???????

次、「ゲームクリア」と「ゲームオーバー」の処理を追加しましょう。

???????

このゲームでは、残機３つがあり、ボールが失ったら残機１つが減る。残機がゼロになった場合は「ゲームオーバー」にする。すでに「ボールが失ったら ...」（ `BallDestroy` ）という処理があるので、これを活用できる。

ゲーム監督クラス「 `GameManager` 」を作成し、残機の数を追跡しましょう。 `BallDestroy` のとき、 `GameManager` を呼び出し、残機を減らす処理を実装してみましょう:

```
// ゲーム監督
public class GameManager : MonoBehaviour
{
    // 残機の数
    private int _lives;

    private void Start()
    {
        _lives = 3; // 3個で始まる
    }

    // ボール廃止されたときに呼び出される
    public void BallDestroyed()
    {
        // 残機が減る
        _lives--;
        if (_lives == 0)
        {
            // とりあえず、コンソールで表示
            Debug.Log("ゲームオーバー");
        }
    }
}
```

そして、 `BallDestroy` 側では ...

```
// 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;

    // GameManagerにも知らせる!
    [SerializeField]
    private GameManager gameManager;

    // 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();
        gameManager.BallDestroyed();
    }
}
```

シーンにゲーム監督スクリプトを新しいゲームオブジェクトにアタッチし、 `BallDestroy` の参照を付けてください:

これで3つのボールをなくして、「ゲームオーバー」になることを確認してください。

BallShooter ?????

ゲームオーバーになっても、次々にボールを投げることができる! これではいけないので、`GameManager`
側でゲームが終わったことがわかれば、`BallShooter`を無効化しましょう:

GameManager に Inspector で参照を付けれるようにする:

```
// ゲーム監督
public class GameManager : MonoBehaviour
{
    // ボールを投げる処理
    [SerializeField]
    private BallShooter ballShooter;

    // あとは同じ…
}
```

そして、ゲームオーバーの時:

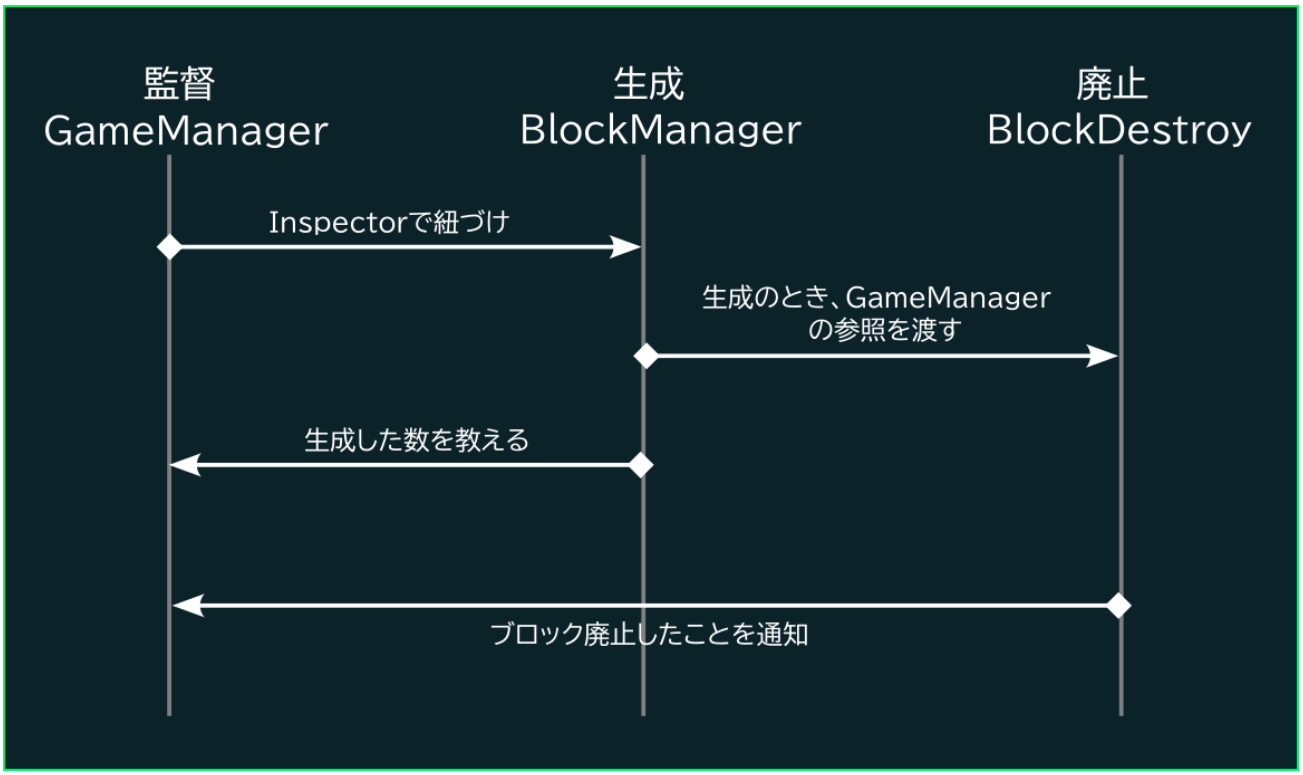
```
if (_lives == 0)
{
    // とりあえず、コンソールで表示
    Debug.Log("ゲームオーバー");

    // スクリプトを無効化する
    ballShooter.enabled = false;
}
```

Unityで参照を設定し、実行してみてください。

??????

このゲームでは、すべてのブロックを削除できれば、ゲームクリアとする。ただし、`GameManager`
がブロックが存在することわからず、実行中生成されるものであるため、Inspector
上での参照を付けることができない!

`BlockManager`がブロックを生成するので、新しいブロックを生成した場合は、`BlockDestroy`が
`GameManager`に通知するように、スクリプトの中で連携を作れば良いでしょう!



GameManager
まず GameManager
から行きましょう。生成された数を受け取るメソッドとブロック廃止の時に呼び出されるメソッドを準備しましょう：

// ゲーム監督
public class GameManager : MonoBehaviour
{
// ボールを投げる処理
[SerializeField]
private BallShooter ballShooter;

// 残機の数
private int _lives;

// ブロックの数
private int _blockCount;

// Start() と BallDestroyed() はそのまま

// ブロックの数を設定
public void SetBlockCount(int count)

```
    {
        _blockCount = count;
    }

    // ブロック廃止のとき呼び出される
    public void BlockDestroyed()
    {
        _blockCount--;
        if (_blockCount == 0)
        {
            // コンソールで表示し、ボール投げれなくなる
            Debug.Log("ゲームクリア");
            ballShooter.enabled = false;
        }
    }
}
```

BlockDestroy

つぎ、ブロック廃止のときに、 `GameManager` に連絡しましょう:

```
// ボールが当たったら、ブロックを消す処理
public class BlockDestroy : MonoBehaviour
{
    // ゲーム監督の参照
    private GameManager _gameManager;

    // ゲーム監督の参照を設定
    public void SetGameManager(GameManager gameManager)
    {
        _gameManager = gameManager;
    }

    // 衝突したら…
    private void OnCollisionEnter2D(Collision2D other)
    {
        // 自分のことを削除
        Destroy(gameObject);
```

```
        // ゲーム監督に通知
        _gameManager.BlockDestroyed();
    }
}
```

BlockManager

最後、中間のスクリプトですべてを繋げましょう:

```
// ブロックの配置を管理する
public class BlockManager : MonoBehaviour
{
    // ブロックのプレハブ
    [SerializeField]
    private GameObject blockPrefab;

    // ゲーム監督クラスの参照をInspectorで設定
    [SerializeField]
    private GameManager gameManager;

    // 行の数
    [SerializeField]
    private int rows;

    // 列の数
    [SerializeField]
    private int cols;

    void Start()
    {
        // プレハブのSpriteRendererを求める
        SpriteRenderer spriteRenderer = blockPrefab.GetComponent<SpriteRenderer>();

        // そこからspriteのサイズを取得
        Vector2 blockSize = spriteRenderer.size;

        // 始まる位置は自分の位置にする
        Vector2 start = transform.position;

        // 各行を表示
```

```
        for (int j = 0; j < rows; j++)
        {
            // 各列を表示
            for (int i = 0; i < cols; i++)
            {
                // すらず量（ブロック幅 × i, ブロック高さ × j）
                // y軸の場合は↓の方に進むので、「-」を付ける!
                Vector2 offset = new Vector2(blockSize.x * i, -blockSize.y * j);

                // インスタンスを作成し、配置
                GameObject copy = Instantiate(blockPrefab);
                copy.transform.position = start + offset;

                // ブロック廃止スクリプトにGameManagerの参照を渡す
                BlockDestroy blockDestroy = copy.GetComponent<BlockDestroy>();
                blockDestroy.SetGameManager(gameManager);
            }
        }

        // ゲーム監督に生成したブロック数を教える
        gameManager.SetBlockCount(rows * cols);
    }
}
```

最後に、　　Unityで参照を設定し、確認しましょう!

※

：確認しにくい場合は、一旦ゲームオーバーの処理をコメントアウトでなしにして、無制限にボールを使えるようにしてください。

Revision #4
Created 2026-04-27 06:36:41 UTC by Menendez Francisco
Updated 2026-04-27 07:38:21 UTC by Menendez Francisco