Лабораторна робота №3

Тема: Анімація, музична композиція та техніка розпізнавання обличь в AR

Мета роботи: набути практичні навички з використання технології доповненої реальності при створенні ігор

Теоретичні відомості: лек.2, лек.3.
https://www.youtube.com/watch?v=OydZM5MXLic
Приклад готового проєкта AR-масок в Unity:
https://github.com/dilmerv/FaceTrackingDemo

Додатково: міні курс по створенню AR-масок https://learn.unity.com/course/create-with-ar-face-filters?uv=2021.3

Методичні рекомендації для:

1. Порядку додавання AR-маски на обличчя з декількома параметрами.
2. Налаштування параметрів AR (Unity AR Setup).
3. Відстеження обличчя в Unity (Face tracking in Unity).

Перейти до завдання на лабораторну роботу.

Порядок додавання AR-маски на обличчя з декількома параметрами.

Здійснити заміну масок для обличчя

Створіть проєкт Unity та налаштуйте параметри AR (див.опис нижче).

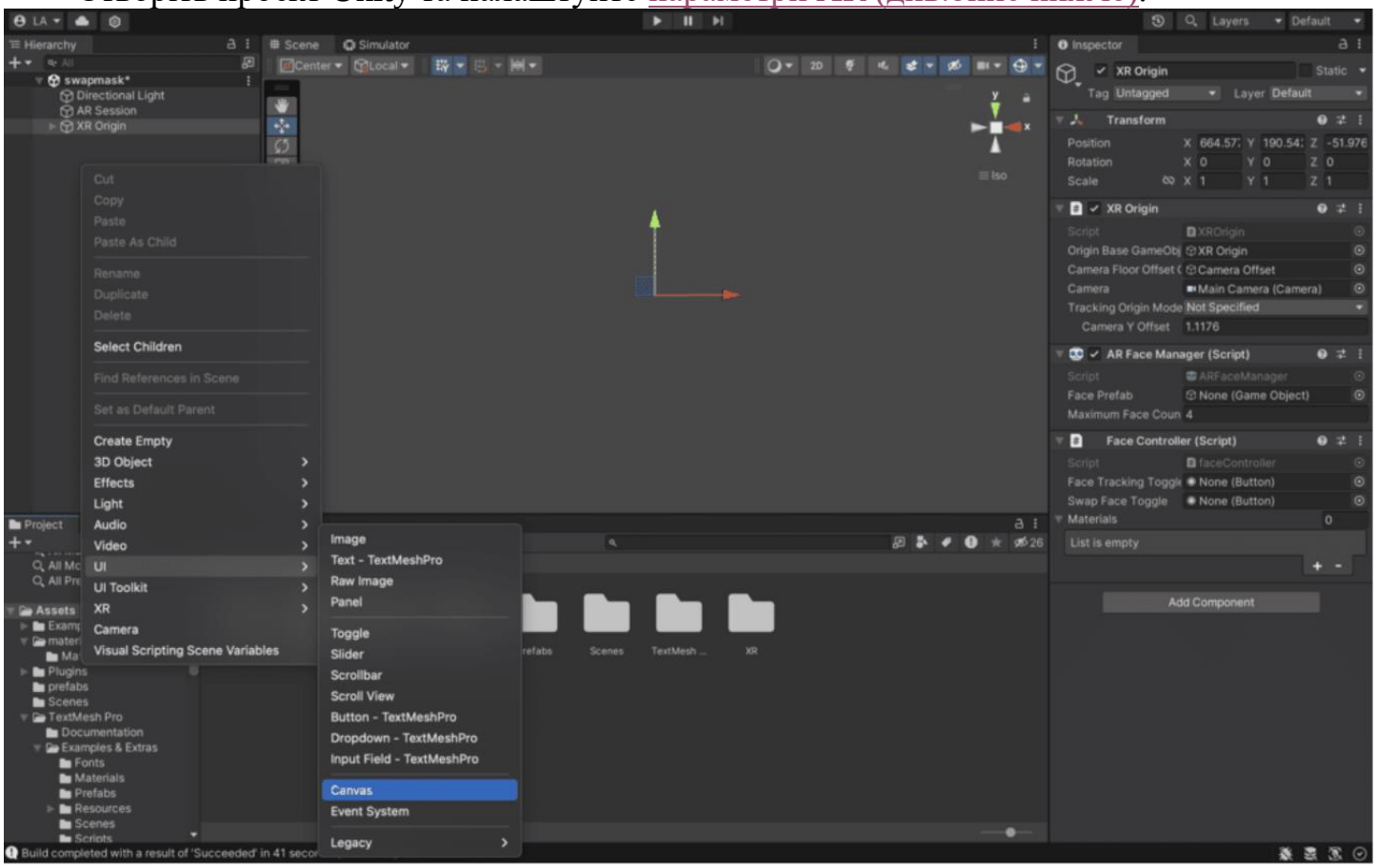

Рис.1.

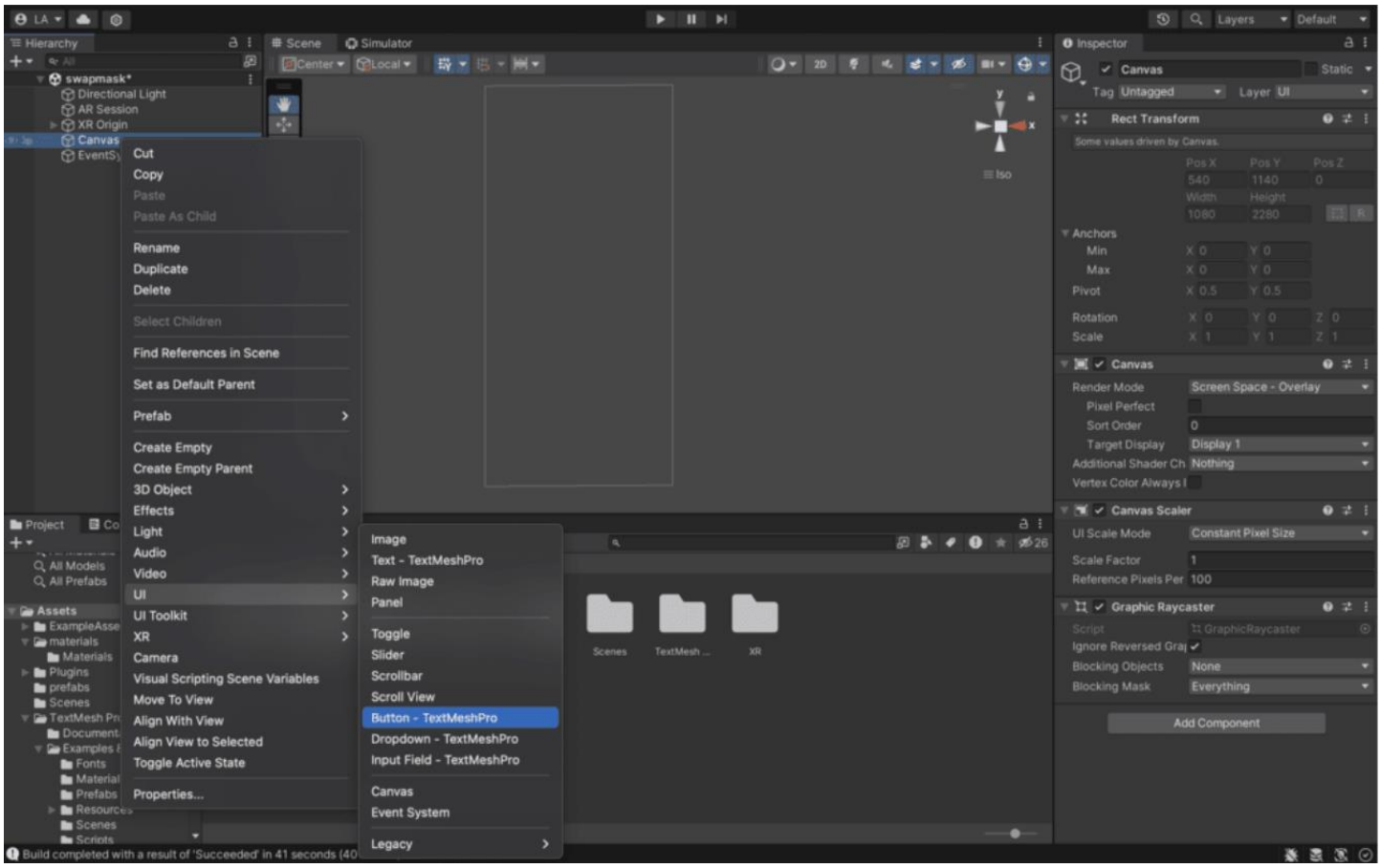
Hierarchy
Scene
Simulator
Inspector
swapmask*
Directional Light
AR Session
XR Origin
Canvas
Cut
Copy
Paste
Paste As Child
Rename
Duplicate
Delete
Select Children
Find References in Scene
Set as Default Parent
Prefab
Create Empty
Create Empty Parent
3D Object
Effects
Light
Audio
Video
UI
UI Toolkit
XR
Camera
Visual Scripting Scene Variables
Move To View
Align With View
Align View to Selected
Toggle Active State
Properties...
Image
Text - TextMeshPro
Raw Image
Panel
Toggle
Slider
Scrollbar
Scroll View
Button - TextMeshPro
Dropdown - TextMeshPro
Input Field - TextMeshPro
Canvas
Event System
Legacy
Rect Transform
Canvas
Canvas Scaler
Graphic Raycaster
Add Component

Рис.2.

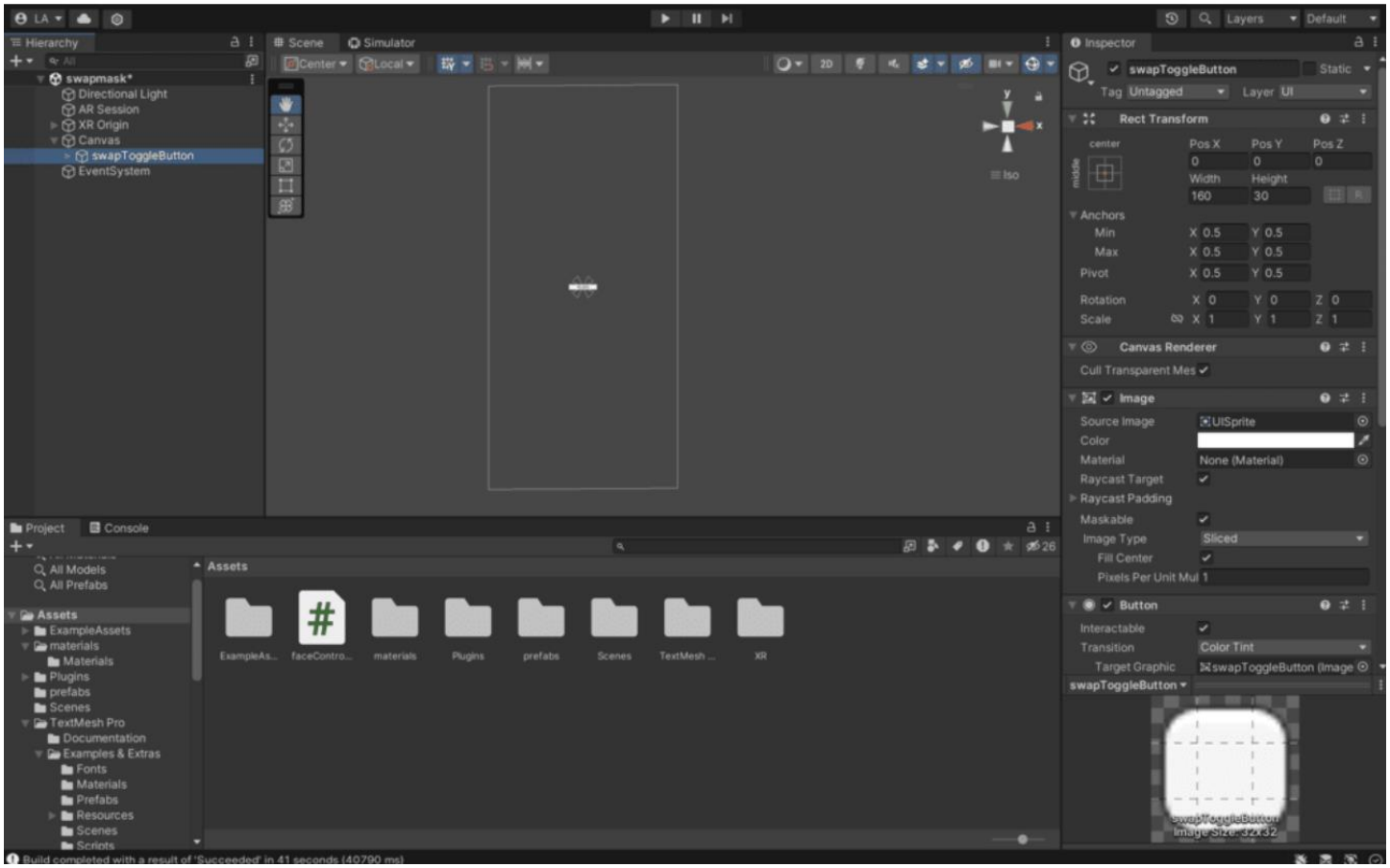
Hierarchy
Scene
Simulator
Inspector
swapmask*
Directional Light
AR Session
XR Origin
Canvas
swapToggleButton
EventSystem
Project
Console
Assets
Rect Transform
Canvas Renderer
Image
Button
Add Component

Рис.3.

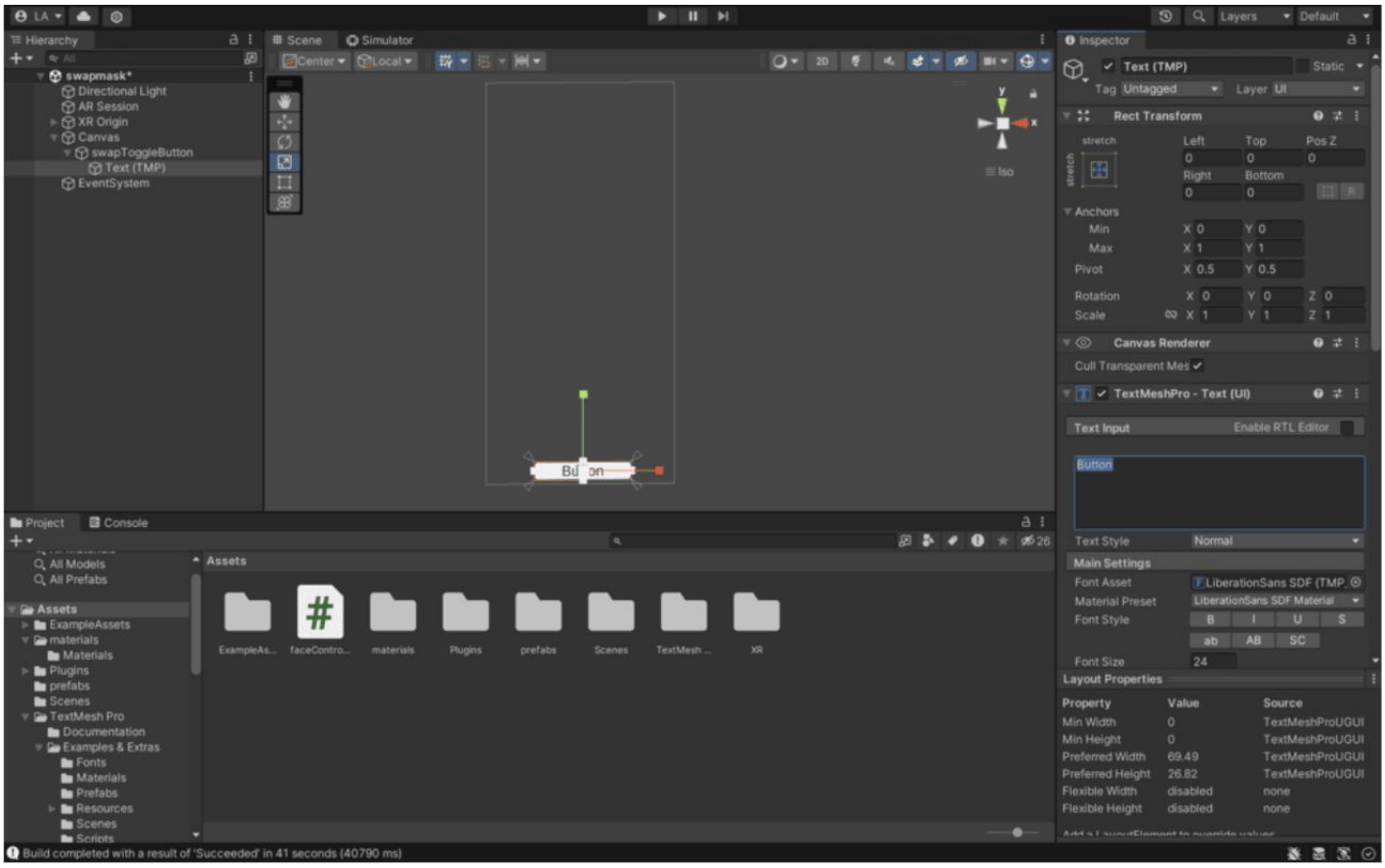

Рис.4.

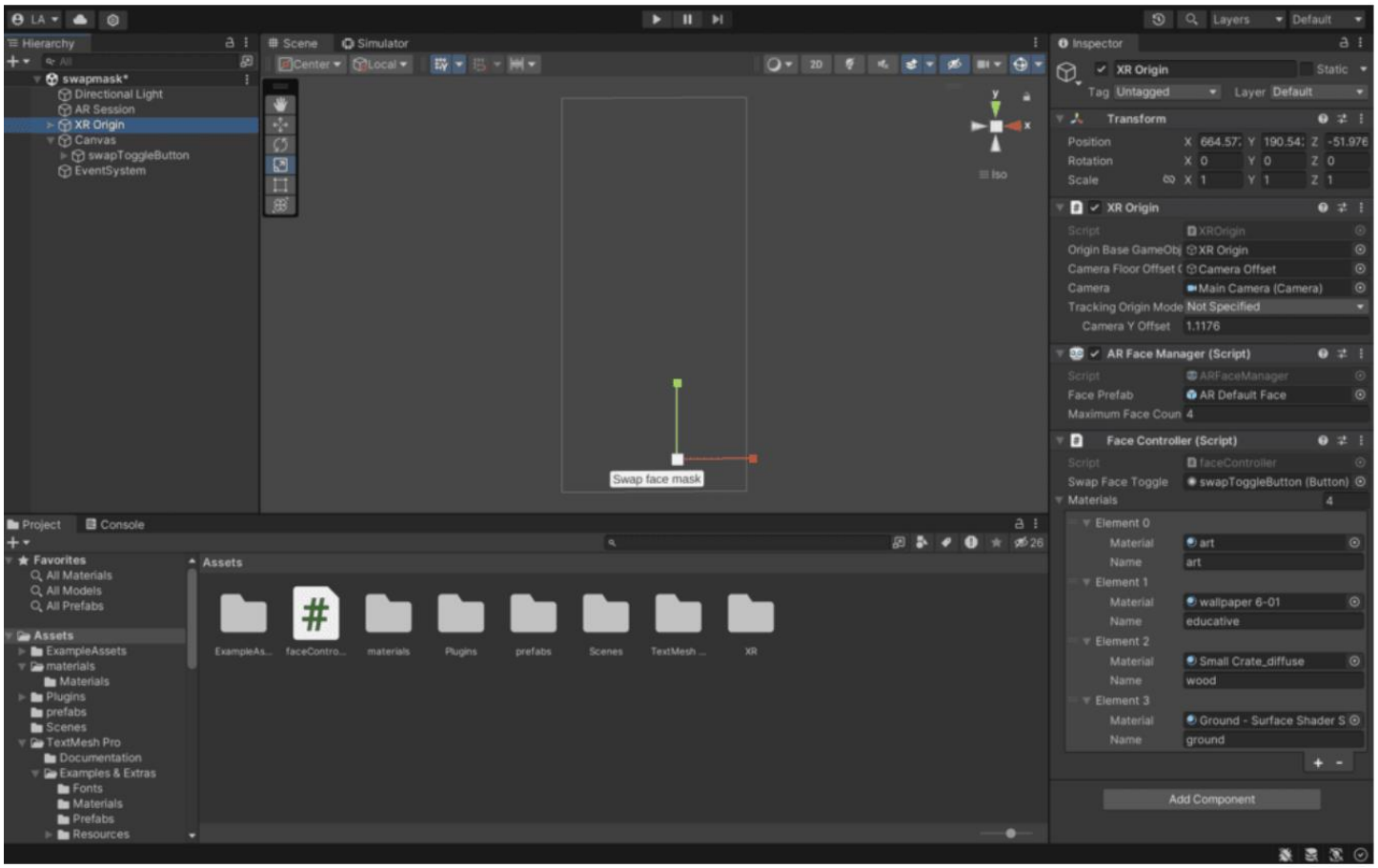

Рис.5.

Canvas

Canvas (полотно) — це компонент інтерфейсу користувача, який діє як контейнер для таких елементів інтерфейсу, як кнопки, текст, зображення та інші елементи керування

інтерфейсом користувача . Canvas відтворює ці елементи інтерфейсу користувача в ігровому світі, накладаючи їх на сцену. Це особливо важливо в програмах AR, де ви повинні надати інтерактивні елементи, які користувачі можуть бачити та взаємодіяти з ними в розширеному середовищі.

Canvas використовується для створення користувацьких інтерфейсів, які можуть реагувати на взаємодії користувача, відображати інформацію та надавати елементи керування в середовищі AR.

Тут створіть полотно та додайте кнопку під назвою "swapToggleButton".

Фейс менеджер

Додайте компонент менеджера обличчя в джерело XR для відстеження обличчя (див.опис нижче) AR – це компонент, який надається базовим пакетом AR Unity, який полегшує відстеження обличчя та керування ним у програмах AR. Відстеження обличчя передбачає ідентифікацію та відстеження рис і виразу обличчя в режимі реального часу за допомогою камери пристрою.

Скрипт контролера обличчя

XR origin, додайте компонент і створіть новий сценарій під назвою «faceController». Сценарій керуватиме заміною матеріалів обличчя на обличчя AR і відповідним чином оновлюватиме текст кнопки інтерфейсу.

```
1   using UnityEngine.UI;
2   using UnityEngine;
3   using UnityEngine.XR.ARFoundation;
4
5
6   [RequireComponent(typeof(ARFaceManager))]
7
8   public class faceController : MonoBehaviour
9   {
10
11      [SerializeField]
12      private Button swapFaceToggle;
13      private ARFaceManager arFaceManager;
14      private bool faceTrackingOn = true;
15      private int swapCounter = 0;
16
17      [SerializeField]
18      public faceMaterial[] materials;
19
20
21      void Awake()
22      {
23          arFaceManager = GetComponent<ARFaceManager>();
24          swapFaceToggle.onClick.AddListener(swapFaces);
25          arFaceManager.facePrefab.GetComponent<MeshRenderer>().material = materials[0].Material;
26      }
27
28      void swapFaces()
29      {
30          swapCounter = swapCounter == materials.Length - 1 ? 0 : swapCounter + 1;
31
32          foreach (ARFace face in arFaceManager.trackables)
33          {
34              face.GetComponent<MeshRenderer>().material = materials[swapCounter].Material;
35          }
36
37          swapFaceToggle.GetComponentInChildren<Text>().text = $"Face Tracking ({materials[swapCounter].Name})";
38      }
39
40      [System.Serializable]
41      public class faceMaterial
42      {
43          public Material Material;
44          public string Name;
45      }
46  }
```

Рис.5.

Пояснення

- Рядки 1–3: ці рядки імпортують необхідні простори імен із механізму Unity для доступу до необхідних класів і функцій для роботи з елементами інтерфейсу користувача, ігровими об’єктами та основою AR.

- Рядок 6: цей атрибут гарантує, що сценарій вимагає *ARFaceManager*прикріплення компонента до того самого GameObject. Якщо компонент відсутній, Unity автоматично додасть його.
- Рядок 8: цей рядок визначає початок класу *faceController*, який успадковує *MonoBehaviour*базовий клас для сценаріїв Unity.
- Рядки 11–12: це створює серіалізоване приватне поле з назвою *swapFaceToggle*типу *Button*. Атрибут *SerializeField*робить це поле видимим в інспекторі Unity, зберігаючи його приватним.
- Рядок 13: оголошено приватне поле *arFaceManager*типу . *ARFaceManager*Це збереже посилання на *ARFaceManager*компонент.
- Рядок 14: приватне логічне поле *faceTrackingOn*оголошується та ініціалізується значенням *true*.
- Рядок 15: приватне ціле число *swapCounter*оголошується та ініціалізується значенням *0*.
- Рядки 17–18: оголошено серіалізоване публічне поле з назвою *materials*, яке є масивом *faceMaterial*об’єктів. У цьому полі зберігатимуться різні матеріали обличчя, які можна міняти місцями.
- Рядки 21–26: метод *Awake*викликається під час ініціалізації екземпляра сценарію. У цьому методі:
  - *arFaceManager*призначається посилання на *ARFaceManager*компонент, прикріплений до того ж GameObject, що й сценарій.
  - Подія *swapFaceToggle*кнопки *onClick*підписується на *swapFaces*метод, тобто *swapFaces*метод буде викликаний, коли кнопка натиснута.
  - Матеріалом *facePrefab* встановлюється матеріал із першого елемента масиву *materials*.
- Рядки 28–38: Метод *swapFaces*визначається як:
  - Збільшується *swapCounter*, повертаючись до 0, якщо досягає кінця масиву *materials*.
  - Цикл виконує ітерацію по кожному *ARFace*об’єкту в *arFaceManager.trackables*колекції (що представляє відстежені обличчя), а матеріал компонента обличчя *MeshRenderer*встановлюється на матеріал за поточним *swapCounter*індексом.
  - Текст кнопки (*swapFaceToggle*) оновлено для відображення поточної назви матеріалу лицьової поверхні.
- Рядки 40–46: Вкладений клас із назвою *faceMaterial*визначається в *faceController*класі:
  - Цей клас позначено як *[System.Serializable]*, що дозволяє відображати його екземпляри в Unity Inspector.
  - Він містить два відкритих поля: *Material*(для збереження матеріалу) і *Name*(для збереження назви матеріалу).

Налаштуйте кнопку

Використовуючи інструменти трансформації у верхній лівій частині сцени, перемістіть кнопку в нижню частину полотна. У вкладках ієрархії у розкривному списку виберіть «Текст (TMP)» і змініть текст на «Змінити маску обличчя».

Контролер обличчя

Перейдіть до джерела XR. На вкладці інспектора ви побачите свій компонент керування обличчям.

- Поруч із перемикачем зміни обличчя перетягніть свою кнопку або виберіть свою кнопку.
- На вкладці «Матеріали» додайте всі матеріали для маски для обличчя , які ви хочете, щоб ваш користувач вибрав.

Налаштування параметрів AR (Unity AR Setup)

Налаштування AR в Unity

Створити проєкт в Unity та виберіть основний шаблон AR.

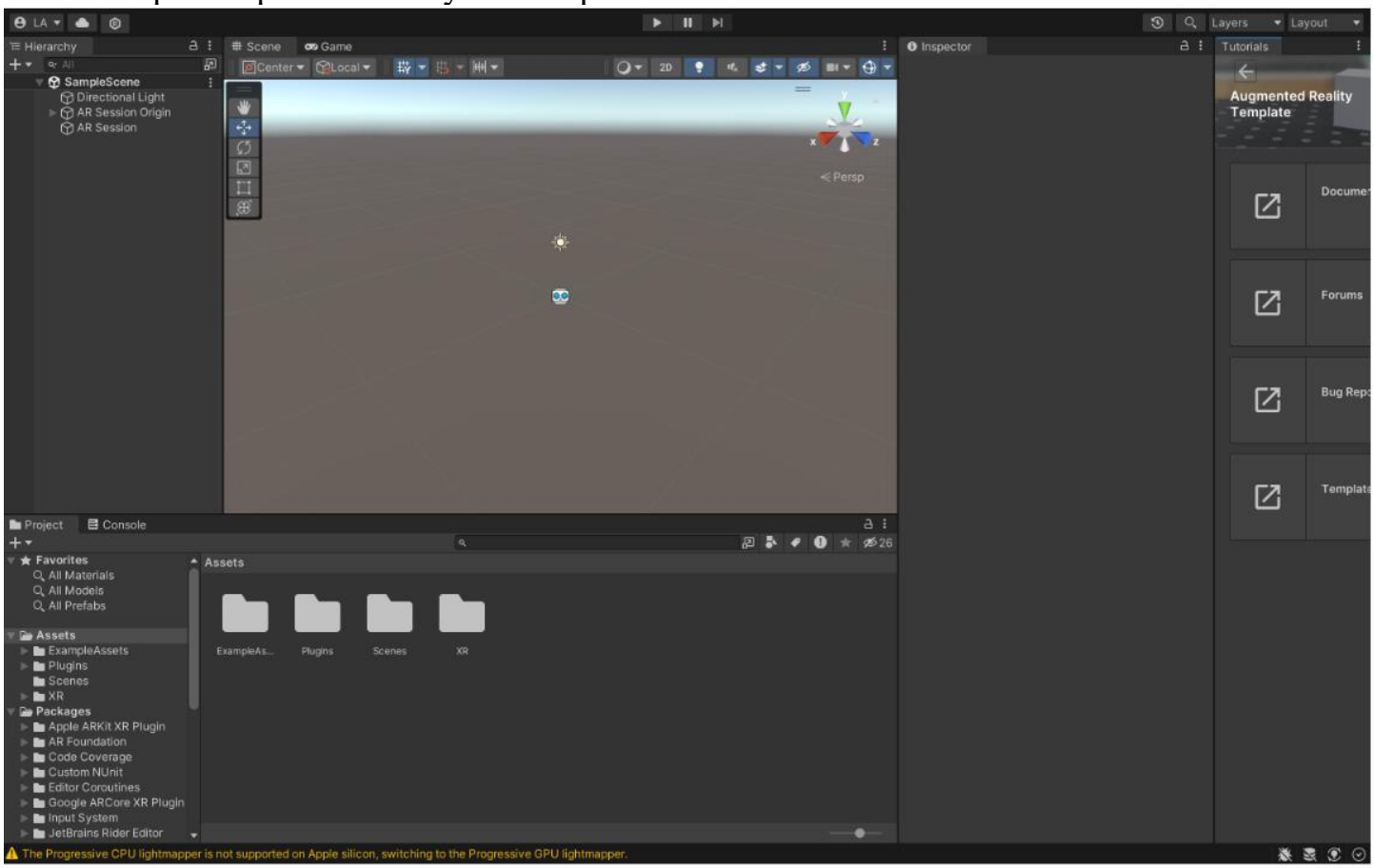

Рис.6.

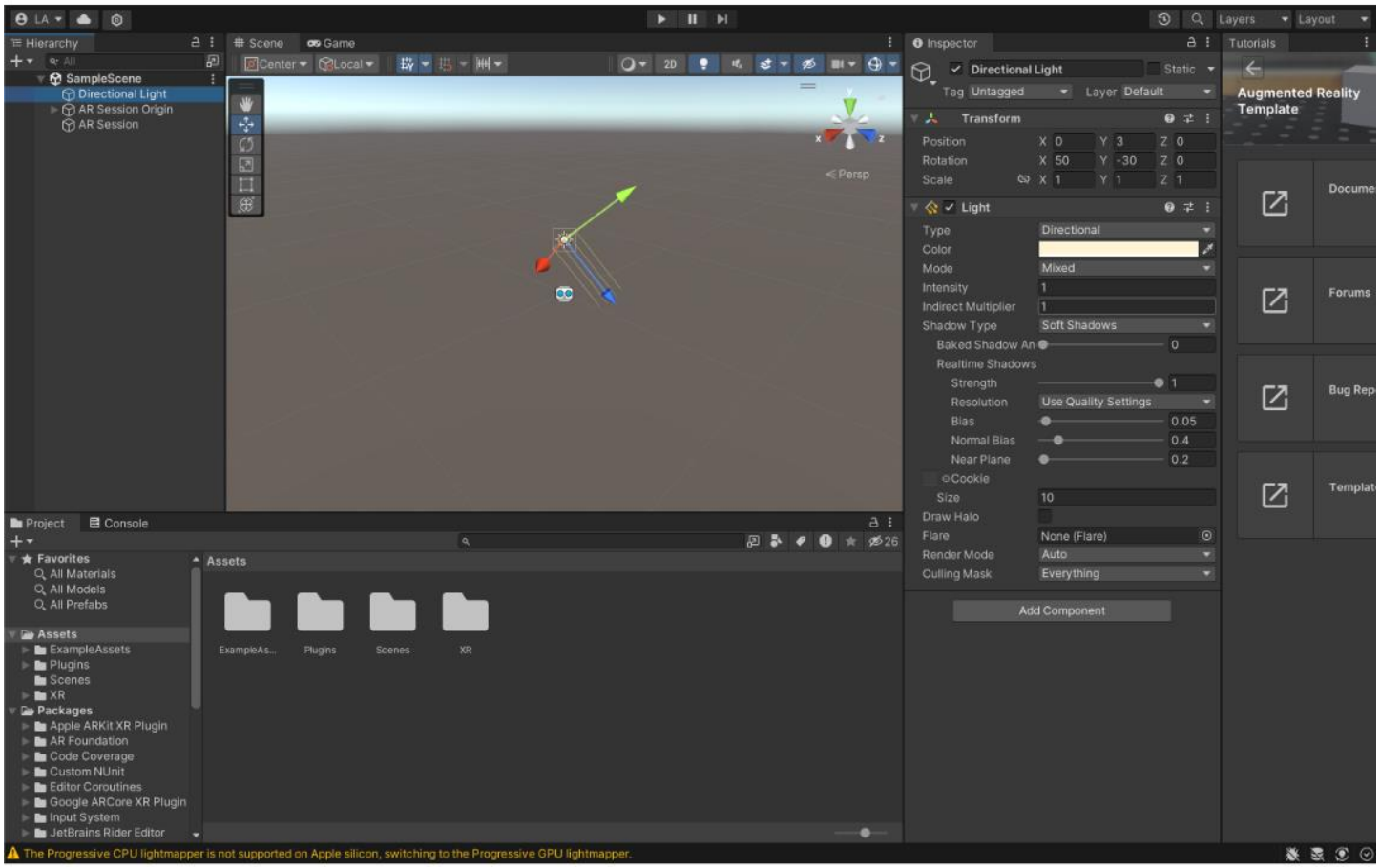
Hierarchy
SampleScene
Directional Light
AR Session Origin
AR Session
Scene
Game
Inspector
Directional Light
Transform
Position
Rotation
Scale
Light
Type
Directional
Color
Mode
Mixed
Intensity
Indirect Multiplier
Shadow Type
Soft Shadows
Realtime Shadows
Strength
Resolution
Use Quality Settings
Bias
Normal Bias
Near Plane
Cookie
Size
Draw Halo
Flare
None (Flare)
Render Mode
Auto
Culling Mask
Everything
Add Component
Tutorials
Augmented Reality Template
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
ExampleAssets
Plugins
Scenes
XR
Packages
Apple ARKit XR Plugin
AR Foundation
Code Coverage
Custom NUnit
Editor Coroutines
Google ARCore XR Plugin
Input System
JetBrains Rider Editor
The Progressive CPU lightmapper is not supported on Apple silicon, switching to the Progressive GPU lightmapper.

Рис.7.

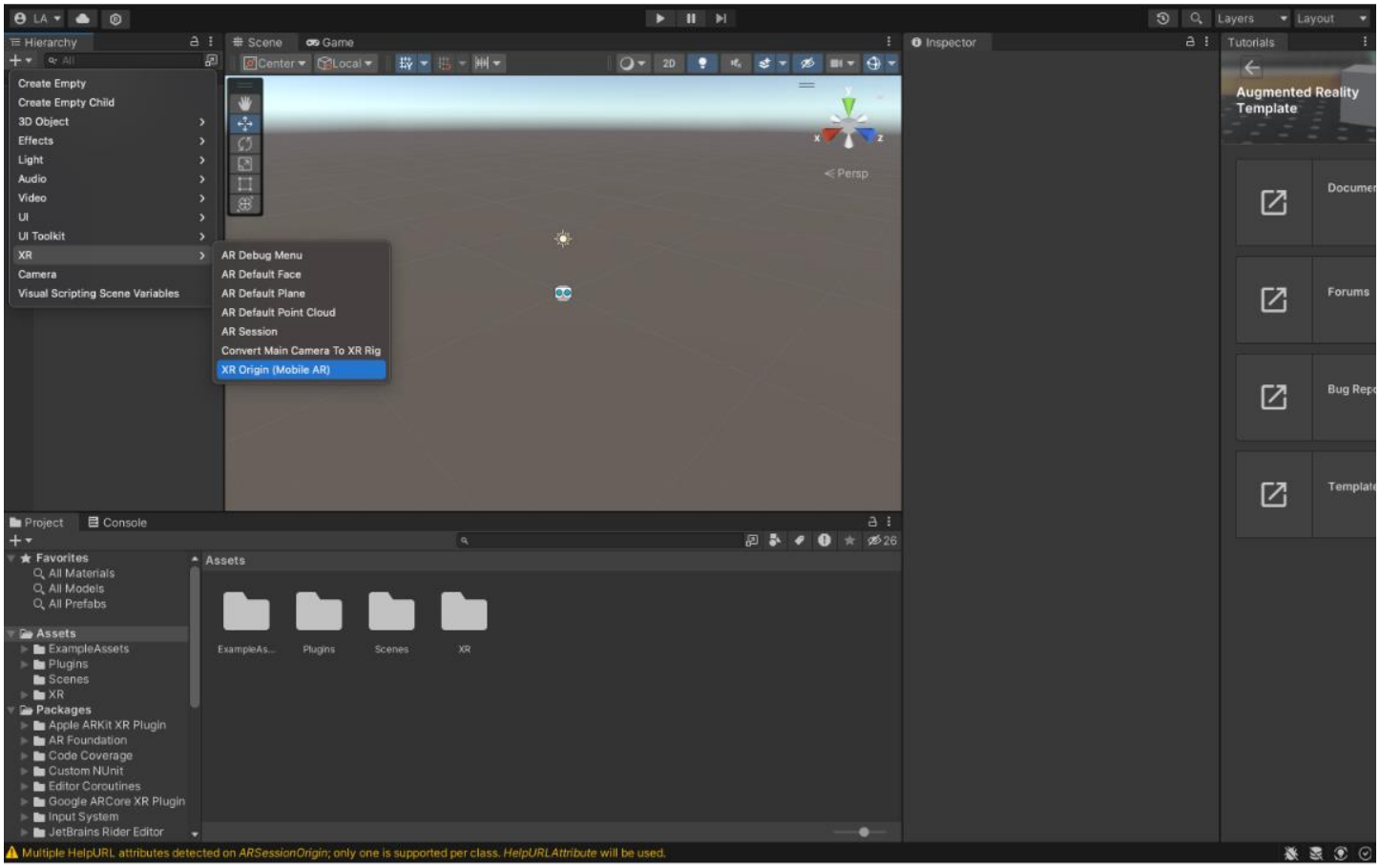
Create Empty
Create Empty Child
3D Object
Effects
Light
Audio
Video
UI
UI Toolkit
XR
Camera
Visual Scripting Scene Variables
AR Debug Menu
AR Default Face
AR Default Plane
AR Default Point Cloud
AR Session
Convert Main Camera To XR Rig
XR Origin (Mobile AR)
Multiple HelpURL attributes detected on ARSessionOrigin; only one is supported per class. HelpURLAttribute will be used.

Рис.8.

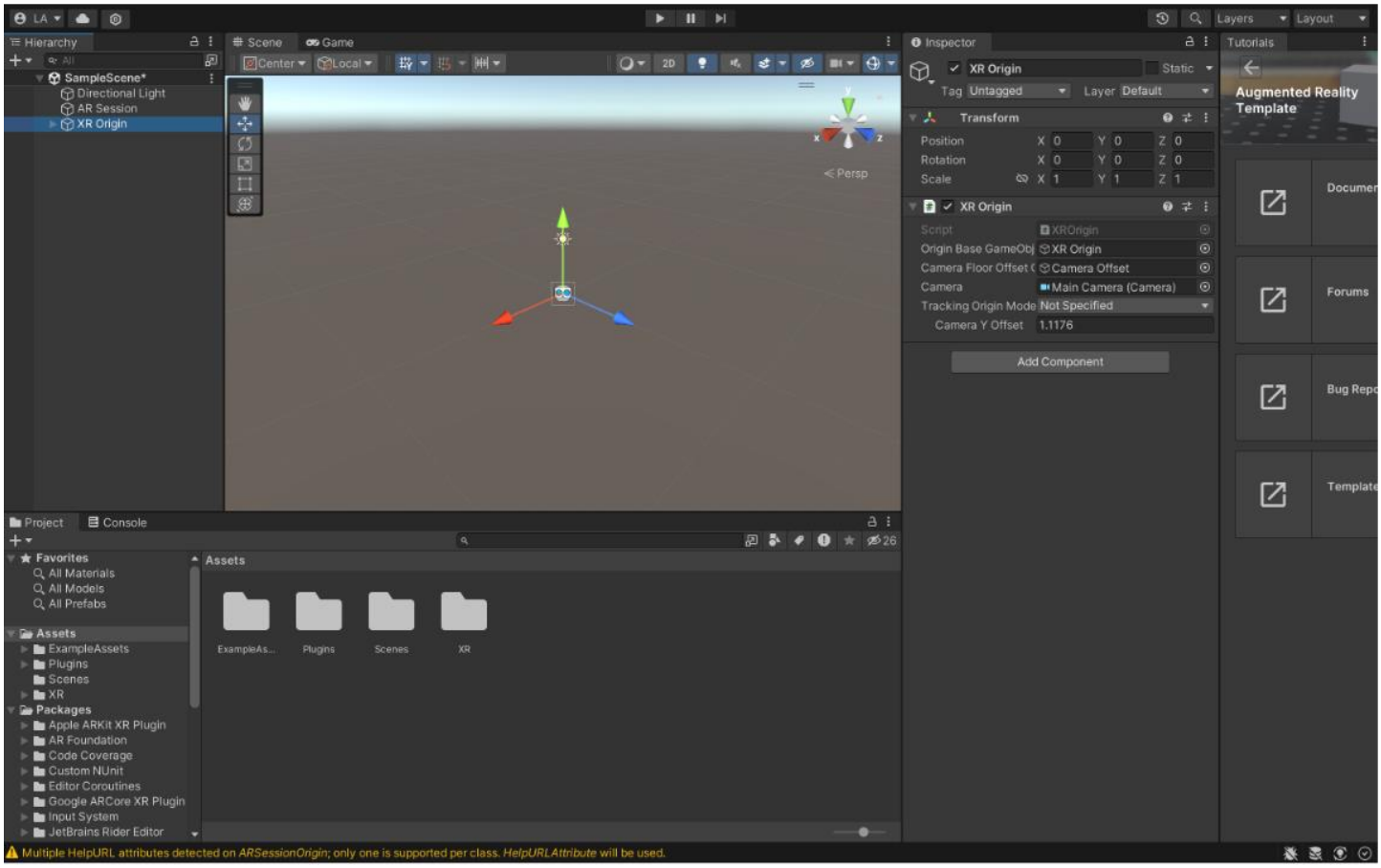

Рис.9.

За замовчуванням для вас буде створено зразок сцени.

Направлене світло

Це тип світла, який імітує сонячне світло та зазвичай використовується для представлення основного джерела світла на вулицях. Він включає такі властивості:

- Тип: ця властивість визначає тип джерела світла. Для спрямованого світла встановлюється тип «Спрямований».
- Колір: властивість кольору дозволяє вибрати колір спрямованого світла. Він визначає відтінок і інтенсивність випромінюваного світла.
- Режим: ця властивість дозволяє вибирати між «У реальному часі» та «Запечений». У режимі Realtime світло обчислюється в реальному часі, тоді як у режимі Baked освітлення попередньо обчислюється для статичних об’єктів під час випікання.
- Інтенсивність: властивість інтенсивності контролює силу світла, випромінюваного спрямованим джерелом світла.
- Непрямий множник: ця властивість регулює інтенсивність непрямого світла , що кидається спрямованим світлом.
- Тип тіні: властивість типу тіні дозволяє вибрати між «Без тіней», «Жорсткі тіні» та «М’які тіні». Він визначає, як спрямоване світло відтворює тіні.
- Намалювати ореол: ця властивість умикає або вимикає відображення відблиску навколо спрямованого світла.
- Відблиск: ця властивість дозволяє призначати спеціальну текстуру відблиску об’єктива спрямованому світлу.
- Режим візуалізації: властивість режиму візуалізації визначає, чи впливає світло на всі об’єкти в сцені, чи воно видиме лише з камери.

- Маска відсічення: властивість маски відсікання визначає, на які шари сцени впливає світло. На об’єкти на шарах, які не включені в маску вибракування, не впливатиме спрямоване світло.

Сеанс AR

Сеанс AR – це компонент Unity, який керує життєвим циклом і конфігурацією досвіду AR. Він виконує такі завдання, як запуск або зупинка сеансу доповненої реальності, керування відстеженням і оновленнями сеансу, а також надання доступу до даних, пов’язаних із доповненою реальністю, як-от канали камери та інформація відстеження. Сеанс AR є фундаментальним компонентом для створення додатків AR в Unity, оскільки він служить мостом між реальним світом і віртуальним вмістом.

Початок сеансу AR

Початок сеансу AR є опорною точкою для всього вмісту, пов’язаного з AR, у програмі доповненої реальності (AR). Він представляє систему координат світу AR і відповідає за відстеження руху пристрою AR і узгодження віртуального вмісту з реальним світом.

Примітка. В останній версії Unity AR Foundation (5.0) джерело сеансу AR застаріло та замінено джерелом XR. Якщо ви використовуєте 5.0 або новішу версію, видаліть джерело сеансу AR і додайте джерело XR. Додати > XR > XR початок.

XR походження

XR origin має подібні компоненти до AR origin. Однак походження XR можна використовувати як для AR, так і для VR.

Основа походження GameObject

Це GameObject, до якого приєднаний компонент XR origin. Він представляє основу початку XR у сцені. Початок XR визначає джерело сеансового простору (0, 0, 0) у сцені XR. Положення та орієнтація цього GameObject визначає точку відліку для всього досвіду XR.

Зміщення підлоги камери

Це змінює положення пристрою XR з камери на простір XR Origin. Це допомагає визначити висоту пристрою XR. Це особливо корисно для режимів відстеження відносно підлоги, коли положення пристрою XR відстежується відносно місця на підлозі.

Камера

Ця властивість представляє камеру, яка використовується для відтворення сцени XR з точки зору пристрою XR. Камера має бути дочірньою частиною GameObject, що містить компонент XR origin. У сценаріях доповненої реальності камера зазвичай використовується для накладання віртуального вмісту на зображення реального світу, зняте камерою пристрою.

Режим відстеження походження

Це визначає, як дані відстеження пристрою XR інтерпретуються відносно джерела XR. Він визначає, де знаходиться позиція (0, 0, 0) у сцені XR у реальному світі. Існує два основних режими:

1. Пристрій: пристрій XR відстежується відносно його початкового положення під час запуску або початку сеансу.
2. Підлога: XR-пристрій відстежується відносно місця на підлозі або поверхні, яке визначається використовуваним XR-пристроєм.

Зсув Y камери

Це висота камери, яка використовується, коли XR Origin перебуває в режимі відстеження джерела «Пристрій». Це зміщення гарантує, що положення камери правильно вирівняно з положенням пристрою XR відносно початку XR.

Разом ці елементи компонента джерела XR відіграють вирішальну роль в управлінні положенням і орієнтацією сцени XR, забезпечуючи належне узгодження між реальним світом і віртуальним вмістом.

Скрипт походження XR

Нижче наведено сценарій джерела XR, написаний мовою C#, який використовується за замовчуванням у джерелі XR.

```
                case TrackingOriginModeFlags.Floor:
                    MoveOffsetHeight(0f);
                    break;
                case TrackingOriginModeFlags.Device:
                    MoveOffsetHeight(m_CameraYOffset);
                    break;
                default:
                    return;
            }
#endif
        }

        /// <summary>
        /// Sets the height of the camera to the given <paramref name="y"/> value by updating the <see cref="CameraFloorOffsetObject"/>.
        /// </summary>
        /// <param name="y">The local y-position to set.</param>
        void MoveOffsetHeight(float y)
        {
            if (m_CameraFloorOffsetObject != null)
            {
                var offsetTransform = m_CameraFloorOffsetObject.transform;
                var desiredPosition = offsetTransform.localPosition;
                desiredPosition.y = y;
                offsetTransform.localPosition = desiredPosition;
            }
        }

        /// <summary>
        /// Repeatedly attempt to initialize the camera.
        /// </summary>
        void TryInitializeCamera()
        {
            if (!Application.isPlaying)
                return;

            m_CameraInitialized = SetupCamera();
            if (!m_CameraInitialized & !m_CameraInitializing)
                StartCoroutine(RepeatInitializeCamera());
        }
```

```
        /// <summary>
        /// Handles re-centering and off-setting the camera in space depending on which
tracking origin mode it is setup in.
        /// </summary>
        bool SetupCamera()
        {
            var initialized = true;

#if XR_MODULE_AVAILABLE
            SubsystemManager.GetInstances(s_InputSubsystems);
            if (s_InputSubsystems.Count > 0)
            {
                foreach (var inputSubsystem in s_InputSubsystems)
                {
                    if (SetupCamera(inputSubsystem))
                    {
                        // It is possible this could happen more than
                        // once so unregister the callback first just in case.
                        inputSubsystem.trackingOriginUpdated -=
OnInputSubsystemTrackingOriginUpdated;
                        inputSubsystem.trackingOriginUpdated +=
OnInputSubsystemTrackingOriginUpdated;
                    }
                    else
                    {
                        initialized = false;
                    }
                }
            }
#endif

            return initialized;
        }

#if XR_MODULE_AVAILABLE
        bool SetupCamera(XRInputSubsystem inputSubsystem)
        {
            if (inputSubsystem == null)
                return false;

            var successful = true;

            switch (m_RequestedTrackingOriginMode)
            {
                case TrackingOriginMode.NotSpecified:
                    CurrentTrackingOriginMode = inputSubsystem.GetTrackingOriginMode();
                    break;
                case TrackingOriginMode.Device:
                case TrackingOriginMode.Floor:
                {
                    var supportedModes = inputSubsystem.GetSupportedTrackingOriginModes();
```

```
                    // We need to check for Unknown because we may not be in a state where we
can read this data yet.
                    if (supportedModes == TrackingOriginModeFlags.Unknown)
                        return false;

                    // Convert from the request enum to the flags enum that is used by the
subsystem
                    var equivalentFlagsMode = m_RequestedTrackingOriginMode ==
TrackingOriginMode.Device
                        ? TrackingOriginModeFlags.Device
                        : TrackingOriginModeFlags.Floor;

                    // ReSharper disable once BitwiseOperatorOnEnumWithoutFlags -- Treated
like Flags enum when querying supported modes
                    if ((supportedModes & equivalentFlagsMode) == 0)
                    {
                        m_RequestedTrackingOriginMode = TrackingOriginMode.NotSpecified;
                        CurrentTrackingOriginMode = inputSubsystem.GetTrackingOriginMode();
                        Debug.LogWarning($"Attempting to set the tracking origin mode to
{equivalentFlagsMode}, but that is not supported by the SDK." +
                            $" Supported types: {supportedModes:F}. Using the current mode of
{CurrentTrackingOriginMode} instead.", this);
                    }
                    else
                    {
                        successful =
inputSubsystem.TrySetTrackingOriginMode(equivalentFlagsMode);
                    }
                }
                break;
                default:
                    Assert.IsTrue(false, $"Unhandled
{nameof(TrackingOriginMode)}={m_RequestedTrackingOriginMode}");
                    return false;
            }

            if (successful)
                MoveOffsetHeight();

            if (CurrentTrackingOriginMode == TrackingOriginModeFlags.Device ||
m_RequestedTrackingOriginMode == TrackingOriginMode.Device)
                successful = inputSubsystem.TryRecenter();

            return successful;
        }

        void OnInputSubsystemTrackingOriginUpdated(XRInputSubsystem inputSubsystem)
        {
            CurrentTrackingOriginMode = inputSubsystem.GetTrackingOriginMode();
            MoveOffsetHeight();
        }
#endif
```

```
        IEnumerator RepeatInitializeCamera()
        {
            m_CameraInitializing = true;
            while (!m_CameraInitialized)
            {
                yield return null;
                if (!m_CameraInitialized)
                    m_CameraInitialized = SetupCamera();
            }
            m_CameraInitializing = false;
        }

        /// <summary>
        /// Rotates the XR origin object around the camera object by the provided <paramref
name="angleDegrees"/>.
        /// This rotation only occurs around the origin's Up vector
        /// </summary>
        /// <param name="angleDegrees">The amount of rotation in degrees.</param>
        /// <returns>Returns <see langword="true"/> if the rotation is performed. Otherwise,
returns <see langword="false"/>.</returns>
        public bool RotateAroundCameraUsingOriginUp(float angleDegrees)
        {
            return RotateAroundCameraPosition(m_OriginBaseGameObject.transform.up,
angleDegrees);
        }

        /// <summary>
        /// Rotates the XR origin object around the camera object's position in world space
using the provided <paramref name="vector"/>
        /// as the rotation axis. The XR Origin object is rotated by the amount of degrees
provided in <paramref name="angleDegrees"/>.
        /// </summary>
        /// <param name="vector">The axis of the rotation.</param>
        /// <param name="angleDegrees">The amount of rotation in degrees.</param>
        /// <returns>Returns <see langword="true"/> if the rotation is performed. Otherwise,
returns <see langword="false"/>.</returns>
        public bool RotateAroundCameraPosition(Vector3 vector, float angleDegrees)
        {
            if (m_Camera == null || m_OriginBaseGameObject == null)
            {
                return false;
            }

            // Rotate around the camera position
            m_OriginBaseGameObject.transform.RotateAround(m_Camera.transform.position,
vector, angleDegrees);

            return true;
        }

        /// <summary>
        /// This function will rotate the XR Origin object such that the XR Origin's up
vector will match the provided vector.
```

```
        /// </summary>
        /// <param name="destinationUp">the vector to which the XR Origin object's up vector
will be matched.</param>
        /// <returns>Returns <see langword="true"/> if the rotation is performed or the
vectors have already been matched.
        /// Otherwise, returns <see langword="false"/>.</returns>
        public bool MatchOriginUp(Vector3 destinationUp)
        {
            if (m_OriginBaseGameObject == null)
            {
                return false;
            }

            if (m_OriginBaseGameObject.transform.up == destinationUp)
                return true;

            var rigUp = Quaternion.FromToRotation(m_OriginBaseGameObject.transform.up,
destinationUp);
            m_OriginBaseGameObject.transform.rotation = rigUp * transform.rotation;

            return true;
        }

        /// <summary>
        /// This function will rotate the XR Origin object around the camera object using the
<paramref name="destinationUp"/> vector such that:
        /// <list type="bullet">
        /// <item>
        /// <description>The camera will look at the area in the direction of the <paramref
name="destinationForward"/></description>
        /// </item>
        /// <item>
        /// <description>The projection of camera's forward vector on the plane with the
normal <paramref name="destinationUp"/> will be in the direction of <paramref
name="destinationForward"/></description>
        /// </item>
        /// <item>
        /// <description>The up vector of the XR Origin object will match the provided
<paramref name="destinationUp"/> vector (note that the camera's Up vector can not be
manipulated)</description>
        /// </item>
        /// </list>
        /// </summary>
        /// <param name="destinationUp">The up vector that the origin's up vector will be
matched to.</param>
        /// <param name="destinationForward">The forward vector that will be matched to the
projection of the camera's forward vector on the plane with the normal <paramref
name="destinationUp"/>.</param>
        /// <returns>Returns <see langword="true"/> if the rotation is performed. Otherwise,
returns <see langword="false"/>.</returns>
        public bool MatchOriginUpCameraForward(Vector3 destinationUp, Vector3
destinationForward)
        {
```

```
            if (m_Camera != null && MatchOriginUp(destinationUp))
            {
                // Project current camera's forward vector on the destination plane, whose
normal vector is destinationUp.
                var projectedCamForward = Vector3.ProjectOnPlane(m_Camera.transform.forward,
destinationUp).normalized;

                // The angle that we want the XROrigin to rotate is the signed angle between
projectedCamForward and destinationForward, after the up vectors are matched.
                var signedAngle = Vector3.SignedAngle(projectedCamForward,
destinationForward, destinationUp);

                RotateAroundCameraPosition(destinationUp, signedAngle);

                return true;
            }

            return false;
        }

        /// <summary>
        /// This function will rotate the XR Origin object around the camera object using the
<paramref name="destinationUp"/> vector such that:
        /// <list type="bullet">
        /// <item>
        /// <description>The forward vector of the XR Origin object, which is the direction
the player moves in Unity when walking forward in the physical world, will match the provided
<paramref name="destinationUp"/> vector</description>
        /// </item>
        /// <item>
        /// <description>The up vector of the XR Origin object will match the provided
<paramref name="destinationUp"/> vector</description>
        /// </item>
        /// </list>
        /// </summary>
        /// <param name="destinationUp">The up vector that the origin's up vector will be
matched to.</param>
        /// <param name="destinationForward">The forward vector that will be matched to the
forward vector of the XR Origin object,
        /// which is the direction the player moves in Unity when walking forward in the
physical world.</param>
        /// <returns>Returns <see langword="true"/> if the rotation is performed. Otherwise,
returns <see langword="false"/>.</returns>
        public bool MatchOriginUpOriginForward(Vector3 destinationUp, Vector3
destinationForward)
        {
            if (m_OriginBaseGameObject != null && MatchOriginUp(destinationUp))
            {
                // The angle that we want the XR Origin to rotate is the signed angle between
the origin's forward and destinationForward, after the up vectors are matched.
                var signedAngle =
Vector3.SignedAngle(m_OriginBaseGameObject.transform.forward, destinationForward,
destinationUp);
```

```
                RotateAroundCameraPosition(destinationUp, signedAngle);

                return true;
            }

            return false;
        }

        /// <summary>
        /// This function moves the camera to the world location provided by <paramref
name="desiredWorldLocation"/>.
        /// It does this by moving the XR Origin object so that the camera's world location
matches the desiredWorldLocation
        /// </summary>
        /// <param name="desiredWorldLocation">the position in world space that the camera
should be moved to</param>
        /// <returns>Returns <see langword="true"/> if the move is performed. Otherwise,
returns <see langword="false"/>.</returns>
        public bool MoveCameraToWorldLocation(Vector3 desiredWorldLocation)
        {
            if (m_Camera == null)
            {
                return false;
            }

            var rot = Matrix4x4.Rotate(m_Camera.transform.rotation);
            var delta = rot.MultiplyPoint3x4(OriginInCameraSpacePos);
            m_OriginBaseGameObject.transform.position = delta + desiredWorldLocation;

            return true;
        }

        /// <summary>
        /// See <see cref="MonoBehaviour"/>.
        /// </summary>
        protected void Awake()
        {
            if (m_CameraFloorOffsetObject == null)
            {
                Debug.LogWarning("No Camera Floor Offset GameObject specified for XR Origin,
using attached GameObject.", this);
                m_CameraFloorOffsetObject = gameObject;
            }

            if (m_Camera == null)
            {
                var mainCamera = Camera.main;
                if (mainCamera != null)
                    m_Camera = mainCamera;
                else
                    Debug.LogWarning("No Main Camera is found for XR Origin, please assign
the Camera field manually.", this);
```

```
            }

            // This will be the parent GameObject for any trackables (such as planes) for
which
            // we want a corresponding GameObject.
            TrackablesParent = (new GameObject("Trackables")).transform;
            TrackablesParent.SetParent(transform, false);
            TrackablesParent.localPosition = Vector3.zero;
            TrackablesParent.localRotation = Quaternion.identity;
            TrackablesParent.localScale = Vector3.one;

            if (m_Camera)
            {
#if INCLUDE_INPUT_SYSTEM && INCLUDE_LEGACY_INPUT_HELPERS
                var trackedPoseDriver =
m_Camera.GetComponent<UnityEngine.InputSystem.XR.TrackedPoseDriver>();
                var trackedPoseDriverOld =
m_Camera.GetComponent<UnityEngine.SpatialTracking.TrackedPoseDriver>();
                if (trackedPoseDriver == null && trackedPoseDriverOld == null)
                {
                    Debug.LogWarning(
                        $"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver
(Input System), " +
                        "so its transform will not be updated by an XR device.  In order for
this to be " +
                        "updated, please add a Tracked Pose Driver (Input System) with
bindings for position and rotation of the center eye.", this);
                }
#elif !INCLUDE_INPUT_SYSTEM && INCLUDE_LEGACY_INPUT_HELPERS
                var trackedPoseDriverOld =
m_Camera.GetComponent<UnityEngine.SpatialTracking.TrackedPoseDriver>();
                if (trackedPoseDriverOld == null)
                {
                    Debug.LogWarning(
                        $"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver, and
com.unity.xr.legacyinputhelpers is installed. " +
                        "Although the Tracked Pose Driver from Legacy Input Helpers can be
used, it is recommended to " +
                        "install com.unity.inputsystem instead and add a Tracked Pose Driver
(Input System) with bindings for position and rotation of the center eye.", this);
                }
#elif INCLUDE_INPUT_SYSTEM && !INCLUDE_LEGACY_INPUT_HELPERS
                var trackedPoseDriver =
m_Camera.GetComponent<UnityEngine.InputSystem.XR.TrackedPoseDriver>();
                if (trackedPoseDriver == null)
                {
                    Debug.LogWarning(
                        $"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver
(Input System), " +
                        "so its transform will not be updated by an XR device.  In order for
this to be " +
                        "updated, please add a Tracked Pose Driver (Input System) with
bindings for position and rotation of the center eye.", this);
```

```
                }
#elif !INCLUDE_INPUT_SYSTEM && !INCLUDE_LEGACY_INPUT_HELPERS
                Debug.LogWarning(
                    $"Camera \"{m_Camera.name}\" does not use a Tracked Pose Driver and
com.unity.inputsystem is not installed, " +
                    "so its transform will not be updated by an XR device.  In order for this
to be " +
                    "updated, please install com.unity.inputsystem and add a Tracked Pose
Driver (Input System) with bindings for position and rotation of the center eye.", this);
#endif
            }
        }

        Pose GetCameraOriginPose()
        {
            var localOriginPose = Pose.identity;
            var parent = m_Camera.transform.parent;

            return parent
                ? parent.TransformPose(localOriginPose)
                : localOriginPose;
        }

        /// <summary>
        /// See <see cref="MonoBehaviour"/>.
        /// </summary>
        protected void OnEnable() => Application.onBeforeRender += OnBeforeRender;

        /// <summary>
        /// See <see cref="MonoBehaviour"/>.
        /// </summary>
        protected void OnDisable() => Application.onBeforeRender -= OnBeforeRender;

        void OnBeforeRender()
        {
            if (m_Camera)
            {
                var pose = GetCameraOriginPose();
                TrackablesParent.position = pose.position;
                TrackablesParent.rotation = pose.rotation;
            }

            if (TrackablesParent.hasChanged)
            {
                TrackablesParentTransformChanged?.Invoke(
                    new ARTrackablesParentTransformChangedEventArgs(this, TrackablesParent));
                TrackablesParent.hasChanged = false;
            }
        }

        /// <summary>
        /// See <see cref="MonoBehaviour"/>.
        /// </summary>
```

```
        protected void OnValidate()
        {
            if (m_OriginBaseGameObject == null)
                m_OriginBaseGameObject = gameObject;

            if (Application.isPlaying && isActiveAndEnabled)
            {
                // Respond to the mode changing by re-initializing the camera,
                // or just update the offset height in order to avoid recentering.
                if (IsModeStale())
                    TryInitializeCamera();
                else
                    MoveOffsetHeight();
            }

            bool IsModeStale()
            {
#if XR_MODULE_AVAILABLE
                if (s_InputSubsystems.Count > 0)
                {
                    foreach (var inputSubsystem in s_InputSubsystems)
                    {
                        // Convert from the request enum to the flags enum that is used by
the subsystem
                        TrackingOriginModeFlags equivalentFlagsMode;
                        switch (m_RequestedTrackingOriginMode)
                        {
                            case TrackingOriginMode.NotSpecified:
                                // Don't need to initialize the camera since we don't set the
mode when NotSpecified (we just keep the current value)
                                return false;
                            case TrackingOriginMode.Device:
                                equivalentFlagsMode = TrackingOriginModeFlags.Device;
                                break;
                            case TrackingOriginMode.Floor:
                                equivalentFlagsMode = TrackingOriginModeFlags.Floor;
                                break;
                            default:
                                Assert.IsTrue(false, $"Unhandled
{nameof(TrackingOriginMode)}={m_RequestedTrackingOriginMode}");
                                return false;
                        }

                        if (inputSubsystem != null && inputSubsystem.GetTrackingOriginMode()
!= equivalentFlagsMode)
                        {
                            return true;
                        }
                    }
                }
#endif
                return false;
            }
```

```
        }

        /// <summary>
        /// See <see cref="MonoBehaviour"/>.
        /// </summary>
        protected void Start()
        {
            TryInitializeCamera();
        }

        /// <summary>
        /// See <see cref="MonoBehaviour"/>.
        /// </summary>
        protected void OnDestroy()
        {
#if XR_MODULE_AVAILABLE
            foreach (var inputSubsystem in s_InputSubsystems)
            {
                if (inputSubsystem != null)
                    inputSubsystem.trackingOriginUpdated -=
OnInputSubsystemTrackingOriginUpdated;
            }
#endif
        }
    }
}
```

Початок XR представляє джерело сеансового простору (0, 0, 0) у сцені XR (розширеної реальності), такій як AR або VR. Це дозволяє маніпулювати положенням і орієнтацією сцени XR, зокрема для переміщення XR і зсуву камери.

Сценарій визначає кілька властивостей і методів для обробки функцій, пов'язаних із XR, наприклад налаштування камери, яка використовується для відтворення сцени XR, визначення батьківського перетворення для відстежуваних об'єктів, керування режимом джерела відстеження (відносно пристрою чи відносно підлоги) та управління позиціонуванням і обертанням камери.

Елементи коду

Ось деякі важливі елементи коду:

Властивості

Сценарій містить різноманітні властивості, такі як *Camera*, *TrackablesParent*, *CameraFloorOffsetObject*, *RequestedTrackingOriginMode*тощ о, до яких можна отримати доступ і змінити їх в Unity Inspector.

методи

Сценарій містить такі функції, як , *RotateAroundCameraPosition*, тощо, які використовуються для керування положенням і орієнтацією сцени XR на основі різних сценаріїв.*MatchOriginUpMoveCameraToWorldLocation*

Подія

Сценарій визначає подію *TrackablesParentTransformChanged*, яка викликається кожного разу, коли *TrackablesParent*змінюється перетворення. Це дозволяє зовнішнім компонентам підписуватися на цю подію та відповідним чином реагувати.

Моноповедінка

Сценарій успадковує MonoBehaviour, що робить його компонентом Unity, який можна приєднати до GameObjects у редакторі Unity.

Загалом, цей сценарій забезпечує базову структуру для керування XR Origin і його зв'язком із камерою та іншими компонентами, пов'язаними з XR, що полегшує створення додатків XR в Unity.

Висновок

Завдяки такому налаштуванню ви тепер можете почати створювати власні інтерактивні та захоплюючі додатки AR. Після їх створення ви можете легко протестувати їх на своїх пристроях Android та iOS.

Відстеження обличчя в Unity (Face tracking in Unity)

Відстеження обличчя означає здатність системи AR виявляти та відстежувати обличчя користувача в режимі реального часу за допомогою камери пристрою. Це дозволяє створювати інтерактивні враження від доповненої реальності, які спеціально прив'язані до міміки та рухів.

Додати відстеження обличчя

Технологія відстеження обличчя може розпізнавати риси та вирази обличчя, наприклад положення та орієнтацію очей, носа, рота та голови. Ось як це можна реалізувати. Створіть новий проект AR в Unity.

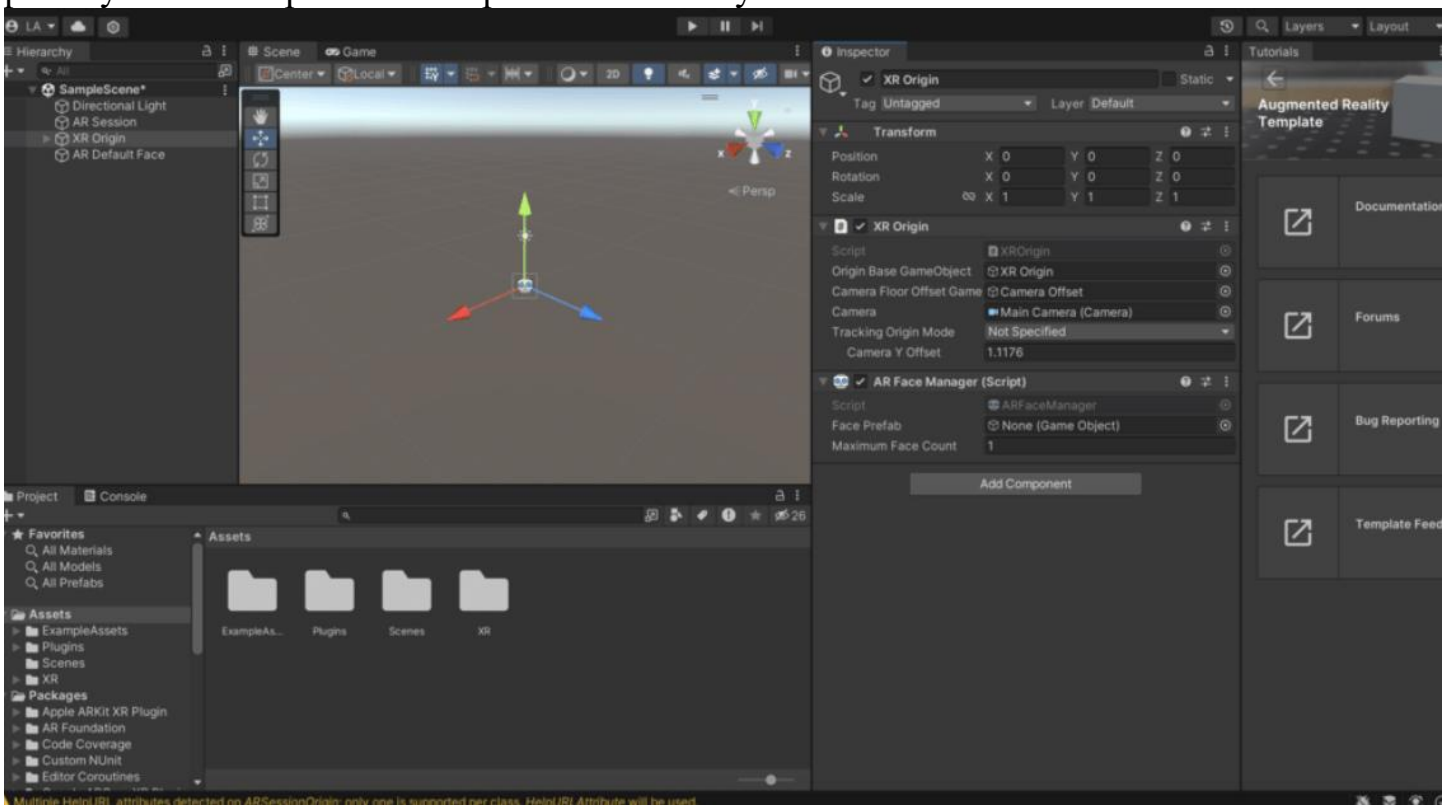

Рис.10.

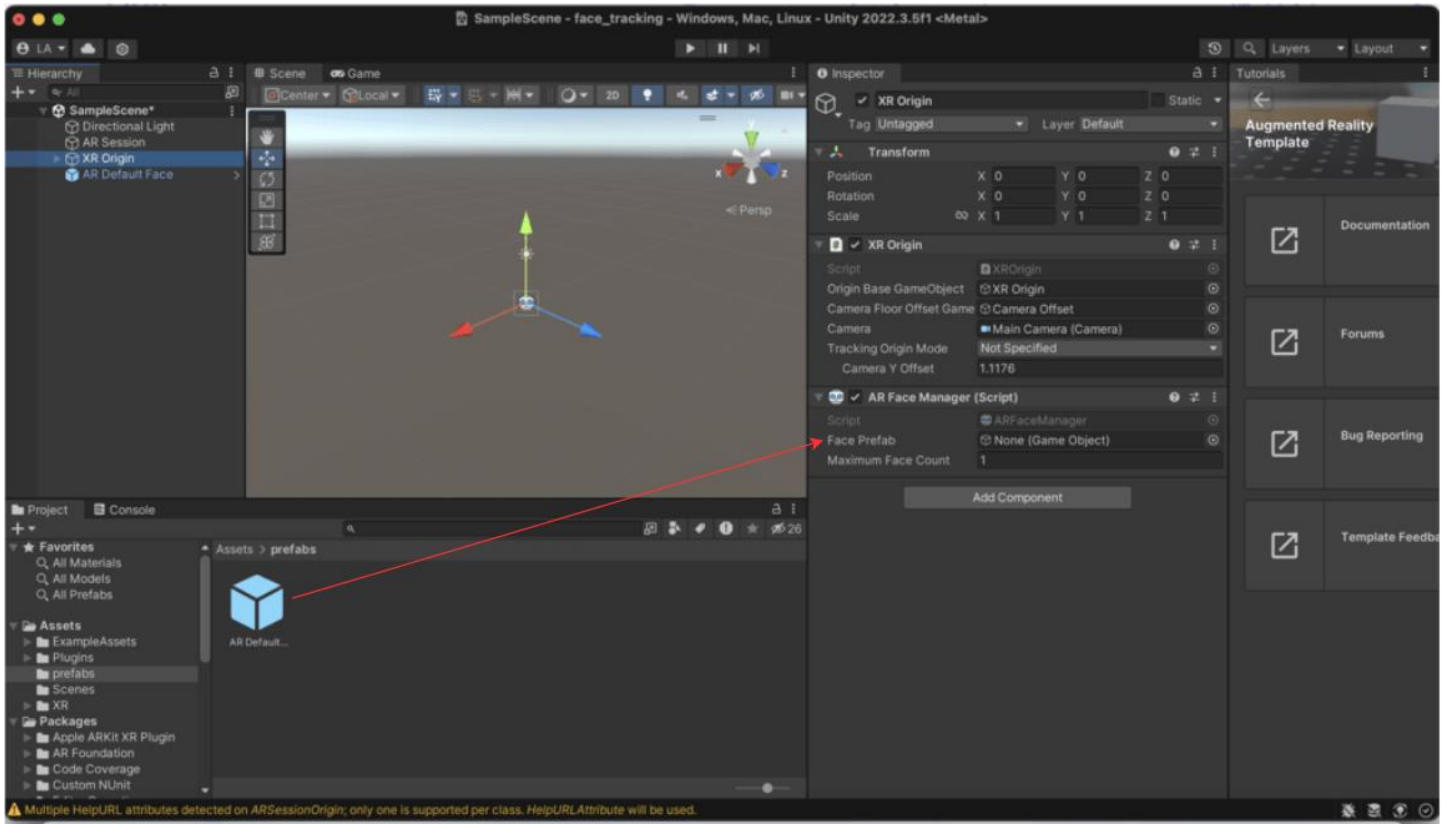

Рис.11.

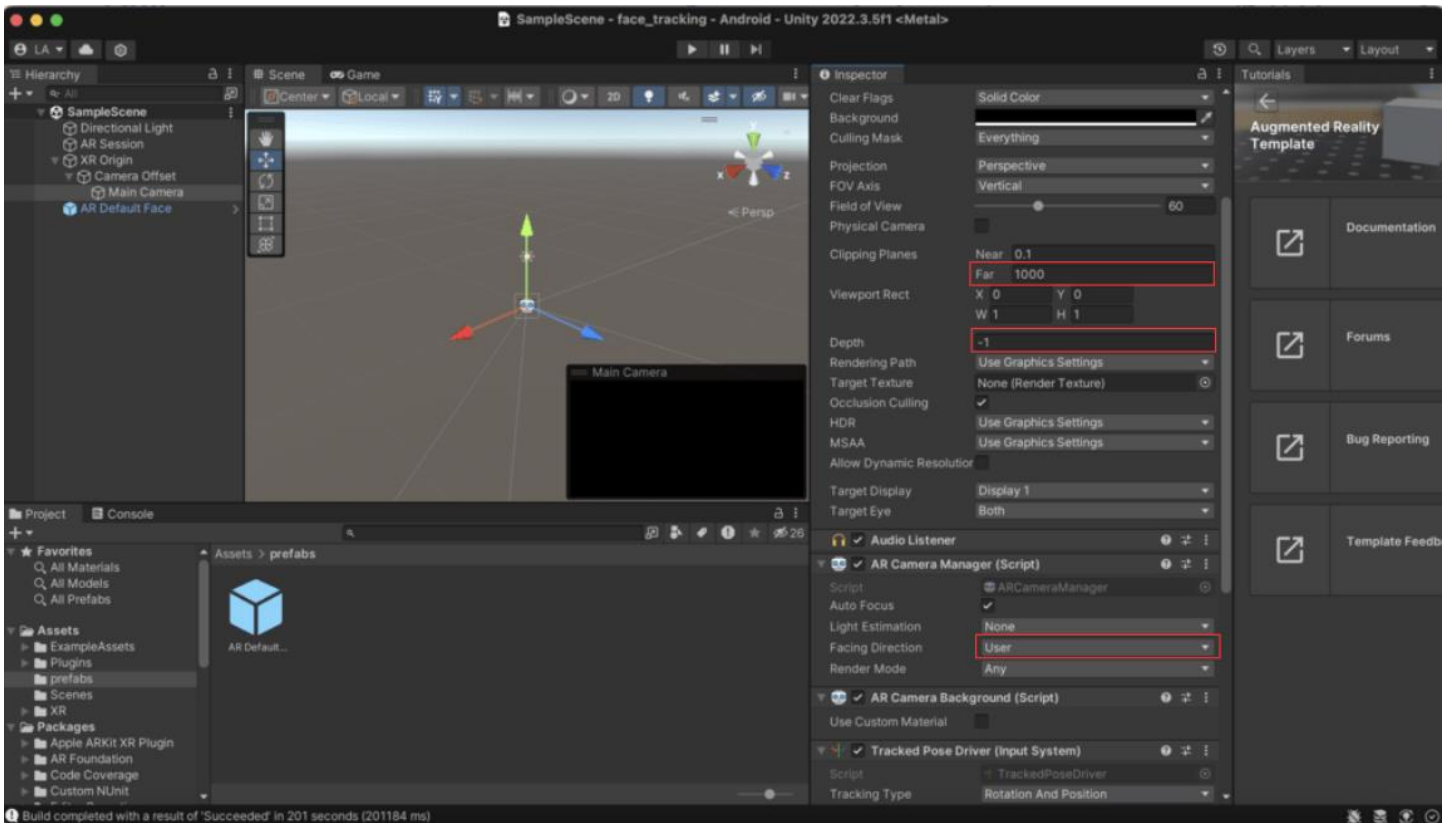

Рис.12.

Компонент менеджера обличчя AR

Менеджер **обличчя AR** – це компонент, наданий базовою структурою AR, який полегшує функцію відстеження обличчя в програмах доповненої реальності. Це дозволяє виявляти та відстежувати обличчя користувача в режимі реального часу за допомогою передньої камери пристрою. У XR origin додайте компонент керування обличчям AR. Ви

можете змінити максимальну кількість облич на кількість облич, які потрібно відстежувати.

Стандартне обличчя AR

Обличчя **AR за замовчуванням** — це стандартний GameObject, попередньо створений Unity. Ми будемо використовувати як префаб для менеджера обличчя AR. На вкладці ієрархії створіть > XR > обличчя за замовчуванням AR . Перетягніть його в папку prefabs. Звідти перетягніть обличчя AR за замовчуванням із папки prefabs до префабу обличчя в джерелі XR.

Основна камера

У XR origin двічі клацніть головну камеру та внесіть такі зміни:

Відсічні площини

Властивість **площини відсікання** основної камери визначає діапазон відстаней, у межах якого відтворюватимуться об'єкти. Він складається з двох значень: значення «Близько» та значення «Далеко». Значення «Near» визначає найближчу від камери відстань, на якій об'єкти почнуть відтворюватися, а значення «Far» визначає максимальну відстань, на якій об'єкти будуть видимі. У цьому випадку змініть значення «Far» на 1000 одиниць. Об'єкти за межами цієї відстані не відтворюватимуться, і вони виглядатимуть вирізаними зі сцени.

Глибина

Властивість глибини визначає пріоритет візуалізації камери порівняно з іншими камерами в сцені **.** Змініть значення глибини на -1, яке вказує на те, що ця камера буде відображена першою в сцені, забезпечуючи належне поєднання вмісту AR із потоком реальної камери та правильним накладанням.

Лицьовий напрямок

Менеджер **камери AR** – це компонент базової структури AR Unity, який керує функціями та налаштуваннями камери AR. Він надає інтерфейс до камери пристрою та обробляє конфігурації сеансу AR. У менеджері камер AR ви можете знайти основну камеру, яка автоматично встановлюється основною камерою для візуалізації AR. Менеджер камери AR спрощує налаштування та конфігурацію камери AR, спрощуючи створення додатків AR без роботи з низькорівневими налаштуваннями камери. Змініть напрямок на «користувач». Якщо встановити напрямок обличчя користувача, відкриється передня камера вашого пристрою.

Завдання на лабораторну роботу

1. Завантажте 3D моделі з анімацією.

Примітка: можна використовувати ресурси тут

2. Налаштуйте Vuforia - Virtual Button для завантаження анімації.
3. Проведіть налаштування анімації 3D моделі.
4. Експортуйте та проведіть тестування AR застосунку на мобільному пристрої.
5. Підготуйте та налаштуйте музичну композицію, що буде програвати разом з анімацією 3D моделі.
6. Проведіть тестування на мобільному пристрої.
7. Створіть проект 5_2.

8. Додайте AR-маску для обличчя з кількома параметрами.
9. Імпортуйте текстуру для використання AR Faces користувача. Поекспериментуйте з різними текстурами.
10. Створіть додаткові маски (разом має бути не менше 4 різних масок), передбачте можливість вибору однієї з масок.
11. Використовуючи генератор випадкових зображень проведіть кодування генератора зображень.

Примітка: ассети тут AR Face Assets

12. Скомпілювати два файли: під ОС Android 7.0 або вище та під ОС iOS 6.0 або вище.
13. Закомітити обидва файли та проєскт на гіт. Надати доступ своєму викладачу.