# ゲームプログラミング2021

## 第5回：スクリプトによるオブジェクト生成とその制御法

北坂　孝幸

# 今日やること

## ❖ブロック崩し（Breakout）を作る

### ▶球，ブロックのPrefab化

➢スクリプトでオブジェクトを生成・配置できるように

- ついでに物理特性等のパラメータも編集

### ▶スクリプトの記述

➢球，ブロック，バーの挙動

➢ブロックの自動生成

➢残り球の管理

# 覚えて欲しいこと

- オブジェクトへのアクセス方法（探し方）
  - GameObject.Find(オブジェクト名)
  - GameObject.FindGameObjectsWithTag(タグ名)
- Prefab化のやり方と呼び出し方
- オブジェクトの諸パラメータの変更方法
  - GetComponent<要素>().パラメータ
- 他のC#スクリプトの変数・関数へのアクセス
  - GetComponent<クラス名>.変数名（又は関数名）

# 動作の設定

- 球
  - BallControl.cs
    - 球が作られた時にある方向に弾く
  - Ballオブジェクトに紐付け

```
public class BallControl : MonoBehaviour {

    public float speed=3f;  //球の速度

    // Use this for initialization
    void Start () {
        Vector3 v = new Vector3 (-1, 0, 1);
        GetComponent<Rigidbody> ().AddForce (v*speed
, ForceMode.VelocityChange);
    }

    // Update is called once per frame
    void Update () {
    }
}
```

- GetComponent<Rigidbody>().AddForce(方向,モード)
  - モード
    - Force: 質量加味の力付加, Acceleration: 質量無視の加速度付加
    - Impulse: 質量加味の速度付加, VelocityChange: 質量無視の速度付加

# バーのキーボード操作 (BarControl.cs)

```
public class BarControl : MonoBehaviour {

    GameObject ground;  //床オブジェクト

    // Use this for initialization
    void Start () {
        ground = GameObject.Find ("Ground");
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetKey (KeyCode.LeftArrow)) {
            // 壁を突き抜けないように制限
            if(transform.position.z-transform.localScale.z/2 > -ground.transform.localScale.z/2)
                transform.Translate (0f, 0f, -0.1f);  //左(z軸負方向)に少し動かす
        }
        if (Input.GetKey (KeyCode.RightArrow)) {
            // 壁を突き抜けないように制限
            if(transform.position.z+transform.localScale.z/2 < ground.transform.localScale.z/2)
                transform.Translate (0f, 0f, 0.1f);  //右(z軸正方向)に少し動かす
        }
    }
}
```

- ❖オブジェクトの検索
  - ▶GameObject.Find(名前)
- ❖オブジェクトの位置(中心)
  - ▶transform.position
- ❖オブジェクトの大きさ
  - ▶transform.localScale

# 実行

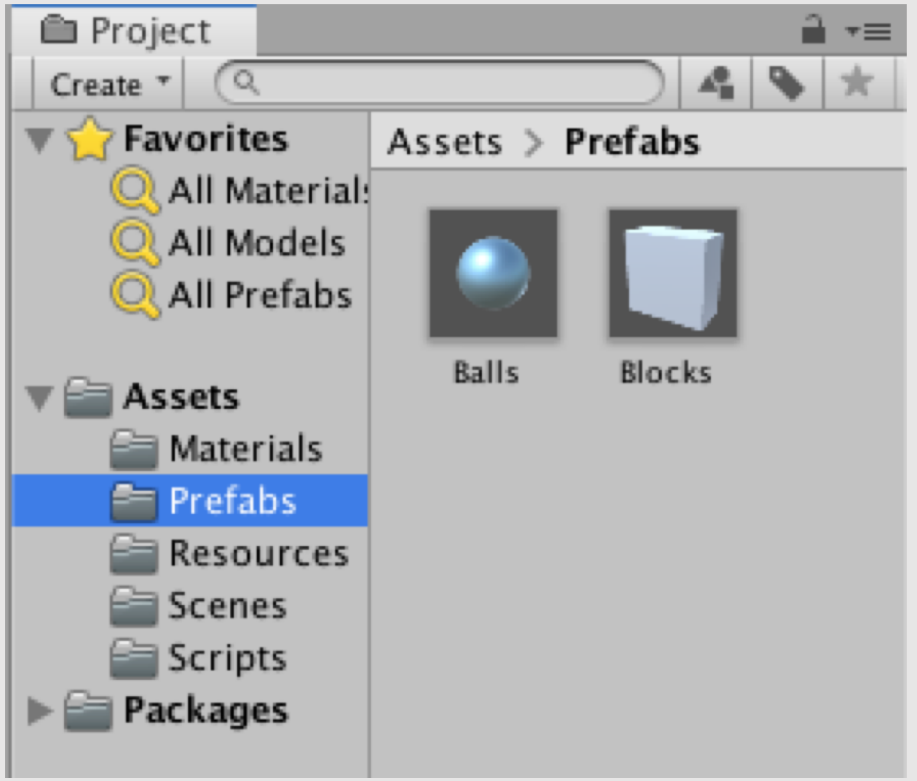
# ブロックの消去

- ブロックにぶつかったら消去
  - BallControl.csの修正
- 実行してブロックが消えることを確認

```
public class BallControl : MonoBehaviour {

      . . . .

    // Update is called once per frame
    void Update () {

    }

    void OnCollisionEnter(Collision obj){
        if (obj.gameObject.tag == "Block")
{
            Destroy (obj.gameObject, 0f);
        }
    }
}
```

# 球とブロックのPrefab化

- <Assets>を右クリック→<Create>→<Folder>
  - ▶名前を“Prefabs”に
- 球のPrefab化
  - ▶”Ball”オブジェクトを<Prefabs>にドラッグ＆ドロップ
    - ➢名前を “Balls”に変更
- ブロックのPrefab化
  - ▶”Block”オブジェクトを<Prefabs>にドラッグ＆ドロップ
    - ➢名前を “Blocks”に変更
- ”Ball”と”Block”はDelete



# ゲームマネージャの設定

## ❖ゲームの管理を司るScript

- ➢ゲーム開始時の挙動
- ➢複数のオブジェクトにまたがる制御
- ➢ゲームの終了

▶<GameObject> - <CreateEmpty>

- ➢空の目に見えないオブジェクトを生成
- ➢名前を “GameManager”に

▶<Assets> - <Create> <C# Script>

- ➢名前を“GameManager”にする
- ➢“GameManager”オブジェクトに紐付け

# GameManager.cs

❖ゲーム起動時に球とブロックを生成

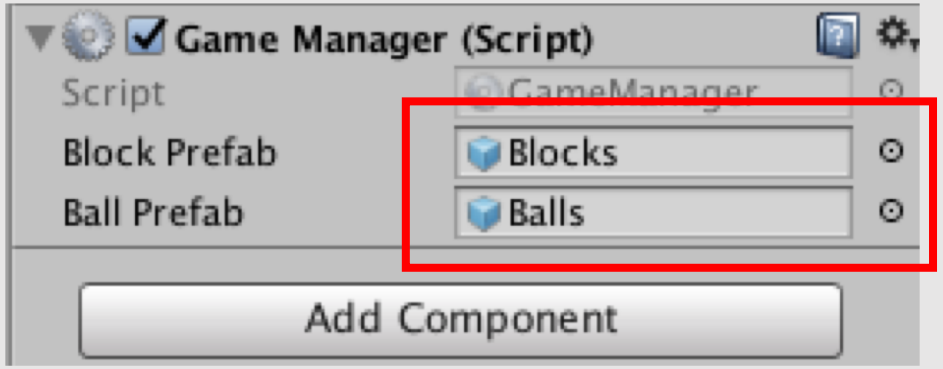
▶プレハブから

➢Instantiate(プレハブ, 位置, 姿勢) as GameObject;

キャスト

GameObjectクラスとして返す

▶GameManagerのインスペクタに対応するプレハブをドラッグ＆ドロップ（忘れずに！）




```
public class GameManager : MonoBehaviour {

    public GameObject blockPrefab;
    public GameObject ballPrefab;
    GameObject block;
    GameObject ball;

    // Use this for initialization
    void Start () {
        block = Instantiate (blockPrefab,
new Vector3(-5f, 1f, 0f),
Quaternion.identity) as GameObject ;
        ball = Instantiate (ballPrefab, tr
ansform.position, Quaternion.identity) as
GameObject ;
    }

    // Update is called once per frame
    void Update () {

    }
}
```

# 動作チェック

❖指定した位置に，ブロックと球が出現

▶球は生成された瞬間にAddForceにより弾かれる

▶球に紐付けたスクリプト(BallControl.cs)もプレハブに梱包される

❖あと，ゲームとして成り立たせるには

▶Start，Quit（→次回）

▶ブロックの自動配置

▶クリアエフェクト（→次回）

▶ゲームオーバー　（→次回）

# ブロックの自動配置 (in GameManager.cs)

## ❖変数の宣言

```
public class GameManager : MonoBehaviour {

    public GameObject blockPrefab;  //ブロックプレハブ
    public GameObject ballPrefab;  //ボールプレハブ
//    GameObject block;  //配列で確保するのでコメントに
    GameObject ball;
    GameObject[,] blocks;   //ブロック用２次元配列
    public int Columns=5;  //ブロックの横の数
    public int Rows=5;  //ブロックの縦の数
    Vector3 blockSize;  //ブロックのサイズ
    Vector3 origin;  //ブロック配置の原点

    // Use this for initialization
    void Start () {
```

# ブロックの自動配置 (in GameManager.cs)

## ❖start()の編集

```
    // Use this for initialization
    void Start () {
//        block = Instantiate (blockPrefab, new Vector3(-
5f,1f,0f), Quaternion.identity) as GameObject;
        /* 床の大きさからブロックのサイズを決定 */
        GameObject ground = GameObject.Find("Ground");  //床オブジェクトを探す
        Vector3 g_size = ground.transform.localScale;  //長いので短い変数に入れ直し
        blockSize.x = blockPrefab.transform.localScale.x;
        blockSize.y = blockPrefab.transform.localScale.y;
        blockSize.z = g_size.z / Columns;  //床の横サイズ÷ブロックの列数
        origin = new Vector3 (-g_size.x/2, blockSize.y/2, -g_size.z/2);  //原点

        //ブロックの自動生成
        GenerateBlocks ();

        //ボールの生成
        ball = Instantiate(ballPrefab, transform.position, Quaternion.identity) as GameObject;
    }
```

# ブロックの自動配置 (in GameManager.cs)

## ❖GenerateBlocks()の作成

```
    // Update is called once per frame
    void Update () {
    }

    //ブロックジェネレータ
    void GenerateBlocks(){
        blocks = new GameObject[Rows, Columns];  //ブロック用配列の生成
        for (int row = 0; row < blocks.GetLength(0); row++) {
            for (int col = 0; col < blocks.GetLength(1); col++) {
                Vector3 v = origin + new Vector3(blockSize.x/2+blockSize.x*row, 0f, blockSize
.z/2+blockSize.z*col);
                blocks [row, col] = Instantiate (blockPrefab, v, Quaternion.identity) as Game
Object;
                blocks [row, col].transform.localScale = blockSize*0.9f;  //みっちり敷き詰めると
境界が見えないので，少し小さく
            }
        }
    }
}
```

次元0　次元1

C#での配列の大きさの取得方法
GetLength(次元番号)

# 動作確認

❖ブロックが自動的に指定した数で配置される

❖ブロックの色を行ごとに変えてみよう

▶オブジェクト(例えばobj)のコンポーネントへのアクセス

➢obj.GetComponent<要素>().パラメータ

- （色の場合）obj.GetComponent<**Renderer**>().**material**

▶Unityで作った色に設定するにはResourcesを利用

➢Resources.Load(“アセット名”) as 型

- “**Resources**”というフォルダを“Assets”の下に作成
  - “magenta”や“red”などのmaterialを追加
  - `Material mat;`
  - `mat = Resources.Load("magenta") as Material;`
- あとはmatをセット　　**拡張子は不要**
  - `blocks[row,col].GetComponent<Renderer>().material = mat;`

**PrefabやテクスチャなどもResourcesフォルダに入れておけば，同様に読み込める**

# 球の発射方法を変える

- 現状：球の生成と同時に射出
- 変更：ランダムな方向に射出する関数Strike()を作成
  - ▶変更 (in BallControl.cs)

```
// Use this for initialization
void Start () {
}
    . . .
void OnCollisionEnter(Collision obj){
    . . .
}

public void Strike(){
    Vector3 v = new Vector3 (Random.Range(-1f,-0.3f), 0f, Random.Range(-1f,1f));
    v = v.normalized;
    GetComponent<Rigidbody> ().AddForce (v*speed, ForceMode.VelocityChange);
}
```

Start()に書いてあったものは消す（あるいはコメントに）

# 球の生成方法の変更 (in GameManager.cs)

❖スペースキーを押したらStrike()を呼び出して発射

```
public class GameManager : MonoBehaviour {
        ・・・
    // Use this for initialization
    void Start () {
        ・・・

        //ボールの生成
        GenerateBall();
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetKey (KeyCode.Space)) {
            if (ball != null) {
                ball.GetComponent<BallControl> ().Strike ();
            }
        }
    }
```

新しくGenerateBall()を作成

# 球の生成方法の変更 (in GameManager.cs)

## ❖球の生成

```
    //ブロックジェネレータ
    void GenerateBlocks(){
        ・・・
    }

    //ボールジェネレータ
    void GenerateBall(){
        //バーの位置を探す
        GameObject bar = GameObject.Find("Bar");  //バーオブジェクトを探す
        float r = ballPrefab.transform.localScale.x/2;    //球の半径
        float bar_thickness = bar.transform.localScale.x / 2;    //バーの厚みの半分
        //球がバーの上にちょうど来るように配置
        Vector3 p = bar.transform.position + new Vector3 (-bar_thickness-r,0,0);
        ball = Instantiate (ballPrefab, p, Quaternion.identity) as GameObject;
    }
}
```

# 課題

## ❖１０～２０秒程度のプレイ動画を提出

▶以下を含めること

➢ブロックの色を行ごとに変える

➢スペースキーで玉をランダムな方向に射出

▶できる人は以下も含める

➢ブロックを全て壊したら「GAME CLEAR !!」と表示

➢ヒント

- GameManager.cs
  - DecreaseBlock()を作成
    - blockCountを一つ減らす
    - blockCountが０になったら “GameClear()”を呼びだす
  - GameClear()を作成
    - “GAME CLEAR !!”と表示
- BallControl.cs
  - ブロックを壊したら，DecreaseBlock()を呼び出してカウントを減らす

## ❖余裕のある人は以下も実装

# Advanced（詳しくは次回）

- ライフの管理とゲームクリアー/オーバーイベントの処理

- GameManager.csのインクルード設定を変更
  - ▶テキストやボタンなどを利用できるように

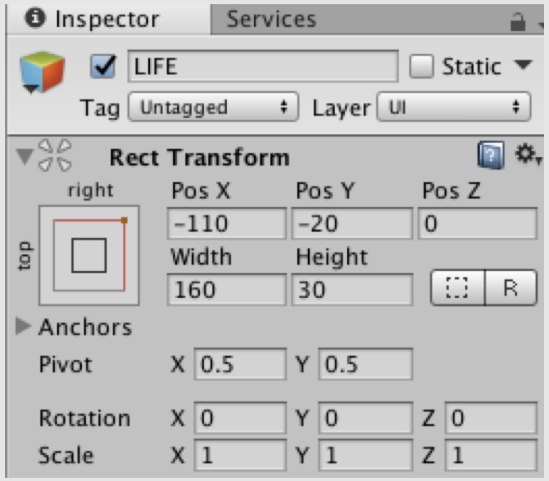
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
```

# ゲーム画面に文字を表示

## ❖ライフ

▶<GameObject> - <UI> - <Text>

➢"LIFE"という名前に

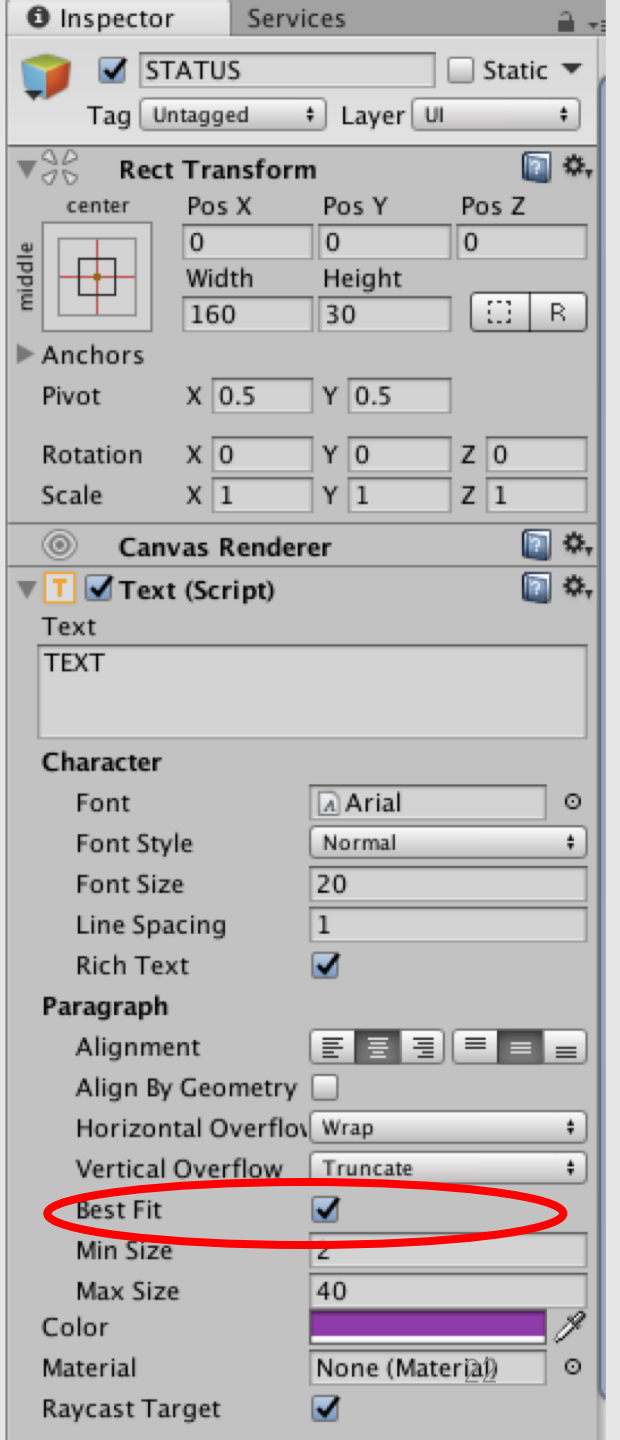



## ❖ゲームステータス

▶<GameObject> - <UI> - <Text>

➢"STATUS"という名前に

- 文字列が長くなった時に自動的に枠内に収めるために，"Best Fit"にチェックしておく



# LIFEの管理 (in GameManager.cs)

## ❖変数の宣言

```
    GameObject lifeText;
    GameObject statusText;
    int life=3;  //ライフ
    bool endFlag=false;  //ゲームオーバー用フラグ
    bool clearFlag=false;  //ゲームクリア用フラグ
    int blockCount;   //残ブロック数を覚えておく変数
```

## ❖start()の編集

```
    void Start () {
        ・・・
        blockCount = columns * rows;   //残ブロック数を初期化
        //LIFEの表示
        lifeText = GameObject.Find("LIFE");
        this.lifeText.GetComponent<Text> ().text = "LIFE : " + life;

        //GameTextの設定
        statusText = GameObject.Find("STATUS");
        this.statusText.GetComponent<Text> ().text = "PRESS Space¥r¥n to START";
    }
```

# LIFEの管理 (in GameManager.cs)

## ❖Update()の編集

```
    void Update () {
        if (Input.GetKey (KeyCode.Space)) {
            if (ball != null) {
                this.statusText.SetActive (false);
                ball.GetComponent<BallControl> ().Strike ();
            } else {  //ボールの作り直し
                GenerateBall();
            }
        }
    }
```

球が動き出したら，STATUSは非表示に

球が無かったら，作り直し

# LIFEの管理 (in GameManager.cs)

## ❖DecreaseLife()の作成

```
    public void DecreaseLife(){
        life--;
        this.lifeText.GetComponent<Text> ().text = "LIFE : " + life;
        if (life == 0) {
            endFlag = true;
            GameOver ();
        }
    }
```

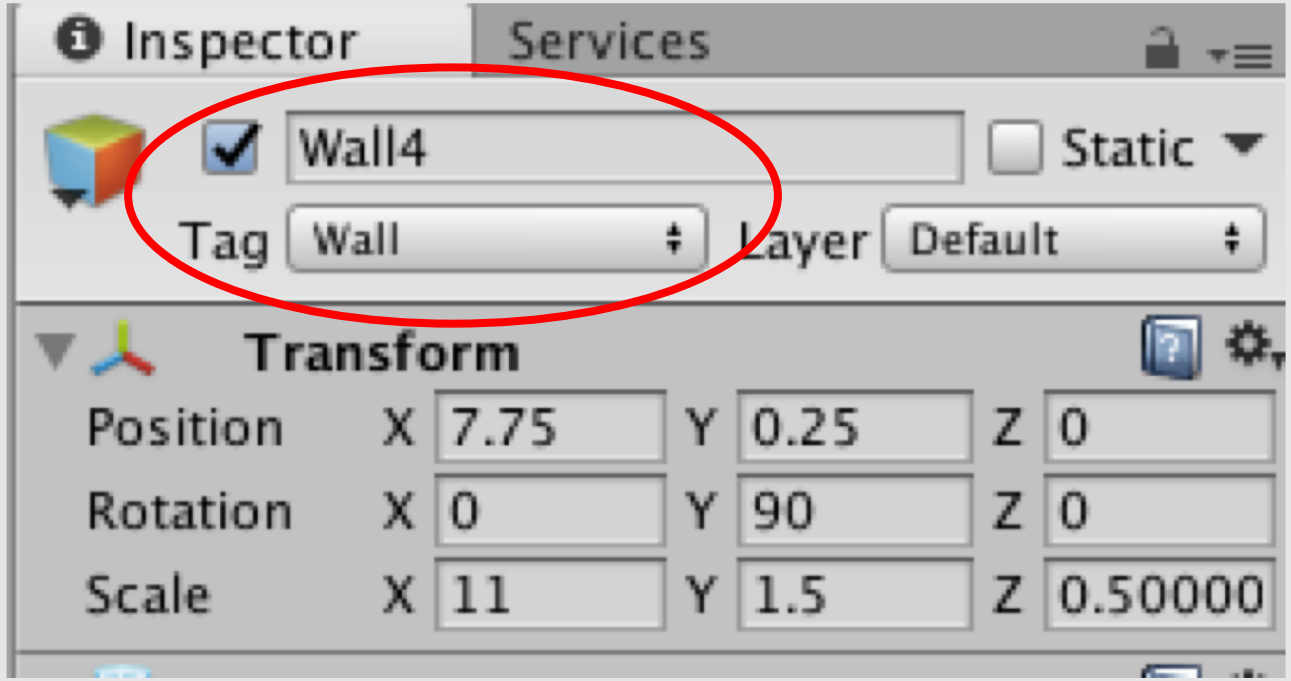
## ❖GameOver()の作成　余裕のある人は作る．詳しくは次回

```
    public void GameOver(){
        //テキスト表示
        Debug.Log("Game Over!");
        statusText.GetComponent<Text> ().text = "GAME OVER!!";
        statusText.SetActive (true);
    }
```

# 壁の設定

❖下の壁に球がぶつかったら，球を壊す

▶壁Wall4に “Wall”タグを付与



# 球の挙動修正 (in BallControl.cs)

```
public class BallControl : MonoBehaviour {

    public float speed=3f;  //球の速度
    GameObject gm;   //ゲームマネージャ

    // Use this for initialization
    void Start () {
        gm = GameObject.Find (“GameManager”); ;   //ゲームマネージャをセット
    }

    // Update is called once per frame
    void Update () {
    }
```

# 球の挙動修正 (in BallControl.cs)

```
    void OnCollisionEnter(Collision obj){
        if (obj.gameObject.tag == "Block") {
            Destroy (obj.gameObject, 0f);
        }else if (obj.gameObject.tag == "Wall") { //"Wall"に当たったら
            Debug.Log("hit the wall");
            Destroy (this.gameObject, 0f); //球（自分自身）を壊す
                gm.GetComponent<GameManager> ().DecreaseLife(); //gmの
DecreaseLife()呼出
        }
    }

    public void Strike(){
        . . .
    }
}
```