??????????

作ったプレハブを利用し、実行中にたくさんのボールを生成しましょう。

????????`BallShooter`

今度のスクリプトは:

- 作成したプレハブを Inspector で指定できるようにする。
- マウスの左ボタンを押したら、プレハブから実際に使うオブジェクトを生成。
- 最大数を制限する。

スクリプトの変数

Scripts フォルダーの中に新しい「 BallShooter」を追加し、 Inspector の変数を準備しましょう:

```
using UnityEngine;


// ボールを生成するスクリプト
public class BallShooter : MonoBehaviour
{
    // Unityでボールのプレハブを指定
    [SerializeField]
    private GameObject ballPrefab;
}
```

当然、プレハブ変数の「型」は「 `GameObject`」になる。

??????????`Update()`

フレーム毎に、マウスボタンのクリックがあったかどうかを確認し、クリックの場合は、プレハブから実物
を生成し、実行中にシーンに追加しましょう:

```
void Update()
{
    // もしも左ボタンをクリックしたら…
    if (Input.GetMouseButtonDown(0))
    {
```

```
        // 「Instantiate」を使い、プレハブから実物（インスタンス）を作成
        // newBallは新しく作ったボールの参照である
        var newBall = Instantiate(ballPrefab);
    }
}
```

ボールを生成するのは、プレーヤーなので、「player」オブジェクト（パドル）にアタッチして、Ball Prefabの中に、前ステップで作ったボールのプレハブを設定してください。

これを追加することにより、プレーヤーは移動だけではなく、ボール生成もできるようになった!

1スクリプト　⇒1責任

??????????

スクリプトが長ければ、長いほど、バグの修正が辛くなり、ゲームの拡張性が狭まる。基礎ルールとして、「1つのスクリプトは1つの責任」の考えすると、大きな課題を解決しやすい細かい課題に分割し、それぞれが別々のスクリプトで実装する。

これで、機能を増やしたり、減らしたりするのは簡単（スクリプトを追加と削除だけ）。また、スクリプトが短くなるので、バグが発生した場合、すぐ解決できることもメリットである。

??????

- 武器を打つ: ShootWeapon
  - CPU
    プレーヤー、人間プレーヤーが関係なく、
    打つことが可能
- 体力を表示: HPGauge
  - なんでもの HP
    を表示（プレーヤー、敵、壊せるものなど
    ）
- プレーヤーを追いかける: FollowPlayer
  - 敵に付けると、プレーヤーを追いかける
  - 仲間に付けると、プレーヤーを追いかける
- アイテム: Item
  - アイテムを表す
- アイテムを拾う: ItemCollect
  - アイテム（ Item
    ）を拾ったら、インベントリに追加
- アイテムを管理: Inventory
  - アイテムどこから来たのか関係なく、アイ
    テムを管理
- アイテムを購入: ItemBuy
  - アイテム（ Item
    ）をショップで購入し、インベントリに追
    加

この状態で実行して確認してみてください。

????

いっぱいのボールが現れて楽しいが、いくつかの問題の確認ができた

1. ボールがたまに遅くなる?
2. プレーヤーの位置が関係なく、ボールがいつも同じ場所から現れる
3. ボールが多すぎる。

1. ?????????

超弾みしたのに、なんでボールが遅くなる? これは正しい物理の計算の影響であるため。ビリヤードのように、ある物体は別の物体を移動すると、速度を伝達する（ 運動量の伝達 という）

固定物体（壁）とぶつかると、強く跳ね返すが … …移動可能な物体とぶつけると、速度が移る!

これを直すには、実行中に速度を調整し、指定した「 speed」を守るようにする。「 `BallMove` 」スクリプトを調整し、フレーム毎は以下の処理にしましょう:

```
// これは「BallMove」スクリプトの更新処理
void Update()
{
    // 現在の速度ベクトルを取得。向きは保護したいが、
    // 長さ（速度）は必ず「speed」であることを保証
    var velocity = _rigidbody.linearVelocity;

    // まず、速度ベクトルの長さは「1」にする（単位ベクトル）
    velocity.Normalize();

    // そして、長さは「speed」にしよう
    velocity *= speed;

    // 速度を上書きする
    _rigidbody.linearVelocity = velocity;
}
```

2. ??????????????

ブールがプレーヤーのいる場所から発射すれば自然な遊び方になるので、 `BallShooter` のスクリプトの直そう。インスタンスを作った後に、適切な場合に配置:

```
// BallShooterのUpdateを更新:
void Update()
{
    // もしも左ボタンをクリックしたら…
    if (Input.GetMouseButtonDown(0))
    {
        //「Instantiate」を使い、プレハブから実物（インスタンス）を作成
        // newBallは新しく作ったボールの参照である
        var newBall = Instantiate(ballPrefab);

        // このスクリプトはパドルにアタッチされているので、
        // そのパドルの位置を取得
        var position = transform.position;
```

```
        // もう少し上に移動
        position.y += 0.5f;

        // ボールの位置を設定
        newBall.transform.position = position;
    }
}
```

3. ????????

このゲームでは、ボールは1個までの制限があるので、今の作りだとボールが多すぎる。これを修正するには、現在生きているボールの数を追跡すれば良いでしょう。

`BallShooter` を編集し、実行中に生成したボールの数を数えましょう。

```
using UnityEngine;

// ボールを生成するスクリプト
public class BallShooter : MonoBehaviour
{
    // Unityでボールのプレハブを指定
    [SerializeField]
    GameObject ballPrefab;

    // 現在生きているボールの数
    private int _ballCounter;

    void Update()
    {
        // もしも左ボタンをクリックしたら…
        // 「かつ」
        // ボールの数は1未満だったら
        if (Input.GetMouseButtonDown(0) && _ballCounter < 1)
        {
            //「Instantiate」を使い、プレハブから実物（インスタンス）を作成
            // newBallは新しく作ったボールの参照である
            var newBall = Instantiate(ballPrefab);
```

```
            // このスクリプトはパドルにアタッチされているので、
            // そのパドルの位置を取得
            var position = transform.position;

            // もう少し上に移動
            position.y += 0.5f;

            // ボールの位置を設定
            newBall.transform.position = position;

            // ボールを数える
            _ballCounter++;
        }
    }
}
```

4. ??????????????????????

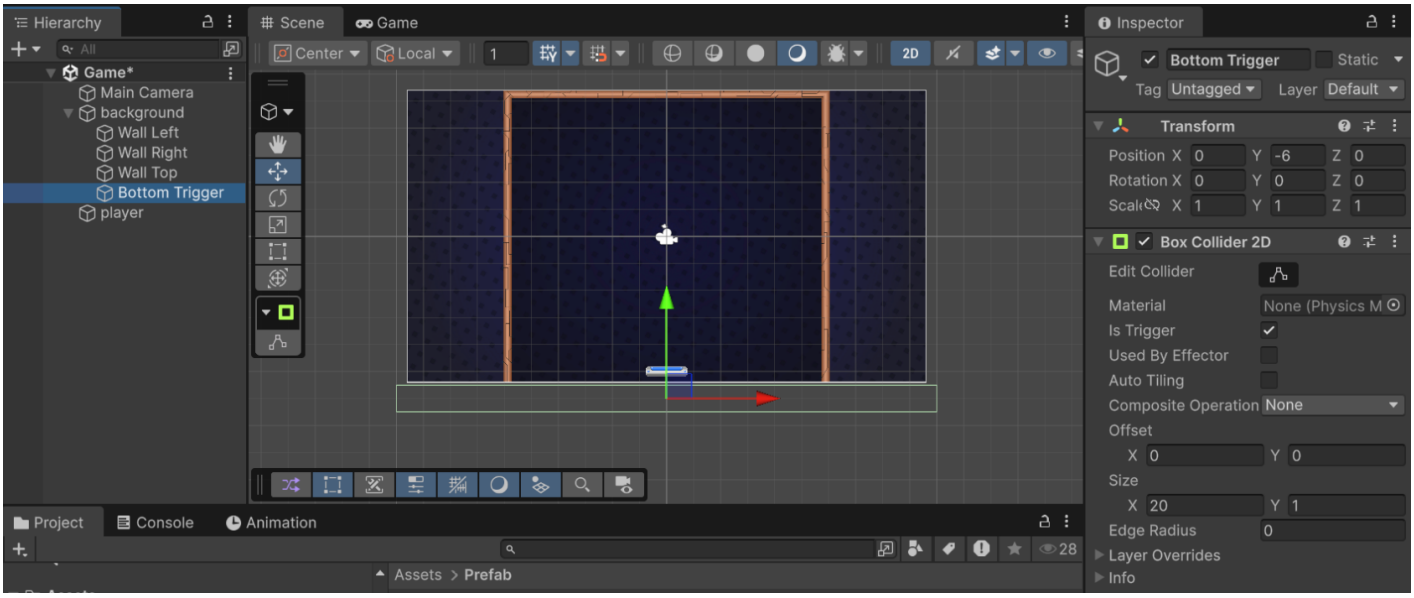
新しい問題が現れた! 確か、ボール１個までの制限ができたが、ボールがなくなっても、次のボールを作れない! 実は、ボールがなくなっていない!

実行中にシーンビューを確認しましょう:

カメラで見えないだけで、ヒエラルキーにも、画面にもボールがあることが確認できる。これを解決するには「トリガー」を使いましょう。画面の真下に透明なトリガーを作成し、ボールが通過したら、ボールを削除し、ボールが失ったことをパドルにお知らせしましょう。

まず、シーンをの設定。 background の中に、空のゲームオブジェクトを作成し、 BoxCollider2D を追加してください。サイズが、画面の下のすべてをカバーするようにしてください。

「 Is Trigger」のチェックを忘れずに！



次「ボールが通過したら、ボール削除し、プレーヤーに通知する」処理を追加しましょう。新しい「 BallDestroy」スクリプトを作成してください。

using UnityEngine;

// ボールを削除し、お知らせする
public class BallDestroy : MonoBehaviour
{
// BallShooterに知らせる！
[SerializeField]
private BallShooter ballShooter;

// トリガーの中に何か入ったら：
private void OnTriggerEnter2D(Collider2D other)
{
// ゲームオブジェクトを廃止（削除）する
Destroy(other.gameObject);

// メソッドを呼び出し、知らせる
ballShooter.BallDestroyed();
}
}

これで実現できるが、 BallShooter 側でお知らせを受け取るメソッド「 BallDestroyed 」をまだ作っていないので、追加しましょう：

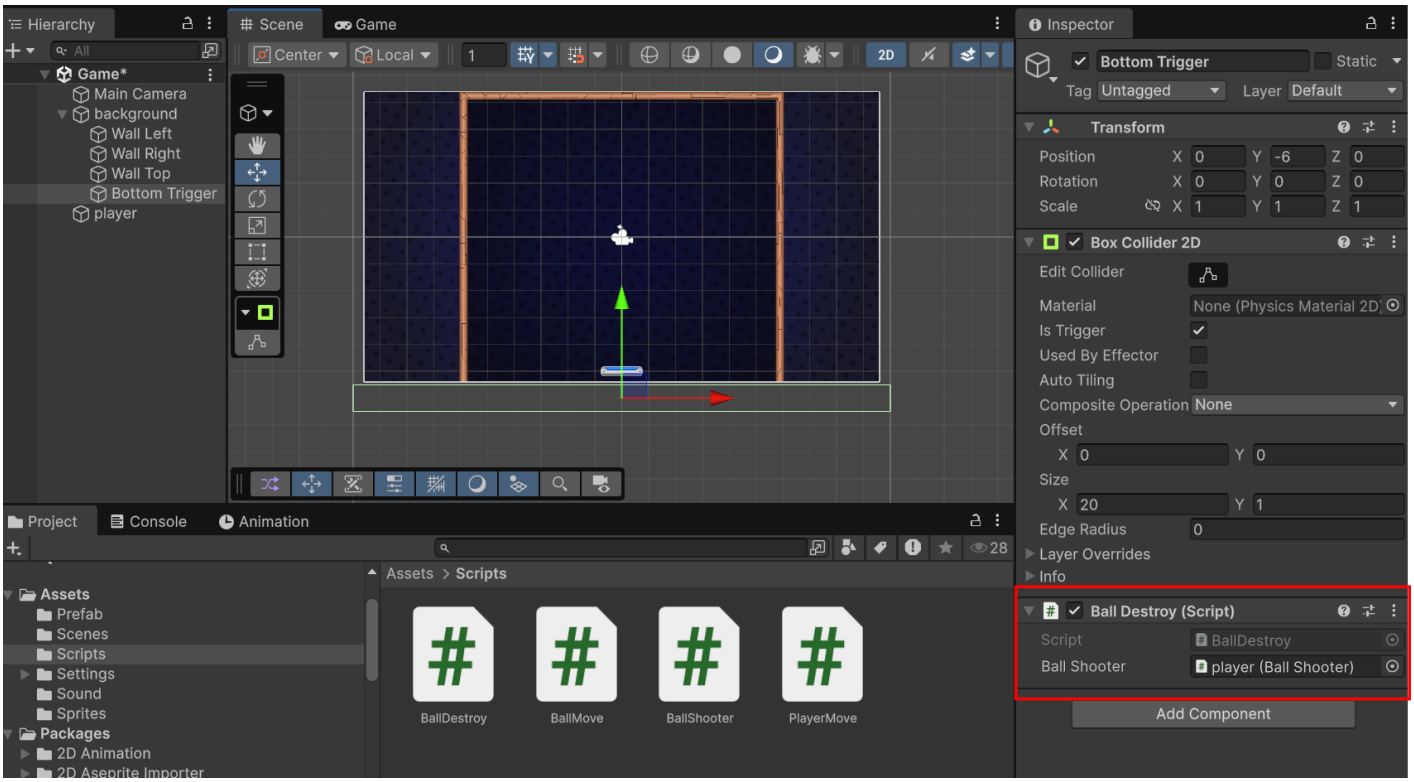
```
using UnityEngine;

// ボールを生成するスクリプト
public class BallShooter : MonoBehaviour
{
    // 省略（変わりがない）

    // ボール廃止されたときに呼び出される
    // 他のスクリプトから呼び出されるので、「public」アクセスにしましょう
    public void BallDestroyed()
    {
        // 最低はゼロであることを確かめる
        if (_ballCounter > 0)
        {
            // 数を減らす
            _ballCounter--;
        }
    }

    void Update()
    {
        // 省略（変わりがない）
    }
}
```

最後、シーンのトリガーに「BallDestroy」スクリプトを追加し、Inspector での必要な連携を設定してください:



Revision #3
Created 2026-04-20 06:07:53 UTC by Menendez Francisco
Updated 2026-04-21 06:33:09 UTC by Menendez Francisco