



箱庭とUnityで組み立てる オレオレロボットのはじめかた



森 崇

(永和システムマネジメント)

4月
17

箱庭チュートリアル会 #3 箱庭とUnityで組み立てるオ レオレロボットのはじめかた

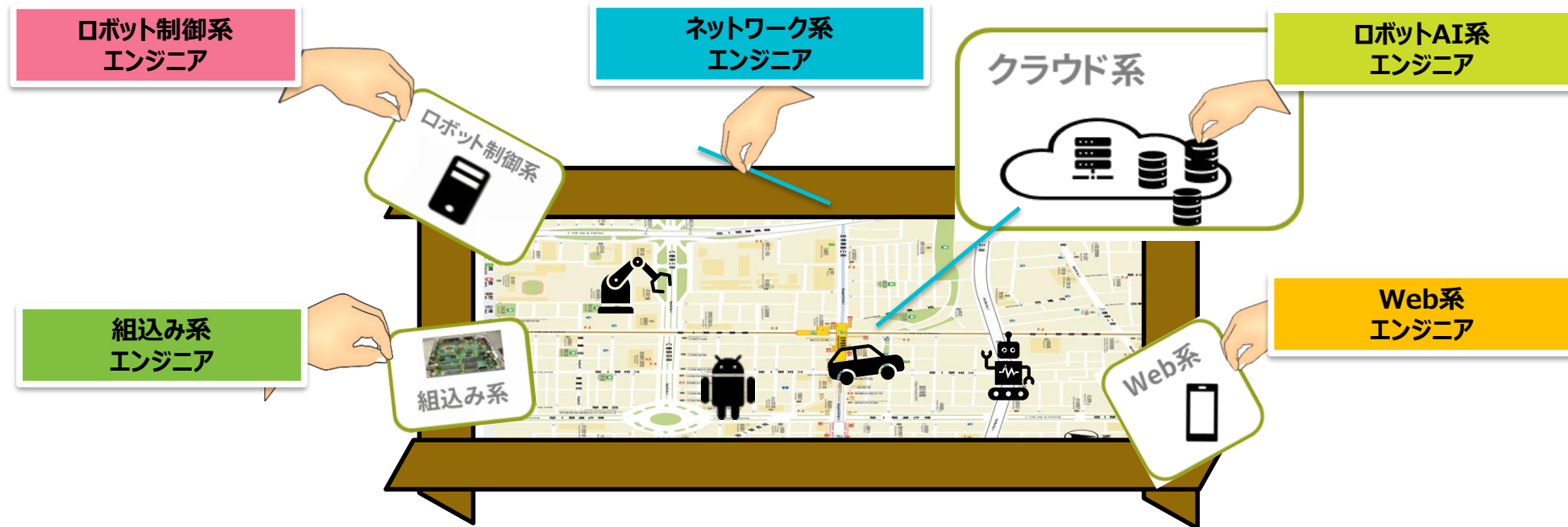
箱庭で1からロボットを作ってみよう！



1. 箱庭とは何か
2. Unityとは何か
3. Unity上のロボットの構造
4. 箱庭機能との接合部分
5. 箱庭があると何が嬉しくなるのか
6. 箱庭ロボットの作り方

箱庭とはなにか

- 箱の中に, **様々なモノ**を**みんなの好み**で配置して, いろいろ試せる!
 - 仮想環境上(**箱庭**)でIoT/ロボット・システムを開発する
- ⇒ 各分野のソフトウェアを持ち寄って, 机上で全体結合&実証実験!



箱庭チュートリアル会の背景



2019-2021

公開中

箱庭プロトタイプモデル開発

2022-2024

箱庭チュートリアル

- 組込み系エンジニア
- ロボット制御系エンジニア
- ロボットAI系エンジニア

箱庭ユーザを広める活動

結合環境構築 (箱庭ベース環境)

箱庭チュートリアル開発 && 広報活動

組込み系エンジニア

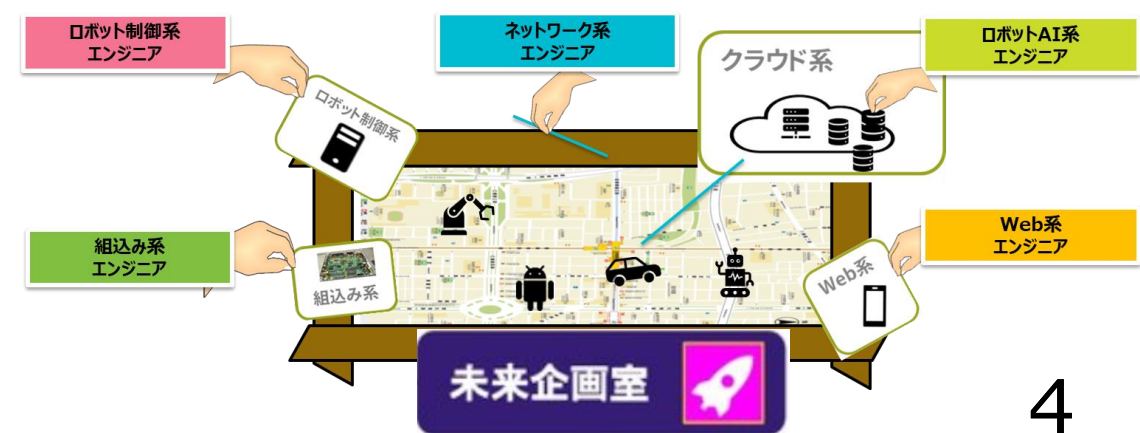
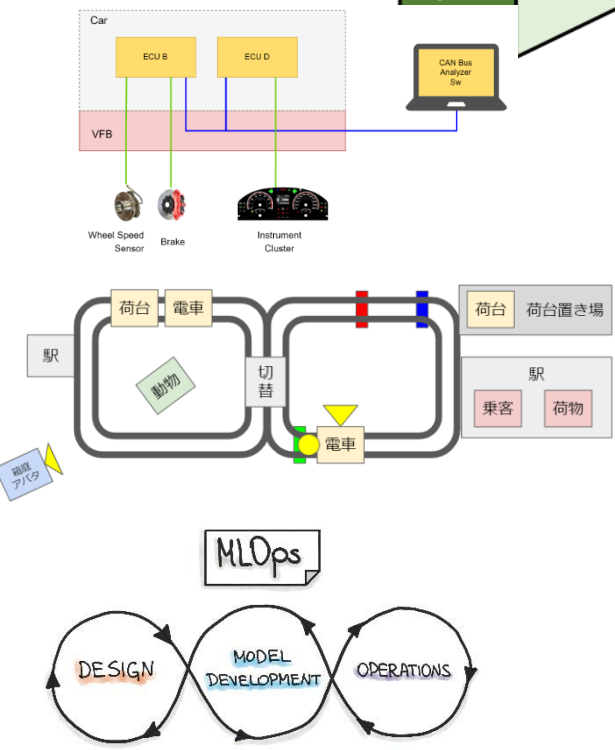
ECU/AUTOSAR開発入門シミュレーション環境

ロボット制御系エンジニア

ロボット制御モデリング/プログラミング入門シミュレーション環境

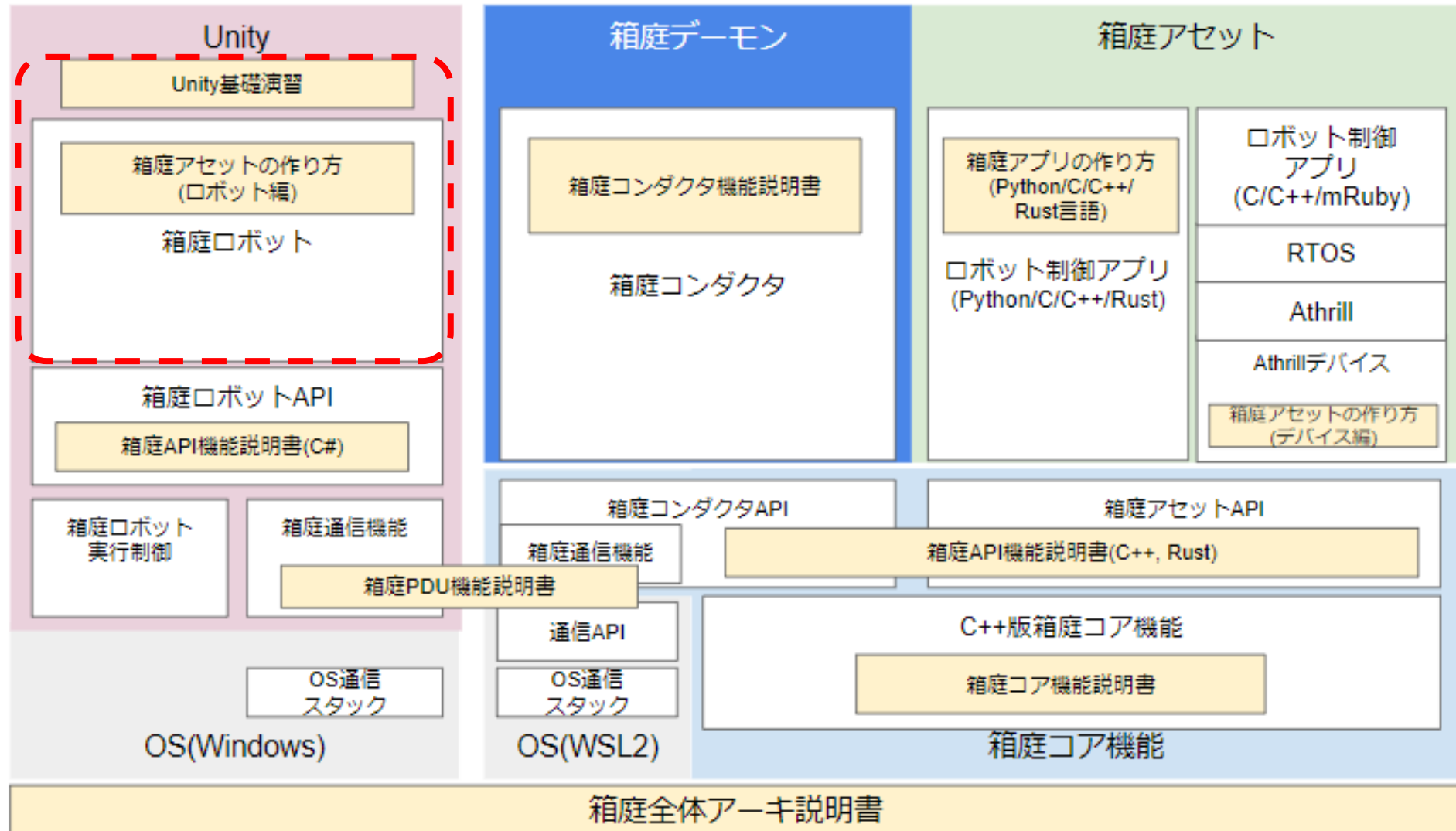
ロボットAI系エンジニア

ロボット制御学習サイクル自動化入門シミュレーション環境





現状の箱庭の全体像 (Windows版)

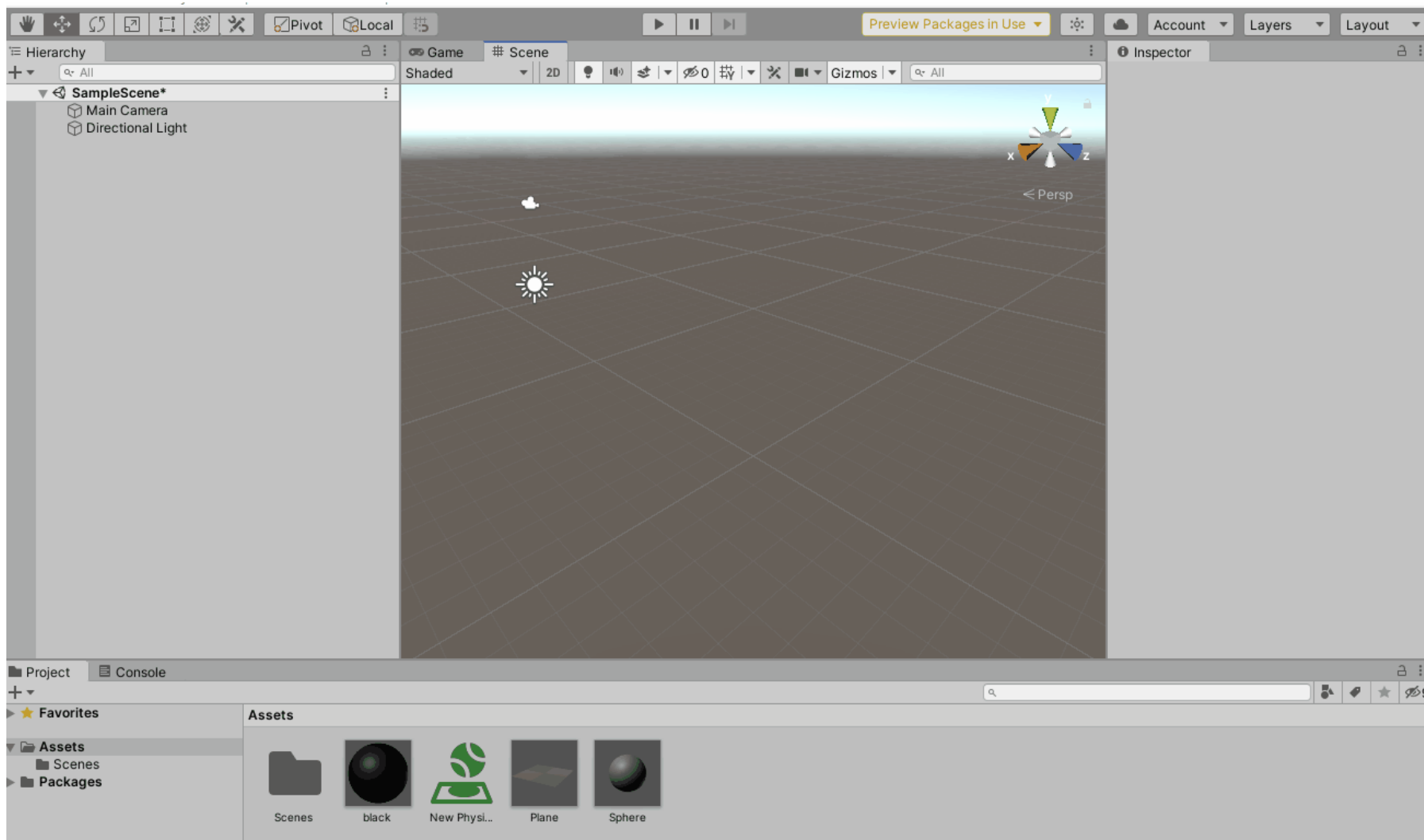




Unityとは何か

- ユニティ・テクノロジーズ社が開発したゲーム開発プラットフォーム
 - 物理演算エンジンが搭載されています
 - 質量・速度・摩擦・風といった、古典力学的な法則をシミュレーションするコンピュータのソフトウェア
 - ※参照元：ウィキペディア
- Unityでできること
 - ロボット作れます
 - 物理的な法則に従ってモノを動かすことができます
 - ロボットの周辺環境も作れます

ボールを置いて、物理シミュレーションを実行するまで⁷



ロボットも動かすことができる

8





Unity上のロボットの構造解説

- Unityの操作
 - Unityエディタの画面構成
 - 基本的な操作方法
- Unityで物理シミュレーションするための基礎知識
- ロボットを動かすためのUnityの構造



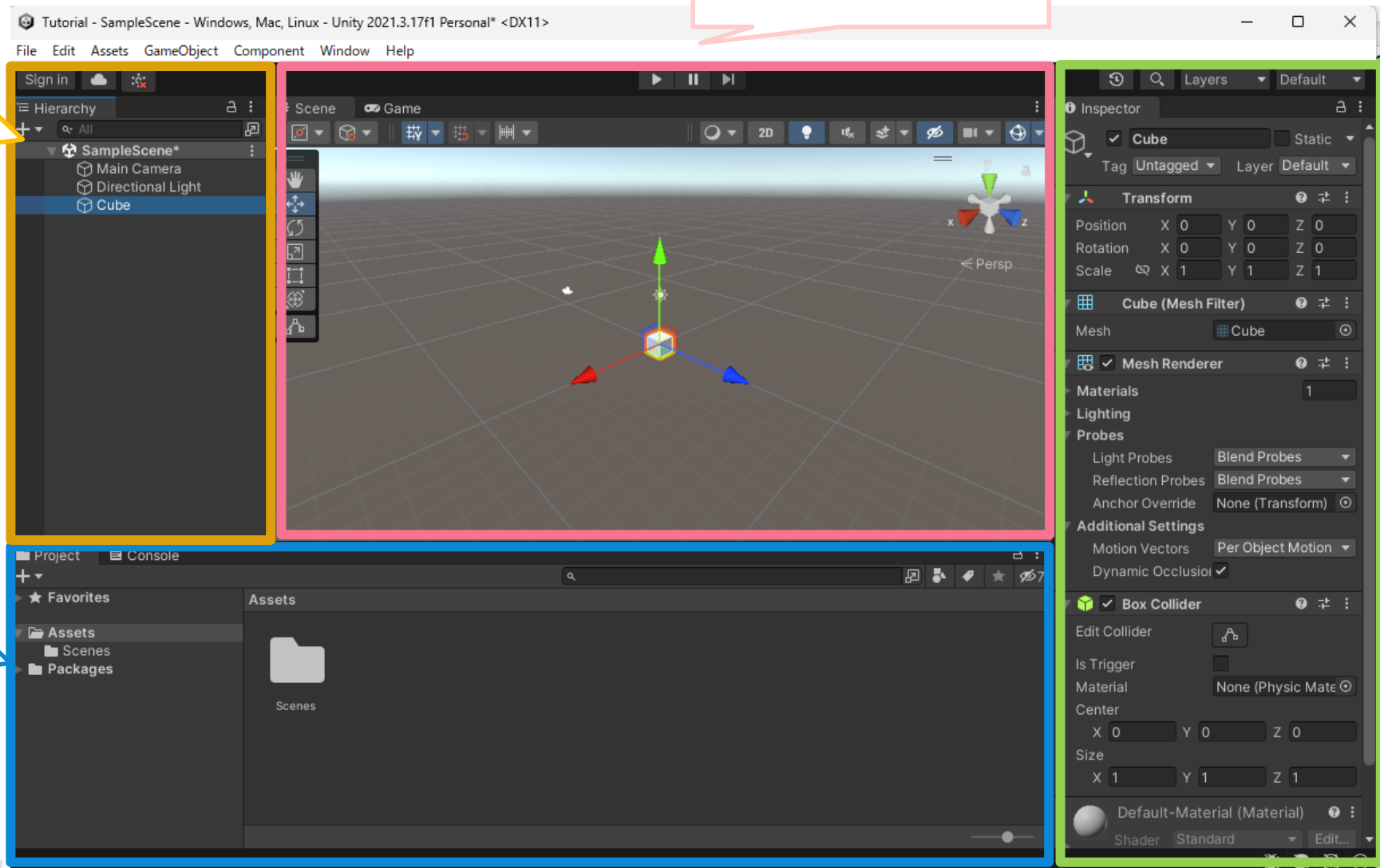
Unityエディタの画面構成 (1 / 2)

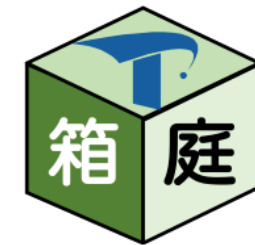
Sceneビュー

Hierarchy
ビュー

Inspector
ビュー

Project
ビュー





Unityエディタの画面構成 (2 / 2)

Gameビュー
Main Cameraを通して移した世界



Console
ビュー



基本的な操作方法

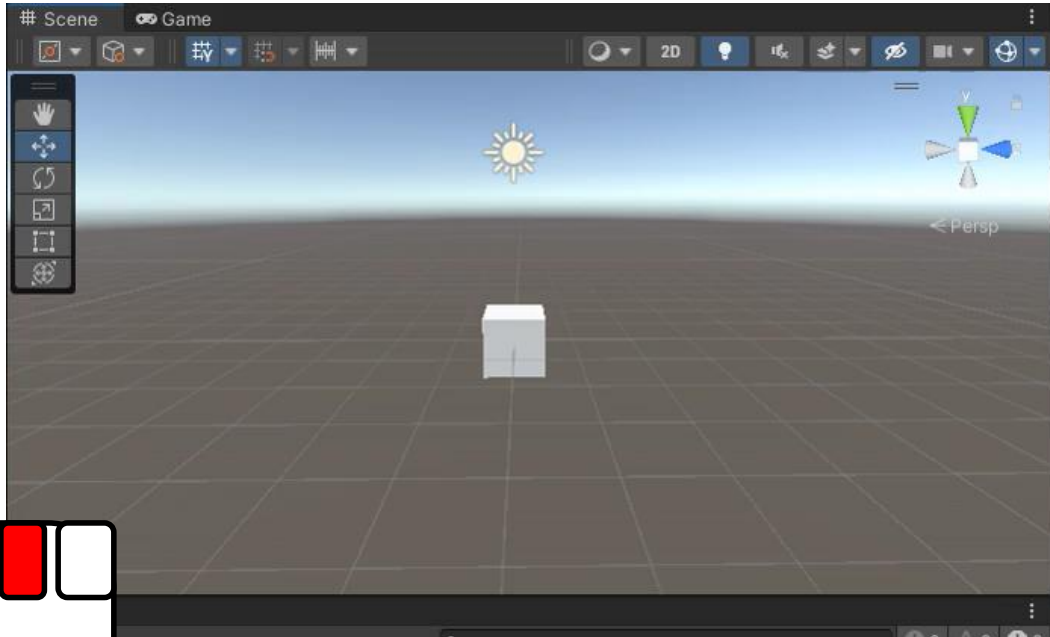
- 視点の操作
 - 視点移動
 - 視点回転
 - ズーム操作
- ゲーム・オブジェクトの操作
 - ゲーム・オブジェクト移動
 - ゲーム・オブジェクト回転
 - ゲーム・オブジェクト拡大・縮小

視点移動

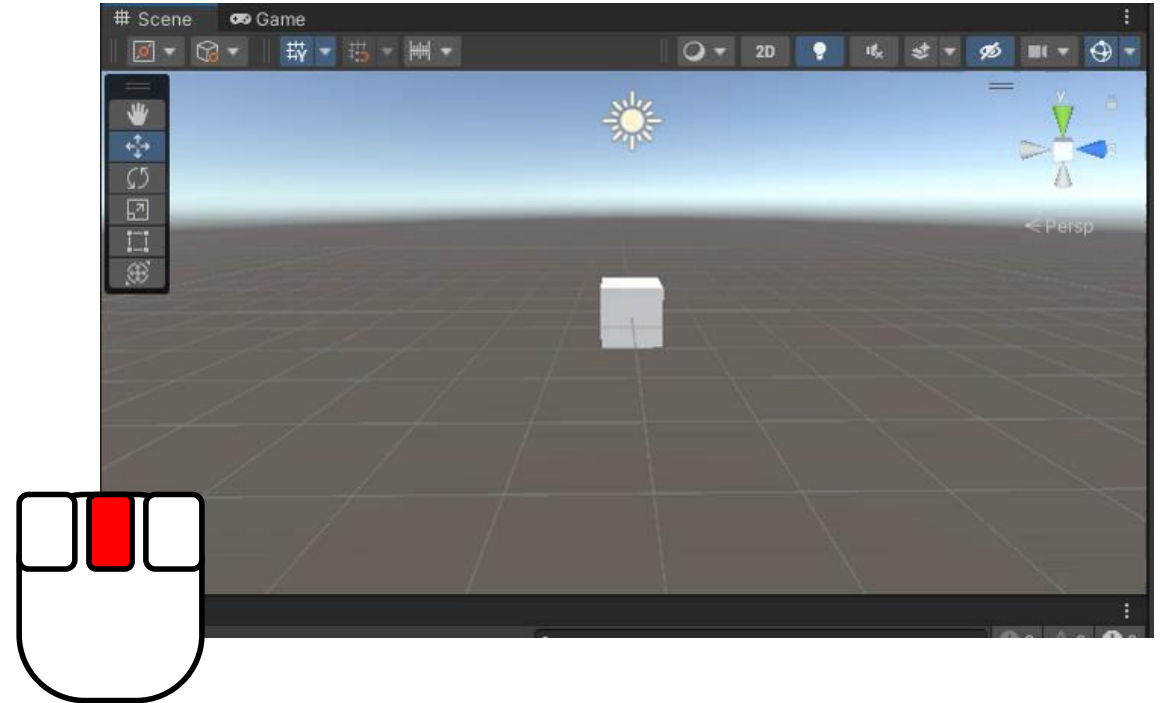
13



マウス・ホイール押しながら右左



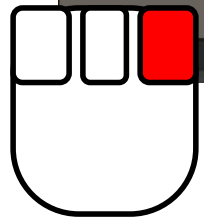
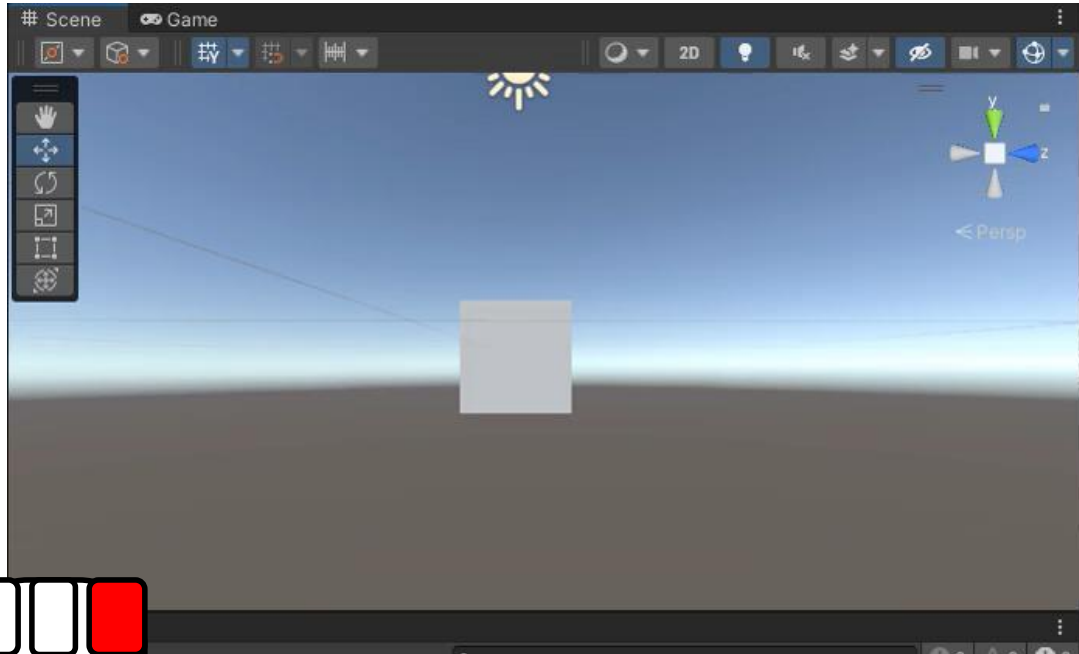
マウス・ホイール押しながら上下



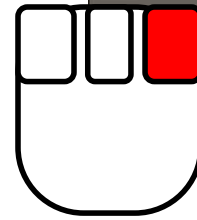
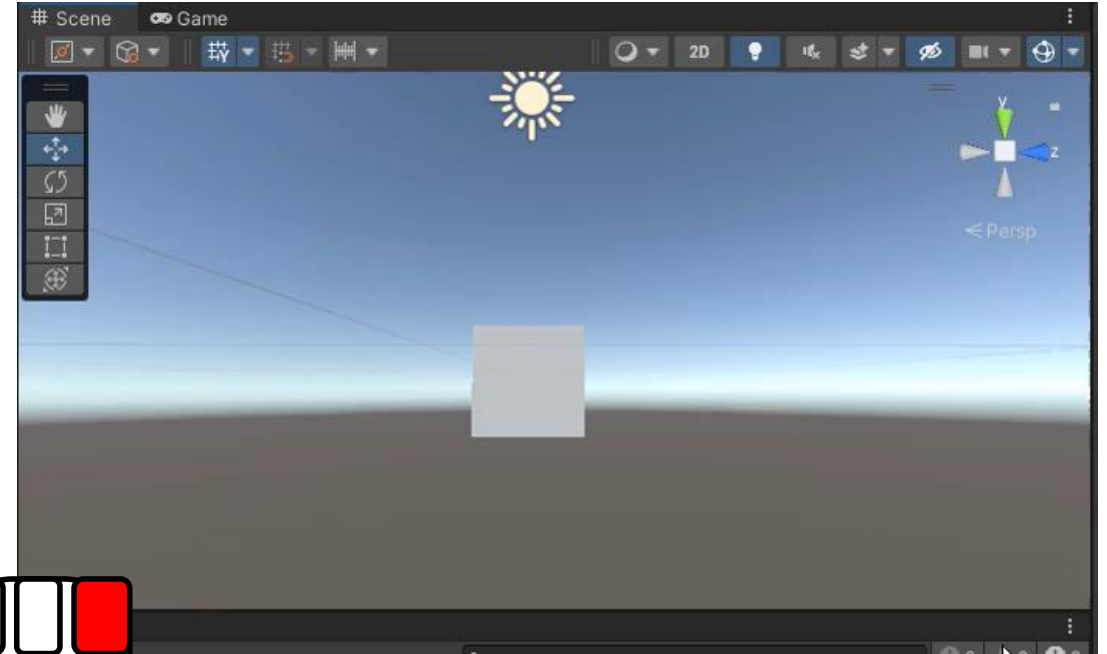


視点回転

右クリックしながら右左



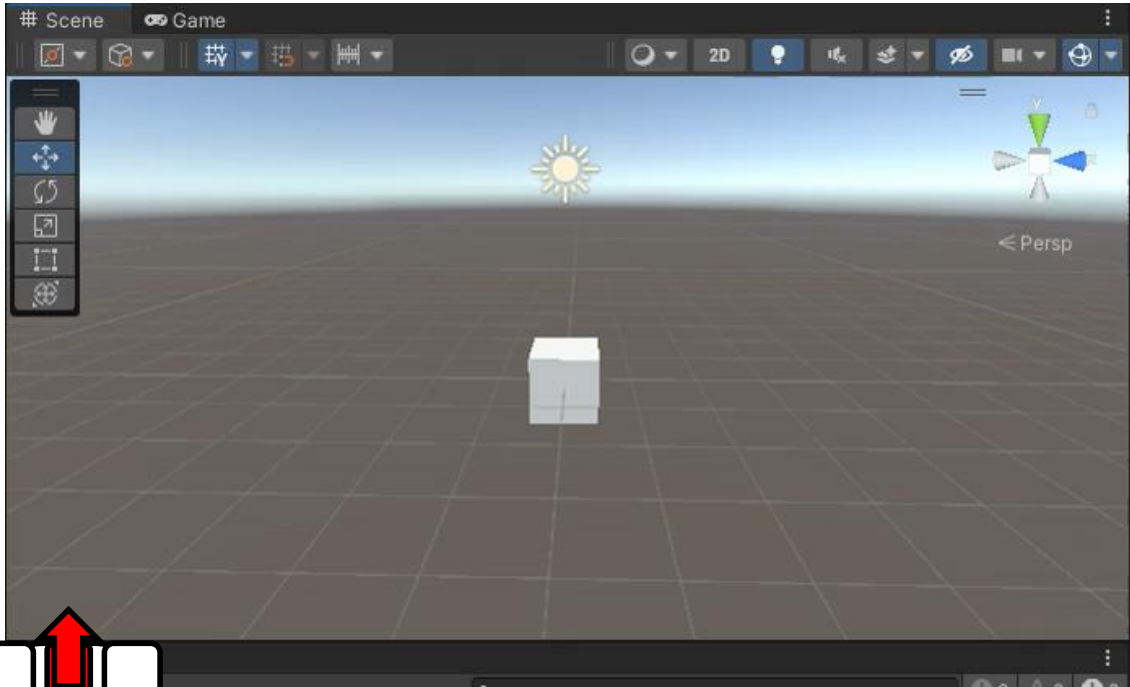
右クリック押しながら上下



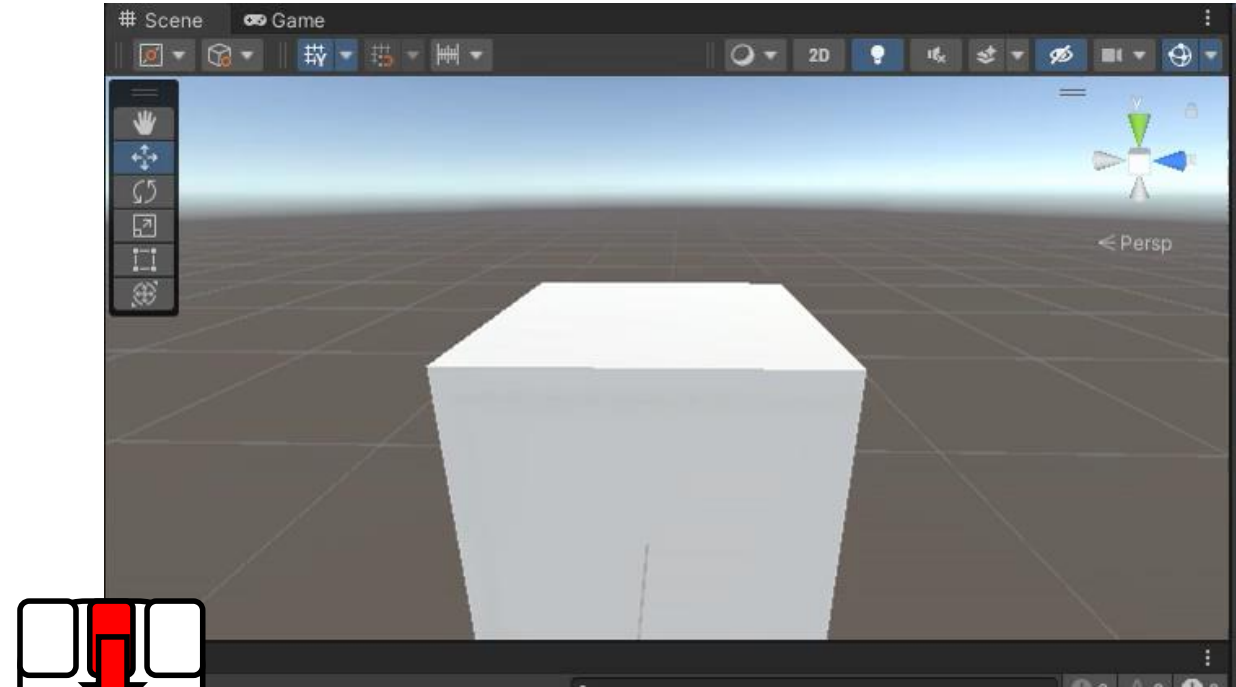
ズーム操作



ズームイン



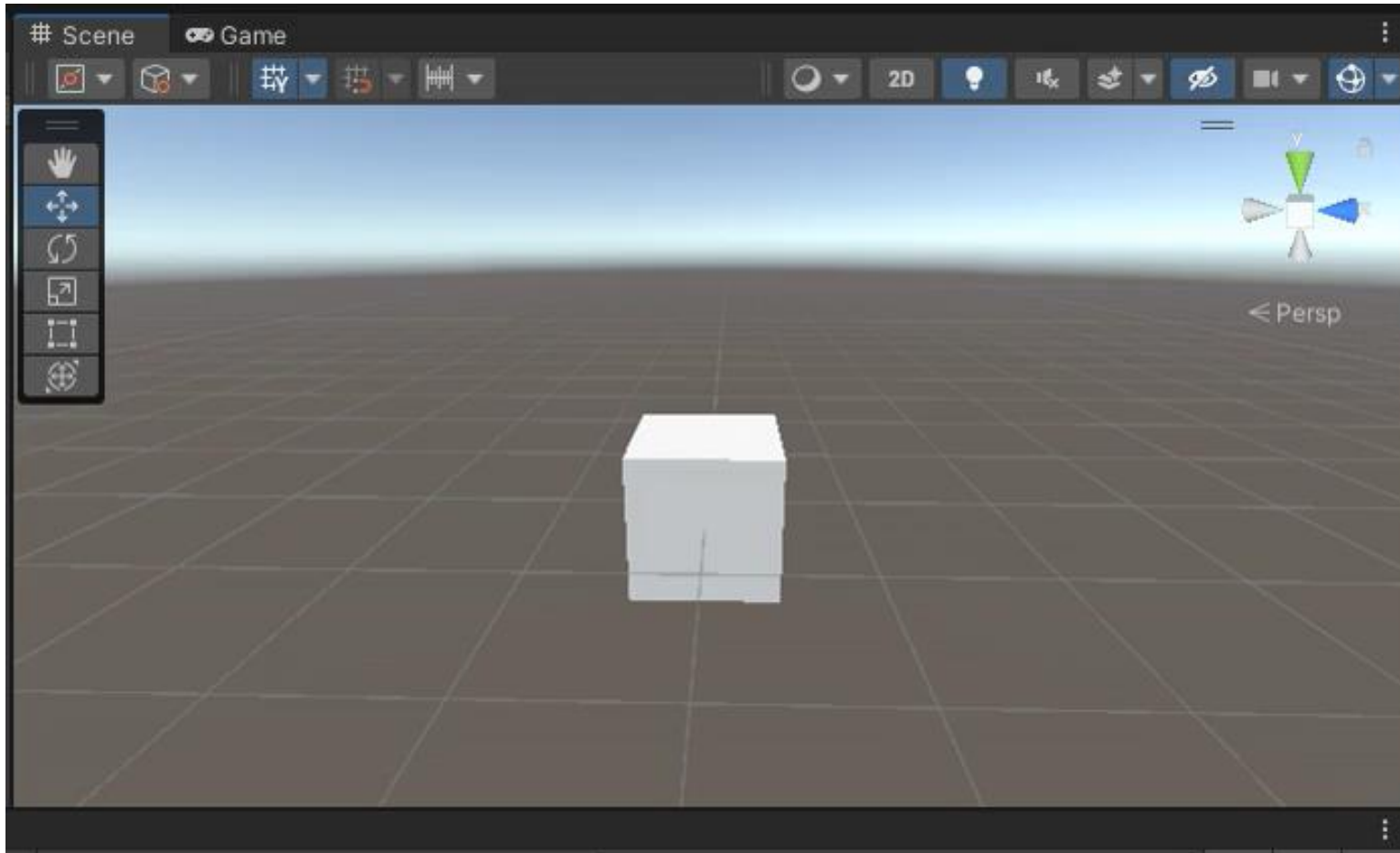
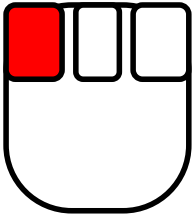
ズームアウト





ゲーム・オブジェクト移動

左クリックしながら移動

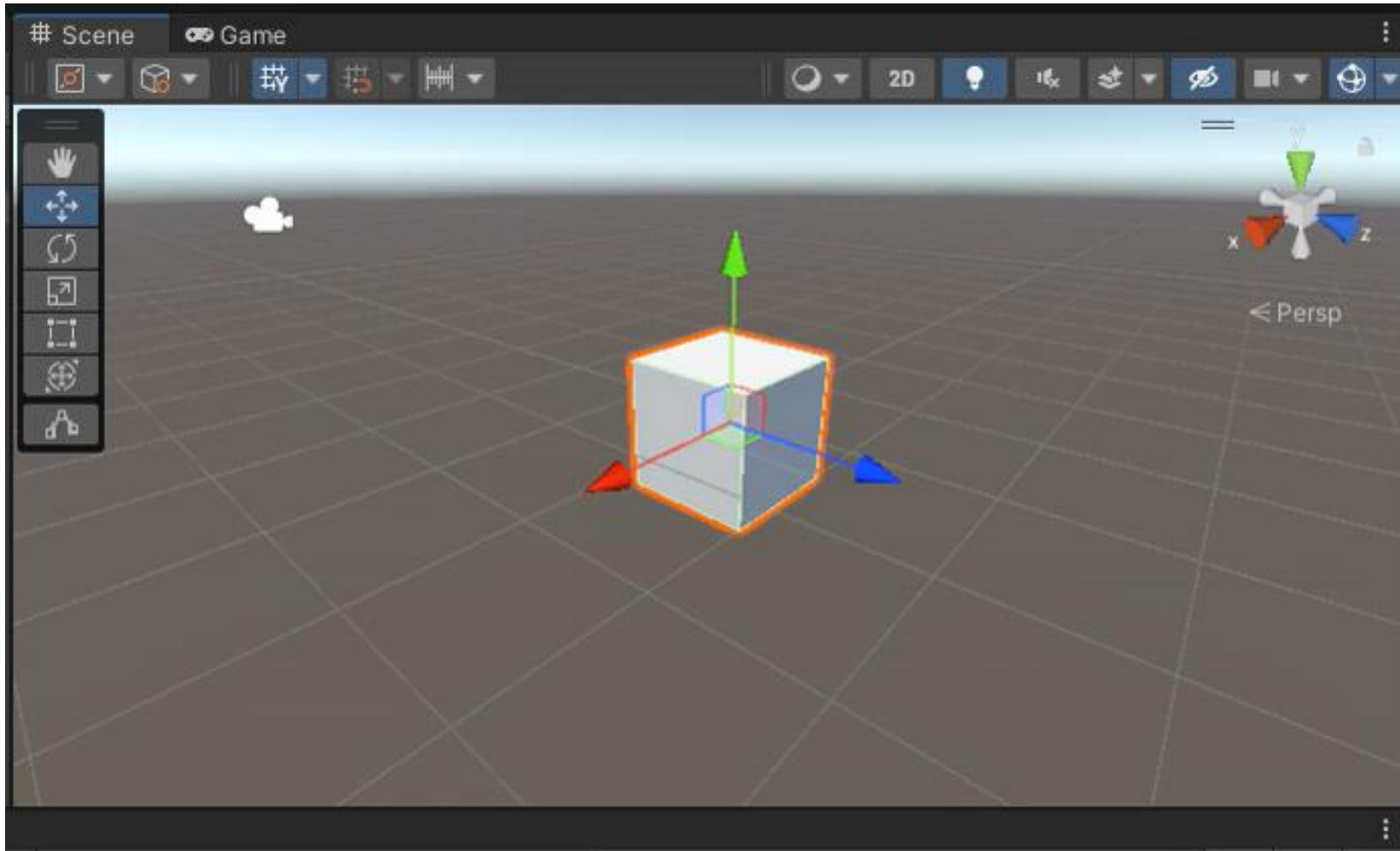
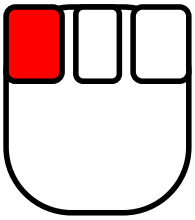


ゲーム・オブジェクト回転

17



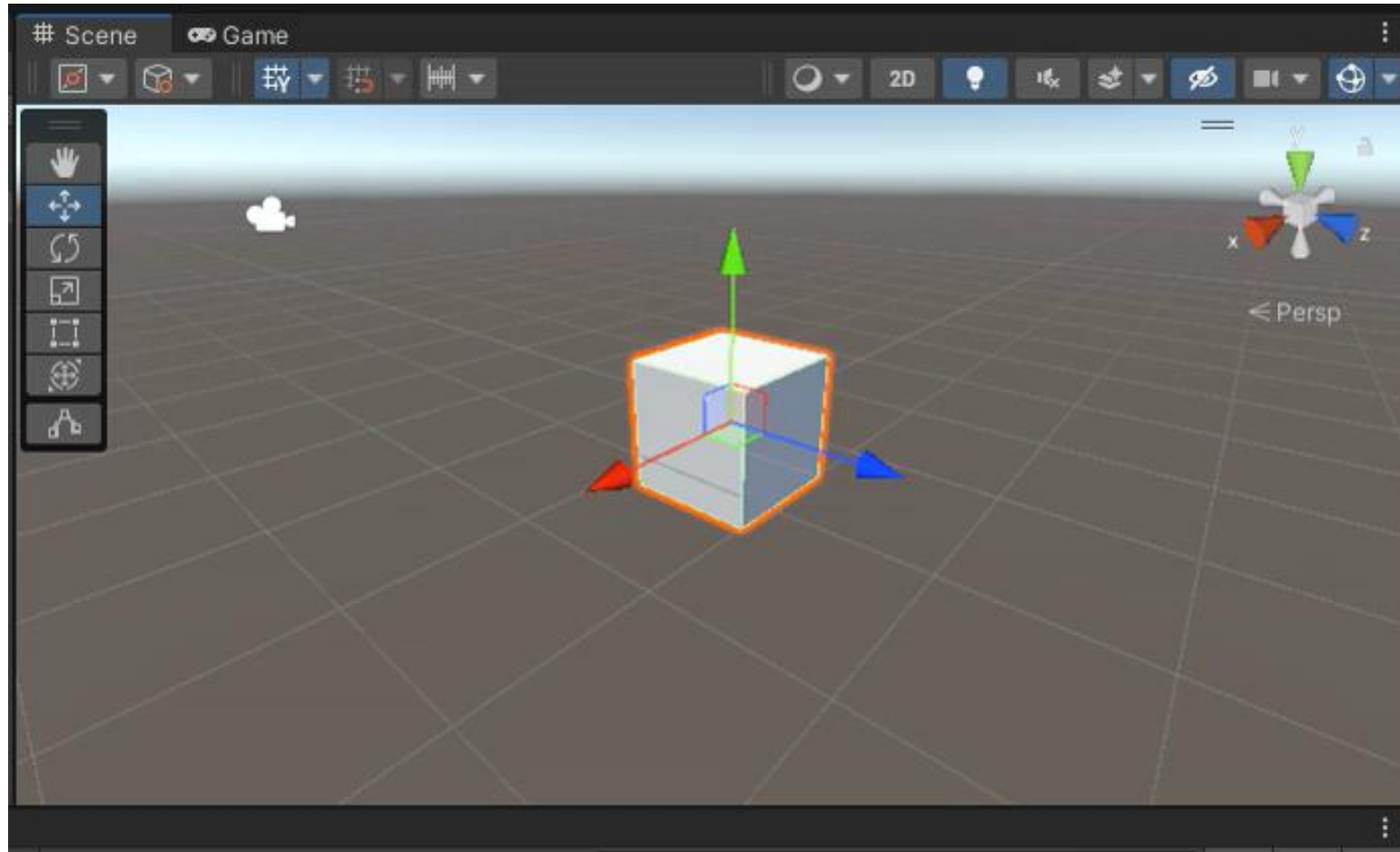
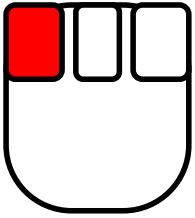
左クリックしながら回転





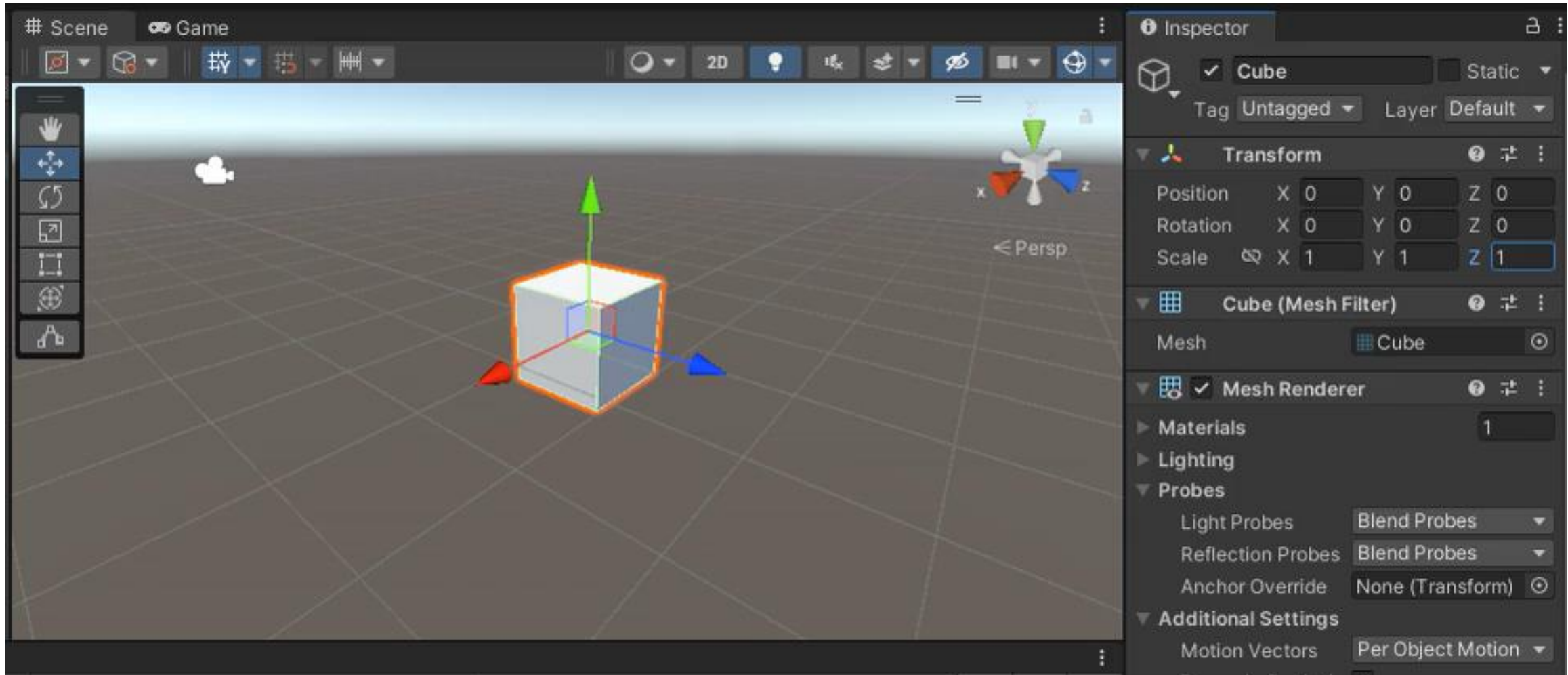
ゲーム・オブジェクト拡大・縮小

左クリックしながら拡大・縮小





Inspectorビューでもできます





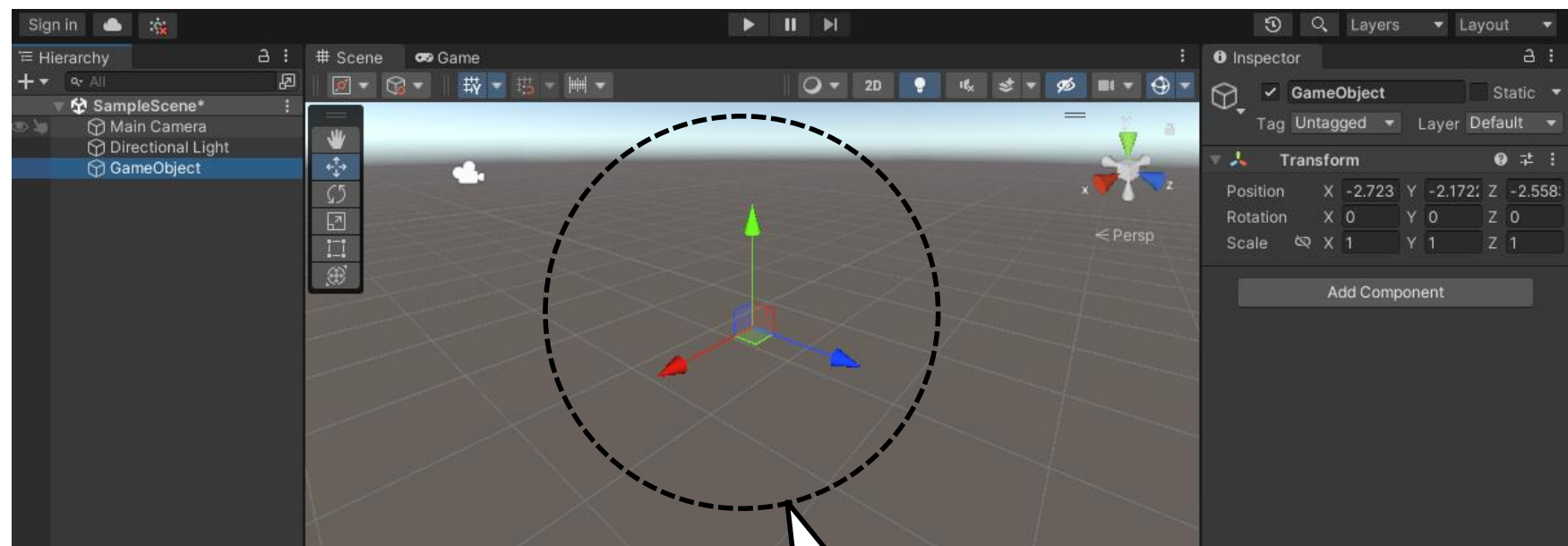
Unityで物理シミュレーションするための基礎知識

- ゲームオブジェクト
- プリミティブな素材
- 衝突判定用のオブジェクト
- 物理挙動
 - Rigidbody
 - Jointコンポーネント
 - ArticulationBody
- 独自処理(Unityスクリプト)の追加方法
- Unityイベント関数の実行順序



ゲームオブジェクト

- Unity上のすべてのキャラやロボットはゲームオブジェクトで出来ています
 - 以下のTransform情報を持ちます
 - 名前
 - 位置(x, y, z)
 - 回転(x, y, z)
 - スケール(x, y, z)

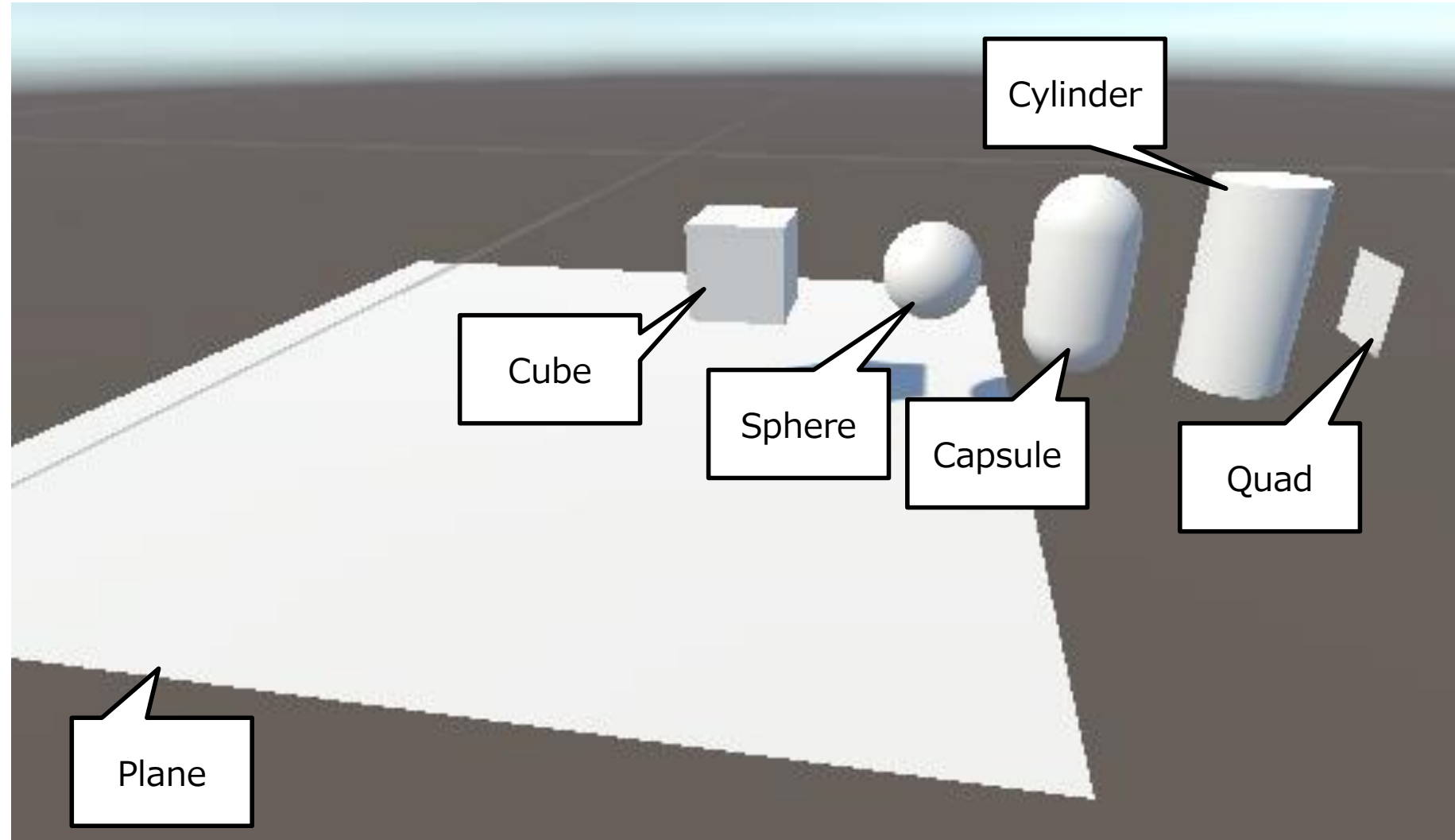


無色、透明です



プリミティブな素材

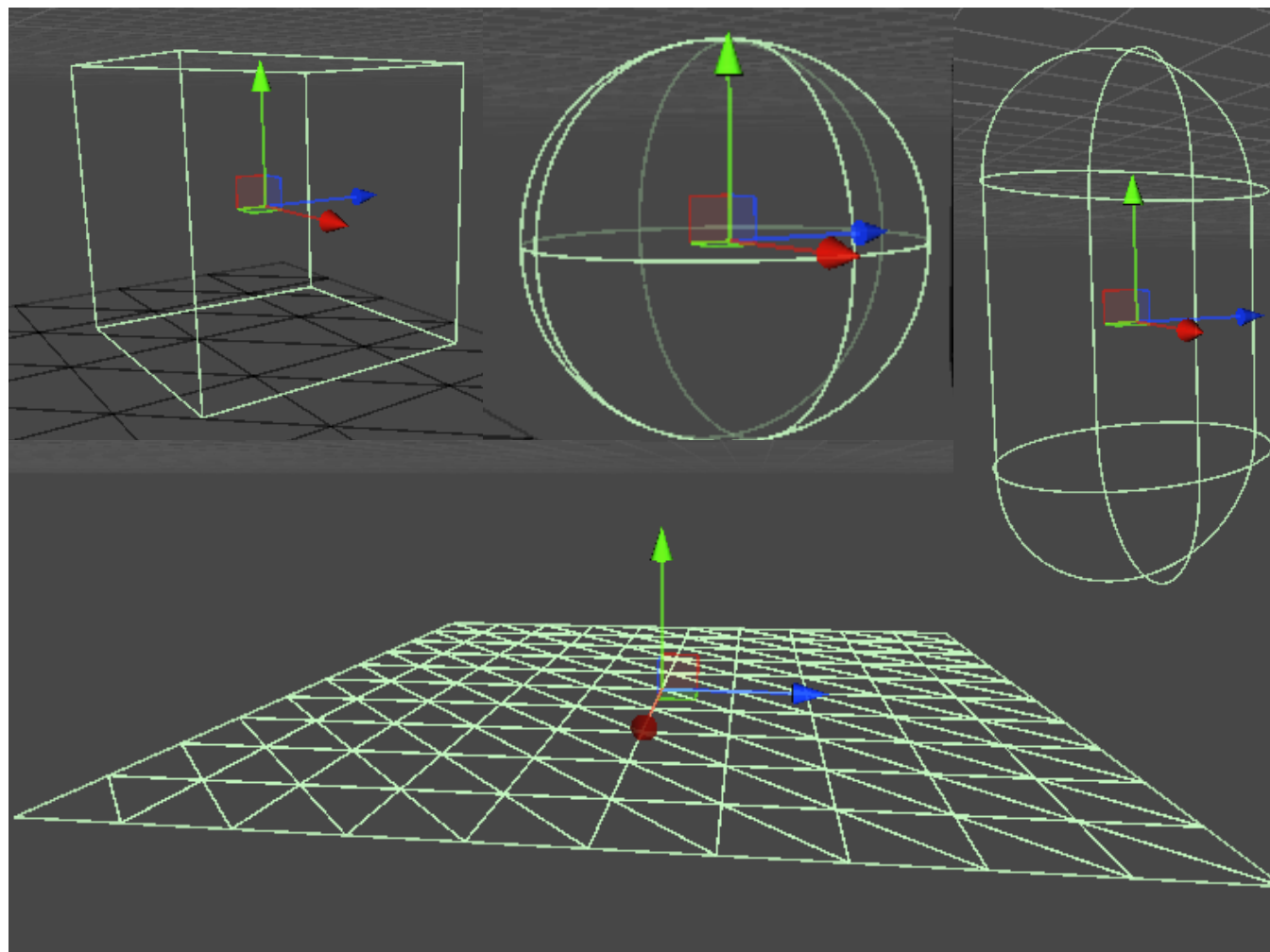
1. Cube
2. Sphere
3. Capsule
4. Cylinder
5. Quad
6. Plane



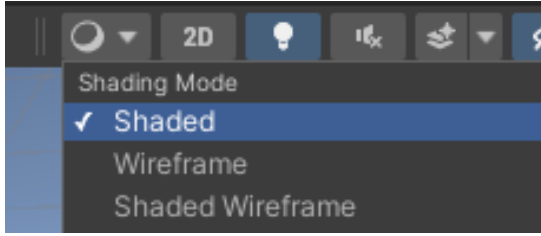


衝突判定用のオブジェクト

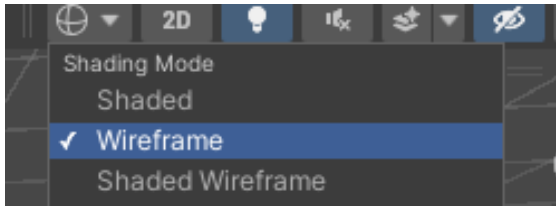
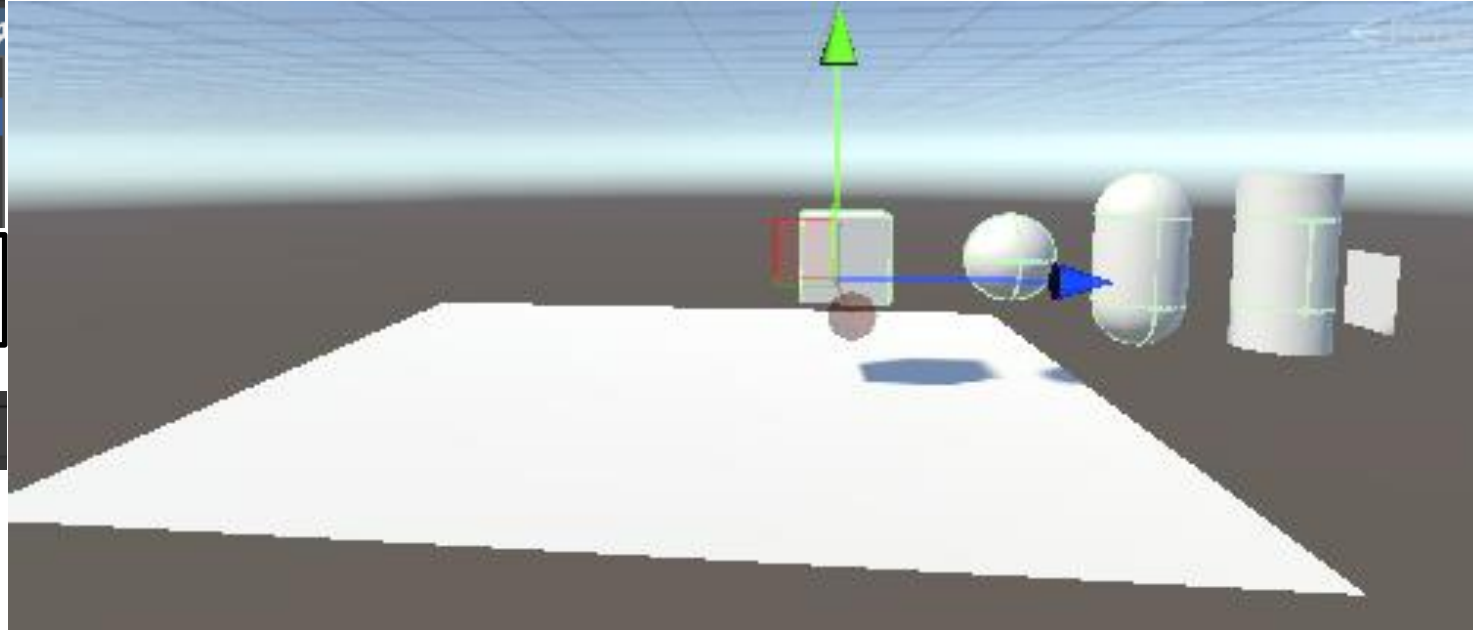
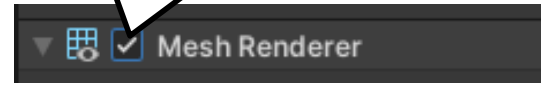
- コライダの種類
 - Boxコライダ
 - Sphereコライダ
 - Capsuleコライダ
 - Meshコライダ



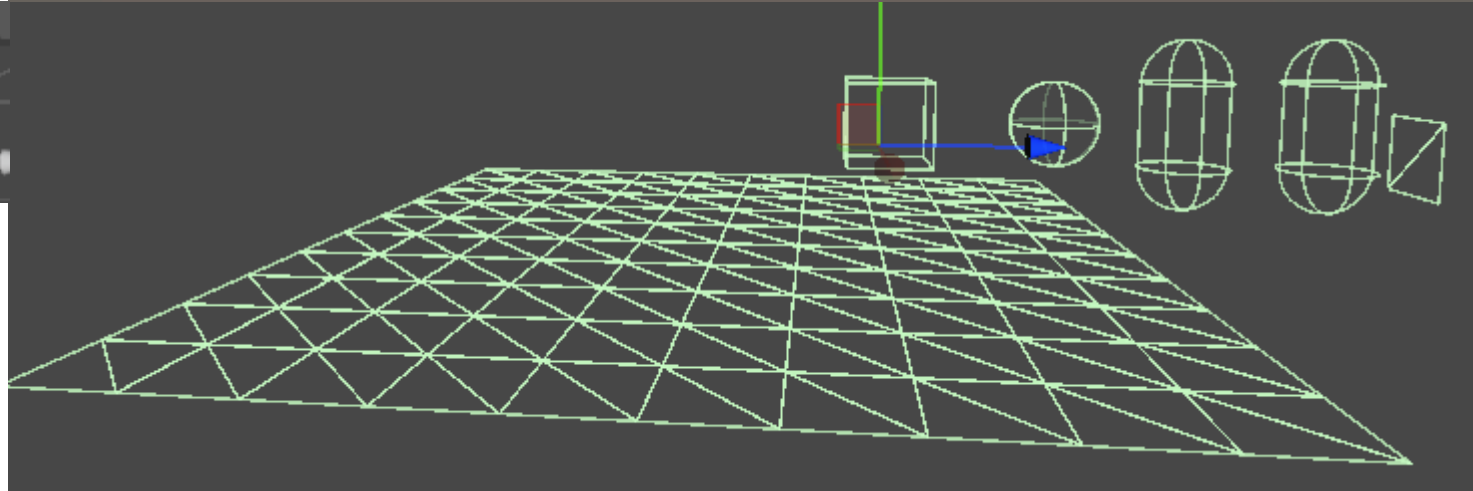
プリミティブな素材のコライダを見てみる



全オブジェクトの
Mesh Rendererをオン



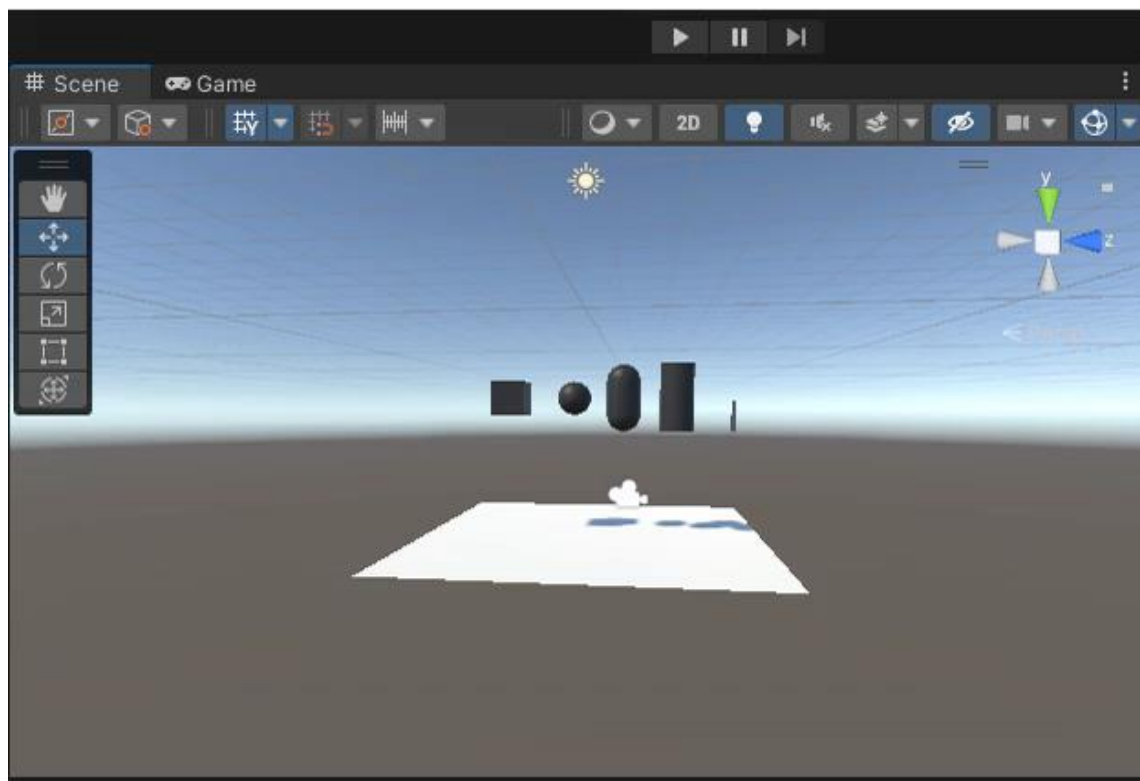
全オブジェクトの
Mesh Rendererをオフ



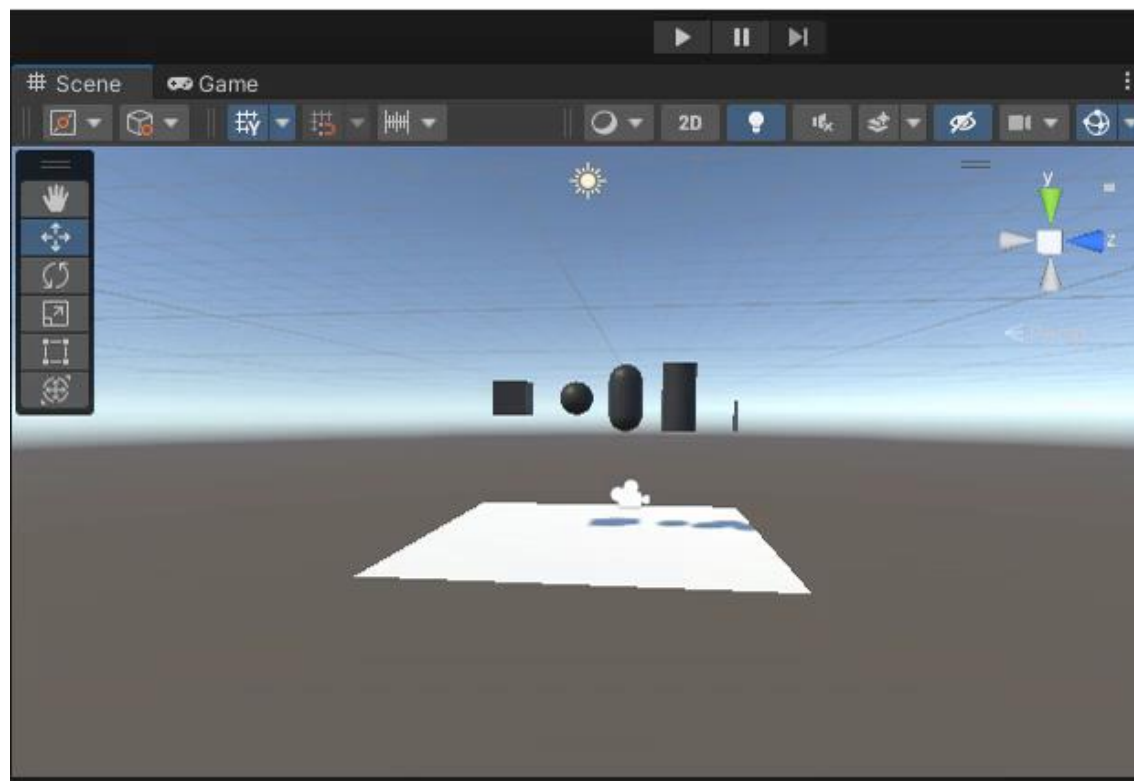


コライダによる衝突の様子

コライダなしの場合：



コライダありの場合：

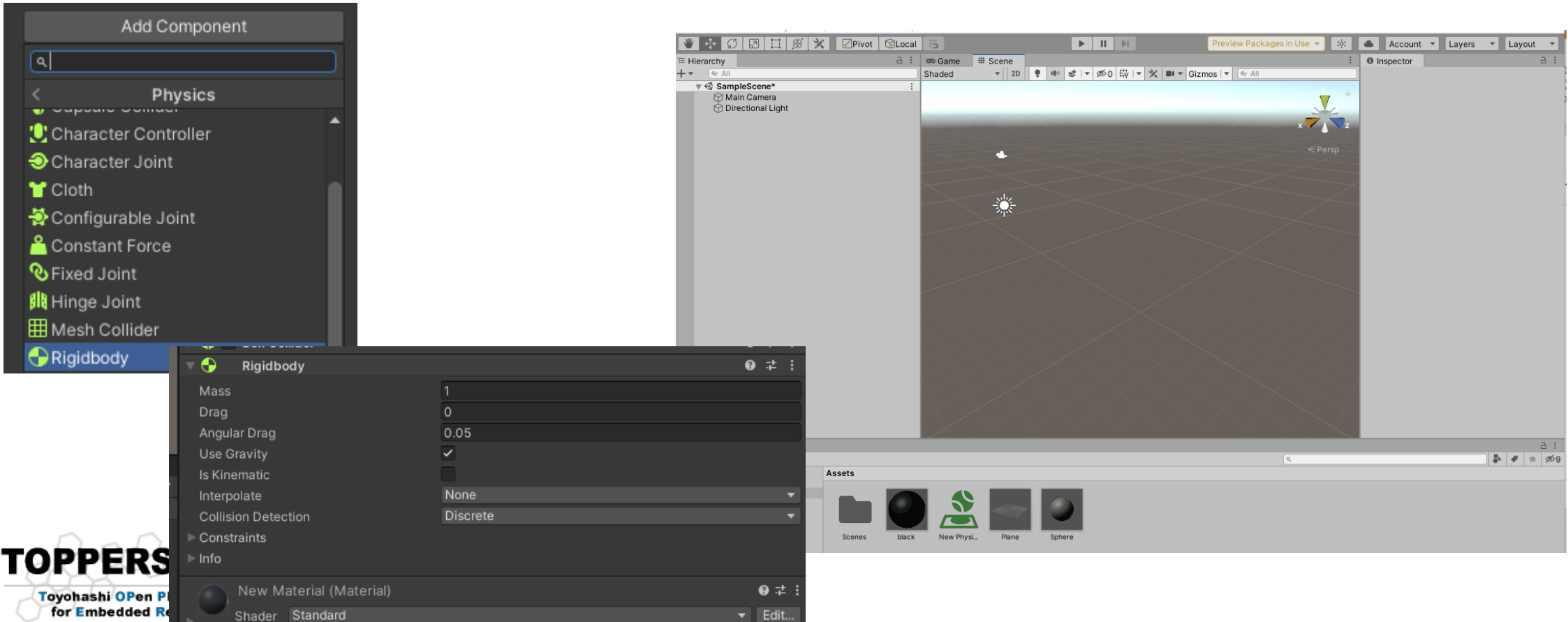




物理挙動

• Rigidbody

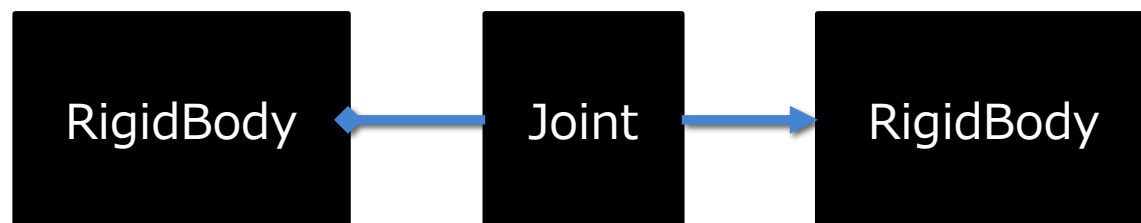
- Rigidbody (リジッドボディ) はオブジェクトに物理挙動を可能にするためのメインコンポーネントです。リジッドボディを加えた瞬間から、オブジェクトは重力の影響を受けようになります。さらに、1つ以上の Collider (コライダー) コンポーネントを加えれば、オブジェクトは衝突の影響によって動くようになります。



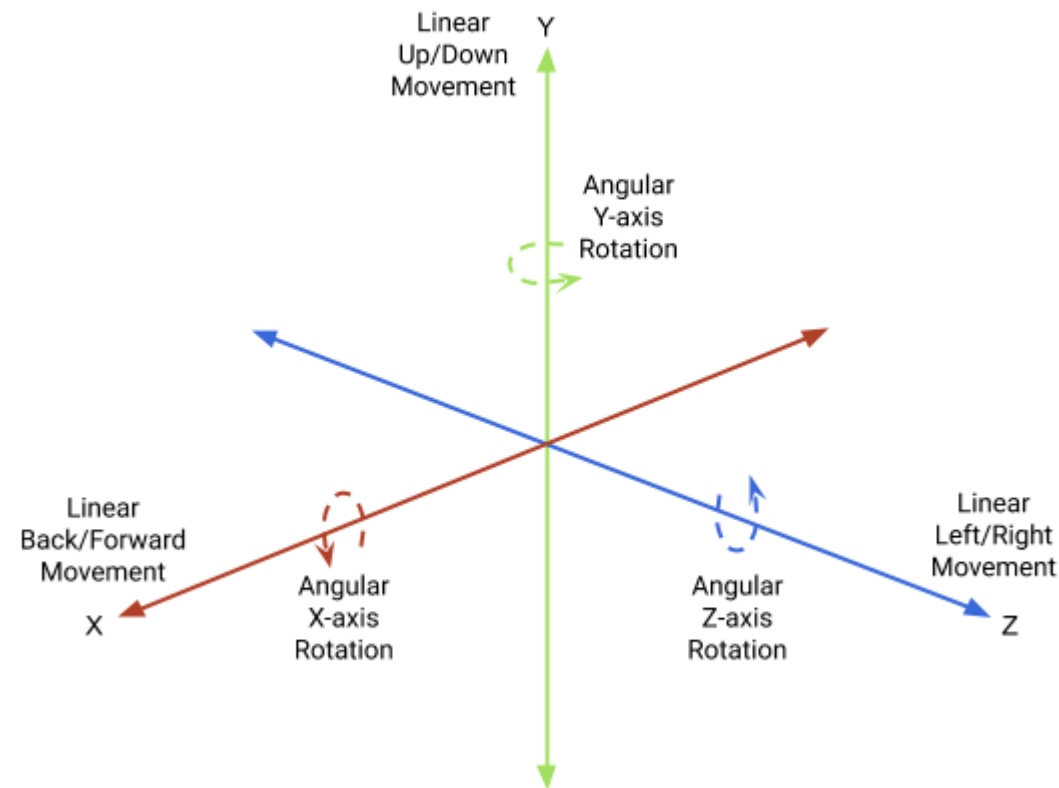


Jointコンポーネント

- Joint (ジョイント) コンポーネントは、Rigidbody (リジッドボディ) を他のRigidbody または空間の固定点に接続します。ジョイントはリジッドボディを動かす力を加え、ジョイントの制限はその動きを制限します。ジョイントはリジッドボディに以下の自由度を与えます。



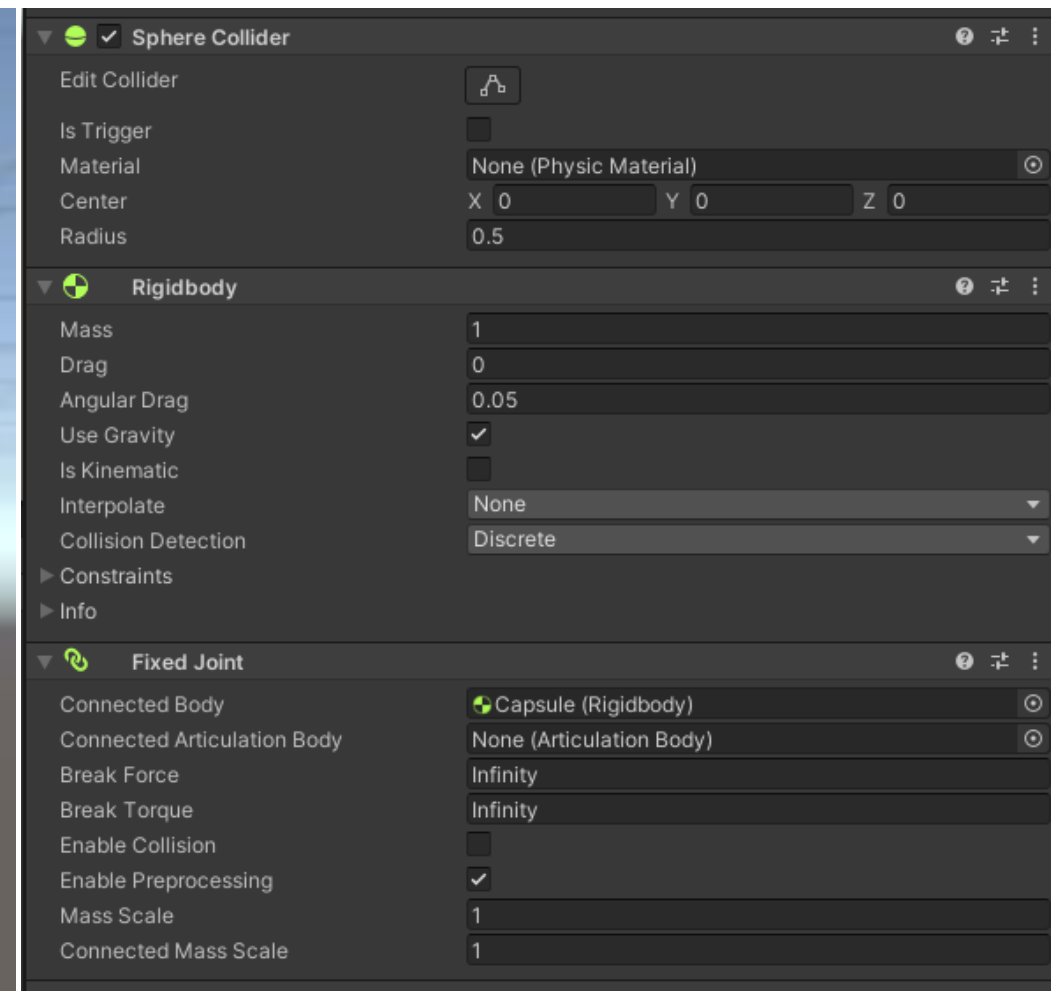
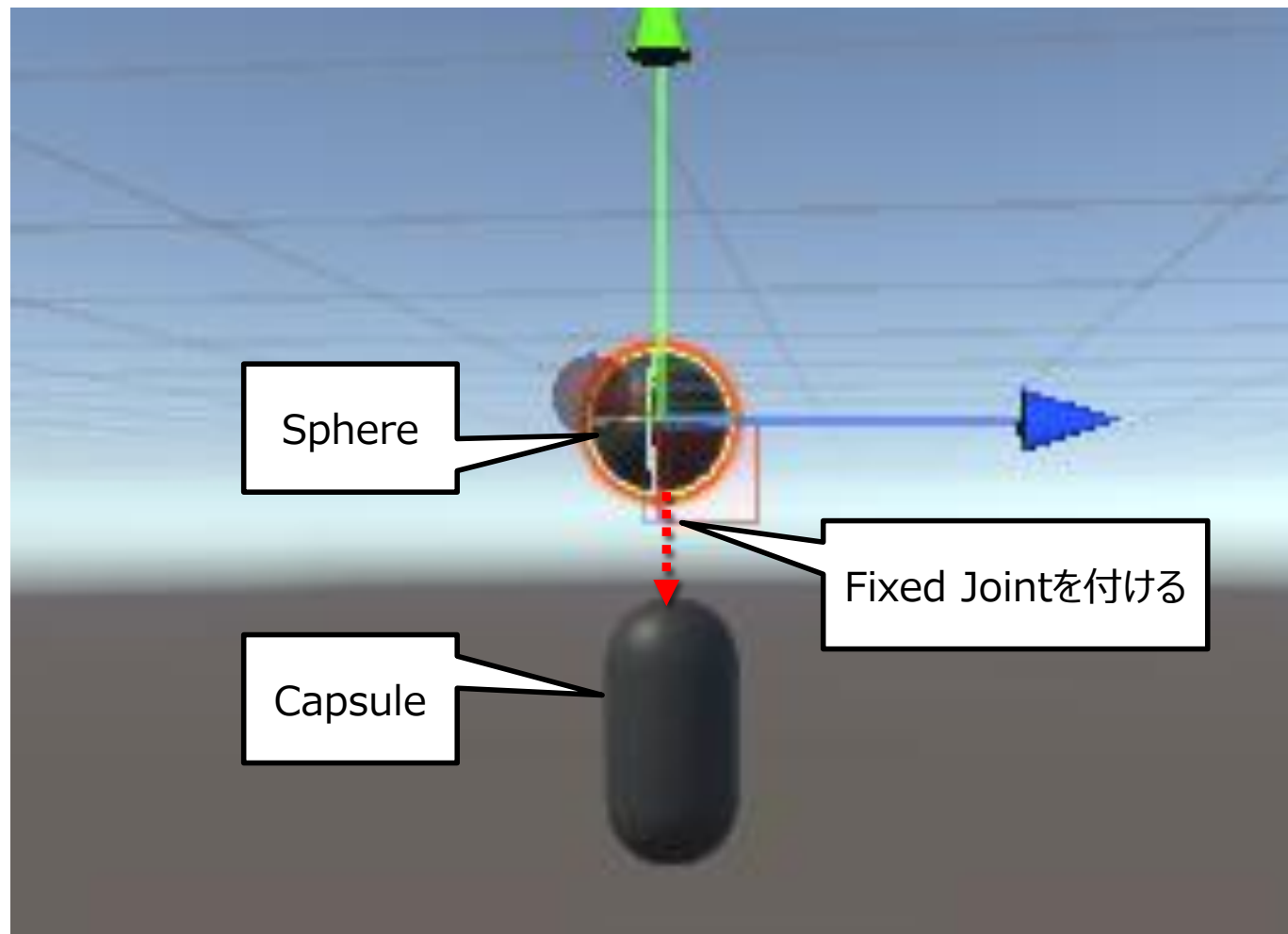
| Jointコンポーネントの種類 | 箱庭での利用有無 |
|--------------------|----------|
| Character Joint | × |
| Configurable Joint | ○ |
| Fixed Joint | ○ |
| Hinge Joint | ○ |
| Spring Joint | × |





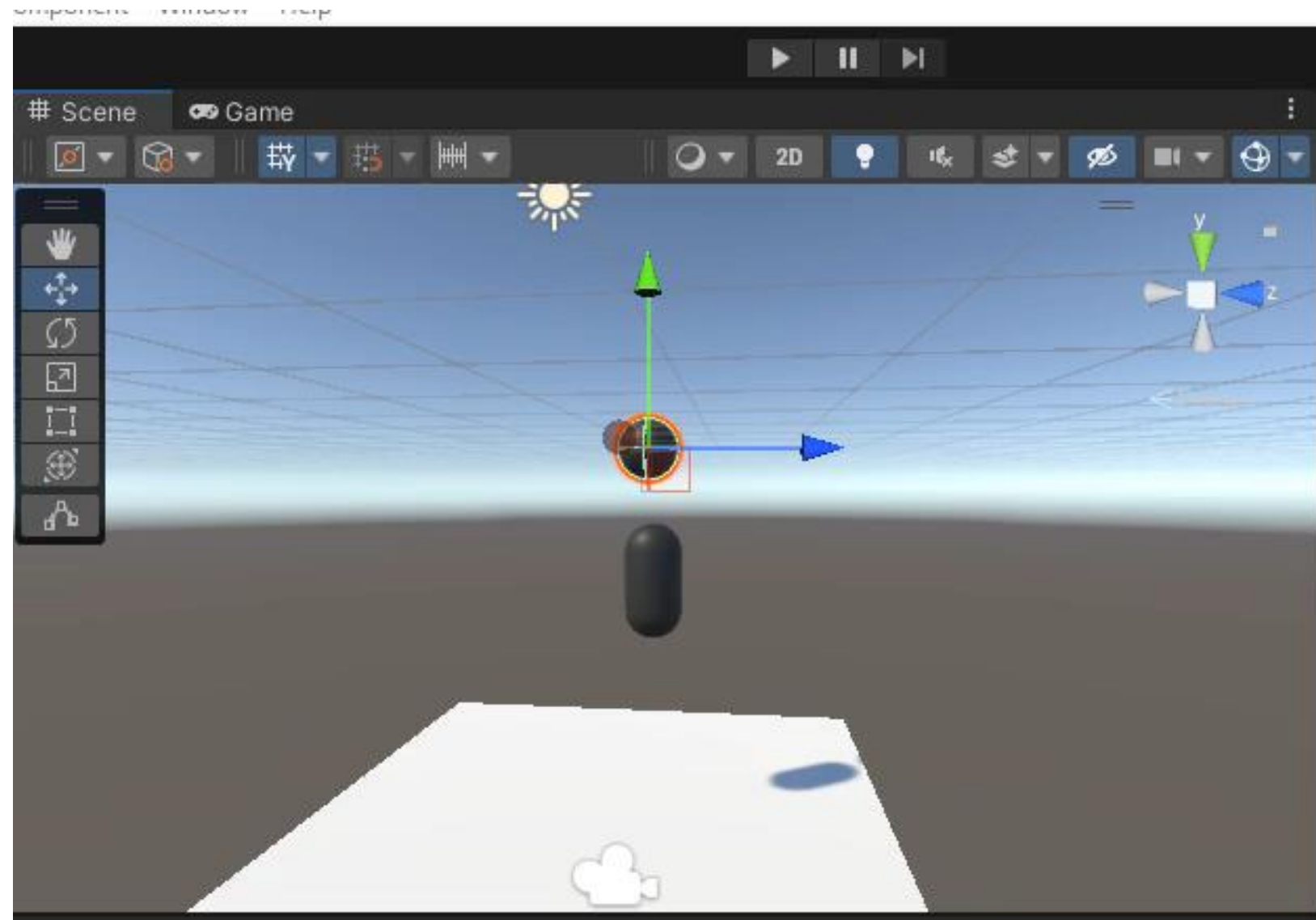
Fixed Joint

- SphereにFixed Jointコンポーネントをアタッチし、Capsuleと接続する



Fixed Joint

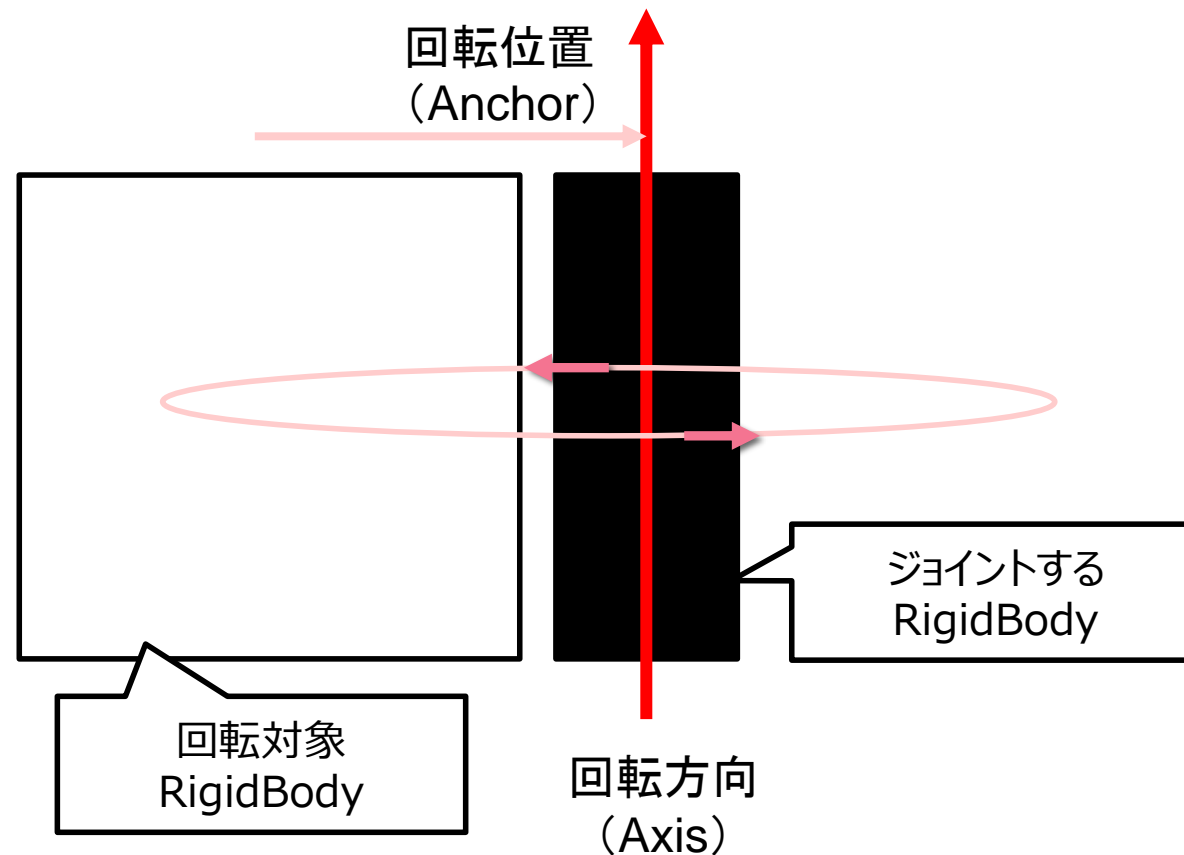
29





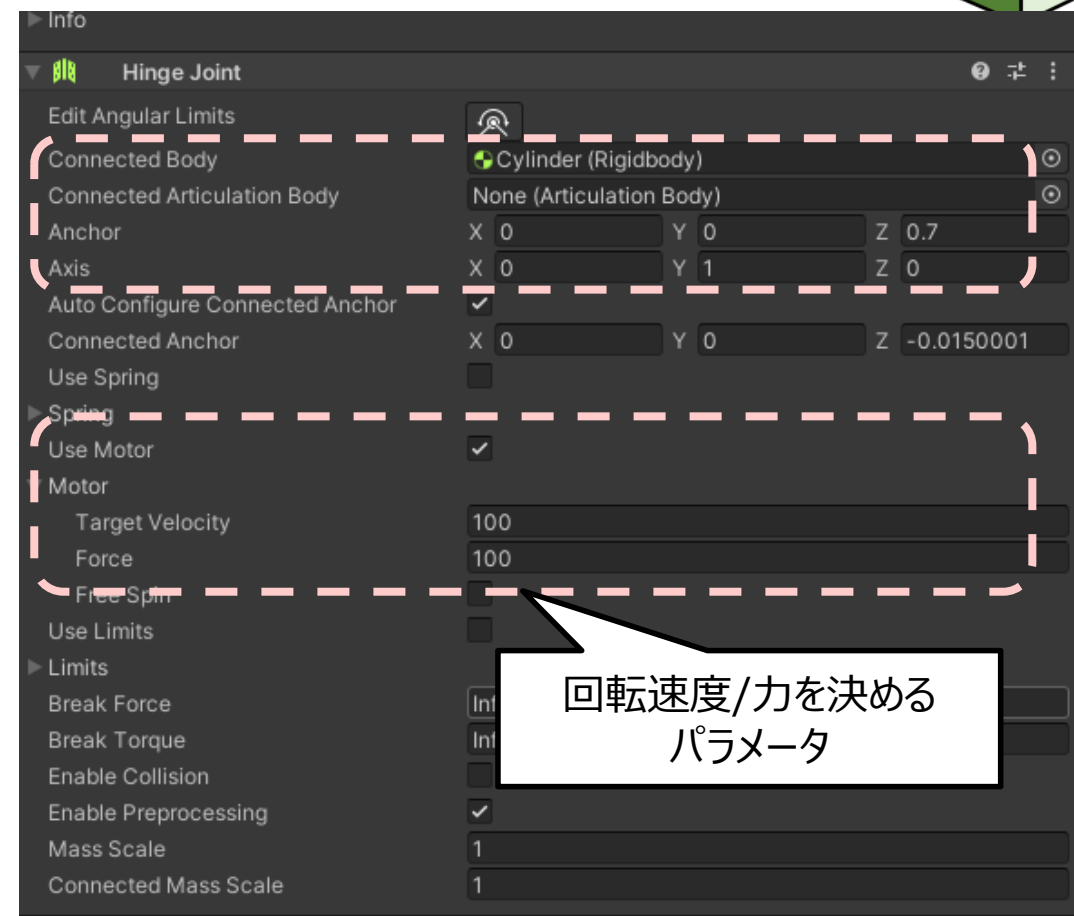
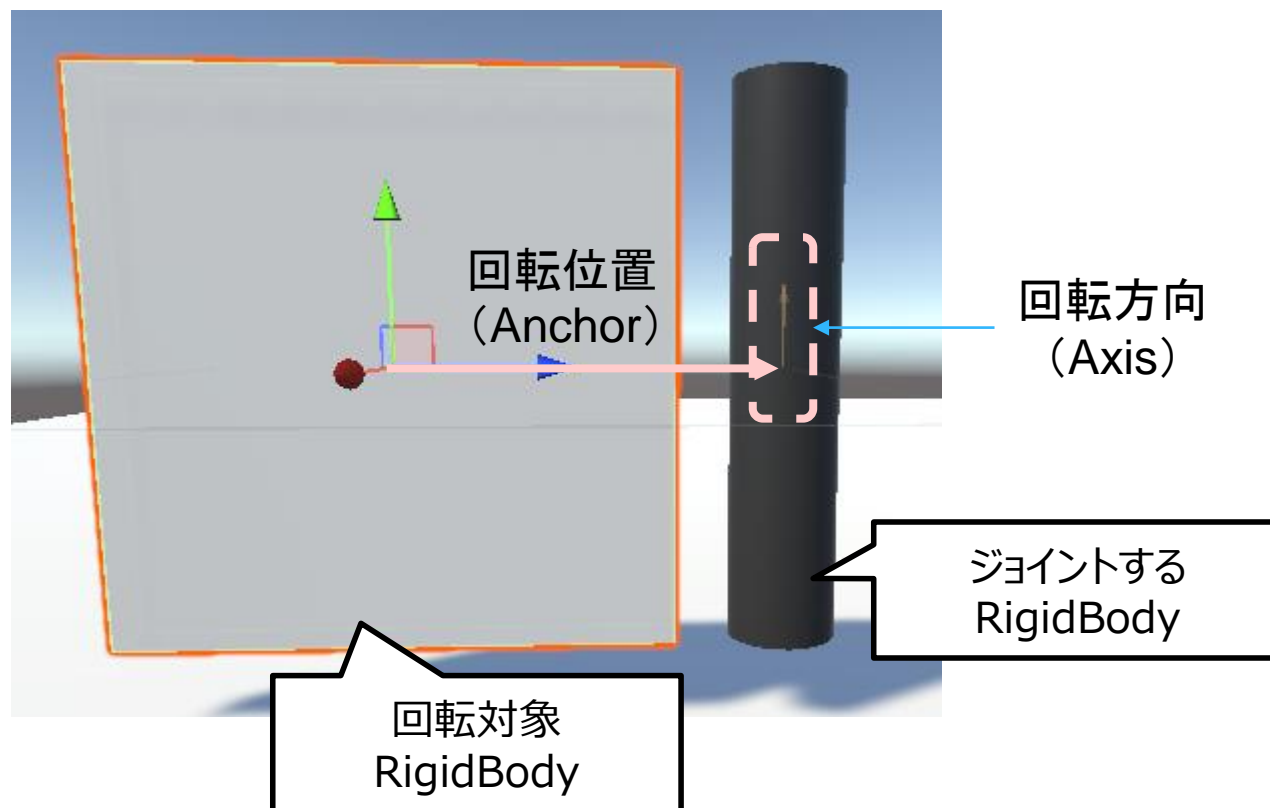
Hinge Joint

- リジッドボディを、他のリジッドボディや空間上の点に、共有された原点で取り付け、その原点の特定の軸を中心に回転させることができます。ドアや指の関節を模倣するのに便利です。



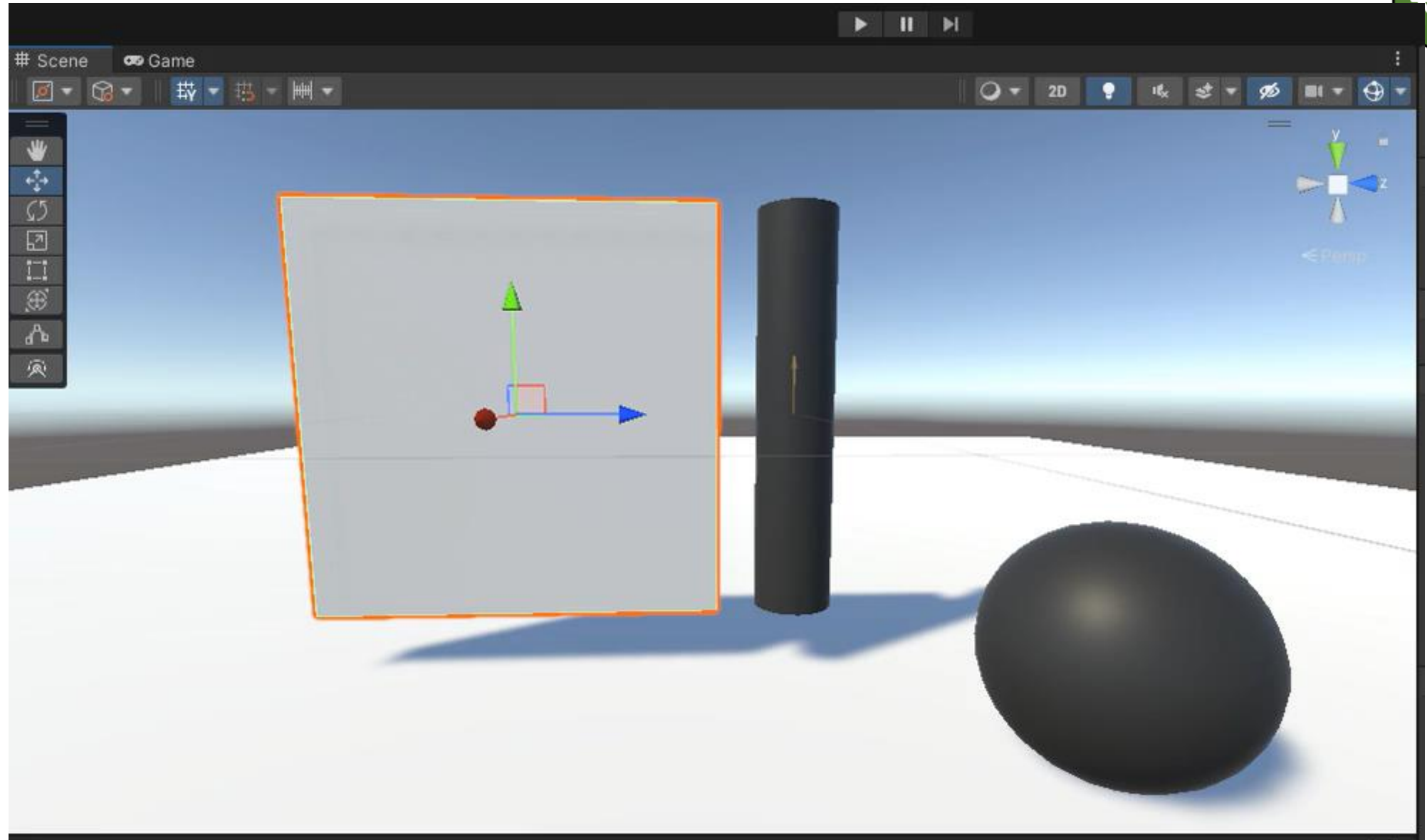
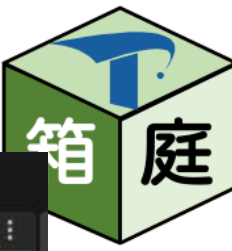


Hinge Jointの例



Hinge Joint

32



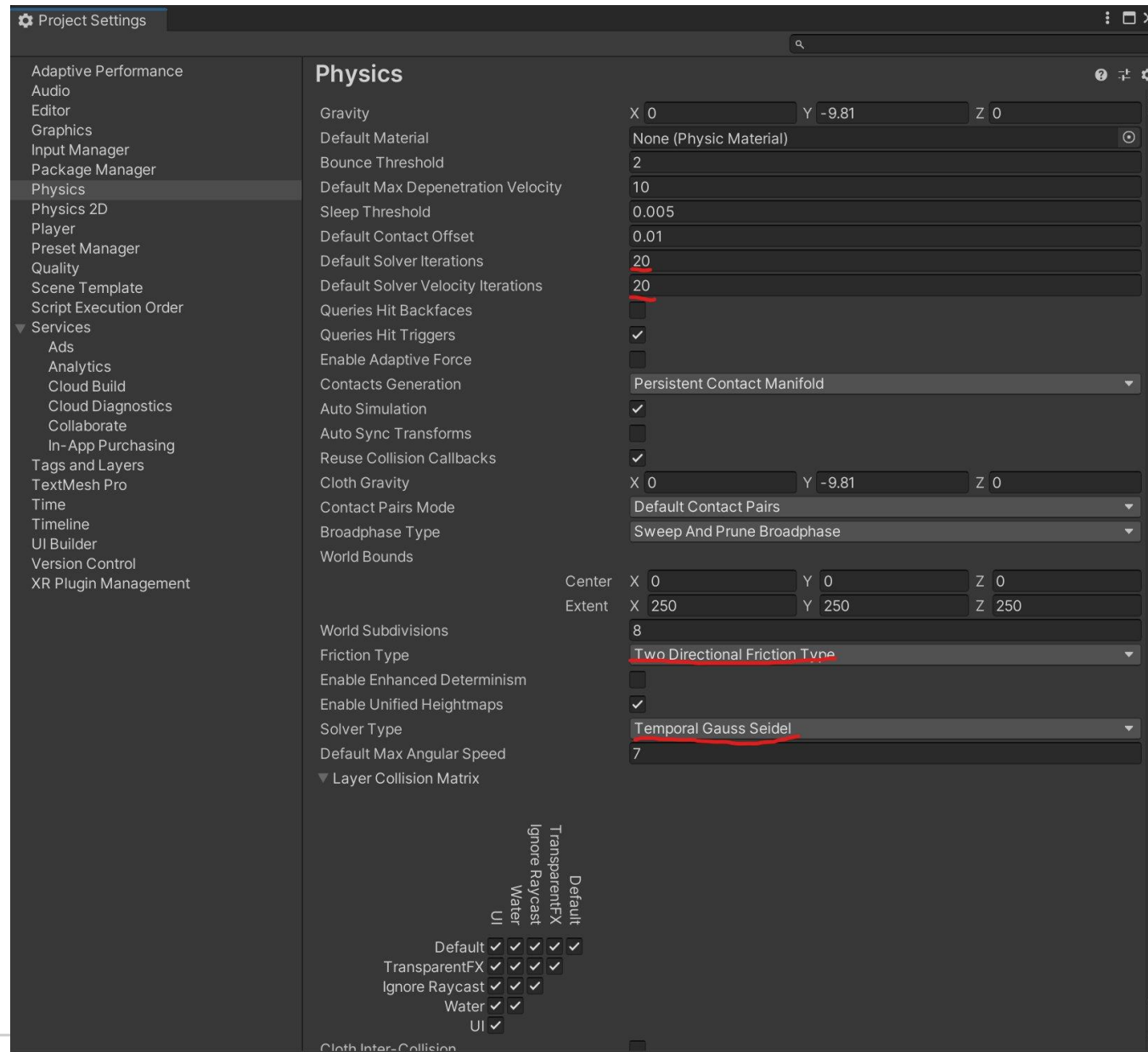


- ArticulationBody

- Articulation Body (連結ボディ) を使うことによって、ゲームオブジェクトを使ったロボットアームやキネマティックチェーンのような 物理的連結 を構築することができます。これらは、産業用アプリケーションのシミュレーションのコンテキストでリアルな物理動作を得るのに役立ちます。
- 連結ボディは、RigidBody や 通常のジョイント といった従来の構成で定義されるプロパティを、1 つのコンポーネントで定義することができます。ただし、これらのプロパティは、階層内のゲームオブジェクトの位置に依存します。



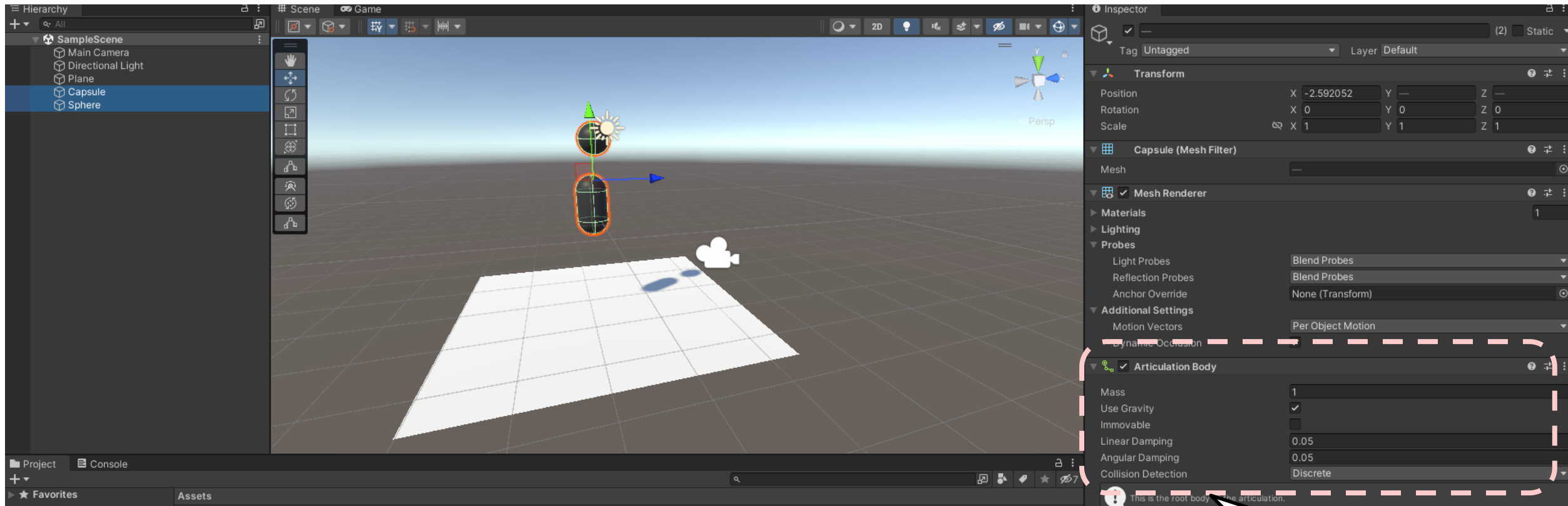
プロジェクト設定を変更する必要があります





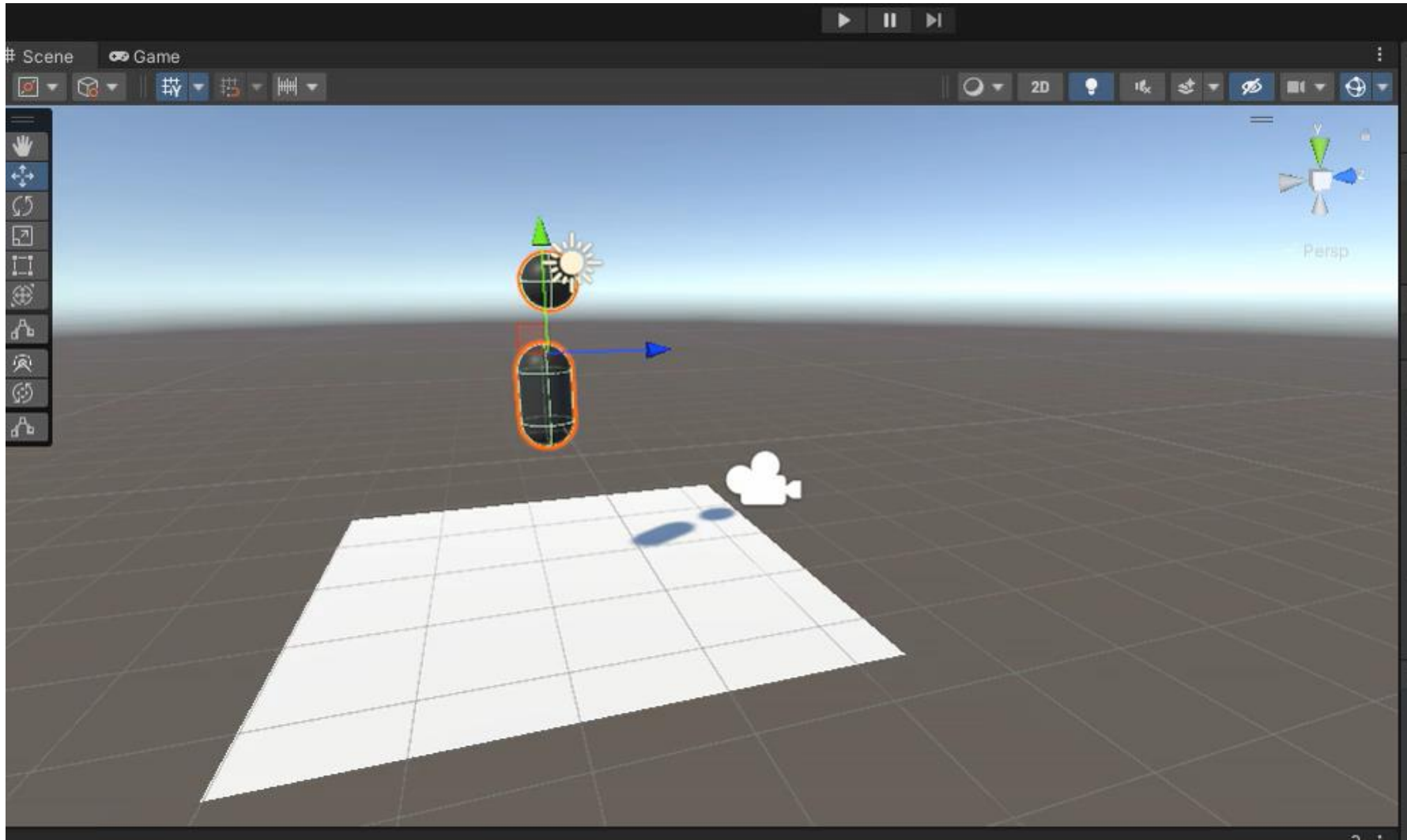
論よりRUN!

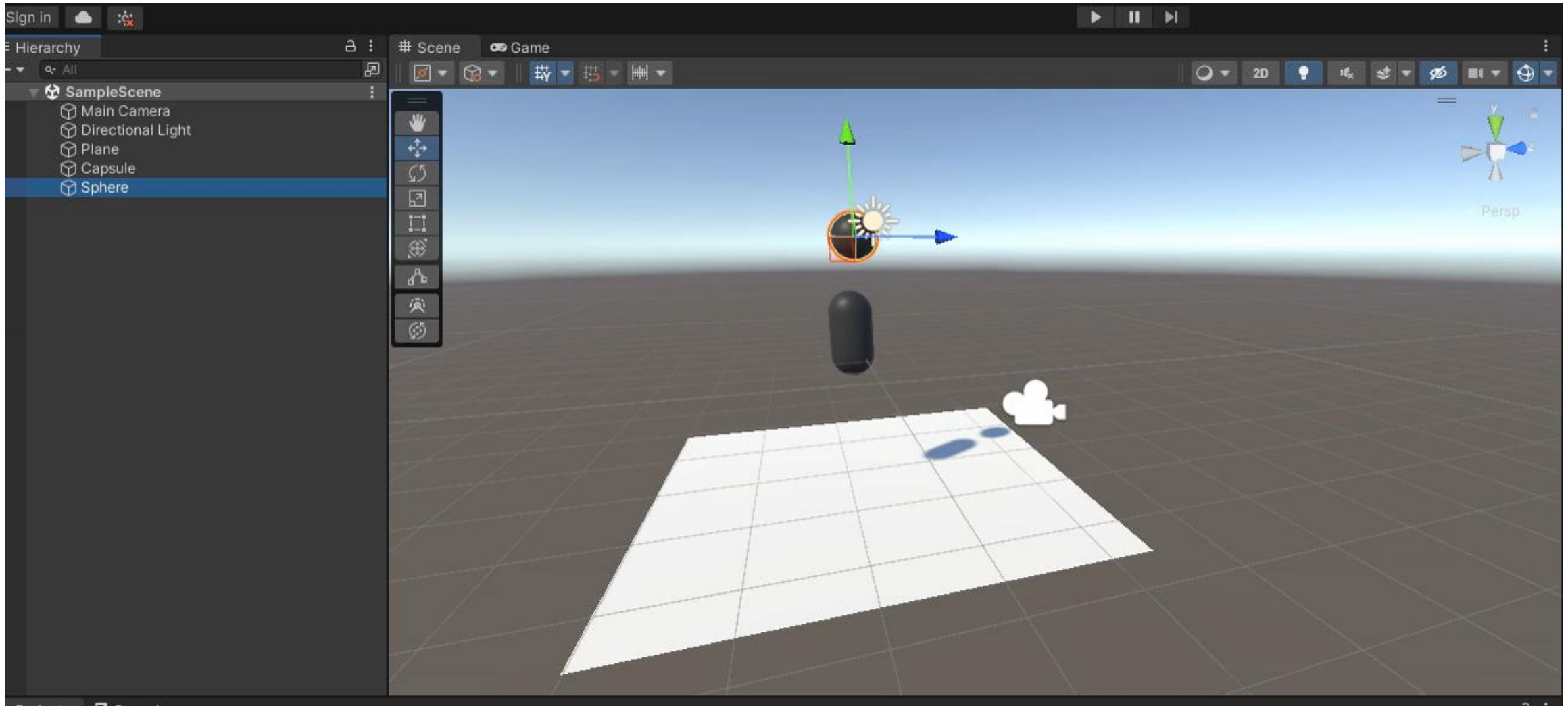
- Rigidbodyをやめて、ArticulationBodyに変更する！

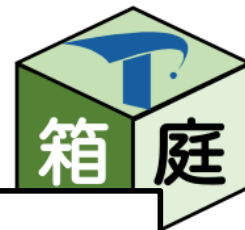


Articulation Bodyをア
タッチするだけ！！

ArticulationBody





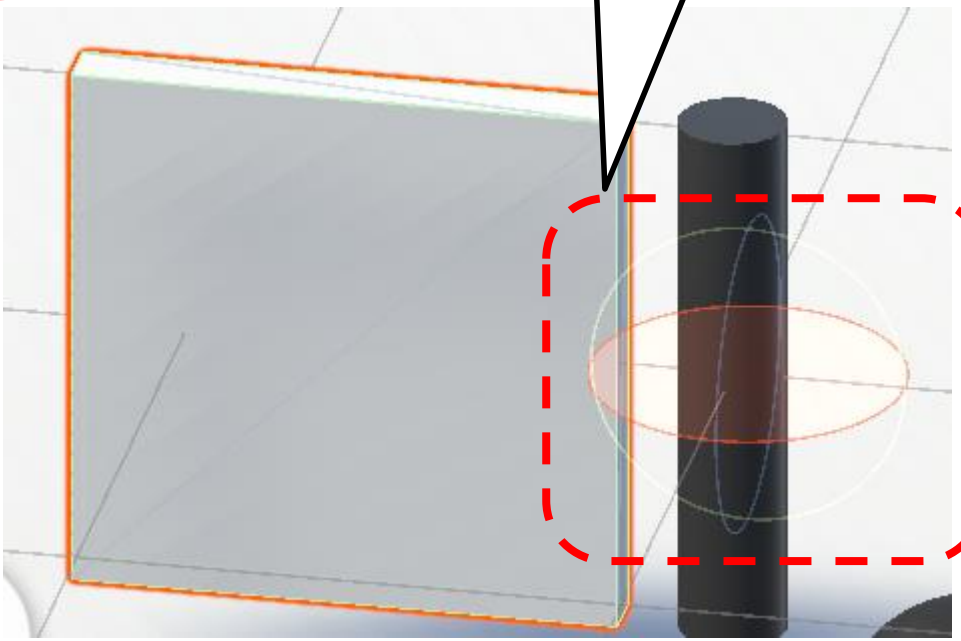


Hinge Jointと同じ設定したい場合

CubeをCylinder配下に移動



CylinderのArticulationBodyの接合部分および回転方向を編集



Joint Typeを Revolute にして、回転設定する

Articulation Body configuration panel:

- Articulation Joint Type: Revolute
- Motion: Free
- X Drive:
 - Stiffness: 0
 - Damping: 20
 - Force Limit: 3.402823e+38
 - Target: 50
 - Target Velocity: 50



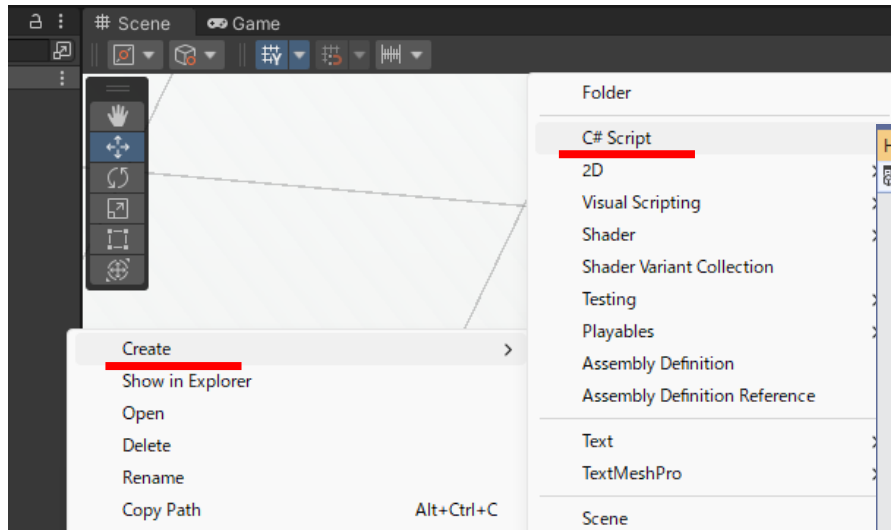
The screenshot displays a Unity development environment. The central scene shows a white grid floor under a blue sky with a sun. A white camera icon is positioned on the floor. A grey plane with a red, green, and blue coordinate system is placed on the grid. A black cylinder and a black sphere are also visible. The Inspector panel on the right shows the following components and their properties:

- Cube (Mesh Filter)**
 - Mesh: Cube
 - Mesh Renderer: Mesh Renderer
 - Materials: 1
 - Lighting: Lighting
 - Probes
 - Light Probes: Blend Probes
 - Reflection Probes: Blend Probes
 - Anchor Override: None (Transform)
 - Additional Settings
 - Motion Vectors: Per Object Motion
 - Dynamic Occlusion:
- Box Collider**
 - Edit Collider: Edit Collider
 - Is Trigger:
 - Material: None (Physic Material)
 - Center: X 0, Y 0, Z 0
 - Size: X 1, Y 1, Z 1
- Articulation Body**
 - Mass: 1
 - Use Gravity:
 - Immovable:
 - Linear Damping: 0.05
 - Angular Damping: 0.05
 - Collision Detection: Discrete
 - Info: This is the root body of the articulation.
 - Speed: 0
 - Velocity: X 0, Y 0, Z 0
 - Angular Velocity: X 0, Y 0, Z 0
 - Inertia Tensor: X 0.1666667, Y 0.08416667, Z 0.08416667
 - Inertia Tensor Rotation: X 0, Y 0, Z 0
 - Local Center of Mass: X 0, Y 0, Z 0
 - World Center of Mass: X 0, Y 0, Z -0.703
 - Sleep State: Awake
 - Body Index: 0
 - Default-Material (Material): Shader Standard



独自処理(Unityスクリプト)の追加方法 (1 / 2)⁴⁰

- Unityのゲームオブジェクトには、Unityスクリプト (C#) を追加できます

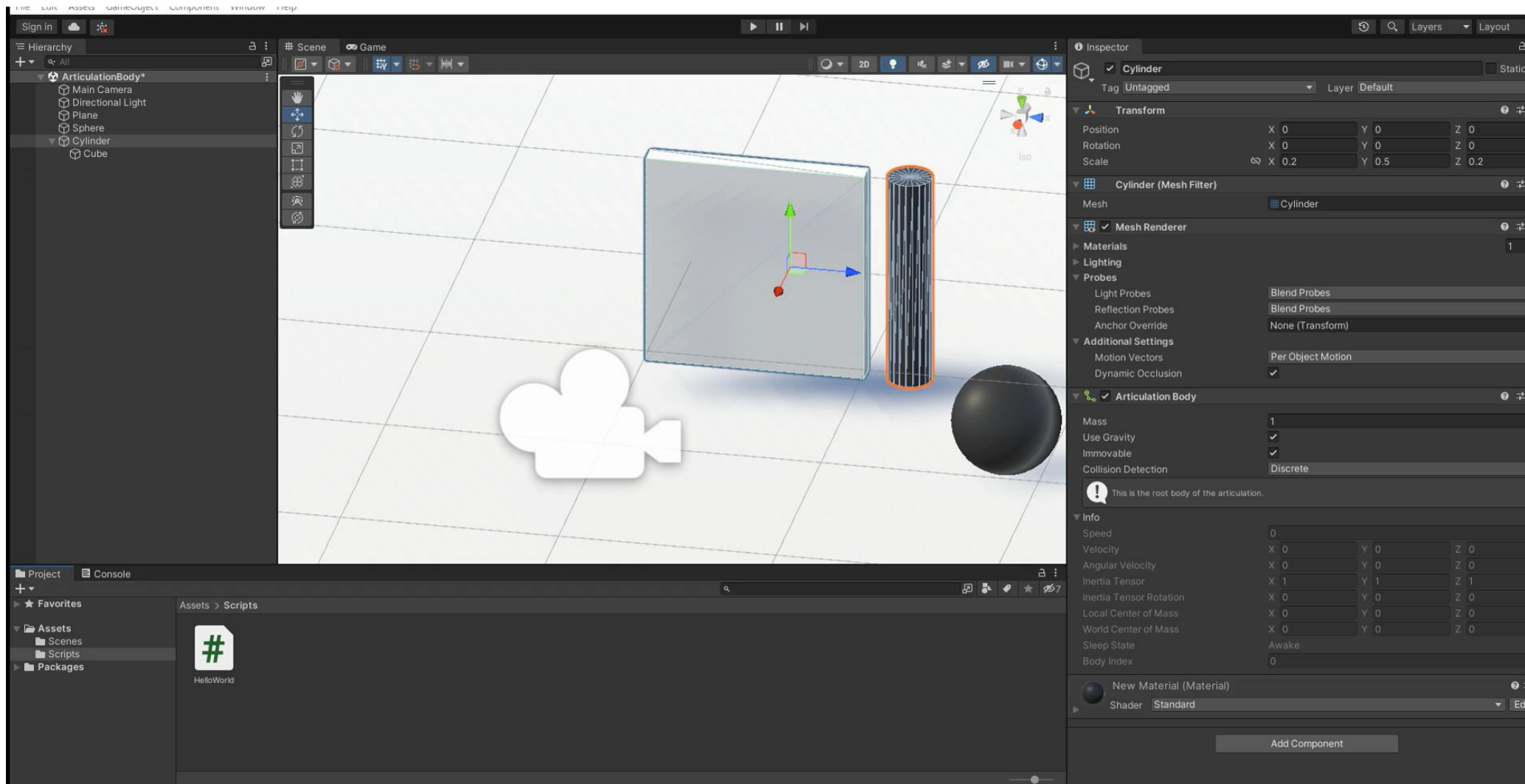


```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class HelloWorld : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10         Debug.Log("Hello World!!");
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16         Debug.Log("Called Update!!");
17     }
18 }
19
```

シミュレーション開始時に呼び出されます

フレーム毎に呼び出されます

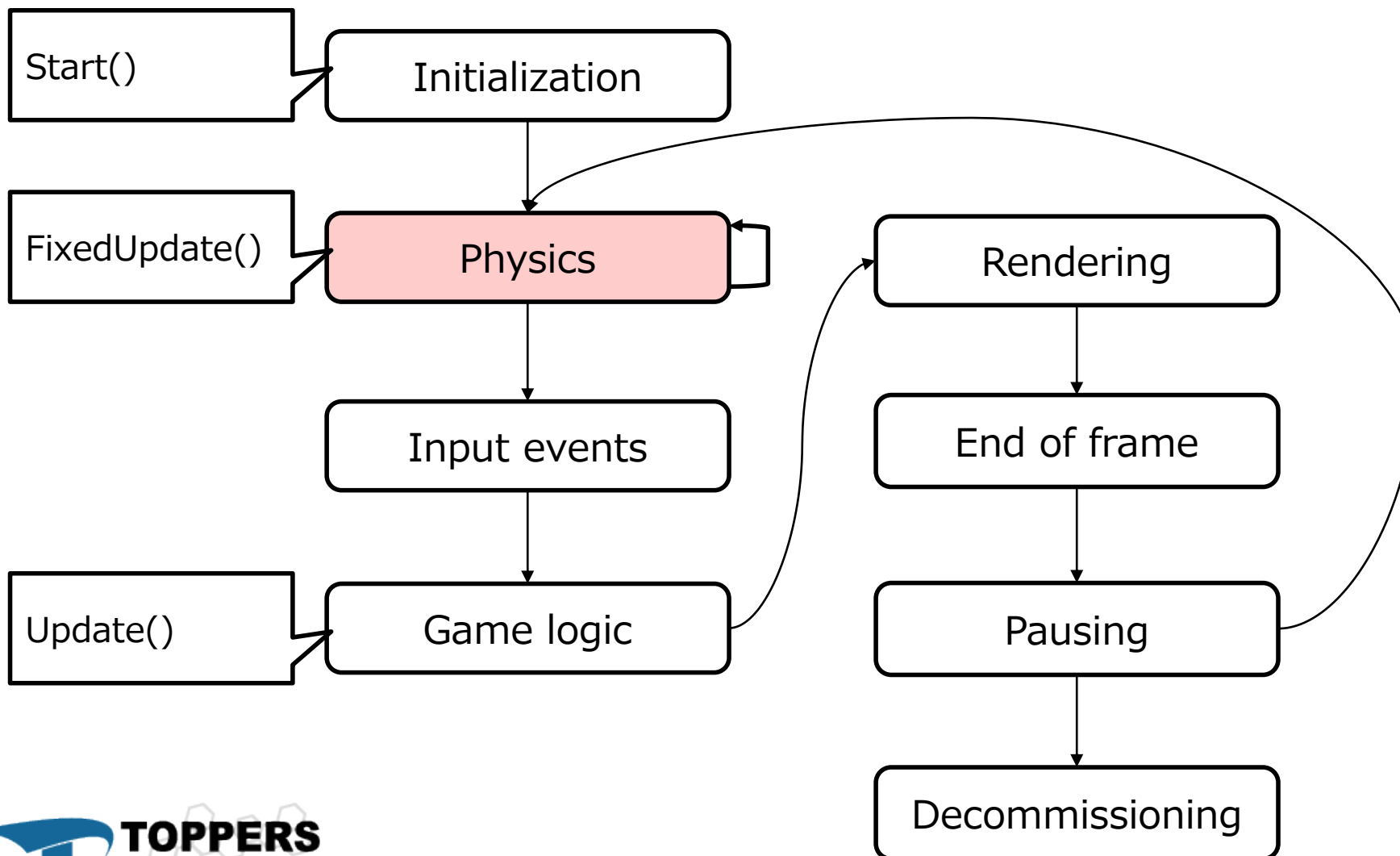
独自処理(Unityスクリプト)の追加方法 (2 / 2)⁴¹

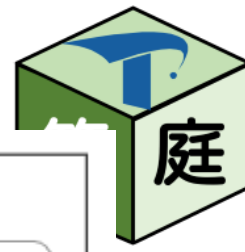




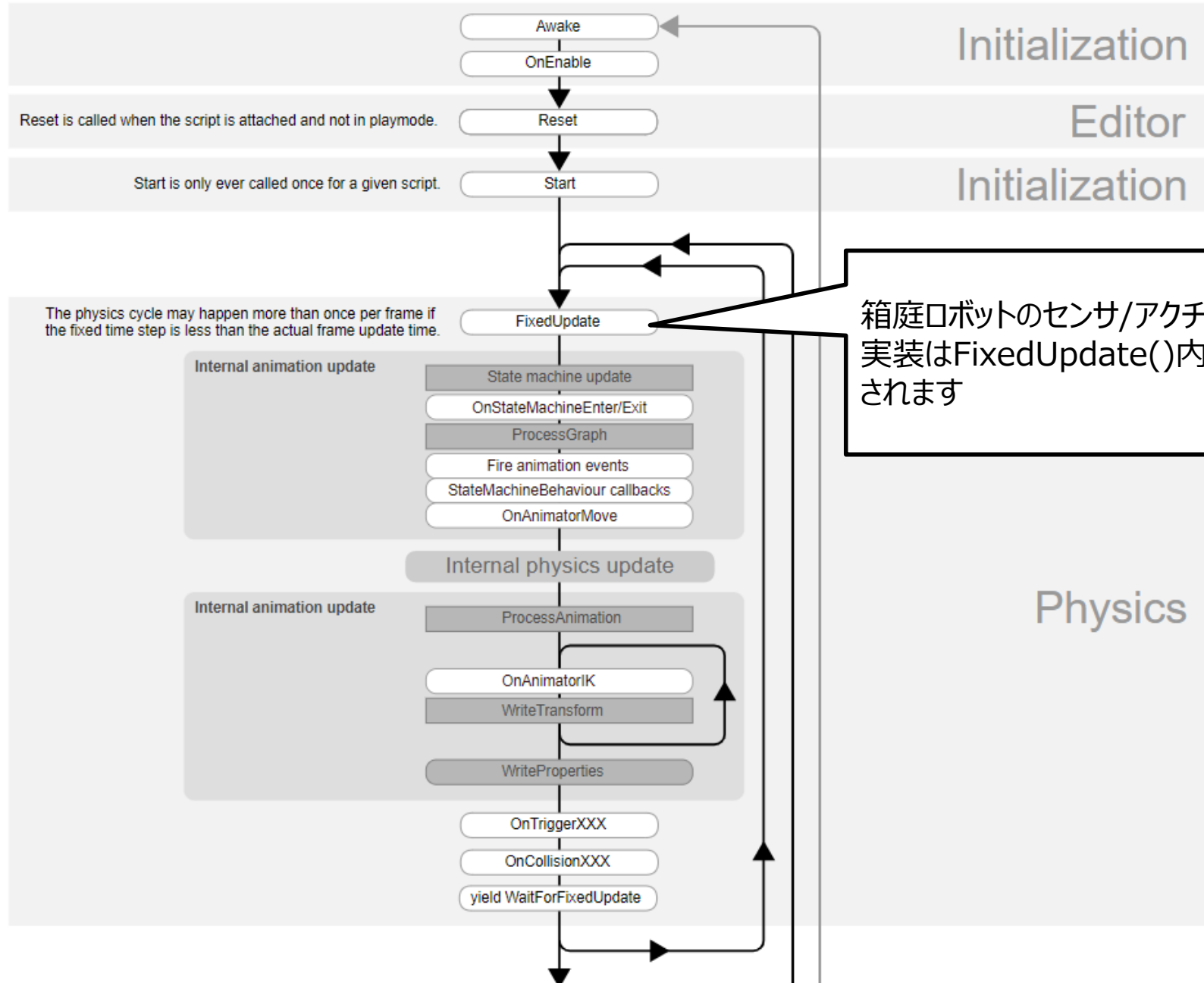
Unityイベント関数の実行順序

- <https://docs.unity3d.com/ja/2023.2/Manual/ExecutionOrder.html>





Unityイベント関数の実行順序



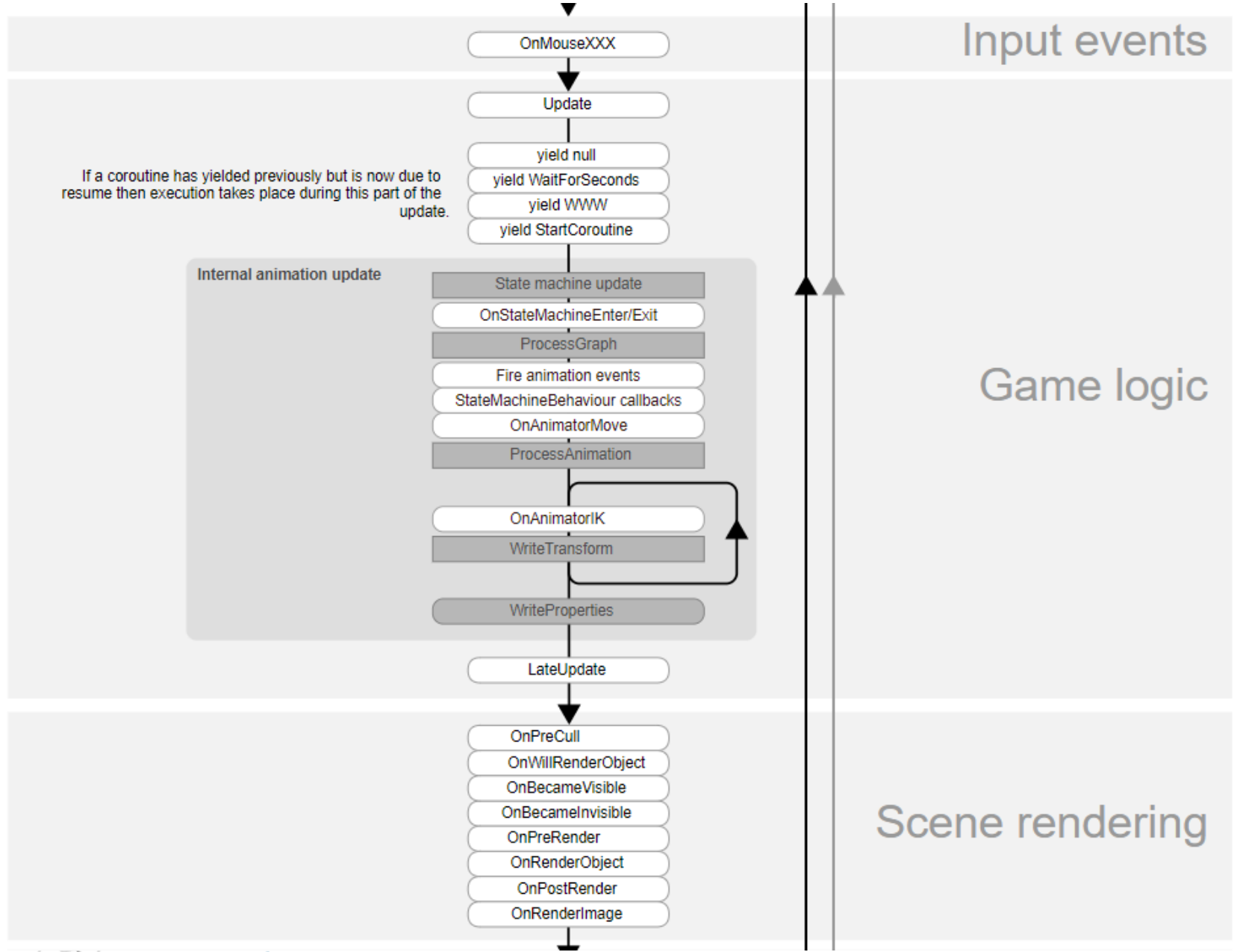
箱庭ロボットのセンサ/アクチュエータ
実装はFixedUpdate()内で実行
されます

Legend

- User callback
- Internal function
- Internal multithreaded function



Unityイベント関数の実行順序



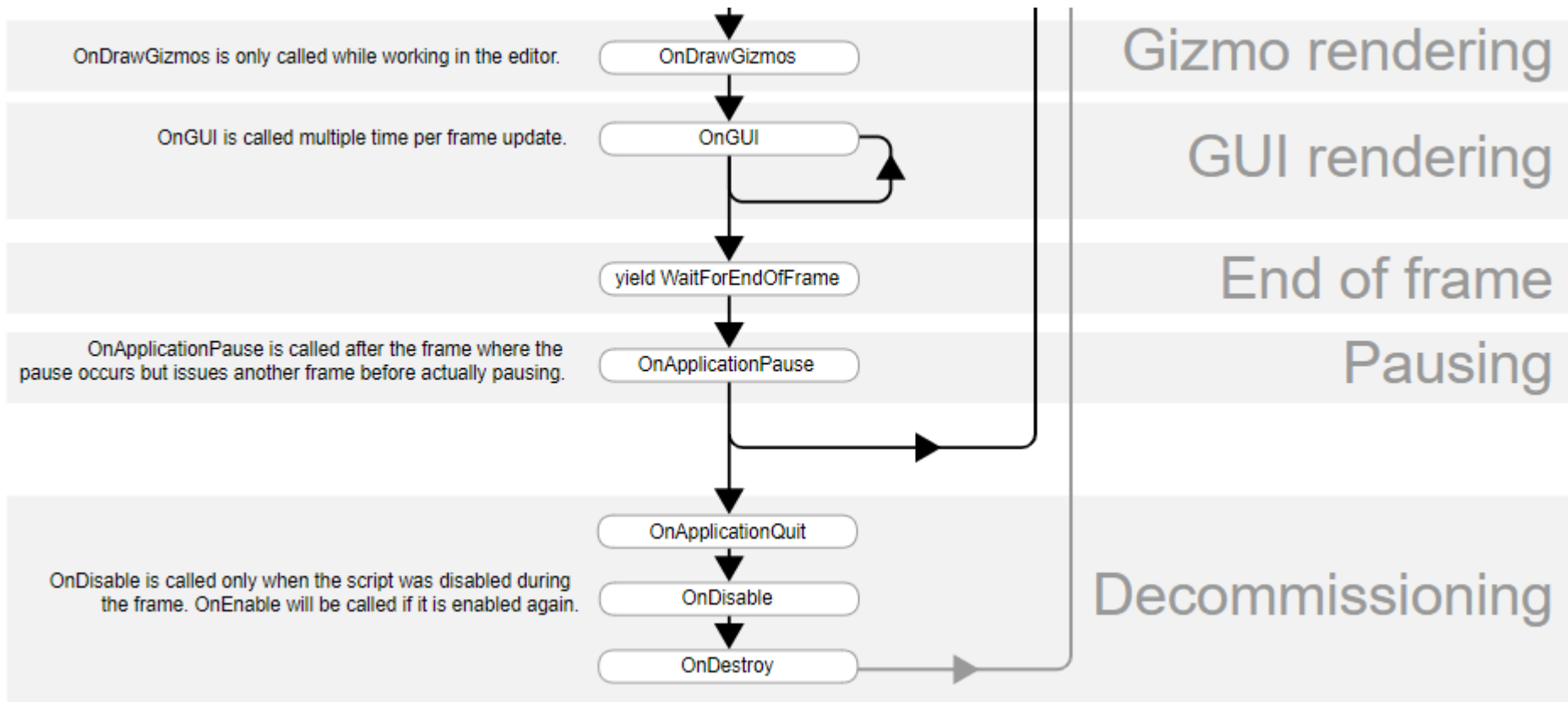
Legend

- User callback
- Internal function
- Internal multithreaded function





Unityイベント関数の実行順序



Legend

User callback

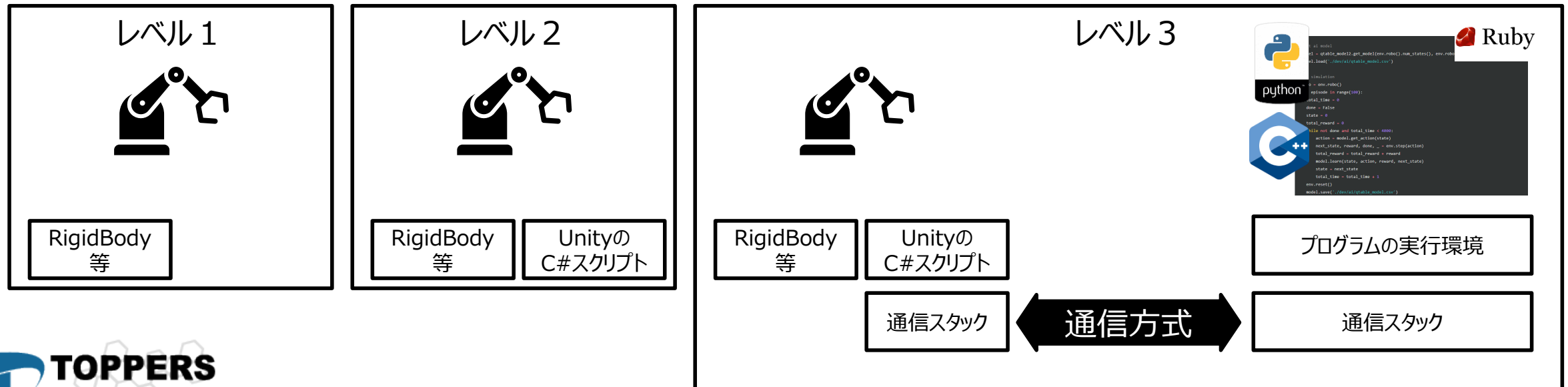
Internal function

Internal multithreaded function



ロボットを動かすためのUnityの構造

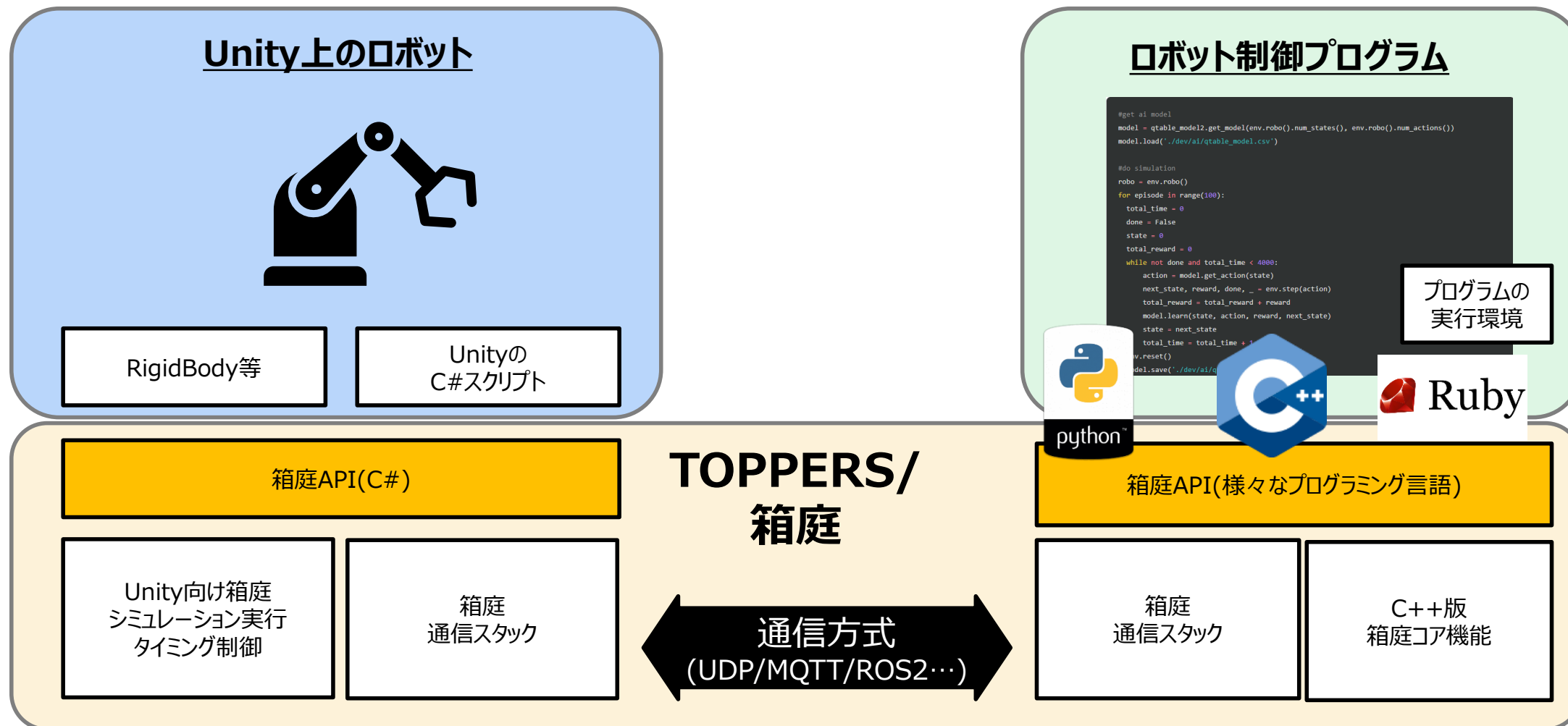
- Unity上でロボットを動かすには以下の3レベルがあります
 - UnityのRigidBodyやArticulationBodyをアタッチしてロボットを動かす
 - UnityのC#スクリプトでロボットを制御する
 - Unity外のプログラム（Python/C++言語など）で、Unity上のロボットを制御する
 - 外部プログラムの実行環境：OS上のプロセス、マイコンシミュレータ上のRTOSタスク等
 - 外部プログラムとの通信方式：ROS/ROS2、MQTT、UDP/TCP、...

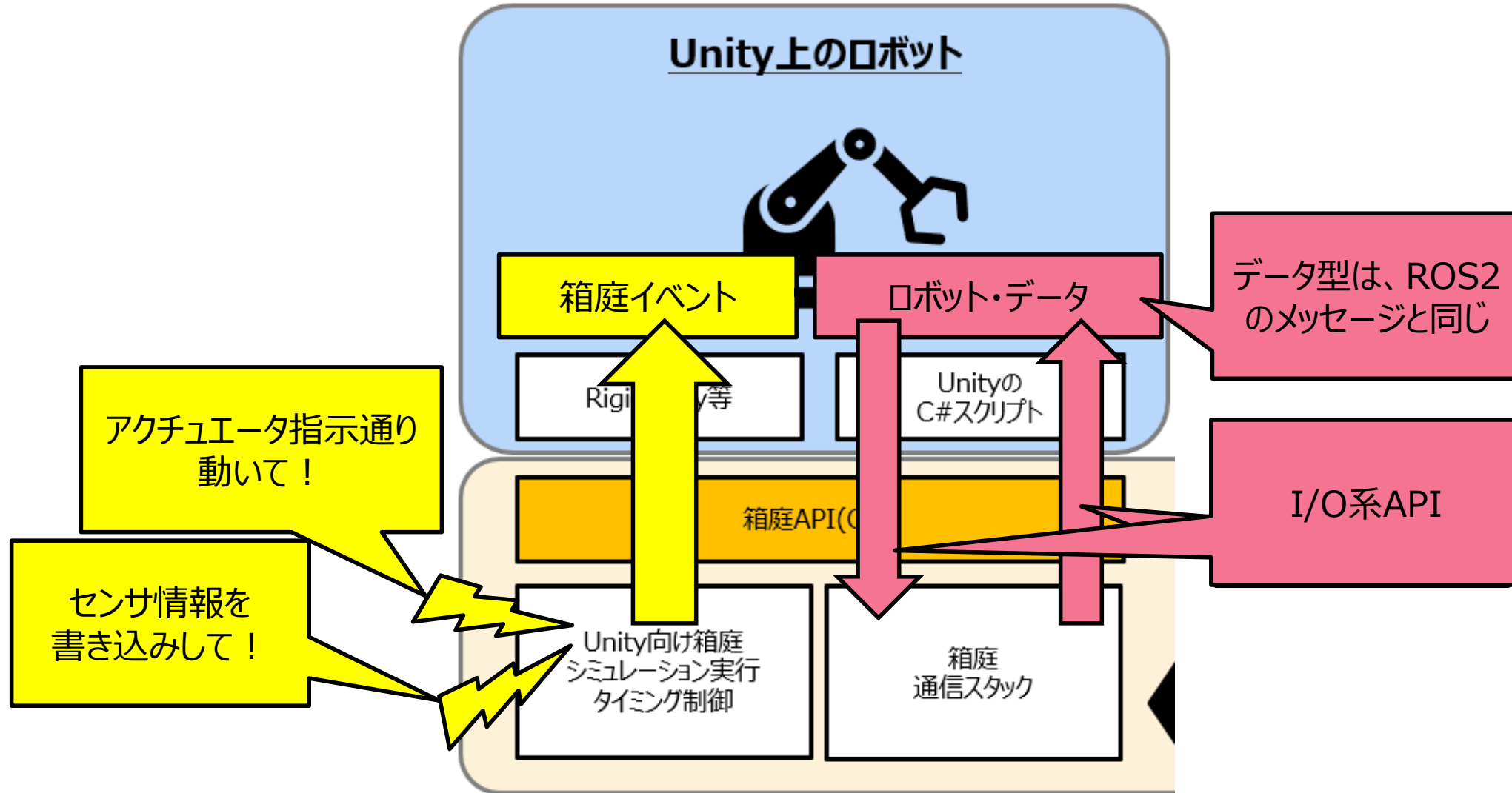




箱庭機能との接合部分

- 箱庭APIを通してUnity上のロボット／制御プログラムを接続する







箱庭API(C#)

- I/O系

- [ロボット・センシングデータの書き込み](#)
- [ロボット・アクチュエータ指示データの読み込み](#)

- データ型

- ROS2のメッセージの[インタフェース定義言語](#)(IDL)で定義します
- 現状、TB3ベースのデータ型は標準サポートしています
 - 必要に応じて、以下で追加することが可能です。
 - <https://github.com/toppers/hakoniwa-ros2pdu>

- 箱庭イベント

- Unityシミュレーション実行向け
- Unityエディタ向け




箱庭があると何が嬉しいのか

Unity

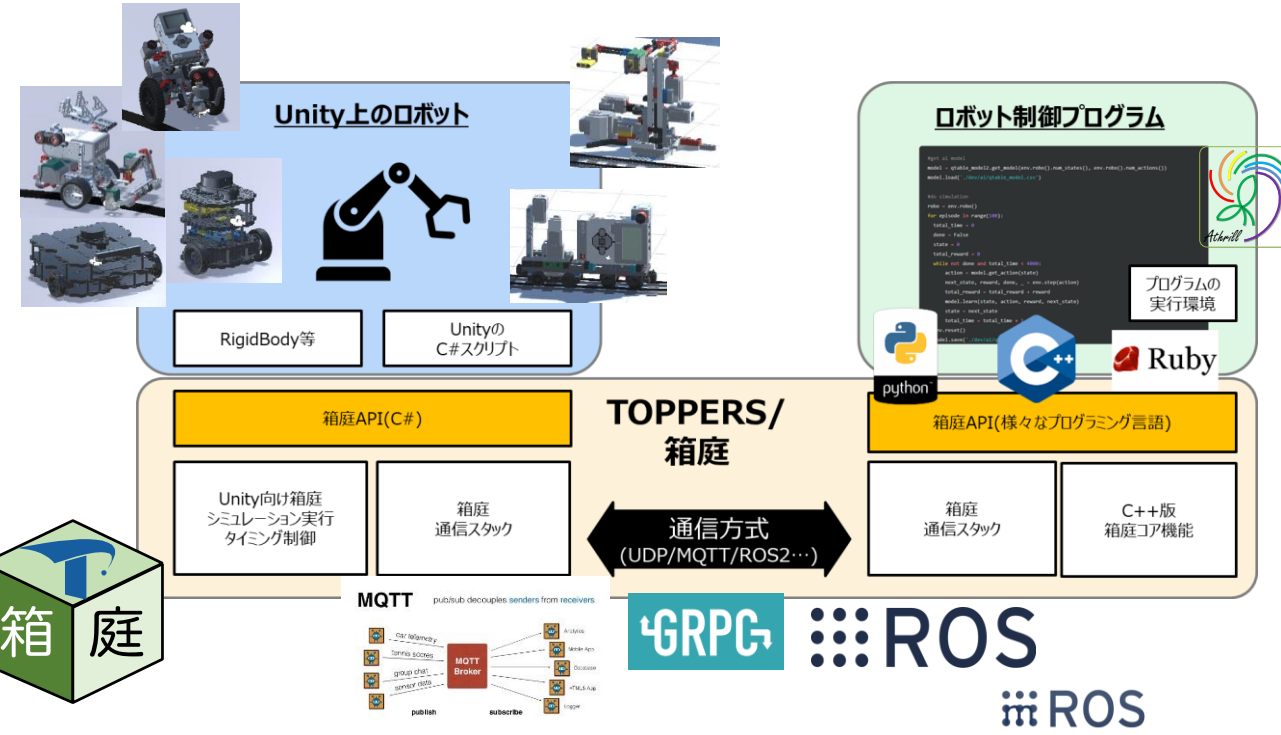
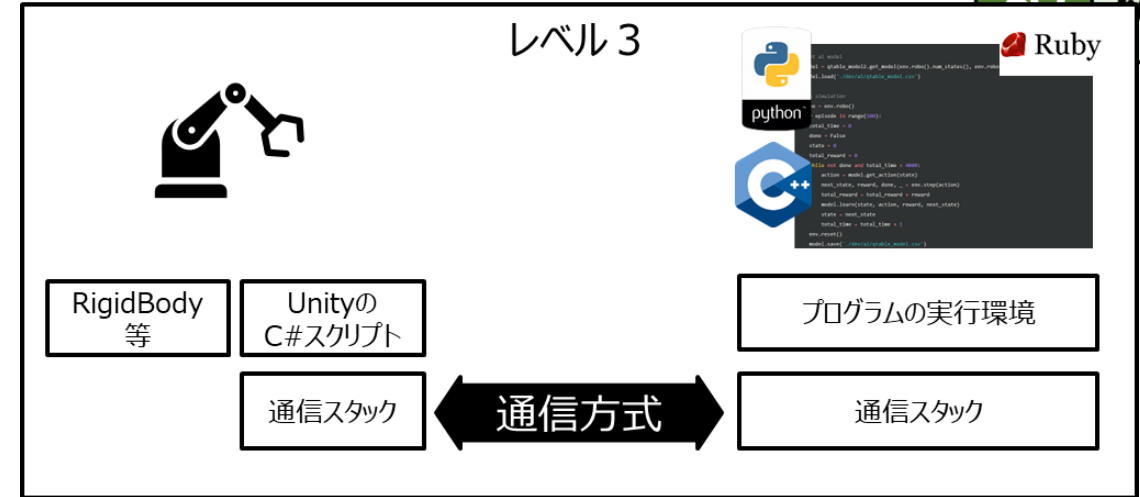
- 通信スタック作らなくても良い
- 通信方式も選べる
 - ROS2/MQTT/gRPC/UDP/MMAP
- 箱庭ロボット・アセットを選べる
 - EV3、Spike、TurtleBot3
- 箱庭ロボット組み立てキットがある
 - アクチュエータ
 - 差動モーター、サーボモーター、LEDなど
 - センサ
 - カメラセンサ、2Dレーザスキャナ、タッチセンサ、IMU

ロボット制御プログラム

- 様々なプログラムの言語を選べる
 - C/C++, Python, mRuby
- マイコンシミュレータも使える
 - Athrill 
 - 専用デバイスも自作可能

オープンソース/ライセンスフリー

- TOPPERSライセンス





箱庭ロボットの作り方

- 箱庭ロボットの組み立て方
- 箱庭ロボット部品の構成
- 箱庭ロボット部品の構造



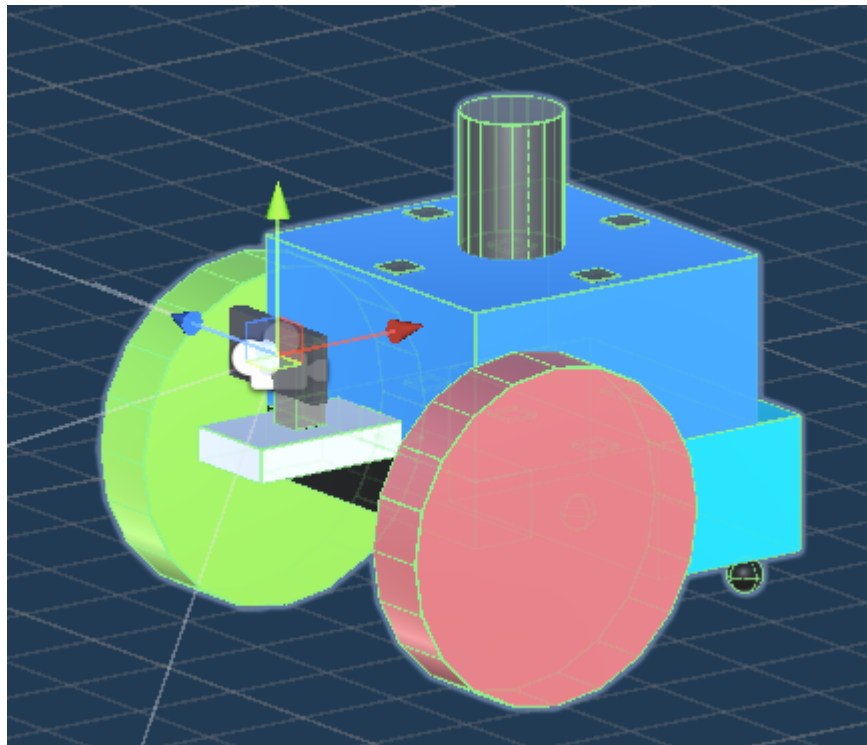
ベースのアイデア

@kanetugu2018 (TOPPERSプロジェクト)

投稿日 2022年01月23日 更新日 2022年05月07日 2878 views

Organization

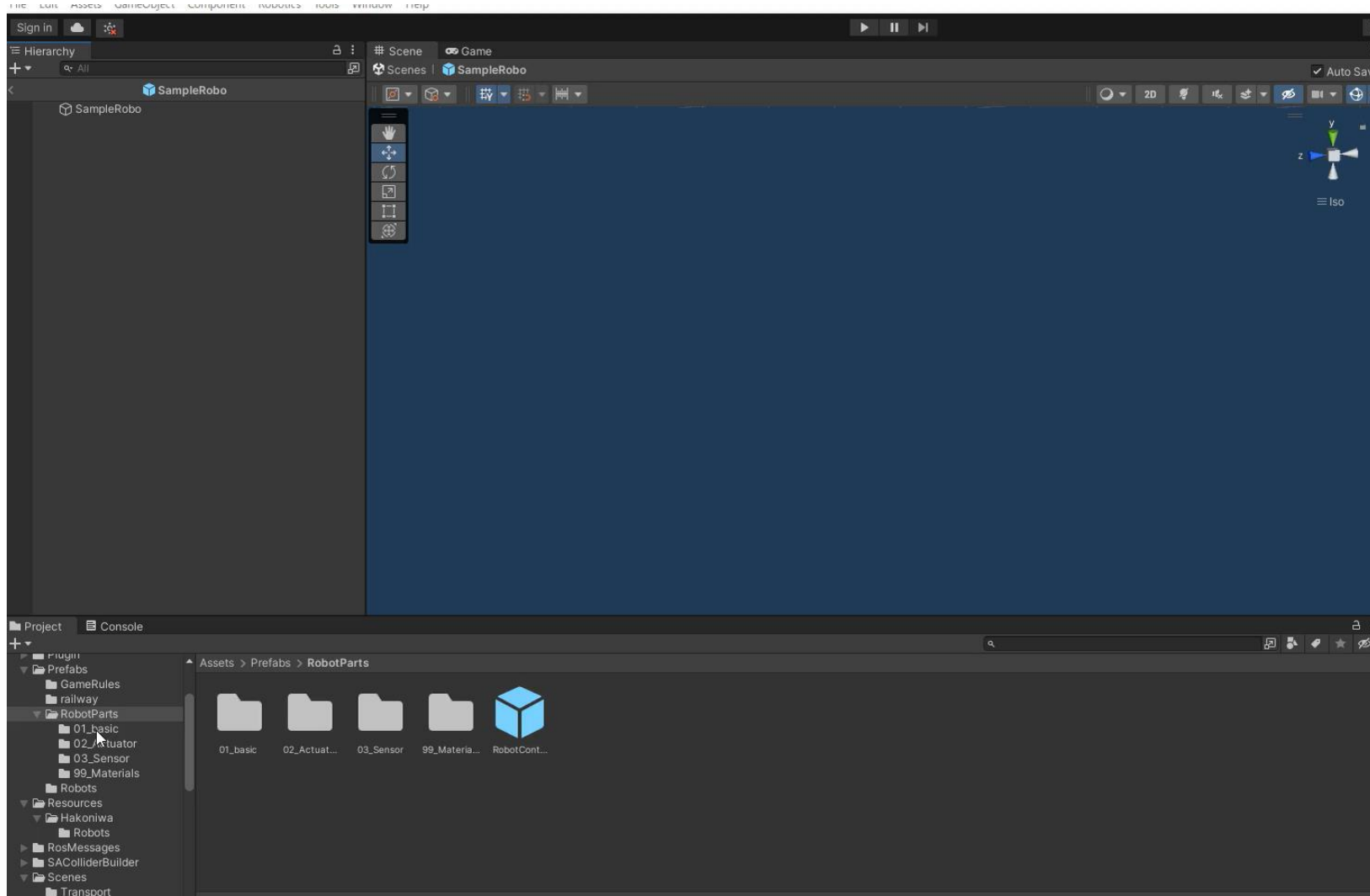
Unity上のロボット部品を組み立てて、ROS2でコントロール



- URL :
 - <https://qiita.com/kanetugu2018/items/29ee98bc434191eee8f1>

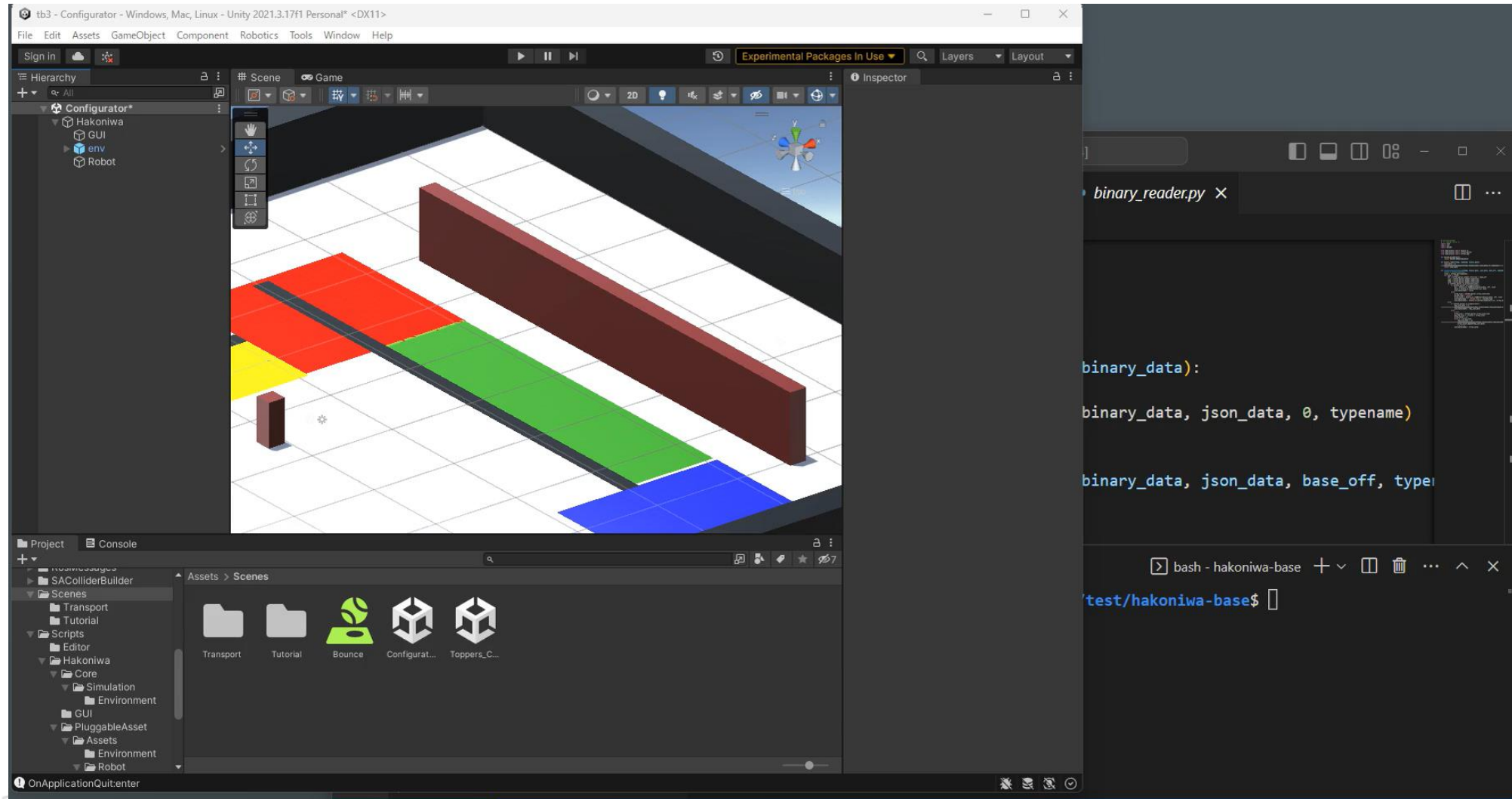
TurtleBot3風のロボットを作ってみよう！

53





作ったロボットを動かしてみよう！





ロボット制御用のPythonプログラム

```
while not done and total_time < 4000:

    f = open('dev/ai/cmd.txt', 'r')
    value = f.readlines()
    f.close()

    sensors = env.hako.execute()

    scan = robo.get_state("scan", sensors)
    scan_ranges = scan['ranges']
    scan_min = min(min(scan_ranges[0:15]), min(scan_ranges[345:359]))
    print("scan=" + str(scan_min))

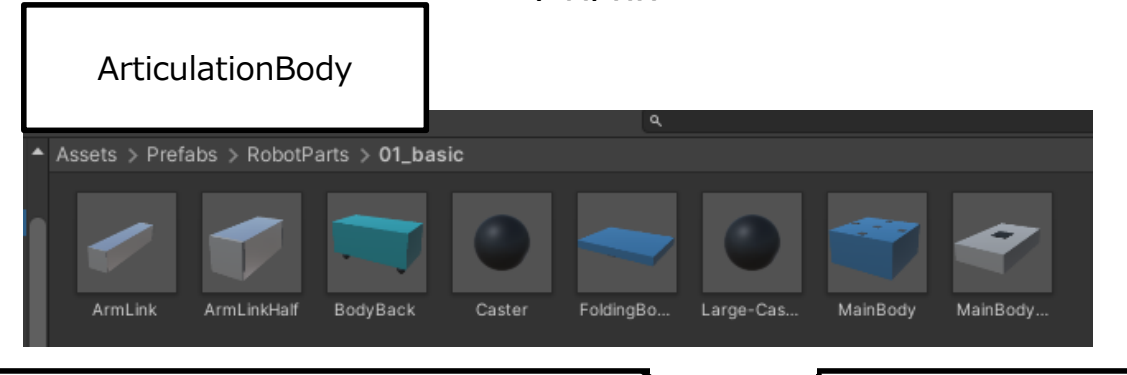
    if (scan_min >= 0.2):
        motor = robo.get_action('cmd_vel')
        motor['linear']['x'] = float(value[0])
        motor['angular']['z'] = 0.0
    else:
        motor = robo.get_action('cmd_vel')
        motor['linear']['x'] = 0.0
        motor['angular']['z'] = -2.0

    for channel_id in robo.actions:
        robo.hako.write_pdu(channel_id, robo.actions[channel_id])
```

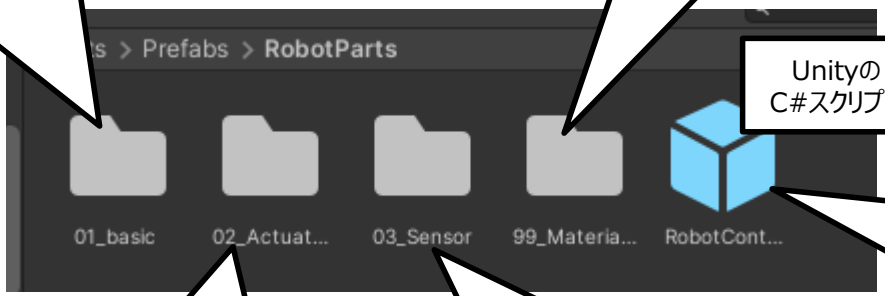


箱庭ロボット部品の構成

基本部品

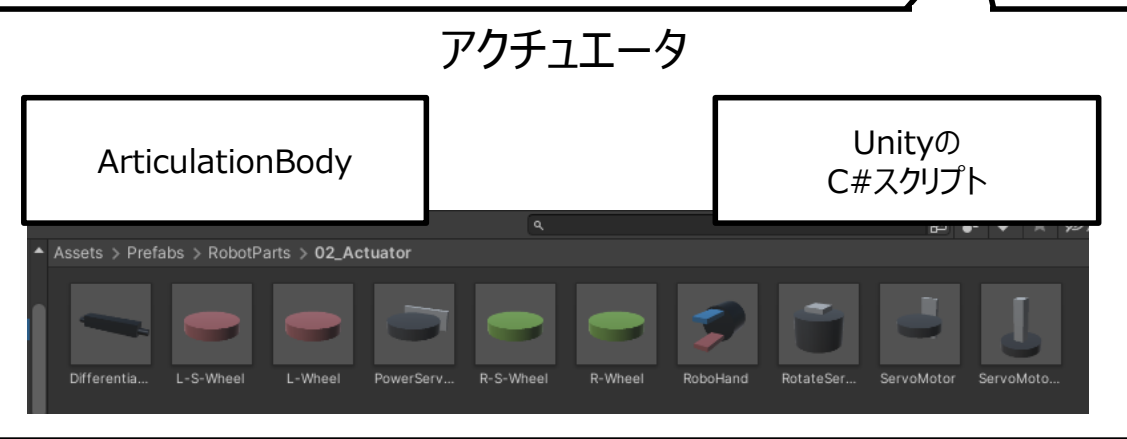


マテリアル

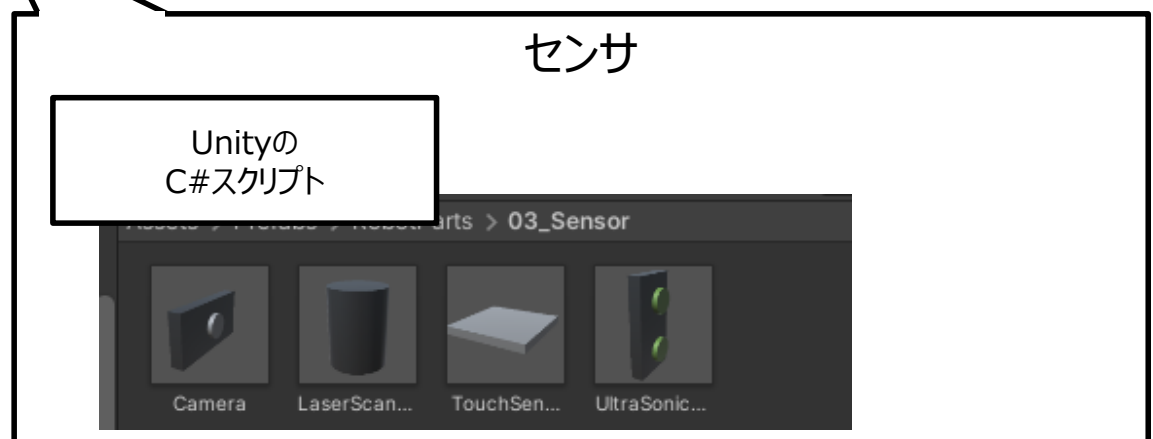


□ロボットルート
 □ロボット毎に1個割り当てする
 (□ロボット全体制御パーツ)

アクチュエータ



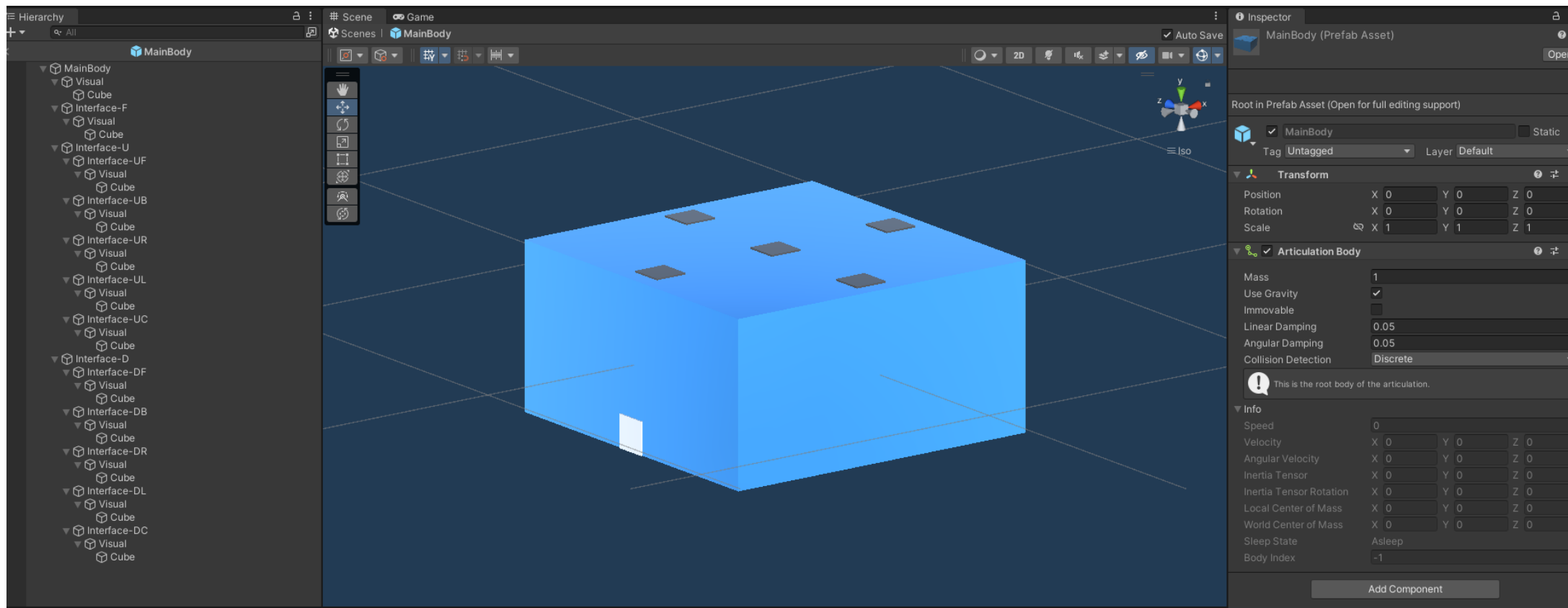
センサ





基本部品例(MainBody)

- ArticulationBodyで構成されている

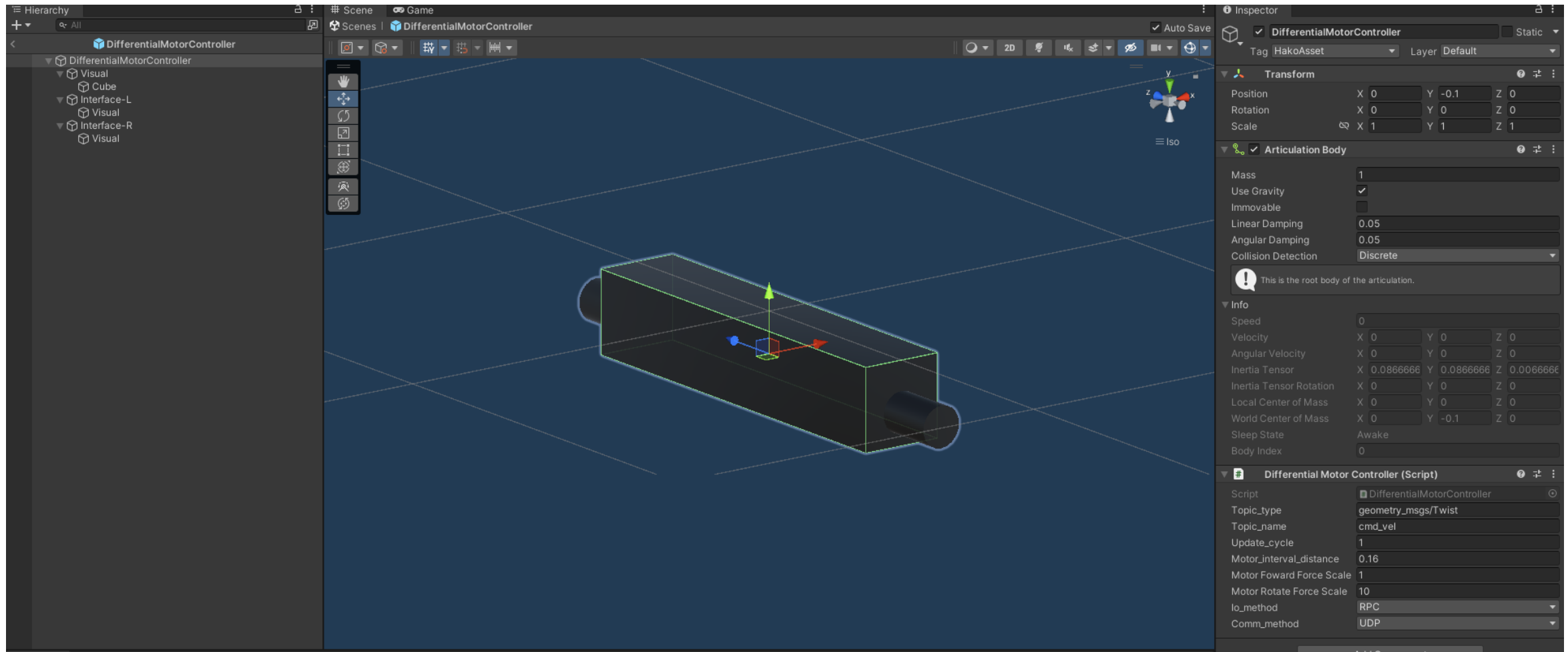


アクチュエータ例(DifferentialMotorController)

58



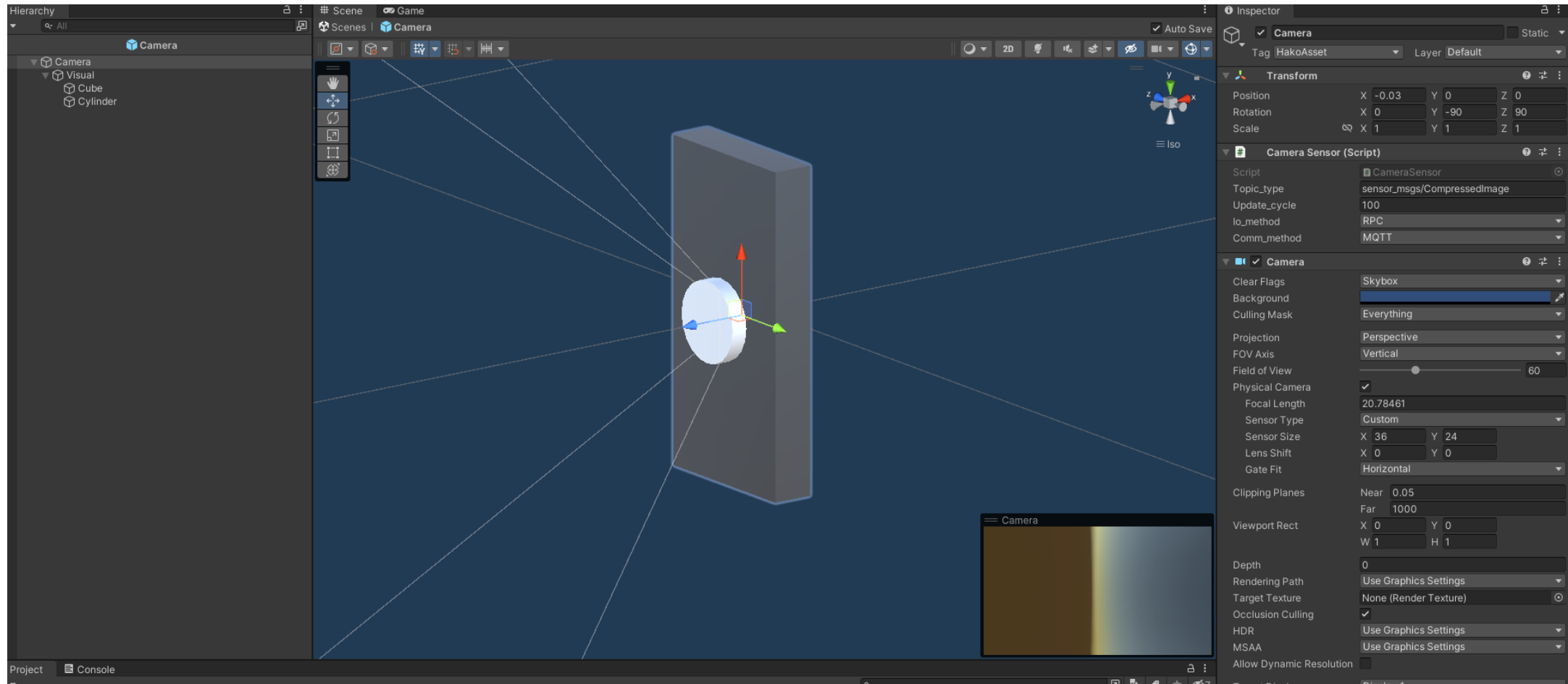
- ArticulationBodyとUnityスクリプトで構成されている





アクチュエータ例(Camera)

- CameraとUnityスクリプトで構成されている



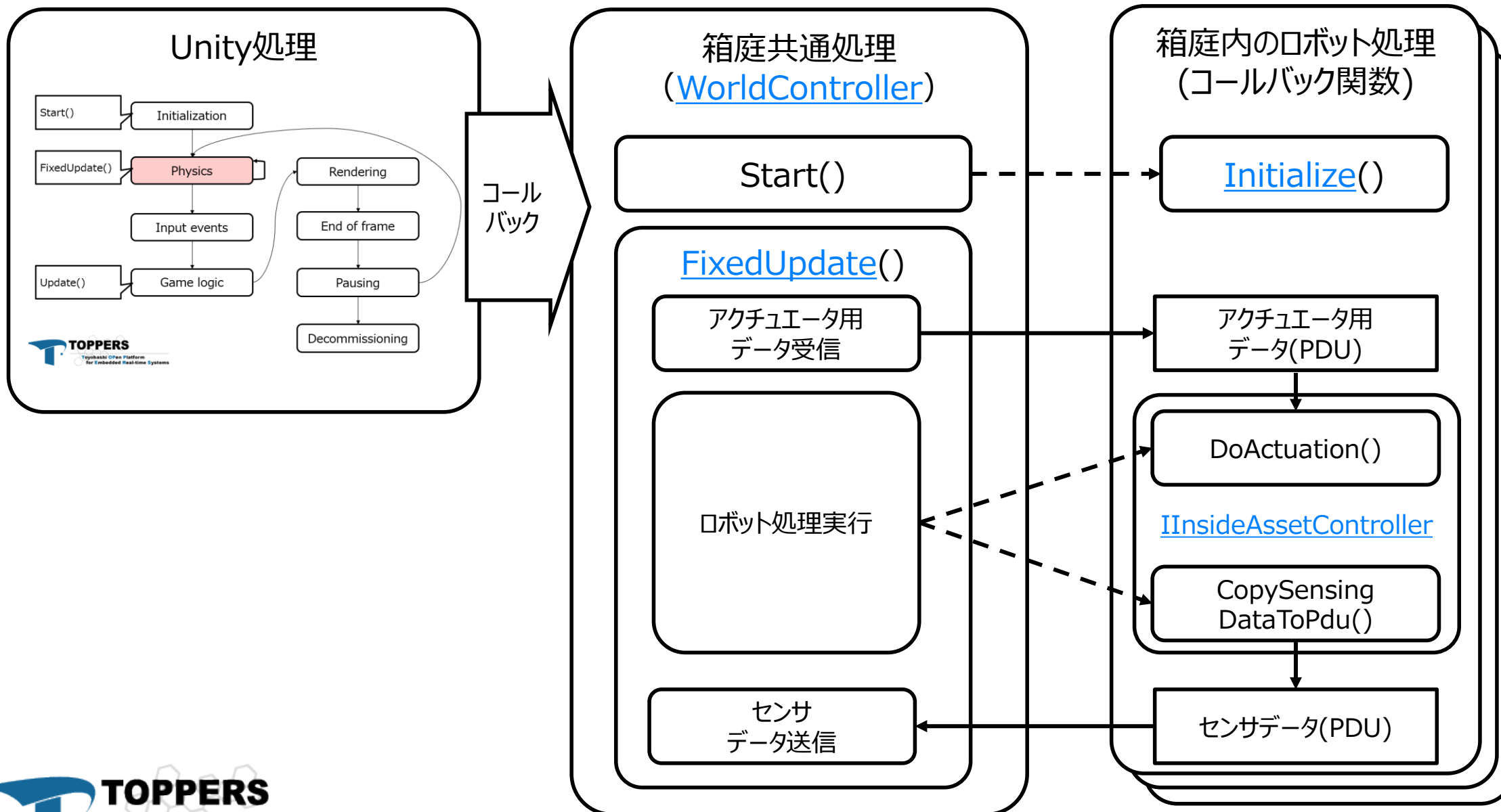


箱庭ロボット部品の構造

- 箱庭の全体的な処理の流れ
- 組み立てた箱庭ロボットの内部処理
- 箱庭ロボットの部品構造



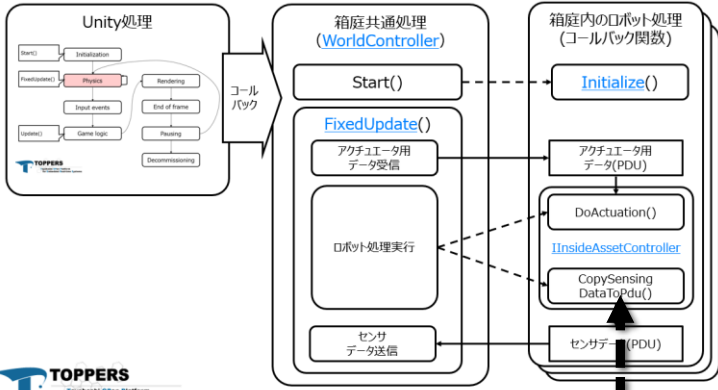
箱庭の全体的な処理の流れ





組み立てた箱庭ロボットの内部処理

- RobotPartsRootがアクチュエータ/センサ部品群を統括し、処理実行する
 - 各種アクチュエータ/センサはインターフェース化されているため、RobotPartsRootは再利用可能
 - 箱庭部品として、アクチュエータ/センサのプロトタイプ実装クラスを用意している



IRobotPartsController(インターフェース)

【役割】
アクチュエータの制御コントローラ用のインターフェース

【インターフェース仕様】
DoControl()関数実装では、PDUデータを取得し、アクチュエータ指示値を関係するアクチュエータに伝達する

IRobotPartsMotor(インターフェース)

【役割】
モーター制御用のインターフェース

【インターフェース仕様】
SetTargetVelocity()関数実装では、指示値に従って、モーターの物理挙動を実現する

**IInsideAssetController
インターフェース実装**

【役割】
すべてのセンサ/アクチュエータの処理を実行する親玉

【補足】
ロボット組み立て時には、本Unityスクリプトを必ずアタッチしないとイケない

RobotPartsRoot

DoActuation()
CopySensing Data To Pdu()

IRobotPartsController

DoControl()

IRobotPartsSensor

UpdateSensor Values()

IRobotPartsSensor(インターフェース)

【役割】
センサ用のインターフェース

【インターフェース仕様】
UpdateSensorValues()関数実装では、センシング処理を実行し、取得したデータをPDUに保存する

IRobotPartsMotor
SetTargetVelocity()

IRobotPartsPincherFinger
UpdateGrip()

IRobotPartsActuator

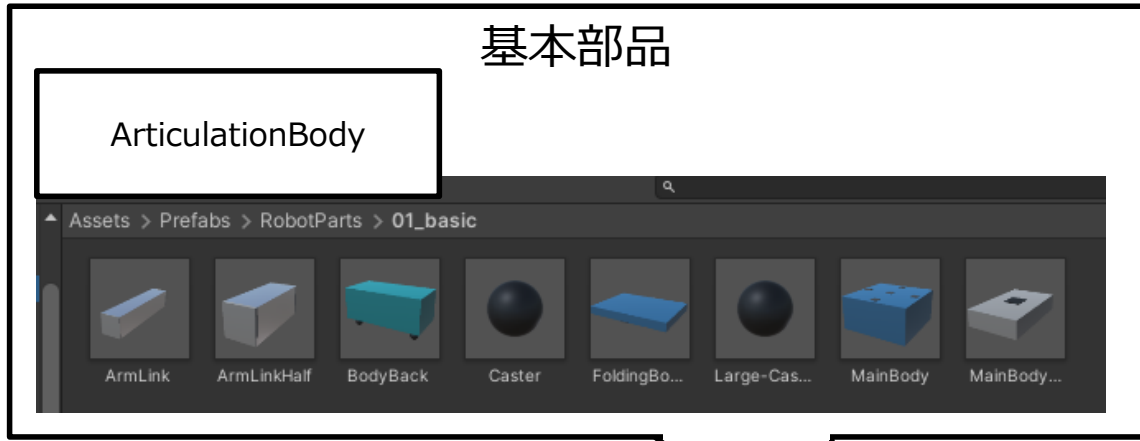
IRobotPartsActuator(インターフェース)

【役割】
アクチュエータ用の共通インターフェース

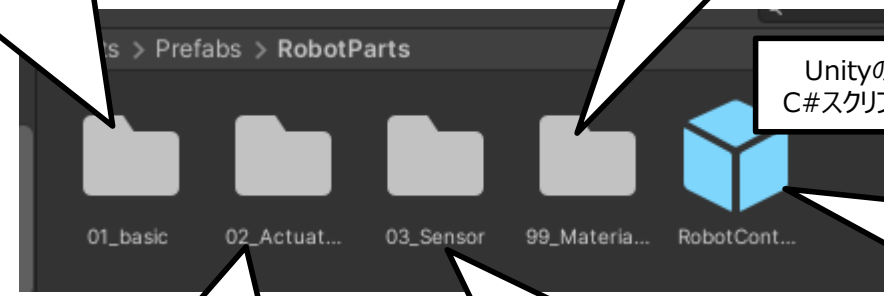


箱庭ロボット部品の構成 (もう一回)

基本部品



マテリアル

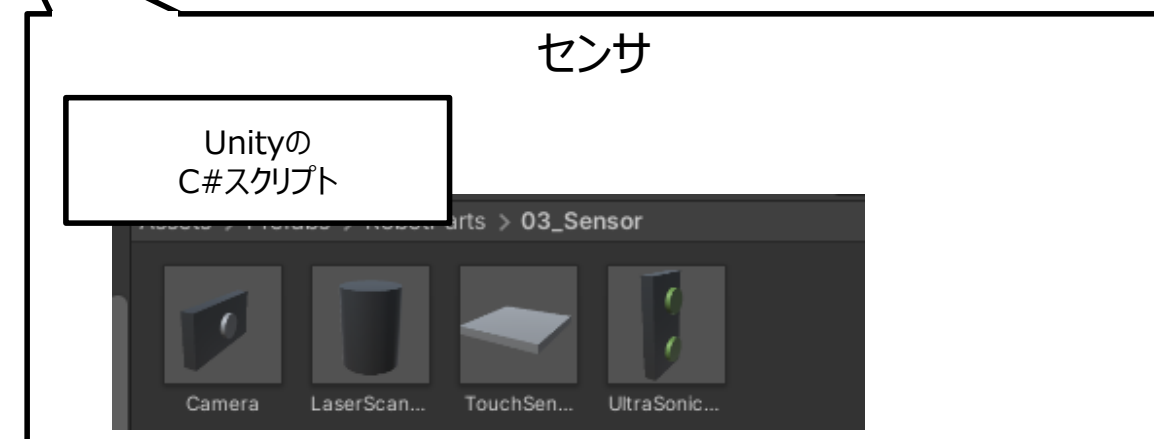


□ロボットルート
□ロボット毎に1個割り当てする
(□ロボット全体制御パーツ)

アクチュエータ

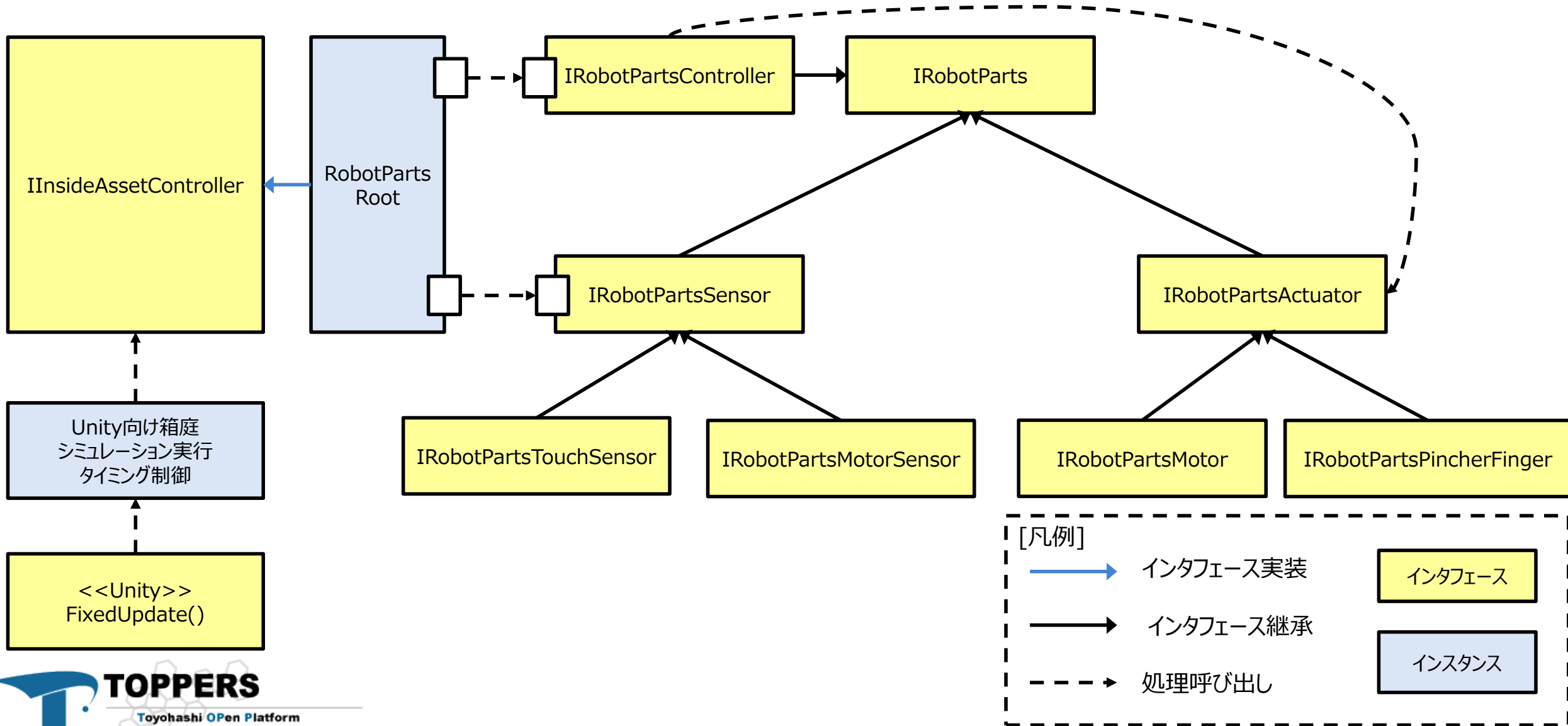


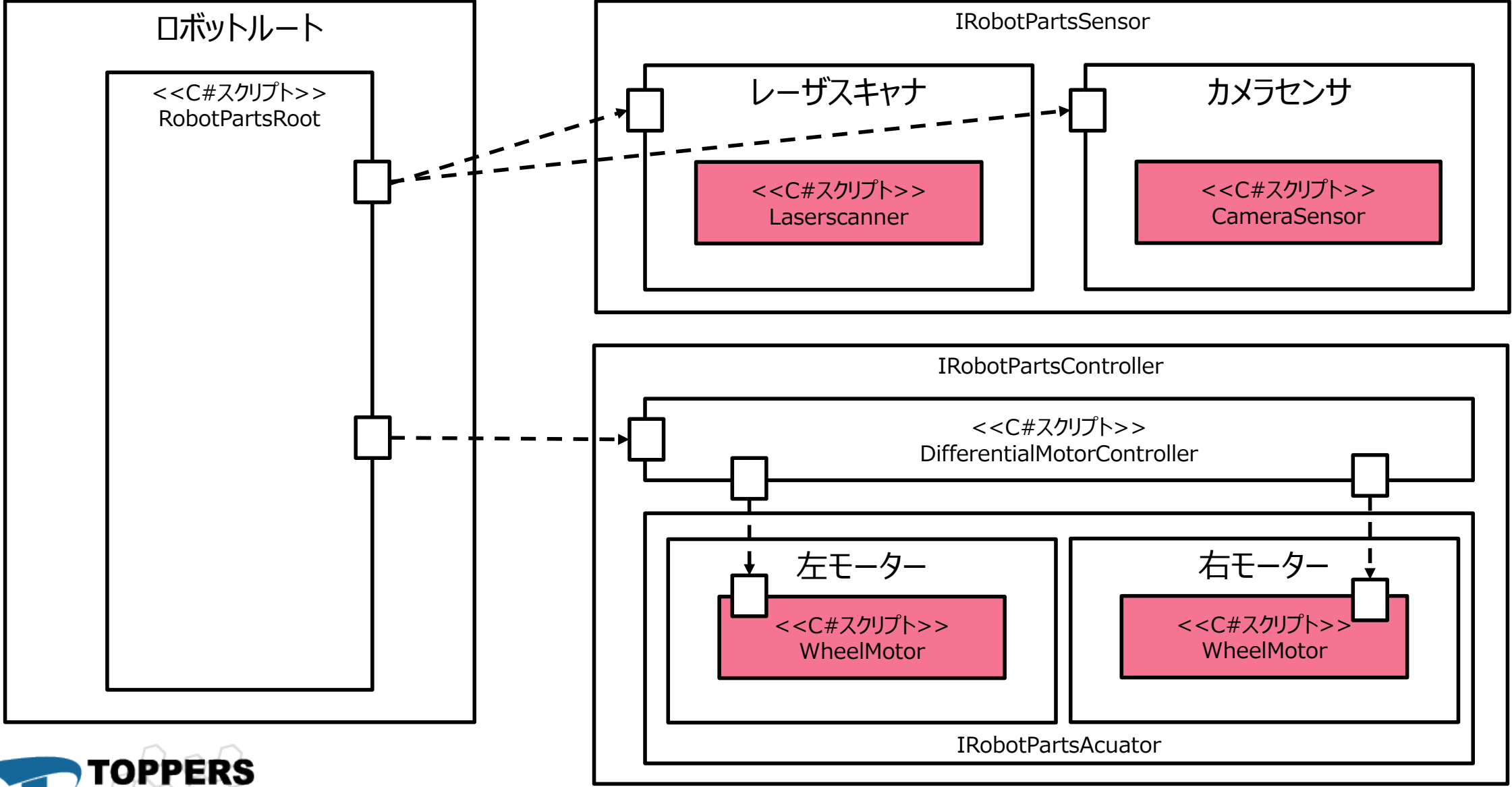
センサ

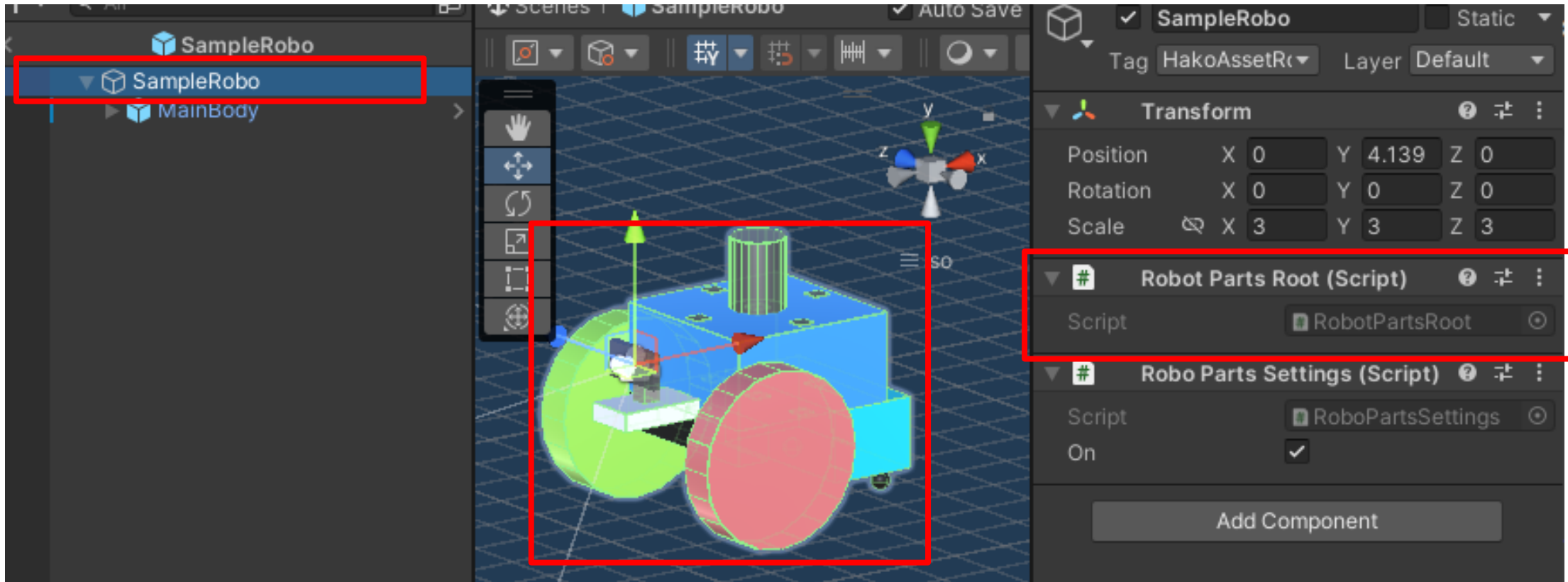




イベント関数インタフェース視点での全体像

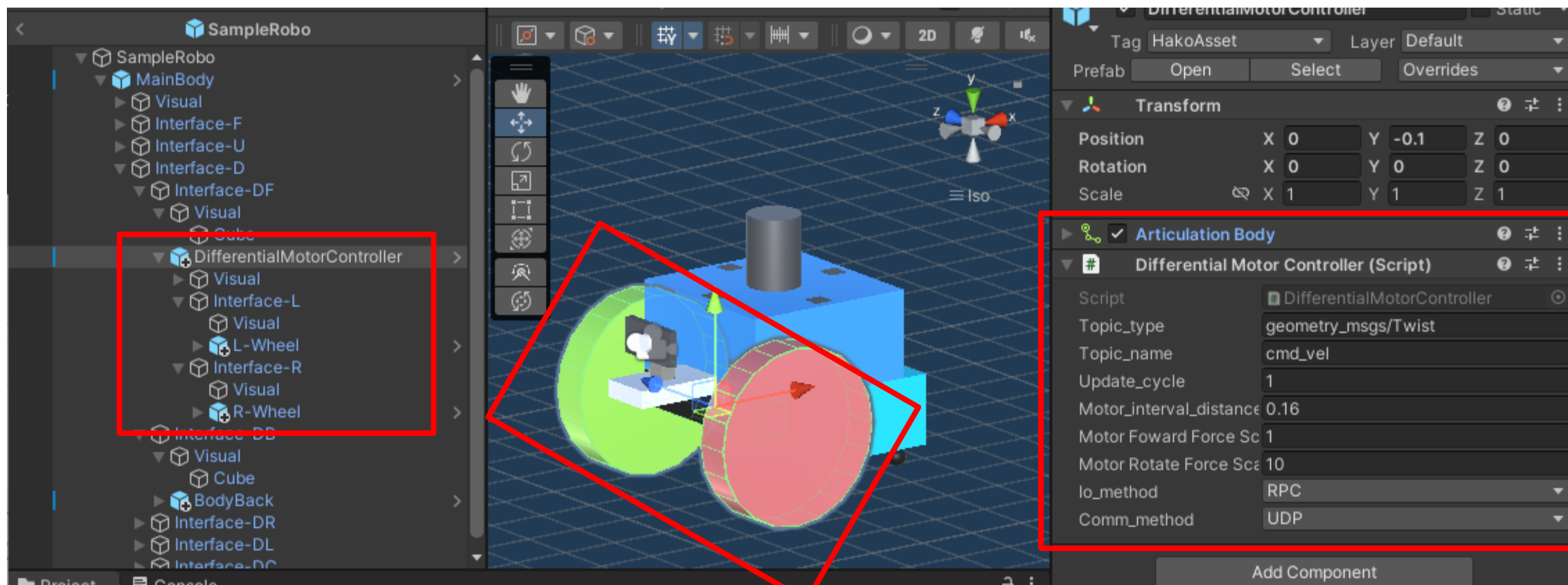








IRobotPartsController : DifferentialMotorController



I Robot Parts Actuator : WheelMotor

68



The screenshot displays a 3D software environment with a robot model in the center. The interface is divided into three main sections:

- Left Panel (Hierarchy):** Shows a tree view of the robot's components. The 'L-Wheel' component is highlighted with a red box.
- Center Panel (3D View):** Shows the robot model. A red box highlights the 'L-Wheel' component, which is a red sphere with a yellow cube at its center.
- Right Panel (Properties):** Shows the properties of the selected 'L-Wheel' component. The 'Articulation Body' section is highlighted with a red box, showing the following properties:

| Property | Value |
|--------------|-----------------------------|
| Script | WheelMotor |
| Motor_radius | 3.3 |
| Wheel | L-Wheel (Articulation Body) |
| Force Limit | 10 |
| Damping | 2 |



IRobotPartsSensor : Laserscanner

The screenshot displays the ROS2 GUI interface for configuring a **LaserScanner** component on a robot model. The interface is divided into three main sections:

- Left Panel (Hierarchy):** Shows the tree structure of the robot model. The **LaserScanner** component is highlighted with a red box.
- Center Panel (3D View):** Shows a 3D view of the robot model. A red box highlights the **LaserScanner** component, which is a cylinder with a green laser beam.
- Right Panel (Properties):** Shows the properties of the selected component. The **Laser Scanner (Script)** component is highlighted with a red box. The properties are as follows:

| Property | Value |
|------------------------|-----------------------|
| Tag | HakoAsset |
| Layer | Default |
| Prefab | Open |
| Select | Select |
| Overrides | Overrides |
| Transform | |
| Position | X 0 Y 0.2 Z 0 |
| Rotation | X 0 Y -90 Z 0 |
| Scale | X 1 Y 1 Z 1 |
| Laser Scanner (Script) | |
| Script | LaserScanner |
| Scale | 1 |
| Max_count | 360 |
| View_interval | 36 |
| Topic_type | sensor_msgs/LaserScan |
| Topic_name | scan |
| Update_cycle | 10 |
| Io_method | RPC |
| Comm_method | MQTT |

The **Add Component** button is visible at the bottom of the right panel.



I RobotPartsSensor : CameraSensor

The screenshot displays the ROS2 GUI interface for a robot model named 'SampleRobo'. The left sidebar shows the object tree, with the 'Camera' object under 'MainBodySubHolder' highlighted in red. The central 3D view shows the robot model with a camera sensor attached to its front. The right sidebar shows the HakoAsset Inspector for the 'Camera Sensor (Script)' object, with the following configuration:

| Property | Value |
|------------------|-----------------------------|
| Script | CameraSensor |
| Topic_type | sensor_msgs/CompressedImage |
| Update_cycle | 100 |
| Io_method | RPC |
| Comm_method | MQTT |
| Camera (checked) | |
| Clear Flags | Skybox |
| Background | [Color Picker] |
| Culling Mask | Everything |
| Projection | Perspective |
| FOV Axis | Vertical |