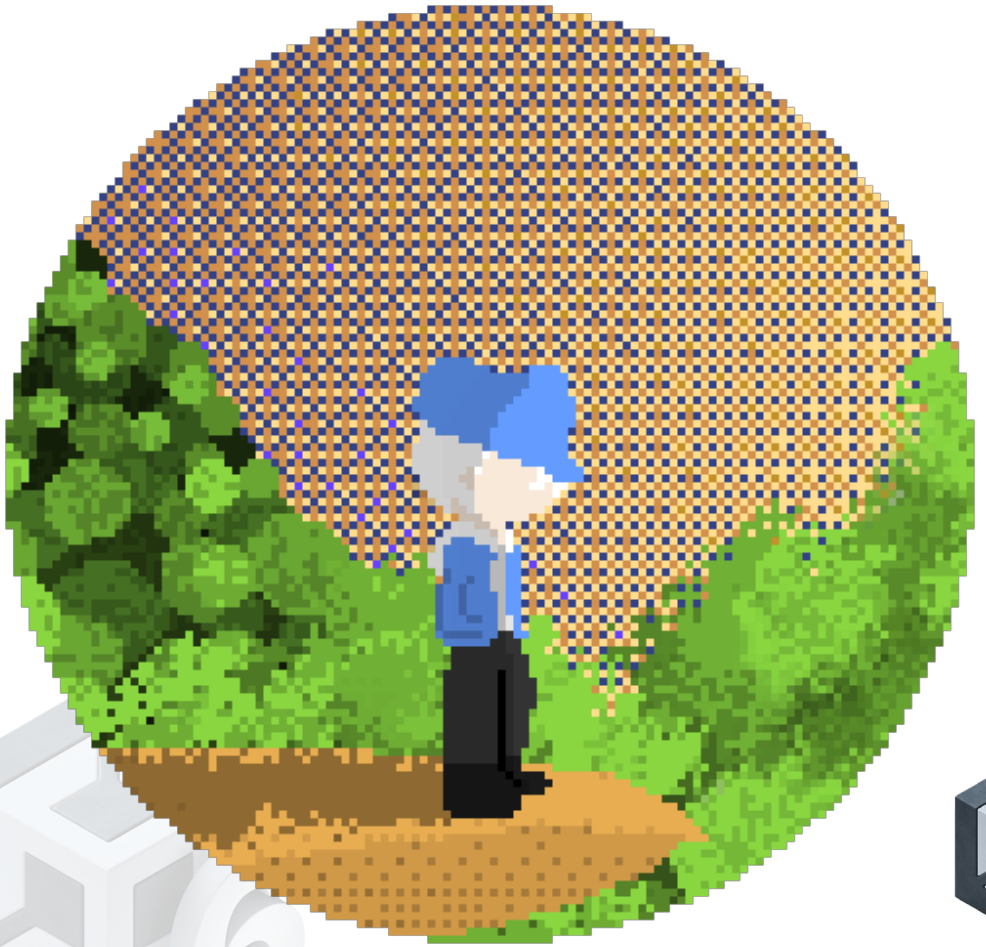
Unity初心者講座

2.敵の導入

自己紹介



21st/部長/インフラ

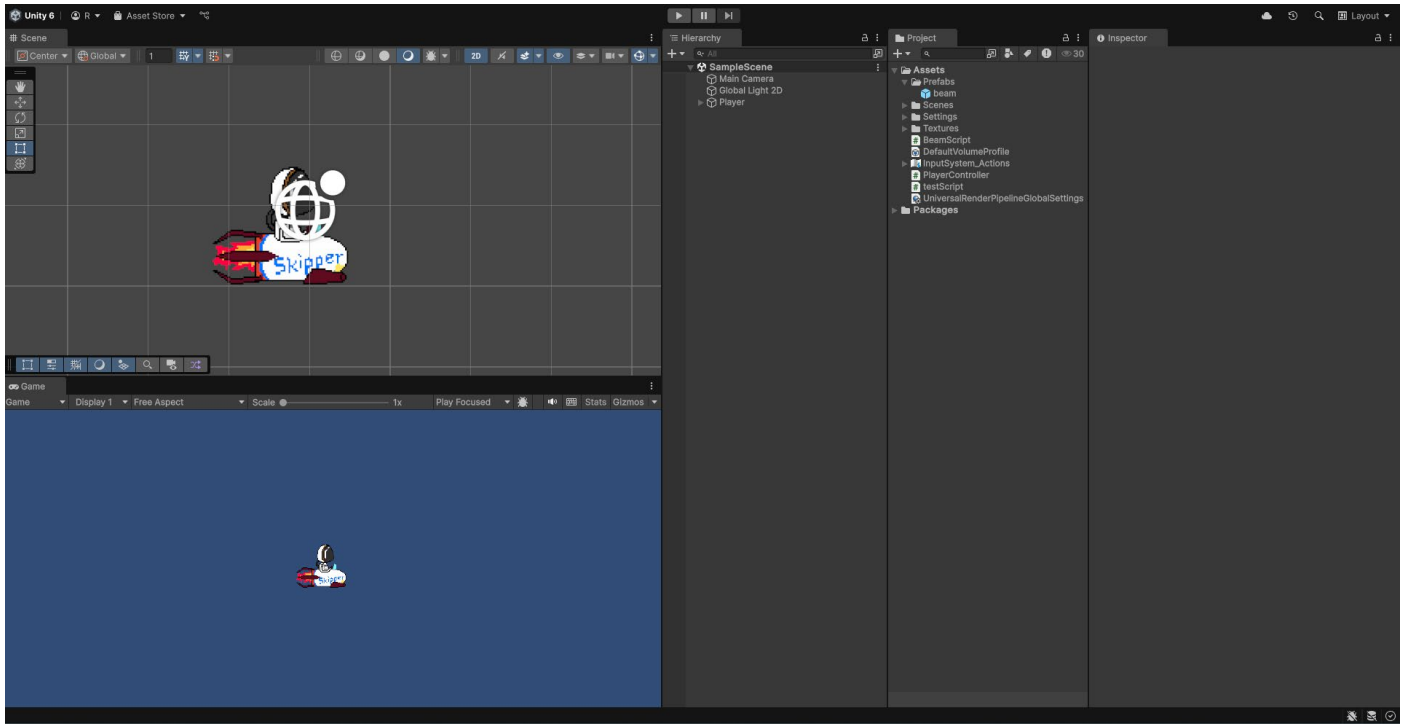
ロペ

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

概要

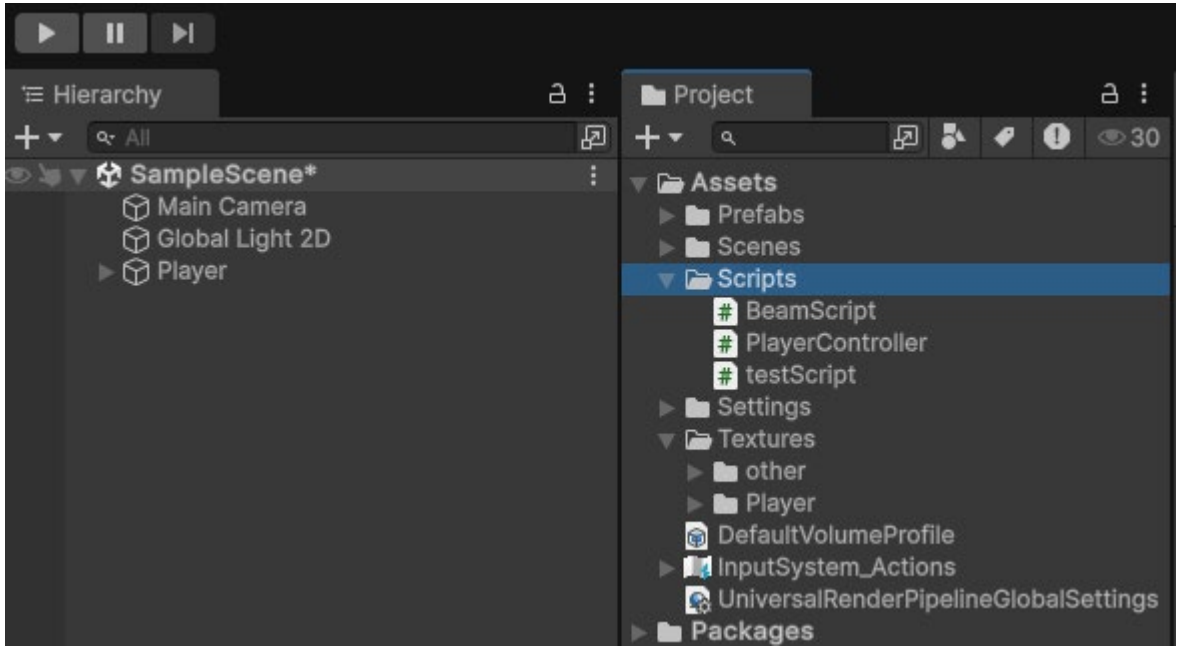
1：敵の追加

前回作成したプジェクトを
UnityHubで開きなおす



1：敵の追加

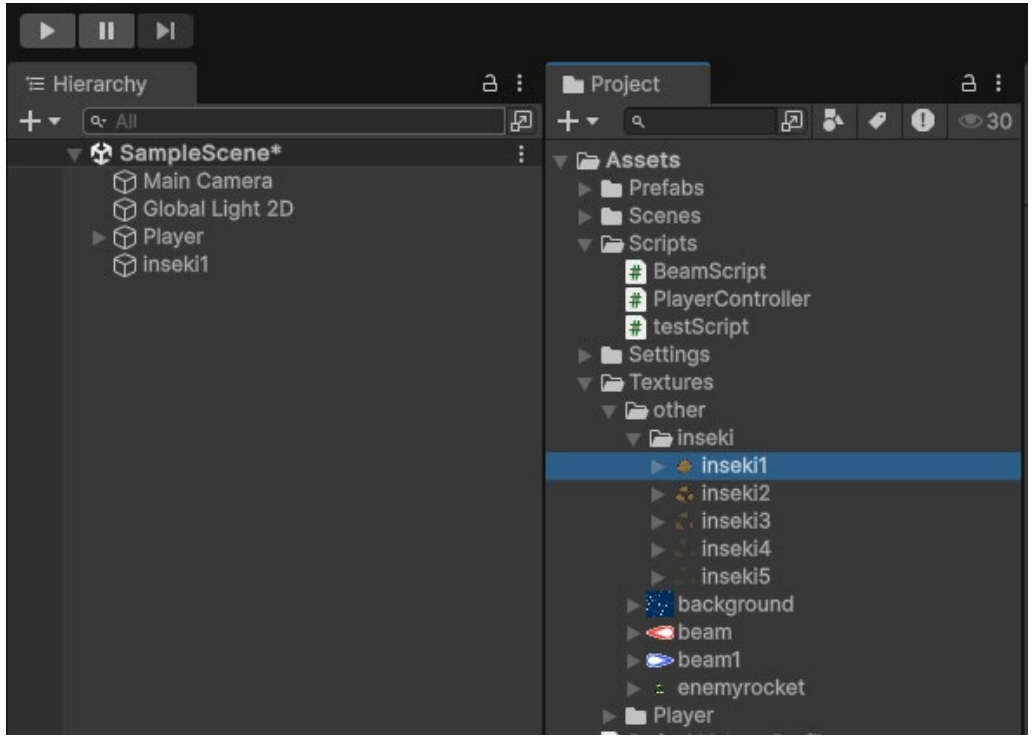
前まで作ってスクリプトが
Projectタブに
散らばっているので
“Scripts”フォルダを作成し
スクリプトを全て放り込んでおく



１：敵の追加

まずは攻撃しない敵（隕石）を追加する

Textures/other/inseki/inseki1を Hierarchyに投げる



1：敵の追加

EnemyScriptの中を
右のコードにうつす

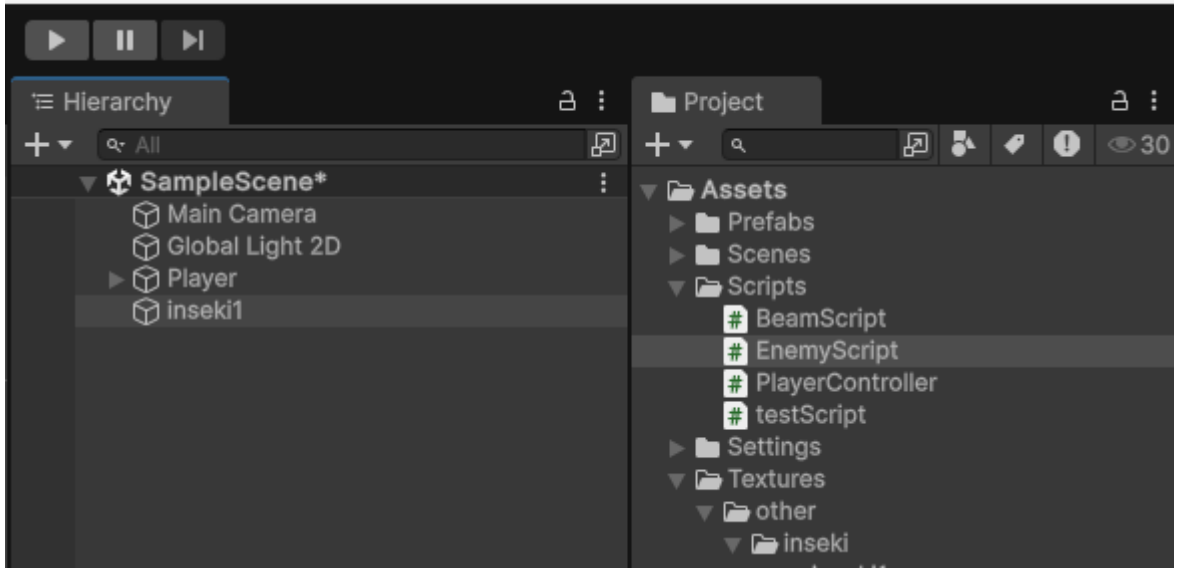
これは今だけは
BeamScriptとまったく同じもの
（後で改変する）

```
public class EnemyScript : MonoBehaviour
{
    public Vector3 MoveVector;
    // Start is called once before the first execution of Update after the MonoBehaviour
    Unity メッセージ | 0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        transform.position += MoveVector * Time.deltaTime;
    }
```

1：敵の追加

うつしてセーブしたら
EnemyScriptをinseki1に
ドラック&ドロップする

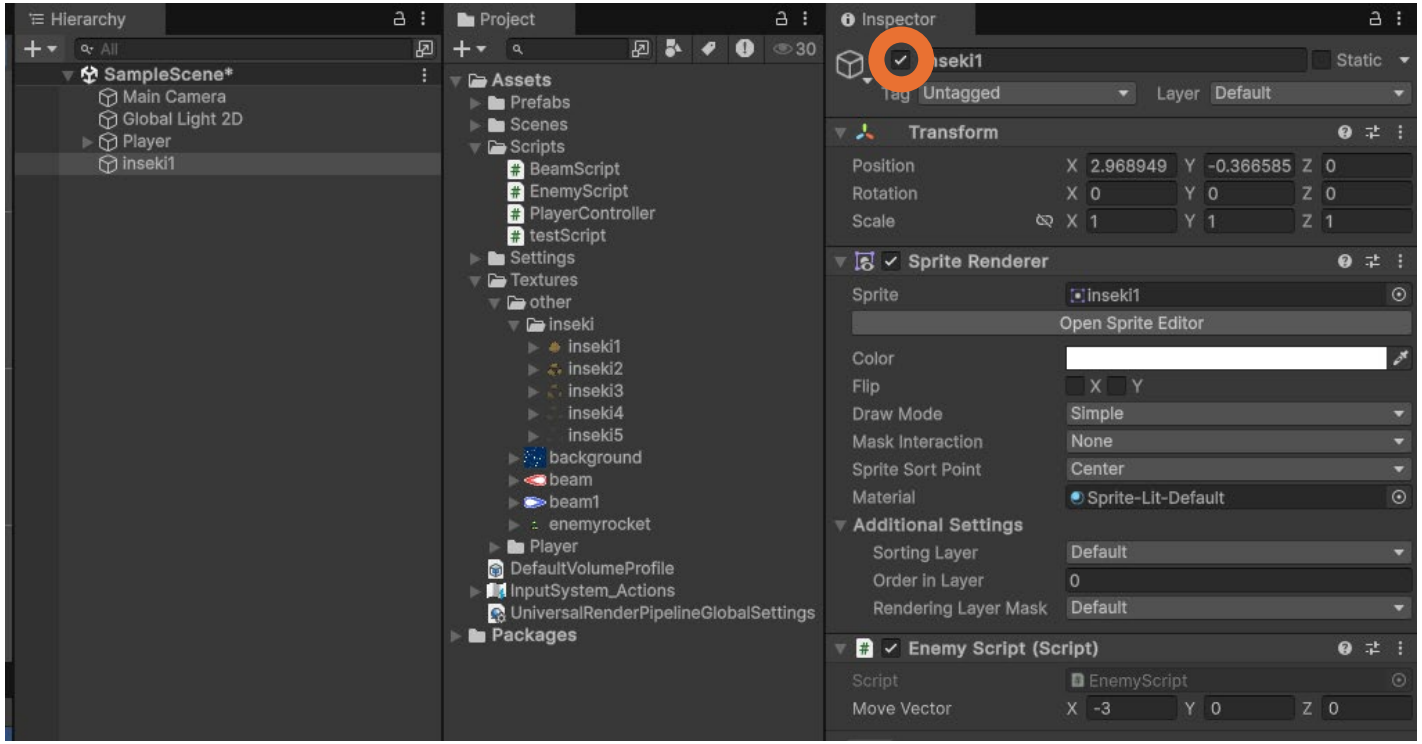
（しないと書いたコードが
inseki1に適応されないので
注意！！）



1：敵の追加

inseki1の
MoveVectorをマイナス方向に
設定して再生すると
隕石が左に移動する

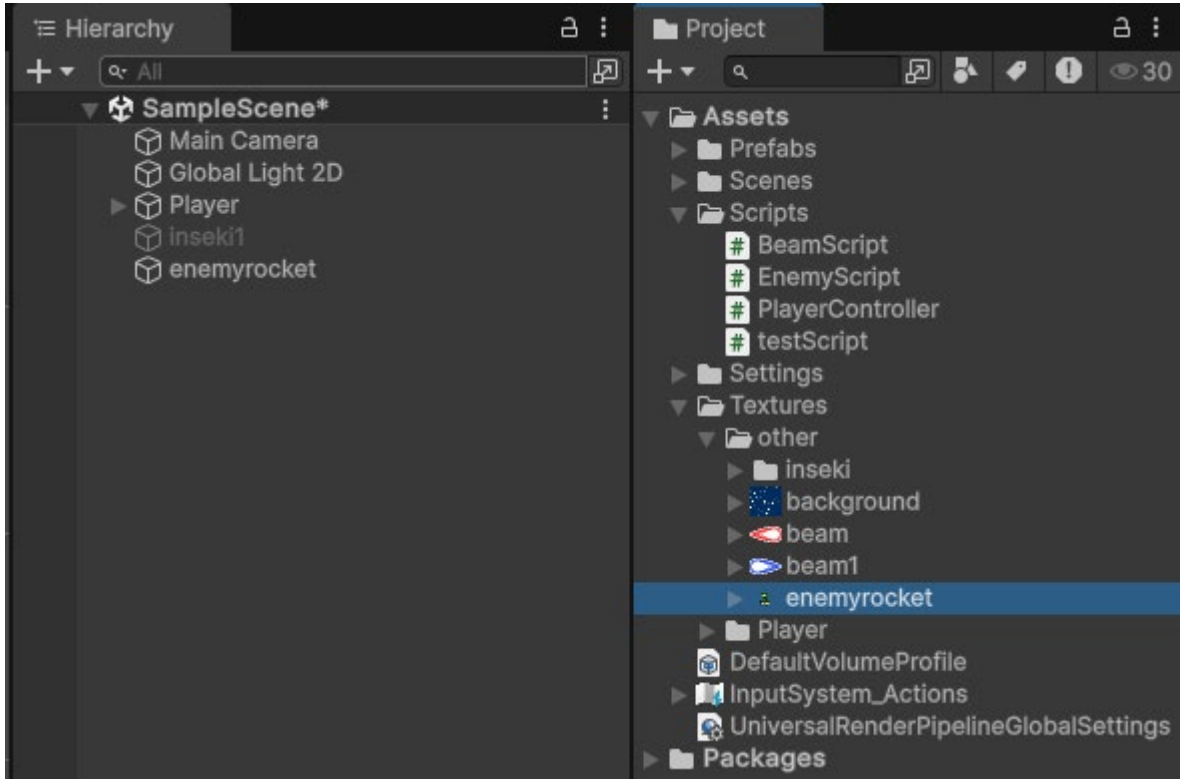
隕石はいったん終わり
確認できたら右上のチェックを
外して見えなくする



1：敵の追加

次に攻撃できる敵を追加する

Textures/other/enemyrocketを
Hierarchyにドラック&ドロップ



1：敵の追加

ドラック&ドロップしたら
EnemyScriptを開いて
コードを書き替える

書き換えたらセーブして
Unityに戻る

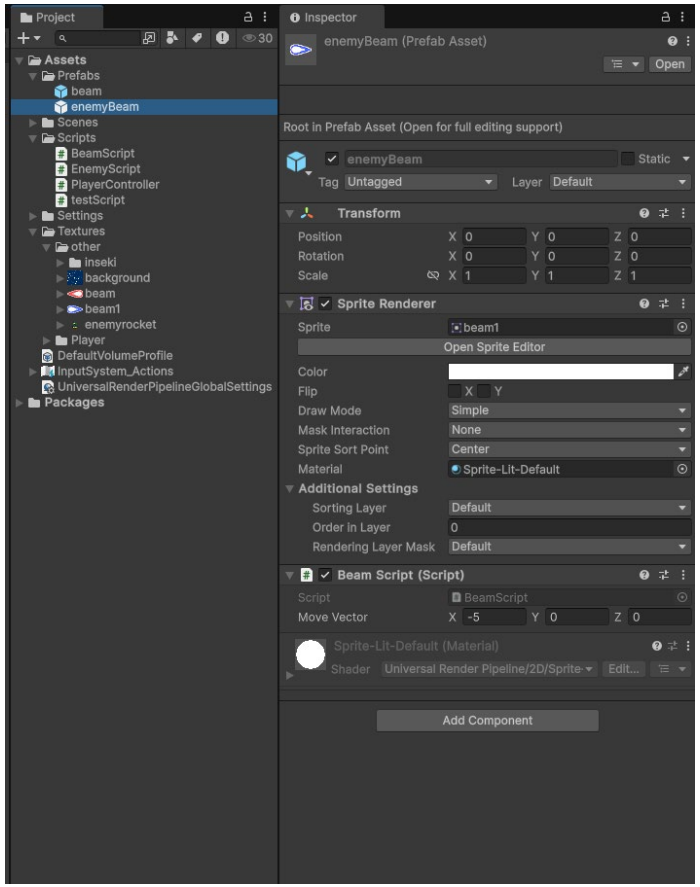
```
public class EnemyScript : MonoBehaviour
{
    public Vector3 MoveVector;

    public bool canAttack;
    public GameObject EnemyBeam;
    public float CoolTime=3;
    float CoolCountTime = 0;

    // Start is called once before the first execution of Update after the MonoBeha
    Unity メッセージ|0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        transform.position += MoveVector * Time.deltaTime;

        if (canAttack == true)
        {
            CoolCountTime -= 1 * Time.deltaTime;
            if (CoolCountTime <= 0)
            {
                Instantiate(EnemyBeam,transform.position,Quaternion.identity);
                CoolCountTime = CoolTime;
            }
        }
    }
}
```

1：敵の追加

戻ったら敵が撃つ弾を
用意する

BeamのPrefabを選び、
Ctrl + Dで複製

複製したものを右クリックして
“Rename”から名前を変更できる

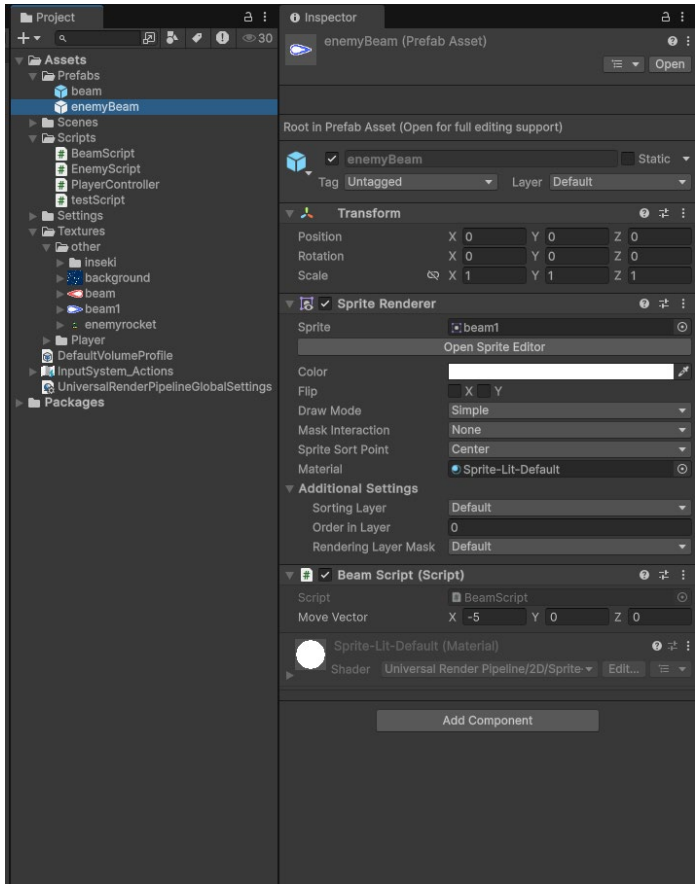


1：敵の追加

変更したら

SpriteRendererの
Spriteを青色の弾に置き換えておき、

BeamScriptの
MoveVectorをマイナス方向に
設定しておく

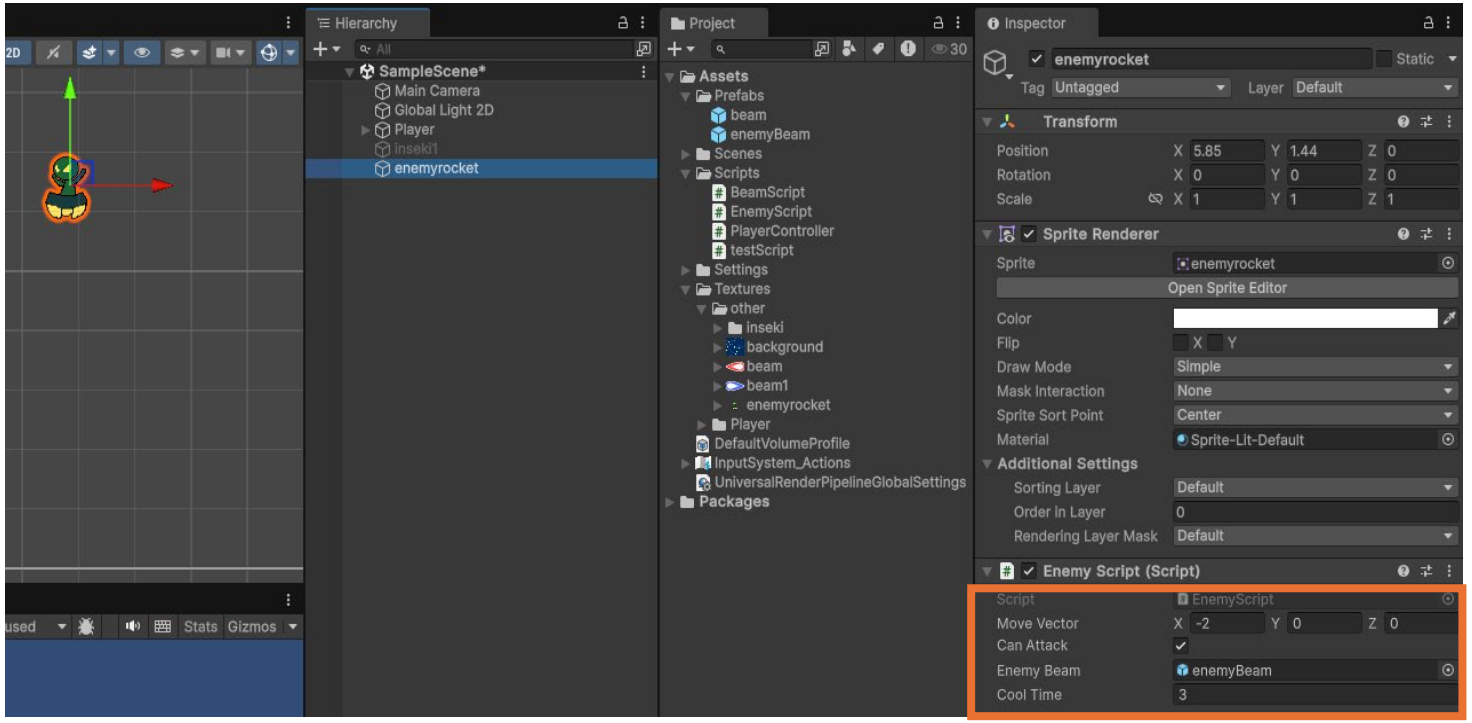


1：敵の追加

HierarchyのEnemyrocketを選び、

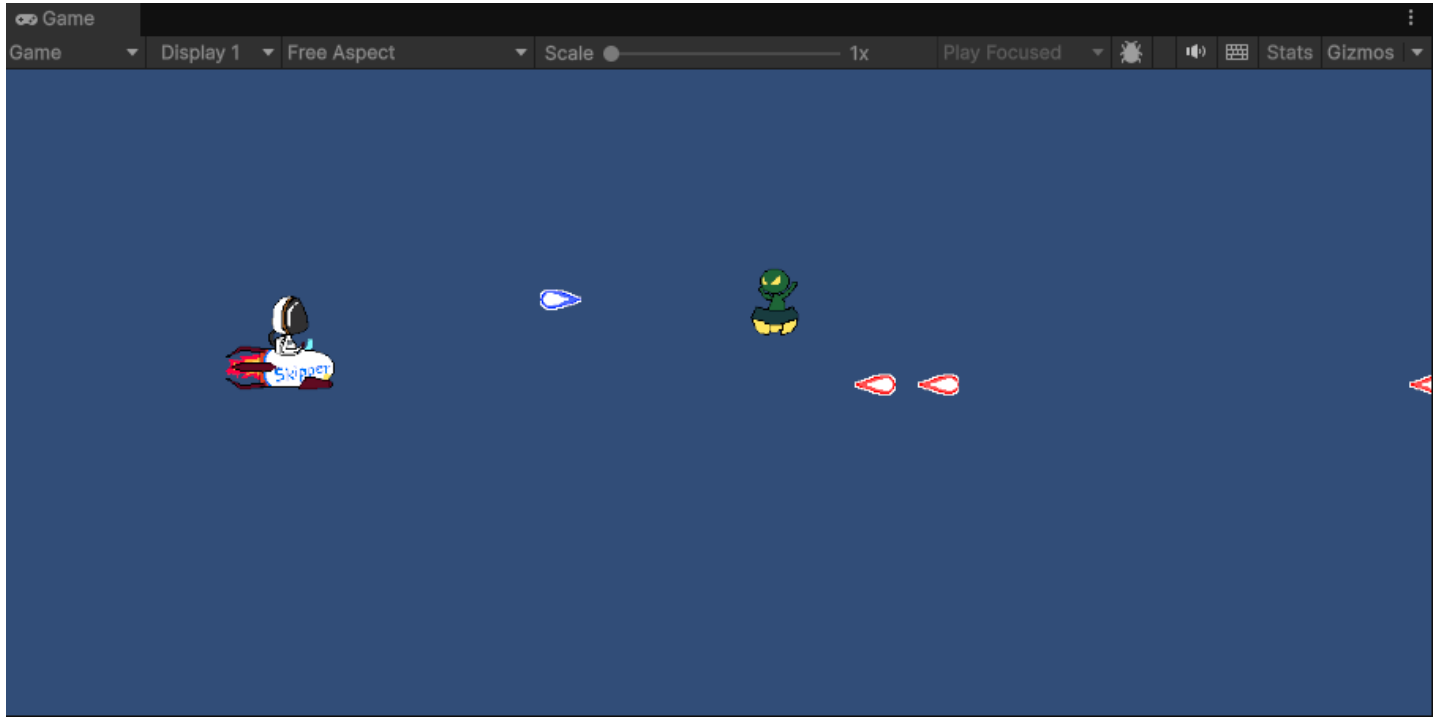
- CanAttackをチェック
- EnemyBeamにPrefabのenemybeamをドラック&ドロップ
- CoolTimeはお好きに

を設定する



1：敵の追加

弾が出てきました



1：敵の追加

コードの解説です
最初のところは
変数、インスタンスを宣言する
ところだと説明した

新しくbool型が追加されている
Bool型は値がtrue,falseしかとらない
変数のことである

Unityだとチェック欄が追加されたはず

```
public class EnemyScript : MonoBehaviour
{
    public Vector3 MoveVector;

    public bool canAttack;
    public GameObject EnemyBeam;
    public float CoolTime=3;
    float CoolCountTime = 0;

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

チェックある：true

チェックない：false

1：敵の追加

下のコードを見てみよう
Update内にあることを忘れずに

「もしcanAttackにチェックいれたら」の条件で隕石での制御か敵の制御かを区別している

その中についても見てみよう

```
if (canAttack == true)
{
    CoolCountTime -= 1 * Time.deltaTime;
    if (CoolCountTime <= 0)
    {
        Instantiate(EnemyBeam,transform.position,Quaternion.identity);
        CoolCountTime = CoolTime;
    }
}
```

Q:Updateってなんだっけ？

１：敵の追加

CoolCountTimeという
変数を１秒に-１している

式が『（左辺）-=（右辺）』の場合は
『（左辺）を（右辺）分減らす』こと
- → + にしたら足し算になる

値にTime.deltaTimeをかければ
１秒かけてその値を設定することができる

```
if (canAttack == true)
{
    CoolCountTime -= 1 * Time.deltaTime;
    if (CoolCountTime <= 0)
    {
        Instantiate(EnemyBeam,transform.position,Quaternion.identity);
        CoolCountTime = CoolTime;
    }
}
```

補足：p77

Q:毎フレームごとに実行されるところ

1：敵の追加

CoolCountTimeを毎秒減らしていって
もしCoolCountTimeが0以下になったら

- InstantiateでEnemyBeamを複製
- CoolCountTimeをCoolTimeに設定
  - 設定しないと無限に弾が出てくる
    （気になったら試してみよう）

を行っている

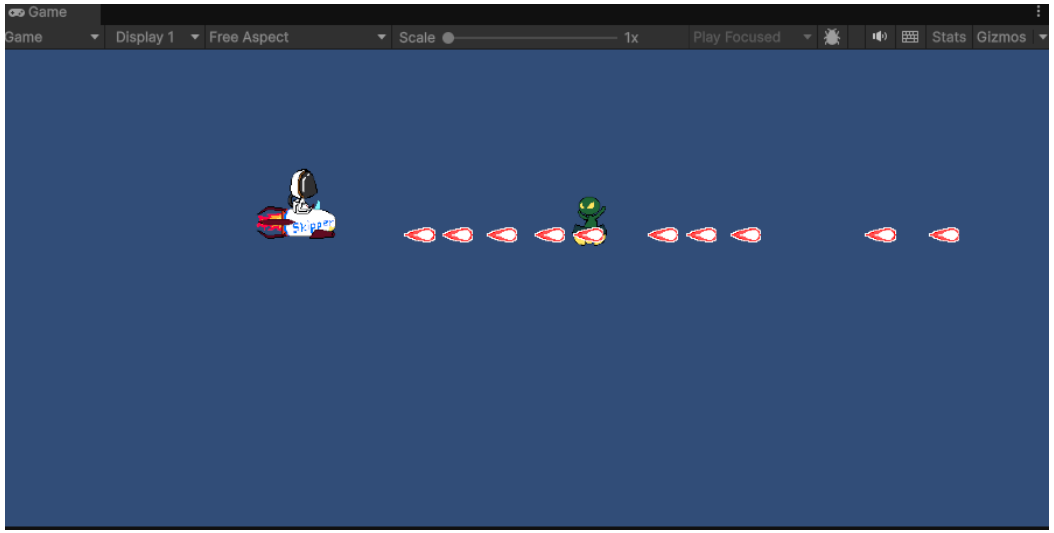
```
if (canAttack == true)
{
    CoolCountTime -= 1 * Time.deltaTime;
    if (CoolCountTime <= 0)
    {
        Instantiate(EnemyBeam,transform.position,Quaternion.identity);
        CoolCountTime = CoolTime;
    }
}
```

Q:毎フレームごとに実行されるところ

2：当たり判定の追加

これで敵の実装はしましたが、
なんか攻撃しても倒れないですね🤔

ここからは攻撃や衝突するための
“当たり判定”をそれぞれ追加していく

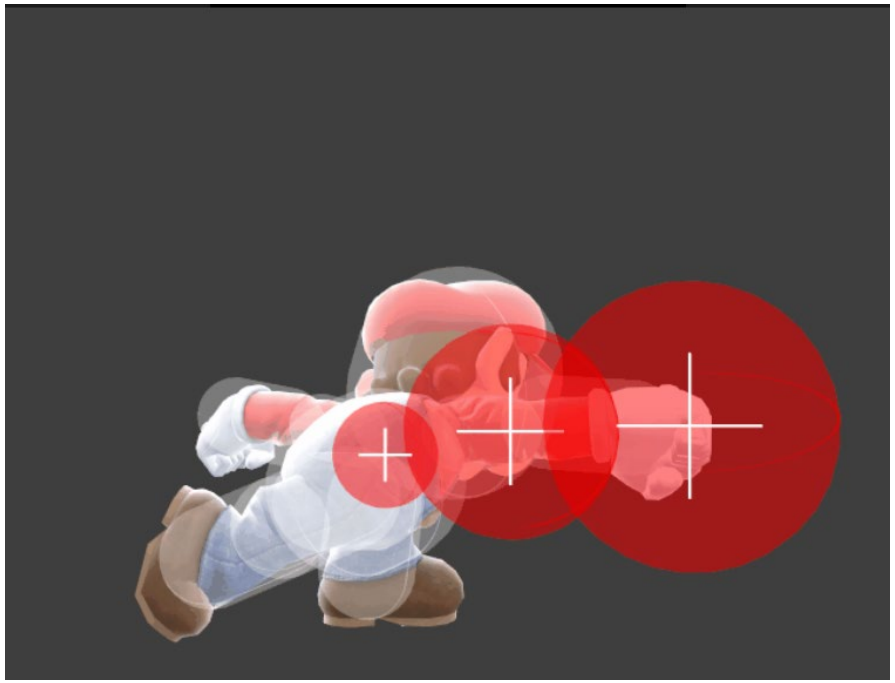


2：当たり判定の追加

そもそも当たり判定とは？

当たり判定は
「物と物がぶつかったことを知る判定」

スマブラの攻撃なら
（本来見えない）赤いエリアに
敵が触れたらダメージをうける、
みたいな感じ

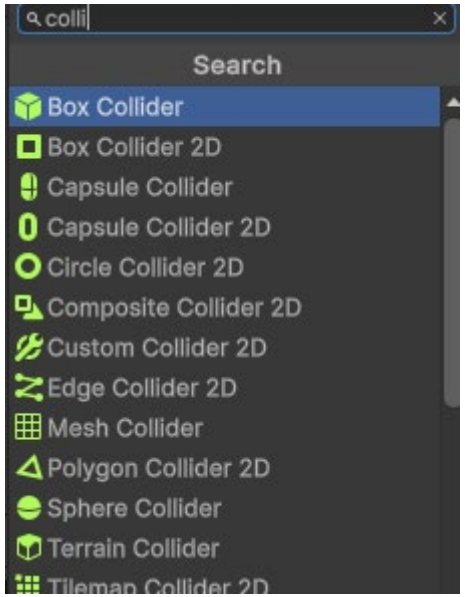


まぁ…実践しないとわかりづらいかも

2：当たり判定の追加

Unityの当たり判定は
全て”Collider”で範囲を設定している

Add Componentで
“Collider”と検索すると
いろいろ出てくるが、範囲の形が違うだけで
やっていることはほとんど同じである

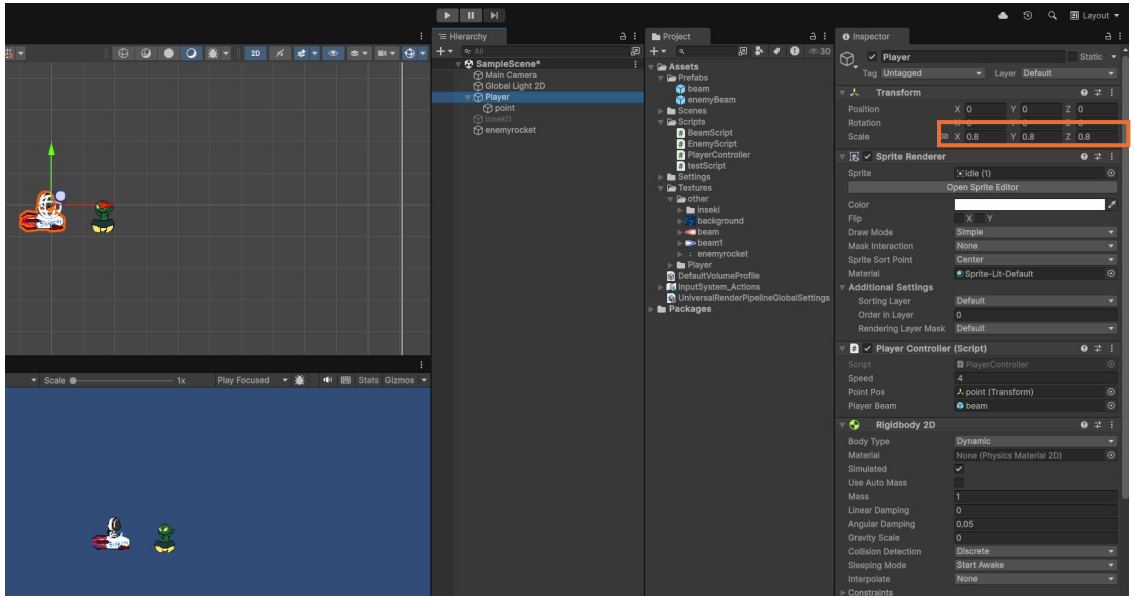


2：当たり判定の追加

まずはPlayerの当たり判定を設定しよう

その前にPlayerの大きさが
やけにデカイので
Transform→Scaleから値を
全て0.8に設定しておく

なお、親オブジェクトの大きさを
設定すると子オブジェクトも小さくなる！



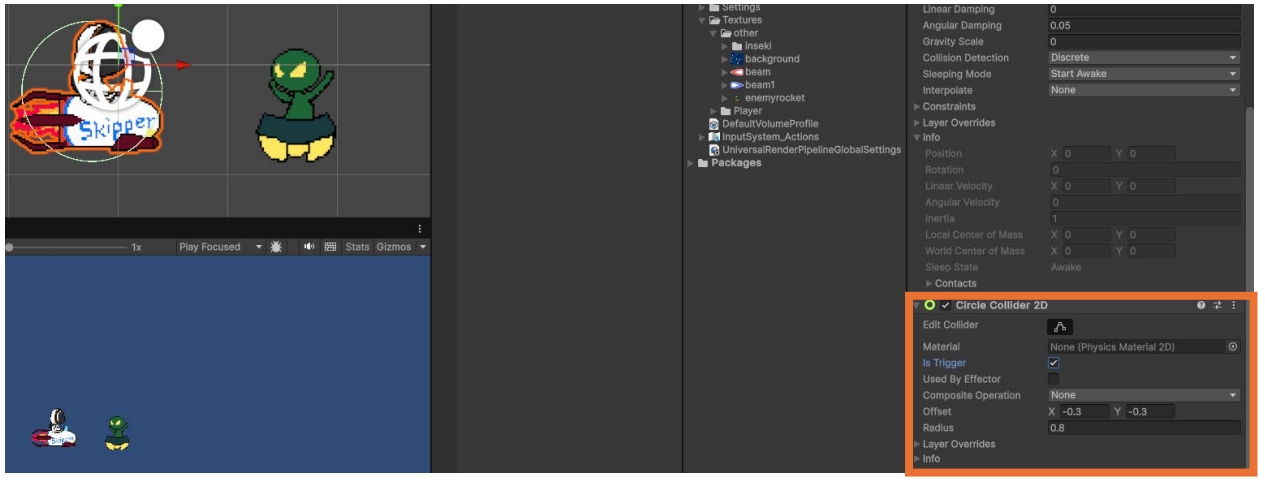
2：当たり判定の追加

PlayerにAdd Componentから
“Circle Collider 2D”を選択

CircleCollider2Dの設定を
画像の通りに設定しておく

左上の円に囲まれた部分が
Playerの当たり判定となる

もし細かく設定したい場合は
“CircleCollider2D”ではなく
”Polygon Collider 2D”を使用してみよう

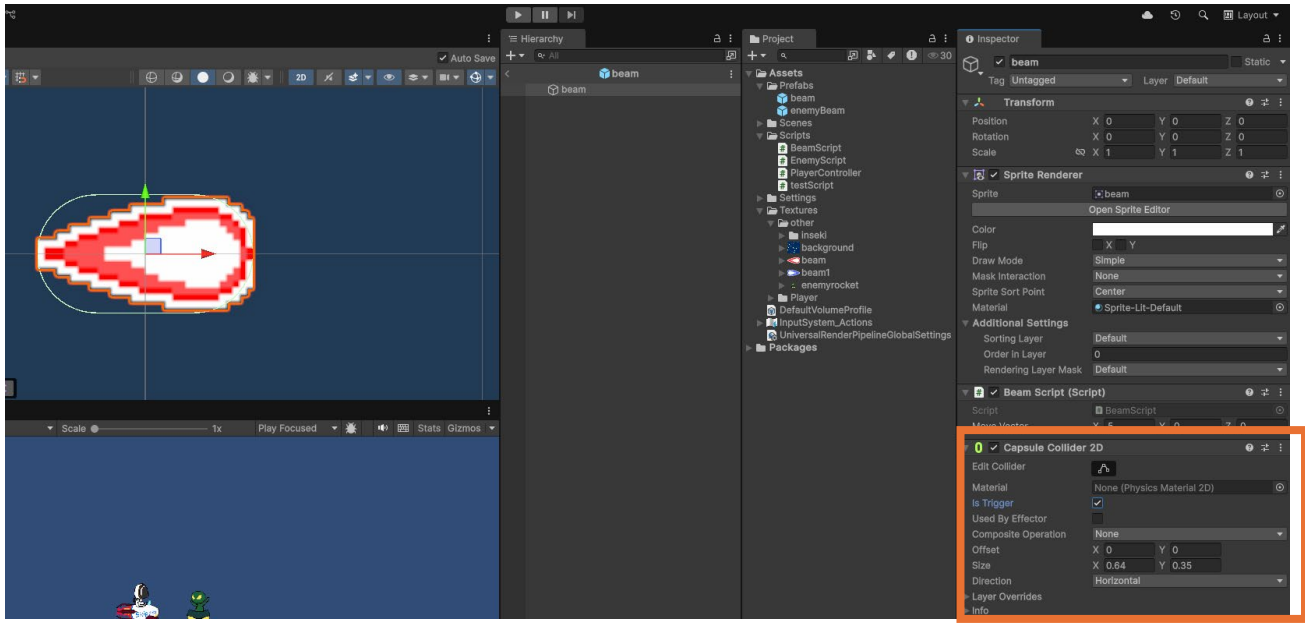


2：当たり判定の追加

次に弾の当たり判定を設定する

PrefabのPlayerBeamを選んで
Add Componentから
“Capsule Collider 2D”を選択

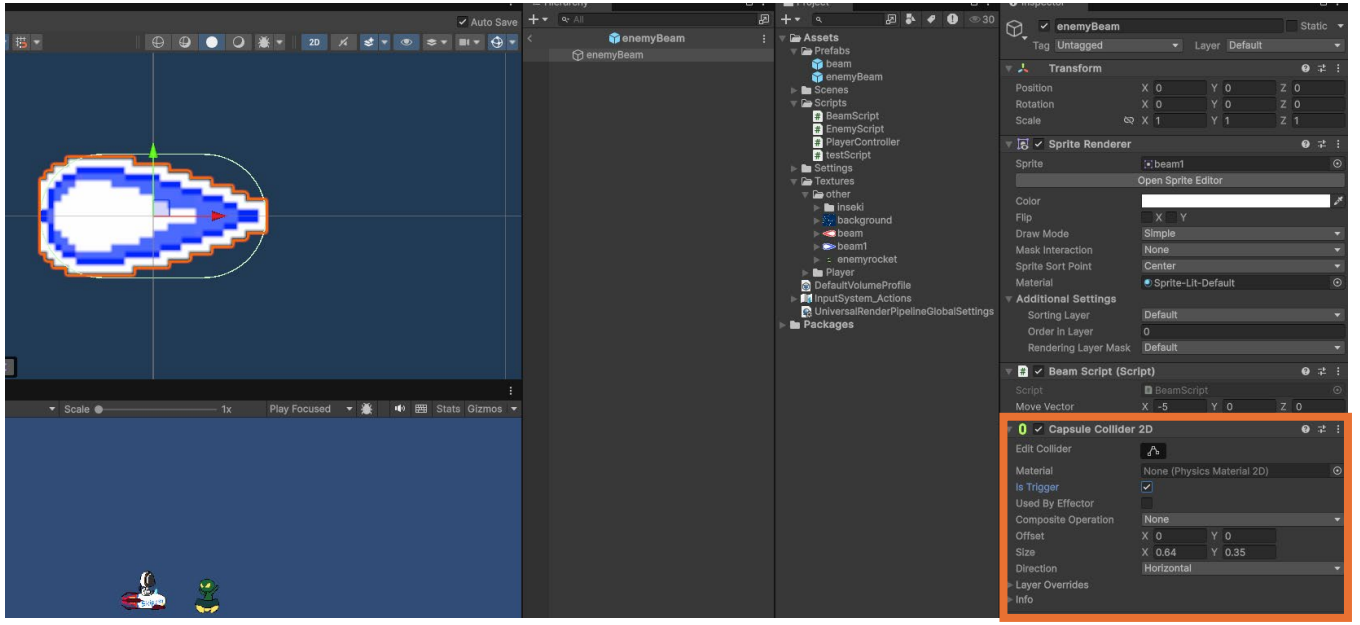
DirectionをHorizontalに
設定してからsizeの値を入れること



2：当たり判定の追加

同様にEnemyBeamも
同じ設定をしておく

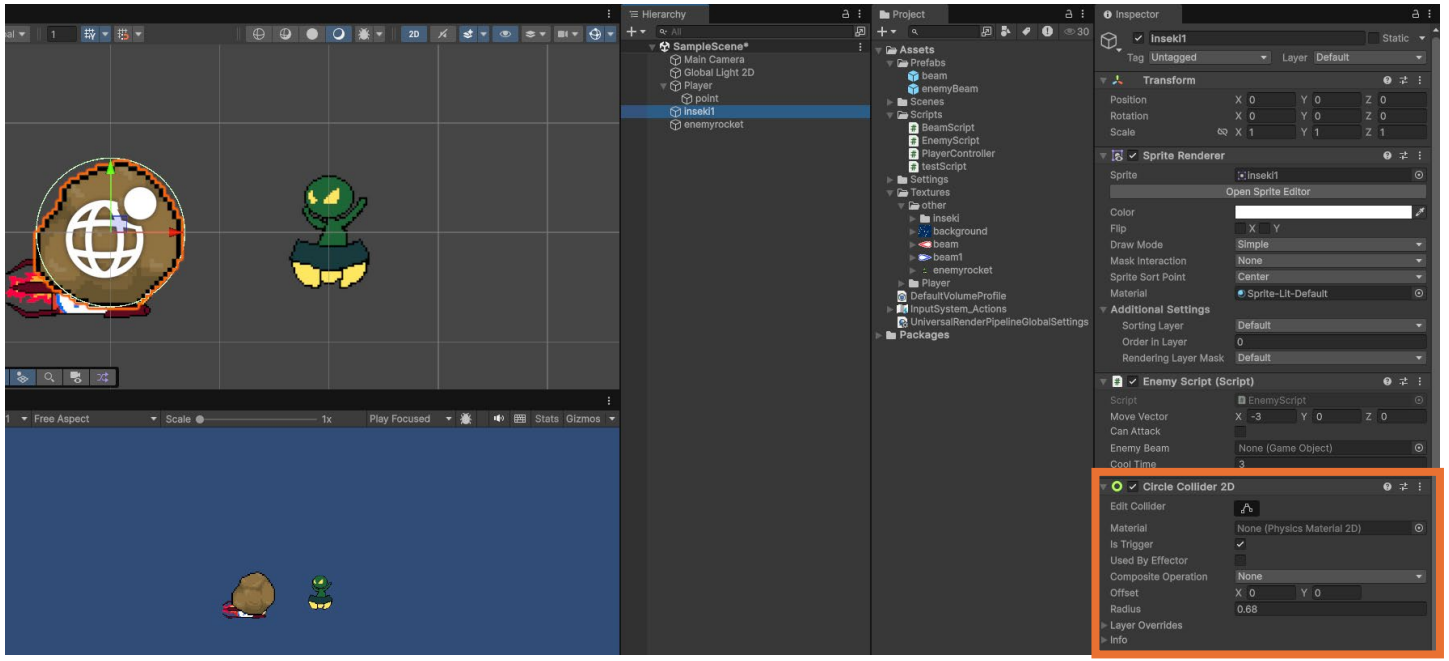
PrefabのEnemyBeamを
開いて同じ設定にすること



2：当たり判定の追加

次に隕石の設定をする

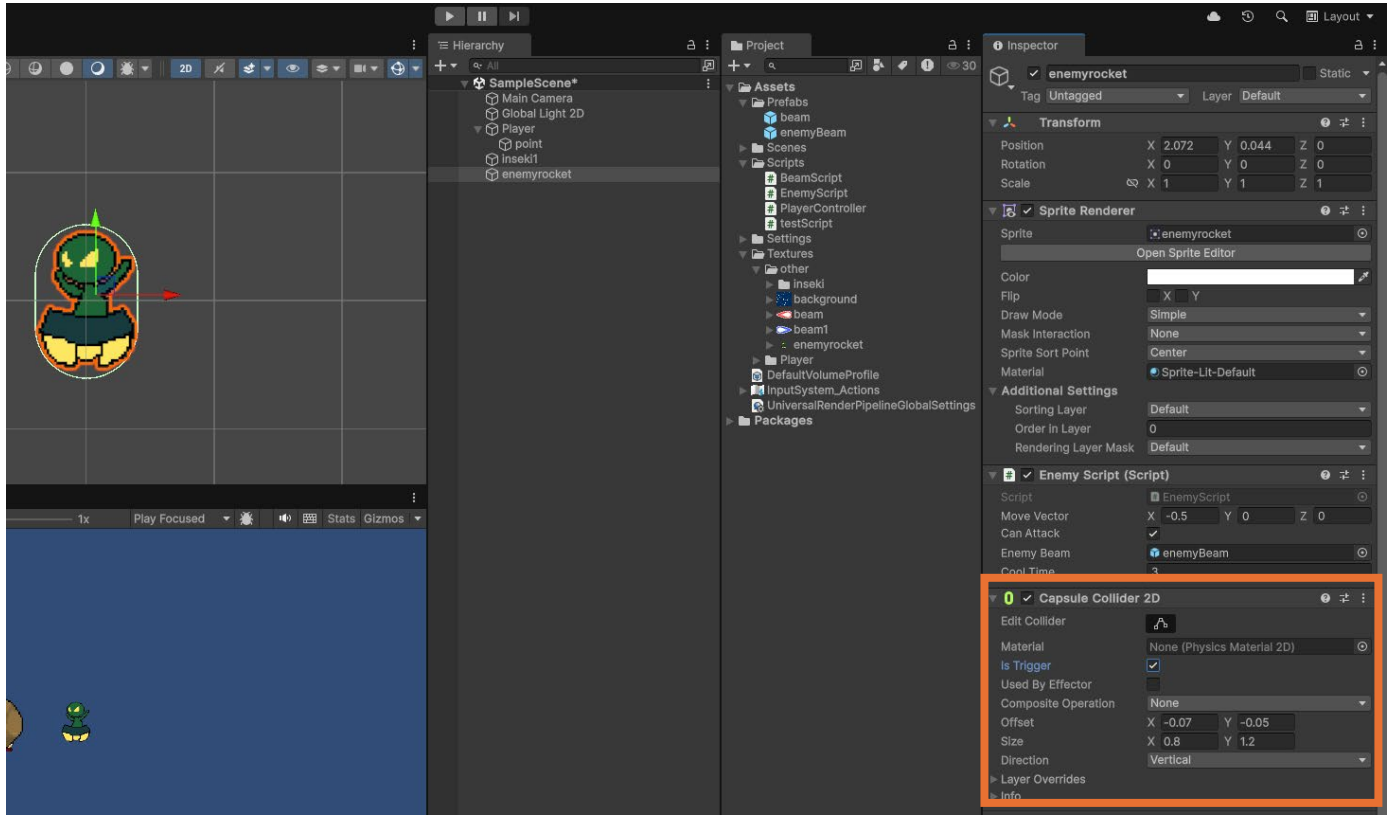
Inseki1を再度表示させて
CircleCollider2Dを追加し、
画像の通りに設定する



2：当たり判定の追加

最後に敵の設定をする

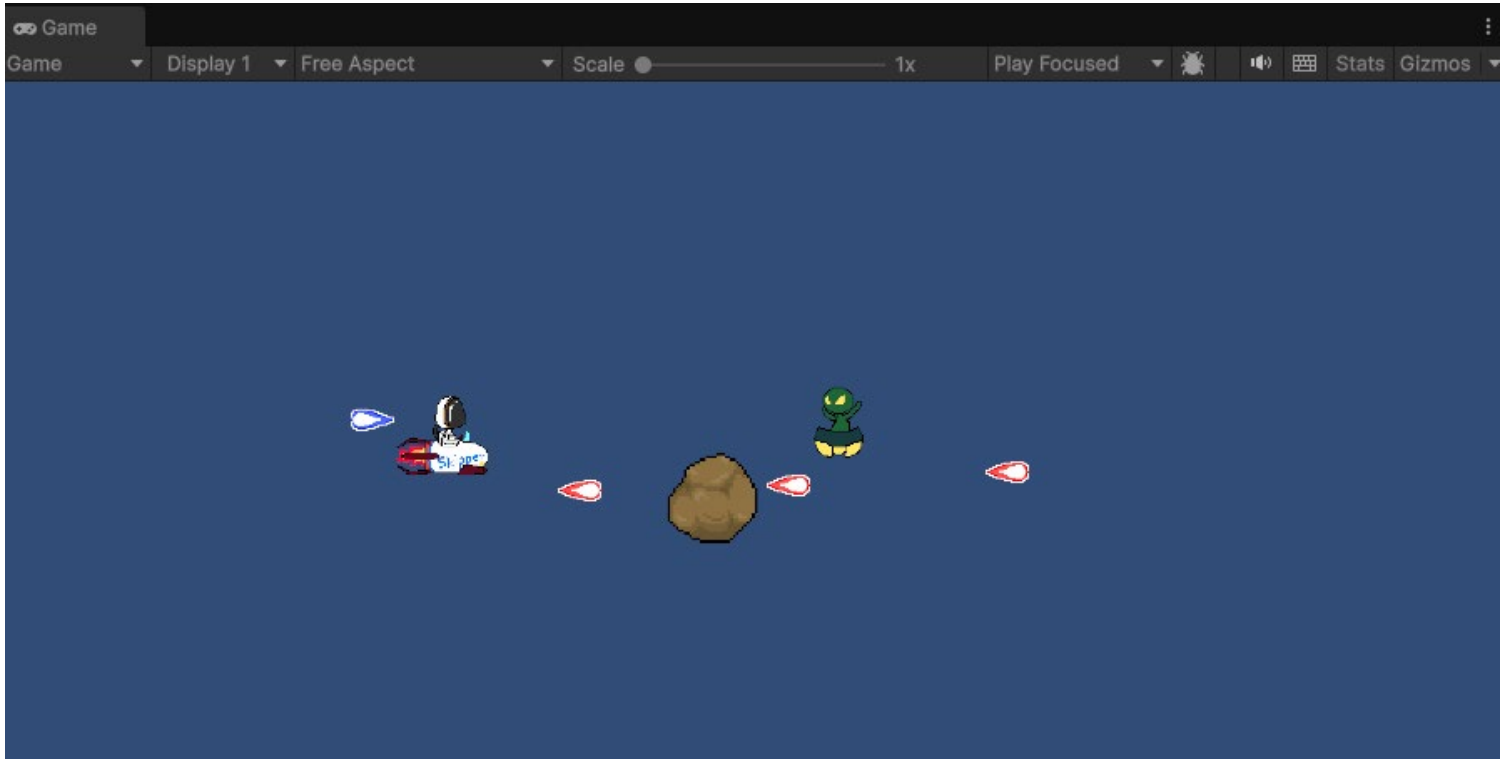
Enemyrocketを選択し、CapsuleCollider2Dを追加画像の通りに設定



2：当たり判定の追加

設定が終わりましたー！

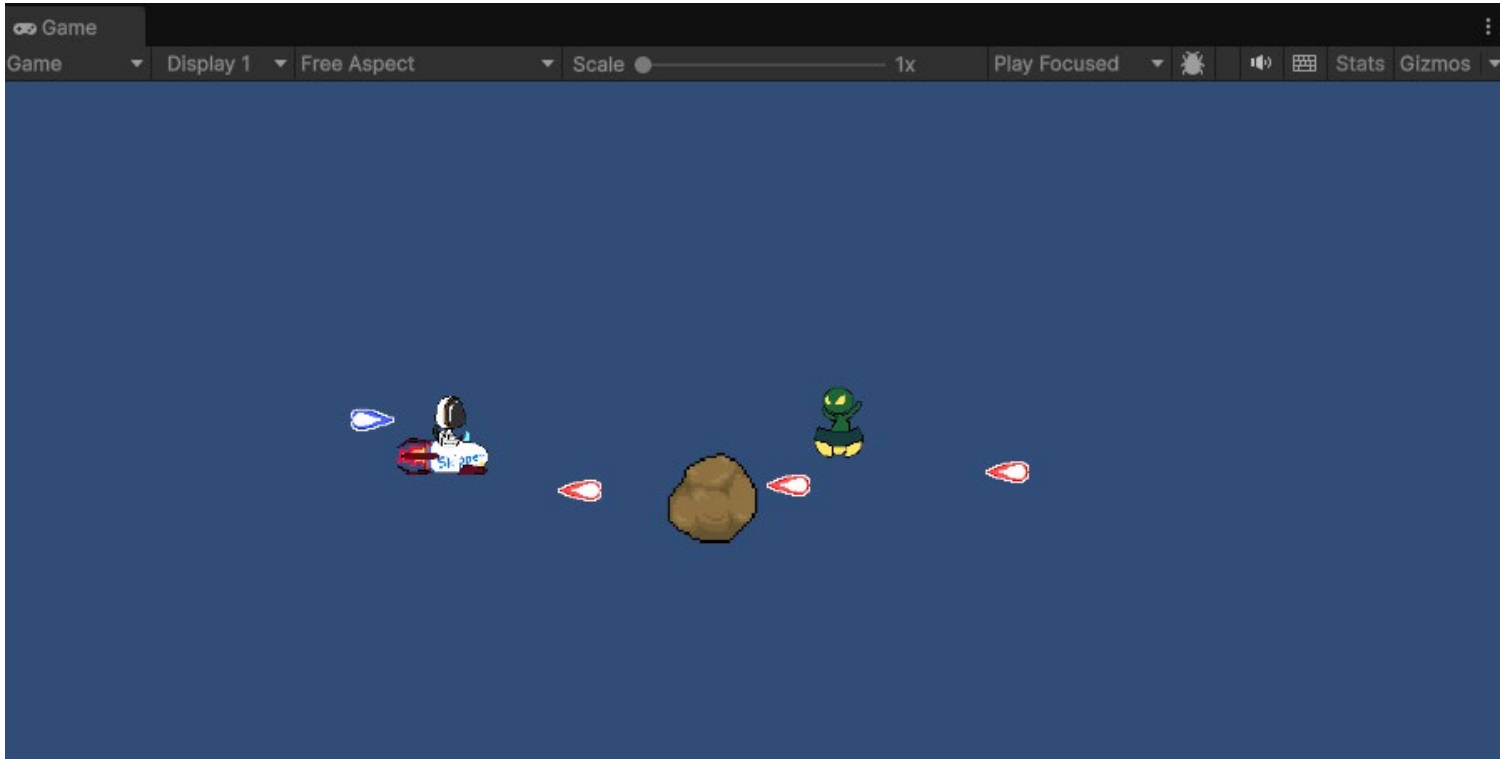
でも攻撃できないな？
🤔🤔🤔



2：当たり判定の追加

当たった時の処理を
まだ書いていないから

加えてオブジェクトの設定が
まだできていないので
やっておきます



2：当たり判定の追加

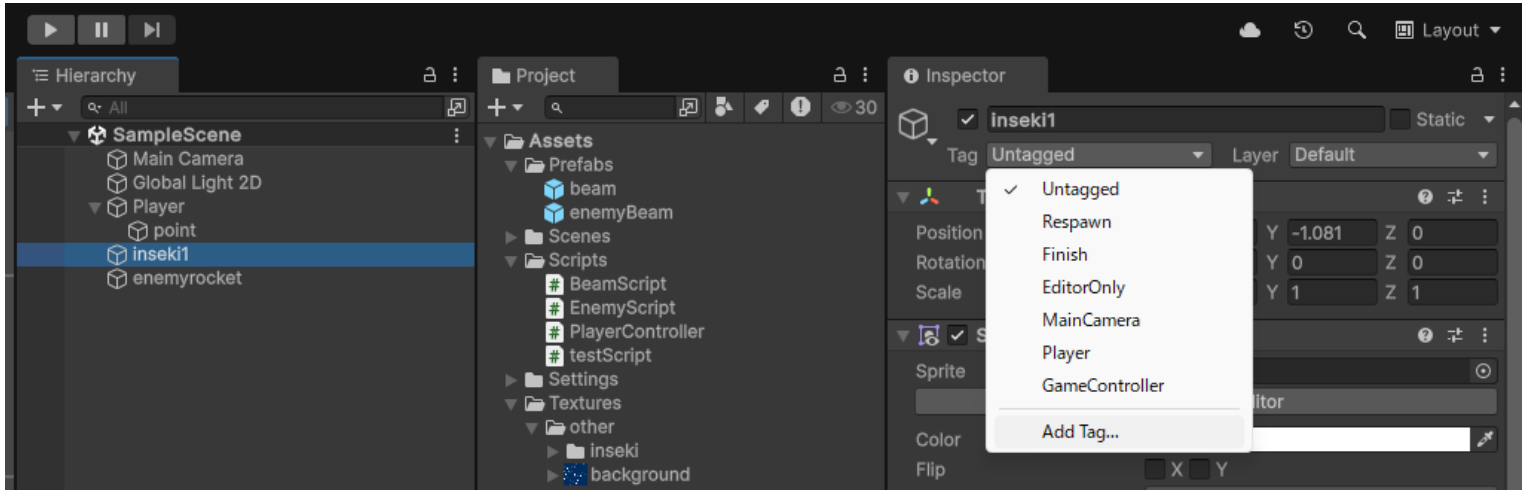
オブジェクトを区別する設定として
“Tag”を設定します

“Tag”は各オブジェクトに１つ設定できるもの

人間からは見た目とかで判断できますが、
Unity、パソコン側では
見た目だけでは区別できないので
このTagを設定して区別します

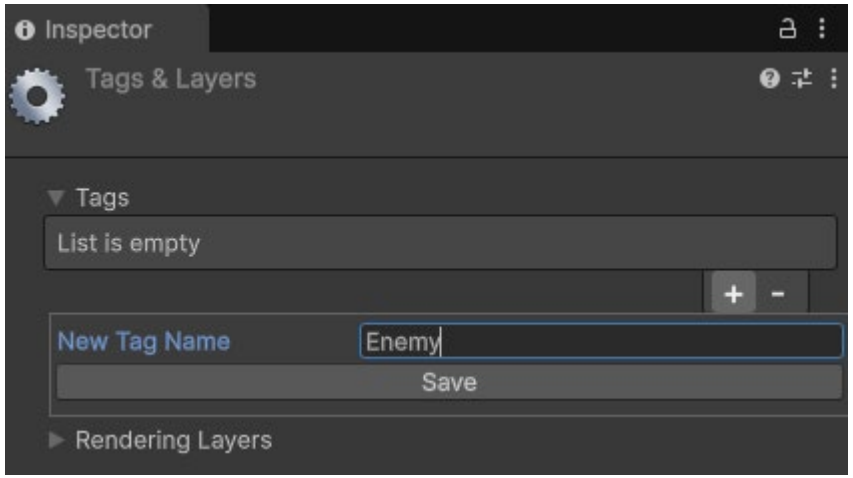
2：当たり判定の追加

まずはinseki1を選んで
Tag→Add Tagを選択



2：当たり判定の追加

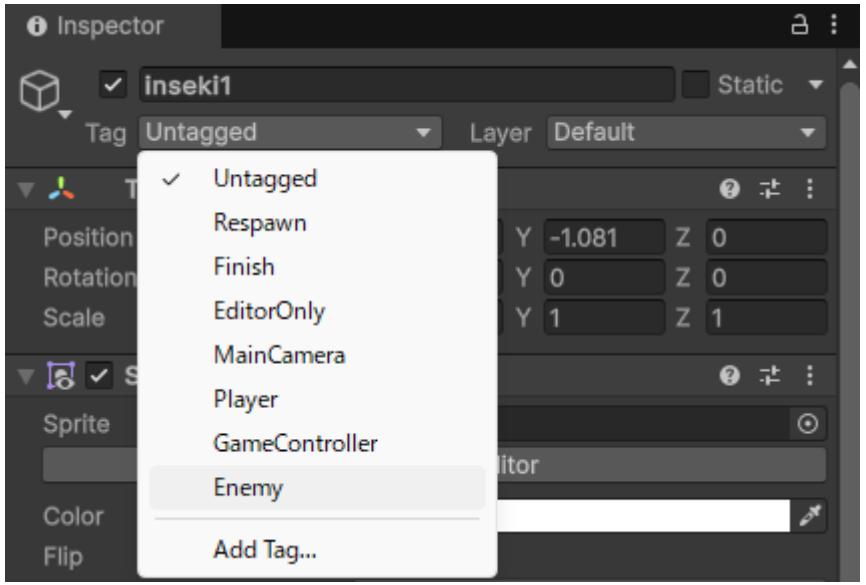
Tagsというものが出てきます
＋を押して名前を入力すれば
新しくTagを増やせます
（今回は”Enemy”という名前）



2：当たり判定の追加

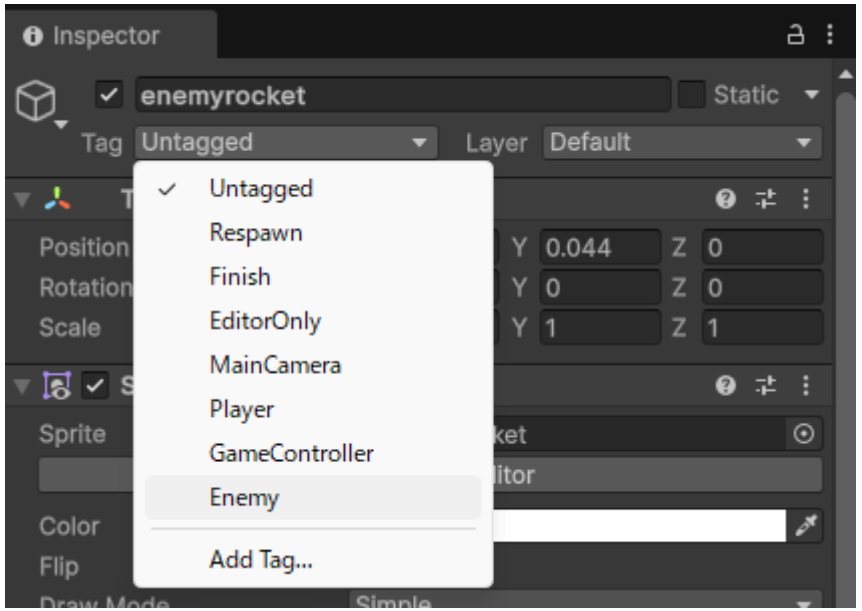
Tagは追加したが、
それをオブジェクトに割り振れていない

再度inseki1を選んで
Tagを見ればEnemyがあるので
選択すれば追加ができる



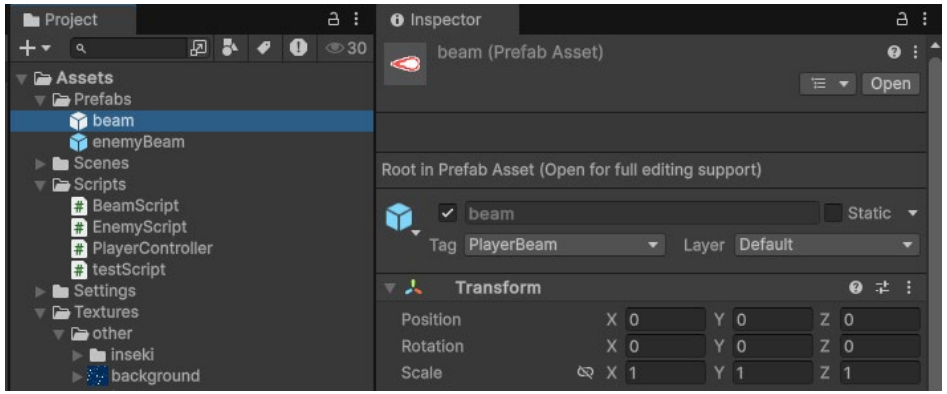
2：当たり判定の追加

Inseki1は完了したので、
Enemyrocketにも
同じtagをつけておく

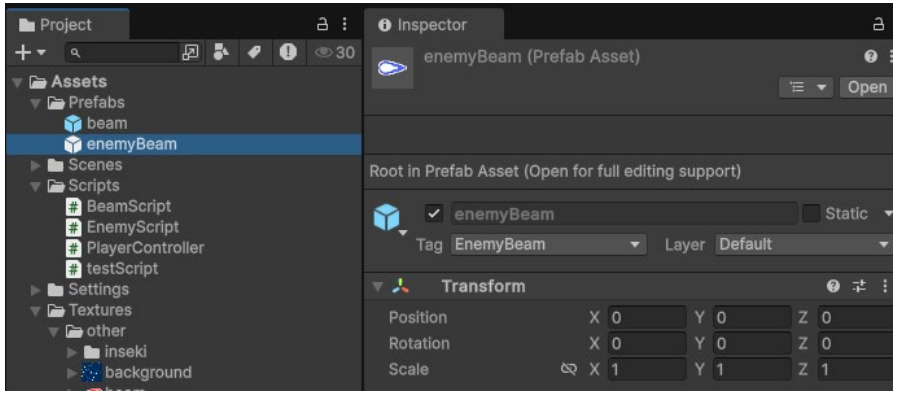


2：当たり判定の追加

次に弾にもtagをつけてみよう
追加する方法は同じ



Beamには”PlayerBeam”
EnemyBeamには”EnemyBeam”という
名前のtagを追加しよう



2：当たり判定の追加

Tagの追加ができたら
攻撃の処理を書いていく

“EnemyScript”を開いて
コードを追記する

Void Updateの外に書くこと！

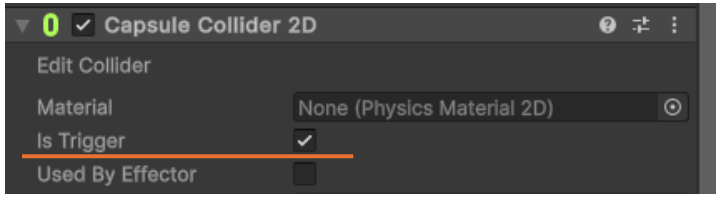
```
// Update is called once per frame
Unity メッセージ | 0 個の参照
void Update()
{
    transform.position += MoveVector * Time.deltaTime;

    if (canAttack == true)
    {
        CoolCountTime -= 1 * Time.deltaTime;
        if (CoolCountTime <= 0)
        {
            Instantiate(EnemyBeam,transform.position,Quaternion.identity);
            CoolCountTime = CoolTime;
        }
    }
}

Unity メッセージ | 0 個の参照
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("PlayerBeam"))
    {
        Destroy(collision.gameObject);
        Destroy(this.gameObject);
    }
}
```

2：当たり判定の追加

先にコードの解説をします
OnTriggerEnterについて

```
Unity メッセージ | 0 個の参照
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("PlayerBeam"))
    {
        Destroy(collision.gameObject);
        Destroy(this.gameObject);
    }
}
```

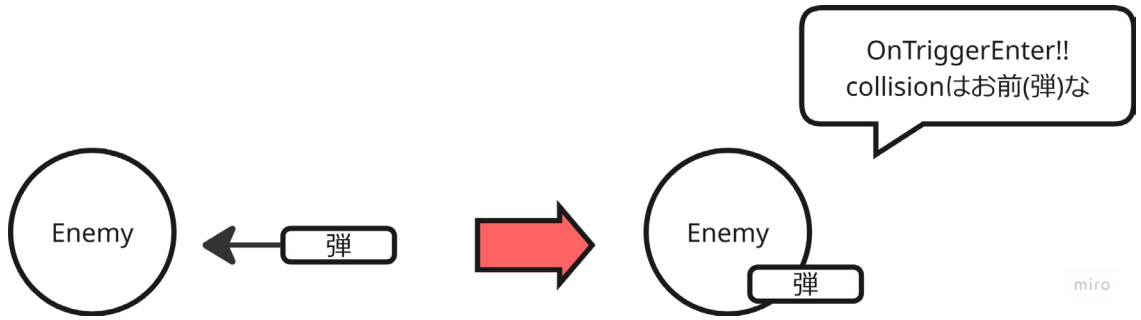
Colliderに”Is Trigger”をチェックした
オブジェクトがEnemyScriptに当たった時、
相手のオブジェクト〔今回は弾〕が
Collisionとして利用される



2：当たり判定の追加

イメージはこんな感じ

補足：p78



2：当たり判定の追加

```
Unity メッセージ | 0 個の参照
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("PlayerBeam"))
    {
        Destroy(collision.gameObject);
        Destroy(this.gameObject);
    }
}
```

onTriggerEnterの中を見る

ぶつかったオブジェクトのTagが
指定した名前と一致していたら、という条件を
調べるためにCompareTagが使用される

前にcollisionがあると
Collisionのtagを調べることができる

2：当たり判定の追加

```
Unity メッセージ | 0 個の参照
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("PlayerBeam"))
    {
        Destroy(collision.gameObject);
        Destroy(this.gameObject);
    }
}
```

Destroyで
指定したオブジェクトを
削除することができる

今回はcollisionのオブジェクトを削除した後に
敵自身を削除している

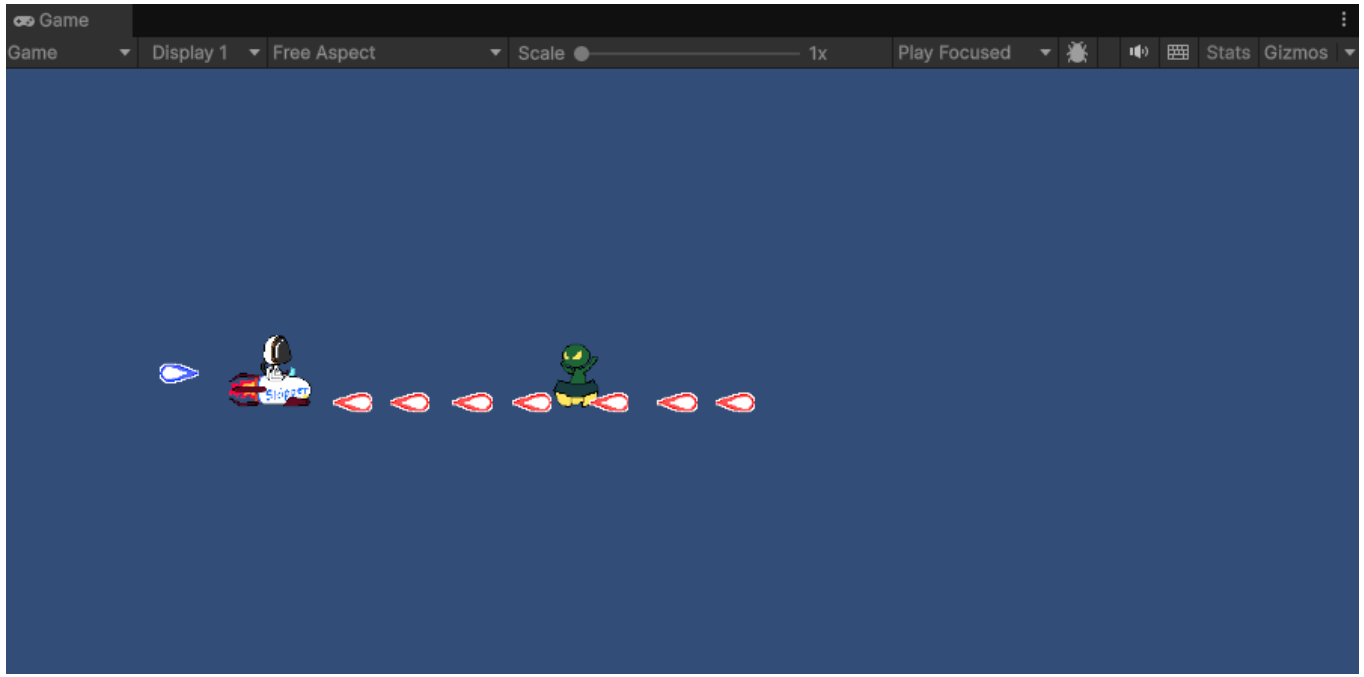
2：当たり判定の追加

これで攻撃ができますね！！

はぁ🤔🤔🤔🤔🤔

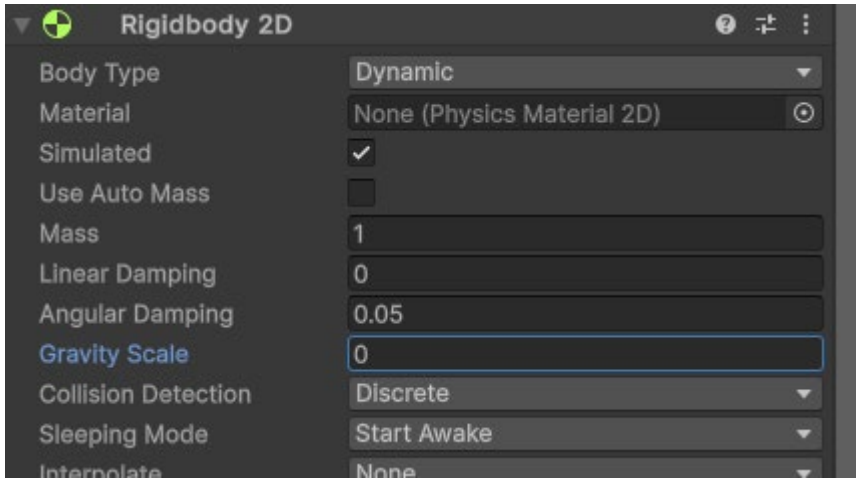
2：当たり判定の追加

実は一番注意してほしい点がある
衝突を検知する際は
“どちらかのオブジェクトに
Rigidbodyがないと反応しない”
という点に注意！！！



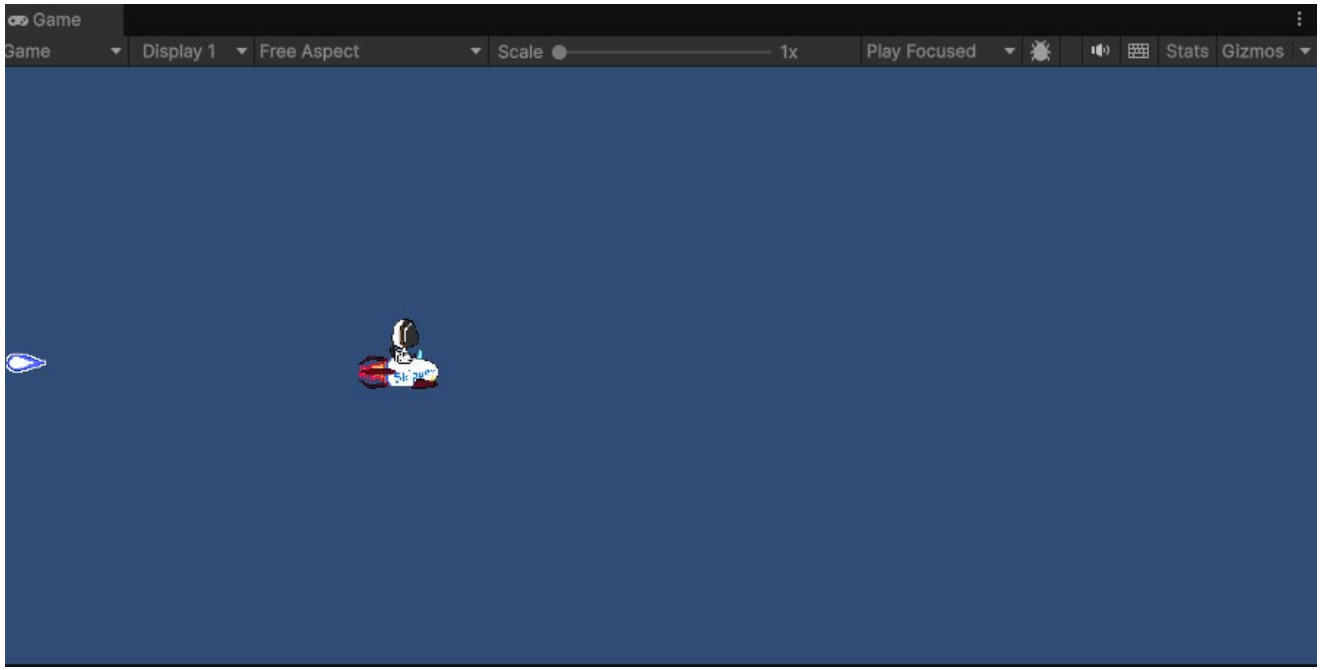
2：当たり判定の追加

Inseki1とEnemyrocketに
Rigidbody 2Dを追加し、
GravityScaleを0に設定
（どちらも！）



2：当たり判定の追加

これでやっと攻撃できました



3：プレイヤーのHP

次にプレイヤーのHPを導入します

おそらくプログラミングの知識があれば
できる人はできそう

3：プレイヤーのHP

流れはこう

- プレイヤーのHPの変数を用意する
- 敵の弾にプレイヤーが当たったらHPを減らす
- 0になったら（いったん）プレイヤーを削除する

3：プレイヤーのHP

とりあえずやってみる

PlayerControllerを開いて
コードを書いてみる

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

public Transform PointPos;
public GameObject PlayerBeam;
// Start is calle
Unity メッセー
void Start()
{
    rb = GetCompo
```

```
void Update()
{
    float GetHorizontal = Input.GetAxis("Horizontal");
    float GetVertical = Input.GetAxis("Vertical");

    Vector3 Vec = new Vector3(GetHorizontal, GetVertical, 0);
    if (Vec.magnitude > 1)
    {
        Vec = Vec.normalized;
    }

    rb.linearVelocity = Vec * speed;

    if (Input.GetKeyDown(KeyCode.Space))
    {
        Instantiate(PlayerBeam, PointPos.position,Quaternion.identity);
    }
}

Unity メッセージ | 0 個の参照
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("EnemyBeam"))
    {
        Destroy(collision.gameObject);

        HP -= 1;
        Debug.Log($"HP:{HP}");
        if (HP <= 0)
        {
            Destroy(gameObject);
        }
    }
```

3：プレイヤーのHP

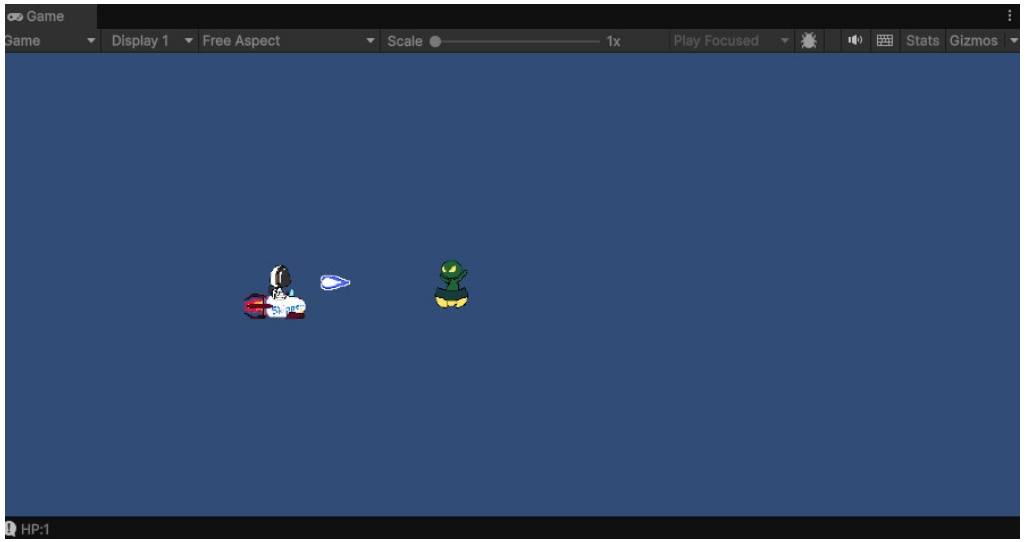
Debug.Logは
デバック用のコード
かっこの中をコンソールに表示できる

$が頭にあると{ }に囲まれた部分に
変数をいれればその中身が表示される

```
void Update()
{
    float GetHorizontal = Input.GetAxis("Horizontal");
    float GetVertical = Input.GetAxis("Vertical");

    Vector3 Vec = new Vector3(GetHorizontal, GetVertical, 0);
    if (Vec.magnitude > 1)
    {
        Vec = Vec.normalized;
    }

    rb.linearVelocity = Vec * speed;

    if (Input.GetKeyDown(KeyCode.Space))
    {
        Instantiate(PlayerBeam, PointPos.position,Quaternion.identity);
    }
}

Unity メッセージ | 0 個の参照
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("EnemyBeam"))
    {
        Destroy(collision.gameObject);

        HP -= 1;
        Debug.Log($"HP:{HP}");
        if (HP <= 0)
        {
            Destroy(gameObject);
        }
    }
}
```

3：プレイヤーのHP

攻撃をくらうと下にHPが
出てくるようになりました



3：プレイヤーのHP

ついでに、敵本体に当たったら
ダメージを受けるように設定をする

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

        HP -= 1;
```

〔条件〕||〔条件〕と書くと
“どちらかの条件を満たしたら”が書ける
実質”OR”みたいな感じ

４：敵の生成

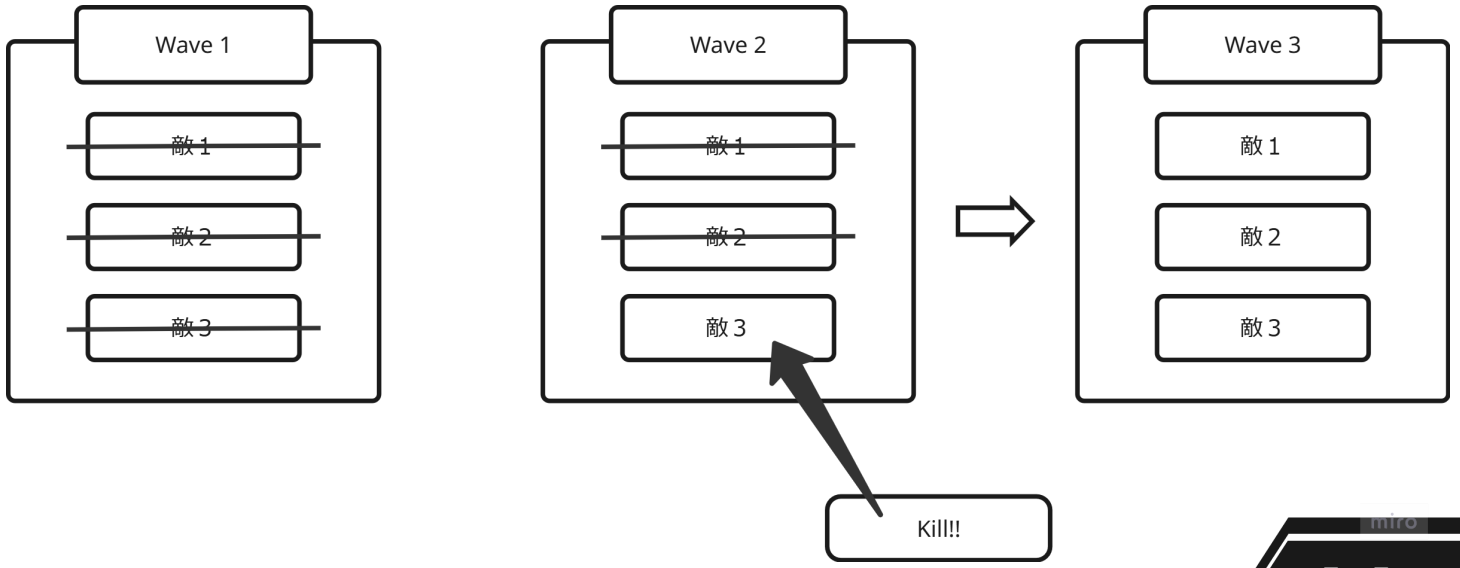
敵の生成をする場合、よくあるパターンは

- ジェネレータを設置して一定間隔に敵を出現する
- ウェーブを導入して敵を全て倒しきったら次のウェーブにはいる

がイメージされる
ゲームジャムに生かしやすいよう、後者の方でやってみるが
前者は気になったら調べてください

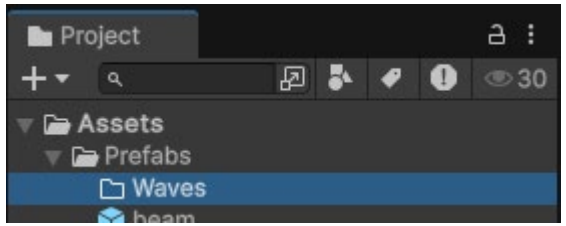
4：敵の生成

Waveで管理する場合は
Waveのオブジェクトの元に
敵を配置し、その数が0になったら次のWaveを生成する仕組み



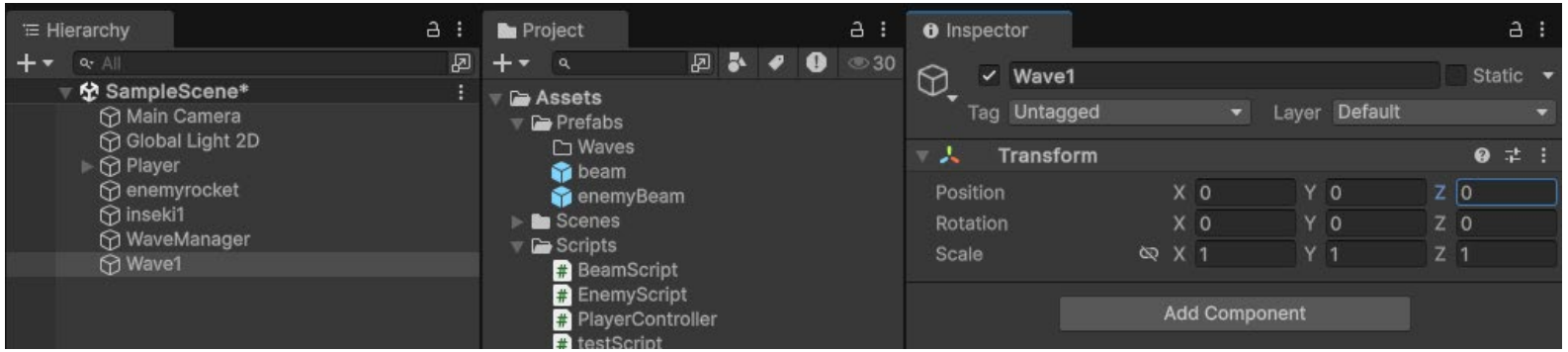
４：敵の生成

さっそくwaveを作ってみよう
まずはPrefabsフォルダに
新しく”Waves”フォルダを用意する



4：敵の生成

作ったら
Hierarchyの
何もないスペースで
右クリックし
“Create Empty”



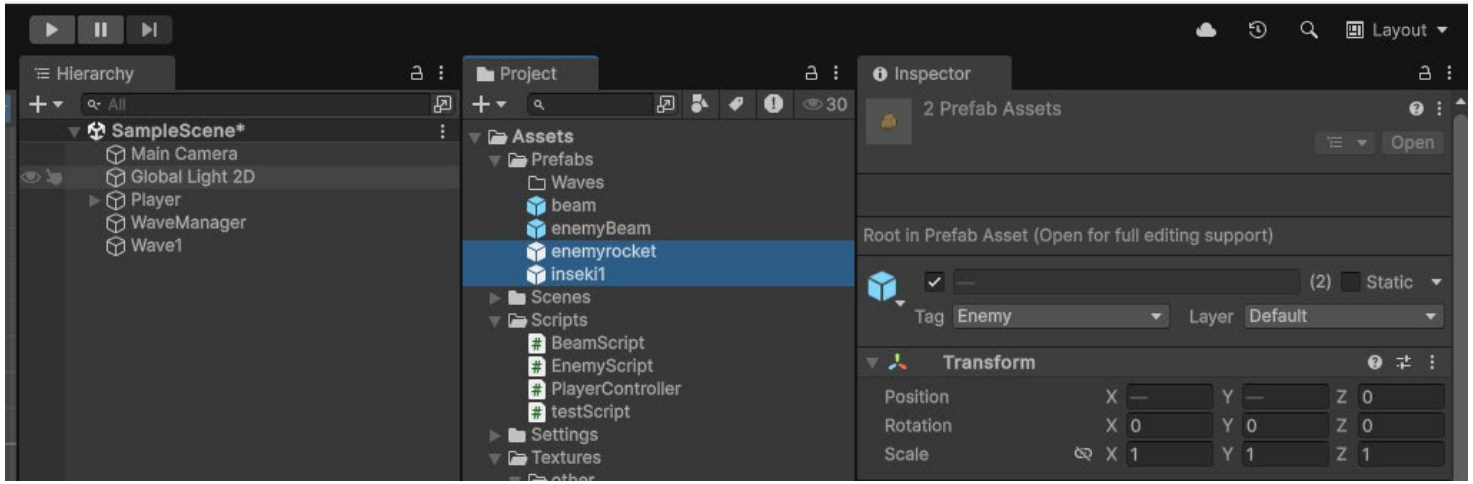
名前はWave1にしておく
位置は全て0にしておくこと

4：敵の生成

次に今まで作ってきた敵を
Prefabにする

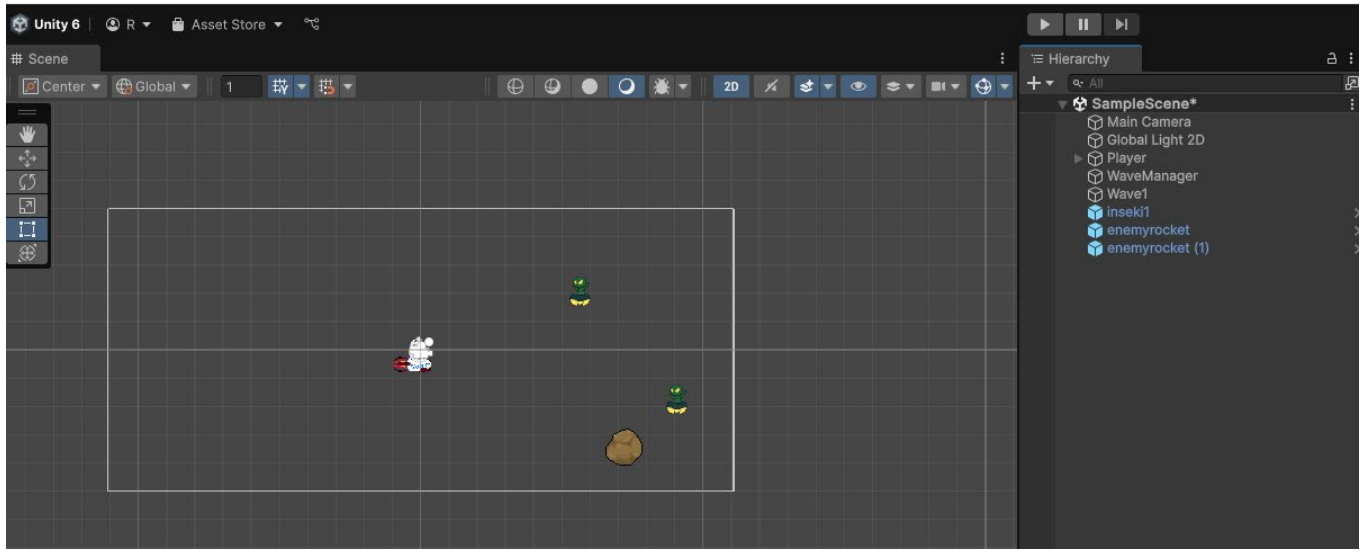
Projectタブに
Inseki1とenemyrocketを
ドラック&ドロップ

Hierarchyの敵を削除し、
Prefabの敵の位置を0にする



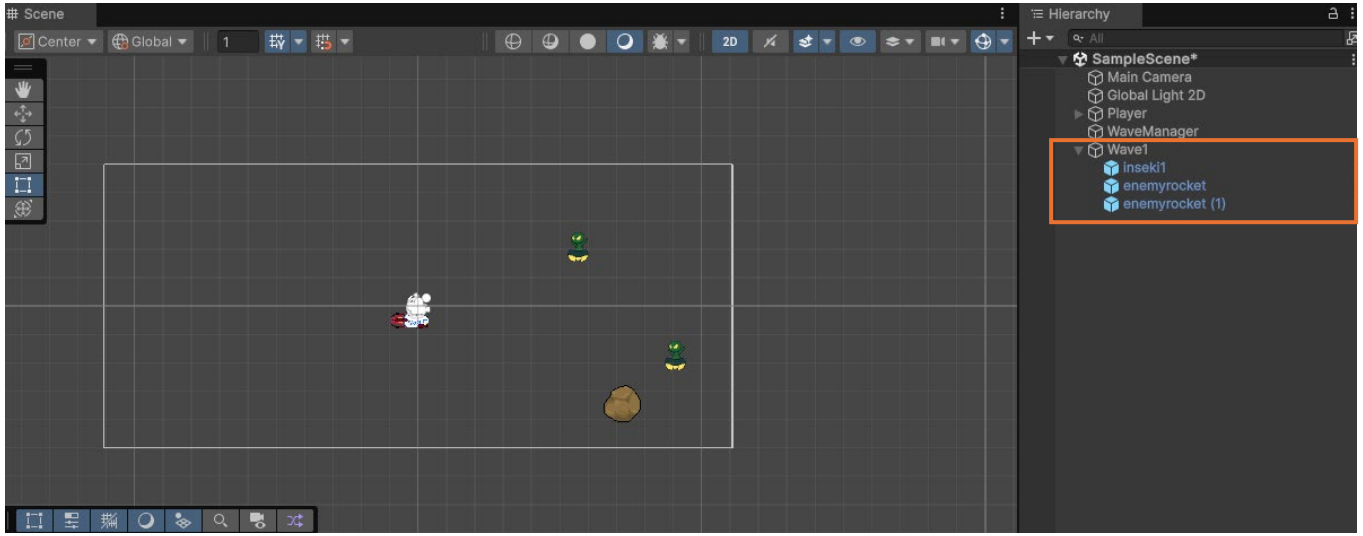
4：敵の生成

Prefabにした敵を
再度、Sceneの好きな位置に
ドラック&ドロップして
配置する



4：敵の生成

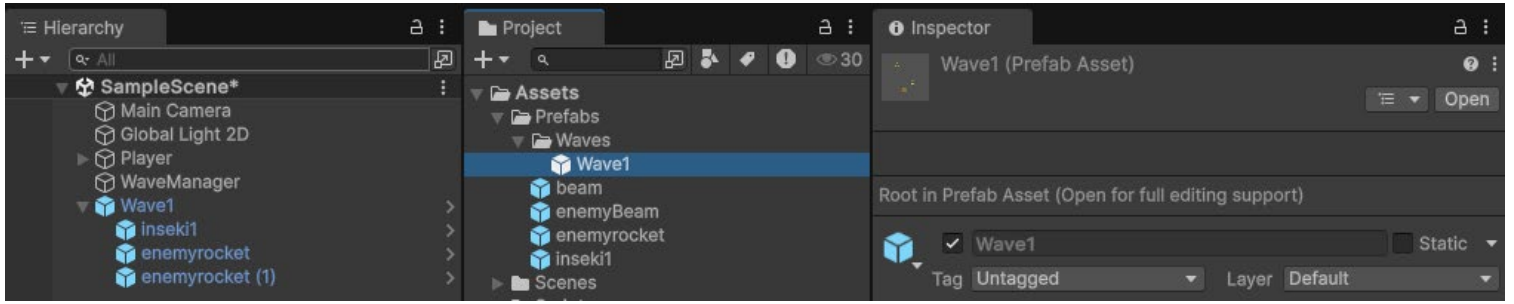
置き終わったらWave1に
全ての敵をドラック&ドロップし、
Wave1の子オブジェクトにする



4：敵の生成

Wave1をPrefabにする

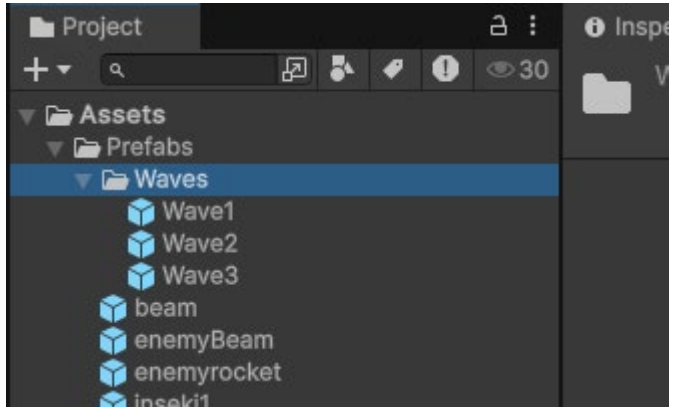
Wave1を
Prefab/Wavesに配置



4：敵の生成

そしたらHierarchyの
Wave1を削除する

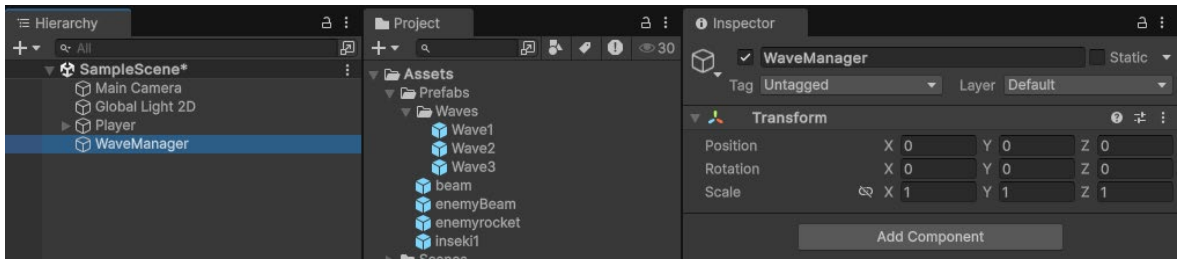
同様にしてWave2,3を
作ってみよう



4：敵の生成

Waveを作ったら
Waveを管理するために
“WaveManager”を作っていく

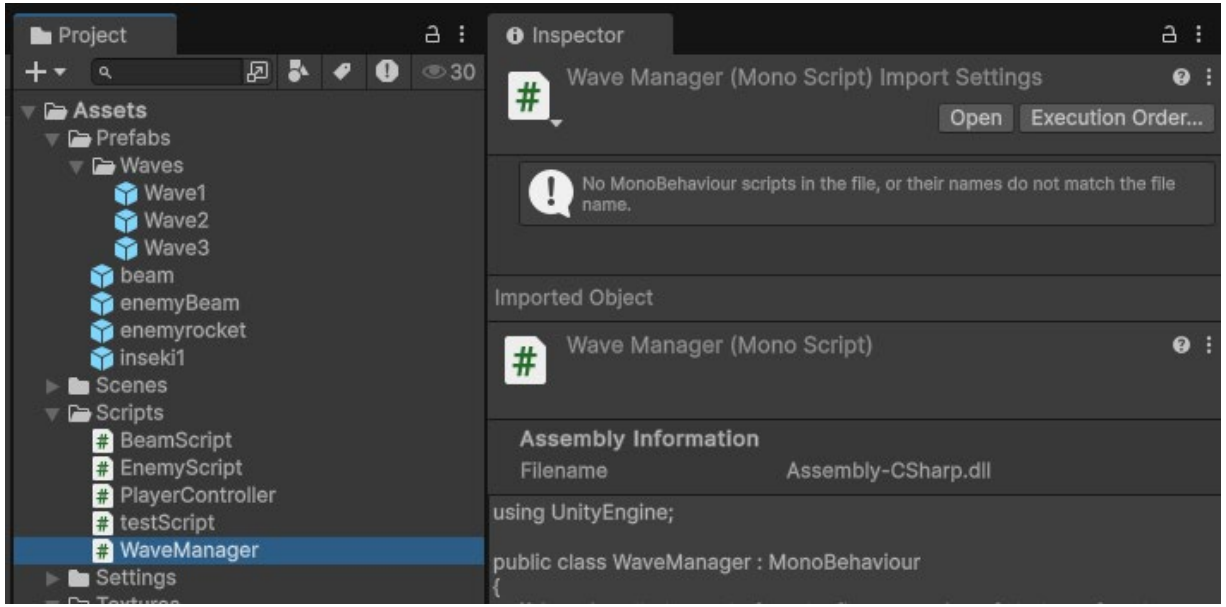
Create Emptyで作って名前を
WaveManagerにしておく



4：敵の生成

作ったら
Scriptsで右クリックし
Create→MonoBehaviourで
生成

名前は”WaveManager”に
しておく



4：敵の生成

“WaveManager”の
コードをうつす

```
public class WaveManager : MonoBehaviour
{
    public GameObject[] Waves;
    public Transform SpawnPos;
    int waveCount = -1;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ | 0 個の参照
    void Start()
    {
        waveCount = -1;
    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        int EnemyCount = GameObject.FindGameObjectsWithTag("Enemy").Length;

        if (EnemyCount == 0)
        {
            waveCount += 1;
            if (waveCount == Waves.Length)
            {
                waveCount = 0;
            }

            Instantiate(Waves[waveCount], SpawnPos.position, Quaternion.identity);
        }
    }
}
```

4：敵の生成

“WaveManager”の
コードをうつす

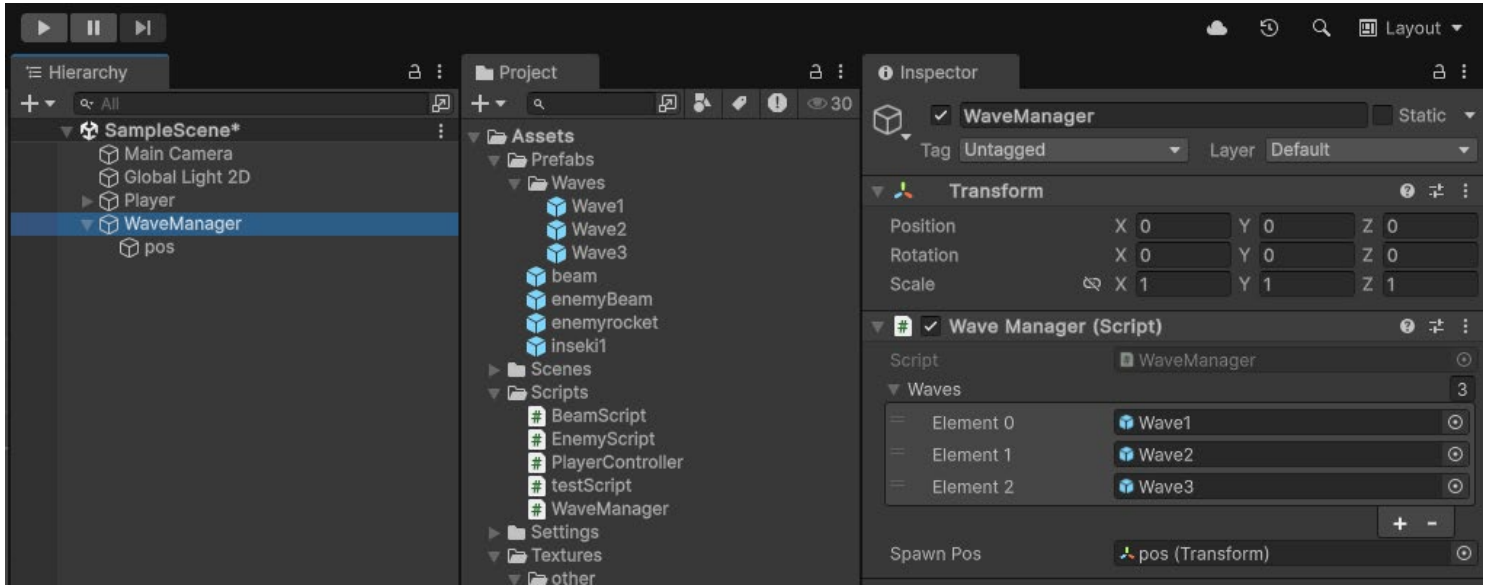
うつしたらセーブして
Unityに戻る

```
public class WaveManager : MonoBehaviour
{
    public GameObject[] Waves;
    public Transform SpawnPos;
    int waveCount = -1;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ | 0 個の参照
    void Start()
    {
        waveCount = -1;
    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        int EnemyCount = GameObject.FindGameObjectsWithTag("Enemy").Length;

        if (EnemyCount == 0)
        {
            waveCount += 1;
            if (waveCount == Waves.Length)
            {
                waveCount = 0;
            }

            Instantiate(Waves[waveCount], SpawnPos.position, Quaternion.identity);
        }
    }
}
```

4：敵の生成

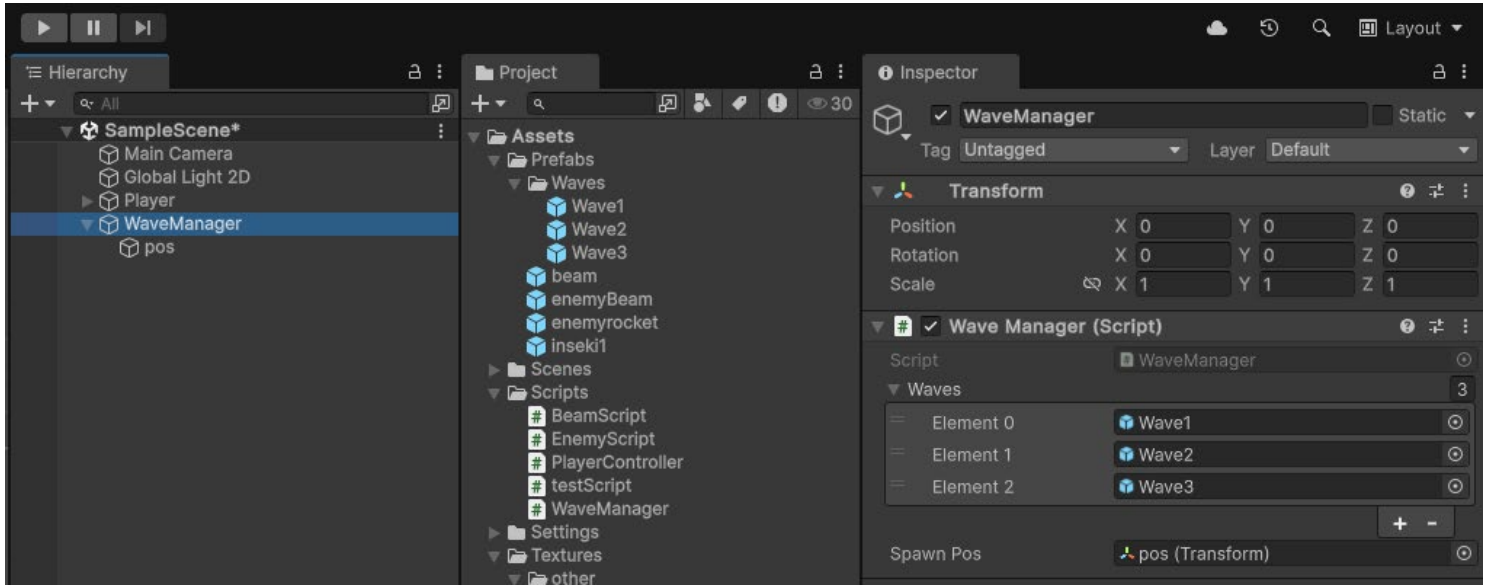
オブジェクトの
WaveManagerに
スクリプトの
WaveManagerを追加する

WavesとSpawnPosが
出てくるので設定をする



4：敵の生成

まずはWavesについて
ここにすべてのWaveを
おいておく
Waveは上から順番に
出現していく

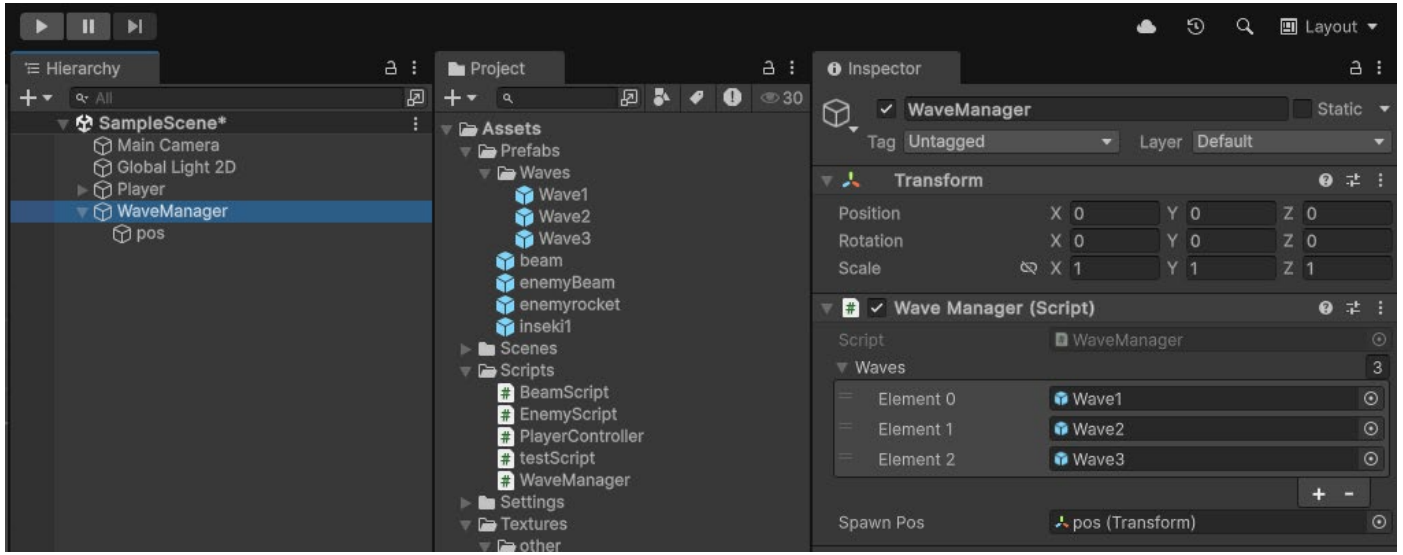


4：敵の生成

次にSpawnPosについて

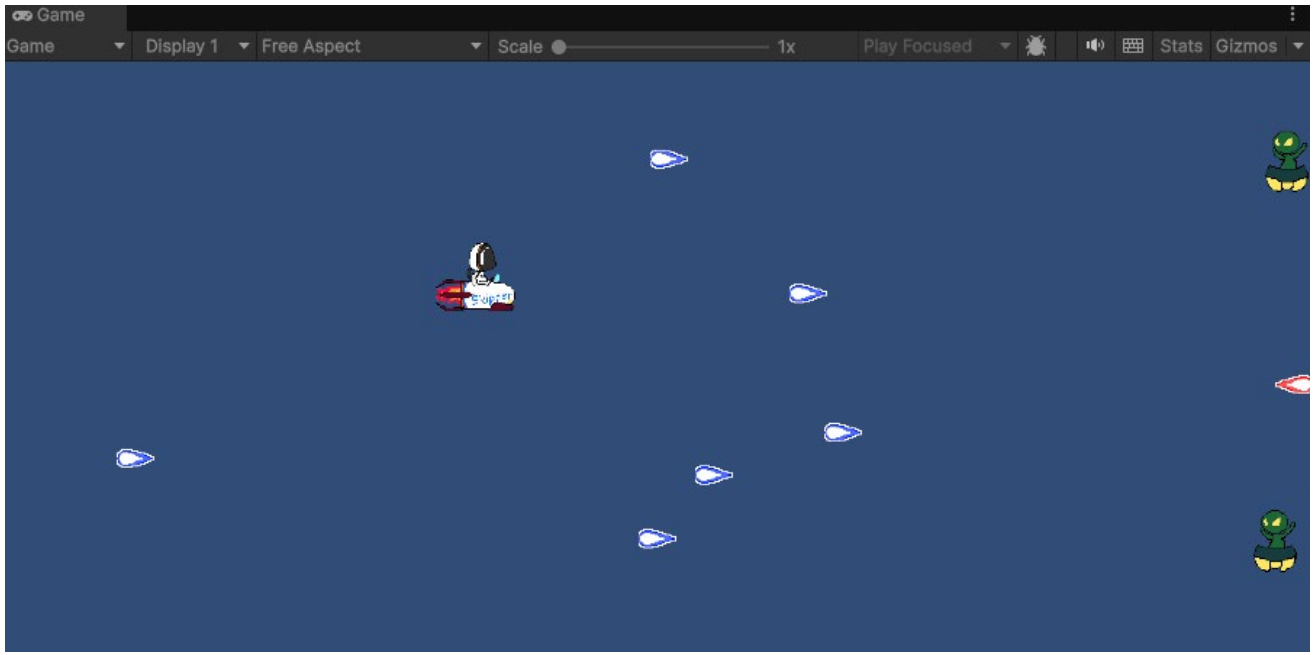
SpawnPosはWaveの
生成する位置を指定している

試しにCreate Emptyから
“pos”を作って位置を調節し、
posをSpawnPosにおいてみよう



4：敵の生成

Waveで敵が出てくる
ようになりました🎉 🎉 🎉



4：敵の生成

コードについて

型に”[]”をつけると
複数の変数、インスタンスを
入れることができる

これを”配列”とよぶ

```
public class WaveManager : MonoBehaviour
{
    public GameObject[] Waves;
    public Transform SpawnPos;
    int waveCount = -1;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ | 0 個の参照
    void Start()
    {
        waveCount = -1;
    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        int EnemyCount = GameObject.FindGameObjectsWithTag("Enemy").Length;

        if (EnemyCount == 0)
        {
            waveCount += 1;
            if (waveCount == Waves.Length)
            {
                waveCount = 0;
            }

            Instantiate(Waves[waveCount], SpawnPos.position, Quaternion.identity);
        }
    }
}
```

4：敵の生成

今回、Wavesは
GameObjectの配列と
なっているので
UnityではPrefabを
入れることができる

なお、配列の番号については
1から数え始めるのではなく
0から数え始めるものとなる

GameObject[] Waves

番号	ウェーブ
0	Wave1
1	Wave2
2	Wave3

4：敵の生成

次にUpdate内について

Int EnemyCount
＝GameObject~~.Length

Unity上で”Enemy”のTagがついた
オブジェクトの個数を代入している

```
public class WaveManager : MonoBehaviour
{
    public GameObject[] Waves;
    public Transform SpawnPos;
    int waveCount = -1;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ | 0 個の参照
    void Start()
    {
        waveCount = -1;
    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        int EnemyCount = GameObject.FindGameObjectsWithTag("Enemy").Length;

        if (EnemyCount == 0)
        {
            waveCount += 1;
            if (waveCount == Waves.Length)
            {
                waveCount = 0;
            }

            Instantiate(Waves[waveCount], SpawnPos.position, Quaternion.identity);
        }
    }
}
```

４：敵の生成

元々、FindGameObjectsWithTagというのは

「指定したTagがついているオブジェクトを全部探して
　見つけたオブジェクトの一覧を配列にしますよ！」

というもの

配列の個数は.Lengthで見れる

4：敵の生成

探してもらった結果、
その個数が0の場合
つまり敵がすべていなくなった場合

WaveCountを+1して
新しくWaveCount番号の
Waveを生成している

```
public class WaveManager : MonoBehaviour
{
    public GameObject[] Waves;
    public Transform SpawnPos;
    int waveCount = -1;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ|0 個の参照
    void Start()
    {
        waveCount = -1;
    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        int EnemyCount = GameObject.FindGameObjectsWithTag("Enemy").Length;

        if (EnemyCount == 0)
        {
            waveCount += 1;
            if (waveCount == Waves.Length)
            {
                waveCount = 0;
            }

            Instantiate(Waves[waveCount], SpawnPos.position, Quaternion.identity);
        }
    }
}
```

4：敵の生成

```
// Update is called once per frame
Unity メッセージ | 0 個の参照
void Update()
{
    int EnemyCount = GameObject.FindGameObjectsWithTag("Enemy").Length;

    if (EnemyCount == 0)
    {
        waveCount += 1;
        if (waveCount == Waves.Length)
        {
            waveCount = 0;
        }

        Instantiate(Waves[waveCount], SpawnPos.position, Quaternion.identity);
    }
}
```

もし+1しつづけて
waveCountが
Wavesの個数と一致したら
0に戻している

これは番号が0から
スタートする影響のためである
（3番目のやつってあるっけ?）

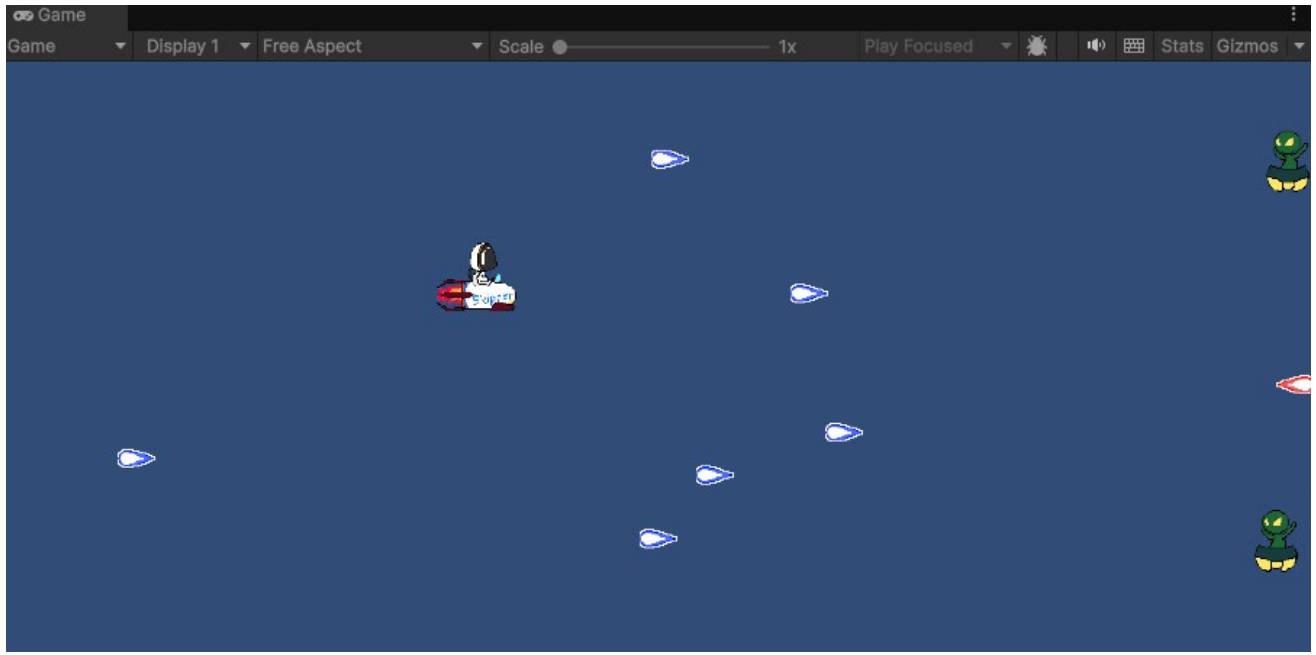
GameObject[] Waves

番号	ウェーブ
0	Wave1
1	Wave2
2	Wave3

4：敵の生成

とりあえずはゲームぽく
なりました

次回はゲームの負荷と装飾の
話になります



Unity初心者講座

2 [illegible]の導入

終わり

ここからは補足です

5：補足

Time.deltaTimeについて

Time.deltaTimeは正確に言えば
“あるフレームから別のフレームに移動する時間を代入”している値のこと

パソコンの機種によっては１秒60フレームの他に１秒１２フレームに
なったりして差が生じるため、このTime.deltaTimeを使用すると
同じような結果になる

（わからなかったら質問してください！！！　大歓迎！！！）

5：補足

OnTriggerEnterについて

IsTriggerにチェック入れた場合はOnTrigger、
チェック入れていない場合はOnCollisionを使用する

Collisionの場合はちゃんとぶつかることができるが、
Triggerの場合は通り過ぎることができる

5：補足

OnTriggerEnterについて

また、衝突が
- し始めたとき（触れたとき）
- している最中（中にいるとき）
- し終わったとき（出ていったとき）

にも場合分けがある

必要に応じて使い分けること
次回多分使います