Input Manager

EventSystem

Камера

NavMash

ProBuilder

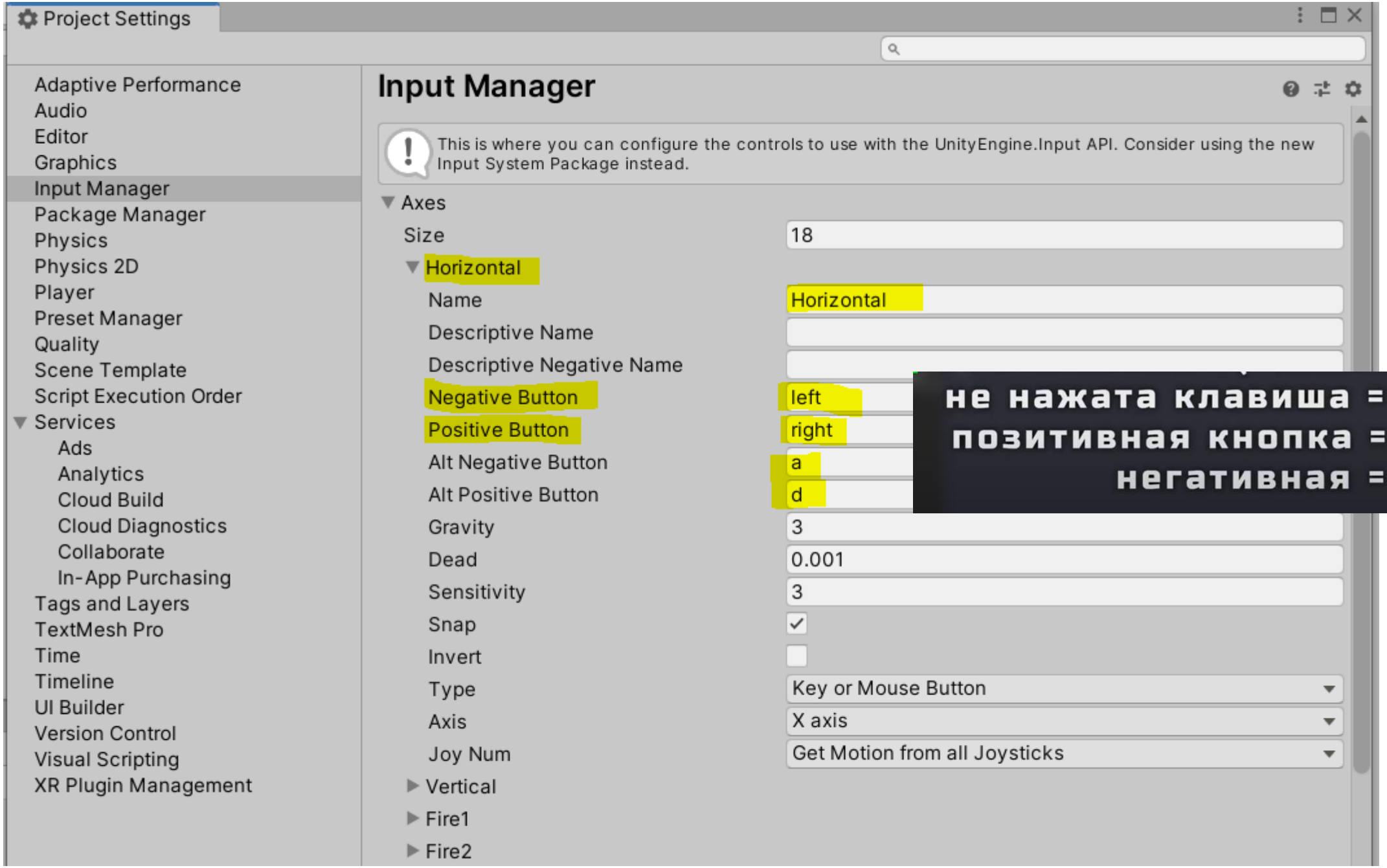
Joints

Менеджер ввода (Input Manager)

Окно диспетчера ввода позволяет вам определять оси ввода и связанные с ними действия для вашего проекта. Чтобы получить к нему доступ, в главном меню Unity выберите **Edit > Project Settings**, затем выберите **Input Manager** в меню навигации справа.

Диспетчер ввода использует следующие типы элементов управления:

- **Key** относится к любой клавише на физической клавиатуре, такой как *W, Shift* или пробел.
- **Button** — это любая кнопка на физическом контроллере (например, геймпадах), например кнопка X на пульте дистанционного управления.
- **Virtual axis** сопоставляется с элементом управления, таким как кнопка или клавиша. Когда пользователь активирует элемент управления, ось получает значение в диапазоне [–1..1]..


Project Settings
Adaptive Performance
Audio
Editor
Graphics
Input Manager
Package Manager
Physics
Physics 2D
Player
Preset Manager
Quality
Scene Template
Script Execution Order
Services
Ads
Analytics
Cloud Build
Cloud Diagnostics
Collaborate
In-App Purchasing
Tags and Layers
TextMesh Pro
Time
Timeline
UI Builder
Version Control
Visual Scripting
XR Plugin Management
Input Manager
This is where you can configure the controls to use with the UnityEngine.Input API. Consider using the new Input System Package instead.
Axes
Size 18
Horizontal
Name Horizontal
Descriptive Name
Descriptive Negative Name
Negative Button left
Positive Button right
Alt Negative Button a
Alt Positive Button d
Gravity 3
Dead 0.001
Sensitivity 3
Snap
Invert
Type Key or Mouse Button
Axis X axis
Joy Num Get Motion from all Joysticks
Vertical
Fire1
Fire2
не нажата клавиша = 0
позитивная кнопка = 1
негативная = -1

Property	Функция
Name	Имя оси. Вы можете использовать это для доступа к оси из скриптов.
Descriptive Name, Descriptive Negative Name	Эти значения устарели и не работают. Раньше они отображались для пользователя на экране Rebind Controls при запуске, но этот экран также устарел.
Negative Button, Positive Buttor	Элементы управления для перемещения оси в отрицательном и положительном направлении соответственно. Это могут быть клавиши на клавиатуре, кнопки на джойстике или мыши.
Alt Negative Button, Alt Positive Button	Альтернативные элементы управления для перемещения оси в отрицательном и положительном направлении соответственно.
Gravity	Скорость в единицах в секунду, с которой ось падает в нейтральное положение при отсутствии входных данных.
Dead	Как далеко пользователь должен переместить аналоговый джойстик, прежде чем ваше приложение зарегистрирует движение. Во время выполнения ввод со всех аналоговых устройств, попадающих в этот диапазон, будет считаться нулевым.
Sensitivity	Скорость в единицах в секунду, с которой ось будет двигаться к целевому значению. Это только для цифровых устройств.
Snap	Если включено, значение оси будет сброшено до нуля при нажатии кнопки, соответствующей противоположному направлению.
Type	Тип ввода, который управляет осью. Выберите одно из следующих значений: - Клавиша или кнопка мыши - Движение мыши - Ось джойстика
Axis	Ось подключенного устройства, которое управляет этой осью.
JoyNum	Подключенный джойстик, управляющий этой осью. Вы можете выбрать конкретный джойстик или запросить ввод со всех джойстиков.

```
float moveSpeed = 10;
//Define the speed at which the object moves.

float horizontalInput = Input.GetAxis("Horizontal");
//Get the value of the Horizontal input axis.

float verticalInput = Input.GetAxis("Vertical");
//Get the value of the Vertical input axis.

transform.Translate(new Vector3(horizontalInput, verticalInput, 0) * moveSpeed * Time.deltaTime);
//Move the object to XYZ coordinates defined as horizontalInput, 0, and verticalInput respectively.
```

Большинство API и структур проектов в Unity идентичны для всех поддерживаемых платформ, однако, существуют принципиальные различия в аппаратных средствах и способах развёртывания.

Ввод

Наиболее очевидным примером различного поведения между платформами является различие в методах ввода, предлагаемых аппаратными средствами.

Клавиатура и джойстик

Функция **Input.GetAxis** очень удобна в применении на настольных платформах, потому что объединяет способы ввода с клавиатуры и джойстика. Тем не менее, эта функция не имеет смысла для мобильных платформ, которые рассчитаны на сенсорный ввод. Более того, стандартный ввод с настольной клавиатуры не переносится на мобильные устройства, потому как предназначен только для набора текста.

https://docs.unity3d.com/ru/current/Manual/CrossPlatformConsiderations.html

Касания и клики

Функции **Input.GetMouseButtonXXX** спроектированы таким образом, что имеют достаточно очевидную интерпретацию на мобильных устройствах, хотя нет никакой “мыши” как таковой. Одиночное касание экрана сообщает о ЛКМ и свойство **Input.mousePosition** передаёт позицию нажатия на экран до тех пор, пока палец касается экрана. Это означает, что игры с простым управлением мышью зачастую могут свободно работать как на настольных , так и на мобильных платформах.

Акселерометр, компас, гироскоп и GPS

Эти устройства ввода появились благодаря мобильности портативных устройств, потому как у настольных компьютеров нет эквивалентных устройств. Например, рулевое управление в гонке может быть реализовано с помощью наклона мобильного устройства (определяется с помощью акселерометра). В подобных случаях, вызовы API ввода, как правило, довольно легко заменить, поэтому ввод с акселерометра может быть заменен нажатием клавиш.

https://docs.unity3d.com/ru/current/Manual/CrossPlatformConsiderations.html

Input

class in UnityEngine

Описание

Интерфейс системы ввода (Input system).

Этот класс используется для чтения информации с осей координат, установленных в Менеджере Ввода, а также для доступа к данным акселерометра/мультитач на мобильных устройствах.

Статические переменные

acceleration	Последнее измеренное линейное ускорение устройства в трехмерном пространстве. (Read Only)
accelerationEventCount	Количество измерений ускорения, произведенных за предыдущий кадр.
accelerationEvents	Возвращает список измерений ускорения, произведенных за предыдущий кадр. (Read Only) (Выделяется память под временные переменные, т.е. в куче создаются лишние объекты).
compass	Свойство для организации доступа к компасу (только для мобильных устройств). (Read
gyro	Возвращает гироскоп по умолчанию.
multiTouchEnabled	Property indicating whether the system handles multiple touches.
stylusTouchSupported	Returns true when Stylus Touch is supported by a device or platform.
touchCount	Количество касаний. Гарантируется, что количество касаний не изменится на протяжении кадра. (Read Only)
touches	Возвращает список объектов, отражающих состояние всех касаний за последний кадр. (Read Only) (Выделяет память под временные переменные, т.е. в куче создаются лишние объекты).
touchPressureSupported	Bool value which let's users check if touch pressure is supported.
touchSupported	Returns whether the device on which application is currently running supports touch input.

Переменные

altitudeAngle	Value of 0 radians indicates that the stylus is parallel to the surface, pi/2 indicates that it is
azimuthAngle	Value of 0 radians indicates that the stylus is pointed along the x-axis of the device.
deltaPosition	The position delta since last change.
deltaTime	Amount of time that has passed since the last recorded change in Touch values.
fingerId	The unique index for the touch.
maximumPossiblePressure	The maximum possible pressure value for a platform. If Input.touchPressureSupported re always be 1.0f.
phase	Describes the phase of the touch.
position	The position of the touch in pixel coordinates.
pressure	The current amount of pressure being applied to a touch. 1.0f is considered to be the pres Input.touchPressureSupported returns false, the value of this property will always be 1.0f.
radius	An estimated value of the radius of a touch. Add radiusVariance to get the maximum touc touch size.
radiusVariance	The amount that the radius varies by for a touch.
rawPosition	The raw position used for the touch.
tapCount	Number of taps.
type	A value that indicates whether a touch was of Direct, Indirect (or remote), or Stylus type.

Структура **Touch** описывает состояние пальца, касающегося экрана.

Поскольку на тачскрине можно зарегистрировать не один, а несколько касаний, то в памяти **сохраняется массив**, в котором записана информация о каждом касании.

Задача – необходимо зарегистрировать касание одним пальцем на экране. Информация о касании записывается в массив. Даже если касание было одно, то у нас всё равно есть массив хоть и состоящий из одного элемента.

Любые сенсорные экраны могут регистрировать разные данные: фаза касания (начало, движение, окончание), позицию на экране и количество быстрых касаний, а также информацию о поведении (траектория движения).

Алгоритм состоит из следующий частей:

1. Зарегистрировать хотя бы одно касание
2. Сохранить интересующий нас элемент массива
3. Получить фазу касания экрана и что-то сделать.

```
if (Input.touchCount > 0)
{
        Touch myTouch = Input.GetTouch(0);
        Vector2 positionOnScreen = myTouch.position;
        Debug.Log(positionOnScreen);
}
```

Тут просто выводится в консоль значение позиции на экране. Позиция на экране задается через **Vector2**. Однако в консоль будет выводится несколько сообщений, так как одно касание может длиться несколько кадров (мы пишем это в **Update**). Если нужна только точка в момент касания, то обращаемся к фазам касания. Код будет выглядеть так:

```
if (Input.touchCount > 0)
{
        Touch myTouch = Input.GetTouch(0);
        if (myTouch.phase == TouchPhase.Began)
        {
                Vector2 positionOnScreen = myTouch.position;
                Debug.Log(positionOnScreen);
        }
}
```

Пример кода, который выводит информацию о нескольких нажатиях

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

public class DebugMultiTouch : MonoBehaviour
{
    List<string> touchInfos = new List<string>();

    void Update()
    {
        touchInfos.Clear();

        for (int i = 0; i < Input.touchCount; i++)
        {
            Touch touch = Input.GetTouch(i);
            string tmp = "Touch #" + (i + 1) + " at " + touch.position.ToString() + ", r=" + touch.radius;
            touchInfos.Add(tmp);
        }
    }
    void OnGUI()
    {
        foreach(string s in touchInfos)
        {
            GUILayout.Label(s);
        }
    }
```

Как создать игру, управлять которой можно и в самом редакторе и на телефоне?

Существуют способы определить с какого девайса идет управление и в зависимости от этого решать как будет идти обработка управления. Такая вещь называется платформенно зависимая компиляция. Она представляет собой **директивы препроцессора**. Задача этих директив – указать, какие участки кода должны выполняться в зависимости от того, с какой платформы работает приложение.

Unity предоставляет инструменты как для определения платформ так и для определения версий с которых запускается код.

Часто используются следующие директивы:

UNITY_STANDALONE – добавляется для части скриптов, которые выполняются при запуске игры из системы Windows или Linux, или Mac OS X.

UNITY_IOS – будет исполняться код для iOS платформы

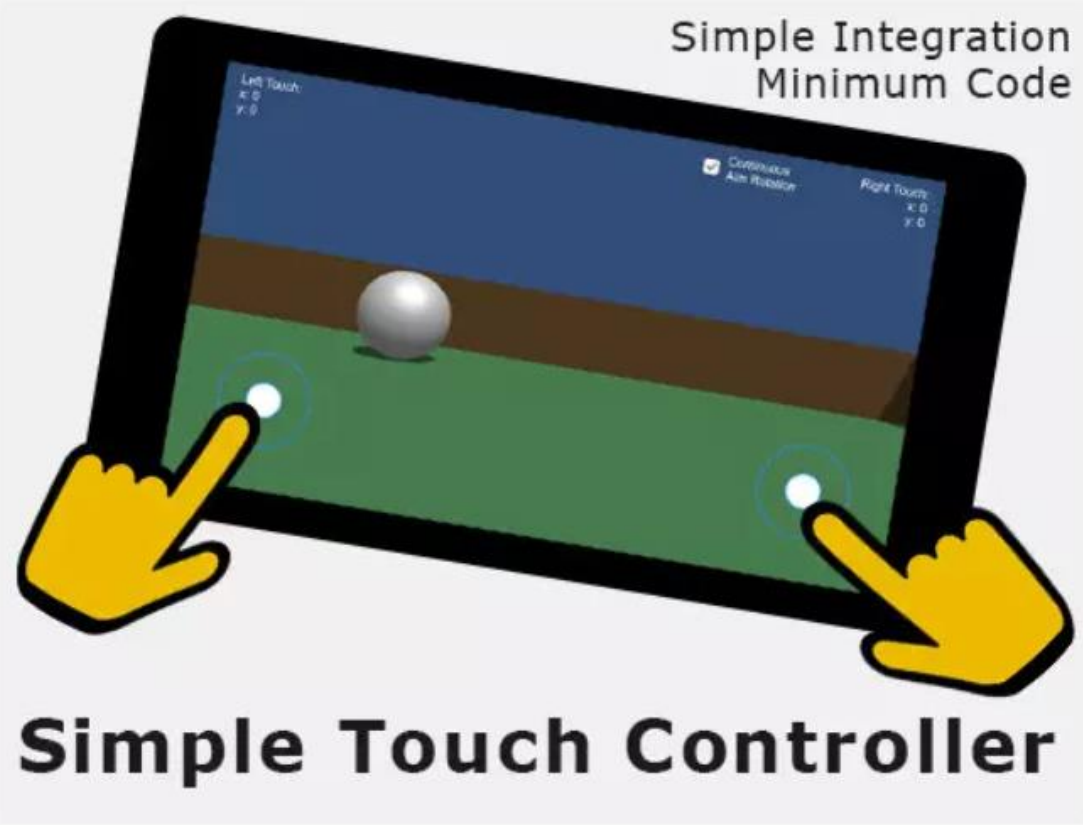
UNITY_ANDROID – директива для платформы Android

https://unity3dschool.ru/upravlenie-v-android-igrah-na-unity.html

Пример кода с директивами.

```
//Проверяем, что код запускается с самостоятельных ОС
#if UNITY_STANDALONE

        //код, который будет выполняться с компьютера. Тут работают Input.GetAxis =)

 #elif UNITY_IOS || UNITY_ANDROID

        //блок который выполняется по условию “иначе”. Хорошей практикой будет учесть
все возможные варианты. Тут учтены как Android так и iOS устройства.

#endif //обязательно указывать по окончанию всех директив
```

Можно скачать Touch Controller с AssetStore



Simple Touch Controller

Daniel4D ★★★★☆ (81) | ♥ (942)

FREE

73 views in the past week

Add to My Assets

GrigoriyChuwak
★★★★★ 21 days ago

its work with unity 5.6.4

I used it for setting direction of wind.
after user stop moving and pull out finger from screen - framework does not fire event with coordinates (0,0...

Read more reviews

Новая система ввода

Старая система ввода обладает рядом проблем. Её трудно расширять, не всегда получается настроить её так, чтобы она удовлетворяла вашей задумке,а также были и технические проблемы, такие как определение подключения геймпада к игре уже после её запуска.

Новая система ввода объединила ввод всех платформ в одном простейшем интерфейсе, который может быть с легкостью расширен для поддержки пользовательских или будущих устройств.
А эти **действия в свою очередь собраны в картах действий**, которые позволяют максимально удобно настроить ряд действий для определенного управления персонажем

Действие — Стрельба

Условие — Нажать кнопку пробел

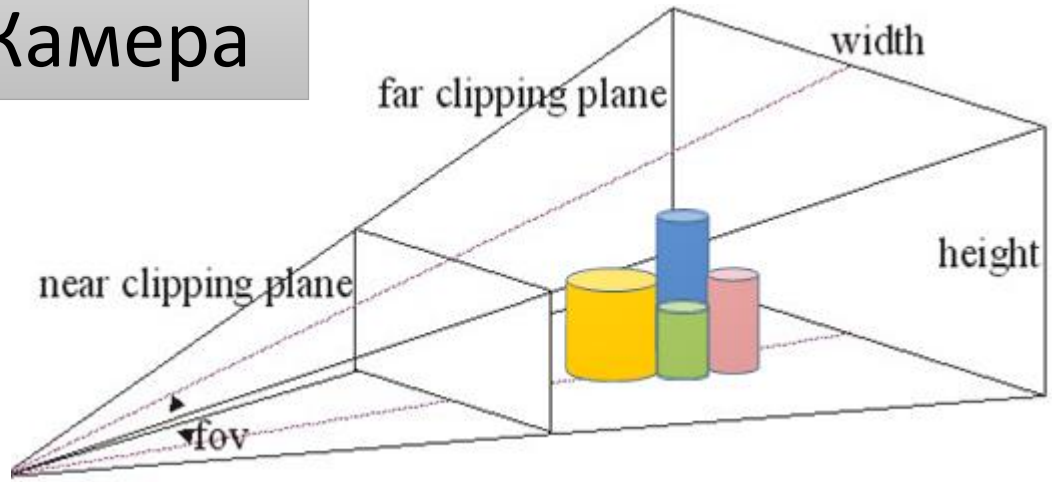
Это простейший пример действия, но действия бывают и сложнее

Действие — Перетащить дерево

Условие — вы должны нажать на экран и удерживать палец в одной точке минимум 1 секунду.

Новая система ввода устанавливается через Package Manager и при установке можно сразу заменить старую систему ввода

Камера



Inspector | Navigation

Main Camera — Static

Tag: MainCamera — Layer: Default

Transform

	X	Y	Z
Position	17.1641	6.6755(	-11.665
Rotation	29.496	-56.723	0
Scale	1	1	1

Camera

Clear Flags	Skybox
Background	
Culling Mask	Mixed...
Projection	Perspective
FOV Axis	Vertical
Field of View	60
Physical Camera	
Clipping Planes	Near 0.3
	Far 1000

Viewport Rect

X	0	Y	0
W	1	H	1

Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
HDR	Use Graphics Settings
MSAA	Use Graphics Settings
Allow Dynamic Reso	
Target Display	Display 1

Audio Listener

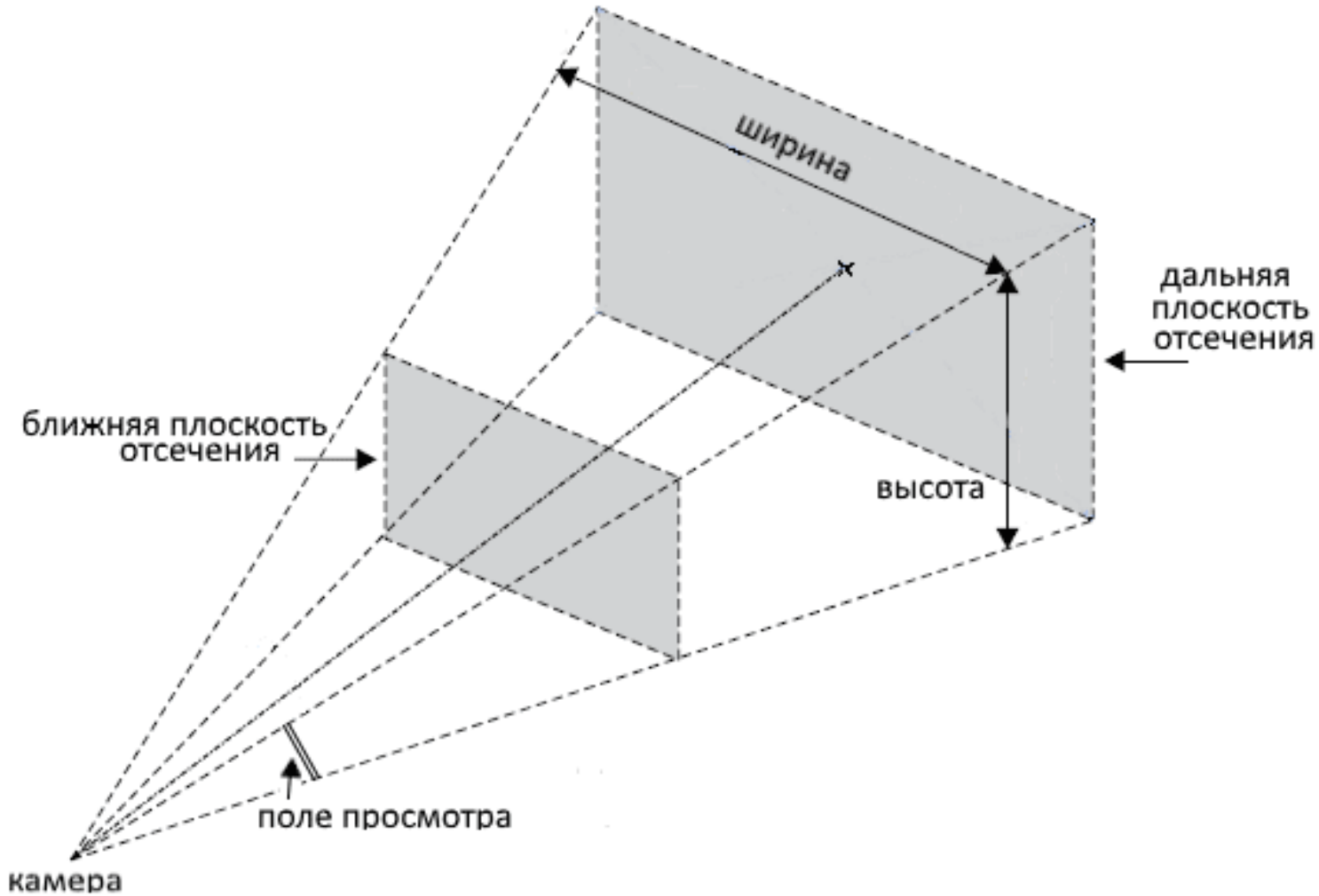
Переключение камер

По умолчанию камера отрисовывает свой вид, чтобы покрыть весь экран, поэтому одновременно может быть виден только один вид камеры (**видимая камера — это та, у которой самое высокое значение свойства глубины**).

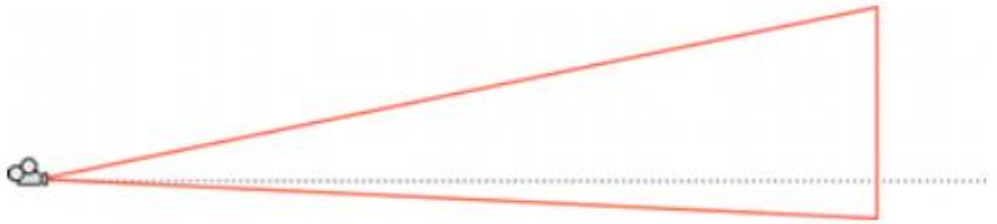
Отключив одну камеру и включив другую из сценария, вы можете переключить одну камеру на другую, чтобы получить разные виды сцены.

```
using UnityEngine;

public class ExampleScript : MonoBehaviour {
    public Camera firstPersonCamera;
    public Camera overheadCamera;

    // Call this function to disable FPS camera,
    // and enable overhead camera.
    public void ShowOverheadView() {
        firstPersonCamera.enabled = false;
        overheadCamera.enabled = true;
    }

    // Call this function to enable FPS camera,
    // and disable overhead camera.
    public void ShowFirstPersonView() {
        firstPersonCamera.enabled = true;
        overheadCamera.enabled = false;
    }
}
```

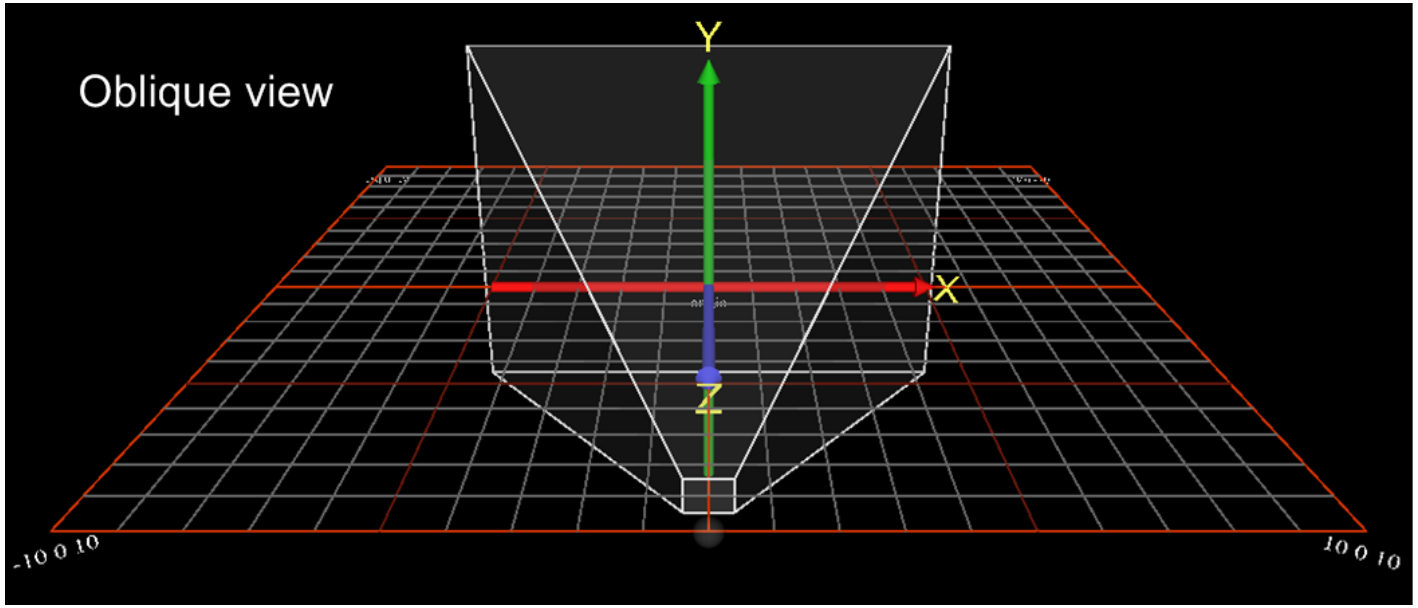
Область видимости камеры



По умолчанию **область отсечения** *(oblique frustum)* располагается симметрично относительно центральной линии камеры, но это не обязательно. Вы можете сделать усеченный конус наклонным, что означает, что одна сторона находится под меньшим углом к центральной линии, чем противоположная сторона.

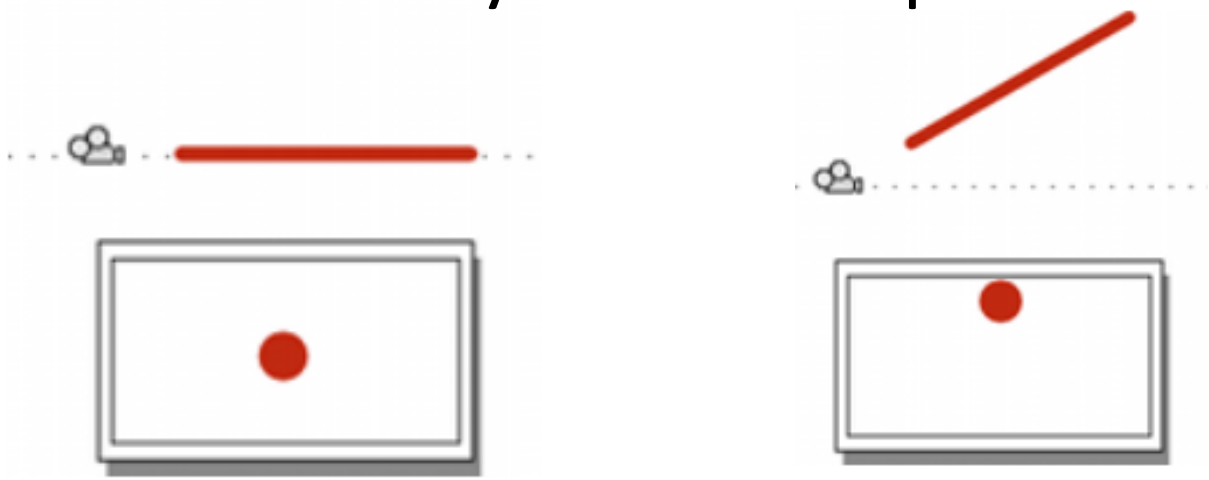


Это делает перспективу на одной стороне изображения более сжатой, создавая **впечатление, что зритель находится очень близко к объекту**, видимому на этом краю. Примером того, как это можно использовать, является игра в автомобильные гонки; если **усеченный нижний край уплощен, зрителю кажется, что они ближе к дороге**, что усиливает ощущение скорости.



Управление из скрипта пример: https://docs.unity3d.com/ru/2020.2/Manual/ObliqueFrustum.html

Лучи из камеры



Любая точка в поле зрения камеры соответствует линии в мировом пространстве. Математически можно представить эту линию в виде объекта Ray (луч). Ray всегда соответствует точке в поле зрения, потому класс Camera предоставляет методы **ScreenPointToRay** и **ViewportPointToRay**.

Различие между ними в том, что ScreenPointToRay ожидает точку в виде пиксельных координат, в то время как ViewportPointToRay получает нормализованные координаты в диапазоне от 0 до 1 (где 0 представляет нижнюю или левую, а 1 - верхнюю или правую часть поля зрения).

Каждая из этих функций возвращает Ray, который состоит из точки испускания (начала) и вектора, показывающего направление линии из этой точки. **Ray берёт начало из ближней плоскости отсечения** вместо точки transform.position камеры.

Рейкастинг

Луч из камеры чаще всего используют для совершения рейкаста в сцену. Рейкаст отправляет воображаемый “лазерный луч” вдоль программного луча из точки испускания до тех пор, пока он не встретит на пути коллайдер в сцене. В результате столкновения с коллайдером, возвращается RaycastHit с координатами и объектом столкновения. Это очень полезный способ нахождения объекта по его изображению на экране. Например, с помощью следующего кода можно определить объект под указателем мышки:

```
using UnityEngine;
using System.Collections;

public class ExampleScript : MonoBehaviour {
    public Camera camera;

    void Start(){
        RaycastHit hit;
        Ray ray = camera.ScreenPointToRay(Input.mousePosition);

        if (Physics.Raycast(ray, out hit)) {
            Transform objectHit = hit.transform;

            // Do something with the object that was hit by the raycast.
        }
    }
}
```

Перемещение камеры вдоль луча

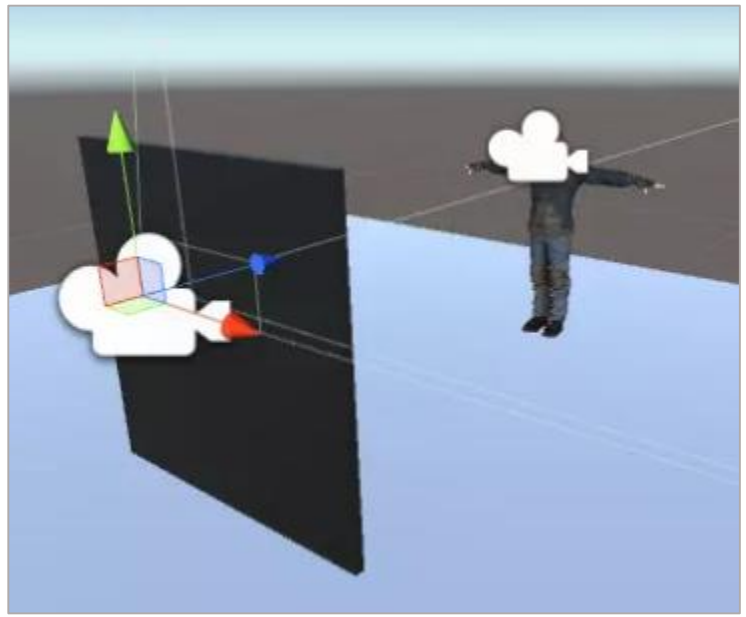
Иногда полезно получить луч, соответствующий точке на экране и затем переместить камеру вдоль этого луча. Например, вы могли бы позволить пользователю выбрать объект мышкой и затем приблизить его, при этом сохраняя его “приколотым” к одному и тому же положению на экране под курсором мышки (это может быть полезно, когда камера смотрит на тактическую карту, например). Код, позволяющий это сделать, довольно прост:

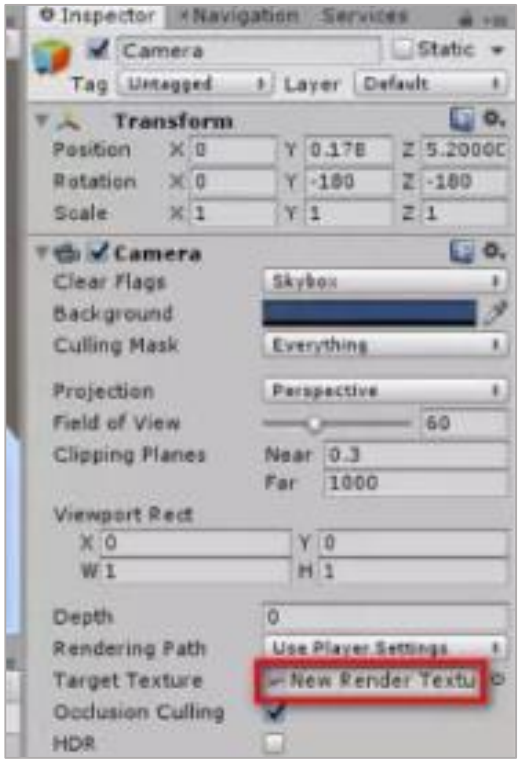
```
using UnityEngine;
using System.Collections;

public class ExampleScript : MonoBehaviour {
    public bool zooming;
    public float zoomSpeed;
    public Camera camera;

    void Update() {
        if (zooming) {
            Ray ray = camera.ScreenPointToRay(Input.mousePosition);
            float zoomDistance = zoomSpeed * Input.GetAxis("Vertical")
* Time.deltaTime;
            camera.transform.Translate(ray.direction * zoomDistance,
Space.World);
        }
    }
}
```

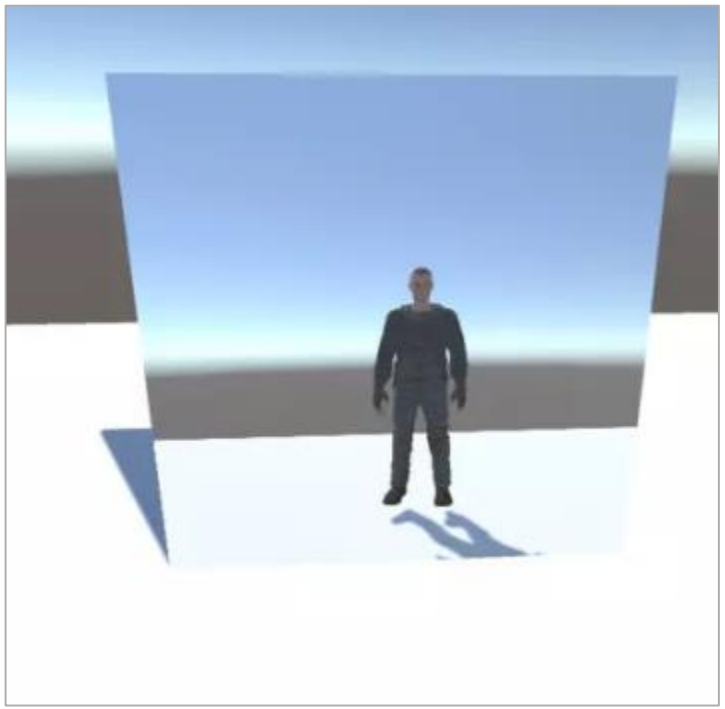
https://docs.unity3d.com/ru/2020.2/Manual/CameraRays.html

Зеркало, миникарта и прочее….

- Создадим объект зеркало (например, просто куб)
- Создадим дополнительную текстуру
- Текстуру поместим на зеркало
- Установим дополнительную камеру, смотрящую из зеркала
- В настройках камеры **в поле Target Texture указываем нашу текстуру**



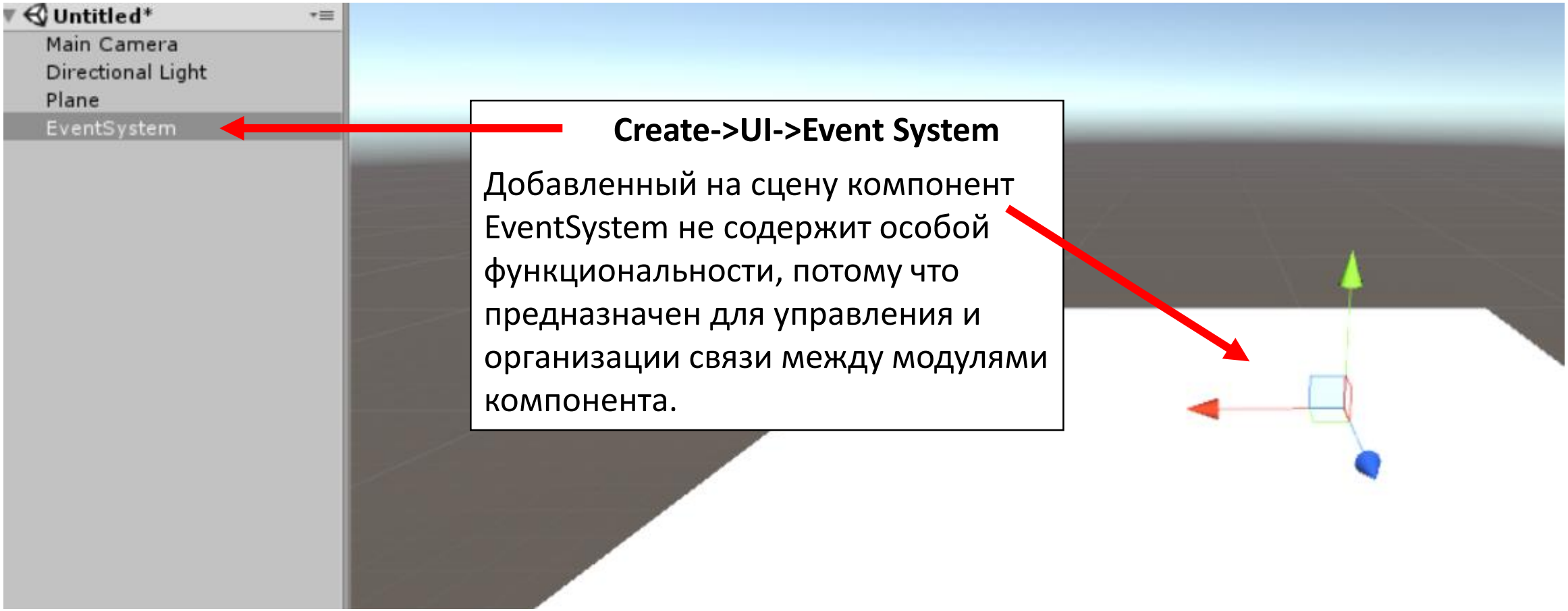




Демо

EventSystem

Система событий - способ отправки событий к объектам в приложении, основанный на вводе с клавиатуры или мыши; с помощью касаний или персональных устройств. Сцена должна содержать только одну систему событий.



Явно указываем что данный сценарий использует систему событий

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

public class Clicker : MonoBehaviour, IPointerClickHandler {

    public void OnPointerClick(PointerEventData eventData){
```

События, которые поддерживаются автономным модулем ввода и сенсорным модулем ввода, предоставляются интерфейсом и могут быть реализованы в MonoBehaviour путем реализации интерфейса.

- IPointerEnterHandler - OnPointerEnter - вызывается, когда указатель входит в объект
- IPointerDownHandler - OnPointerDown - вызывается, когда указатель нажимается на объект
- IPointerClickHandler - OnPointerClick - Вызывается, когда указатель нажимается и отпускается на том же объекте
- IBeginDragHandler - OnBeginDrag - вызов объекта перетаскивания при начале перетаскивания
- IDragHandler - OnDrag - вызов объекта перетаскивания при перетаскивании
- IScrollHandler - OnScroll - вызывается при прокрутке колесика мыши
- IDeselectHandler - OnDeselect - вызывается выбранный объект
- IMoveHandler - OnMove - вызывается, когда происходит событие перемещения (влево, вправо, вверх, вниз, ect)
- . . .

Реакция на щелчок

Реализуем один из интерфейсов.
Интерфейс реализующий щелчок мышки это IPointerClickHandler
Реализуем метод OnPointerClick

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

public class Clicker : MonoBehaviour, IPointerClickHandler {

    public void OnPointerClick(PointerEventData eventData){
```

Реакция на щелчок

```
public void OnPointerClick(PointerEventData eventData){

    float red = Random.Range (.0f, 1.0f);
    float green = Random.Range (.0f, 1.0f);
    float blue = Random.Range (.0f, 1.0f);

    Color color = new Color (red, green, blue);
    gameObject.GetComponent<Renderer> ().material.color = color;
```

Любое взаимодействие с объектами в игровом мире происходит через камеру.

Изначально камера не умеет пропускать щелчки мышью и передавать их системе событий это связано с тем, что камере приходится проецировать щелчки мыши на двумерном экране на трехмерную игровую сцену.

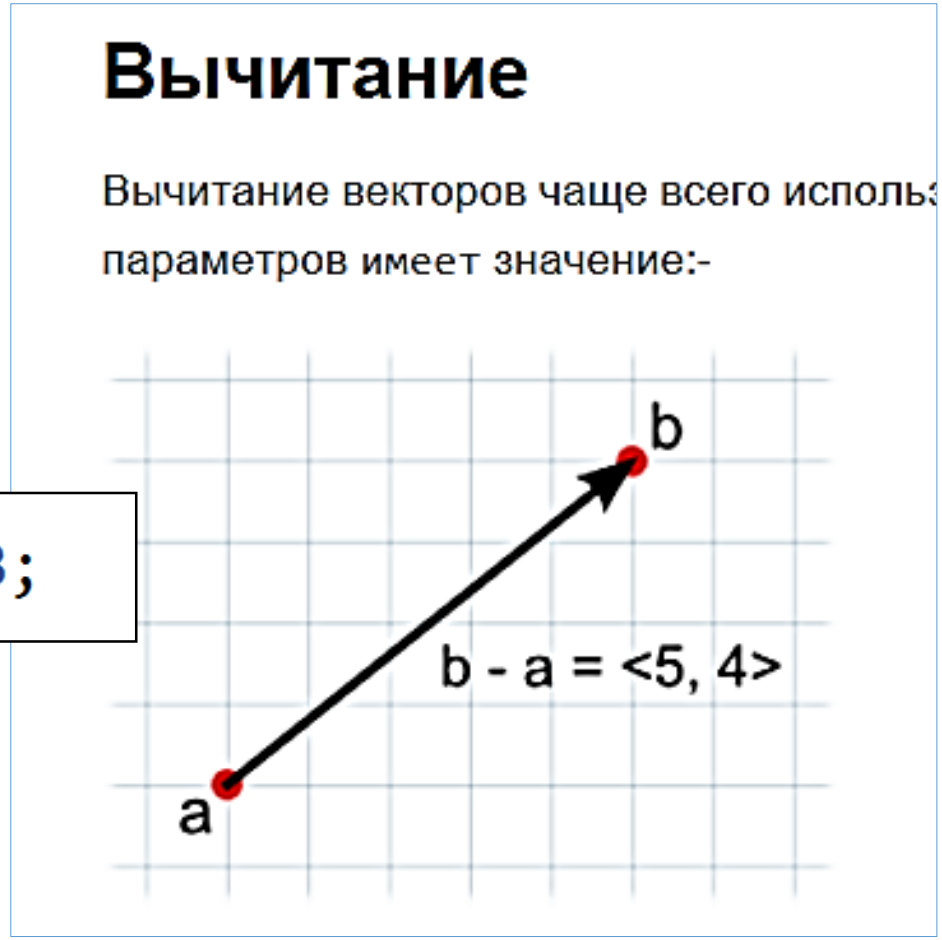
Эта возможность по умолчанию отключена (поэтому данный скрипт и не работает), чтобы ее включить **нужно назначить камере компонет Physics Raycaster.**

2. Для толчка применяем **AddForceAtPosition**

Что бы определить направление, в котором будем толкать куб, вспомним векторную алгебру.

Направление от B к A равно разности точек B и A.

```
Vector3 direction = pointA - pointB;
```



```
//учитываем направление

Vector3 pointA=eventData.pointerPressRaycast.worldPosition;
Vector3 pointB = Camera.main.transform.position;
Vector3 direction = pointA - pointB;
direction = direction.normalized;


Vector3 force = direction * 500;
Vector3 target = eventData.pointerPressRaycast.worldPosition;
gameObject.GetComponent<Rigidbody> ().AddForceAtPosition (force, target);
```

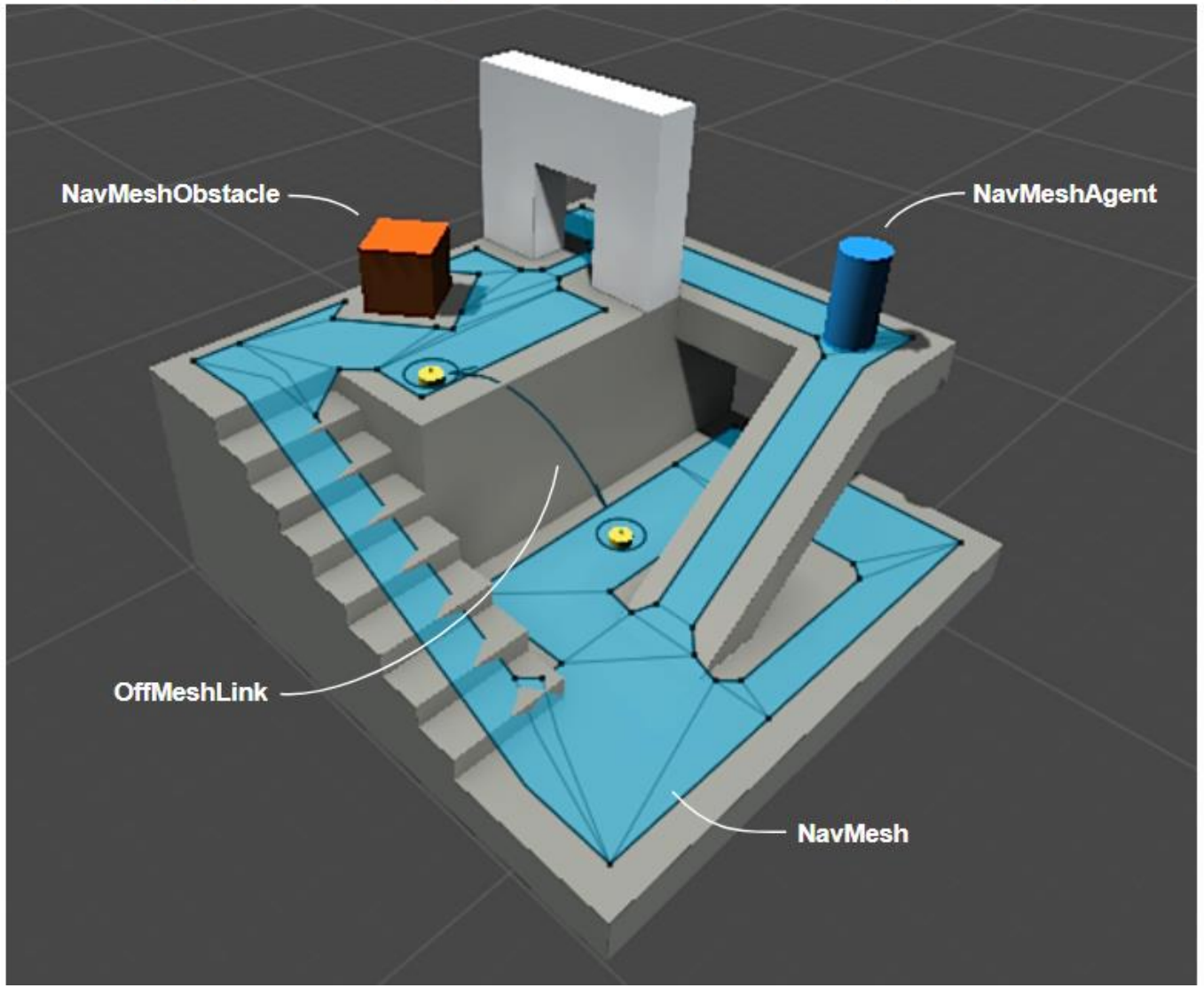
В нашем случае А-координаты точки на кубе, В-координаты камеры.

Для доступа к камере используем класс Camera и свойство main

```
//учитываем направление

Vector3 pointA=eventData.pointerPressRaycast.worldPosition;
Vector3 pointB = Camera.main.transform.position;
Vector3 direction = pointA - pointB;
direction = direction.normalized;


Vector3 force = direction * 500;
Vector3 target = eventData.pointerPressRaycast.worldPosition;
gameObject.GetComponent<Rigidbody> ().AddForceAtPosition (force, target);
```

Параметр **eventData** содержит дополнительную информацию о событии и из него можно получить мировые координаты точки на кубе по которому мы щелкаем (или касаниемся экрана).

Navigation System in Unity

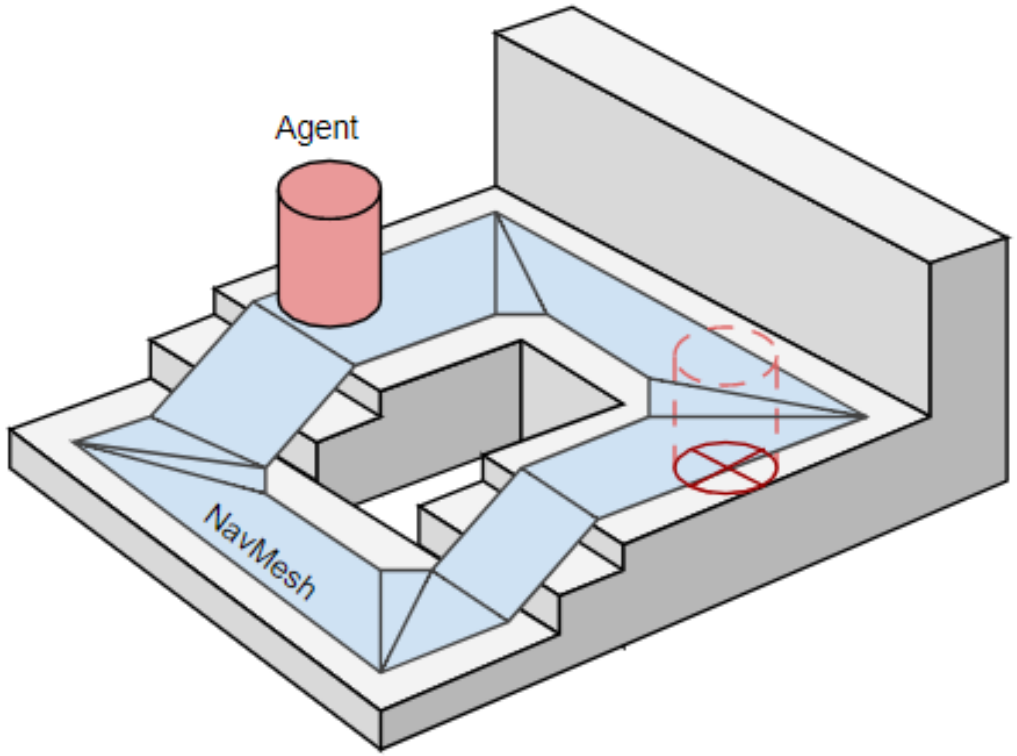


Walkable Areas

Проходимые области определяют места в сцене, где агент может стоять и двигаться. В Unity агенты описываются как цилиндры.

Проходимая область создается из геометрии сцены, проверяя места, где может стоять агент.

Эта поверхность называется навигационной сеткой (для краткости **NavMesh**).

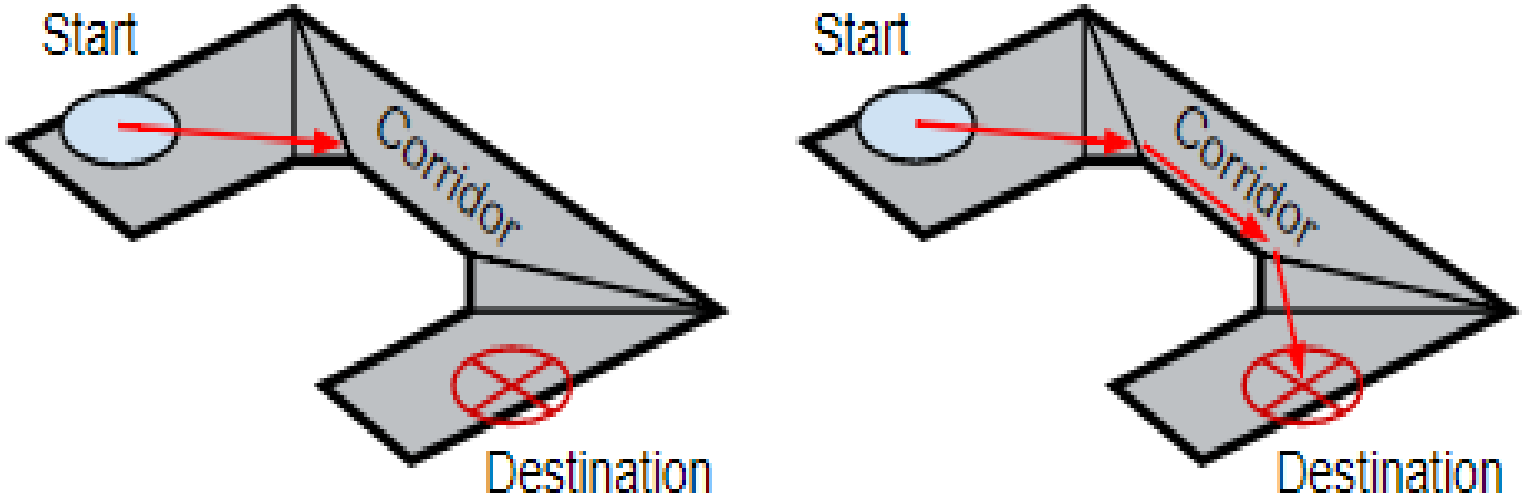


NavMesh сохраняет эту поверхность как выпуклые многоугольники. Выпуклые многоугольники являются полезным представлением, поскольку мы знаем, что между любыми двумя точками внутри многоугольника нет препятствий.

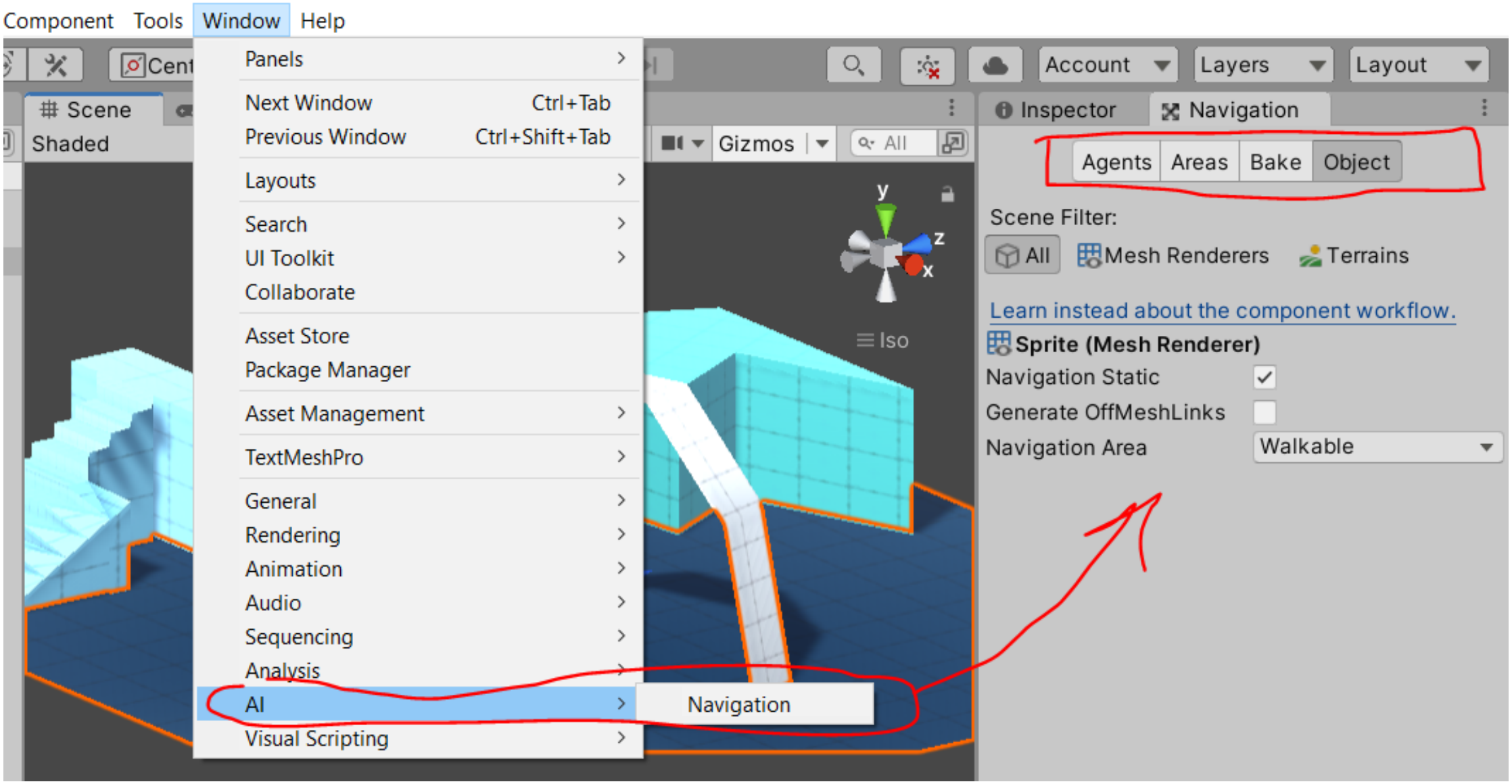
Поиск путей

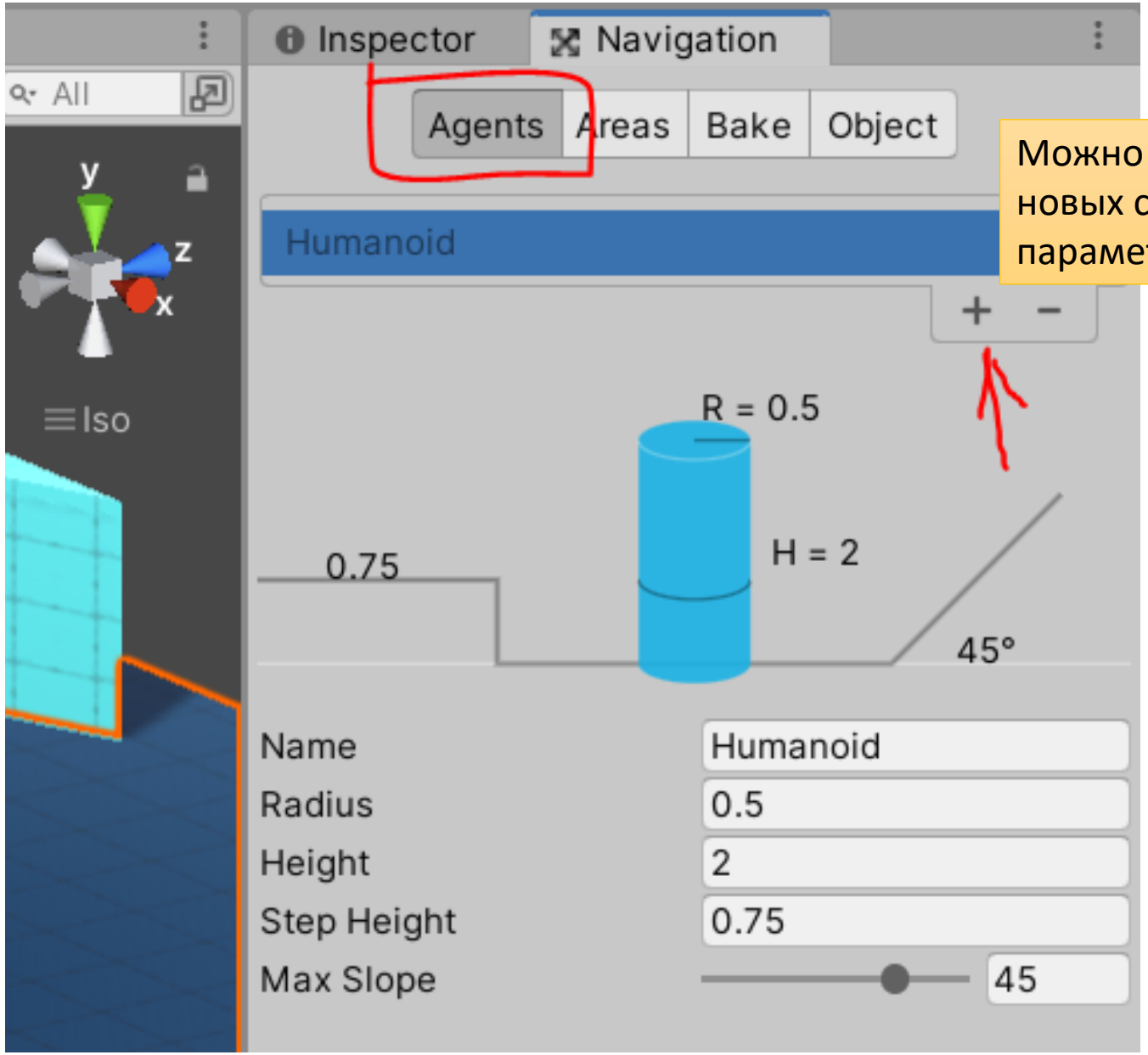
Последовательность многоугольников, которые описывают путь от начала до целевого многоугольника, называется **коридором**.

Агент достигнет пункта назначения, всегда направляясь **к следующему видимому углу коридора.**



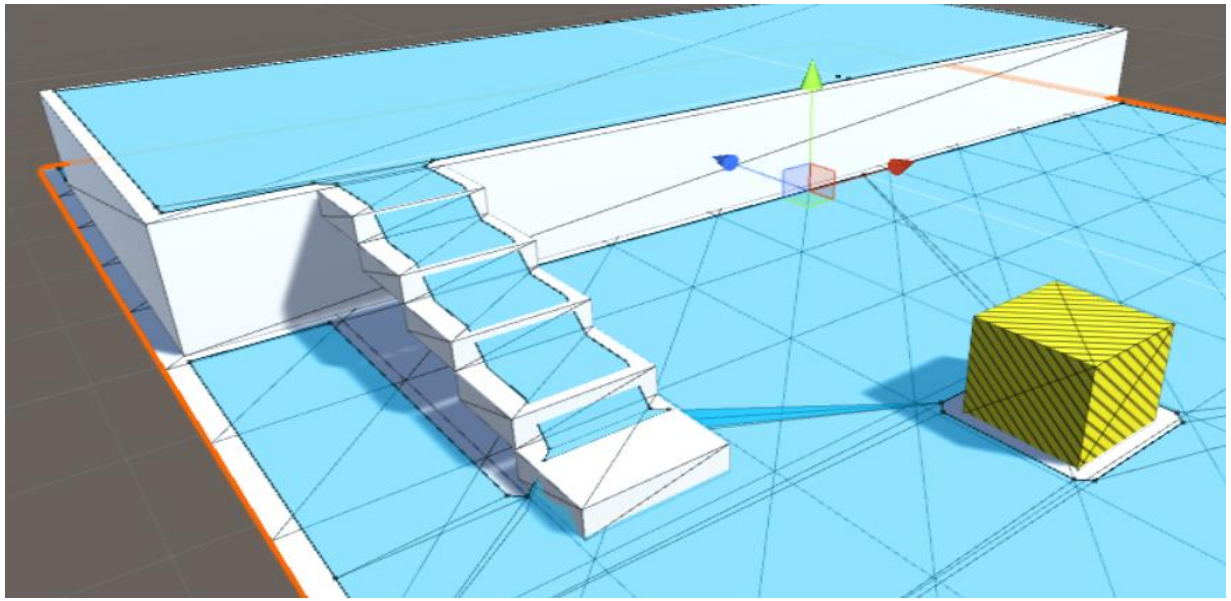
Создание NavMesh

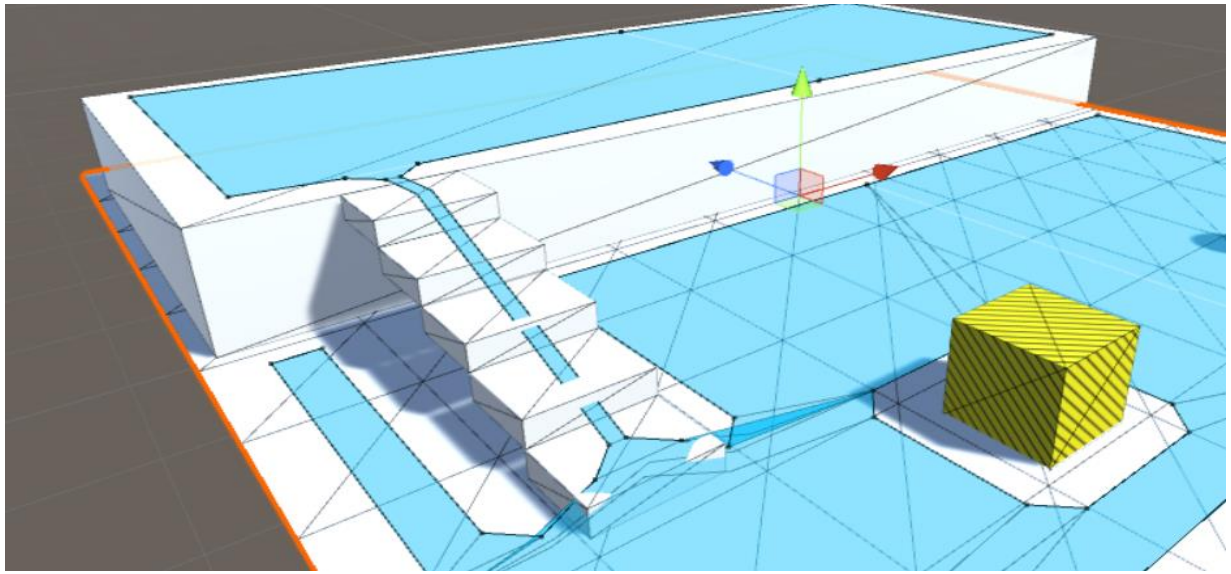


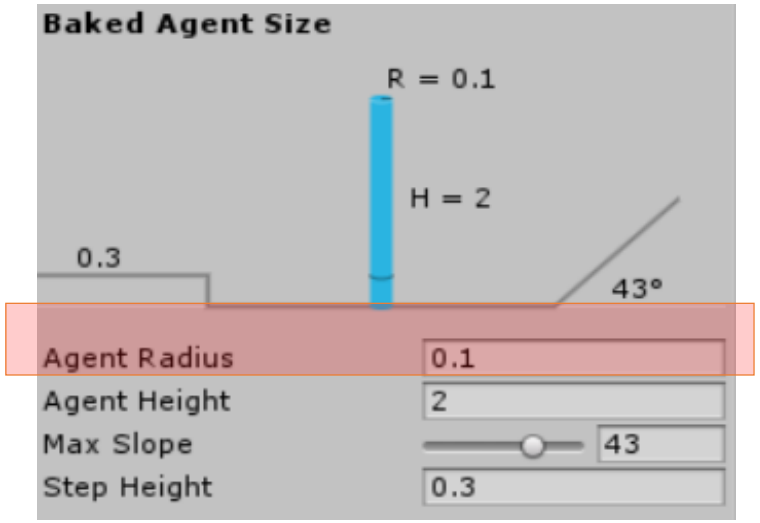


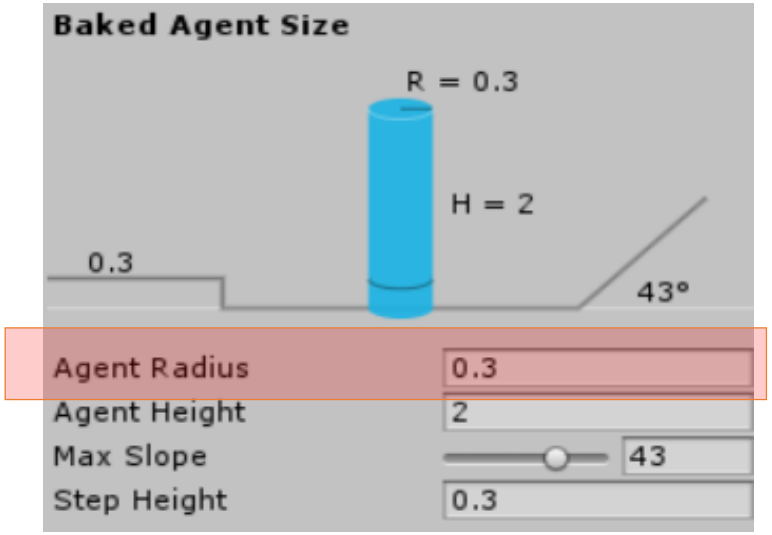
Radius определяет насколько близко агент может подойти к стене или уступу,
Height определяет низко могут находится объекты под которыми должен пройти агент,
Step Height определяет насколько высоки могут быть препятствия на которые агент может подняться
Max Slope определяет максимальный угол пандусов по которым агент способен взобраться

Пример как проходимость зависит от геометрии (радиуса) агента

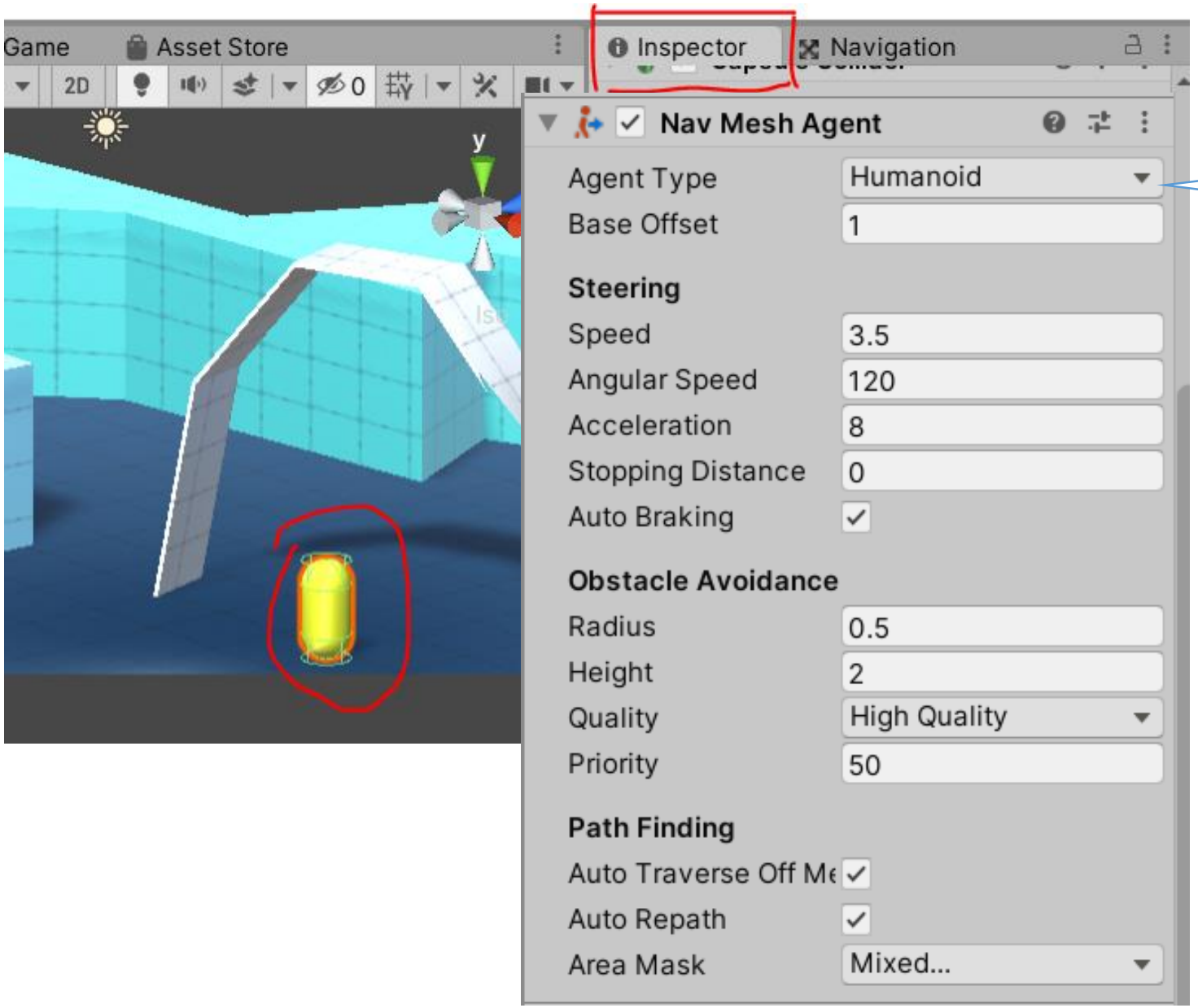








Герою добавляем компонент **Nav Mesh Agent**



Если создано несколько агентов, то выбираем тип агента из списка

Рулевое управление

- Скорость
- Угловая скорость
- Ускорение
- Дистанция до препятствия
- Остановится перед непроходим

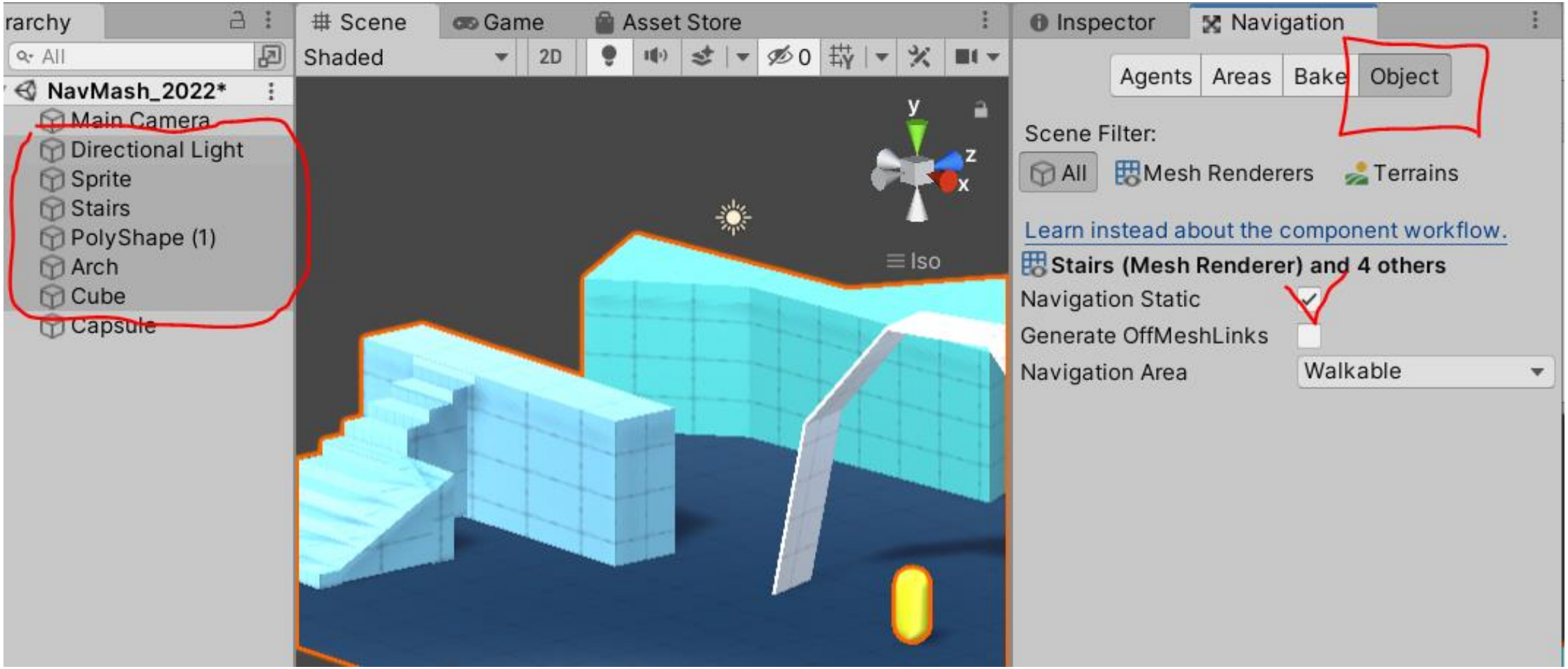
Обход препятствий

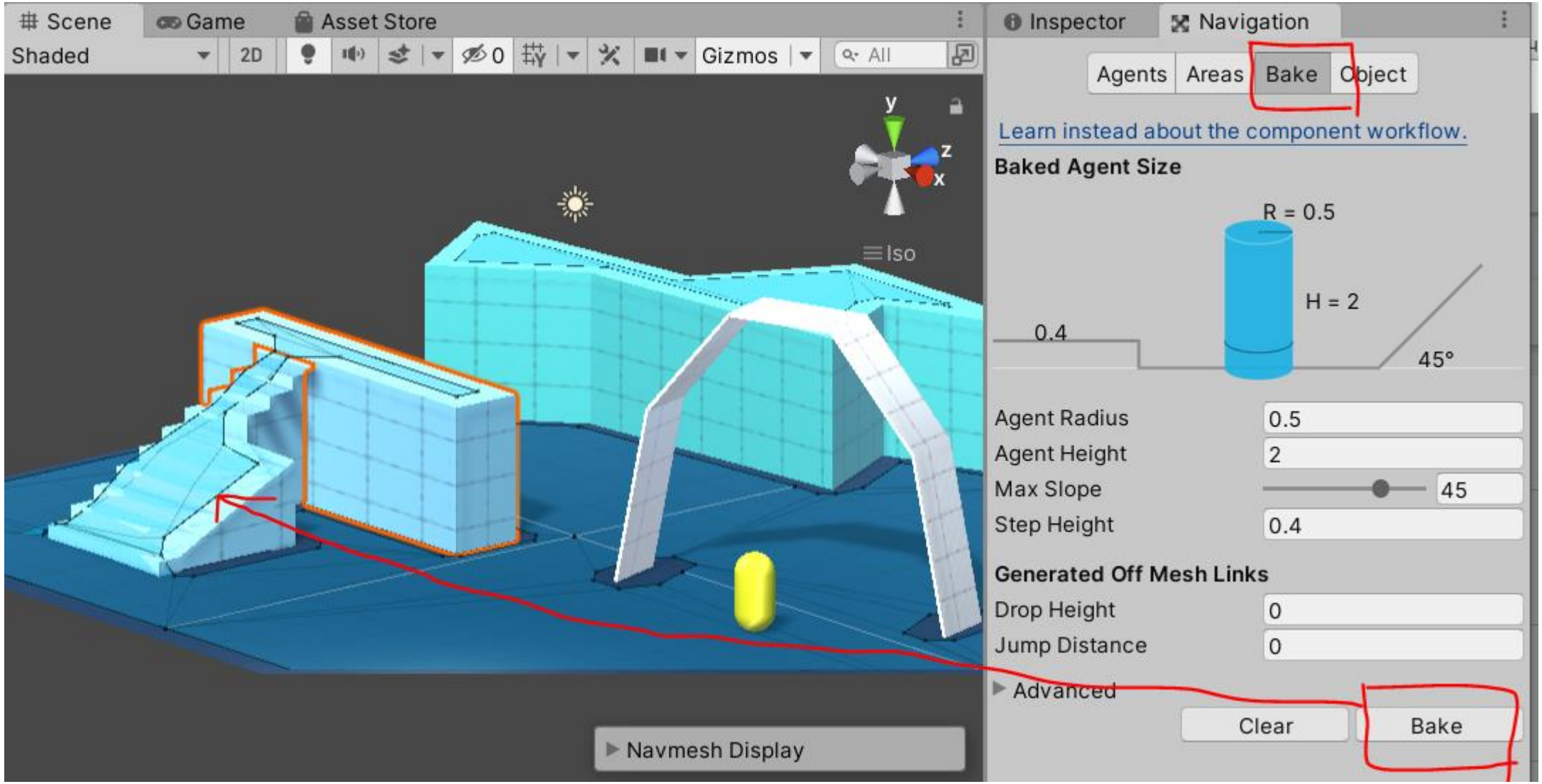
- Радиус
- Высота
- Смещение
- Качество обхода препятствий
- Агенты с более низким приоритетом будут игнорироваться

Поиск пути

- Автоматически перемещаться по сетке
- Автоматически перестраивать путь
- Маска области

Все статичные объекты (стены, лестницы, поверхности) необходимо пометить флажком Navigation Static. Тогда поверхности предназначенные для ходьбы будут образованы на основе их Mesh Renderers или Terrains.

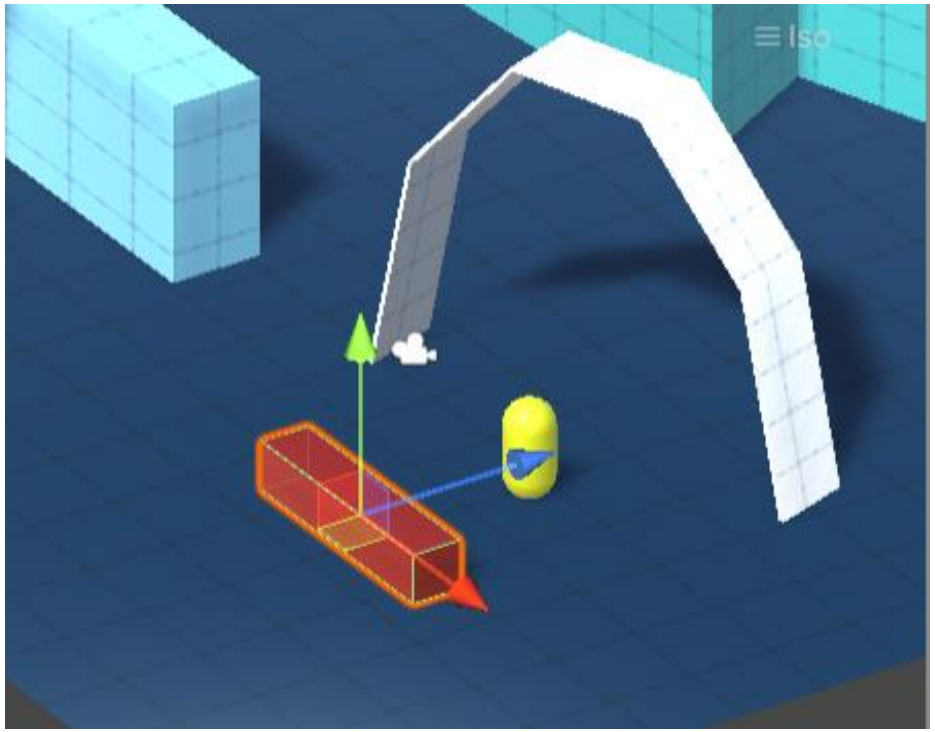


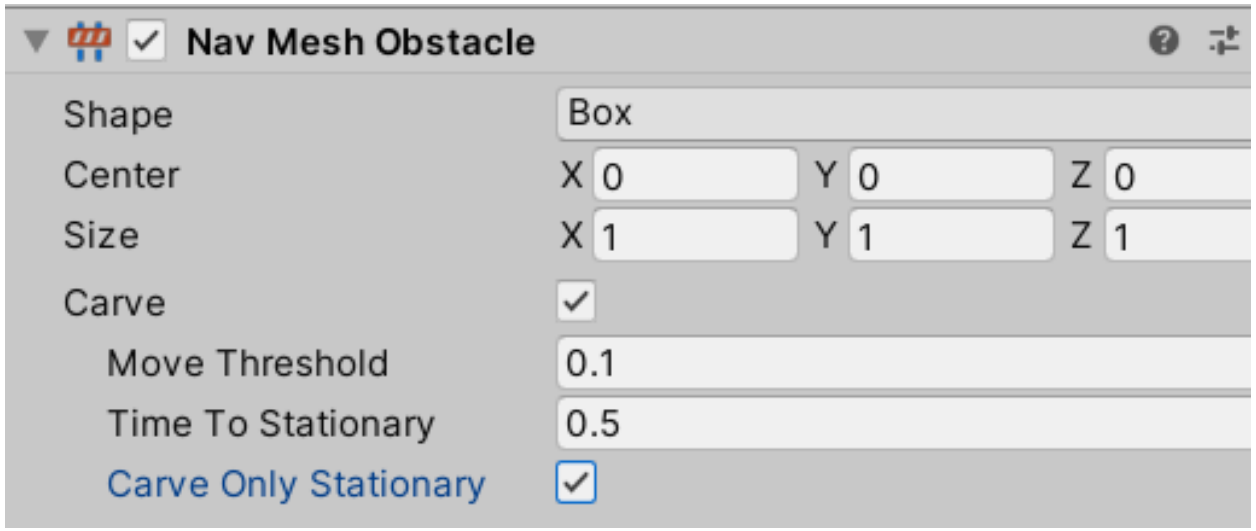


После запекания создастся файл NavMesh в папке, название которой будет соответствовать названию сцены, для которой было произведено запекание. Он будет автоматически использоваться агентами, находящимися на данной сцене.

Препятствиям добавляем компонент **Nav Mesh Obstacle**

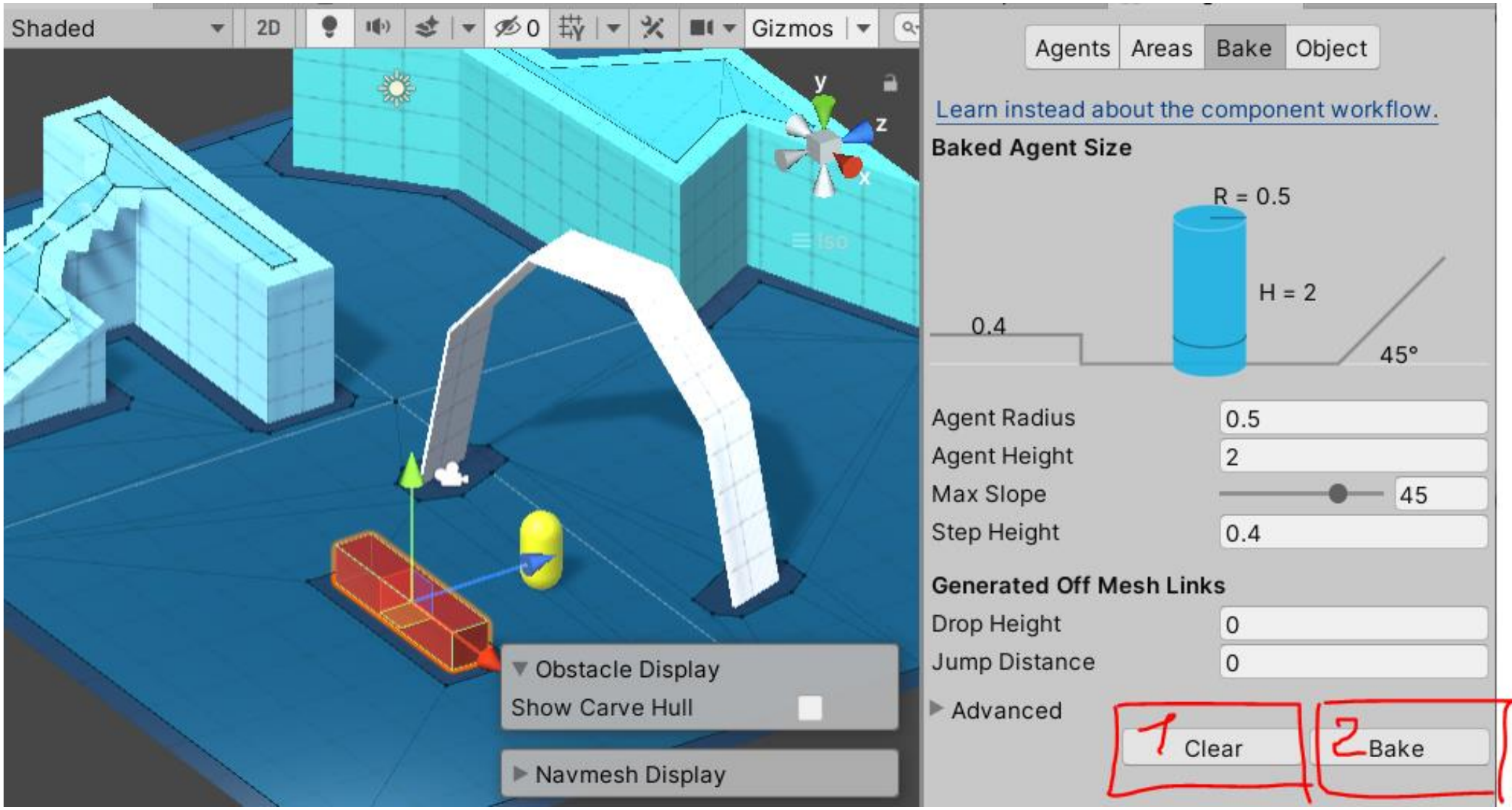
Препятствия могут двигаться, в отличие от статических объектов и на них не создается область для прохождения агента



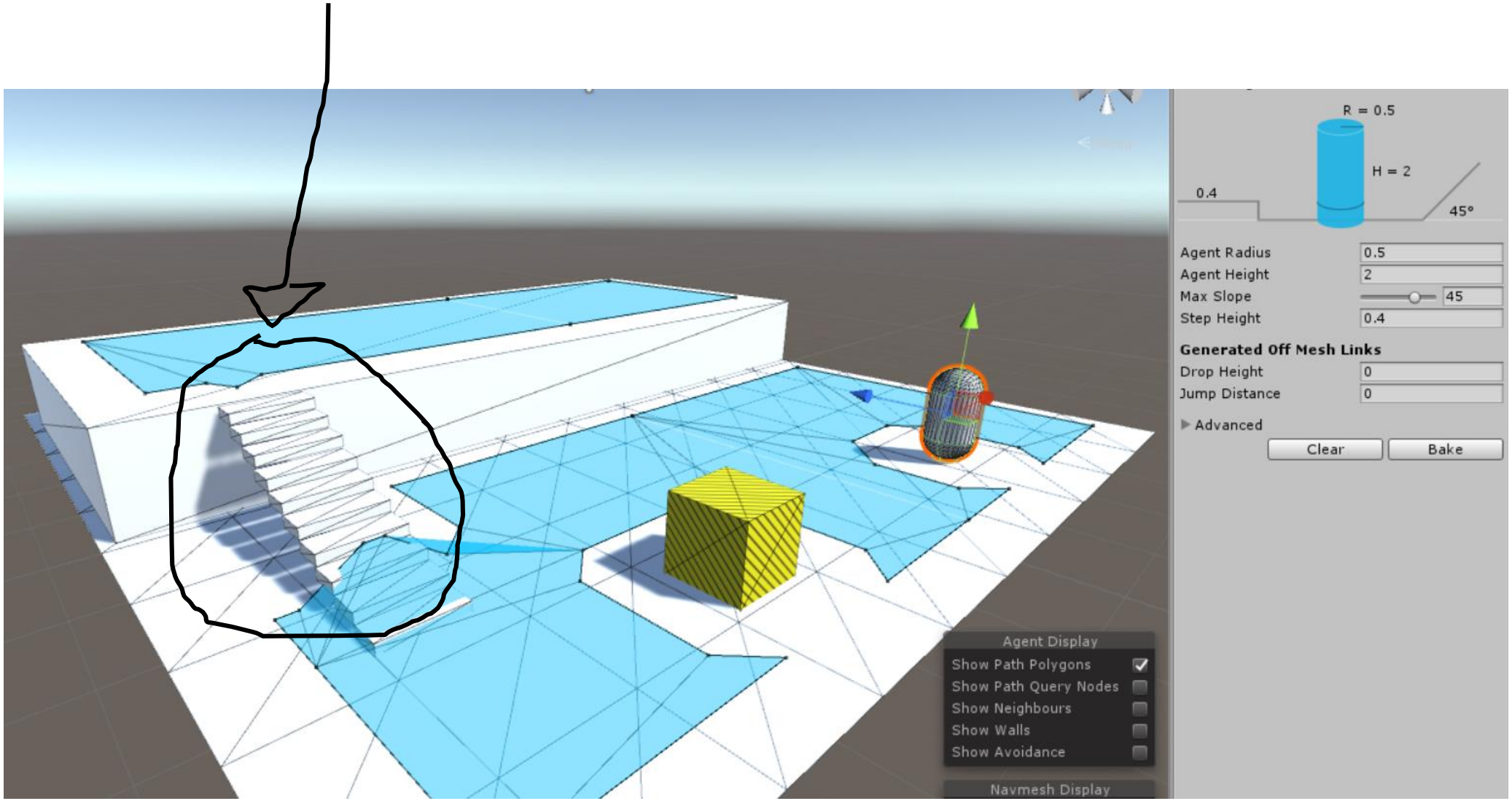


Carve
При включении препятствие прорежет дыру в NavMesh. Стоит включить для препятствий, таких как ящики и бочки, которые обычно блокируют навигацию, но могут быть перемещены игроком или другими игровыми событиями.
Carve Only Stationary
Когда эта функция включена, препятствие будет вырезано только тогда, когда оно неподвижно.

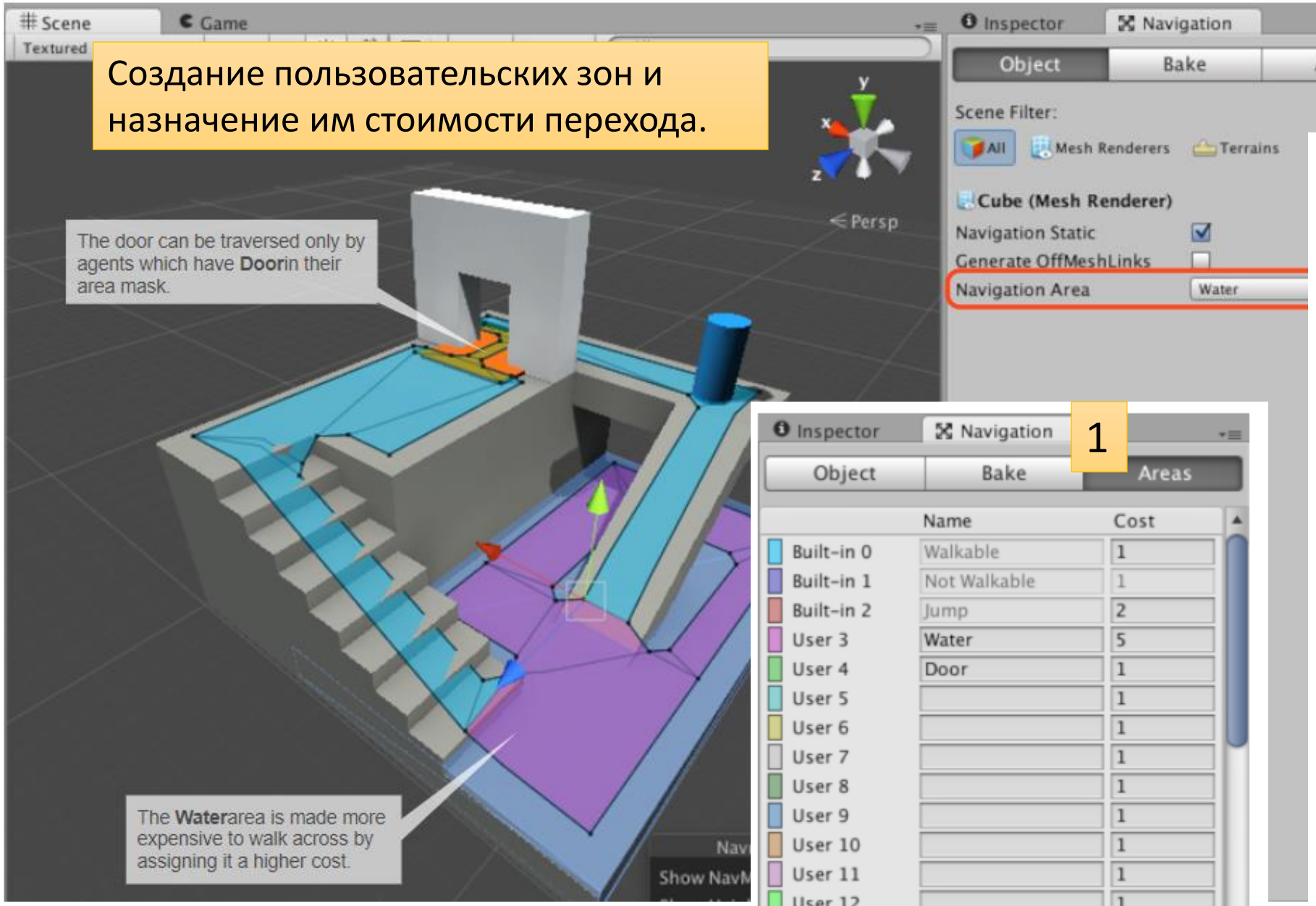
Если в сцену вносились изменения (в нашем случае добавилось препятствие), то запеченный NavMesh очищают (1) и запекают заново (2)

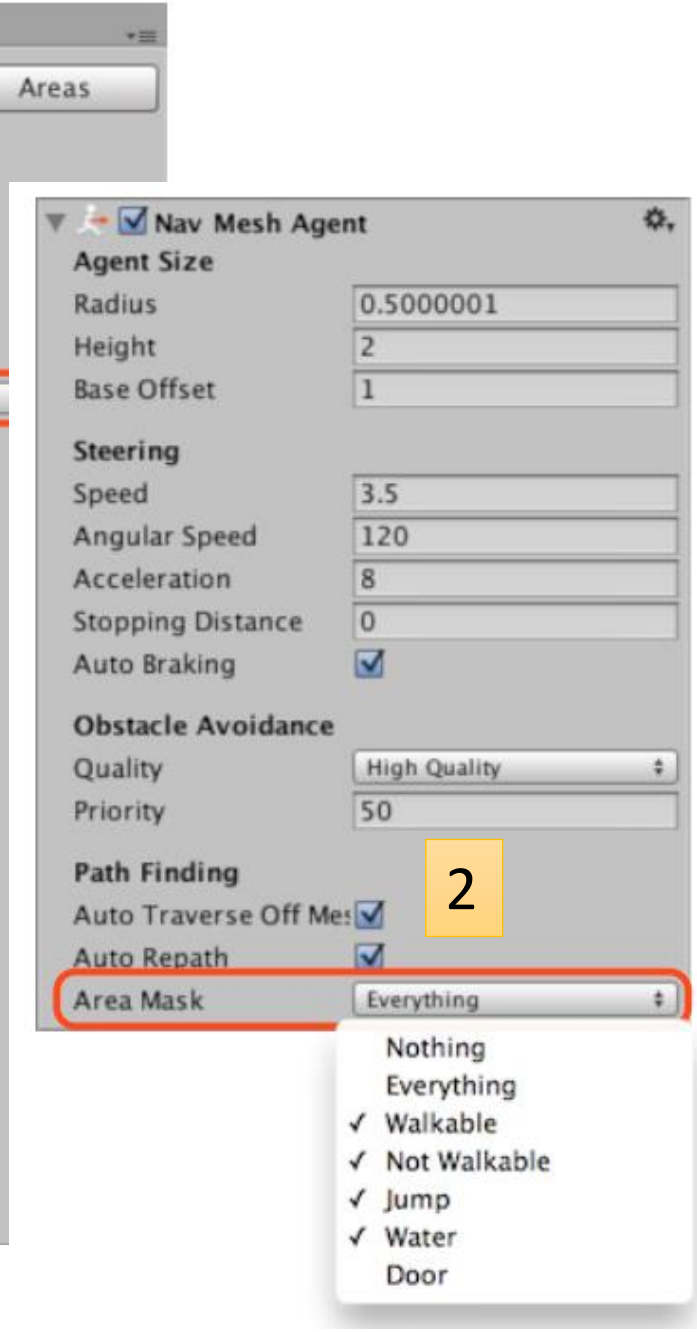


Если путь запекся не так, как нужно (есть непроходимые области, там где их быть не должно), нужно пересмотреть параметры агента или в данном случае увеличить ширину ступенек.



Создание пользовательских зон и назначение им стоимости перехода.

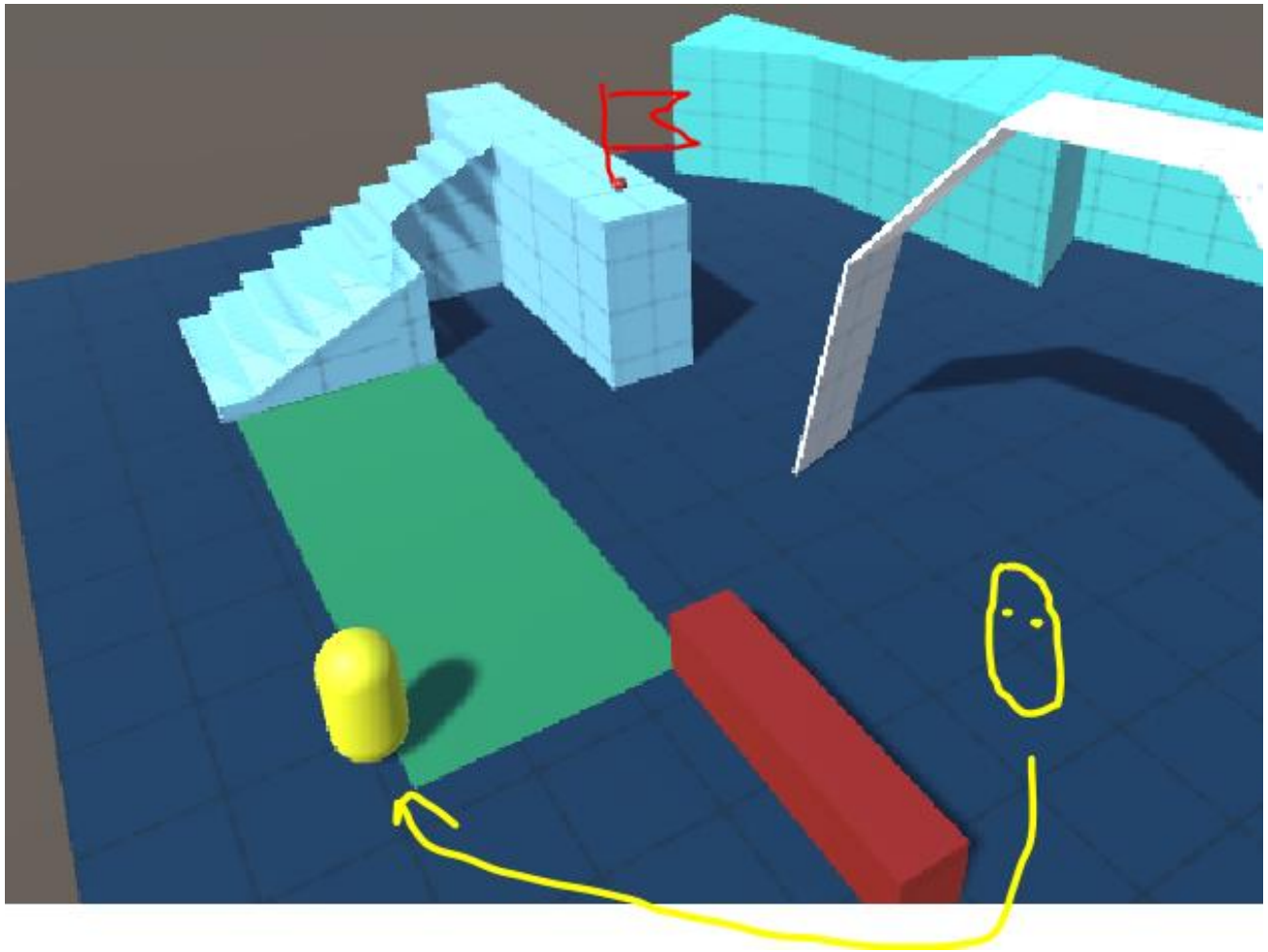


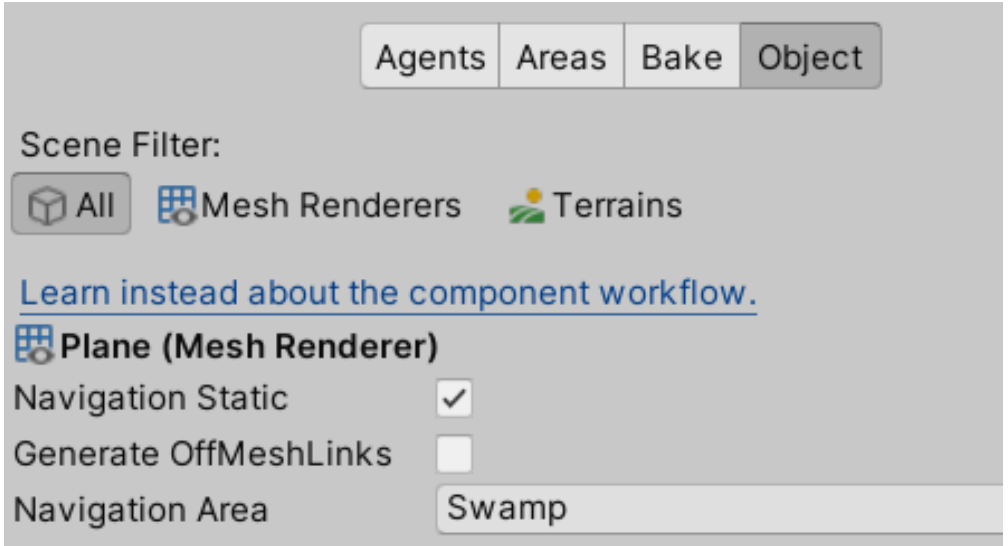


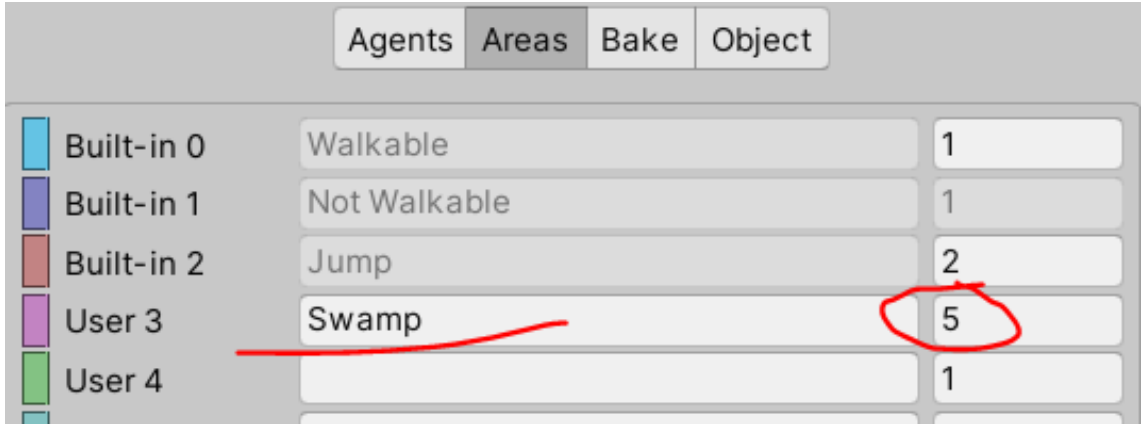
Движение агента к заданной цели

Код по ссылке
https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent-destination.html

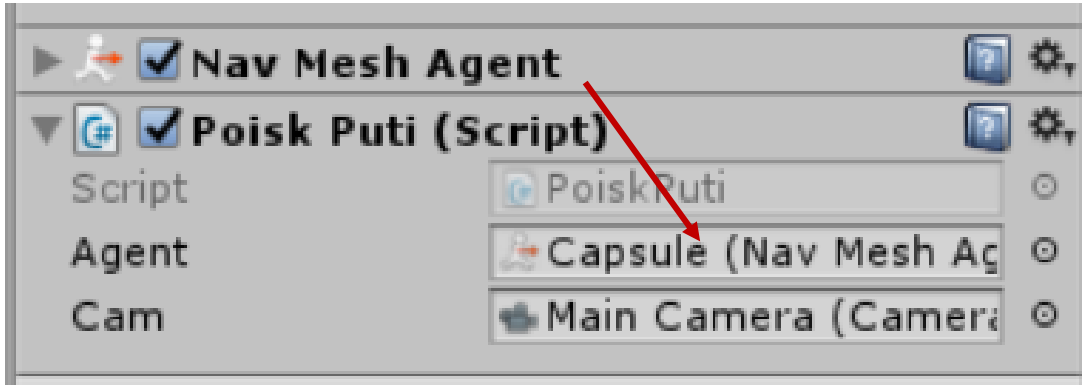
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

Ссылок: 0
public class MoveToTarget : MonoBehaviour
{
    public Transform target;
    Vector3 destination;
    NavMeshAgent agent;

    Ссылок: 0
    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        agent.destination=target.position;
    }

    void Update()
{
    //Обновить пункт назначения, если цель перемещается на одну единицу
    if (Vector3.Distance(destination, target.position) > 1.0f)
    {
        destination = target.position;
        agent.destination = destination;
    }
}
```




Agents
Areas
Bake
Object
Scene Filter:
All
Mesh Renderers
Terrains
Learn instead about the component workflow.
Plane (Mesh Renderer)
Navigation Static
Generate OffMeshLinks
Navigation Area
Swamp


Agents
Areas
Bake
Object
Built-in 0
Walkable
1
Built-in 1
Not Walkable
1
Built-in 2
Jump
2
User 3
Swamp
5
User 4
1

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

public class PoiskPuti : MonoBehaviour {
    [SerializeField] NavMeshAgent agent;
    [SerializeField] Camera cam;


    void Update () {
        if (Input.GetMouseButton (0))
        {

            Ray ray = cam.ScreenPointToRay (Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast (ray, out hit)) {
                agent.SetDestination (hit.point);
            }

        }
    }
}
```

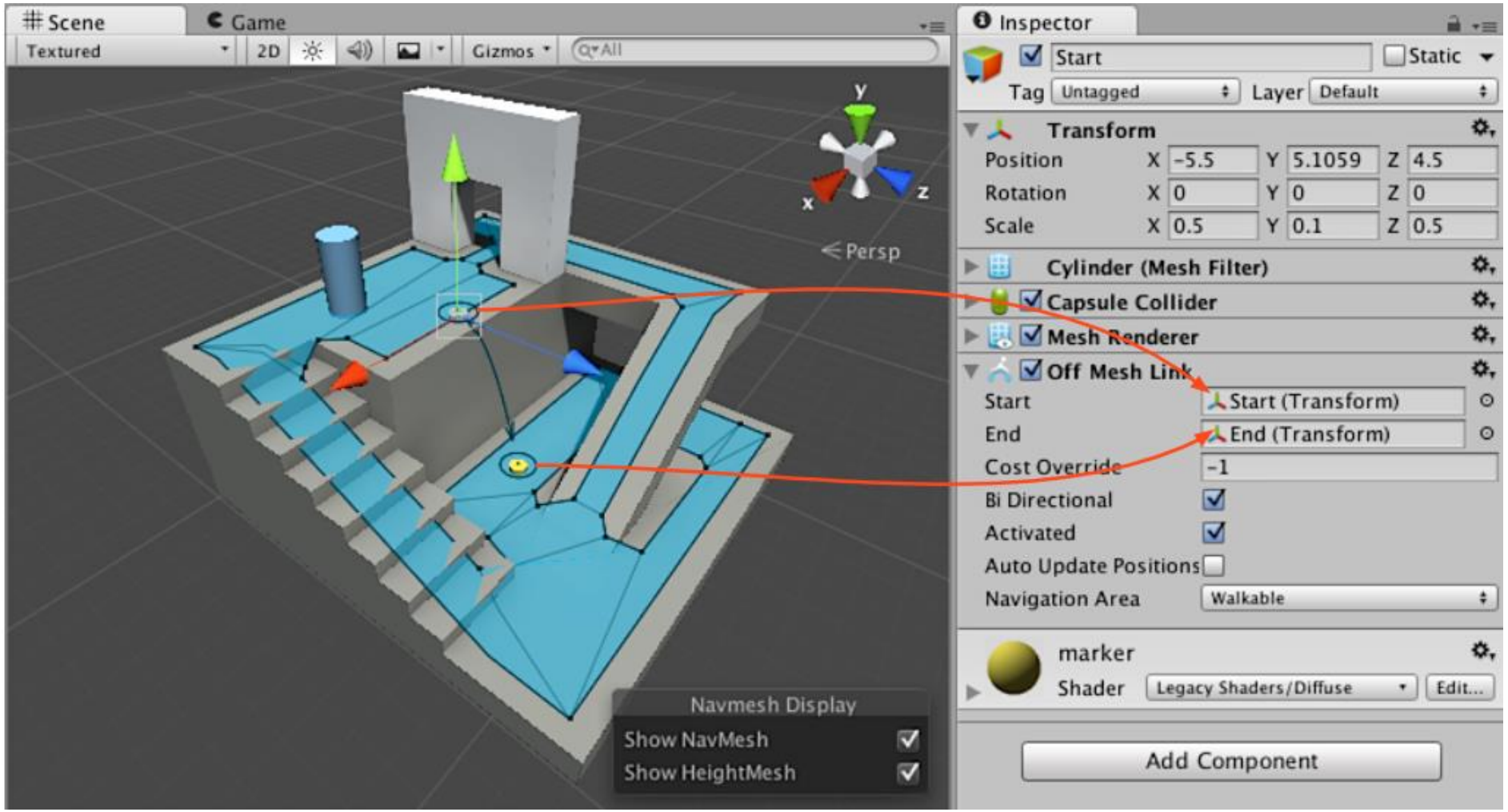


Код для управления движением агента при помощи щелчка мыши.

Некоторые скрипты можно посмотреть по ссылке на статью «Использование NavMesh для навигации ИИ в Unity» https://habr.com/ru/post/646039/

Off-mesh Link используются для того, чтоб создавать альтернативные пути перемещения для агентов. Для работы таких ссылок необходимо в компоненте Off-mesh Link указать положения начала и конца ссылки, а также стоимость такого перехода.

Инструкция https://docs.unity3d.com/Manual/nav-CreateOffMeshLink.html

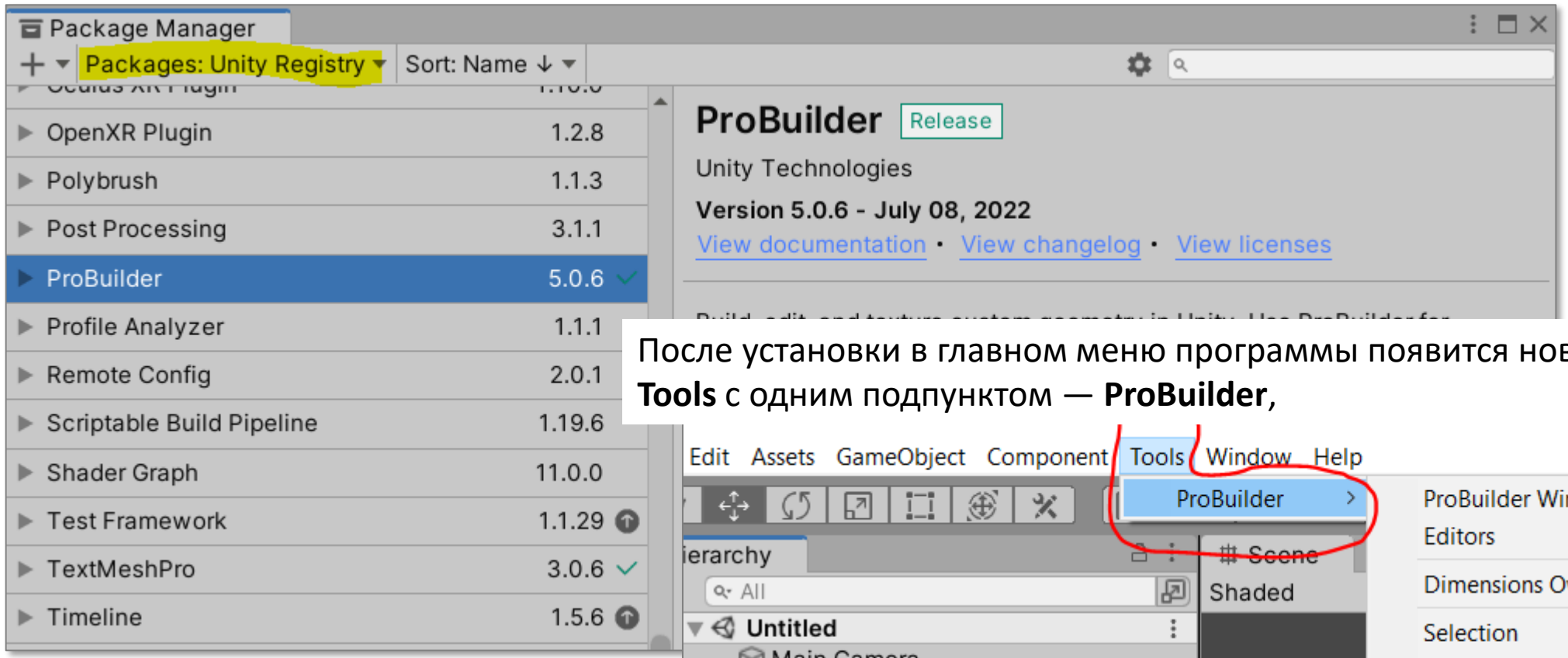


ProBuilder

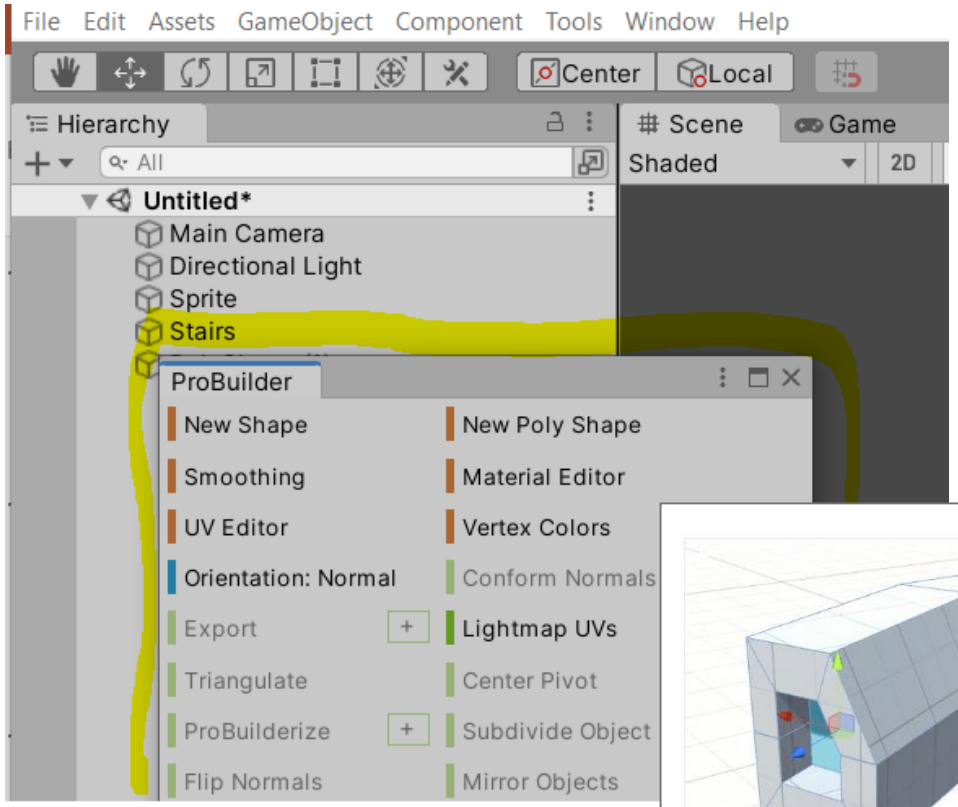
https://docs.unity3d.com/Packages/com.unity.probuilder@5.0/manual/index.html

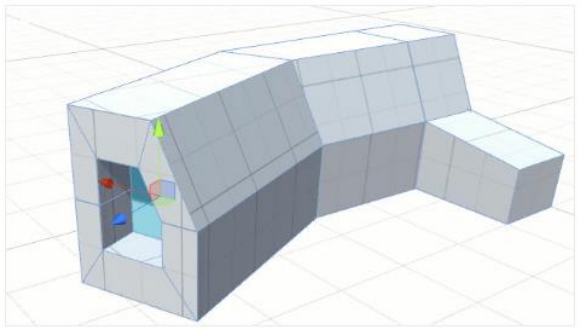
ProBuilder — это уникальный набор средств для построения двумерных и трехмерных геометрических форм с возможностью их дальнейшего редактирования и текстурирования.

Пункт меню Windows — **Package Manager**.

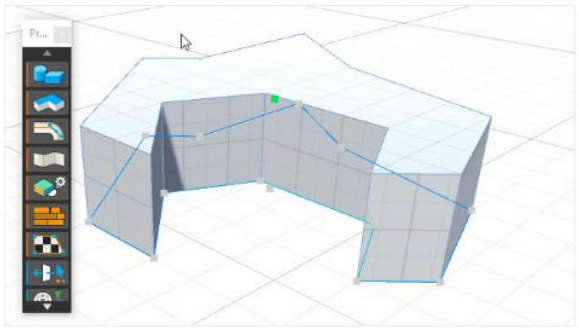


После установки в главном меню программы появится новый пункт **Tools** с одним подпунктом — **ProBuilder**,

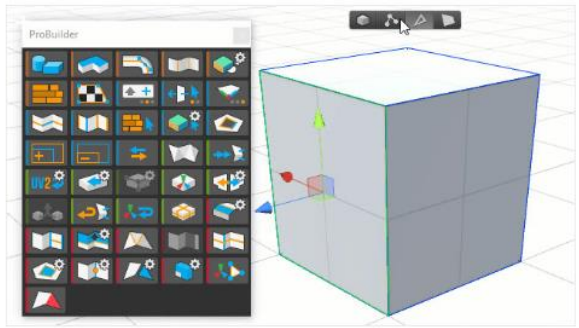




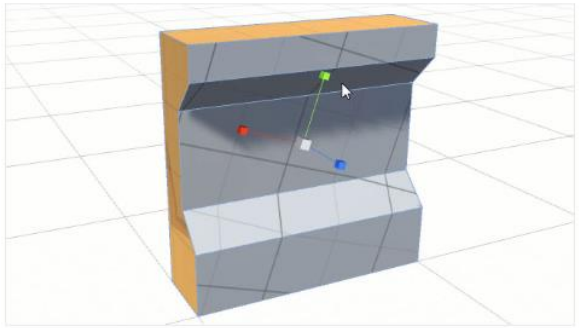
Extrude and inset



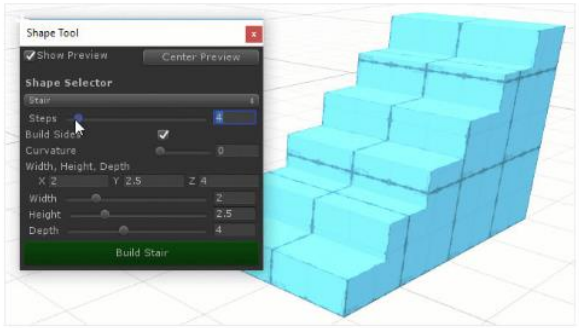
Versatile Poly Shapes



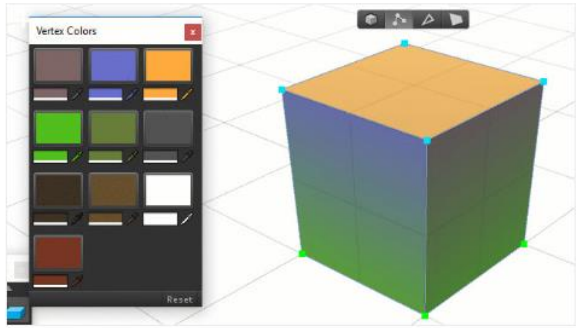
Dynamic user interface



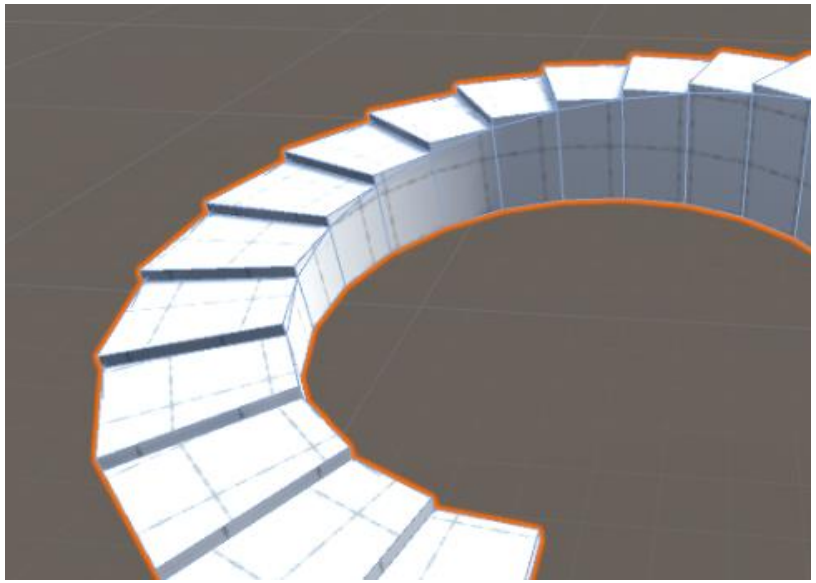
In-scene UV controls

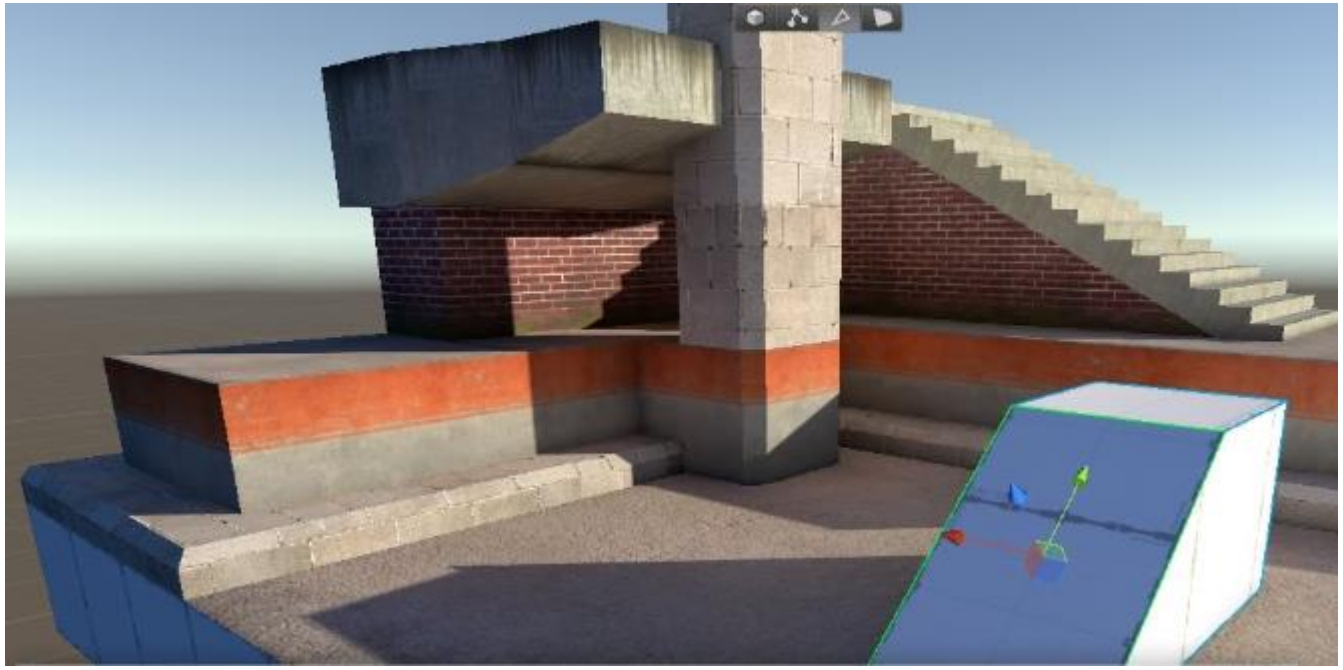


Procedural shapes



Vertex coloring

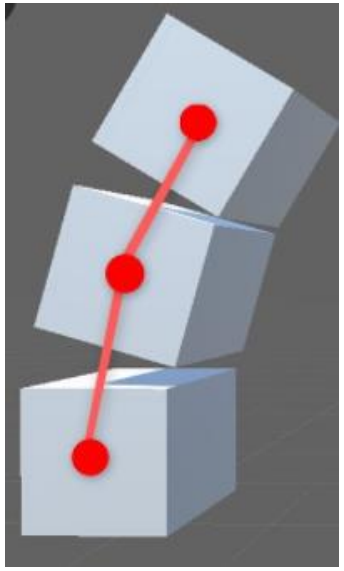


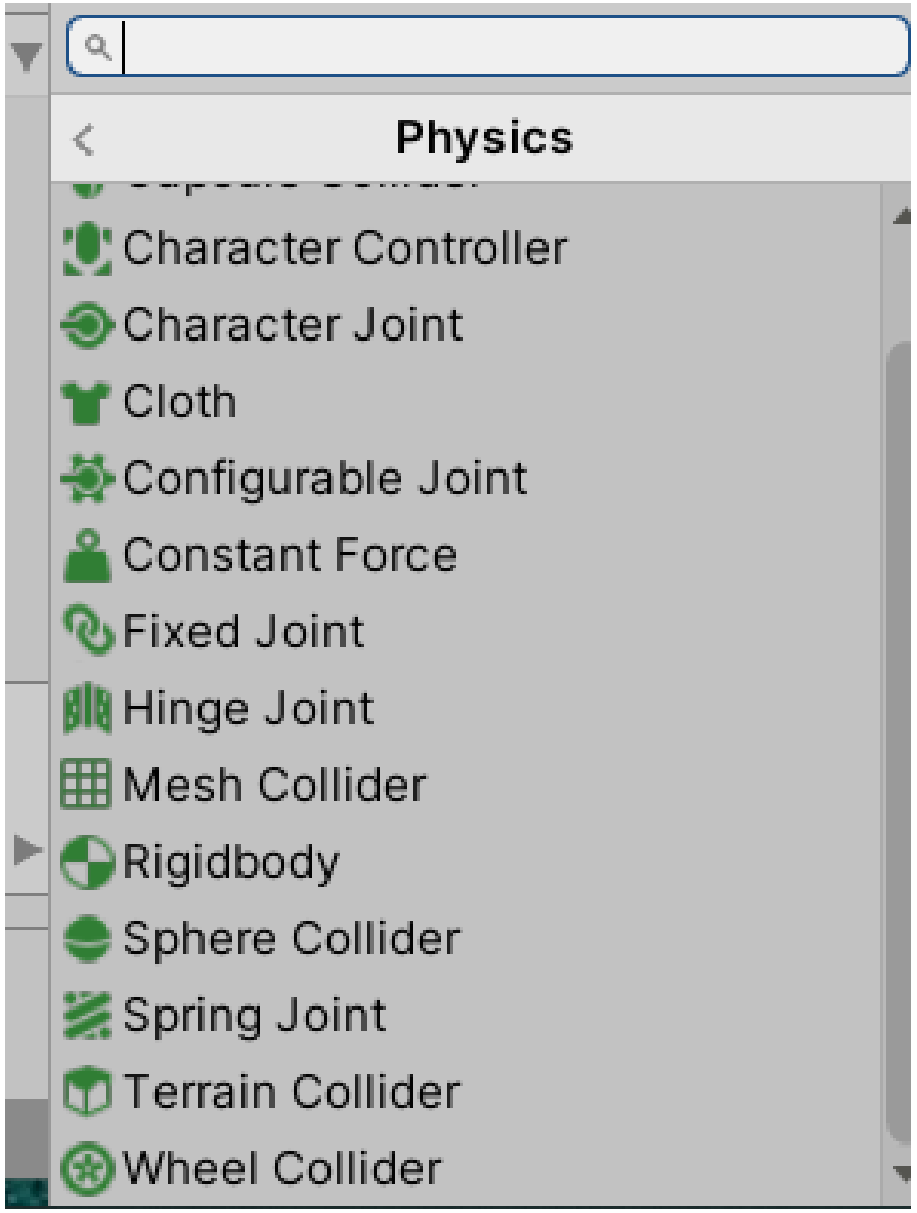


Joints

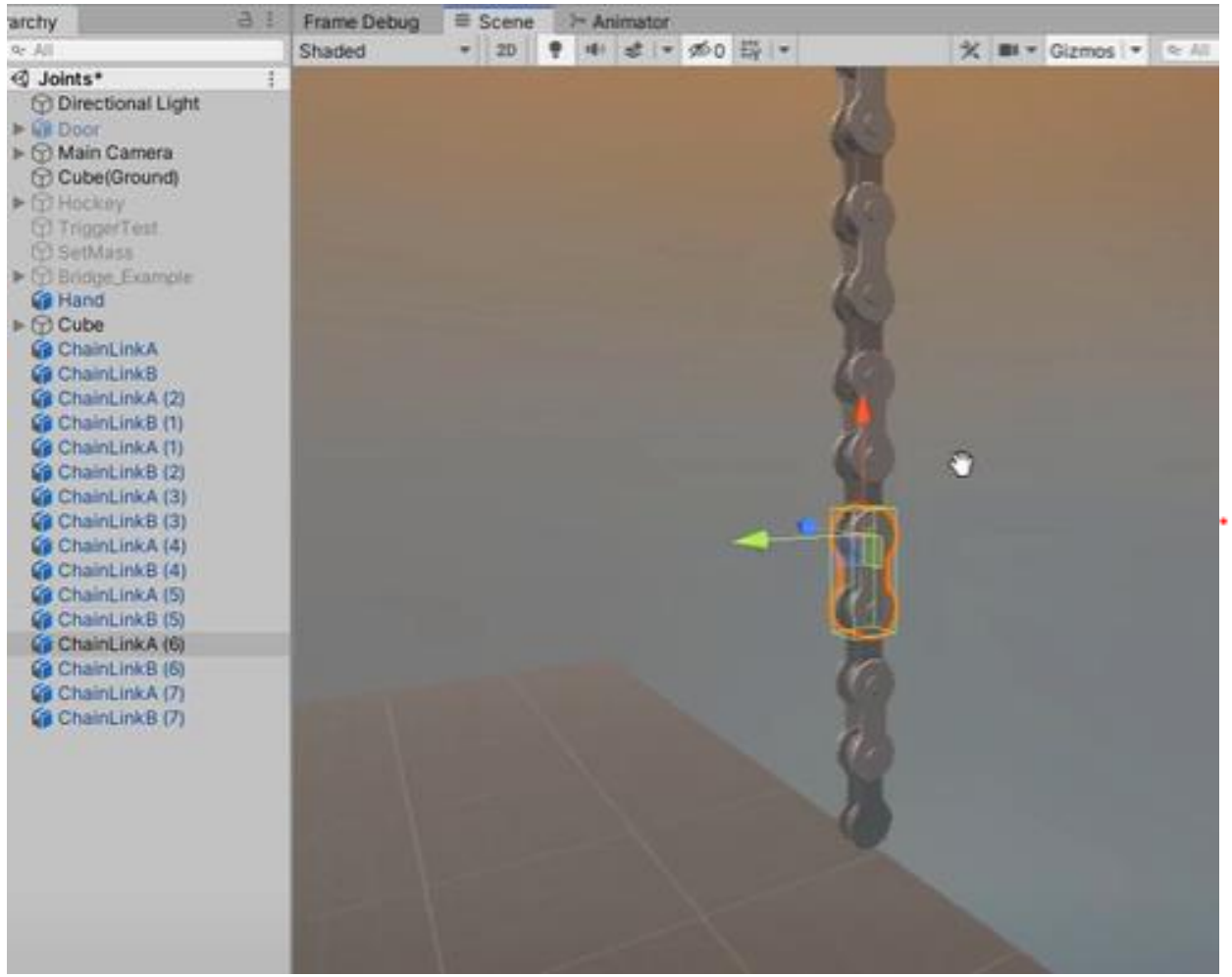
Компонент **Joint** соединяет Rigidbody с другим Rigidbody или фиксированной точкой в пространстве.

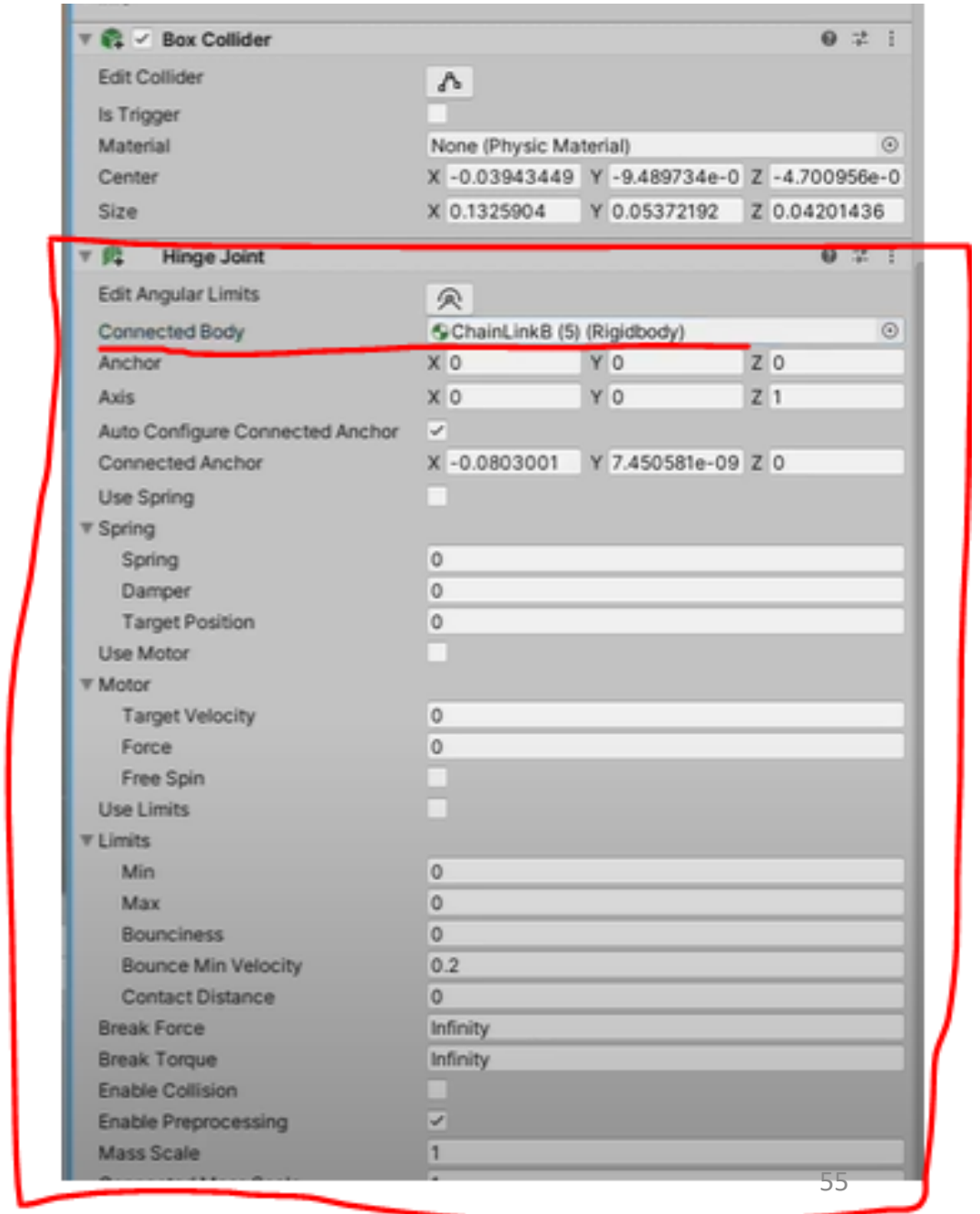
Unity предоставляет следующие соединения, которые применяют разные силы и ограничения к компонентам Rigidbody и, следовательно, придают этим телам разное движение:





Hinge Joint





https://www.youtube.com/watch?v=C8evrkExl34&list=PL8C4SmiVZY0wUFQrUXKQCS3BhRk2MIs8v&index=11