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)

TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

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

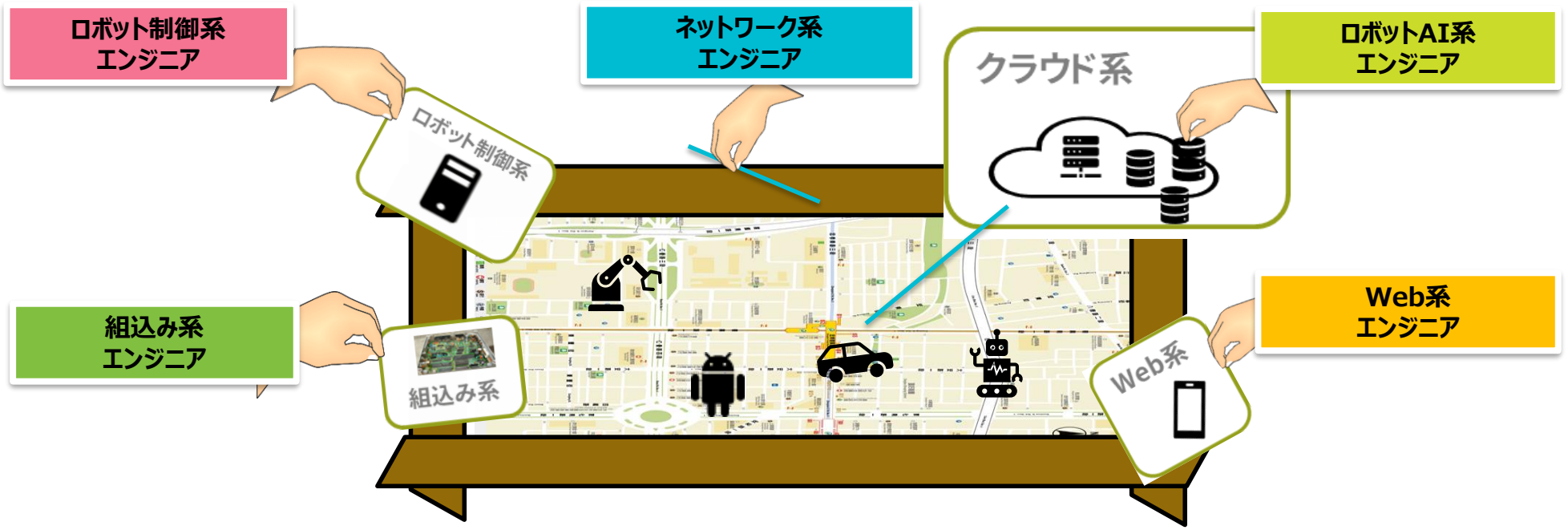
箱庭で１からロボットを作ってみよう！

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

TOPPERS
Toyohashi OPen Platform
for Embedded Real-time Systems

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

結合環境構築
(箱庭ベース環境)

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

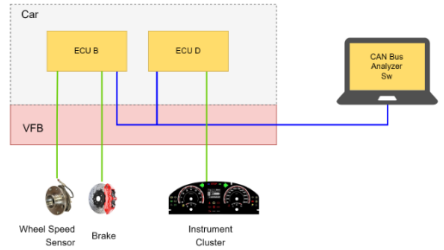
箱庭ユーザを
広める活動

箱庭チュートリアル開発
&&広報活動

2025 目指せ大阪万博！

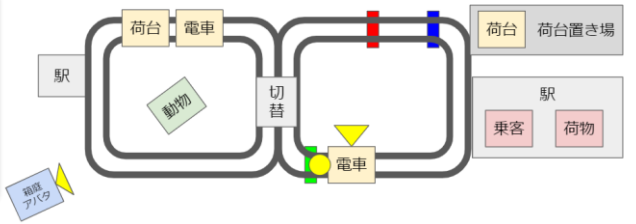
組込み系エンジニア

ECU/AUTOSAR開発入門シミュレーション環境



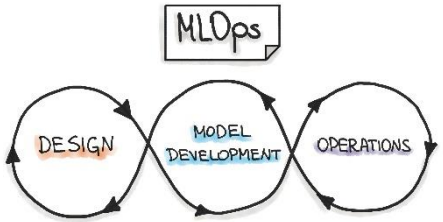
ロボット制御系エンジニア

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



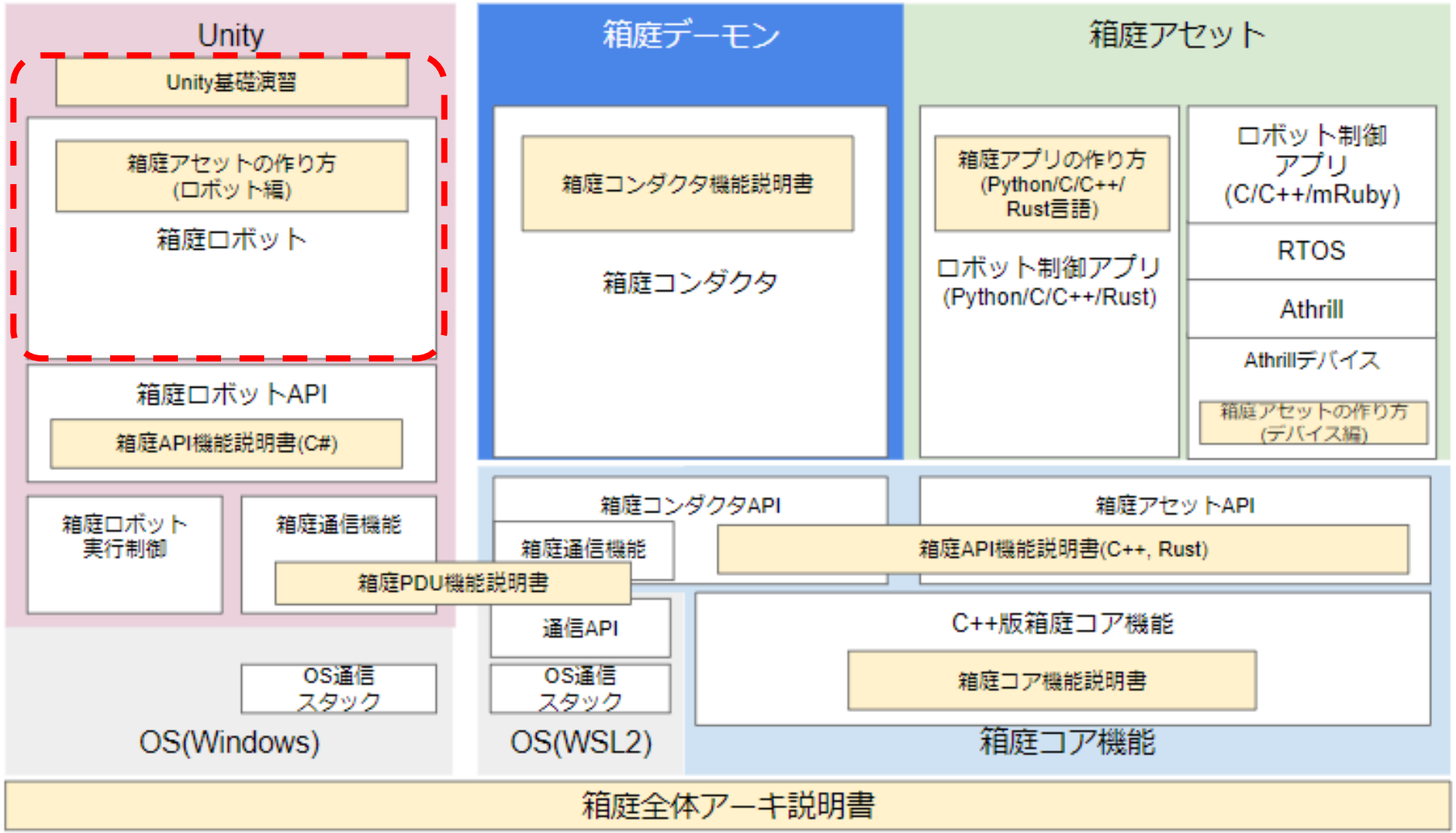
ロボットAI系エンジニア

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



TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

現状の箱庭の全体像（Windows版）

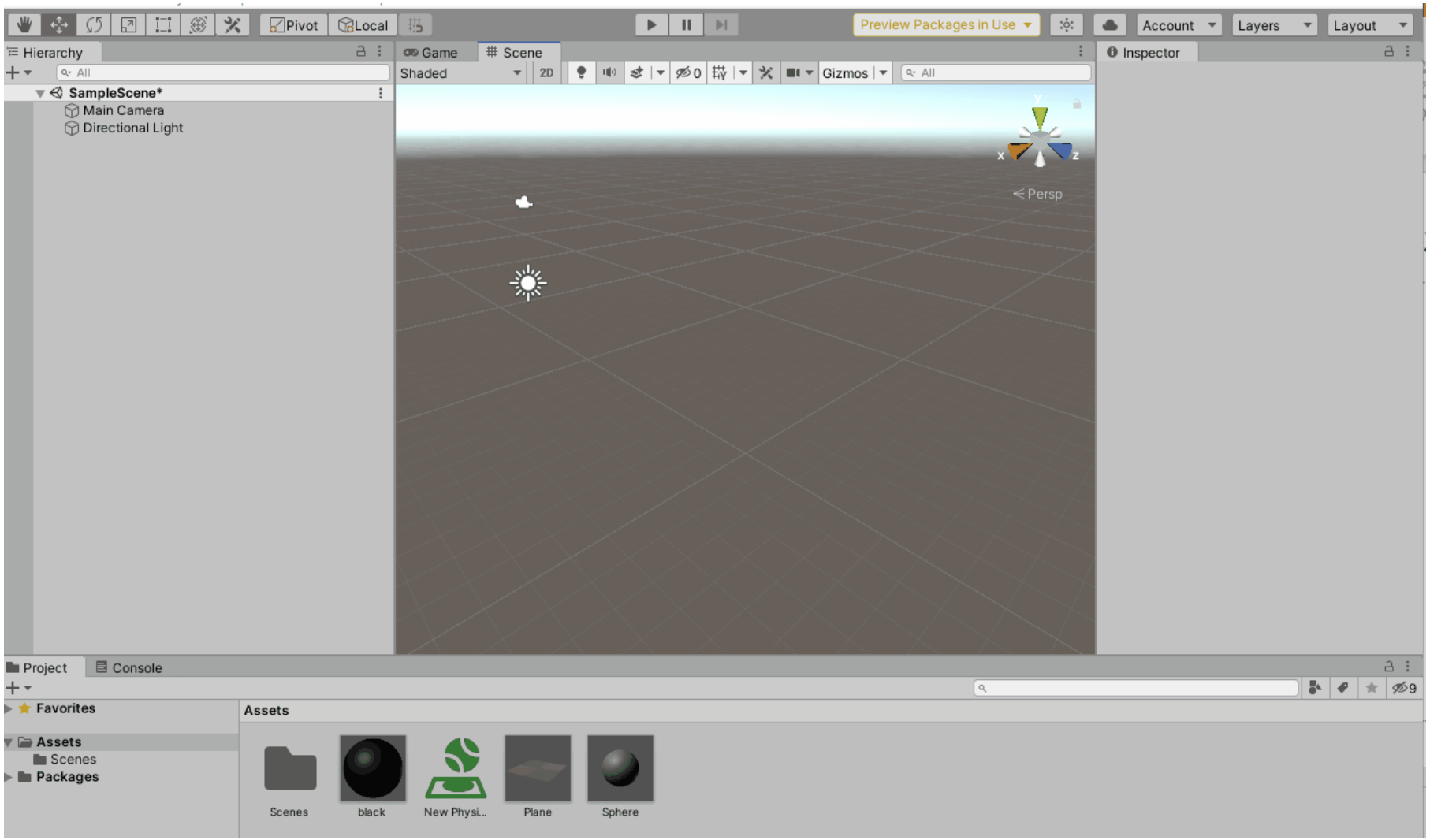


Unityとは何か

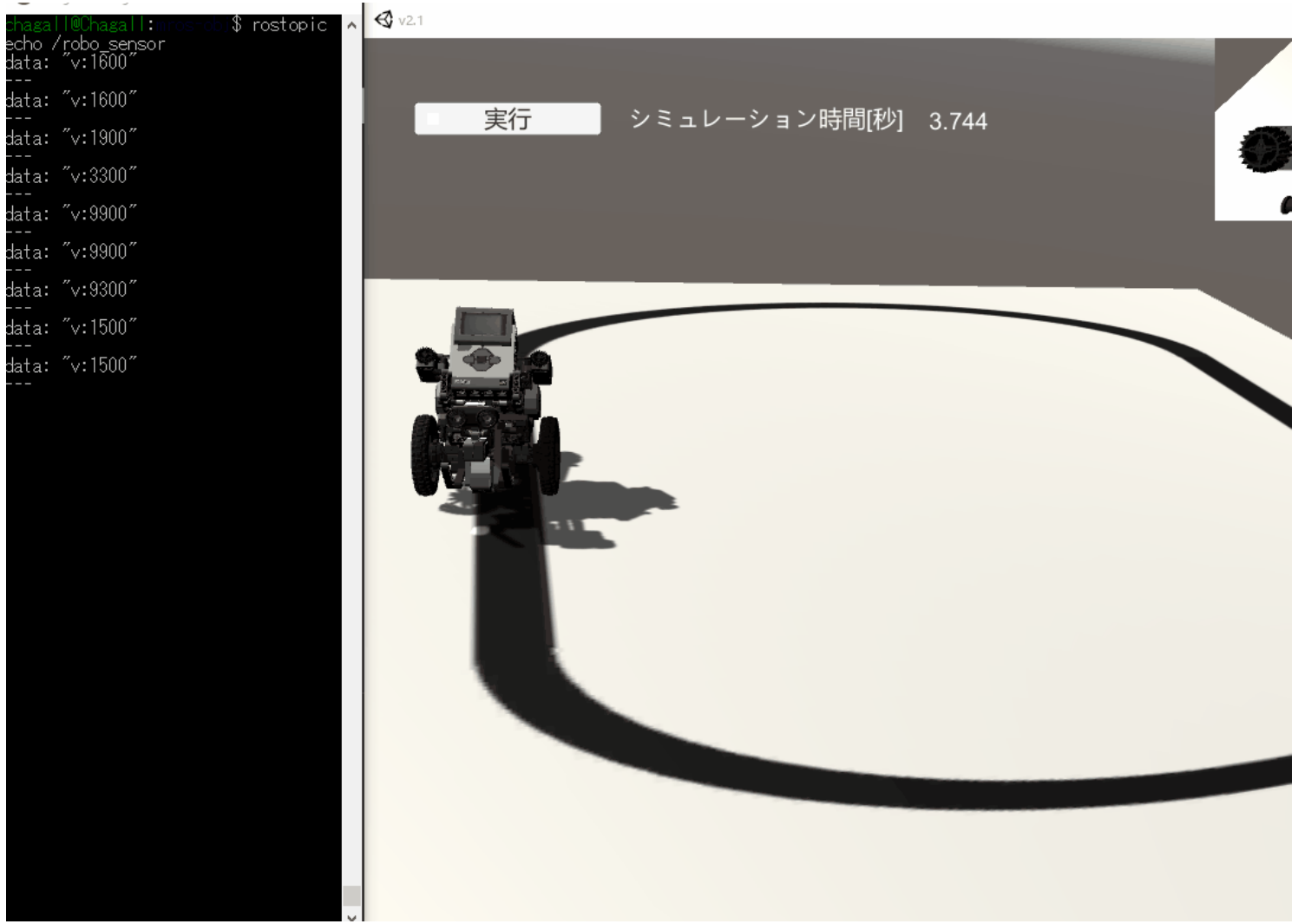
- ユニティ・テクノロジーズ社が開発したゲーム開発プラットフォーム
  - 物理演算エンジンが搭載されています
    - 質量・速度・摩擦・風といった，古典力学的な法則をシミュレーションするコンピュータのソフトウェア
    - ※参照元：ウィキペディア

- 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



ロボットも動かすことができる



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

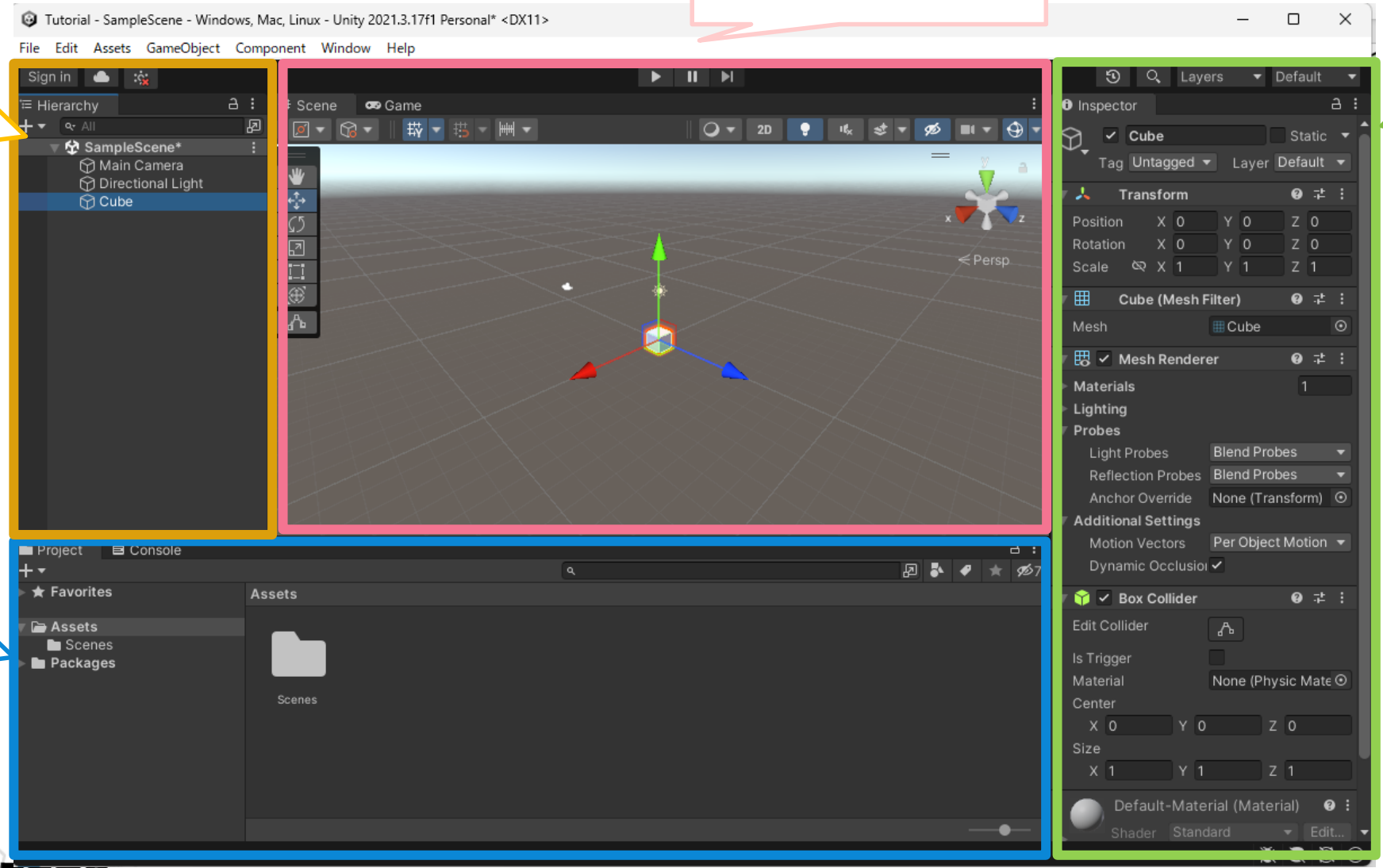
Unityエディタの画面構成（１／２）

Hierarchy
ビュー

Sceneビュー

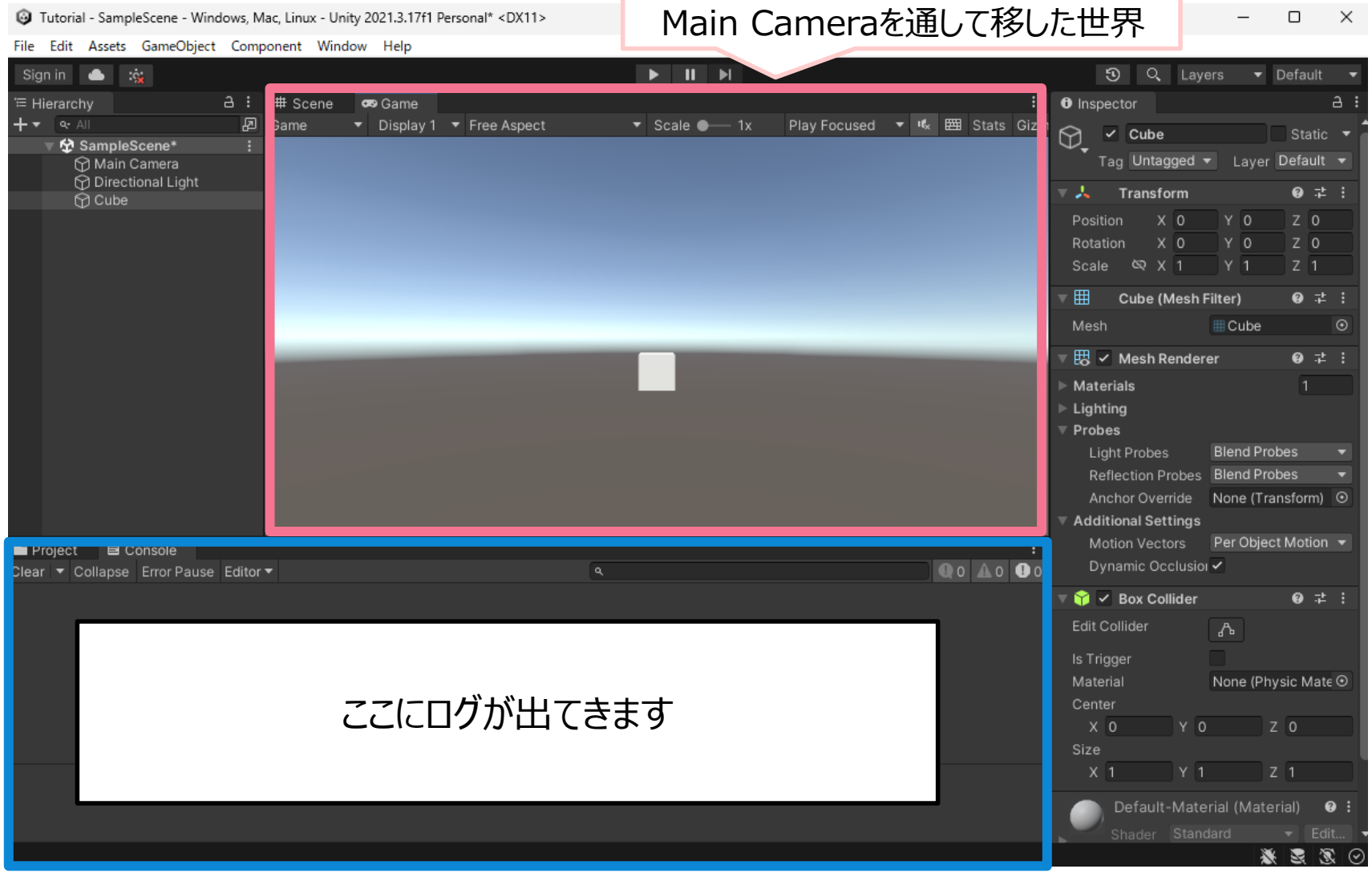
Inspector
ビュー

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



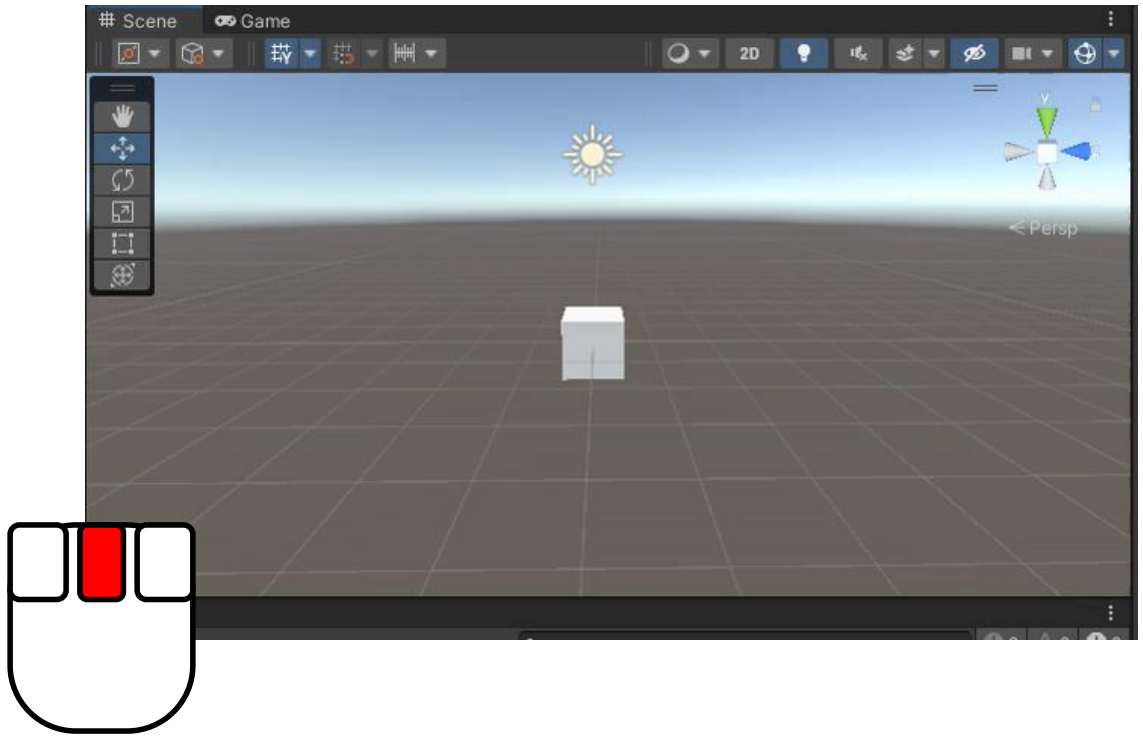
Console
ビュー

基本的な操作方法

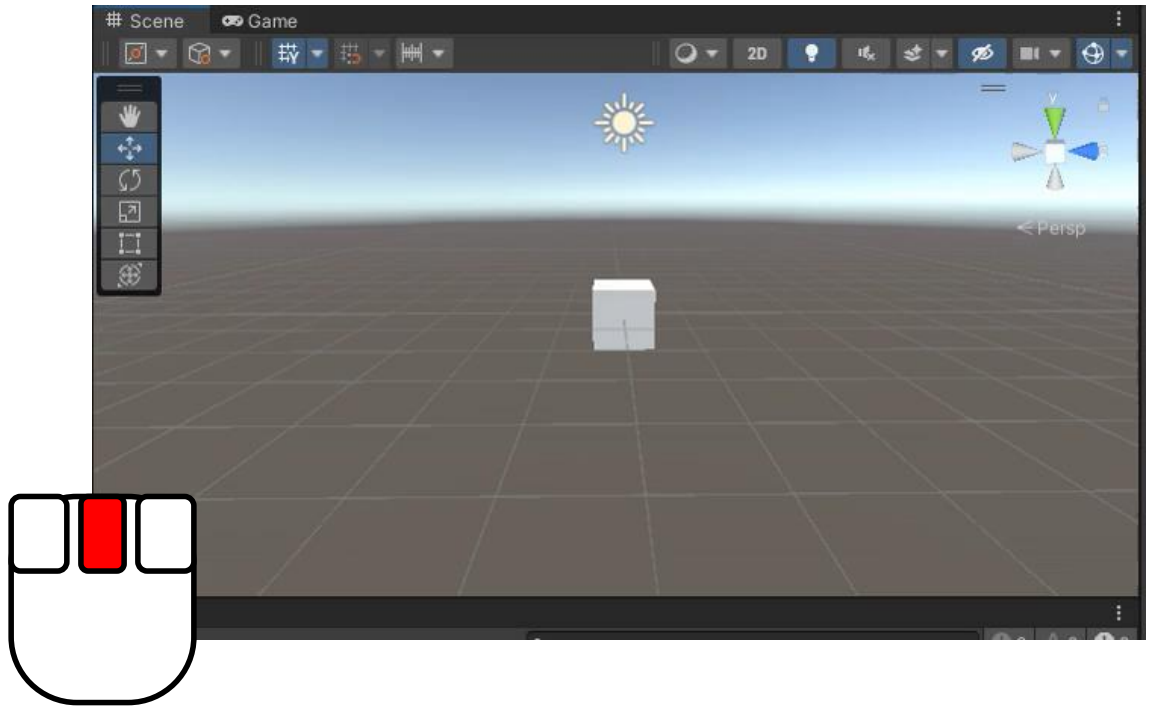
- 視点の操作
  - 視点移動
  - 視点回転
  - ズーム操作
- ゲーム・オブジェクトの操作
  - ゲーム・オブジェクト移動
  - ゲーム・オブジェクト回転
  - ゲーム・オブジェクト拡大・縮小

視点移動

マウス・ホイール押しながら右左

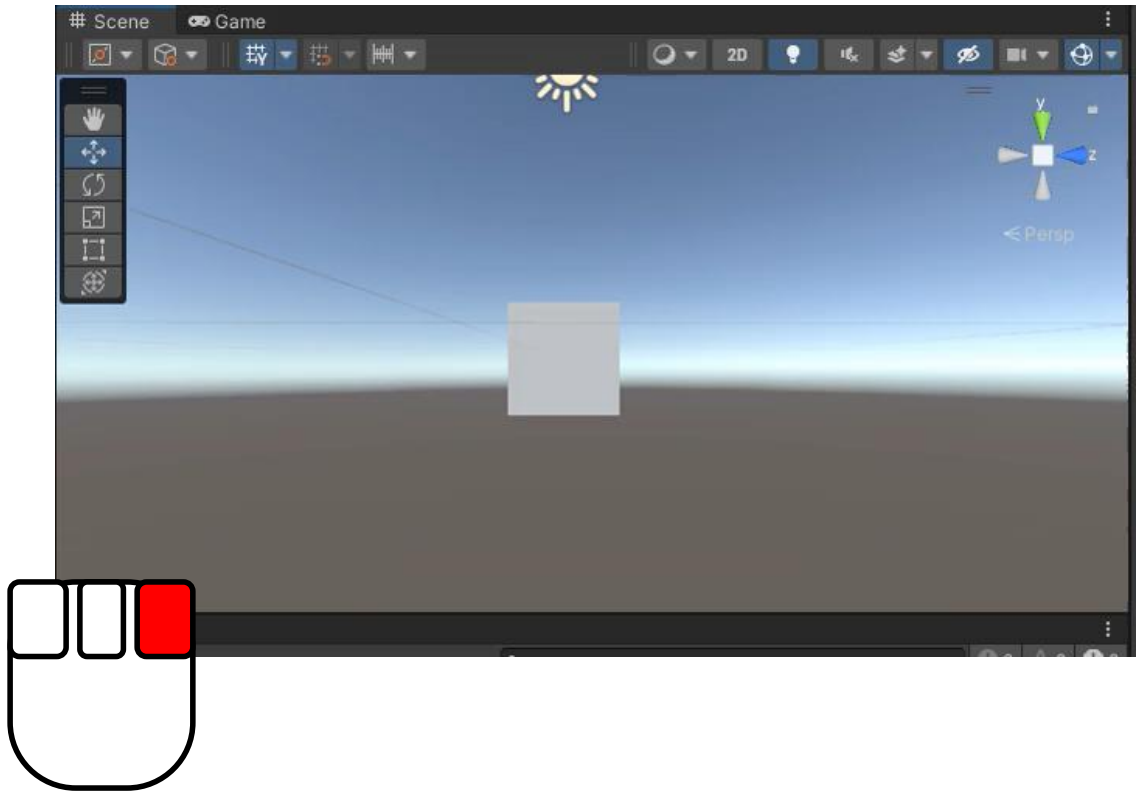


マウス・ホイール押しながら上下

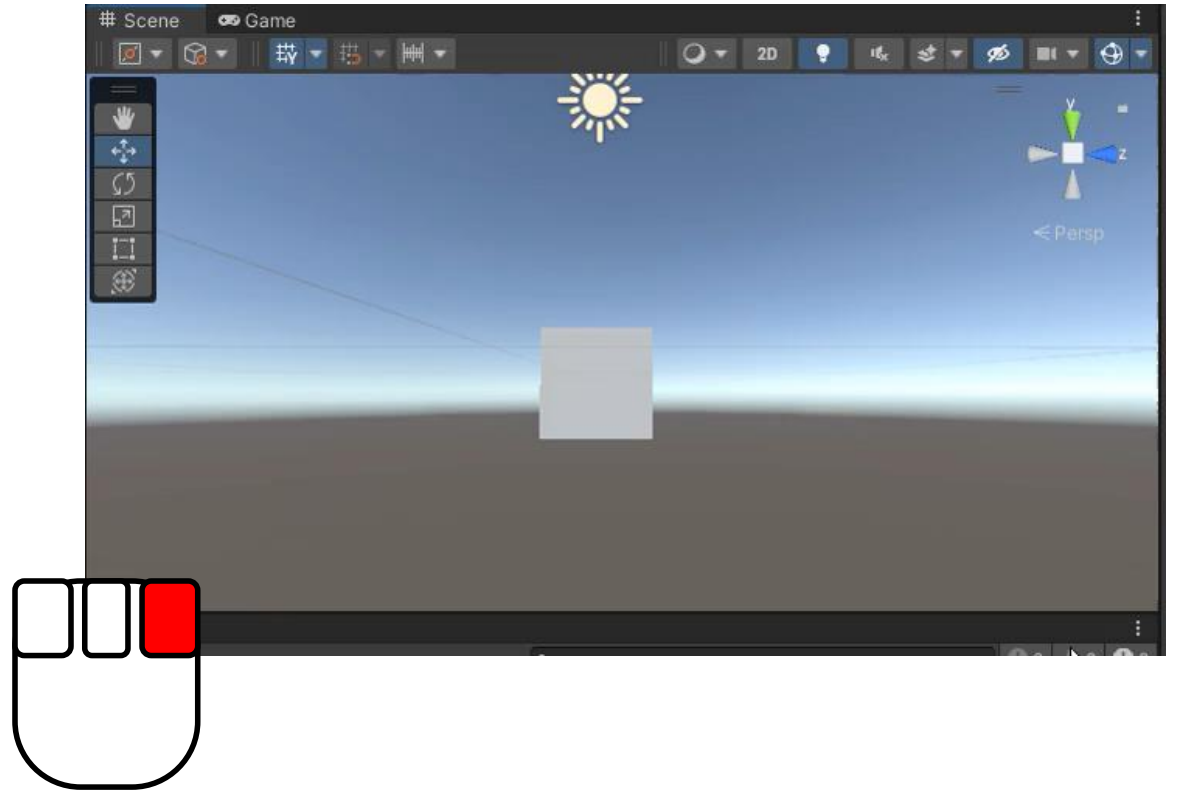


視点回転

右クリックしながら右左

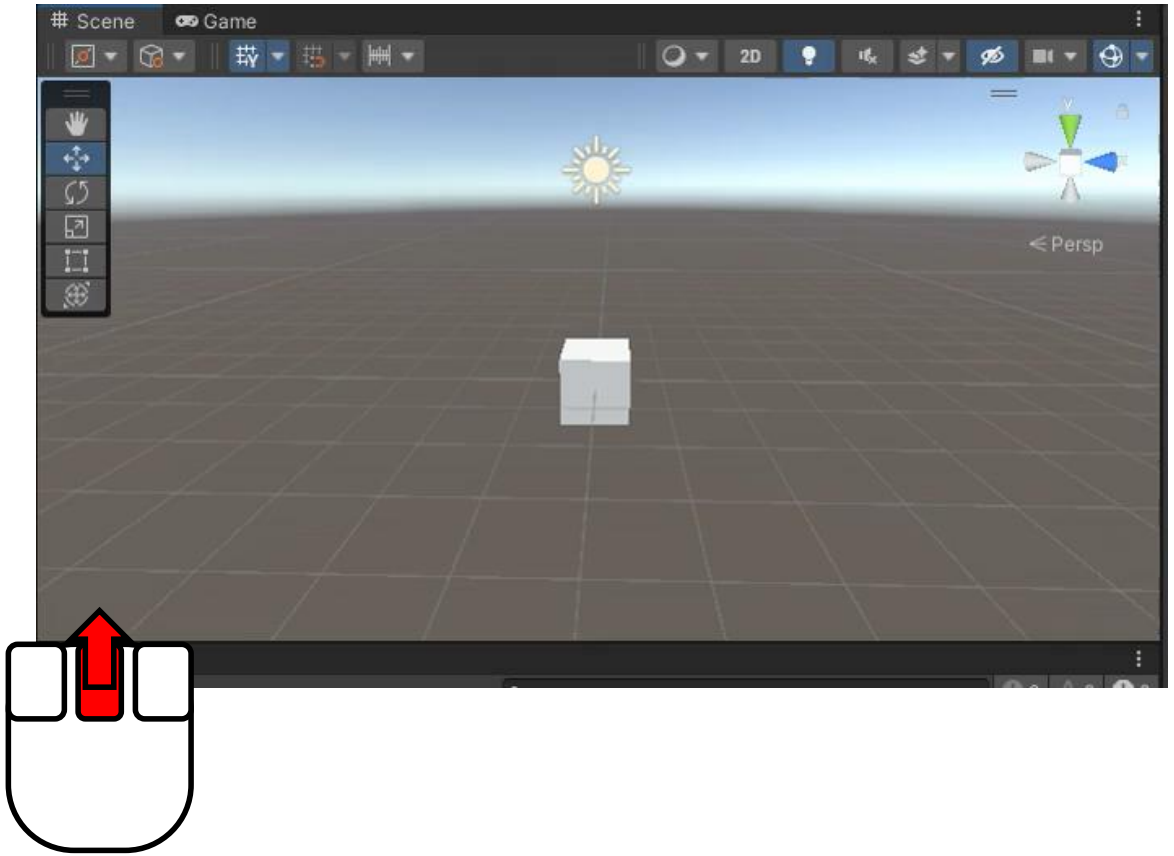


右クリック押しながら上下

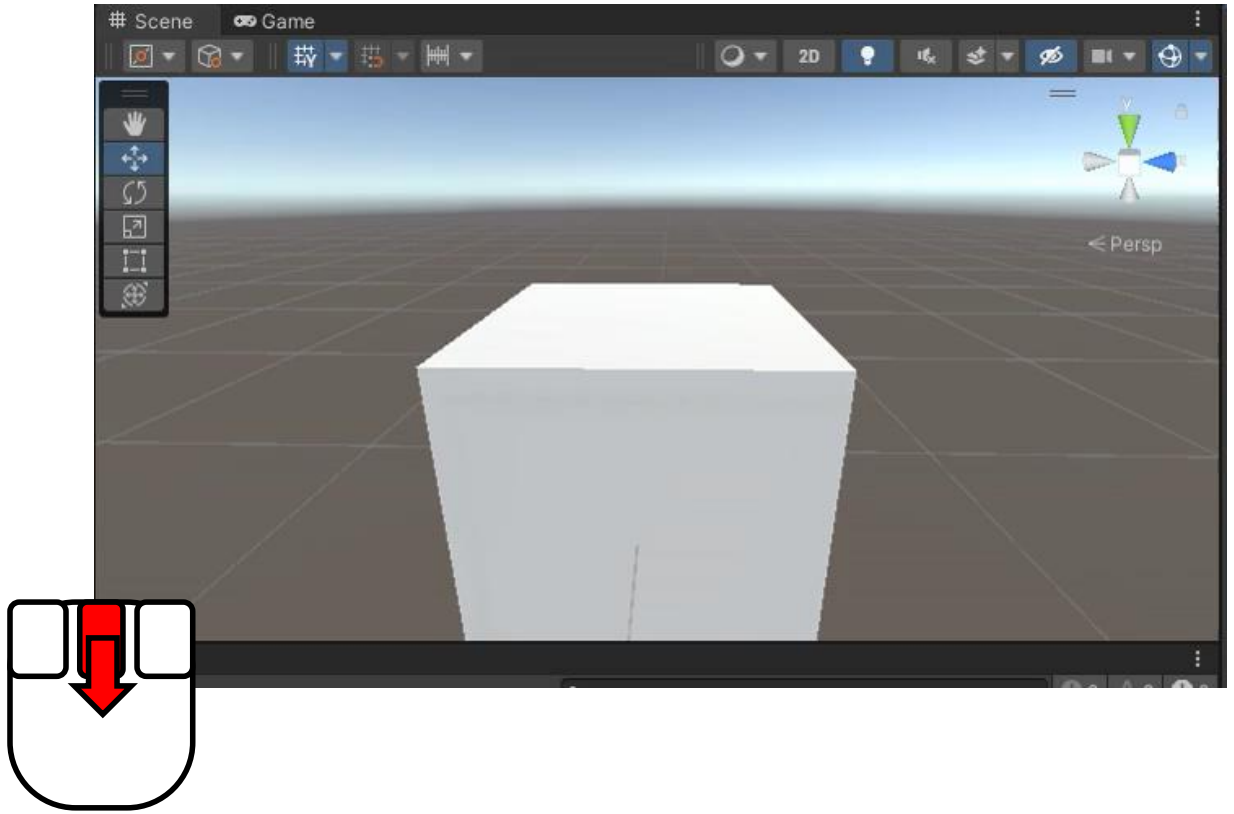


ズーム操作

ズームイン

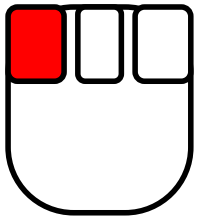


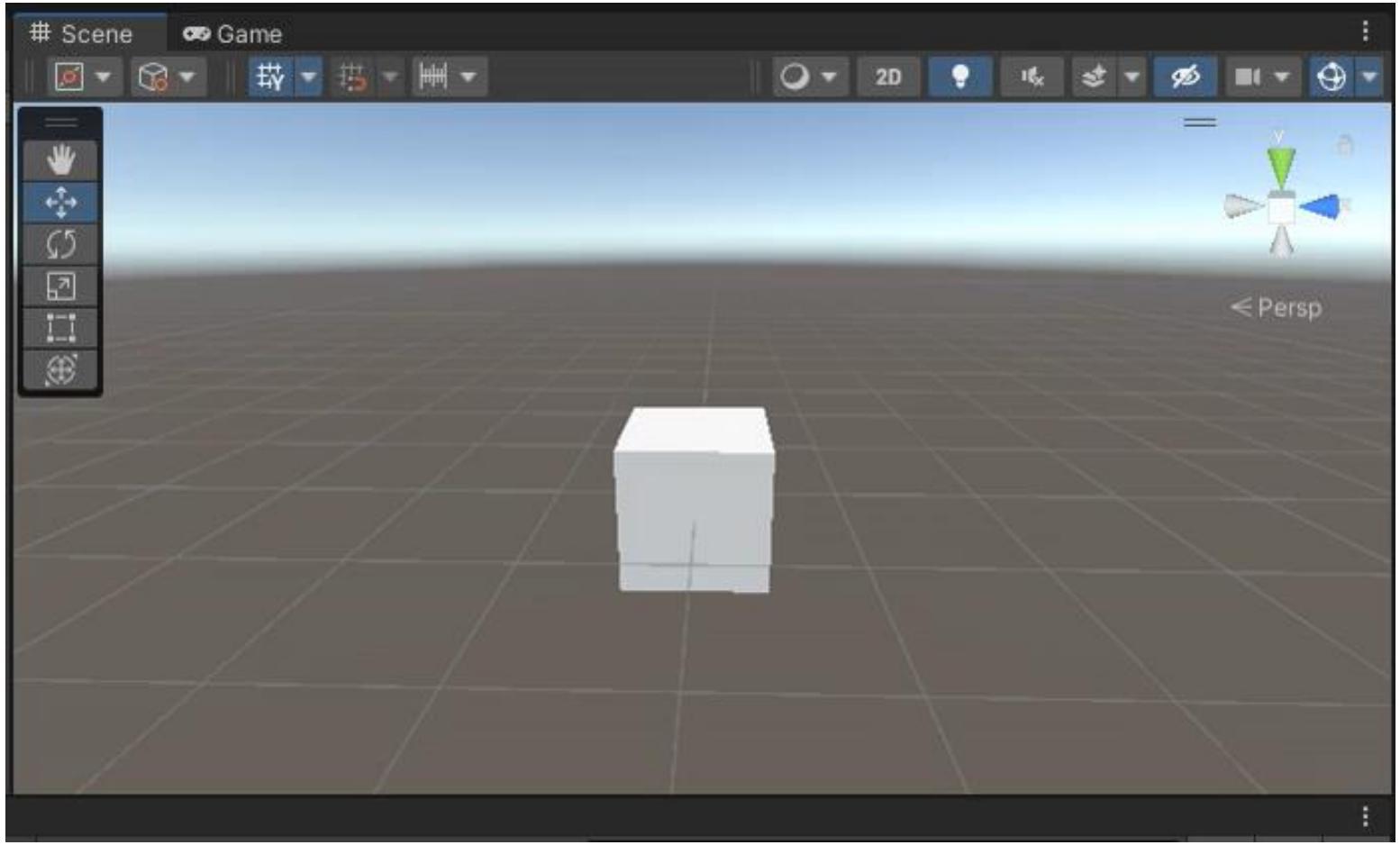
ズームアウト



ゲーム・オブジェクト移動

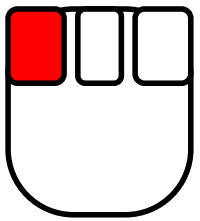
左クリックしながら移動

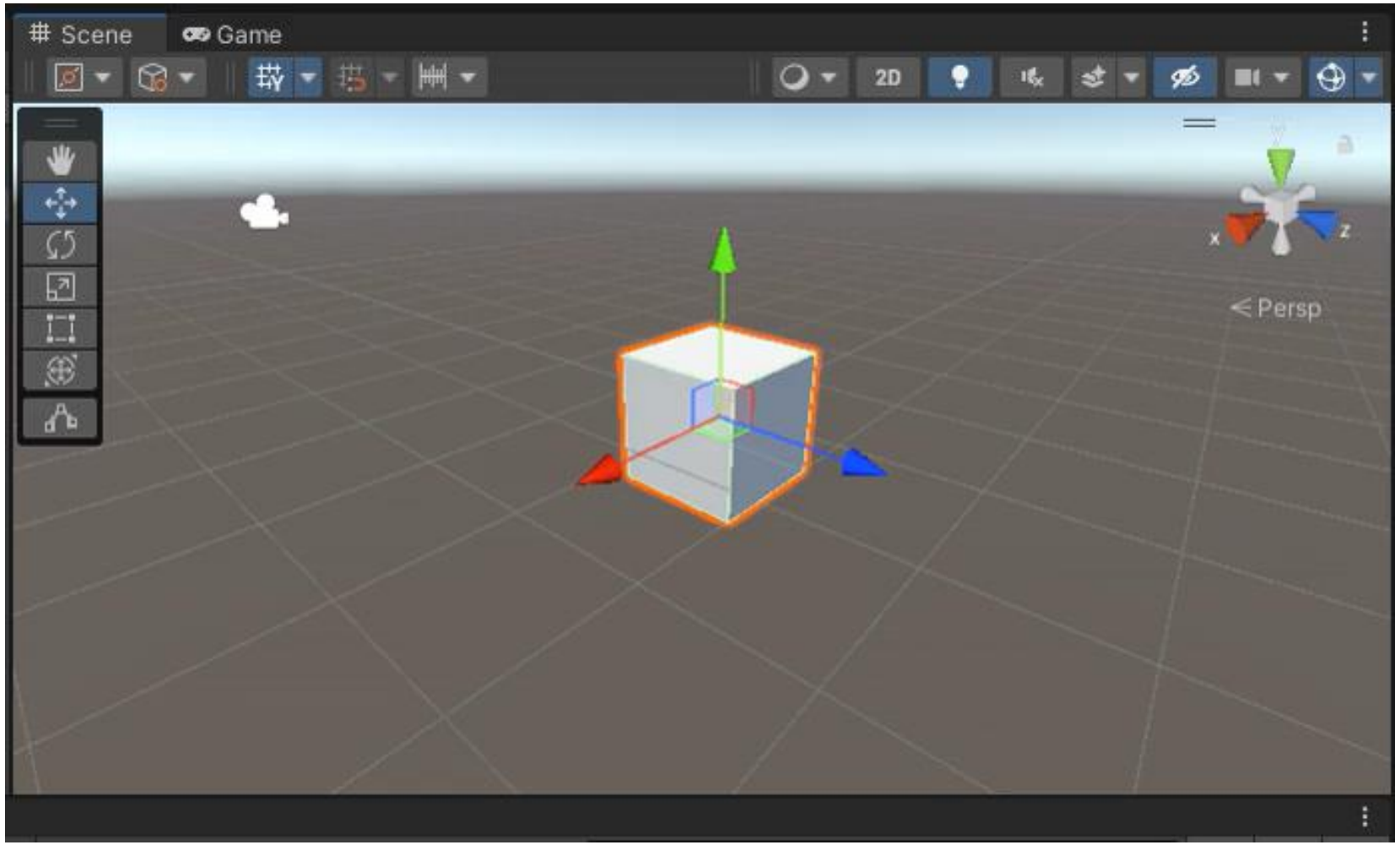




ゲーム・オブジェクト回転

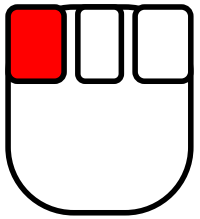
左クリックしながら回転

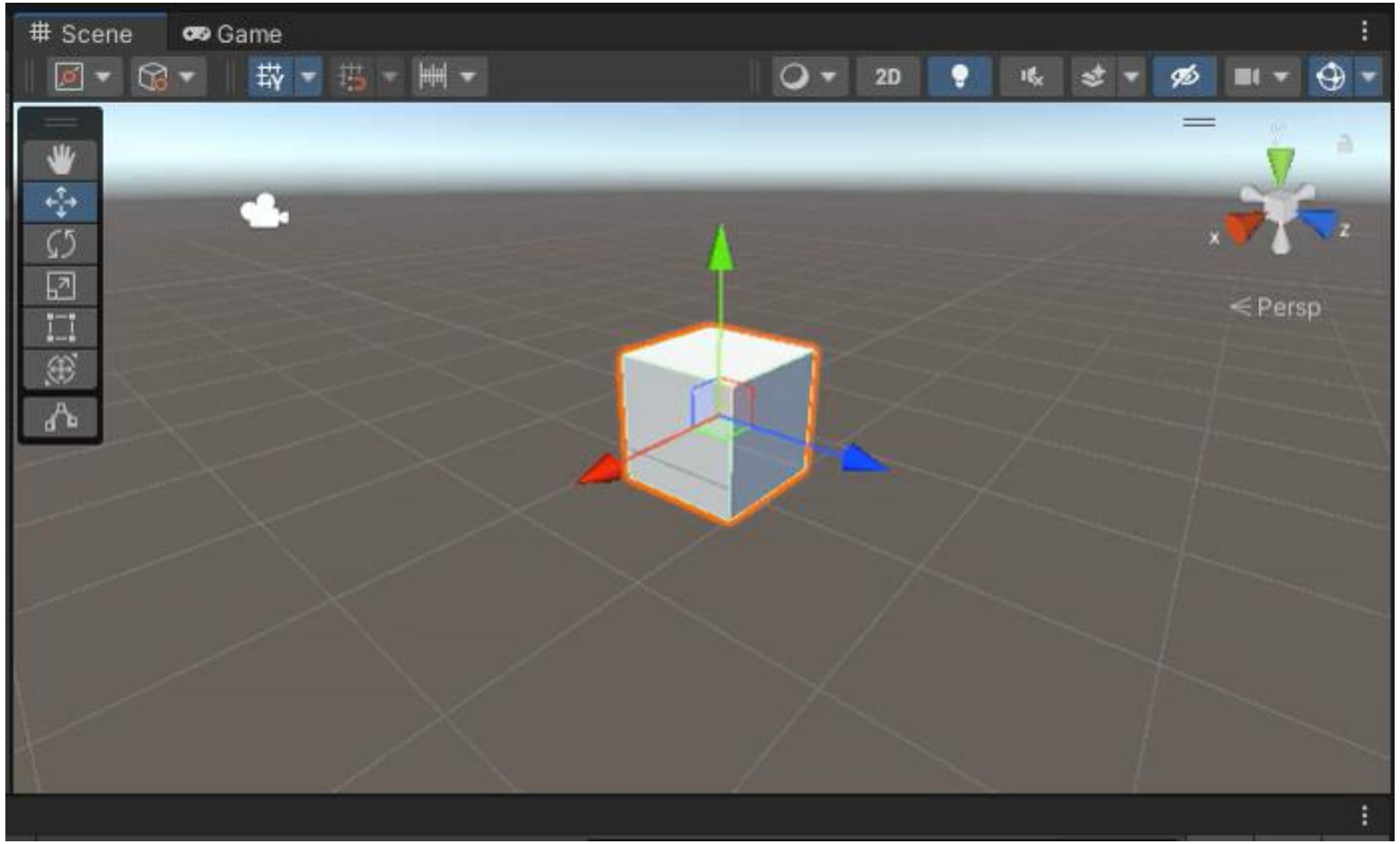




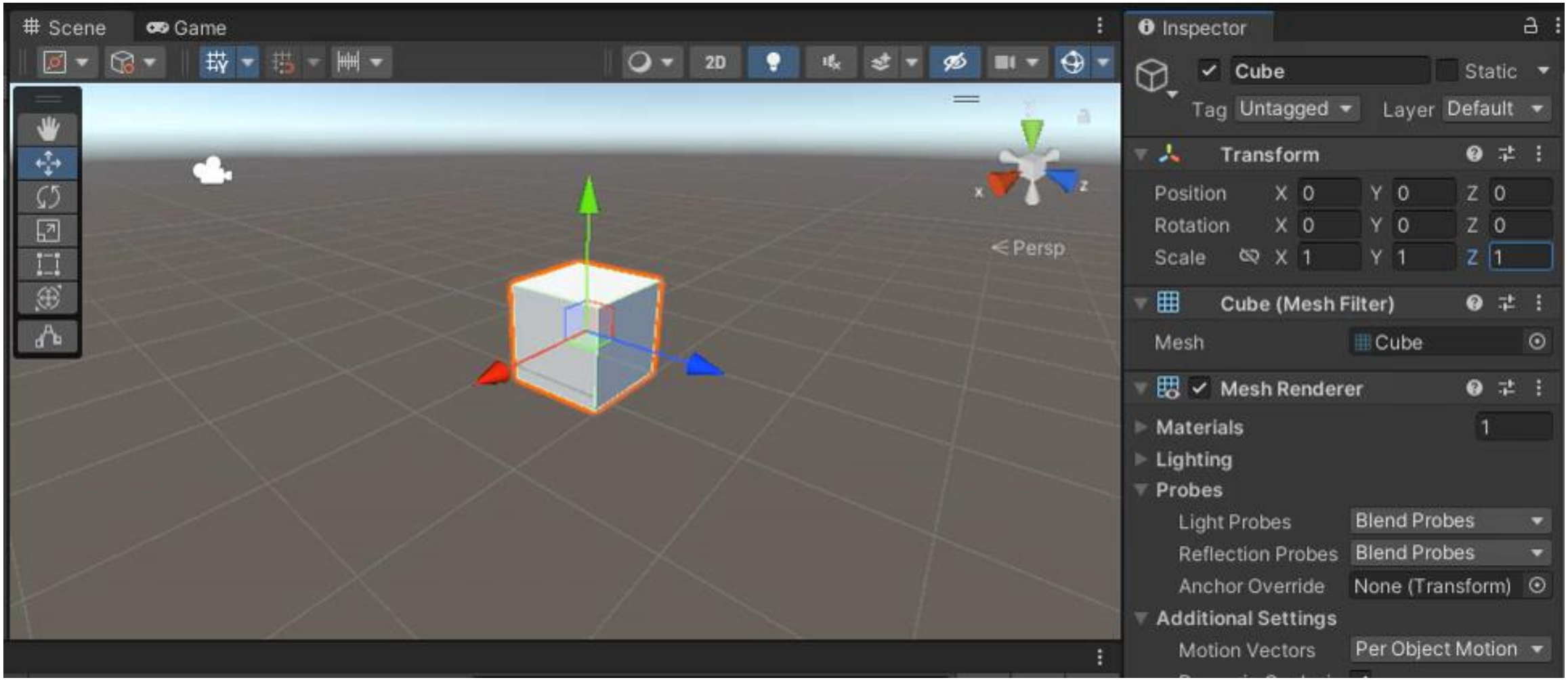
ゲーム・オブジェクト拡大・縮小

左クリックしながら拡大・縮小





Inspectorビューでもできます

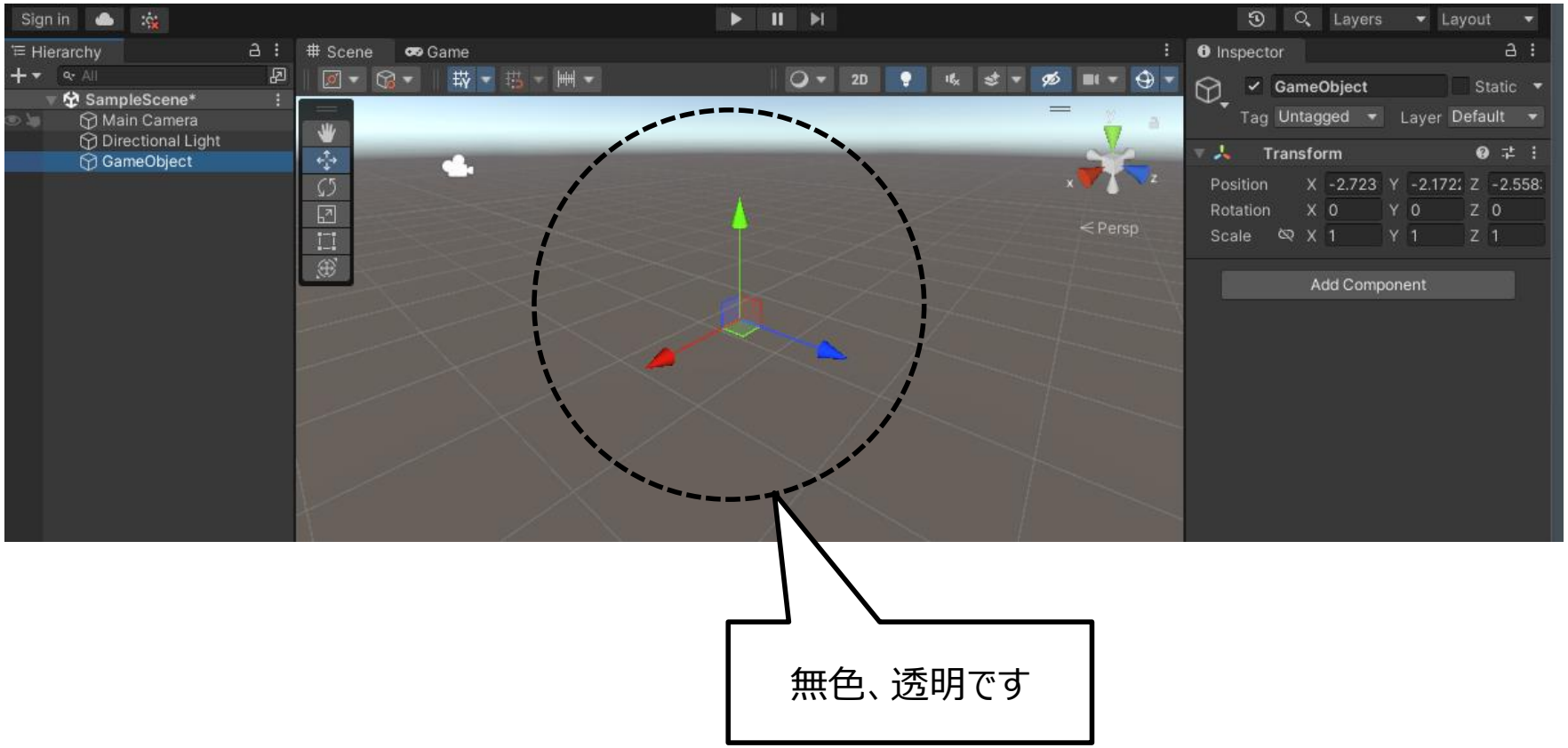


Unityで物理シミュレーションするための基礎知識

- 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

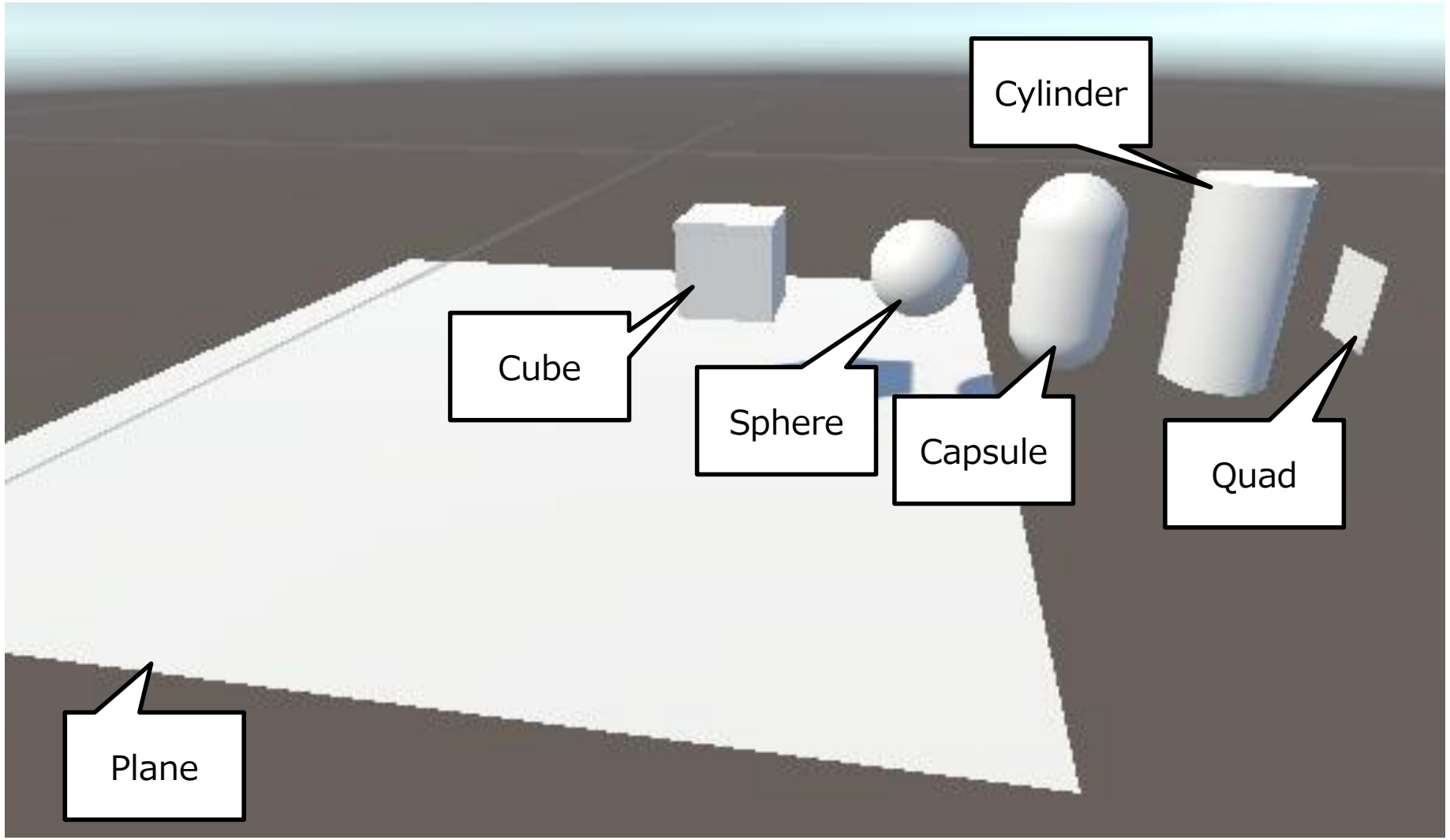
ゲームオブジェクト

- 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)



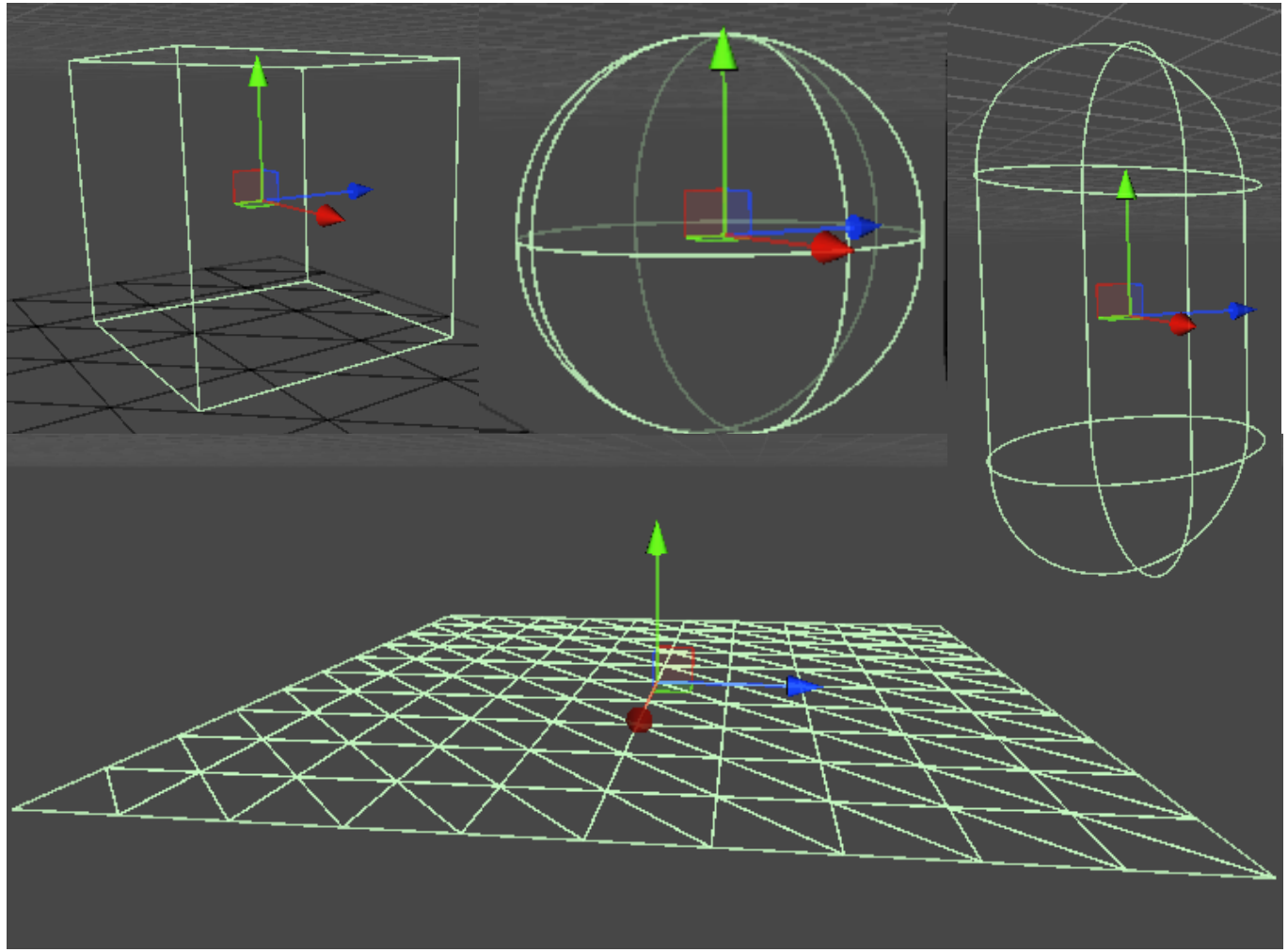
プリミティブな素材

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

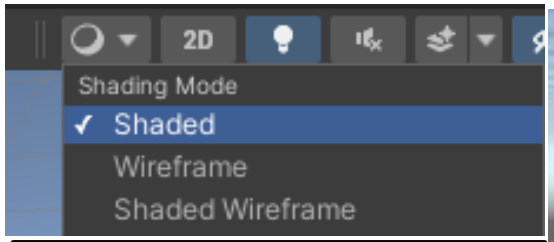


衝突判定用のオブジェクト

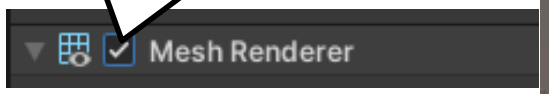
- コライダの種類
  - Boxコライダ
  - Sphereコライダ
  - Capsuleコライダ
  - Meshコライダ

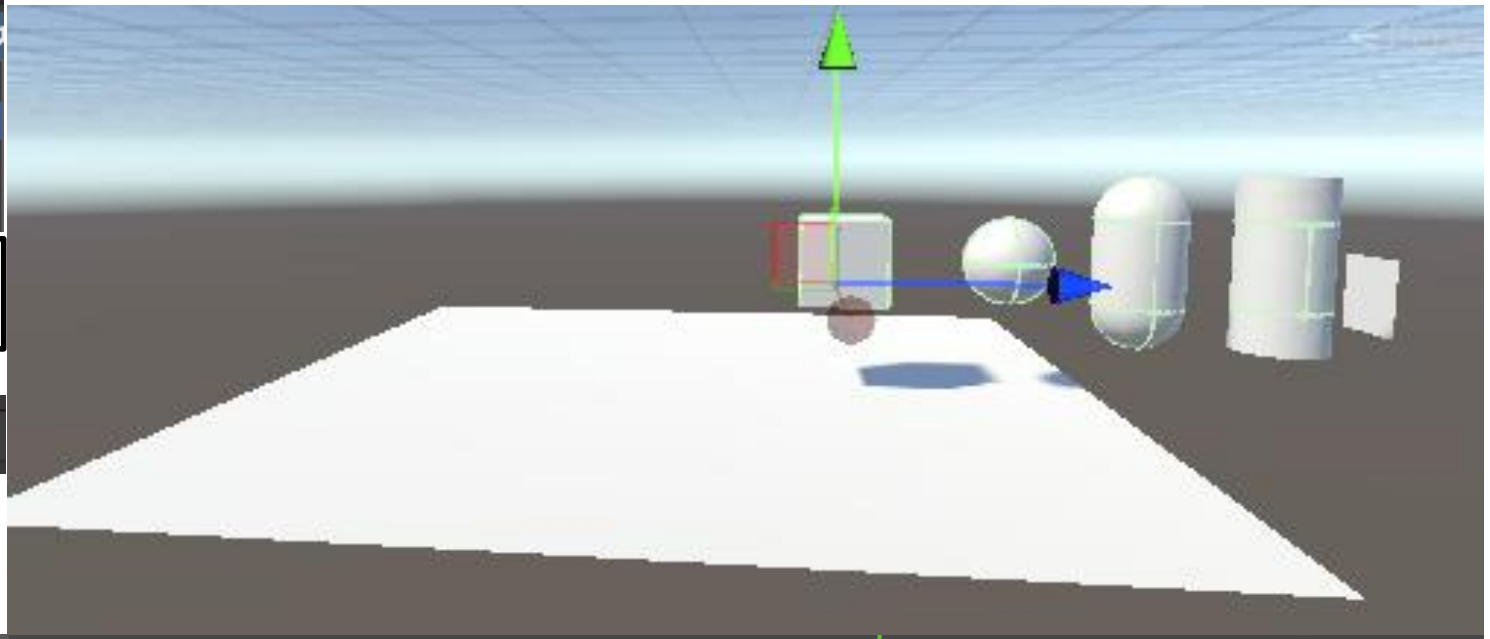


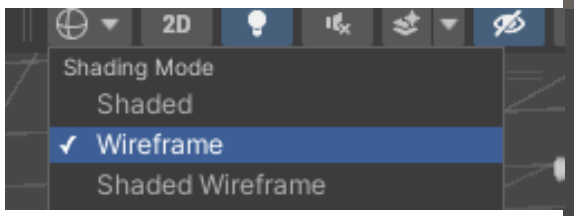
プリミティブな素材のコライダを見てみる



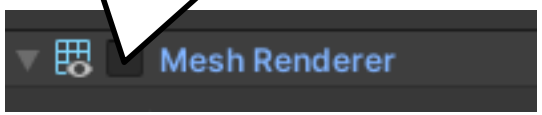
全オブジェクトの
Mesh Rendererをオン

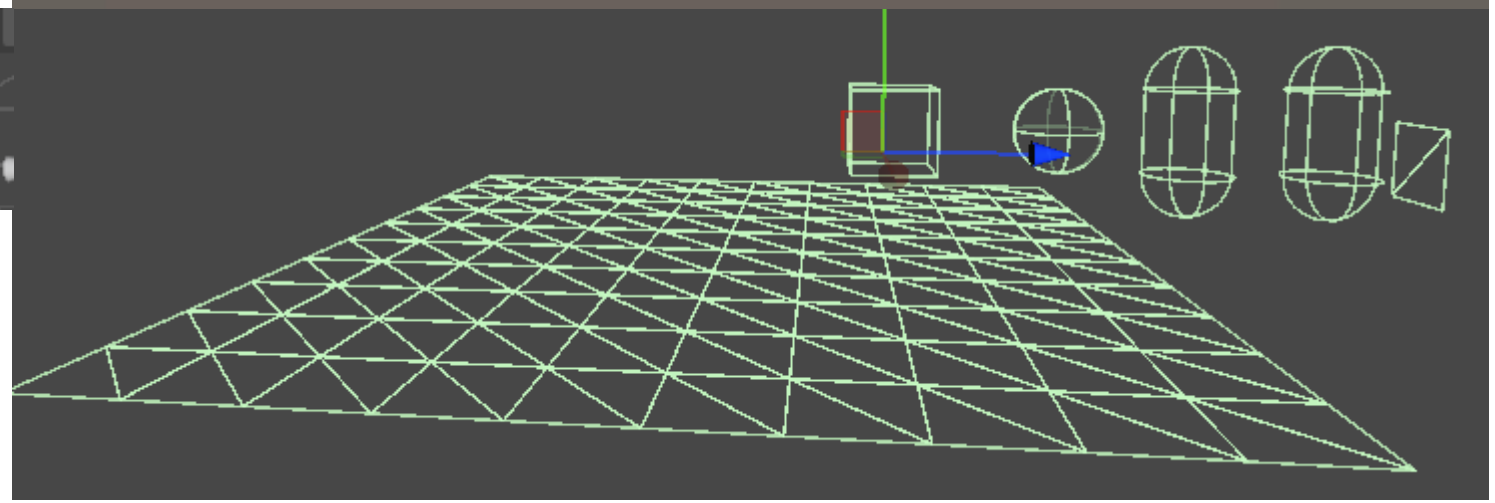






全オブジェクトの
Mesh Rendererをオフ

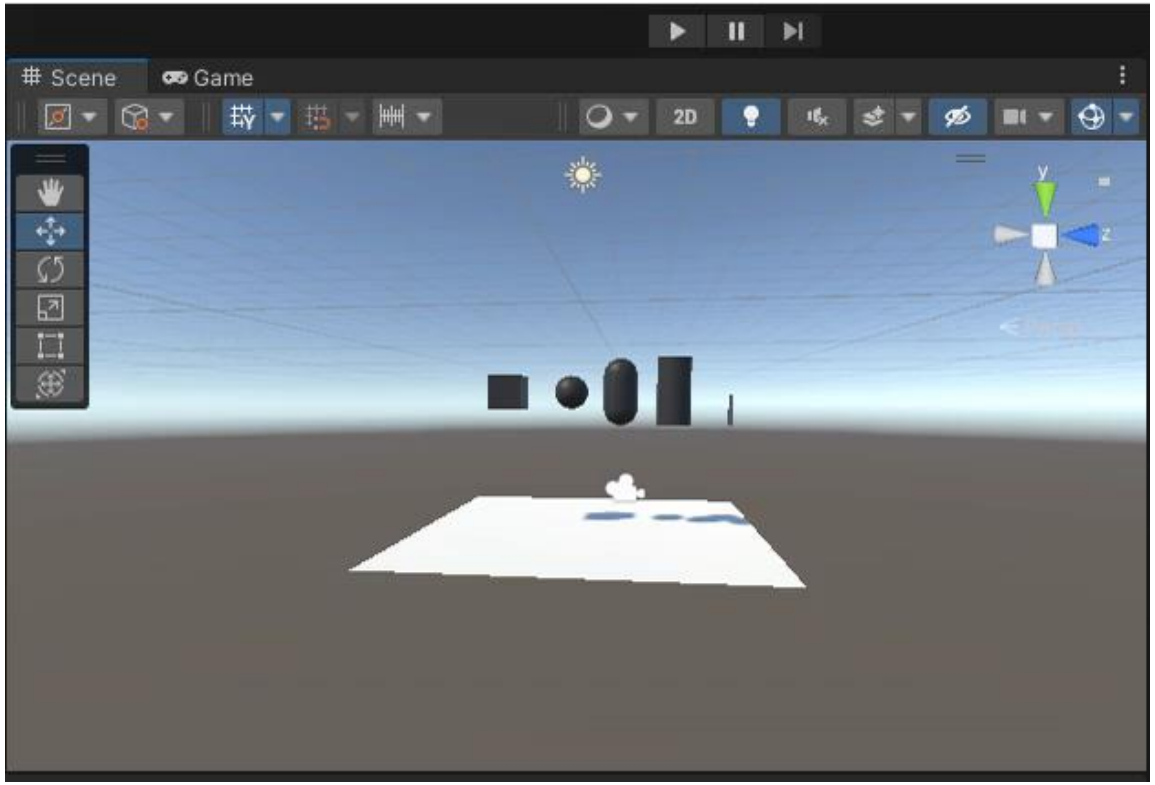




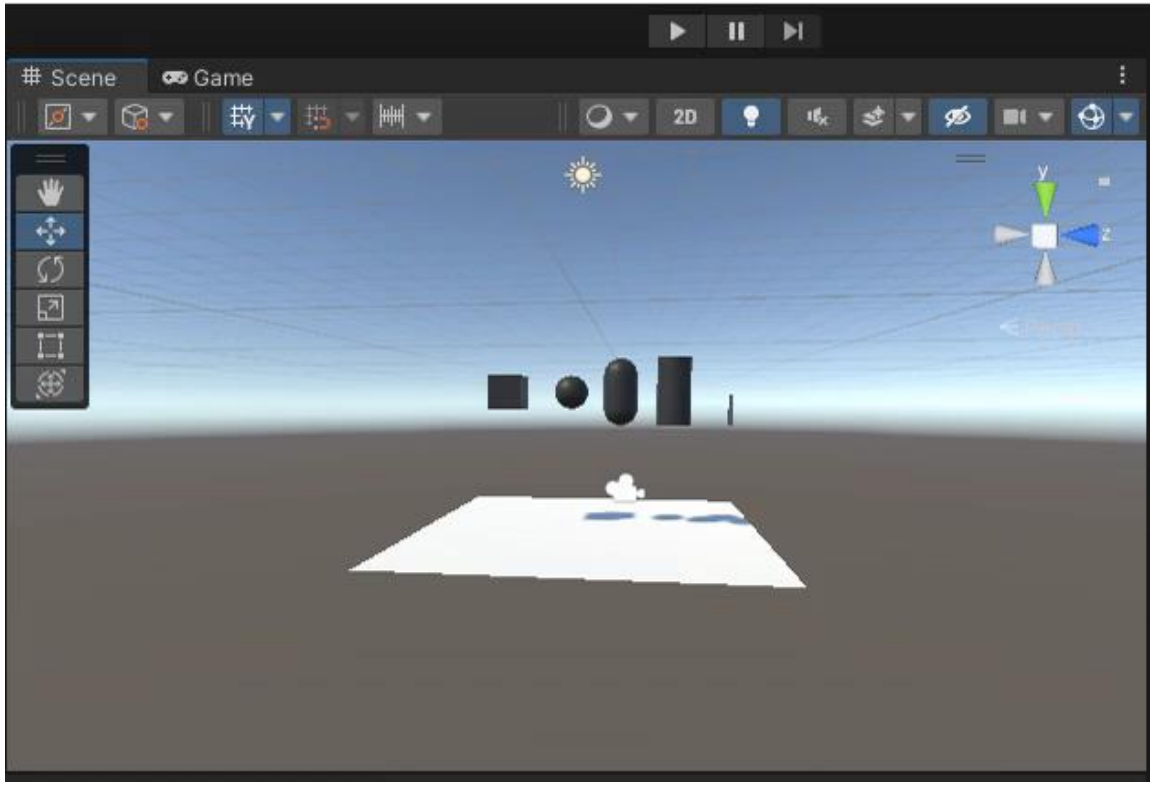
TOPPERS
Toyohashi OPen Platform
for Embedded Real-time Systems

コライダによる衝突の様子

コライダなしの場合：

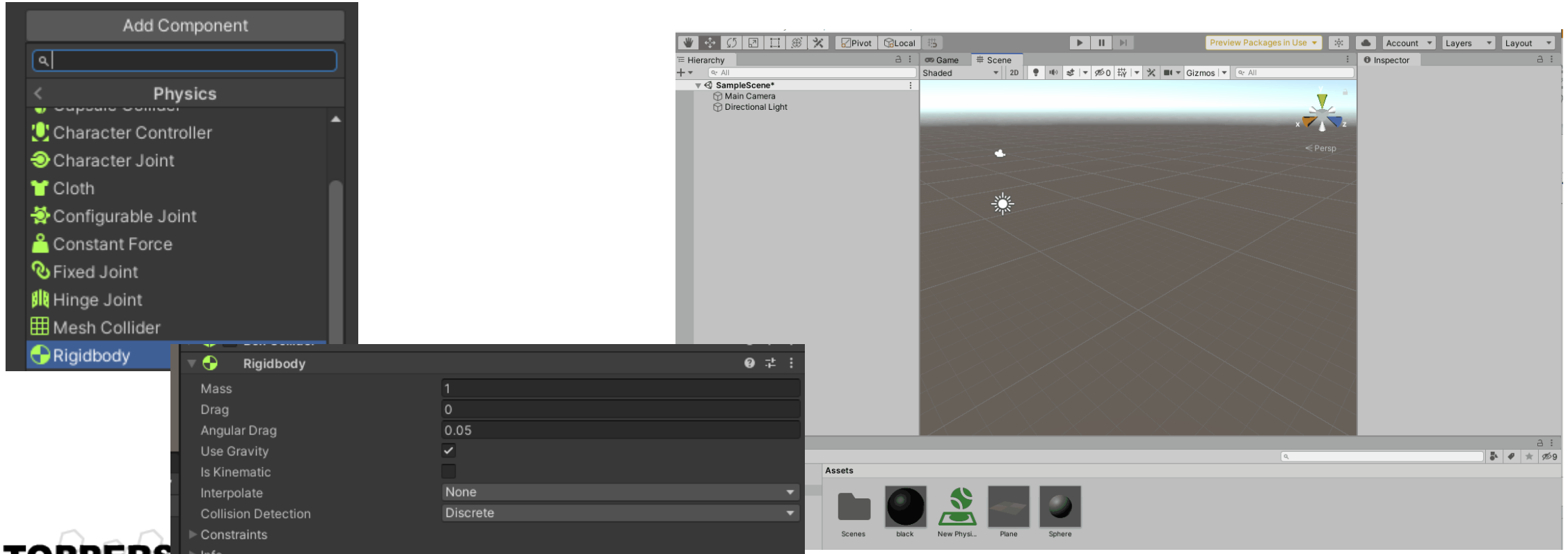


コライダありの場合：



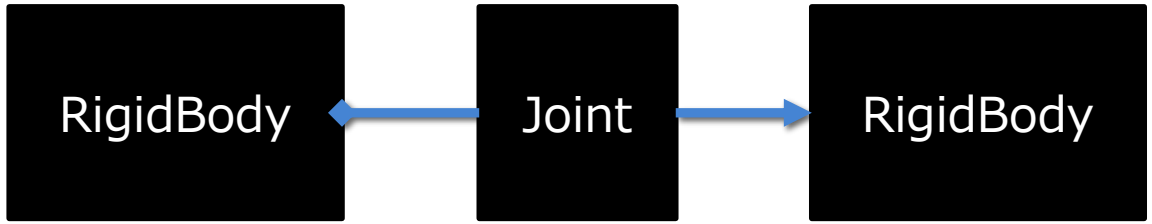
物理挙動

- 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るようになります。さらに、1つ以上の Collider (コライダー) コンポーネントを加わえれば、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。

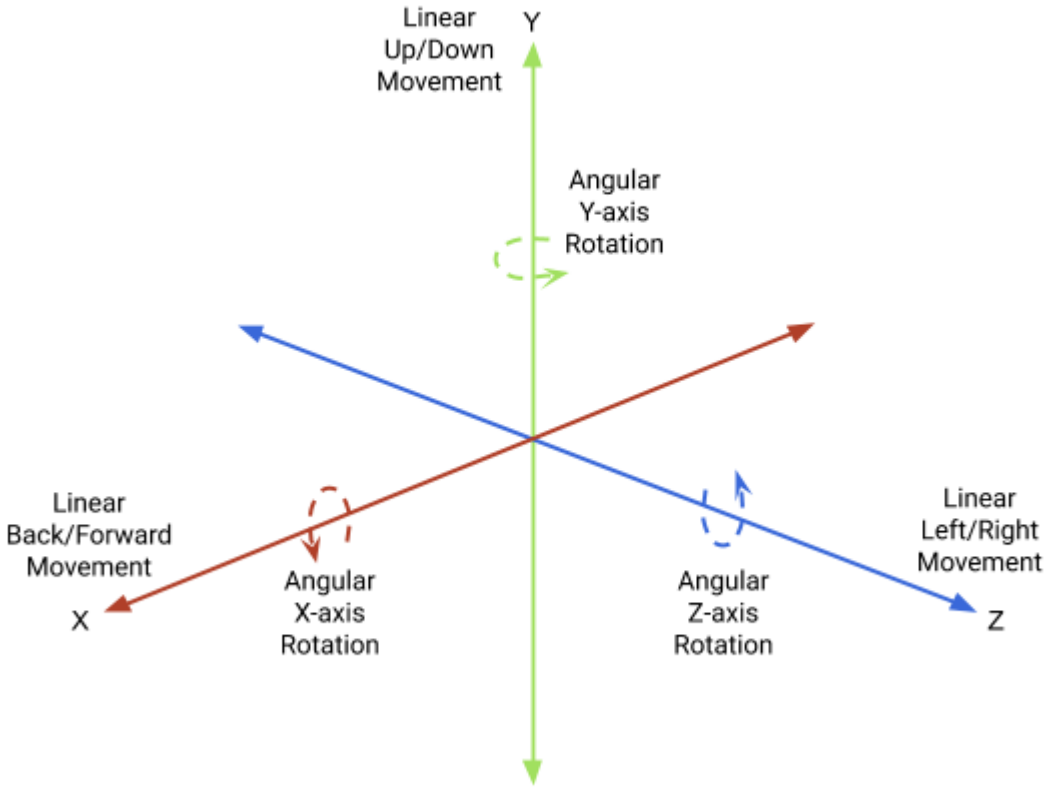


Jointコンポーネント

- 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。

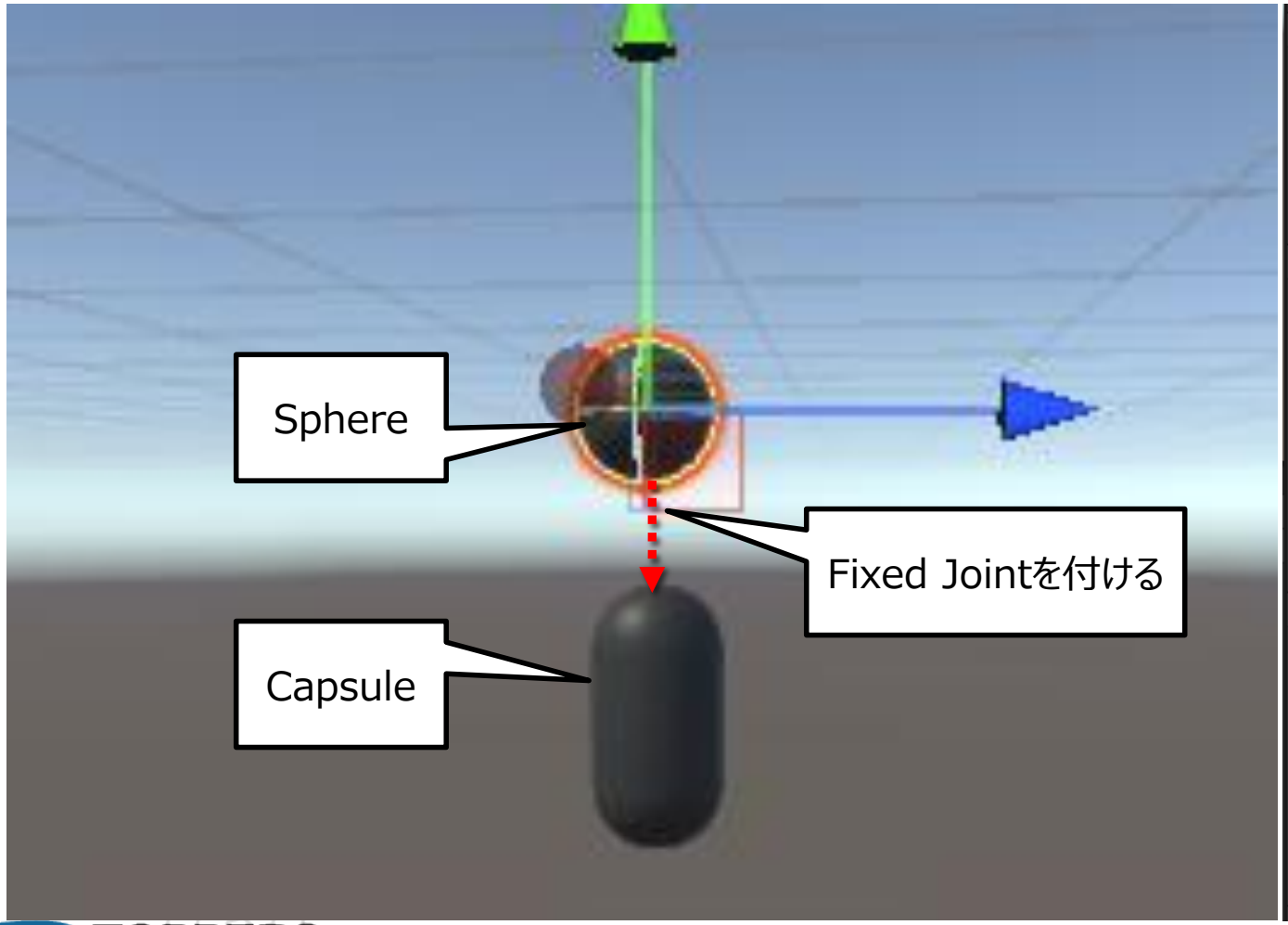


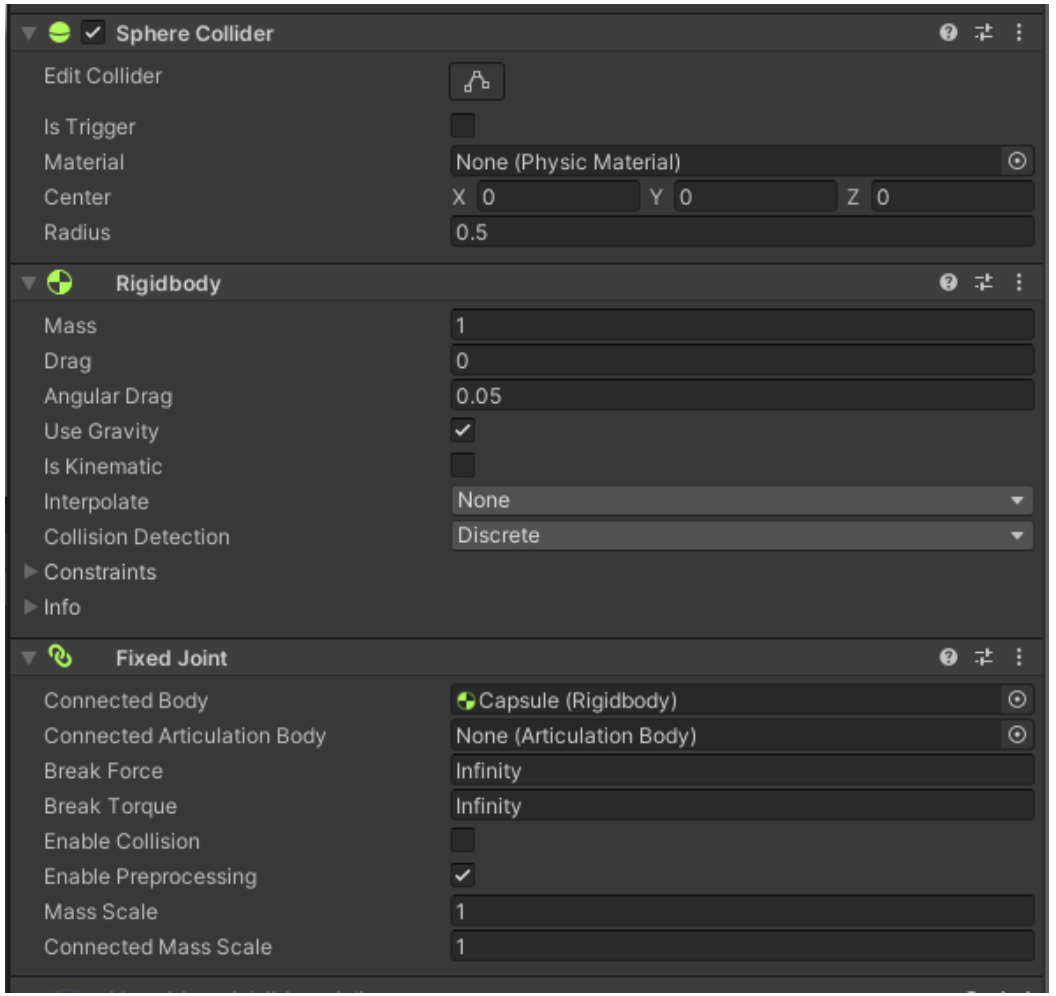
Jointコンポーネントの種類	箱庭での利用有無
Character Joint	×
Configurable Joint	○
Fixed Joint	○
Hinge Joint	○
Spring Joint	×



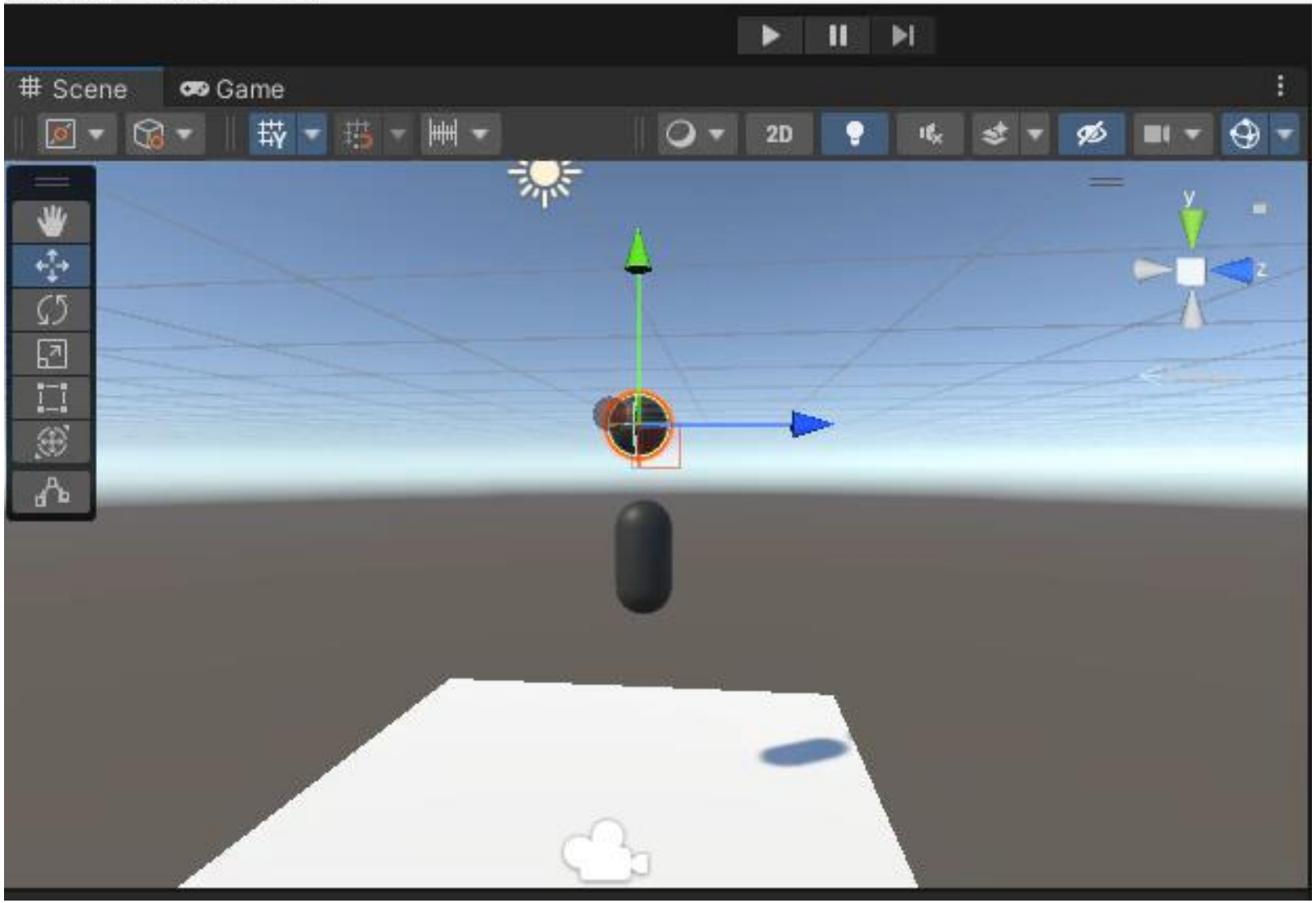
Fixed Joint

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



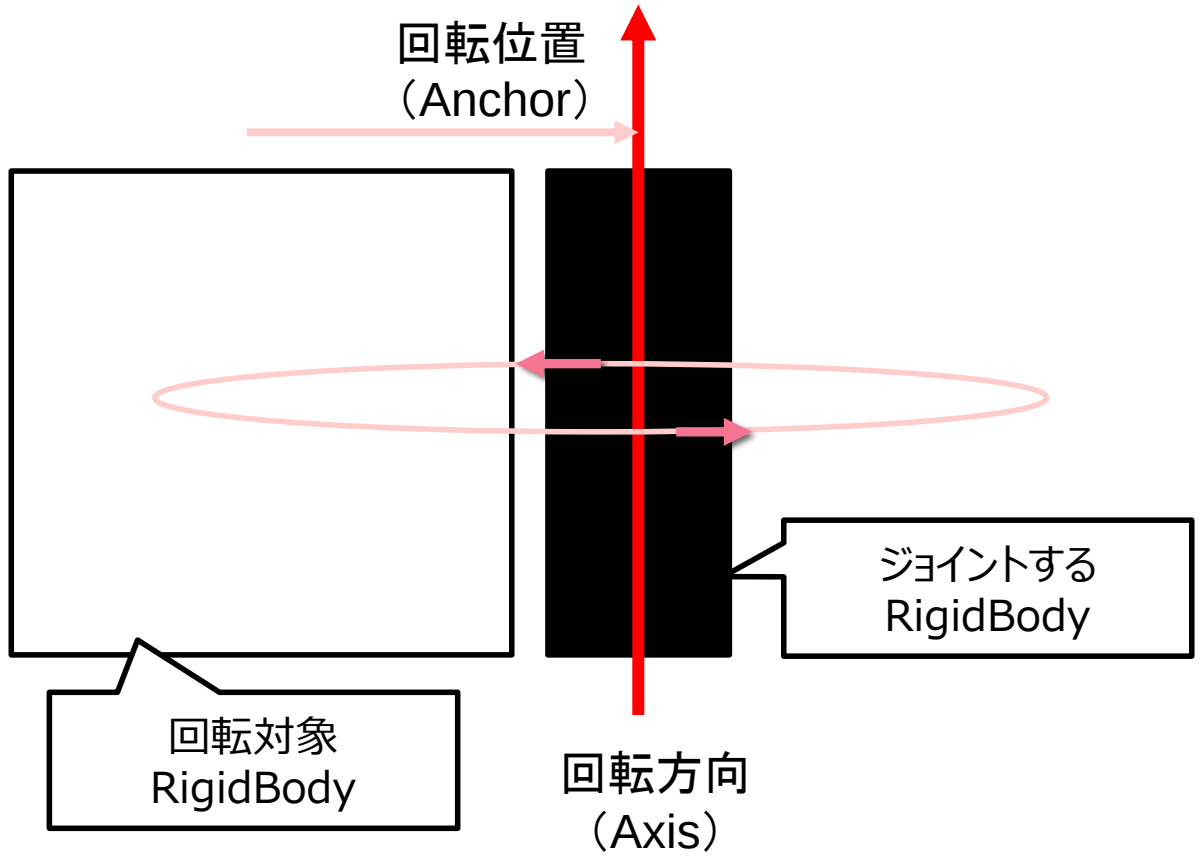


Fixed Joint

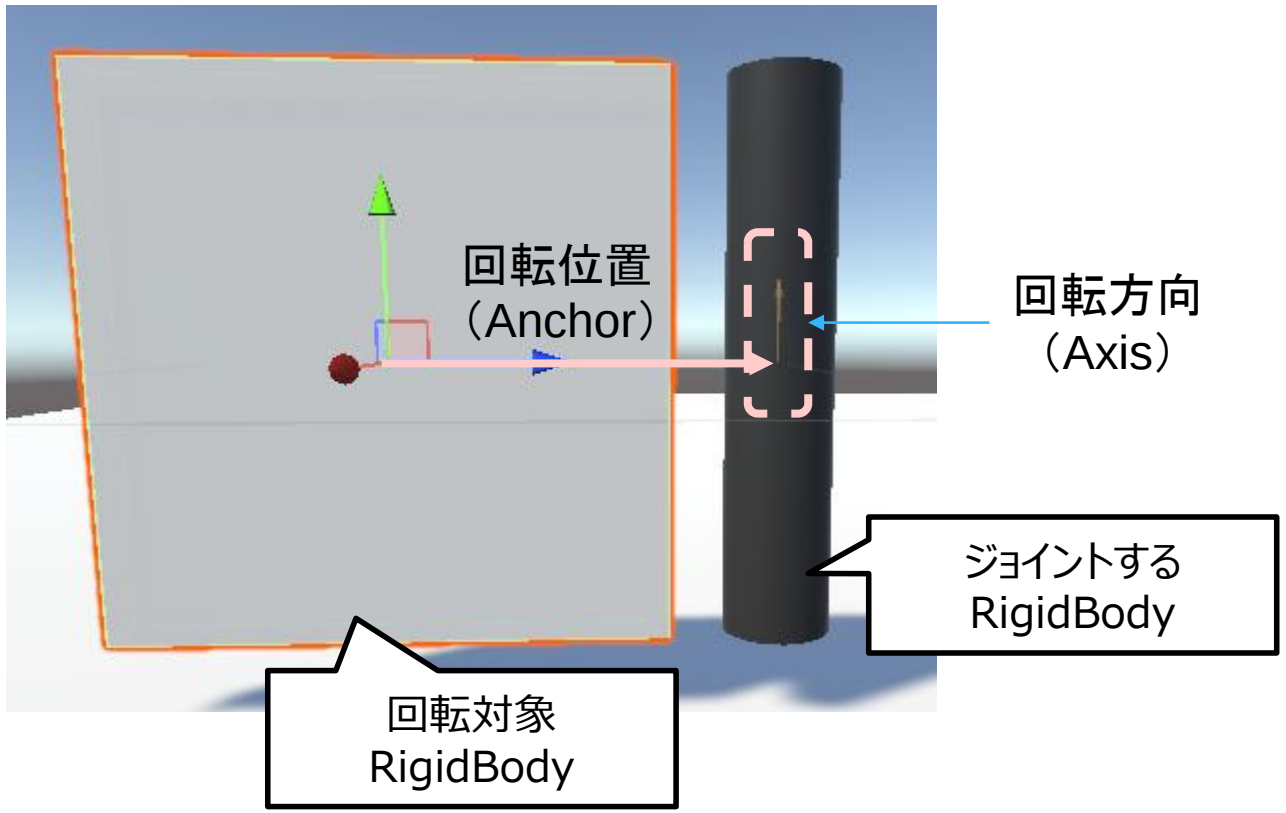


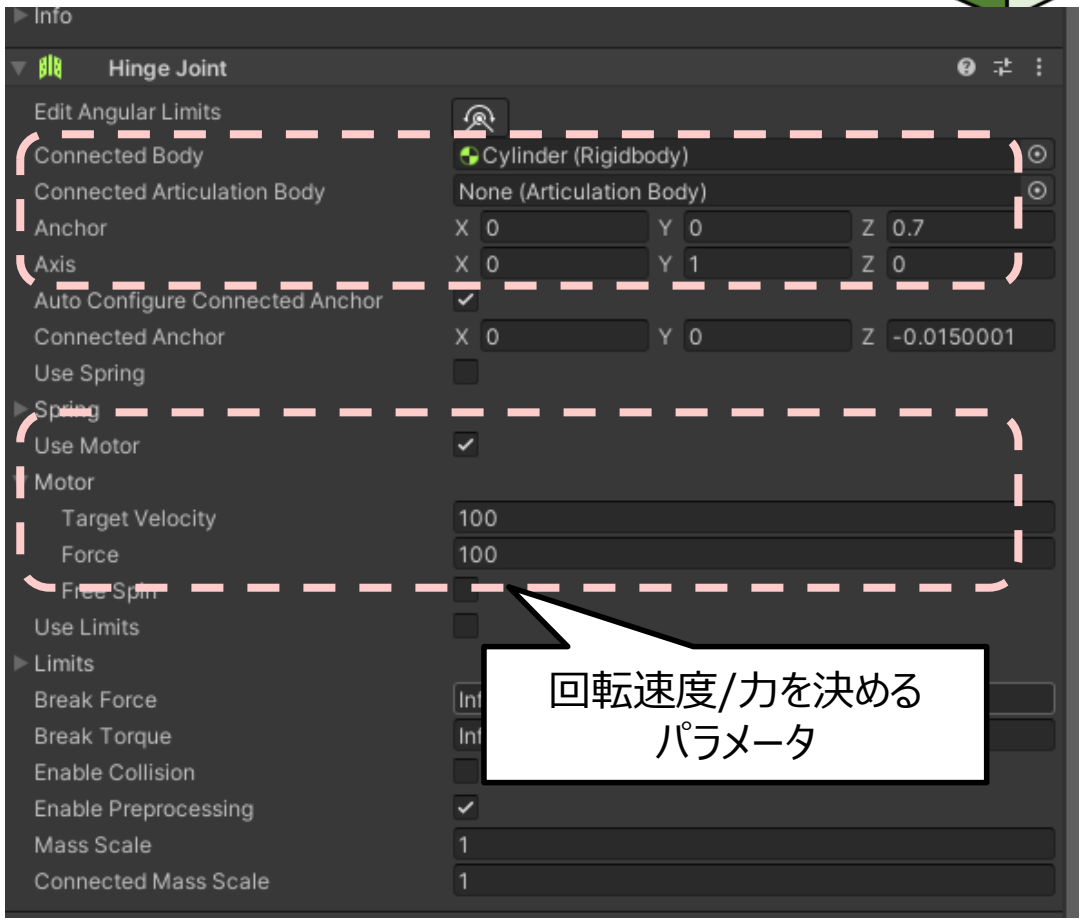
Hinge Joint

- 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。

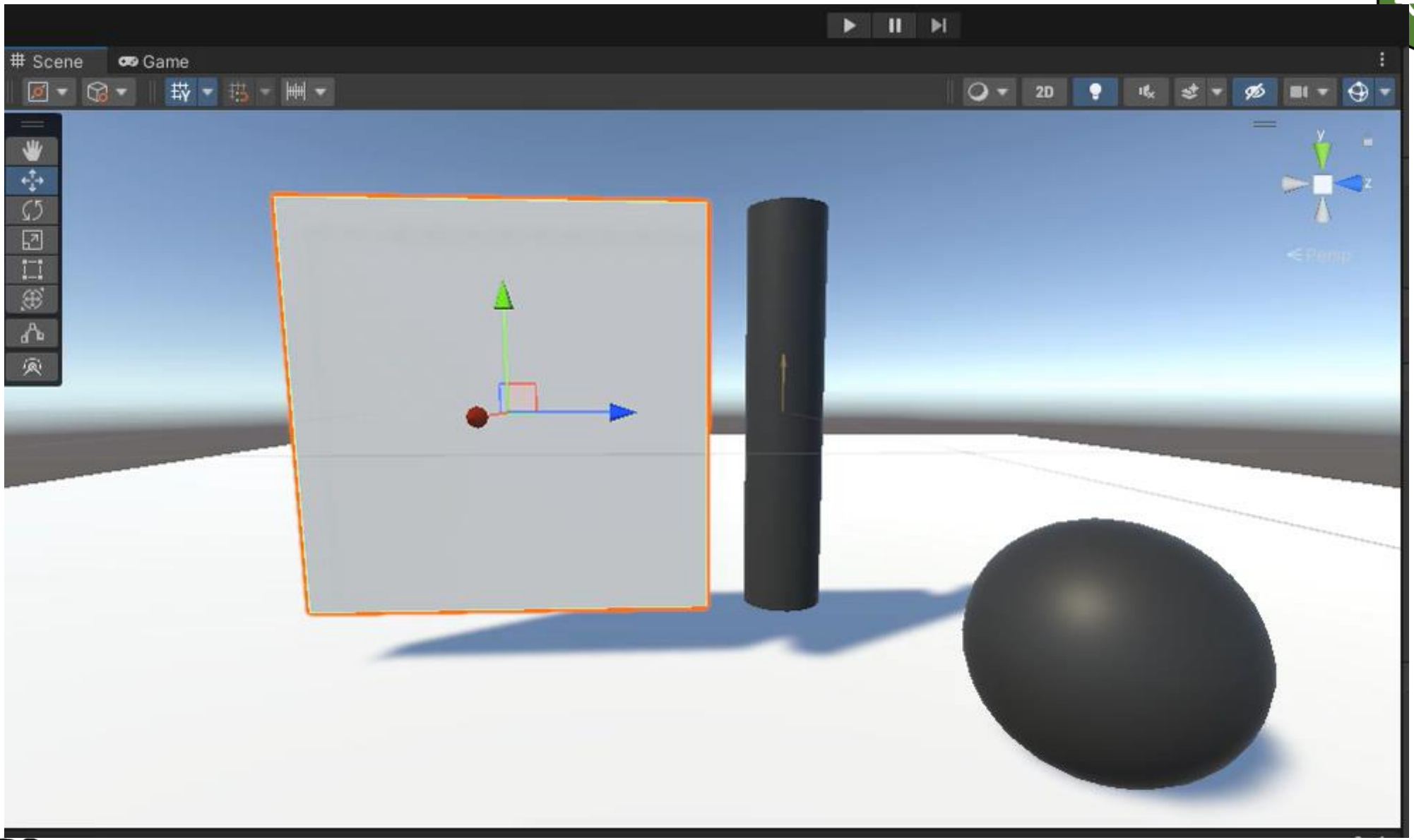


Hinge Jointの例





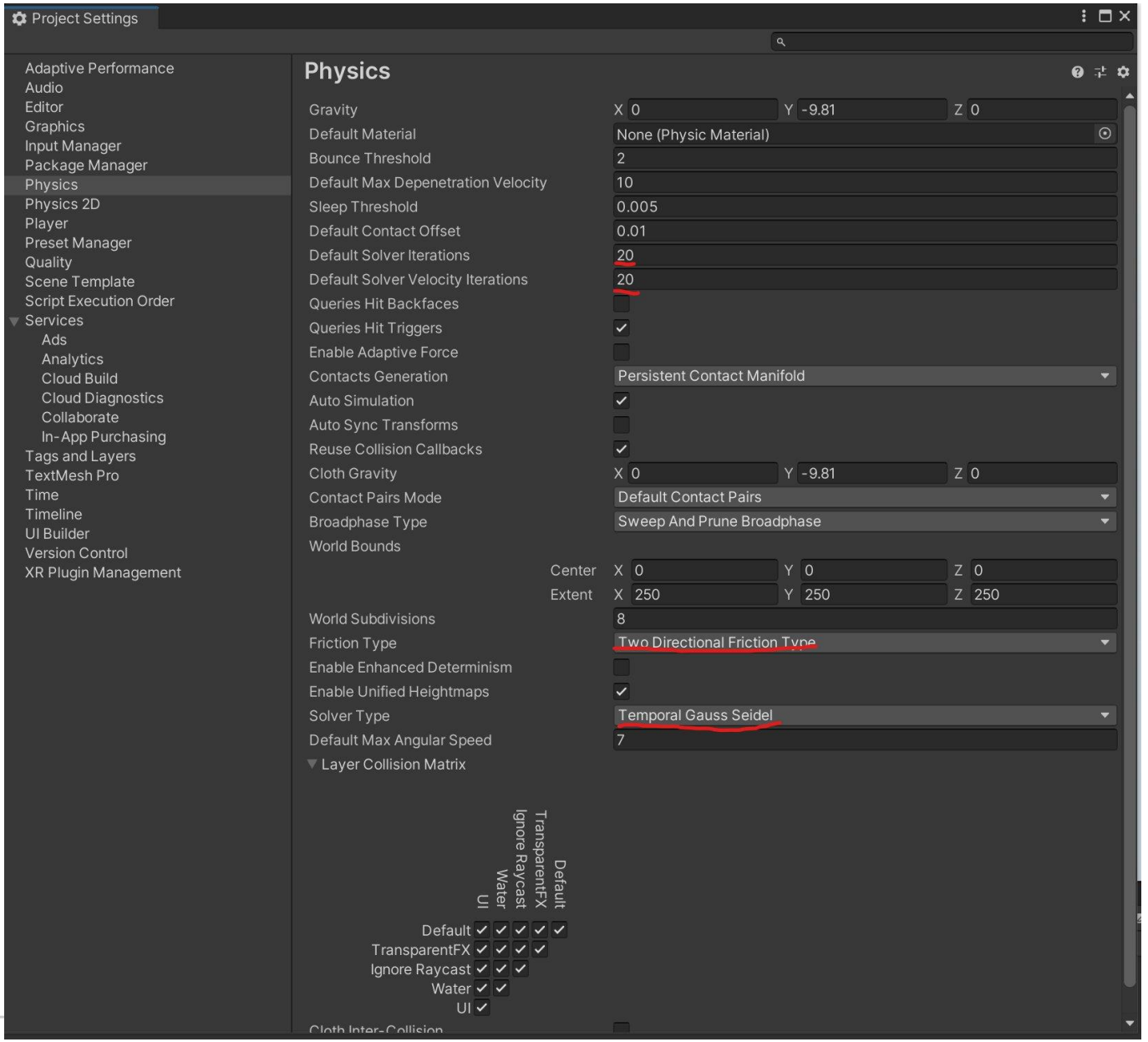
Hinge Joint



物理挙動

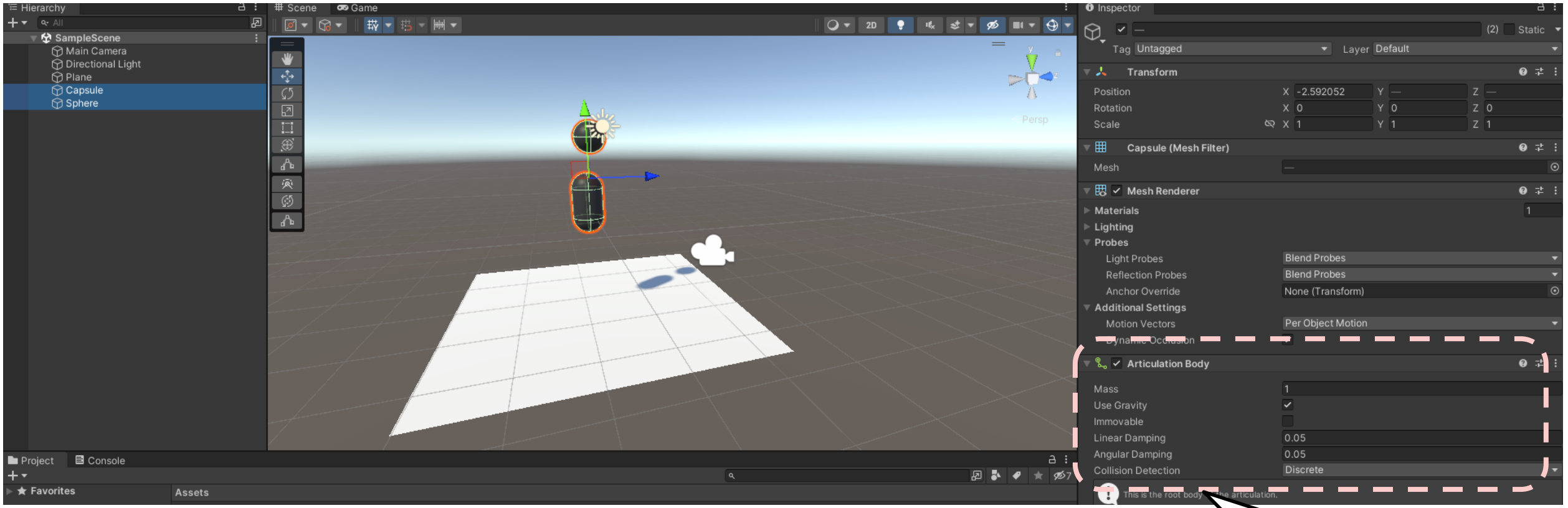
- ArticulationBody
  - Articulation Body (連結ボディ) を使うことによって、ゲームオブジェクトを使ったロボットアームやキネマティックチェーンのような 物理的連結 を構築することができます。これらは、産業用アプリケーションのシミュレーションのコンテキストでリアルな物理動作を得るのに役立ちます。
  - 連結ボディは、RigidBody や 通常のジョイント といった従来の構成で定義されるプロパティを、1 つのコンポーネントで定義することができます。ただし、これらのプロパティは、階層内のゲームオブジェクトの位置に依存します。

プロジェクト設定を変更する必要あります



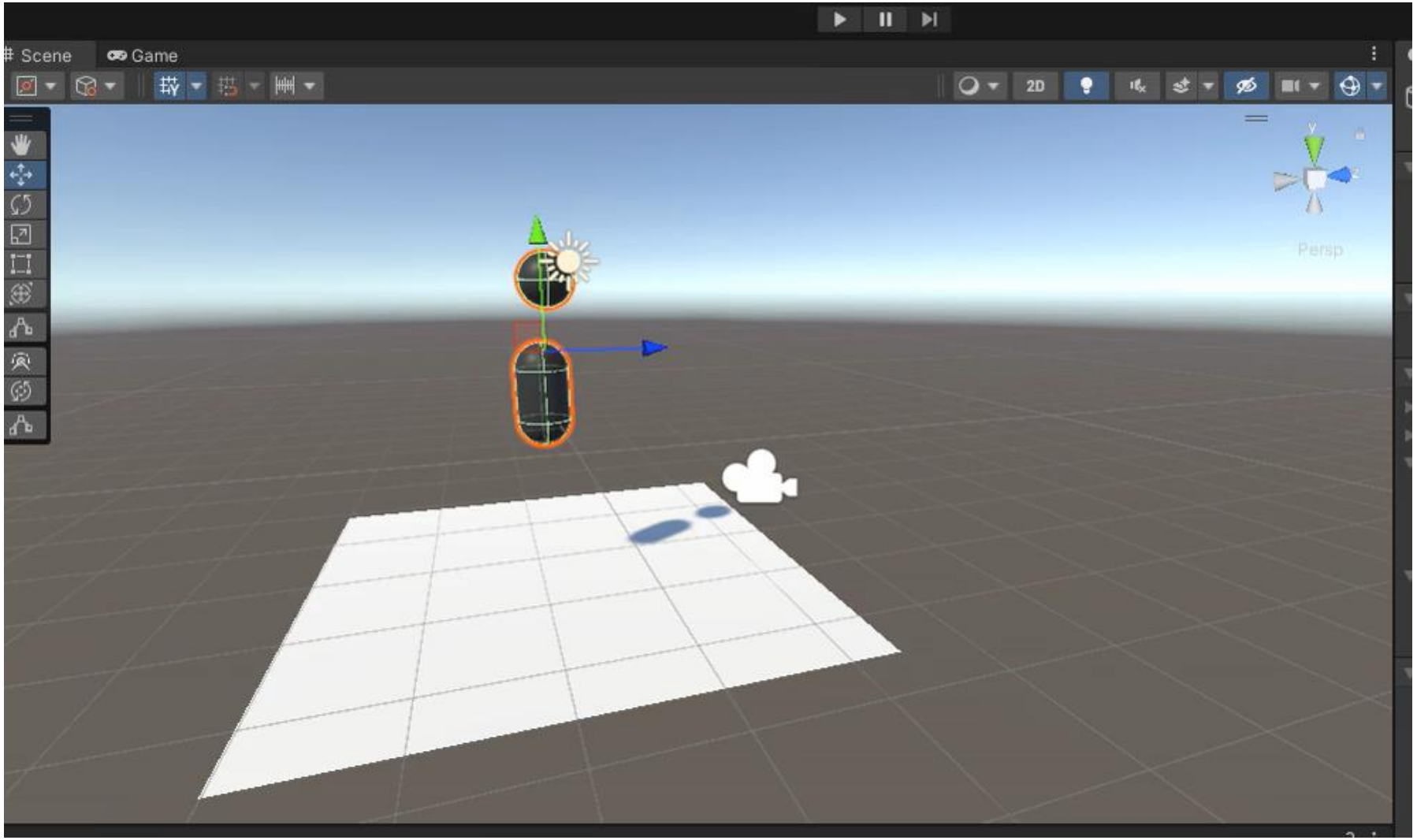
論よりRUN!

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

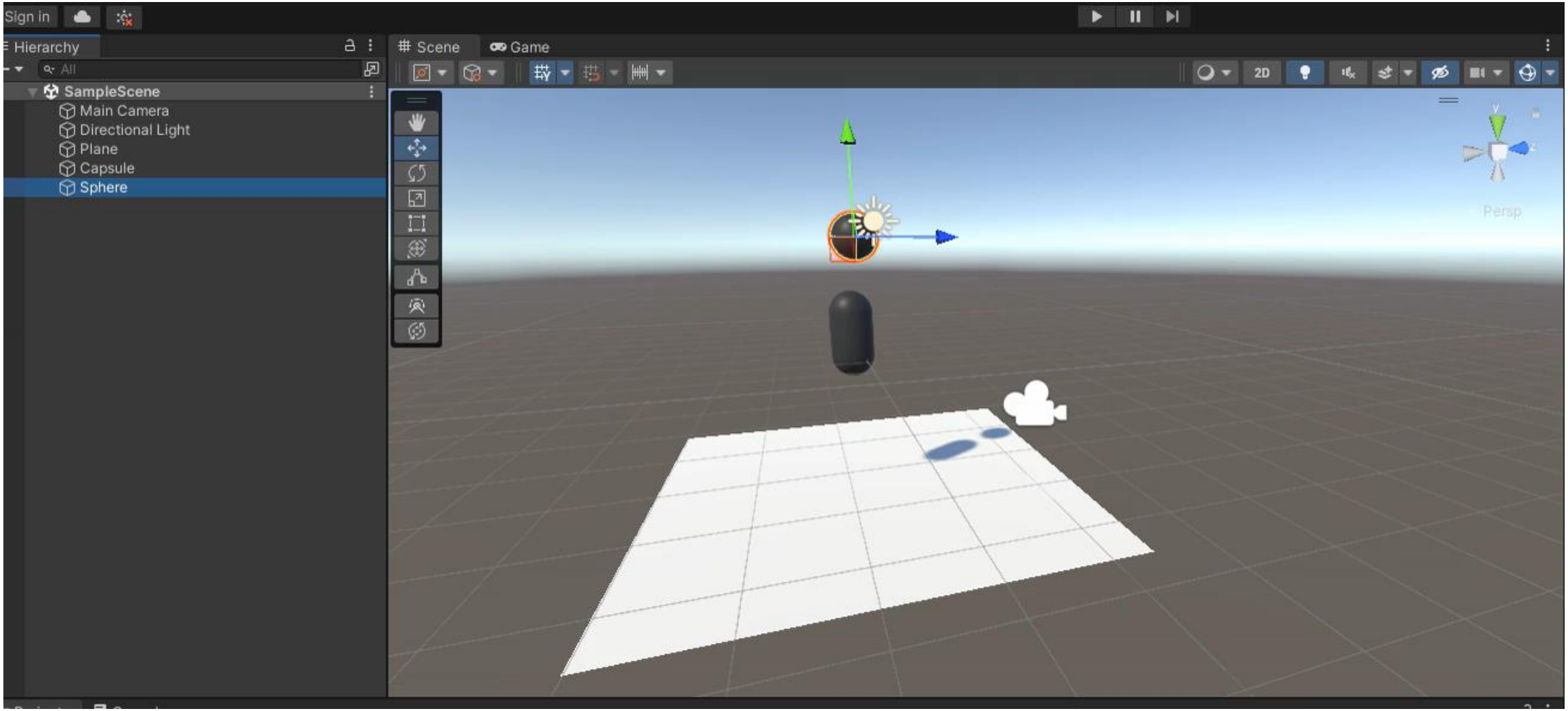


Articulation Bodyをアタッチするだけ！！

ArticulationBody



Fixed

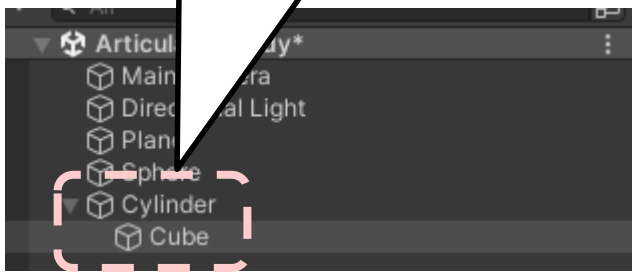

Sign in
Hierarchy
All
SampleScene
Main Camera
Directional Light
Plane
Capsule
Sphere
Scene
Game
2D
Persp

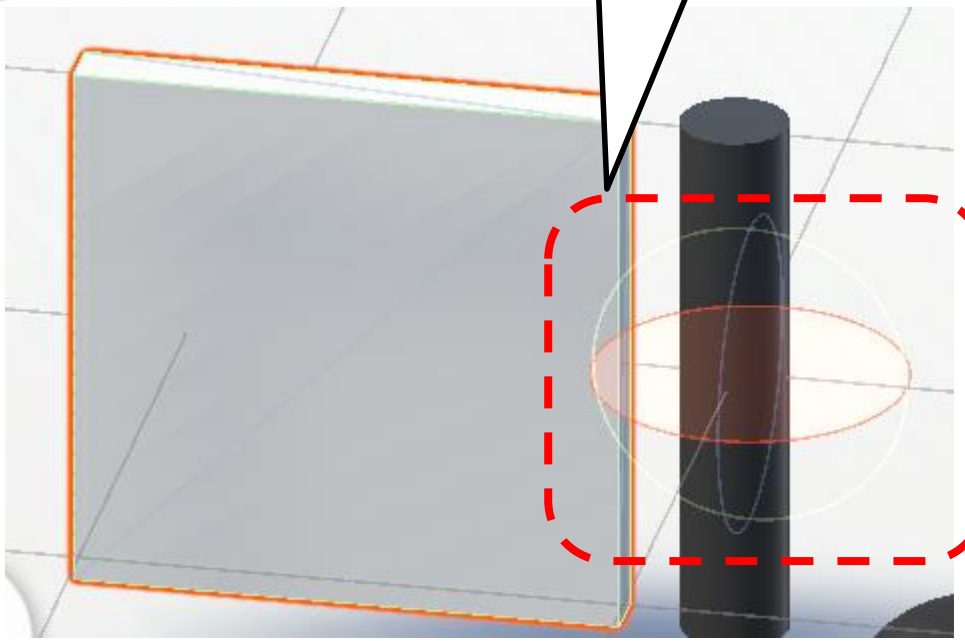
Hinge Jointと同じ設定したい場合

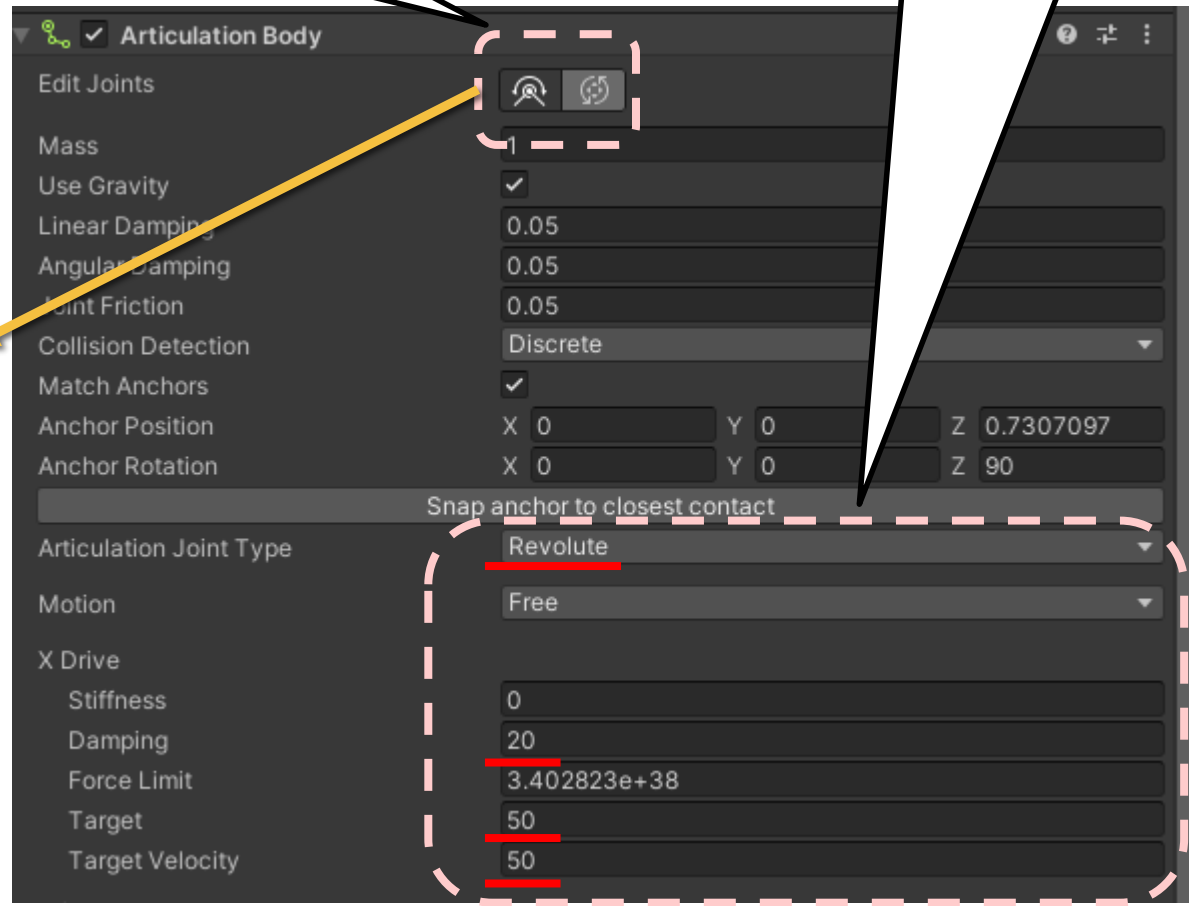
CubeをCylinder配下に移動

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

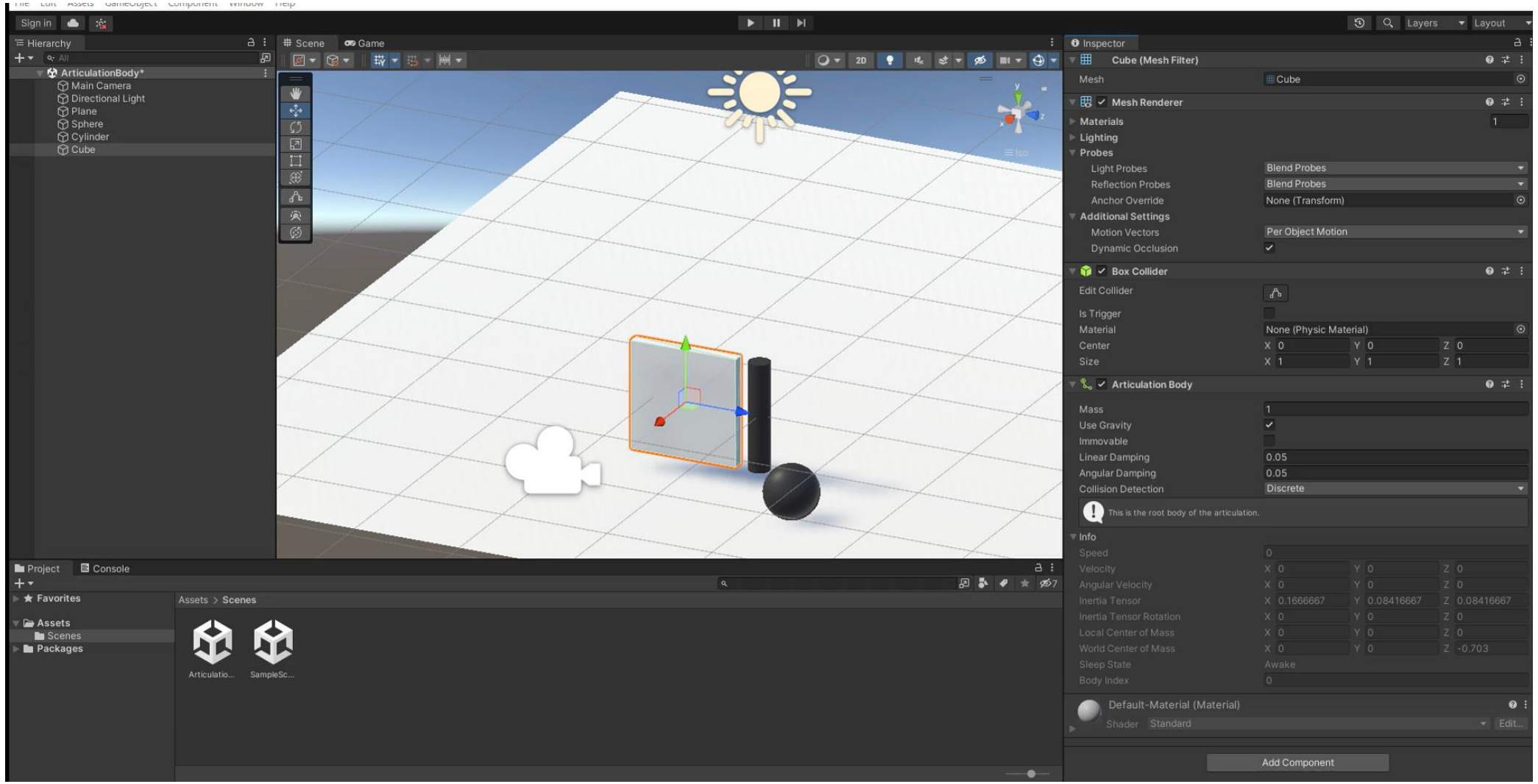






TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

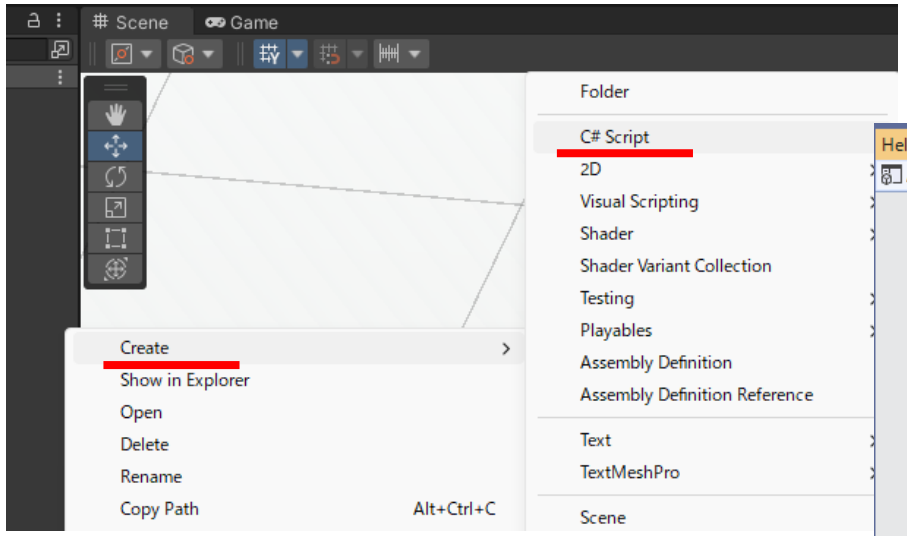
Revolute



TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

独自処理(Unityスクリプト)の追加方法（１／２）

- 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

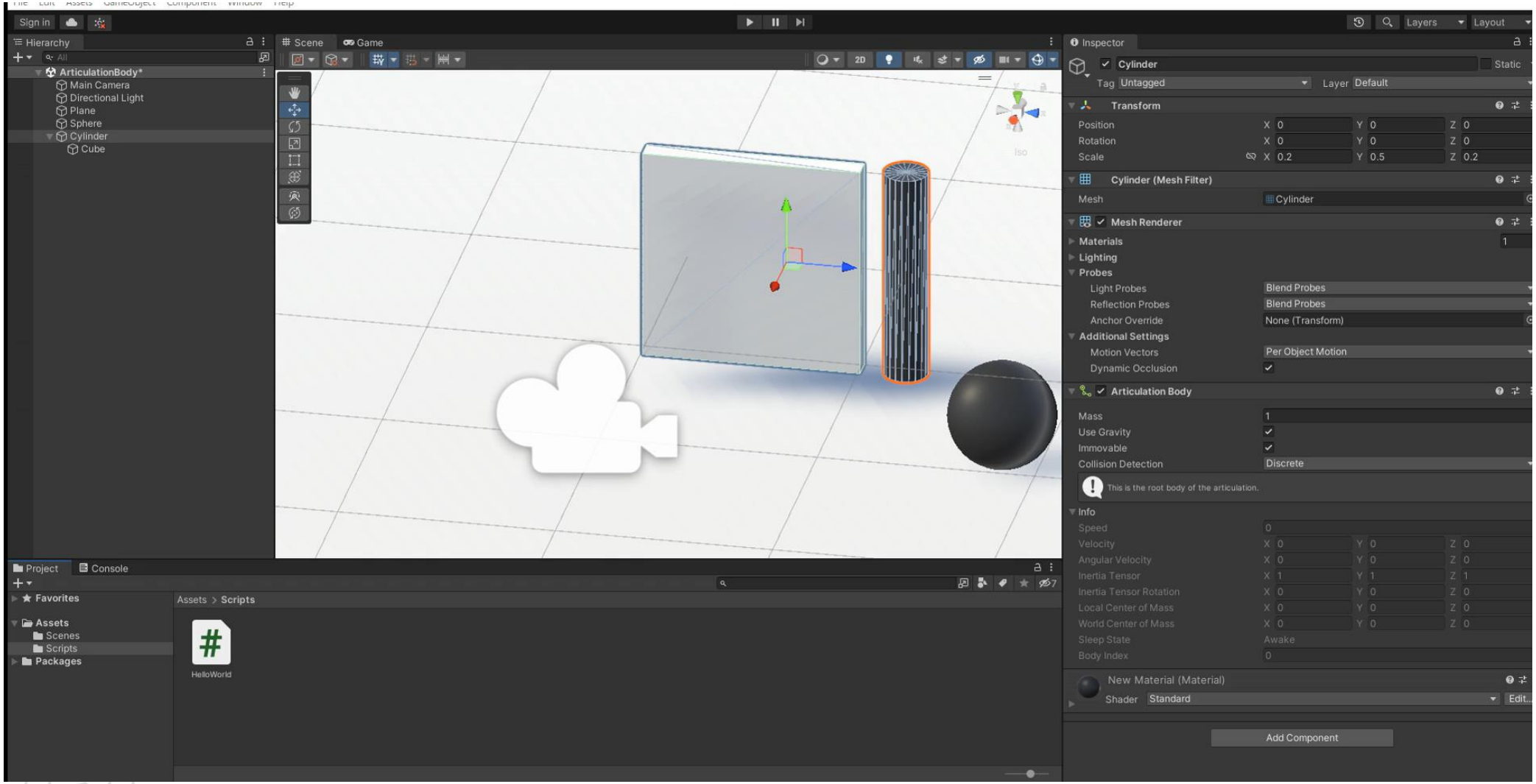



```
HelloWorld.cs
Assembly-CSharp   HelloWorld   Update()
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    Unity スクリプト | 0 個の参照
5   public class HelloWorld : MonoBehaviour
6   {
7       // Start is called before the first frame update
        Unity メッセージ | 0 個の参照
8       void Start()
9       {
10          Debug.Log("Hello World!!");
11      }
12
13      // Update is called once per frame
        Unity メッセージ | 0 個の参照
14      void Update()
15      {
16          Debug.Log("Called Update!!");
17      }
18  }
19
160 %   問題は見つかりませんでした   行: 16   文字: 33   SPC   CRLF
```

シミュレーション開始時に呼び出されます

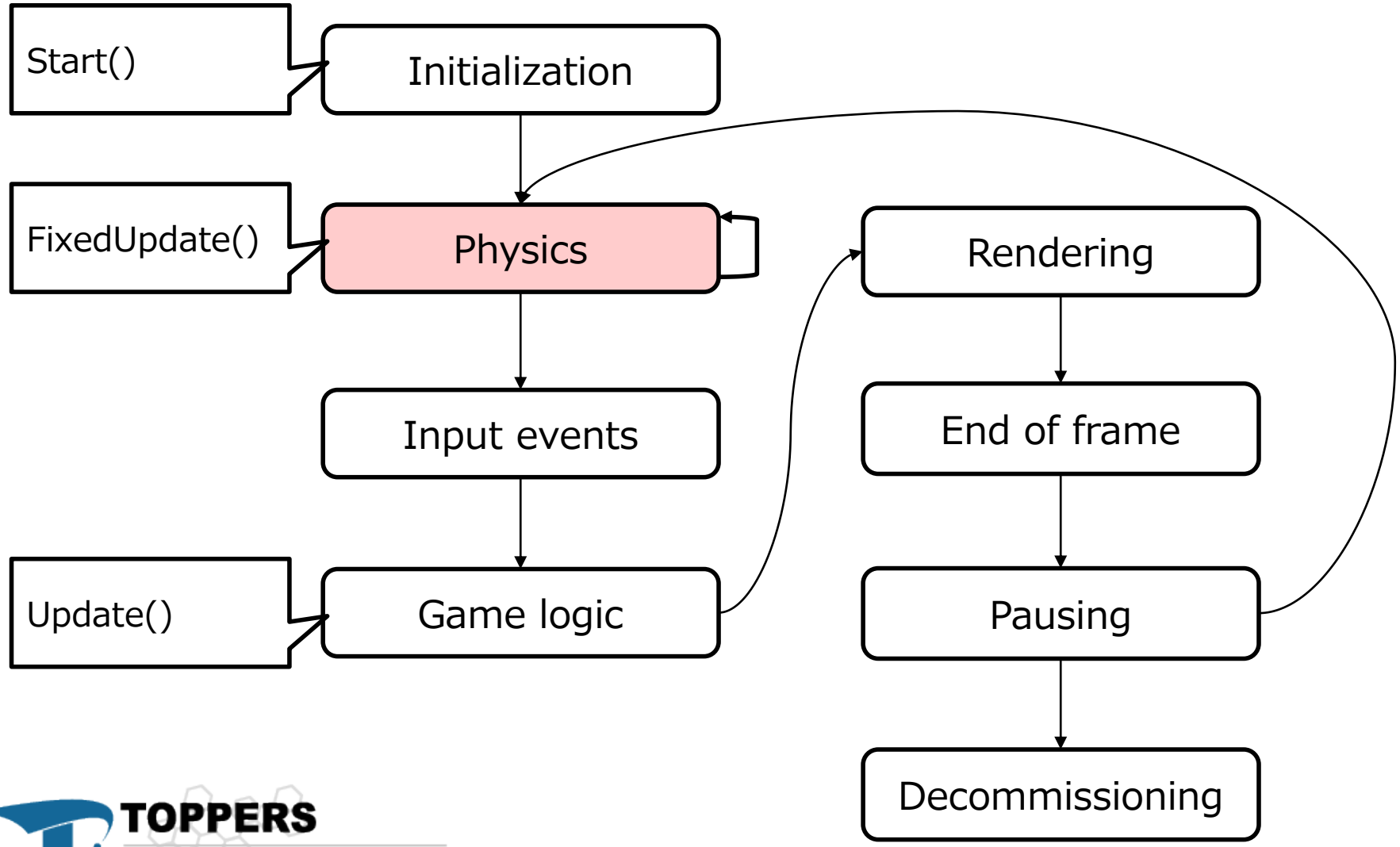
フレーム毎に呼び出されます

独自処理(Unityスクリプト)の追加方法（２／２）

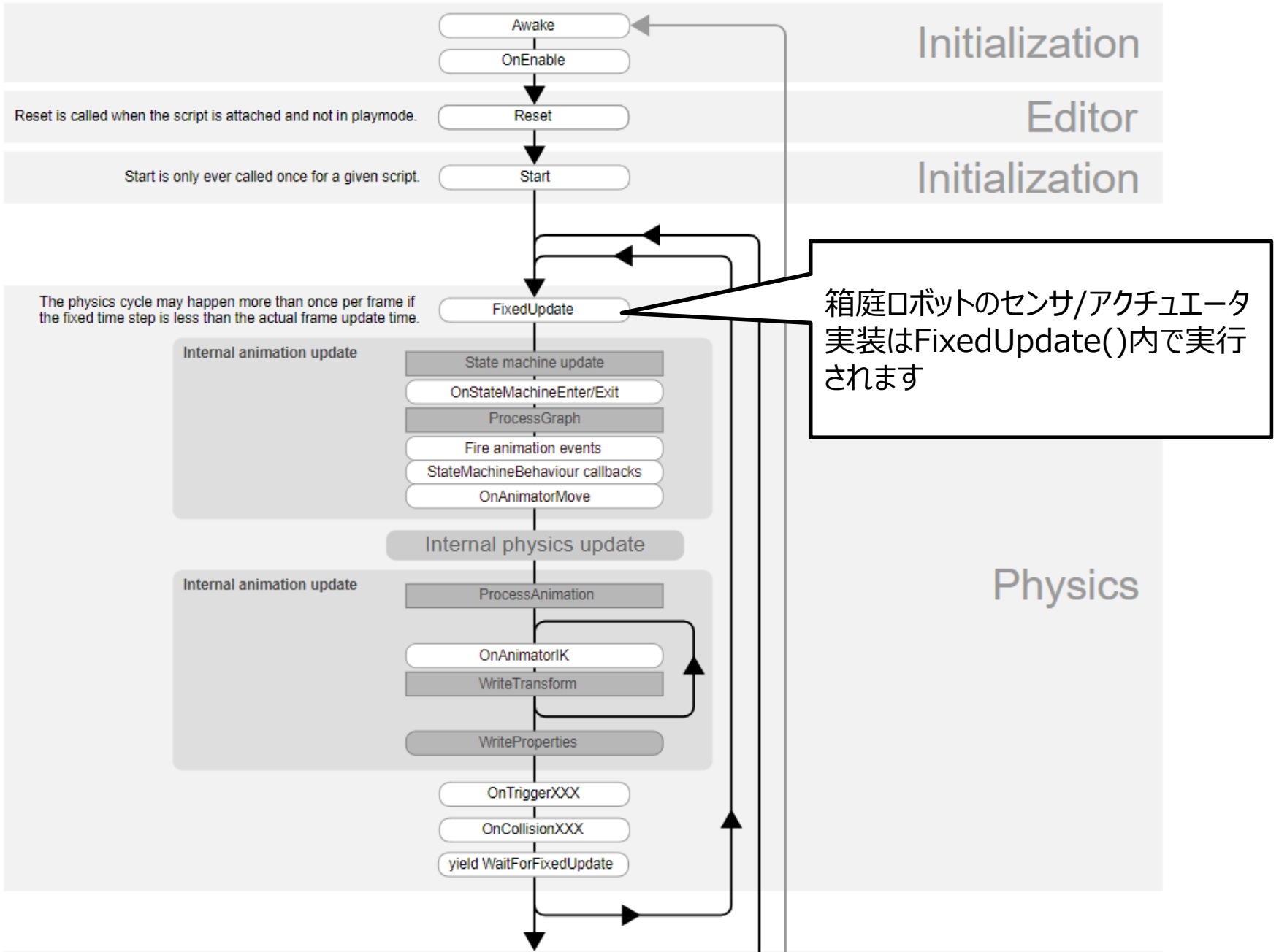


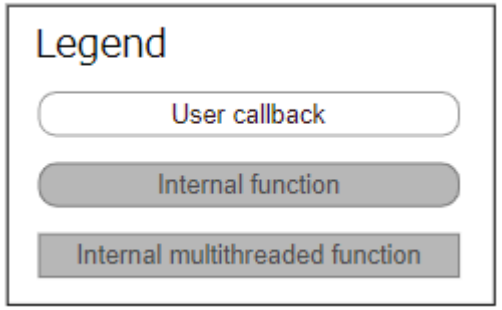
Unityイベント関数の実行順序

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



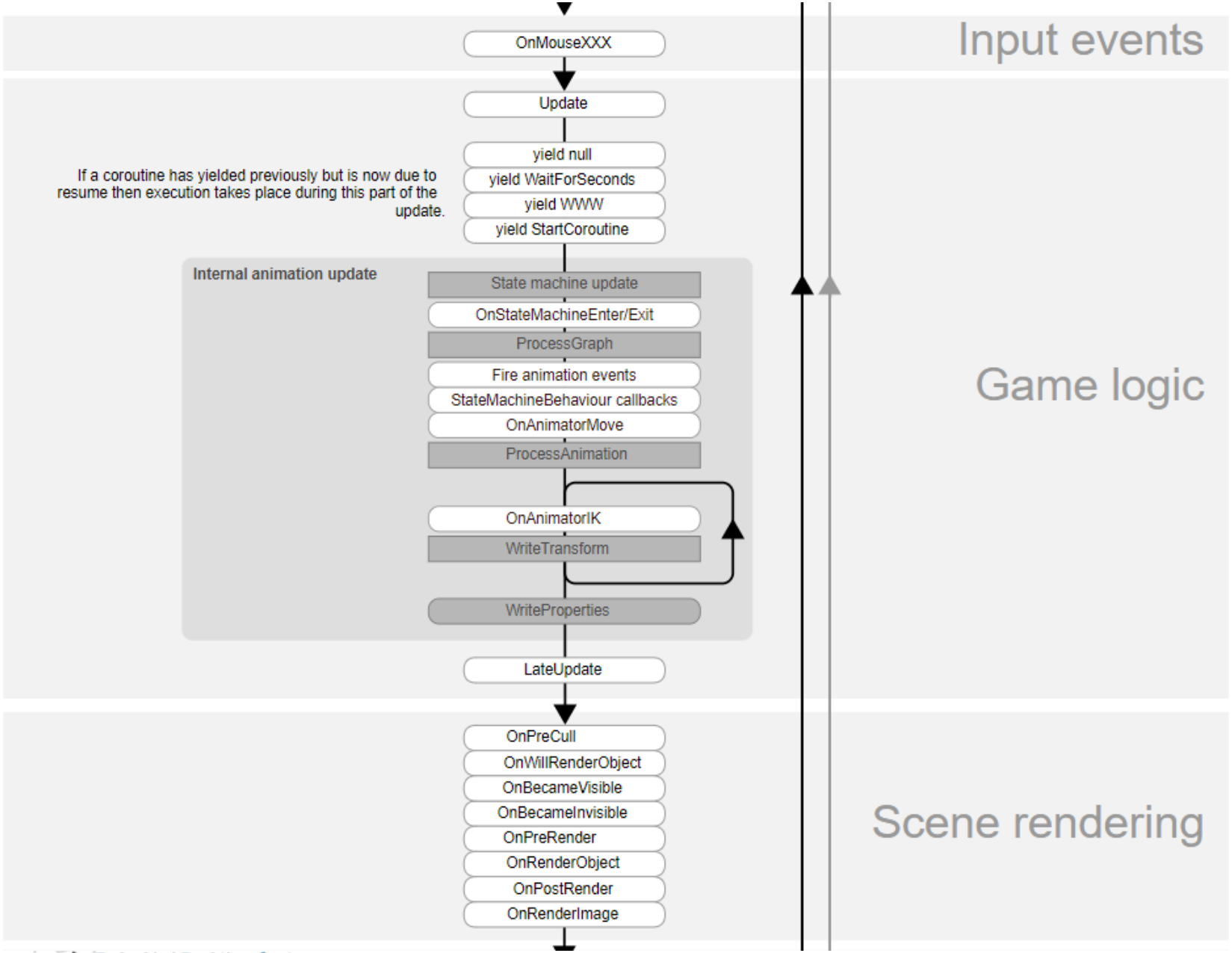
Unityイベント関数の実行順序

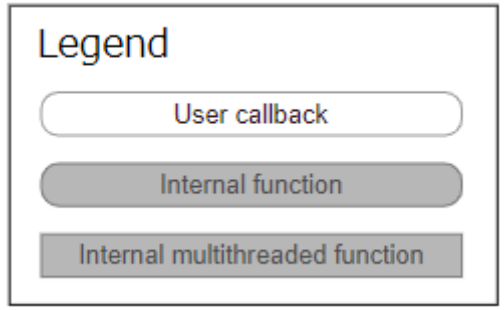




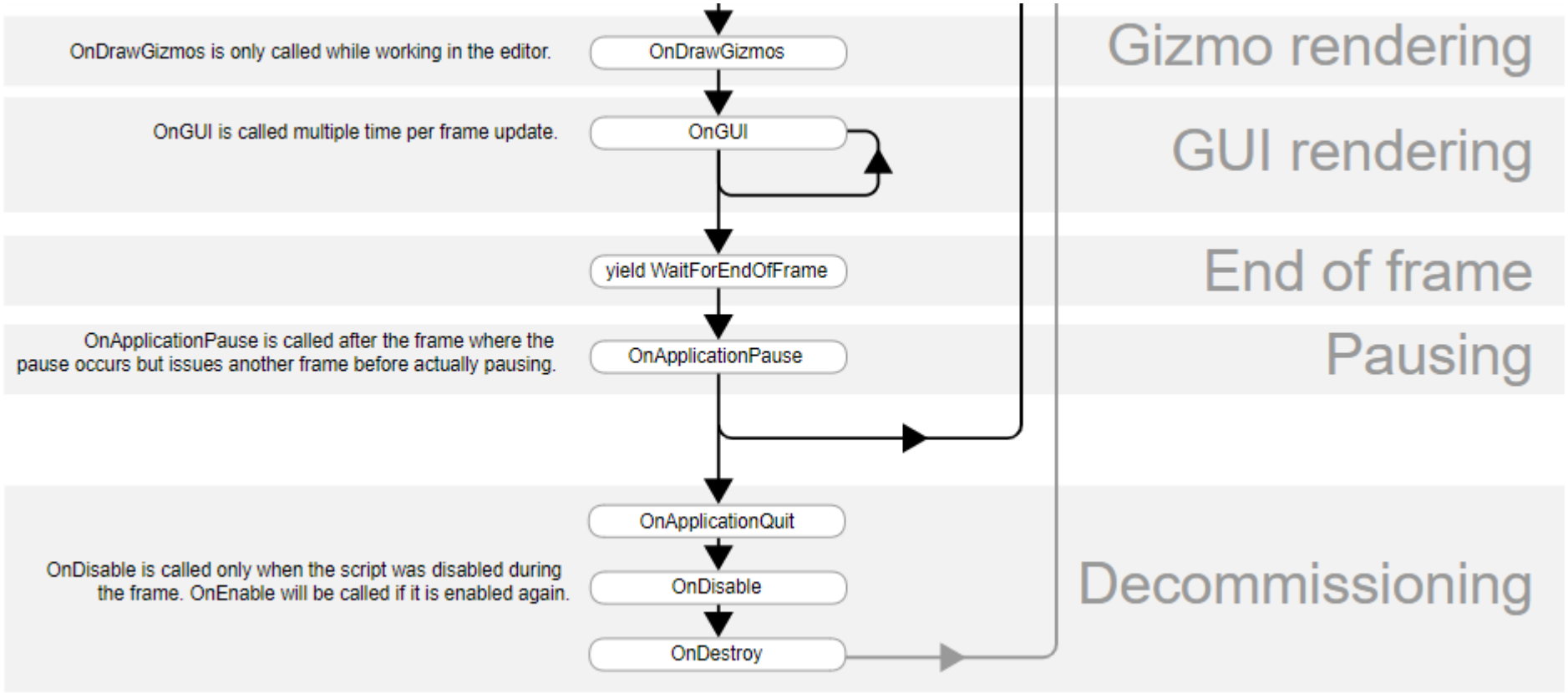
箱庭ロボットのセンサ/アクチュエータ実装はFixedUpdate()内で実行されます

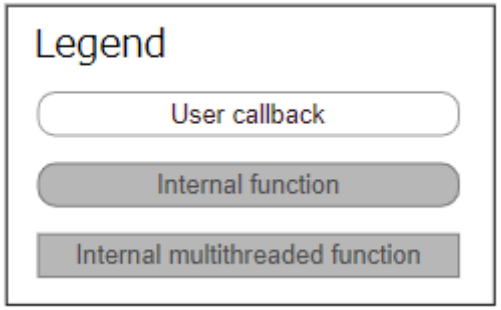
Unityイベント関数の実行順序





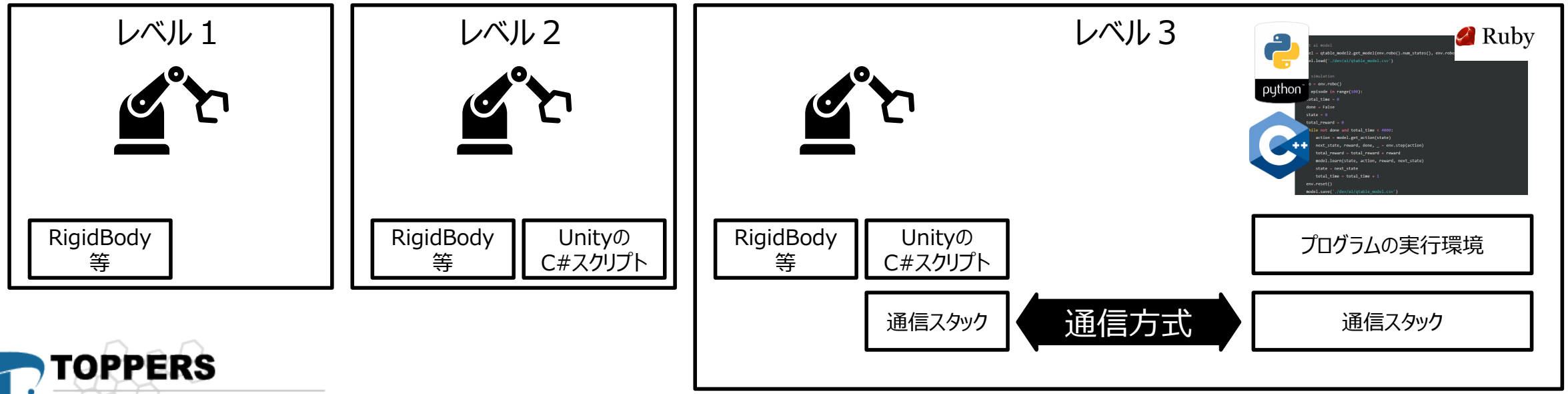
Unityイベント関数の実行順序





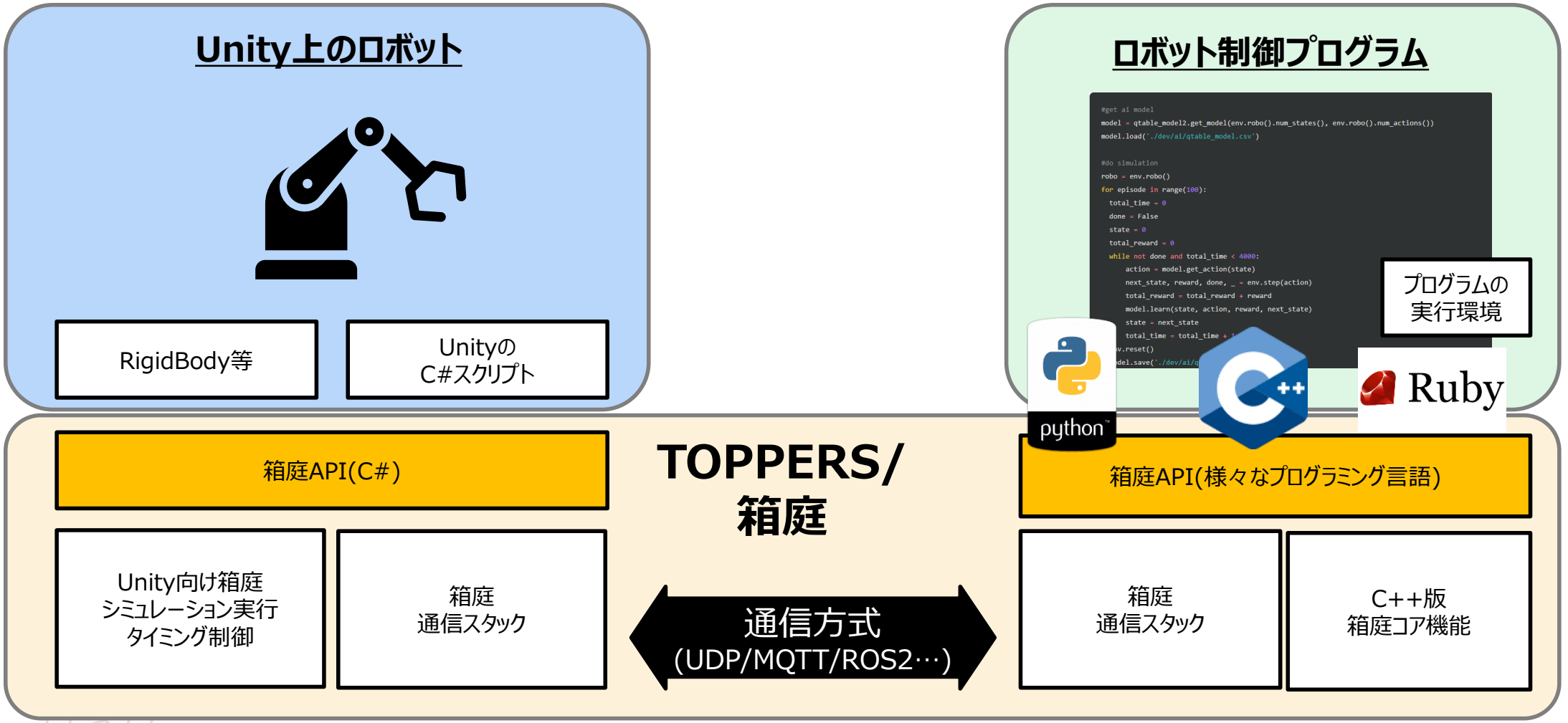
ロボットを動かすためのUnityの構造

- 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
  1. UnityのRigidBodyやArticulationBodyをアタッチしてロボットを動かす
  2. UnityのC#スクリプトでロボットを制御する
  3. 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、、、

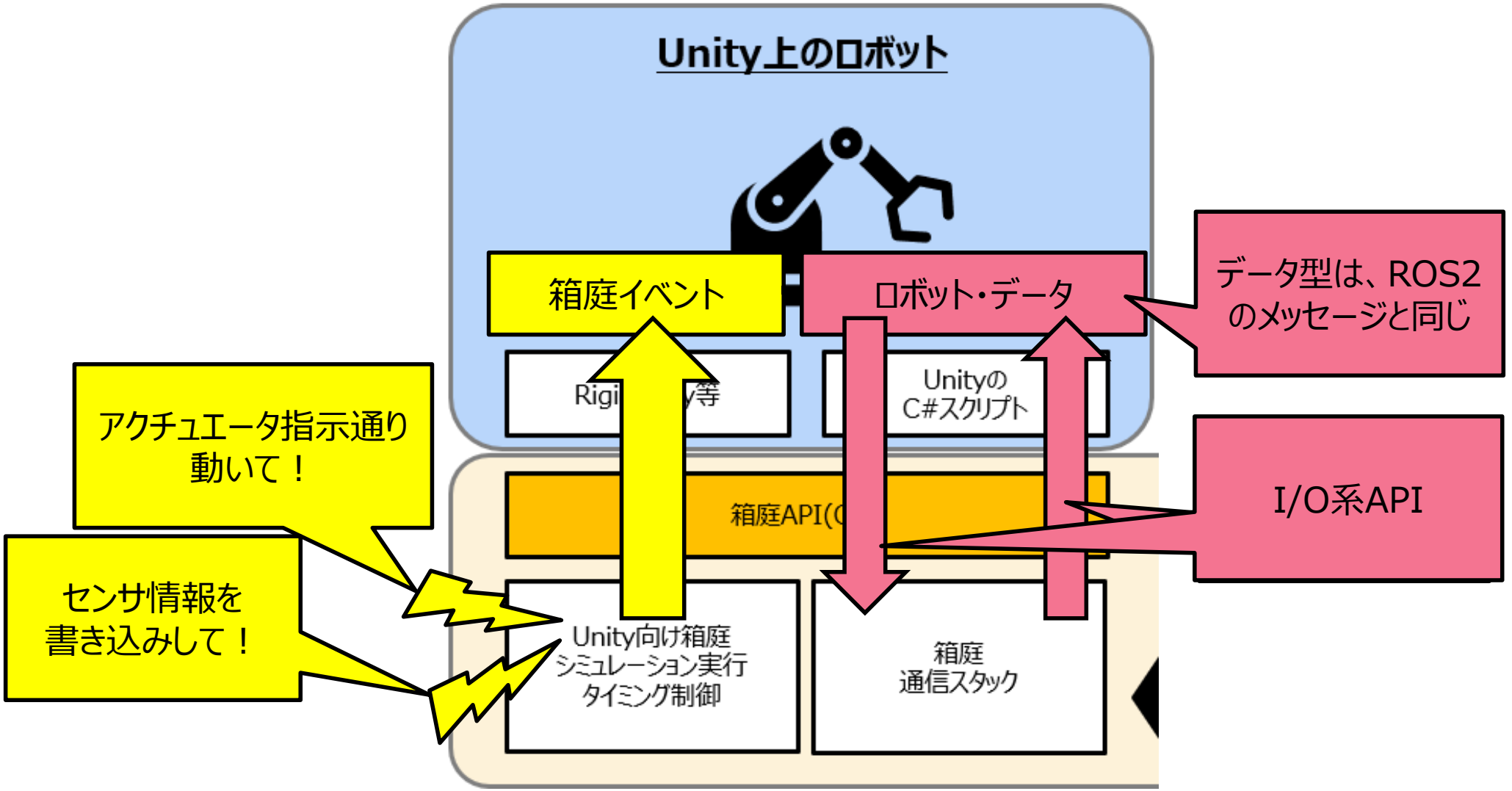


箱庭機能との接合部分

- 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概要

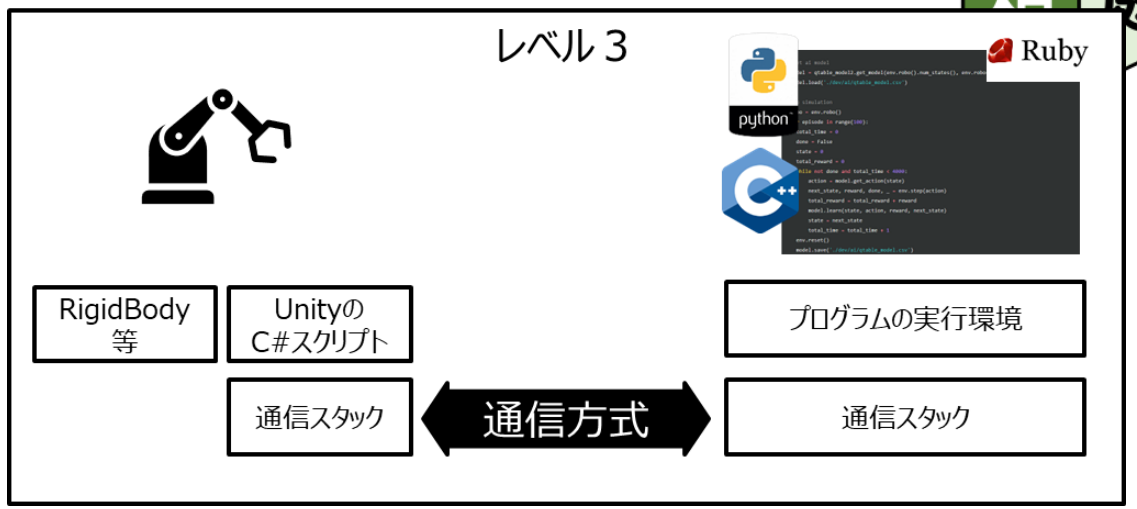

Unity上のロボット
箱庭イベント
ロボット・データ
データ型は、ROS2
のメッセージと同じ
Rigi
等
Unityの
C#スクリプト
アクチュエータ指示通り
動いて！
箱庭API(C
I/O系API
センサ情報を
書き込みして！
Unity向け箱庭
シミュレーション実行
タイミング制御
箱庭
通信スタック

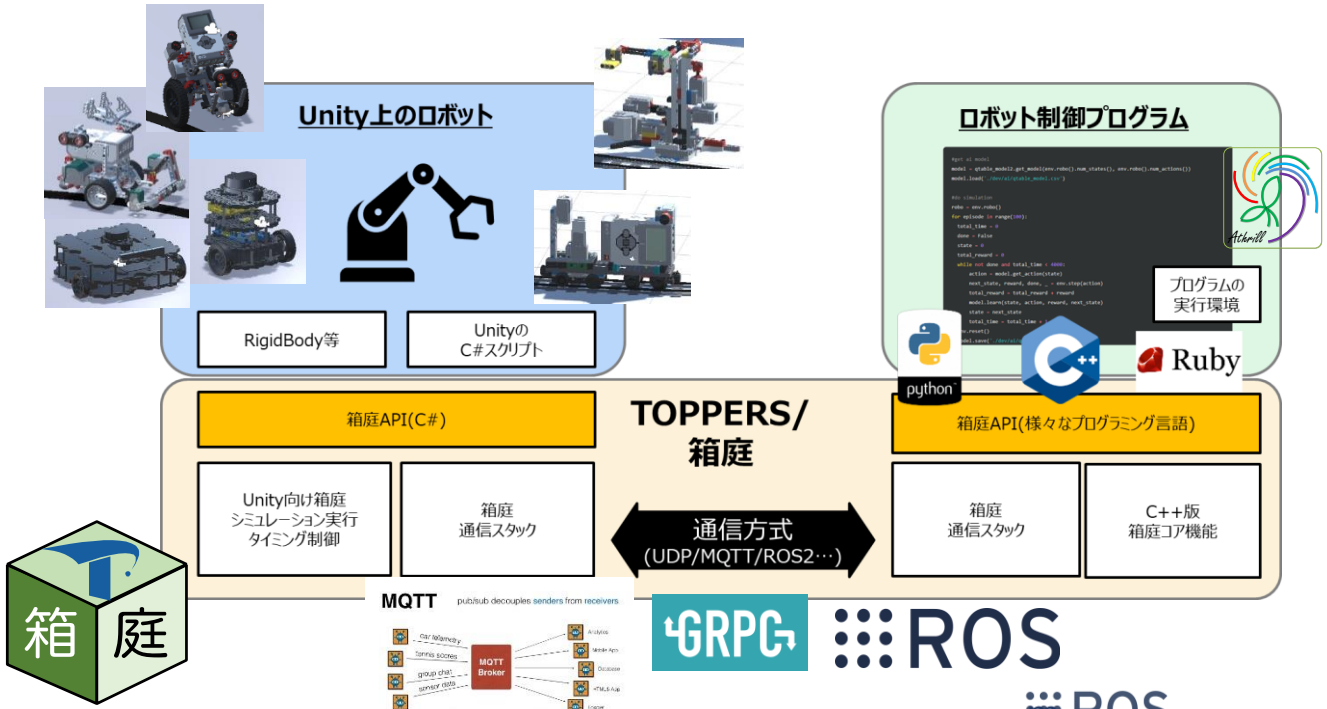
箱庭API(C#)

- 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くなるのか

- 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





TOPPERS
Toyohashi OPen Platform
for Embedded Real-time Systems

箱庭ロボットの作り方

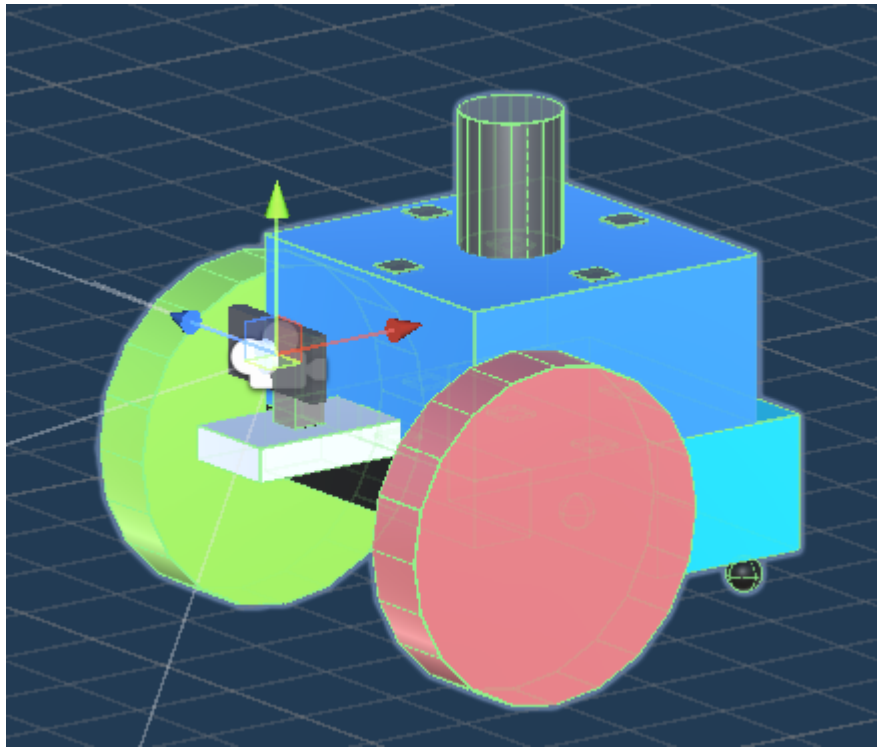
- **箱庭ロボットの組み立て方**
- 箱庭ロボット部品の構成
- 箱庭ロボット部品の構造

ベースのアイディア

@kanetugu2018 (TOPPERSプロジェクト)　Organization

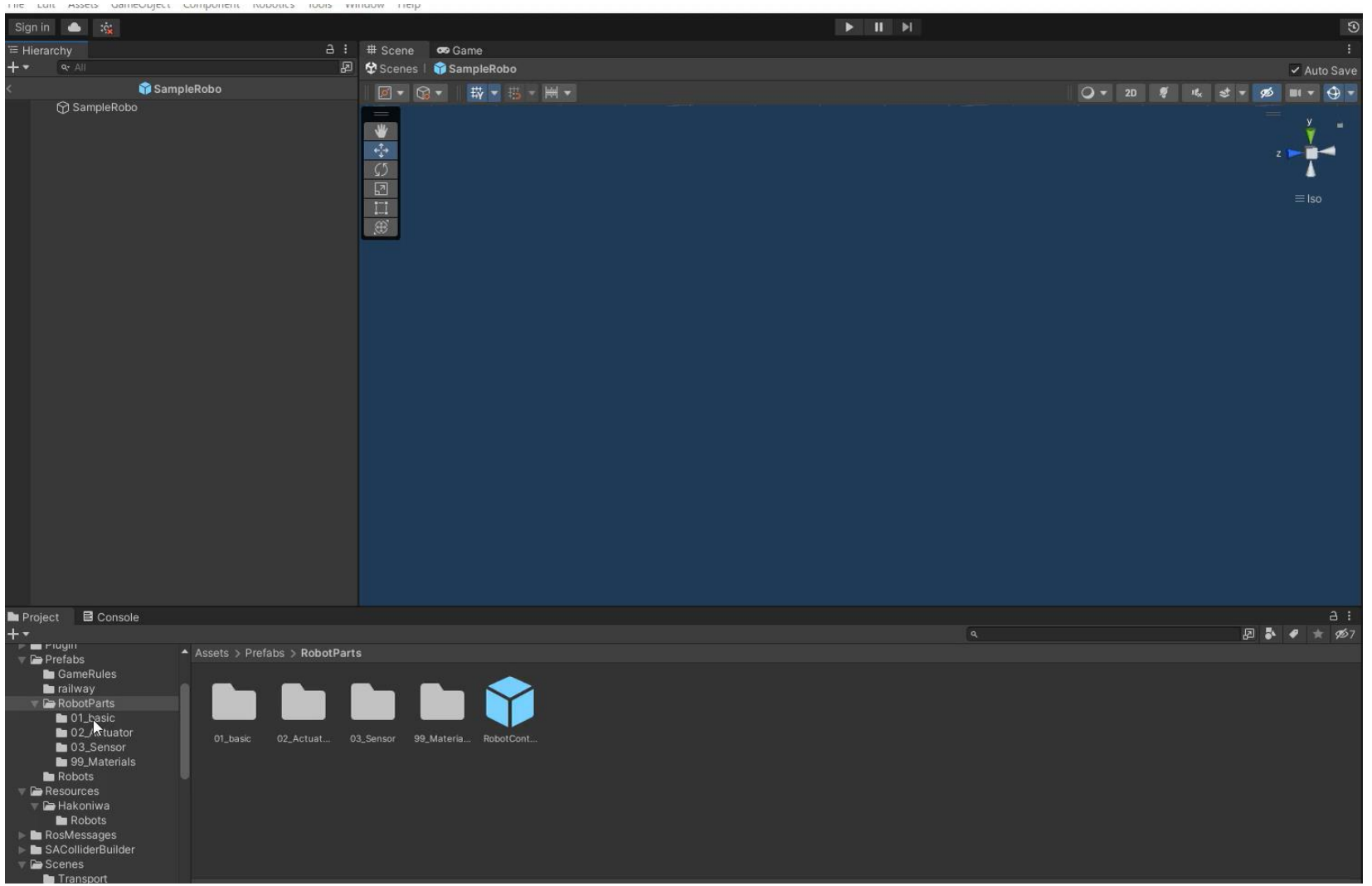
投稿日 2022年01月23日　更新日 2022年05月07日　2878 views

Unity上のロボット部品を組み立てて，ROS2でコントロール

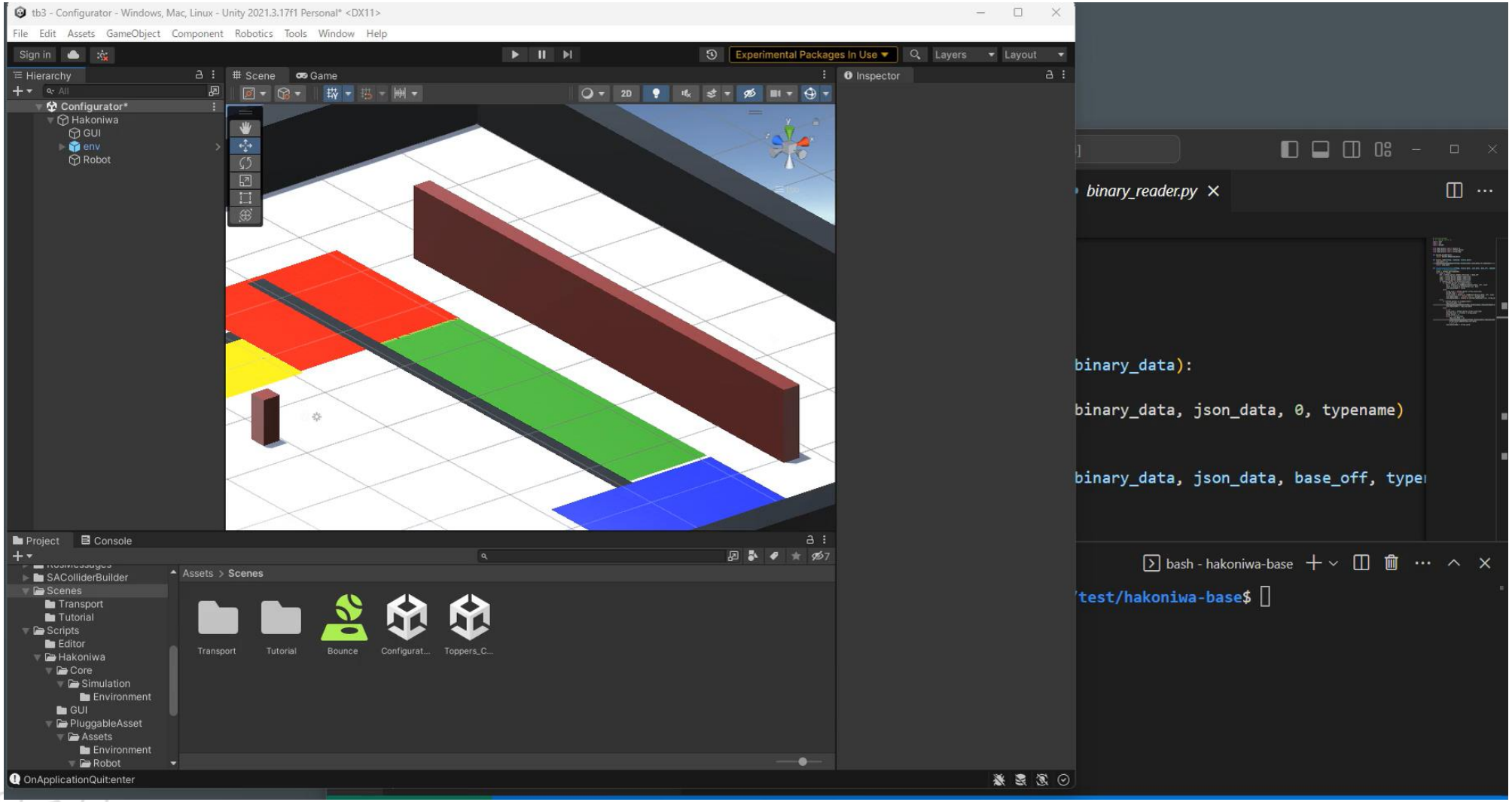


- URL：
  - https://qiita.com/kanetugu2018/items/29ee98bc434191eee8f1

TurtleBot3風のロボットを作ってみよう！



作ったロボットを動かしてみよう！



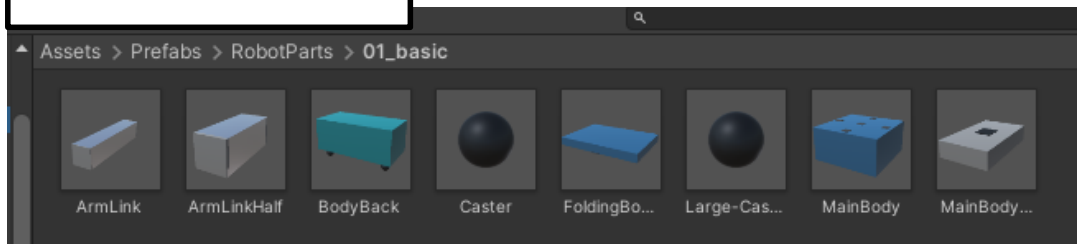
ロボット制御用のPythonプログラム

```
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])
```

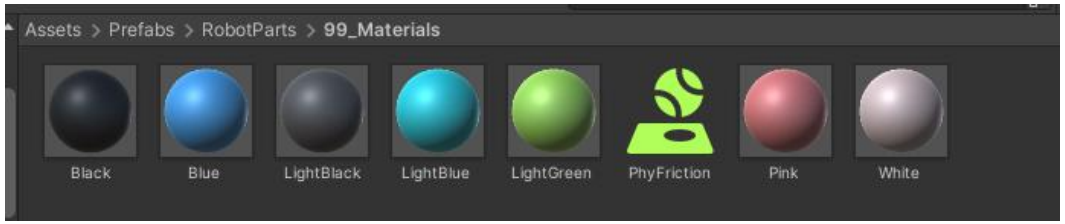
箱庭ロボット部品の構成

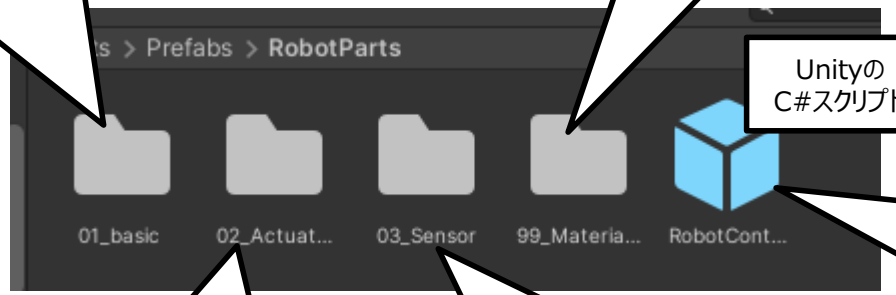
基本部品

ArticulationBody



マテリアル





Unityの
C#スクリプト

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

ArticulationBody

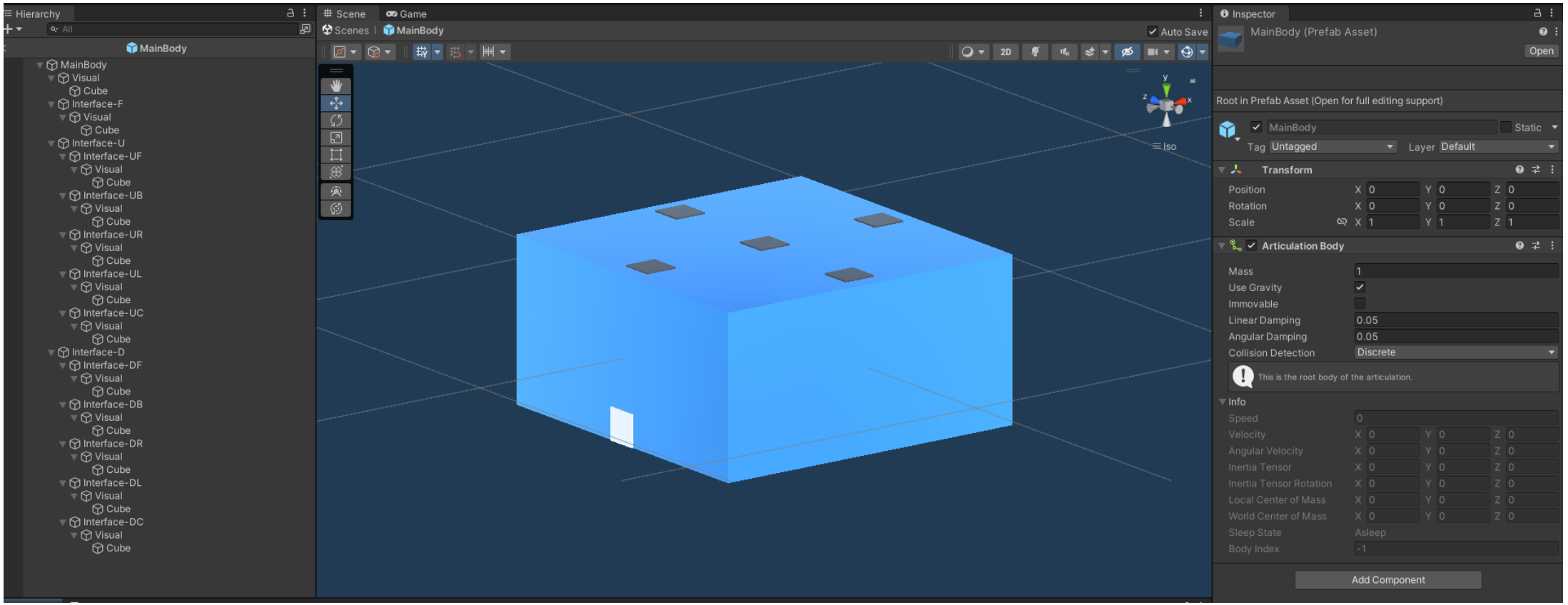
Unityの
C#スクリプト

センサ

Unityの
C#スクリプト

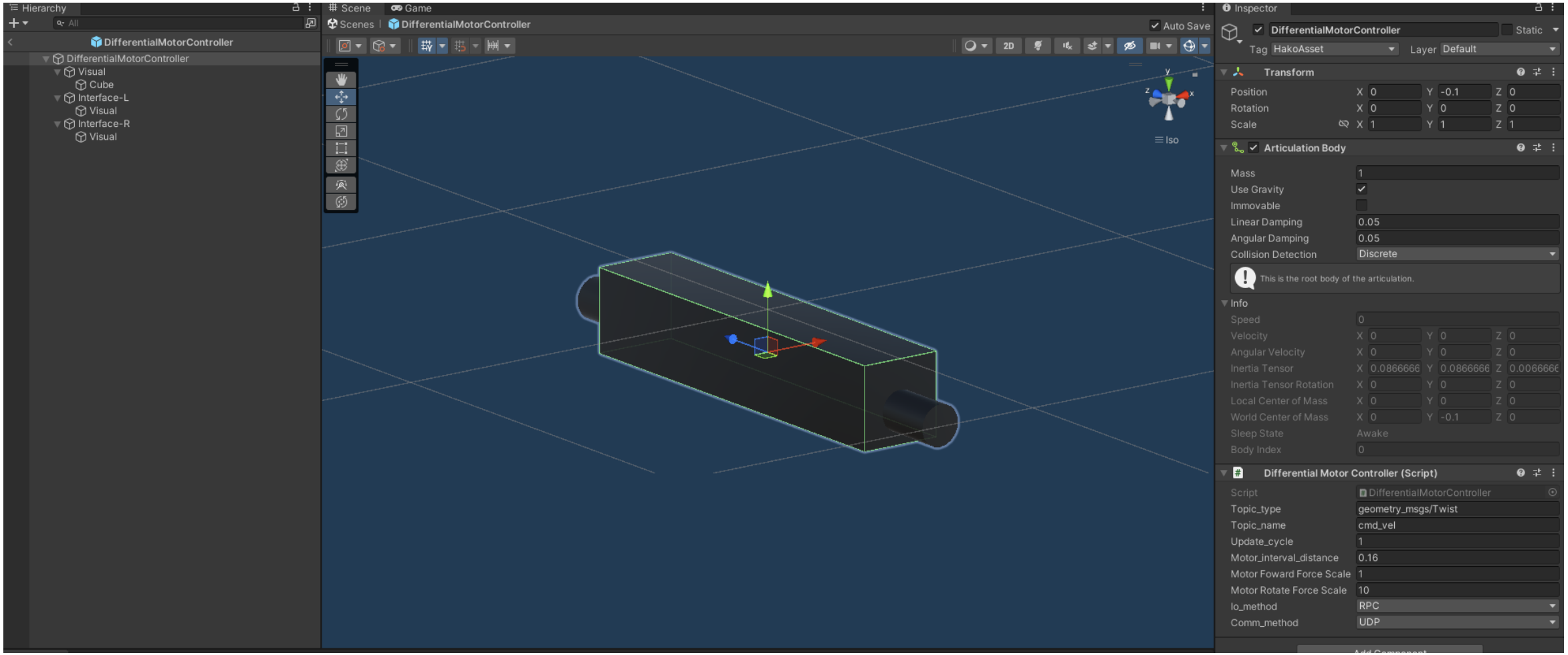
基本部品例(MainBody)

- ArticulationBodyで構成されている



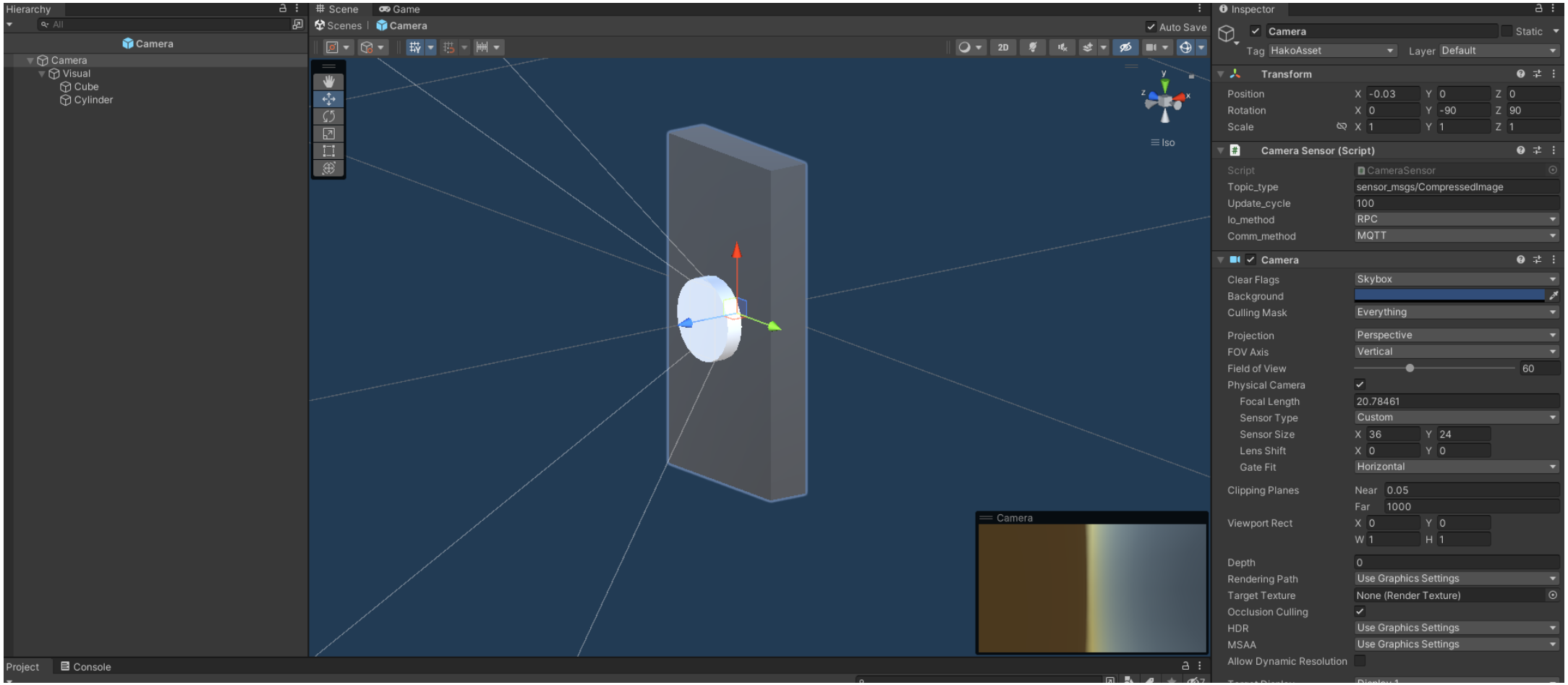
アクチュエータ例(DiffrentialMotorController)

- 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)

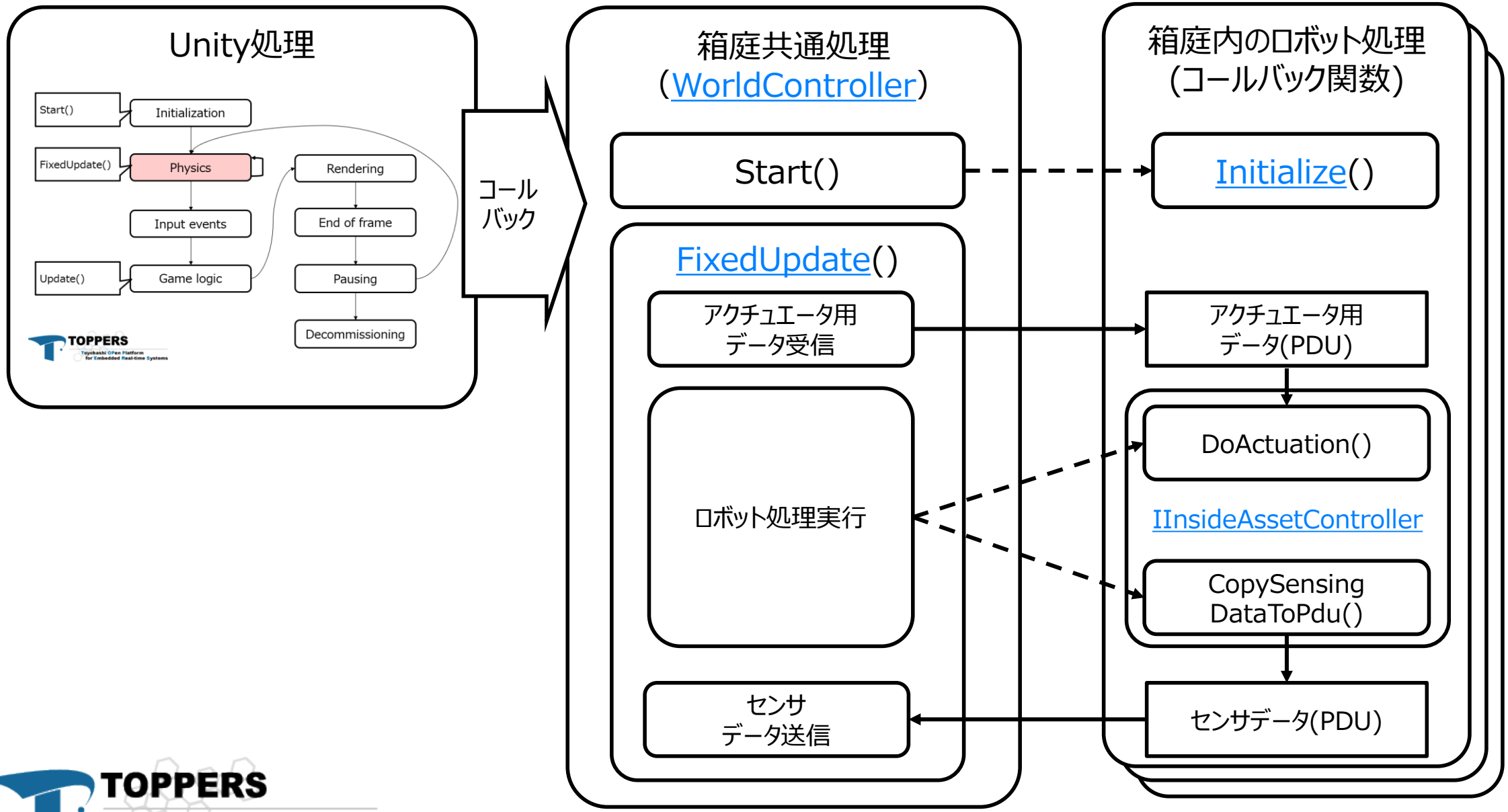
- CameraとUnityスクリプトで構成されている



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

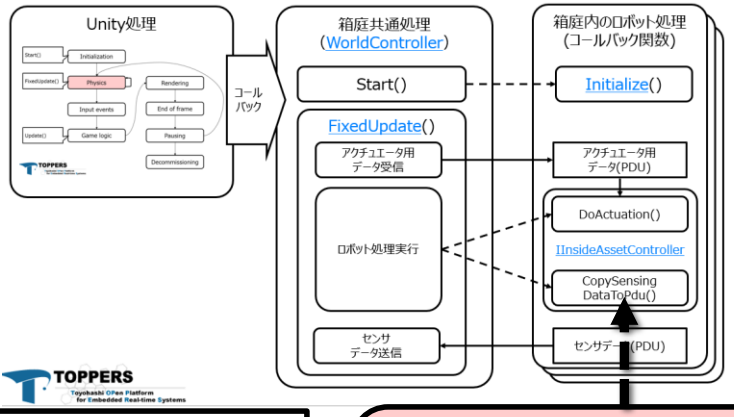
箱庭の全体的な処理の流れ



TOPPERS
Toyohashi OPen Platform
for Embedded Real-time Systems

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



IInsideAssetController
インタフェース実装

【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いけない

RobotPartsRoot

DoActuation()

CopySensing DataToPdu()

IRobotPartsController(インタフェース)
【役割】
アクチュエータの制御コントローラ用のインタフェース
【インタフェース仕様】
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

IRobotPartsController

DoControl()

IRobotPartsSensor

UpdateSensor Values()

IRobotPartsSensor(インタフェース)
【役割】
センサ用のインタフェース
【インタフェース仕様】
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(インタフェース)
【役割】
モーター制御用のインタフェース
【インタフェース仕様】
SetTargetVeolcity()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

IRobotPartsMotor

SetTargetVelocity()

IRobotPartsPincherFinger

UpdateGrip()

IRobotPartsActuator

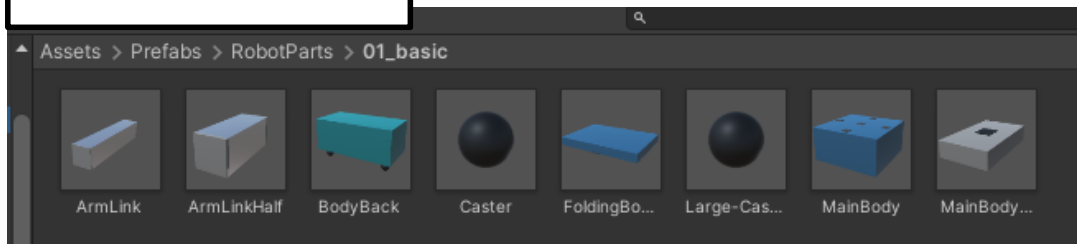
IRobotPartsActuator(インタフェース)
【役割】
アクチュエータ用の共通インタフェース

TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

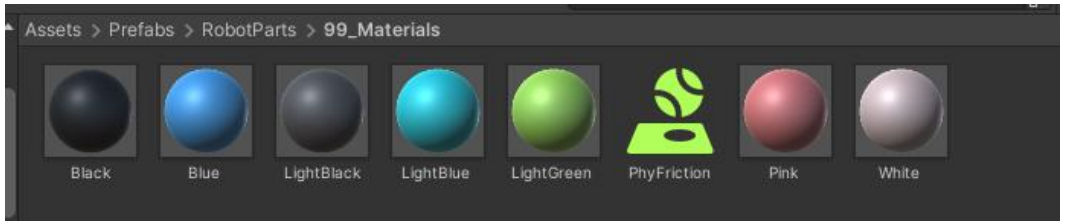
箱庭ロボット部品の構成（もう一回）

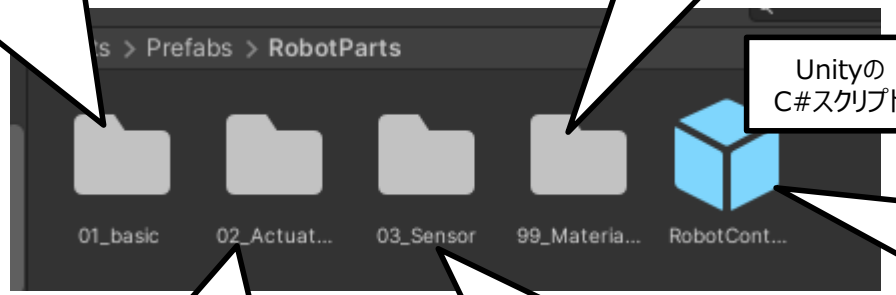
基本部品

ArticulationBody



マテリアル





Unityの
C#スクリプト

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

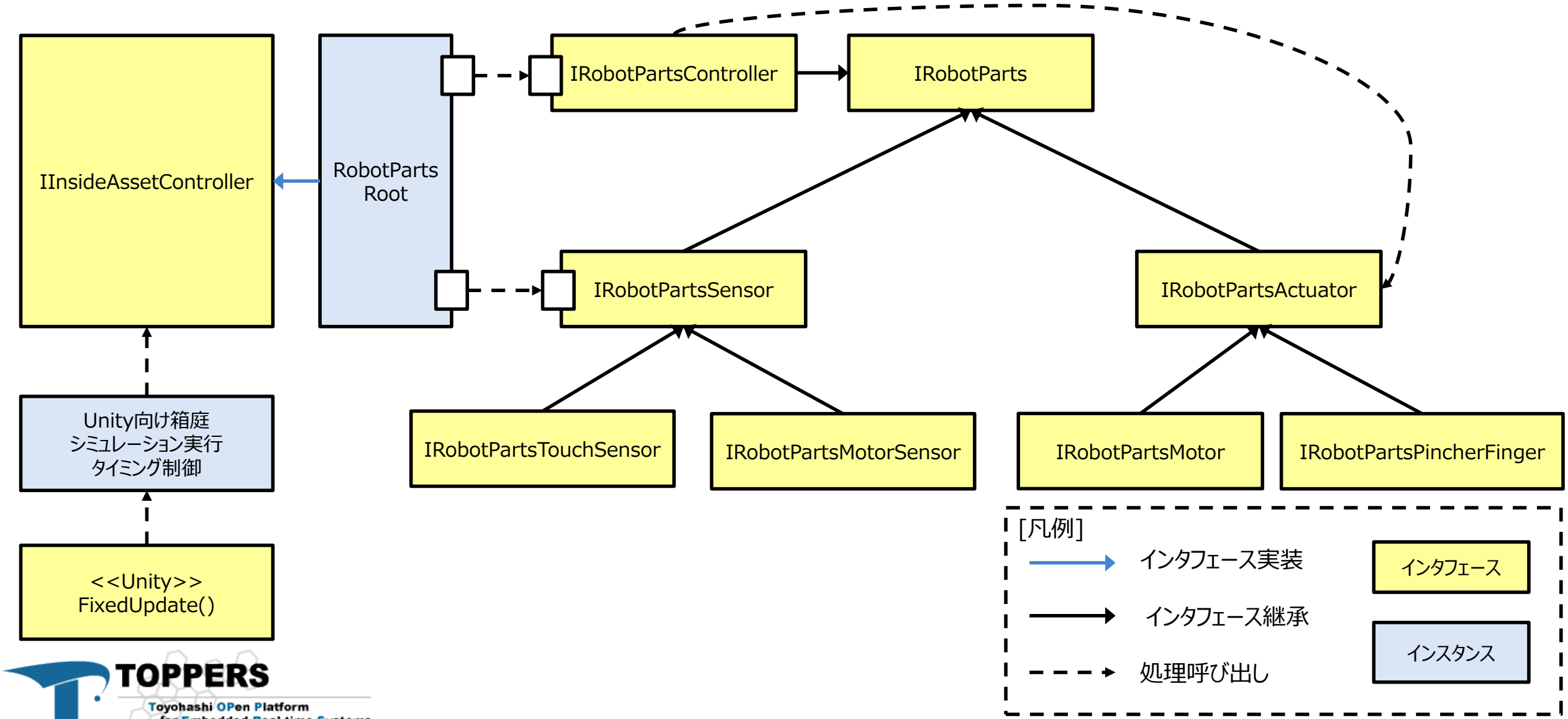
ArticulationBody

Unityの
C#スクリプト

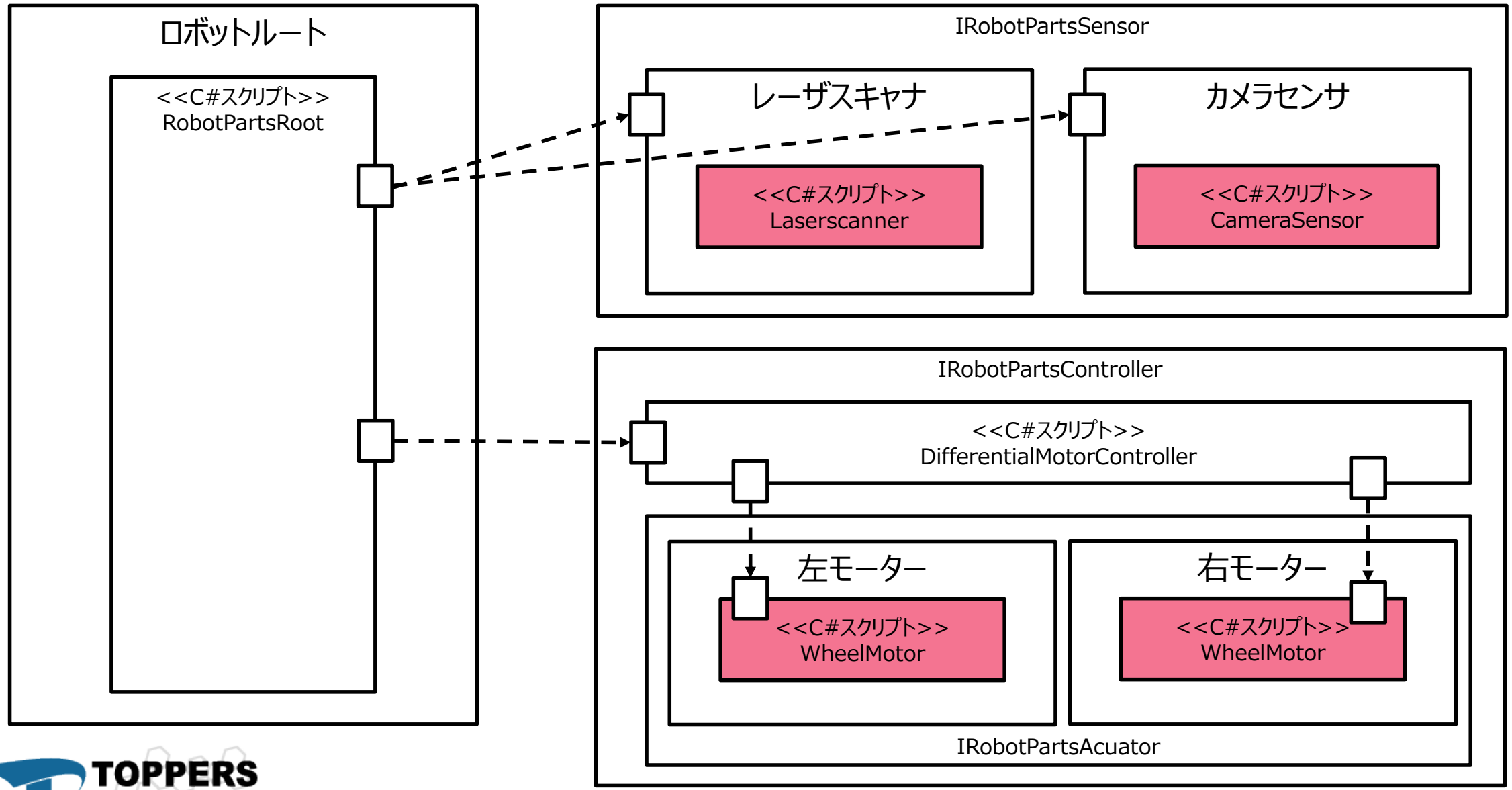
センサ

Unityの
C#スクリプト

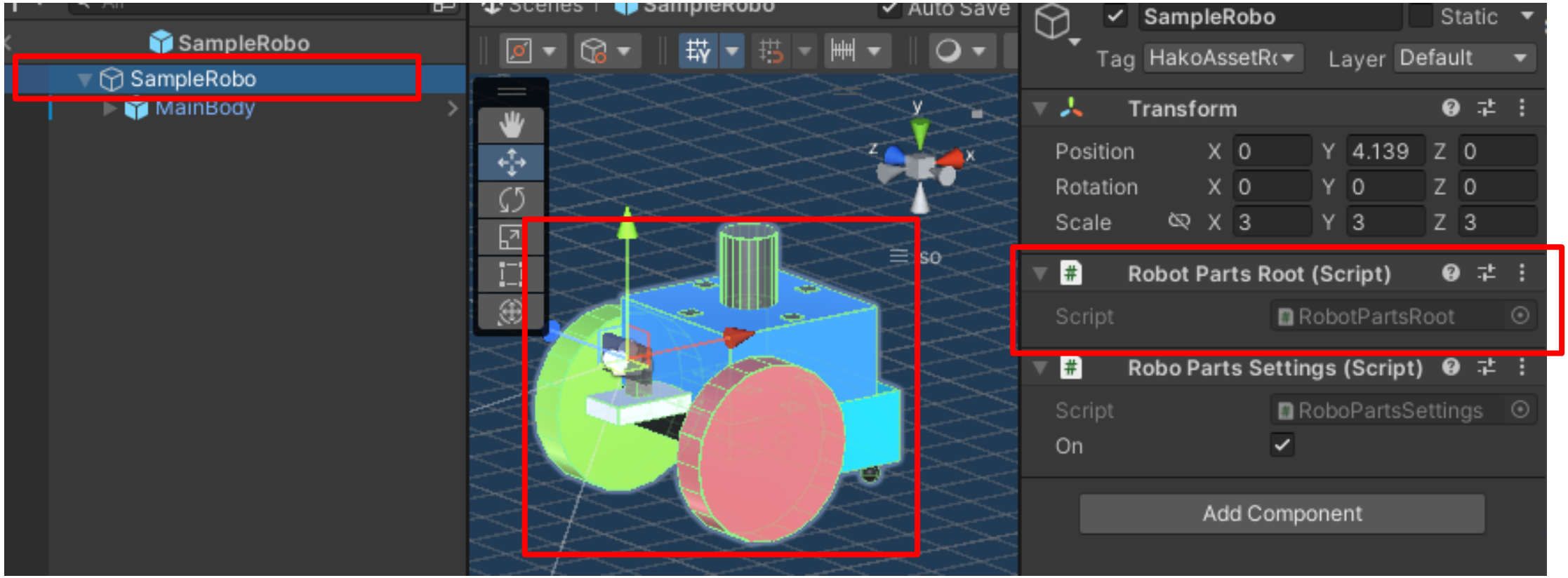
イベント関数インタフェース視点での全体像



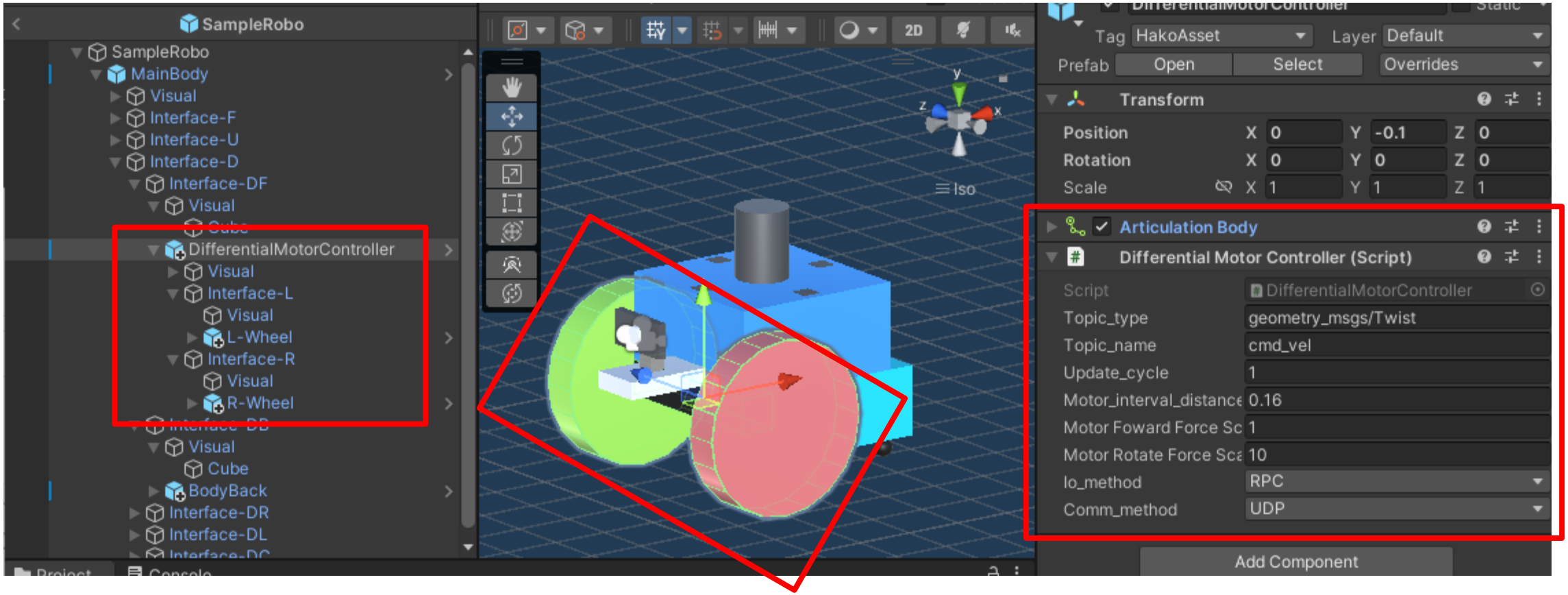
インスタンス例


ロボットルート
<<C#スクリプト>>
RobotPartsRoot
IRobotPartsSensor
レーザスキャナ
<<C#スクリプト>>
Laserscanner
カメラセンサ
<<C#スクリプト>>
CameraSensor
IRobotPartsController
<<C#スクリプト>>
DifferentialMotorController
左モーター
<<C#スクリプト>>
WheelMotor
右モーター
<<C#スクリプト>>
WheelMotor
IRobotPartsAcuator

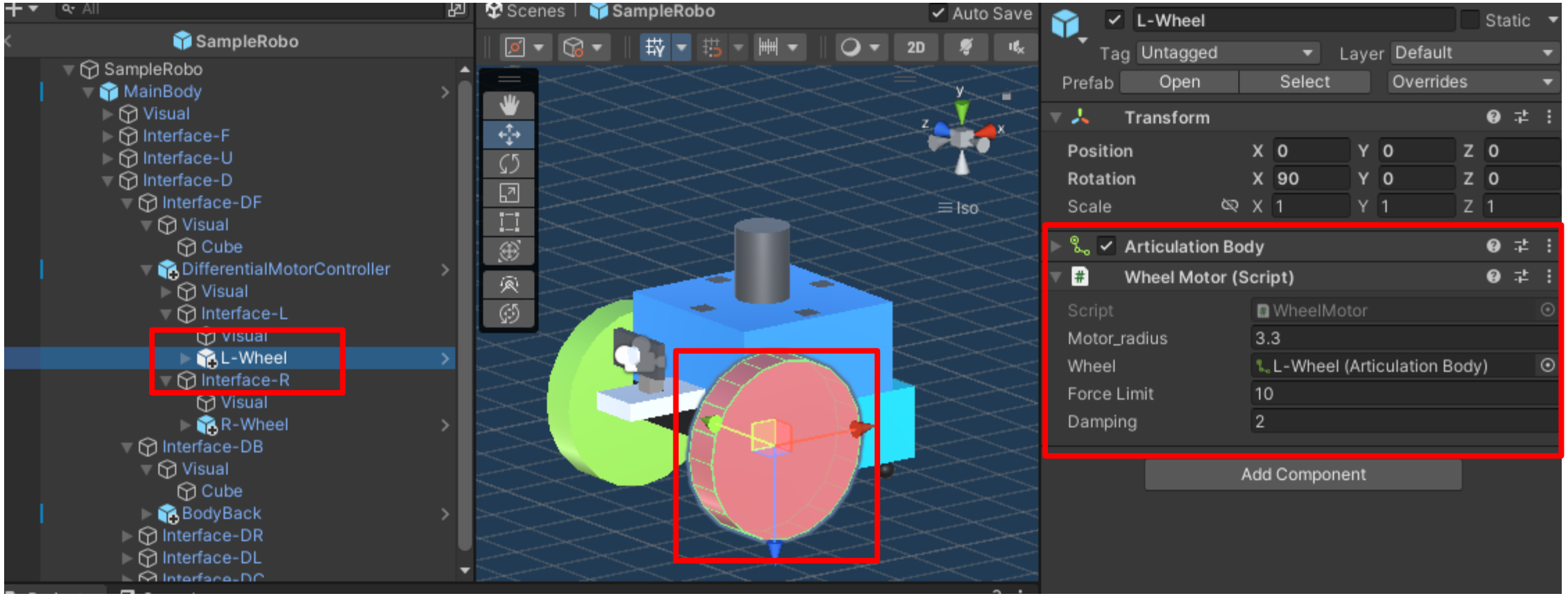
RobotPartsRoot



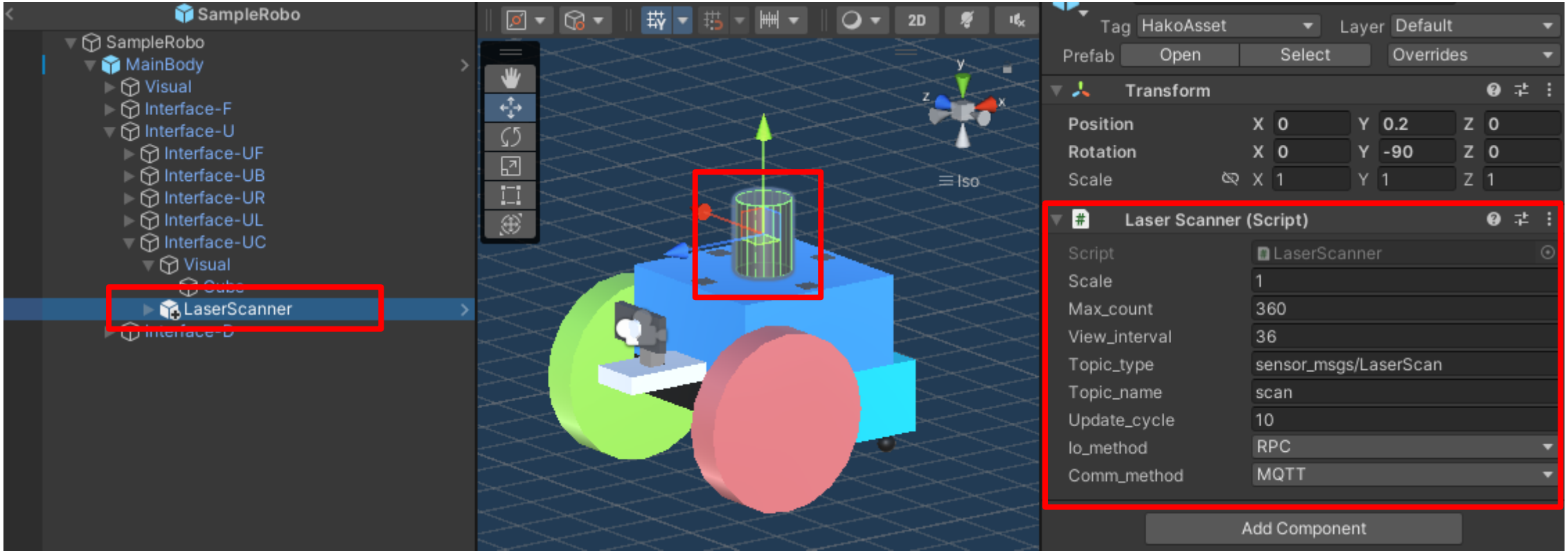
IRobotPartsController：DifferentialMotorController



IRobotPartsActuator : WheelMotor



IRobotPartsSensor : Laserscanner



IRobotPartsSensor : CameraSensor