

# Взаимодействие с объектами на сцене

EventSystem

Raycast

NavMash

Joints

# EventSystem

Система событий - способ отправки событий к объектам в приложении, основанный на вводе с клавиатуры или мыши; с помощью касаний или персональных устройств.

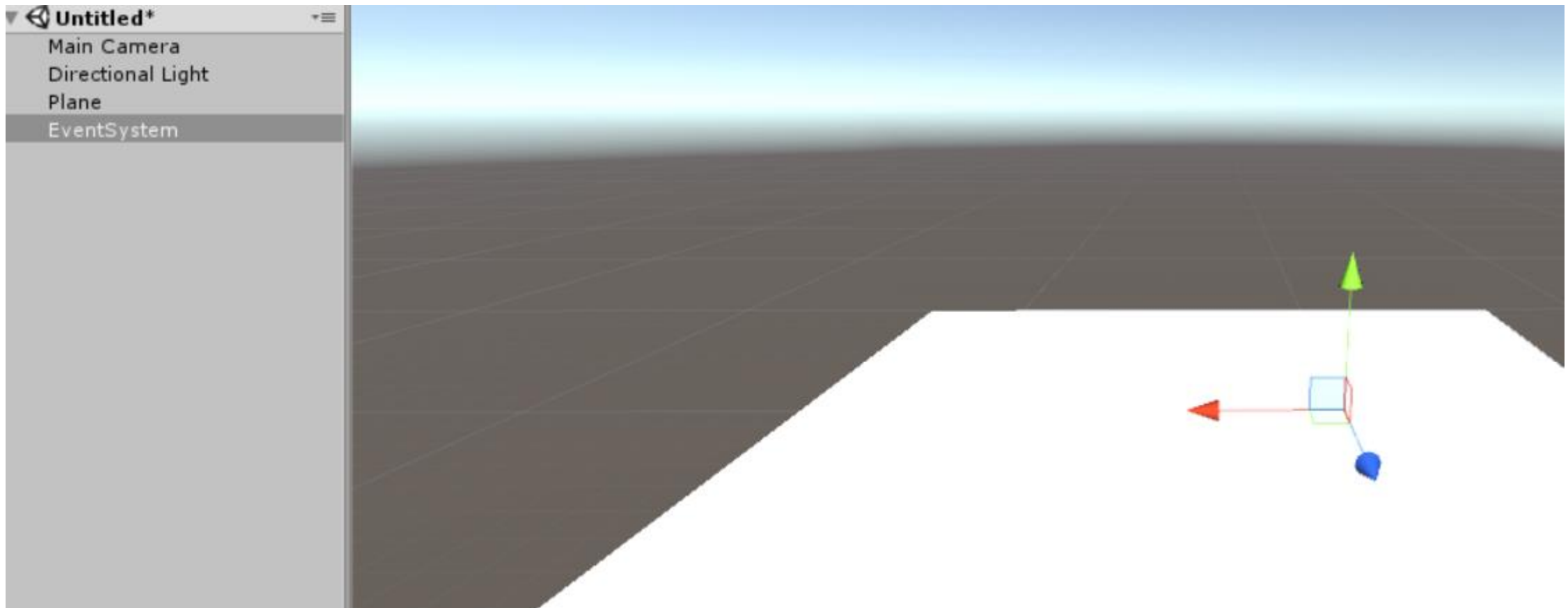
Сцена должна содержать только одну систему событий.

## **Create->UI->Event System**

Добавленный на сцену компонент EventSystem не содержит особой функциональности, потому что предназначен для управления и организации связи между модулями компонента.

# EventSystem

В окне Hierarchy контекстное меню UI->EventSystem



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, чтобы ее включить **нужно назначить камере компонент Physics Raycaster**

Добавляем компонент **Physics Raycaster** для камеры



## 2. Для толчка применяем **AddForceAtPosition**

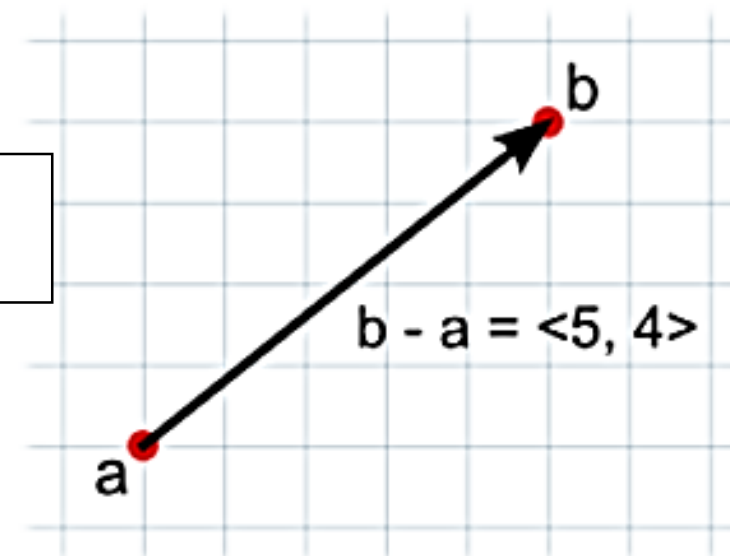
Что бы определить направление, в котором будем толкать куб, вспомним векторную алгебру.

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

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

### Вычитание

Вычитание векторов чаще всего используется для определения направления. Параметр `pointB` имеет значение:-



*//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);
```

*//учитываем направление*

```
Vector3 pointA=eventData.pointerPressRaycast.worldPosition;
```

```
Vector3 pointB = Camera.main.transform.position;
```

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

```
direction = direction.normalized;
```

```
Vector3 force = direction * 500;
```

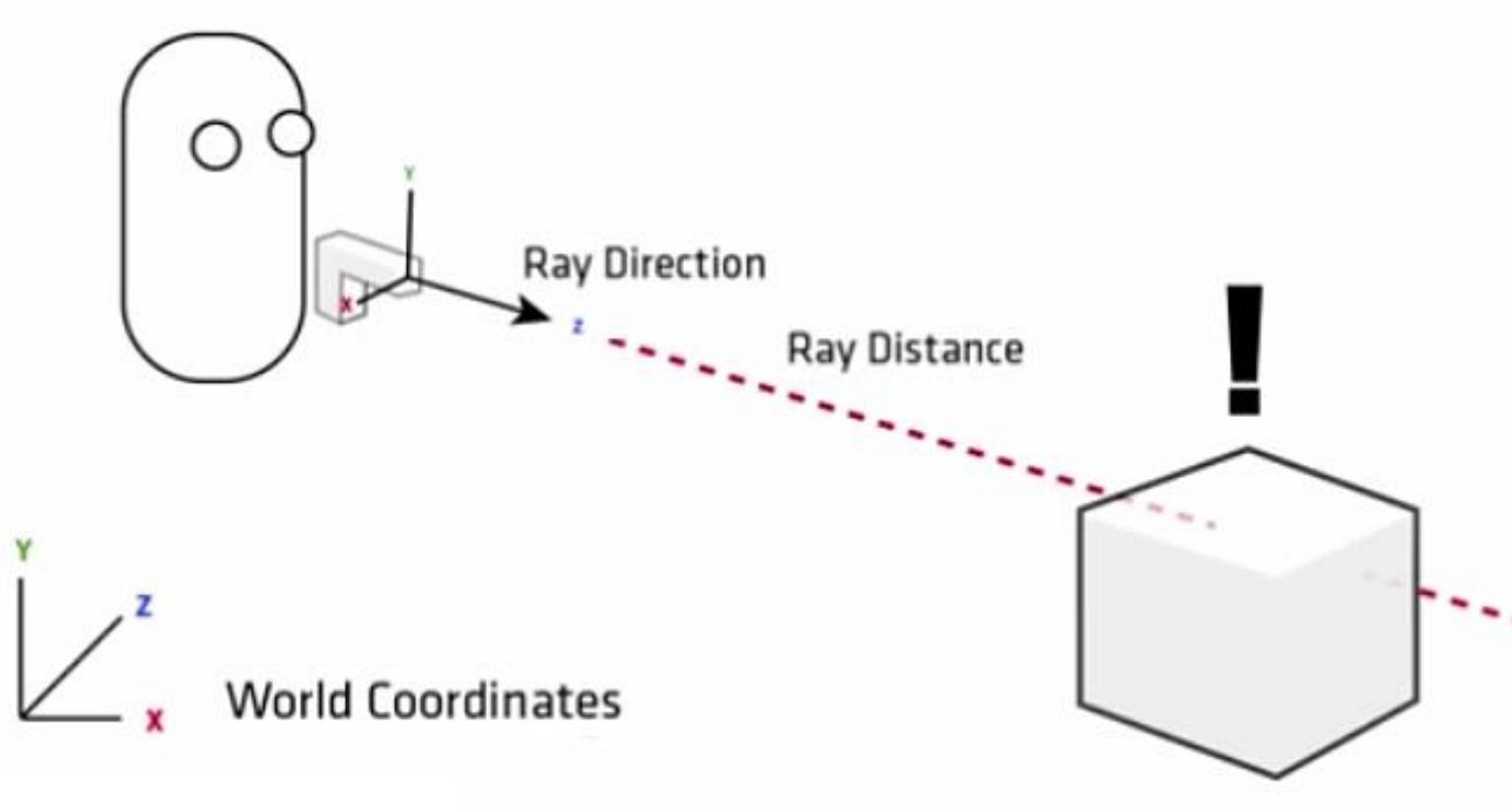
```
Vector3 target = eventData.pointerPressRaycast.worldPosition;
```

```
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 касанием экрана).

# Raycast

## Physics.Raycast



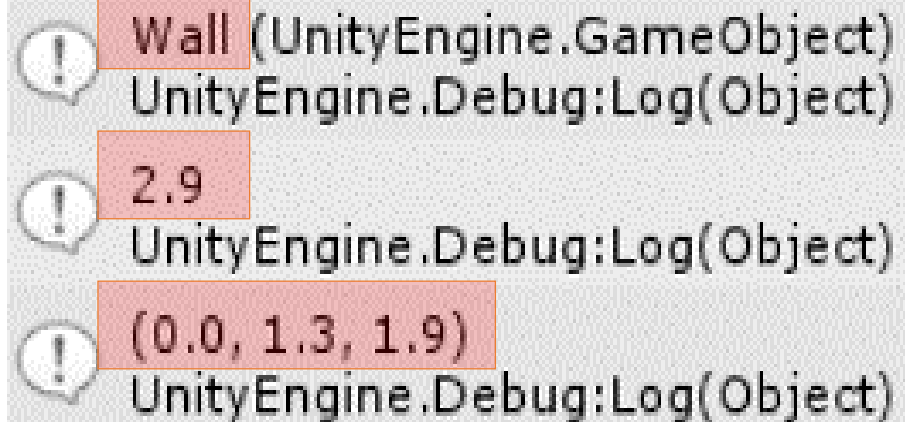
```
public static bool Raycast (  
    Vector3 origin,  
    Vector3 direction,  
    out RaycastHit hitInfo,  
    float maxDistance,  
    int layerMask,  
    QueryTriggerInteraction queryTriggerInteraction  
)
```

## Raycast

origin	Начальная точка луча в мировых координатах.
direction	Направление луча.
maxDistance	Максимальное расстояние, которое луч должен проверять на наличие столкновений.
layerMask	Маска слоя, которая используется для выборочного игнорирования коллайдеров при бросании луча.
queryTriggerInteraction	Указывает, должен ли этот запрос удалять триггеры.

# Raycast

```
RaycastHit info;  
if (Physics.Raycast (transform.position,transform.forward*4f, out info,4f))  
{  
    Debug.Log(info.collider.gameObject);  
    Debug.Log(info.distance);  
    Debug.Log(info.point);  
}
```



The screenshot shows three log entries in the Unity console, each preceded by a speech bubble icon containing an exclamation mark. The first entry is 'Wall (UnityEngine.GameObject)' with 'UnityEngine.Debug:Log(Object)' below it. The second entry is '2.9' with 'UnityEngine.Debug:Log(Object)' below it. The third entry is '(0.0, 1.3, 1.9)' with 'UnityEngine.Debug:Log(Object)' below it. The text 'Wall', '2.9', and '(0.0, 1.3, 1.9)' are highlighted with red boxes.

Wall (UnityEngine.GameObject)  
UnityEngine.Debug:Log(Object)  
2.9  
UnityEngine.Debug:Log(Object)  
(0.0, 1.3, 1.9)  
UnityEngine.Debug:Log(Object)

info.rigidbody

.....

info.transform

.....

## Как можно задавать направление?

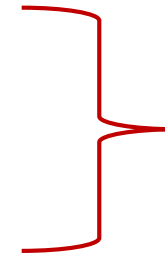


```
RaycastHit info;  
if (Physics.Raycast (transform.position, transform.forward*4f, out info,4f))  
{
```

Три основных направления совпадают с направлением осей:

transform.up  
transform.forward  
transform.right

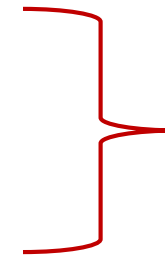
transform.-up  
transform.-forward  
transform.-right



Локальные  
координаты

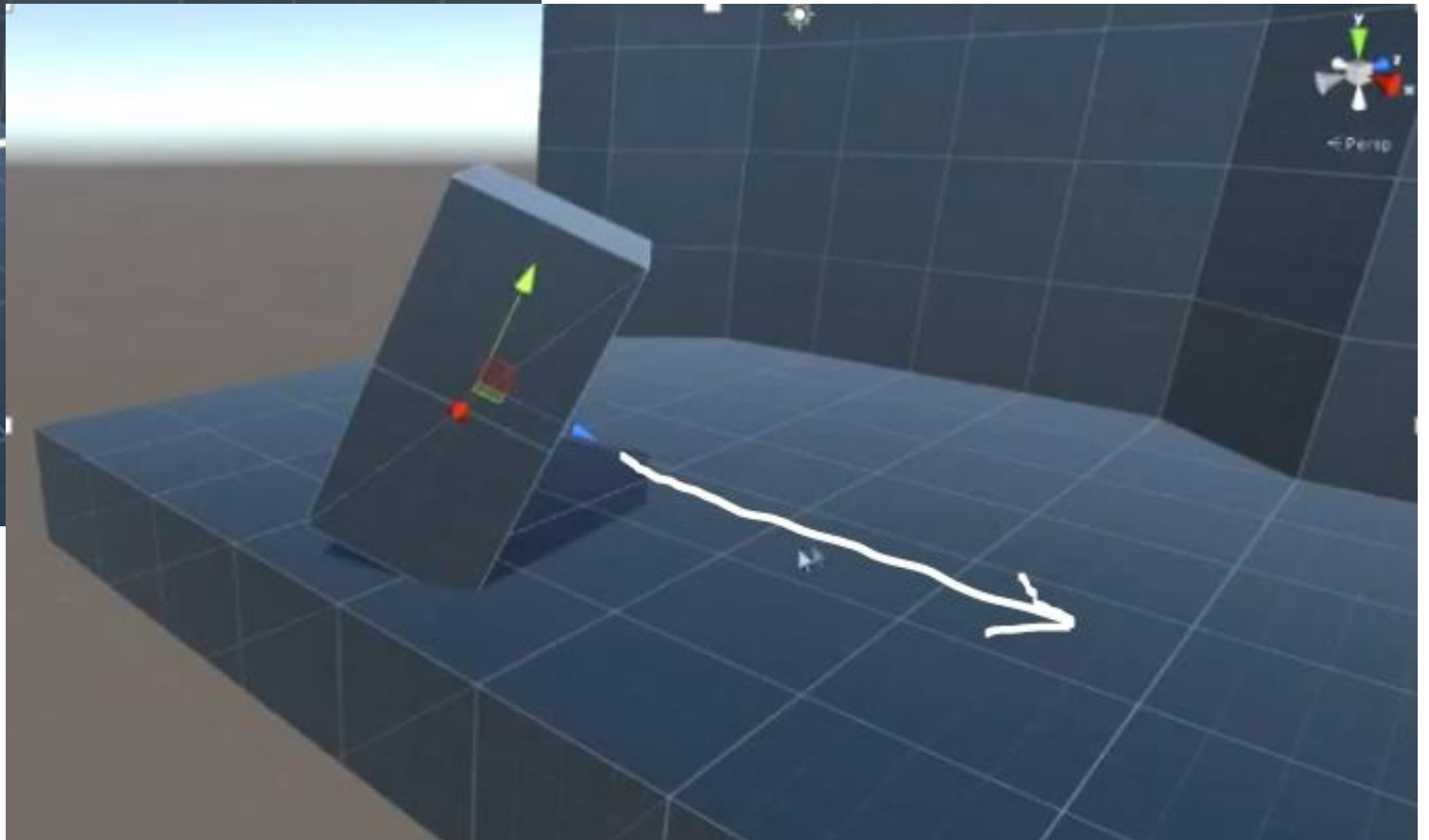
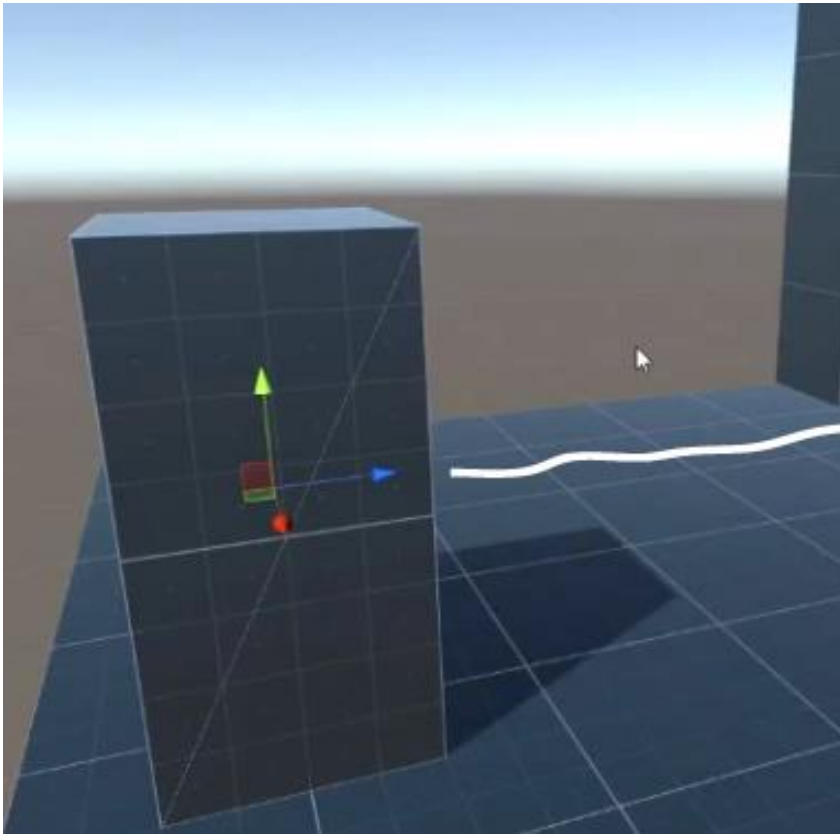
Vector3.up  
Vector3.forward  
Vector3.right

Vector3.-up  
Vector3.-forward  
Vector3.-right



Глобальные  
координаты



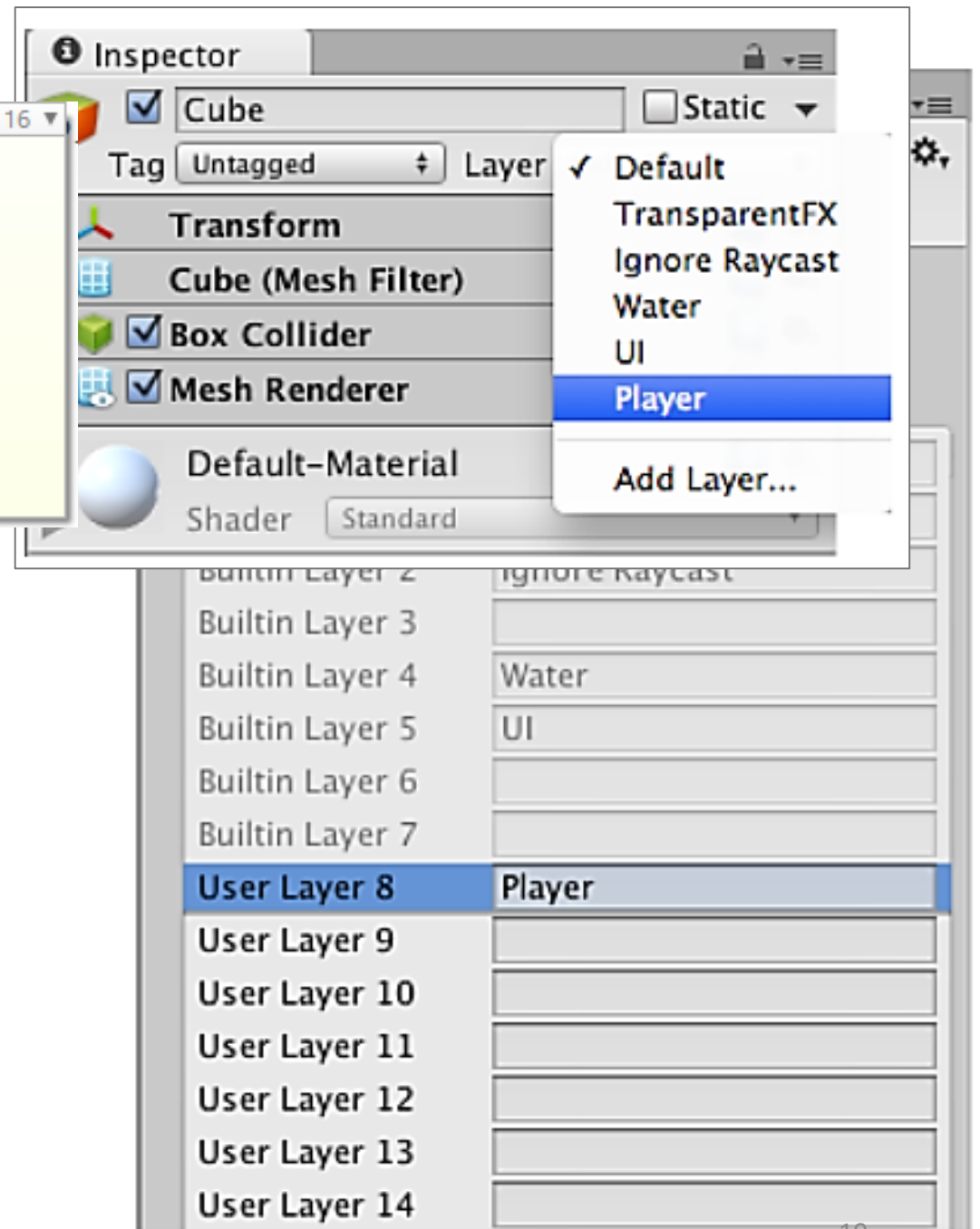


transform.up\*7.0f



transform. - forward \*1.0f

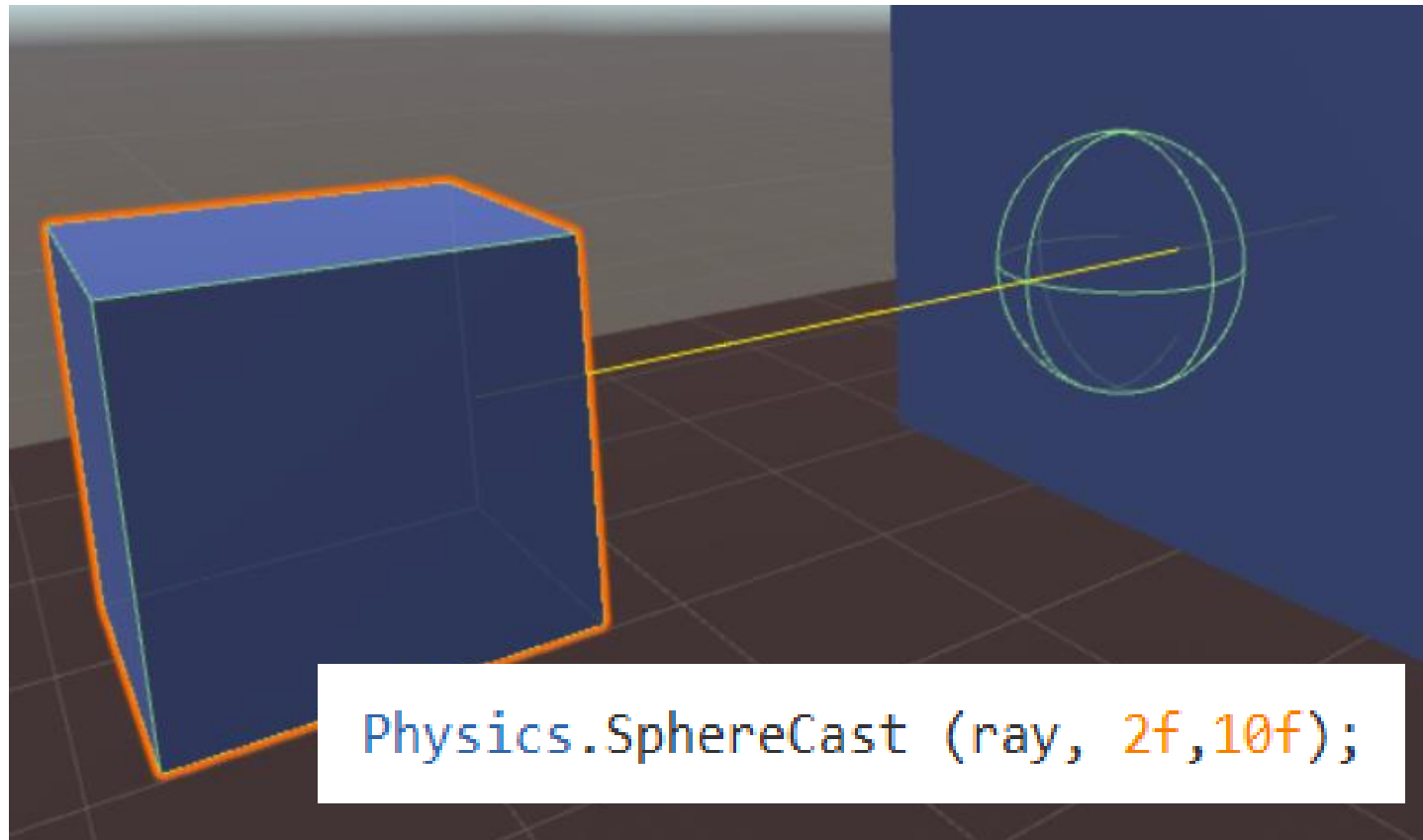
```
public static bool Raycast (
    Vector3 origin,
    Vector3 direction,
    out RaycastHit hitInfo,
    float maxDistance,
    int layerMask,
    QueryTriggerInteraction queryTriggerInteraction
)
```



`int layerMask = DefaultRaycastLayers`

**Слои** чаще всего используются Камерами для визуализации только части сцены и Светами для освещения только частей сцены. Но их также можно использовать с помощью **raycasting**, чтобы выборочно игнорировать коллайдеры или создавать столкновения.

## Physics.SphereCast

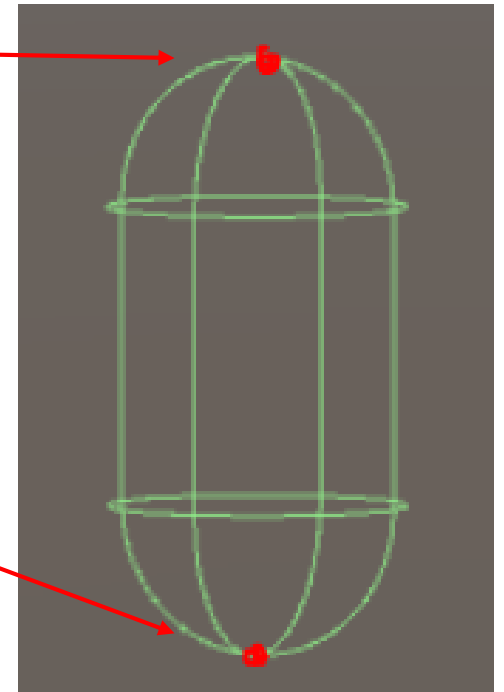


## Physics.CapsuleCast

```
public static bool CapsuleCast (  
    Vector3 point1,  
    Vector3 point2,  
    float radius,  
    Vector3 direction  
)
```

```
public static bool CapsuleCast (  
    Vector3 point1,  
    Vector3 point2,  
    float radius,  
    Vector3 direction,  
    out RaycastHit hitInfo,  
    float maxDistance,  
    int layerMask,  
    QueryTriggerInteraction queryTriggerInteraction  
)
```

## Physics.CapsuleCast



```
public float speed = 5.0f;  
public float obstacleRange = 0.5f;
```

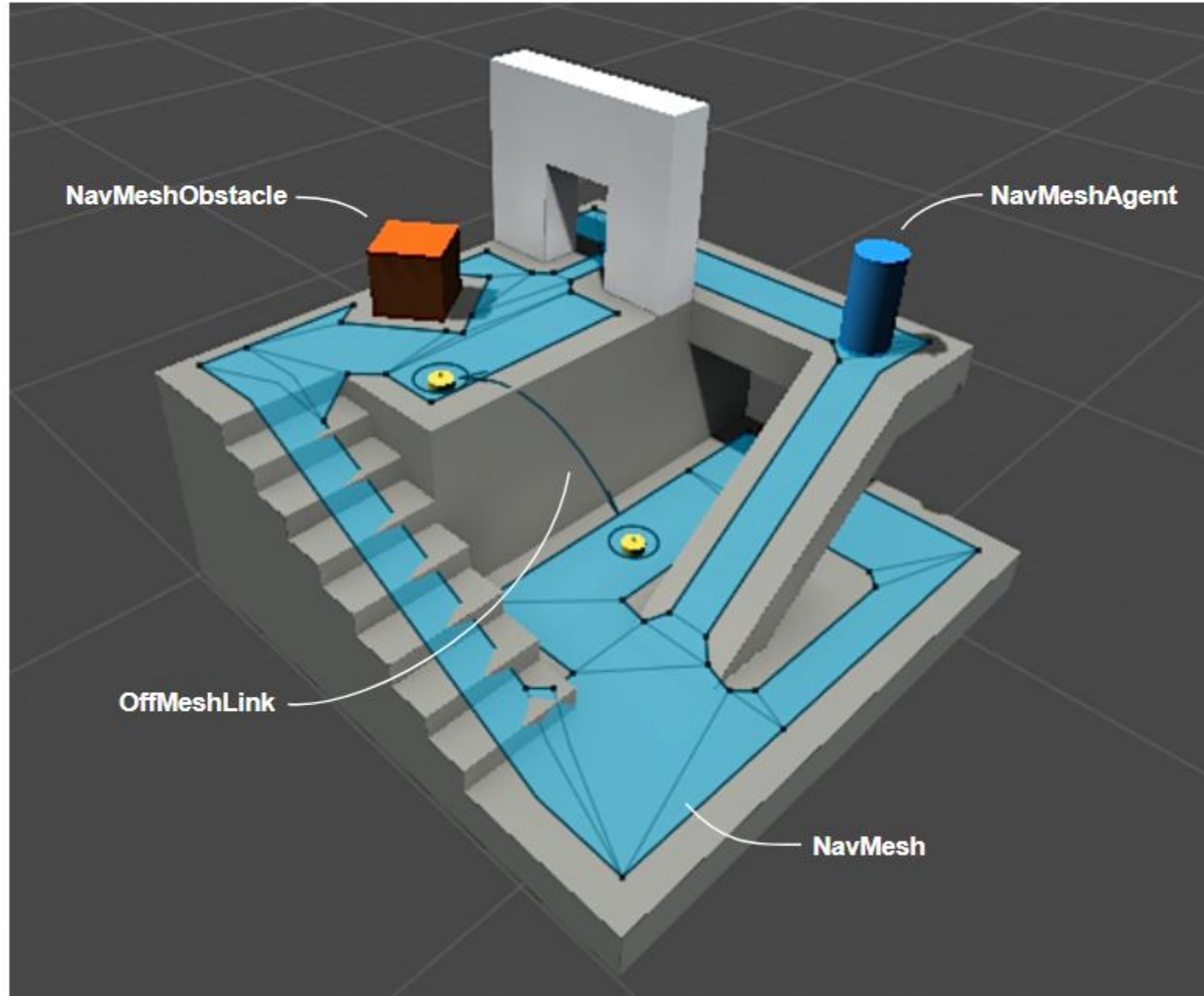


Демо

ССЫЛОК: 0

```
void Update()  
{  
    transform.Translate(0, 0, speed * Time.deltaTime);  
    Ray ray = new Ray(transform.position, transform.forward);  
    RaycastHit hit;  
    if (Physics.SphereCast(ray, 1.0f, out hit))  
    {  
        if (hit.distance < obstacleRange)  
        {  
            float angle = Random.Range(-110, 110);  
            transform.Rotate(0, angle, 0);  
        }  
    }  
}
```

# Navigation System in Unity



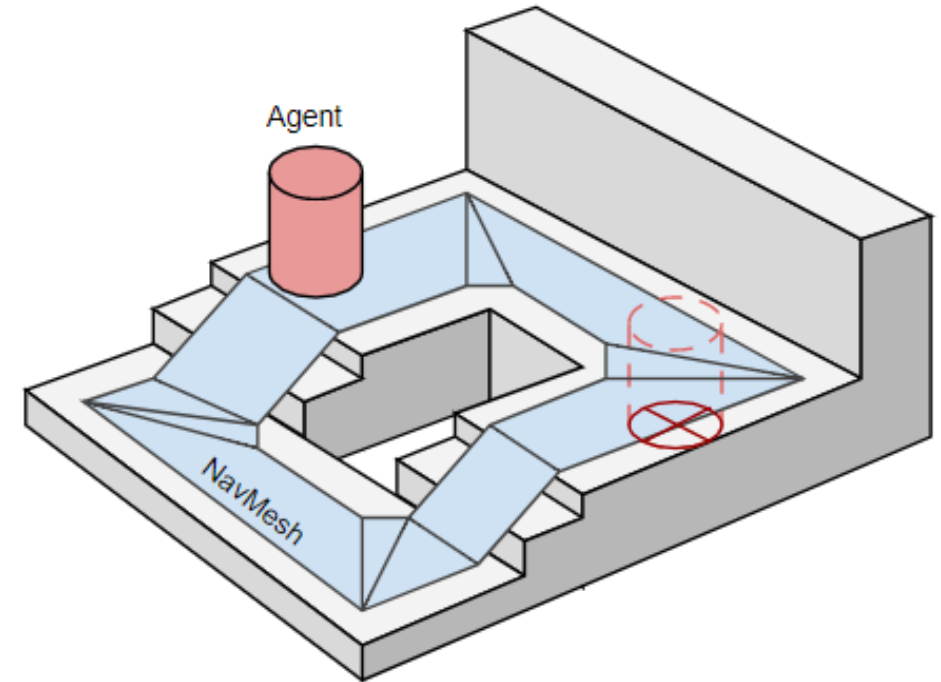
Проходимые области определяют места в сцене, где агент может стоять и двигаться. В Unity агенты описываются как цилиндры.

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

Затем местоположения соединяются с поверхностью, лежащей поверх геометрии сцены. 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.

## Walkable Areas

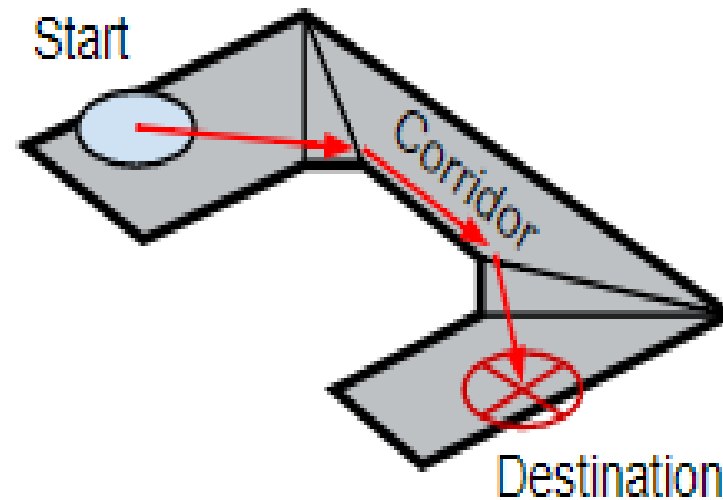
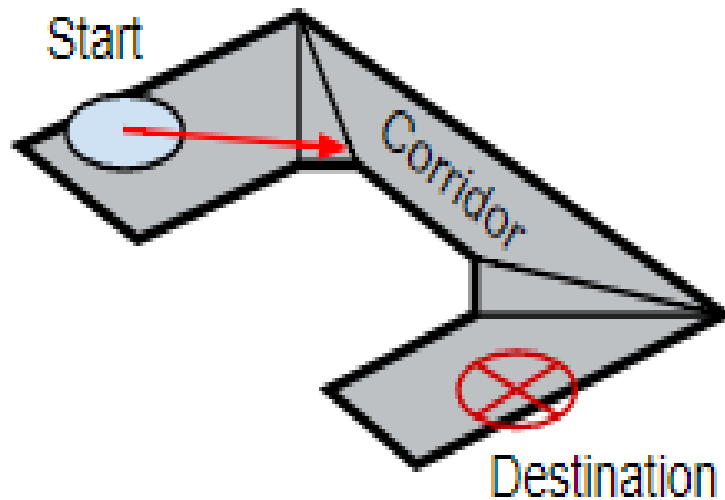




# Поиск путей

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

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



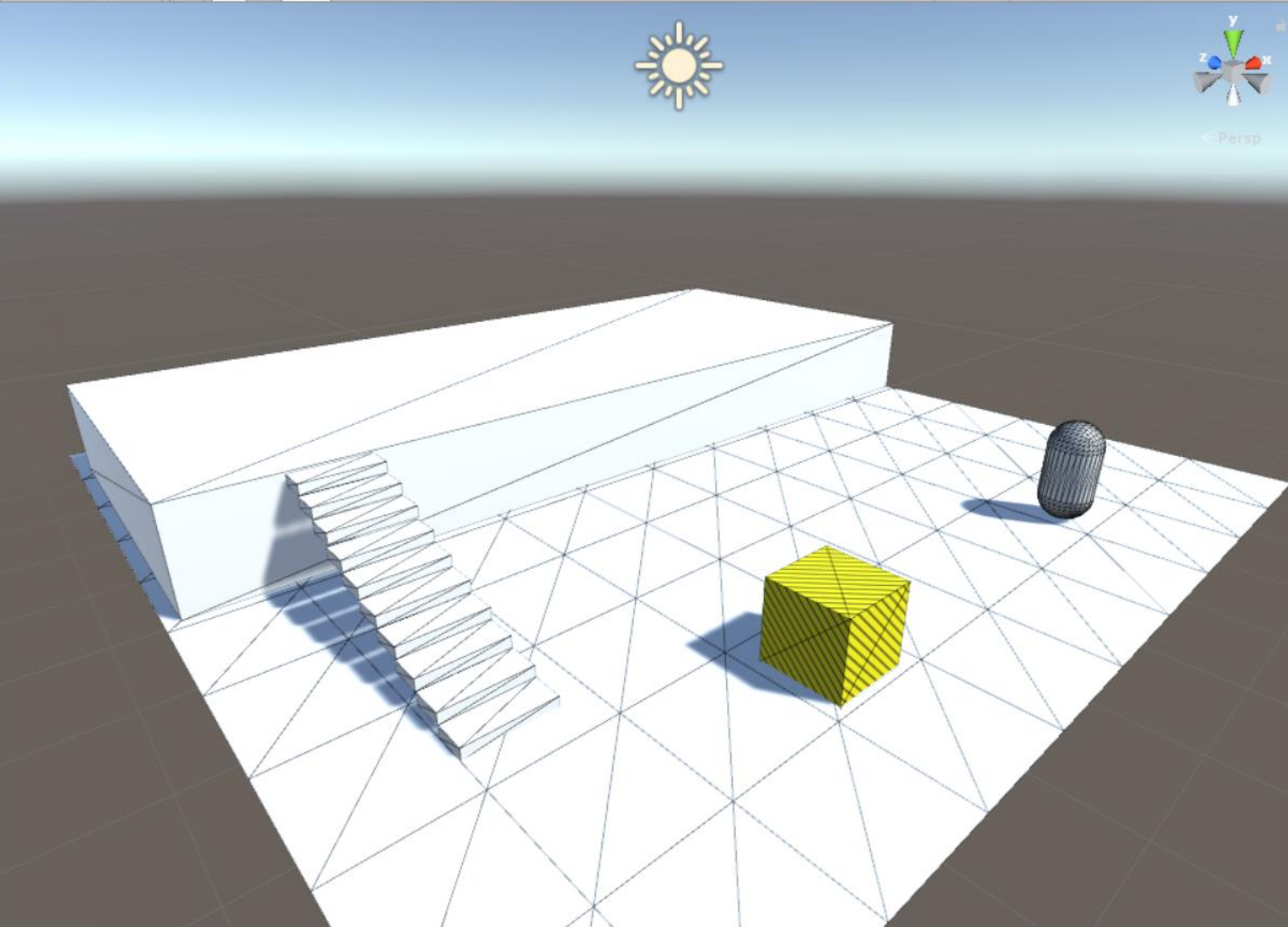
- scene\_2\*
- Capsule
- Cube (2)
- Cube (3)
- Cube (4)
- Cube (5)
- Cube (6)
- Cube (7)
- Cube (8)
- Cube (9)
- Cube (10)
- Cube (11)
- Obstacle
- Plane
- Platform

Path:

Project

Console

- Create All Models All Prefabs
- Assets**
- DigitalKons
  - Prototyp
  - Demo
  - Der
  - Resou
  - Mat
  - Mes
  - B
  - C
  - C
  - C
  - C
  - C
  - C
  - E
  - L
- DigitalKons...
- scene\_1
- scene\_2



Asset Labels

AssetBundle

None

None



Center Global



Collab



Account

Layers

Layout

Hierarchy

Create MeshRenderer

- scene\_2\*
- Capsule
- Cube (2)
- Cube (3)
- Cube (4)
- Cube (5)
- Cube (6)
- Cube (7)
- Cube (8)
- Cube (9)
- Cube (10)
- Cube (11)
- Obstacle
- Plane
- Platform

Path:

- Cube (2)
- scene\_2

Project

Create Console

All Models All Prefabs

Assets

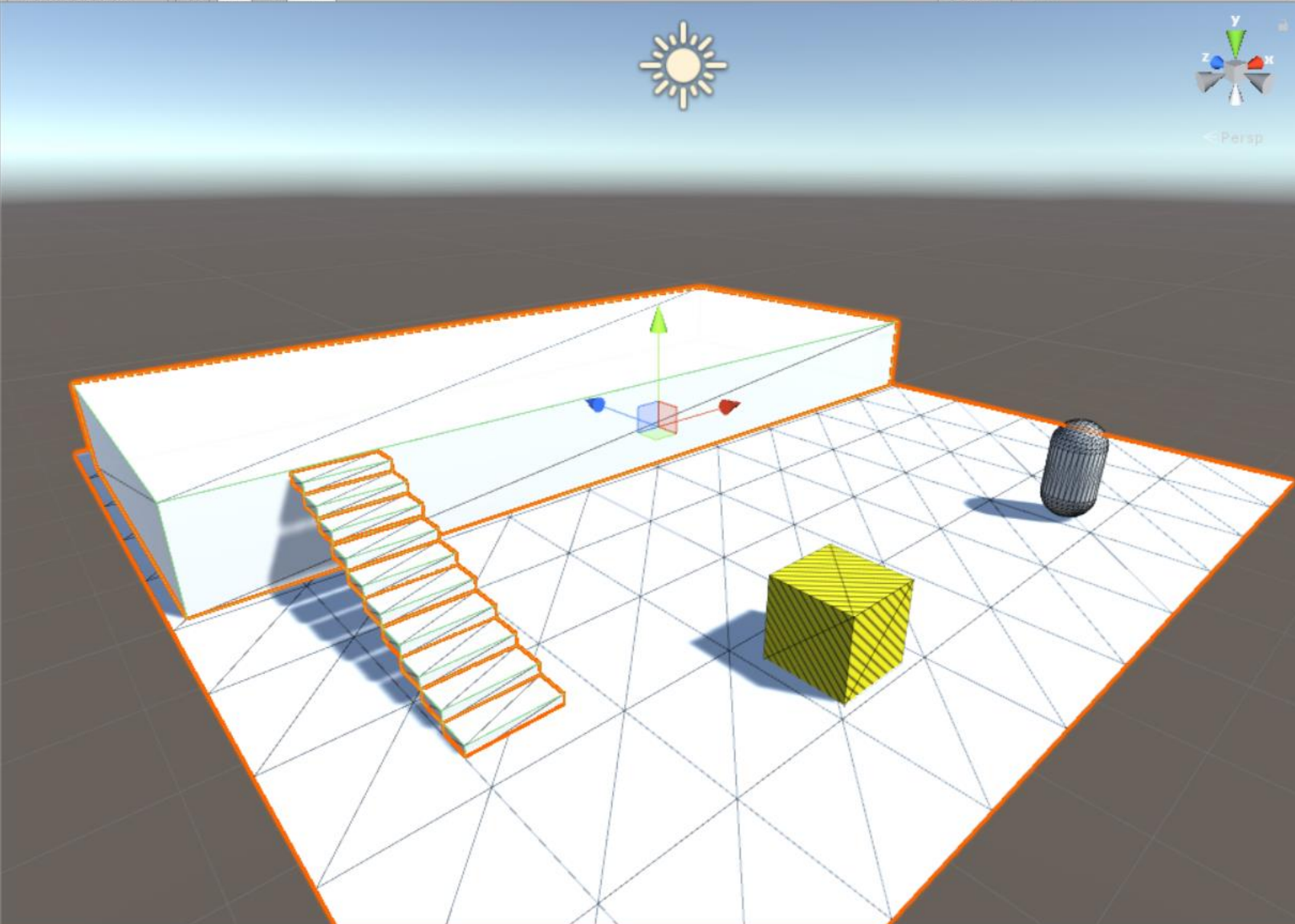
- DigitalKons...
- scene\_1
- scene\_2

# Scene Game Asset Store

Shaded Wireframe 2D

Gizmos

All



Inspector

Services Navigation

Static

- Nothing
- Everything
- Lightmap Static
- Occluder Static
- Batching Static
- Navigation Static
- Occludee Static
- Off Mesh Link Generation
- Reflection Probe Static

Default-Material Shader Standard

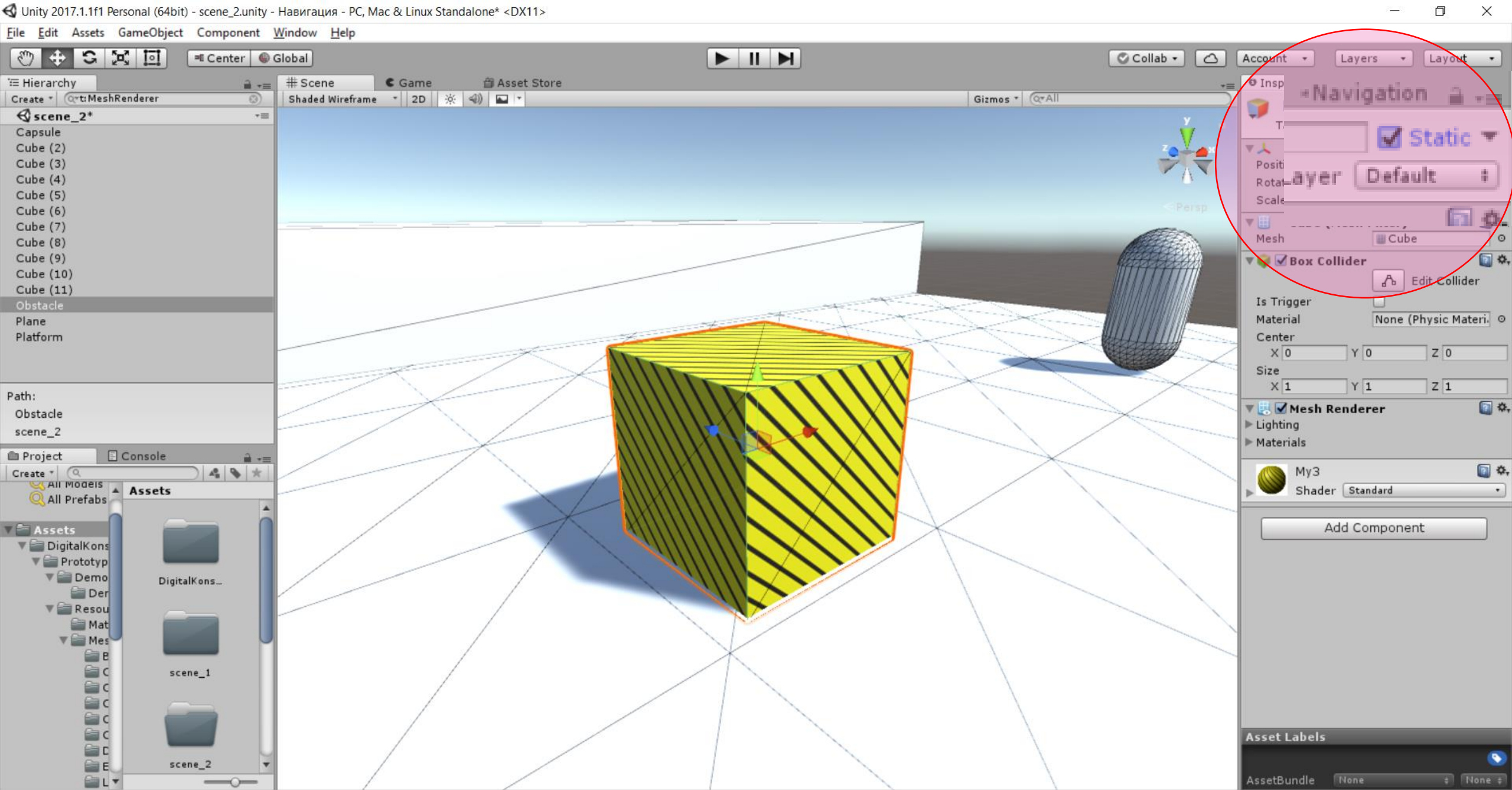
Components that are only on some of the selected objects cannot be multi-edited.

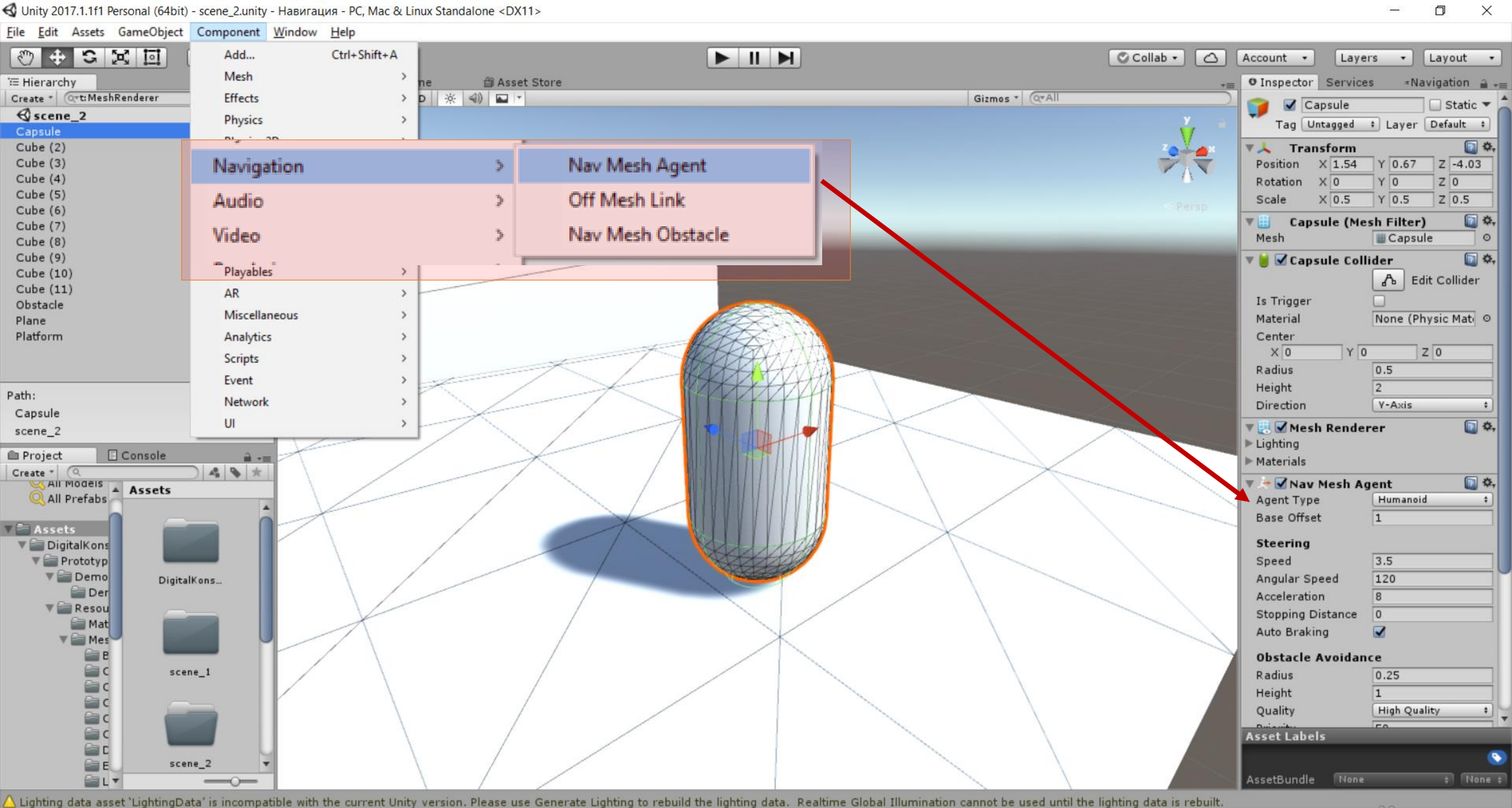
Add Component

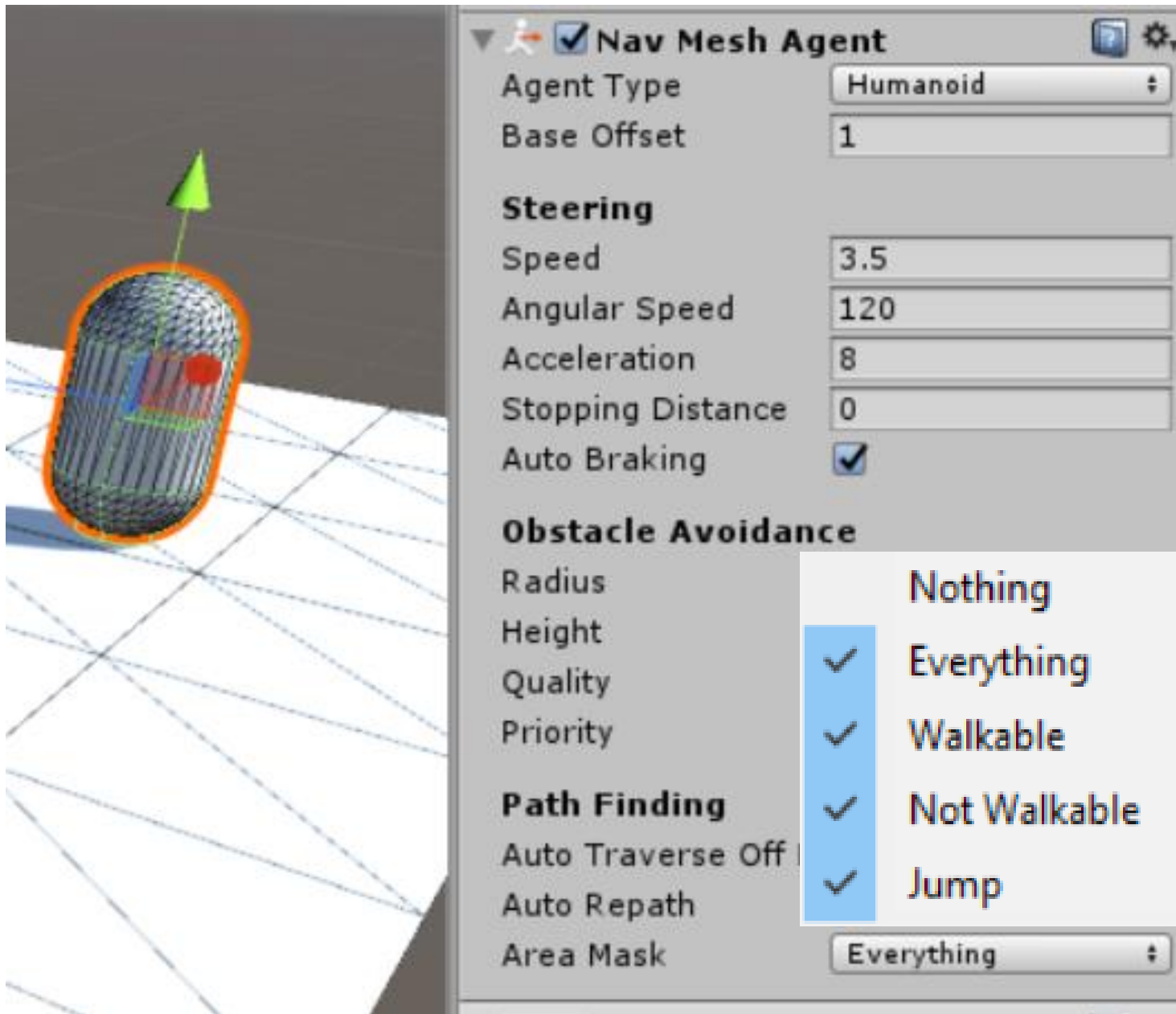
Asset Labels

AssetBundle None None









## Рулевое управление

Скорость  
Угловая скорость  
Ускорение  
Дистанция до препятствия  
Остановится перед непроходим

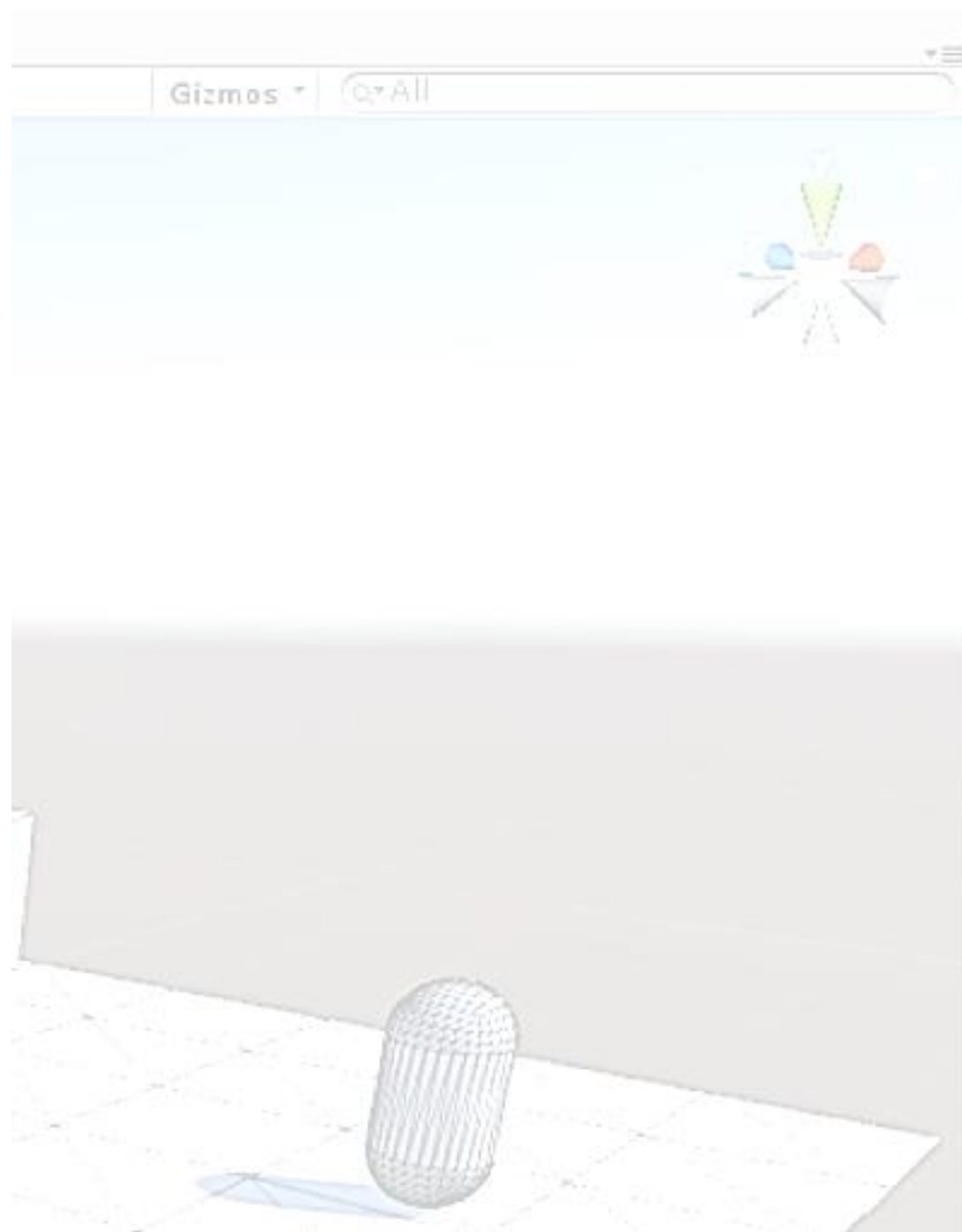
## Обход препятствий

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

## Поиск пути

Автоматически перемещаться по сетке  
Автоматически перестраивать путь  
Маска области





Inspector Services **Navigation**

Agents Areas Bake Object

Agent Types

Humanoid

**Capsule**

+ -

$R = 0.5$

$H = 2$

0.4 45°

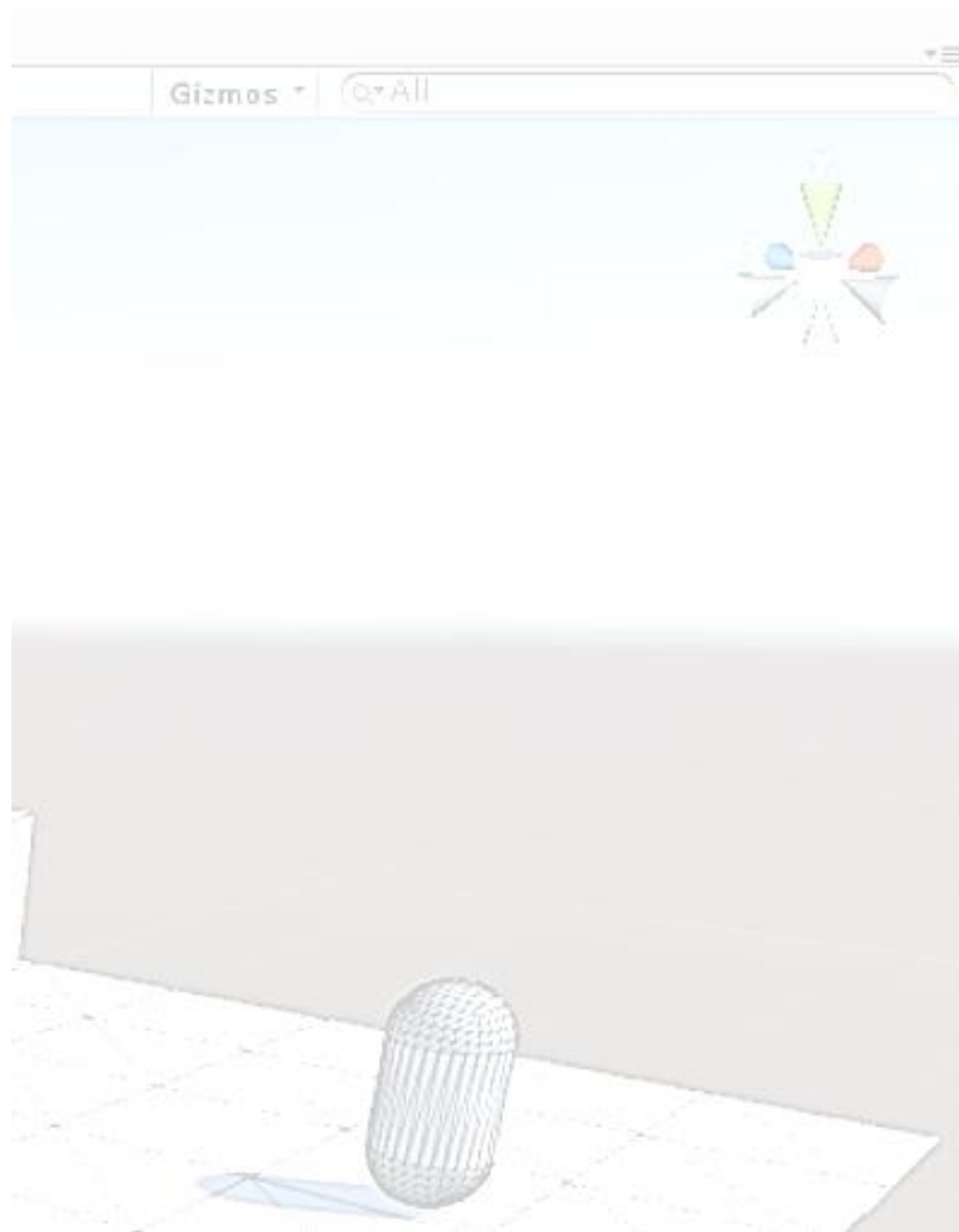
Name Capsule

Radius 0.5

Height 2

Step Height 0.4

Max Slope



Inspector Services **Navigation**

Agents Areas **Bake** Object

[Learn instead about the component workflow.](#)

**Baked Agent Size**

R = 0.5  
H = 2

0.4 45°

Agent Radius 0.5

Agent Height 2

Max Slope 45

Step Height 0.4

**Generated Off Mesh Links**

Drop Height 0

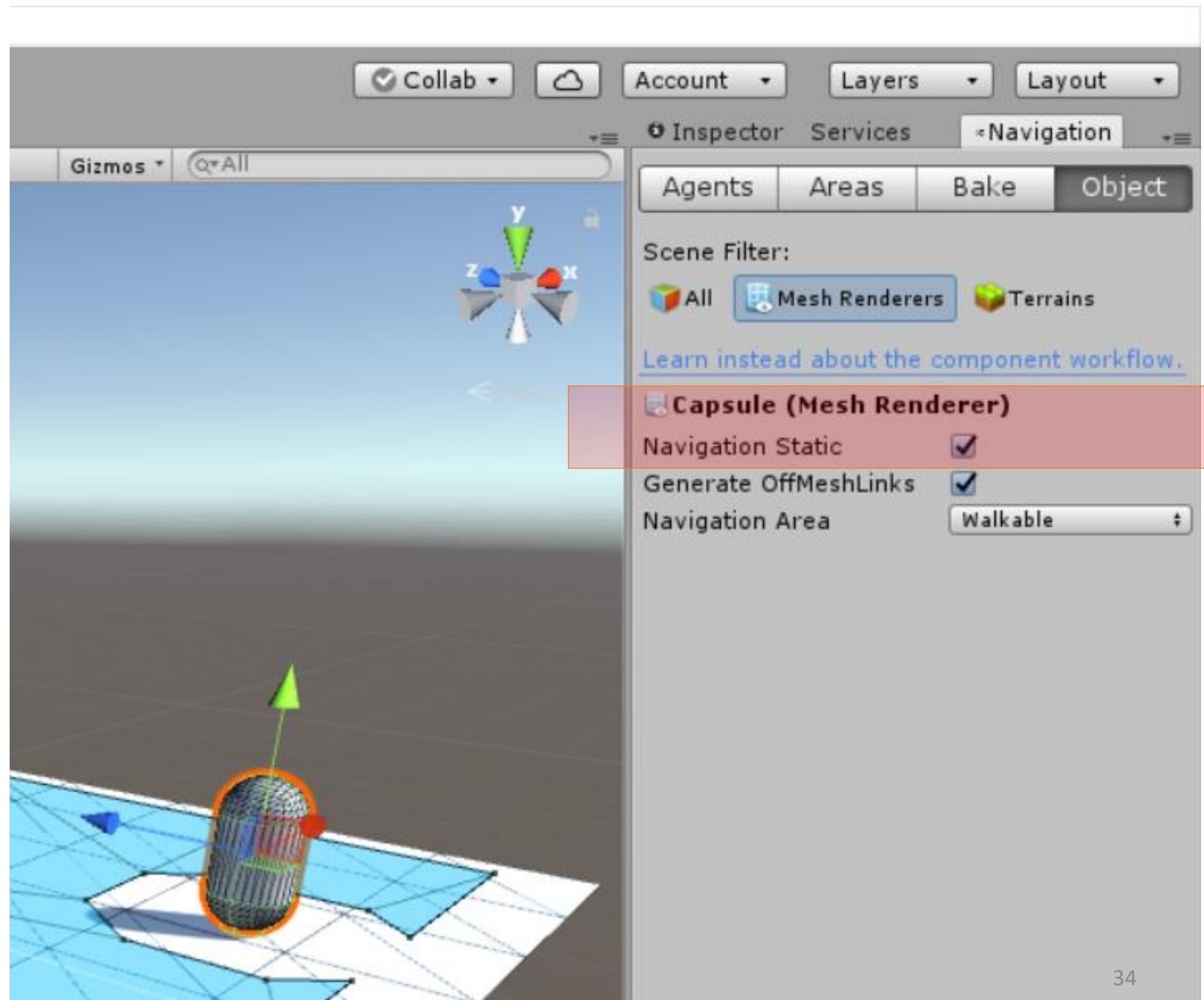
Jump Distance 0

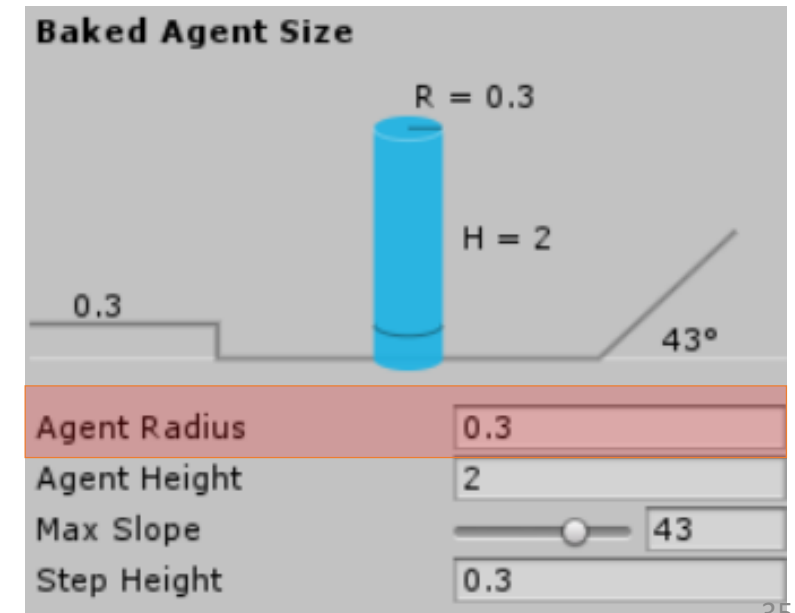
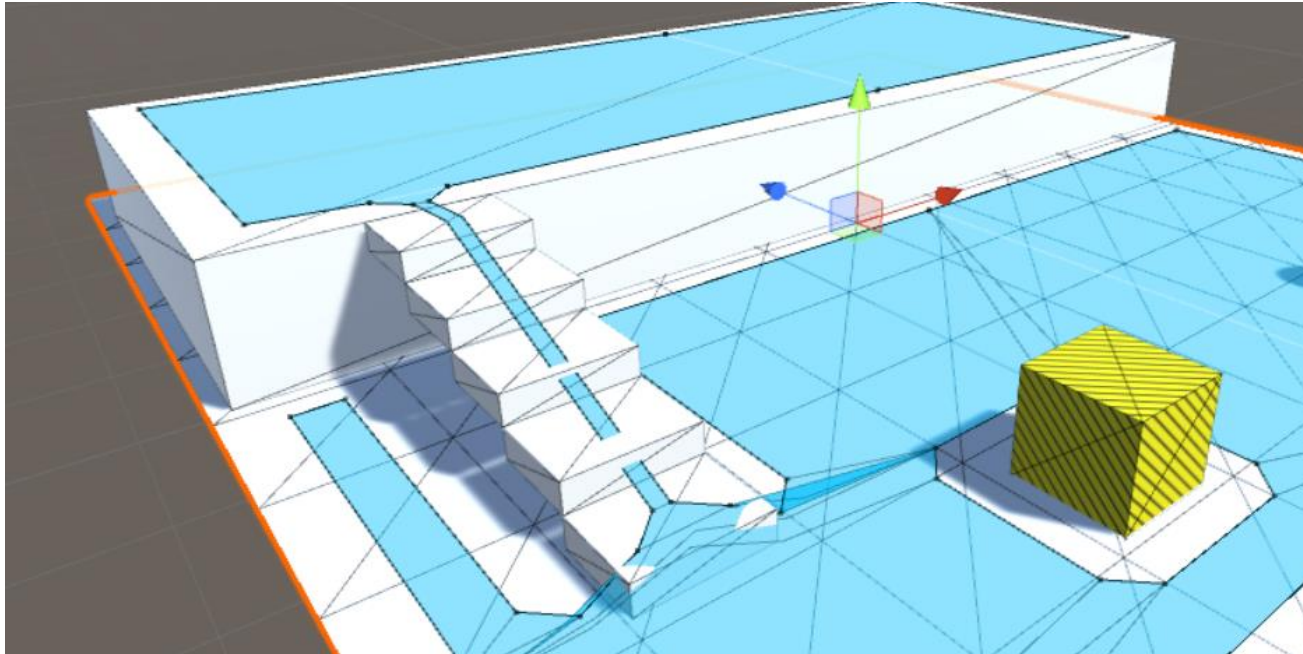
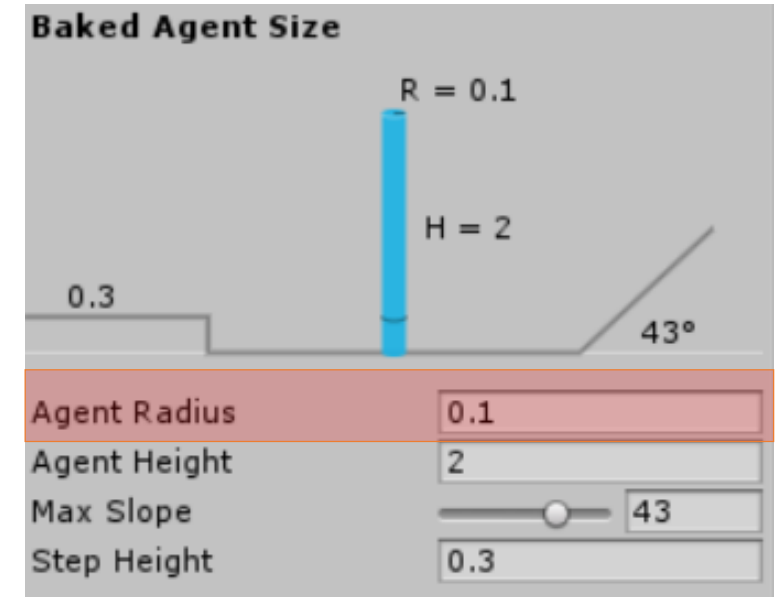
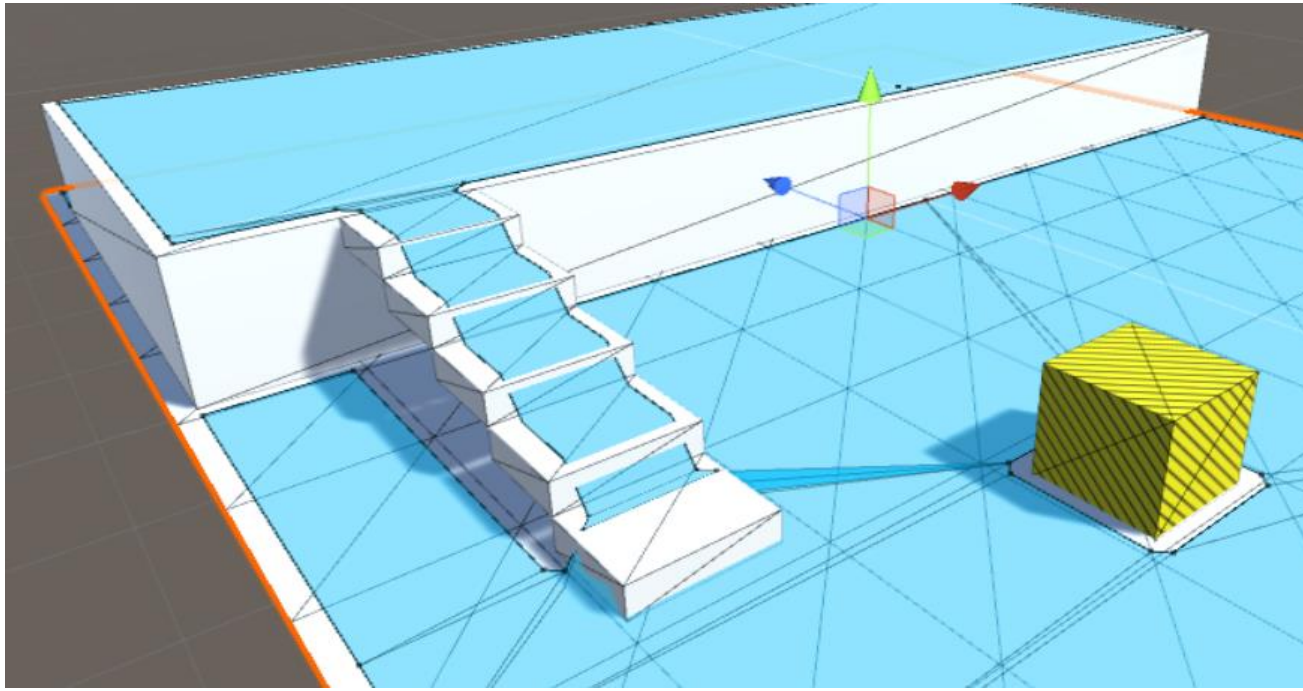
► Advanced

Clear Bake









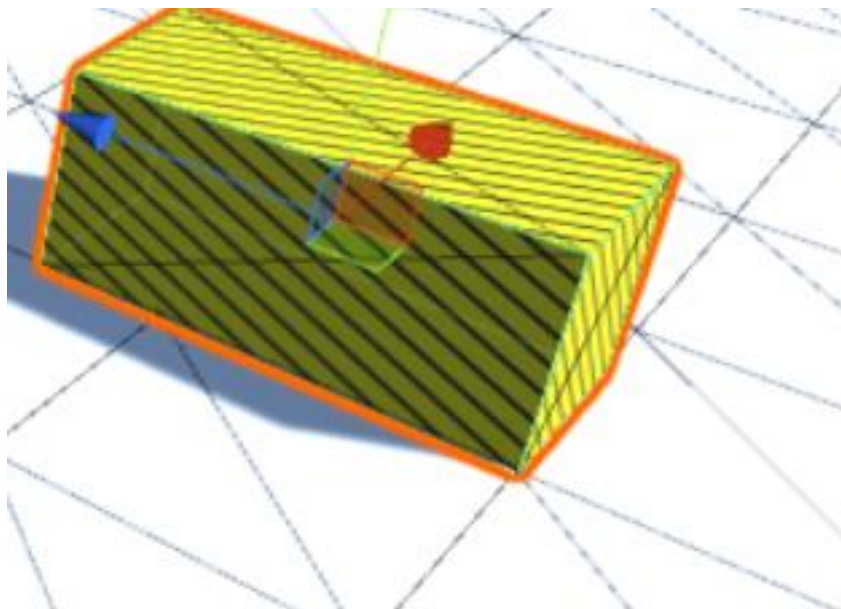
```
// MoveTo.cs
using UnityEngine;
using System.Collections;




public class MoveTo : MonoBehaviour {

    public Transform goal;

    void Start () {
        NavMeshAgent agent = GetComponent<NavMeshAgent>();
        agent.destination = goal.position;
    }
}
```





▼  ☒ **Nav Mesh Obstacle**  

Shape

Center  
X  Y  Z

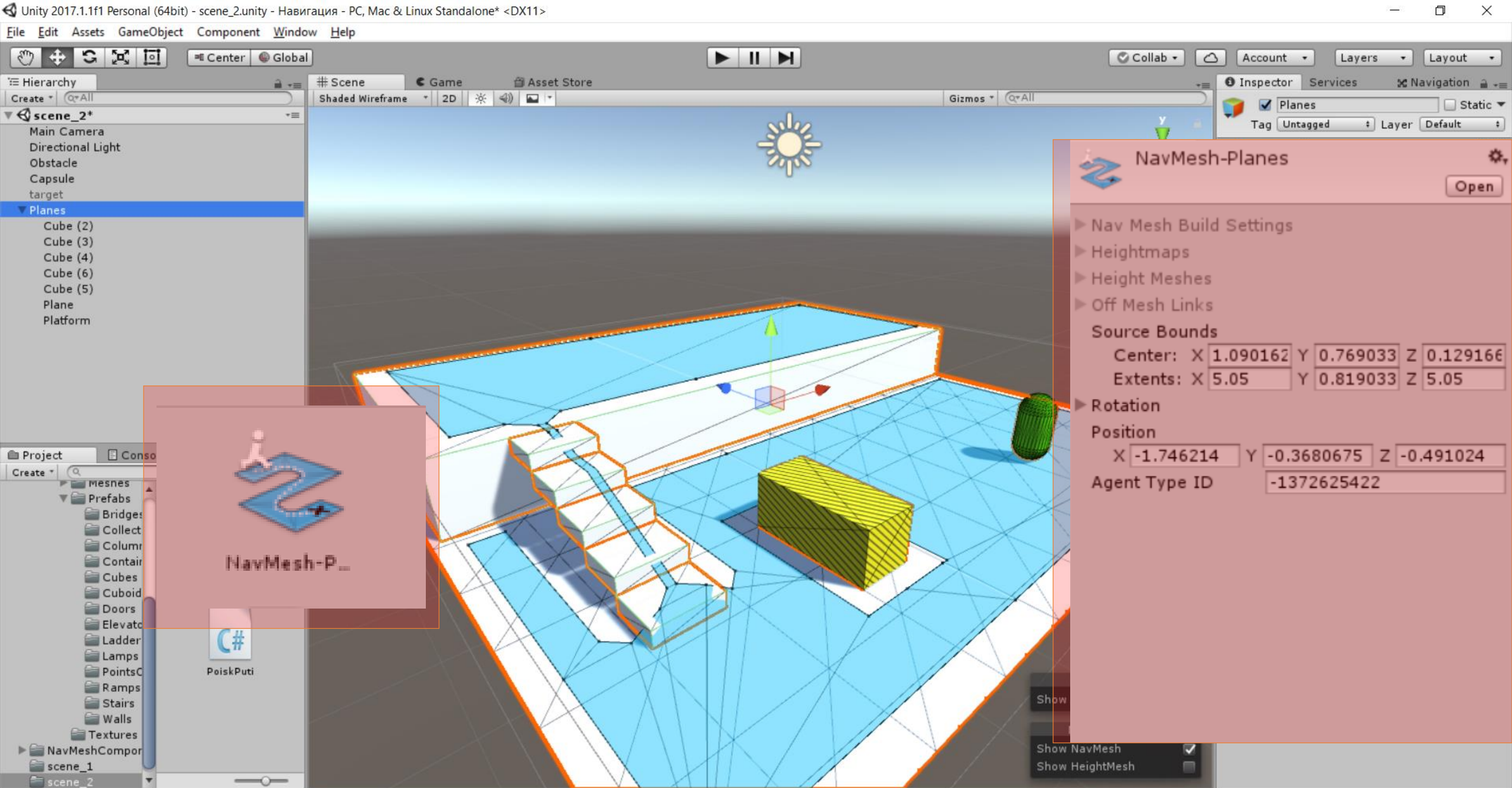
Size  
X  Y  Z

Carve ☒

Move Threshold

Time To Stationar

Carve Only Statio ☒



```

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))
        {

```

```

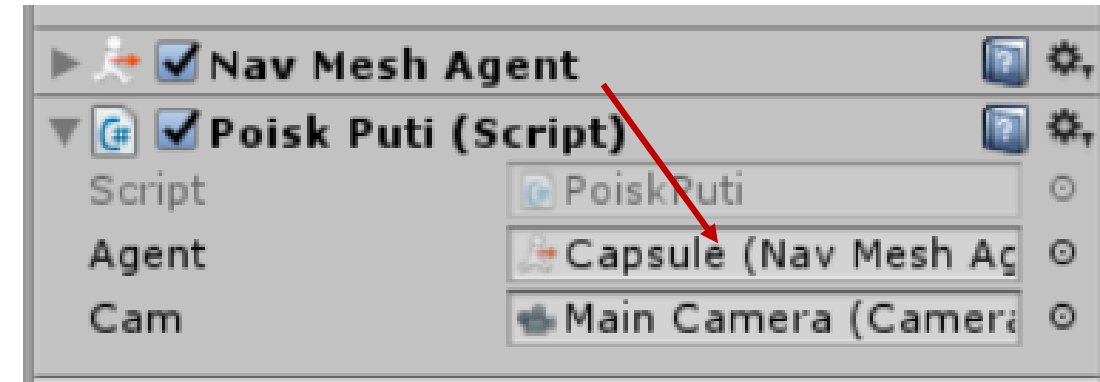
            Ray ray = cam.ScreenPointToRay (Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast (ray, out hit)) {
                agent.SetDestination (hit.point);
            }

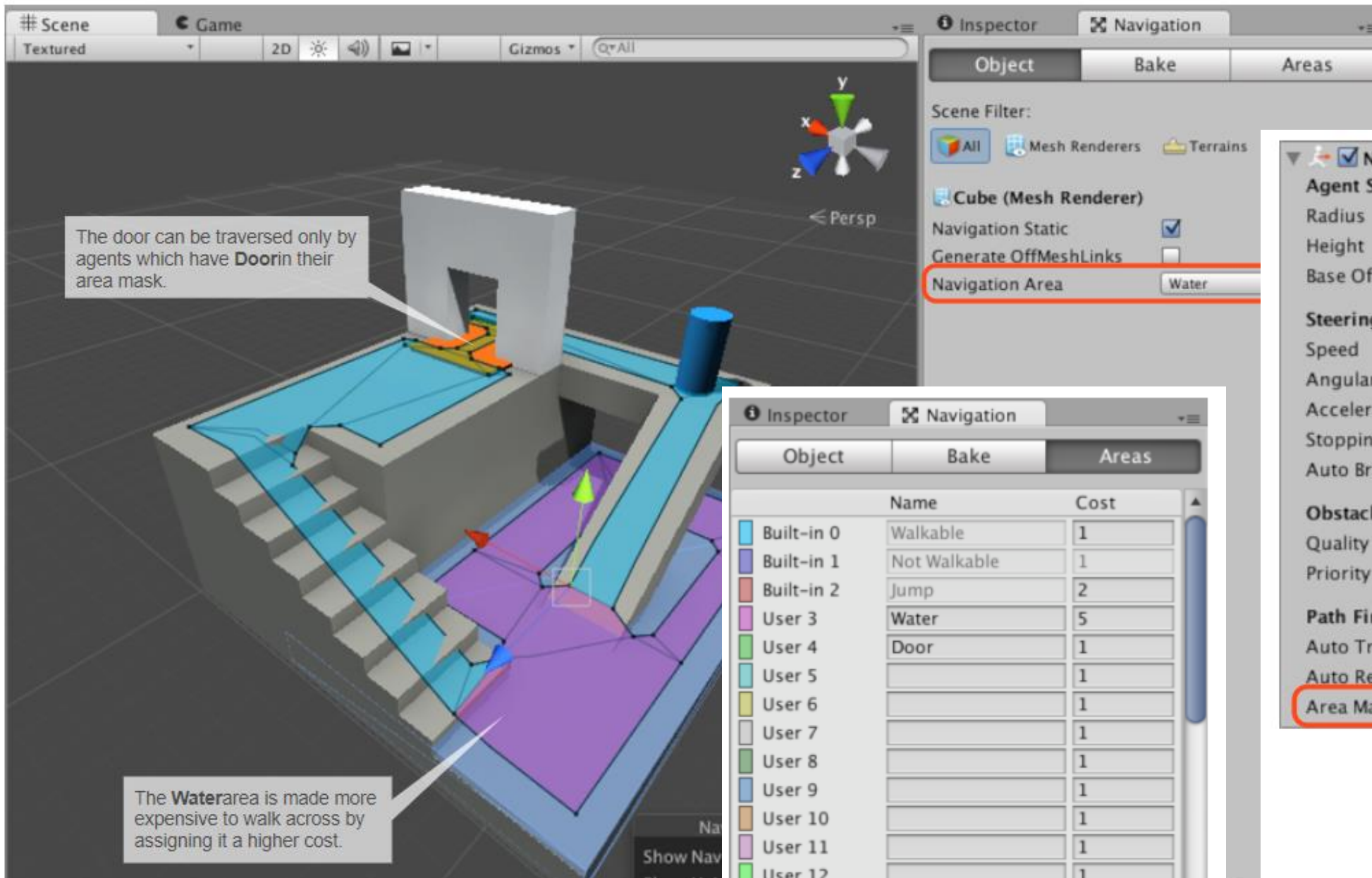
```

```

        }
    }
}

```







# Неуничтожаемые объекты

По умолчанию Unity рассматривает каждый объект, как существующий в пространстве и времени только одной активной сцены. Разница между сценами подобна разнице между отдельными вселенными. Как следствие, объекты не могут существовать вне сцены, которой они принадлежат, то есть они уничтожаются при смене активной сцены.

Решение функция

**DontDestroyOnLoad()**

```
void Start()
{
    // Сделать объект неуничтожаемым
    DontDestroyOnLoad(gameObject);
}
```

Но возникает проблема дублирования объектов. Если вернуться к исходной сцене, где был создан неуничтожаемый объект, сценарий создаст его копию.

Решение применить **singleton**

# Лучи из камеры

Любая точка в поле зрения камеры соответствует линии в мировом пространстве.



# Лучи из камеры

Любая точка в поле зрения камеры соответствует линии в мировом пространстве. Иногда полезно иметь математическое представление этой линии и Unity может предоставить его в виде объекта Ray (луч).

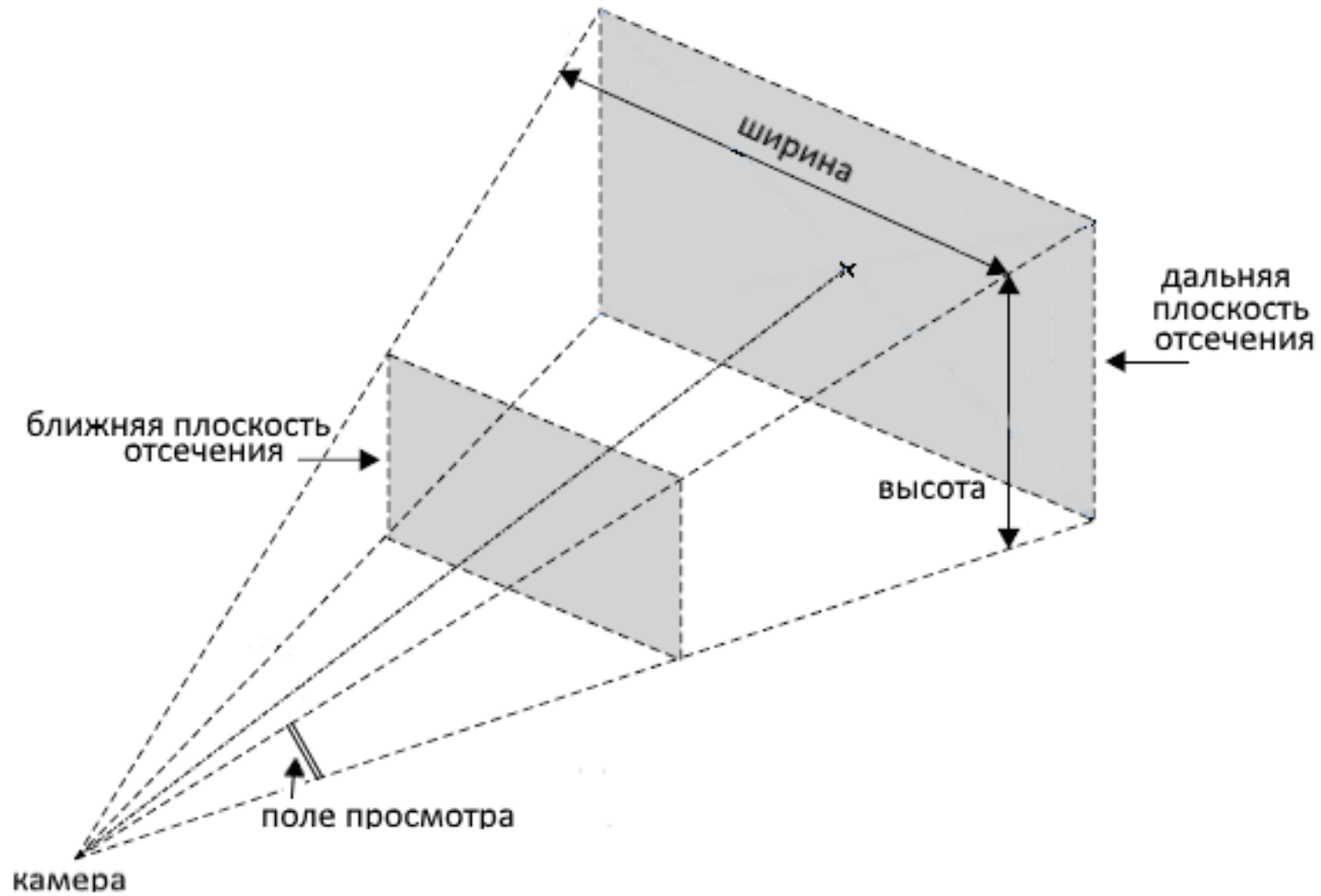
Класс Camera предоставляет методы **ScreenPointToRay** и **ViewportPointToRay**.

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.

# Область видимости камеры



```
        // Draws a line in the scene view going through a point 200 pixels
        // from the lower-left corner of the screen
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    Camera camera;

    void Start() {
        camera = GetComponent<Camera>();
    }

    void Update() {
        Ray ray = camera.ScreenPointToRay(new Vector3(200, 200, 0));
        Debug.DrawRay(ray.origin, ray.direction * 10, Color.yellow);
    }
}
```

Что позволяет определить данный код ?

```
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;
        }
    }
}
```

# Дополнительные материалы для изучения

## Использование Raycast

<https://www.youtube.com/watch?v=OCIXUXBnzlg>

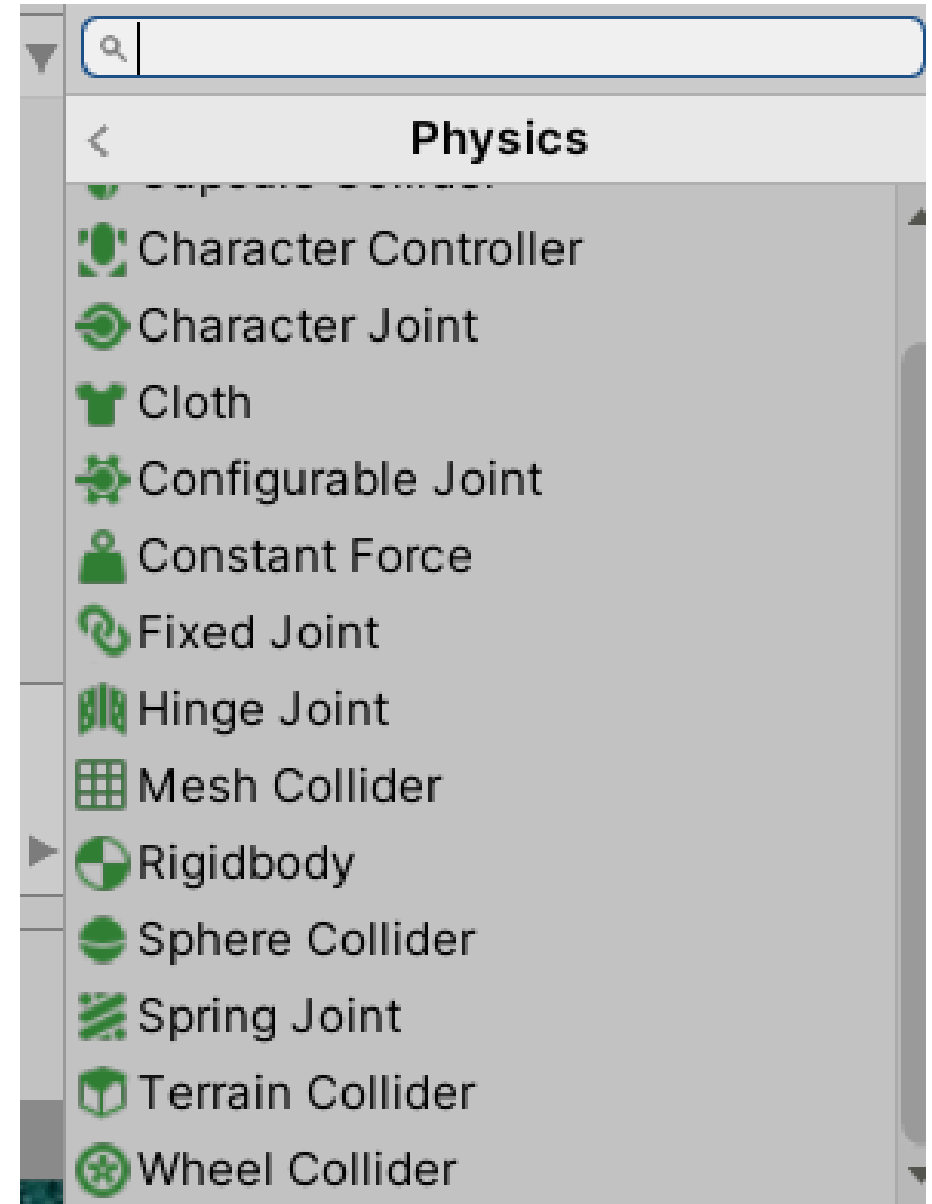
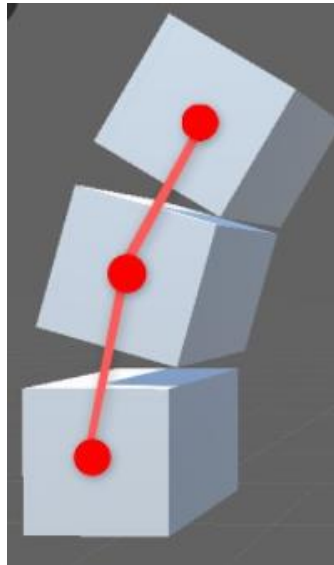
## Стрельба с помощью Raycast

[https://www.youtube.com/watch?v=1Sc\\_Ad3DESs](https://www.youtube.com/watch?v=1Sc_Ad3DESs)

# Joints

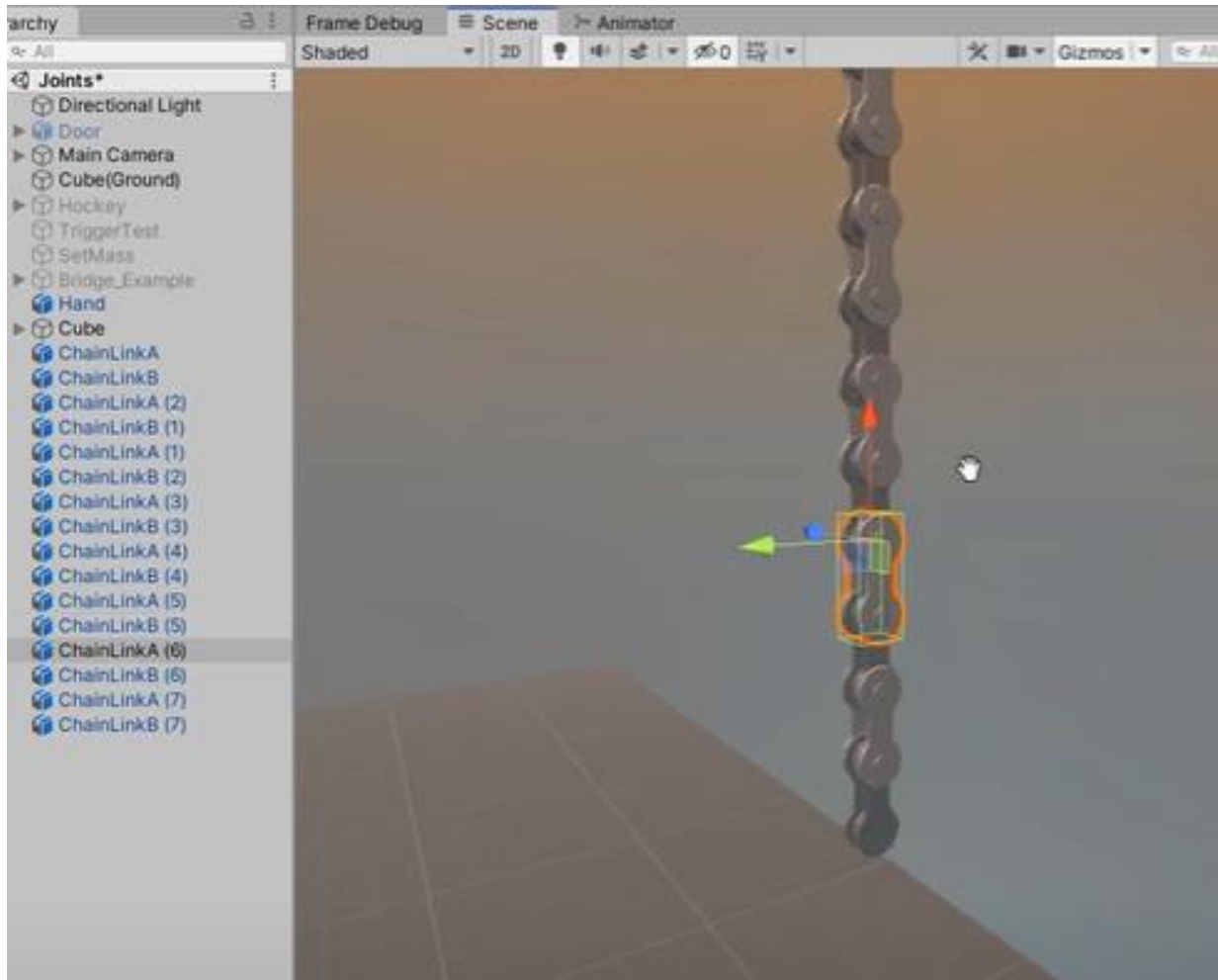
Компонент **Joint** соединяет Rigidbody с другим Rigidbody или фиксированной точкой в пространстве.

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>

