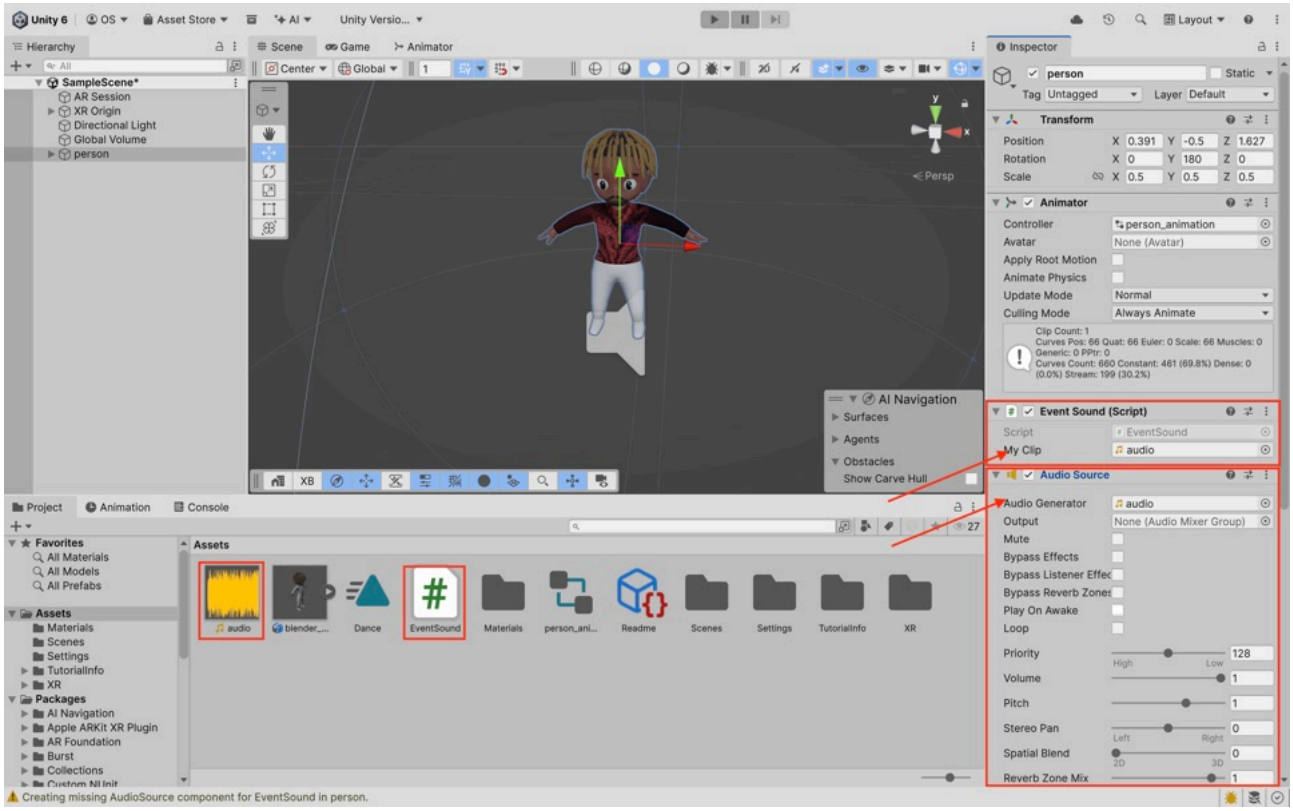


Рис. 2.29 – Налаштування звукового супроводу для *3D*-об’єкта

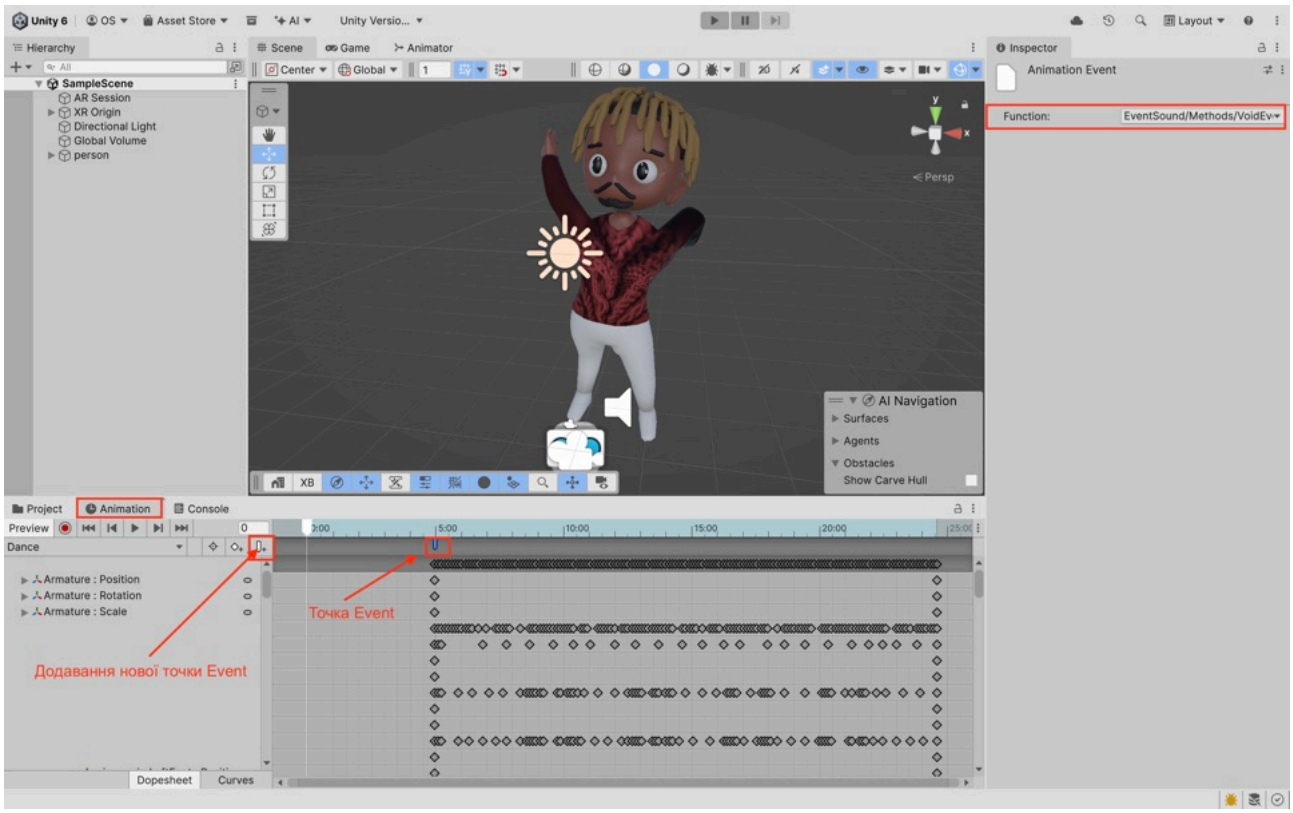


Рис. 2.30 – Створення точок *Event*

Для проєктування маркерної доповненої реальності у вікні *Inspector* для ігрового об’єкта *XR Origin* спочатку необхідно додати компоненту *AR Tracked Image Manager*. Наступним необхідно створити бібліотеку для зберігання маркерів. Для цього у вікні *Assets* необхідно натиснути правою кнопкою миші, обрати *Create-> XR-> Reference Image Library* та дати назву бібліотеці. Далі необхідно імпортувати зображення маркеру доповненої реальності (рис. 2.31), додати до бібліотеки та налаштувати (рис. 2.32).

Рис. 2.31 – Зображення маркеру

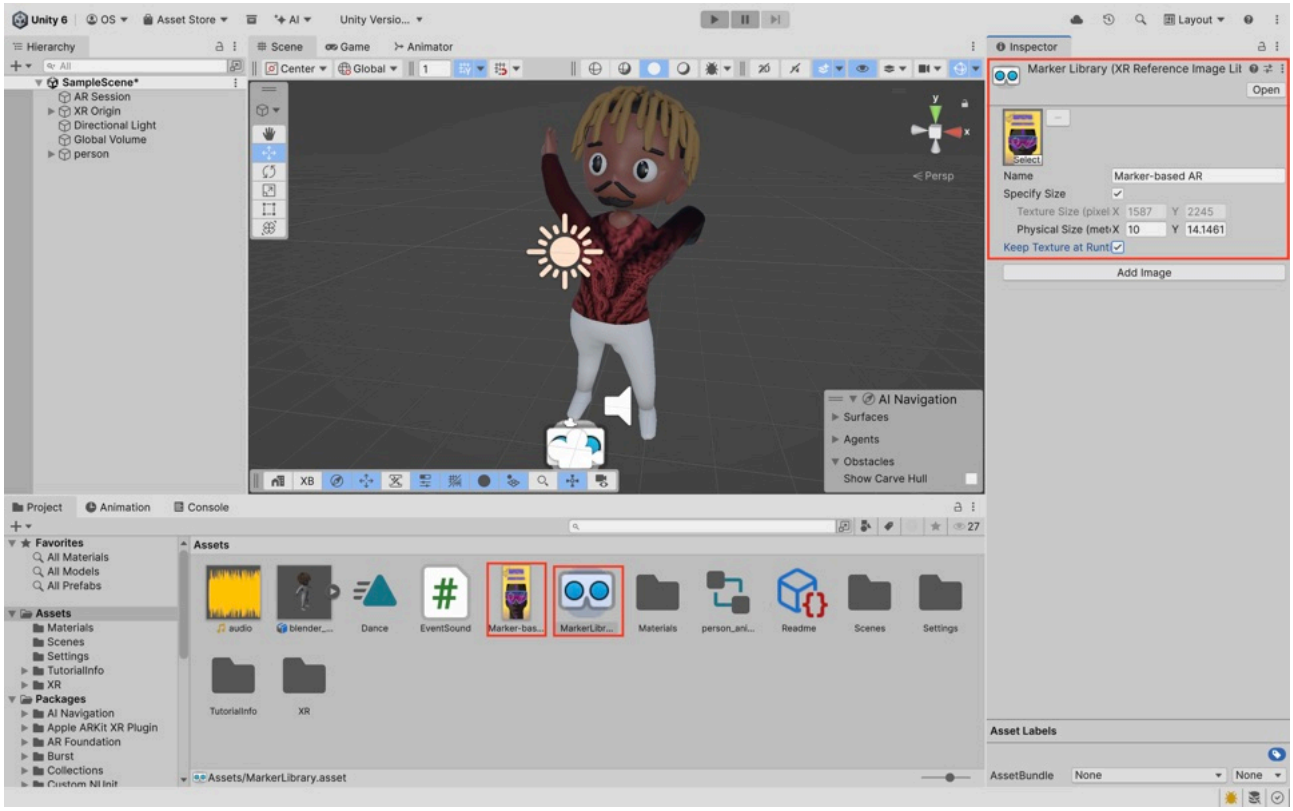


Рис. 2.32 – Налаштування бібліотеки маркерів

Останнім необхідно налаштувати параметри компоненти *AR Tracked Image Manager y* вікні *Inspector*, зокрема необхідно додати бібліотеку маркерів, створити та додати префаб *3D*-об'єкта (рис. 2.33). Після створення префабу, ігровий об'єкт необхідно видалити з вікна *Hierarchy*.

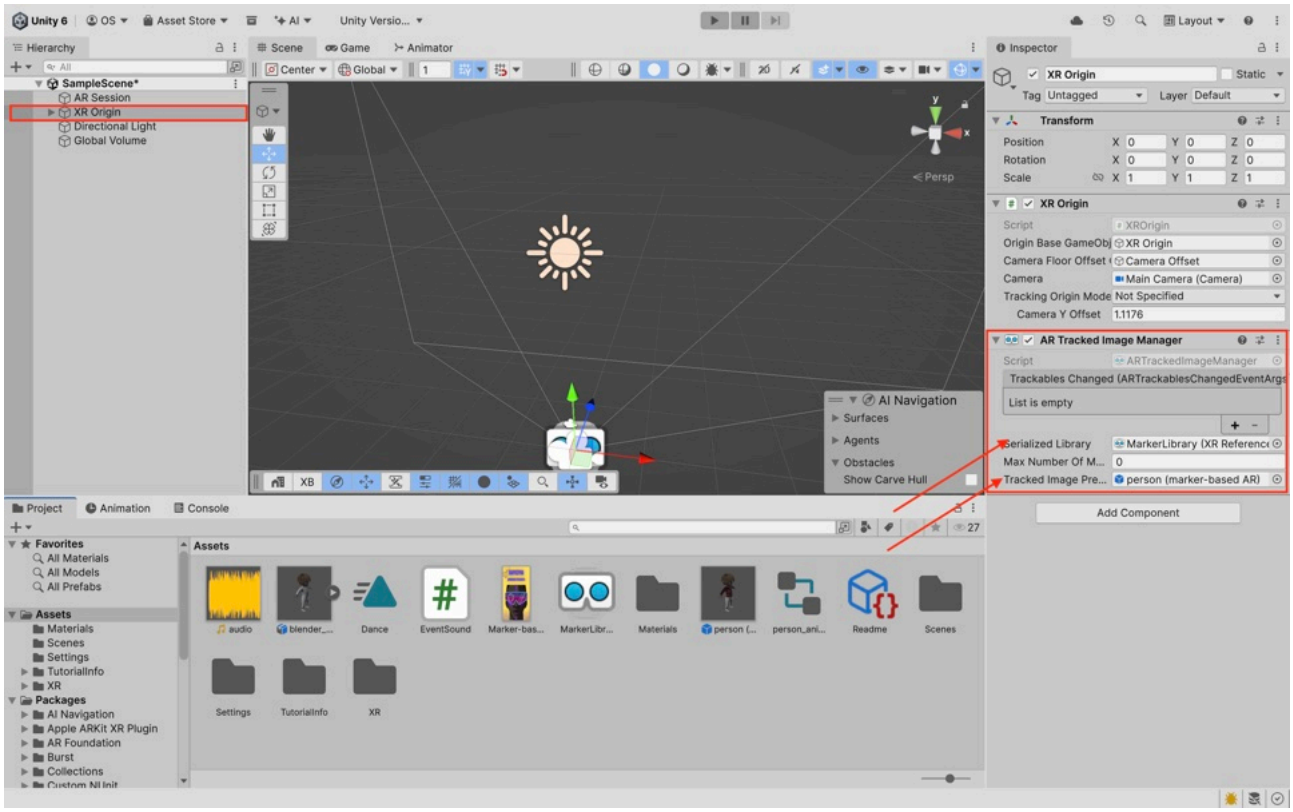
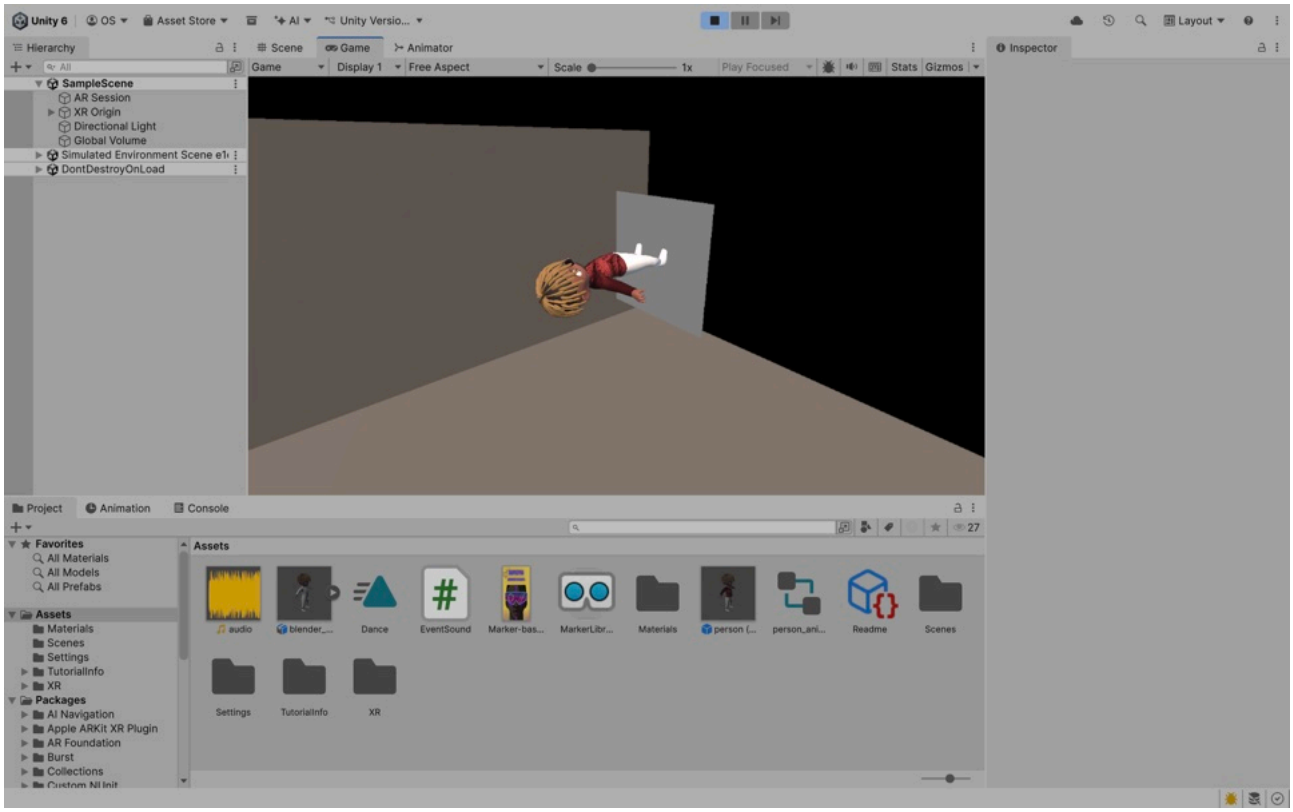
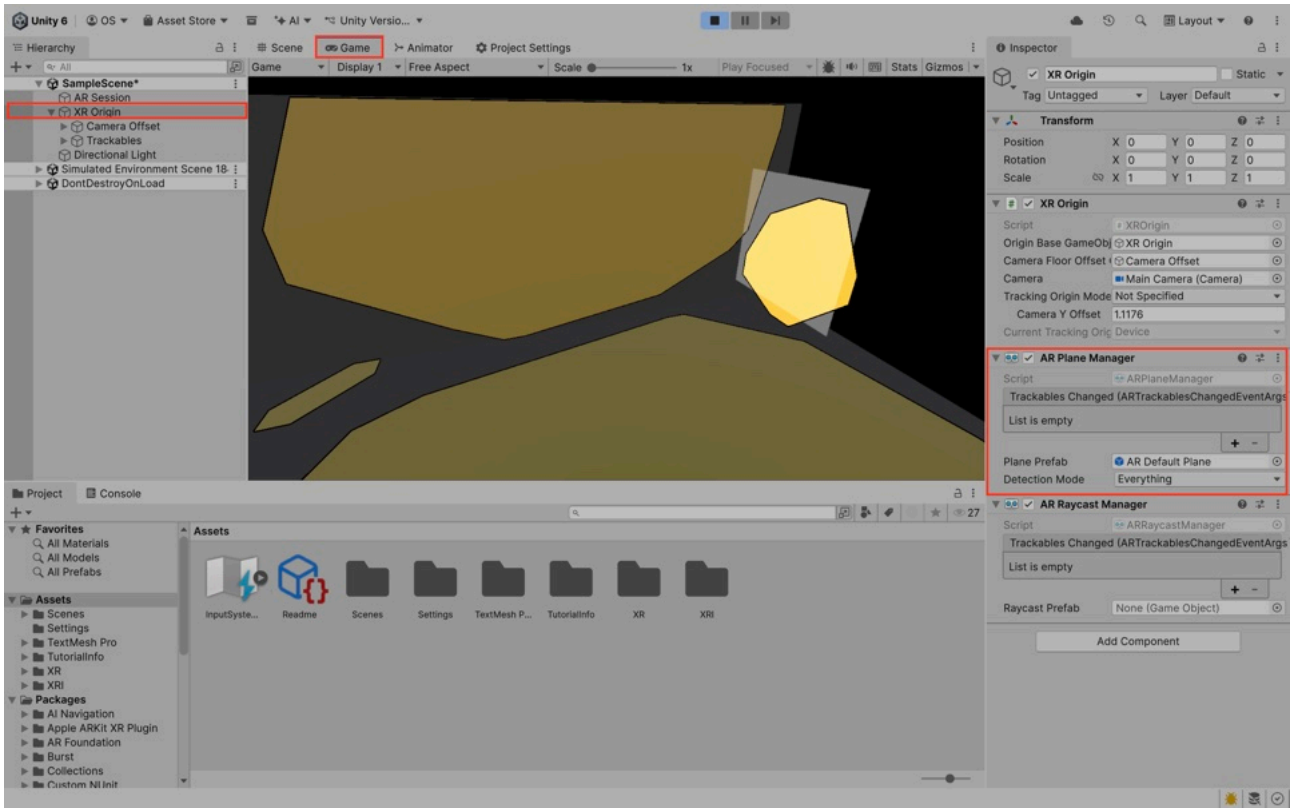


Рис. 2.33 – Налаштування компоненти *AR Tracked Image Manager*

У режимі *Play Mode* тестування маркерної доповненої реальності в *Unity* здійснюється за допомогою *XR Simulation* (рис. 2.34). *XR Simulation* є інструментом *Unity* для тестування *XR*-сцен у редакторі без використання реального *XR*-пристрою. Для підключення *XR Simulation* необхідно обрати *Edit-> Project Settings-> XR Plug-in Management* та натиснути на *XR Simulation*.



а)



б)

Рис. 2.34 – *XR Simulation*: а) виявлення зображення; б) виявлення поверхонь

XR Simulation імітує сканування фізичного середовища та створює віртуальні поверхні для розміщення *XR*-об’єктів (параметри *Environment Scan*), створює візуальні підказки про виявлені поверхні (параметри *Plane Discovery*) при використанні в *Unity*-проєкті компоненти *AR Plane Manager*. Також *XR Simulation* імітує розпізнавання зображення (параметри *Tracked Image Discovery*) для маркерної доповненої реальності при використанні в *Unity*-проєкті компоненти *AR Tracked Image Manager*. Параметри *Environment Probe Discovery* відповідають за імітацію виявлення інформації про освітлення навколишнього середовища для визначення освітлення *XR*-об’єктів при використанні в *Unity*-проєкті компоненти *AR Environment Probe Manager*. Параметри *Anchor Discovery* в *XR Simulation* імітують створення та виявлення якорів для фіксації *XR*-об’єктів у фізичному середовищі при використанні в *Unity*-проєкті компоненти *AR Anchor Manager*. Параметри *Bounding Box Discovery* імітують визначення обмежувальних областей для цифрових або фізичних об’єктів для коректної взаємодії *XR*-об’єктів з фізичним середовищем.

Для завершення розроблення застосунку маркерної доповненої реальності, на цьому етапі необхідно перейти до розділу 2.2.5 Збирання та розгортання *AR*-застосунку.

2.2.4 Проєктування просторової доповненої реальності

У цьому розділі розглядається проєктування просторової доповненої реальності, що ґрунтується на технології розпізнавання горизонтальних та вертикальних поверхонь у фізичному середовищі та подальшому розміщенні цифрового об’єкта на цих поверхнях шляхом дотику користувача по екрану *AR*-пристрою. Налаштування *3D*-об’єкта розглядається у розділі 2.2.2.

Спочатку до ігрового об’єкта *XR Origin* необхідно додати компоненту *AR Plane Manager* для відстеження горизонтальних та вертикальних поверхонь. Для цього у вікні *Inspector* необхідно обрати *Add Component-> XR-> AR Foundation-> AR Plane Manager* (рис. 2.35). Для параметра *Detection Mode* необхідно обрати значення *Everything*, щоб відстежувати одночасно

горизонтальні та вертикальні поверхні. Далі необхідно у вікні *Hierarchy* натиснути правою кнопкою миші та обрати *XR-> AR Default Plane.* Для створеного ігрового об'єкта необхідно створити префаб, обрати створений префаб для параметра *Plane Prefab* компоненти *AR Plane Manager* та видалити ігровий об'єкт з вікна *Hierarchy* (рис. 2.35). Наступним необхідно додати компоненту *AR Raycast Manager* для реалізації взаємодії користувача з *AR*-досвідом шляхом визначення точок перетину між *Raycast*-променем (дотик користувача по екрану) та поверхнями фізичного середовища (рис. 2.35).

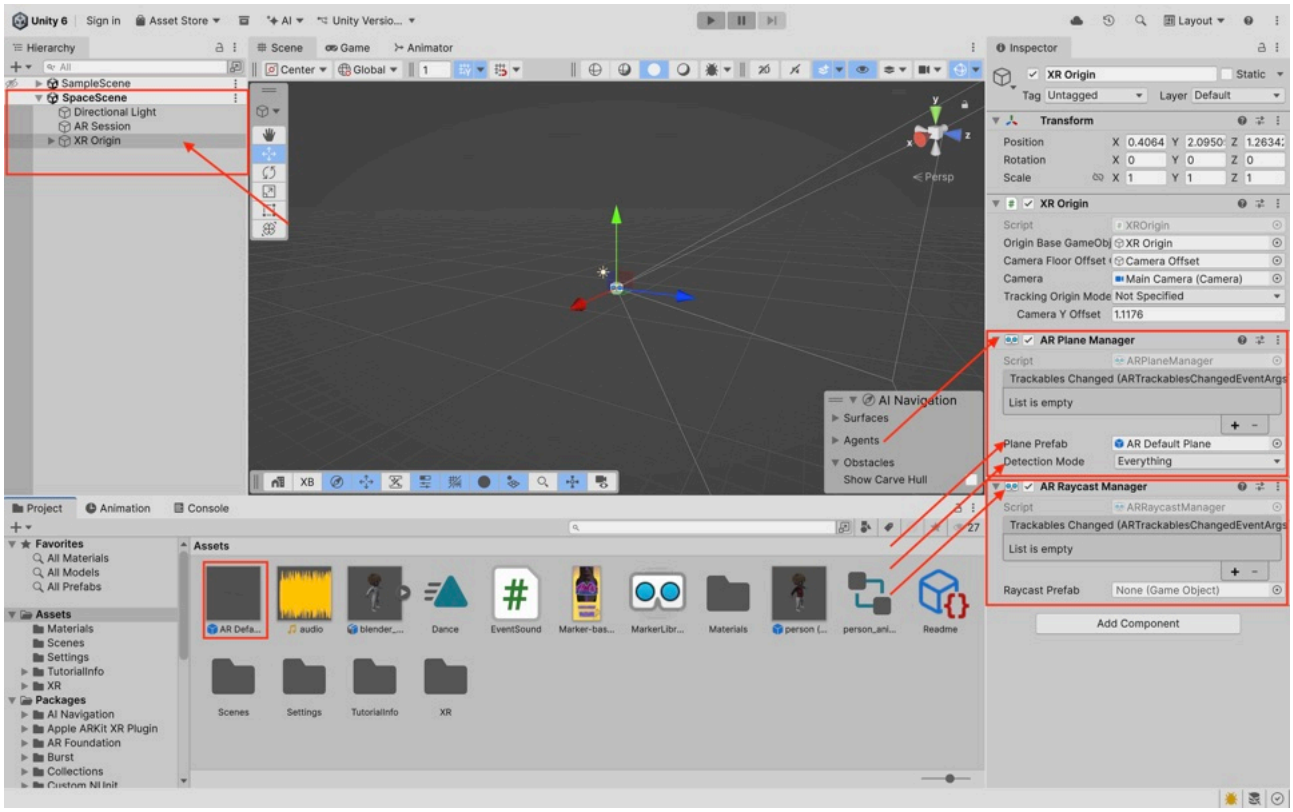


Рис. 2.35 – Налаштування компонент *AR Plane Manager* та *AR Raycast Manager*

Наступним необхідно додати систему опису та оброблення дій користувача *Input Actions* для реалізації взаємодії користувача з *AR*-досвідом. Відповідно до ідеї проєктування *AR*-застосунку, користувач може переміщати та розташовувати анімований *3D*-об'єкт у фізичному середовищі. Спочатку необхідно у вікні *Assets* натиснути правою кнопкою миші, обрати *Create-> Input Actions* та дати назву створеному файлу, напр., *TouchControls*. Далі необхідно відкрити створений файл *TouchControls*, створити та назвати групу

дій (*Action Maps),* напр., *control*, дію (*Actions)* напр., *touch* та налаштувати параметри оброблення дії (рис. 2.36).

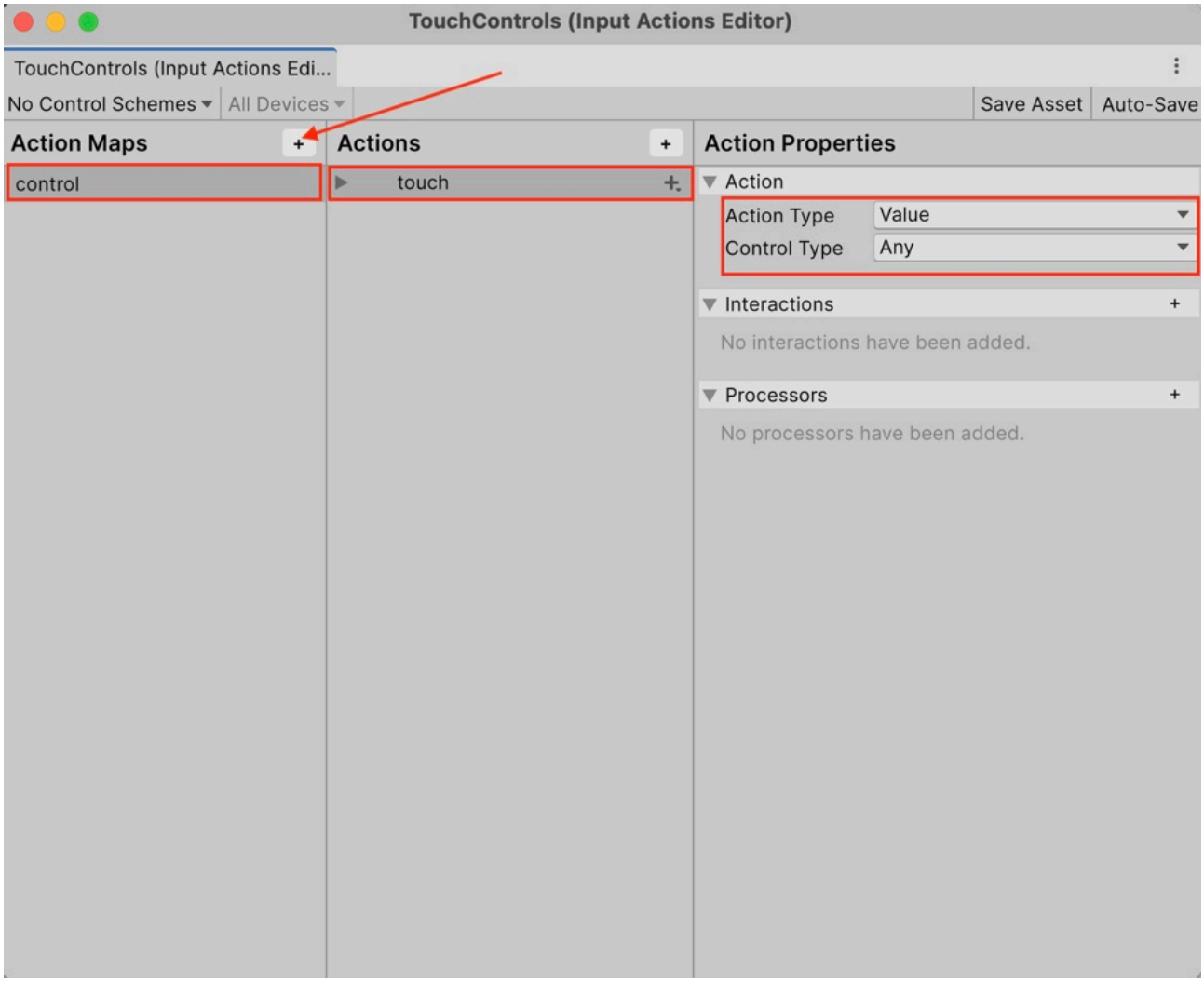


Рис. 2.36 – Додавання та налаштування системи *Input Actions*

Далі створену дію дотику необхідно налаштувати (*Binding)* та визначити конкретний елемент введення (напр., кнопка, жест, дотик, контролер). Відповідно до ідеї проєктування, необхідно обрати *Pointer-> Press* (рис. 2.37) та натиснути *Save Asset*.

Далі необхідно на основі файлу системи *Input Actions* у вікні *Inspector* згенерувати *C#*-скрипт (рис. 2.38).

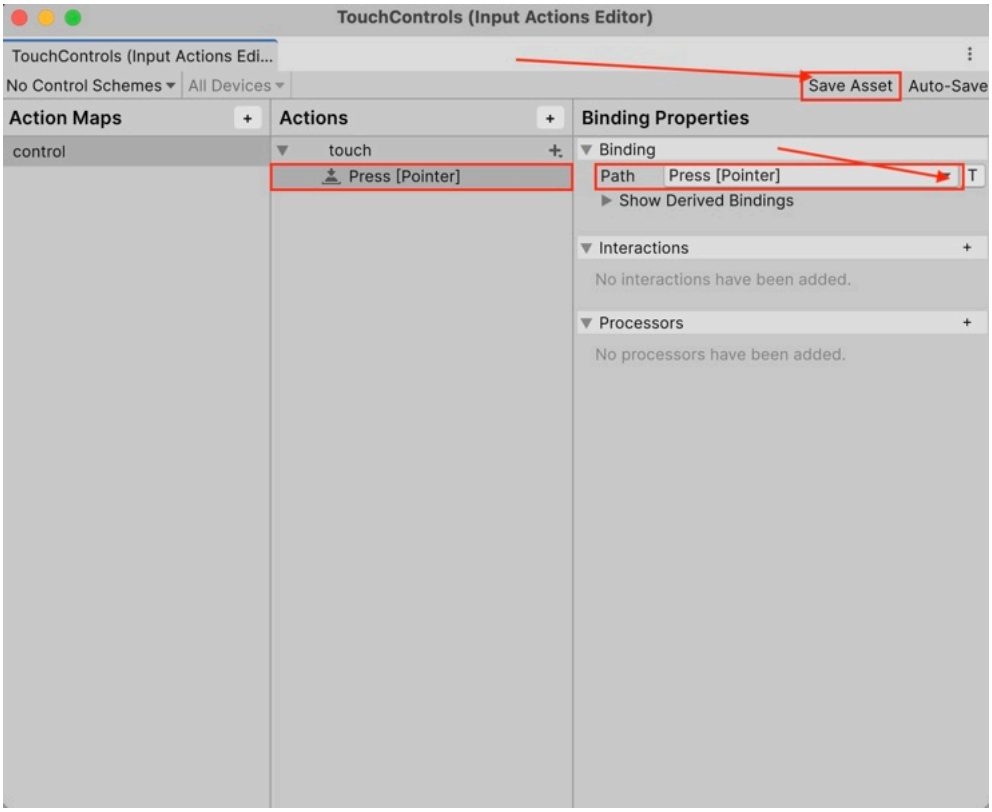


Рис. 2.37 – Визначення елементів введення

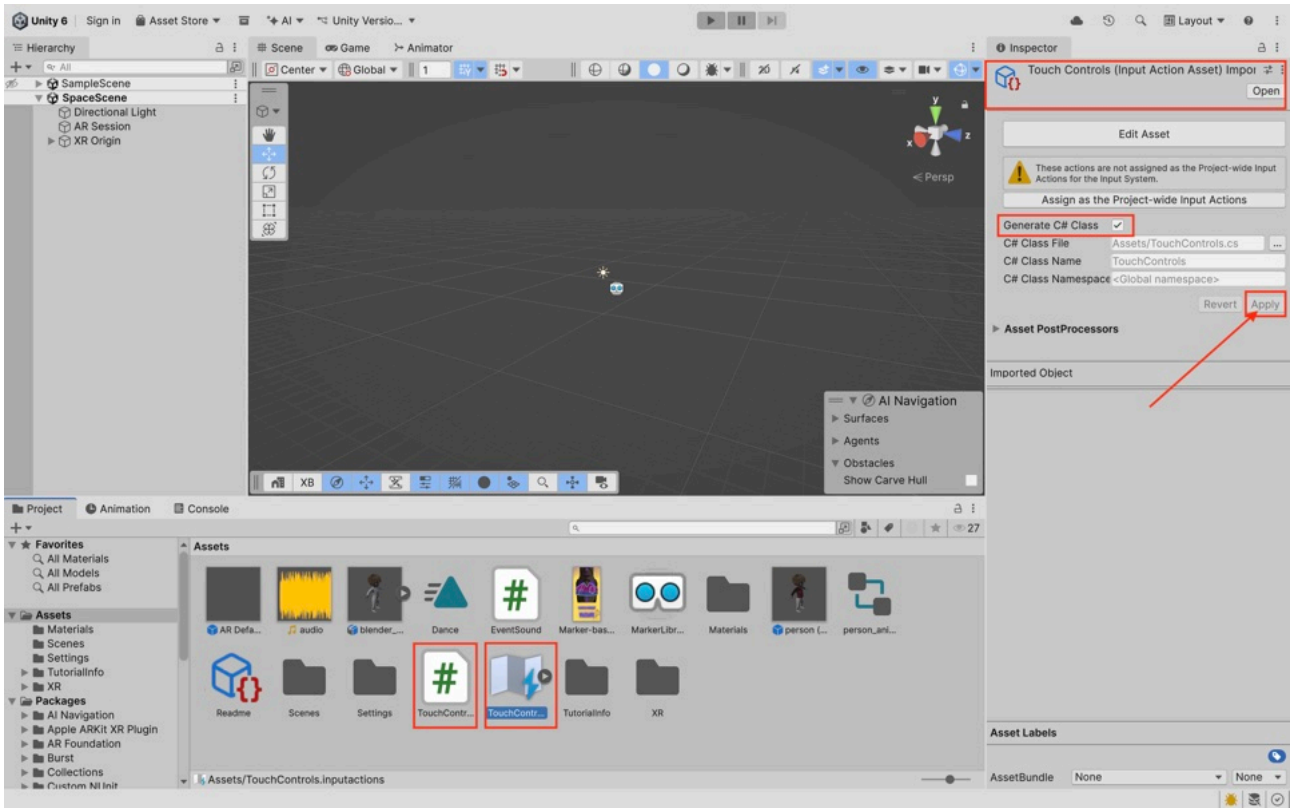


Рис. 2.38 – Генерування C#-скрипт для системи *Input Actions*

Останнім необхідно створити скрипт-компоненту для ігрового об’єкта *XR Origin.* У вікні *Inspector* необхідно натиснути *Add Component-> New script*, назвати файл та натиснути *Create and Add*. У створеному файлі необхідно написати програмний код (мова *C#*):

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using UnityEngine.InputSystem;

// Гарантує, що на об’єкті є ARRaycastManager, який потрібен для визначення AR-площин
[RequireComponent(typeof(ARRaycastManager))]

public class PlaceOnPlaneNewInputSystem : MonoBehaviour
{
    [SerializeField]
    [Tooltip("Instantiates this prefab on a plane at the touch location.")]

// Префаб, який буде розміщуватись на AR-сцені
    GameObject placedPrefab;

// Посилання на створений у сцені об'єкт
    GameObject spawnedObject;

// Клас для обробки touch-введення у (New Input System)
    TouchControls controls;

// Перевіряє, чи користувач торкається екрана
    bool isPressed;

// Забезпечує пошук та ідентифікацію AR-поверхонь у фізичному середовищі
    ARRaycastManager aRRaycastManager;

// Список результатів попадання Raycast-променя на AR-поверхню
    static List<ARRaycastHit> hits = new List<ARRaycastHit>();

    private void Awake()
    {

// Отримуємо компонент ARRaycastManager з об'єкта
        aRRaycastManager = GetComponent<ARRaycastManager>();

// Ініціалізація керування дотиками
        controls = new TouchControls();

// Подія початку дотику
        controls.control.touch.performed += _ => isPressed = true;

// Подія завершення дотику
```

```
        controls.control.touch.canceled += _ => isPressed = false;
    }


    void Update()
    {

// Перевірка дотику, якщо немає активного дотику — код не виконується
        if (Pointer.current == null || isPressed == false)
            return;

// Отримання позиції дотику
        var touchPosition = Pointer.current.position.ReadValue();

// Визначає, чи торкається користувач знайденої AR-поверхні
        if (aRRaycastManager.Raycast(touchPosition, hits,
TrackableType.PlaneWithinPolygon))
        {
            var hitPose = hits[0].pose;

// Якщо об'єкт ще не створений — створюємо його
            if (spawnedObject == null)
            {
                spawnedObject = Instantiate(placedPrefab, hitPose.position,
hitPose.rotation);
            }
            else
            {
// Якщо об'єкт вже створений — переміщуємо його
                spawnedObject.transform.position = hitPose.position;
                spawnedObject.transform.rotation = hitPose.rotation;
            }

// Обчислюємо напрямок до камери
            Vector3 lookPos = Camera.main.transform.position -
spawnedObject.transform.position;
            lookPos.y = 0;

// Повертаємо об'єкт до камери
            spawnedObject.transform.rotation = Quaternion.LookRotation(lookPos);
        }
    }

// Увімкнення touch-керування
    private void OnEnable()
    {
        controls.control.Enable();
    }

// Вимкнення touch-керування
    private void OnDisable()
    {
        controls.control.Disable();
```

```
    }
}
```

Для створеної скрипт-компоненти необхідно додати префаб з розділу 2.2.2 (рис. 2.39).

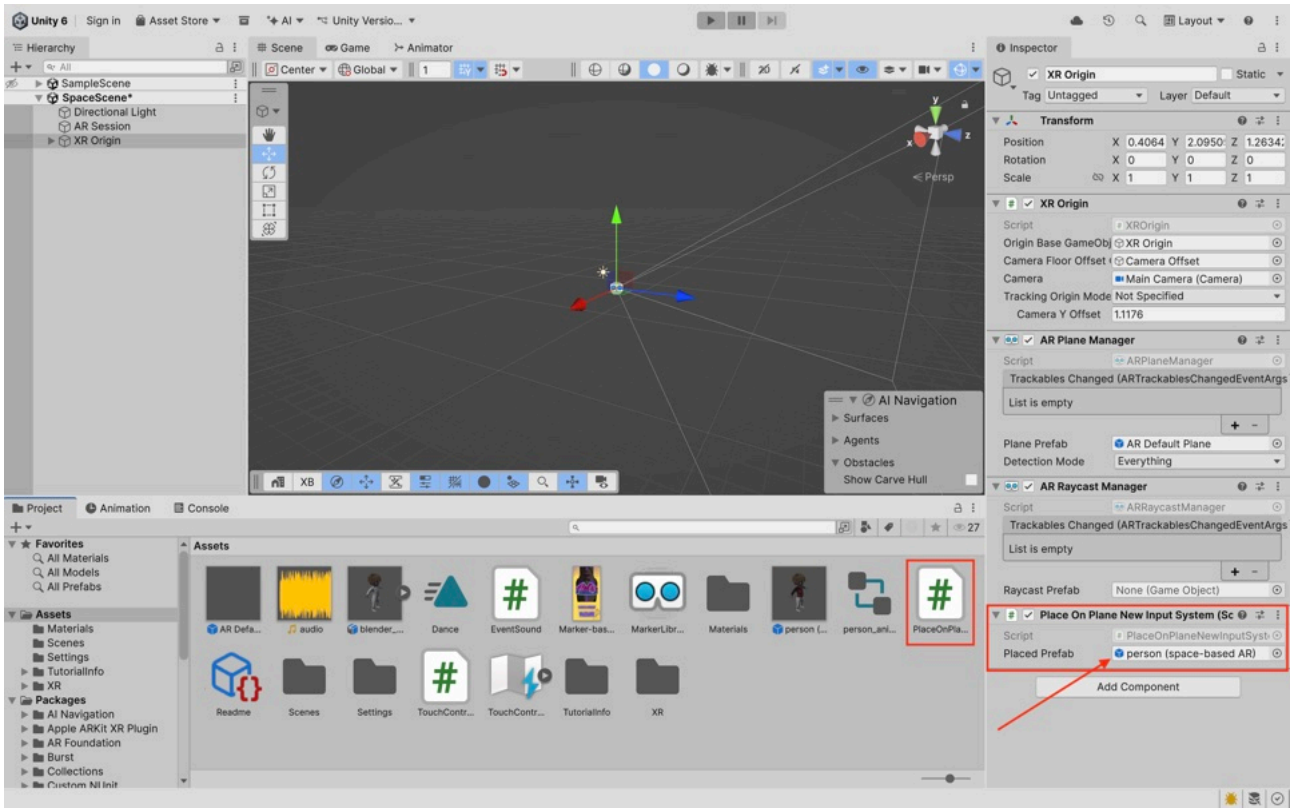


Рис. 2.38 – Налаштування скрипт-компоненти

2.2.5 Збирання та розгортання *AR*-застосунку

1) Збирання та розгортання *AR*-застосунку на основі *iOS*-платформи.

Для збирання *AR*-застосунку необхідно обрати *File-> Build And Run*, створити та назвати теку, де буде зберігатись збірка, напр., *iOS_Build* та натиснути *Choose* (рис. 2.39). Збирання *AR*-застосунку виконується протягом певного часу.

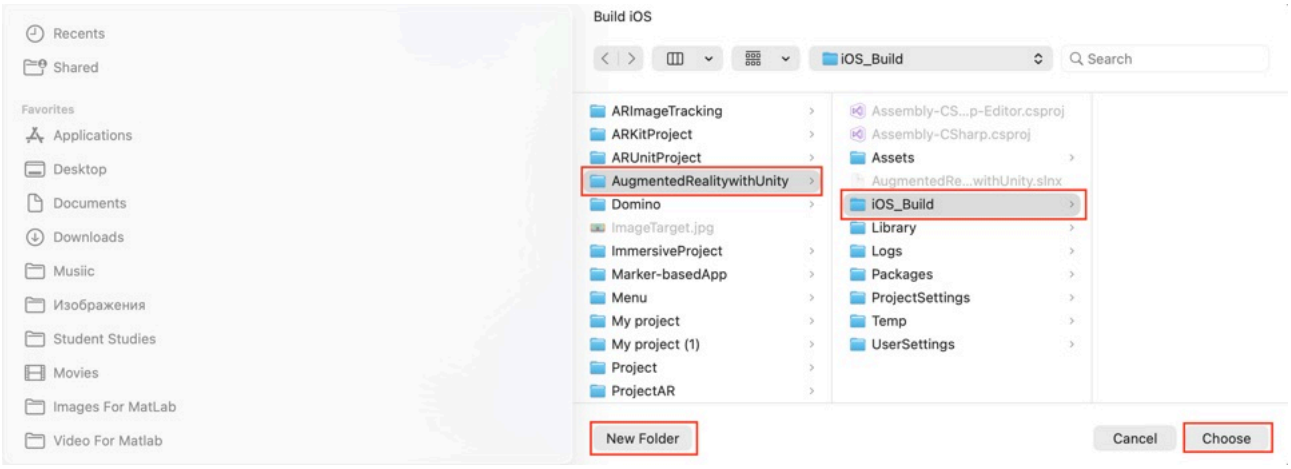


Рис. 2.39 – Збирання *AR*-застосунку

Для розгортання *AR*-застосунку на основі *iOS*-платформи спочатку автоматично відкриється файл *Unity-iPhone.xcodeproj* в середовищі *Xcode* (рис. 2.40). Далі необхідно активувати функцію *Automatically manage signing-> Enable Automatic* та додати профіль розробника (рис. 2.40), якщо розроблення застосунків для *iOS*-платформи відбувається вперше, якщо не вперше – необхідно обрати вже створений профіль розробника.

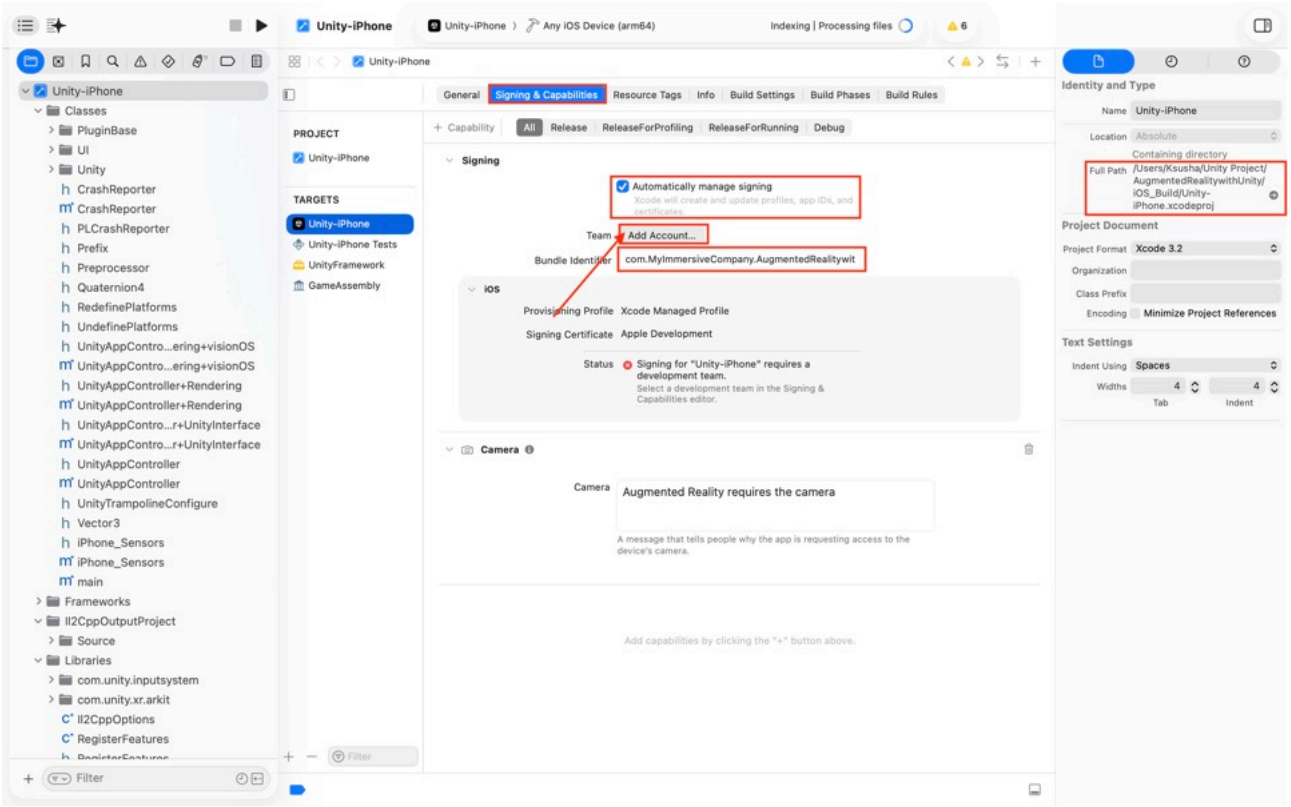
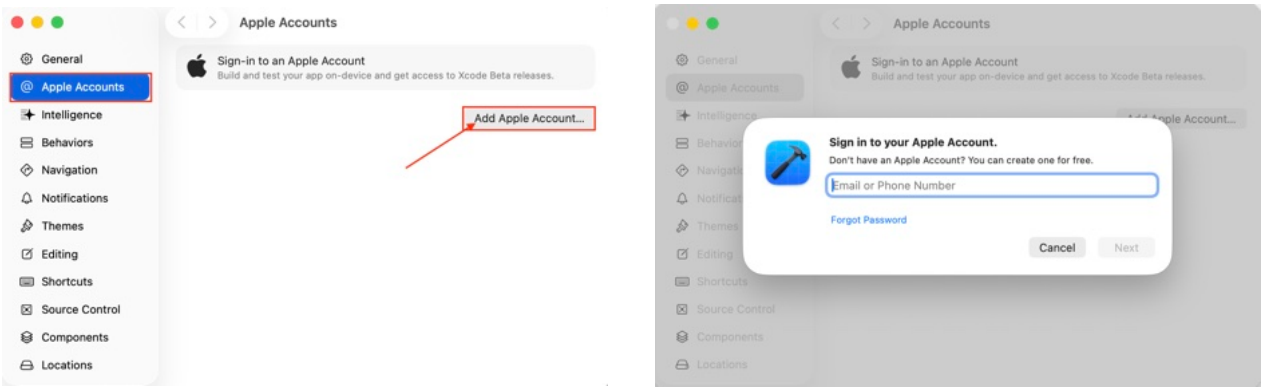
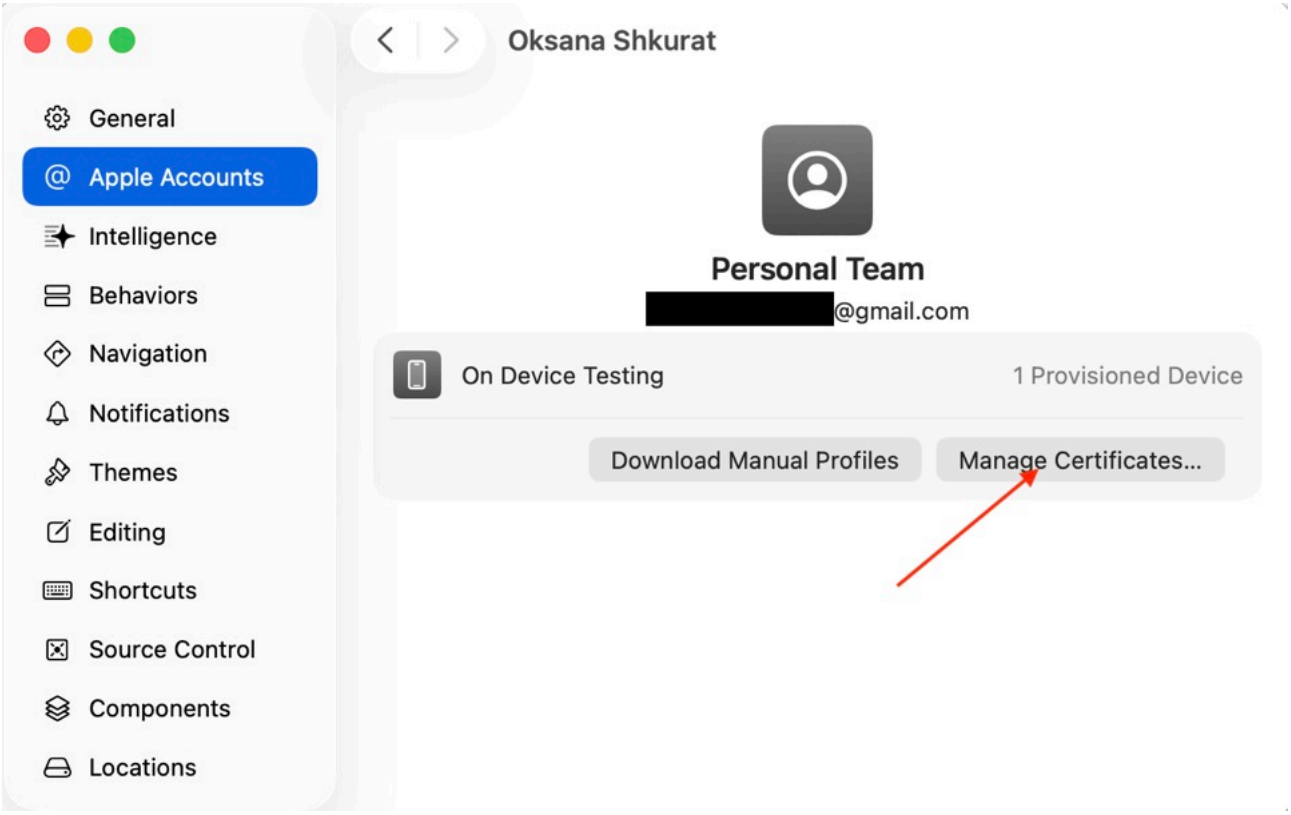


Рис. 2.40 – Середовище *Xcode*

Наступним необхідно створити профіль розробника (якщо розроблення застосунків для *iOS*-платформи відбувається вперше). Для цього в середовищі *Xcode* натиснути *Add Account…* (рис. 2.40), обрати *Apple Accounts* (рис. 2.41) та натиснути *Add Apple Account…*. Далі необхідно створити профіль або авторизуватись в *Apple Account* та налаштувати *iOS*-пристрій (рис. 2.41).



а) б)



в)

Рис. 2.41 – Створення профілю розробника: а) додавання *Apple Account*; б) авторизація в *Apple Account*; в) підключення та налаштування *iOS*-пристрою

Створений профіль розробника (рис. 2.42) необхідно обрати в *Xcode* (рис. 2.44).

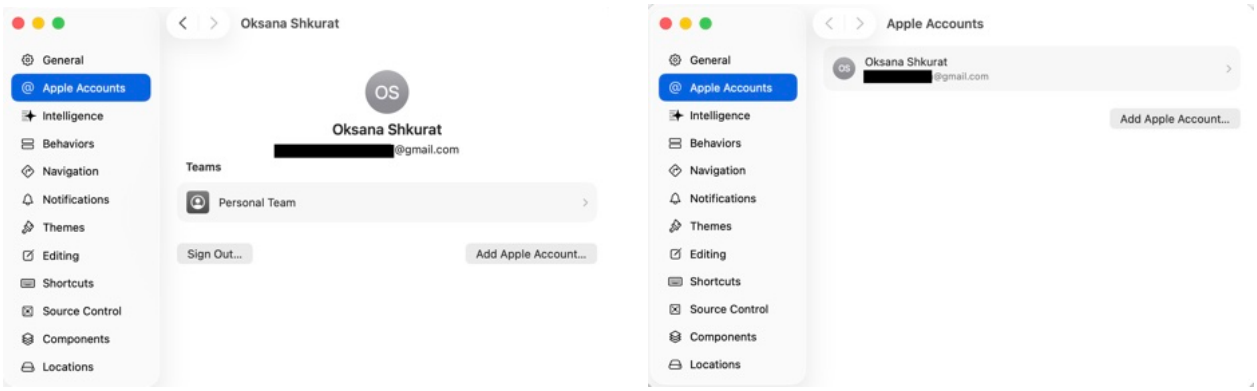


Рис. 2.42 – Створений профіль розробника

Останнім необхідно налаштувати *iOS*-пристрій, на якому буде розгорнуто *AR*-застосунок. Для цього необхідно підключити *iOS*-пристрій до персонального комп'ютера та обрати *Параметри* (*Налаштування*)-> *Приватність і безпека*-> *Режим розробника*, ввімкнути *Режим розробника*. Перезавантаження *iOS*-пристрою виконується протягом певного часу.

Далі необхідно підтвердити надійність розробника. Для цього на *iOS*-пристрої необхідно обрати *Параметри*-> *Загальні*-> *VPN і керування пристроєм*, натиснути *Довіряти* розробнику та *Дозволити* програмам відкриватись та отримувати доступ до даних *iOS*-пристрою (рис. 2.43).

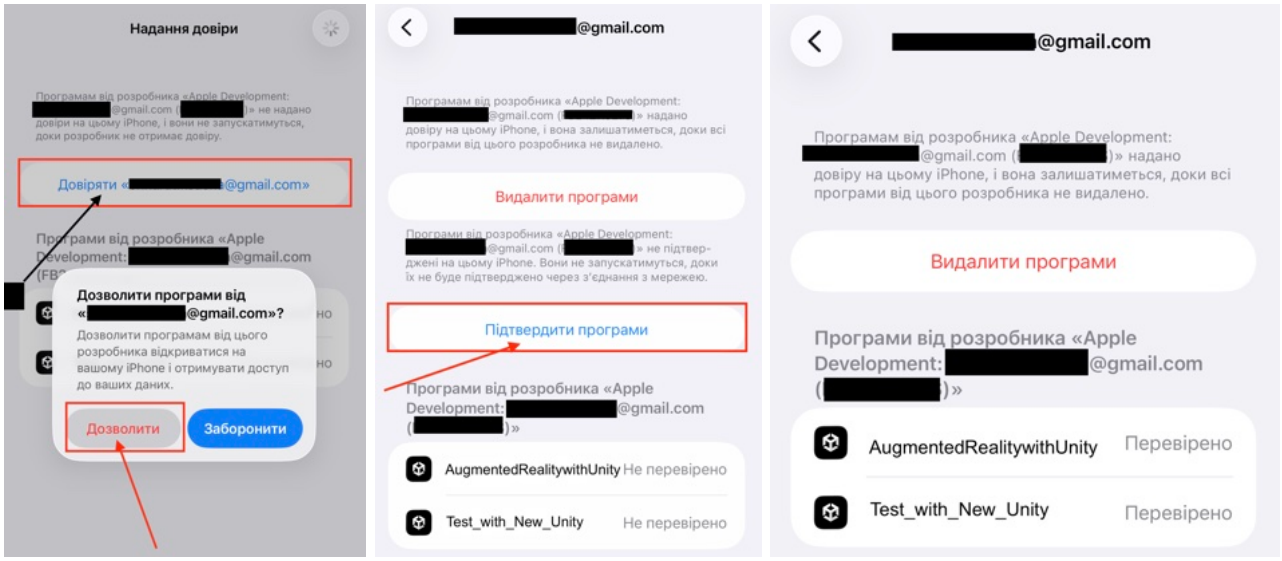


Рис. 2.43 – Підтвердження надійності розробника

В середовищі *Xcode* необхідно натиснути кнопку відтворення та виконати розгортання *AR*-застосунку на фізичному *iOS*-пристрої (рис. 2.44).

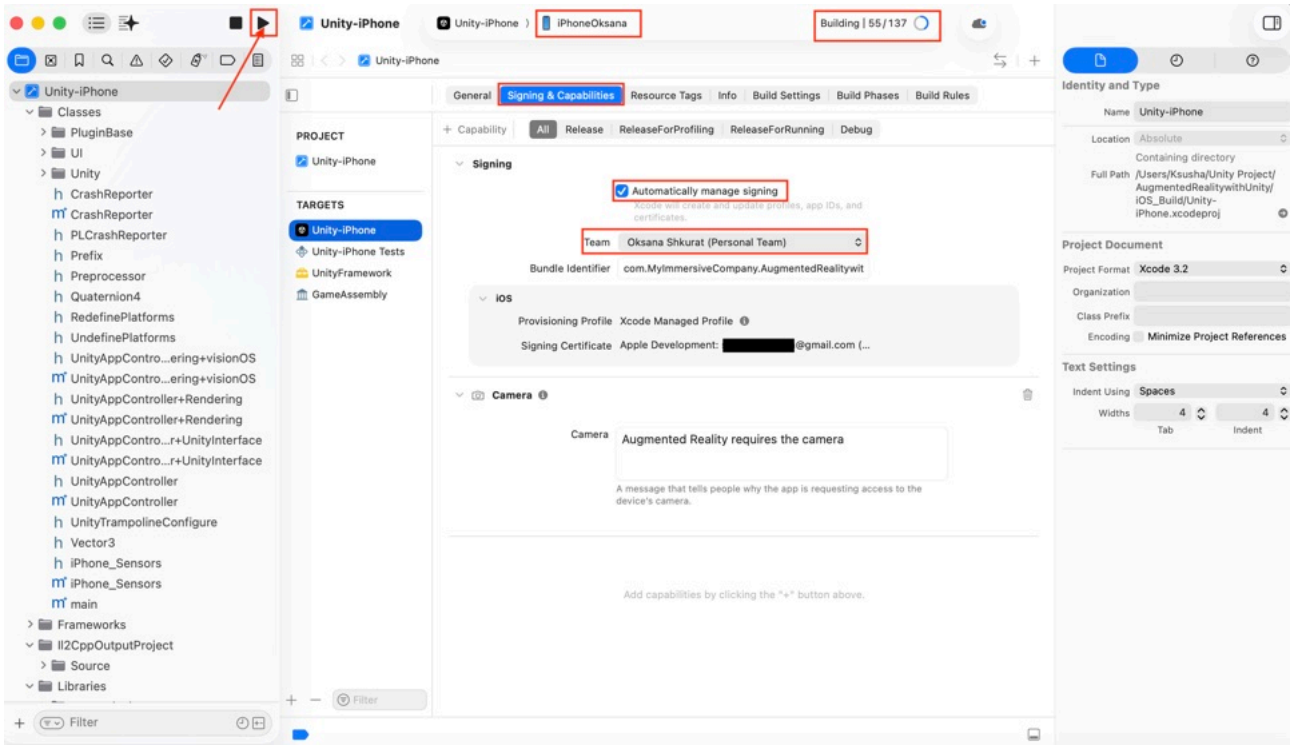


Рис. 2.44 – Розгортання *AR*-застосунку на фізичному *iOS*-пристрої

Розгортання *AR*-застосунку виконується протягом певного часу. Після завершення розгортання *AR*-застосунок встановлюється та запускається на підключеному *iOS*-пристрої.

2) Збирання та розгортання *AR*-застосунку на основі *Android*-платформи.

Для налаштування *Android*-пристрою необхідно обрати *Налаштування-> Про планшет* та натиснути декілька разів на *Версія системи* (рис. 2.45). В результаті з’явиться запит на розблокування *Android*-пристрою та повідомлення, що ви є розробником.

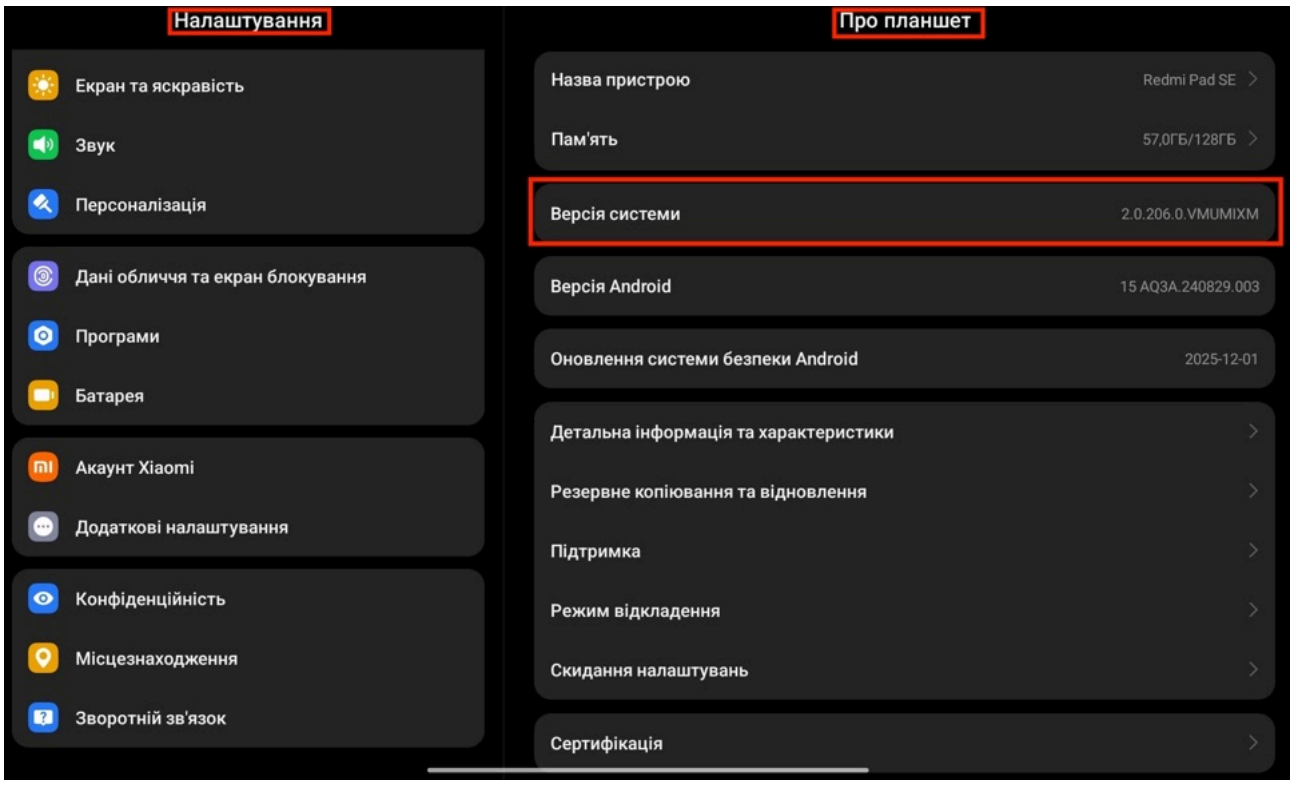


Рис. 2.45 – Розблокування *Android*-пристрою

Наступним необхідно обрати *Налаштування-> Додаткові налаштування-> Параметри розробника* та ввімкнути *Режим розробника* (рис. 2.46).

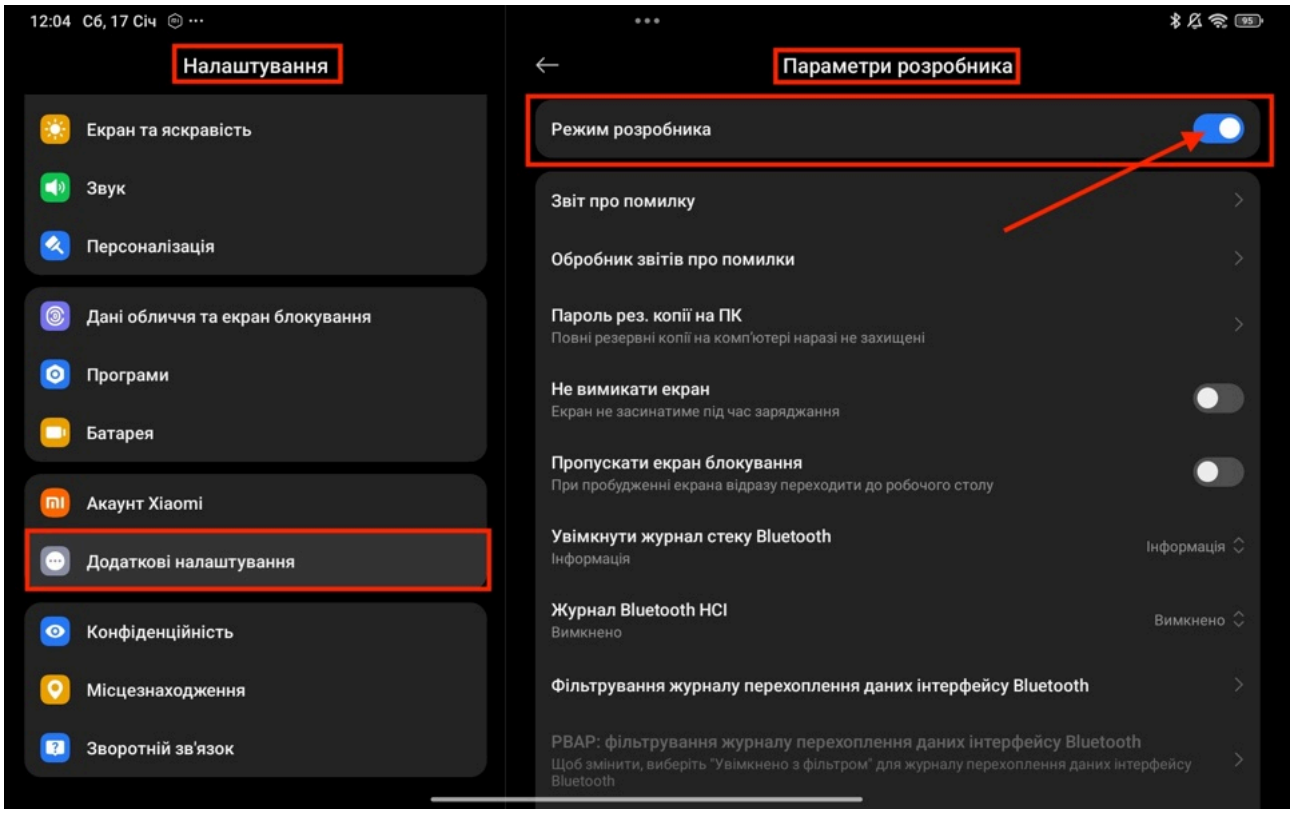


Рис. 2.46 – Ввімкнення режиму розробника на *Android*-пристрої

Останнім необхідно ввімкнути налаштування *Налагодження USB* (рис. 2.47). Для деяких *Android*-пристроїв для встановлення програм також необхідно ввімкнути *Встановити за допомогою USB*.

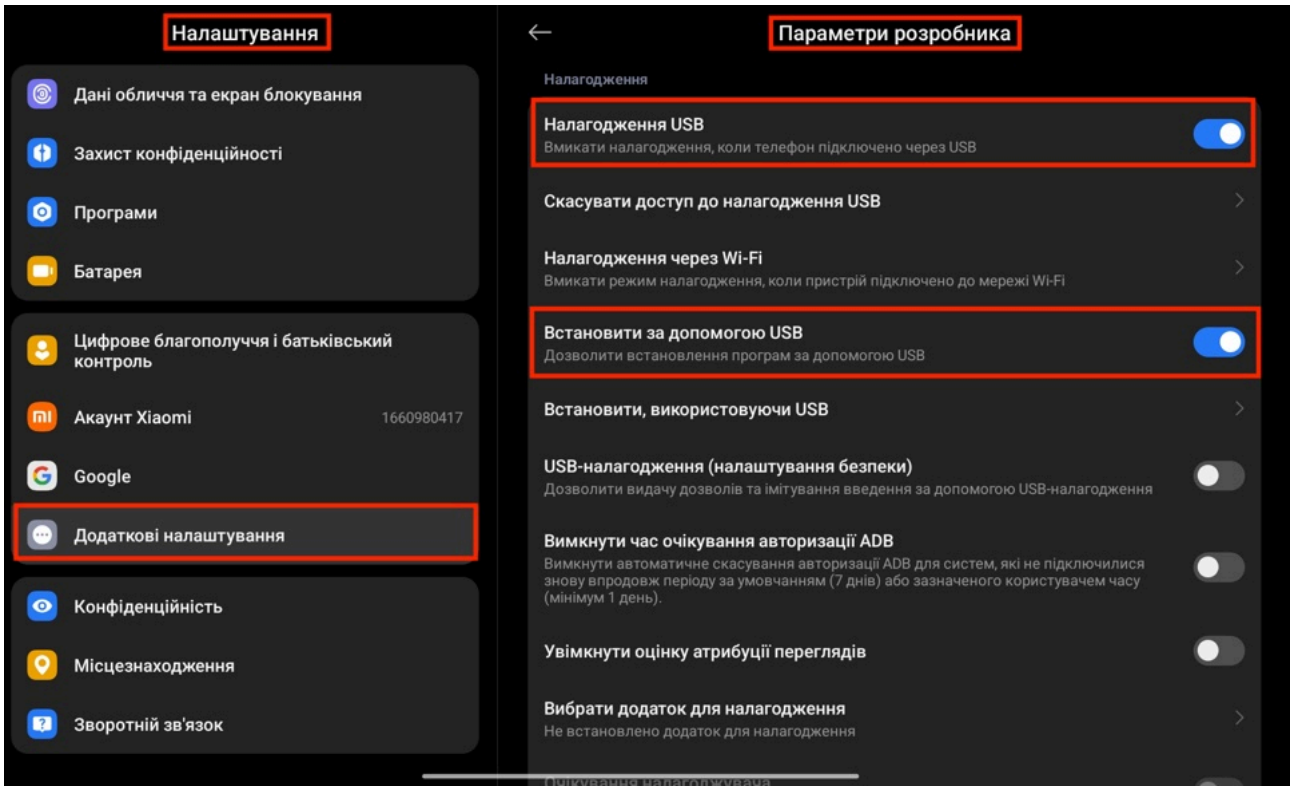


Рис. 2.47 – Налаштування параметрів розробника на *Android*-пристрої

Для збирання та розгортання *AR*-застосунку необхідно обрати *File-> Build Profiles*. Далі необхідно підключити *Android*-пристрій до персонального комп'ютера та натиснути *Refresh* (рис. 2.48). Після виконання, *Android*-пристрій відобразиться у списку пристроїв (рис. 2.49) та необхідно натиснути *Build And Run*.

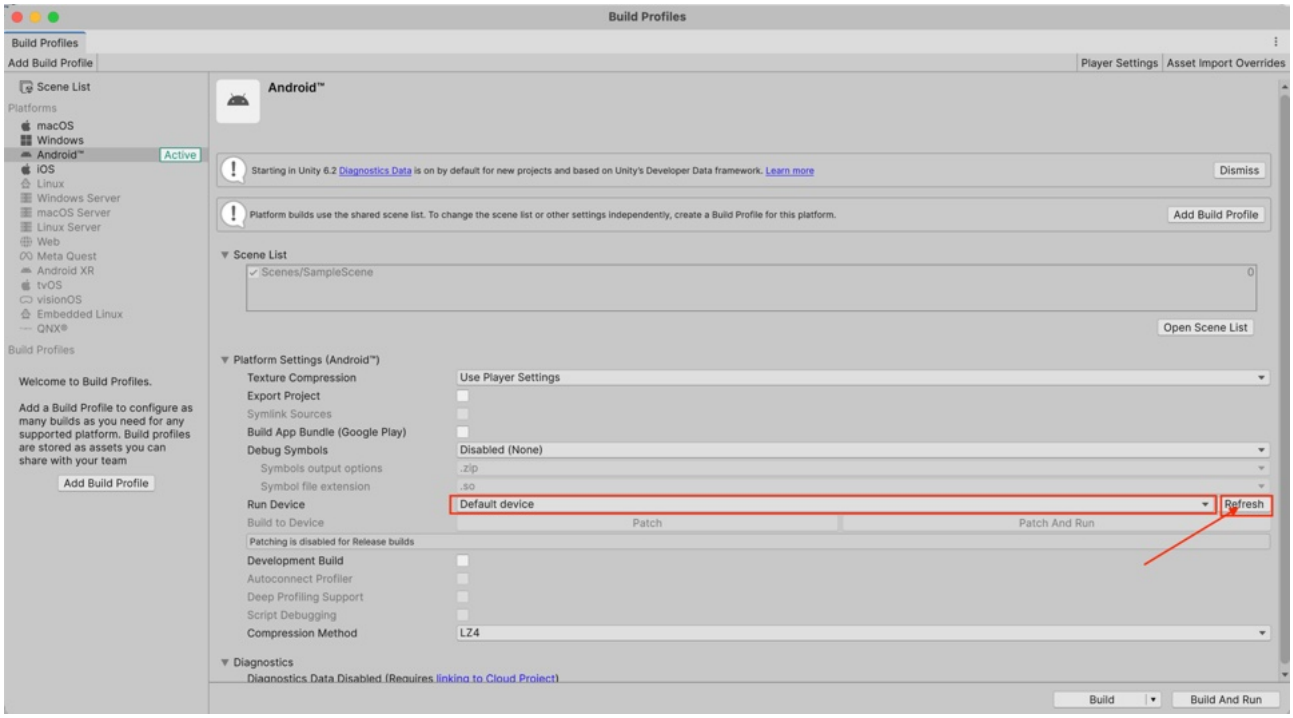


Рис. 2.48 – Підключення *Android*-пристрою для збирання та розгортання *AR*-застосунку

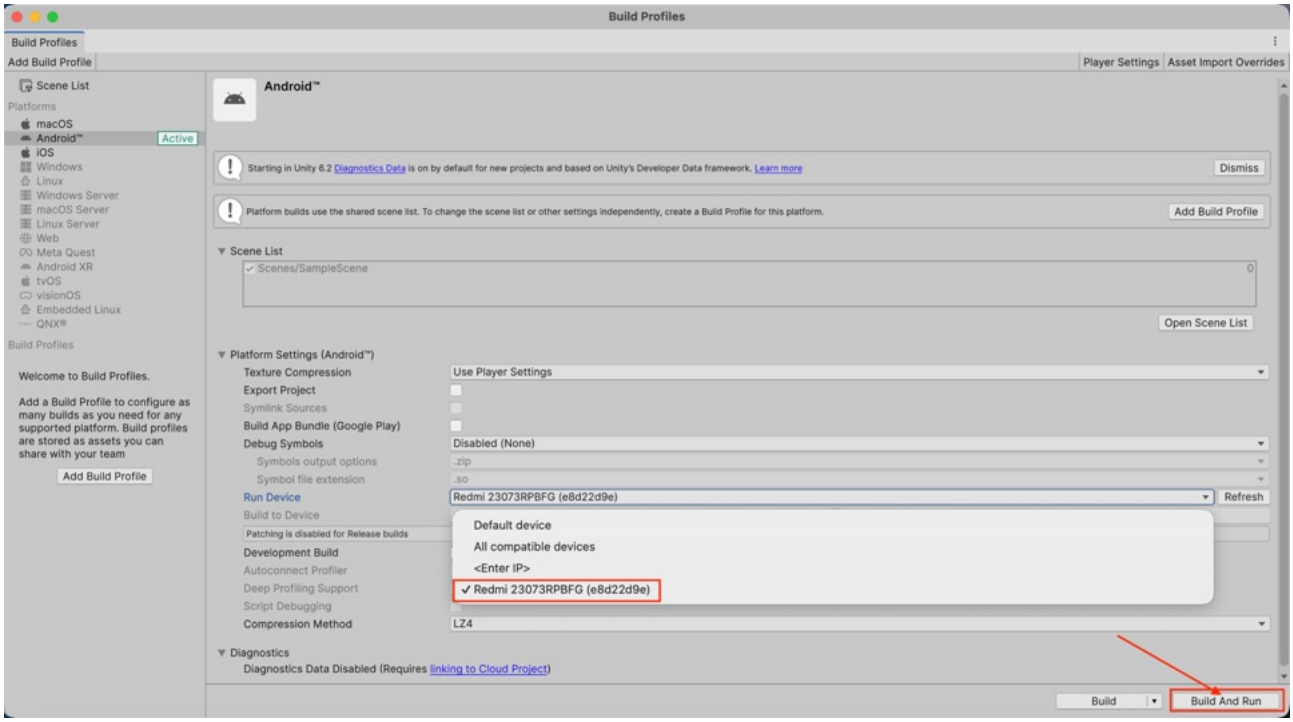


Рис. 2.49 – Збирання та розгортання *AR*-застосунку на фізичному *Android*-пристрої

Останнім необхідно дати назву *.apk* файлу, створити та назвати теку, де буде зберігатись *.apk* файл (рис. 2.50) та натиснути *Create*, потім *Save*.

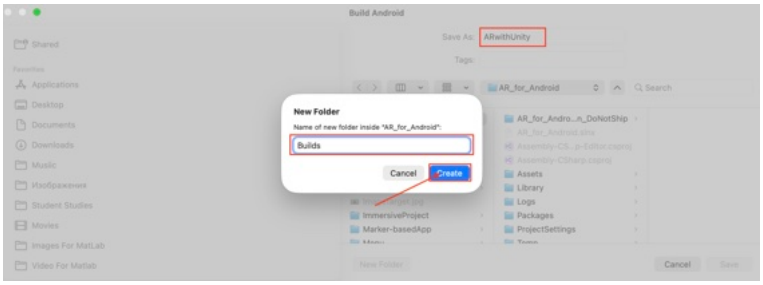


Рис. 2.50 – Зберігання *.apk* файлу *ARwithUnity* в теці *Builds*

Збирання та розгортання *AR*-застосунку виконується протягом певного часу. Після завершення розгортання *AR*-застосунок встановлюється та запускається на підключеному *Android*-пристрої (рис. 2.51).

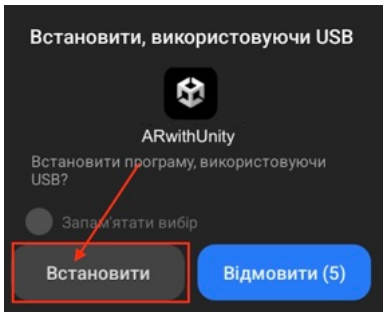
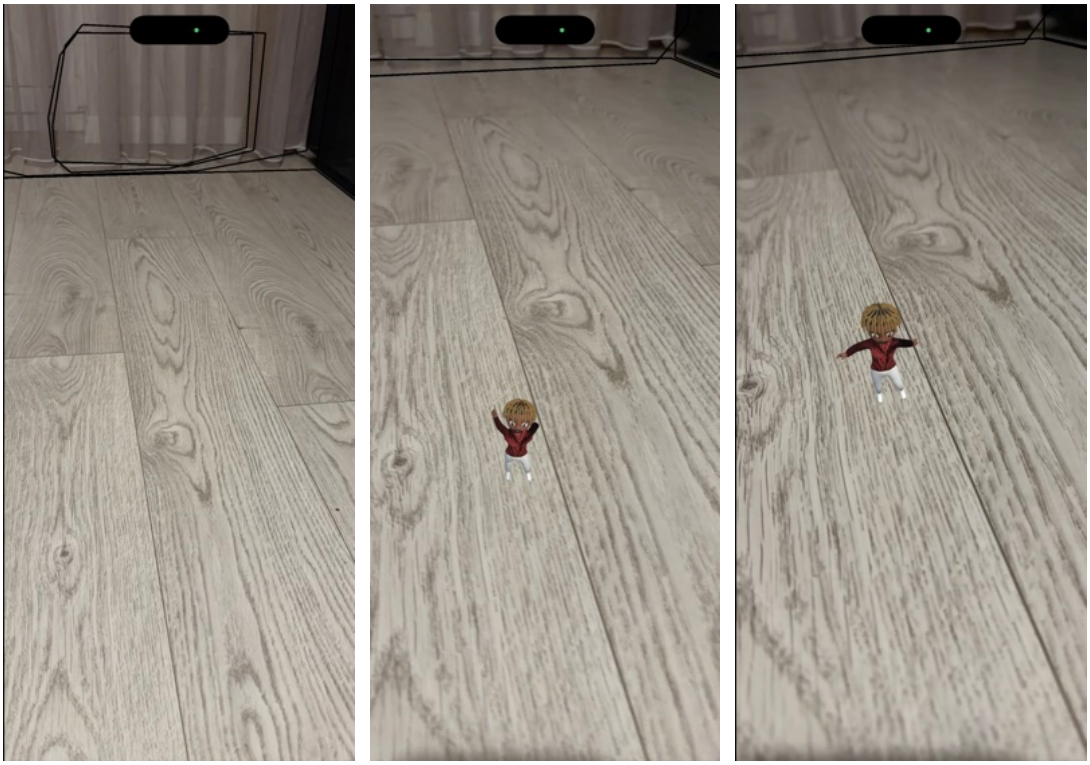


Рис. 2.51 – Запит на встановлення *AR*-застосунку

2.2.6 Демонстрація роботи *AR*-застосунків

Результат роботи *AR*-застосунку на основі розпізнавання зображення представлено на рис. 2.52. Результат роботи *AR*-застосунку на основі розпізнавання горизонтальних та вертикальних поверхонь представлено на рис. 2.53.

Рис. 2.52 – Приклад роботи *AR*-застосунку маркерної доповненої реальності



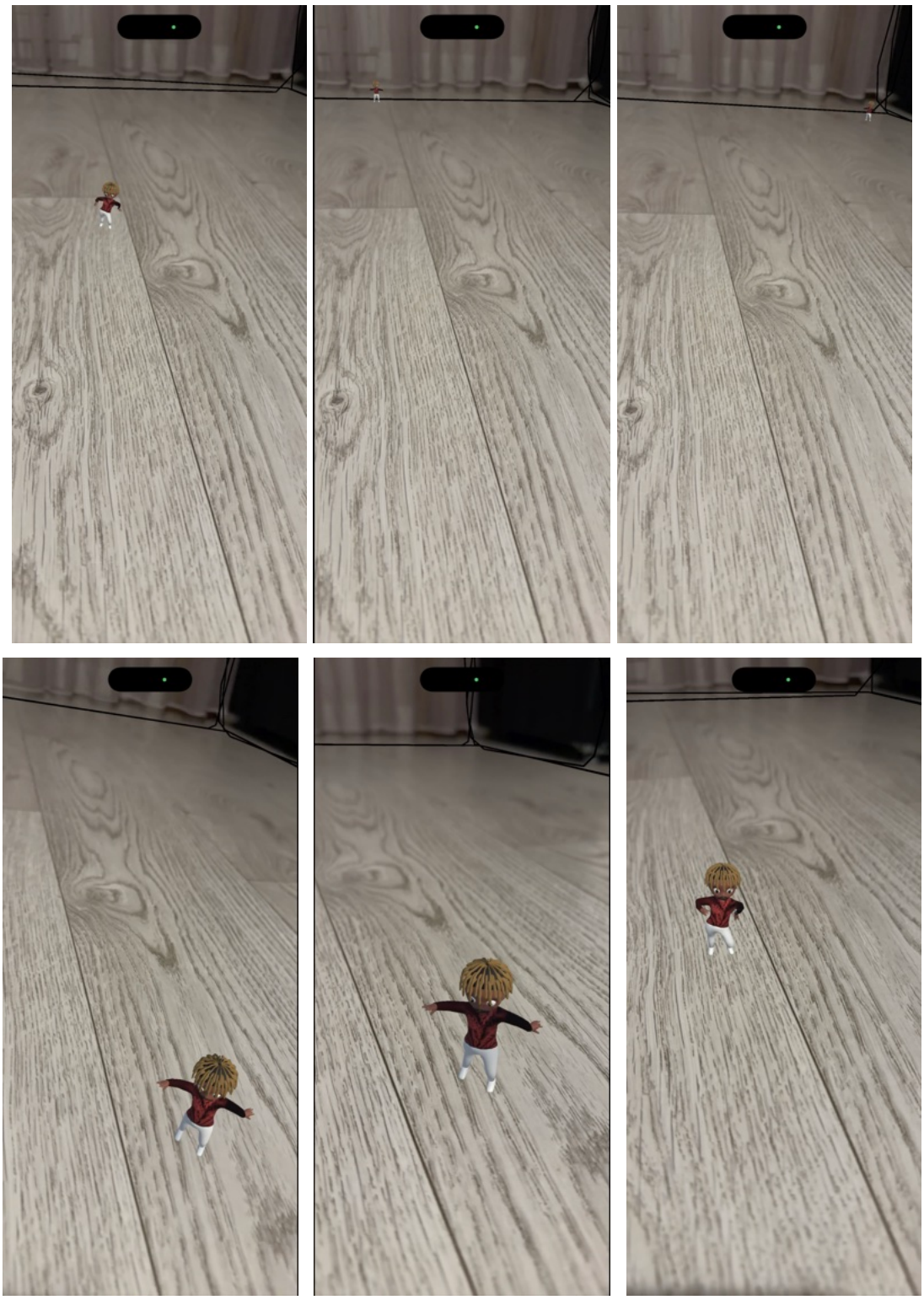


Рис. 2.53 – Приклад роботи *AR*-застосунку просторової доповненої реальності

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Для чого використовується *XR Plug-in Management* в *Unity*-проєктах?
2. Які кроки необхідно виконати, щоб використовувати *XR Plug-in Management* в *Unity*-проєкті?
3. Які *XR*-плагіни можна підключити в *Unity*-проєкт за допомогою *XR Plug-in Management*?
4. Для чого використовується *XR Simulation* в *Unity*-проєктах?
5. Які кроки необхідно виконати, щоб використовувати *XR Simulation* в *Unity*-проєкті?
6. Що таке *AR Foundation* в *Unity*-проєкті?
7. Які типи доповненої реальності підтримує *AR Foundation*?
8. Чому *ARKit* та *ARCore* є ключовими технологіями для *AR Foundation*?
9. Опишіть структуру *Unity*-проєкту для створення доповненої реальності.
10. Яке призначення ігрового об’єкта *AR Session*?
11. Опишіть структуру та призначення ігрового об’єкта *XR Origin*.
12. Яке призначення компоненти *AR Session*?
13. Яке призначення компоненти *AR Session Origin*?
14. Яке призначення компоненти *AR Camera*?
15. Яке призначення компоненти *AR Tracked Image Manager* та як реалізуються її функції?
16. Яке призначення компоненти *AR Plane Manager* та як реалізуються її функції?
17. Яке призначення компоненти *AR Raycast Manager* та як реалізуються її функції?
18. Яке призначення системи *Input Actions* в *Unity* для опису та оброблення дій користувача? Яким чином реалізуються її функції для оброблення введення з різних пристроїв (сенсорний екран, клавіатура, контролери)?
19. Інструменти *Unity* для роботи з *3D*-об’єктами та матеріалами.

20. Які інструменти *Unity* застосовуються для анімації ігрових об’єктів? Які компоненти та системи використовуються для створення, редагування і керування анімаціями у проєкті?

ЛАБОРАТОРНИЙ ПРАКТИКУМ №3. РОЗРОБЛЕННЯ ЗАСТОСУНКУ ЗМІШАНОЇ РЕАЛЬНОСТІ В ІГРОВОМУ РУШІЇ *UNITY*

Лабораторний практикум присвячений розробленню *MR*-застосунку в ігровому рушії *Unity*. Для створення змішаної реальності на основі *iOS*-платформи використовується фреймворк *ARKit*, який доступний в *Unity* через плагін *Apple ARKit XR Plugin*. Для створення доповненої реальності на основі *Android*-платформи використовується *SDK ARCore*, який доступний в *Unity* через плагін *Google ARCore XR Plugin*. Для реалізації взаємодії з цифровими об'єктами застосовується *Unity*-фреймворк *AR Foundation*.

В розділі «Порядок виконання» розглянутий процес розроблення *MR*-застосунку на основі фреймворку *AR Founation* з реалізацією оклюзії та взаємодії з цифровим *3D*-об'єктом (розміщення, зсув, обертання, масштабування) за допомогою сенсорного введення користувача.

ЗАВДАННЯ

Розробити *MR*-застосунок в ігровому рушії *Unity*. *MR*-застосунок повинен містити *3D*-об'єкт(и) з лабораторного практикуму №1 (обов'язково), а також мультимедійні дані – *3D*-об'єкти з відкритих сторонніх джерел, аудіо, відео, текст (за бажанням). Для розробленого *3D*-об'єкта(ів) повинна бути реалізована анімація, а також застосовані відповідні текстури та матеріали. В *MR*-застосунку повинна бути реалізована взаємодія користувача (розміщення, зсув, обертання, масштабування) з розробленим *3D*-об'єктом(ами). *MR*-застосунок повинен містити ігрове вікно з інструкцією щодо правил взаємодії з цифровими об'єктами.

Звіт з лабораторного практикуму повинен містити:

1. Титульний аркуш, формулювання завдання, текстовий опис процесу розроблення *MR*-застосунку, зокрема інформація щодо використаних інструментів та технологій, фрагменти програмного коду, скриншоти проміжних та ключових результатів (*.pdf* файл).

2. Демонстрацію роботи *MR*-застосунку (*.mp4* файл).

3. Архів програми (*.zip* файл).

3.1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Змішана реальність є *XR*-технологією, що створює гібридне середовище, в якому цифрові об’єкти інтегруються та взаємодіють з фізичними об’єктами в режимі реального часу.

Взаємодією між фізичними та цифровими об’єктами в *MR*-системах може бути зміна положення, орієнтації або масштабу цифрових об’єктів шляхом введення користувача. Взаємодією також може бути зміна параметрів рендерингу цифрових об’єктів відповідно до зміни освітлення фізичного середовища в режимі реального часу. Найбільш реалістичною взаємодією є оклюзія, коли цифрові об’єкти частково або повністю перекриваються фізичними та навпаки в режимі реального часу. Для реалізації оклюзії у системах доповненої та змішаної реальності застосовуються методи відстеження положення та орієнтації фізичних об’єктів. На основі отриманих даних генерується карта фізичного середовища, що дозволяє визначати взаємне перекриття між цифровими та фізичними об’єктами, що забезпечує інтеграцію цифрового середовища в фізичне середовище.

Змішана реальність ґрунтується на технологіях відстеження рухів рук (*hand tracking*), погляду (*gaze tracking*) або контролерів (*MR control tracking*) для реалізації взаємодії цифрового середовища з фізичним середовищем. В *Unity* для реалізації різних видів взаємодії використовується *XR Interaction Toolkit*.

XR Interaction Toolkit є одним із ключових прикладним програмним інтерфейсом (*API*) в *Unity* для розроблення взаємодії користувача у доповненій, змішаній та віртуальній реальностях, який надає уніфіковану, модульну та розширювану архітектуру для роботи з введенням користувача на основі відстеження рук, погляду та *XR*-контролерів. *XR Interaction Toolkit* абстрагує апаратно-залежні механізми введення, що дозволяє реалізовувати універсальні сценарії взаємодії для різних *XR*-пристроїв. *XR Interaction Toolkit*

використовується для створення стандартних механік взаємодії, зокрема захоплення, натискання, обертання, переміщення, телепортації.

Для завантаження *XR Interaction Toolkit* в *Unity*-проєкт необхідно обрати *Window-> Package Management-> Package Manager*. Далі необхідно обрати реєстр пакетів *Unity-> XR Interaction Toolkit* та натиснути *Install* (рис. 3.1).

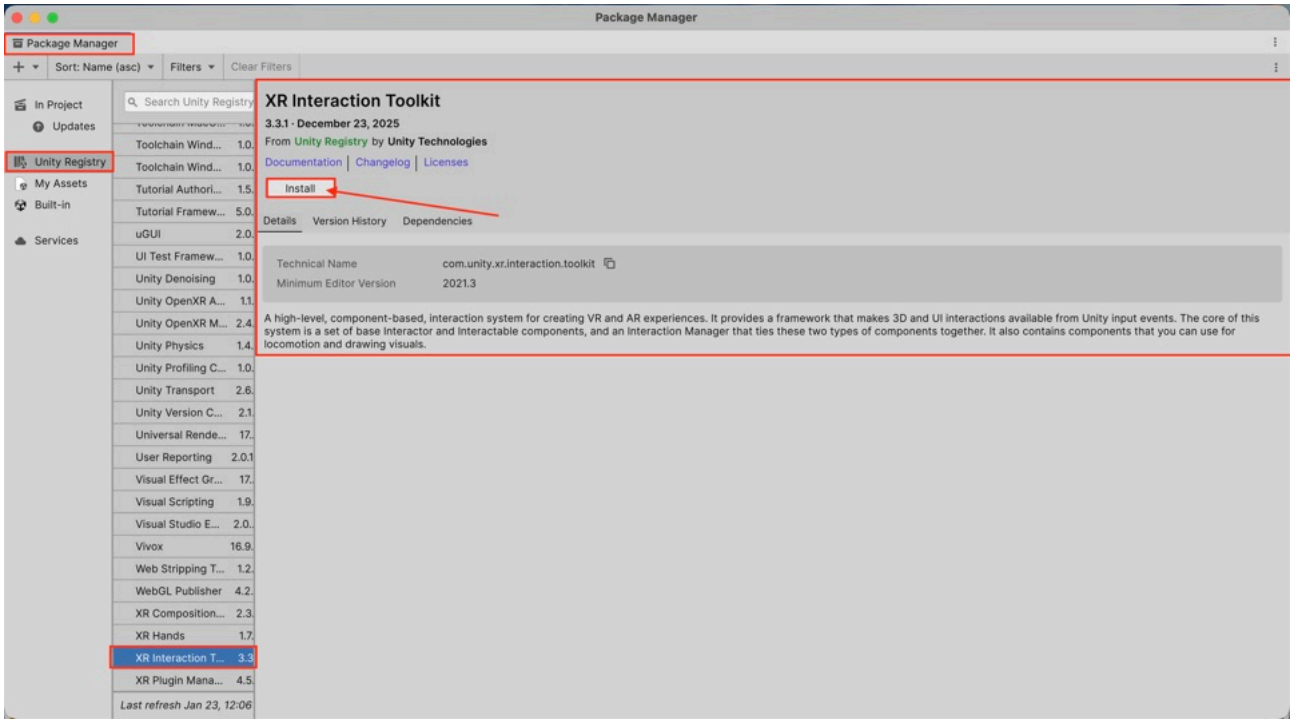


Рис. 3.1 – Встановлення *XR Interaction Toolkit* в *Unity*-проєкт

XR Interaction Toolkit надає пакети готових асетів, які прискорюють розроблення *MR*-сцени, зокрема *Starter Assets* для контролерів і рук, *XR Interaction Simulator* для тестування взаємодії в *Unity*-редакторі без реальних пристроїв (рис. 3.2).

Starter Assets в *XR Interaction Toolkit* є головним набором ресурсів, який містить префаби *XR Origin*, лівого та правого контролерів з прямим керування та керуванням *Ray*-променя, а також *XRI Default Input Actions* для керування контролерами та руками. Завдяки цим інструментам можна одразу створювати *XR*-сцени, де користувач може захоплювати, переміщати, обертати та масштабувати об’єкти. Для тестування без *XR*-шолома *Unity*-сцену можна додати префаб *XR Interaction Simulator*, який симулює контролери, дозволяючи працювати з руками типу *grab*, *pinch* та *select* прямо в *Unity*-редакторі (рис. 3.3).

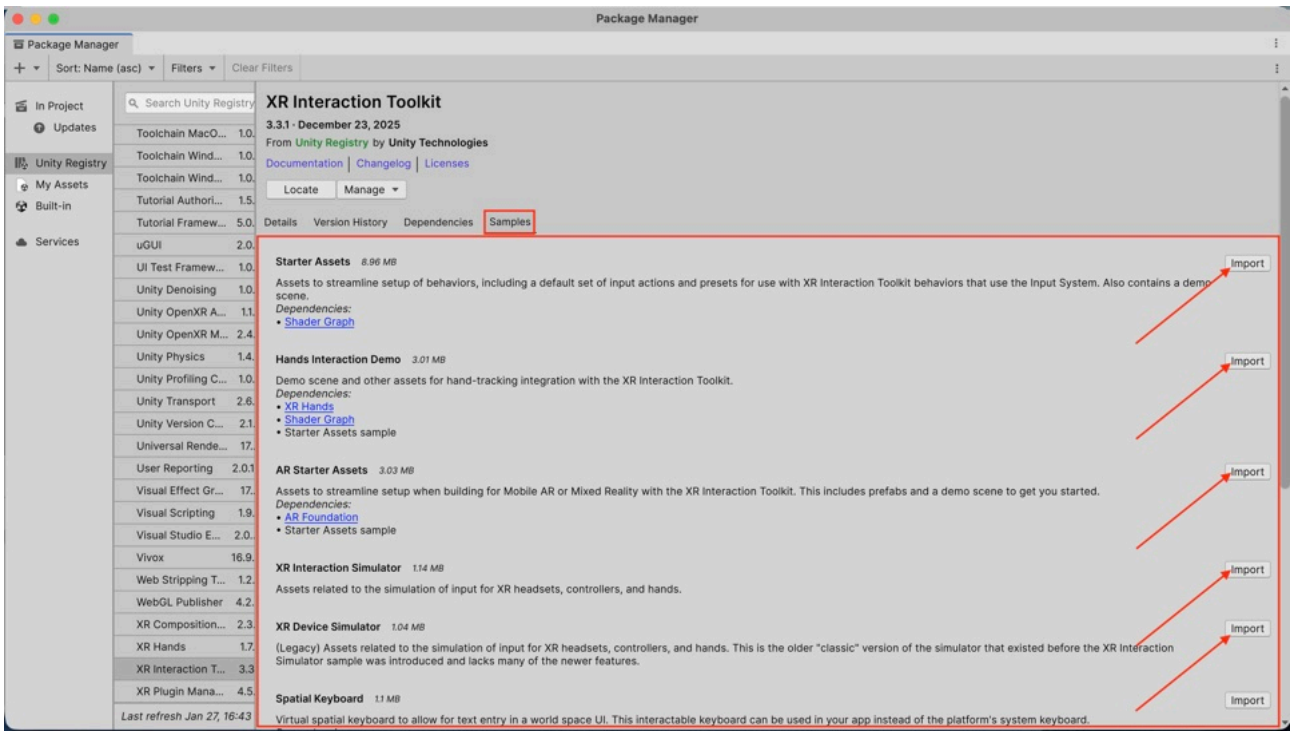


Рис. 3.2 – Імпортування асетів *XR Interaction Toolkit*

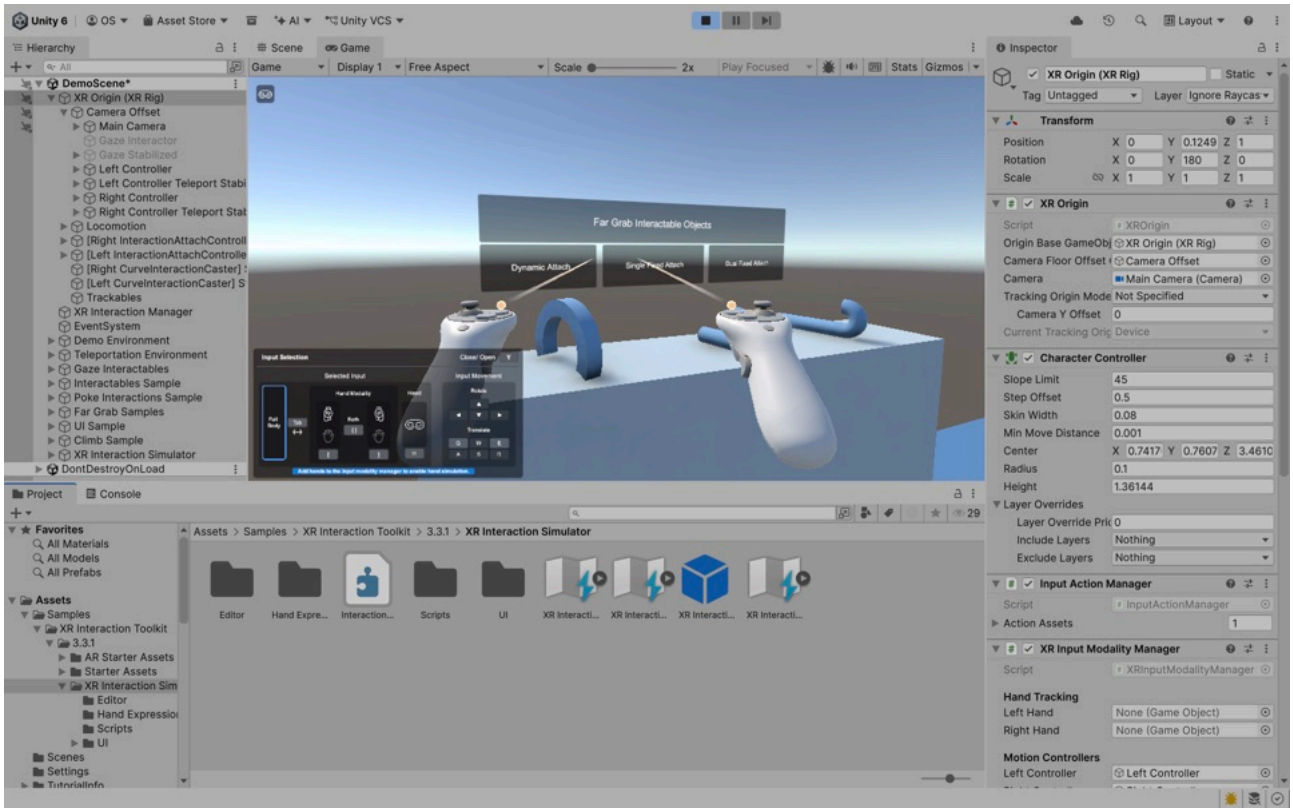


Рис. 3.3 – *Unity*-сцена з *XR Interaction Simulator*: симуляція контролерів і захват об’єктів

В *XR Interaction Toolkit* основним ігровим об'єктом є *XR Rig*, який представляє користувача у *XR*-середовищі та містить дочірні ігрові об'єкти – *XR*-камеру та контролери. *XR Controller* – це ігровий об'єкт, що відповідає за фізичний *XR*-пристрій користувача (лівий або правий контролер, ліва або права рука) та необхідний для зчитування користувацького введення та передачу його інтеракторам при реалізації взаємодії з цифровими *3D*-об'єктами.

Всі взаємодії в *XR Interaction Toolkit* будуються на поняттях інтеракторів (*Interactors*) та інтерактивних об'єктах (*Interactables*).

Інтерактори є компонентами *XR*-сцени, які додаються до ігрових об'єктів (наприклад, *XR Controller*) та ініціюють взаємодію з інтерактивними об'єктами. *XR Interaction Toolkit* підтримує різні типи інтеракторів. *Direct Interactor* використовується для реалізації взаємодії з цифровими *3D*-об'єктами безпосередньо, тобто користувач фізично «торкається» цифрового *3D*-об'єкта або підносить його рукою/контролером. *Ray Interactor* використовується для реалізації взаємодії з цифровими об'єктами (не тільки з *3D*-об'єктами, але й з *UI*-елементами) на відстані за допомогою віртуального променя, який дозволяє наводити, вибирати та активувати інтерактивні цифрові об'єкти без прямого контакту. *Gaze Interactor* використовується для реалізації взаємодії з цифровими об'єктами (*UI*-елементами) на основі відстеження напрямку погляду користувача, без використання контролерів або прямого контакту. *Socket Interactor* використовується для реалізації автоматичного розташування інтерактивних цифрових *3D*-об'єктів у фіксовані місця. *Poke Interactor* використовується для реалізації взаємодії з цифровими об'єктами шляхом фізичного дотику або натискання, наприклад для кнопок та *UI*-елементів у *XR*-середовищі. *Near-Far Interactor* поєднує близьку та дистанційну взаємодію, автоматично перемикаючись між прямим дотиком і взаємодією за допомогою променя залежно від відстані до *3D*-об'єкта.

Інтерактивними об'єктами є об'єкти *XR*-сцени, з якими користувач може взаємодіяти. Інтерактивні об'єкти мають різні стани взаємодії, напр., наведення променя на об'єкт, вибір, захоплення та відпускання, що ініціюються

інтерактором, а *XR Interaction Manager* керує всією системою в *XR*-сцені, зокрема координує, який інтерактор взаємодіє з яким інтерактивним об'єктом і викликає правильні події. Наприклад, компонента *XR Grab Interactable* для ігрового об'єкта дозволяє вибирати, захоплювати, переміщувати та відпускати цей ігровий об'єкт. Компонента *XR Simple Interactable* для ігрового об'єкта дозволяє вибирати або активувати ігровий об'єкт, без захоплення й відпускання (напр., кнопки ігрового вікна).

Фреймворк *AR Foundation* не забезпечує повного набору функцій для розроблення змішаної реальності, проте є базовим інструментом для її реалізації. Для змішаної реальності *AR Foundation* забезпечує відстеження положення та орієнтації *MR*-пристрою в просторі, виявлення поверхонь, визначення точок перетину між *touch*-введенням користувача та поверхнями фізичного середовища на основі *Raycast*-променів, оклюзію та аналіз освітлення для візуальної узгодженості реальних і цифрових об'єктів.

У наступному розділі 3.2 розглянута покрокова інструкція реалізації взаємодії (розміщення, зсув, обертання, масштабування) з цифровим об'єктом у фізичному середовищі для проєктування змішаної реальності на основі фреймворку *AR Foundation*, вибір якого залежав від доступного апаратного забезпечення для *MR*.

3.2 ПОРЯДОК ВИКОНАННЯ

3.2.1 Створення та налаштування *Unity*-проєкту

В *Unity Hub* необхідно обрати *Projects*, натиснути *New Project*, обрати шаблон проєкту *Universal 3D*, дати назву проєкту (напр., *MixedRealitywithUnity*) та натиснути *Create project* (рис. 2.6).

Наступним необхідно обрати платформу для якої буде розроблений *MR*-застосунок (рис. 2.8), встановити та налаштувати *AR Foundation* (рис. 2.9)*, SDK* доповненої реальності (рис. 2.10-2.11) та *XR Plug-in Management* (рис. 2.12-2.13). Пояснення та покрокові інструкції для цього етапу наведено у розділі 2.1.

Останнім необхідно налаштувати *Unity*-проєкт відповідно до платформи розгортання. Опис цього етапу наведено у розділі 2.2.1.

3.2.2 Імпортування та налаштування *MR*-об'єктів

В *Unity*-проєкт спочатку необхідно додати ігрові об'єкти *ARSession* та *XR Origin*, ігровий об'єкт *Main Camera* необхідно видалити (рис. 3.3). Опис цього етапу та порядок його виконання наведено в розділі 2.1.

Для створення сцени змішаної реальності необхідно використати *3D*-об'єкт(и) лабораторного практикуму №1. Експорт *3D*-об'єкта в *Blender* необхідно здійснити через меню *File->Export* з вибором формату *.fbx*, після чого необхідно вибрати каталог збереження, назвати файл та натиснути *Export FBX*.

Наступним необхідно імпортувати файл *3D*-об'єкта та налаштувати матеріал, текстуру та анімацію. Пояснення та покрокові інструкції для цього етапу наведено у розділі 2.2.2. Останнім необхідно створити префаб з налаштованого *3D*-об'єкта, а сам ігровий об'єкт видалити з вікна *Hierarchy*.

3.2.3 Проєктування змішаної реальності на основі *AR Foundation*

У цьому розділі розглядається проєктування змішаної реальності, що ґрунтується на технології відстеження горизонтальних та вертикальних поверхонь у фізичному середовищі, розміщенні цифрового об'єкта на цих поверхнях та взаємодії з цифровими об'єктами (розміщення, зсув, обертання, масштабування) шляхом жестів *touch*-введення користувача, а також технології часткового або повного перекривання цифрового *3D*-об'єкта різними об'єктами фізичного середовища (оклюзія).

Для проєктування змішаної реальності використовується фреймворк *AR Foundation* та *SDK* доповненої реальності відповідно до платформи розгортання. *AR Foundation* забезпечує взаємодію цифрових об'єктів із фізичним середовищем шляхом відстеження поверхонь, просторового позиціонування та використання *Raycast*-променів для визначення точок

перетину між *touch*-введенням користувача та поверхнями фізичного середовища. Оброблення *touch*-введення користувача реалізується за допомогою відповідних скриптів. *AR Foundation* забезпечує оклюзію, але підтримка оклюзії є обмеженою та залежить від апаратних можливостей пристрою та платформи, зокрема вона доступна на пристроях з *LiDAR*-сенсорами (наприклад, *iPad Pro* та *iPhone* серій *Pro*) або на окремих *Android*-пристроях, що підтримують *ARCore Depth API*.

Спочатку до ігрового об'єкта *XR Origin* необхідно додати компоненту *AR Plane Manager*, для параметра *Detection Mode* обрати значення *Everything*. У вікні *Hierarchy* додати ігровий об'єкт *AR Default Plane*, створити його префаб та додати створений префаб для параметра *Plane Prefab* компоненти *AR Plane Manager*. Ігровий об'єкт *AR Default Plane* необхідно видалити у вікні *Hierarchy* (рис. 3.4). Опис цього етапу та порядок його виконання наведено в розділі 2.2.4.

Наступним необхідно додати компоненту *AR Raycast Manager* до ігрового об'єкта *XR Origin* та додати префаб *3D*-об'єкта для параметра *Raycast Prefab* (рис. 3.4).

Наступним необхідно створити та додати до ігрового об'єкта *XR Origin* скрипт-компоненту *AR Place Object* для визначення точок перетину *Raycast*-променів з відстежуваними фізичними поверхнями, а також розміщення та зсув *3D*-об'єкта у знайденій точці, керування затримкою між дотиками (рис. 3.4).

Програмний код скрипт-компоненти *AR Place Object* (мова *C#*):

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

public class ARPlaceObject : MonoBehaviour
{
    // Змінна raycastManager – це компонента AR Foundation, яка використовує Raycast-
промені для визначення точок перетину у фізичному середовищі
    [SerializeField] private ARRaycastManager raycastManager;
    bool isPlacing = false;

    // Додаємо змінну для збереження об'єкта
    private GameObject placedObject;
    // Якщо ARRaycastManager не призначений, нічого не відбувається
```

```
    void Update()
    {
        if (!raycastManager) return;

    // Відстеження дотику на телефоні або кліку мишею в Unity Editor, якщо зараз об’єкт ще не розміщується (!isPlacing)
        if ((Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began ||
            Input.GetMouseButtonDown(0)) && !isPlacing)
        {
            isPlacing = true;
Vector2 touchPos = (Input.touchCount > 0) ? Input.GetTouch(0).position :
(Vector2)Input.mousePosition;
            PlaceOrMoveObject(touchPos);
        }
    }

    void PlaceOrMoveObject(Vector2 touchPosition)
    {

    // Визначає Raycast-промені та їх точки перетину з поверхнями фізичного середовища
        var rayHits = new List<ARRaycastHit>();
        raycastManager.Raycast(touchPosition, rayHits, TrackableType.AllTypes);

    // Якщо точка перетину знайдена, отримуються дані положення та орієнтації, в точці перетину створюється префаб raycastManager.raycastPrefab, який задано в ARRaycastManager у вікні Inspector
        if (rayHits.Count > 0)
        {
            Vector3 hitPosePosition = rayHits[0].pose.position;
            Quaternion hitPoseRotation = rayHits[0].pose.rotation;
            if (placedObject == null)
            {
                // Якщо об’єкта ще немає — створюємо його
                placedObject = Instantiate(raycastManager.raycastPrefab,
hitPosePosition, hitPoseRotation);
            }
            else
            {
                // Якщо об’єкт вже є — просто переміщуємо його
                placedObject.transform.position = hitPosePosition;
                placedObject.transform.rotation = hitPoseRotation;
        }
        }

    // Після 0.25 секунди затримки можна знову ставити об’єкт
        StartCoroutine(SetIsPlacingToFalseWithDelay());
    }

    IEnumerator SetIsPlacingToFalseWithDelay()
    {
        yield return new WaitForSeconds(0.25f);
        isPlacing = false;
    }
```

}

Для параметра *Raycast Manager* скрипт-компоненти *AR Place Object* необхідно додати ігровий об’єкт *XR Origin* (рис. 3.4).

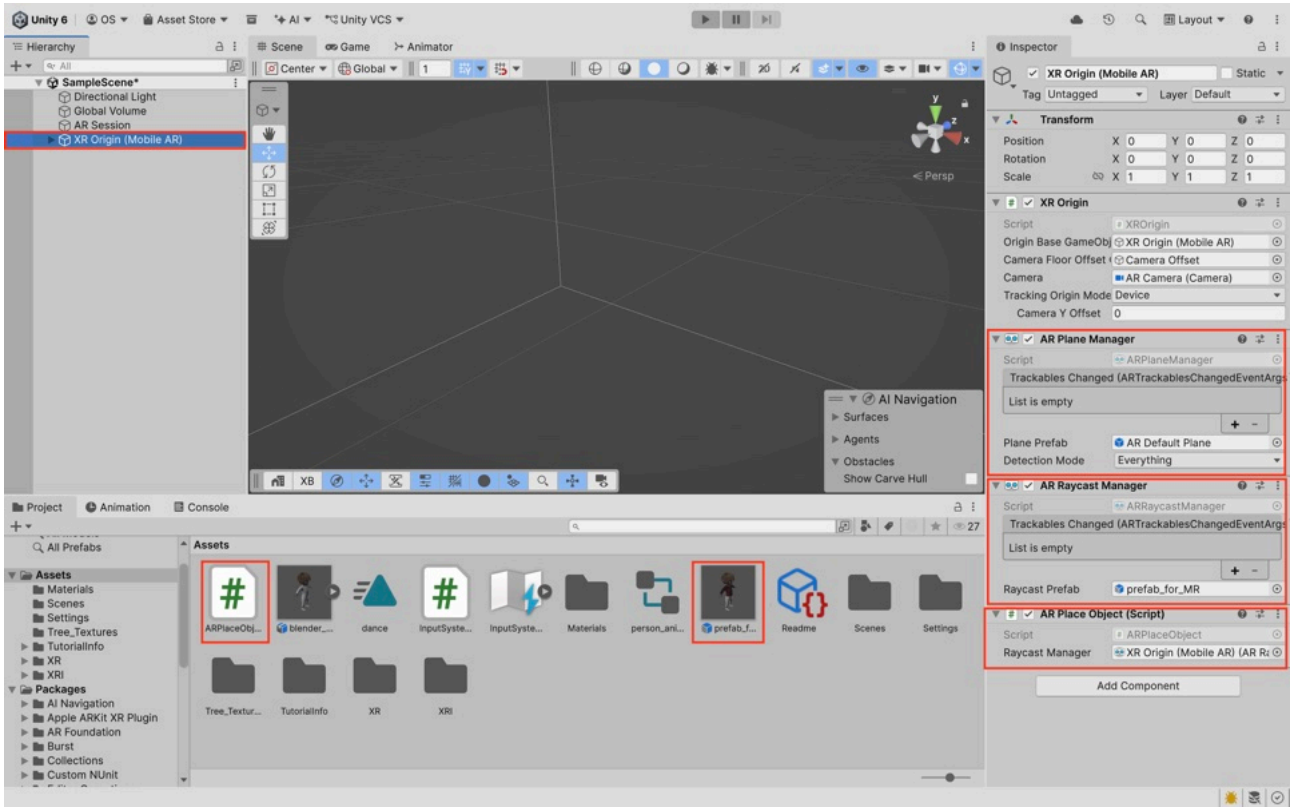


Рис. 3.4 – Налаштування ігрового об’єкта *XR Origin*

Для реалізації оклюзії необхідно до ігрового об’єкта *AR Camera* (дочірній ігровий об’єкт *XR Origin*) додати компоненту *AR Occlusion Manager*. Спочатку необхідно обрати *Add Component-> XR-> AR Foundation-> AR Occlusion Manager* та налаштувати параметри компоненти, зокрема якість генерування текстури глибини фізичного середовища (*Environment Depth Mode*), використання фільтрації для зменшення шуму та більш гладкої оклюзії (*Temporal Smoothing*), застосування оклюзії для фізичних об’єктів складної геометричної форми – людей та її якість (*Human Segmentation Depth Mode*, *Human Segmentation Stencil Mode*), пріоритет оклюзії, зокрема для фізичних об’єктів або людей (рис. 3.5). Також необхідно налаштувати параметри компоненти *AR Camera Manager*, яка додається автоматично до ігрового об’єкта *AR Camera* (рис. 3.5).

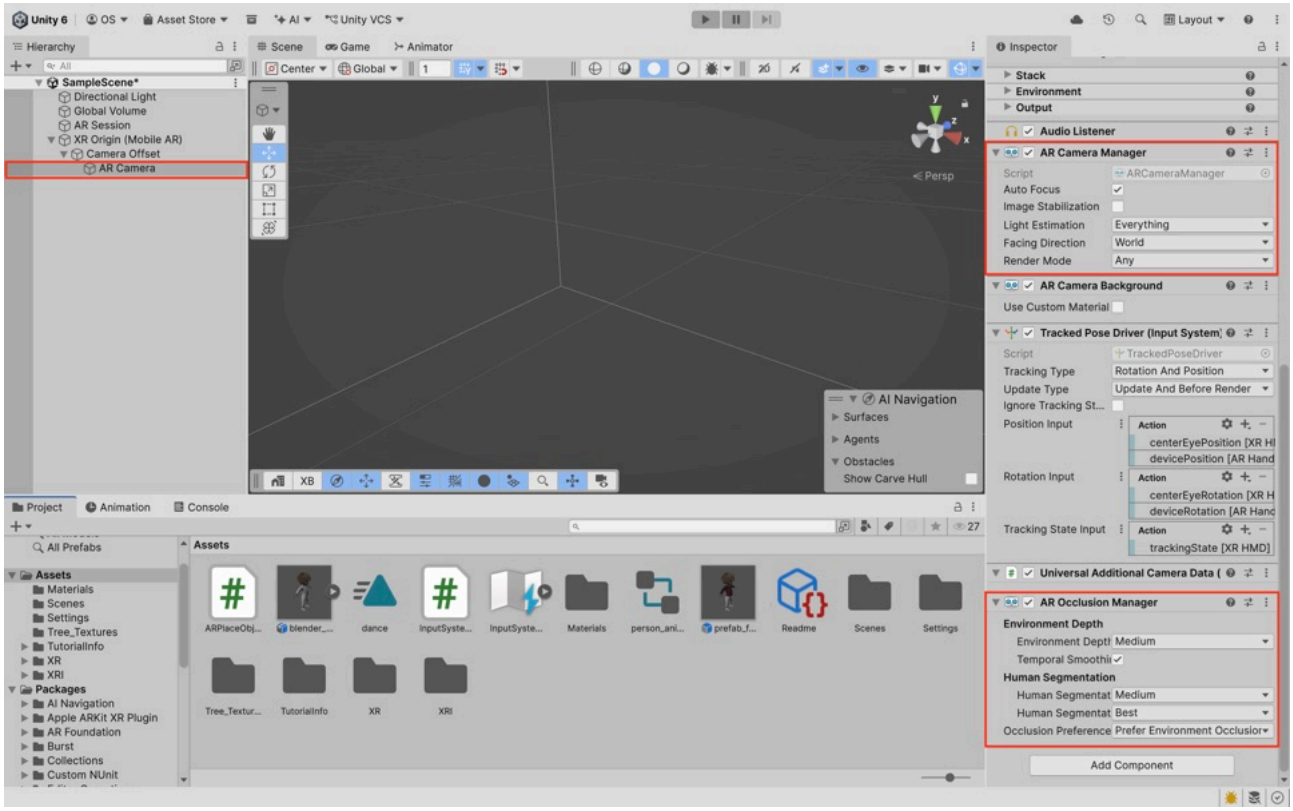


Рис. 3.5 – Налаштування ігрового об’єкта *AR Camera*

Останнім необхідно реалізувати взаємодію з цифровим *3D*-об’єктом (зсув, обертання, масштабування) шляхом жестів *touch*-введення користувача. Для цього необхідно створити скрипт-компоненту для префабу *3D*-об’єкта та у створеному файлі необхідно написати програмний код (мова *C#*):

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using System.Collections.Generic;

// Скрипт для маніпуляцій об'єктом у MR: переміщення, обертання та масштабування
public class ARObjectManipulator : MonoBehaviour
{
// Компонент для AR Raycast (пошук поверхонь)
    private ARRaycastManager raycastManager;
// Камера AR, яка випромінює Raycast-промінь
    private Camera arCamera;

// Початкова відстань між пальцями для масштабування
    private float initialDistance;
// Початковий масштаб об'єкта для масштабування
    private Vector3 initialScale;

    void Start()
    {
```

```
// Знаходимо ARRaycastManager на сцені
        raycastManager = FindObjectOfType<ARRaycastManager>();
// Беремо основну камеру AR
        arCamera = Camera.main;
    }

    void Update()
    {
// Якщо один дотик – переміщуємо об'єкт
        if (Input.touchCount == 1)
        {
            MoveObject(Input.GetTouch(0));
        }
// Якщо два дотики – обертаємо та масштабуємо об'єкт
        else if (Input.touchCount == 2)
        {
            RotateAndScale();
        }
    }

// Реалізація переміщення об’єкта
    void MoveObject(Touch touch)
    {
// Рухаємось тільки якщо дотик рухається
        if (touch.phase != TouchPhase.Moved) return;

// Створюємо промінь від камери через точку дотику на екрані
        Ray ray = arCamera.ScreenPointToRay(touch.position);

// Перевіряємо, чи промінь потрапив на об'єкт
        if (Physics.Raycast(ray, out RaycastHit hit))
        {
            if (hit.transform == transform)
            {
// Створюємо список для зберігання результатів AR Raycast
                List<ARRaycastHit> hits = new List<ARRaycastHit>();

// Виконуємо AR Raycast по поверхні
                raycastManager.Raycast(touch.position, hits,
TrackableType.PlaneWithinPolygon);

// Якщо знайшли поверхню – переміщаємо об'єкт туди
                if (hits.Count > 0)
                {
                    transform.position = hits[0].pose.position;
                }
            }
        }
    }

// Реалізація обертання та масштабування
    void RotateAndScale()
    {
```

```
        Touch t0 = Input.GetTouch(0);
        Touch t1 = Input.GetTouch(1);

// Поточна відстань між пальцями
        float currentDistance = Vector2.Distance(t0.position, t1.position);

        if (t1.phase == TouchPhase.Began)
        {

// Запам'ятовуємо початкову відстань та масштаб об'єкта
            initialDistance = currentDistance;
            initialScale = transform.localScale;
        }
        else
        {

// Вираховуємо коефіцієнт масштабування
            float scaleFactor = currentDistance / initialDistance;

// Застосовуємо новий масштаб
            transform.localScale = initialScale * scaleFactor;
        }

// Вектор між пальцями у попередньому кадрі
        Vector2 prevDir = (t0.position - t0.deltaPosition) - (t1.position -
t1.deltaPosition);

// Вектор між пальцями у поточному кадрі
        Vector2 currDir = t0.position - t1.position;

// Обчислюємо кут між попереднім і поточним положенням пальців
        float angle = Vector2.SignedAngle(prevDir, currDir);

// Множник для збільшення швидкості обертання
        float rotationMultiplier = 5.0f;

// Обертаємо об'єкт по трьох осях одночасно
        transform.Rotate(Vector3.up, angle * rotationMultiplier, Space.World);
        transform.Rotate(Vector3.right, angle * rotationMultiplier, Space.World);
        transform.Rotate(Vector3.forward, angle * rotationMultiplier, Space.World);
    }
}
```

Скрипт *ARObjectManipulator* реалізує базову взаємодію користувача з *3D*-об’єктом у середовищі змішаної реальності. Він забезпечує за допомогою *touch*-введення розміщення цифрового об’єкта на виявленій фізичній поверхні, зсув, обертання та масштабування об’єкта в тривимірному просторі. Розміщення цифрового об’єкта відбувається за допомогою *tap*-дотику користувача. Коли користувач торкається екрана, на виявленій поверхні розміщується *3D*-об’єкт,

попередній об’єкт переміщується на нову позицію, якщо він уже існує. Зсув цифрового об’єкта відбувається за допомогою *drag*-дотику користувача. Обертання цифрового об’єкта відбувається за допомогою *twist*-дотику по всіх трьох осях одночасно. Масштабування цифрового об’єкта відбувається за допомогою *pinch*-дотику.

3.2.4 Розроблення ігрового вікна

Для розроблення ігрового вікна з метою опису *touch*-введення для взаємодії з цифровим *3D*-об’єктом необхідно створити ігровий об’єкт *Canvas* в *Unity*-проєкті. Спочатку необхідно натиснути правою кнопкою миші у вікні *Hierarchy* та обрати *UI(Canvas)-> Canvas,* ігровий об’єкт *EventSystem* створиться автоматично. Основними параметрами ігрового об’єкта *Canvas* є режим рендерингу, розташування та розміри (рис. 3.6), які визначають спосіб відображення та позиціонування *UI*-елементів у *XR*-сцені.

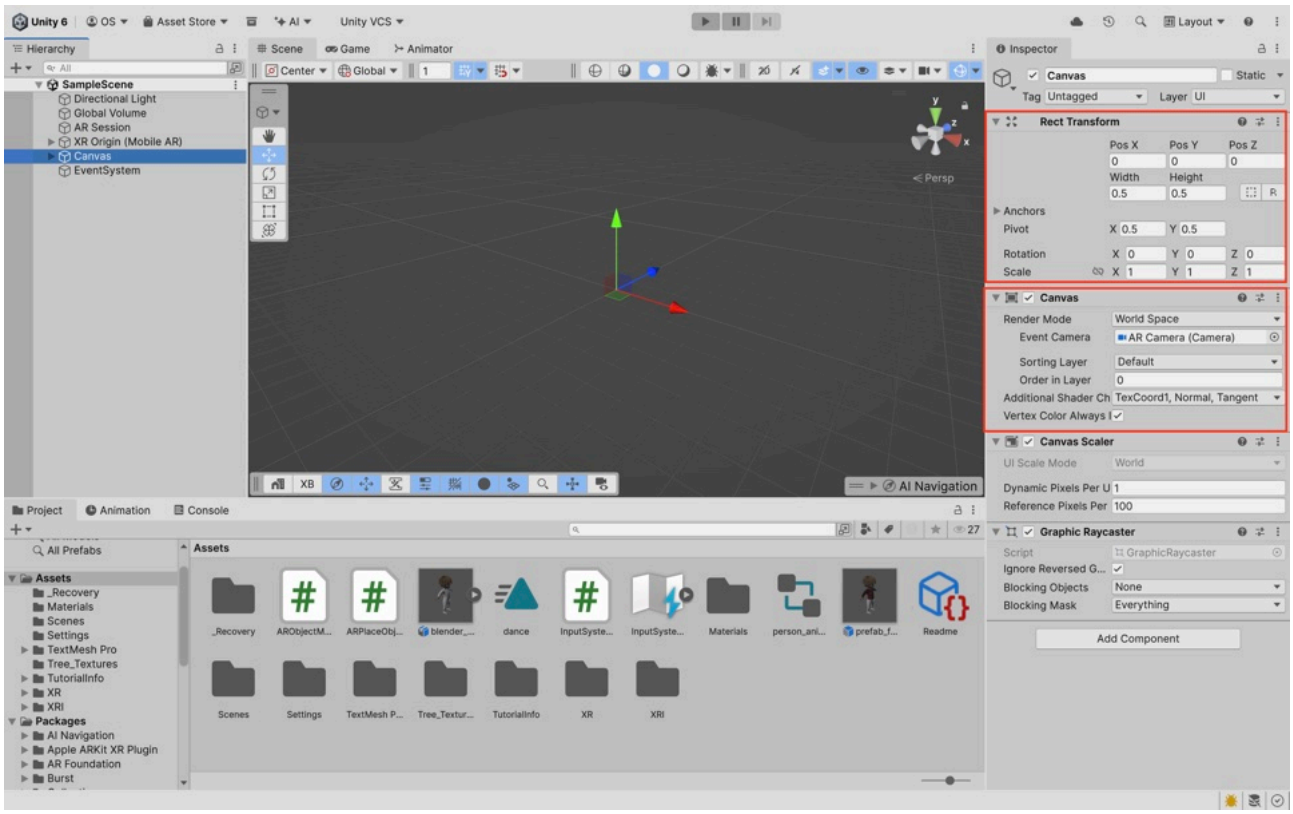


Рис. 3.6 – Налаштування ігрового об’єкта *Canvas*

Наступним необхідно створити дочірній ігровий об’єкт *Image* для конфігурування фону ігрового меню. У вікні *Hierarchy* натиснути правою кнопкою миші та обрати *UI(Canvas)-> Image* (рис. 3.7).

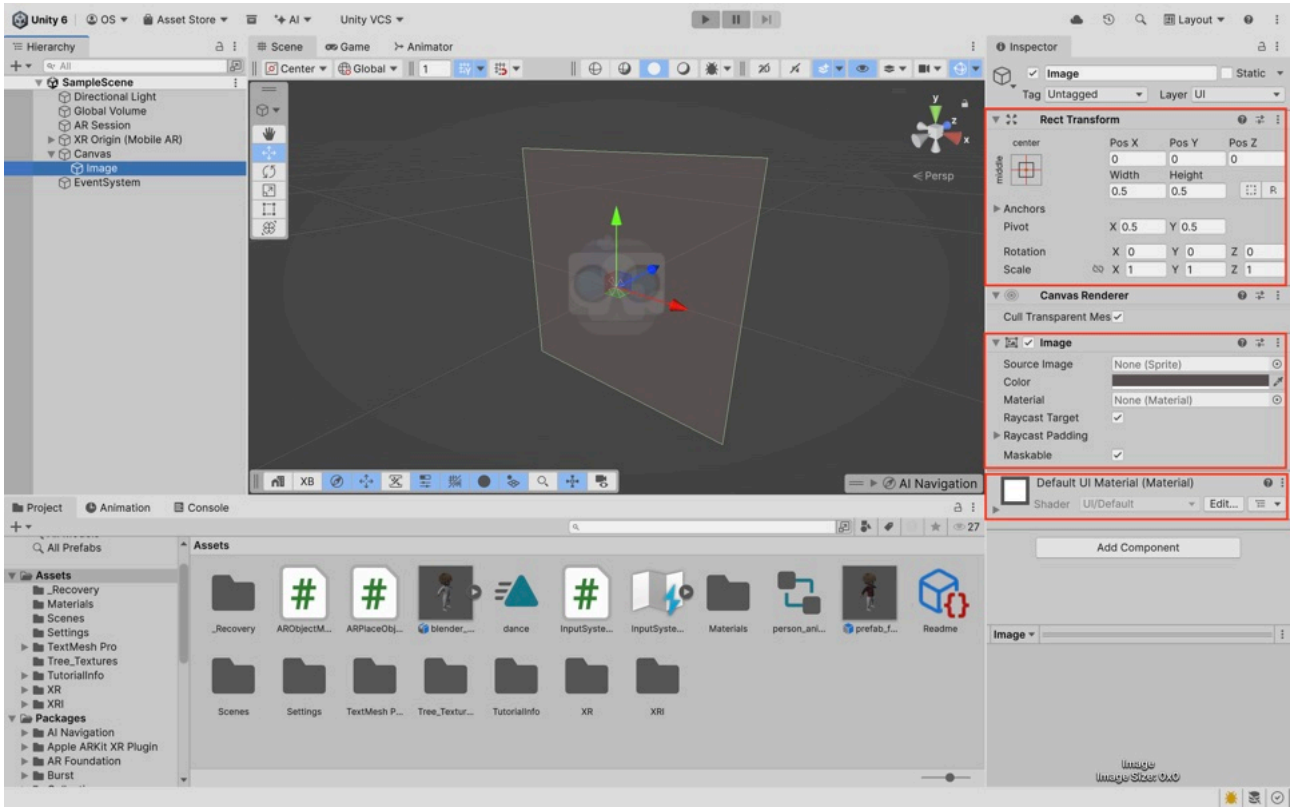


Рис. 3.7 – Налаштування ігрового об’єкта *Image*

Наступним необхідно створити дочірні ігрові об’єкти типу *Button*, при натисканні на які в режимі *Play Mode* виконуються різні події, зокрема відображення інформації про правила взаємодії з цифровим *3D*-об’єктом та закриття ігрового вікна. Ігрові об’єкти типу *Button* міститимуть текстові підписи, тому спочатку необхідно встановити та налаштувати інструмент для конфігурування тексту.

Для завантаження в *Unity* інструмента конфігурації тексту необхідно обрати *Window-> Package Management-> Asset Store* та написати пошуковий запит, напр., *TextMesh Pro* (рис. 3.8).

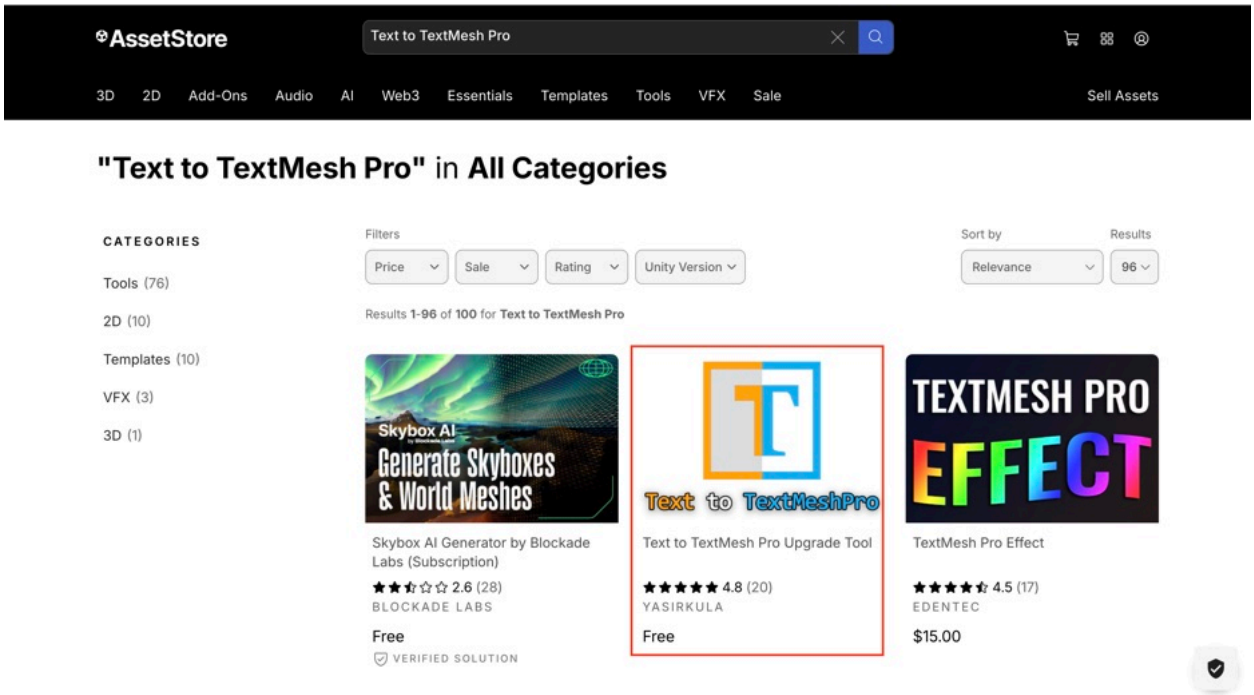


Рис. 3.8 – Інструменти конфігурації тексту для *Unity*-проєктів

Наступним необхідно обрати інструмент конфігурації тексту та відкрити в *Unity*-середовищі (рис. 3.9), натиснути *Download*, а потім натиснути *Import* для імпортування інструмента до поточного *Unity*-проєкту (рис. 3.10).

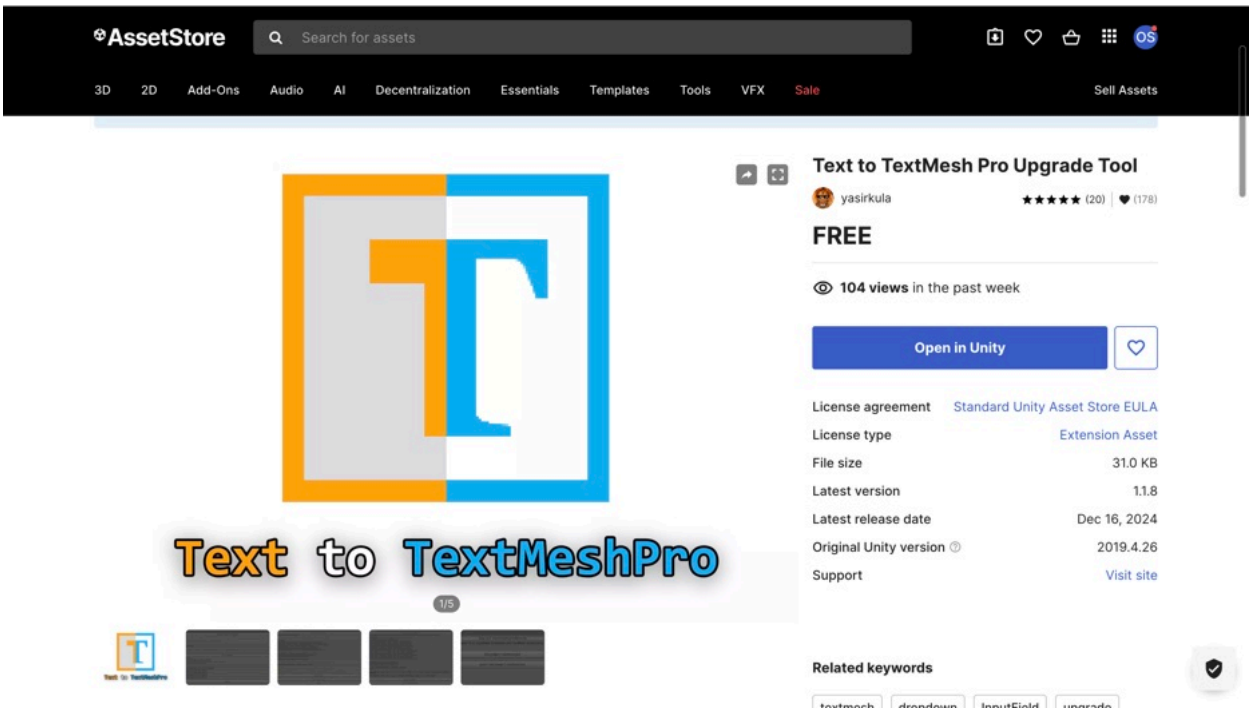


Рис. 3.9 – Інструмент для конфігурування тексту *Text to TextMesh Pro Upgrade Tool*

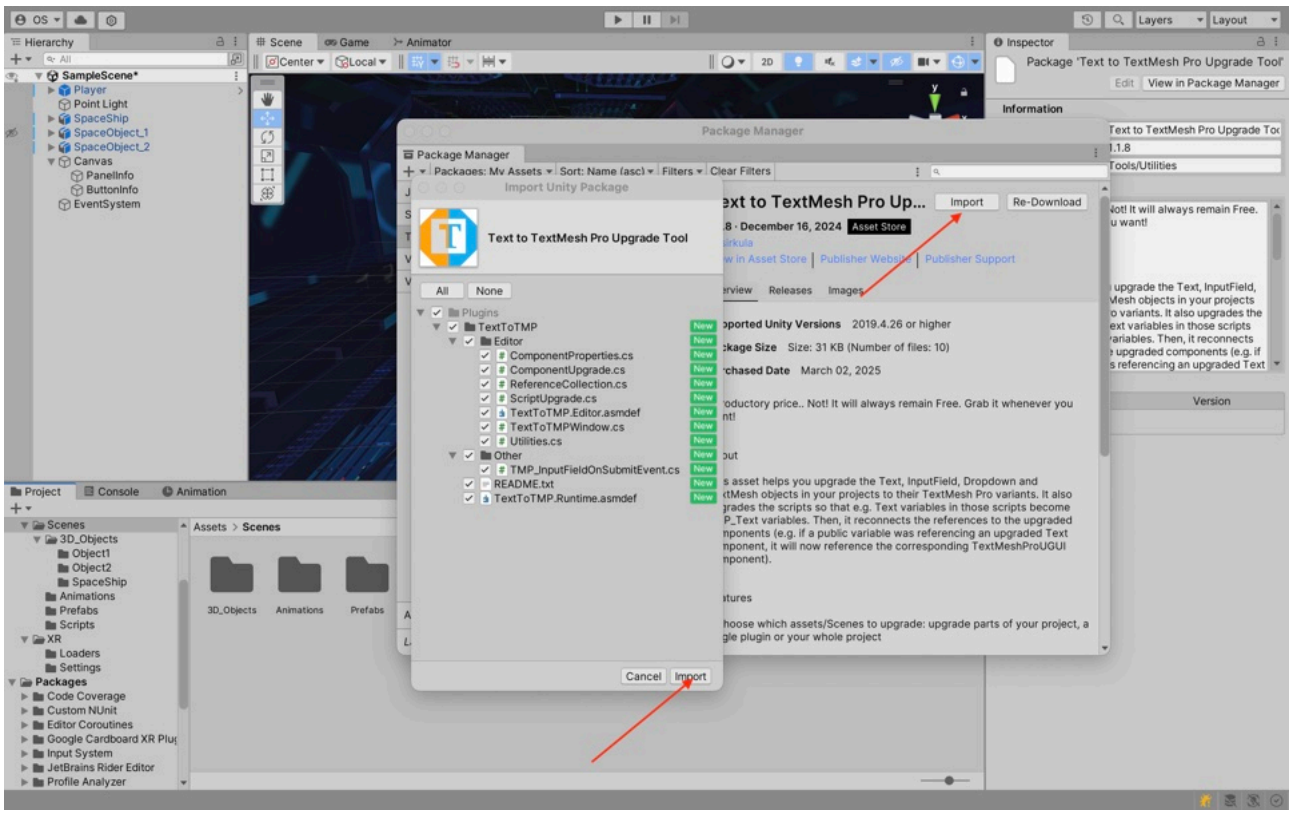


Рис. 3.10 – Імпортування *Text to TextMesh Pro Upgrade Tool* в *Unity*-проєкт

Якщо інструмент конфігурування тексту використовувався раніше в *Unity*-проєктах, необхідно обрати *Window-> Package Management-> Package Manager- > Packages: My Assets*, обрати відповідний інструмент та натиснути *Import*. В результаті *Text to TextMesh Pro Upgrade Tool* буде встановлено.

Останнім необхідно прийняти оновлення *Text to TextMesh Pro Upgrade Tool*, якщо будуть запропоновані або у вікні *Project* обрати *Packages-> TextMeshPro-> Package Resources* та імпортувати *TMP Essentials* та *TMP Examples & Extras* в *Unity*-проєкт.

Для додавання ігрового об’єкта типу *Button* у вікні *Hierarchy* необхідно натиснути правою кнопкою миші та обрати *UI(Canvas)-> Button-TextMeshPro*. Основними параметрами ігрового об’єкта *Button* є режим розташування, розміри, колір та стиль, події та текстовий дочірній ігровий об’єкт для підпису (рис. 3.11). Основними параметрами текстового об’єкта є сам текстовий вміст, шрифт, розмір, колір тощо.

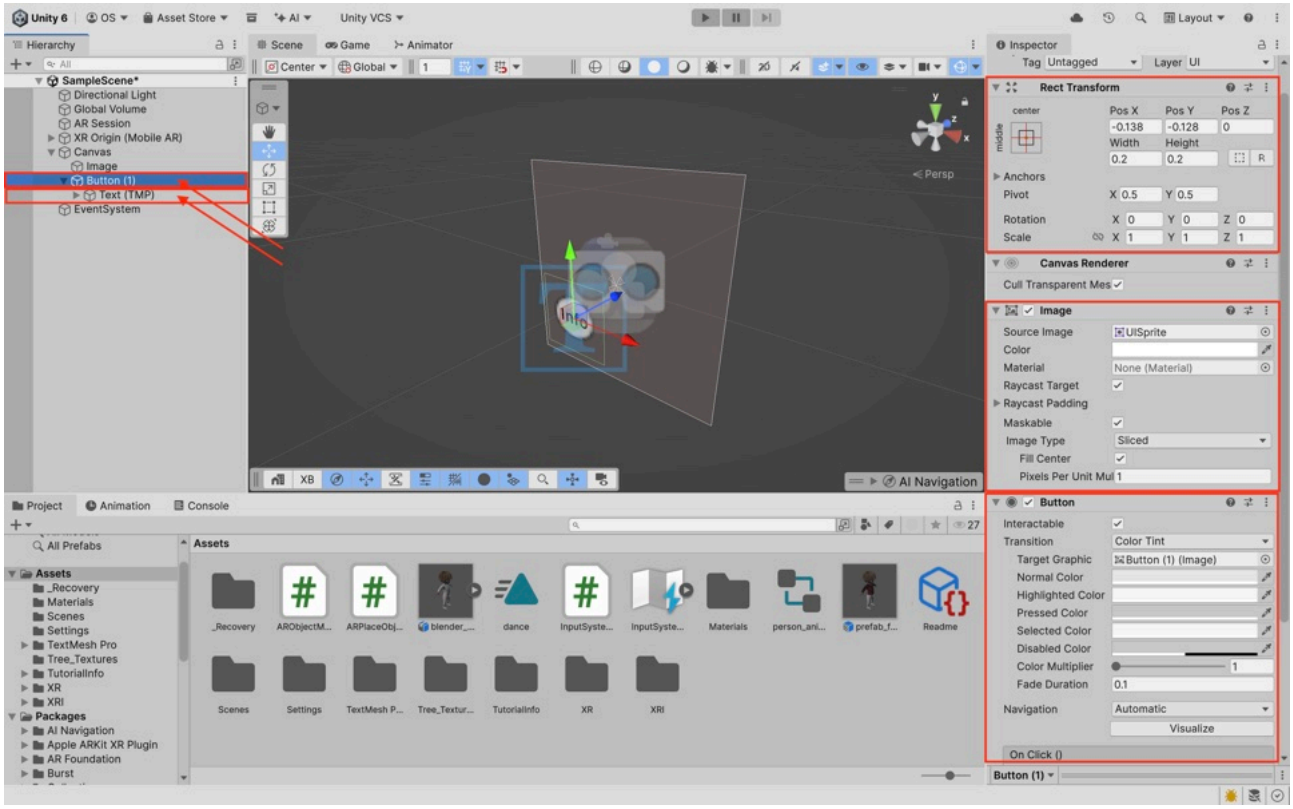


Рис. 3.11 – Налаштування ігрового об’єкта *Button*

Необхідно створити два ігрові об’єкти типу *Button*: один для відображення інформації, а другий – для закриття ігрового вікна.

В *Unity* для створення подій необхідно створити пустий ігровий об'єкт, щоб він виступав як контейнер для скриптів, пов’язаних з кнопкою. Для цього у вікні *Hierarchy* необхідно натиснути правою кнопкою миші та обрати *Create Empty* та назвати, напр., *UIHandler* (рис. 3.12). Наступним необхідно створити скрипт-компоненту для ігрового об’єкта *UIHandler* та у створеному файлі необхідно написати програмний код (мова *C#*):

```
using UnityEngine;
using TMPro;

// Клас для обробки подій, пов’язаних з кнопками UI
public class WindowEvents : MonoBehaviour
{
// Посилання на текстовий компонент TextMeshPro, у якому буде відображатися інформація
    public TMP_Text text;

// Посилання на ігровий об’єкт вікна (Canvas), яке необхідно відкривати або закривати
    public GameObject window;
```

```
// Метод, який викликається при натисканні кнопки та встановлює текст з інструкціями для користувача
    public void ClickButton()
    {
        text.text =
            "Tap touch – to place the object.\n" +
            "Drag touch – to move (translate) the object.\n" +
            "Twist touch – to rotate the object.\n" +
            "Pinch touch – to scale the object.";
    }

// Метод для закриття ігрового вікна шляхом вимкнення відповідного ігрового об’єкта
    public void CloseWindow()
    {
        window.SetActive(false);
    }
}
```

Наступним кроком необхідно створити дочірній ігровий об’єкт *Text*, за допомогою якого відображатиметься інформація щодо взаємодії з цифровим *3D*-об’єктом. Для цього у вікні *Hierarchy* необхідно натиснути правою кнопкою миші та обрати *UI(Canvas)-> Text- TextMeshPro*. Основними параметрами ігрового об’єкта *Text* є текстовий вміст, шрифт, розмір, колір тощо (рис. 3.12).

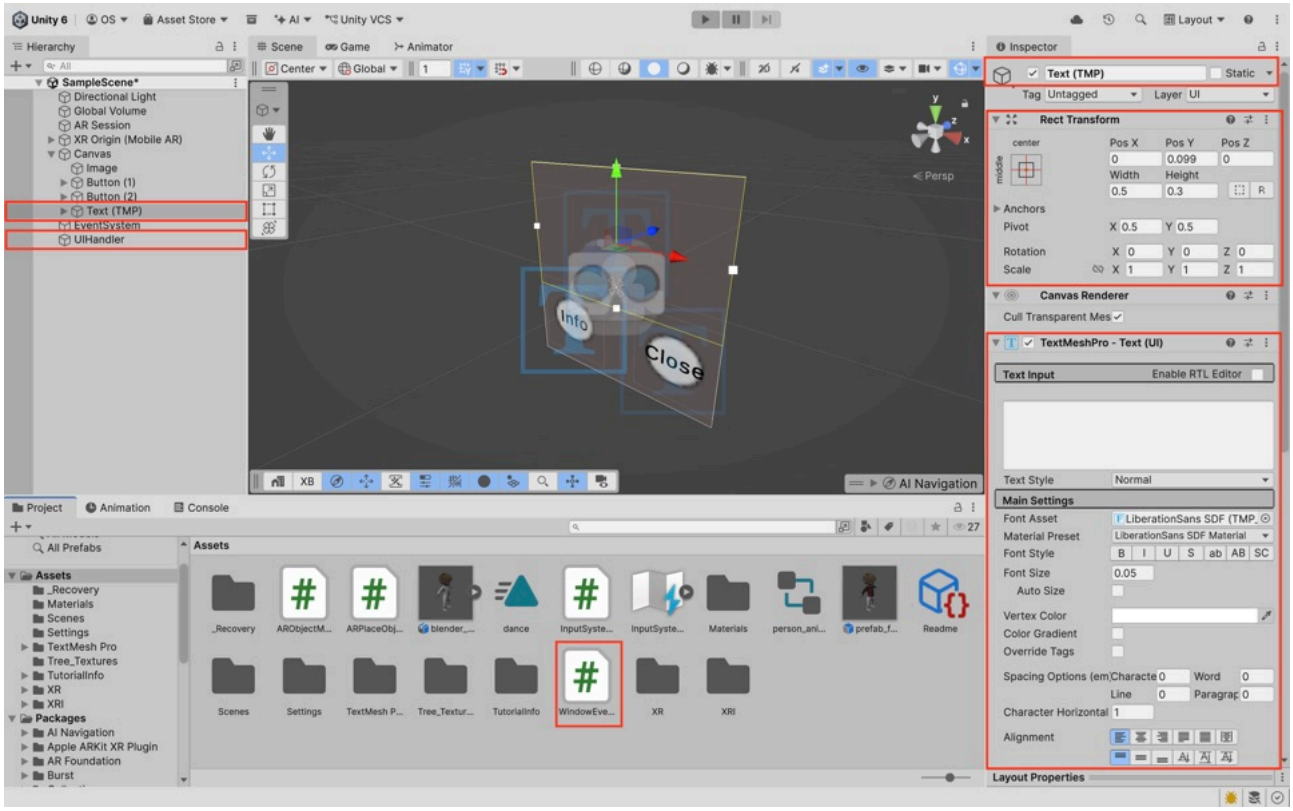


Рис. 3.12 – Налаштування ігрового об’єкта *Text*

Наступним необхідно налаштувати параметри скрипт-компоненти *Window Events* ігрового об’єкта *UIHandler*, зокрема додати ігрові об’єкти *Text* та *Canvas* (рис. 3.13).

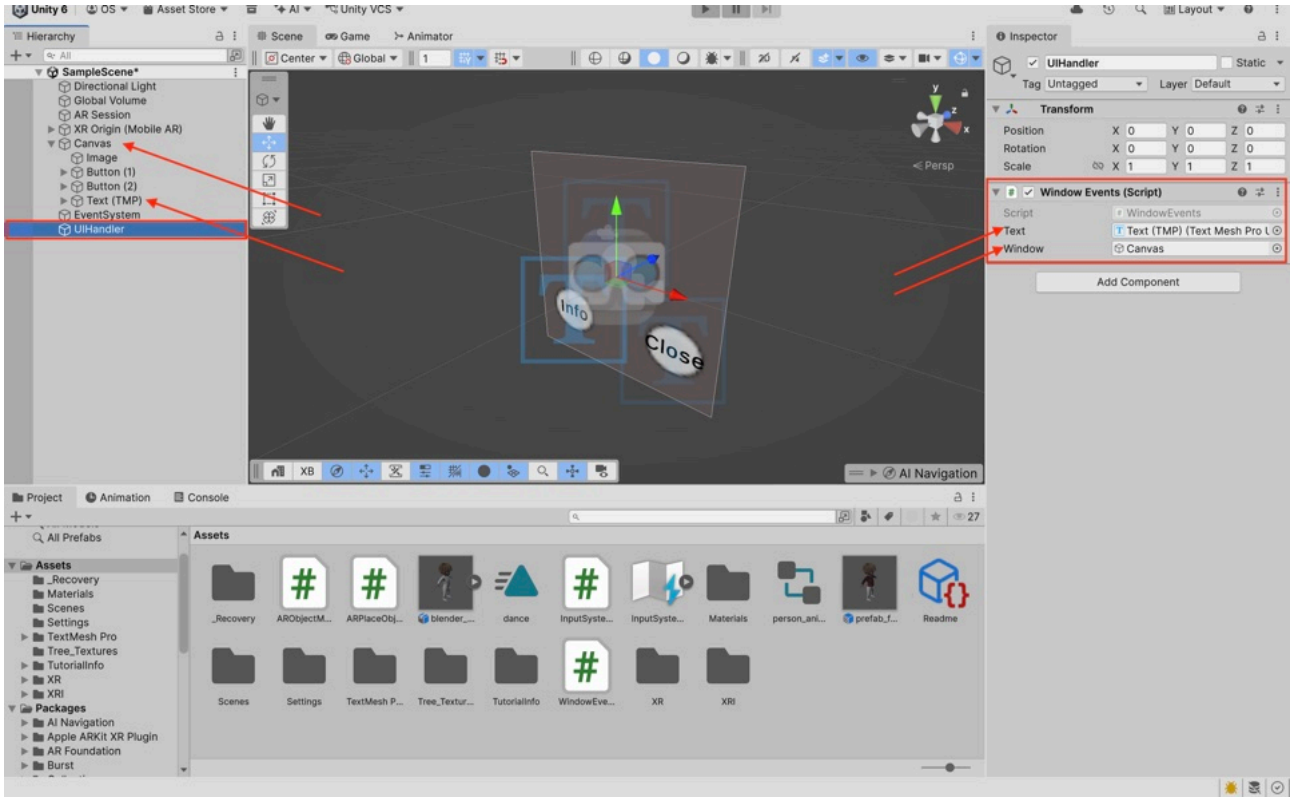


Рис. 3.13 – Налаштування ігрового об’єкта *UIHandler*

Останнім необхідно налаштувати події для ігрових об’єктів *Button.* У вікні *Hierarchy* необхідно вибрати відповідний ігровий об’єкт *Button*, після чого у вікні *Inspector* для компоненти *Button* додати подію *OnClick()*, перетягнути ігровий об’єкт *UIHandler* зі скриптом обробки подій та обрати відповідний публічний метод (наприклад, *WindowEvents-> ClickButton()* або *WindowEvents-> CloseWindow()*), що забезпечить коректну взаємодію кнопки з користувачем (рис. 3.14 та 3.15).

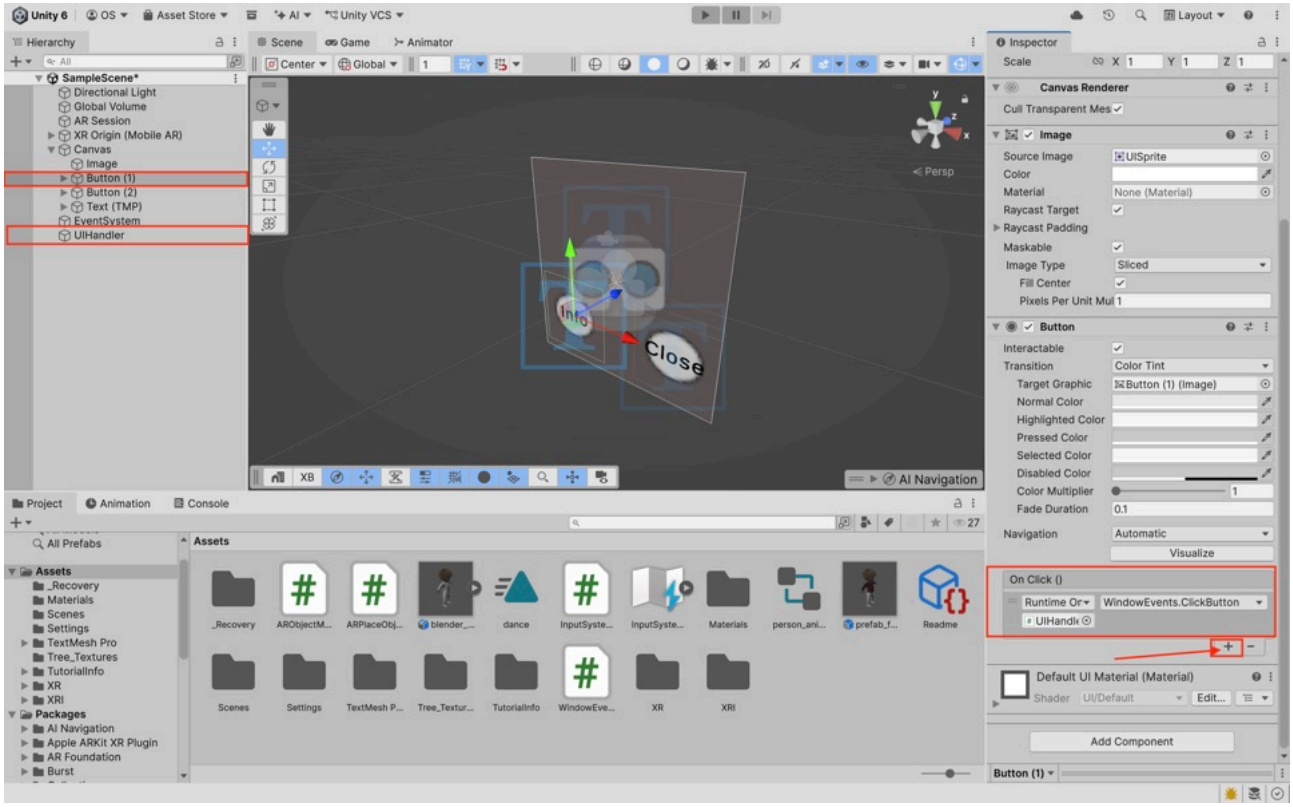


Рис. 3.14 – Налаштування ігрового об’єкта *Button(1)*

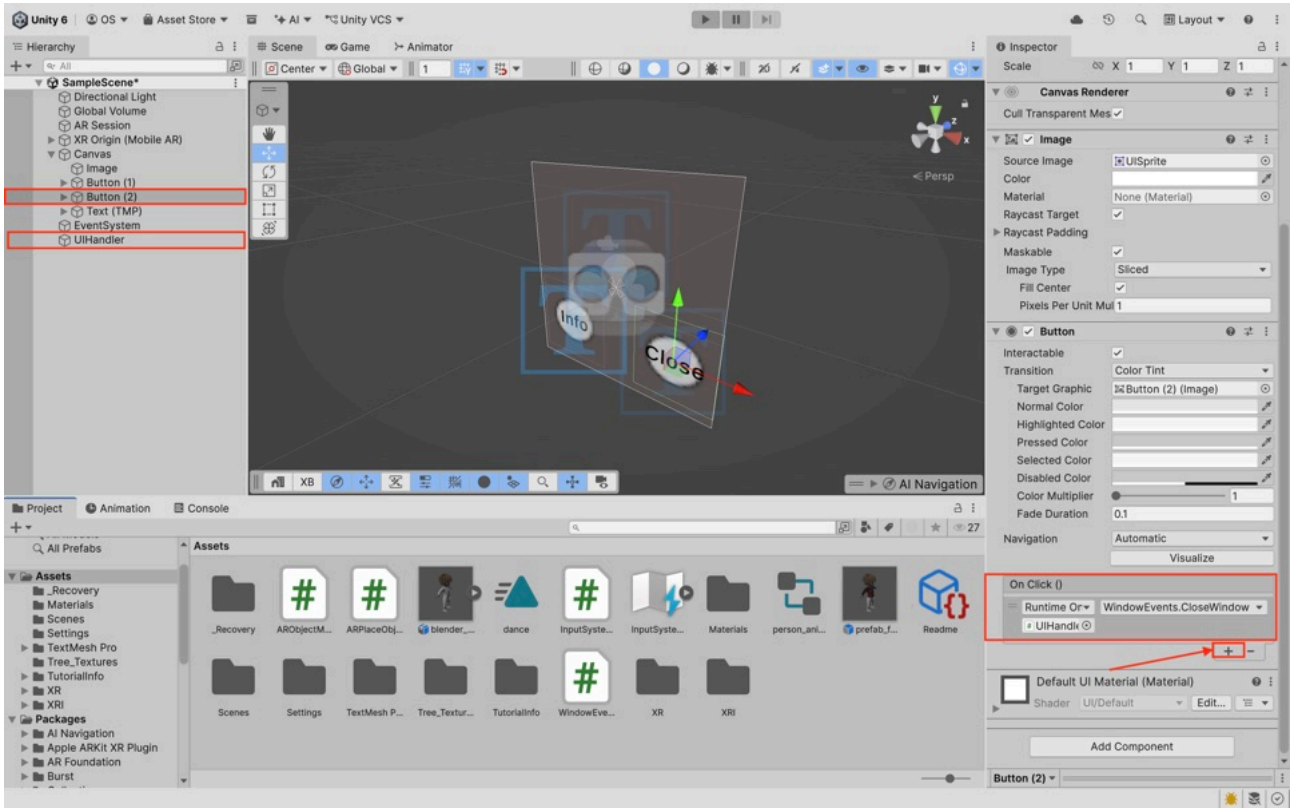


Рис. 3.15 – Налаштування ігрового об’єкта *Button(2)*

3.2.5 Збирання та розгортання *MR*-застосунку

Процес збирання та розгортання *MR*-застосунку в середовищі *Unity* є однаковим в порівнянні з процесом збирання та розгортання *AR*-застосунку для *Apple ARKit XR Plugin* та *Google ARCore XR Plugin*. Використання фреймворку *AR Foundation* та *XR Interaction Toolkit API* забезпечує уніфікований підхід до розроблення *XR*-досвідів незалежно від цільової платформи. Пояснення та покрокові інструкції для цього етапу наведено у розділі 2.2.5.

3.2.6 Демонстрація роботи *MR*-застосунку

Результат роботи *MR*-застосунку представлено на рис. 3.16.

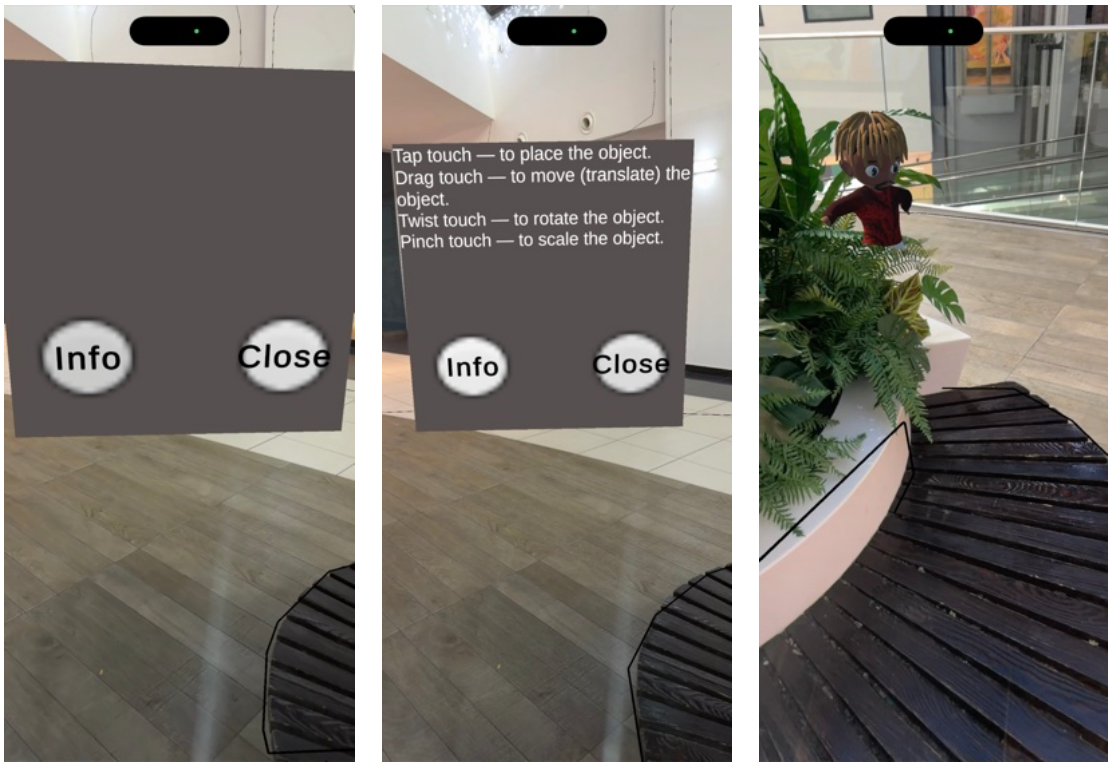


Рис. 3.16 – Приклад роботи *MR*-застосунка

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які основні *Unity*-фреймворки використовуються для реалізації взаємодії в *MR*-застосунках і яку роль вони виконують?

2. Які типи взаємодії з цифровими об’єктами можна реалізувати в *MR*-середовищі за допомогою фреймворку *AR Foundation* та *XR Interaction Toolkit API*?

3. У чому полягає відмінність між реалізацією взаємодії через сенсорне введення та через *XR Interaction Toolkit*?

4. Які компоненти *AR Foundation* використовуються для реалізації *MR*-середовища?

5. Як компоненти *ARRaycastManager* та *ARPlaneManager* використовуються для реалізації взаємодії з цифровими об’єктами в *MR*-середовищі?

6. Яке призначення компоненти *AR Occlusion Manager* та як реалізуються її функції?

7. Поняття інтерактор (*Interactor*) в *XR Interaction Toolkit* та яку функцію він виконує в *XR*-сцені?

8. Які типи інтеракторів підтримуються в *XR Interaction Toolkit*?

9. Яке призначення *Direct Interactor* в *XR Interaction Toolkit* і як він використовується для взаємодії з об’єктами?

10. Яке призначення *Ray Interactor* в *XR Interaction Toolkit* і як він використовується для взаємодії з об’єктами?

11. Чим відрізняється *Left/Right Controller* від *Interactor* в *XR Interaction Toolkit* та яку роль кожен з них виконує у взаємодії з об’єктами?

12. Які основні типи інтерактивних об’єктів підтримуються в *XR Interaction Toolkit*?

13. Яке призначення компоненти *XR Grab Interactable* та як реалізуються її функції?

14. Як інтерактор забезпечує взаємодію з інтерактивними об’єктами в *XR Interaction Toolkit*?

15. Яке призначення *XR Interaction Manager* в *XR Interaction Toolkit*?

16. Які події взаємодії обробляє *XR Interaction Manager*?

17. Які основні *UX/UI* інструменти доступні в *Unity*?

18. Для чого використовується система *Canvas* в *Unity*?

19. Яка роль ігрового об’єкта *Event System* у взаємодії з *UI* в *Unity*?

20. Як реалізувати взаємодію користувача з кнопками ігрового меню в *Unity*-проєкті?

ЛАБОРАТОРНИЙ ПРАКТИКУМ №4. РОЗРОБЛЕННЯ ЗАСТОСУНКУ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ В ІГРОВОМУ РУШІЇ *UNITY*

Лабораторний практикум присвячений розробленню мобільного *VR*-застосунку в ігровому рушії *Unity*. Для створення віртуальної реальності на основі *iOS*- та *Android*-платформ використовується *Google Cardboard SDK*.

В розділі «Порядок виконання» розглянутий процес розроблення мобільного *VR*-застосунку на основі *Google Cardboard SDK* з реалізацією відстеження положення та орієнтації голови користувача, а також взаємодії за допомогою *gaze-based* механізму (визначення напрямку погляду).

ЗАВДАННЯ

Розробити *VR*-застосунок в ігровому рушії *Unity*. *VR*-застосунок повинен містити *3D*-об'єкт(и) з лабораторного практикуму №1 (обов'язково), а також мультимедійні дані – *3D*-об'єкти з відкритих сторонніх джерел, аудіо, відео, текст (за бажанням). Для розробленого *3D*-об'єкта(ів) повинна бути реалізована анімація, а також застосовані відповідні текстури та матеріали. У *VR*-застосунку повинна бути реалізована взаємодія користувача з розробленим *3D*-об'єктом(ами). *VR*-застосунок повинен містити цифрове середовище (розроблене або завантажене з відкритих сторонніх джерел) та ігрове вікно з інструкцією щодо правил взаємодії з цифровими об'єктами та/або середовищем.

Звіт з лабораторного практикуму повинен містити:

1. Титульний аркуш, формулювання завдання, текстовий опис процесу розроблення *VR*-застосунку, зокрема інформація щодо використаних інструментів та технологій, фрагменти програмного коду, скриншоти проміжних та ключових результатів (*.pdf* файл).

2. Демонстрацію роботи *VR*-застосунку (*.mp4* файл).

3. Архів програми (*.zip* файл).

4.1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Віртуальна реальність є *XR*-технологією, що створює цифрове середовище, в якому об'єкти, події та реакції на події комп'ютерно-генеруються в режимі реального часу.

Ініціатором події у системі віртуальної реальності є введення користувача, що реалізуються через технології відстеження рухів рук (*hand tracking*), погляду (*gaze tracking*) або контролерів (*VR control tracking*). В *Unity* для реалізації різних механік взаємодії – захоплення об'єктів, натискання кнопок, обертання, переміщення та телепортацію – використовується *XR Interaction Toolkit*, інтерфейс (*API*) для реалізації взаємодії користувача в *XR*-середовищі. В розділі 3.1 викладені основні теоретичні відомості про *XR Interaction Toolkit*. Функції віртуальної реальності реалізовані в платформо-залежних *SDK – PICO SDK* (для *VR*-пристроїв *PICO*), *SteamVR SDK* (для *VR*-пристроїв *HTC Vive*, *Valve Index* та інших), *Vive Wave SDK* (для автономних *VR*-пристроїв *HTC Vive*). Відстеження положення та орієнтації користувача (контролерів, голови, рук, а в окремих *VR*-пристроях – очей), стереоскопічний рендеринг, відтворення просторового звуку, підтримка *XR*-контролерів (зчитування натискань кнопок, джойстиків, генерування вібровідгуку) належать до основних функцій систем віртуальної реальності.

OpenXR є відкритим кросплатформеним *API*-стандартом для розроблення застосунків розширеної реальності (*VR*, *MR*, *AR*) та забезпечує відстеження положення та орієнтації *XR*-пристроїв та користувача, стереоскопічний рендеринг та інші базові функції *XR*. *OpenXR* підтримується сучасними ігровими рушіями (*Unity*, *Unreal Engine*, *Godot*), а також може використовуватись в нативних середовищах розроблення (*Microsoft Visual Studio*, *Android Studio*, *Xcode*) через *OpenXR SDK* із застосуванням мов програмування *C/C++*. *OpenXR* дозволяє створювати застосунки для різних *XR*-пристроїв незалежно від платформи та виробника.

В *Unity OpenXR* підтримується через плагін *OpenXR Plug-in,* який забезпечує взаємодію рушія з *XR*-пристроями. Для завантаження *OpenXR* в

Unity-проєкт спочатку необхідно обрати *Window-> Package Management-> Package Manager*. Далі необхідно обрати реєстр пакетів *Unity-> OpenXR Plugin* та натиснути *Install* (рис. 4.1). Для підключення плагіну в *Unity*-проєкт необхідно обрати *Edit- > Project Settings- > XR Plug-in Management* та обрати *OpenXR* для платформи (рис. 4.2), для якої буде розроблений *VR*-застосунок.

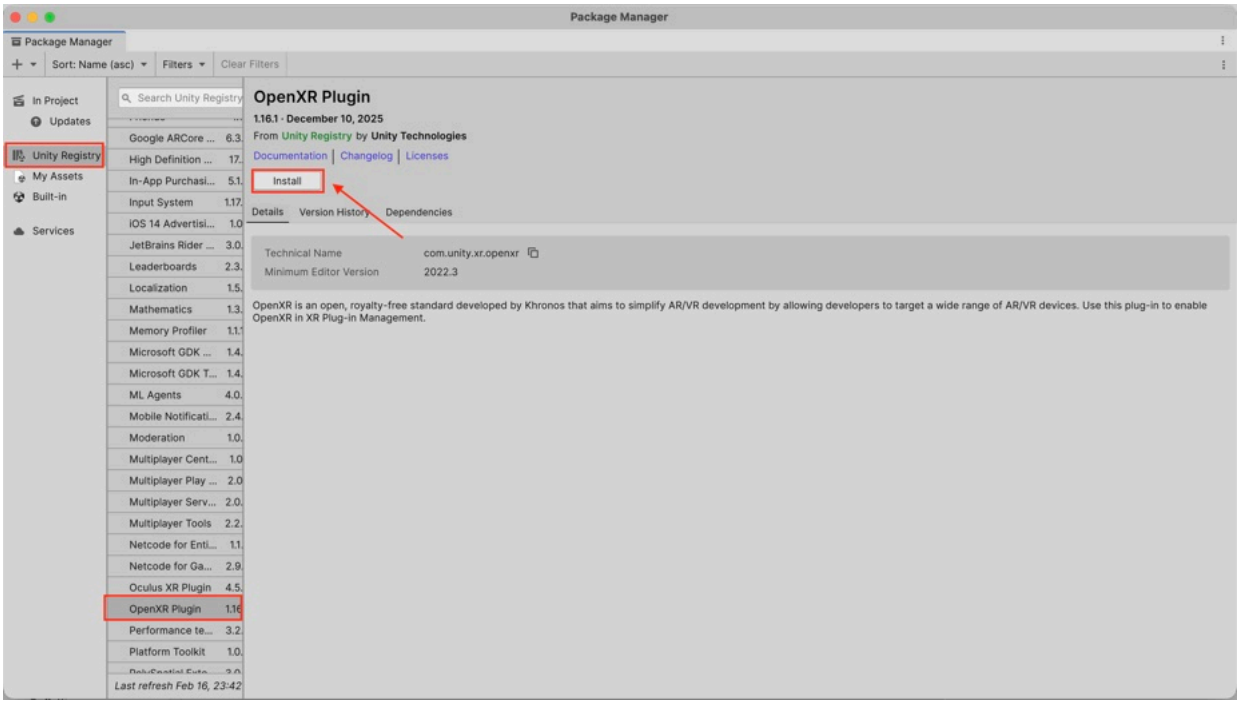


Рис. 4.1 – Завантаження та встановлення *OpenXR Plugin* в *Unity*-проєкт

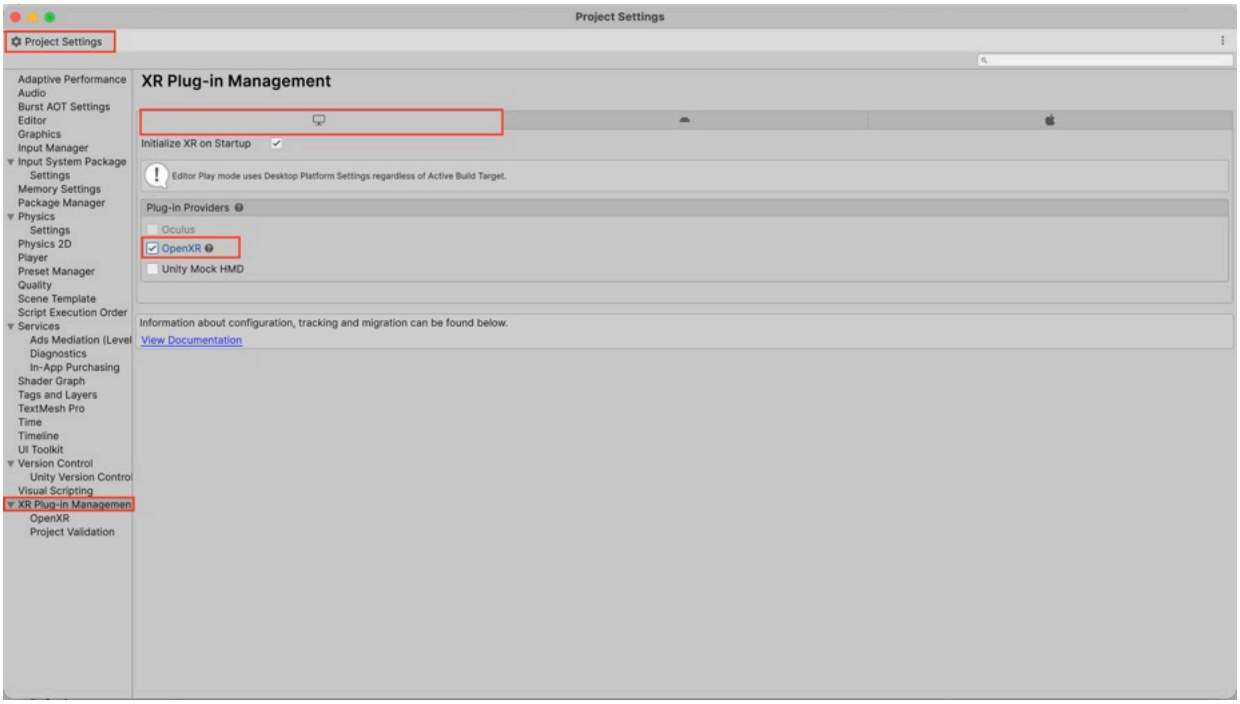


Рис. 4.2 – Підключення плагіну *OpenXR Plugin* в *Unity*-проєкт

Google Cardboard SDK є відкритим кросплатформеним *SDK* мобільної віртуальної реальності та забезпечує відстеження голови користувача, стереоскопічний рендеринг, а також взаємодію через сенсорне введення (натискання на екран) та механізм *gaze-based* (наведення погляду на об'єкти у сцені без натискання). Нативне розроблення з *Google Cardboard SDK* передбачає створення *VR*-застосунків для платформ *Android* (*Java/Kotlin*, *Android Studio*) та *iOS* (*Objective-C/Swift*, *Xcode*) із використанням бібліотек *Google Cardboard SDK*.

В *Unity Cardboard SDK* підтримується через плагін *Google Cardboard XR Plugin for Unity*, який забезпечує відстеження руху голови користувача, стереоскопічний рендеринг *VR*-камери та оброблення вводу користувача для мобільних *VR*-застосунків на платформах *iOS* та *Android*. Для завантаження *Google Cardboard SDK* в *Unity*-проєкт спочатку необхідно обрати *Window-> Package Management-> Package Manager*. Далі необхідно обрати +- > *Add package from git URL* (рис. 4.3), ввести *https://github.com/googlevr/cardboard-xr-plugin.git* та натиснути *Add*. Далі відбуватиметься завантаження та встановлення *Google Cardboard SDK* (рис. 4.4).

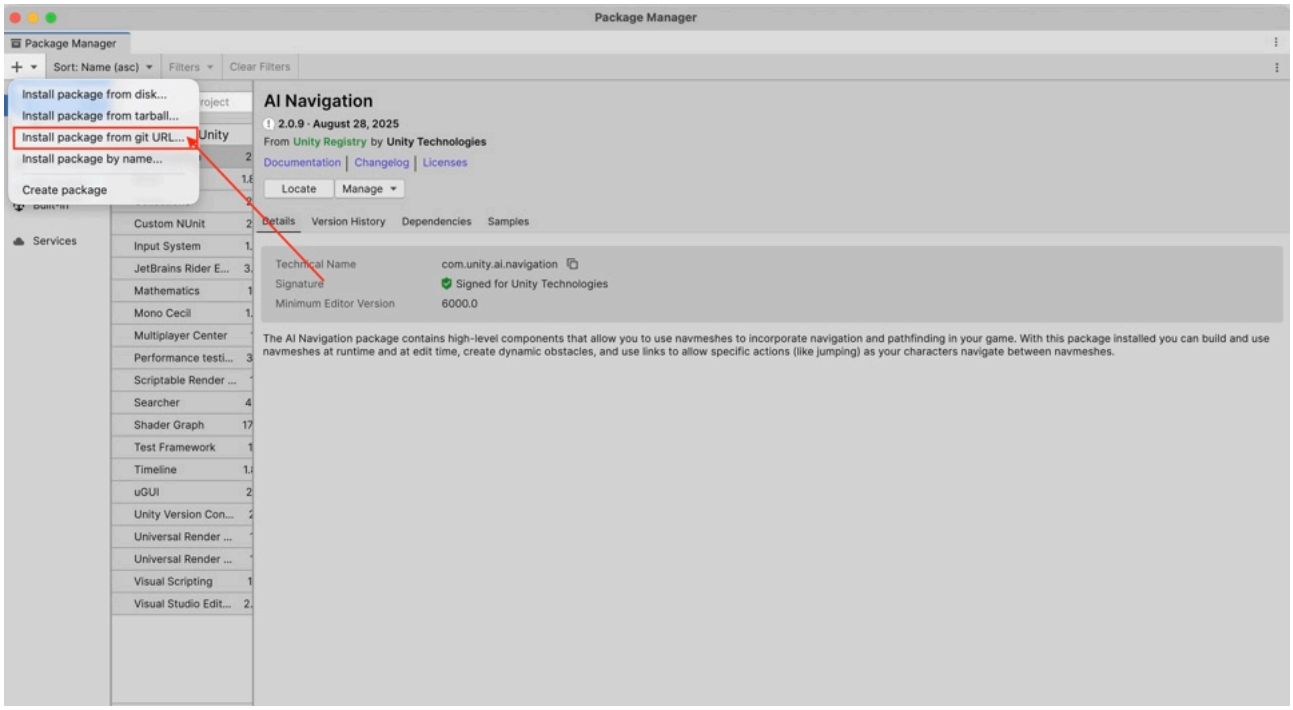


Рис. 4.3 – Додавання бібліотек, пакетів, *SDK* безпосередньо з репозиторію *Git*

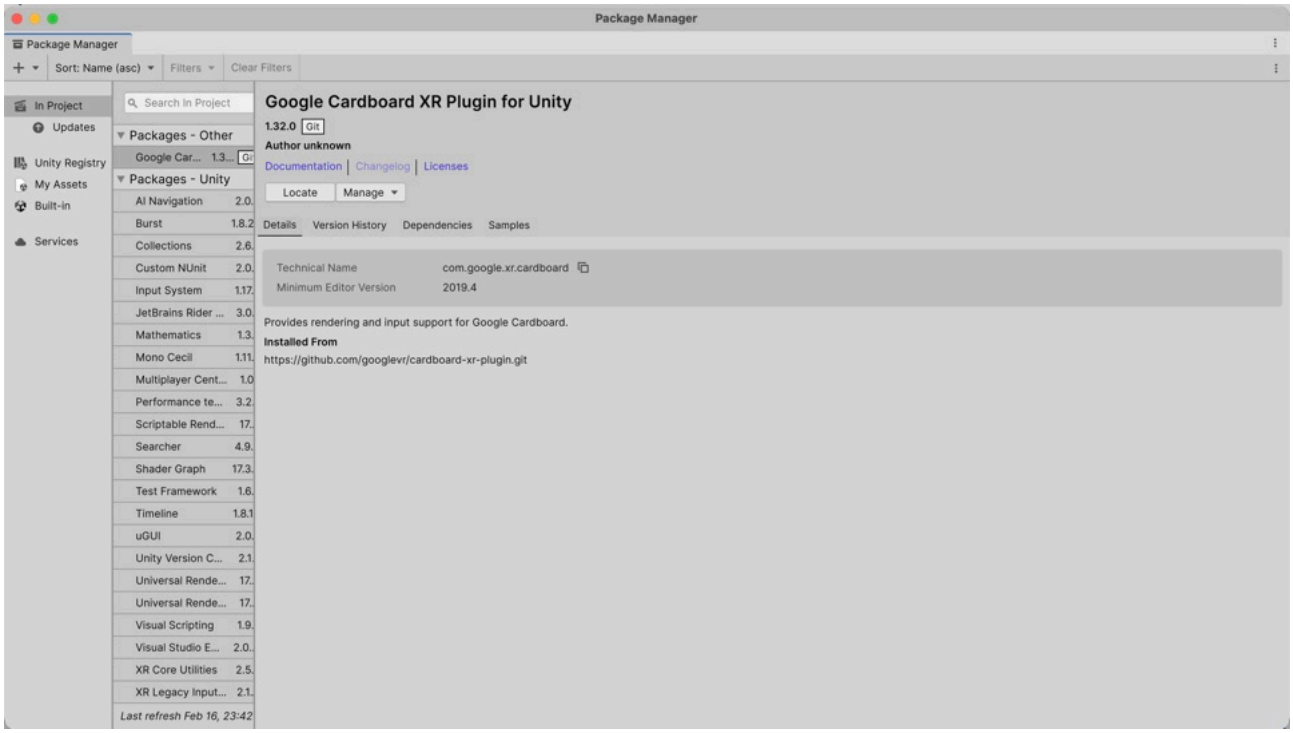


Рис. 4.4 – Завантаження та встановлення *Google Cardboard SDK* в *Unity*-проєкт

В плагіні *Google Cardboard XR Plugin for Unity* вбудовані шаблони, зокрема *Hello Cardboard* – це демонстраційна сцена для ознайомлення з базовими *VR*-функціями, та *VRMode* – це шаблон для розроблення власних *VR*-середовищ, що підтримують відстеження (голови та погляду) та стереорендеринг. Для завантаження шаблонів, в *Package Manager* необхідно обрати вікно *Samples* та натиснути *Import* (рис. 4.5).

Для підключення плагіну в *Unity*-проєкт необхідно обрати *Edit- > Project Settings- > XR Plug-in Management* та обрати *Cardboard XR Plugin* для платформи (рис. 4.6), для якої буде розроблений *VR*-застосунок.

VR-шаблони зберігаються в *Assets/Samples/Google Cardboard XR Plugin for Unity/<version>/Hello Cardboard/Scenes* (рис. 4.7).

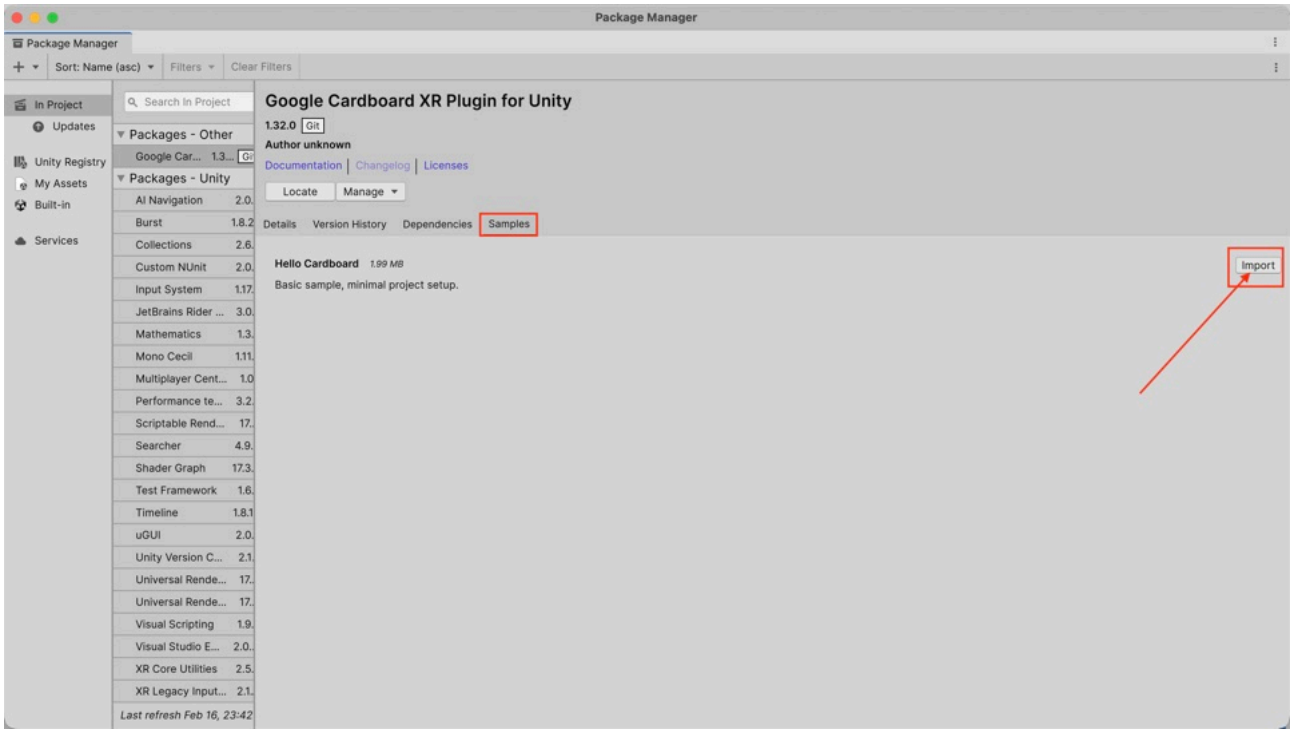


Рис. 4.5 – Завантаження шаблонів плагіну *Google Cardboard XR Plugin for Unity*

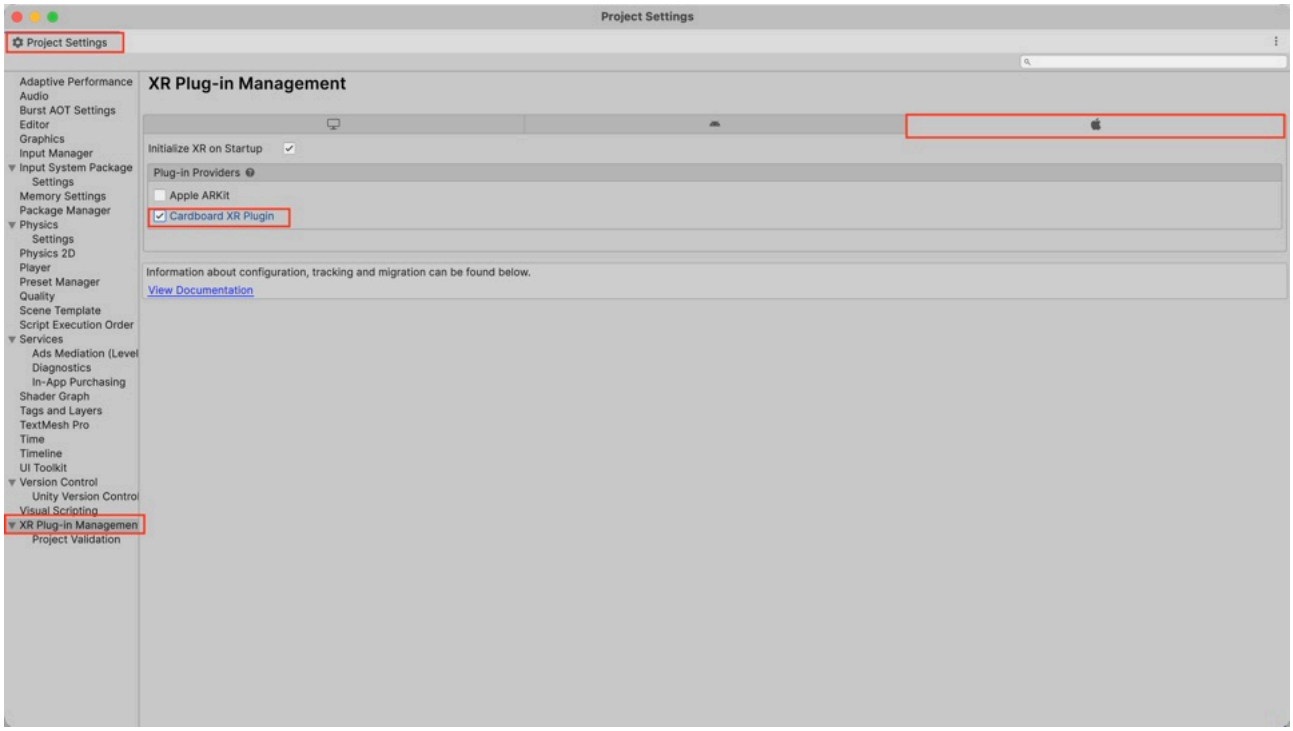


Рис. 4.6 – Підключення плагіну *Cardboard XR Plugin* в *Unity*-проєкт

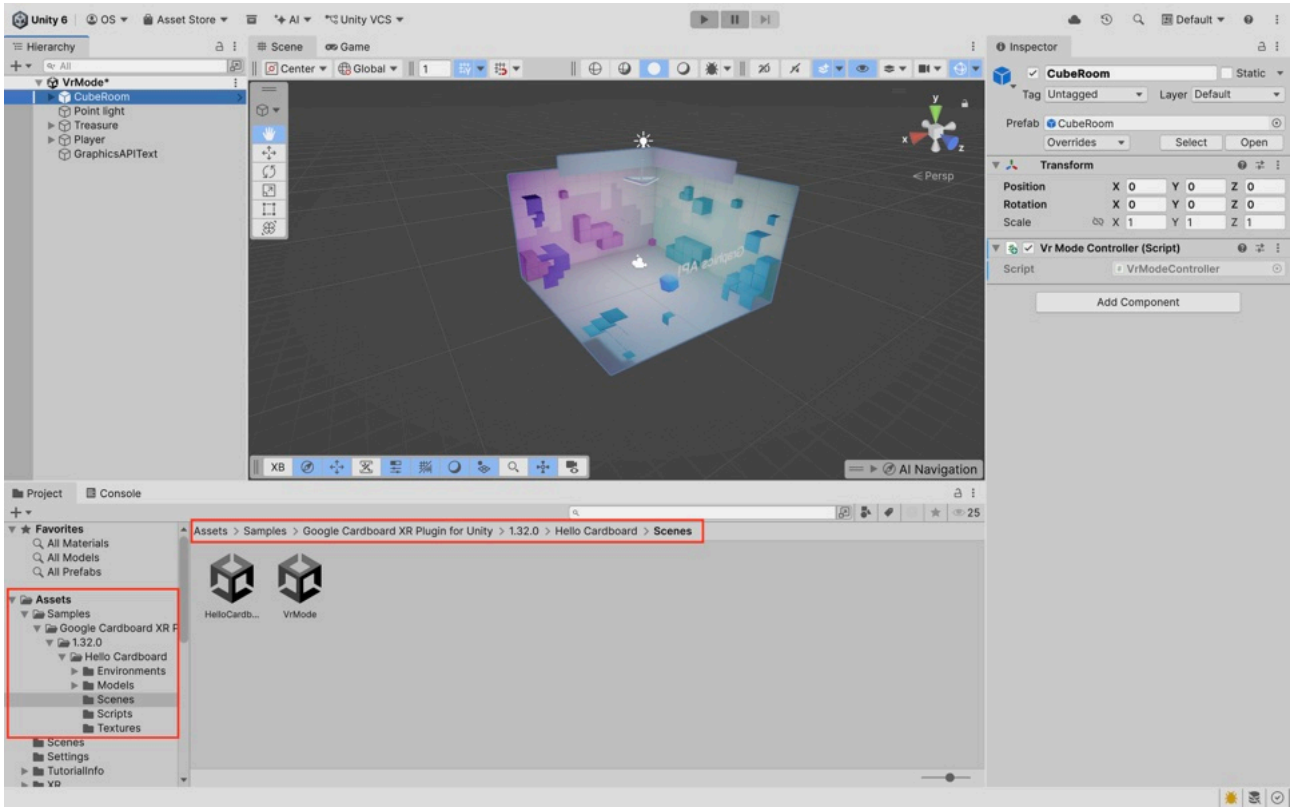


Рис. 4.7 – Шаблонні *VR*-середовища *Google Cardboard*

Якщо на цьому етапі перейти до збирання та розгортання *VR*-застосунку, то віртуальне середовище буде відтворюватись відповідно до шаблонних налаштувань: при виборі (наведенні погляду) *3D*-об’єкта *VR*-сцени, його текстура змінюється, при взаємодії (натискання на екран) – *3D*-об’єкт зникає та у *VR*-сцені розміщується новий *3D*-об’єкт. Цей процес відбувається безперервно.

Налаштування *Unity*-проєкту для реалізації *VR*-застосунку з використанням *Google Cardboard SDK* для *iOS*- та *Android*-платформ в розділі 4.2.1. Для збирання *VR*-застосунку необхідно обрати *File-> Build Profiles* та натиснути *Open Scene List-> Add Open Scenes*. Далі необхідно обрати *Samples/ Google Cardboard XR Plugin for Unity/<version>/Hello Cardboard/Scenes/VrMode* та натиснути *Build*. Далі виконати дії для розгортання *VR*-застосунку, які описані в розділі 4.2.4. Демонстрація роботи *VR*-застосунку в розділі 4.2.5.

Google Cardboard SDK не реалізує самостійно функцію відтворення просторового звуку в *VR*-застосунках. Тому для реалізації просторового звуку в

VR-застосунках на основі *Google Cardboard SDK* необхідно використовувати зовнішні *SDK*.

Технологія просторового звуку

У реальному фізичному середовищі будь-які звуки є просторовими звуками. Джерело звуку характеризується положенням у середовищі, амплітудою, частотою, напрямком розповсюдження звукових хвиль. Середовище розповсюдження звукових хвиль характеризується розміром, наявністю, кількістю та розмірами фізичних перешкод, матеріалами (напр., концертний зал, кімната тощо). Приймач звукових хвиль характеризується розташуванням. Джерелом просторового звуку в *XR*-застосунку є цифровий віртуальний об’єкт, який імітує просторовий звук, середовище розповсюдження – цифровий об’єкт, приймач звукових хвиль – користувачі *XR*-застосунку. Просторовий звук в будь-яких *XR*-застосунках є штучним звуком, реалістичність якого створюється на основі застосування технологій перетворення звуку.

Технології перетворення звуку враховують різницю в часі, коли звукова хвиля надходить до правого та лівого вуха користувача, що дозволяє визначити горизонтальне положення джерела низькочастотного звуку. Ця різниця в часі буде тим більша, чим далі ліворуч чи праворуч розташоване джерело звуку від користувача. Для визначення горизонтального положення джерела високочастотного звуку враховуються відмінності в розподілі гучності та частоти звукової хвилі між правим і лівим вухами, що викликані акустичною тінню голови користувача. Для визначення вертикального положення джерела враховується частота звукової хвилі, що відбивається від внутрішньої сторони зовнішнього вуха користувача.

Технології перетворення звуку також враховують направленість та відбиття звукової хвилі. При розповсюдженні звукової хвилі відбувається її відбиття від усіх поверхонь навколишнього середовища та створюється складна система, що включає прямий звук, ранні відбиття звукової хвилі та

реверберацію. Прямий звук розповсюджується від джерела до користувача. При збільшенні відстані між джерелом звуку та користувачем відбувається зменшення енергії звуку. Тому звуки, які розташовані далі від користувача мають меншу гучність, ніж звуки, які розташовані ближче. Перші відбиті звукові хвилі, які досягають користувача відомі як ранні відбиття. Ранні відбиття звукової хвилі визначають параметри середовища розповсюдження звукової хвилі, напр., розміри, форму. З часом щільність відбиття та одночасного поглинання звукових хвиль збільшується та відбувається реверберація. Реверберація визначається кількістю, щільністю, тоном та тривалістю розповсюдження звукових хвиль. Реверберація звукової хвилі визначає параметри середовища розповсюдження звукової хвилі, напр., матеріали, кількість та щільність перешкод.

Технології перетворення звуку враховують оклюзію звукової хвилі. При розповсюдженні звукової хвилі відбувається її блокування об'єктами, які розташовуються на шляху від джерела до користувача. Високочастотні компоненти звукові хвилі блокуються більше, ніж низькочастотні.

В *Resonance Audio SDK* реалізовані технології перетворення та відтворення просторового звуку для *Web*-платформи, мобільних платформ (*iOS*, *Android*), операційних систем *Windows*, *MacOS*, *Linux* та ін. *Resonance Audio SDK* також підтримується ігровими рушіями, як *Unity* та *Unreal Engine.*

Для використання *Resonance Audio SDK* в *Unity*-проєкті спочатку необхідно завантажити файл *ResonanceAudioForUnity_*.unitypackage* за посиланням: *https://github.com/resonance-audio/resonance-audio-unity-sdk/releases*. Далі в середовищі *Unity* необхідно обрати *Assets- > Import Package- > Custom Package* та обрати завантажений файл. У діалоговому вікні обрати всі файли та натиснути *Import* (рис. 4.8). Останнім необхідно прийняти оновлення *Resonance Audio API*, якщо будуть запропоновані.

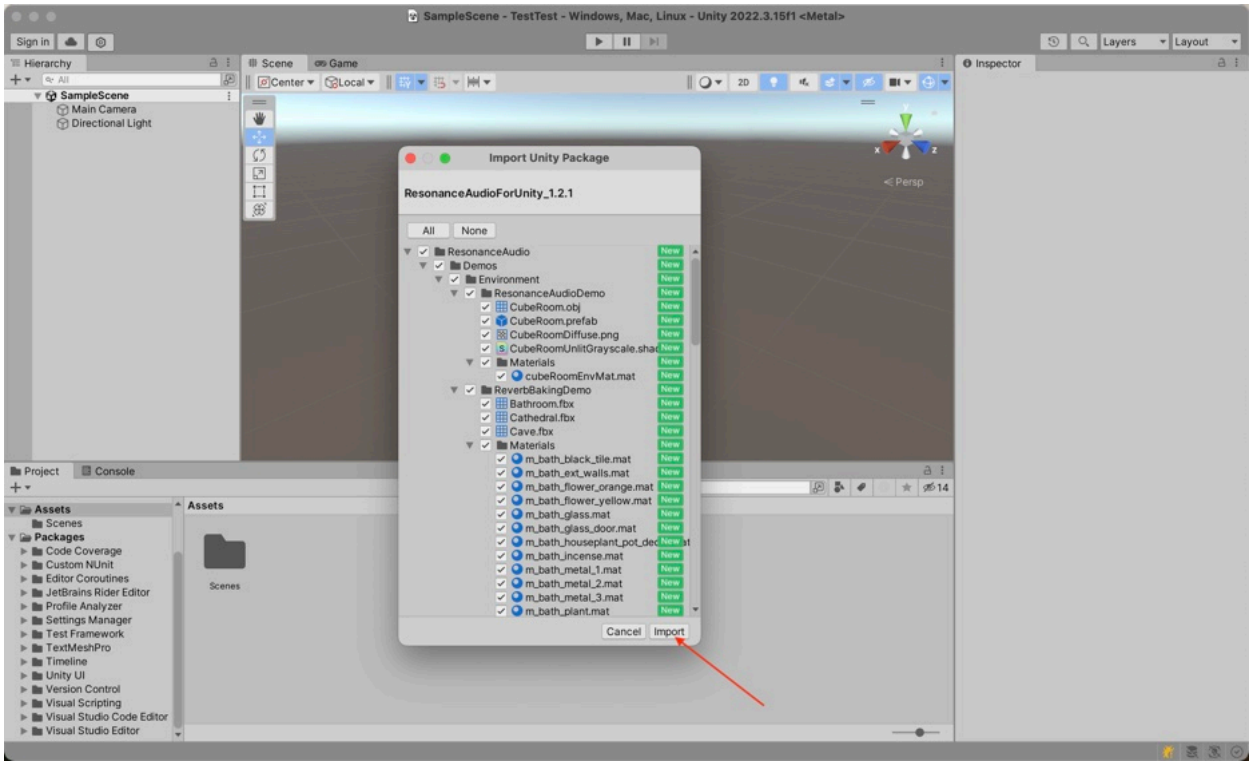


Рис. 4.8 – Імпортування *Resonance Audio* в *Unity*-проєкт

Наступним необхідно налаштувати *Resonance Audio*. *В Unity*-проєкті необхідно обрати *Edit- > Project Settings- > Audio* та для параметрів *Spatializer Plugin* та *Ambisonic Decoder Plugin* обрати *Resonance Audio* (рис. 4.9).

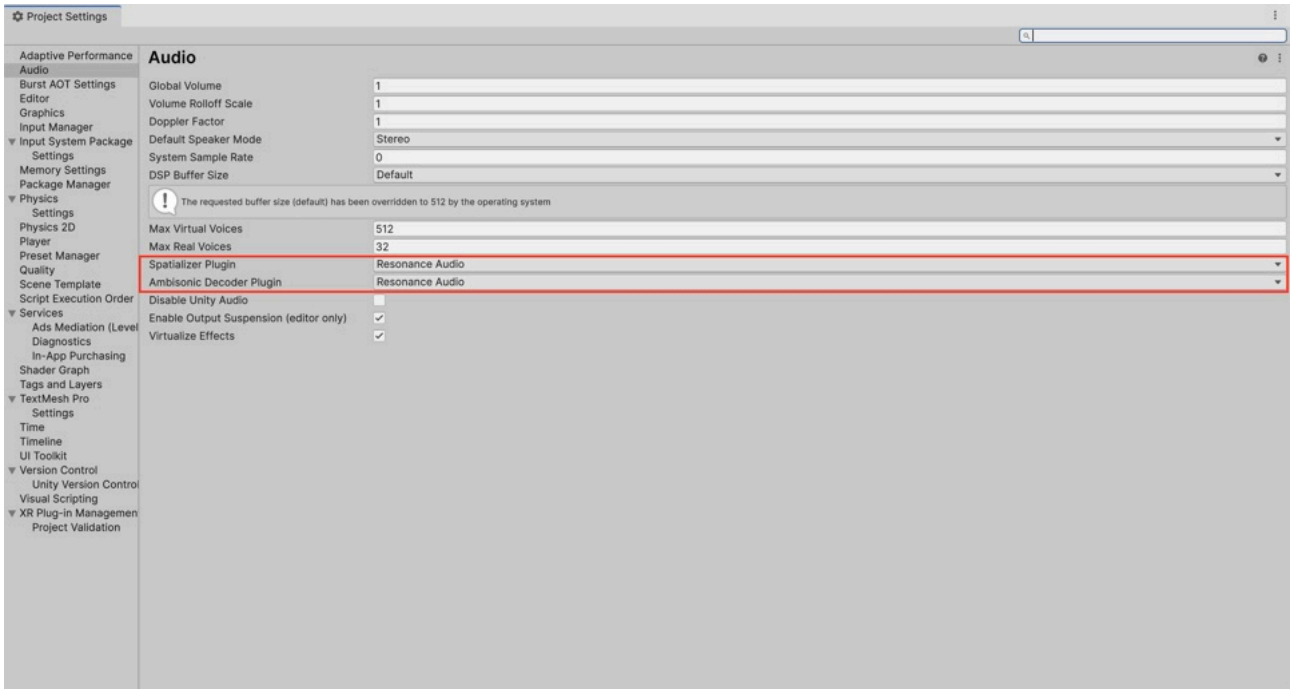


Рис. 4.9 – Налаштування *Resonance Audio* в *Unity*-проєкті

Демонстраційні сцени використання можливостей *Resonance Audio* для відтворення просторового звуку знаходяться в *Assets-> ResonanceAudio-> Demos-> Scenes*.

Для відтворення просторового звуку *Resonance Audio* реалізує наступні компоненти в *Unity*:

– *ResonanceAudioSource*: розширює функції компоненти *Audio Source* шляхом реалізації параметрів спрямованості, оклюзії та якості рендерингу звуку. Для ігрового об'єкта необхідно визначати обидві компоненти, *Audio Source* та *ResonanceAudioSource*.

– *ResonanceAudioListener*: розширює функції компоненти *Audio Listener* шляхом реалізації параметрів глобального посилення та маски оклюзії джерела звуку. Для ігрового об'єкта необхідно визначати обидві компоненти, *Audio Listener* та *ResonanceAudioListener*.

– *ResonanceAudioRoom*: реалізує параметри відтворення звуку в просторі відповідно до заданих налаштувань (матеріалів простору), а також параметри відбивної здатності та реверберації. Відтворення звуку відбувається, коли об'єкт з компонентою *Audio Listener* знаходиться всередині вказаних меж простору.

– *ResonanceAudioReverbProbe*: реалізує параметри реверберації звуку.

4.2 ПОРЯДОК ВИКОНАННЯ

4.2.1 Створення та налаштування *Unity*-проєкту

В *Unity Hub* необхідно обрати *Projects*, натиснути *New Project*, обрати шаблон проєкту *Universal 3D*, дати назву проєкту (напр., *VirtualRealitywithUnity*) та натиснути *Create project* (рис. 2.6).

Наступним необхідно обрати платформу для якої буде розроблений *VR*-застосунок (рис. 2.8), встановити та налаштувати *Google Cardboard SDK*. Пояснення та покрокові інструкції для цього етапу наведено у розділі 4.1.

Останнім необхідно налаштувати *Unity*-проєкт відповідно до платформи розгортання. Спочатку необхідно обрати *Edit-> Project Settings-> Player*. Далі

необхідно налаштувати параметри *Unity*-проєкту відповідно до Таблиці 4.1 або Таблиці 4.2.

Таблиця 4.1. Параметри налаштування *Unity*-проєкту для розроблення *VR*-застосунку на основі *Google Cardboard SDK* для *iOS*-платформи

Назва параметра	Значення
Company Name	Поле призначене для вказання розробника *VR*-застосунку та використовується під час формування ідентифікаторів і параметрів збірки *Unity*-проєкту. Необхідно ввести назву, напр., *MyImmersiveCompany.* Рекомендується змінити початкові значення параметра.
Default Orientation	Поле призначене для визначення орієнтації *VR*-застосунку на екрані (горизонтальне, вертикальне або автоповорот). Необхідно обрати *Landscape Left* або *Landscape Right*.
Bundle Identifier	Поле призначене для ідентифікації *VR*-застосунку. В полі автоматично встановиться значення відповідно до шаблону *com.CompanyName.ProductName.* Напр., *com.MyImmersiveCompany.AugmentedRealitywithUnity*.
Camera Usage Description	Поле призначене для вказання причини використання камери *VR*-пристрою застосунком, яка відображається користувачу під час запиту на дозвіл використання камери. Необхідно ввести значення, напр., *Cardboard SDK requires camera permission to read the QR code (required to get the encoded device parameters).*

Таблиця 4.2. Параметри налаштування *Unity*-проєкту для розроблення *VR*-застосунку на основі *Google Cardboard SDK* для *Android*-платформи

Назва параметра	Значення
Company Name	Поле призначене для вказання розробника *VR*-застосунку та використовується під час формування ідентифікаторів і параметрів збірки *Unity*-проєкту. Необхідно ввести назву, напр., *MyImmersiveCompany.*
Optimized Frame Pacing	Поле призначене для застосування механізму забезпечення рівномірної синхронізації кадрів, що підвищує плавність візуалізації. Зняти позначку за наявності.
Default Orientation	Поле призначене для визначення режиму відображення *VR*-застосунку залежно від орієнтації екрана пристрою – горизонтальний, вертикальний або з автоповоротом. Необхідно обрати *Landscape Left* або *Landscape Right*.
Graphics API	Поле призначене для визначення інтерфейсу, через який *AR*-застосунок взаємодіє з графічним процесором для відображення графіки. Обрати графічний *API*, зокрема *Vulkan*, *OpenGLES2*, *OpenGLES3* або будь-яка їх комбінація. Рекомендовано: видалити *Vulkan* та обрати *OpenGLES3*. Якщо обрано *Vulkan* як графічний *API*, у *Vulkan Settings* зняти *Apply display rotation during rendering*, у *Texture compression format* обрати *ETC2* (для версії *Unity* 2021.2 або вище).
Minimum API Level	Поле призначене для визначення мінімального рівня *Android API*, необхідного для встановлення та роботи *VR*-застосунку. Необхідно обрати значення *Android 8.0 'Oreo'* (*API level 26*) або вище.
Target API	Поле призначене для визначення рівня *Android API* для якого

Level	оптимізований *VR*-застосунок. Обрати *API level 33* або вище.
Target Architectures	Поле призначене для визначення процесорних архітектур, для яких виконується збірка *VR*-застосунку. Рекомендовано обрати *ARMv7*.
Internet Access	Поле призначене для встановлення дозволу підключатись до мережі *VR*-застосунку. Обрати *Require*.

Build
 	
Custom Main Gradle Template
	Поставити позначку.
	У файлі *Assets/Plugins/Android/mainTemplate.gradle* додати:
	
	`implementation 'androidx.appcompat:appcompat:1.6.1'`
	`implementation 'com.google.android.gms:play-services-vision:20.1.3'`
	`implementation 'com.google.android.material:material:1.6.1'`
	`implementation 'com.google.protobuf:protobuf-javalite:3.19.4'`

Custom Gradle Properties Template	Поставити позначку.
	У файлі *Assets/Plugins/Android/gradleTemplate.properties* додати:
	
	`android.enableJetifier=true`
	`android.useAndroidX=true`

Application Entry Point	Поле визначає *Android*-клас, з якого починається виконання *VR*-застосунку. Обрати *Activity* та прибрати *GameActivity* (для версії *Unity* 2023.1 або вище).

4.2.2 Імпортування та налаштування *VR*-об’єктів

Для імпортування *3D*-об’єктів необхідно обрати *Assets-> Import New Asset…* та завантажити файли в теку *Assets*. Далі необхідно вилучити матеріали для *3D*-об’єктів та зберегти їх в новій окремій теці, напр., *Materials*. Далі необхідно створити нові або налаштувати параметри існуючих матеріалів для

3D-об’єктів, імпортувати текстури тощо. Покрокові інструкції цього етапу наведені в розділі 2.2.2

Останнім необхідно розмістити *3D*-об’єкти у *VR*-сцені та налаштувати освітлення (рис. 4.10). Для цифрового *VR*-середовища необхідно застосувати декілька джерел освітлення (рис. 4.10). У вікні *Hierarchy* натиснути ПКМ-> *Light- > Directional Light*, розмістити об’єкти, визначити інтенсивність освітлення, колір та інші параметри.

Відповідно до вимог лабораторного практикуму, *3D*-модель середовища віртуальної реальності можна розробити в *Blender* або завантажити з відкритих джерел, зокрема *Sketchfab* (*https://sketchfab.com/feed*), *CGTrader* (*https://www.cgtrader.com/*) та інші.

4.2.3 Проєктування віртуальної реальності

Проєктування віртуальної реальності на основі *Google Cardboard SDK* починається з налаштування ігрового об’єкта *Player.* Ігровий об’єкт *Player* відображає віртуальне середовище та реагує на рухи та дії користувача (представляє користувача *VR*-застосунку в цифровому світі). Для додавання ігрового об’єкта *Player* в *Unity*-сцену, спочатку необхідно створити префаб в теці *Assets*. Для цього необхідно відкрити шаблонну сцену *Samples/ Google Cardboard XR Plugin for Unity/<version>/Hello Cardboard/Scenes/VrMode* та розмістити ігровий об’єкт *Player* в теці *Assets*. Далі необхідно відкрити *Unity*-сцену, обрати *Assets- > Scenes- > SampleScene*, розмістити на сцені створений префаб, видалити ігровий об’єкт *Main Camera*. Для ігрового об’єкта *Player* у вікні *Inspector* поставити тег *MainCamera* (рис. 4.10).

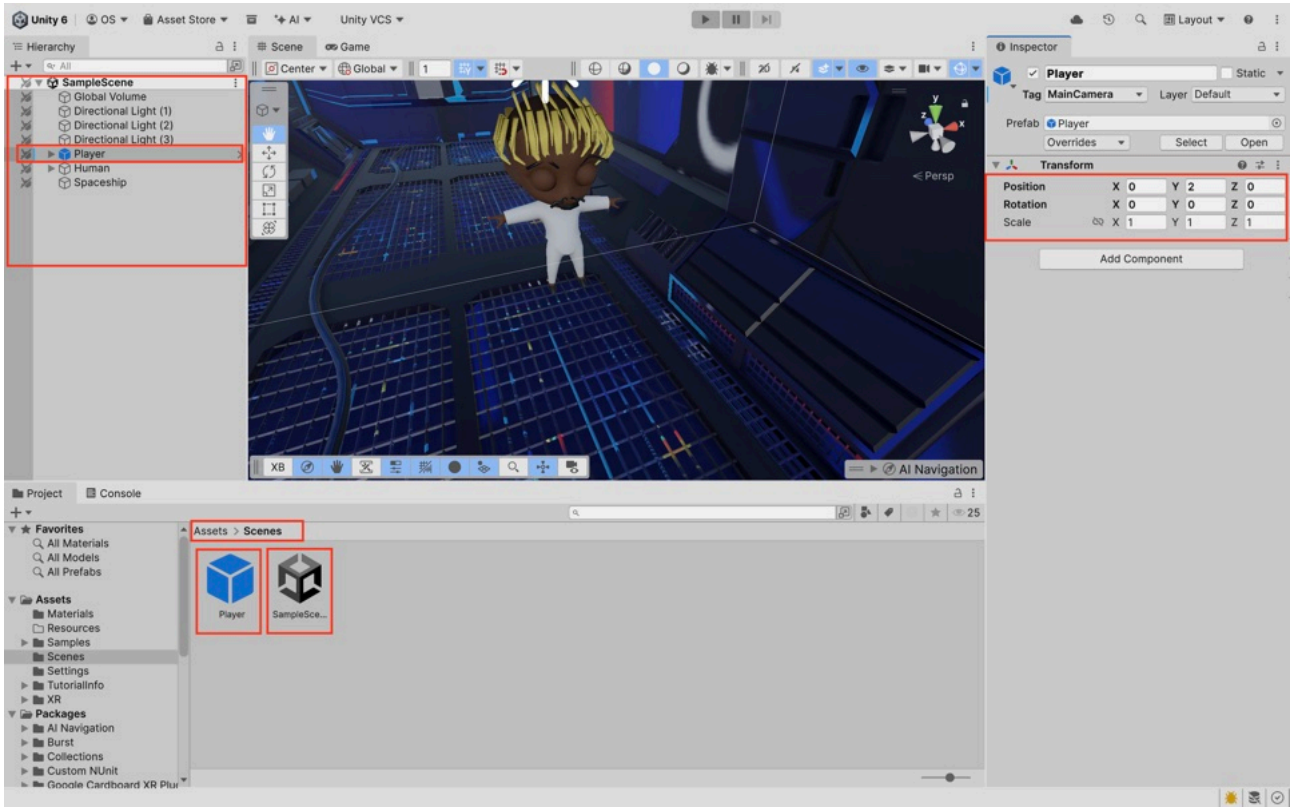


Рис. 4.10 – Розташування *3D*-об’єктів *VR*-сцени

1) Відповідно до ідеї віртуальної реальності, що розроблюється в цьому прикладі, користувач застосунку може переміщуватись по *VR*-сцені, визначивши напрямок руху нахилом голови та зафіксувавши погляд. *VR*-сценою є космічний корабель (до ігрового об’єкта *Spaceship* необхідно додати компоненту *Mesh Colider* для визначення простору переміщення).

Для руху користувача в окулярах *Google Cardboard* у віртуальному середовищі, спочатку необхідно створити скрипт-компоненту для ігрового об’єкта *Player* та у створеному файлі написати програмний код (мова *C#*):

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

// Скрипт для руху користувача у VR за напрямком, куди «дивиться» камера
public class VRLookWalk : MonoBehaviour
{

// Посилання на Transform VR-камери, яку будемо використовувати для визначення напрямку руху
    public Transform vrCamera;

// Кут нахилу голови, після якого користувач починає рухатись вперед
```

```
    public float toggleAngle = 30.0f;

// Швидкість руху користувача
    public float speed = 3.0f;

// Змінна, що вказує, чи потрібно рухатись вперед
    public bool moveForward;

// Компонента CharacterController, яка керує рухом об’єкта Player
    private CharacterController cc;

    void Start()
    {
// Отримання компоненти Character Controller з об’єкта
        cc = GetComponent<CharacterController>();
    }

    void Update()
    {
// Перевірка кута нахилу VR-камери по осі X для застосування руху
// Якщо голова користувача нахилена вниз на кут більше toggleAngle, але менше 90
градусів, відбувається рух вперед
        if (vrCamera.eulerAngles.x >= toggleAngle && vrCamera.eulerAngles.x < 90.0f)
        {
            moveForward = true;
        }
        else
        {
            moveForward = false;
        }
// Застосування вектора руху до об'єкта, для якого доданий скрипт
        if (moveForward)
        {

// TransformDirection перетворює локальний вектор вперед у світові координати
            Vector3 forward = vrCamera.TransformDirection(Vector3.forward);

// Використовуємо SimpleMove для руху CharacterController
// SimpleMove автоматично враховує гравітацію
            cc.SimpleMove(forward * speed);
        }
    }
}
```

Наступним необхідно налаштувати скрипт-компоненту *VRLookWalk*, зокрема додати ігровий об’єкт *Main Camera*, встановити кут нахилу голови користувача (*VR*-камери), при якому відбуватиметься рух у віртуальному середовищі, швидкість руху та обрати функцію руху назад (рис. 4.11).

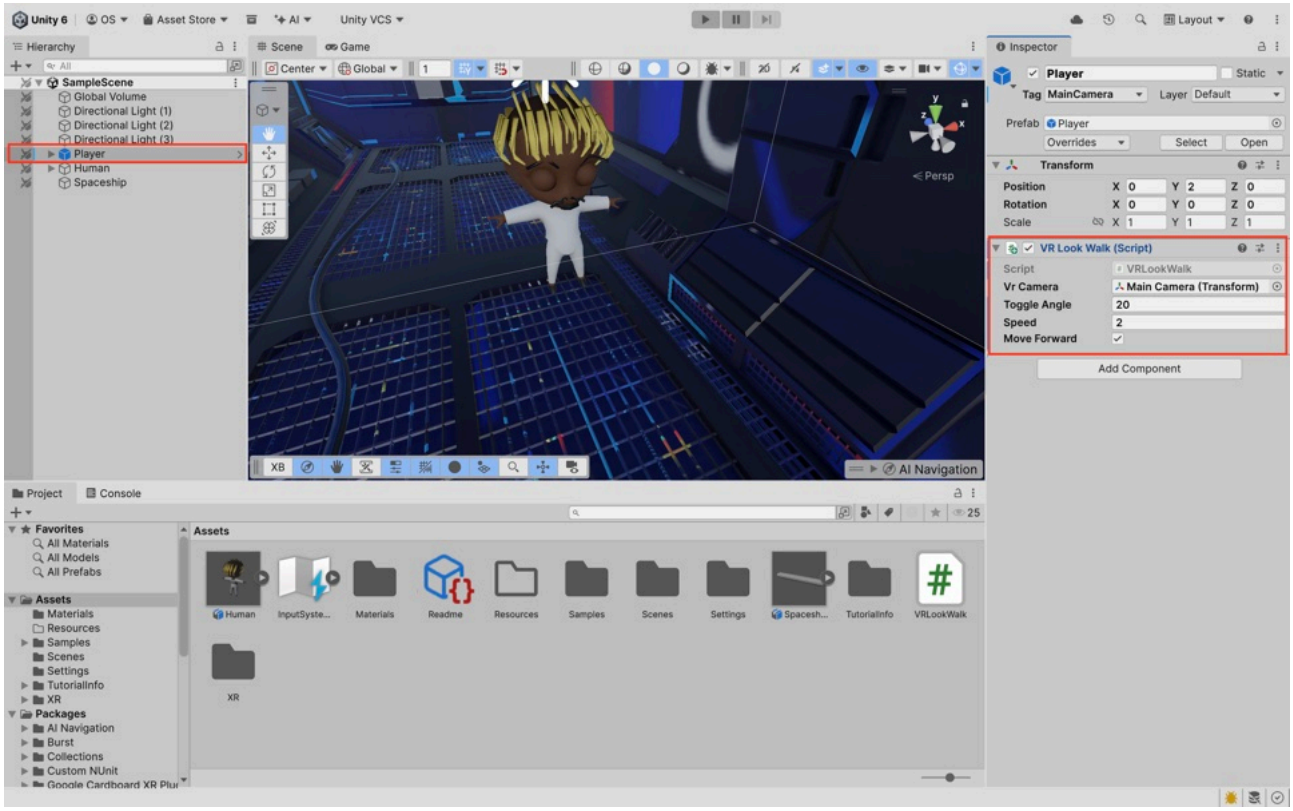


Рис. 4.11 – Налаштування скрипт-компоненти *VRLookWalk*

Наступним необхідно додати та налаштувати компоненту *Character Controller*. У вікні *Inspector* необхідно натиснути *Add Component-> Physics-> Character Controller*. Далі необхідно визначити розташування центру контрролера, висоту та радіус (рис. 4.12). Висота контролера має бути вдвічі більшою за *Y*-координату його центру, при цьому контролер повинен залишатися чітко в межах цифрового простору (ігрового об'єкта *Spaceship*). Ігровий об'єкт *Player* також необхідно розташувати чітко в межах цифрового простору.

Останнім необхідно додати та налаштувати компоненту *Rigidbody*. У вікні *Inspector* необхідно натиснути *Add Component-> Physics-> Rigidbody*. У параметрах компоненти увімкніть використання гравітації та заморожування положення ігрового об'єкта *Player* по осі *Y*.

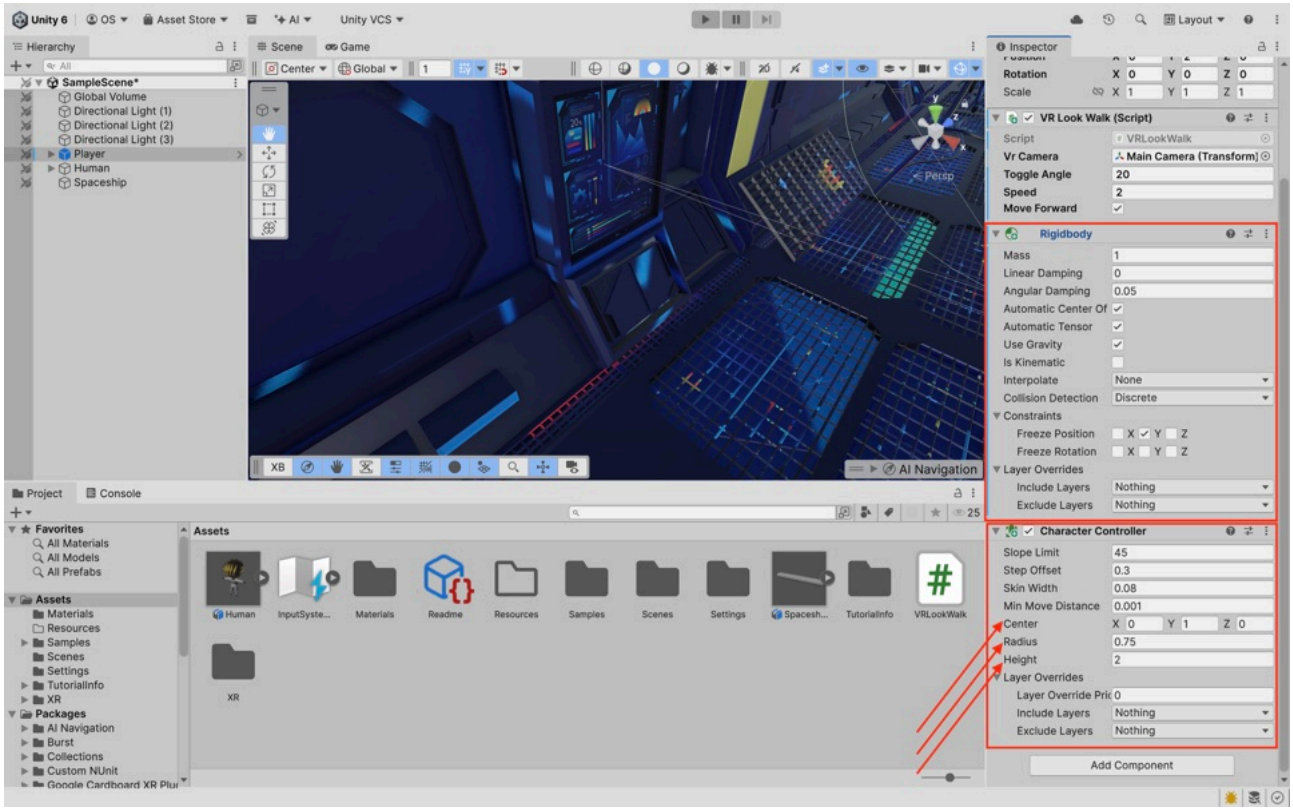


Рис. 4.12 – Налаштування компонент *Character Controller* та *Rigidbody*

Якщо на цьому етапі перейти до збирання та розгортання застосунку, то *VR*-сцена буде відтворюватись відповідно до поточних налаштувань *Unity*-проєкту. У *VR*-застосунку користувач зможе рухатись по сцені (змінювати положення та орієнтацію), визначивши напрям руху поворотом голови та зафіксувавши погляд.

2) Відповідно до ідеї віртуальної реальності, що розроблюється в цьому прикладі, користувач застосунку може взаємодіяти з *VR*-середовищем, а саме з ігровим вікном, наблизившись до ігрового вікна та зафіксувавши погляд на елементі керування (кнопці).

Для створення ігрового вікна в *Unity*-проєкті, спочатку необхідно створити та налаштувати ігровий об’єкт *Canvas*. У вікні *Hierarchy* необхідно натиснути ПКМ-> *UI (Canvas)-> Canvas* та визначити параметри ігрового вікна – розташування, масштаб, режим рендерингу, камеру для керування взаємодією та інше (рис. 4.13).

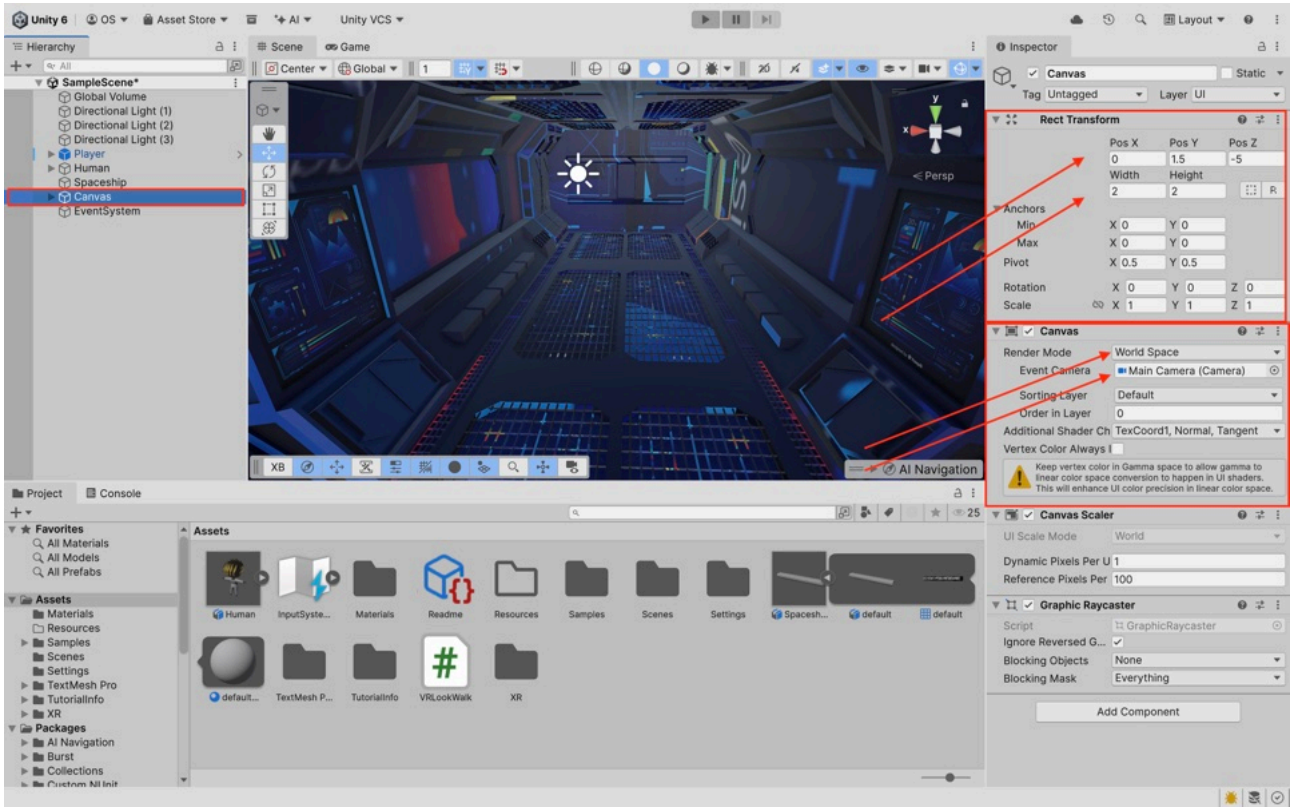


Рис. 4.13 – Налаштування ігрового об’єкта *Canvas*

Наступним необхідно створити інтерфейс ігрового вікна. Спочатку необхідно створити ігровий об’єкт *Panel*, як дочірній об’єкт *Canvas*. У вікні *Hierarchy* необхідно натиснути ПКМ-> *UI (Canvas)-> Panel*, назвати об’єкт, напр., *PanelInfo*, та визначити параметри об’єкта – положення, орієнтацію, масштаб, фонове зображення, колір, матеріал та інше (рис. 4.14).

Наступним необхідно створити ігровий об’єкт *Button*, як дочірній об’єкт *Canvas*. У вікні *Hierarchy* необхідно натиснути ПКМ-> *UI (Canvas)-> Button*, назвати об’єкт, напр., *ButtonInfo*, та визначити параметри об’єкта (рис. 4.15). Ігровий об’єкт *ButtonInfo* є активним елементом ігрового вікна – кнопкою. Ігровому об’єкту *ButtonInfo* необхідно дати назву. Для цього необхідно налаштувати дочірній ігровий об’єкт *Text (TMP)* – визначити положення, орієнтацію, масштаб, шрифт, розмір, колір та інше. Якщо інструмент для конфігурації тексту (*TextMeshPro*) раніше був завантажений, то достатньо буде імпортувати *TMP Essential Resources* та *TMP Examples & Extras* в *Unity*-проєкт.

В іншому випадку, покрокова інструкція завантаження та налаштування плагіну *TextMeshPro* наведена в розділі 3.2.4.

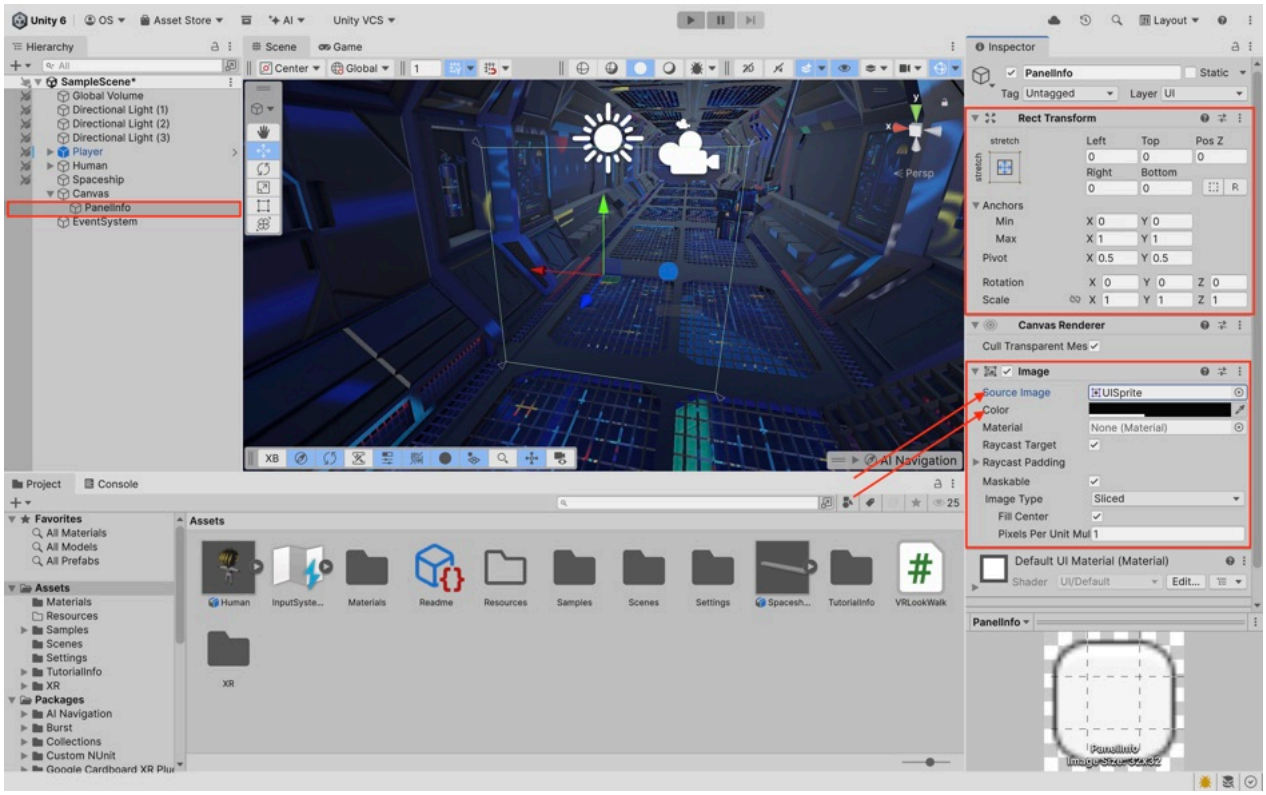


Рис. 4.14 – Налаштування ігрового об’єкта *PanelInfo*

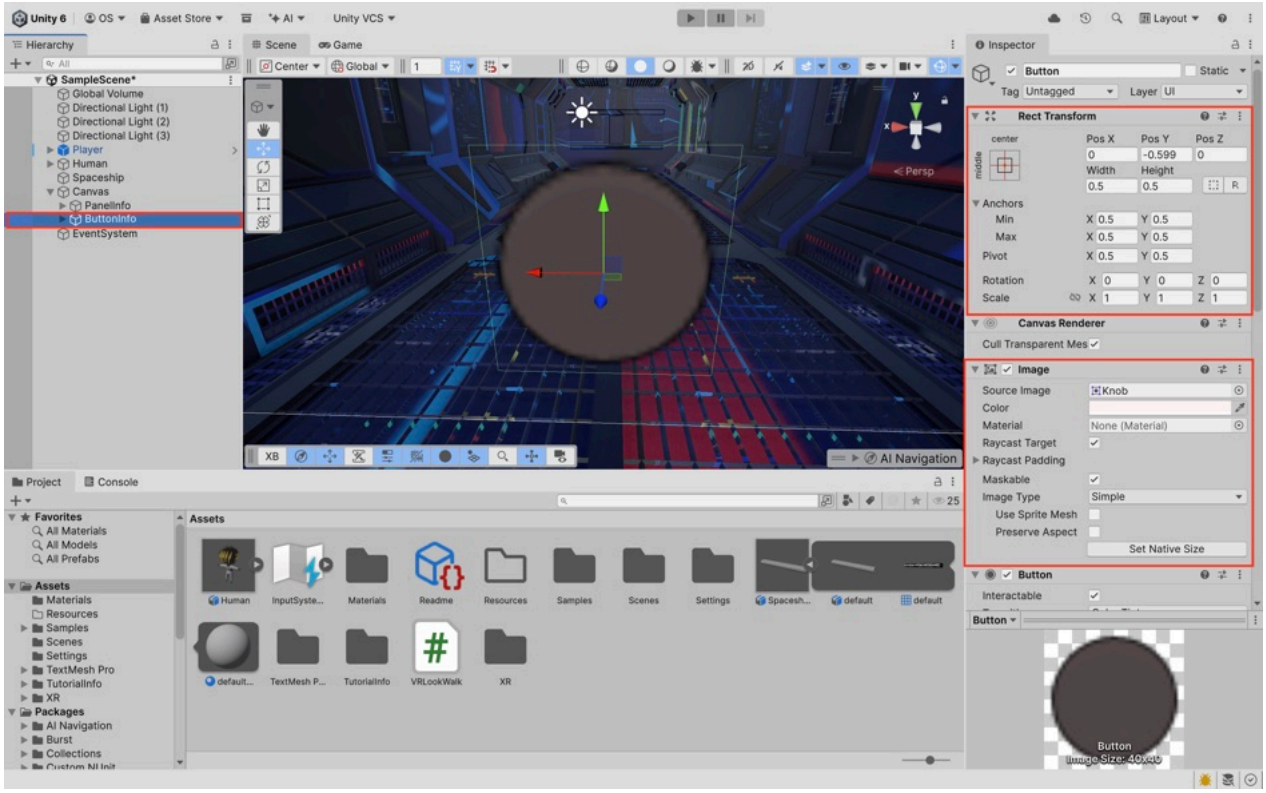


Рис. 4.15 – Налаштування ігрового об’єкта *ButtonInfo*

Наступним необхідно створити ігровий об’єкт *Image*, як дочірній об’єкт *PanelInfo*. У вікні *Hierarchy* необхідно натиснути ПКМ-> *UI (Canvas)-> Image*, назвати об’єкт, напр., *Info*, та визначити параметри об’єкта (рис. 4.16). Для ігрового об’єкта *Info* необхідно створити дочірній об’єкт *Text*. У вікні *Hierarchy* необхідно натиснути ПКМ-> *UI (Canvas)-> Text- TextMeshPro*, назвати об’єкт, напр., *TextInformation*, та визначити параметри – текст, шрифт, розмір, положення та орієнтація, масштаб (рис. 4.17). Відповідно до завдання лабораторного практикуму, ігровий об’єкт *TextInformation* відображає інструкцією щодо правил взаємодії з цифровими об’єктами та/або середовищем.

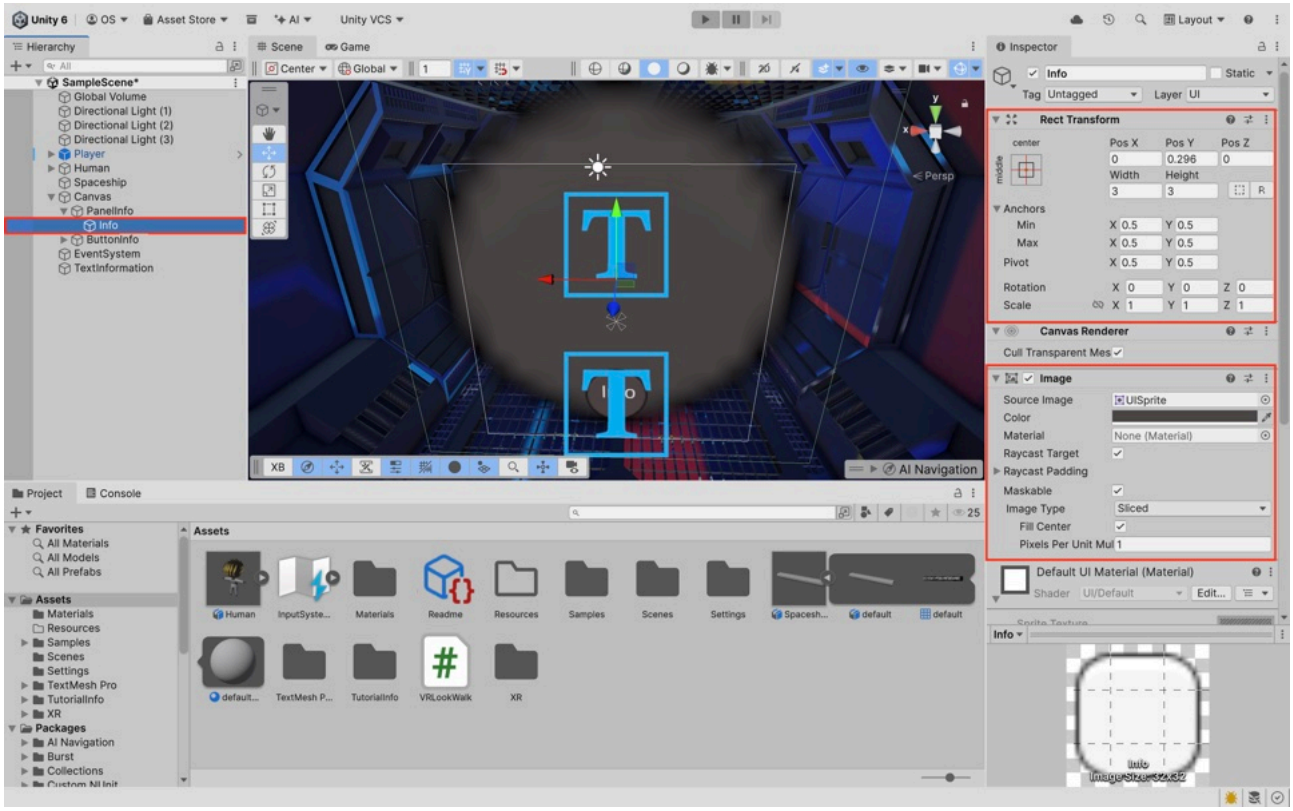


Рис. 4.16 – Налаштування ігрового об’єкта *Info*

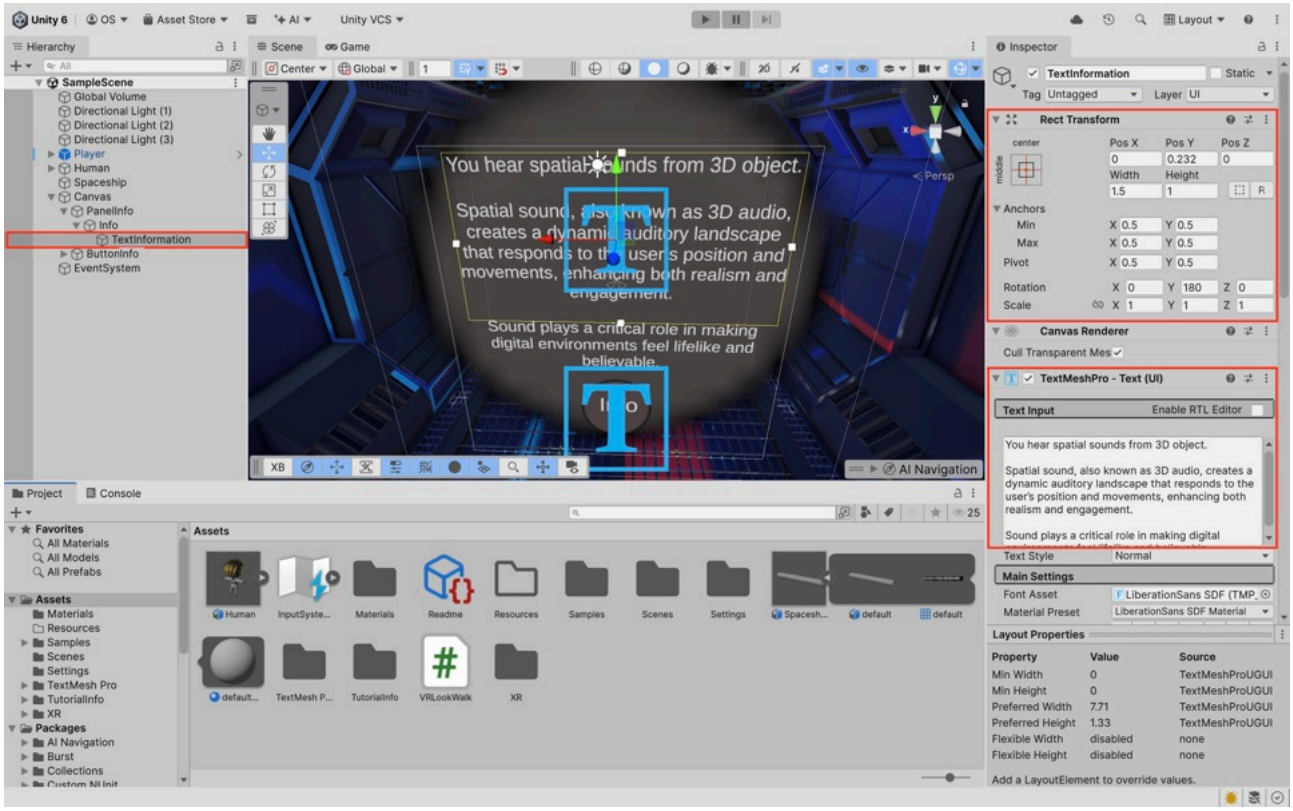


Рис. 4.17 – Налаштування ігрового об’єкта *TextInformation*

Для відображення, а також приховування на ігровому вікні текстової інструкції при наведенні погляду користувача на відповідну кнопку, необхідно для ігрового об’єкта *Info* (ігровий об’єкт типу *Image*) створити скрипт-компоненту (мова *C#*):

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

public class ShowHideInfo : MonoBehaviour
{
    public GameObject InfoObject;

// Приватна змінна для збереження поточного стану об'єкта
    // false – об'єкт прихований, true – показаний
    private bool Show = false;

    void Start()
    {

    }

    void Update()
    {
```

```
    }

// Метод для перемикання стану об'єкта (відобразити/приховати)
    // Викликається при натисканні кнопки ігрового вікна
    public void ShowAndHideInfo()
    {

// Перевіряємо, чи об'єкт зараз прихований
        if (!Show)
        {

// Активуємо об'єкт (робимо його видимим)
            InfoObject.SetActive(true);

// Встановлюємо стан "візуалізувати"
            Show = true;
        }
        else
        {

// Якщо об'єкт уже візуалізований — деактивуємо його
            InfoObject.SetActive(false);

// Встановлюємо стан на "приховати"
            Show = false;
        }
    }
}
```

Останнім необхідно налаштувати скрипт-компоненту *ShowHideInfo* для ігрового об’єкта *Info* (рис. 4.18).

Наступним необхідно для ігрового об’єкта *ButtonInfo* створити скрипт-компоненту, щоб користувач мав змогу взаємодіяти з ігровим вікном. Коли користувач наблизиться до ігрового вікна та зафіксує погляд на кнопці керування протягом визначеного часу (за замовченням 2 секунди) – в ігровому вікні відобразиться текстова інформація. При повторному фіксуванні погляду на кнопці – текстова інформація зникне з ігрового вікна. Коли користувач фокусуватиме погляд на кнопці тільки один раз, то текстова інформація в ігровому вікні буде візуалізуватись протягом всієї *VR*-сесії. Протягом *VR*-сесії користувач також може ігнорувати ігрове вікно та не взаємодіяти з ним.

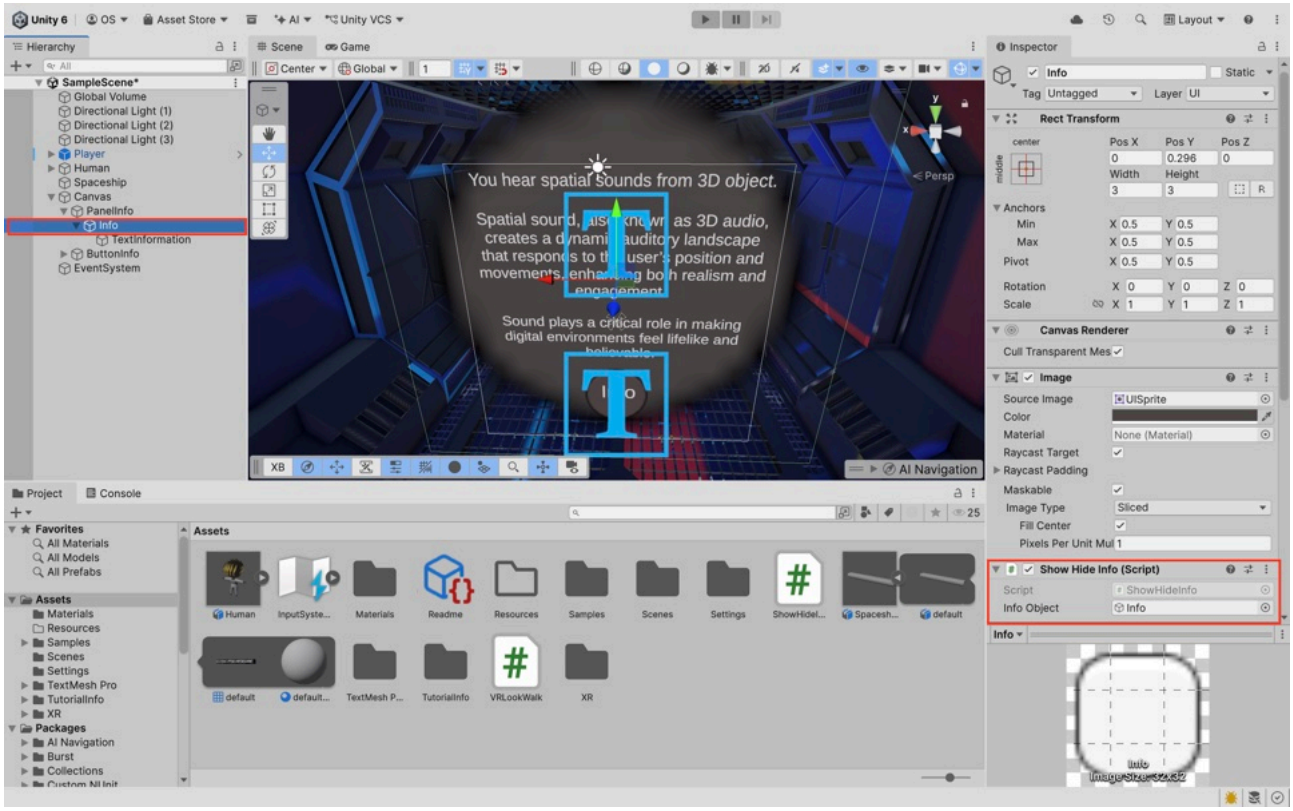


Рис. 4.18 – Налаштування компоненти *ShowHideInfo*

У вікні *Inspector* спочатку необхідно натиснути *Add Component->New script* та назвати файл, напр. *GazeInteraction,* натиснути *Create and Add*. У створеному файлі необхідно написати код (мова *C#*):

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class GazeInteraction : MonoBehaviour
{

// Час (в секундах), протягом якого користувач повинен дивитися на об'єкт, щоб
спрацювала подія
    public float gazeTime = 2f;

// Лічильник часу, який накопичується, поки користувач дивиться на об'єкт
    private float timer;

// Стан, чи користувач дивиться на об'єкт
    private bool gazedAt;

    void Start()
    {

    }
```

```
    void Update()
    {

// Якщо користувач дивиться на об'єкт
        if (gazedAt)
        {

// Накопичуємо час, що пройшов
            timer += Time.deltaTime;

// Якщо користувач дивився достатньо (gazeTime)
            if (timer >= gazeTime)
            {
// Викликаємо подію "натискання" (PointerDown) через Event System
// Це дозволяє об'єкту реагувати як на натискання, але через погляд
                ExecuteEvents.Execute(gameObject, new
PointerEventData(EventSystem.current), ExecuteEvents.pointerDownHandler);

// Скидаємо таймер для повторного відліку
                timer = 0f;
            }
        }

    }

// Метод викликається, коли користувач починає дивитися на об'єкт
    public void PointerEnter()
    {
        gazedAt = true;
        Debug.Log("PointerEnter");
    }

// Метод викликається, коли користувач відводить погляд від об'єкта
    public void PointerExit()
    {
        gazedAt = false;
        Debug.Log("PointerExit");
    }

// Метод викликається при спрацьовуванні "натискання"
    public void PointerDown()
    {
        Debug.Log("PointerDown");
    }
}
```

Останнім необхідно додати та налаштувати компоненту *Event Trigger* для ігрового об’єкта *ButtonInfo*. Спочатку у вікні *Inspector* необхідно натиснути *Add Component- > Event Trigger*. Далі необхідно додати події та налаштувати їх. Для

додавання події *Enter* необхідно натиснути *Add New Event Type- > PointerEnter - > +*. Для параметра *Object* необхідно обрати *ButtonInfo*, для параметра *Function* обрати *GazeInteraction- > PointerEnter ()* (рис. 4.19).

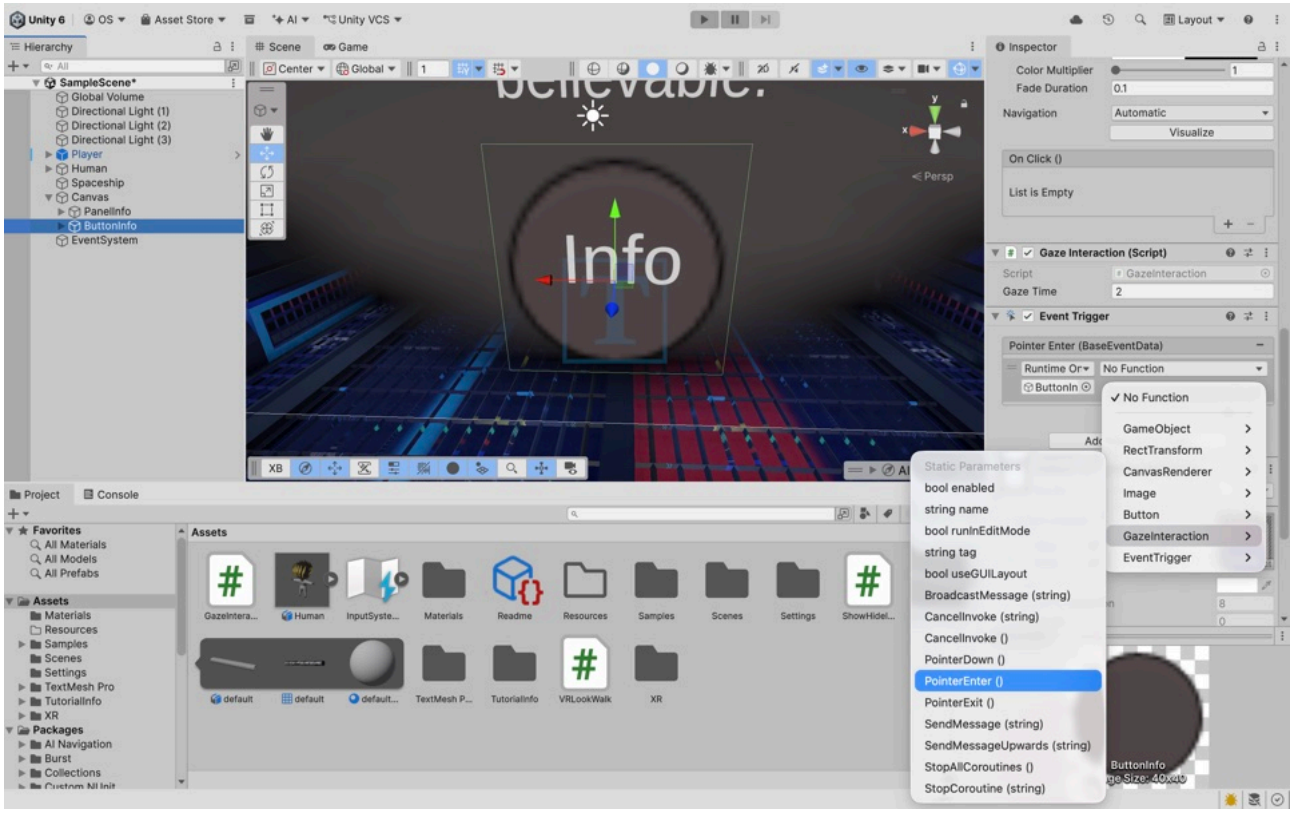


Рис. 4.19 – Налаштування події *Enter*

Для додавання події *Exit* необхідно натиснути *Add New Event Type- > PointerExit- > +*. Для параметра *Object* необхідно обрати *ButtonInfo*, для параметра *Function* обрати *GazeInteraction- > PointerExit ()*.

Для додавання події *Down* необхідно натиснути *Add New Event Type- > PointerDown- > +*. Для параметра *Object* необхідно обрати *Info*, для параметра *Function* обрати *ShowHideInfo- > ShowAndHideInfo ()* (рис. 4.20).

Останнім необхідно вимкнути ігровий об’єкт *Info* (рис. 4.21) та налаштувати ігровий об’єкт *Player* (рис. 4.22) для візуалізації взаємодії користувача з об’єктами *VR*-застосунку. Радіус ігрового об’єкта *Player* буде збільшуватись при переміщенні, повороті та фокусуванні погляду користувача, що буде свідчити про можливість взаємодії з елементами *VR*-застосунку.

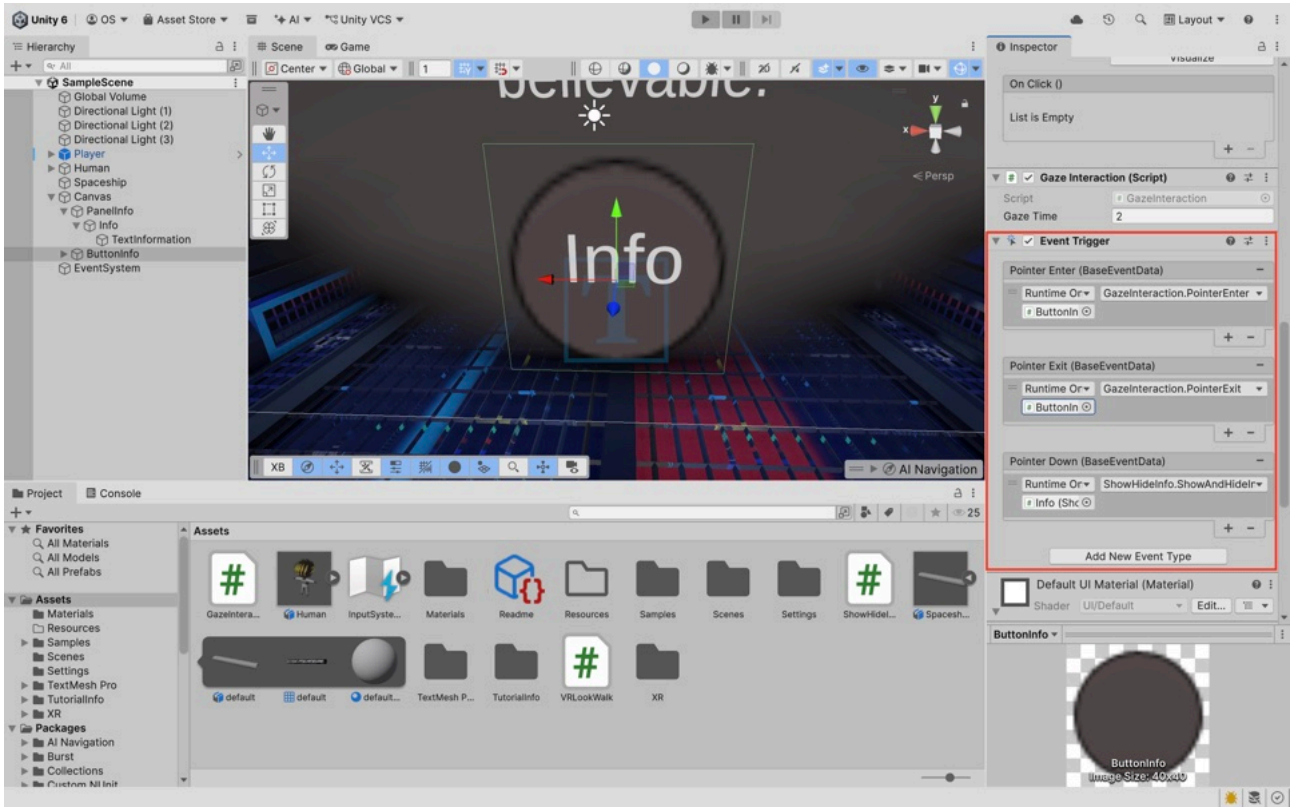


Рис. 4.20 – Налаштування компоненти *Event Trigger*

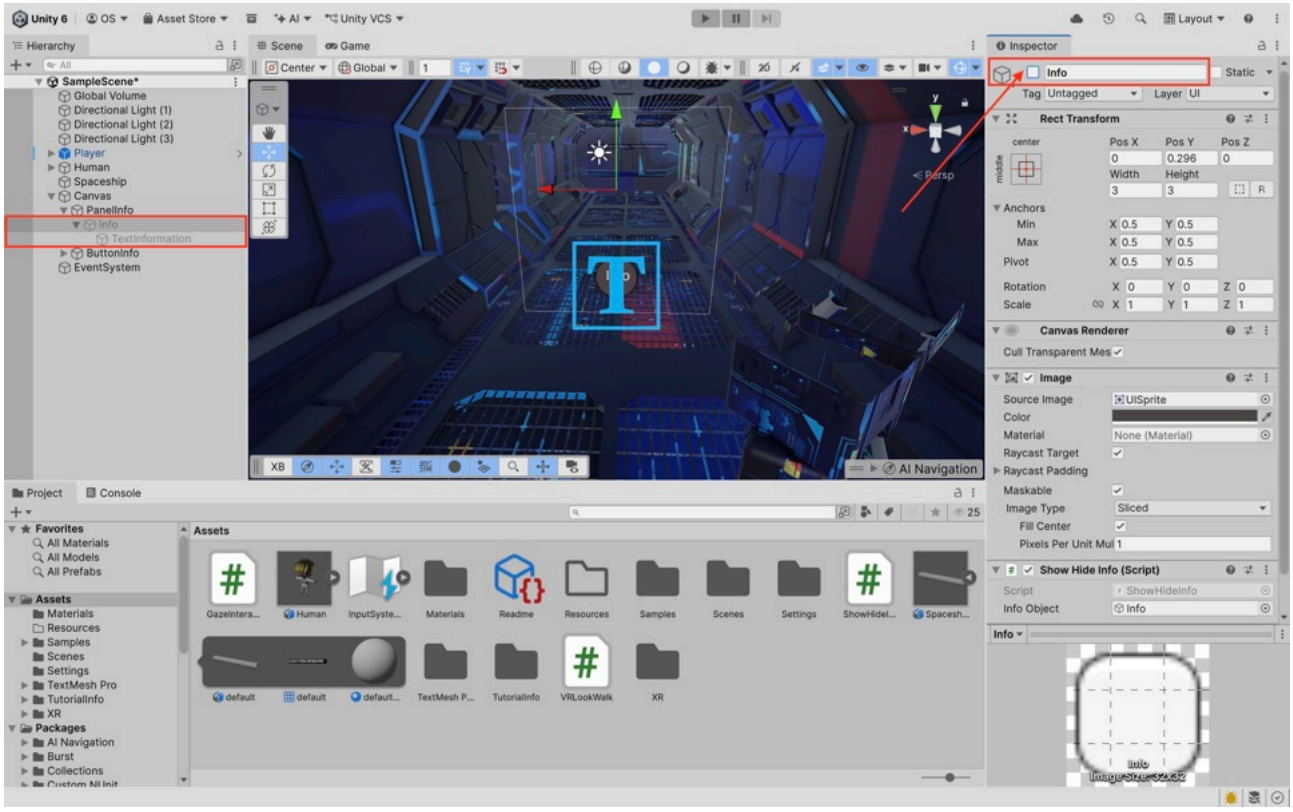


Рис. 4.21 – Налаштування ігрового об’єкта *Info* перед збиранням *Unity*-проєкту

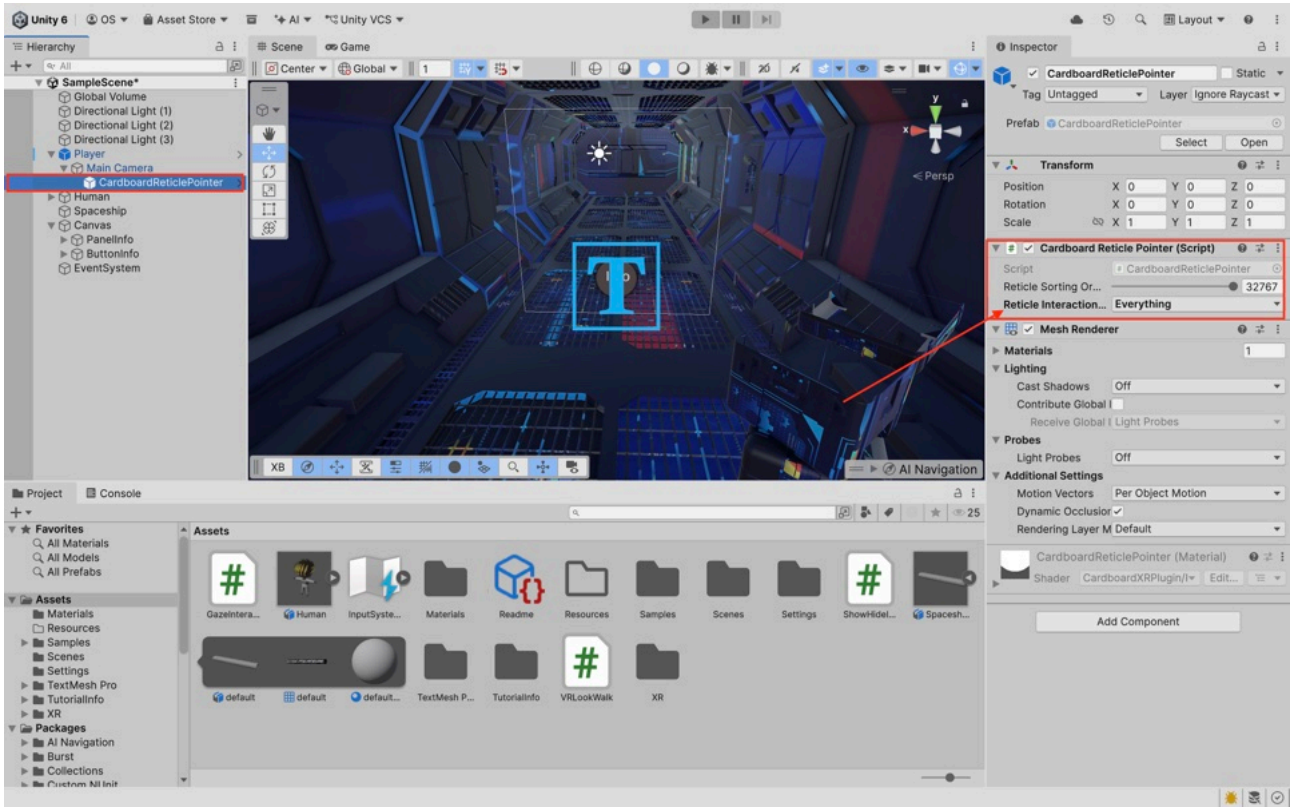


Рис. 4.22 – Налаштування ігрового об’єкта *Player* перед збиранням *Unity*-проєкту

3) Відповідно до ідеї віртуальної реальності, що розроблюється в цьому прикладі, коли цифровий *3D*-об’єкт рухається у віртуальному середовищі відтворюється просторовий звук.

Для анімування *3D*-об’єкта необхідно створити або імпортувати анімаційні кліпи (*Animation Clips*), які визначають рухи об’єкта або зміни його властивостей у часі. Для створення анімаційного кліпу використовується інструмент *Timeline* в *Unity*. Покрокова інструкція створення анімаційного кліпу та анімування самого *3D*-об’єкта наведена у розділі 2.2.2. Ці кліпи додаються до компоненти *Animator*, що прикріплюється до *3D*-об’єкта у вікні *Inspector*, після чого можна налаштувати стани анімацій та переходи між ними в *Animator Controller*.

Для імпортування анімаційного кліпу можна скористатись платформою *Mixamo* (https://www.mixamo.com/#/). *Mixamo* є онлайн-платформою для анімування *3D*-персонажів (рис. 4.23). *Mixamo* містить велику бібліотеку *3D*-

об’єктів (персонажів) та готових анімацій. Параметрами налаштування анімацій є швидкість, тривалість, інтенсивність рухів. *Mixamo* підтримує експорт *3D*-моделей у форматах *FBX* або *OBJ* для використання в ігрових рушіях *Unity*, *Unreal Engine* та інших платформах.

Mixamo підтримує імпорт *3D*-моделей (формати *FBX*, *OBJ*) та створює скелетну анімацію шляхом автоматизованого визначення позиції ключових точок моделі (рис. 4.24).

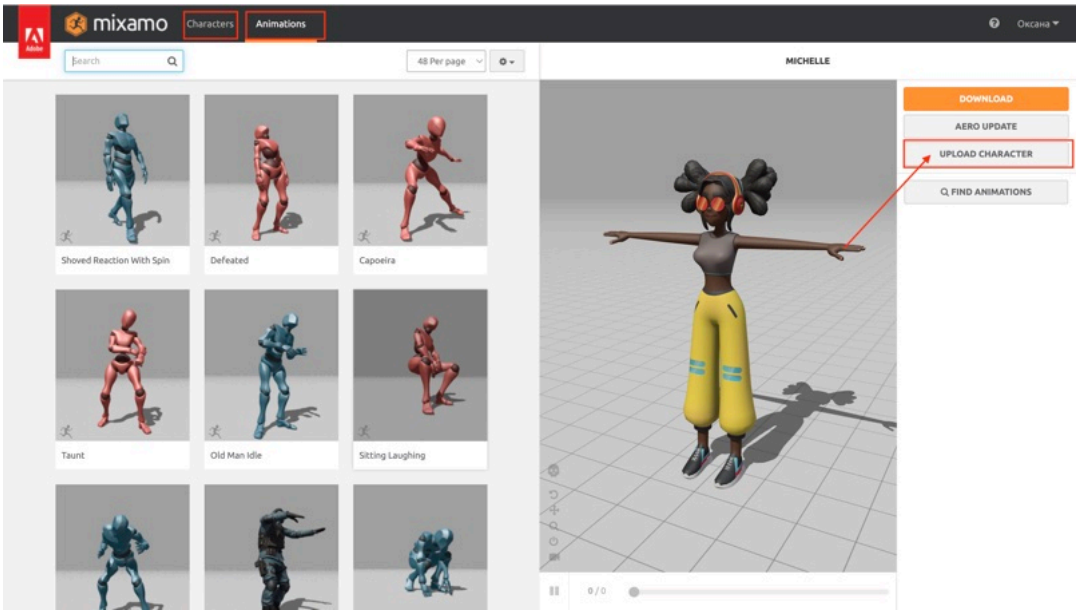


Рис. 4.23 – Платформа *Mixamo*

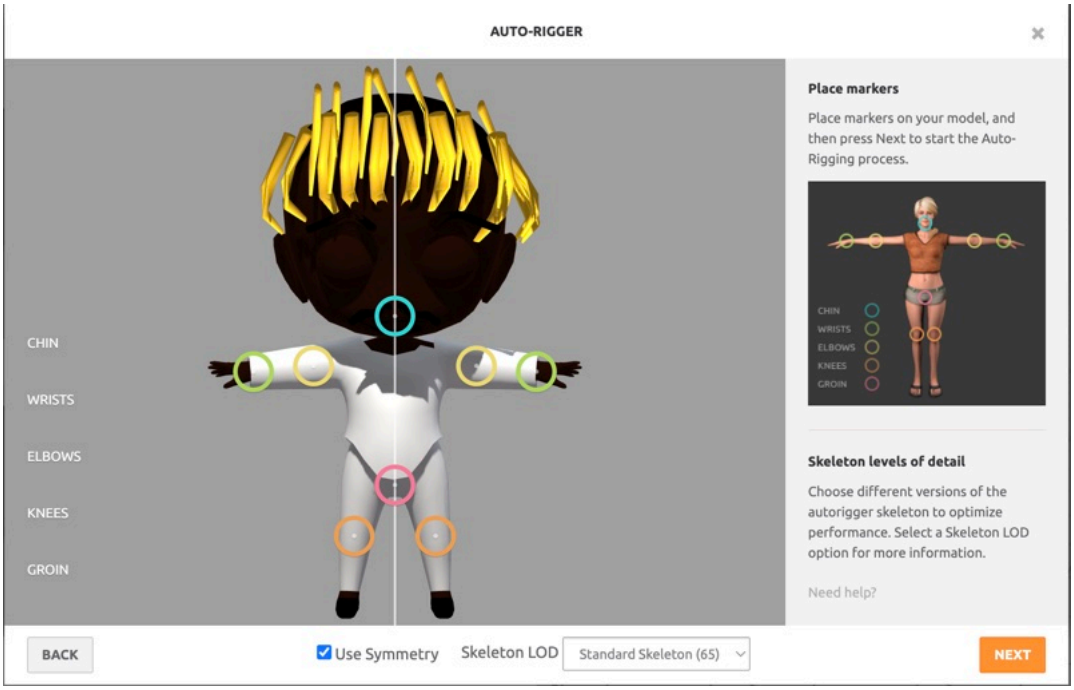


Рис. 4.24 – Створення скелетної анімації за допомогою платформи *Mixamo*

Джерелом просторового звуку є цифрові віртуальні об’єкти. Відповідно до різних переміщень, наближень цифрових об’єктів до точки спостереження змінюються параметри звукової хвилі, напр., інтенсивність, направленість тощо.

Для реалізації просторового звуку в *VR*-застосунку, спочатку необхідно завантажити та налаштувати *Resonance Audio SDK*. Покрокова інструкція наведена у розділі 4.1.

Наступним необхідно імпортувати *.mp3* файл, зокрема обрати *Assets- > Import New Asset…* та додати компоненту *Audio Source* у вікні *Inspector* для *3D*-об’єкта (напр., *Human*). Далі необхідно обрати імпортований *.mp3* файл для параметра *Audio Generator*, для параметра *Output* встановити *Master (ResonanceAudioMixer)*, увімкнути параметри *Spatialize* та *Spatialize Post Effects*, для параметра *Spatial Blend* встановити 1(*3D*) та налаштувати інші параметри за необхідності (рис. 4.25).

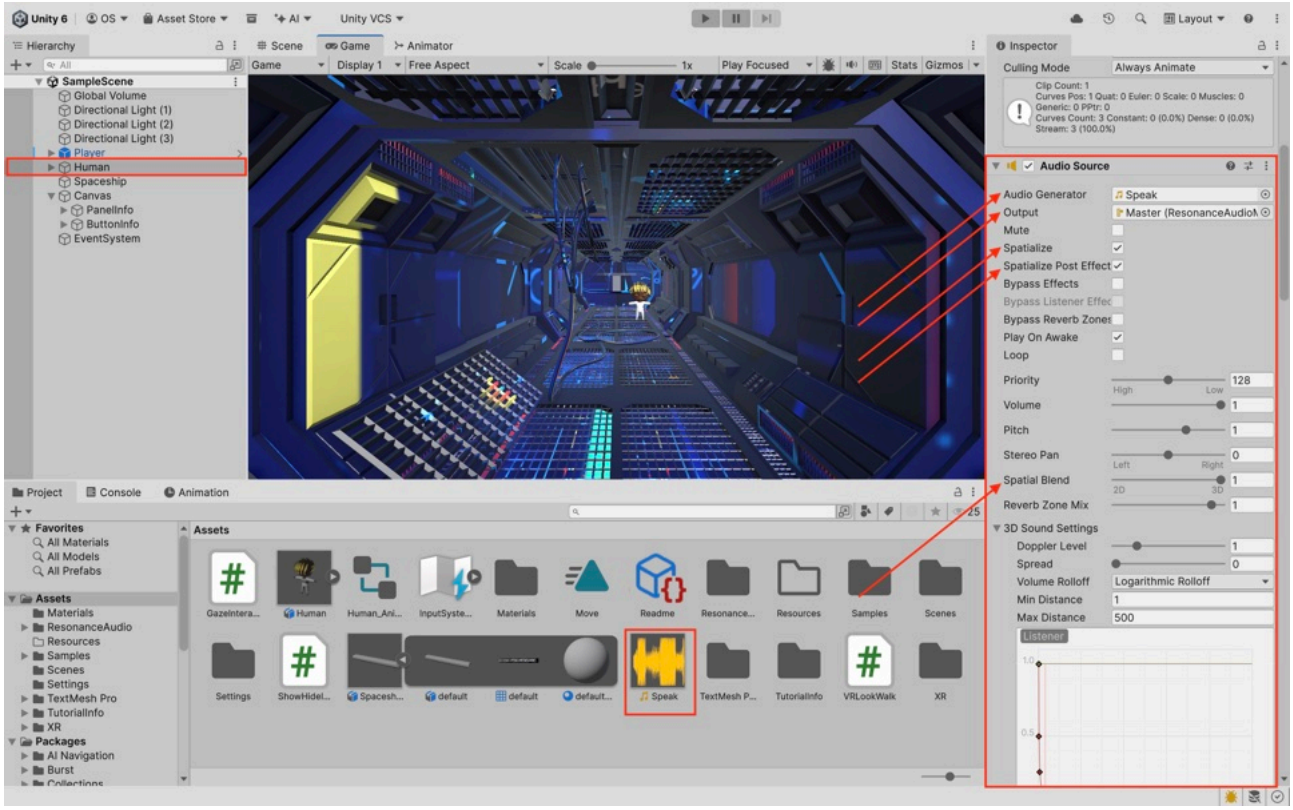


Рис. 4.25 – Налаштування компоненти *Audio Source*

Наступним необхідно додати компоненту *ResonanceAudioSource* (рис. 4.26) для налаштування розширених параметрів просторового звуку.

Resonance Audio SDK реалізує технології оброблення та відтворення просторового звуку на основі заданих параметрів цифрового середовища, які реалізовані в *Unity* в компоненті *ResonanceAudioRoom*. Параметрами налаштування *ResonanceAudioRoom* є матеріали поверхонь середовища, відбивна здатність, модифікатори реверберації та розмір середовища.

Для шести поверхонь цифрового середовища (ліва, права, передня та задня стіни, підлога, стеля) можна призначити акустичні матеріали. Такими акустичними матеріалами для компоненти *ResonanceAudioRoom* є чистий або пофарбований бетонний блок, чиста або пофарбована цегла, полірований бетон або плитка, акустична плитка для стелі, лінолеум на бетонній підлозі, паркет на бетонній підлозі, тонке або товсте скло, скловолоконний утеплювач, тонка або груба штукатурка, фанерна панель, важка завіса, шпакльована поверхня, дерев'яна стеля або панель, поверхня з води або льоду, прозорий матеріал, трава, мармур, метал. Кожен акустичний матеріал визначає ступінь поглинання, відбиття звукових хвиль різних частот. Напр., матеріал важка завіса поглинає більшість високих частот звукової хвилі, що призводить до відтворення «сухого» звуку в цифровому середовищі, а, напр., матеріал полірований бетон або плитка відбивають більшість звукових хвиль на різних частотах, що призводить до відтворення ехо в середовищі.

Параметр відбивна здатність компоненти *ResonanceAudioRoom* визначає силу відбивання звукової хвилі у цифровому середовищі, тим самим визначаючи розмір, форму середовища. При зменшенні значення параметра відбувається відтворення звуків, що можна порівняти зі звуками в тісно замкнутому невеликому приміщенні.

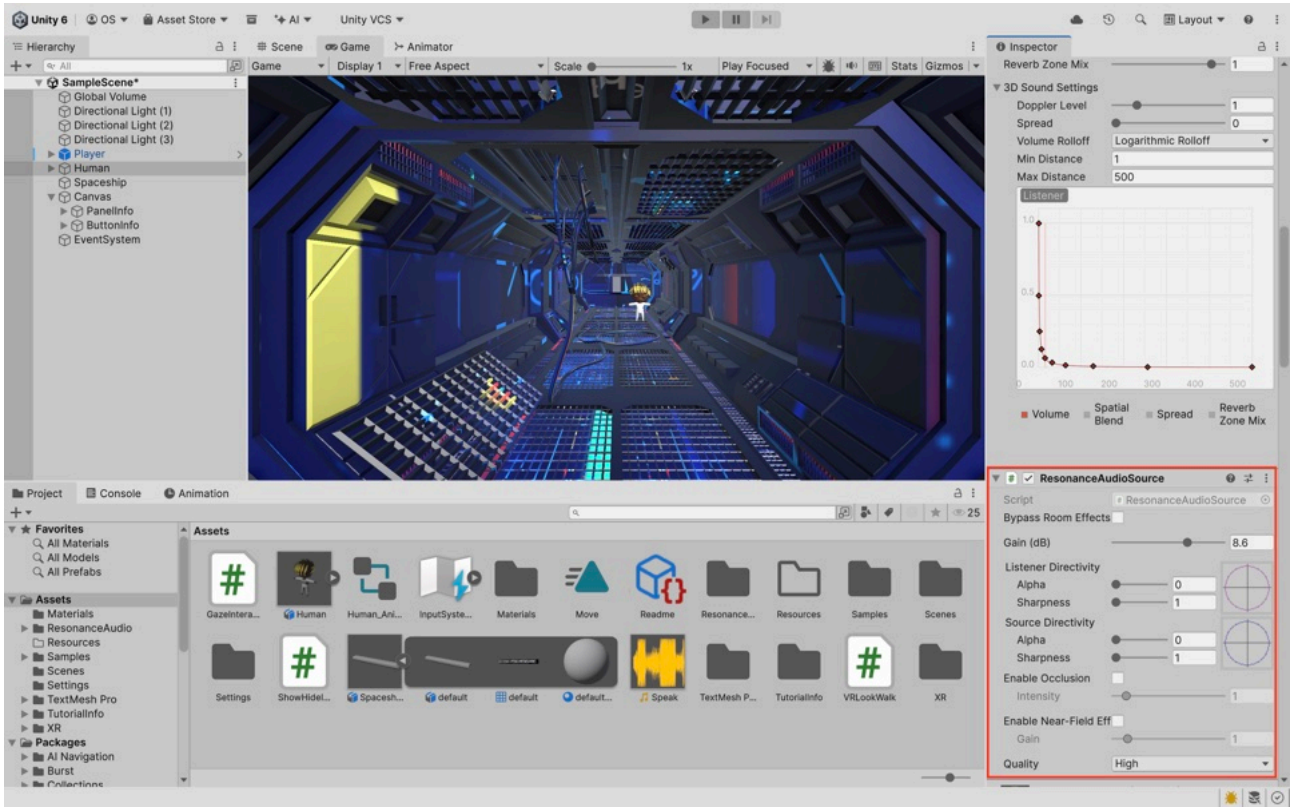


Рис. 4.26 – Налаштування компоненти *ResonanceAudioSource*

Параметрами налаштування реверберації є гучність відтворення звукових ефектів у середовищі, інтенсивність та тривалість. Інтенсивність реверберації визначає кількість низьких або високих частот у реверберації цифрового середовища, які характеризуються різною швидкістю згасання. Цей параметр визначає, наскільки наповнене середовище. Напр., при зменшенні інтенсивності реверберації відбувається відтворення звуків, що можна порівняти з приміщенням, наповненим багатьма предметами або людьми. За замовченням тривалість реверберації встановлюється на основі матеріалів поверхонь та розмірів середовища, які були обрані для компоненти *ResonanceAudioRoom*. Налаштування тривалості реверберації дозволяє збільшити або зменшити довжину реверберації.

Параметр розмір середовища компоненти *ResonanceAudioRoom* визначає межі, при перетинанні яких відбувається вмикання або вимкнення налаштувань просторового звуку.

3D-об’єкт *Spaceship* є простором, де відтворюються просторові звуки. Для об’єкта *Spaceship* у вікні *Inspector* необхідно додати компоненту *ResonanceAudioRoom*. У вікні сцени з’являється жовтий куб, який показує початкові межі простору, що необхідно налаштувати за допомогою параметра *Size*. Наступним необхідно визначити акустичні матеріали поверхонь, відбивну здатність та параметри реверберації цифрового середовища (рис. 4.27).

Отже, при рухах *3D*-об’єкта (*Human*) та самого користувача в *VR*-застосунку будуть відтворюватись просторові звуки.

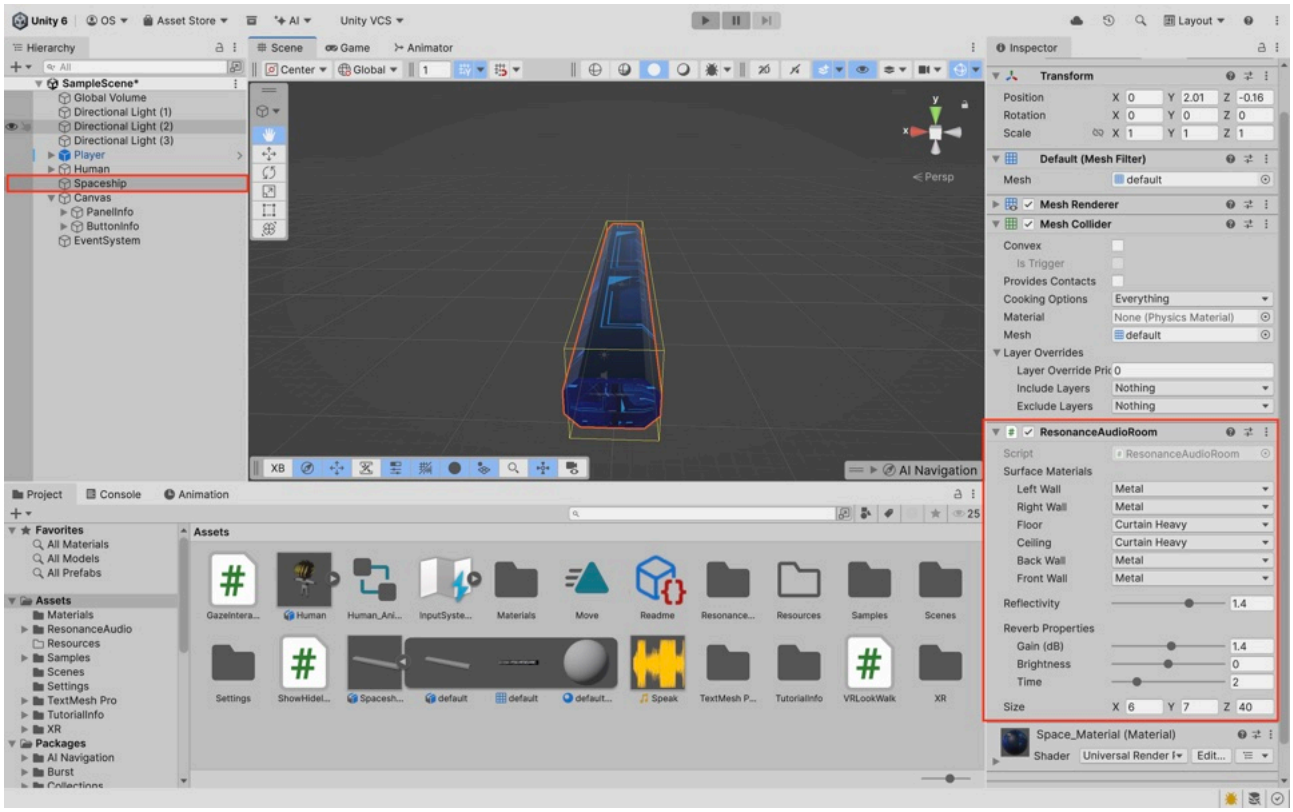


Рис. 4.27 – Налаштування компоненти *ResonanceAudioRoom*

4.2.4 Збирання та розгортання *VR*-застосунку

Процес збирання та розгортання *VR*-застосунку на основі *Google Cardboard SDK* в середовищі *Unity* є однаковим в порівнянні з процесом збирання та розгортання *AR*- та *MR*-застосунків. Покрокові інструкції описані в розділі 2.2.5.

4.2.5 Демонстрація роботи *VR*-застосунків

На рис. 4.28 представлене шаблонне *VR*-середовище *Google Cardboard*. Користувач може здійснювати обертальні рухи з 3 *DoF*, обирати (наводити погляд) та взаємодіяти (затримувати погляд) з *3D*-об'єктами. Результат роботи розробленого *VR*-застосунку представлено на рис. 4.29. Користувач може здійснювати обертальні та поступальні рухи з 6 *DoF*, обирати (наводити погляд) та взаємодіяти (затримувати погляд) з *3D*-об'єктами.

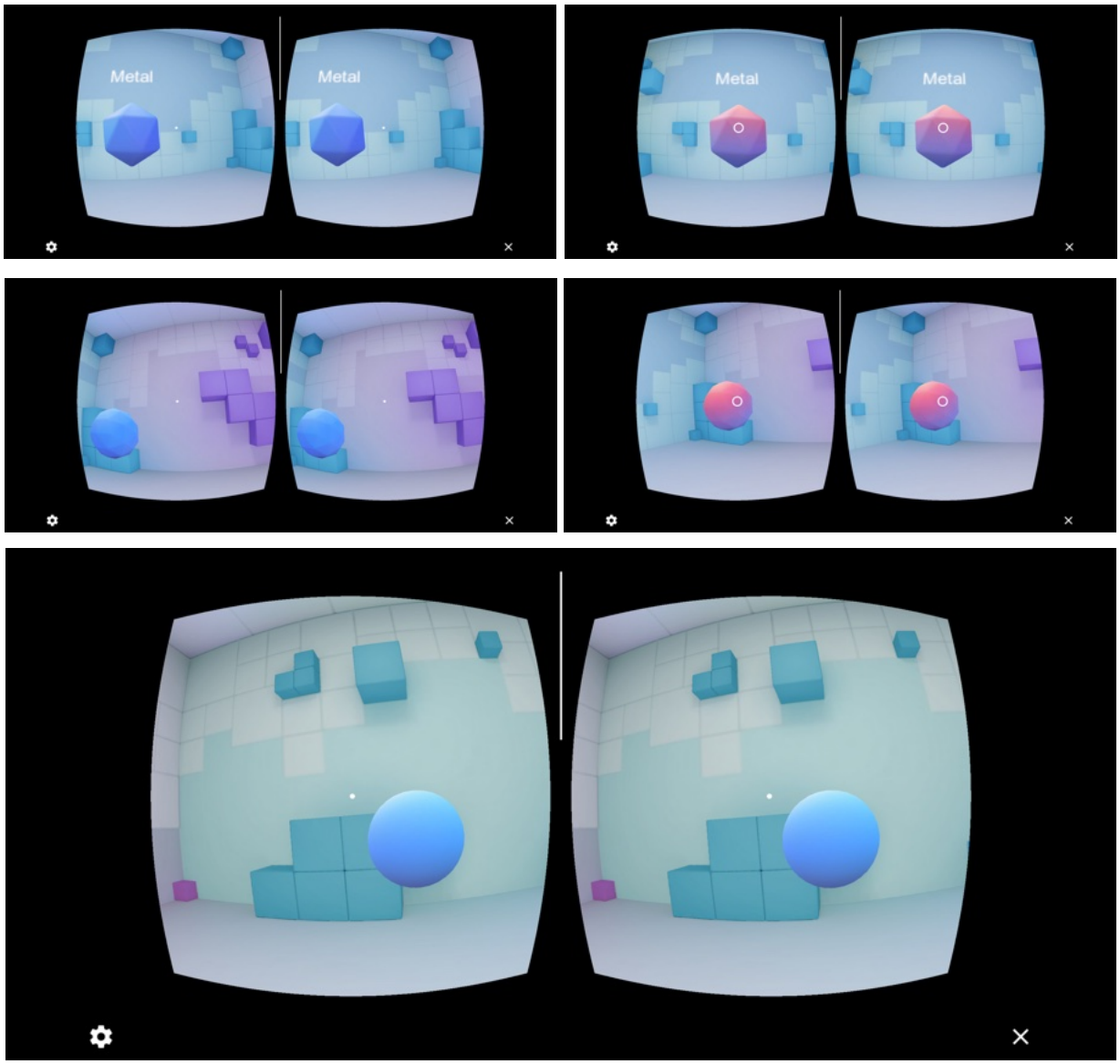


Рис. 4.28 – Приклад роботи *VR*-застосунку на основі шаблонного середовища *Google Cardboard SDK*

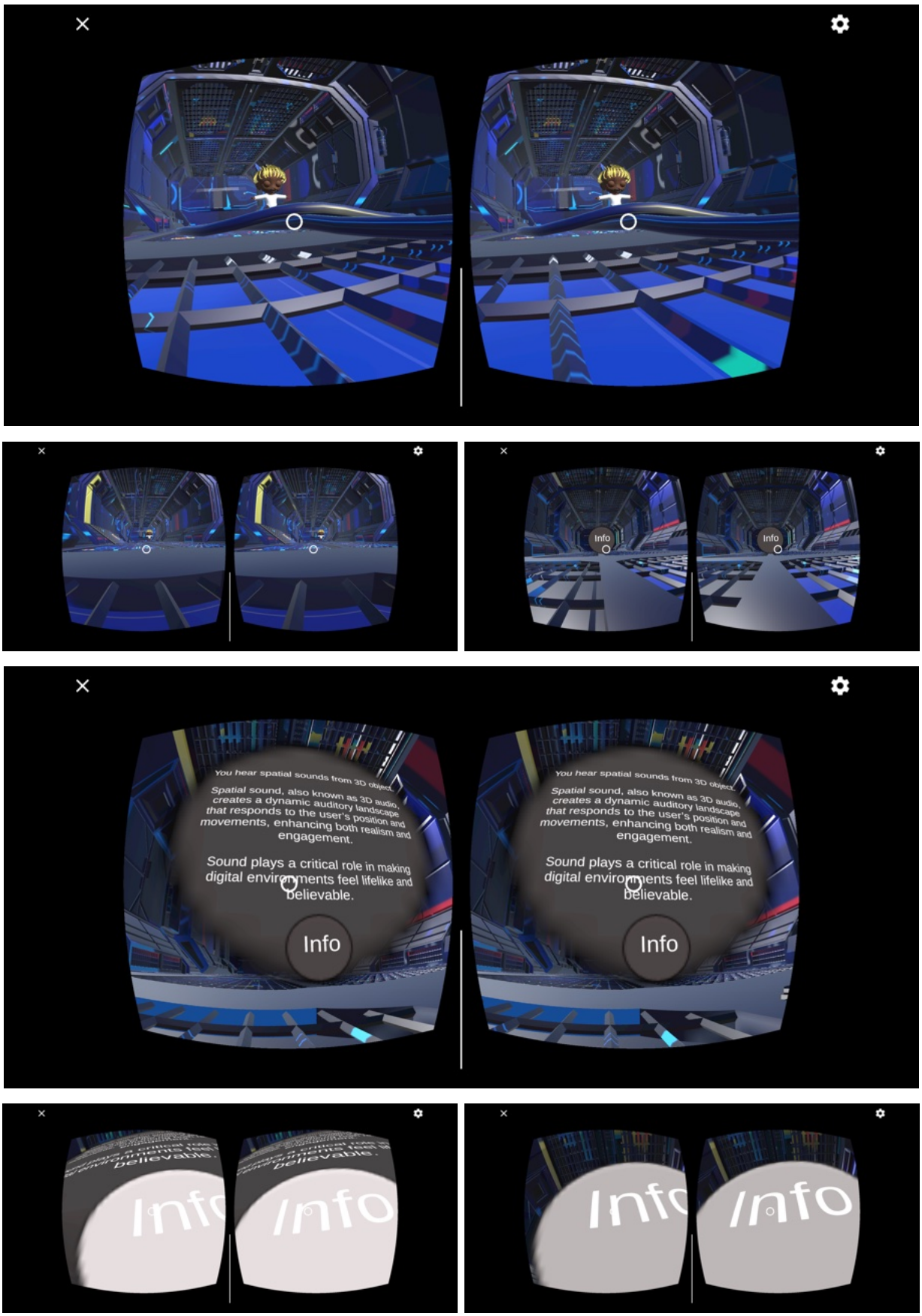


Рис. 4.29 – Приклад роботи розробленого *VR*-застосунку

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яке призначення *XR Interaction Toolkit* у *VR*-застосунках?
2. Які механіки взаємодії можна реалізувати в *VR*-застосунках за допомогою *XR Interaction Toolkit*?
3. Які *SDK* належать до платформо-залежних засобів реалізації *VR*-функцій?
4. У чому полягає відмінність між *XR Interaction Toolkit* та *VR SDK*?
5. У чому полягає відмінність між *OpenXR* та платформо-залежними *VR SDK*?
6. Які базові *VR*-функції забезпечує *OpenXR*?
7. Які типи *XR*-застосунків можна розробляти з використанням *OpenXR*?
8. Як *OpenXR* використовується в нативних середовищах розроблення?
9. Які кроки необхідно виконати, щоб використовувати *OpenXR* в *Unity*-проєкті?
10. Які *VR*-плагіни можна підключити в *Unity*-проєкт за допомогою *XR Plug-in Management*?
11. Які основні функції забезпечує *Google Cardboard SDK* у мобільних *VR*-застосунках?
12. Як реалізується взаємодія користувача у *Google Cardboard SDK*?
13. З якими середовищами розроблення та платформами можна інтегрувати *Google Cardboard SDK* для створення мобільних *VR*-додатків?
14. Які кроки необхідно виконати, щоб використовувати *Google Cardboard SDK* в *Unity*-проєкті?
15. Які *VR*-функції підтримують шаблони *Hello Cardboard* та *VRMode*?
16. Які типи відстеження підтримуються у *VRMode*-шаблоні?
17. Які платформи підтримує *Resonance Audio SDK* для реалізації просторового звуку?
18. Які кроки необхідно виконати, щоб використовувати *Resonance Audio SDK* в *Unity*-проєкті?
19. Яке призначення компоненти *ResonanceAudioSource* у складі *Resonance Audio SDK* та які додаткові параметри звуку вона реалізує порівняно зі стандартною компонентою *Unity Audio Source*?

20. Яке призначення компоненти *ResonanceAudioRoom* у складі *Resonance Audio SDK*? У чому полягає відмінність між компонентами *ResonanceAudioRoom* та *ResonanceAudioReverbProbe* щодо реалізації реверберації звуку?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Unity Manual. URL : https://docs.unity3d.com/Manual/.
2. ARKit Documentation. URL : https://developer.apple.com/documentation/arkit/.
3. ARCore Documentation. URL : https://developers.google.com/ar/develop.
4. Open XR Documentation and Extensions: Procedures and Conventions. URL : https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html.
5. Google Gardboard. URL : https://developers.google.com/cardboard/develop.
6. Resonance Audio SDK. URL : https://resonance-audio.github.io/resonance-audio/.