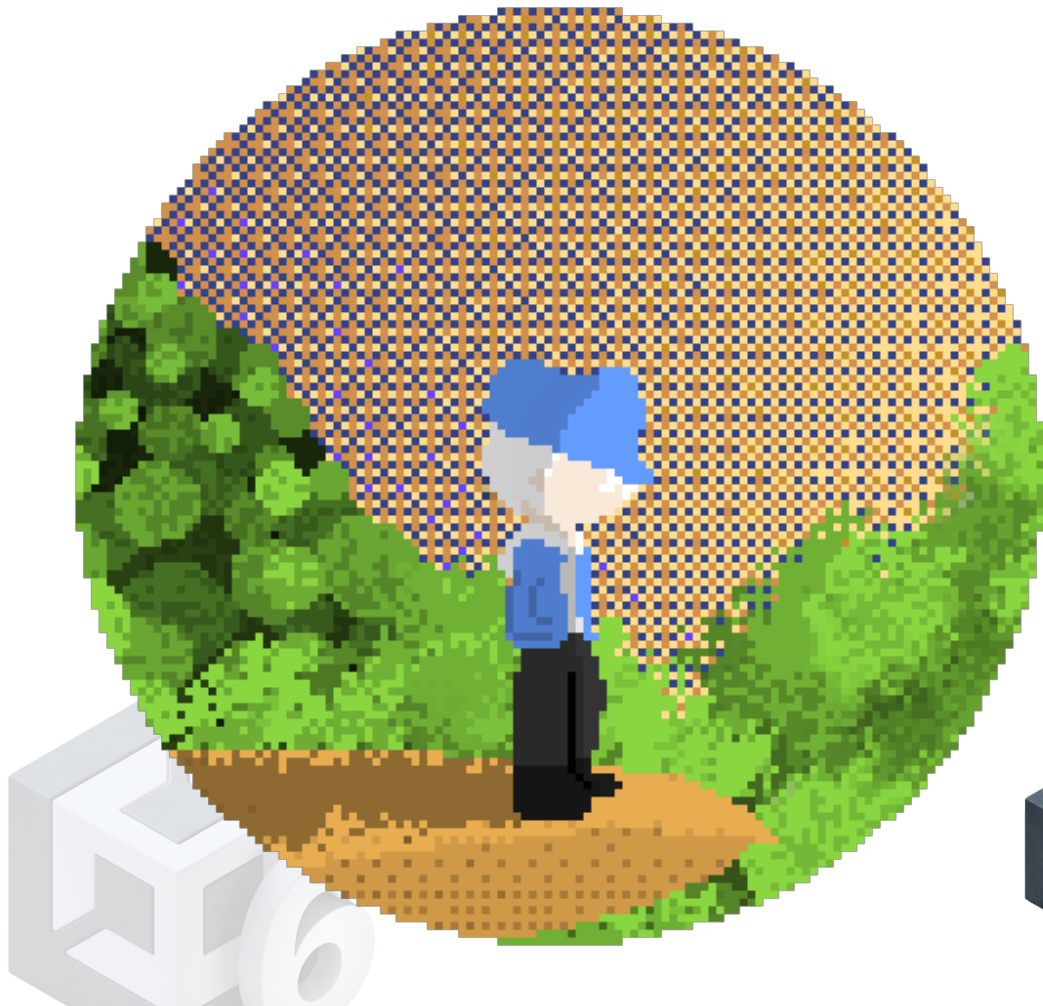


Unity 初心者講座

3. 細かい設定

自己紹介



21st/部長/インフラ

ロペ

Unityでゲームを作りながら
ドット絵でお絵描きしている
電子情報システム学科の2年



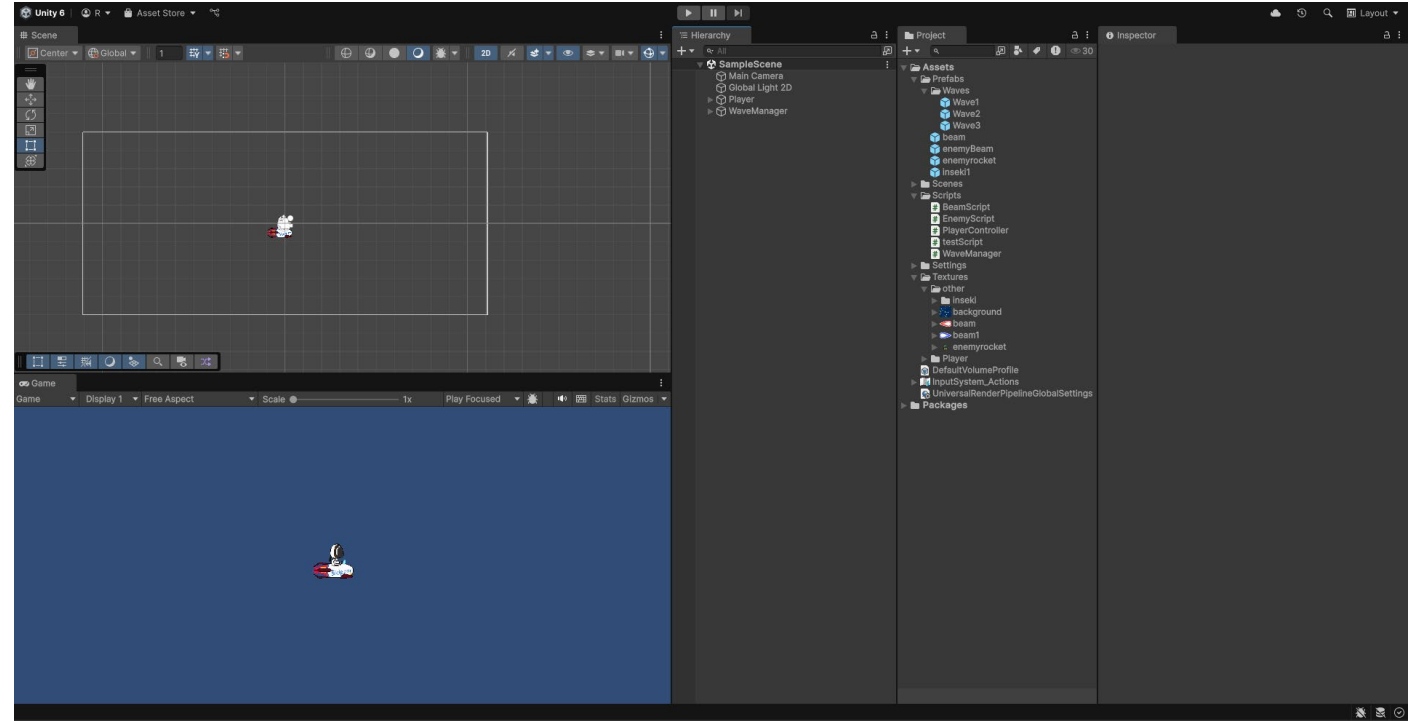
Unity初心者講座
-簡単シューティングゲーム-

- 1 : 画面外の削除
- 2 : テキストの追加
- 3 : 背景スクロール
- 4 : アニメーション
- 5 : ゲームオーバー



1 : 画面外の削除

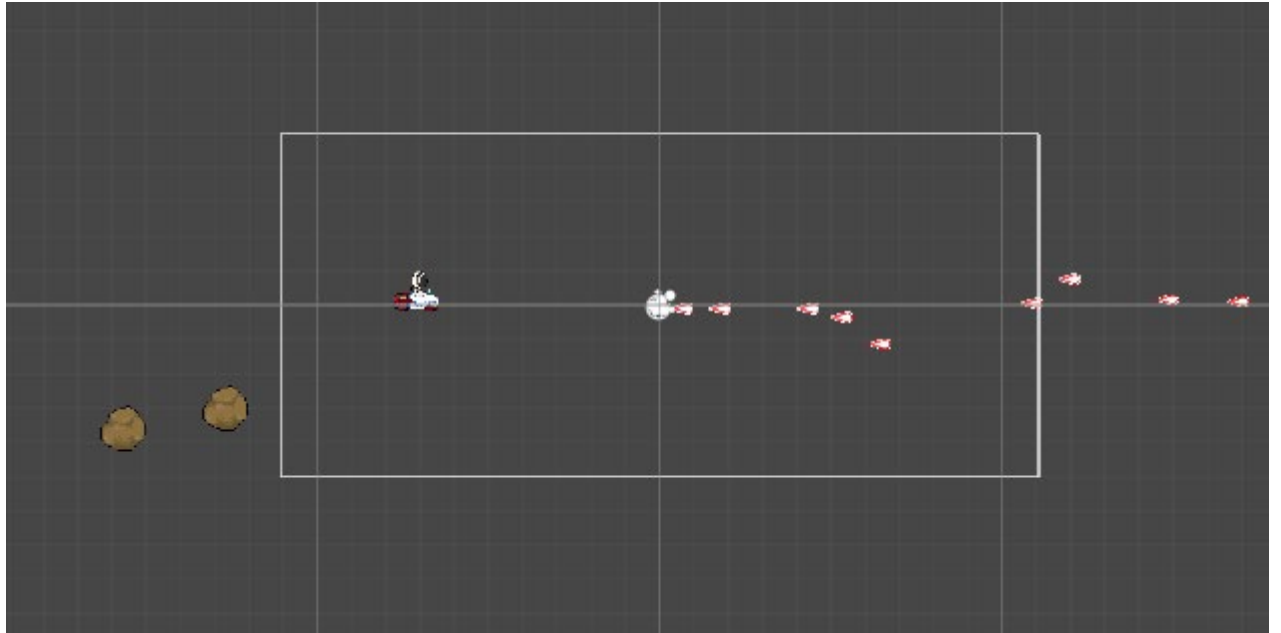
前回作成したプロジェクトを
UnityHubで開きなおす



1：画面外の削除

ある程度ゲームはできている
状態ですが、画面外に出た場合
敵が消えることなく
ずっと残ってしまってます

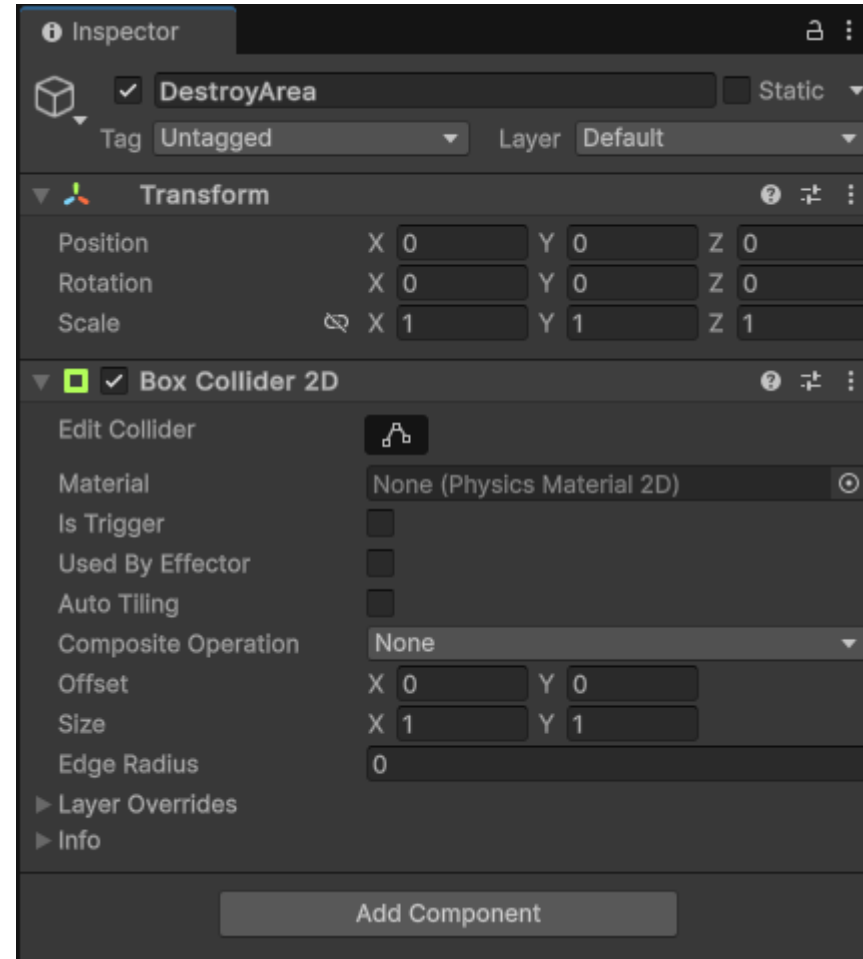
これがいくつもあると負荷に
なるので、削除することにします



1：画面外の削除

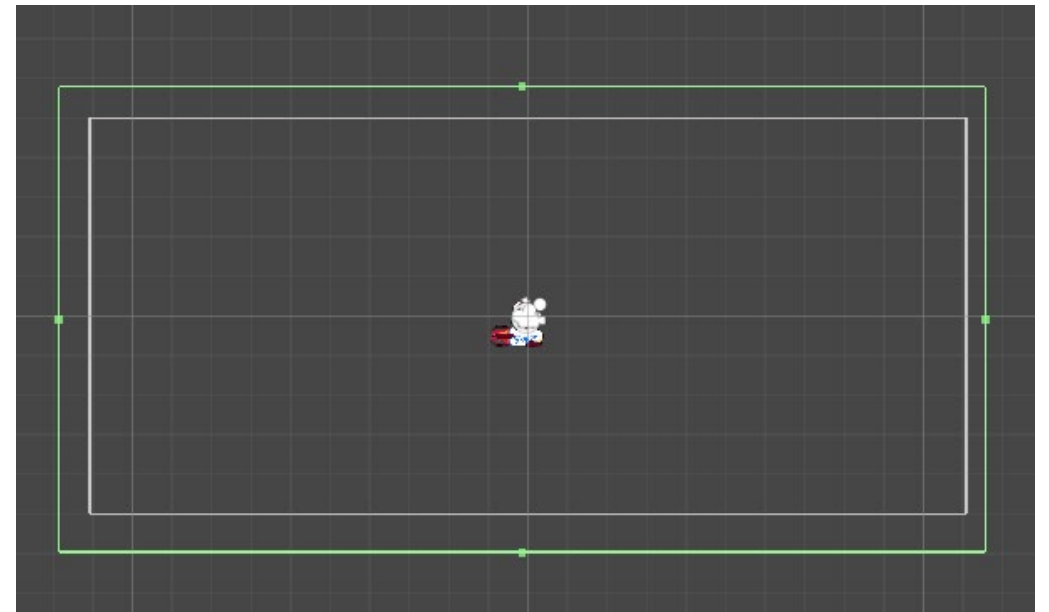
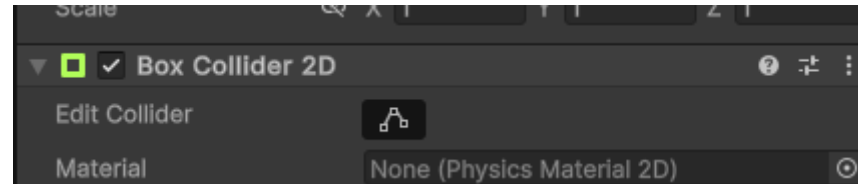
Create Emptyから
“DeatroyArea”を作成する

作成したらAddComponentから
“BoxCollider2D”を
追加し、カメラの枠に
合わせて当たり判定を調節する



1：画面外の削除

当たり判定を調節するときは
Sizeの値を設定してもいいが、
“Edit Collider”から
簡単に調節することができる



1 : 画面外の削除

判定を調節できたら
BoxCollider2Dの
“IsTrigger”をオンにする

Rigidbody2Dを追加して
GravityScaleを0にする

Tagは変える必要はない

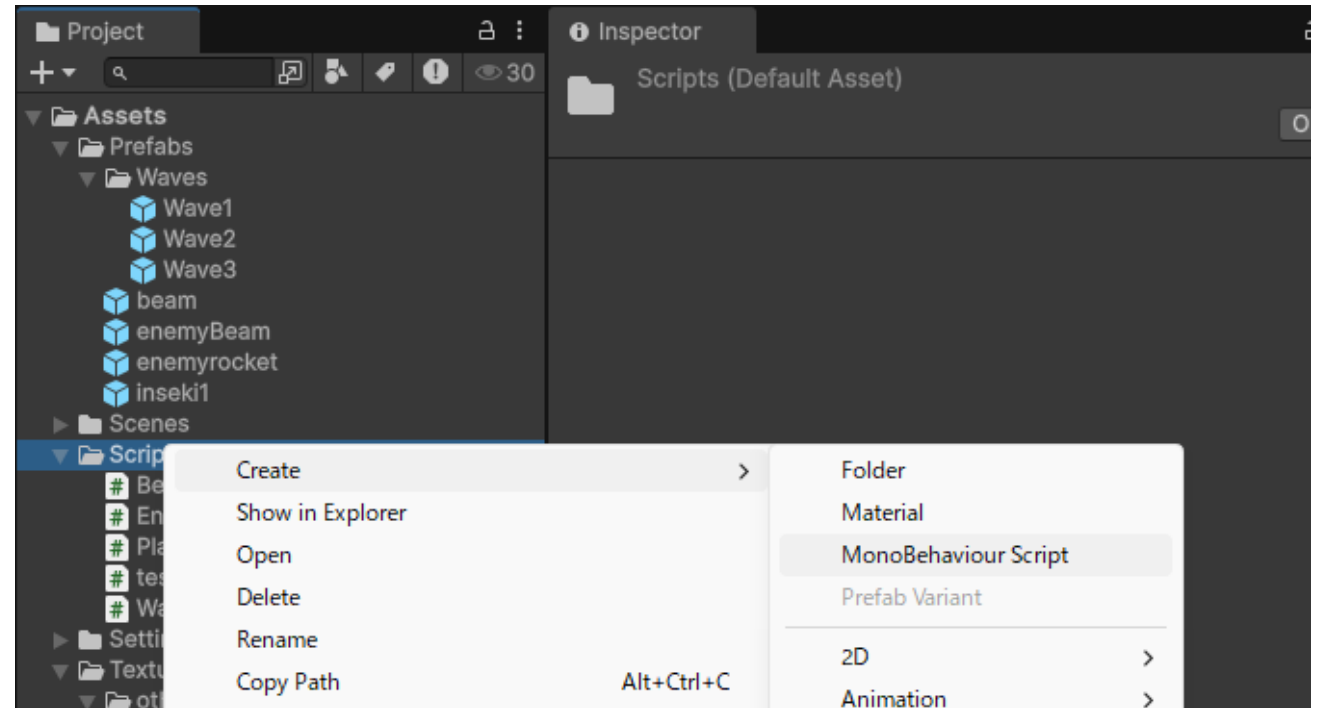


1：画面外の削除

設定し終わったら
スクリプトを作成する

Scriptsを右クリックし
MonoBehaviour Scriptを
作成する

名前は”DestroyScript”にしておく



1 : 画面外の削除

DestroyScriptの コードをうつす

なお、
void Startとvoid Updateは
使用してないので消してもOK



```
public class DestroyScript : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    // Unity メッセージ10 個の参照
    void Start()
    {
    }

    // Update is called once per frame
    // Unity メッセージ10 個の参照
    void Update()
    {
    }

    // Unity メッセージ10 個の参照
    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.CompareTag("Enemy") || collision.CompareTag("EnemyBeam") || collision.CompareTag("PlayerBeam"))
        {
            Destroy(collision.gameObject);
        }
    }
}
```

1：画面外の削除

コード解説

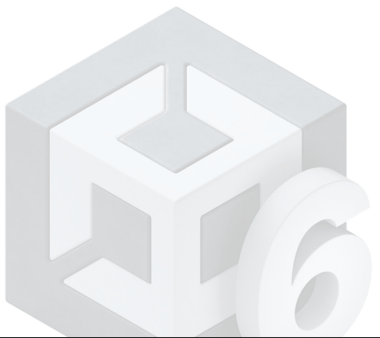
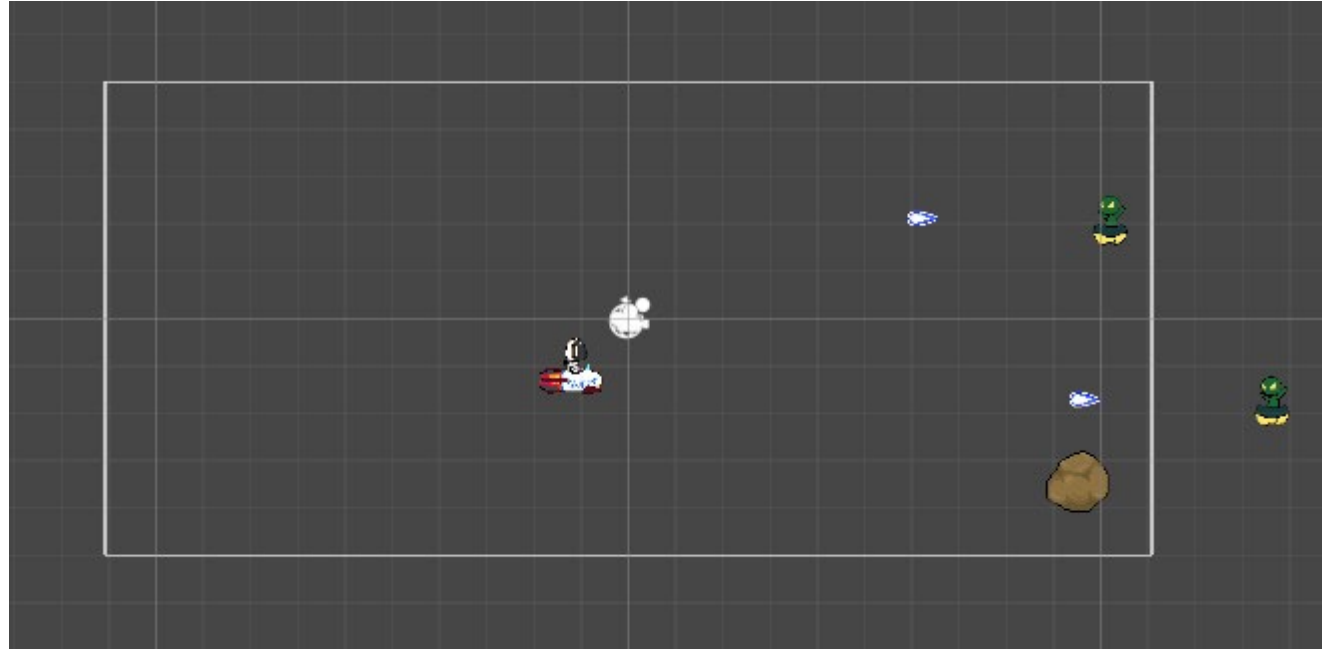
**DestroyAreaから
出ていくオブジェクトの
Tagが右の3つなら、
オブジェクトを削除する**

**Exitの場合が出ていくときの処理
(2章の補足を参照)**

```
Unity メッセージ10 個の参照
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.CompareTag("Enemy") || collision.CompareTag("EnemyBeam") || collision.CompareTag("PlayerBeam"))
    {
        Destroy(collision.gameObject);
    }
}
```

1：画面外の削除

**DestroyAreaの範囲外に
オブジェクトが行ったら
削除されるようになりました**



1：画面外の削除

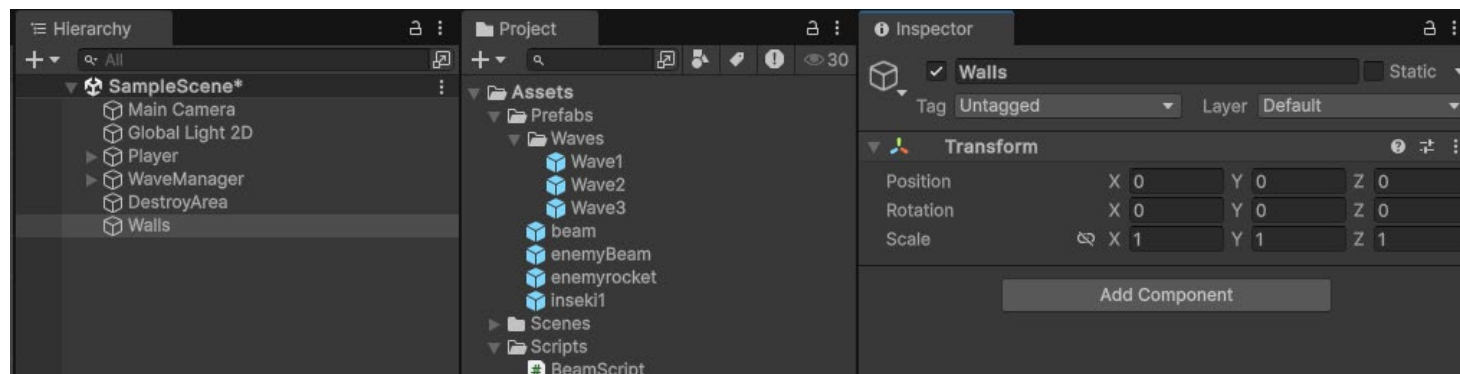
Playerが画面の外に
行けてしまうので
その処理をします

今回、移動の方法が
RigidbodyのlinearVelocityを
使用しているので
少し面倒な制御をしています



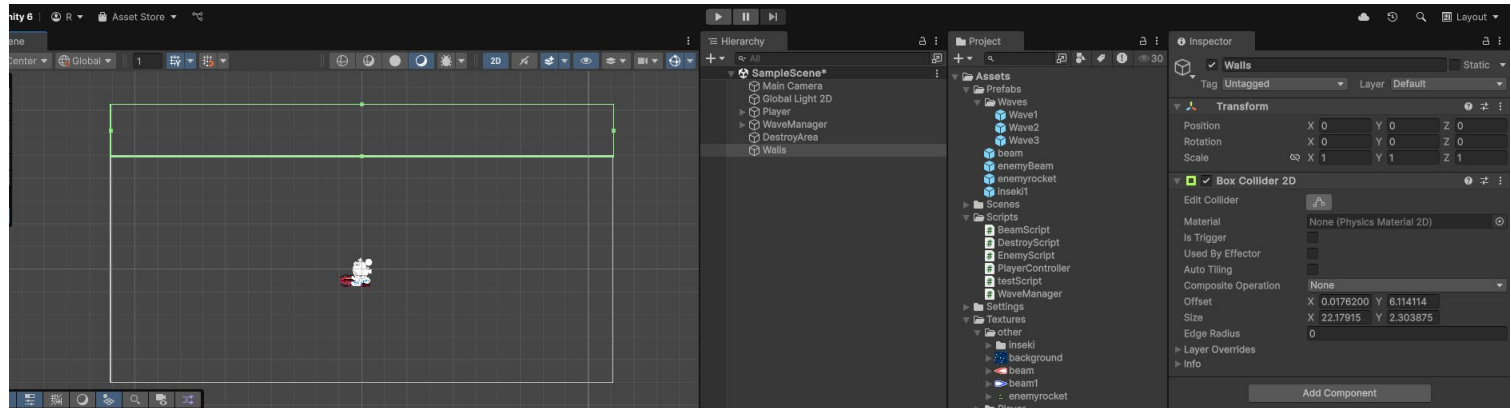
1 : 画面外の削除

Create Emptyで
“Walls”を作る



1 : 画面外の削除

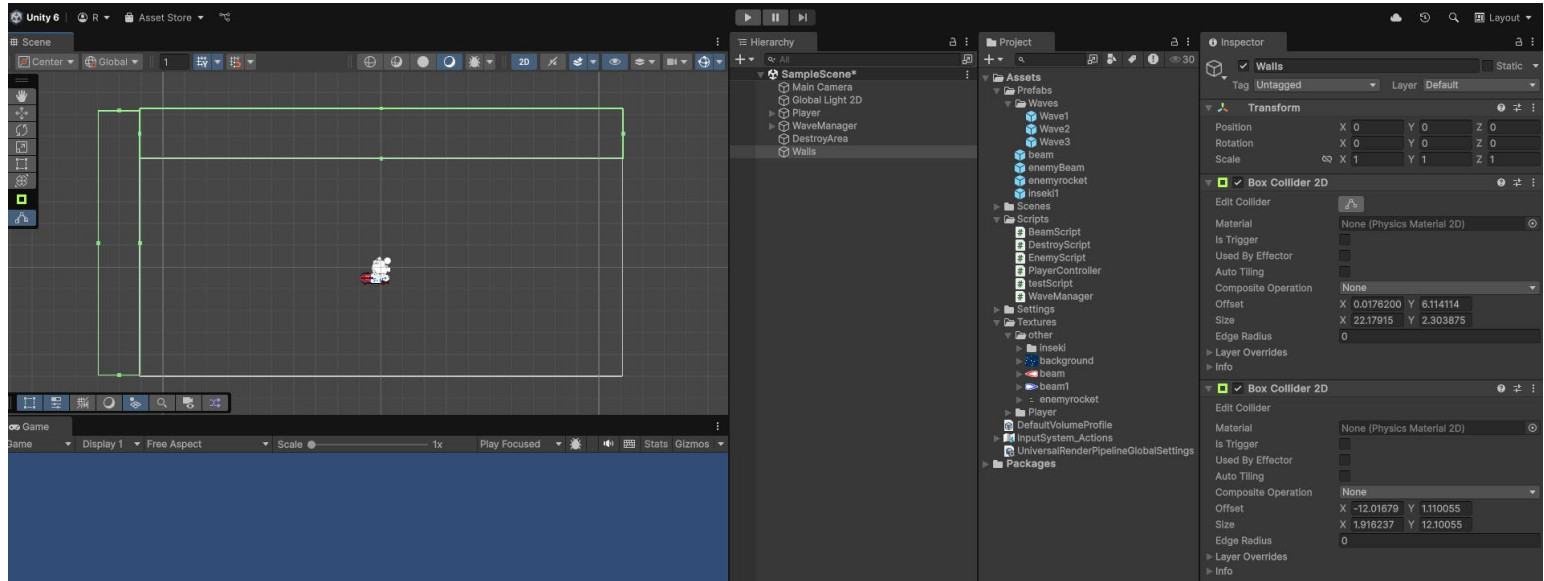
**Wallsに
BoxCollider2Dを追加し、
カメラの外の一辺を
覆いつくすように配置する**



1：画面外の削除

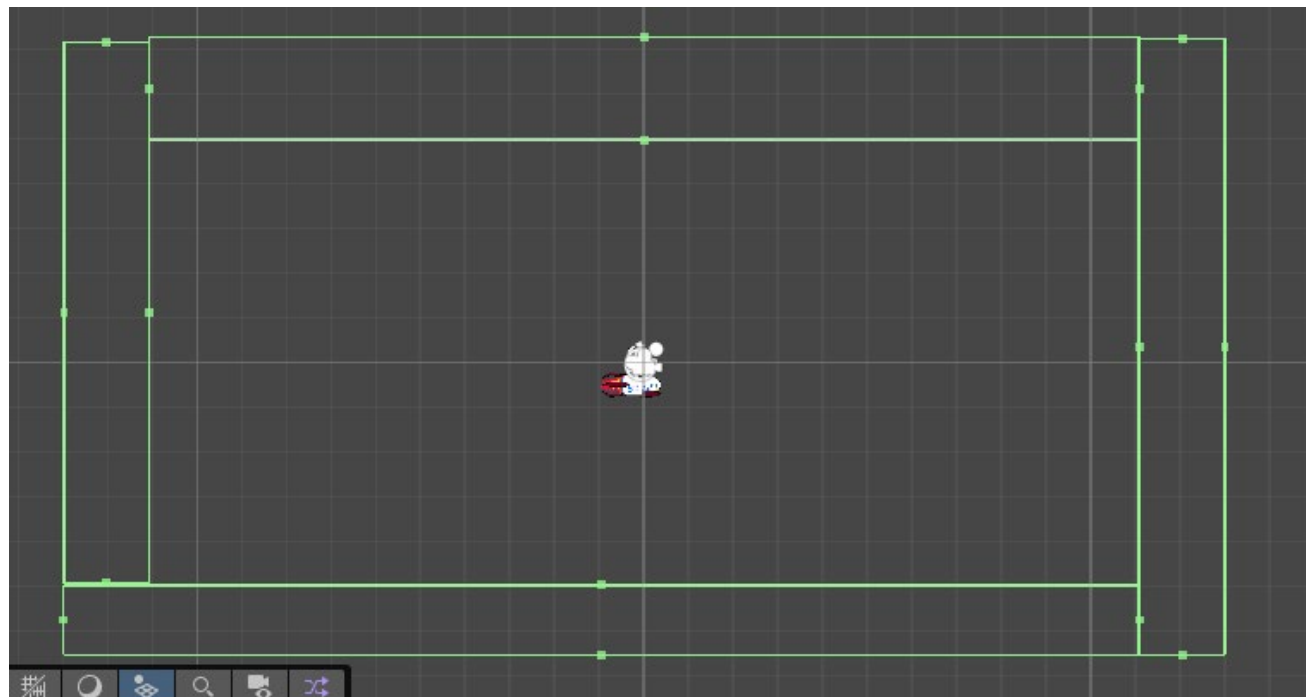
配置したら新しく
BoxCollider2Dを追加し、
空いている辺に対して
囲ってあげる

(Edit Colliderを使用する
ときは一番上のBoxColliderのものを
選ぶとすべて調節できるようになる)



1：画面外の削除

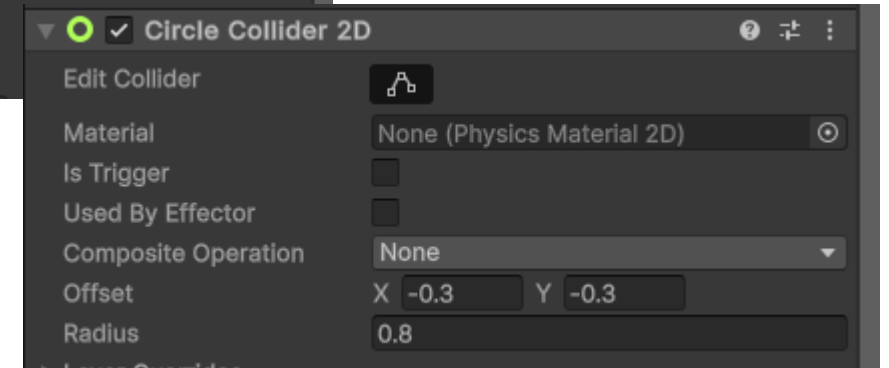
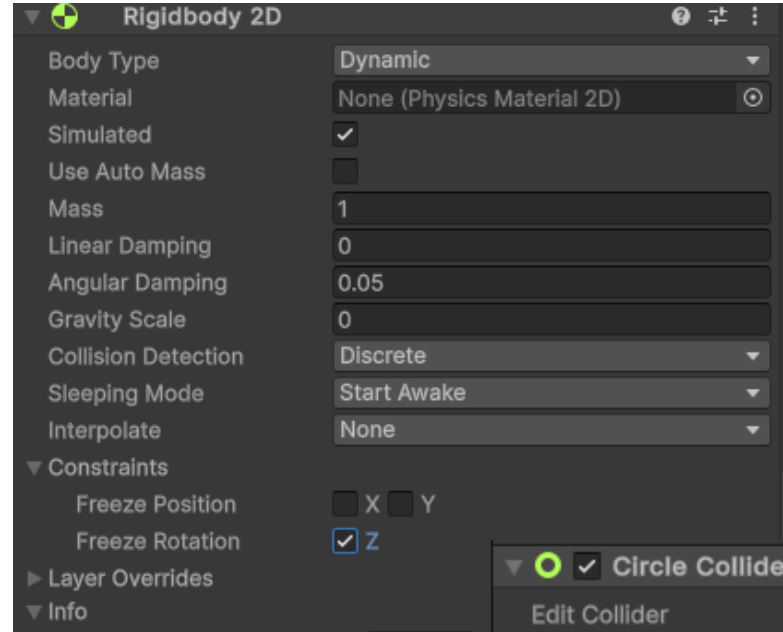
全部やる



1 : 画面外の削除

やったら
Playerの設定をする

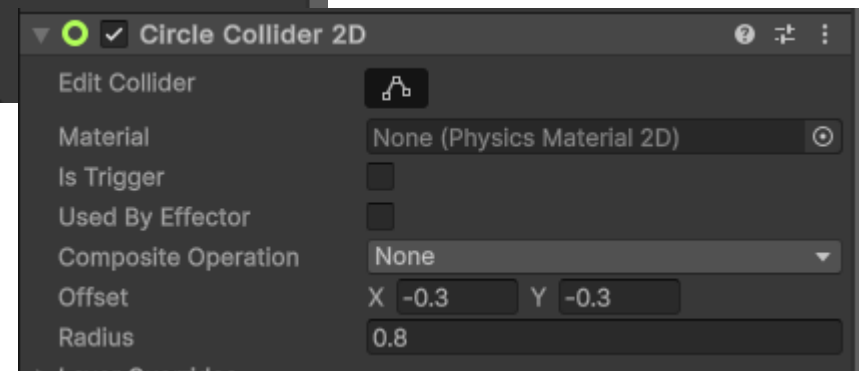
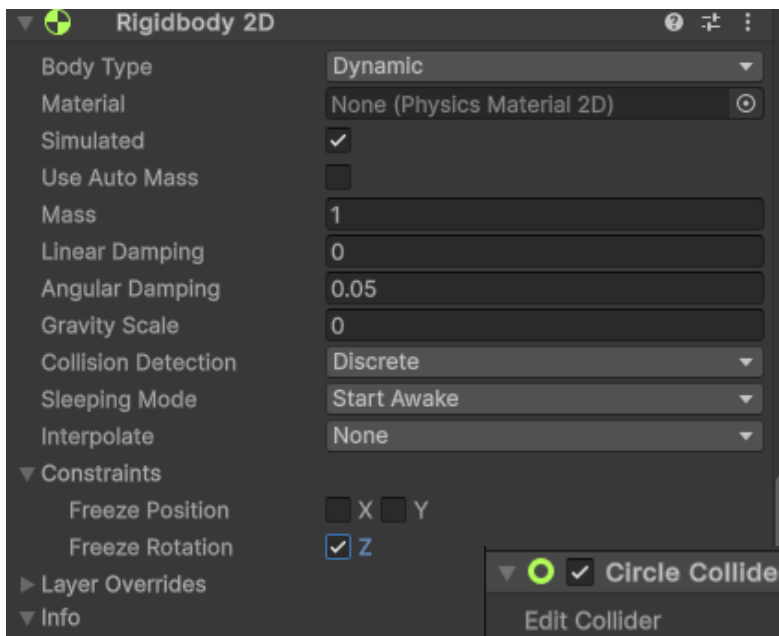
- Rigidbody2Dの
Constraintsの
“Freeze Rotation Z”を☒
- CircleCollider2Dの
“Is Trigger”の☐を外す



1 : 画面外の削除

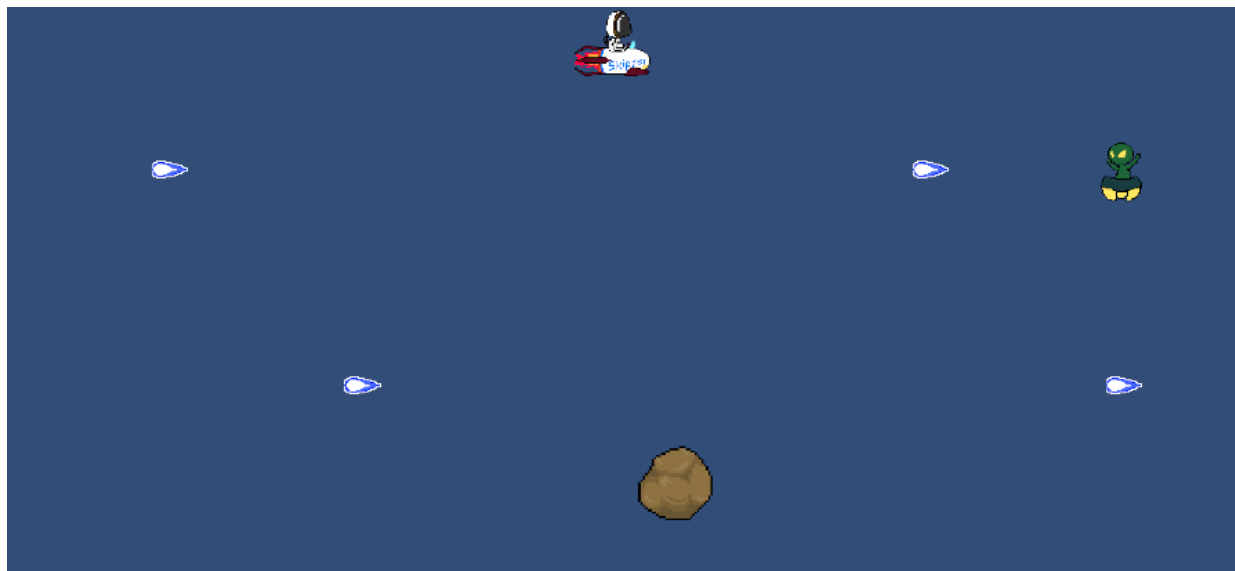
やったら
Playerの設定をする

- Rigidbody2Dの
Constraintsの
“Freeze Rotation Z”を☒
- CircleCollider2Dの
“Is Trigger”の☐を外す



1：画面外の削除

出ていかなくなりました



2：テキストを追加

HPの表示がDebugでしか
対応していないので、
画面上にHPの情報を載せます

ここではUIの話になります



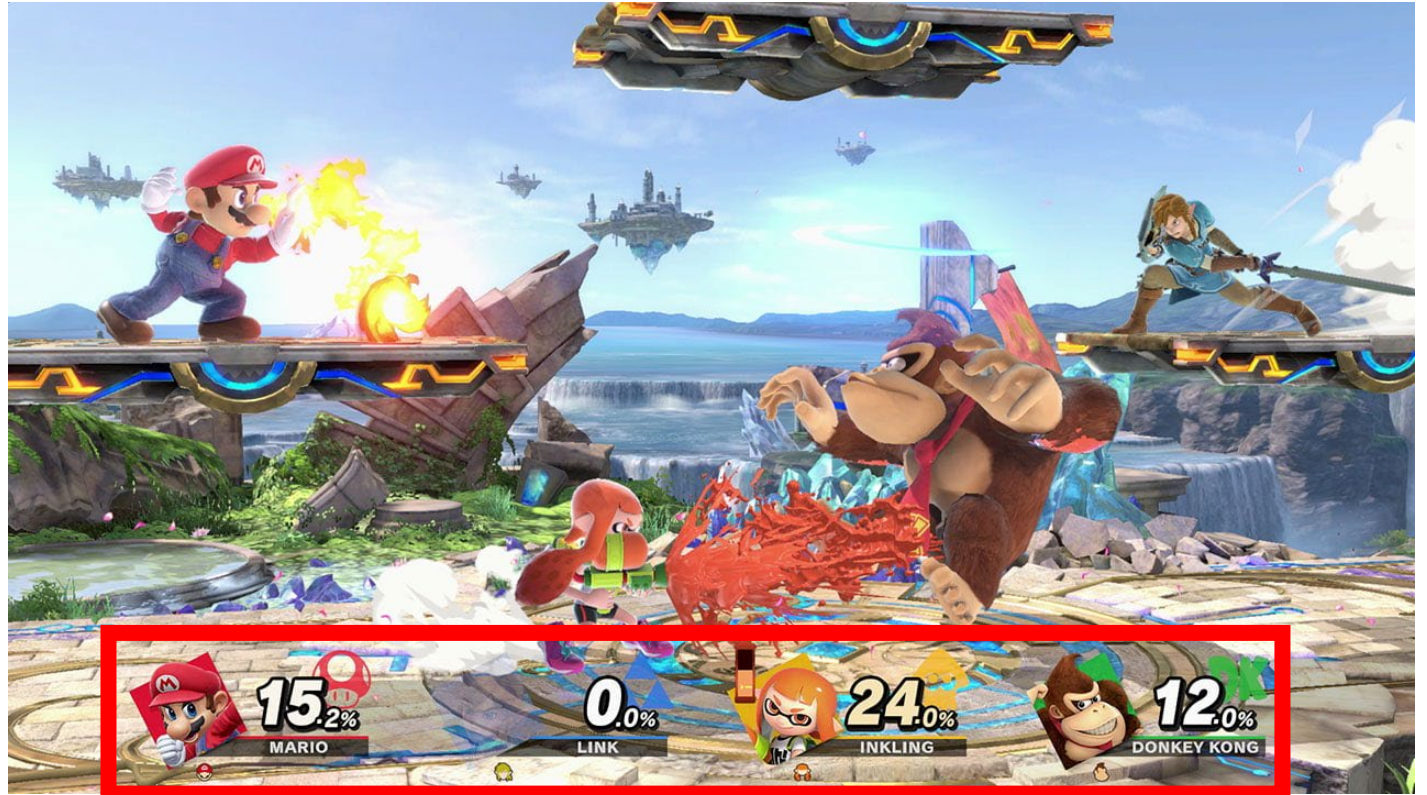
2：テキストを追加

そもそもUIとは？

UI：User Interfaceの略称

ゲーム上の情報を
画像やテキストなどで
プレイヤーに伝えるものを
UIと呼びます

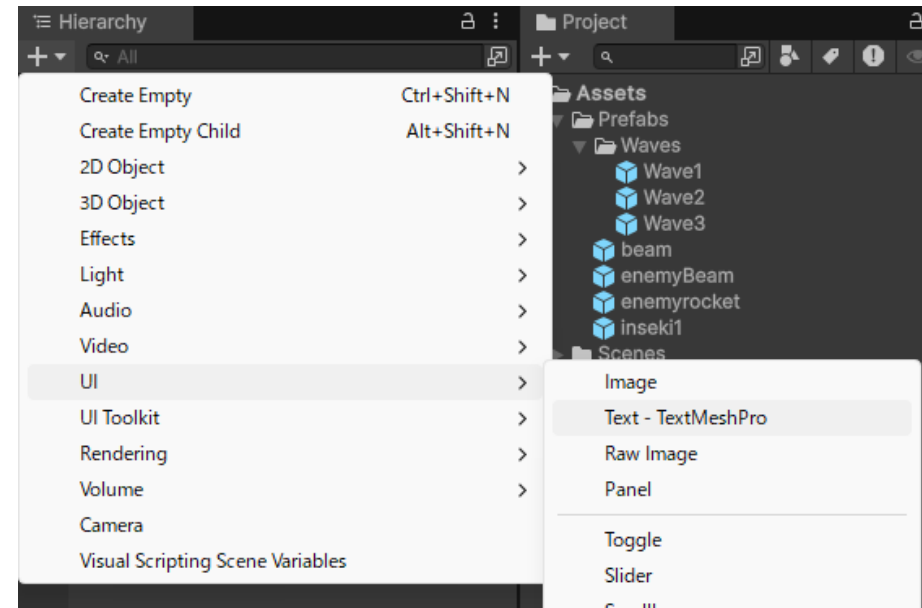
スマブラのUIだったら
下のアイコン、ダメージがUI
(マリオのアイコン、こんなのだけ???)



2：テキストを追加

今回はHPの情報をテキストでプレイヤーに伝えるもの、という感じでいく

まずはHierarchyの
プラス（もしくは右クリック）から
“UI / Text – TextMeshPro”
を選択

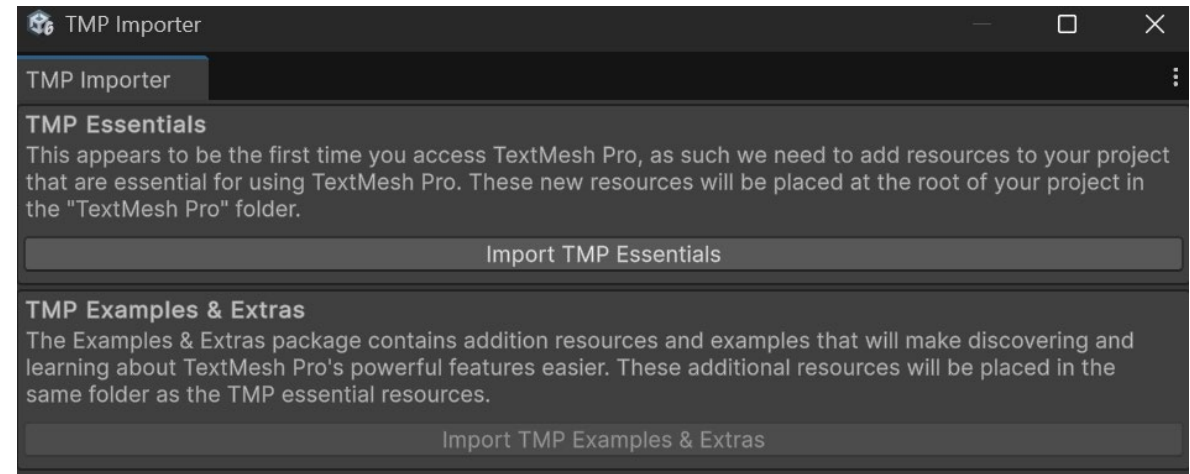


2：テキストを追加

追加すると
TMP Importerが出てくる

“Import TMP Essentials”を
押してテキストの素材をダウンロード

押した後、下のボタンが押せるようになるが、これは押さなくてもいい



2：テキストを追加

ダウンロードした後、
Hierarchyには
“Canvas”と”Text(TMP)”が
追加されている

Canvasというのは
UIを置くための土台のようなもの

Canvasの子オブジェクトに
テキストなどを置けば表示される

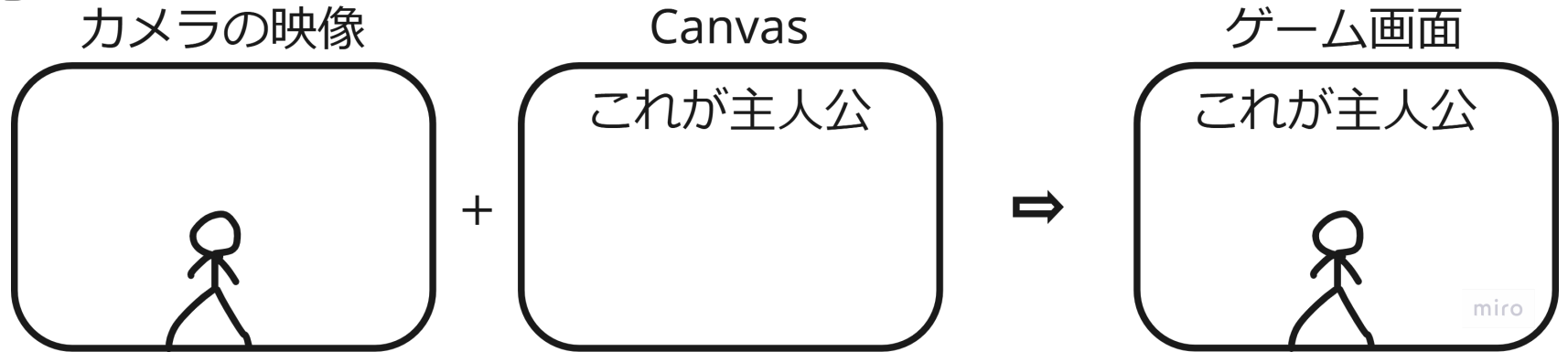


Sceneのカメラを離すと
くそでか四角形があるが、これがCanvas
左下の小さな四角形にプレイヤーがいます

2：テキストを追加

イメージはこう

Main Cameraに映っている映像と
Canvasにおいた情報を
合体させて
ゲーム画面として
表示している感じ

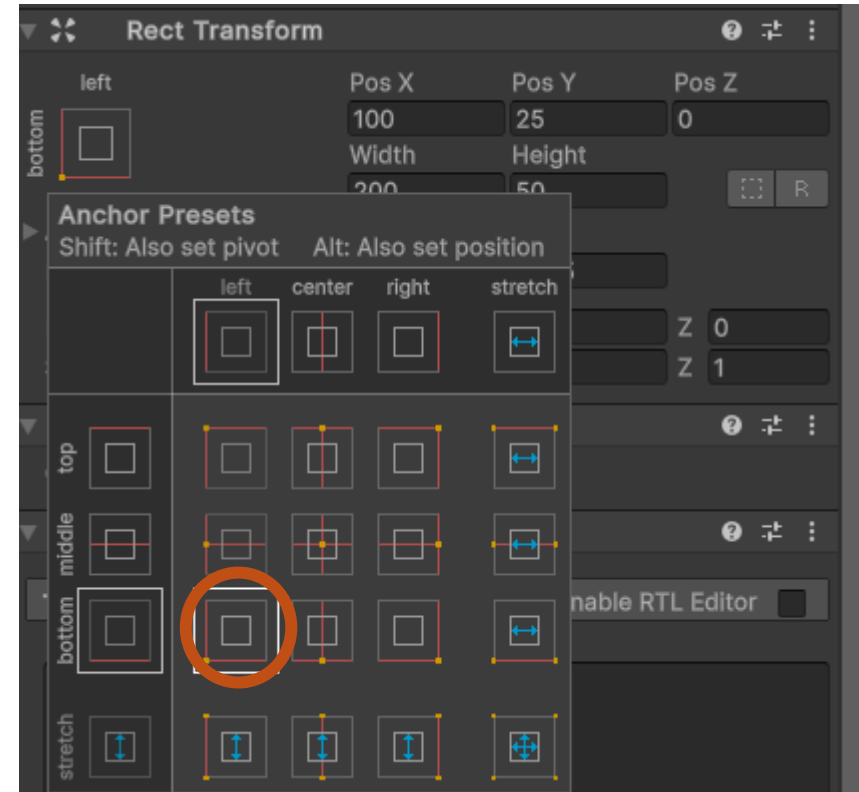


2：テキストを追加

テキストの位置を調節する

先に”Rect Transform”の
左に四角形のものがあるので、
それを画像のアイコンに設定

その後、
Pos X : 100
Pos Y : 25
に設定する

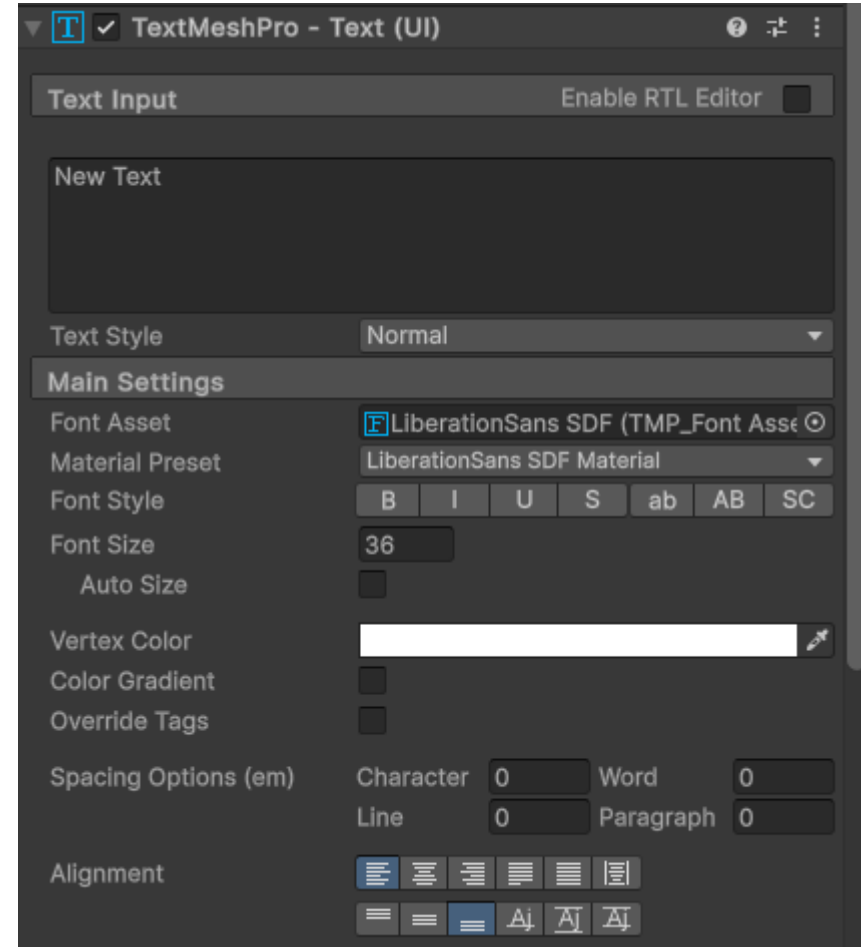


2：テキストを追加

Text[TMP]を下にいくと
“TextMeshPro”という
コンポーネントがある

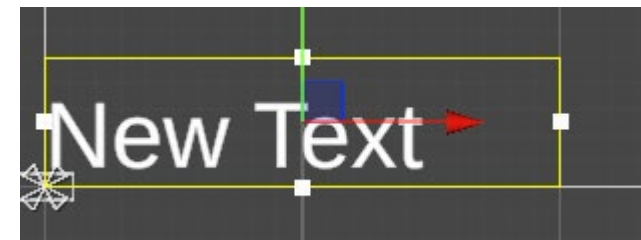
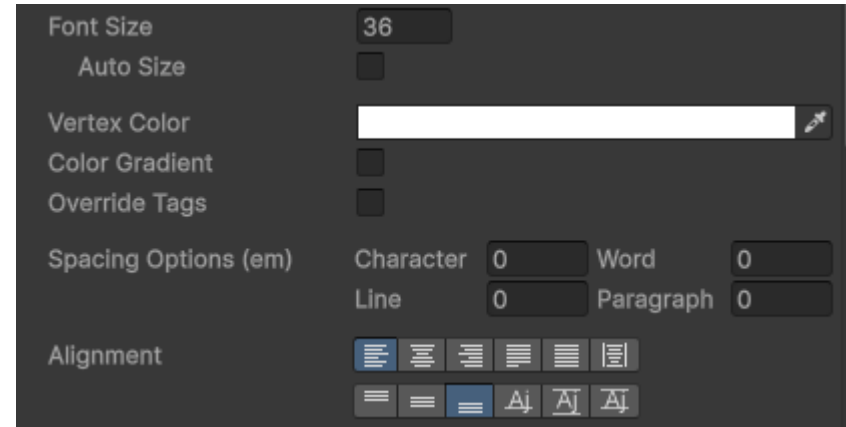
ここでテキストの詳細を
調節することができる

- 一番上のスペース
 - 表示したい文字はここに書く
- Font Asset
 - フォントを変更できる
 - フォントの素材を追加するときは[これ](#)参考に



2：テキストを追加

- Font Size
 - 文字の大きさ
- Vertex Color
 - 文字の色が変えられる
- Alignment
 - 文字の枠の中（黄色の枠）でどこに配置するか

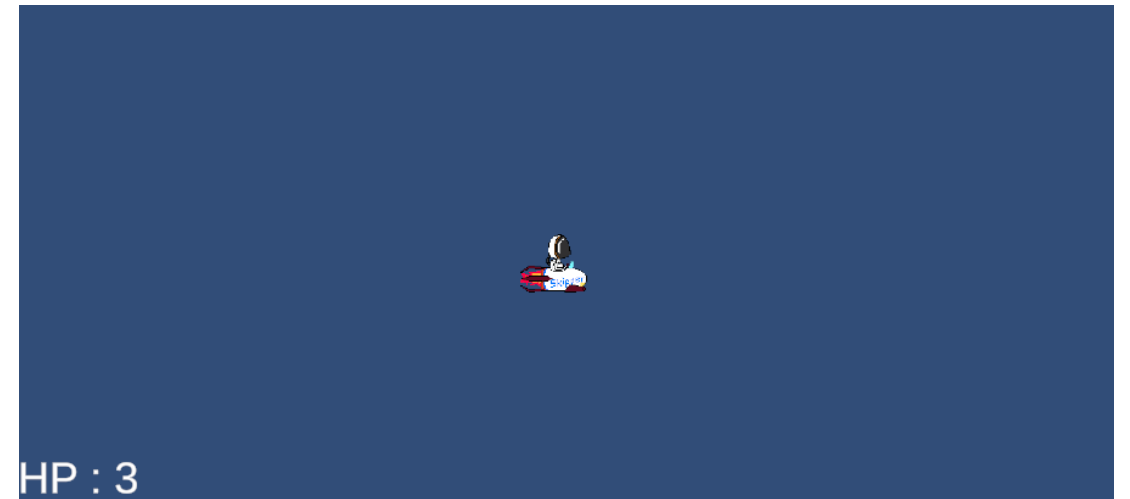
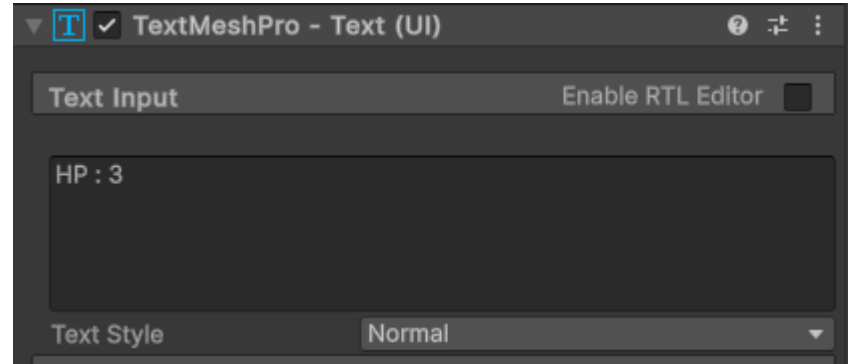


今回、Alignmentは
左端と下に配置しています（変えて検証してみよう）

2：テキストを追加

テキストを変更しよう

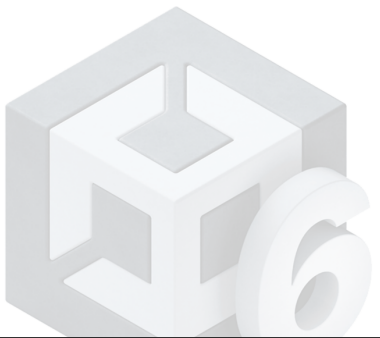
大きな枠に文字を入れて
反映させてみよう



2：テキストを追加

テキストの準備はできたが、
ダメージを受けた際にこの数を
減らしていきたいので
それを実装する

HPの管理はどこでしていたか？



2：テキストを追加

HPはPlayerControllerに
実装していたので、それを活用する

PlayerControllerを開いて
コードをうつしてみる

うつす場所をよく見ること！



```
using UnityEngine;
using TMPro;
```

```
Unity スクリプト (1 件のアセット参照) 10 個の参照
```

```
public class PlayerController : MonoBehaviour
```

```
{
    Rigidbody2D rb;
    public int speed;
    public int HP = 3;
```

```
    public Transform PointPos;
    public GameObject PlayerBeam;
```

```
    public TextMeshProUGUI HPText;
```

```
    // Start is called once before the first execution of Update after
```

```
    Unity メッセージ 10 個の参照
```

```
private void OnTriggerEnter2D(Collider2D collision)
```

```
{
    if (collision.CompareTag("EnemyBeam") || collision.CompareTag("Enemy"))
```

```
    {
        Destroy(collision.gameObject);
```

```
        HP -= 1;
```

```
        HPText.text = $"HP : {HP}";
```

```
        Debug.Log($"HP: {HP}");
```

```
        if (HP <= 0)
```

```
        {
            Destroy(gameObject);
```

```
        }
```


2：テキストを追加

TextMeshProを使う際には
一番上に

“using TMPro”

と追加しないといけない

これはTextMeshProがもともとは
Uniry公式で作られてないもので、
後からUnity公式に取り入れられたからである

```
using UnityEngine;  
using TMPro;
```

```
Unity スクリプト (1 件のアセット参照) 10 個の参照  
public class PlayerController : MonoBehaviour  
{  
    Rigidbody2D rb;  
    public int speed;  
    public int HP = 3;  
}
```



2：テキストを追加

そして
Textを変更するインスタンスの型は

“TextMeshProUGUI”

となっている

これでテキストを
変更することができる



```
public int speed;  
public int HP = 3;  
  
public Transform PointPos;  
public GameObject PlayerBeam;  
public TextMeshProUGUI HPTText;  
// Start is called once before the first execution of Update after  
Unity メッセージ 10 個の参照
```

2：テキストを追加

テキストの文字を変える場合は


`HPText.text = [右辺]`

で変更することができる

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("EnemyBeam") || collision.CompareTag("Enemy"))
    {
        Destroy(collision.gameObject);

        HP -= 1;
        HPText.text = $"HP : {HP}";

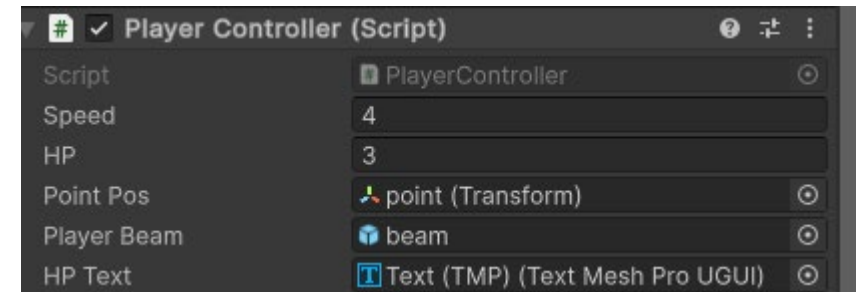
        Debug.Log($"HP: {HP}");
        if (HP <= 0)
        {
            Destroy(gameObject);
        }
    }
}
```



Q:文字と認識する際、""で囲む必要があるが
\$はどういうときに使う？

2：テキストを追加

最後にセーブをしてUnityに戻り、
PlayerのPlayerControllerを見ると
HPTextがあるので
Canvasの子オブジェクトである
"Text(TMP)"を入れてあげる



A:変数をそのまま文字として使用するため
{ }に変数を入れれば表示ができる

2：テキストを追加

これでダメージを受けたら
HPの数が減る
テキストを追加できました

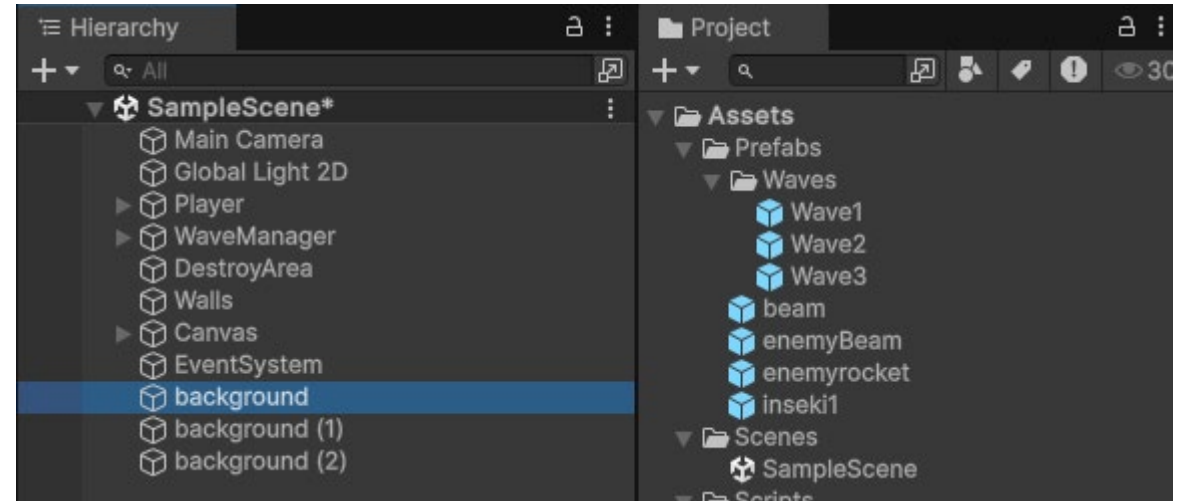


3：背景スクロール

背景がまだ実装していないので
背景スクロールを実装します

まずは
Textures/other/backgroundを
Hierarchyに投げる

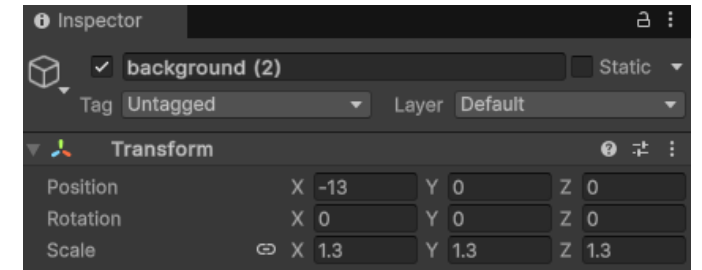
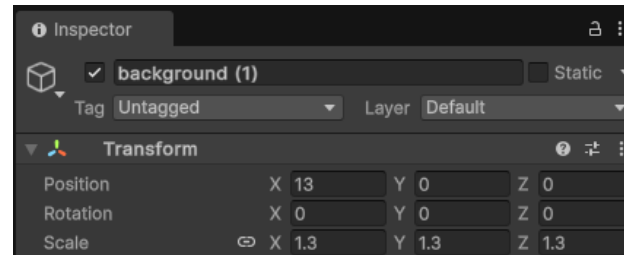
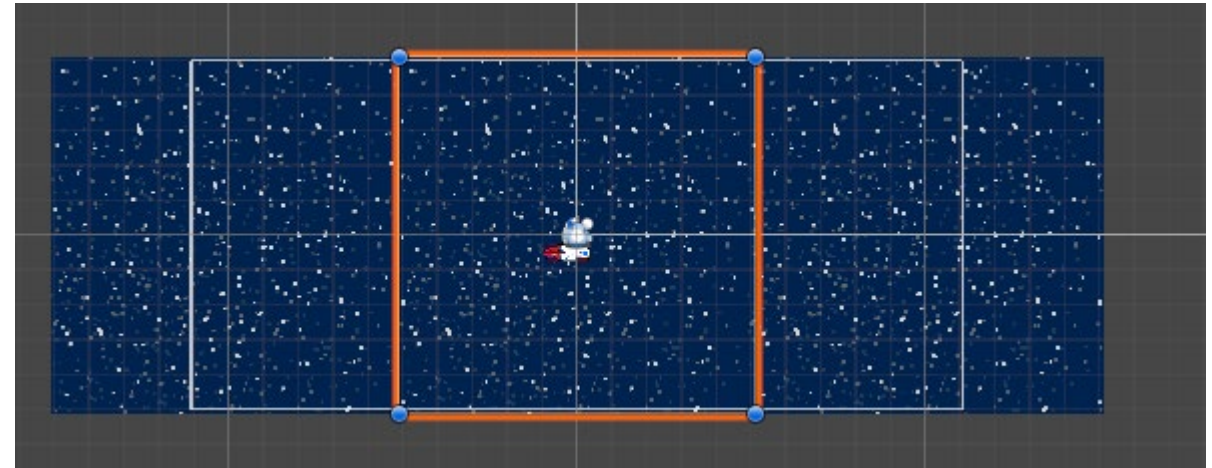
backgroundを複製する
backgroundを選択して
Ctrl + Dで複製ができる



3：背景スクロール

複製すると位置が重なっているので
座標をずらして配置する

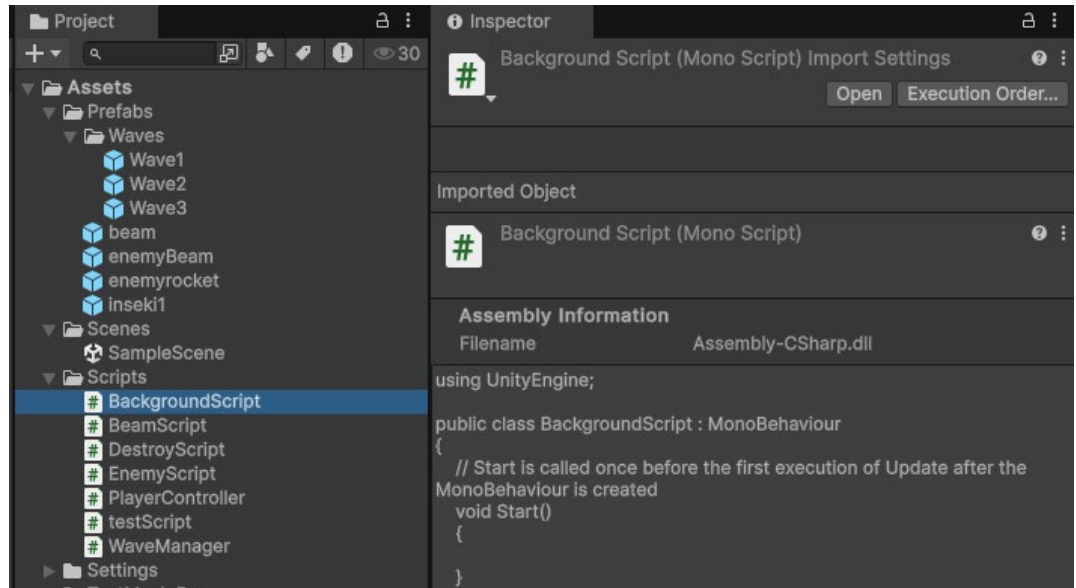
Background[1],[2]の位置を
画像の位置に配置しておき、
全てのBackgroundの
Scaleを全て1.3にする
(多分これがちょうどよさそう)



3：背景スクロール

background用のスクリプトを書く

Scriptsで右クリックで
MonoBehaviour Script
を生成し、名前を
“BackgroundScript”にする



3：背景スクロール

BackgroundScriptには

- ・ 背景の位置を毎フレーム左方向にずらす
- ・ ある程度ズレたら[今回はxが-17.5より小さかったら]
右の方に位置を再設定する

というプログラムを書く
次のページにプログラムがあるが、
考えたい人は自分でコードを考えてみよう



3：背景スクロール

こちらがBackgroundScriptです！
うっひょー！！！！



```
public class BackgroundScript : MonoBehaviour
{
    public Vector3 MoveVector;
    // Start is called once before the first execution of Update after
    // the MonoBehaviour is created
    // Unity メッセージ 10 個の参照
    void Start()
    {
    }

    // Update is called once per frame
    // Unity メッセージ 10 個の参照
    void Update()
    {
        transform.position -= MoveVector * Time.deltaTime;

        if(transform.position.x <= -17.5f)
        {
            transform.position = new Vector3(17.5f, 0, 0);
        }
    }
}
```

3：背景スクロール

位置をずらすときは
Transform.positionに
『-=』 『+=』 で変更できたが、
直接変更するときは
そのまま『=』で変更できる



```
public class BackgroundScript : MonoBehaviour
{
    public Vector3 MoveVector;
    // Start is called once before the first execution of Update after
    // the MonoBehaviour is instantiated
    // Unity メッセージ 10 個の参照
    void Start()
    {
    }

    // Update is called once per frame
    // Unity メッセージ 10 個の参照
    void Update()
    {
        transform.position -= MoveVector * Time.deltaTime;

        if(transform.position.x <= -17.5f)
        {
            transform.position = new Vector3(17.5f, 0, 0);
        }
    }
}
```

3：背景スクロール

これで背景のスクロールが
できました



4：アニメーションの追加

次にプレイヤーのイラストが
止まったままになっています

これを用意した画像を使用して
連続にイラストを表示する
アニメーションを追加します



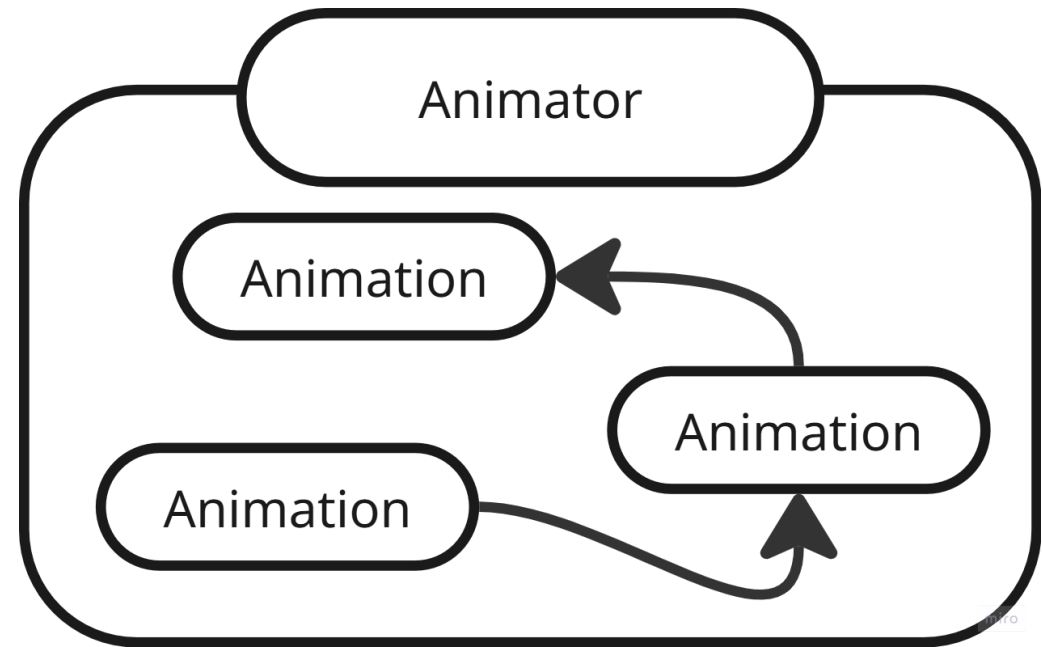
イメージ→



4：アニメーションの追加

Unityでアニメーションを
操作するためには
Animatorと
Animationを操作します

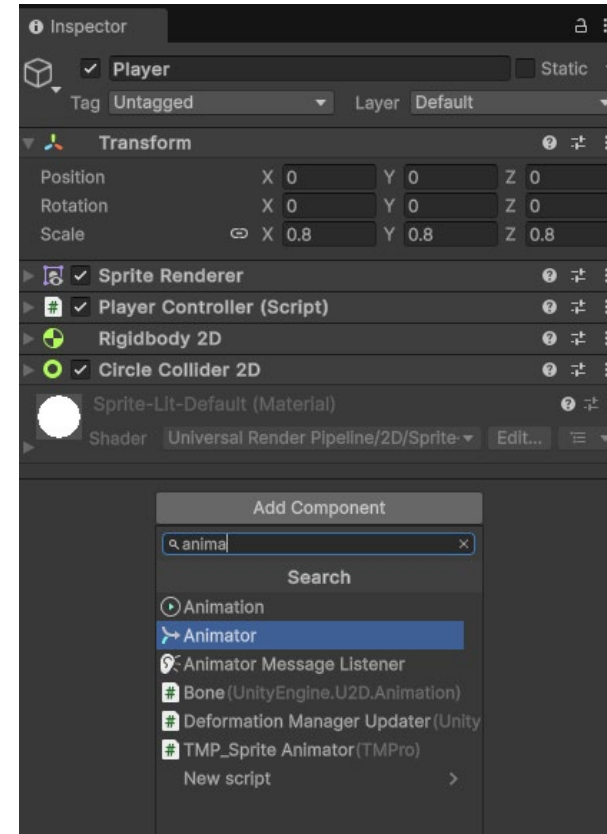
Animationはそのまま
そのオブジェクトのアニメーションを、
AnimatorはAnimationを管理する
ものです



4：アニメーションの追加

まずはAnimationを管理するための
AnimatorをPlayerに追加する

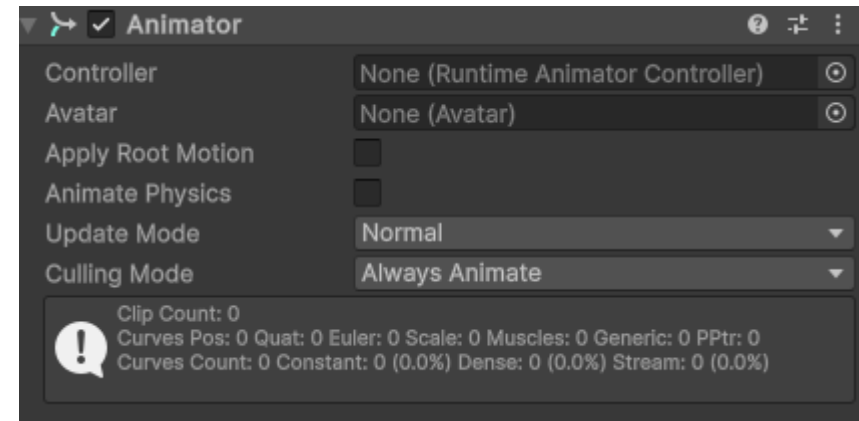
Playerを選択し、
AddComponentから
“Animator”を追加する



4：アニメーションの追加

追加するとこんなのが出てきます

“Controller”がアニメーションを管理するもので、AnimatorはこのControllerの中身を見て管理をしています



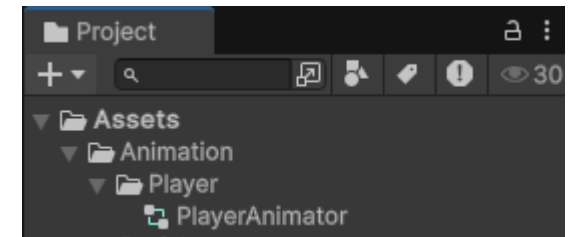
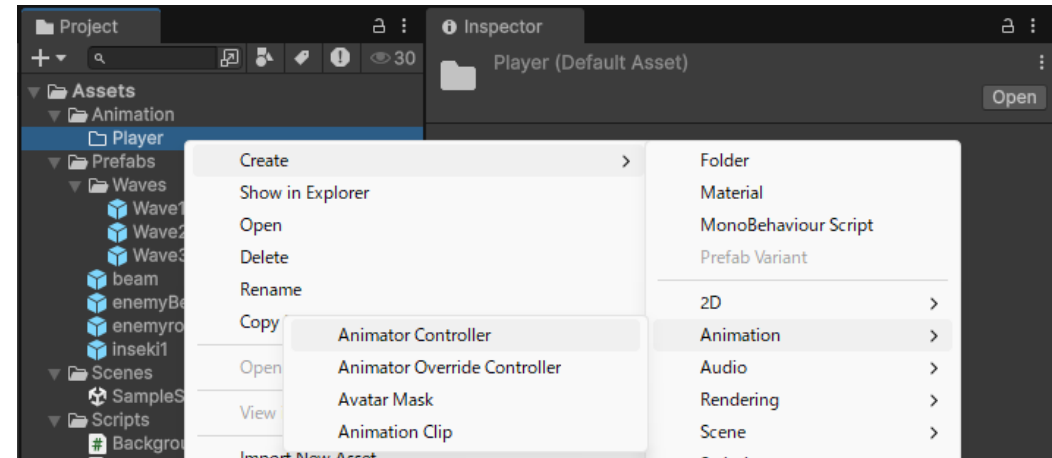
4：アニメーションの追加

ではControllerを追加しよう

Animationフォルダと
その中にPlayerフォルダを作成

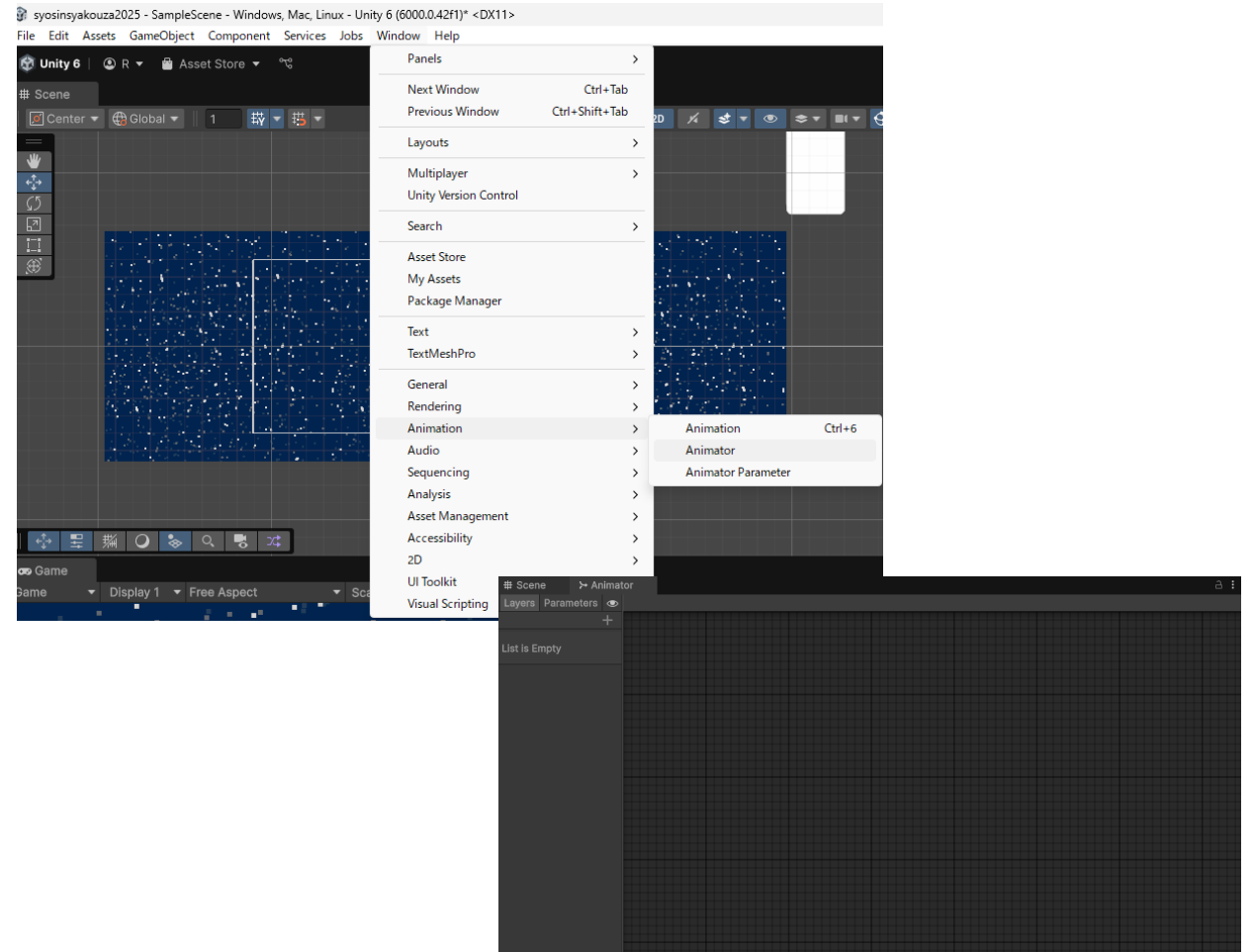
その後、Playerフォルダで右クリックし
Create/Animation/Animator Controllerを追加

名前は”PlayerAnimator”にしておく



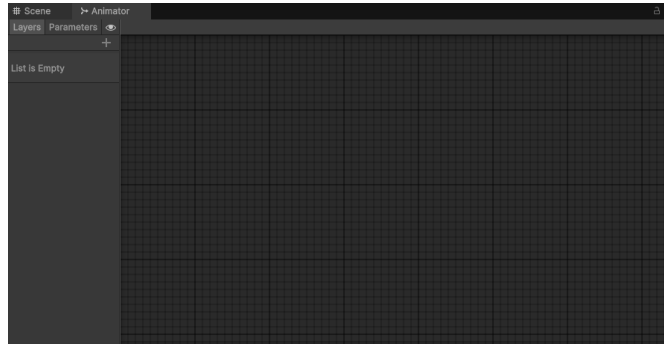
4：アニメーションの追加

**Controllerを追加したら
一番上の
Windows/Animation/Animatorを
クリックし、
Animatorタブを追加する**

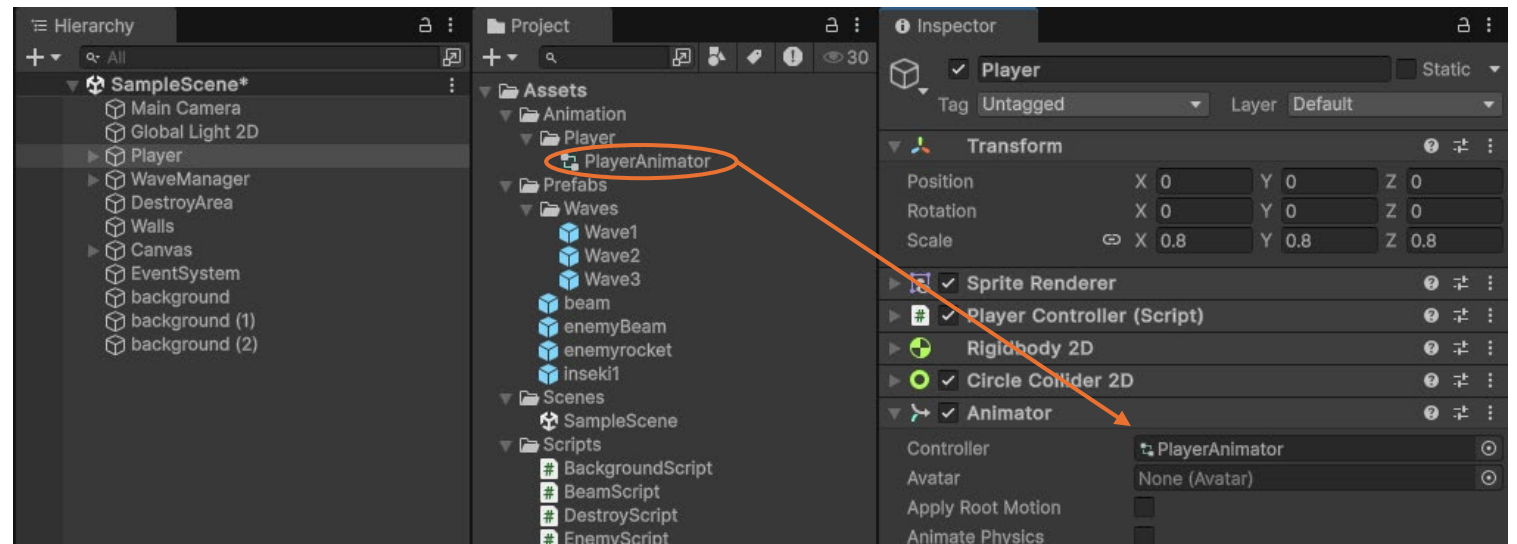


4：アニメーションの追加

Animatorを追加すると
何も無い画面になる



PlayerのAnimatorに
作ったPlayerAnimatorを
Controllerに入れると
少し中身が追加される

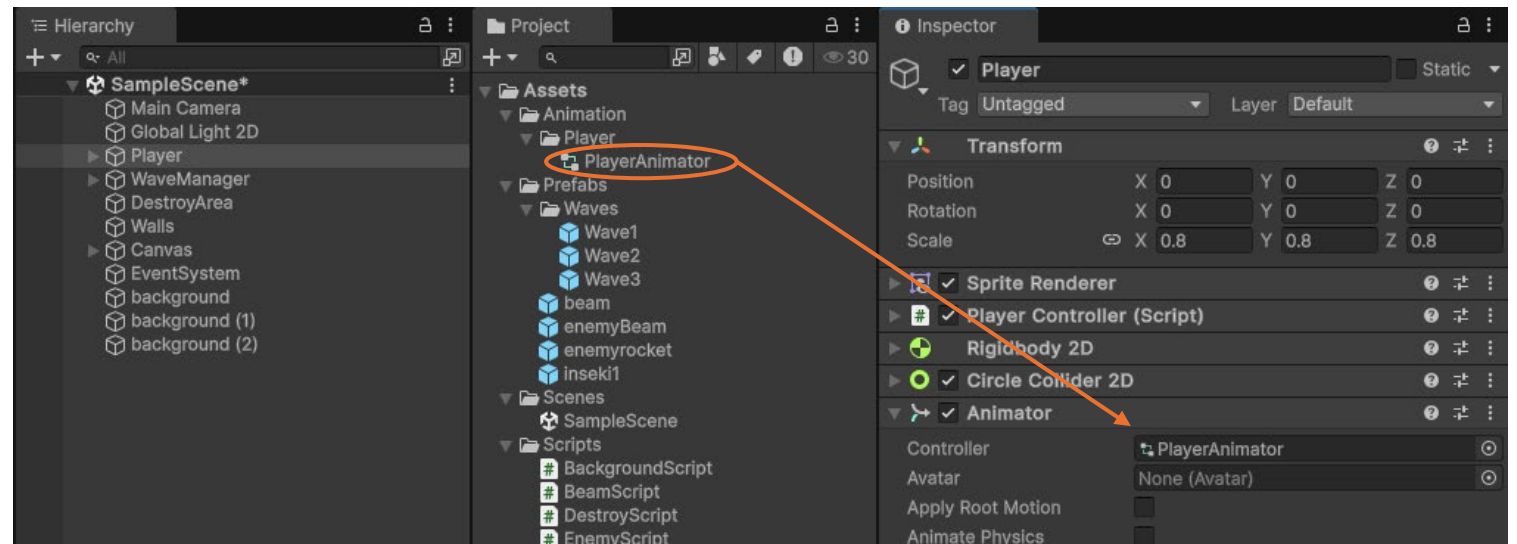


4 : アニメーションの追加

Animatorを追加すると
何も無い画面になる

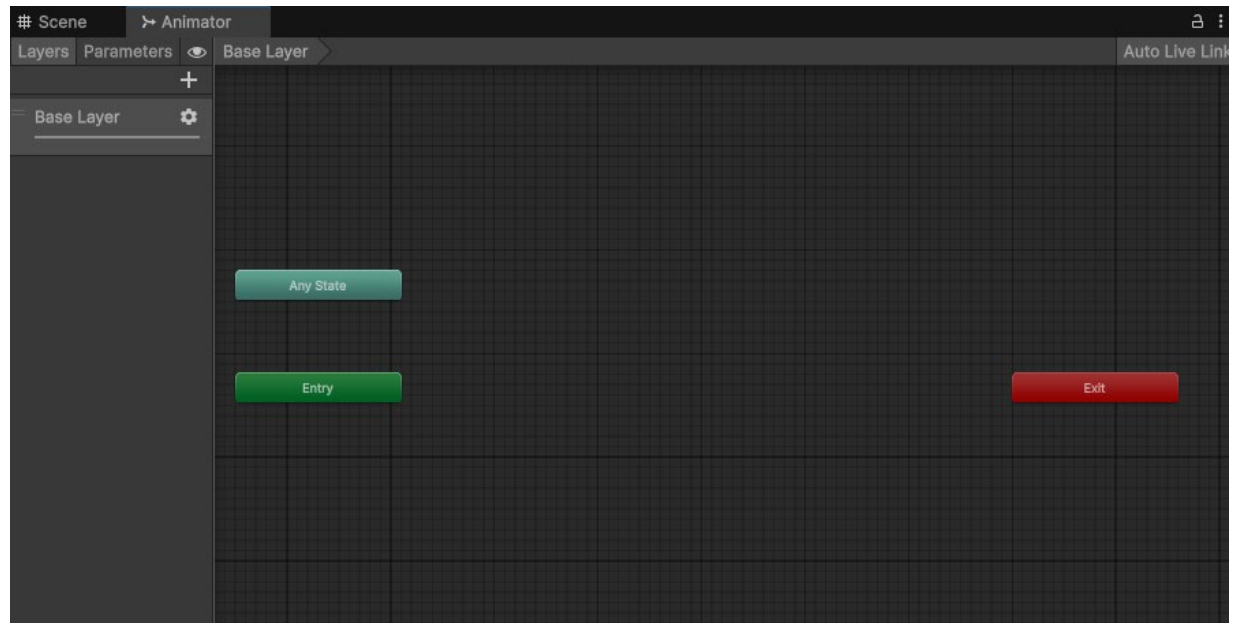


PlayerのAnimatorに
作ったPlayerAnimatorを
Controllerに入れると
少し中身が追加される



4：アニメーションの追加

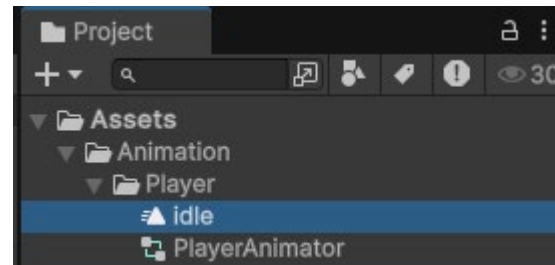
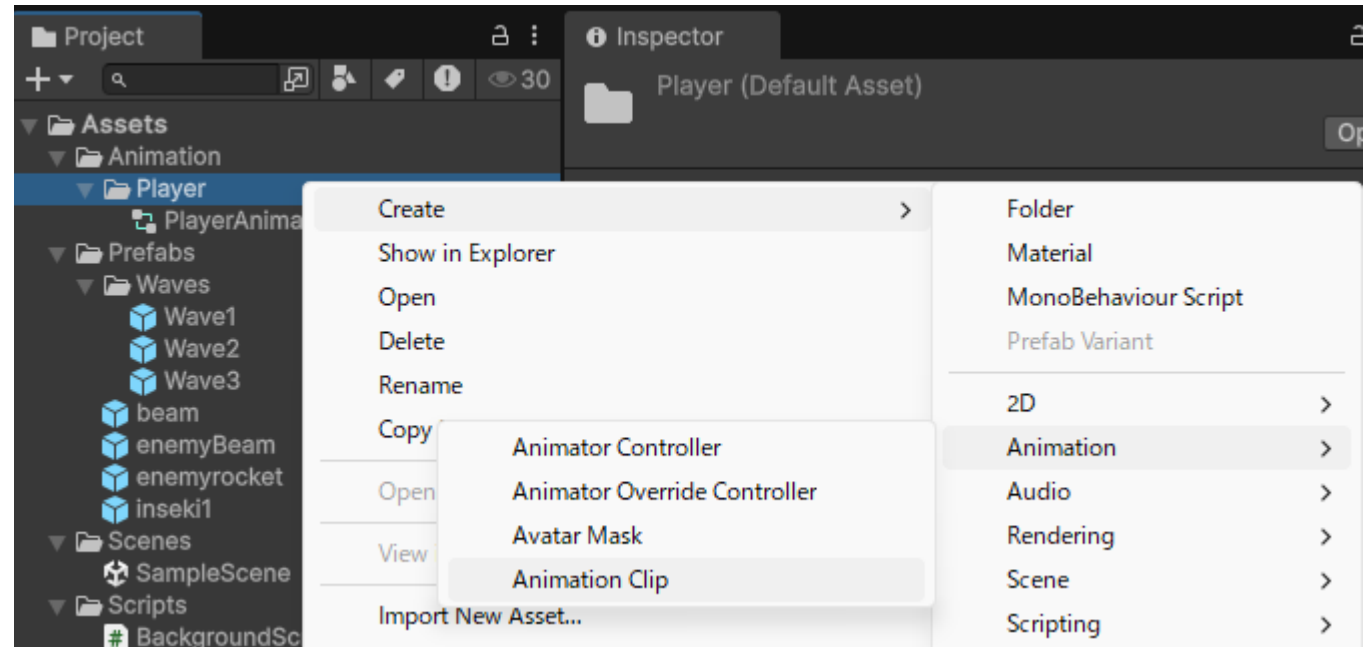
こんな感じの見た目になる
これがAnimatorで、
ここに1つ1つの
アニメーションを組み立てていく



4：アニメーションの追加

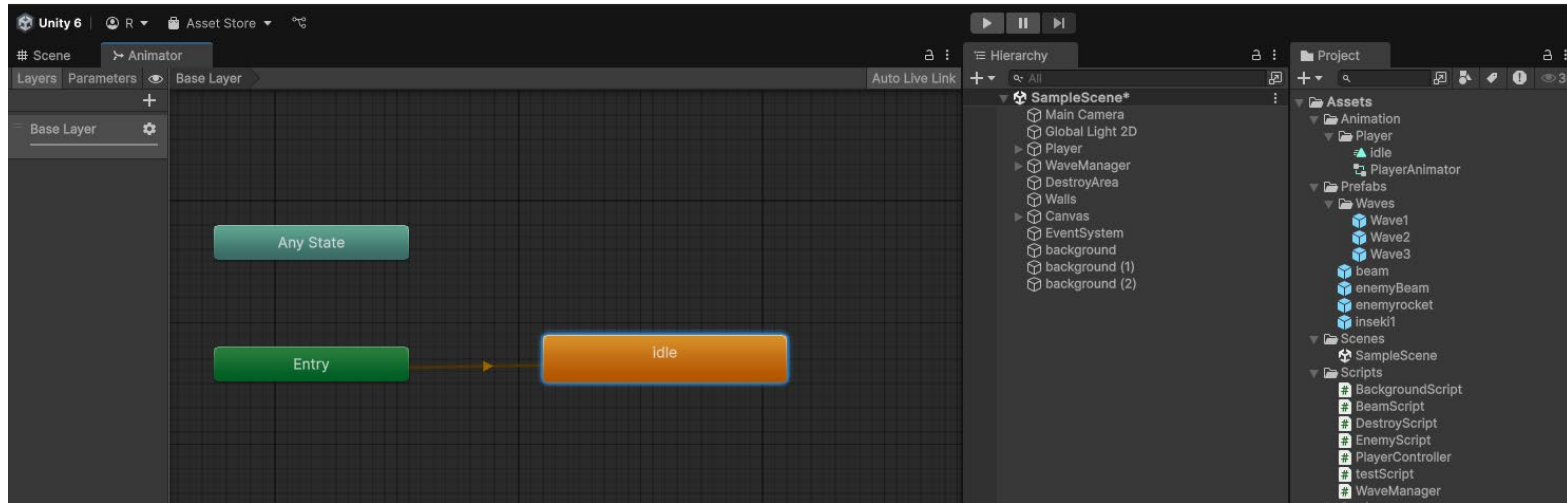
次にアニメーションを追加する
Animation/Playerに
右クリックして
Create/Animation/Animation Clip
を追加する

名前は"idle"にする



4：アニメーションの追加

作ったらAnimatorタブに
作った”idle”を
ドラック&ドロップ



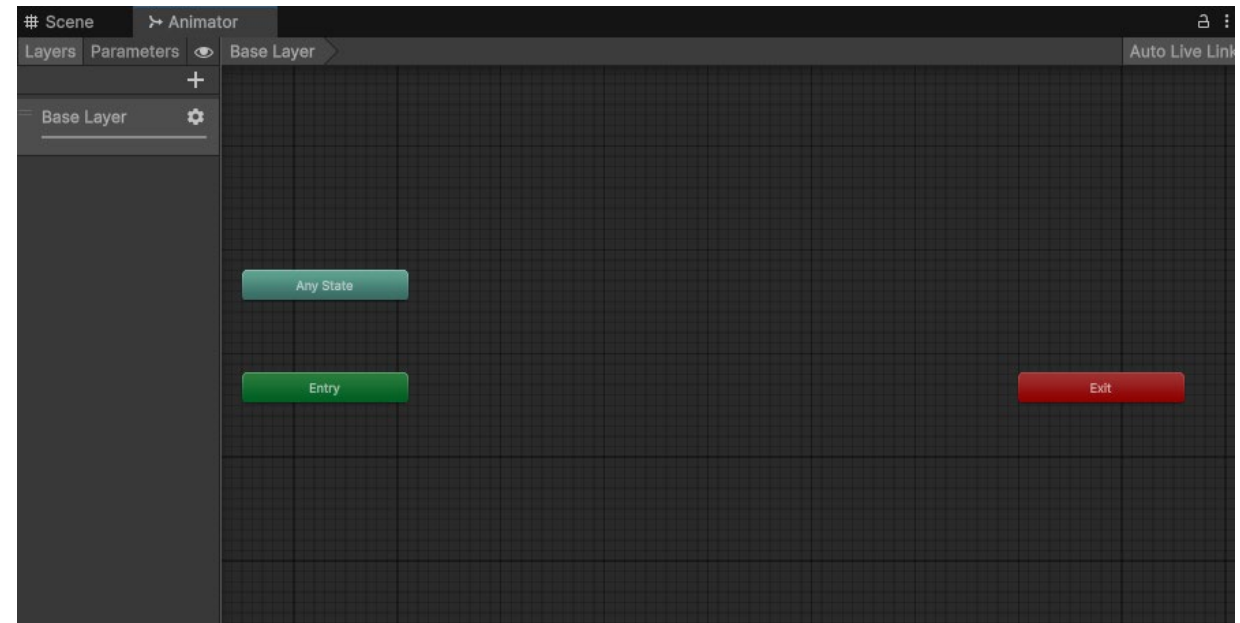
そうすると
Entry → idleにアニメーションがつながる



4：アニメーションの追加

元々おいてあったものについて
軽く解説すると、

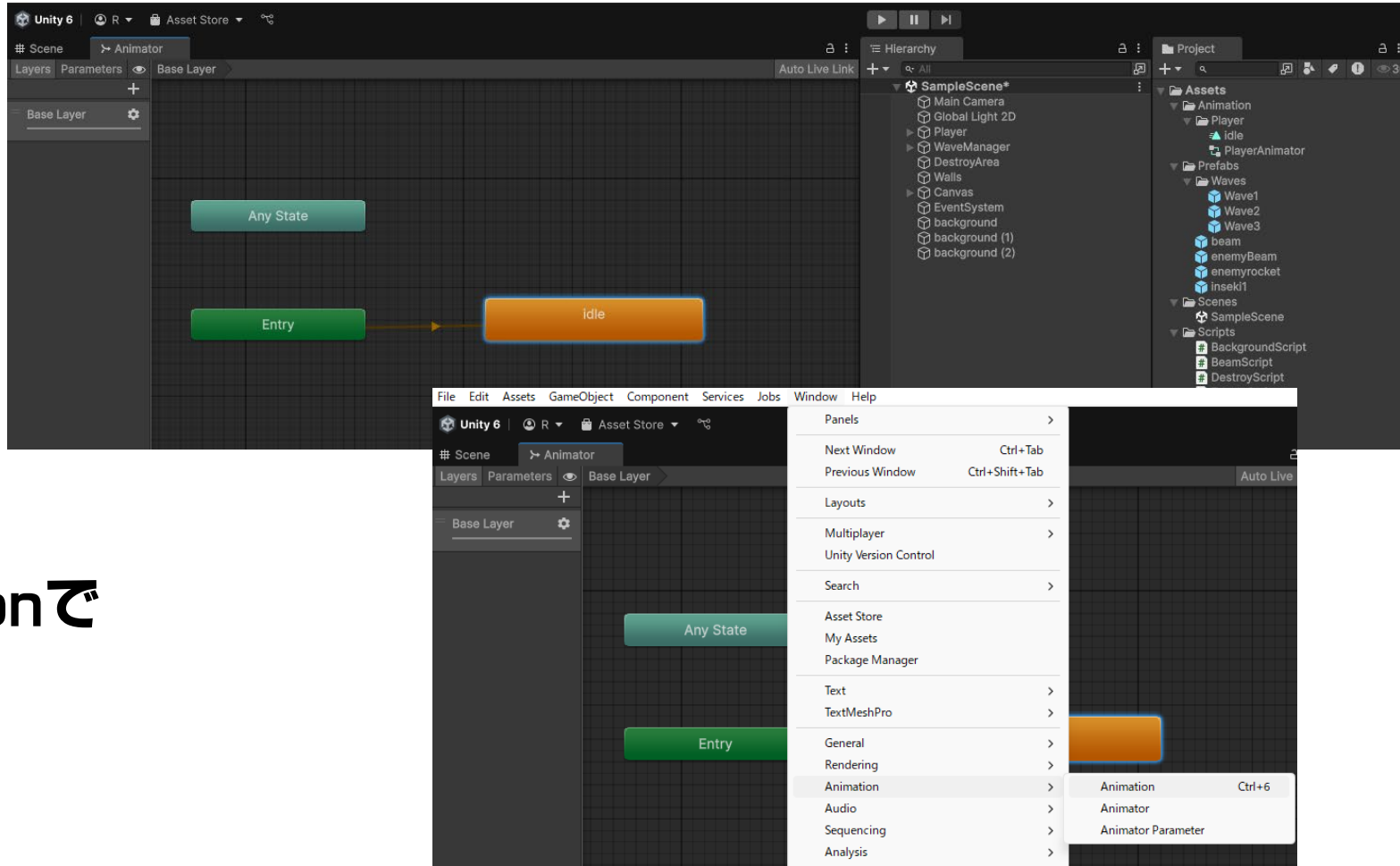
- Entry
 - ゲームを再生し始めたときの入り口
- AnyState
 - どんな状況でも繋がられるもの
- Exit
 - 矢印をつなげると
アニメーションの再生が停止する



4：アニメーションの追加

Idleにはつながりましたが
Idleの中身はまだないので
それを設定します

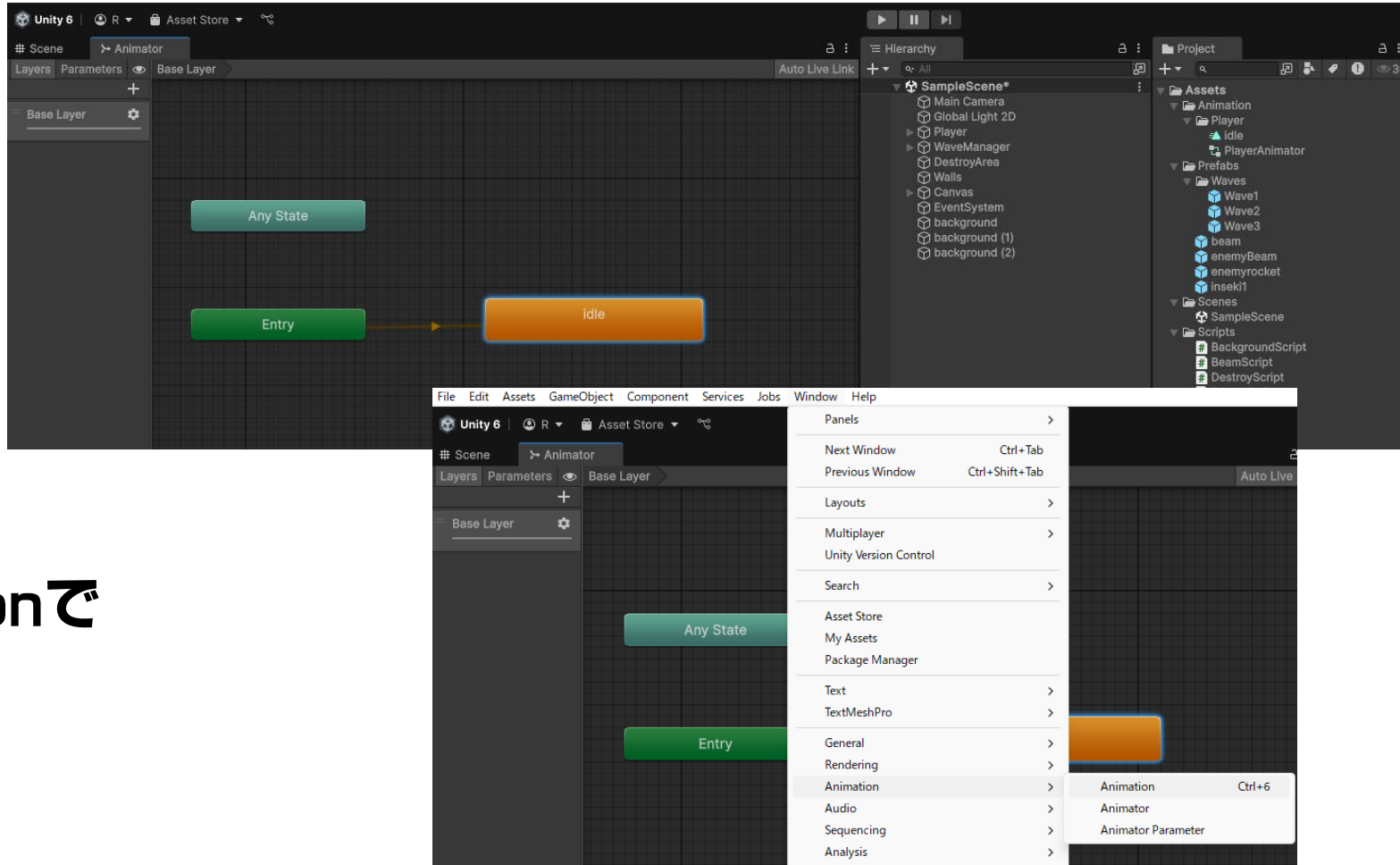
Window/Animation/Animationで
Animationタブを追加



4：アニメーションの追加

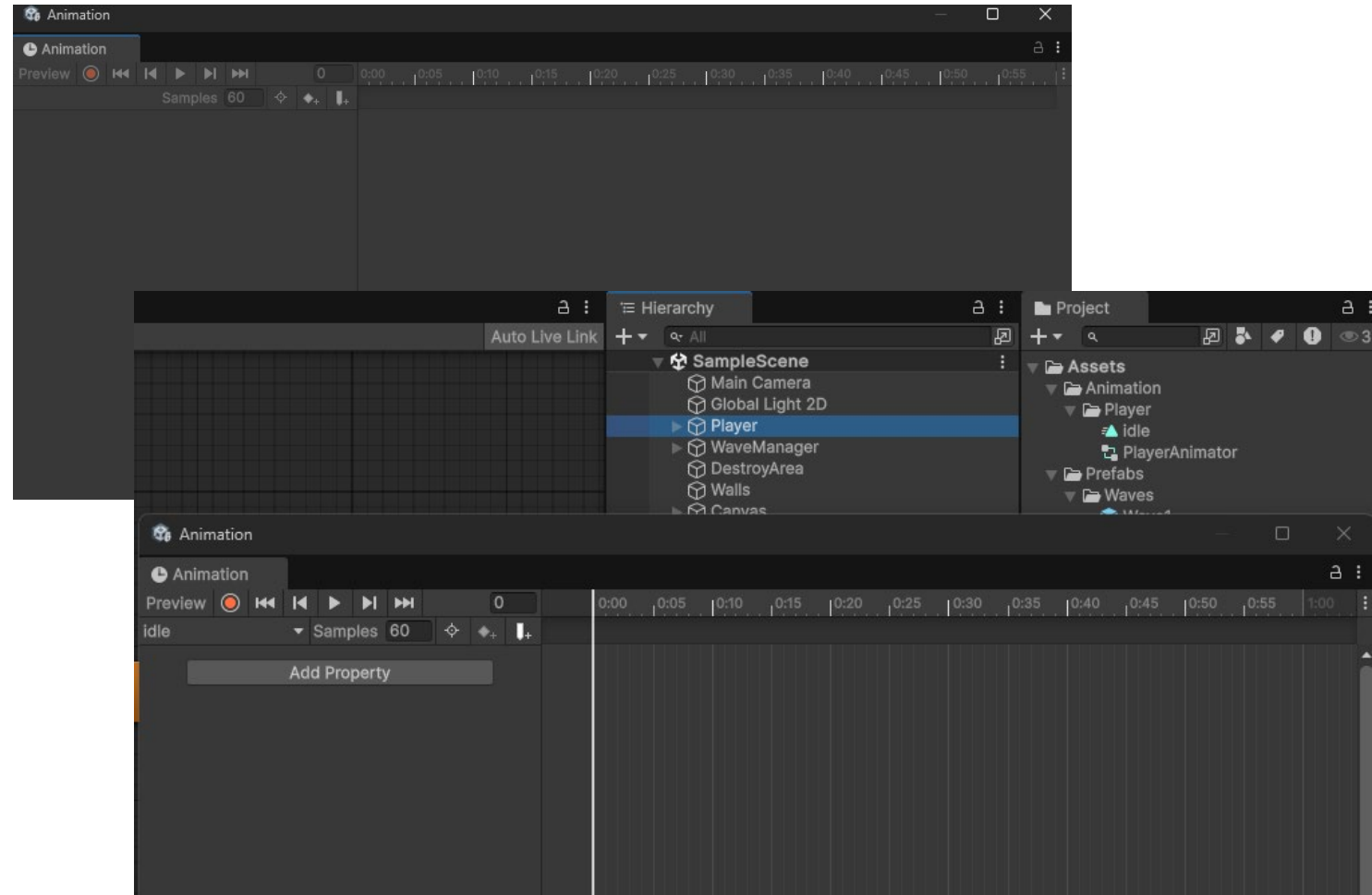
Idleにはつながりましたが
Idleの中身はまだないので
それを設定します

Window/Animation/Animationで
Animationタブを追加



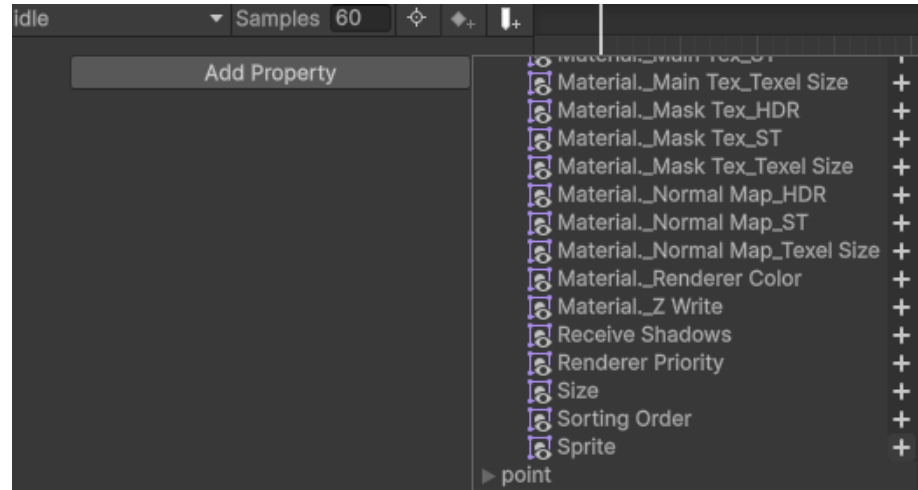
4：アニメーションの追加

見ると中身が何もありませんが、
Playerを選択すると
PlayerのAnimatorの
一番初めに再生される
“idle”が見れます

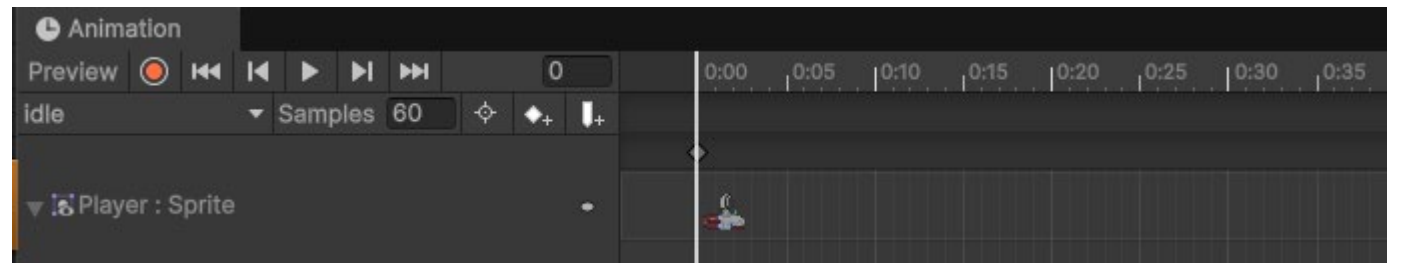


4：アニメーションの追加

とりあえず設定をします
“Add Property”から
Sprite Renderer/Spriteを選択



するとAnimationに
Spriteが出てきます

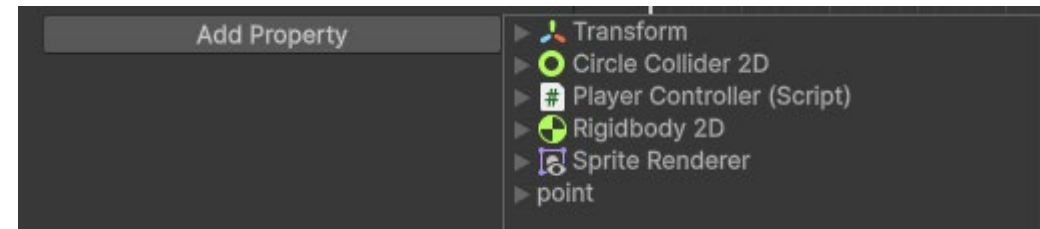


4：アニメーションの追加

“Add Property”では
オブジェクトのどの要素を
アニメーションにするかを設定できる

要素の一覧には
そのオブジェクトについている
コンポーネントが出てくる

「そのコンポーネントの
どの部分を使用するか？」
を選択している感じ
他にも子オブジェクトもここに出てくる

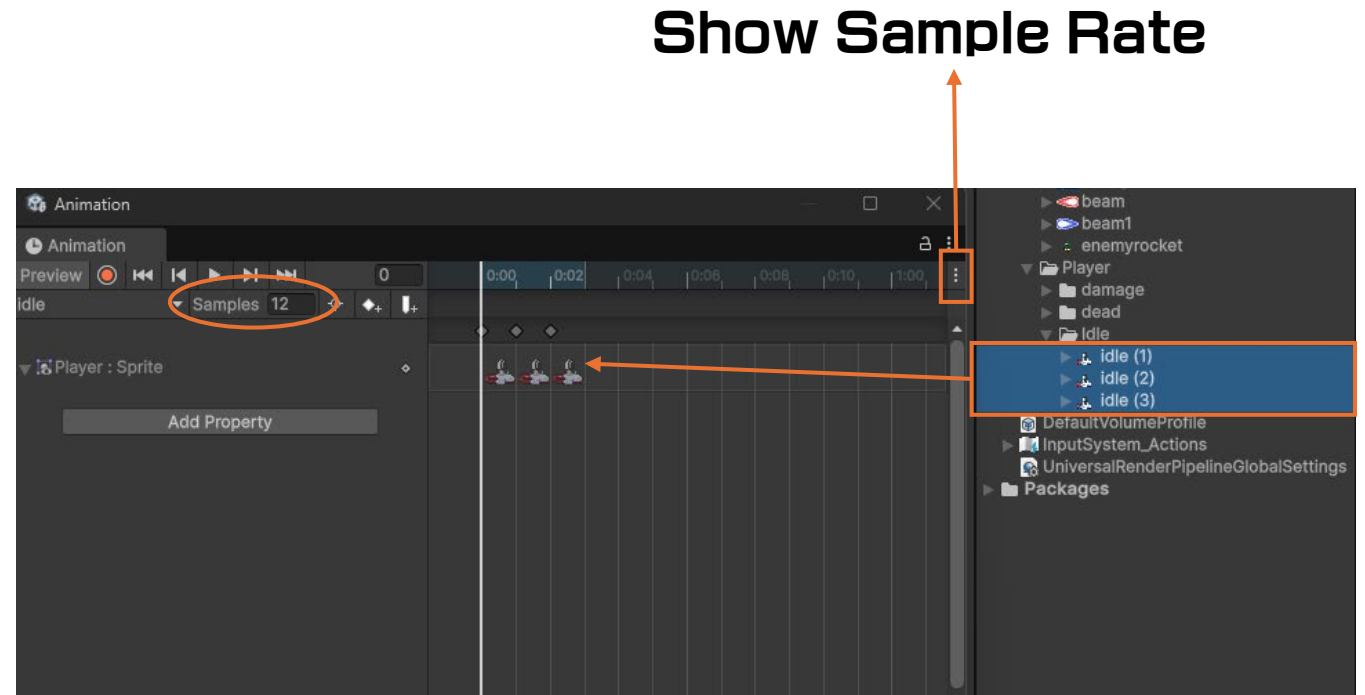


4：アニメーションの追加

アニメーションの設定をする

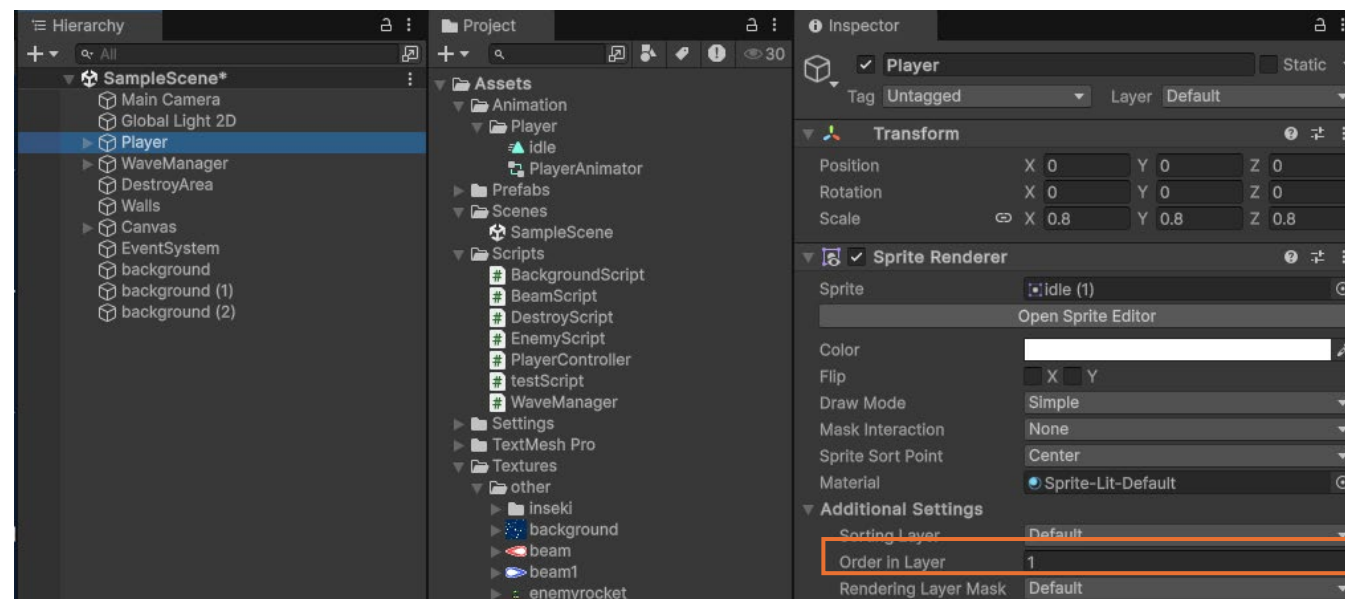
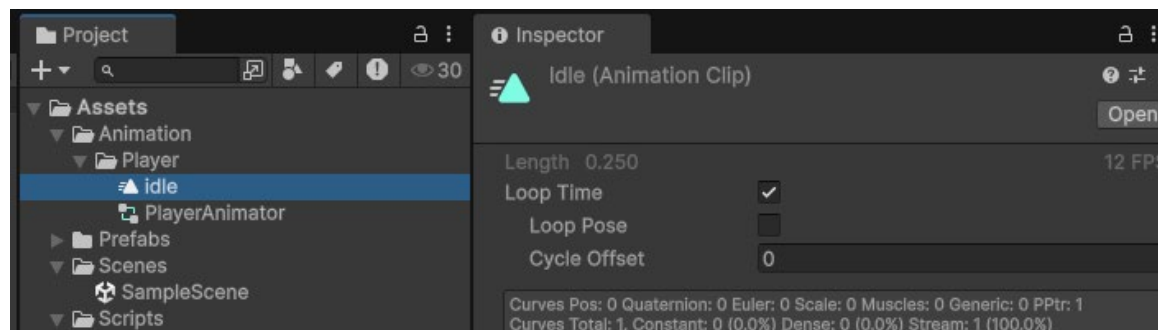
時間の右にある三つの点から
“Show Sample Rate”をオン
出てくる“Samples”を12にする

その後、
Textures/player/idleにある
3つの画像をドラック&ドロップ
(idle 1 を選択してシフトを押しながらidle3をクリックすると
複数選択が可能！)



4：アニメーションの追加

最後に
Idleを見て
“Loop Time”をオンにし、
PlayerのSpriteRendererの
“Order in Layer”を1にすれば
アニメーションは終わり



5：ゲームオーバー

ラストに、
プレイヤーのHPが0になったら
ゲームオーバーを表示するようにします

ゲームオーバーの表示はテキストで
“ゲームオーバー”と文字を表示させればOKです

では、文字ってどうやって作ったのか？

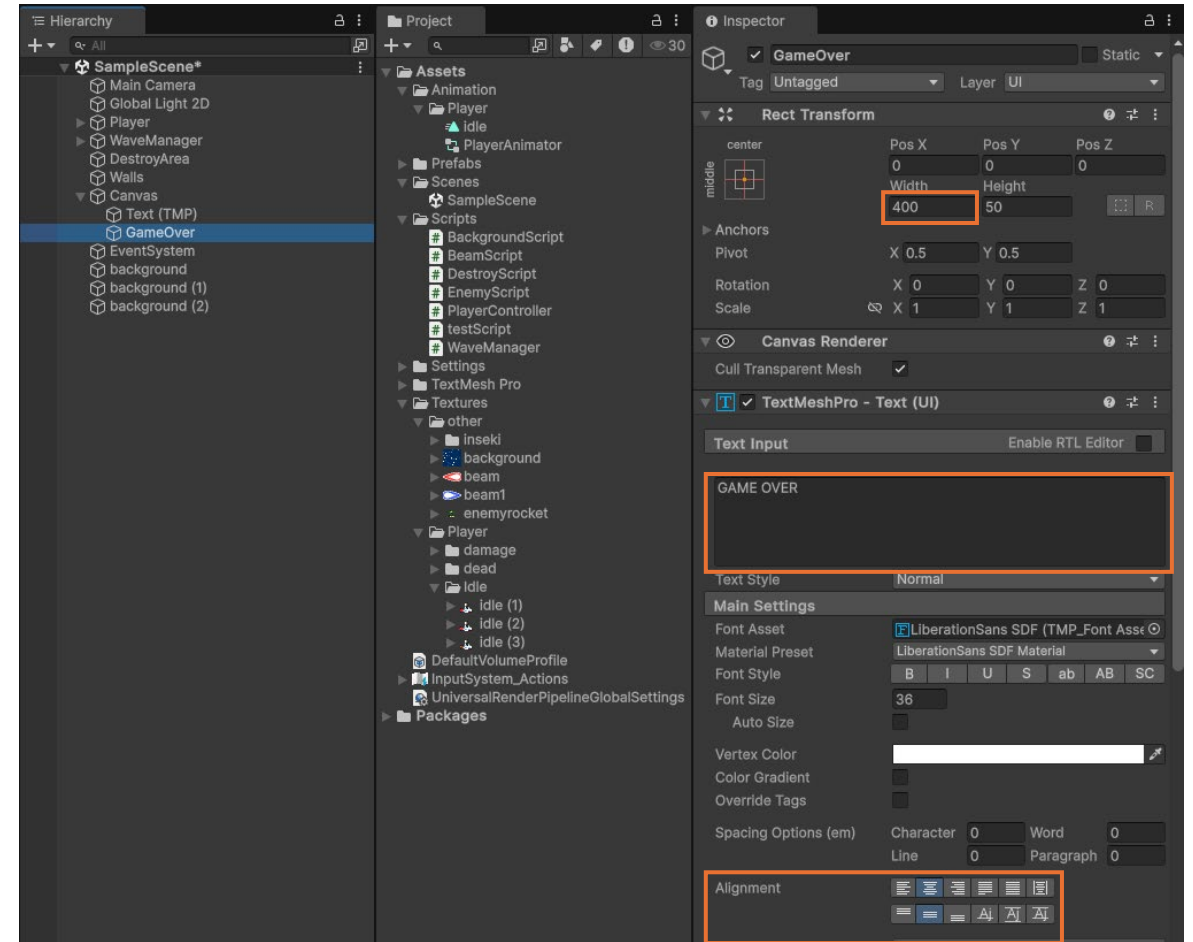


5：ゲームオーバー

文字はUIから作成した

Canvasを右クリックし、
UI/Text – TextMeshProをクリック

作成したら名前を”GameOver”にし、
画像のように設定をする

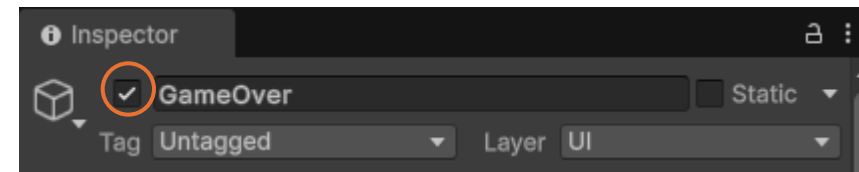
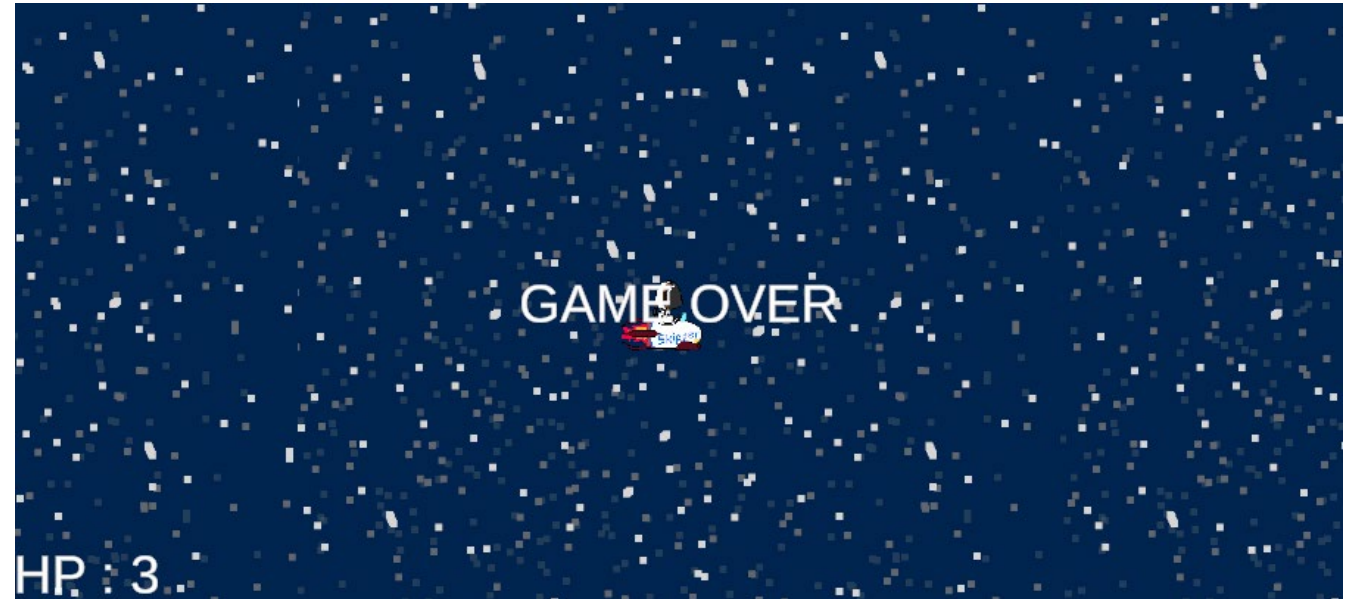


5：ゲームオーバー

Gameビューは
こんな感じになるかと
思います

色とか変えてもらってOK

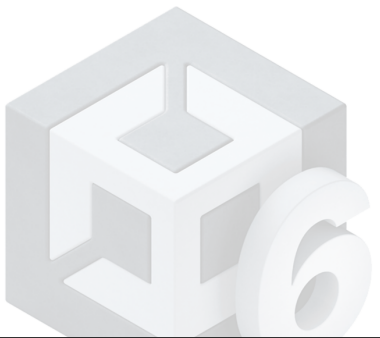
設定できたらこの☑を外そう
見えなくなるはずだ



5：ゲームオーバー

プレイヤーのHPが0になったら
ゲームオーバーを表示したい

そうしたらプレイヤーのHPが
0になるときのコードは
どこに書いたのだろうか？



5 : ゲームオーバー

PlayerControllerの
OnTriggerEnterに書いたはず

HPを減らしていったら
0以下になったら削除される
設定だったはずだ



```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("EnemyBeam") || collision.CompareTag("Enemy"))
    {
        Destroy(collision.gameObject);

        HP -= 1;
        HPText.text = $"HP : {HP} ";

        Debug.Log($"HP : {HP} ");
        if (HP <= 0)
        {
            Destroy(gameObject);
        }
    }
}
```

5 : ゲームオーバー

HPが0になるときに
文字を出すコードと
その文字を指定するインスタンスを
宣言させよう

右の箇所の通りに
コードをうつそう



```
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rb;
    public int speed;
    public int HP = 3;

    public Transform PointPos;
    public GameObject PlayerBeam;

    public TextMeshProUGUI HPText;
    public TextMeshProUGUI GameOverText;
    // Start is called once before the first execution
    // Unity メッセージ 10 個の参照
    void Start()
    {
```

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("EnemyBeam") || collision.CompareTag("Enemy"))
    {
        Destroy(collision.gameObject);

        HP -= 1;
        HPText.text = $"HP : {HP}";

        Debug.Log($"HP: {HP}");
        if (HP <= 0)
        {
            GameOverText.gameObject.SetActive(true);
            Destroy(gameObject);
        }
    }
}
```

5：ゲームオーバー

gameObject.SetActive()

これはそのオブジェクトを
表示/非表示にするコードである
Trueを指定したら表示、
Falseを指定したら非表示

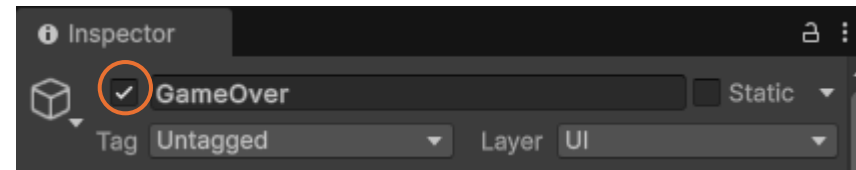
先ほど、☒を外すと非表示になったが
この動作をコードで行っている



```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("EnemyBeam") || collision.CompareTag("Enemy"))
    {
        Destroy(collision.gameObject);

        HP -= 1;
        HPText.text = $"HP : {HP}";

        Debug.Log($"HP: {HP}");
        if (HP <= 0)
        {
            GameOverText.gameObject.SetActive(true);
            Destroy(gameObject);
        }
    }
}
```



5：ゲームオーバー

最後に、
PlayerControllerの
“GameOverText”に
GameOverのテキストの
オブジェクトを入れる

ゲームオーバーが実装できた！！！！



6：最後に

完成したゲームの動画を
ここに乗せようかと思いますが、
もう少しあとにします…



6：最後に

とりあえずは、ゲーム制作お疲れさまでした！！
ざっと解説していきましたが、
正直覚えるところはたくさんあります

覚えたところで、それをどうやって組み立てていくか？と
考えていくところがゲーム開発の面白い部分だと
思いますので、これをきっかけに
Unityのゲーム開発を楽しんでみてください！！



6：最後に

まだできていない要素としては

- ・ 音
- ・ 設定画面

が候補になりますが、
それは気になったら個人で調べてみてください…！

再度重ねて申し上げますが、
お疲れさまでした！！！！





Unity 初心者講座

2 導入

初歩

ここからは補足です