キャッチゲーム　りんご Unity版

1. 素材の準備

スクラッチを起動　https://scratch.mit.edu/projects/349513276/

スプライトで、Appleを追加
「コスチューム」タブを選択

[ビットマップに変換]ボタンを押す

appleを右クリック

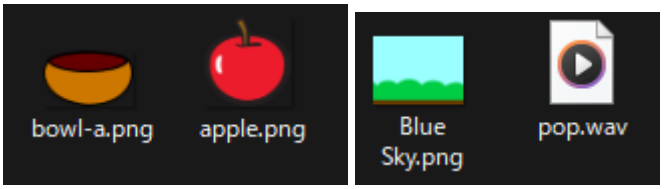
「書き出し」を選択

スプライトの bowl、背景の blue sky もビットマップに変換して、ファイルへ書き出す

bowlを選択
「音」タブを選択
popを右クリック

「書き出し」を選択

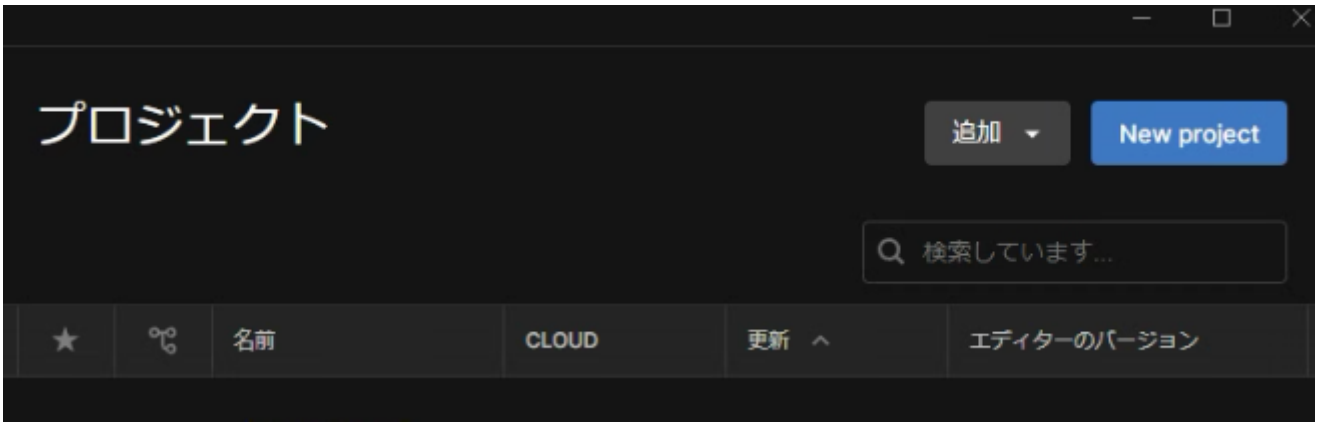
4つのファイルが作成できた



※Scratchの素材をお借りしてUnityで利用します。本来の利用目的と違うので、自分の学習のためだけに使うようにしましょう。

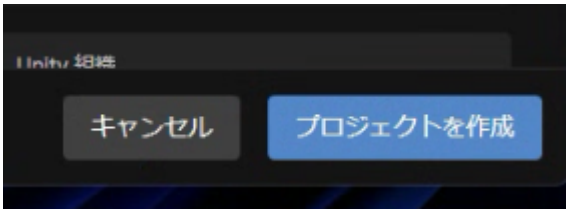
2. プロジェクトの作成

Unity Hubを起動
[New Project]ボタンを押す

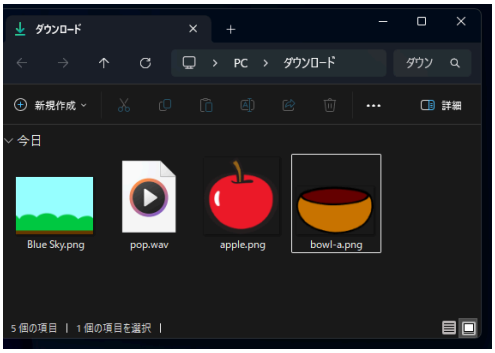


Universal 2D を選択
プロジェクト名に「キャッチりんご」と入力 （名前はなんでもOK）

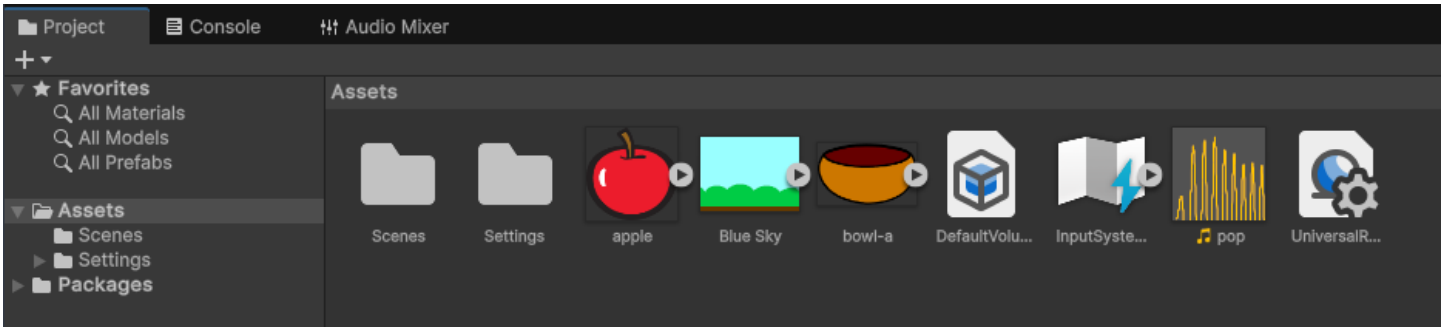
[プロジェクトを作成]ボタンを押す
しばらくすると、作成したプロジェクトが開く



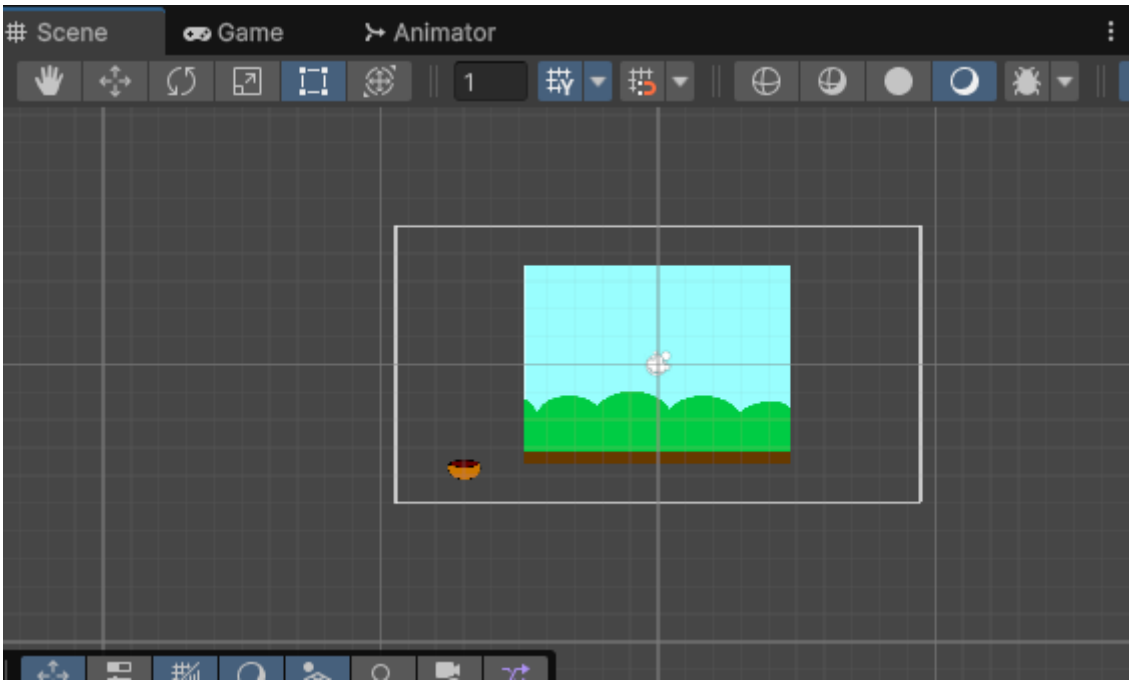
書き出したファイルをダウンロードフォルダーからProject/Assets へドラック＆ドロップして追加



ファイルエクスプローラーのファイルを Unityへ
ドラック＆ドロップ

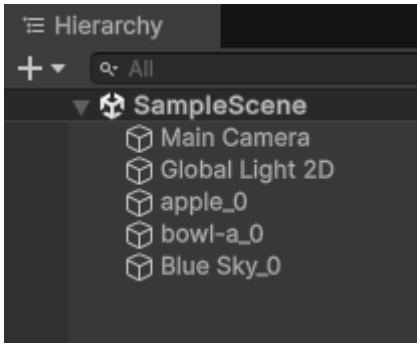


追加した画像と音を Sceneにドラック&ドロップ



Project/Assetsから
Sceneへ
ドラック&ドロップ

Sceneに置くと、Hierarchyに aplle_0, bowl-a_0, Blue Sky_0 が作成される

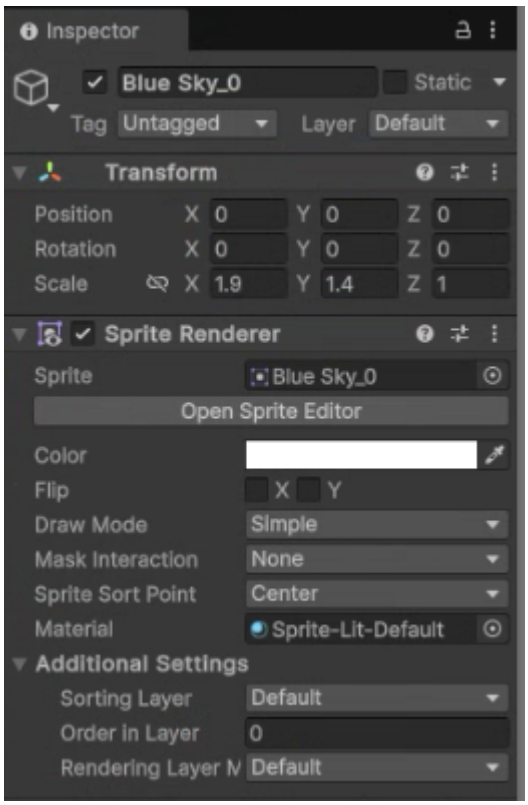


apple_0
bowl-a_0
Blue Sky_0

3. 背景とボールをつくる

3. 1 Blue Sky 背景

Hierarchyの Blue Sky_0を選択
Blue Sky_0のInspectorを変更

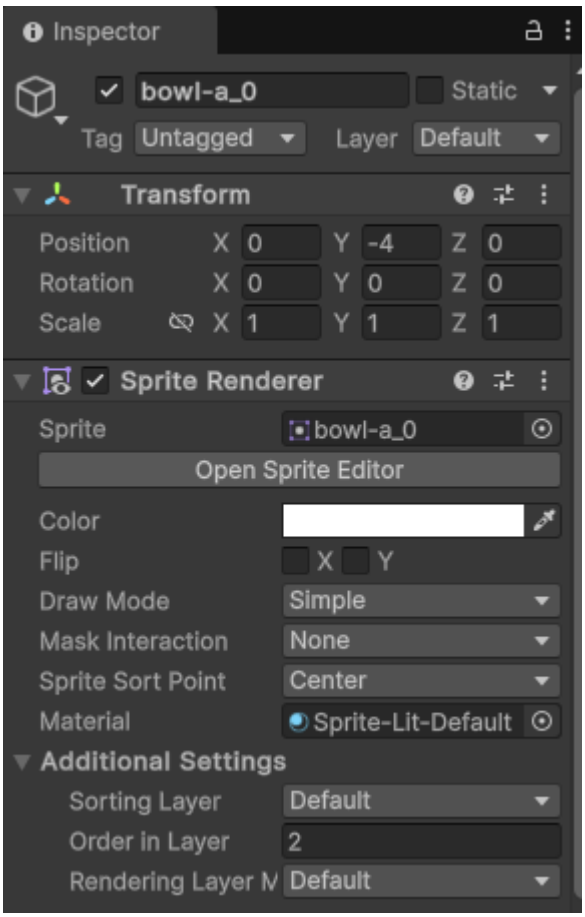


Position X:0 Y:0 Z:0

Scale X:1.9 Y:1.4
※Blue Skyが小さいので、Scaleで拡大

3．2 ボール

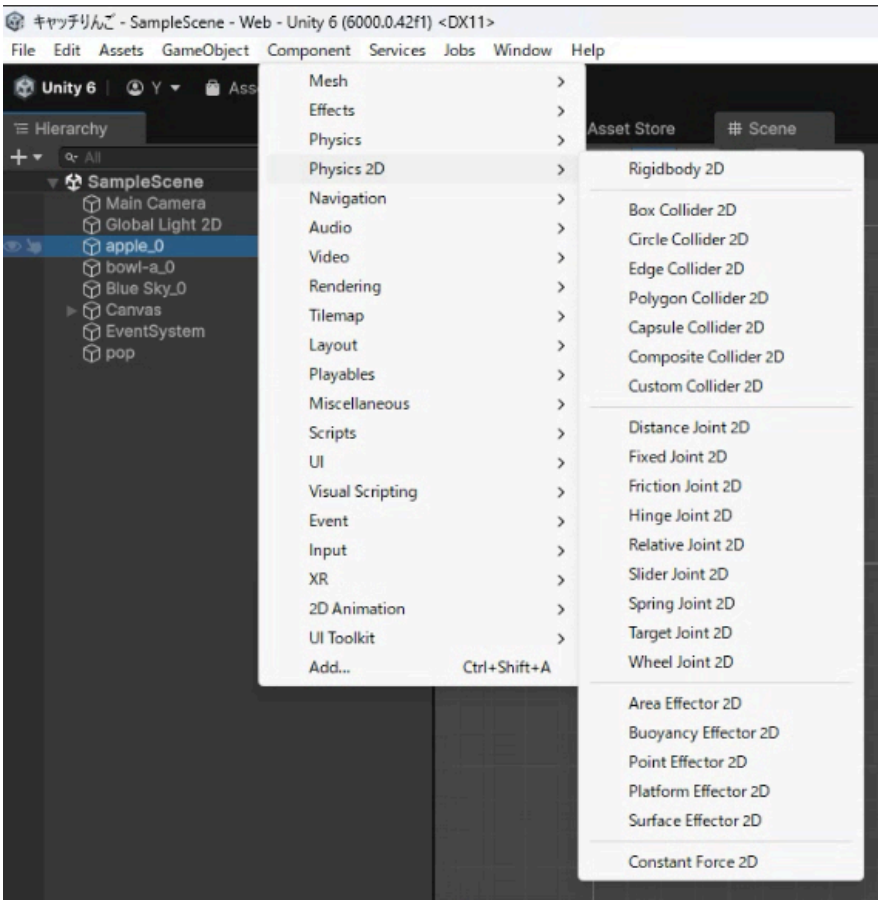
Hierarchyの bowl-a_0を選択
bowl-a_0のInspectorを変更



Position X:0 Y:-4 Z:0

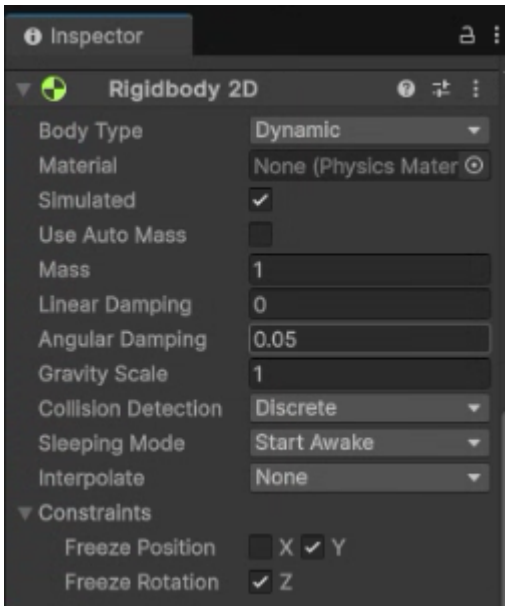
※背景とりんごの前にくるようOrder in Layerを2にする
Additional Settings/Order in Layer: 2

メニューにて Component/Physics 2D/**Rigidbody 2D** を選択して追加　　(重力など物理演算を使う)
メニューにて Component/Physics 2D/**Box Collider 2D** を選択して追加　(しょうとつを検知するときに使う)



Physics 2D
　Rigidbody 2D
　Box Collider 2D

ボールが落ちないように Rigidbody 2Dの Yを固定、回転しないように Zを固定

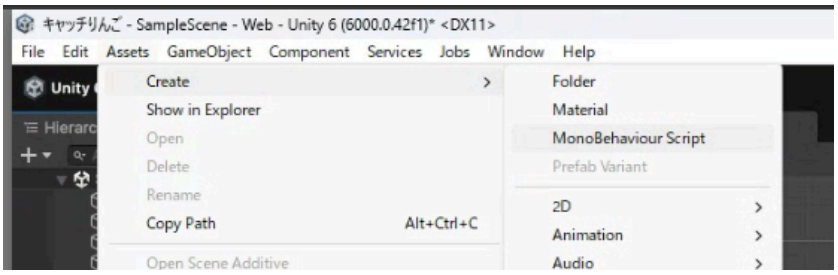


Constraints/Freeze Position ☑Y
Constraints/Freeze Rotation ☑ Z

3．3 ボールをキーボードで動かす

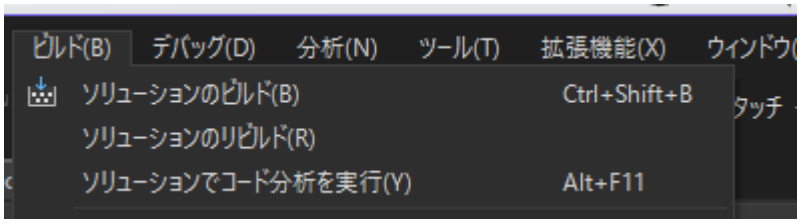
3．3．1 スクリプト

メニューの Assets/Create/**MonoBehavior Script**を選択
Projectに、新しいスクリプトができ、名前が変更できる状態にあるので、「BowlController」と変更
※ 一度名前が決まると、後から変更してもプログラム側は、変わらない



ProjectのBowlControllerをダブルクリックして開く
プログラム ***BowlController*** を入力　(コメント // 以降は、説明用の文字で、入力しなくてもいいです。)

エディターのメニュー「ビルド」「ソリューションのビルド」を選択

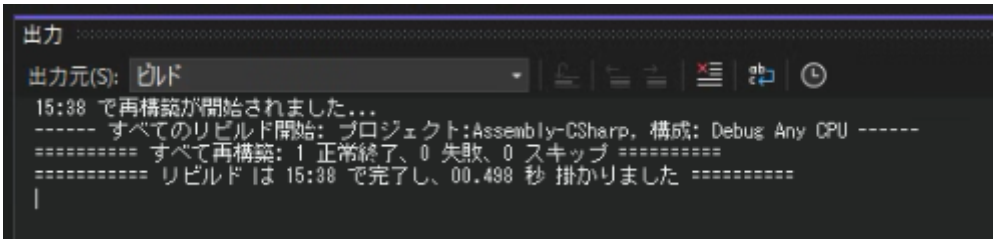


ビルド

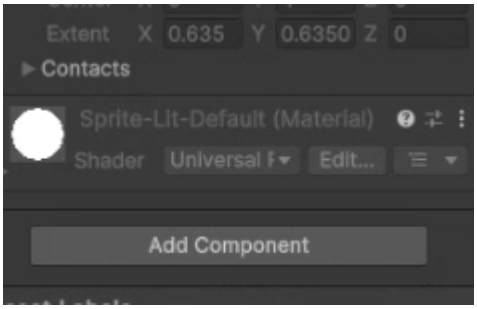
ソリューションのビルド

※void Start()は、最初に1回だけ呼ばれる。
※void Update() は、フレーム毎に呼ばれる。
※Time.deltaTime は、フレームが異なっても同じスピードを出せるように調整してくれる。

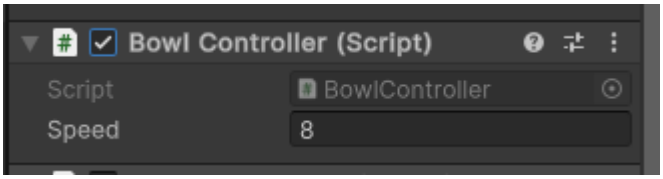
エラーが無くなるまで、プログラムを修正
エラーが無くなったら



Hierarchyの bowl-a_0を選択
Inspectorを一番下まで表示
Project のBowlController をInspectorの[Add Componet]へめがけて、ドラック&ドロップして追加

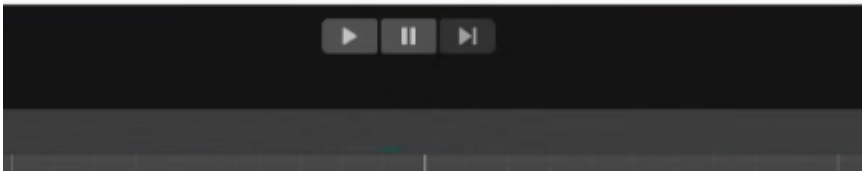


publicの付いている変数 speed は、Inspectorで、変更できる。
実行中も変更できます。ただし、停止すると値は、戻ってしまうので、注意が必要。



3．４ 動かしてみる

▶️ボタンを押して、キーボードの左矢印と右矢印でボールが動くか確認



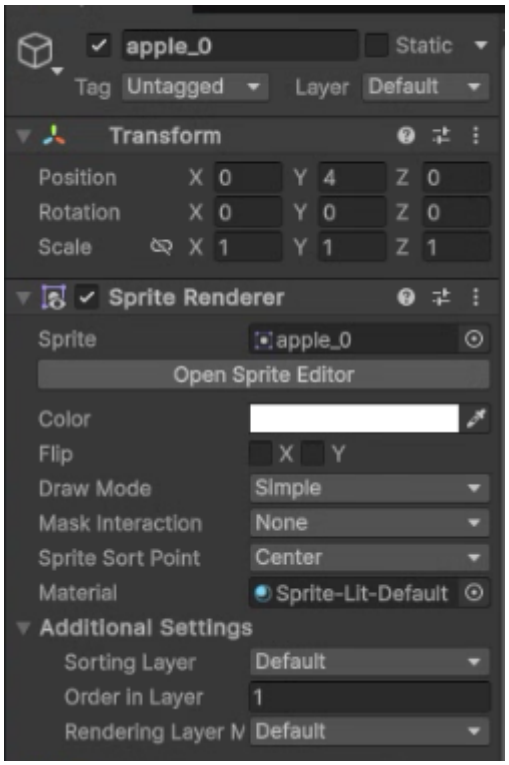
BowlController

```
using UnityEngine;

public class BowlController : MonoBehaviour
{
    public float speed = 8f; // 移動速度

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow)) // 左矢印キーが押されたとき
        {
            transform.position -= speed * transform.right * Time.deltaTime; // 左に移動
        }
        else if (Input.GetKey(KeyCode.RightArrow)) // 右矢印キーが押されたとき
        {
            transform.position += speed * transform.right * Time.deltaTime; // 右に移動
        }
    }
}
```

4. りんごをつくる

4. 1りんご

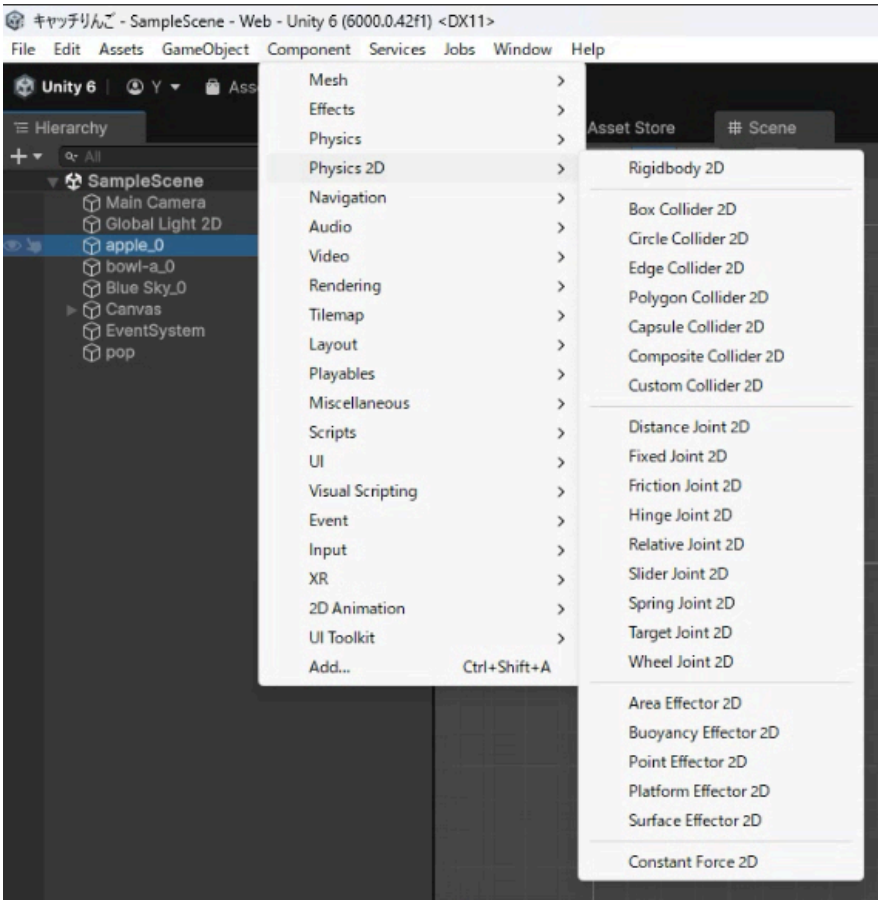
Hierarchyの apple_0を選択



Position X:0 Y:4 Z:0

Additional Settings/Order in Layer: 1
※背景の前にくるようOrder in Layerを1にする

メニューにて Component/Physics 2D/**Rigidbody 2D** を選択して追加　　(重力など物理演算を使う)
メニューにて Component/Physics 2D/**Circle Collider 2D** を選択して追加 (しょうとつを検知するときに使う)

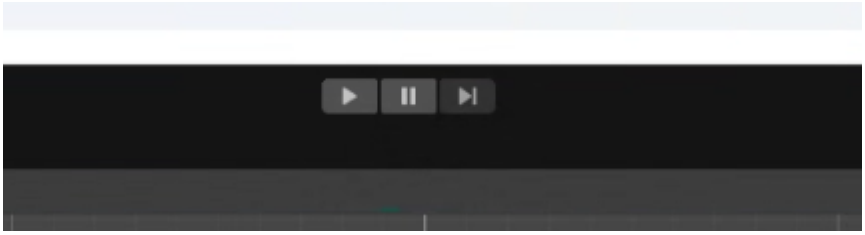


Physics 2D
　Rigidbody 2D

　Circle Collider 2D

4．2 動かしてみる

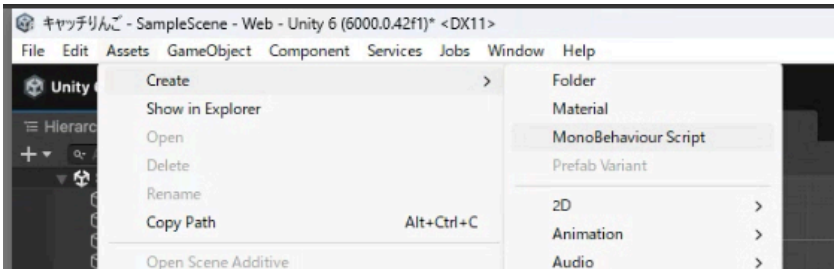
▶️ボタンボタンを押して、りんごが落ちるか確認



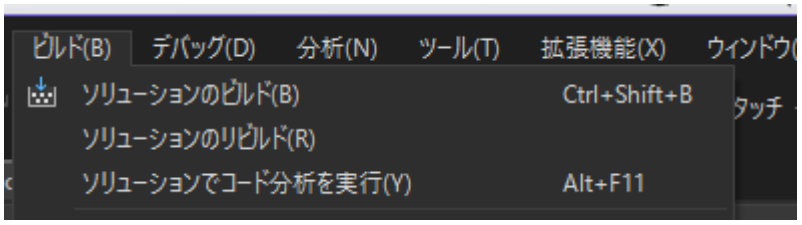
4．3 りんごがボールに当たったら

4．3．1 スクリプト

メニューの Assets/Create/**MonoBehavior Script**を選択
Projectに、新しいスクリプトができ、名前が変更できる状態にあるので、「AppleController」と変更
※ 一度名前が決まると、後から変更してもプログラム側は、変わらない



ProjectのAppleControllerをダブルクリックして開く
プログラム ***AppleController*** を入力 (コメント // 以降は、説明用の文字で、入力しなくてもいいです。)
エディターのメニュー「ビルド」「ソリューションのビルド」を選択

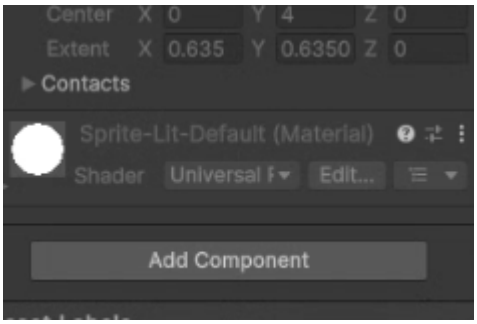


ビルド

ソリューションのビルド

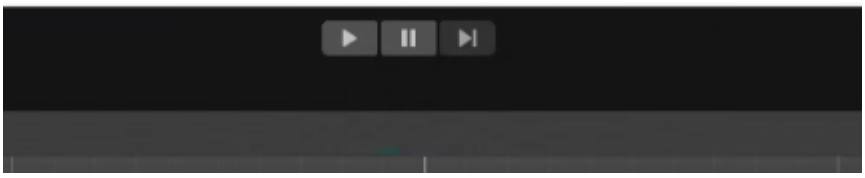
エラーが無くなるまで、プログラムを修正
※ワーニングは、今のところ無視しても良いです。
エラーが無くなったら

Hierarchyの apple_0を選択
Inspectorを一番下まで表示
Project のAppleController をInspectorの[Add Componet]へめがけて、ドラック＆ドロップして追加



4. 3 動かしてみる

▶️ボタンボタンを押して、ボールに当たったらりんごが消えるか確認



AppleController

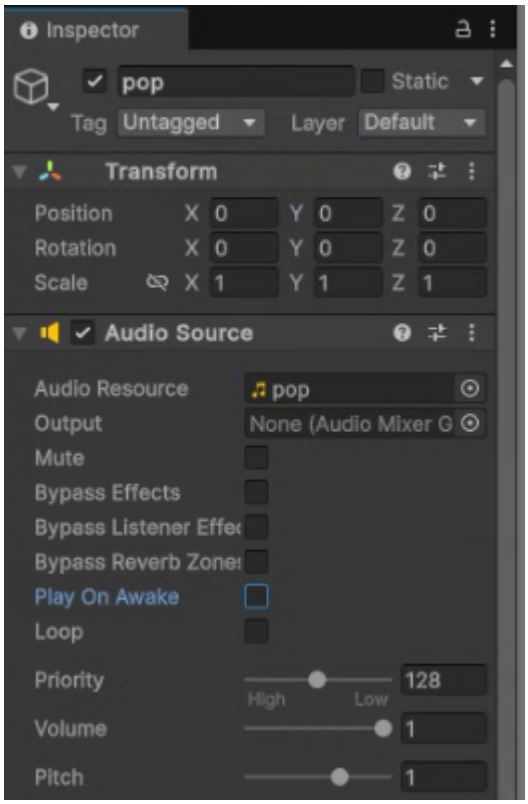
```
using UnityEngine;

public class AppleController : MonoBehaviour
{
    public int maxScore = 10; //ライフの初期値
    public UnityEngine.UI.Text scoreText; // スコア表示用のテキスト
    public GameObject gameOver; // ゲームオーバー表示用のオブジェクト
    public AudioSource sePop; // SE用のAudioSource

    private int score; // 得点

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        score = 0; // スコアの初期化
        Next(); // りんごの初期位置を設定
    }

    // Update is called once per frame
    void Update()
    {
       // Debug.Log("y: " + transform.position.y);
        if (transform.position.y < -5) // りんごが画面外に出たとき
        {
                Next(); // りんごの位置を初期化
        }
    }

    // りんごが他のオブジェクトと衝突したときに呼ばれるメソッド
    void OnCollisionEnter2D(Collision2D collision)
    {
        // sePop.Play(); // POPを再生

        score += 1; // スコアを加算
        // scoreText.text = "とくてん " + score.ToString(); // スコアを表示

        if (score == maxScore) // スコアが最大値を超えた場合
        {
          // gameOver.SetActive(true); // ゲームオーバー表示
           Destroy(gameObject); // りんごを削除
        }
        else
        {
           Next(); // りんごの位置を初期化
        }
    }

    // りんごの位置を初期化するメソッド
    private void Next()
    {
        gameObject.transform.position = new Vector3(Random.Range(-6, 6), 4, 0); // 位置の初期化
```

```
    }
}
```

5. 音をならす

5. 1 ポップ音

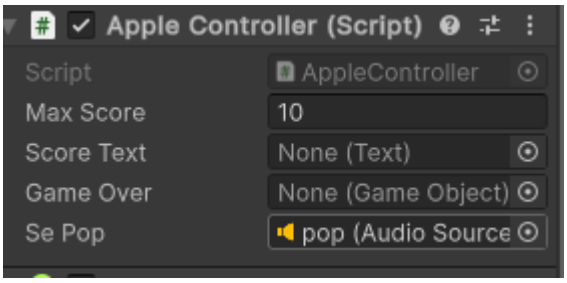
Hierarchyのpopを選択
popのInspectorを変更



Play On Awake □　チェックを外す
※ 開始時に、音が鳴るのを防ぐ

5. 2 プログラムから音をならす

Hierarchyの apple_0を選択
apple_0のInspectorを変更
AppleControllerに値を入れる



Se Pop: Hierarchyのpopをドラック＆ドロップ

※ ここで指定した値やオブジェクトが、プログラムで利用できる

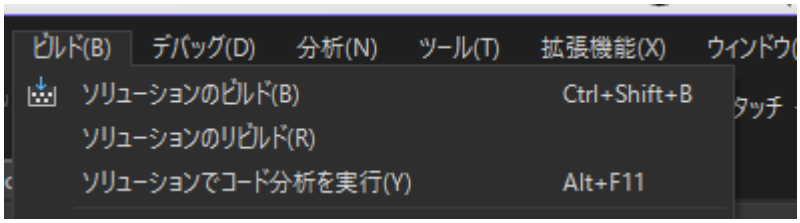
AppleController スクリプトの sePop.Play()のコメント // をはずす。命令文が無い時は追加。

前
```
//sePop.Play(); // POPを再生
```

後

```
sePop.Play(); // POPを再生
```

エディターのメニュー「ビルド」「ソリューションのビルド」を選択



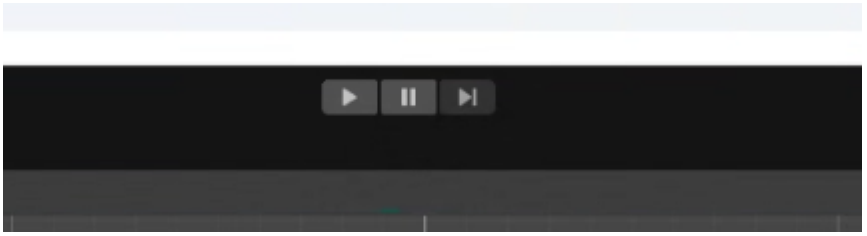
ビルド

ソリューションのビルド

エラーが無くなるまで、プログラムを修正

5．3 動かしてみる

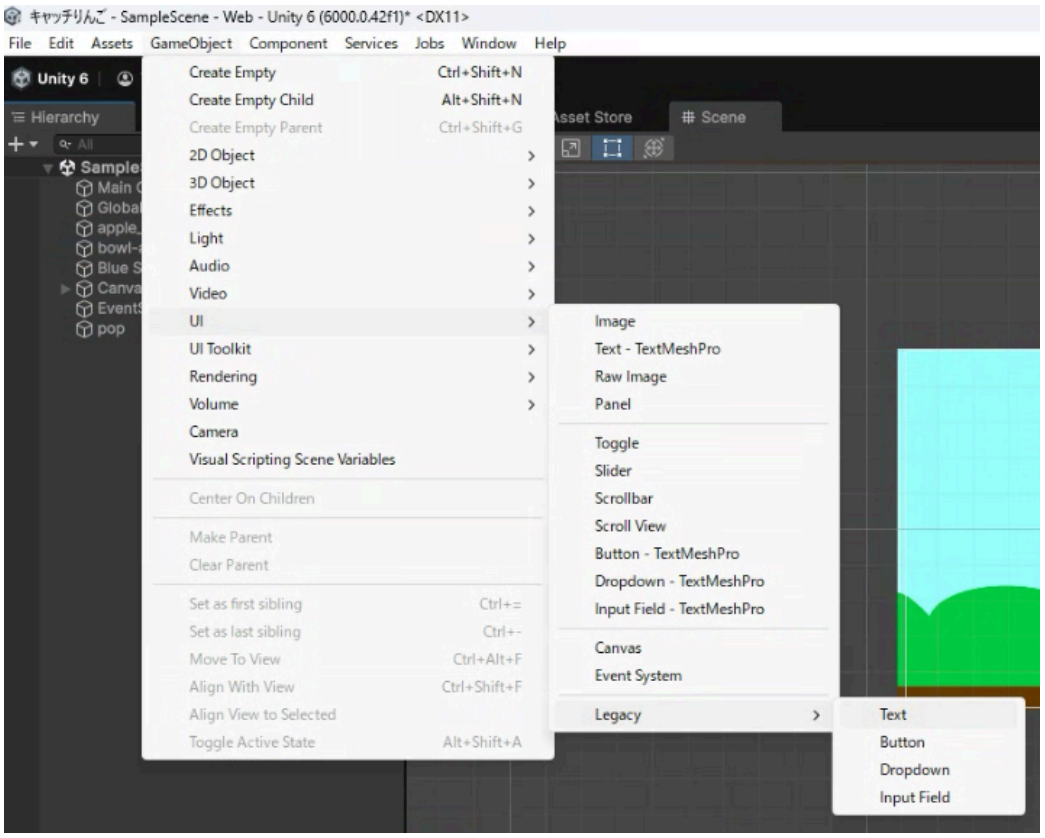
▶️ボタンを押して、動作するか確認



6. とくてんとゲームオーバーの文字をつくる

6. 1 とくてんとゲームオーバーをつくる

メニューのComponet/UI/Legacy/**Text** を選択し、CanvasとTextを追加

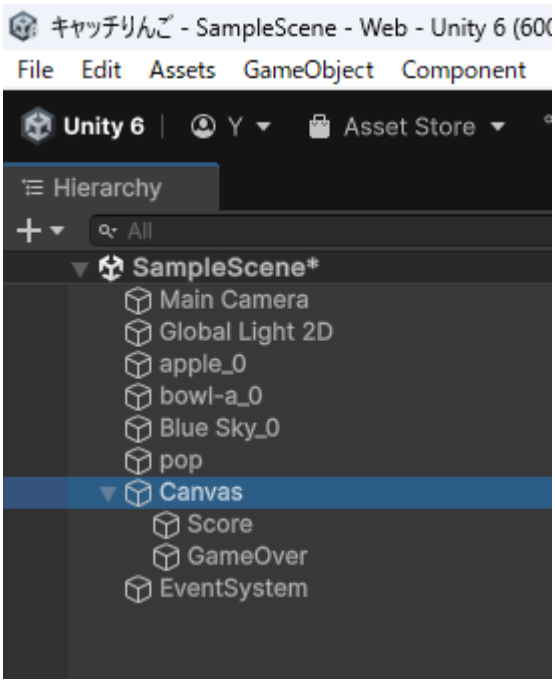


UI

Legacy
Text

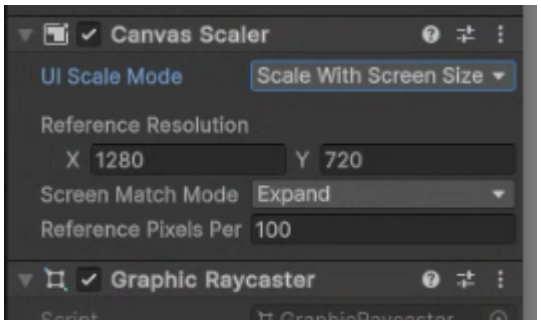
HierarchyのCanvas/**Text**を選択し、名前をScoreと変える

メニューのComponet/UI/Legacy/**Text** を選択し、もう一つTextを追加
HierarchyのCanvas/**Text**を選択して、名前をGameOverと変える



6．2 Canvasを設定

HierarchyのCanvasを選択
CanvasのInspectorを変更



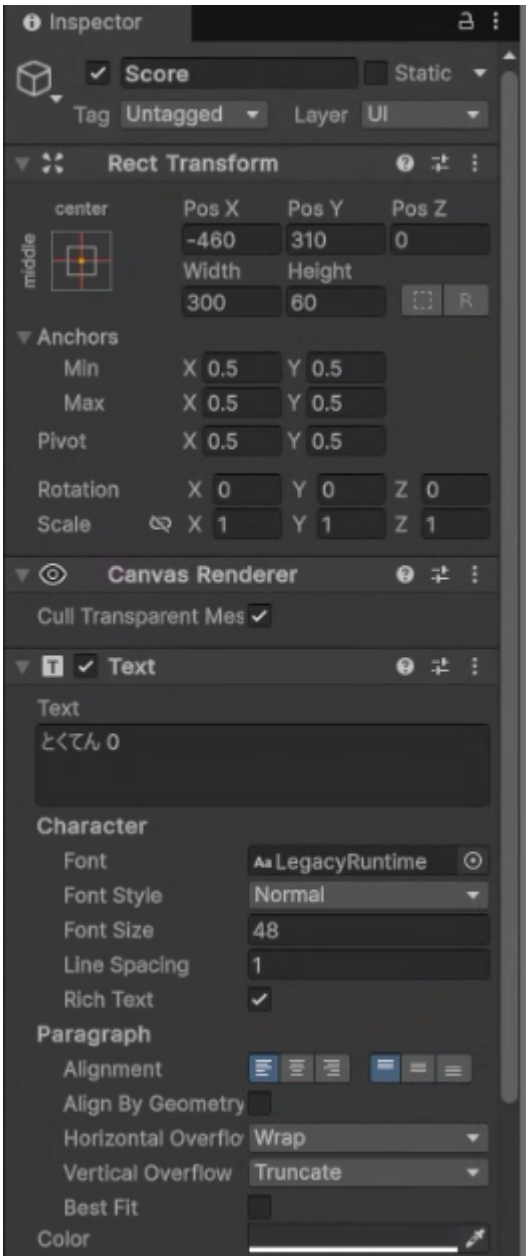
Canvas ScalerのUI Scale Mode: Scale With Screen Size
※スクリーンサイズに調整

Reference Resolution X: 1280 Y:720
Screen Match Mode: Expand
※HDの大きさ。さらに、拡大縮小できるようにする

6．3 とくてんの

詳細を設定

HierarchyのScoreを選択
ScoreのInspectorを変更



Rect Transformの Pos X: -460 Pos Y: 310
Width 300: Height: 60

Textに「とくてん 0」

Font Size: 48

6．4 ゲームオーバーの詳細を設定

HierarchyのGameOverを選択
GameOverのInspectorを変更

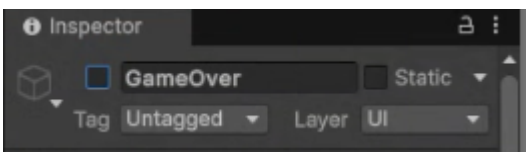
Rect Transformの Pos X: 0 Pos Y: 0
Width: 720 Height: 120

Textに「ゲームオーバー」

Font Size: 98

Alignment: 中　中

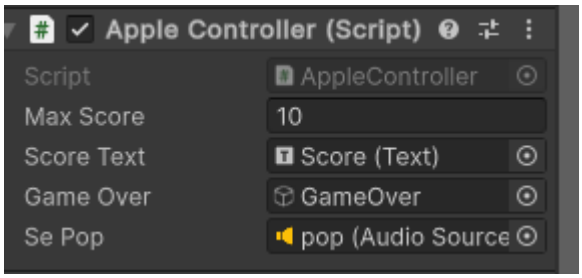
開始時に表示されないようにする



GameOverの右側の☑を外し、非表示する

6. 5 スクリプトで使えるようにする

Hierarchyのapple_0を選択
Inspectorの Apple Controller(Script) に値を入れる



Score Text: HierarchyのScoreをドラック＆ドロップ
Game Over: HierarchyのGmaeOverをドラック＆ドロップ

※ ここで指定した値やオブジェクトが、プログラムで利用できる

AppleController スクリプトの scoreText.text とgameOver.SetActive のコメント // をはずす。
命令文が無い時は、追加

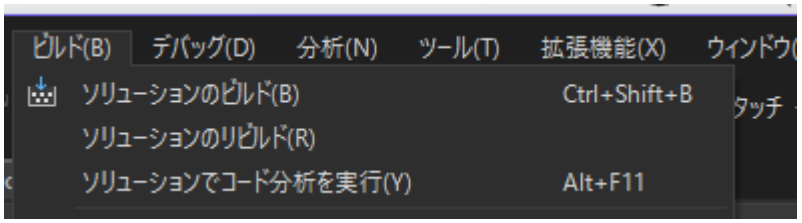
前
// scoreText.text = "とくてん " + score.ToString(); // スコアを表示

// gameOver.SetActive(true); // ゲームオーバー表示

後
  scoreText.text = "とくてん " + score.ToString(); // スコアを表示

  gameOver.SetActive(true); // ゲームオーバー表示

エディターのメニュー「ビルド」「ソリューションのビルド」を選択



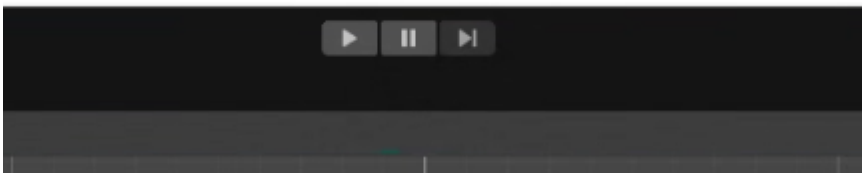
ビルド

ソリューションのビルド

エラーが無くなるまで、プログラムを修正

6. 5 動かしてみる

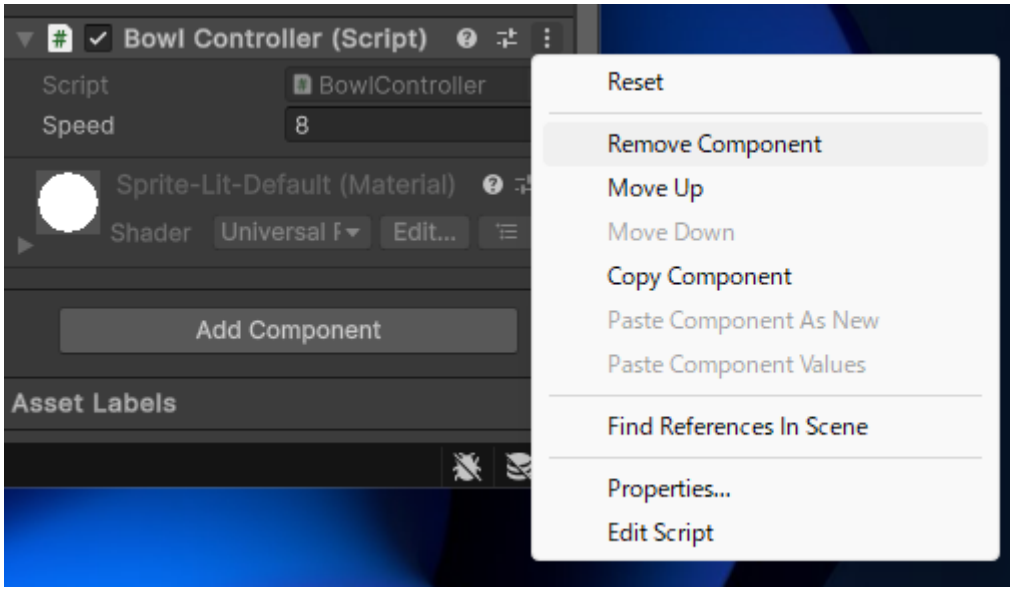
▶️ボタンを押して、とくてんとゲームオーバーが表示を確認



チャレンジ

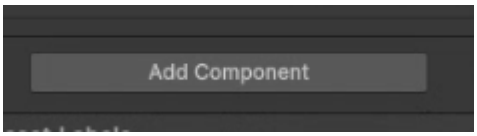
物理演算を利用したボールの移動

スクリプト BowlController1 を作成
Hierarchyの bowl-a_0を選択
BowlController を Remove Component で削除



Remove Component

Inspectorを一番下まで表示
Project のBowlController1 をInspectorの[Add Componet]へめがけて、ドラック&ドロップして追加



速度を与えるlinearVelocityと力を与えるAddForceの2通りある。スクリプトでは、キーとゲームパッドの両方を利用できる Input.GetAxis() もためそう。利用する行のみコメントをはずして、ビルドして実行

```
rb2d.linearVelocity = keySpeed; // X軸の速度設定
//rb2d.linearVelocity = axisSpeed; // X軸の速度設定

//rb2d.AddForce(keySpeed); // X軸に力を加える
//rb2d.AddForce(axisSpeed); // X軸に力を加える
```

BowlController1

```
using UnityEngine;

public class BowlController1 : MonoBehaviour
{
    public float speed = 8f; // 移動速度
    private Rigidbody2D rb2d; // Rigidbody2Dコンポーネント
    private float playerSpeed; // プレイヤーの速度
    private float moveHorizontal; // 水平方向の移動量

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        rb2d = GetComponent<Rigidbody2D>(); // Rigidbody2Dの取得
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            playerSpeed = -speed; // 左に移動
        }
        else if (Input.GetKey(KeyCode.RightArrow))
        {
            playerSpeed = speed; // 右に移動
        }
        else
        {
            playerSpeed = 0; // 停止
        }

        moveHorizontal = Input.GetAxis("Horizontal"); // 水平方向の入力取得

    }

    // 物理演算ごとに呼ばれるメソッド
    void FixedUpdate()
    {
        Vector2 keySpeed = new Vector2(playerSpeed, 0); // キー入力からの値
        Vector2 axisSpeed = new Vector2(moveHorizontal * speed, 0); // キー、ゲームパッドからの値

        rb2d.linearVelocity = keySpeed; // X軸の速度設定
        //rb2d.linearVelocity = axisSpeed; // X軸の速度設定

        //rb2d.AddForce(keySpeed); // X軸に力を加える
        //rb2d.AddForce(axisSpeed); // X軸に力を加える
    }
}
```

りんごの動きを変える

りんごのキャッチに失敗すると、りんごの落ちるスピードが早くなっていきます。また、ボールに当たるとりんごが回転したり、斜めになったりします。これらは、Rigiedbody2Dコンポーネントを追加して物理演算を行い現実の世界に近い動きをさせているから起きている動きの変化です。

Scratchと同じようにキャッチに失敗したとき、次に落ちてくるとき、次の3つの状態を最初にりんごを落としたときと同じ状態にプログラムを変更してみよう。

- 回転を ゼロ にする rotation
- 速度を ゼロ にする linearVelocity
- 回転の速度（角速度）を ゼロ にする angularVelocity

プログラムを使わない方法もあるかも。

回答例
Nextに3行追加

```
private void Next()
    {
        gameObject.transform.position = new Vector3(Random.Range(-6, 6), 4, 0); // 位置の初期化
        gameObject.transform.rotation = Quaternion.identity; // りんごの回転を初期化
        gameObject.GetComponent<Rigidbody2D>().linearVelocity = new Vector2(0, 0); // りんごの速度を初期化
        gameObject.GetComponent<Rigidbody2D>().angularVelocity = 0; // りんごの角速度を初期化
    }
```

X Z の動きを固定にすることで、回転を止めることができる。

ボールの動きを変える

りんごがホールの右端にぶつかるとボールに左方向への力がかかり、ボールは左に移動します。反対側の左端にぶつかるとボールに右方向への力がかかりボール右に移動します。そのままにしておくと、画面からはみ出し戻ってこなくなります。これらは、Rigiedbody2Dコンポーネントを追加して物理演算を行い現実の世界に近い動きをさせているから起きている動きの変化です。

ボールが画面からはみ出さないようにしてみましょう。方法は、プログラムを利用する方法、プログラムを利用しない方法など、いくつかの方法があります。いろいろなアイデアを考え対応してみましょう。

回答例

プログラムを利用する方法
- ボールが画面の橋の位置にいるときに、それ以上位置を動かさない
- キー入力がないときボールの移動速度を ゼロ にする

プログラムを利用しない方法
- ボールの質量を大きくして、衝突時に動きにくくする
- 画面の両端に物をおいて、それ以上移動しないようにする
- 摩擦のある素材を下において、移動にブレーキをかける

ゲームのルールを変更する

AppleController は、10回キャッチするまで、りんごが落ちてきます。Scratchの「キャッチゲームりんご」は、10回りんごが落ちてくる間に何個獲れるかと違ったルールになってます。Scratchの「キャッチゲームりんご」と同じルールとなるようにプログラムを変更してみましょう。

補足

プログラムコードのScratchとの比較（ボール）

```
public float speed = 8f; // 移動速度


void Update()
{
    if (Input.GetKey(KeyCode.LeftArrow)) // 左矢印キーが押されたとき
    {
        transform.position -= speed * transform.right * Time.deltaTime; // 左に移動
    }
    else if (Input.GetKey(KeyCode.RightArrow)) // 右矢印キーが押されたとき
    {
        transform.position += speed * transform.right * Time.deltaTime; // 右に移動
    }
}
```

```
public float speed = 8f; // 移動速度
```

公開 浮動小数点数 名前 = 8浮動小数点数。コメント

公開：他のプログラムから見れる
浮動小数点数：少数を含む数
名前：変数（プロパティ）の名前
コメント：説明文

変数の宣言と同時に数字を指定できる。

```
void Update()
```

戻り値の型 名前 引数

戻り値の型：void　空、なし
名前：関数（メソッド）の名前
引数：関数に渡す変数 () なし

```
if (Input.GetKey(KeyCode.LeftArrow)) // 左矢印キーが押されたとき
```

```
if (　　)
{
```

評価の結果、条件が成立した（真、true）なら、この場所を実行

```
}
else if (　　)
{
```

評価の結果、条件が成立した（真、true）なら、この場所を実行

```
}
```

```
Input.GetKey(　　)
```

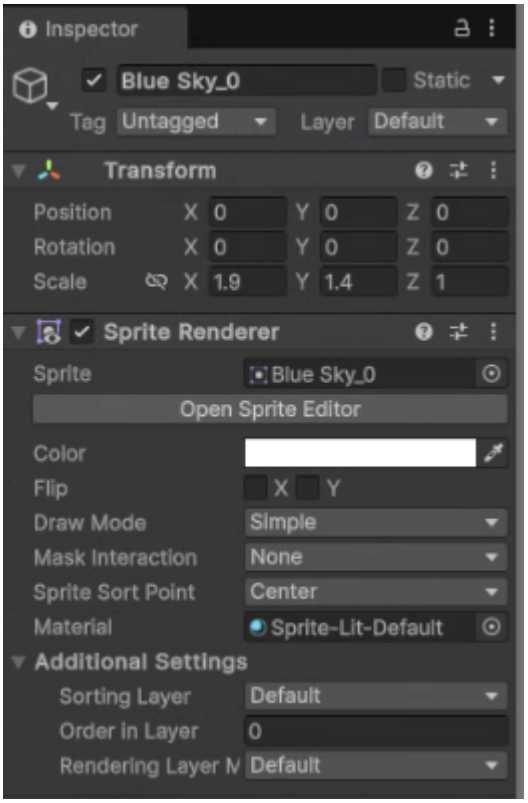
引数で指定されたキーが押されているか確認

```
KeyCode.LeftArrow
```

キーコード 左矢印

transform.position -= speed * transform.right * Time.deltaTime; // 左に移動

transform.position とは？



Tag タグ 名札

Transform トランスフォーム 変形させる
Position ポジション 位置
Rotaion ローテーション 回転
Scale スケール 縮尺

プログラムでは、ポジションを TransformのPositionを transform.position と表す。
positionは、x,y,xの3つの情報（3次元の情報）を持っている。3次元の情報は、Vector3 ベクター3 で表す

変数の型
Scratchで全角の数字を入れても数字として認識されない。文字として認識してしまう。数字を指定したい場合は、半角の数字を入れることで、Scratchが数字と判断してくれる。Scratchは、数字と文字の2つの型がある。コンピューターでは、情報に型があり型を指定することで、コンピューター内部の処理方法を変えている。

どんな型がある？
-2, -1, 0 ,1 ,2 は、整数で Int インテジャー　（イント）で表す。
-2.1, -1.0, 0.0, -1.5, -2.0 は、浮動小数点数 float フロートで表す。

整数、浮動小数点数は、掛けたり割ったり、足したり引いたりできる。

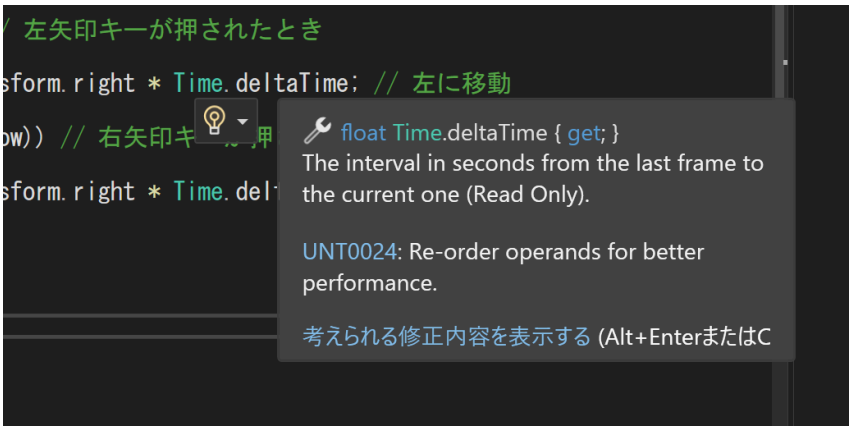
Vector3も同様に足したり引いたりできる。

```
Vector3 a = new Vector3(1, 2, 3) + new Vector3(4,5,6);
```

Vector3とVector3を掛けたり割ったりすることは、できない。

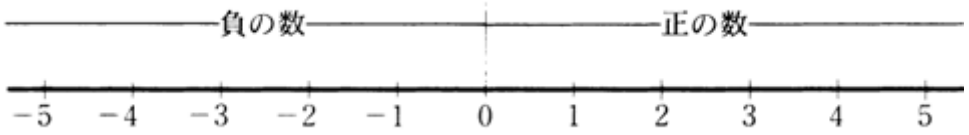
```
Vector3 a = new Vector3(1, 2, 3) * new Vector3(4,5,6);
```

計算で利用している変数の型を確認
transform.position: Vector3
speed: float
transform.right: Vector3
Time.deltaTime: float



Q.なんで、int float があるの？floatだけあれば、すべて計算できるでしょう。
A.コンピューターにとって、浮動小数点数の計算は、不得意。整数の方が、計算が早いしデータ量も少ない。

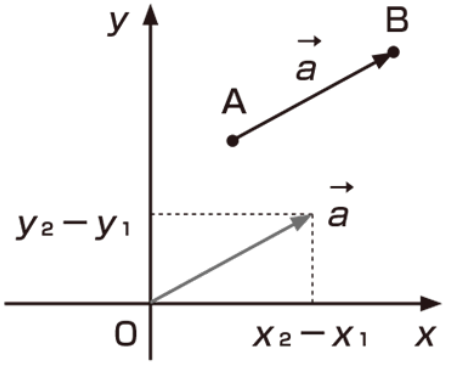
※
スカラー(Int, float) 1つの数値で表す。数直線（左から右の1本線）上の数値。



ベクトル(英語では、ベクター、Vector3, Vector2) 大きさと方向を持つ。複数の数値で表す。平面や立体上の数値。
計算方法は、「 3次元ベクトルの計算」で検索すると難しい式が出てくる。

---

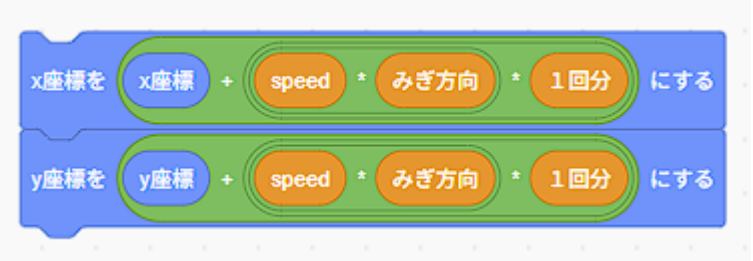
xy平面上に2点$A(x_1, y_1)$, $B(x_2, y_2)$をとると$\overrightarrow{AB}$が定まる。



(終点B)−(始点A)=(x_2-x_1, y_2-y_1)を$\overrightarrow{AB}$の成分という

プログラムの計算例

transform.position += speed * transform.right * Time.deltaTime; // 右に移動



⇓　transform.position += は、計算結果を足し込む。上と下の式は、同じ意味

transform.position = transform.position + speed * transform.right * Time.deltaTime;

speed * transform.right 右方向の **Vector3 (1, 0,0)**に **speed**倍した値を計算
speed * Vector3 (1, 0, 0)
Vector3(speed * 1, speed * 0, speed * 0)
Vector3 (speed, 0,0)

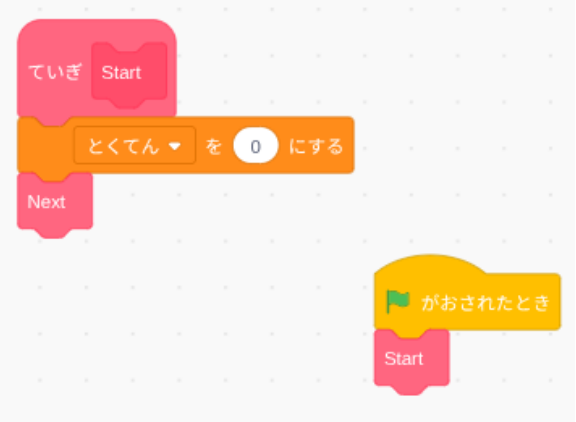
*Time.deltaTime **1**フレーム分の値に補正
Vector3(speed, 0,0) * Time.deltaTime
Vector3(speed * Time.deltaTime, 0 * Time.deltaTime, 0 * Time.deltaTime)
Vector3 (speed * Time.deltaTime, 0)

transform.position + 元の位置に計算結果を足す
Vector3(x, y, z) + Vector3 (speed * Time.deltaTime, 0)
Vector3(x + speed * Time.deltaTime, y + 0, z + 0)
Vector3(x + speed * Time.deltaTime, 0, 0)

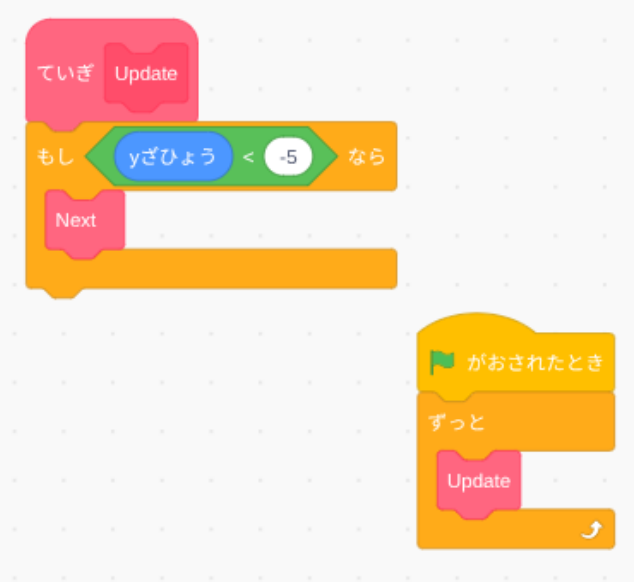
transform.position = すべての計算結果に置き換える
Vector3(x + speed * Time.deltaTime, 0, 0)

プログラムコードのScratchとの比較（リンゴ）

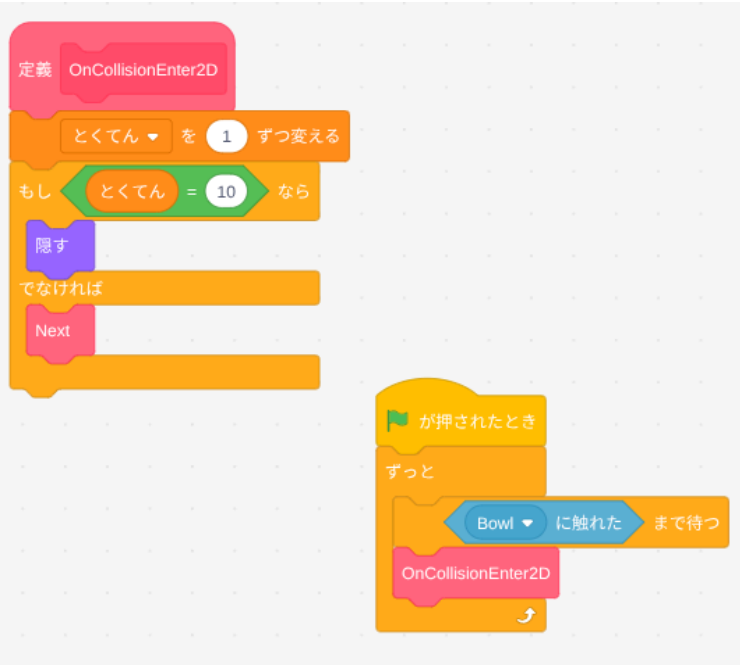
```
void Start()
{
    score = 0; // スコアの初期化
    Next(); // りんごの初期位置を設定
}
```




```
void Update()
{
   // Debug.Log("y: " + transform.position.y);
    if (transform.position.y < -5) // りんごが画面外に出たとき
    {
        Next(); // りんごの位置を初期化
    }
}
```




```
void OnCollisionEnter2D(Collision2D collision)
{
    score += 1; // スコアを加算
    if (score == maxScore) // スコアが最大値を超えた場合
    {
        Destroy(gameObject); // りんごを削除
    }
    else
    {
        Next(); // りんごの位置を初期化
    }
}
```



```
private void Next()
{
    gameObject.transform.position =
        new Vector3(Random.Range(-6, 6), 4, 0); // 位置の初期化
}
```

インプットシステムの確認

Input Manager と Input System Packageの2つのシステムが入っていると、競合するので、実行時にConsoleにエラーが出る。

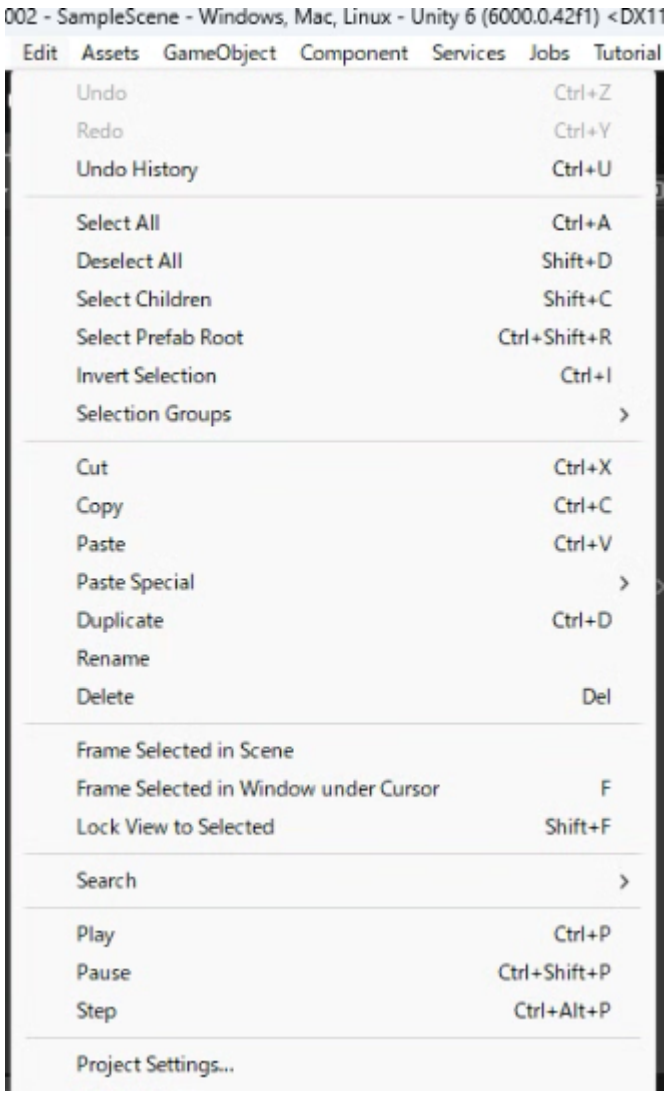
Unity6.1だと次のエラーが発生するかも

エラー内容
InvalidOperationException: You are trying to read Input using the UnityEngine.
Input class, but you have switched active Input handling to Input System package in Player Settings.

InvalidOperationException: You are trying to read Input using the UnityEngine.Input class, but you have switched active Input handling to Input System package in Player Settings.
UnityEngine.Input.GetKey (UnityEngine.KeyCode key) (at /Users/bokken/build/output/unity/unity/Modules/InputLegacy/Input.bindings.cs:414)
BowlCont.Update () (at Assets/BowlCont.cs:16)

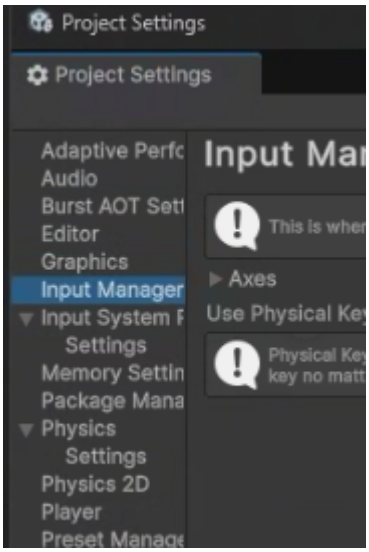
確認と回避方法

メニューよりEdit/ProjectSettings.. を選択

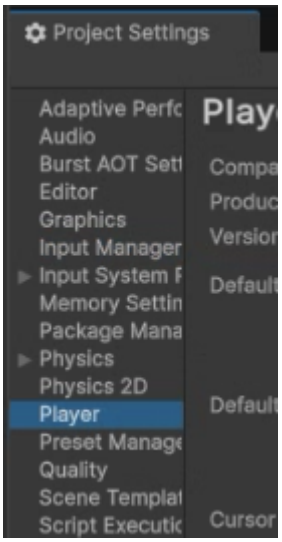


Project Settings Windowで

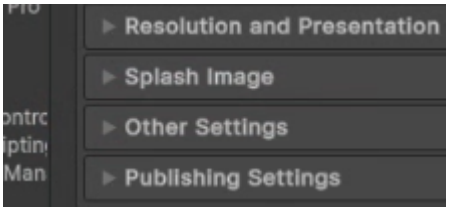
Input Manager と Input System Package があるか確認。



Playerを選択



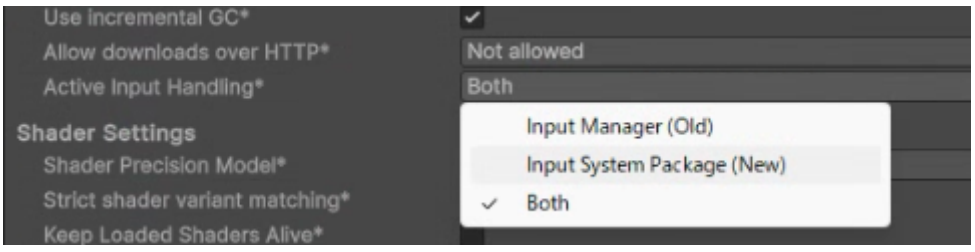
Other Settingsの▸をクリックして開く



Configuration

Active Input Handling の項目を確認

Bothになってないときは、Bothにチェックを入れる



[Apply] をクリック
Unityを再起動

【Unity】Input Systemの使い方入門
https://nekojara.city/unity-input-system-intro