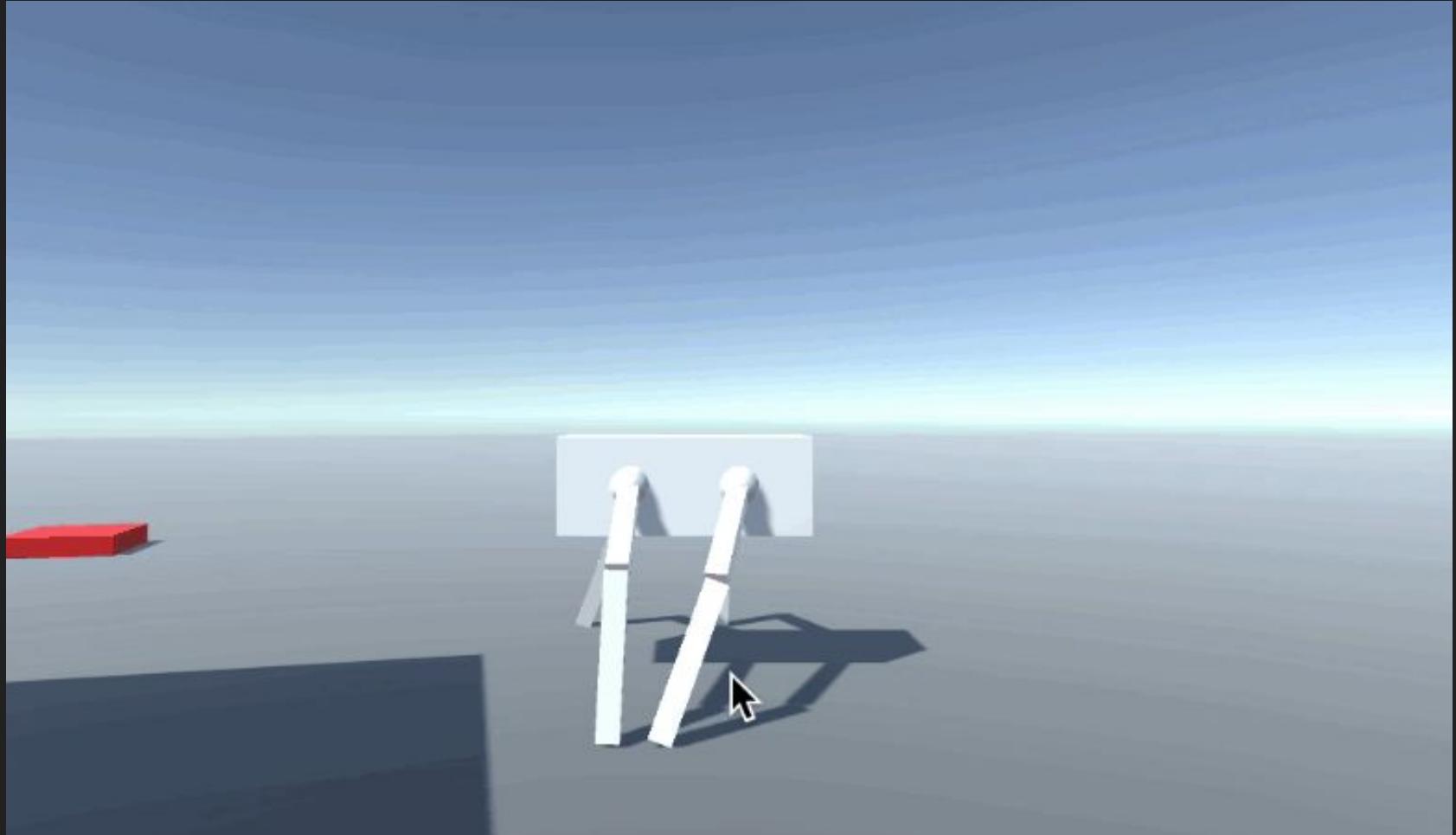
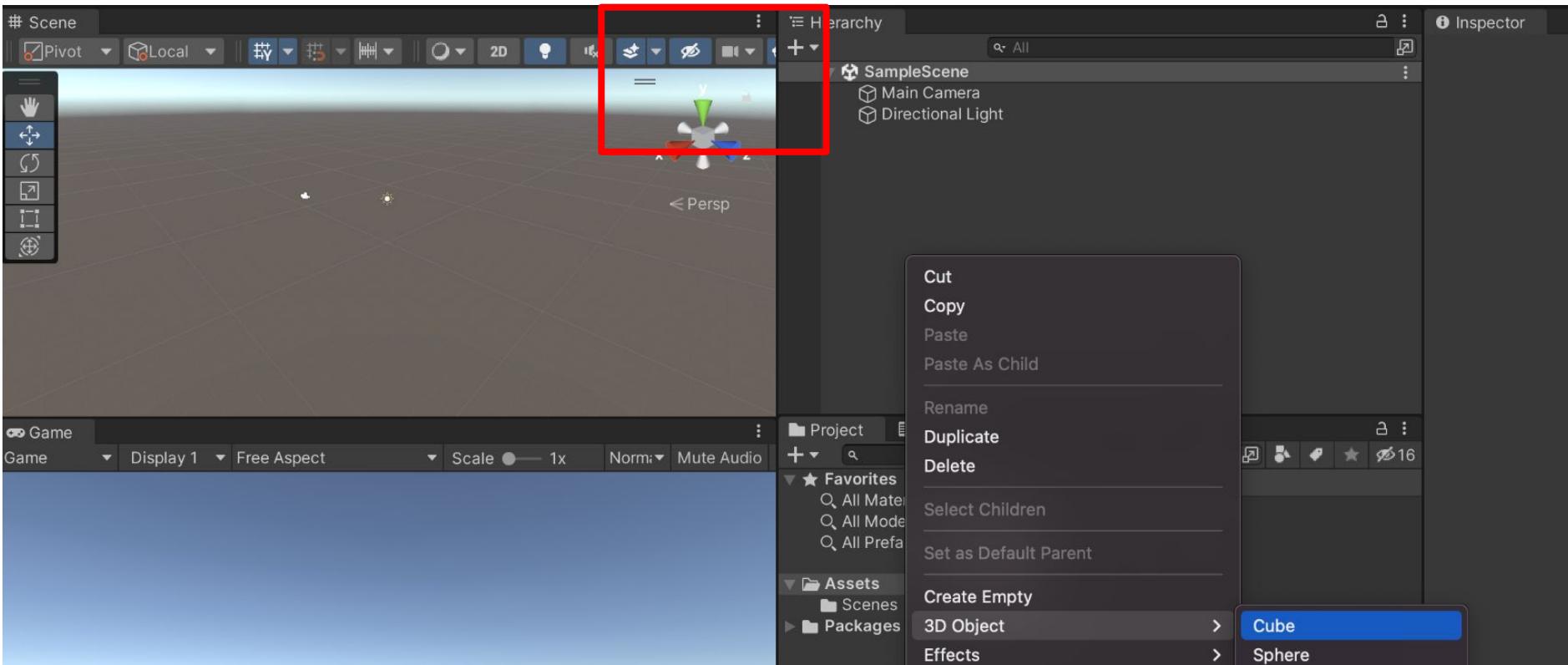
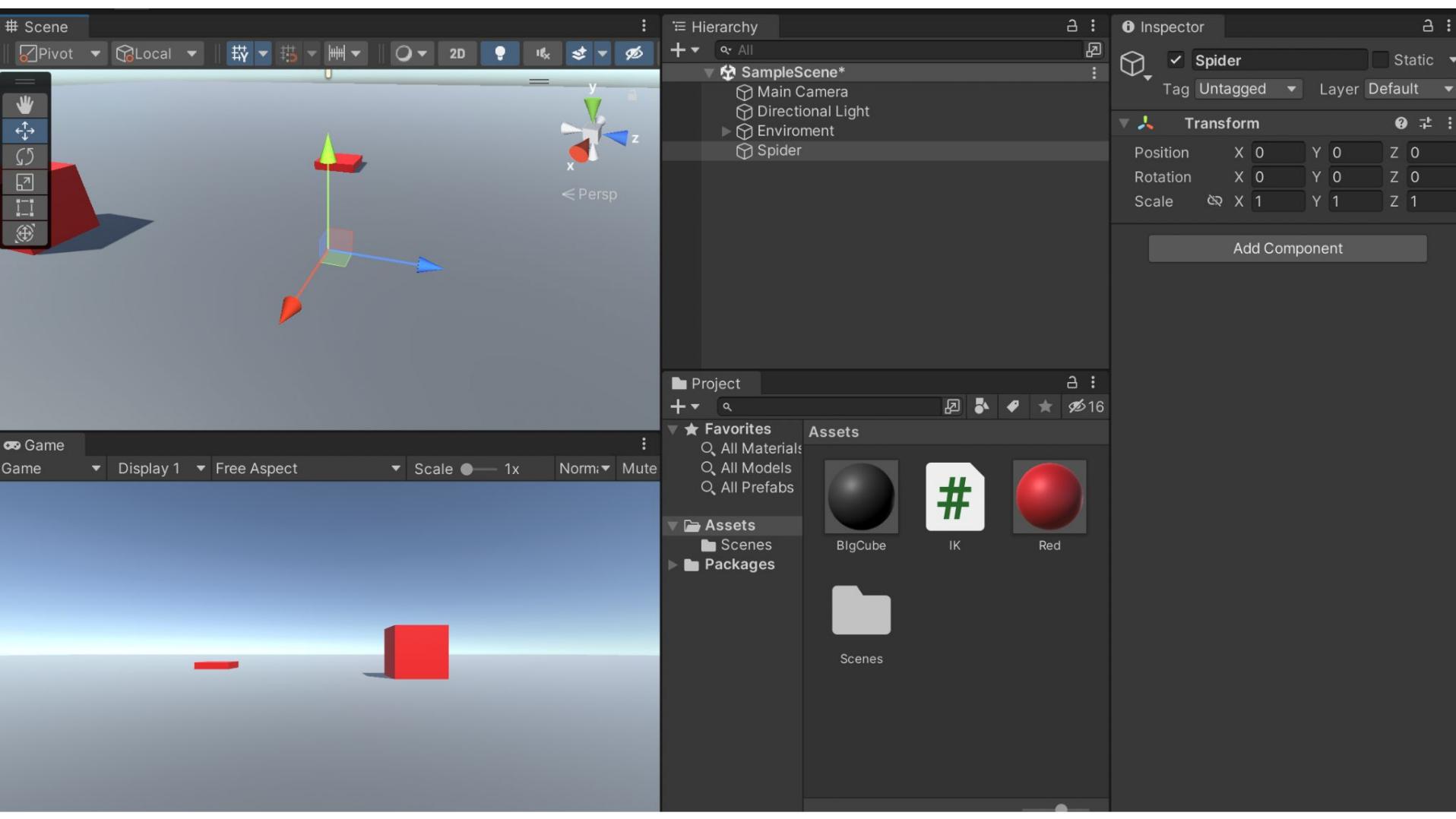


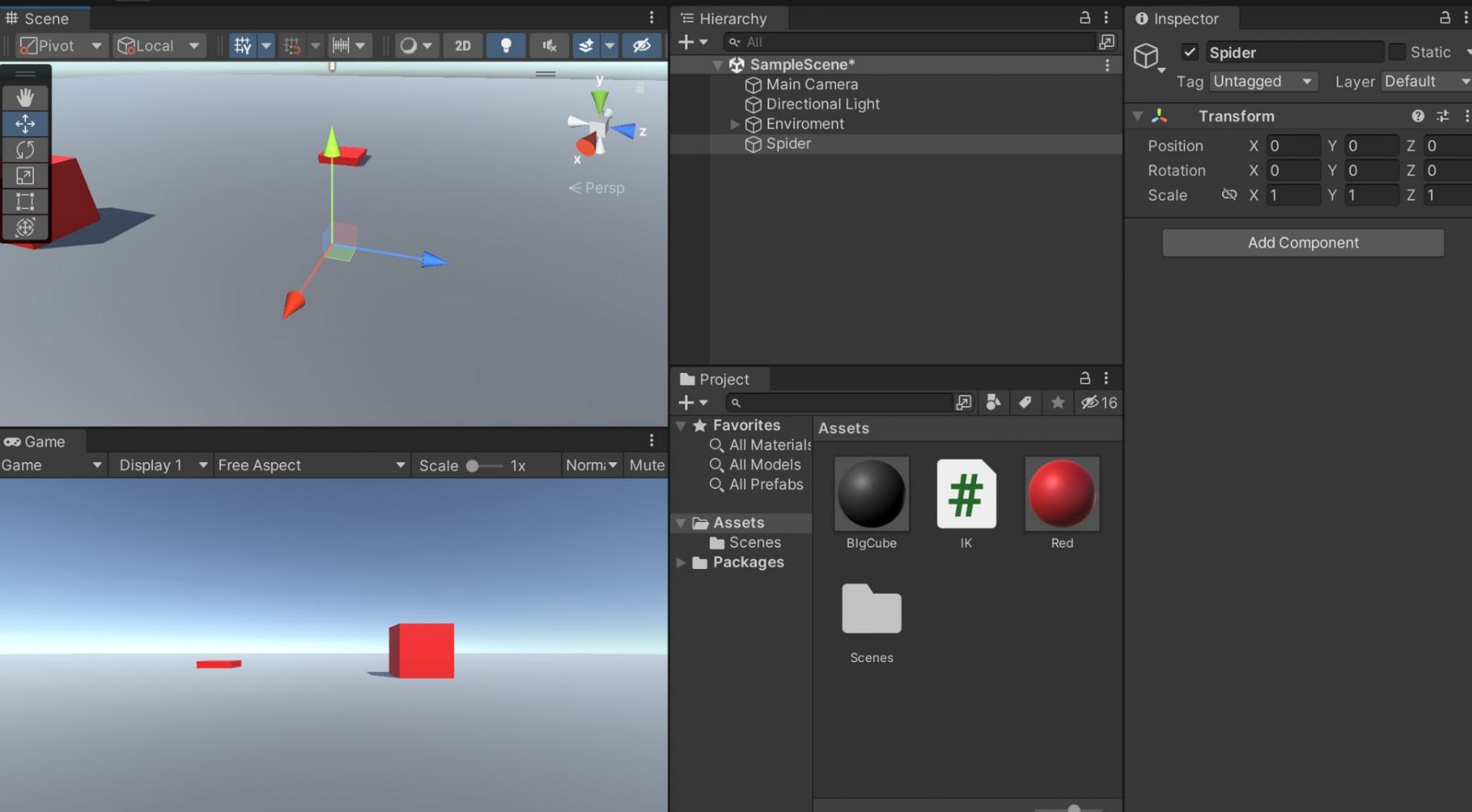
Procedural Animation

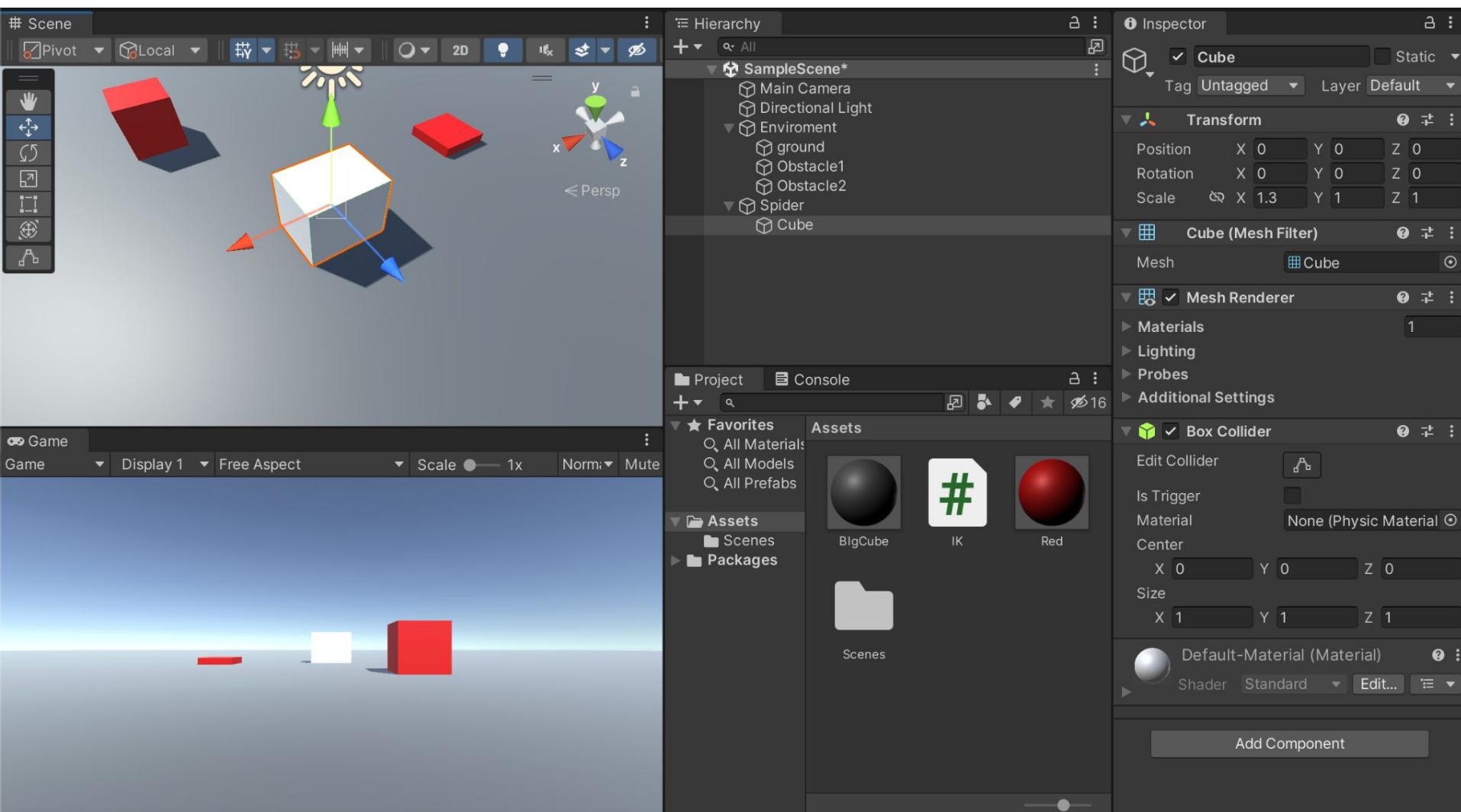


蜘蛛 (歩く)







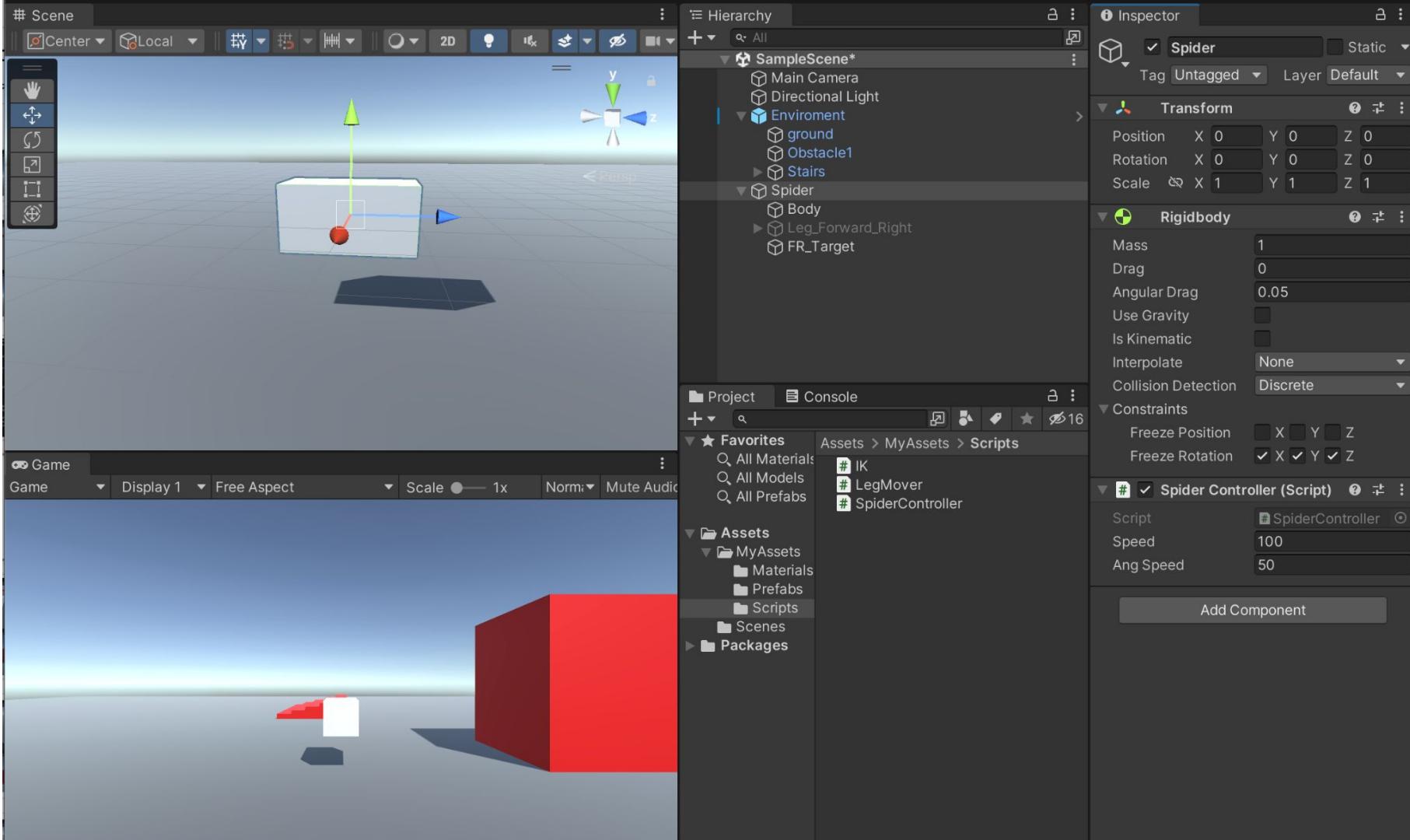


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

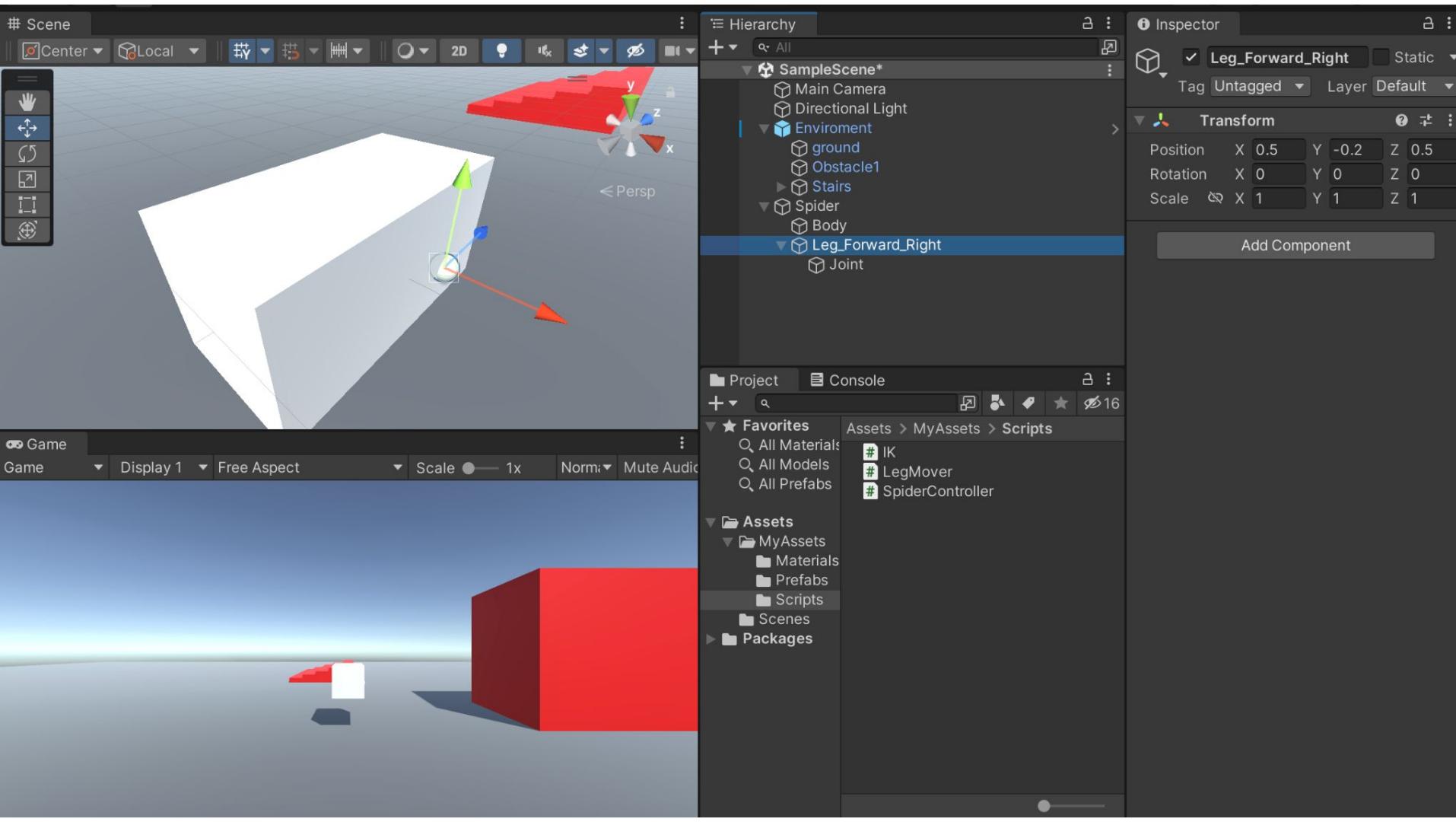
public class SpiderController : MonoBehaviour
{
    Rigidbody rb;
    [SerializeField] float speed = 100f, angSpeed = 50f;
    float mov_x, mov_y;

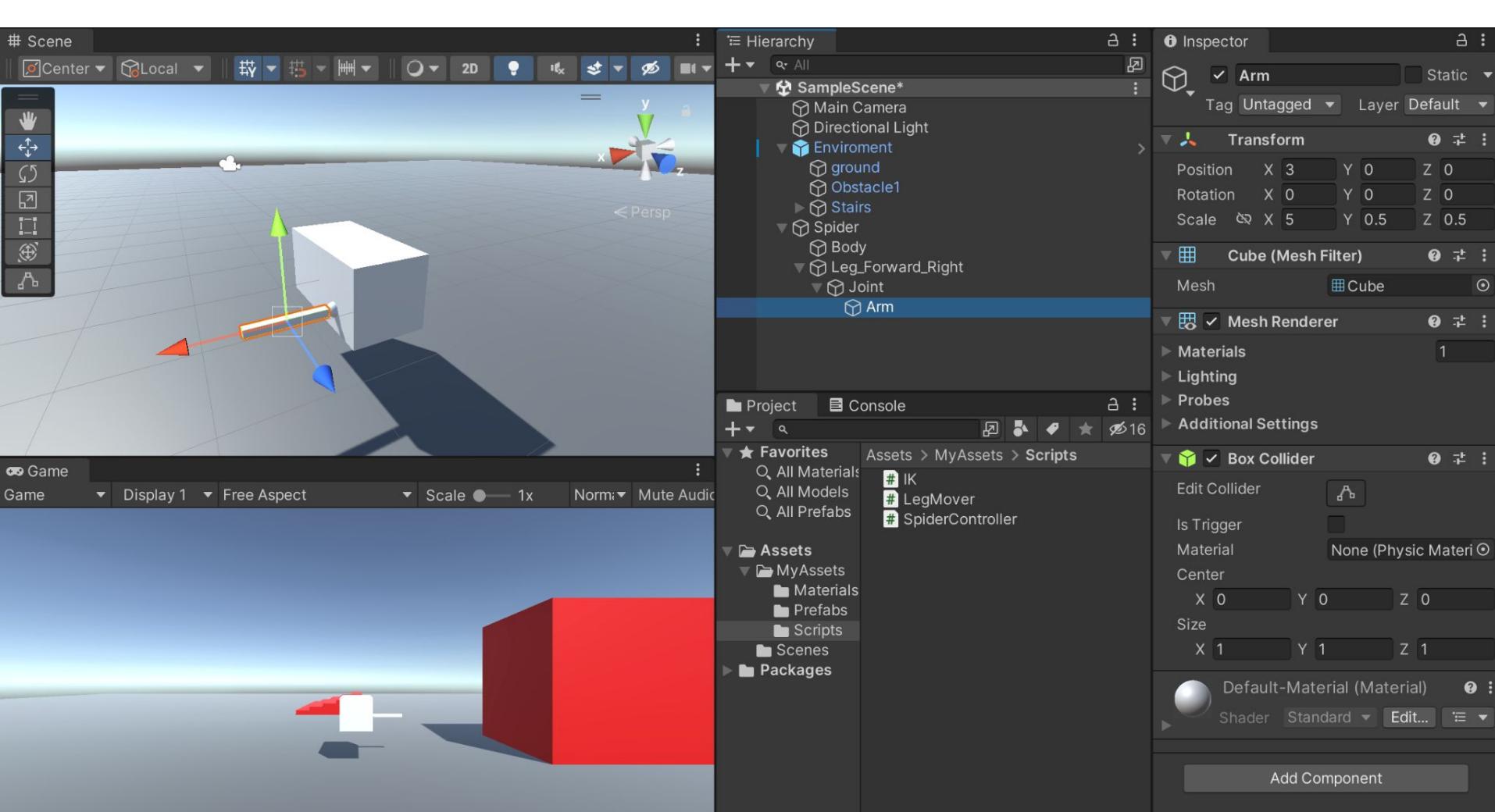
    void Start()
    {
        TryGetComponent(out rb);
    }

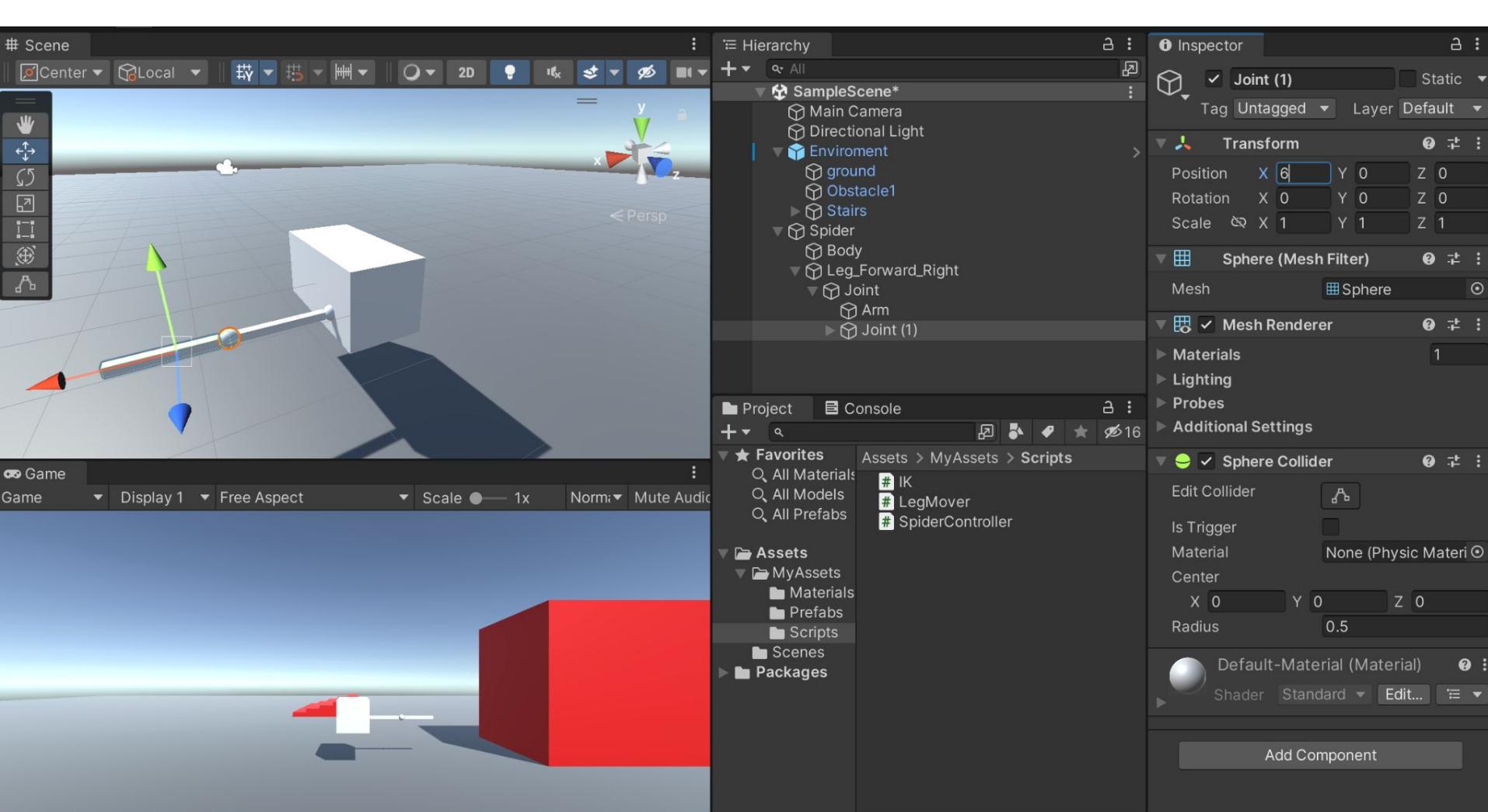
    void Update()
    {
        mov_y = Input.GetAxis("Vertical"); //直進に使う
        mov_x = Input.GetAxis("Horizontal"); //回転に使う
        //移動のベクトル
        Vector3 movement = transform.forward * mov_y * speed * Time.deltaTime;
        rb.velocity = movement;
        transform.Rotate(0, mov_x * angSpeed * Time.deltaTime, 0); //y軸中心の回転
    }
}
```



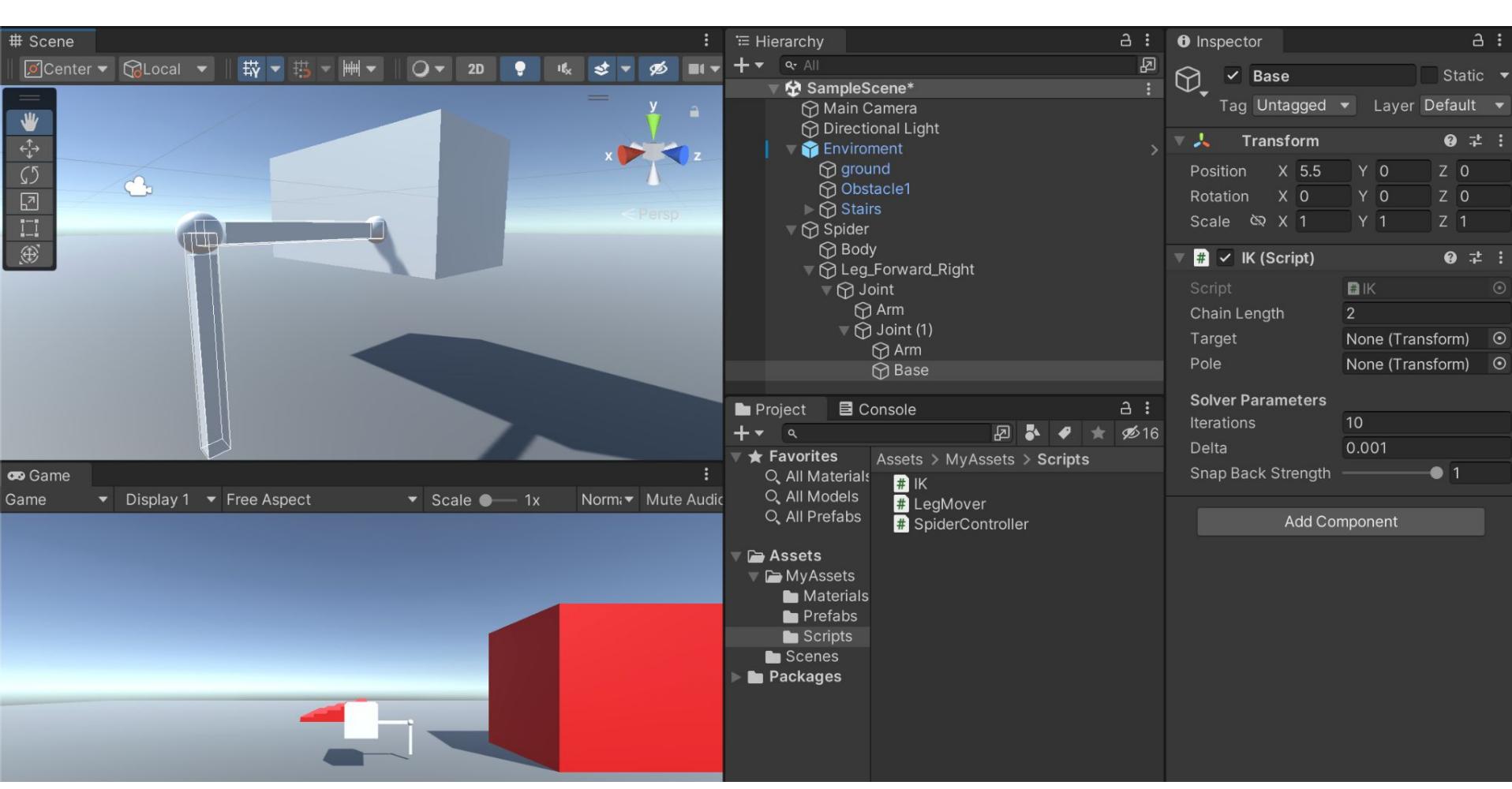
実行 確認

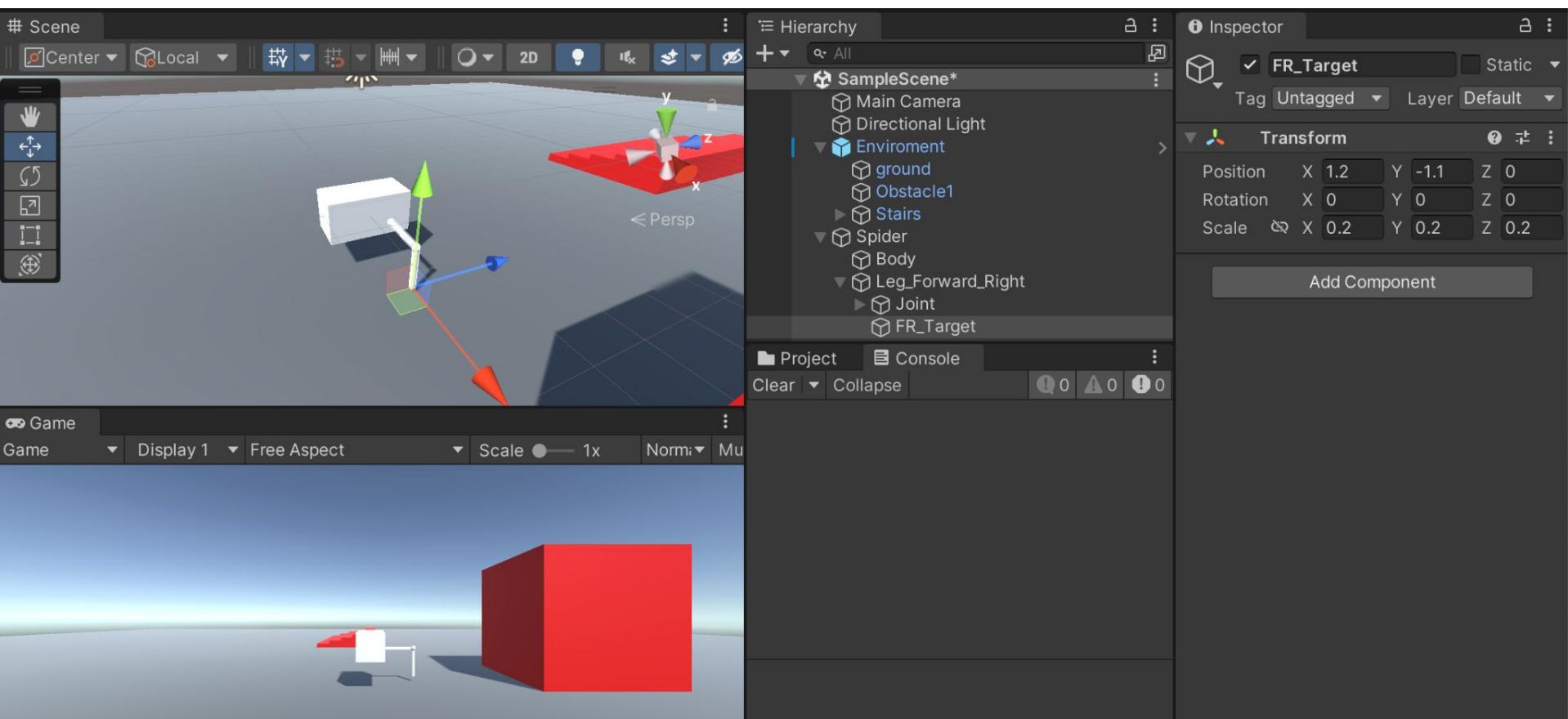




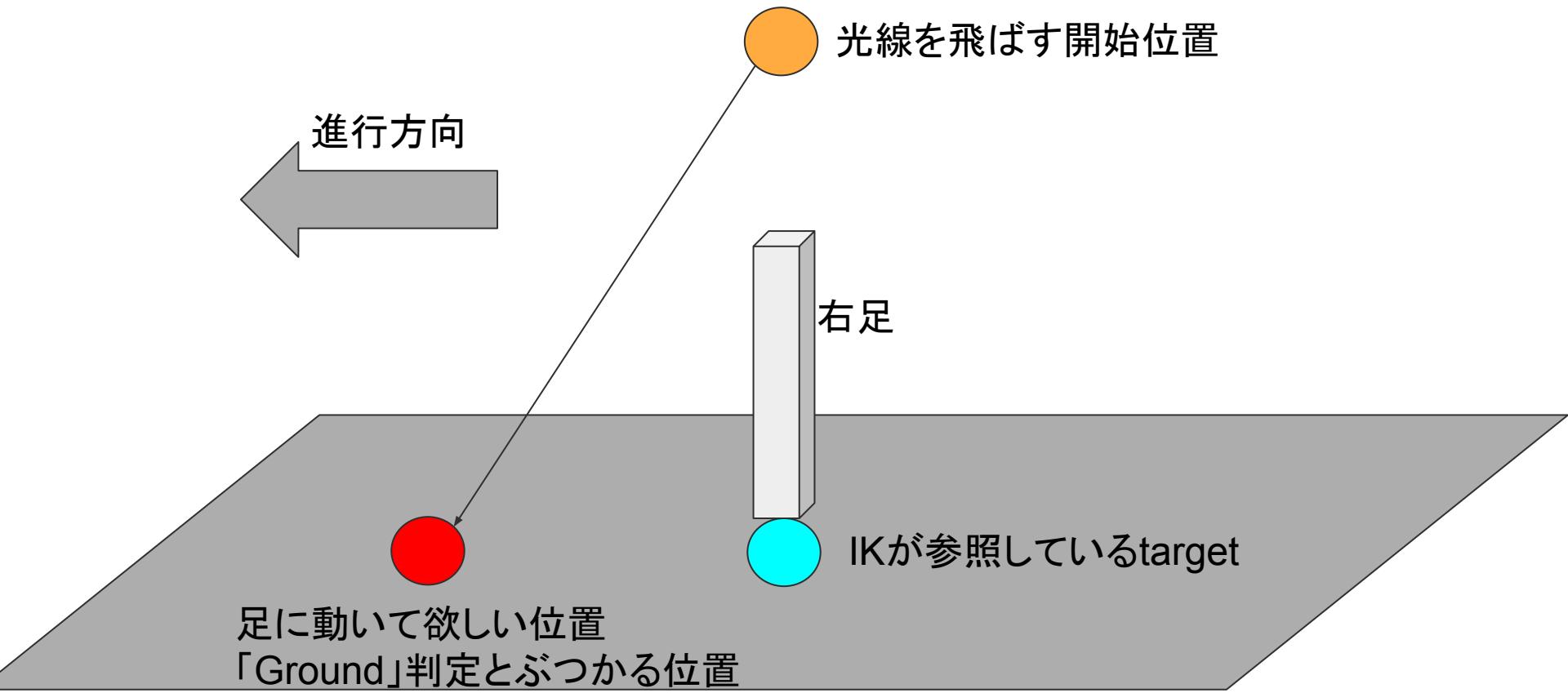


IKプログラム 設定



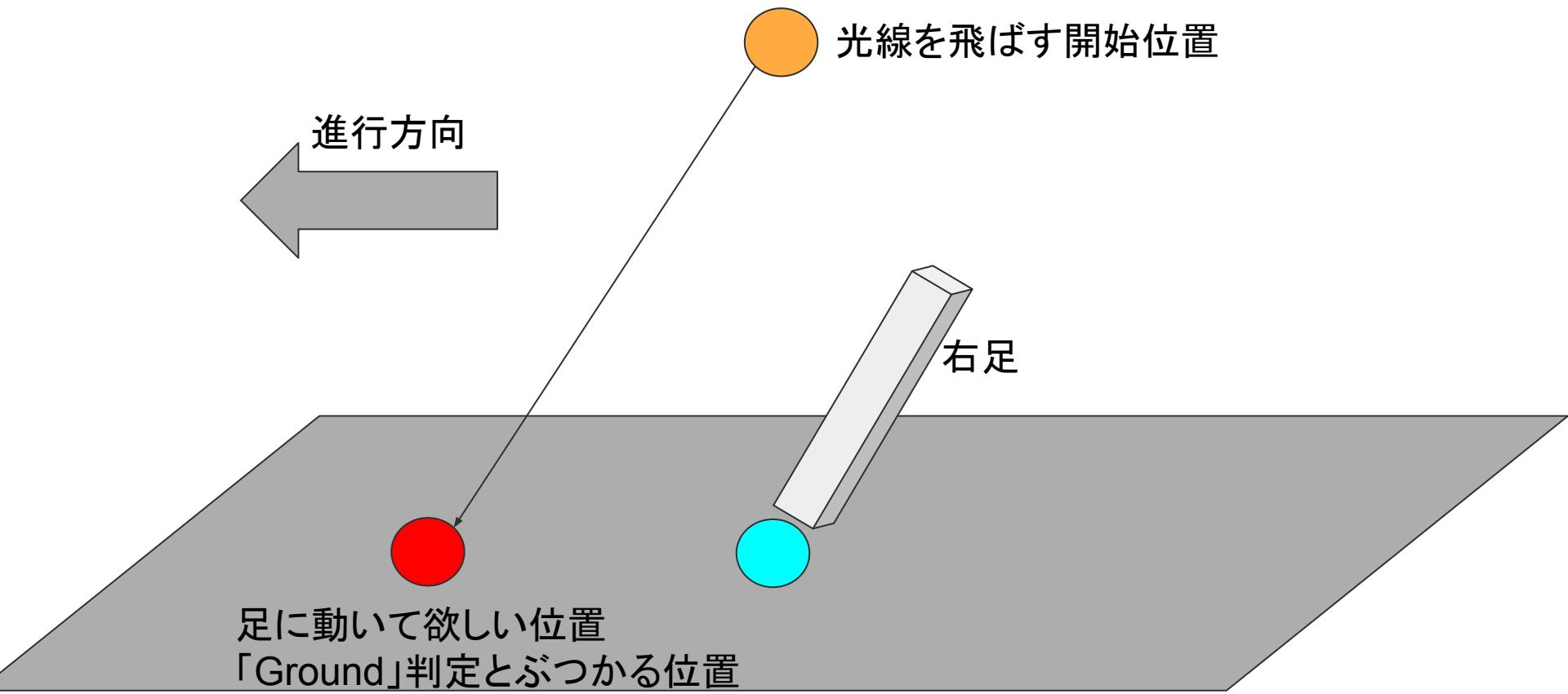


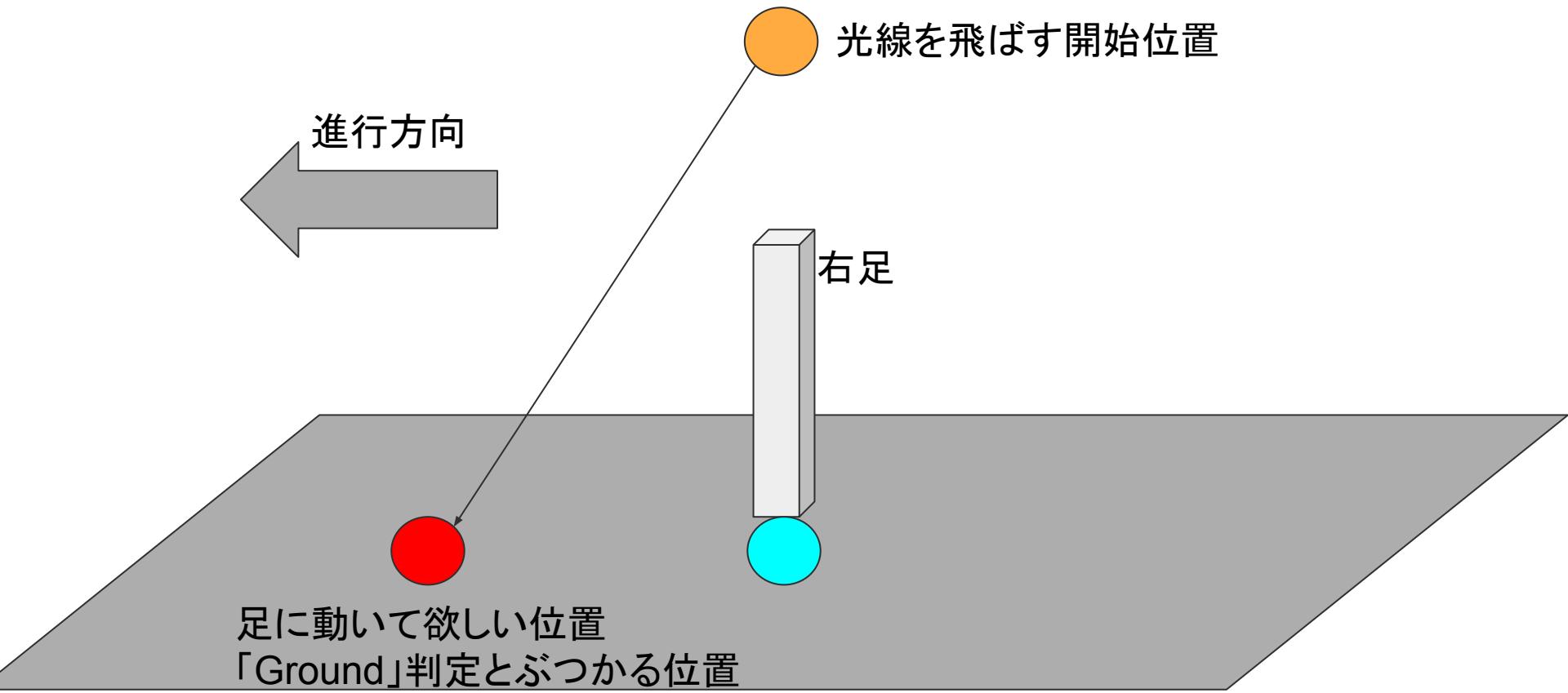
ターゲットに追従するか確認

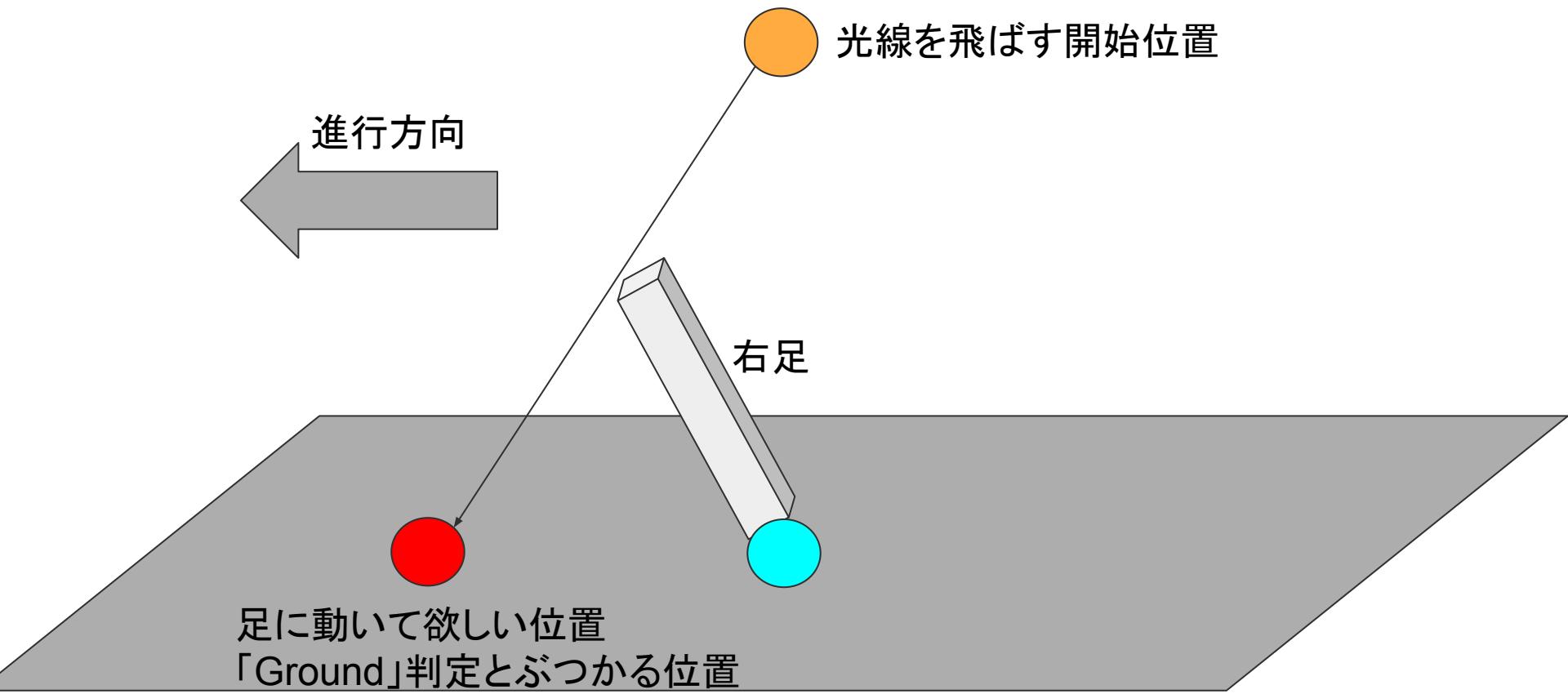


(IKのターゲットが動かす)胴体が動くと

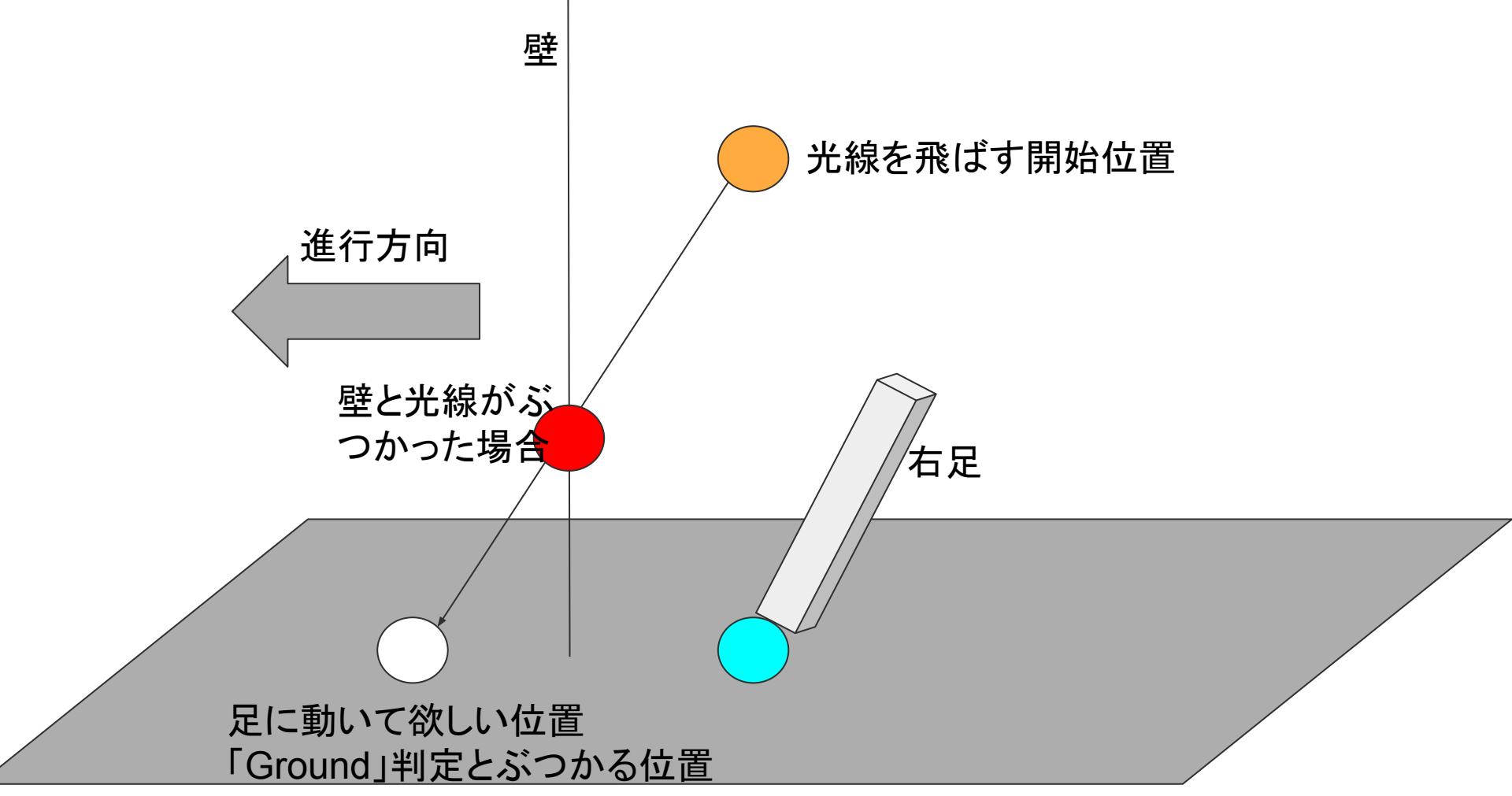
足の動き的には

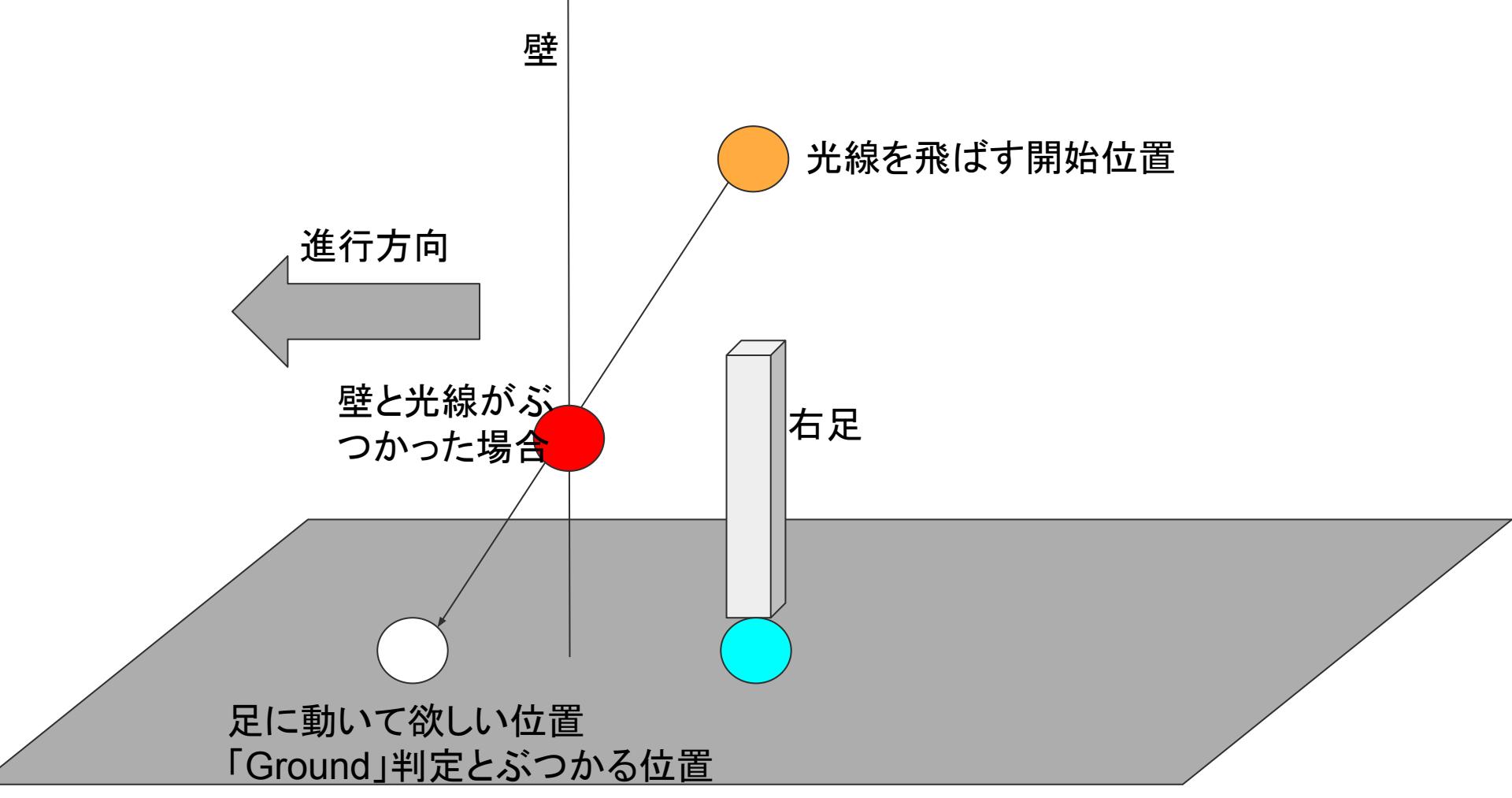


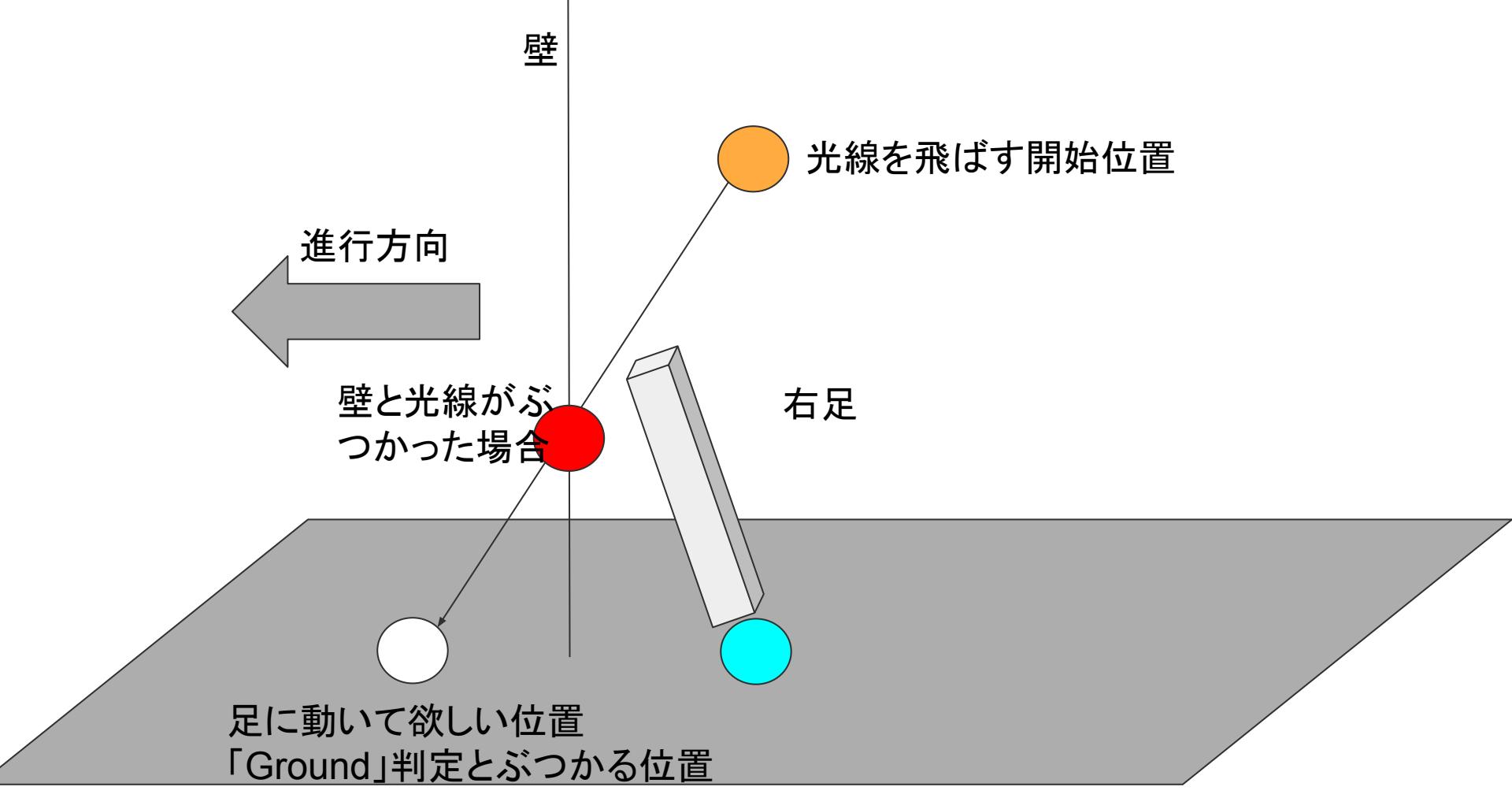




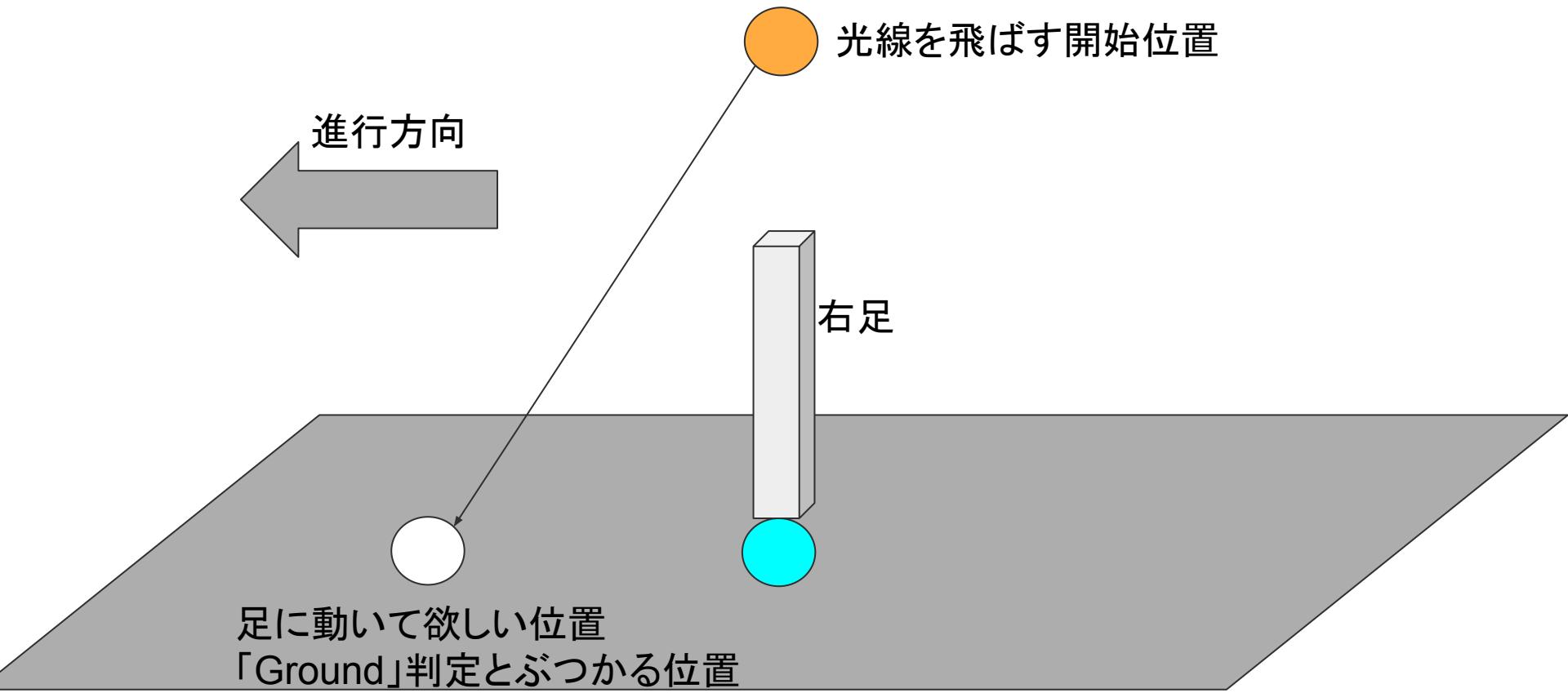
壁がある場合

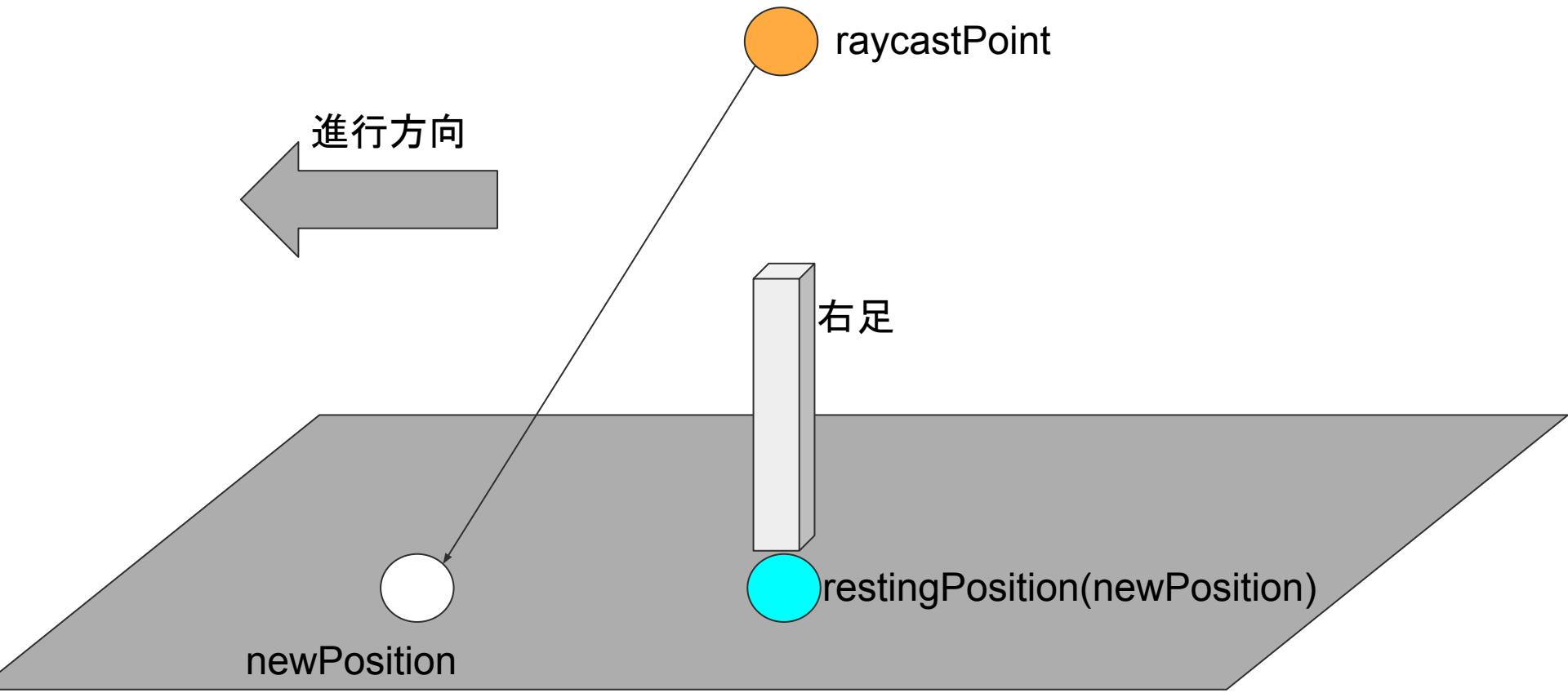


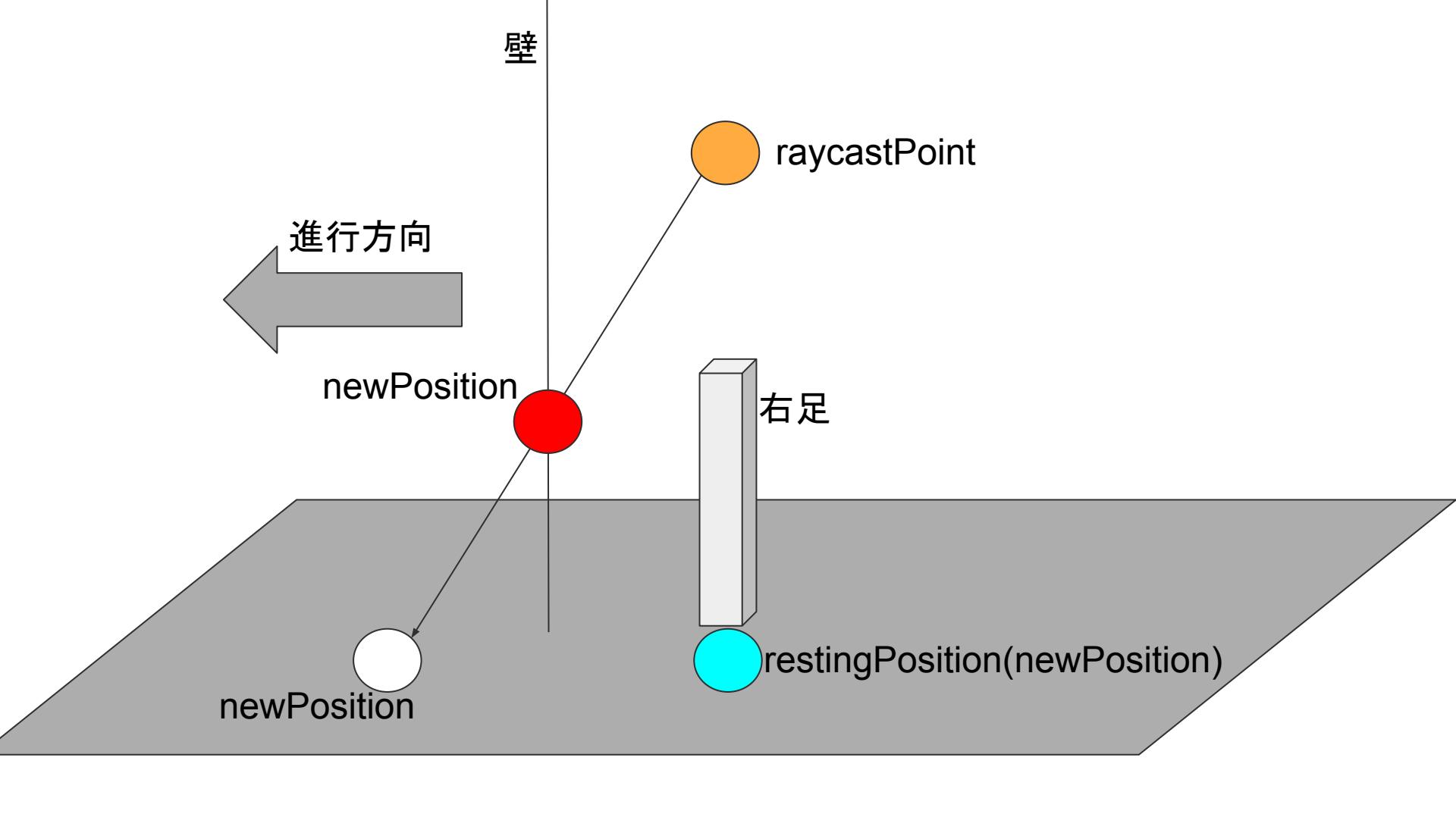




変数に置き直す





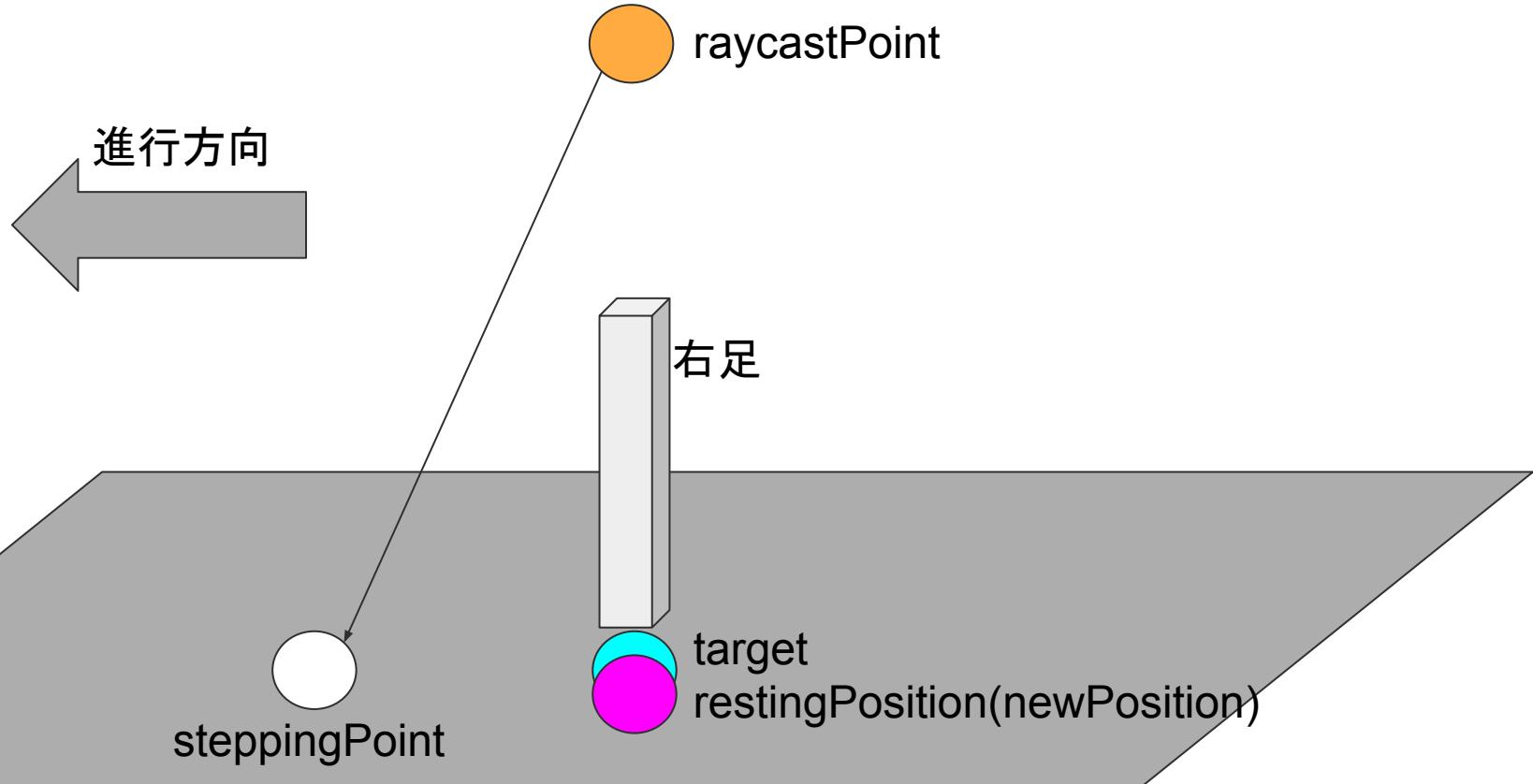


プログラム上で歩くとは？

場合分け

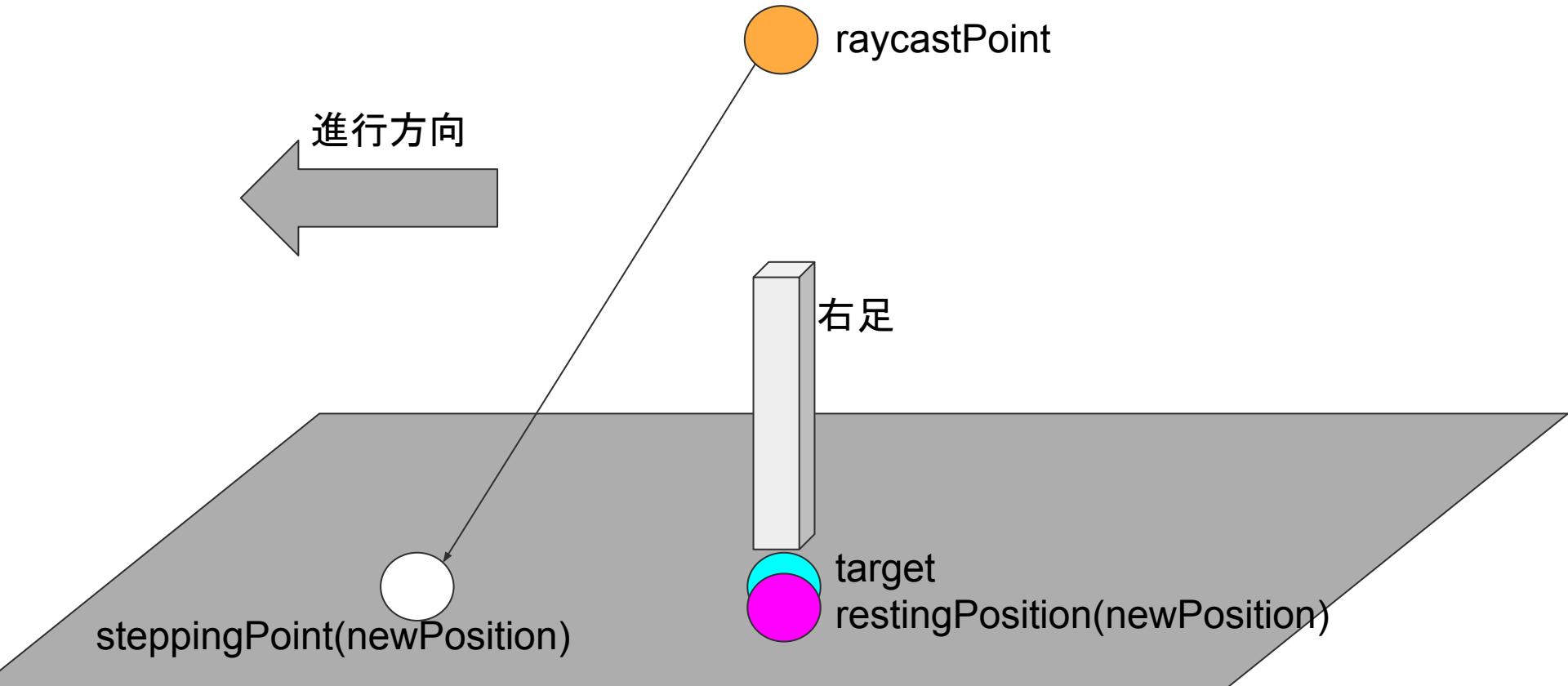
足をあげる前

「steppingPoint」と「restingPosition」の距離が離れていれば
実際に移動する処理を行う

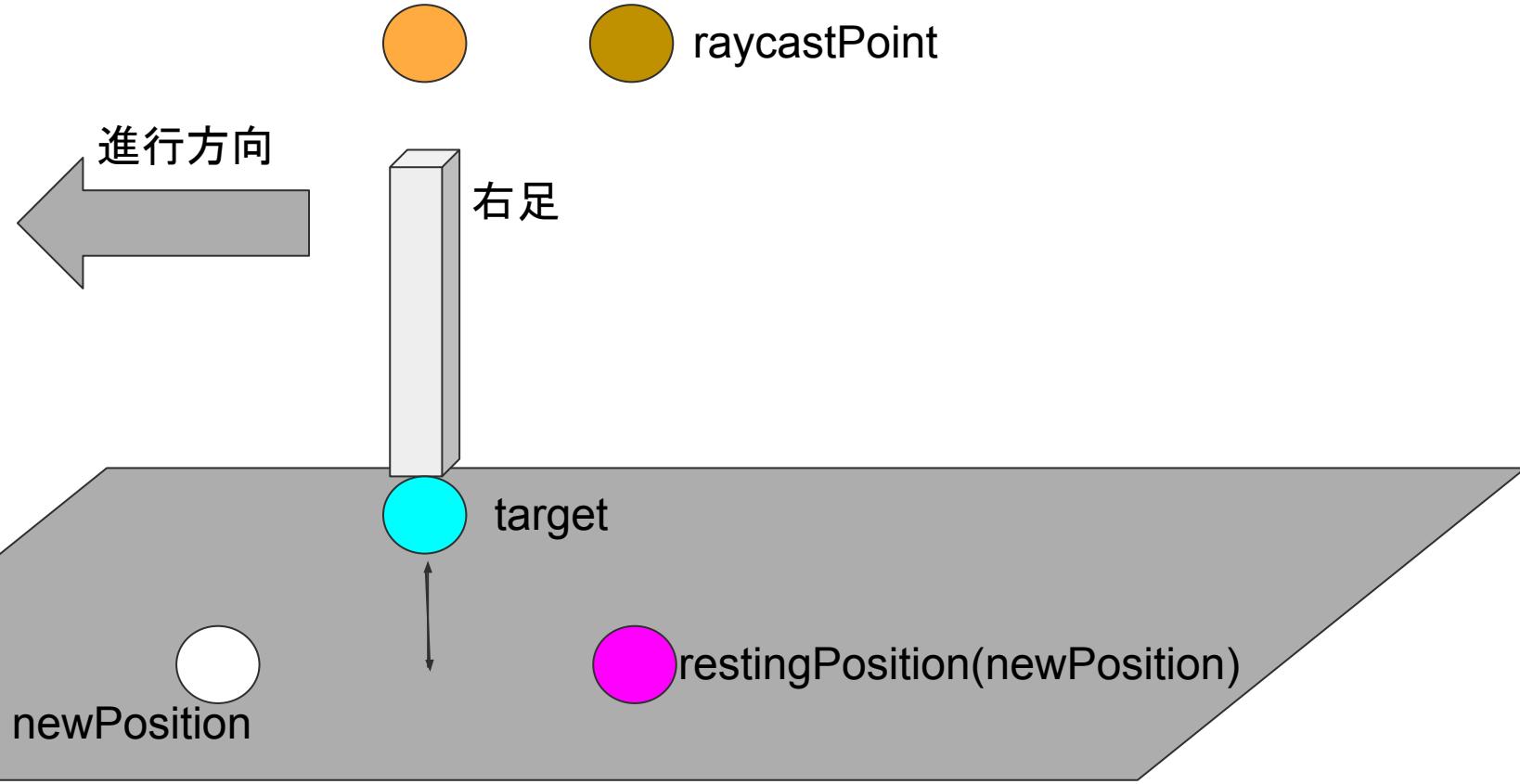


足をあげる前

「steppingPoint」と「restingPosition」の距離が離れていれば
実際に移動する処理を行う

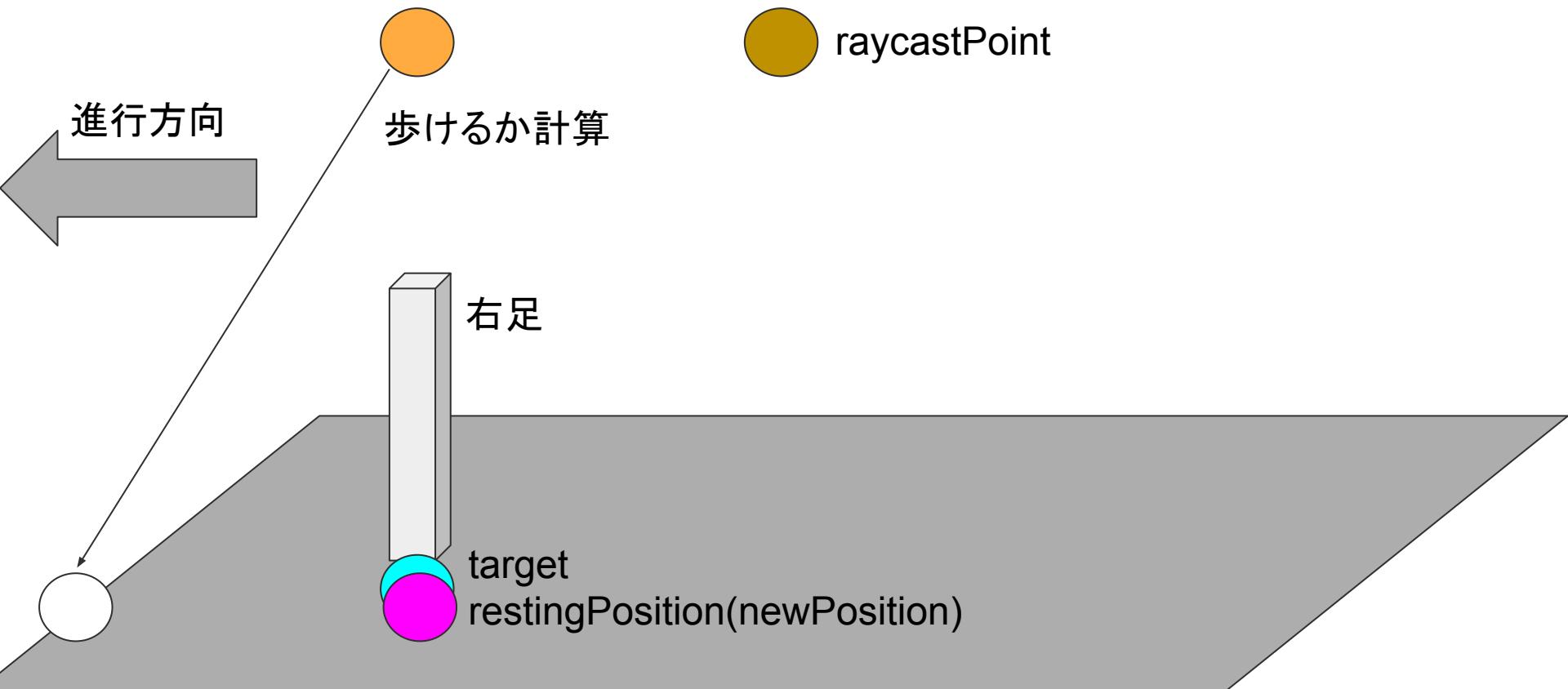


足をあげ終えた



足を降ろし終えた

「newPosition」と「restingPosition」の距離が離れていれば
実際に移動する処理を行う



```
public class LegMover : MonoBehaviour
{
    /* 変数 */
    [Header("レイを飛ばす始点")] [SerializeField] Transform raycastPoint;
    [Header("IKで設定する目標点")] [SerializeField] Transform target;
    [Header("歩くことができるかを探る点")] [SerializeField] Transform steppingPoint;

    Vector3 newPosition; //計算した結果を保存する変数( 次のフレームにいて欲しい足の位置 )

    [Header("衝突処理をするレイヤー")] [SerializeField] LayerMask mask;

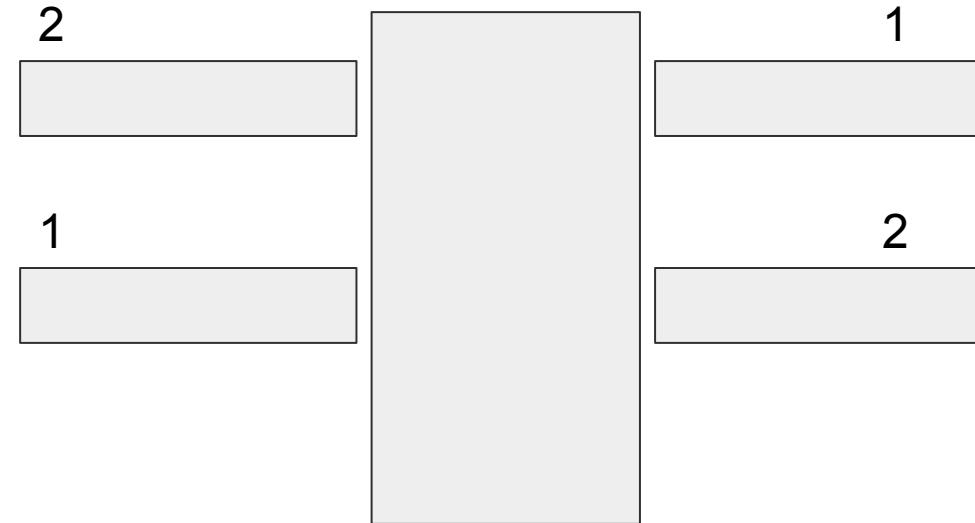
    Vector3 restingPosition; //蜘蛛の動きを作るが、ある程度動くまで動かさない足がある。その位置。//一休みポジ

    [Header("移動動作を始め出す距離")] [SerializeField] float moveDistance = 0.7f; //((一休みポジ と 移動予定位置 の距離)
    [Header("足のアニメーション速度")] [SerializeField] float speed = 10f; //足を上や下に動かす時の速さ (足のアニメーションの速さ)
    [Header("足のvalue")]
    [Tooltip("「『前右足・後ろ左足』セット」が動いている=1\n「『前左足・後ろ右足』セット」が動いている=2")]
    [SerializeField] int legSetValue = 1;

    int currentLegSetValue; //現在の動いて欲しい足セット

    bool legGrounded; //足が地面にあるかどうか
    bool moving; //移動中かどうか
    bool movingDown; //足を降ろしているところかどうか
    bool hasMoved; //移動が終わったかどうか
    /*
     */
}
```

前



後ろ

```
void Start()
{
    //一番初めはtargetと同じ位置に置く
    restingPosition = target.position;
}

void Update()
{
    // 足の動かしたいポジを大まかに計算 光線とGameObjectが当たった位置
    newPosition = CalculatePoint(steppingPoint.position); //動かせる座標 もしくは 一休みポジ

    if (Vector3.Distance(restingPosition, newPosition) > moveDistance){
        // 実際に足を動かす処理 (動かしたい距離と一休みする距離が一定の距離離れたら実行)
        Step(newPosition);
    }
}
```

```
//動く場所の計算
public Vector3 CalculatePoint(Vector3 stepPos) {
    Vector3 movDir = stepPos - raycastPoint.position;// 進みたい方向 = stepPos - レイ の位置
    RaycastHit hit;//光線を飛ばした時に当たった物(Colliderの付いたGameObject)の情報を入れる変数

    if (Physics.SphereCast(raycastPoint.position, 1f, movDir, out hit, 5f, mask)){
        stepPos = hit.point;//光線が衝突したポイントの座標
    }else{//何もぶつからなかった場合
        stepPos = restingPosition;//一休みポジ を返す
    }
    return stepPos; //位置を返す
}
```

```
//実際に移動 //足のアニメーションを実装する //targetの位置を変える
//newPosition 足をどこに動かすか計算したもの
void Step(Vector3 stepPos){

    //if (currentLegSetValue == legSetValue)
    //}

    //場合分けする（状態によってしたい処理を考える）（後々 「前右足・後ろ左足」 セットか前左足・後ろ右足」 セットかの判断も）
    /*①*/
    //足をあげようとしている状態(足を上げ終わるまで続ける)

    legGrounded = false; //地面に足はついていない判定に変える
    moving = true; //動いている判定に変える
    //movingDown = false; //足は降ろしていない (falseのまま)
    hasMoved = false; //動きが完了したわけでもない(falseに変える)

    // 「target.position(IKが追従する点)が元々いた場所」 から 「目的地(移動予定点からy方向1ぶん上など)」 まで
    // 「target.position(IKが追従する点)」 を「一定速度」で移動させてくれる関数
    target.position = Vector3.MoveTowards(target.position, stepPos + Vector3.up, speed * Time.deltaTime);
    //一休みポジを更新
    restingPosition = Vector3.MoveTowards(target.position, stepPos + Vector3.up, speed * Time.deltaTime);
```

実行

足をあげるというアニメーションが動作しそうだということがわかる

実行しながら調整...

```
void Update()
{
    // 足の動かしたいポジを大まかに計算 光線とGameObjectが当たった位置
    newPosition = CalculatePoint(steppingPoint.position); // 動かせる座標 もしくは 一休みポジ

    // アニメーション起動 かつ 地面に足がついている
    // もしくは アニメーション中
    if (Vector3.Distance(restingPosition, newPosition) > moveDistance && legGrounded || moving ){
        // 実際に足を動かす処理 (動かしたい距離と一休みする距離が一定の距離離れたら実行)
        Step(newPosition);
    }
    // IKの位置を一休み位置に
    UpdateIK();
}
```

```
//実際に移動 //足のアニメーションを実装する //targetの位置を変える
//newPosition 足をどこに動かすか計算したもの
void Step(Vector3 stepPos){

    //if (currentLegSetValue == legSetValue)
    //}

    //場合分けする（状態によってしたい処理を考える）（後々 「前右足・後ろ左足」 セットか前左足・後ろ右足」 セットかの判断も）
    /*①*/
    //足をあげようとしている状態(足を上げ終わるまで続ける)

    legGrounded = false; //地面に足はついていない判定に変える
    moving = true; //動いている判定に変える
    //movingDown = false; //足は降ろしていない (falseのまま)
    hasMoved = false; //動きが完了したわけでもない(falseに変える)

    // 「target.position(IKが追従する点)が元々いた場所」 から 「目的地(移動予定点からy方向1ぶん上など)」 まで
    // 「target.position(IKが追従する点)」 を「一定速度」で移動させてくれる関数
    target.position = Vector3.MoveTowards(target.position, stepPos + Vector3.up, speed * Time.deltaTime);
    //一休みポジを更新
    restingPosition = Vector3.MoveTowards(target.position, stepPos + Vector3.up, speed * Time.deltaTime);
```

```
/*②*/
//足をおろそうとしている状態 (足を上げ終わり、足を降ろすまで)
// 足をstepPos + Vector3.up まで持ってきたら(足を上げ終わったら)
if (target.position == stepPos + Vector3.up)
{
    movingDown = true; //足を振り下ろす動作に移行
}
if (movingDown == true)
{
    target.position = Vector3.MoveTowards(target.position, stepPos, speed * Time.deltaTime);
    restingPosition = Vector3.MoveTowards(target.position, stepPos, speed * Time.deltaTime);
}
```

Hierarchy

SampleScene*

- Main Camera
- Directional Light
- Enviroment
 - ground
 - Obstacle1
 - Stairs
- Spider
 - Body
- Leg_Forward_Right
 - FR_Joint1
 - FR_Arm1
 - FR_Joint2
 - FR_Arm2
 - FR_Base
 - FR_Target
 - FR_SteppingPoint
 - FR_RaycastPoint

Inspector

Transform

Position X 0.5 Y -0.2 Z 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Leg Mover (Script)

Script # LegMover

レイを飛ばす始点 Raycast Point FR_RaycastPoint (Transform)

IKで設定する目標点 Target FR_Target (Transform)

歩くことができるかを探る点 Stepping Point FR_SteppingPoint (Transform)

衝突処理をするレイヤー Mask Ground

移動動作を始め出す距離 Move Distance 1

足のアニメーション速度 Speed 10

足のvalue Leg Set Value 0

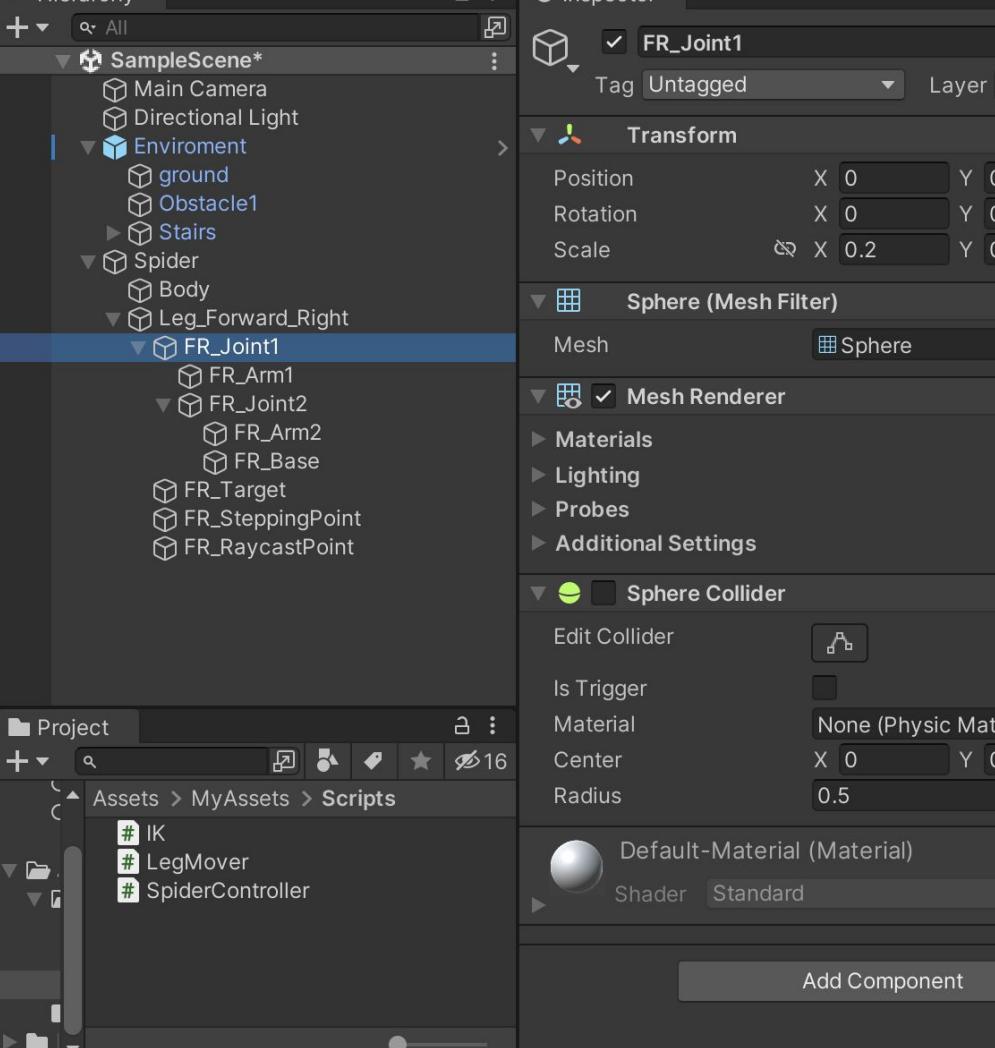
セットの足 Otherleg None (Leg Mover)
Offset 0

Project

Assets > MyAssets > Scripts

- # IK
- # LegMover
- # SpiderController

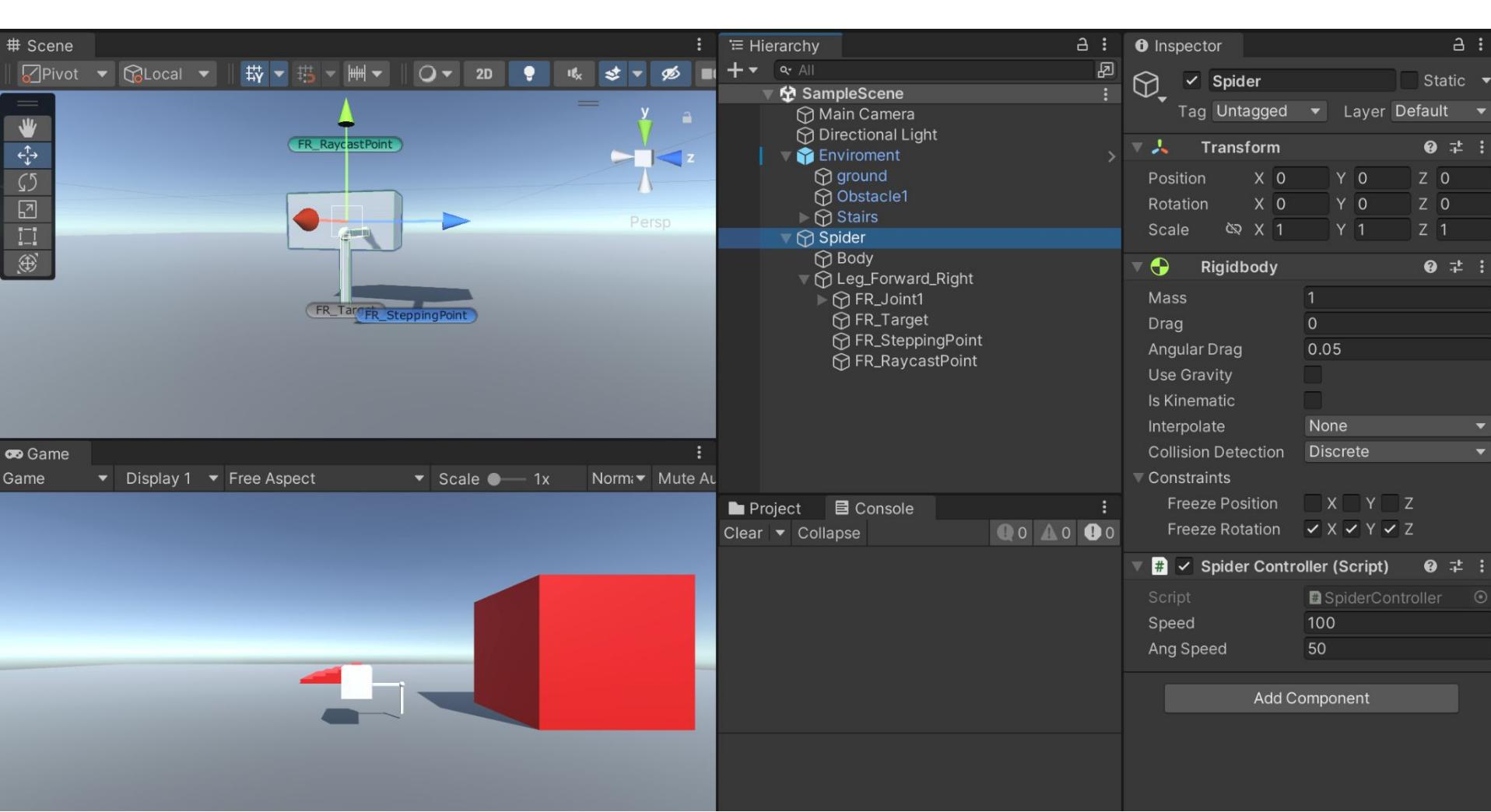
足に関するColliderをオフ



前足後ろ足

合計 4本にする

交互に動かす



```
public class LegMover : MonoBehaviour
{
    /* 変数 */
    [Header("レイを飛ばす始点")] [SerializeField] Transform raycastPoint;
    [Header("IKで設定する目標点")] [SerializeField] Transform target;
    [Header("歩くことができるかを探る点")] [SerializeField] Transform steppingPoint;

    Vector3 newPosition; //計算した結果を保存する変数( 次のフレームにいて欲しい足の位置 )

    [Header("衝突処理をするレイヤー")] [SerializeField] LayerMask mask;

    Vector3 restingPosition; //蜘蛛の動きを作るが、ある程度動くまで動かさない足がある。その位置。//一休みポジ

    [Header("移動動作を始め出す距離")] [SerializeField] float moveDistance = 0.7f; //((一休みポジ と 移動予定位置 の距離)
    [Header("足のアニメーション速度")] [SerializeField] float speed = 10f; //足を上や下に動かす時の速さ (足のアニメーションの速さ)
    [Header("足のvalue")]
    [Tooltip(" 「『前右足・後ろ左足』セット」が動いている=1\n 「『前左足・後ろ右足』セット」が動いている=2")]
        [SerializeField] int legSetValue = 1;

    int currentLegSetValue; //現在の動いて欲しい足セット
    [Header("セットの足")] [SerializeField] LegMover otherleg;

    bool legGrounded; //足が地面にあるかどうか
    bool moving; //移動中かどうか
    bool movingDown; //足を降ろしているところかどうか
    bool hasMoved; //移動が終わったかどうか
    /*
        */
}
```

Hierarchy Inspector

SampleScene*

- Main Camera
- Directional Light
- Enviroment
 - ground
 - Obstacle1
 - Stairs
 - Spider
 - Body
 - Leg_Foward_Right
 - Leg_Foward_Left
 - Leg_Back_Right
 - Leg_Back_Left

Transform

Position X 0.5 Y -0.2 Z 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Leg Mover (Script)

Script # LegMover

レイを飛ばす始点 Raycast Point FR_RaycastPoint (Transform)

IKで設定する目標点 Target FR_Target (Transform)

歩くことができるかを探る点 Stepping Point FR_SteppingPoint (Transform)

衝突処理をするレイヤー Mask Ground

移動動作を始め出す距離 Move Distance 1

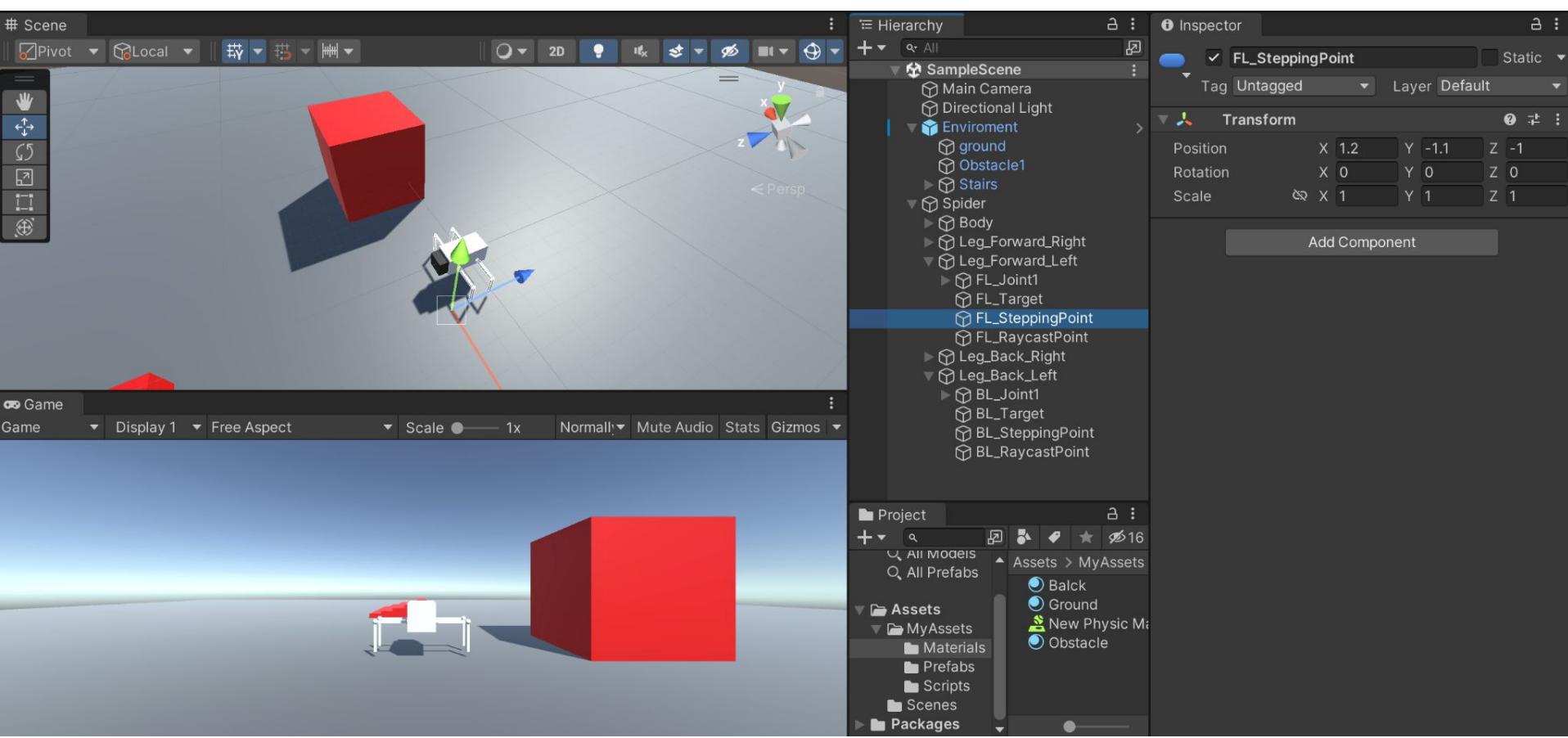
足のアニメーション速度 Speed 10

足のvalue Leg Set Value 1

セットの足 Otherleg Leg_Back_Left (Leg Mover)

Project Console

Clear Collapse Q 0 A 0 ! 0



```
[Header("レイを飛ばす始点")] [SerializeField] Transform raycastPoint;
[Header("IKで設定する目標点")] [SerializeField] Transform target;
[Header("歩くことができるかを探る点")] [SerializeField] Transform steppingPoint;

Vector3 newPosition; //計算した結果を保存する変数( 次のフレームにおいて欲しい足の位置)

[Header("衝突処理をするレイヤー")] [SerializeField] LayerMask mask;

Vector3 restingPosition; //蜘蛛の動きを作るが、ある程度動くまで動かさない足がある

[Header("移動動作を始め出す距離")] [SerializeField] float moveDistance = 0.7f;
[Header("足のアニメーション速度")] [SerializeField] float speed = 5f; //足を上げる速さ
[Header("足のvalue")]
[Tooltip("「『前右足・後ろ左足』セット」が動いている=1\n「『前左足・後ろ右足』セット」が動いている=2")]
[SerializeField] int legSetValue = 1;

static int currentLegSetValue = 1; //現在の動いて欲しい足セット
[Header("セットの足")] [SerializeField] LegMover otherleg;

bool legGrounded = true; //足が地面にあるかどうか
bool moving = false; //移動中かどうか(アニメーション中かどうか)
bool movingDown = false; //足を降ろしているところかどうか
bool hasMoved = true; //移動が終わったかどうか
/* */
```

```
void Step(Vector3 stepPos)
{
    if (currentLegSetValue == legSetValue) {
        //場合分けする(状態によってしたい処理を考える)(後々 「前右足・後ろ左足」セットか前左足・後ろ右足」+ 
        /*①*/
        //足をあげようとしている状態(足を上げ終わるまで続ける)

        legGrounded = false; //地面に足はついていない判定に変える
        moving = true; //動いている判定に変える
        //movingDown = false; //足は降ろしていない(falseのまま)
        hasMoved = false; //動きが完了したわけでもない(falseに変える)

        //「target.position(IKが追従する点)が元々いた場所」から「目的地(移動予定点からy方向1ぶん上など)」ま
        //「target.position(IKが追従する点)」を「一定速度」で移動させてくれる関数
        target.position = Vector3.MoveTowards(target.position, stepPos + Vector3.up, speed * Ti
        //一休みポジを更新
        restingPosition = Vector3.MoveTowards(target.position, stepPos + Vector3.up, speed * Ti

        /*②*/
        //足をおろそうとしている状態(足を上げ終わり、足を降ろすまで)
        //足をstepPos + Vector3.upまで持ってきたら(足を上げ終わったら)
        if (target.position == stepPos + Vector3.up)
        {
            movingDown = true; //足を振り下ろす動作に移行
        }
        if (movingDown == true)
        {
            //足を下ろす動作
        }
    }
}
```

```
/*③*/
//一休みしている状態
//target.positionがひと休みポジと一緒に
if (target.position == stepPos)
{
    legGrounded = true; //地面に足がついている
    moving = false; //アニメーションしなくて良い
    movingDown = false; //
    hasMoved = true; //

    //「動かしたい足セット」と「実際に動かしている足セット」が同じ かつ 他の足が動き終わっているなら
    if (currentLegSetValue == legSetValue)
    {
        currentLegSetValue = currentLegSetValue * -1 + 3; //「動かしたい足セット」を変える
    }
}
```

// 足の動かしたいポジを大まかに計算

//動かせる座標 もしくは 一休みポジ

// 実際に足を動かす処理

//場合分けする(状態によってしたい処理を考える)

(後々 「前右足・後ろ左足」セットか

「前左足・後ろ右足」セットかの判断も)

//足をあげている状態

//足をおろそうとしている状態

//一休みしている状態