Лабораторная работа № 3. Проект – разработка простого AR-Приложения для Android-устройства (смартфон, планшет и пр.). Создание в графическом редакторе Unity 3D сцены дополненной реальности: визуализация управляемой анимации 3D-Модели с использованием виртуальной кнопки. Часть 2. Создание виртуальной кнопки кнопки для организации управляемой анимации 3D-Модели, подготовленной в ЛР № 3, Часть 1.

Работа по созданию простейшей анимации простейшего виртуального объекта **3D Модели была выполнена в первой части ЛР №3**.

Предметом исследования в первой части ЛР №3 стали элементы меню окна **Animation**, которые **позволяют вам создавать трансформации любой сложности на любом временном интервале.**

На выходе **ЛР №3, Часть 1** вами были разработаны 3 Приложения ДР для:

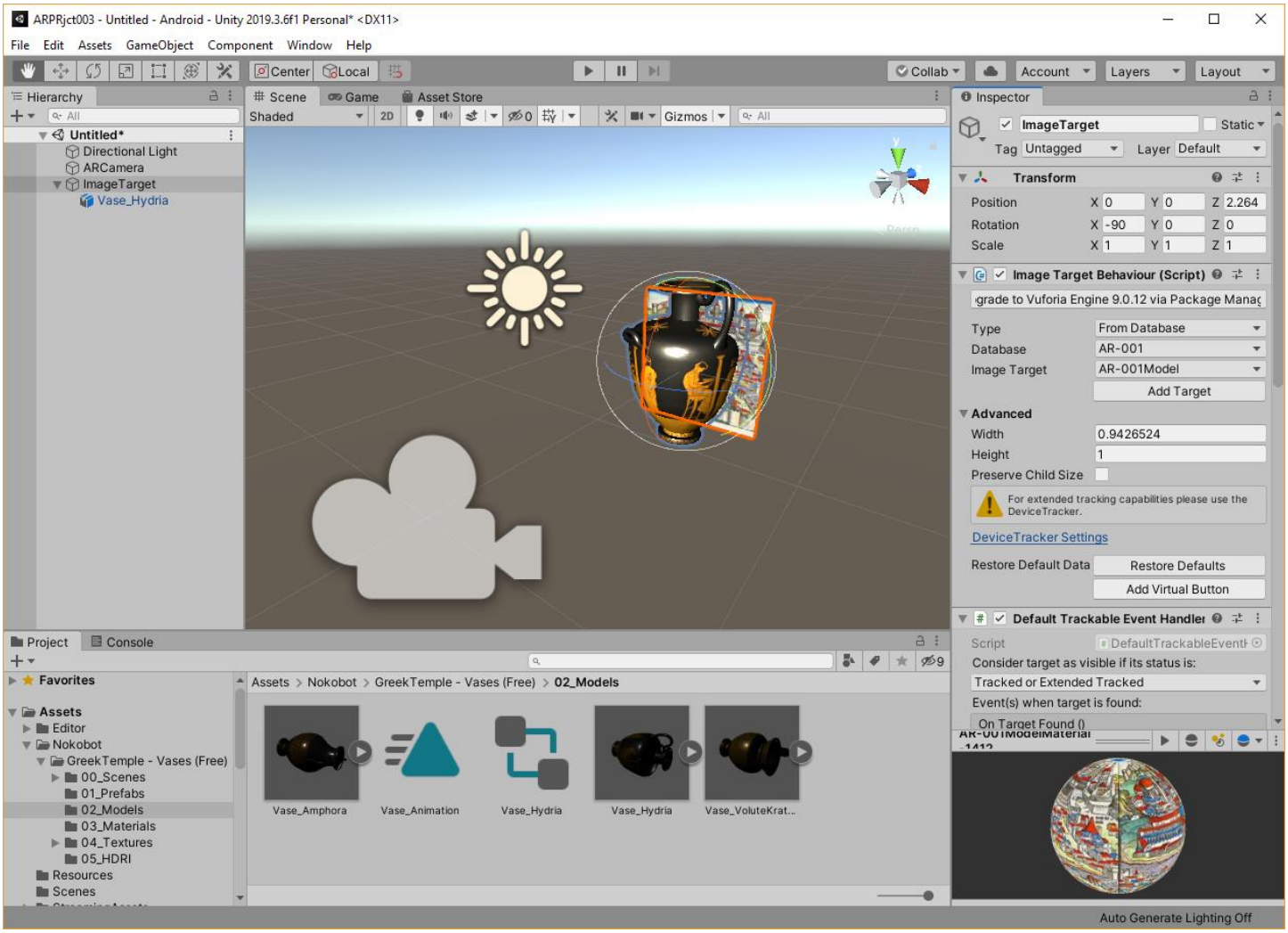
- Вращения выбранной **3D-модели** вокруг выбранной оси (в зависимости от модели);
- Перемещения выбранной **3D-модели** по диагонали, например – из нижнего левого угла в верхний правый;
- **«Наезда»/«Удаления»** вращающейся модели на зрителя.

Одну (а может быть и все три, попытки такой реализации приветствуются!) из разработанных в рамках этого задания анимаций в ЛР №3, Часть 2 можно использовать для связывания с виртуальной кнопкой для создания с ее помощью управляемой анимации.

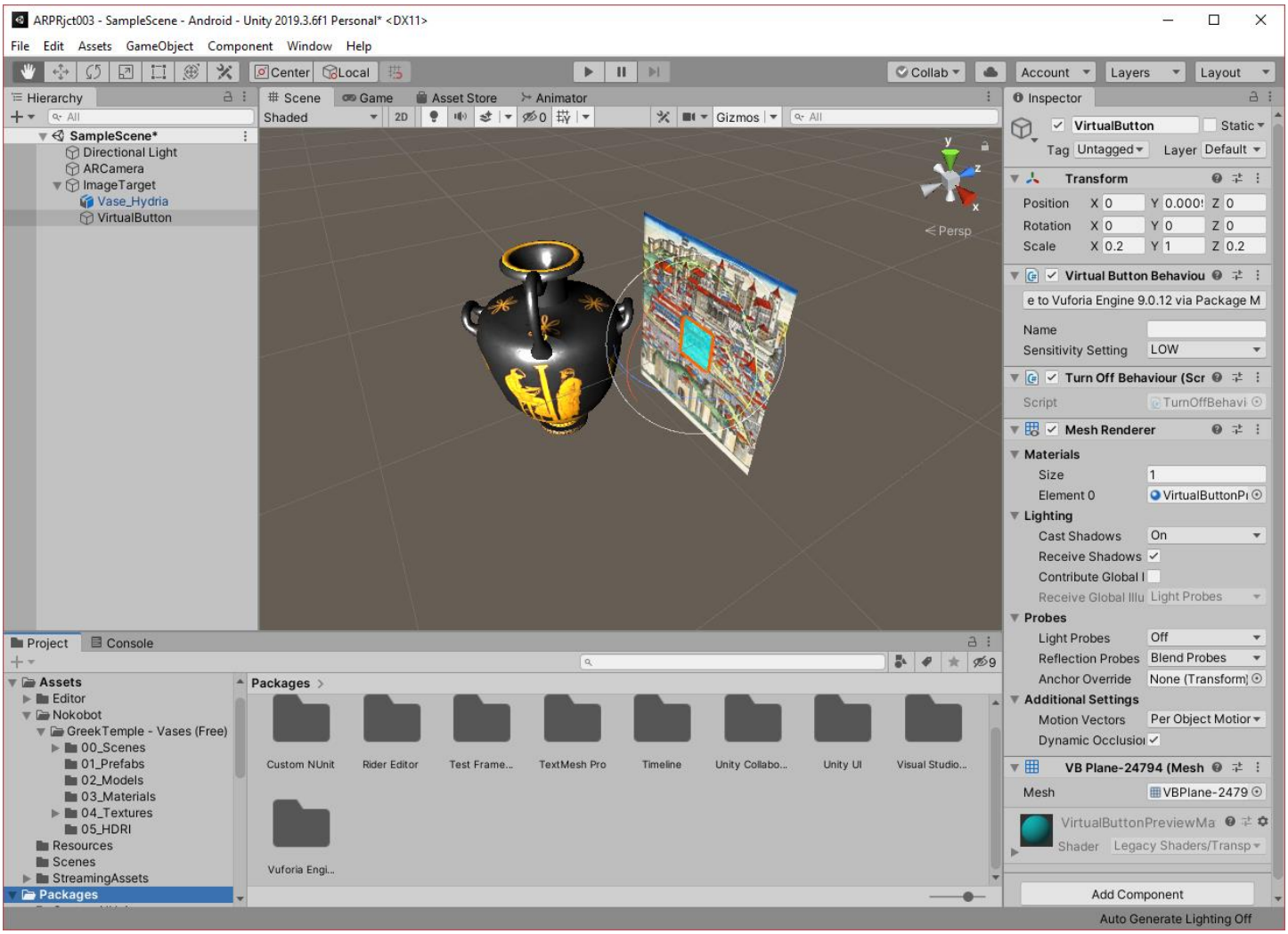
В данном описании вашему вниманию предлагается подробное описание шагов, которые необходимо выполнить для добавления в **Проект виртуальной кнопки**, с помощью которой может быть осуществлено управление простейшей анимацией (вращение вокруг вертикальной оси) 3D-модели греческой амфоры.

Итак, по завершении создания простейшей анимации в **ЛР №3, Часть 1**, вам предлагалось закрыть окно **Animation** и убедиться в том, что в **Assets окна Project** появился новый **Asset** – **Vase_Animation**. Это имя будет использоваться в дальнейшем при разработке виртуальной кнопки и связывании ее посредством скриптинга с анимацией.

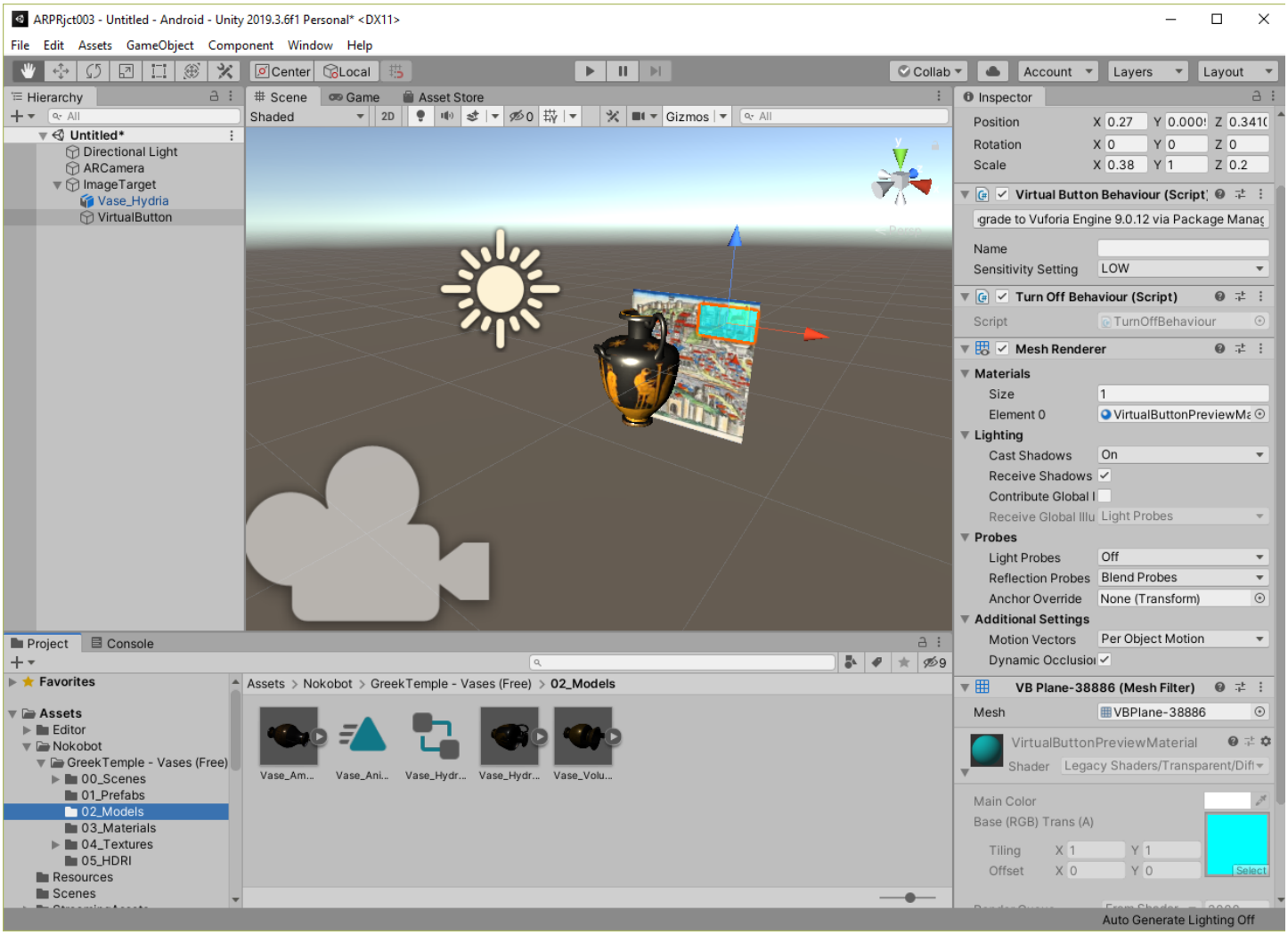
1. **Добавим виртуальную кнопку**. Как вы помните, этот объект мы располагаем тоже относительно таргета. Выбираем позицию **Image Target в Hierarchy** окна редактора **Unity 3D**. В окне **Inspector** в разделе **Image Target Behaviour (Script)→меню Advanced.** Убедитесь, что в установленной вами версии **Unity** в этом меню появилась кнопка **Add Virtual Button.**



Добавляем виртуальную кнопку, убеждаемся, что она появилась в окне **Hierarchy**, как потомок **Image Target**:

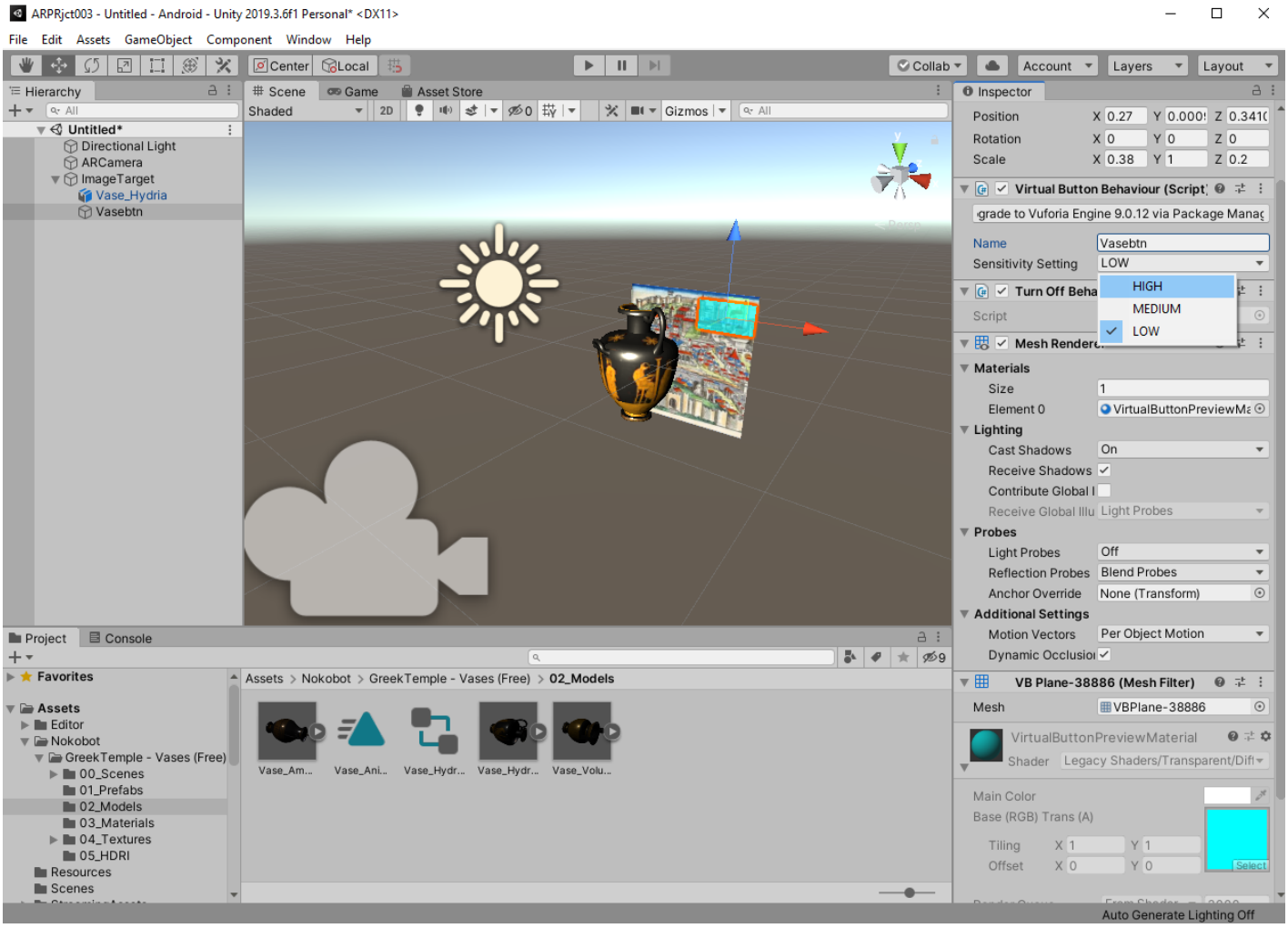


Вспомним, что в нашем таргете есть места наибольшего скопления распознаваемых точек – **зоны чувствительности**. Именно они являются наиболее предпочтительными для размещения виртуальной кнопки. Перемещаем кнопку в нужное место (в правый верхний угол таргета) и масштабируем ее:

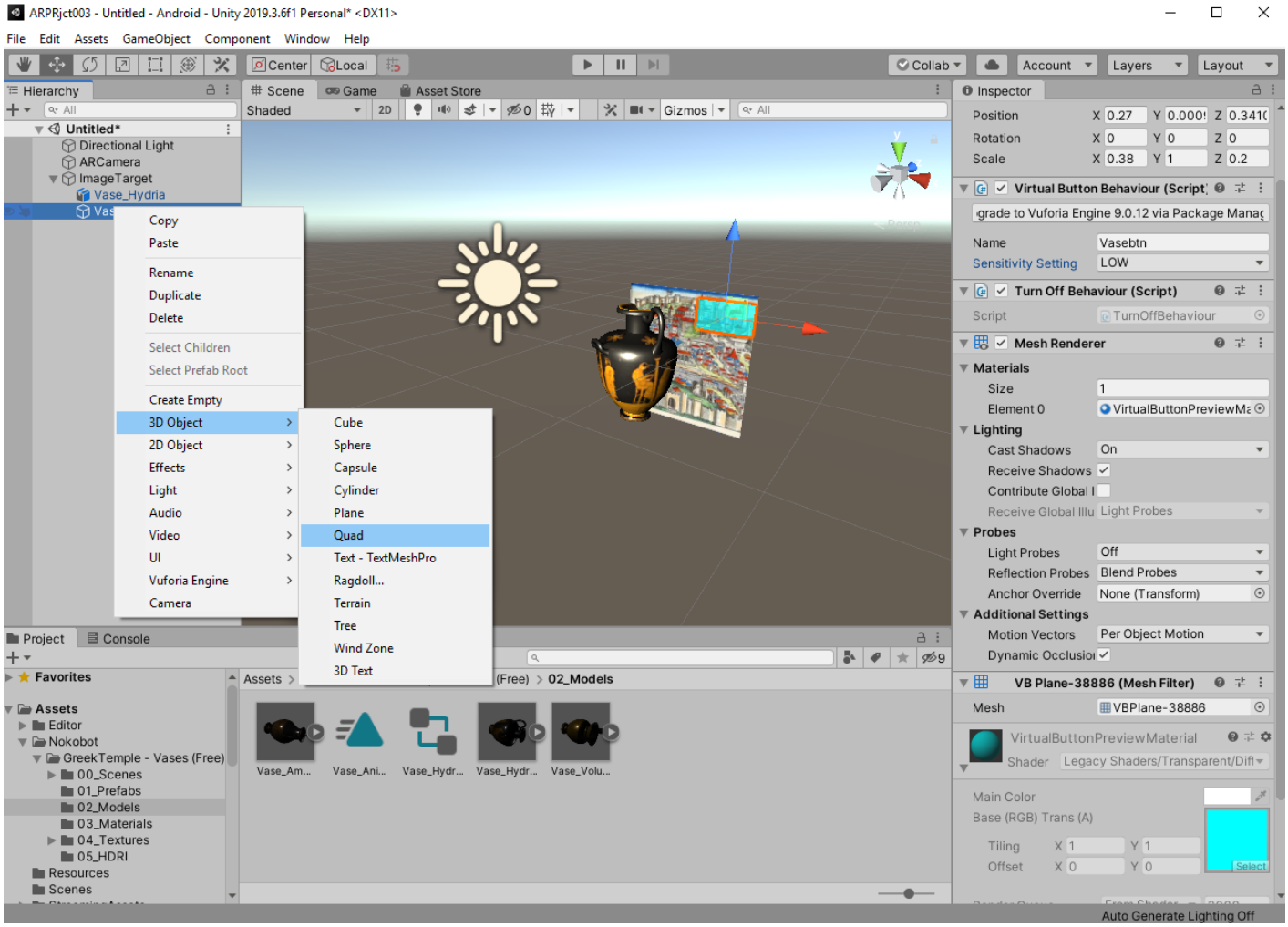


Для дальнейшей работы присвоим виртуальной кнопке имя. Переименуем **Virtual Button**, в **Vasebtn** (по правой кнопке в области **Hierarchy→Rename**), и запомним это новое имя.

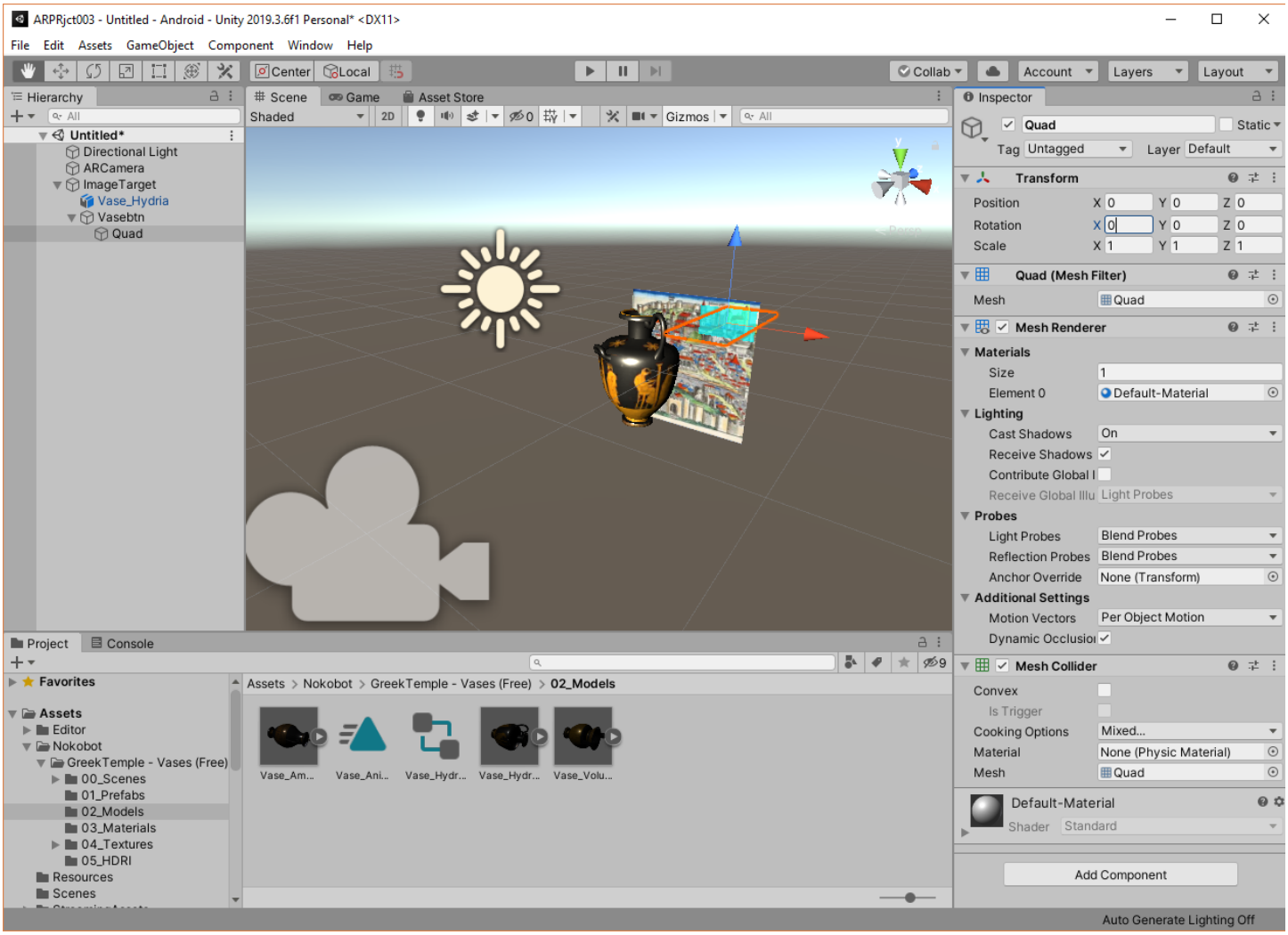
Это имя вручную надо внести для виртуальной кнопки в области **Inspector** в области **Virtual Button Behaviour (Script)**, и в этой же области повысим уровень чувствительности **Sensitivity Setting→HIGH:**

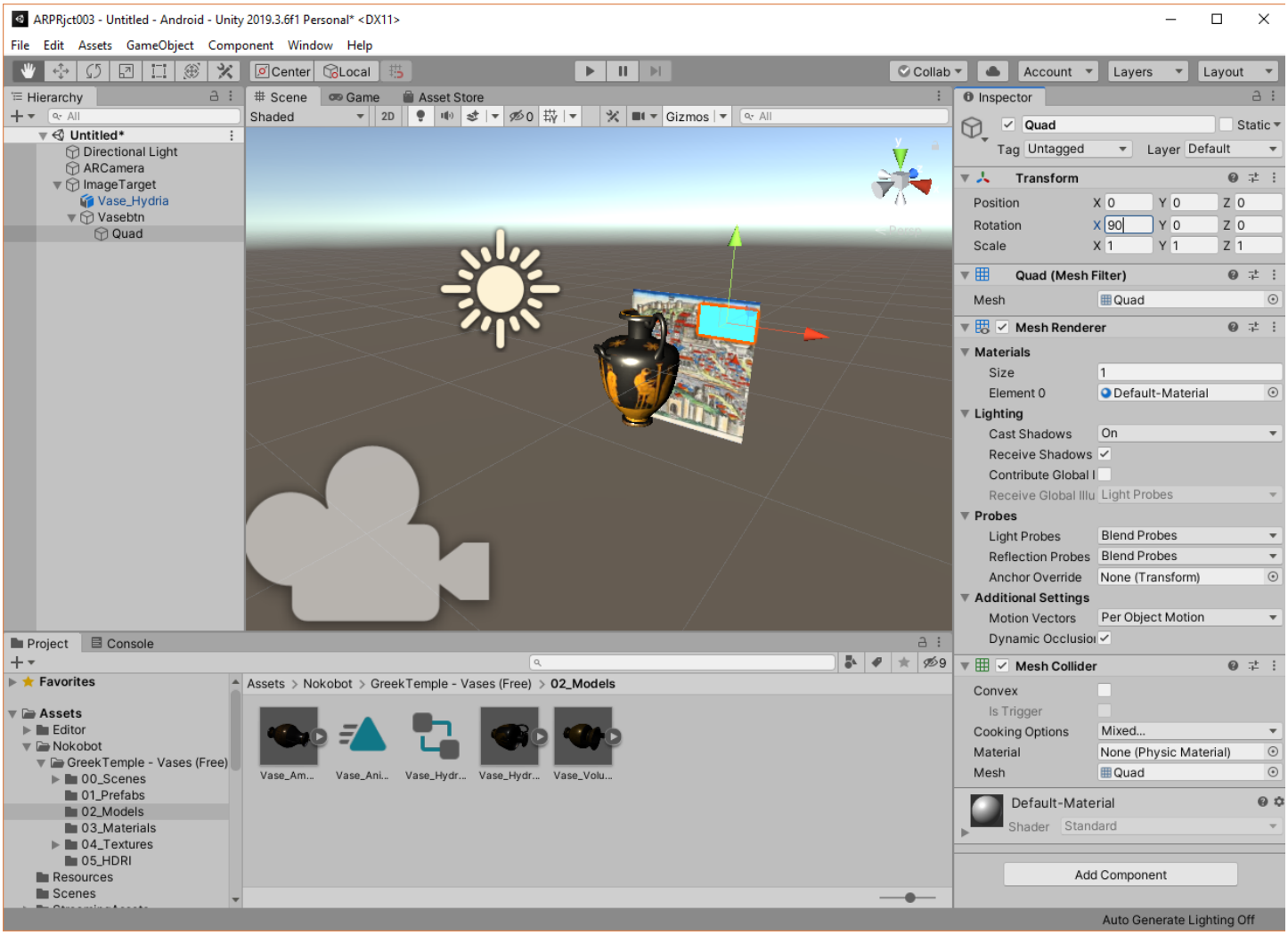

ARPRjct003 - Untitled - Android - Unity 2019.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Hierarchy
Untitled*
Directional Light
ARCamera
ImageTarget
Vase_Hydria
Vasebtn
Scene Game Asset Store
Shaded 2D Gizmos
Inspector
Position X 0.27 Y 0.000 Z 0.341
Rotation X 0 Y 0 Z 0
Scale X 0.38 Y 1 Z 0.2
Virtual Button Behaviour (Script)
grade to Vuforia Engine 9.0.12 via Package Manag
Name Vasebtn
Sensitivity Setting LOW
HIGH
MEDIUM
LOW
Turn Off Beha
Script
Mesh Render
Materials
Size 1
Element 0 VirtualButtonPreviewMa
Lighting
Cast Shadows On
Receive Shadows
Contribute Global I
Receive Global Illu Light Probes
Probes
Light Probes Off
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusio
VB Plane-38886 (Mesh Filter)
Mesh VBPlane-38886
VirtualButtonPreviewMaterial
Shader Legacy Shaders/Transparent/Diffu
Main Color
Base (RGB) Trans (A)
Tiling X 1 Y 1
Offset X 0 Y 0
Select
Project Console
Favorites
Assets
Editor
Nokobot
GreekTemple - Vases (Free)
00_Scenes
01_Prefabs
02_Models
03_Materials
04_Textures
05_HDRI
Resources
Scenes
Assets > Nokobot > GreekTemple - Vases (Free) > 02_Models
Vase_Am...
Vase_Ani...
Vase_Hydr...
Vase_Hydr...
Vase_Volu...
Auto Generate Lighting Off

Для того, чтобы кнопка была плоским объектом, свяжем ее с объектом **Quad** (как вы это неоднократно делали в предыдущих ЛР):



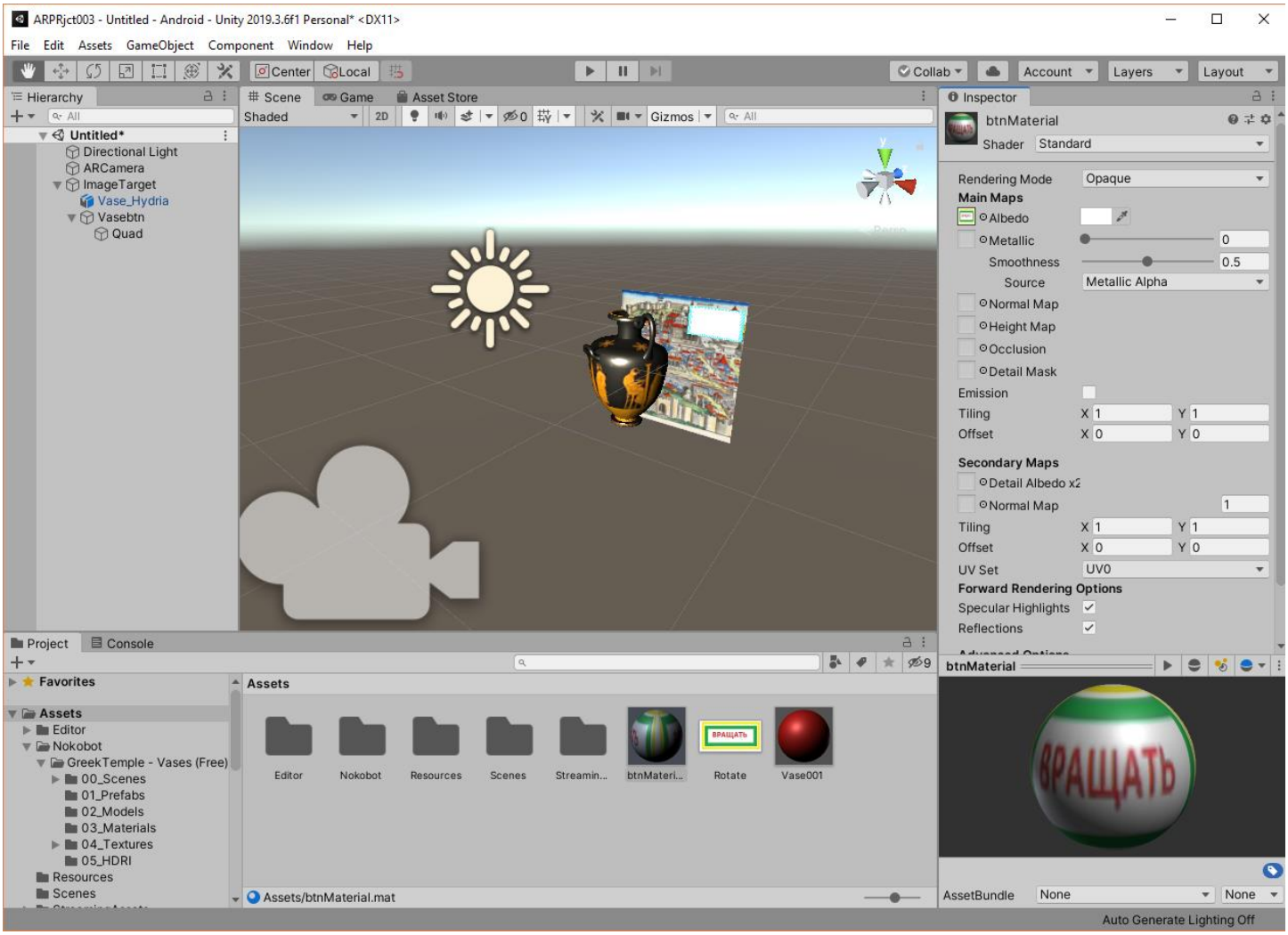
Разместим этот **Quad** в размерах и пропорциях, соответствующих нашей виртуальной кнопке→ развернув его на **90 градусов по X**:


ARPRjct003 - Untitled - Android - Unity 2019.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Hierarchy
Untitled*
Directional Light
ARCamera
ImageTarget
Vase_Hydria
Vasebtn
Quad
Scene Game Asset Store
Shaded 2D Gizmos
Inspector
Quad
Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Quad (Mesh Filter)
Mesh Quad
Mesh Renderer
Materials
Size 1
Element 0 Default-Material
Lighting
Cast Shadows On
Receive Shadows
Contribute Global I
Receive Global Illu Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusio
Mesh Collider
Convex
Is Trigger
Cooking Options Mixed...
Material None (Physic Material)
Mesh Quad
Default-Material
Shader Standard
Add Component
Project Console
Favorites
Assets
Editor
Nokobot
GreekTemple - Vases (Free)
00_Scenes
01_Prefabs
02_Models
03_Materials
04_Textures
05_HDRI
Resources
Scenes
Assets > Nokobot > GreekTemple - Vases (Free) > 02_Models
Vase_Am...
Vase_Ani...
Vase_Hydr...
Vase_Hydr...
Vase_Volu...
Auto Generate Lighting Off

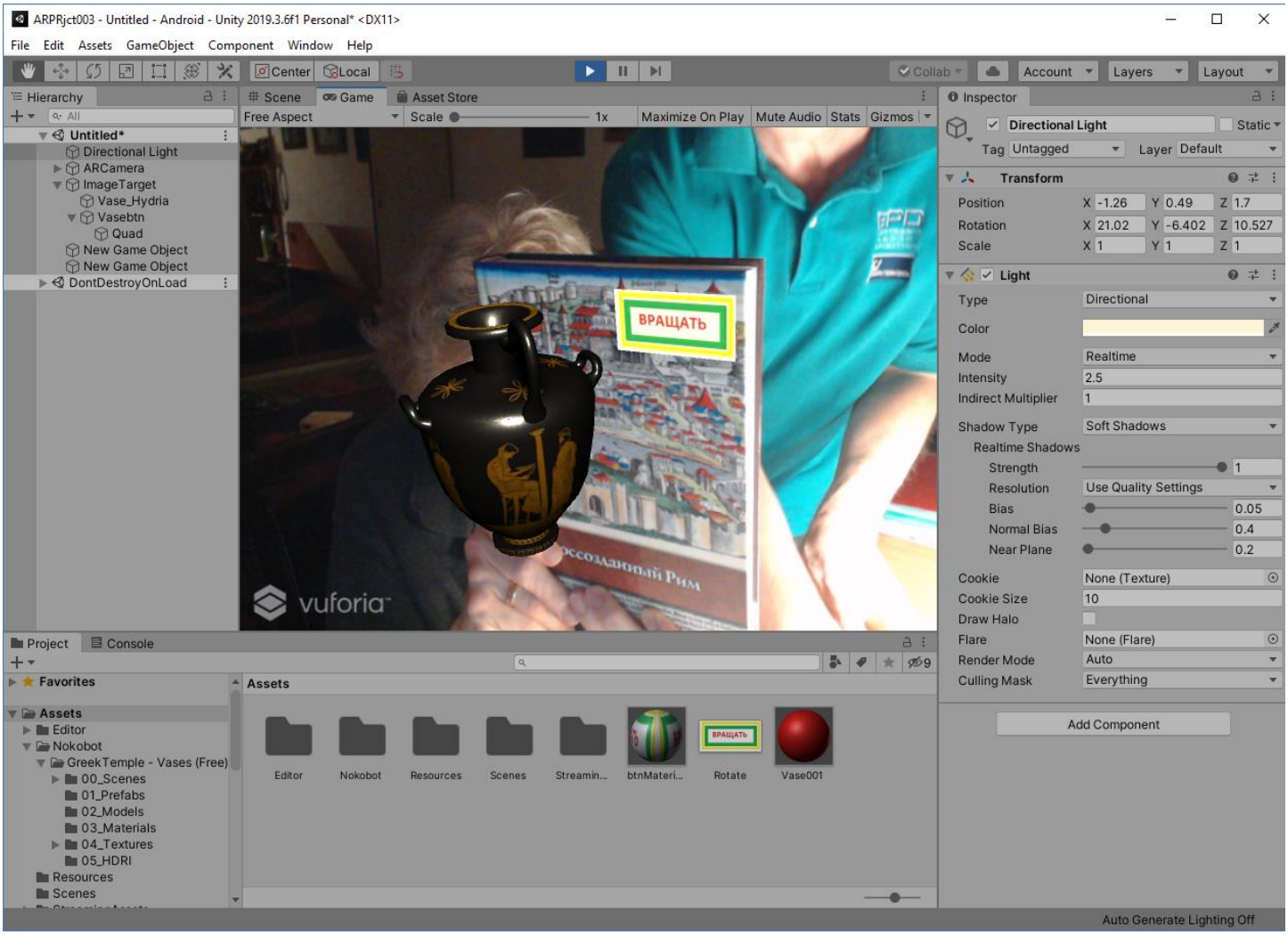


Кнопка должна будет содержать картинку/слова/пиктограмму и пр. Подготовим наполнение кнопки заранее в виде изображения, которое затем будем связано с **Quad** через материал → «Обернем» кнопку материалом.

Подобные действия были произведены вами в **ЛР№2, Часть 2.**



Результат – вращение модели и видимость кнопки с выбранным наполнением можно проанализировать в режиме **Play** в редакторе **Unity**. Добиться нужного качества визуализации, например, вы можете с помощью изменения интенсивности освещенности (в **Inspector'e** объекта **Directional Light→Light→Intensity**), взаимного расположения **Quad** и **Кнопки** и т.д.



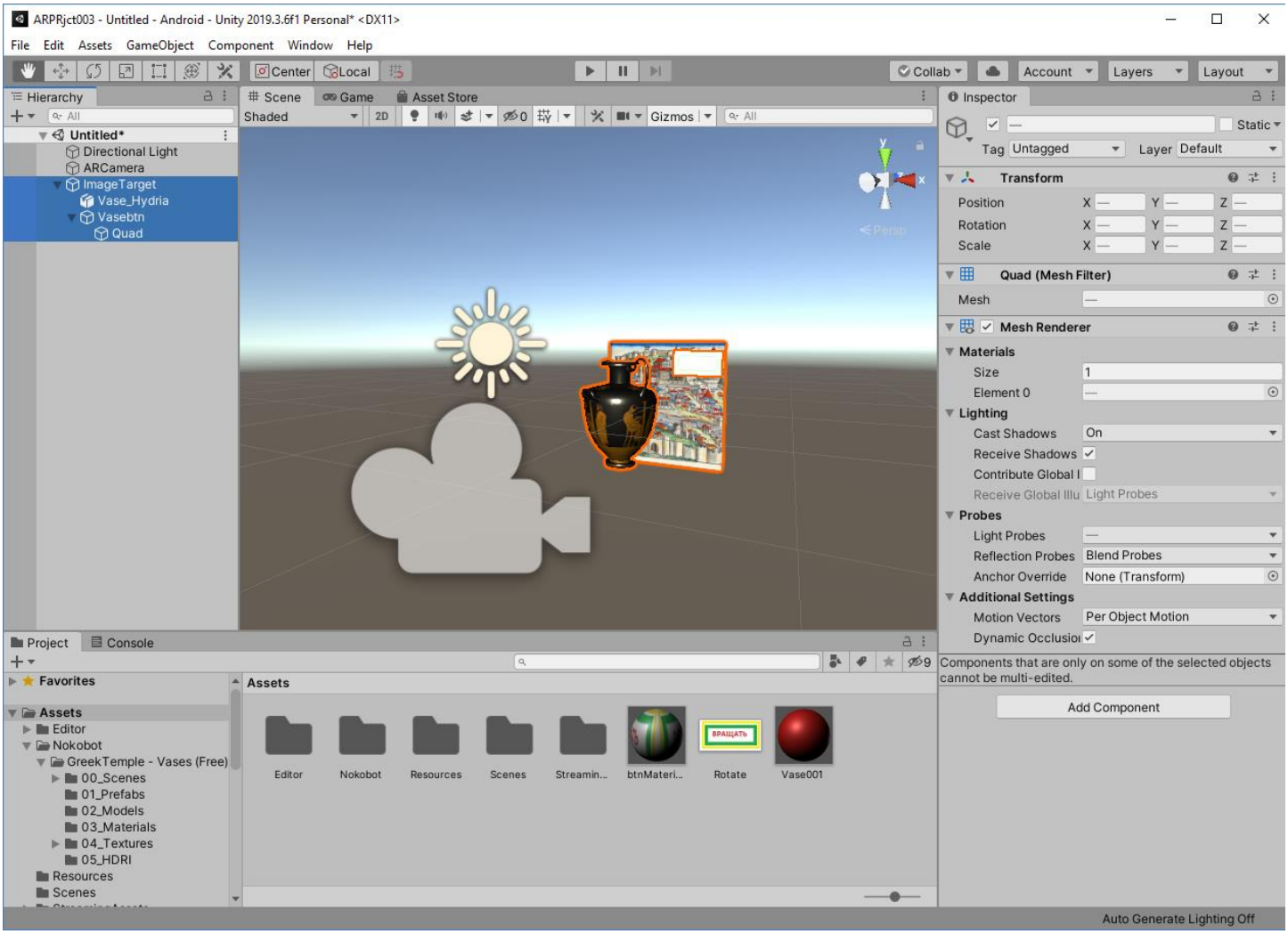
2. **Таким образом, на данном шаге мы имеем следующие объекты в Проекте:**

- **AR Camera;**
- **Таргет;**
- **3D Модель с анимацией;**
- **Составной объект, реализующий виртуальную кнопку.**

Вспомним: **Задача ЛР №3** – разработка элементов виртуального интерфейса для работы в ДР, т.е. установление связей между поведением (**behaviour**) виртуальной **3D-модели** (у нас - **«анимация»**) и состоянием виртуального элемента управления (**«виртуальная кнопка» → отжата/нажата**).

Для установления указанных связей нам потребуется программирование. По умолчанию в **Unity 3D** предлагается писать программы (т.н. скрипты) для установления связей между объектами **Unity 3D** на языке **C#**.

ВАЖНО!! **Центральным объектом** в этой структуре является таргет. Как видно в **Hierarchy**, все интересующие нас объекты интерфейса являются потомками **Image target:**

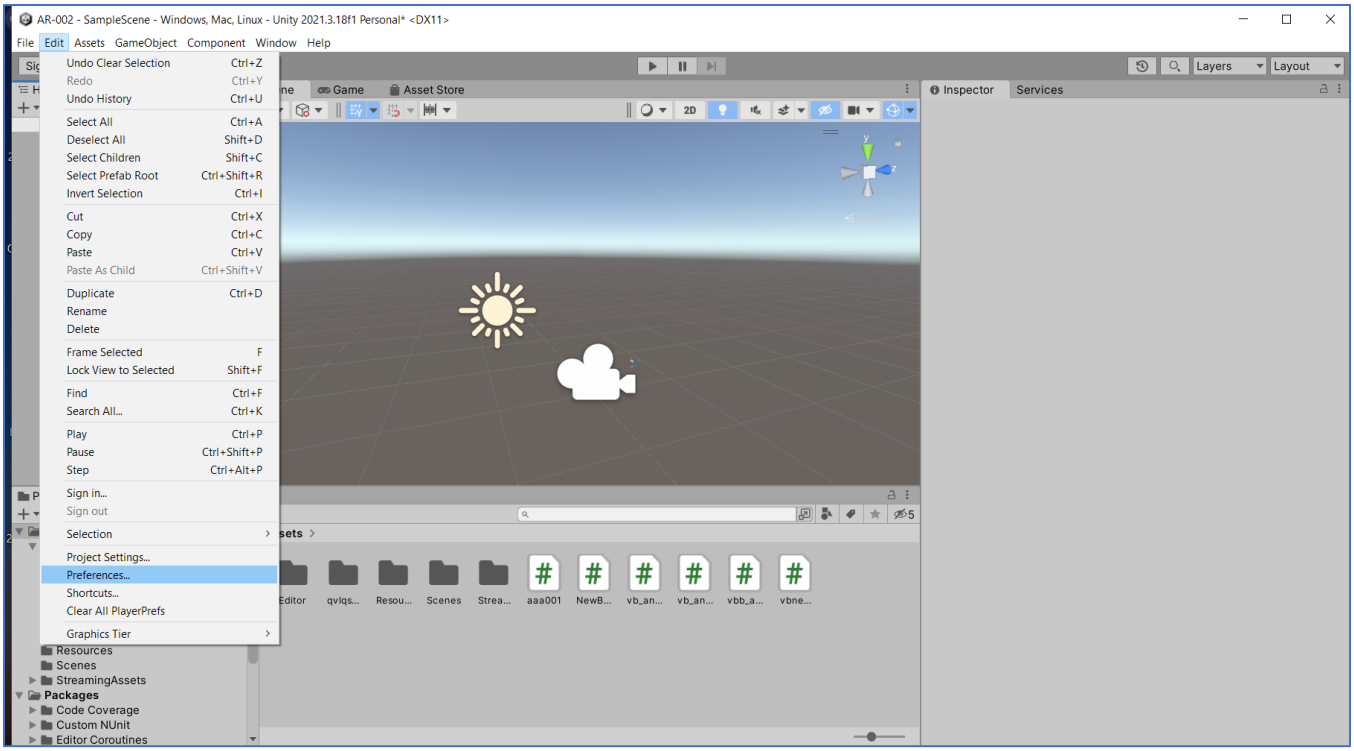


Следовательно, скрипт установления связей должен входить в состав Image target'a.

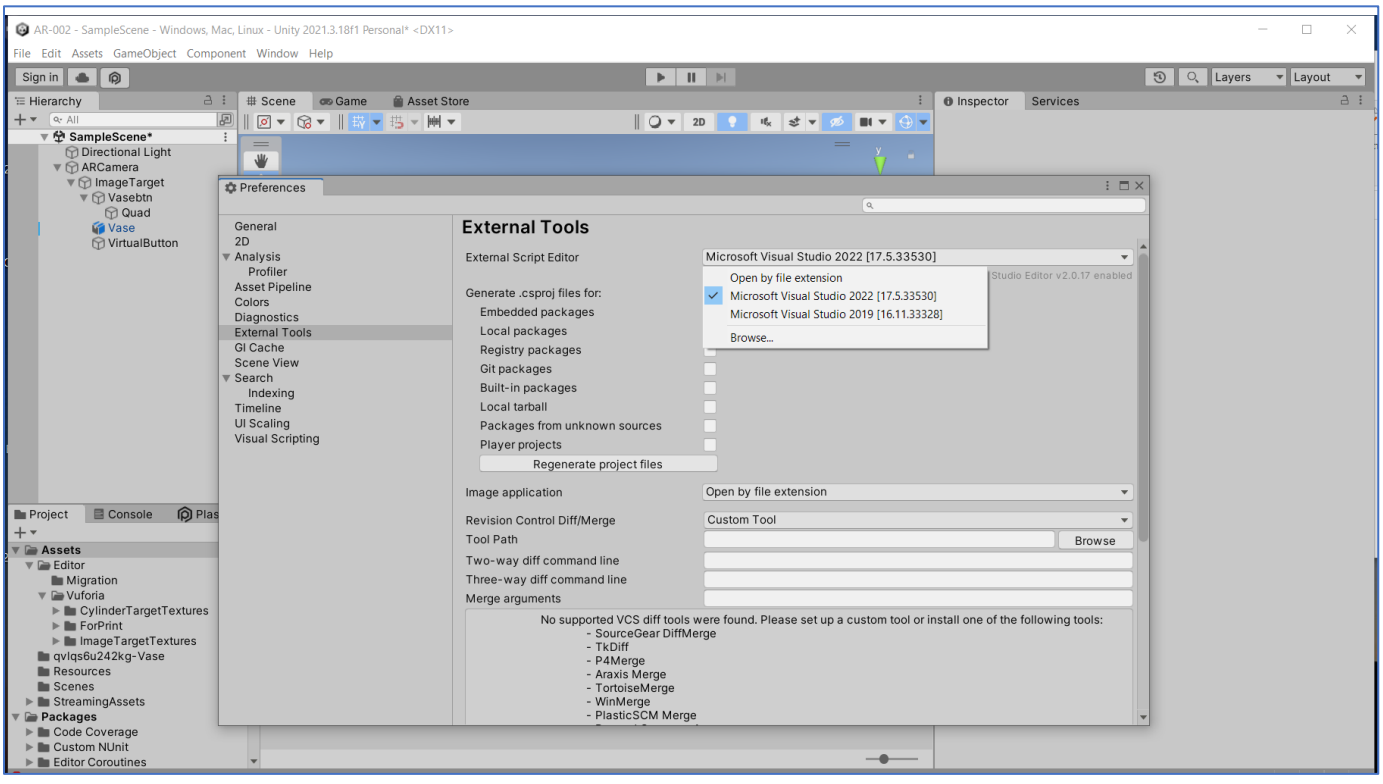
Механизм создания скрипта для **Image Target'a** заключается в следующем: в **Inspector'e** для данного объекта добавляется объект **Script**, уже содержащий минимально необходимый набор **лексем C#**. Разработчик в любой среде программирования (у нас это будет **Microsoft Visual Studio**, которая может быть уже установлена в процессе инсталляции **Unity 3D**, если вы это сделали в свое время правильно в процессе выполнения **ЛР №1**) создает программу – скрипт, управляющий взаимодействием между объектами.

В первую очередь убедитесь в том, что на Вашем локальном компьютере установлена **Microsoft Visual Studio**. Если она не установлена (что иногда происходит), установите ее из **Microsoft Store**. Предпочтительные варианты для работы с Unity версий 2020 и выше – **Visual Studio Community 2022** или **Visual Studio Community 2022.**

Для интеграции **Visual Studio** с **Unity** в меню **Edit** выберите опцию **Preferences**:

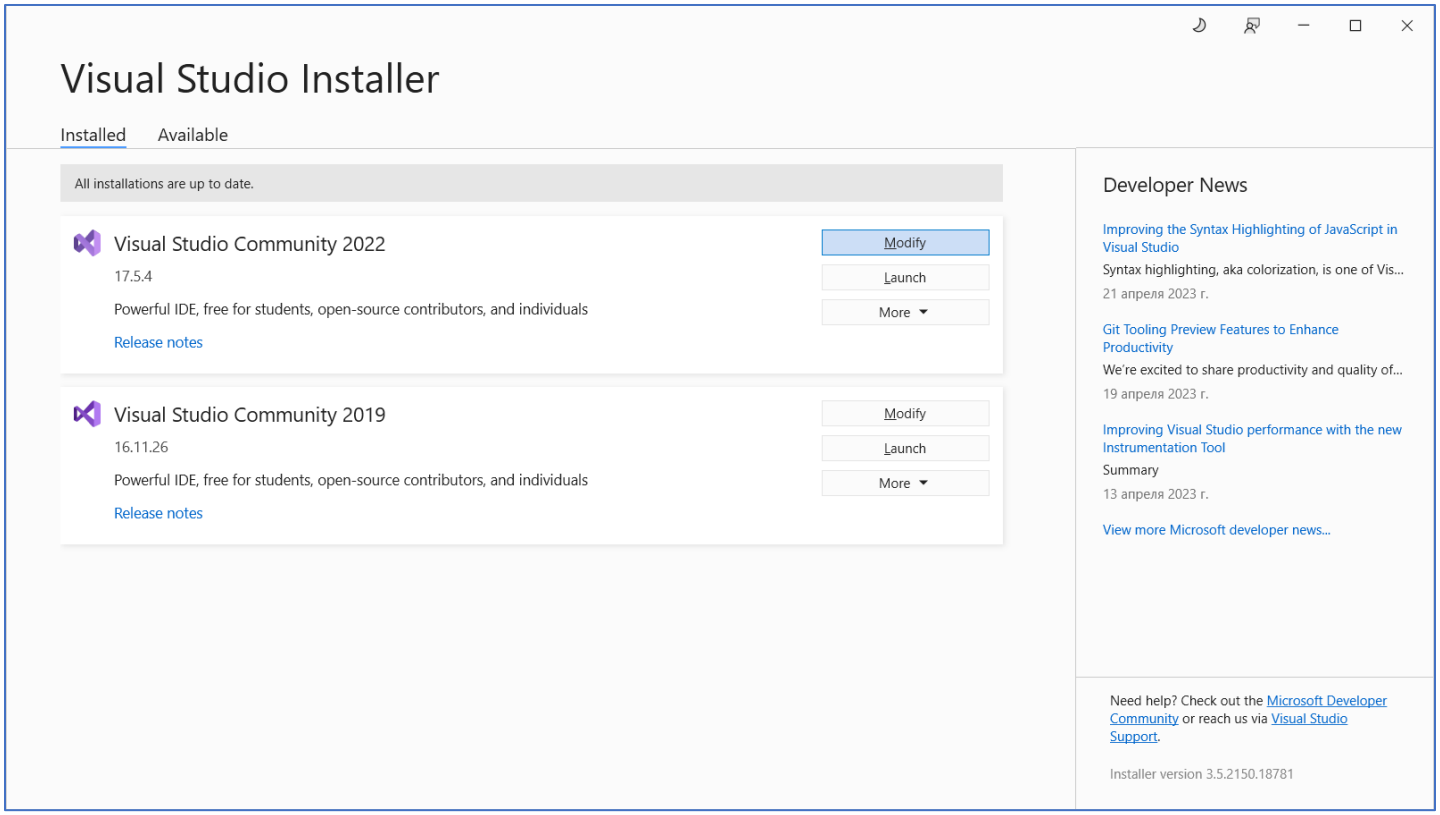


И в открывшемся окне выберите раздел **External Tools** и выполните необходимые установки:

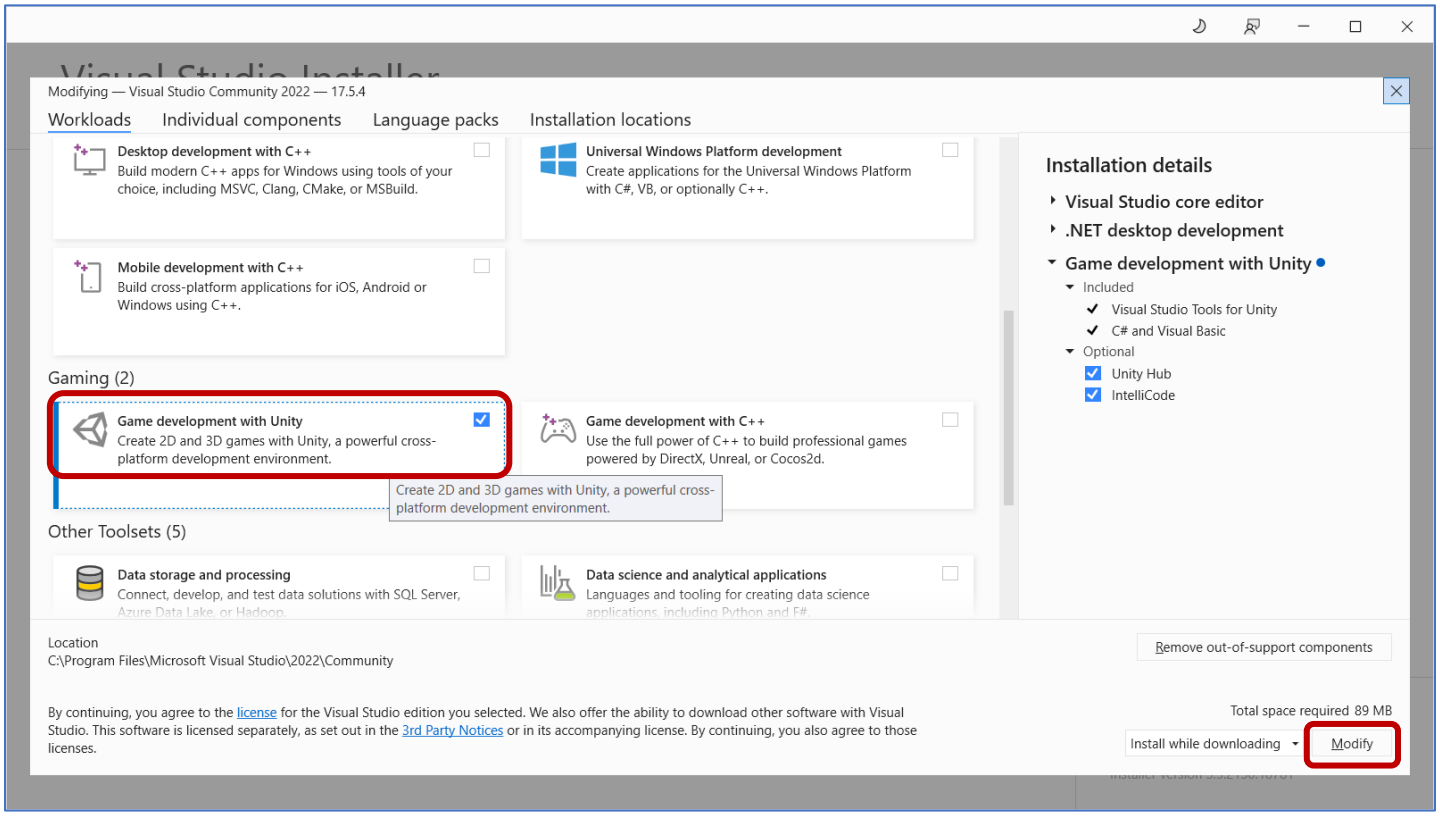


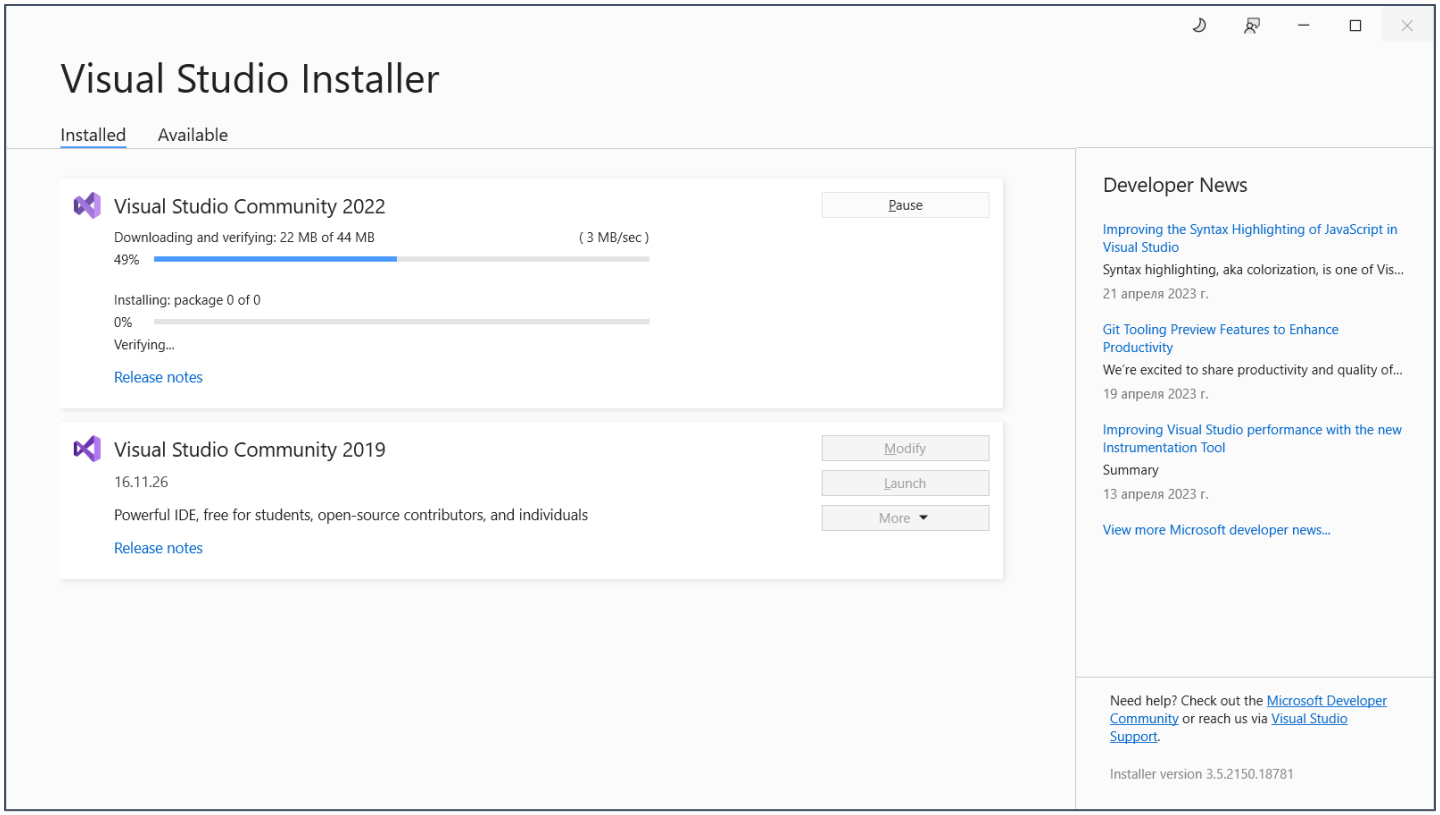
Однако, на этом не оканчивается работа по интеграции Visual Studio (новых релизов) и Unity.

Выполните модификацию **Visual studio**:



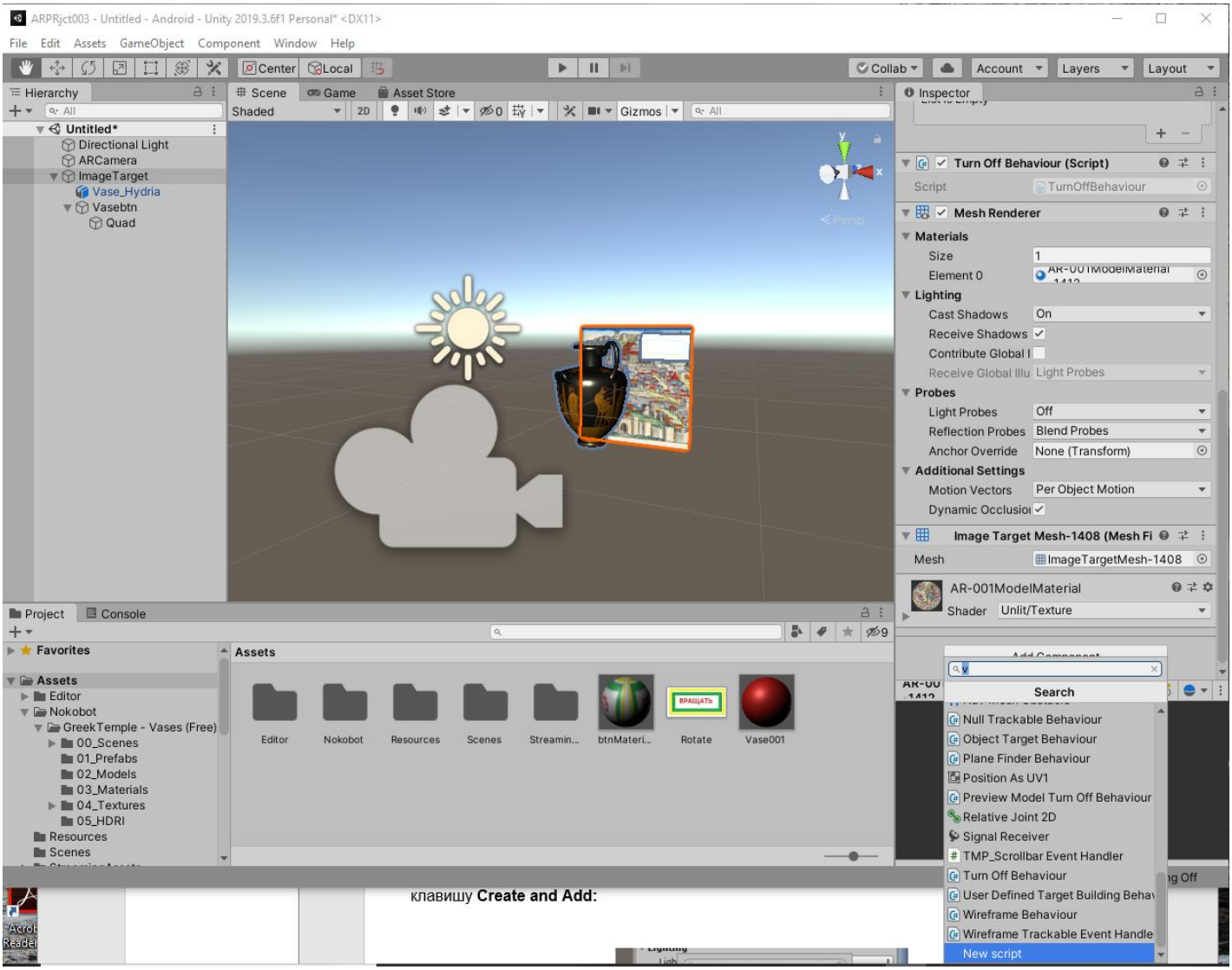
В открывающемся меню выбирайте компонент **Game Development with Unity**:



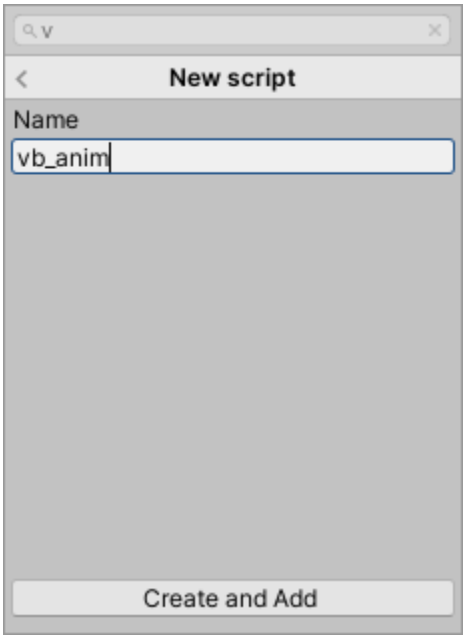


Переходим к разработке скрипта.

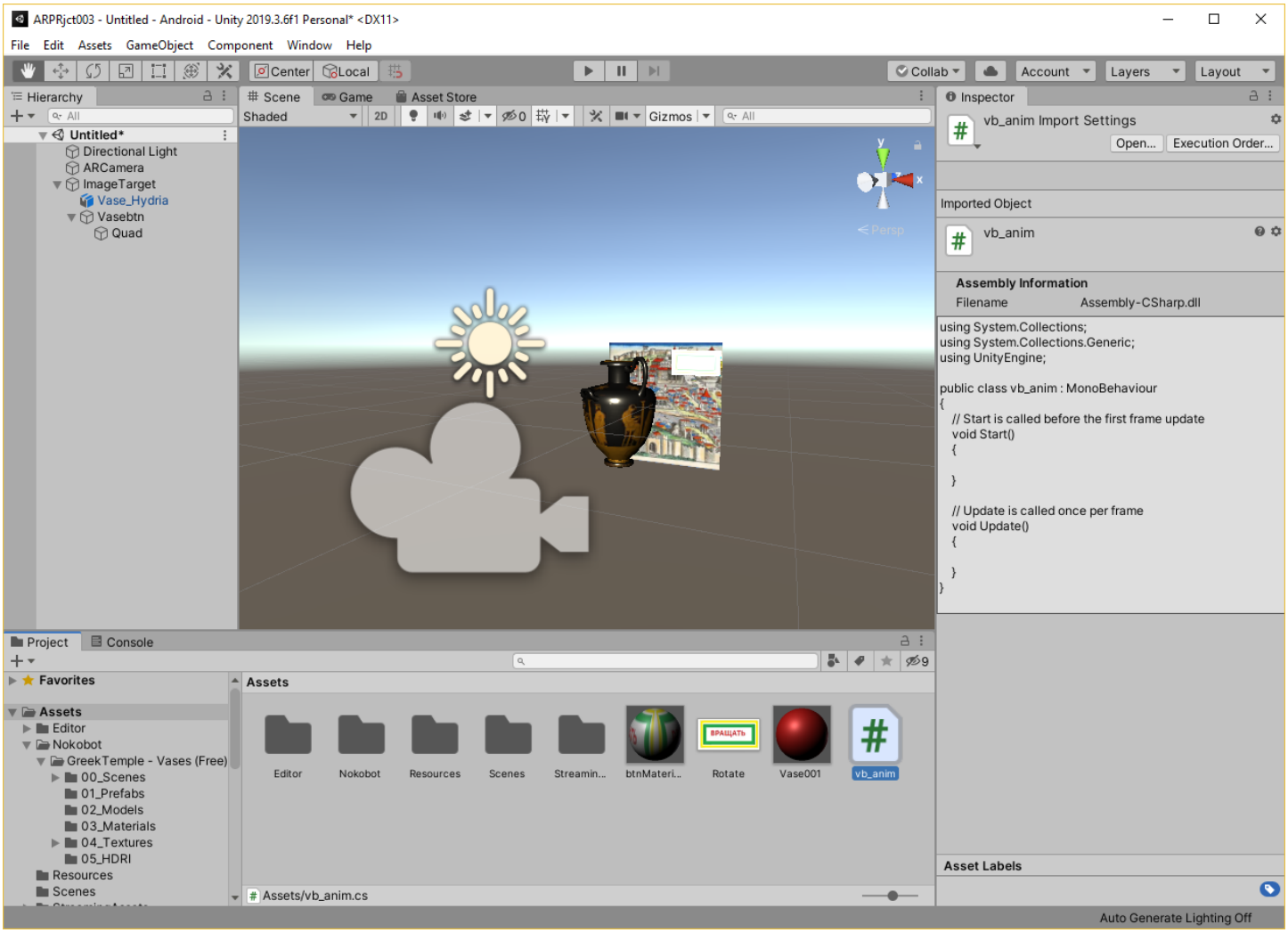
Для выделенного в **Hierarchy Image Target'a в Inspector'e** выбираем клавишу **Add Component** и в выпадающем меню ищем позицию **New Script**:



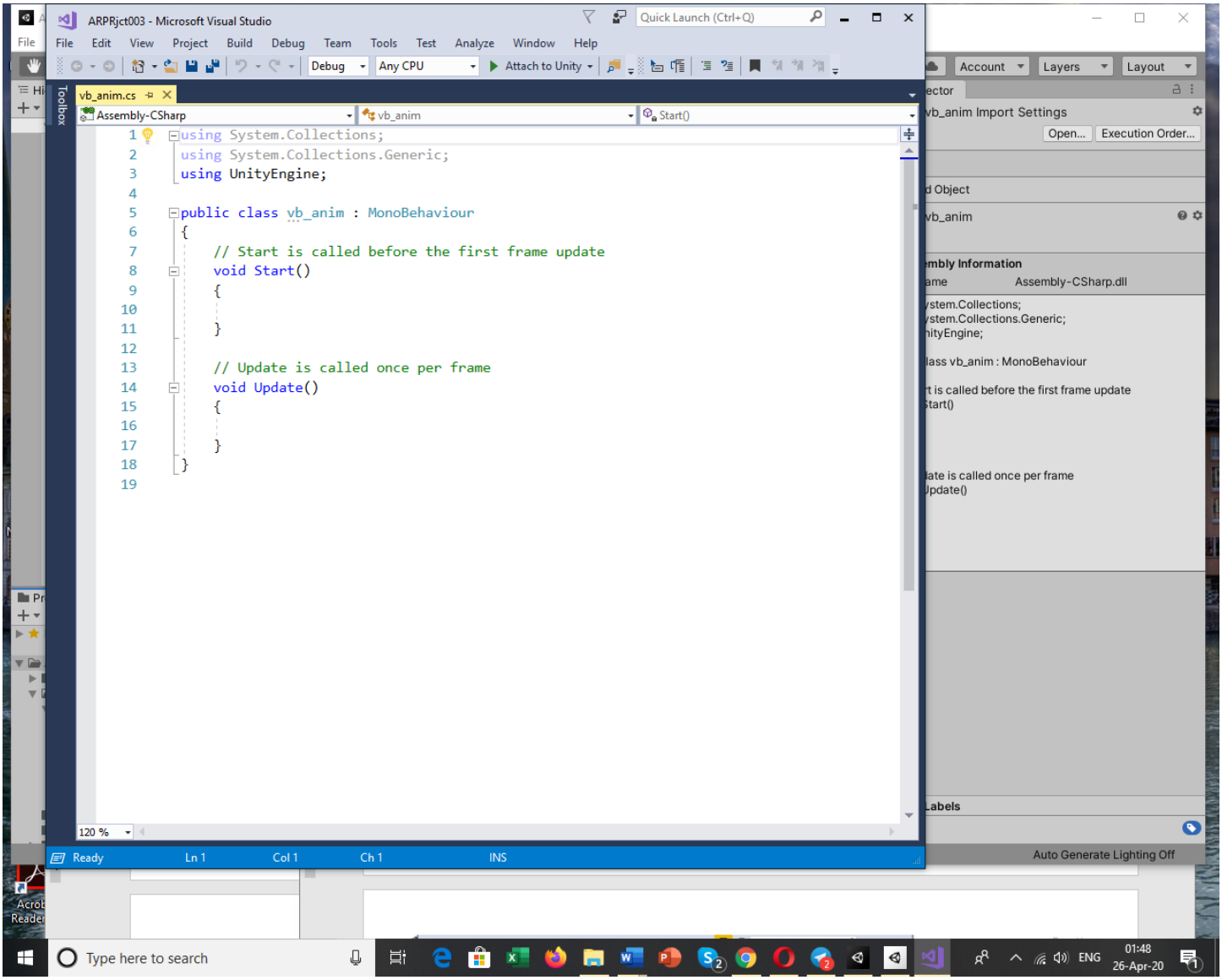
В открывшемся окне **New Script** в поле имени задаем имя, у нас **– vb_anim,** и нажимаем клавишу **Create and Add:**



В результате мы получаем шаблон будущего скрипта. Убедиться в этом можно, выделив в области **Project→Assets** пиктограмму скрипта **vb_anim**:



Для доработки шаблона надо перейти в среду программирования, дважды кликнув на иконку скрипта. В результате открывается среда разработки Microsoft Visual Studio 2022:



Модель поведения Приложения ДР в нашем случае следующая – при обнаружении события **«кнопка нажата»** заготовленная нами анимация (вращение) должна срабатывать; при обнаружении события **«кнопка отжата»** анимация прекращается.

Отображенный выше шаблон нужно дополнить методами и объектами из библиотеки **Vuforia** для реализации описанного сценария.

Реализация этого сценария обеспечивается библиотекой **Vuforia**, которую надо добавить в формируемый **Script**:

```
using Vuforia;
```

Реализация модели поведения выполняется с помощью подключения библиотек

```
UnityEngine;
UnityEngine.Events;
```

Обработка событий **«кнопка нажата»** и **«кнопка отжата»** реализуется с помощью программных конструкций

```
vbBtnObj.GetComponent<VirtualButtonBehaviour>().RegisterOnButtonPressed(OnButtonPressed);

vbBtnObj.GetComponent<VirtualButtonBehaviour>().RegisterOnButtonReleased(OnButtonReleased);
```

и

```
public void OnButtonPressed(VirtualButtonBehaviour vb) {
VaseAni.Play("Vase_Animation"); Debug.Log("Button pressed"); }

public void OnButtonReleased(VirtualButtonBehaviour vb) {
VaseAni.Play("none"); Debug.Log("Button released"); }
```

где `vbBtnObj и VaseAni` – локальные объекты, введенные в структуру шаблона и которым должна быть назначена связь с объектами нашей сцены в **Unity**:

```
public GameObject vbBtnObj;
public Animator VaseAni;
```

`vbBtnObj = GameObject.Find("Vasebtn"); и VaseAni.GetComponent<Animator>();` – обеспечивают связь с актуальными объектами нашей сцены в Unity.

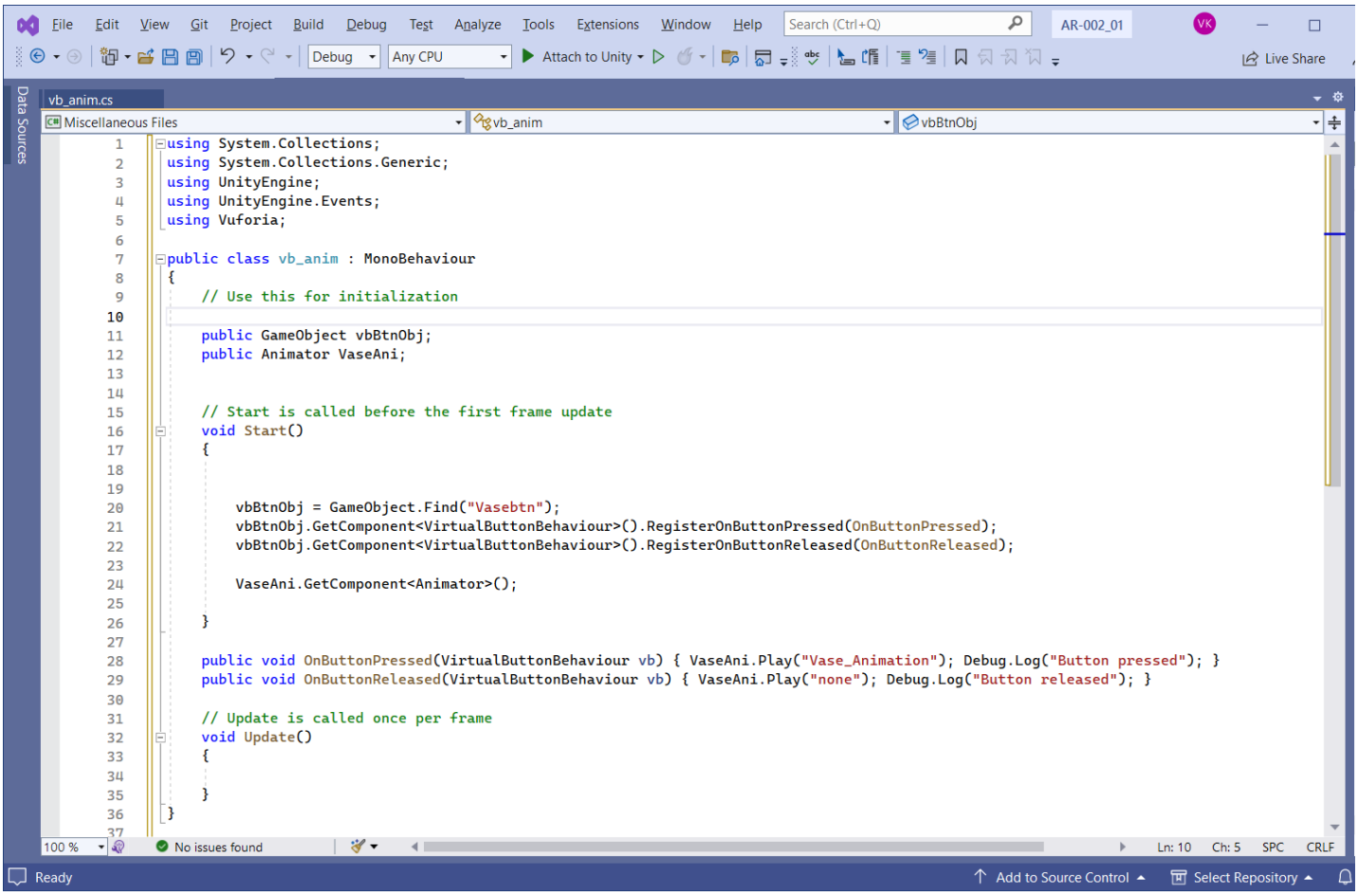
Для более подробного понимания возможностей работы в **Unity** с объектами «Виртуальная кнопка» вы можете обратиться к материалам **Unity** по теме: «**VirtualButtonBehaviour Class Reference**»

https://library.vuforia.com/sites/default/files/references/unity/classVuforia_1_1VirtualButtonBehaviour.html

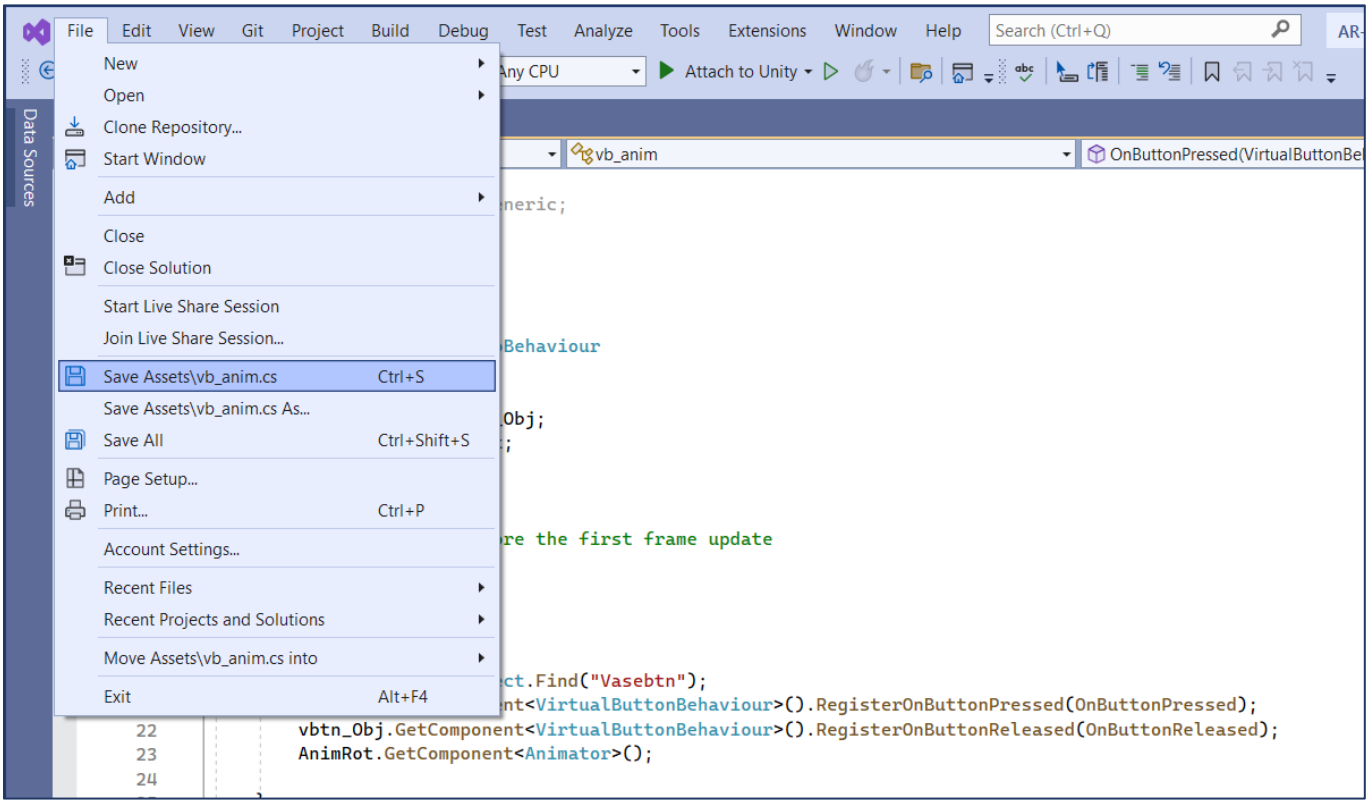
Предлагаем вашему вниманию работающий вариант построенного скрипта для нашего примера. Мы должны получить в среде разработки **Visual Studio** скрипт **vb_anim.cs**

Желтым цветом выделены доработки предложенного шаблона:

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

public class vb_anim : MonoBehaviour
{
   // Use this for initialization

    public GameObject vbBtnObj;
    public Animator VaseAni;


    // Start is called before the first frame update
    void Start()
    {


        vbBtnObj = GameObject.Find("Vasebtn");

vbBtnObj.GetComponent<VirtualButtonBehaviour>().RegisterOnButtonPressed(OnBu
ttonPressed);

vbBtnObj.GetComponent<VirtualButtonBehaviour>().RegisterOnButtonReleased(OnB
uttonReleased);

VaseAni.GetComponent<Animator>();

    }


public void OnButtonPressed(VirtualButtonBehaviour vb) {
VaseAni.Play("Vase_Animation"); Debug.Log("Button pressed"); }


public void OnButtonReleased(VirtualButtonBehaviour vb) {
VaseAni.Play("none"); Debug.Log("Button released"); }

    // Update is called once per frame

void Update()
    {

    }
 }
```

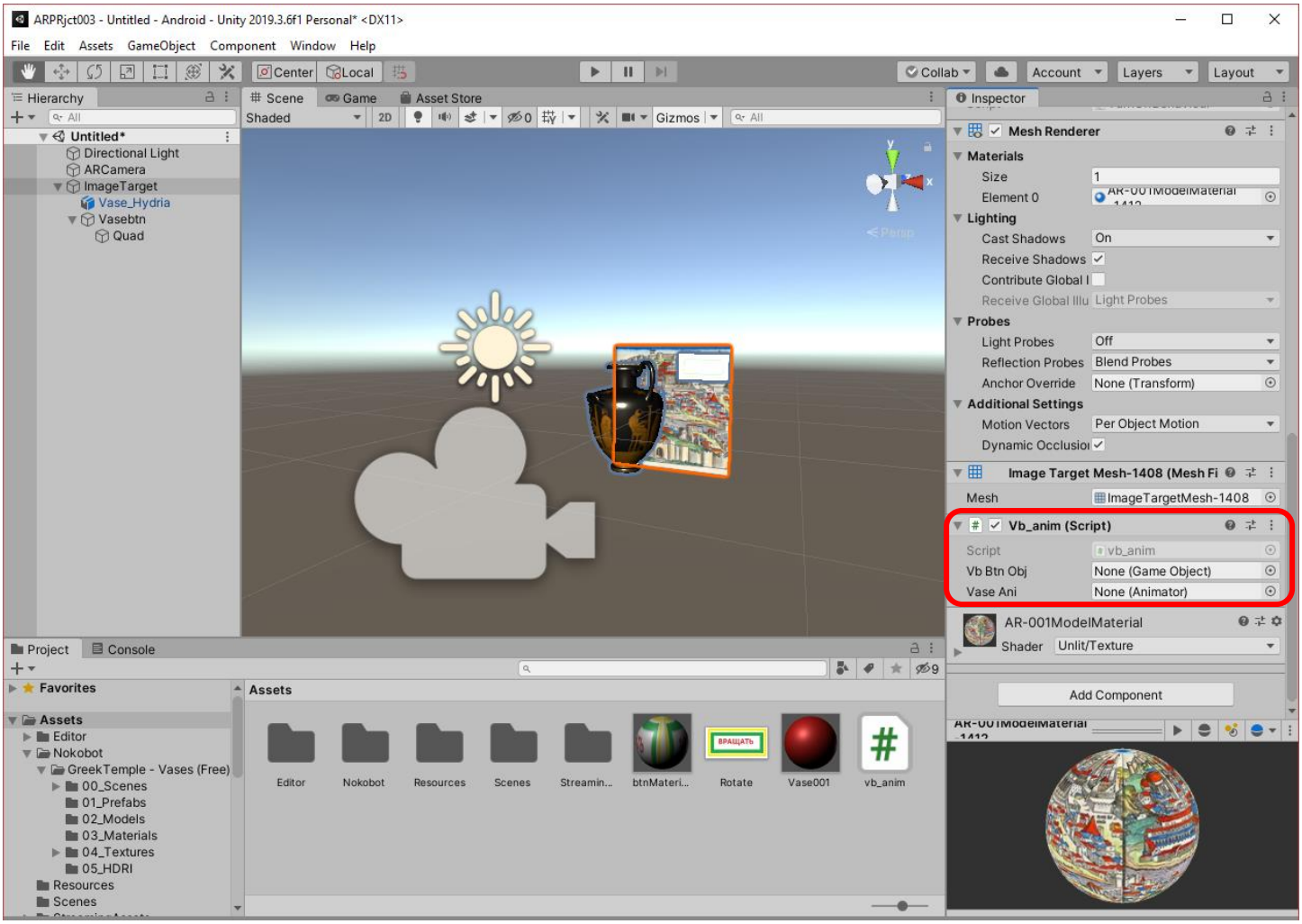


Убеждаемся в том, что скрипт не содержит синтаксических ошибок и сохраняем его в проекте в локальной ФС. (в **Vusual Studio: File → Save Assets/vb_anim.cs**) →



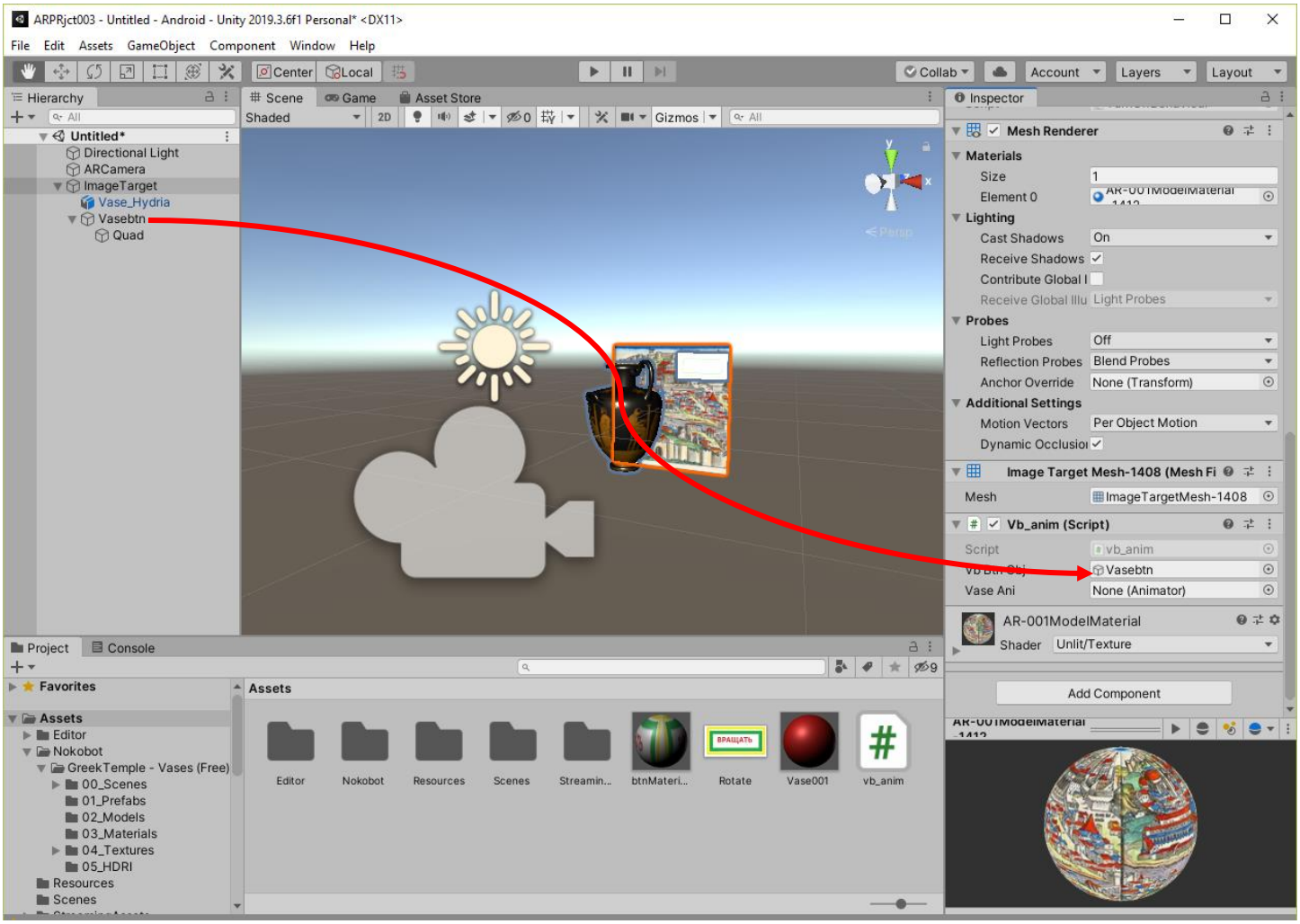
В результате в **Unity 3D в Inspector’e** мы увидим измененный и доработанный скрипт:

Возвращаемся к объекту **Image target.** В **Inspector’e** для него мы видим раздел, связанный со скриптом:

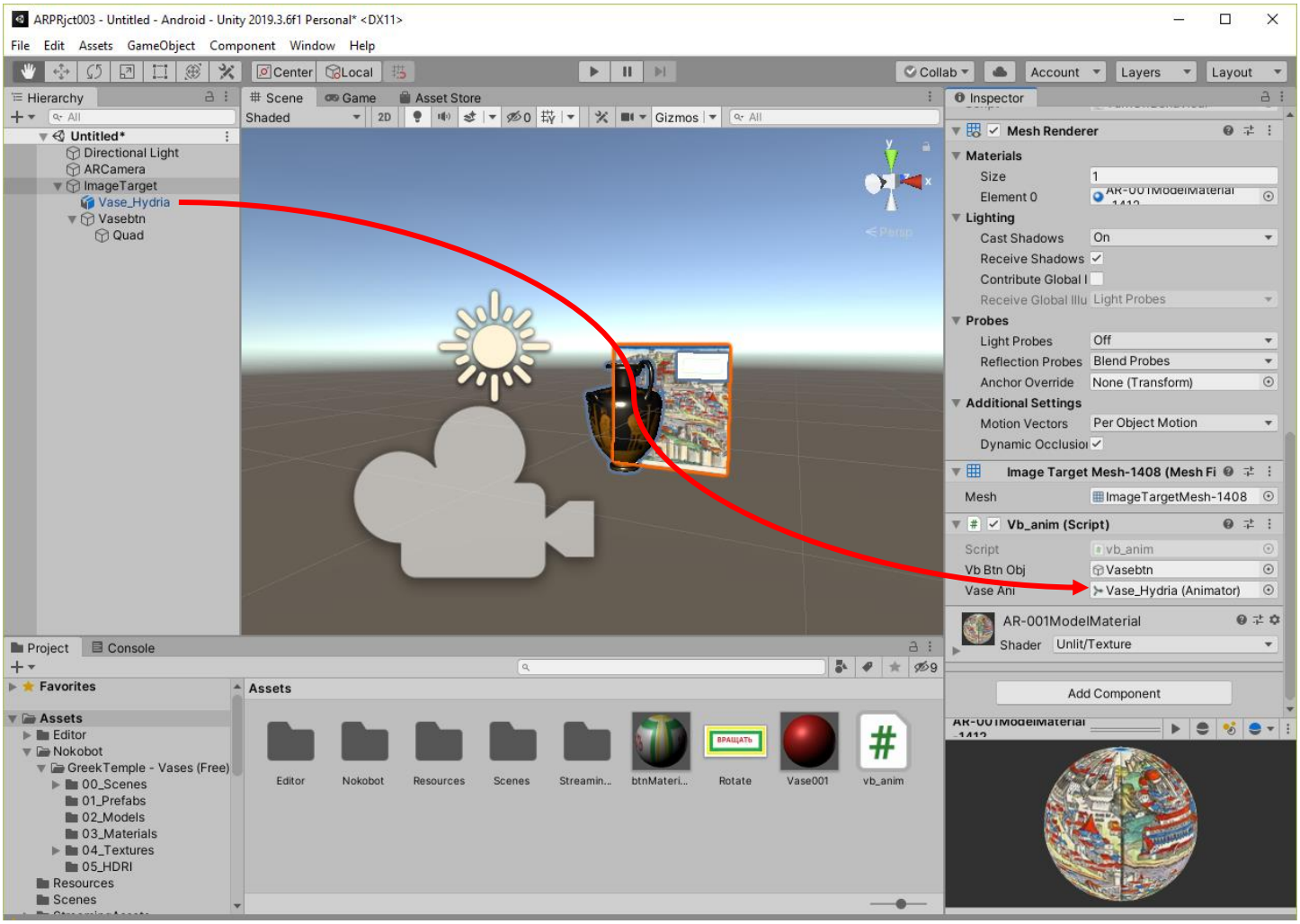


Обратите внимание, что скрипт в **Inspector'е** представлен двумя точками входа – **Vb Btn Obj** и **Vase Ani**, которым соответствуют заведенные нами в скрипте объекты **vbBtnObj** и **VaseAni**. Один из них отвечает за **виртуальную кнопку** (например, **vbBtnObj**), второй – за анимационную последовательность (например **VaseAni**). При этом в соответствующих полях **Inspector'а** пока против этих точек входа нет ни одного объекта (**None** – соответствующий квалификатор).

Привяжем к объекту **Vb Btn Obj** скрипта **vb_anim** виртуальную кнопку, подготовленную нами на предыдущих этапах, **Vasebtn**. Для этого операцией **drag-n-drop из Hierarchy** перетаскиваем кнопку **Vasebtn** в поле **Inspector'а**:



Аналогичные действия производим с объектом **Vase Ani –** связываем с ним объект **Vase_Hydria (Animator)**, который нам доступен как ссылка **Vase_Hydria в Hierarchy**. См. скрипт. Обратите внимание – аниматор является компонентом описанного объекта **Vase_Hydria.**

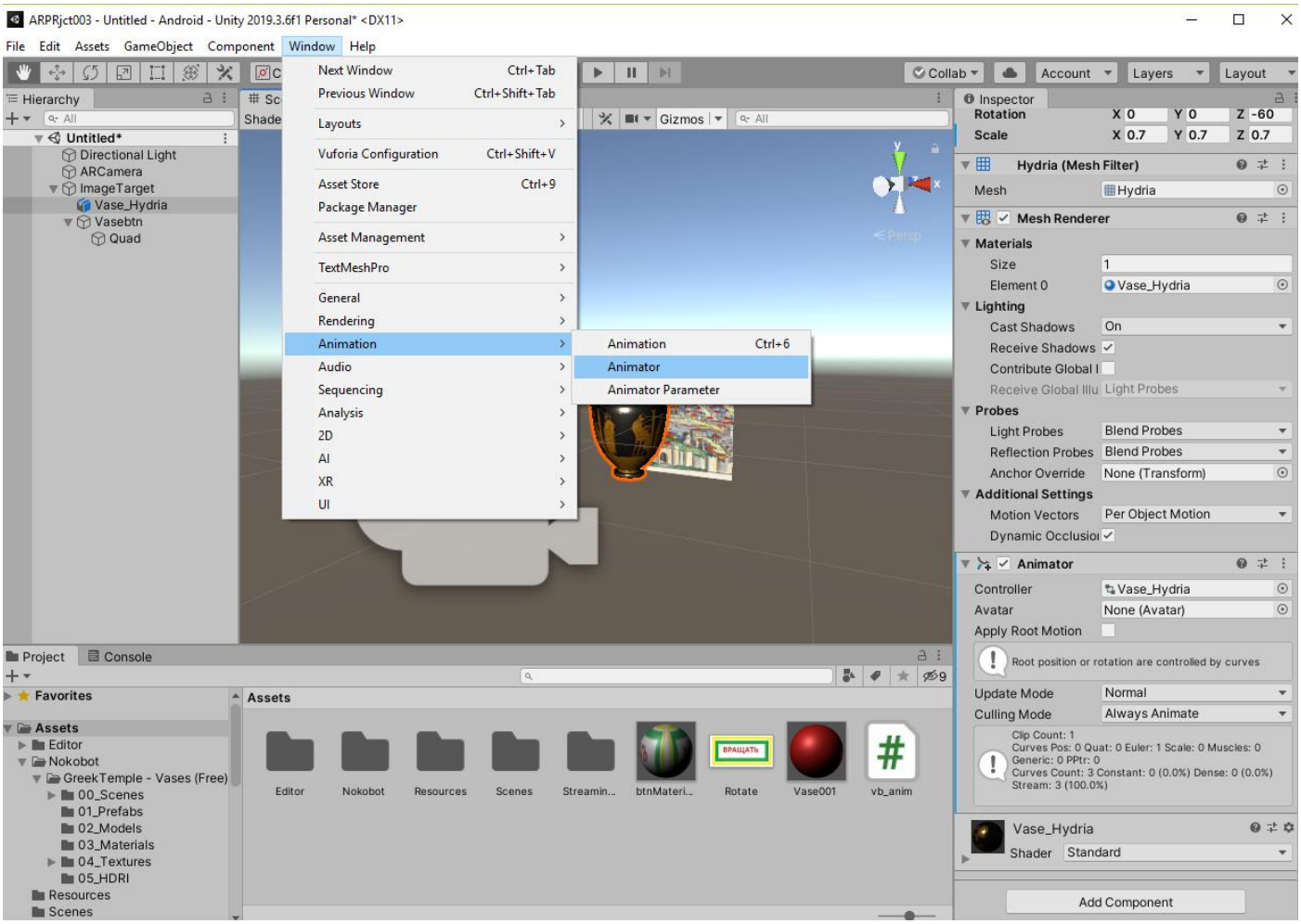


Доработаем взаимодействие со скриптом при наступлении события **`OnButtonReleased.`**

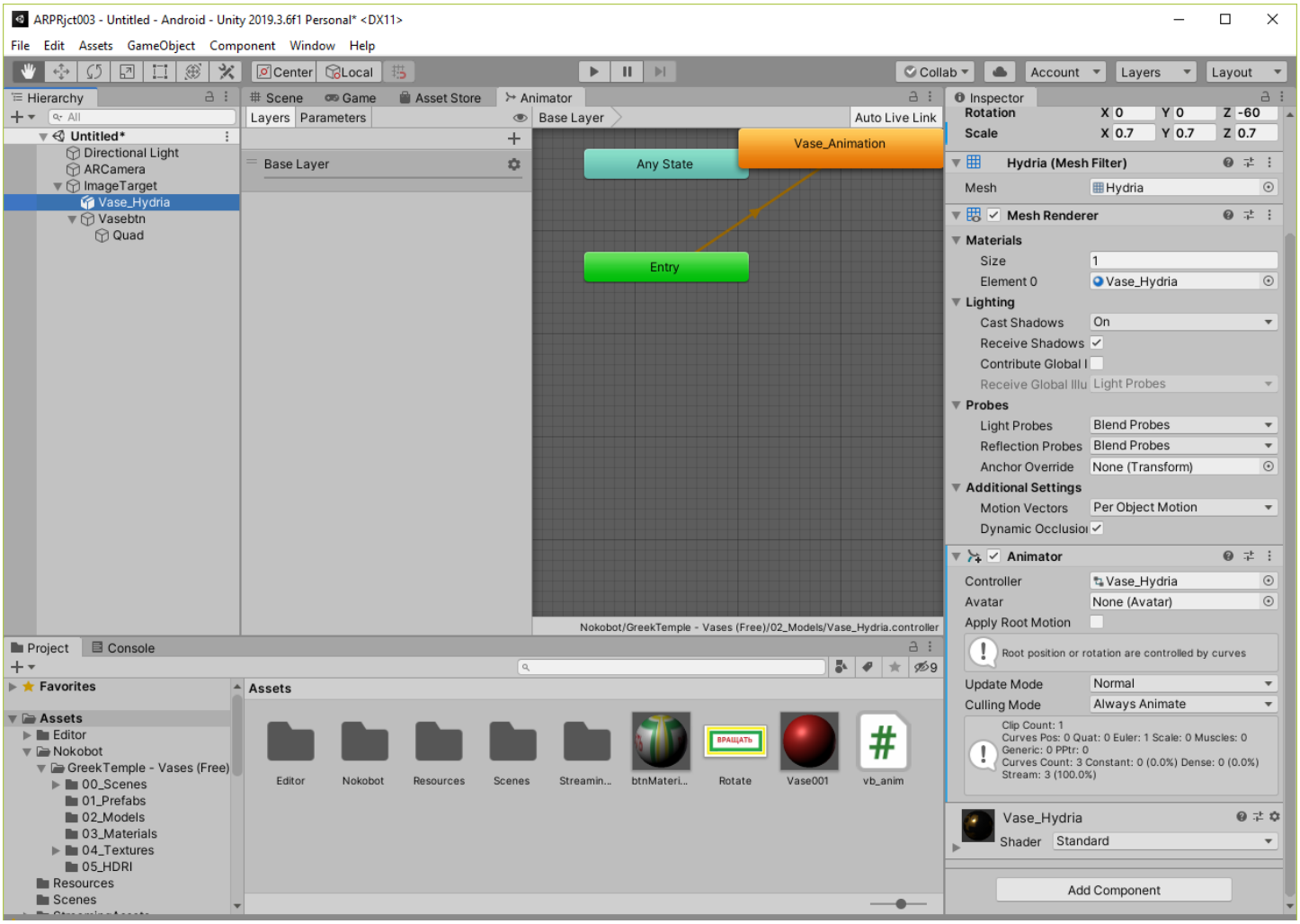
Для этого необходимо реализовать сценарий, при котором при отжатой кнопке (событие **`OnButtonReleased`**) анимация отключена – в скрипте это соответствует оператору

```
VaseAni.Play("none");
```

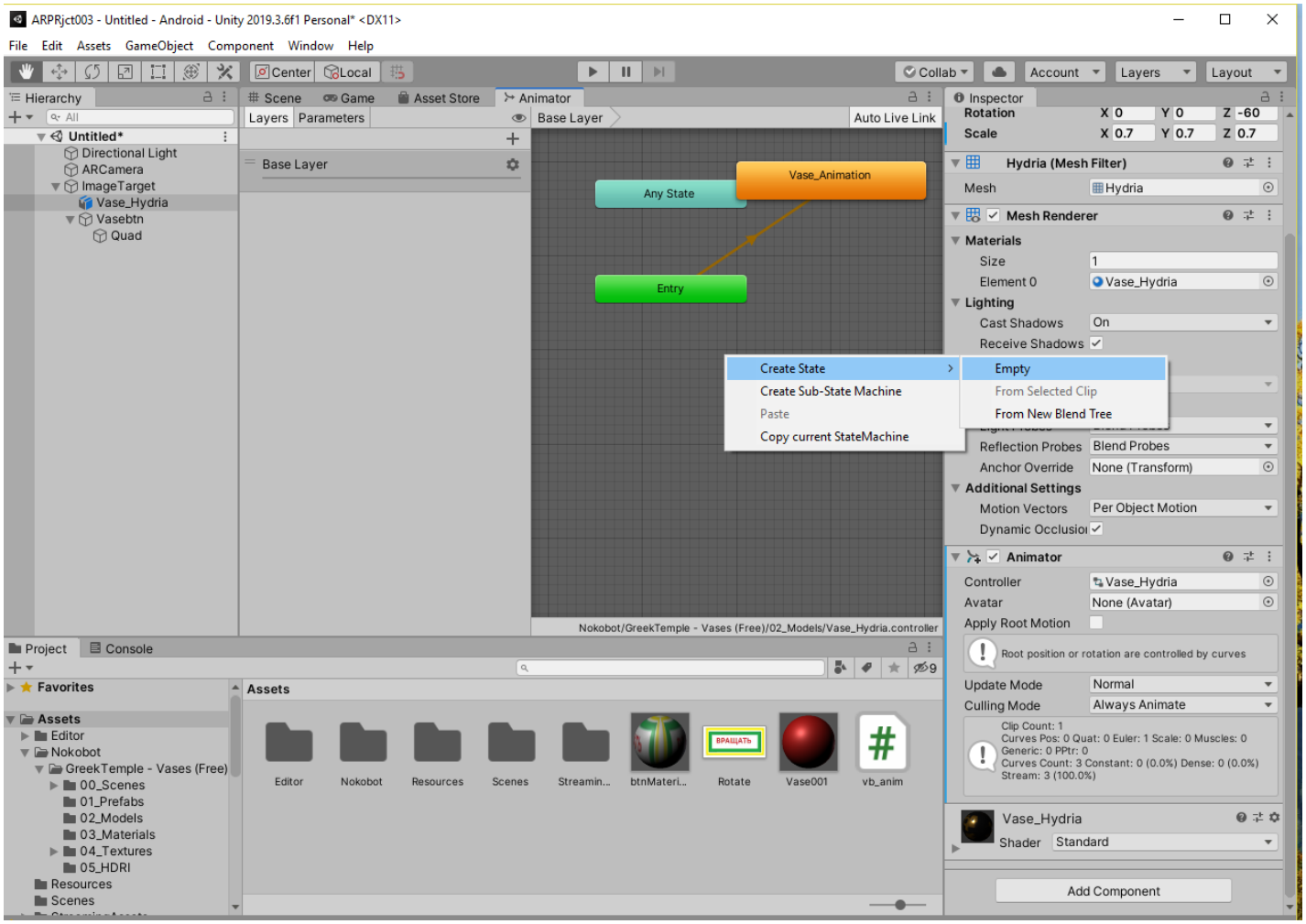
Это означает, что мы должны скорректировать компонент **Animator**, добавив в него **«none»,** которого пока еще нет. Выделяем в **Hierarchy** нашу **3D Модель Vase_Hydria**, а в основном меню выполняем последовательность вызовов **Windows→Animation→Animator**:



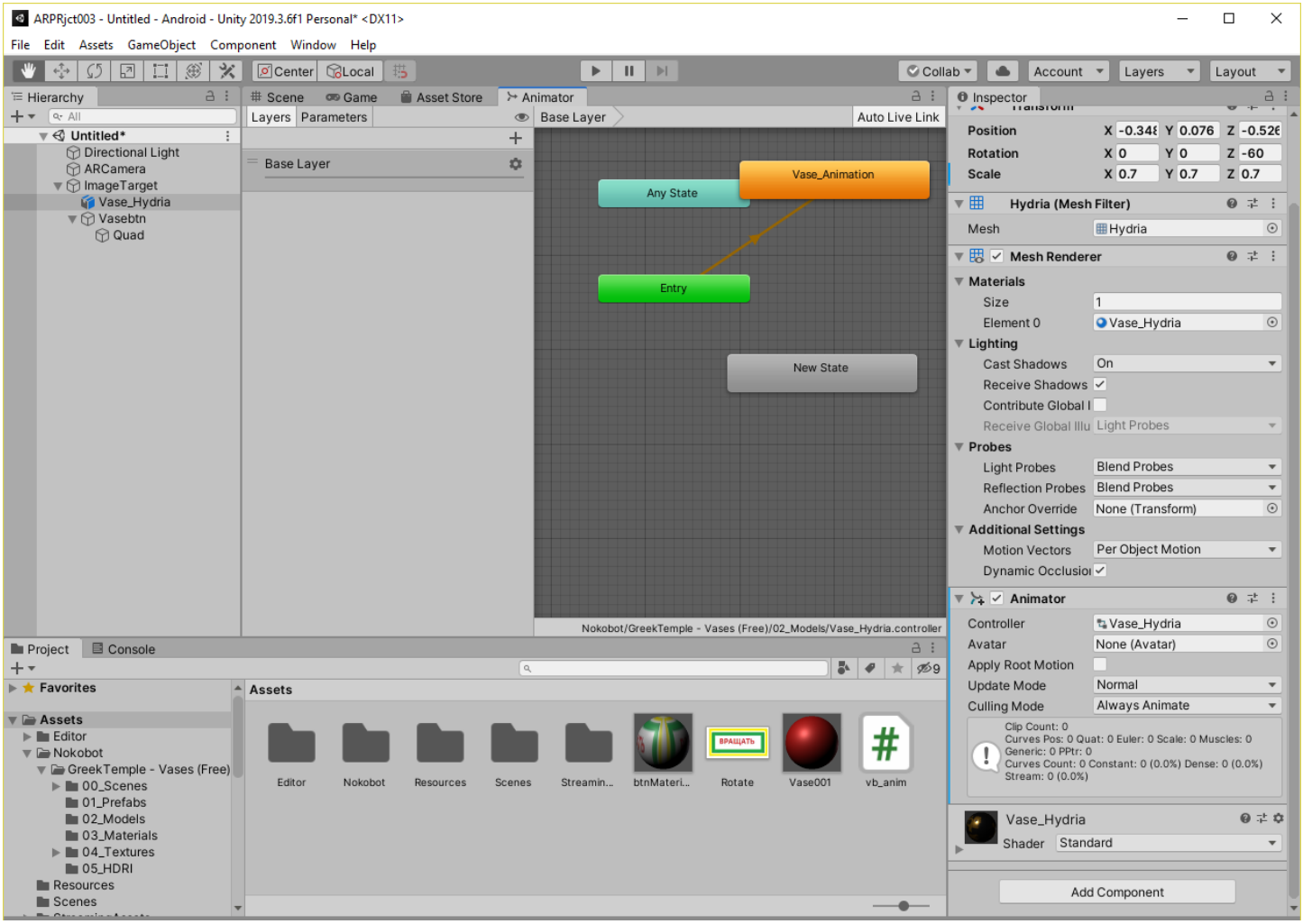
В результате получаем новое окно:



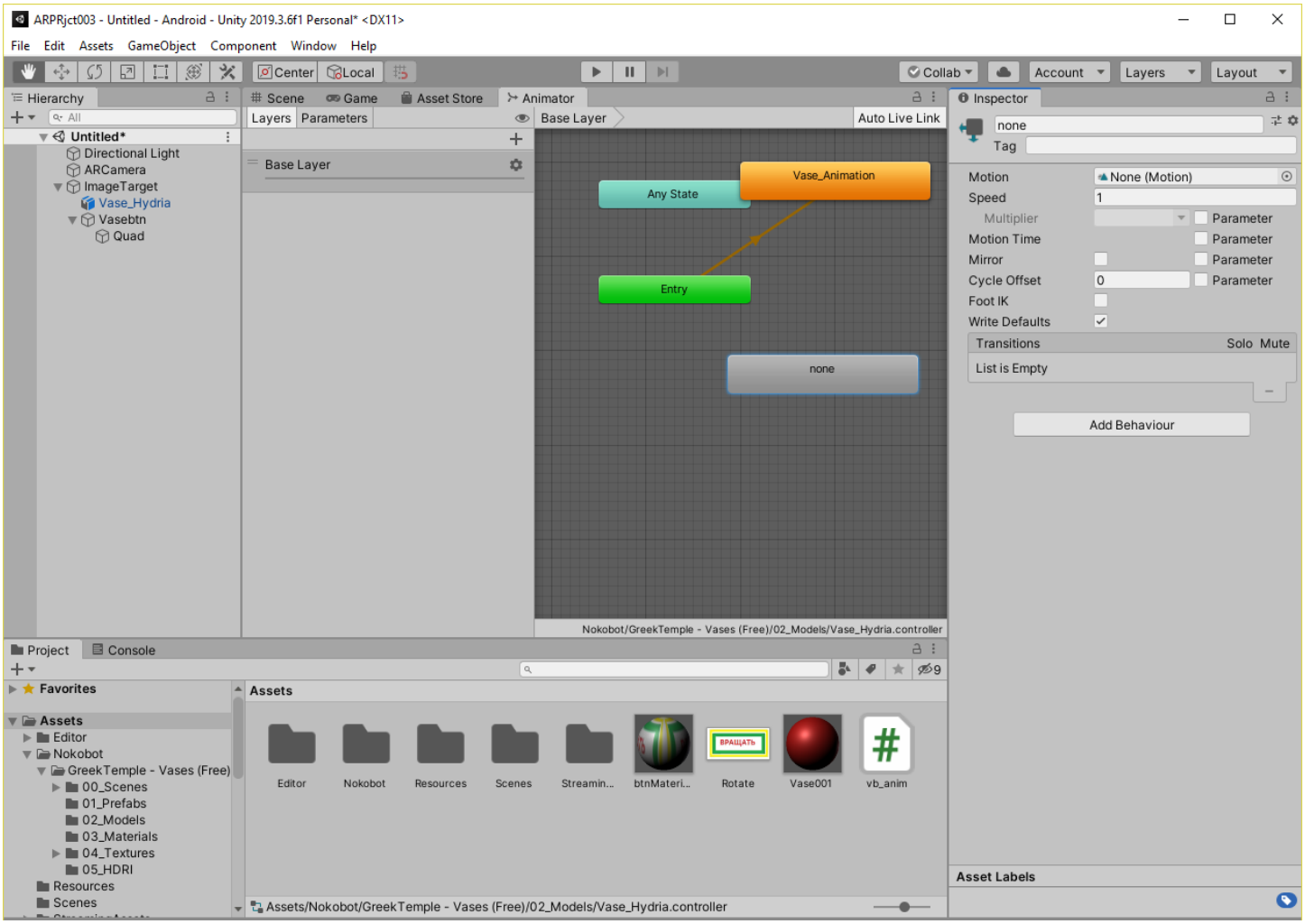
Добавляем в этом окне компонент **none.** Для этого в пустом месте окна **Animator** кликаем правой клавишей и добавляем состояние **Empty:**


ARPRjct003 - Untitled - Android - Unity 2019.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Untitled*
Directional Light
ARCamera
ImageTarget
Vase_Hydria
Vasebtn
Quad
Scene Game Asset Store Animator
Layers Parameters
Base Layer
Any State
Vase_Animation
Entry
Create State
Create Sub-State Machine
Paste
Copy current StateMachine
Empty
From Selected Clip
From New Blend Tree
Inspector
Hydria (Mesh Filter)
Mesh Renderer
Materials
Lighting
Additional Settings
Animator
Controller
Avatar
Apply Root Motion
Update Mode
Culling Mode
Add Component
Project Console
Favorites
Assets
Editor
Nokobot
GreekTemple - Vases (Free)
00_Scenes
01_Prefabs
02_Models
03_Materials
04_Textures
05_HDRI
Resources
Scenes
Vase_Hydria
Shader
Standard

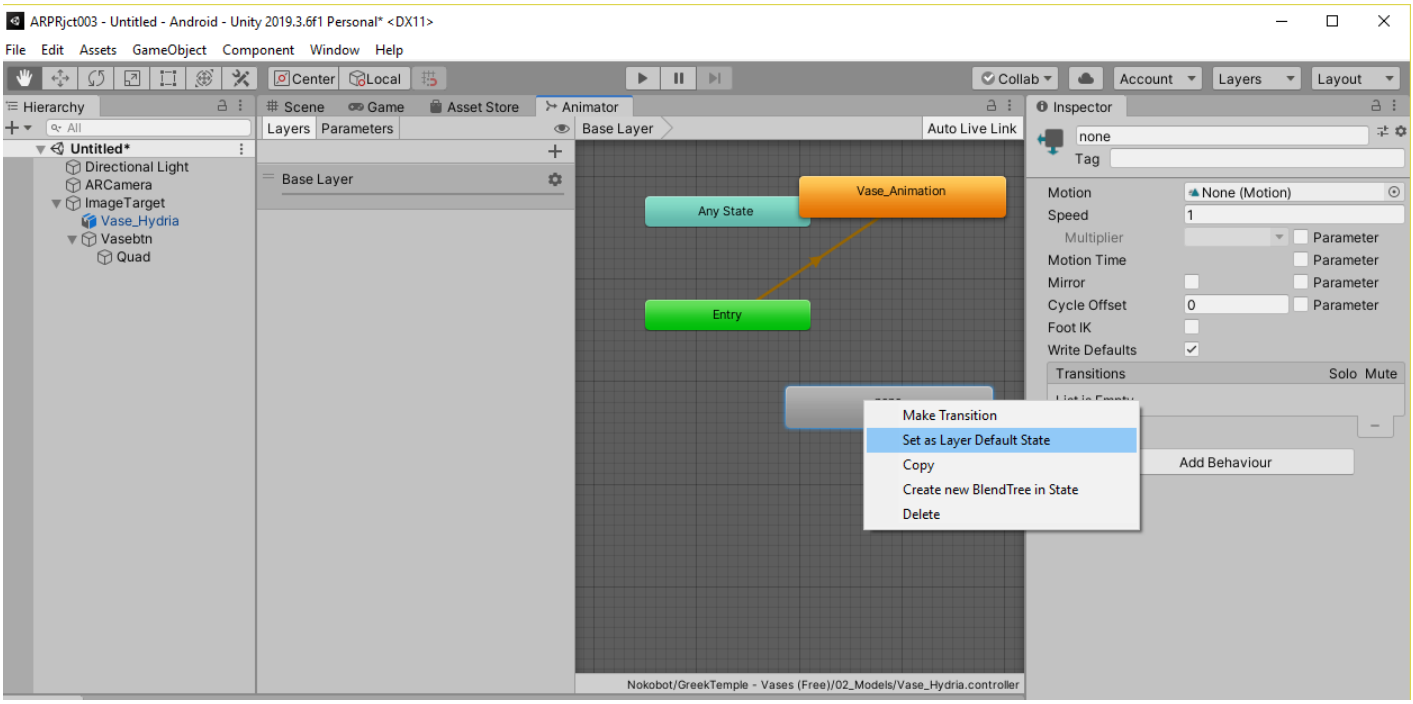
Результат:



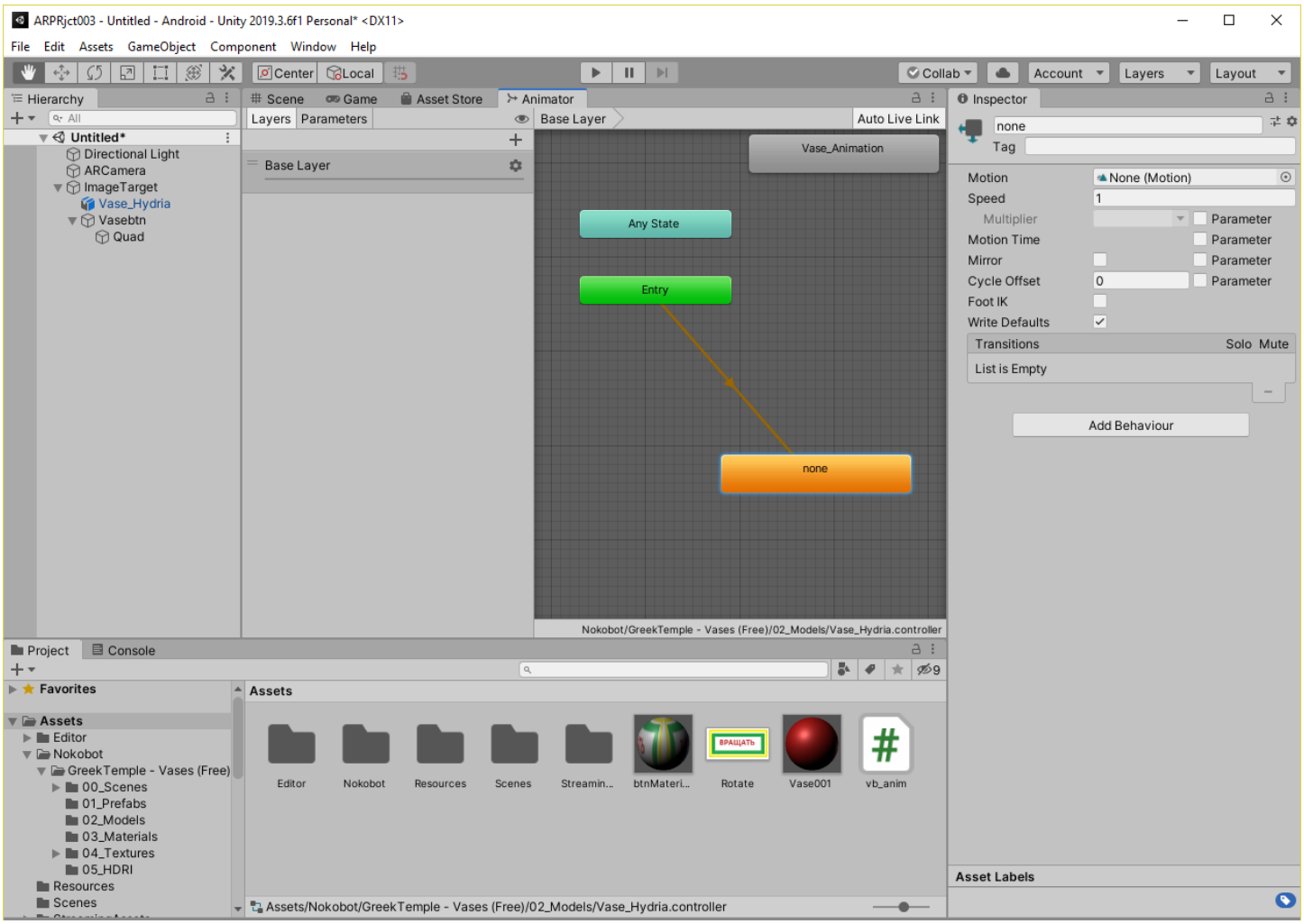
Переименовываем **New State** на **none**, что делается непосредственно в **Inspector'e** в первом текстовом поле:



Как видно из графического представления программы, анимация (**Vase_Animation**) запускается сразу после входа (**Entry**) в компонент **Аниматор** при любом начальном состоянии клавиши – состояние по умолчанию. Нам необходимо блокировать анимацию при старте программы, т.к. по условиям сценария анимация производится только при наступлении события **«нажатая клавиша»**. Для этого мы должны в качестве состояния по умолчанию использовать не **Vase_Animation,** а **none.** (см. скрипт). Для этого подводим курсор в окне **Animator** к **none**, по правой клавише мыши выбираем действие **Set as Layer Default State:**



В результате Аниматор будет выглядеть следующим образом:



Работа по созданию сцены завершена. Проверка достигнутого в разрабатываемой сцене выполняется в режиме **Game (Play)** с использованием камеры локального компьютера.

3. **Создание** файла **.apk** для загрузки Приложения ДР на **Android**-МУ.

Все установки, настройки и манипуляции с окнами и функциями **Unity 3D** для создания файла **.apk** идентичны тем, что вы выполняли в ЛР№2, части 1, 2, 3. Не забудьте снабдить свое загружаемое на МУ Приложение иконкой!

Сохраняем собранный файл **.apk** в локальной файловой системе разработчика, загружаем и устанавливаем его на **Android -** МУ любым известным вам способом.

Видео разработанного в данной ЛР Приложения ДР доступно по ссылке:

https://youtu.be/BL9qEuMptd0

Разработанное в рамках ЛР № 3, Часть 2 AR-Приложение необходимо продемонстрировать преподавателю. Для этого загрузите свой .apk на любой файлообменник или доступное облако и пришлите мне ссылку!

Не забудьте про таргет!