Unity講座
～ブロック崩し③～

3年　　海苔　威

1. クリア時の初期化

1-1. スコアの初期化

- 初期化でせっかくのスコアが消えるのは
  もったいないので最高得点を保存しよう
- 「S02_Score」を開く
- 右のように変更
- Reset_scoreの下にOnGUIが続いてます
- 出来たら保存

```
using UnityEngine;
using System.Collections;

public class S02_Score : MonoBehaviour {

    private static int score = 0;
    private static int max_score = 0;
    private bool flag = false;
    public GUIStyle gui_score;

    public void Additional_score(int value)
    {
        score += value;
    }

    public void GameClear_score()
    {
        if (max_score < score)
        {
            max_score = score;
        }
        flag = true;
    }

    public void Reset_score()
    {
        score = 0;
    }
```

```
void OnGUI() {
    if (!flag)
    {
        GUI.Label(new Rect(0, 0, Screen.width - 10, 30), "SCORE : " + score, gui_score);
    }
    else
    {
        GUI.Label(new Rect(0, Screen.height - 100, Screen.width - 100, 30), "  今回の得点：" + score, gui_score);
        GUI.Label(new Rect(0, Screen.height - 50, Screen.width - 100, 30), "  過去最高得点：" + score, gui_score);
    }
}
```

- GameClear_score関数は呼ばれるとmax_scoreとscoreを比較し
  max_scoreがsocreより小さい、つまり今の得点が最高得点を更新している場合
  max_scoreの値をscoreの値に変更する
  上の処理とは別にflagをtrueにする（trueだとゲームクリア状態となる）

- Reset_score関数は呼ばれるとscoreの値を0にする

- OnGUi関数はflagがtrueかfalseで表示する内容を分岐する
- falseなら現在の得点を、trueなら今回の得点と過去最高得点を表示する

- 作成した関数を呼び出す部分は「S03_ending」に書く

- 「S03_ending」を開く
- 右のプログラムを書く

- Start関数でGameClear_scoreを呼び、
  最高得点の更新を行う

- Update関数で5秒経過したときに
  Reset_scoreを呼びscoreを0にして初期化した後
  titleシーンを呼び出す

- これでスコアを初期化してタイトル画面に戻れる

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class S03_ending : MonoBehaviour {
    private float step_timer = 0.0f;
    private S02_Score s02_score;

    void Start()
    {
        s02_score = GetComponent<S02_Score>();
        s02_score.GameClear_score();
    }


    void Update () {
        step_timer += Time.deltaTime;
        if (step_timer > 5.0f)
        {
            s02_score.Reset_score();
            SceneManager.LoadScene(0);
        }
    }
}
```

1-2. ステージ管理番号の初期化

- ゲームクリア時のstage_noは3のため、2回目以降は1ステージ目のクリアで
  エンディングに遷移してしまう
- stage_noもクリア時に初期化する必要がある

- stage_noを管理する「S02_Stage」はブロック生成もしている
  →クリア時にstage_noを1にすると1ステージ目と認識してブロック生成をする
  かもしれない…

- こうならないようにするには
  →endingシーンでstage_noを初期化したらそれ以降は
  「S02_Stage」を読まないようにする

- まず「S02_Stage」を開く
- OnGUIの前あたりに関数を追加する

```
public void stageNo_clear()
{
    stage_no = 1;
}

void OnGUI()
{
```

- このstageNo_clearを呼ぶことでステージ番号を初期化する
- 「S03_ending」を開く
- 赤枠追加

```
public class S03_ending : MonoBehaviour {
    private float step_timer = 0.0f;
    private S02_Score s02_score;
    private S02_Stage s02_stage;

    void Start()
    {
        s02_stage = GetComponent<S02_Stage>();
        s02_stage.stageNo_clear();
        s02_stage.enabled = false;

        s02_score = GetComponent<S02_Score>();
        s02_score.GameClear_score();
    }
```

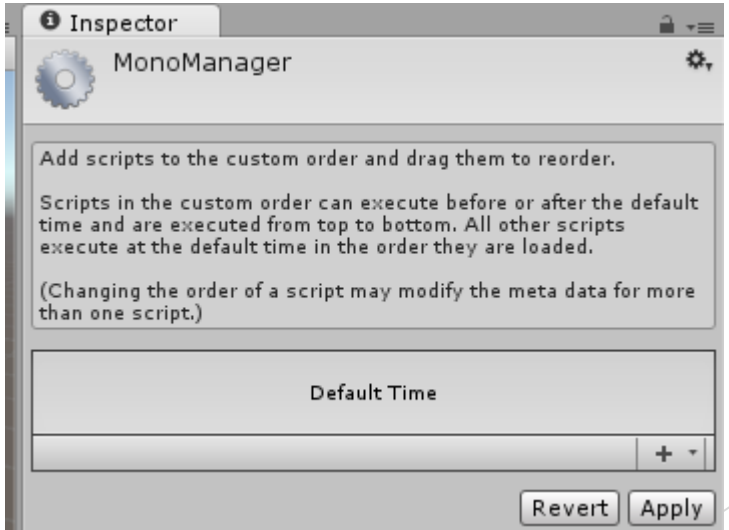
- stageNo_clearで初期化した後
  enabledをfalseにする
  →これはInspectorのチェックボックス
  を外すのをプログラム上で行っている
  のと同じ

- ここまでで初期化はできる
- けど、endingシーンで「S02_Stage」と「S03_ending」の内どちらのStart関数が先に呼ばれるかはランダム

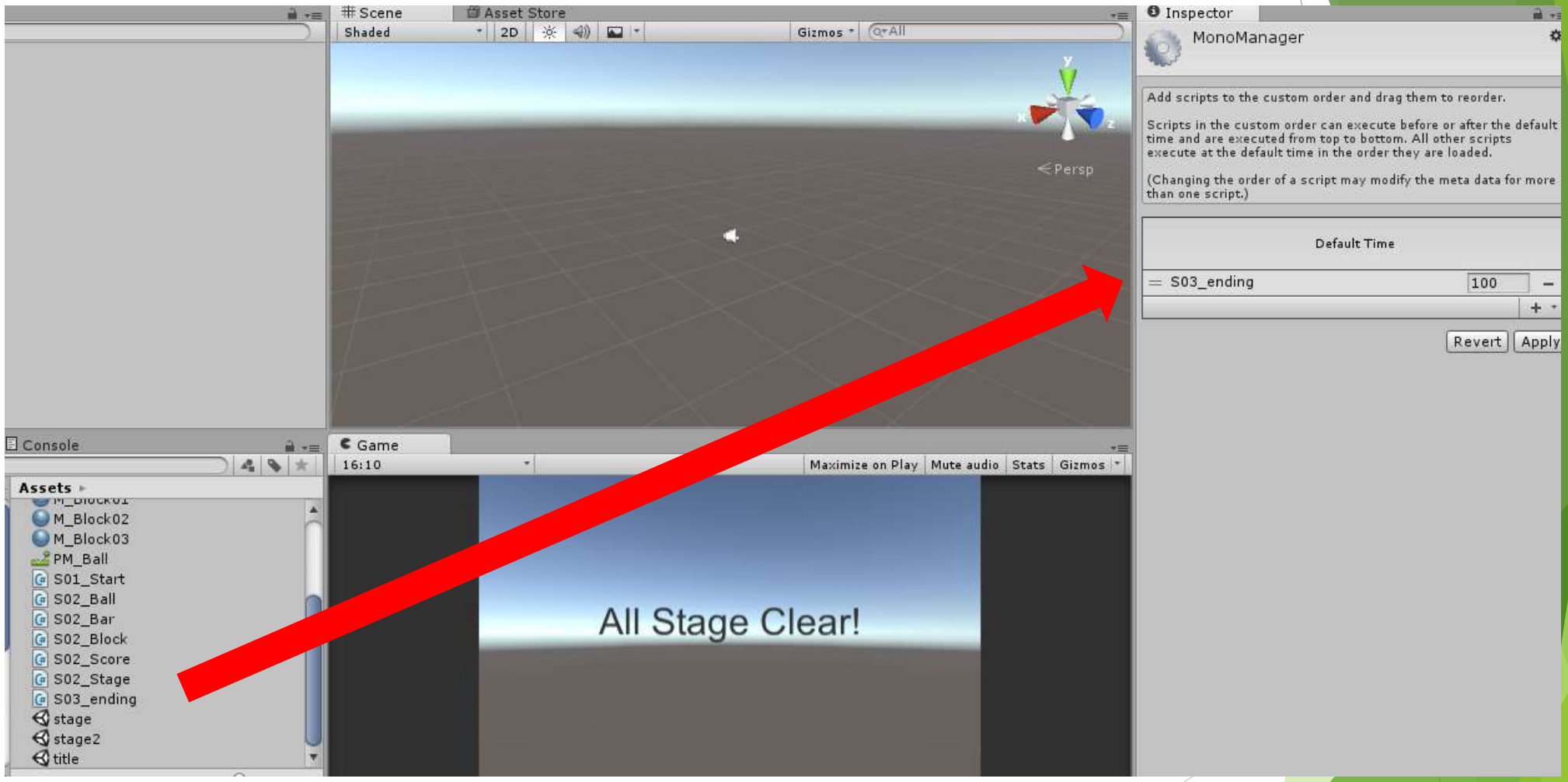
  →同一オブジェクト（今回はGameRoot）にあるスクリプトのStart関数は呼ばれる順番がランダム

- 特定のスクリプトのStart関数を優先する方法
- 「ending」シーンを開いて
- 「Edit」→「Project Setting」→「Script Execution Order」

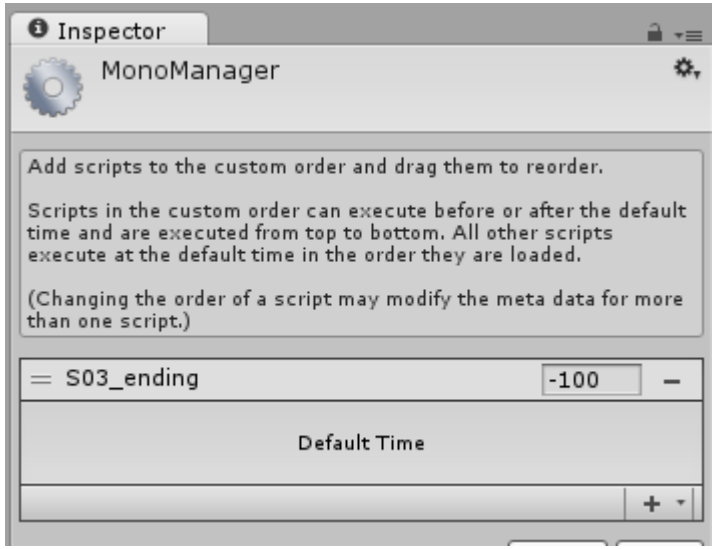
- Inspectorがこう変化する



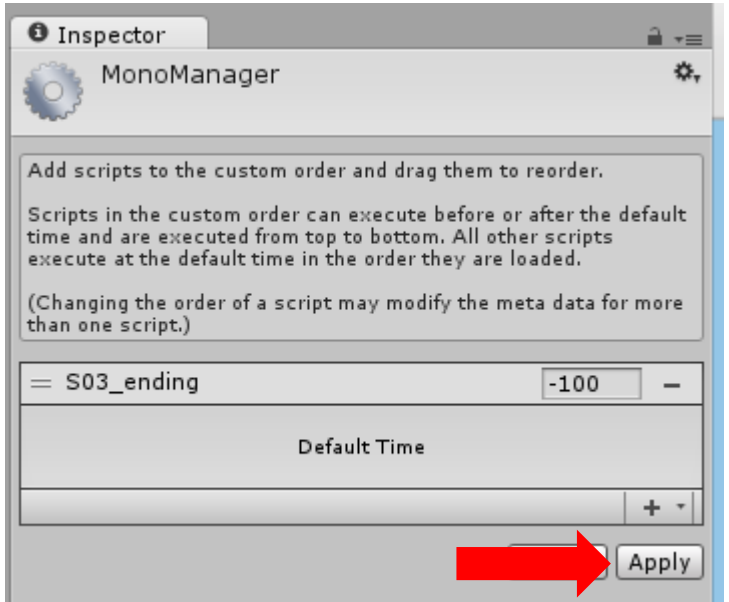
- Assetsの「S03_ending」をドラッグ＆ドロップ



- 「S03_ending」をDefault Timeの上に移動



- これで「S03_ending」のStart関数が優先される。できたらApplyをクリック



- Assetsの「S02_Stage」をHierarchyの「GameRoot」に追加
- ゲーム実行

- 各ステージにブロックが生成されるとデバッグしにくいので
  →のように1ブロックだけ生成するように
  　書き換えとくとラク

```
private void blockSetting1()
{
    GameObject block_right1 = GameObject.Instantiate(prefab[0]) as GameObject;
    block_right1.transform.position = new Vector3(18.0f , 19.0f, 0.0f);
    /*for (int i = 0; i < 4; i++)
    {
        GameObject block_right1 = GameObject.Instantiate(prefab[0]) as GameObject;
        GameObject block_right2 = GameObject.Instantiate(prefab[1]) as GameObject;
        GameObject block_right3 = GameObject.Instantiate(prefab[2]) as GameObject;
        GameObject block_left1 = GameObject.Instantiate(prefab[0]) as GameObject;
        GameObject block_left2 = GameObject.Instantiate(prefab[1]) as GameObject;
        GameObject block_left3 = GameObject.Instantiate(prefab[2]) as GameObject;

        block_right1.transform.position = new Vector3(3.0f + (5 * i), 19.0f, 0.0f);
        block_right2.transform.position = new Vector3(3.0f + (5 * i), 21.0f, 0.0f);
        block_right3.transform.position = new Vector3(3.0f + (5 * i), 23.0f, 0.0f);
        block_left1.transform.position = new Vector3(-3.0f - (5 * i), 19.0f, 0.0f);
        block_left2.transform.position = new Vector3(-3.0f - (5 * i), 21.0f, 0.0f);
        block_left3.transform.position = new Vector3(-3.0f - (5 * i), 23.0f, 0.0f);
    }*/
}

private  void blockSetting2()
{
    GameObject block_right1 = GameObject.Instantiate(prefab[0]) as GameObject;
    block_right1.transform.position = new Vector3(18.0f, 19.0f, 0.0f);
    /*for (int i = 0; i < 3; i++){
        for(int j = 0; j < 4; j++)
        {
            GameObject block_r = GameObject.Instantiate(prefab[i]) as GameObject;
            GameObject block_l = GameObject.Instantiate(prefab[i]) as GameObject;
```

2. バーの挙動

2-1. スクリプトの改善

- 左右キーだけでなく他のキーでも動くように
- 今の書き方だと追加が大変…
- 拡張しやすい形に書き換える

- KEYという構造体を用意しあるキーが押されたらその値を格納する、というような関数を自作し使用する

- 意味わかんないとおもうので実際に書いてみる

2-2. 実際に書く

- 「S02_Bar」を開く
- もはや全て書き直しレベル

```
public class S02_Bar : MonoBehaviour {

    private struct KEY
    {
        public bool left;
        public bool right;
        public bool moveUP;
    }

    private KEY key;
    private float MAX_X = 18.5f;
    private float MIN_X = -18.5f;
    private int speed = 10;

    void Update()
    {
        getKeyInput();
        setSpeed();

        if (key.left&&transform.position.x>MIN_X)
        {
                transform.position += Vector3.left * speed * Time.deltaTime;
        }
        else if (key.right&&transform.position.x<MAX_X)
        {
                transform.position += Vector3.right * speed * Time.deltaTime;
        }
    }
```

- さらに続く
- できたら保存

- ゲーム実行

- 「S02_Stage」のブロック生成をコメントにしたままの人は解除して試す

- 左右のシフトキーを押しながら移動すると速い！
- マウス持ってる人は左右クリックだけで動く！

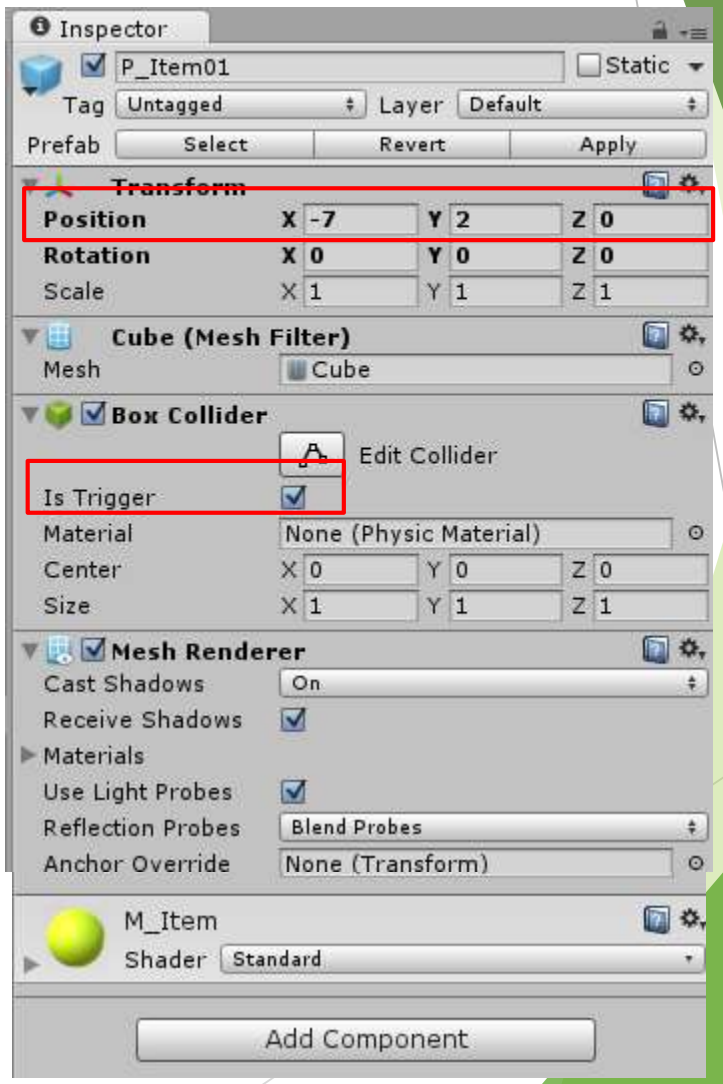
```
private void getKeyInput()
{
    key.left = false;
    key.right = false;
    key.moveUP = false;

    key.left |= Input.GetKey(KeyCode.LeftArrow);
    key.left |= Input.GetKey(KeyCode.F);
    key.left |= Input.GetMouseButton(0);

    key.right |= Input.GetKey(KeyCode.RightArrow);
    key.right |= Input.GetKey(KeyCode.J);
    key.right |= Input.GetMouseButton(1);

    key.moveUP |= Input.GetKey(KeyCode.LeftShift);
    key.moveUP |= Input.GetKey(KeyCode.RightShift);
}

private void setSpeed()
{
    if (!key.moveUP)
    {
        speed = 10;
    }
    else
    {
        speed = 20;
    }
}
```

3. アイテムを追加

3-1. アイテム(仮)とバーの設定

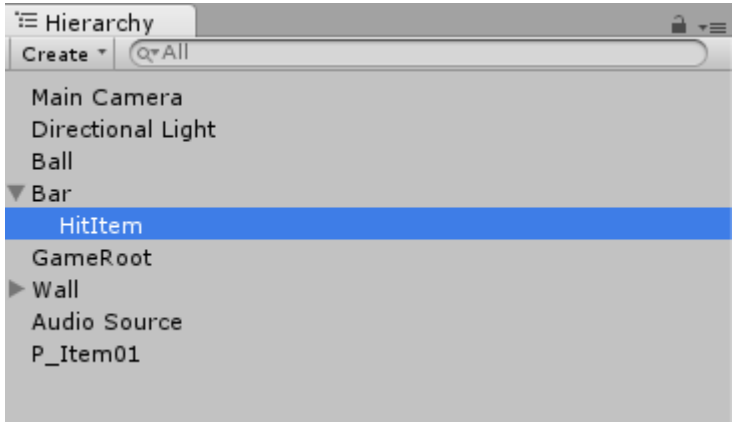
- まずはアイテムとなるオブジェクトを作る
- 「stage」シーンを開いて
- 「GameObject」→「3D Object」→「Cube」
- 名前は「P_Item01」
- Inspectorを開いて赤枠変更

- 適宜「Assets」→「Create」→「Material」で色でも付けといてください



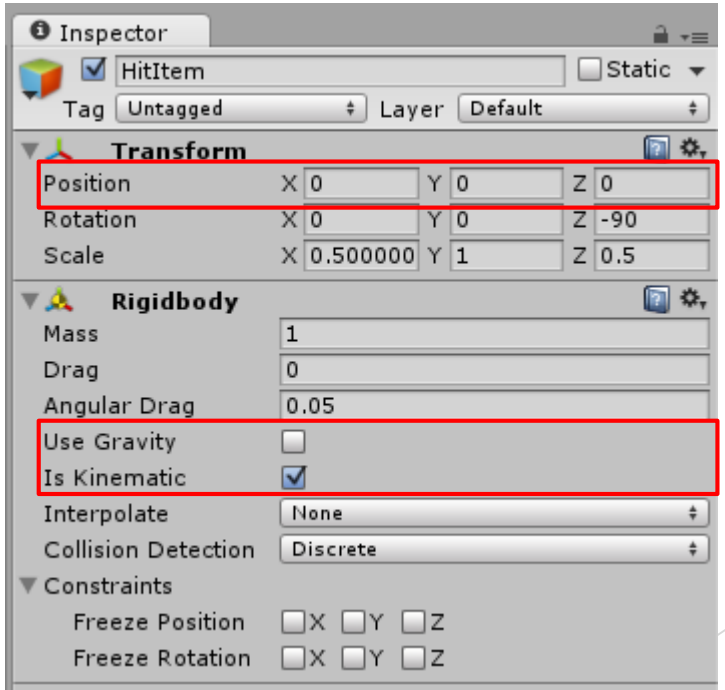
- バーにアイテムが接触した時に取る、ようにしたい
- バーにRigidbodyを直接持たせると変な動きをする

  →空のオブジェクトを作ってRigidbodyを持たせる

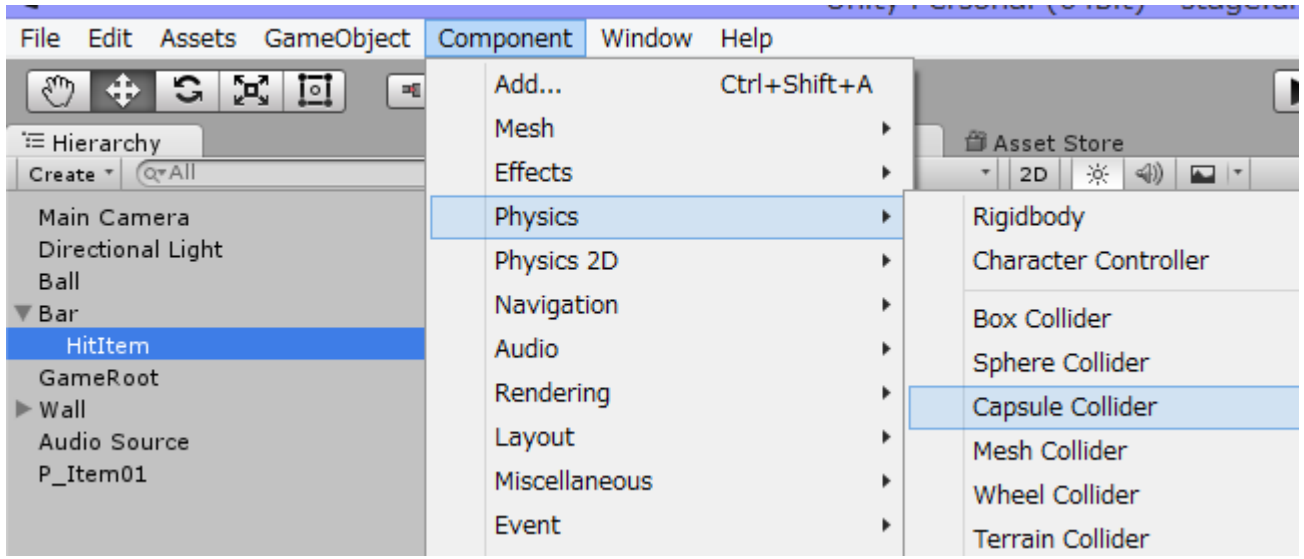


- 「GameObject」→「Create Empty」
- 名前は「HitItem」
- Hierarchyの「Bar」にドラッグ＆ドロップ

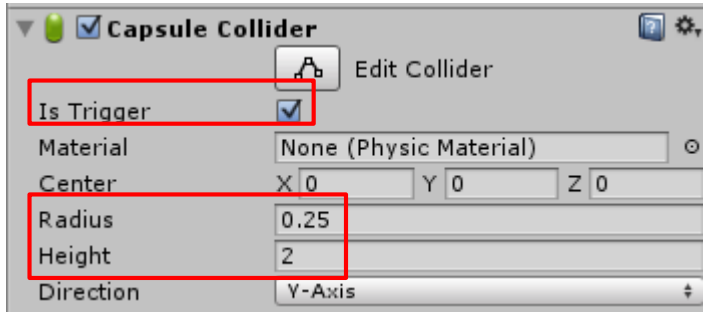
- 「HitItem」にRigidBodyを追加
- 赤枠変更



- 続けて「HitItem」にCapsule Colliderを追加
- 「HitItem」を選択した状態で「Component」→「Physics」→「Capsule Collider」



- 赤枠変更

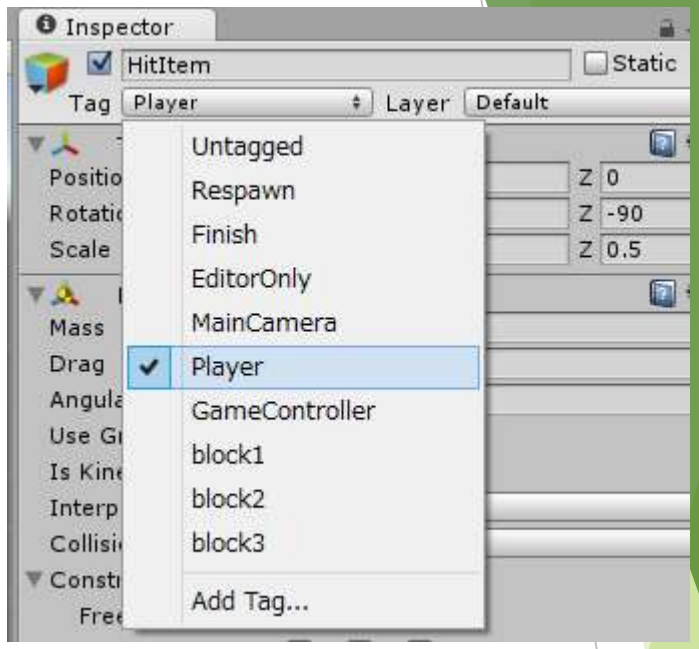


3-2. アイテムの動き

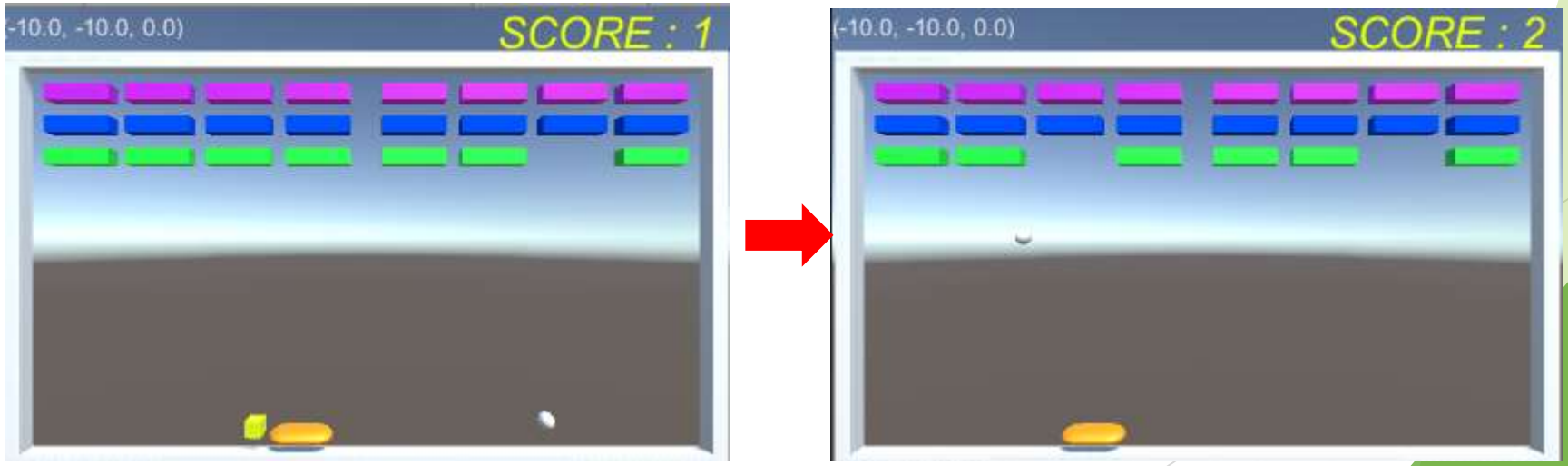
- アイテムの動きはプログラムで制御
- 「Assets」→「Create」→「C# Script」
- 名前は「S02_Item」
- 「P_Item01」にドラッグ＆ドロップ
- →書く

```
using UnityEngine;
using System.Collections;

public class S02_Item : MonoBehaviour {

    void Update () {
        transform.Rotate(Vector3.up * 90 * Time.deltaTime);

        if (transform.position.y < -1.0f)
        {
            Destroy(gameObject);
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            Destroy(gameObject);
        }
    }
}
```

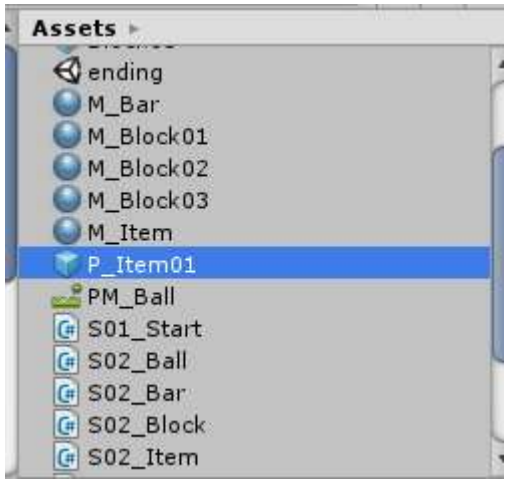
- 書けたらバーにタグを付ける
- 「Bar」が持つ「HitItem」のタグを「Player」に変更



- 出来たらゲーム実行
- アイテムにバーが衝突した時にアイテムが消えるか確認



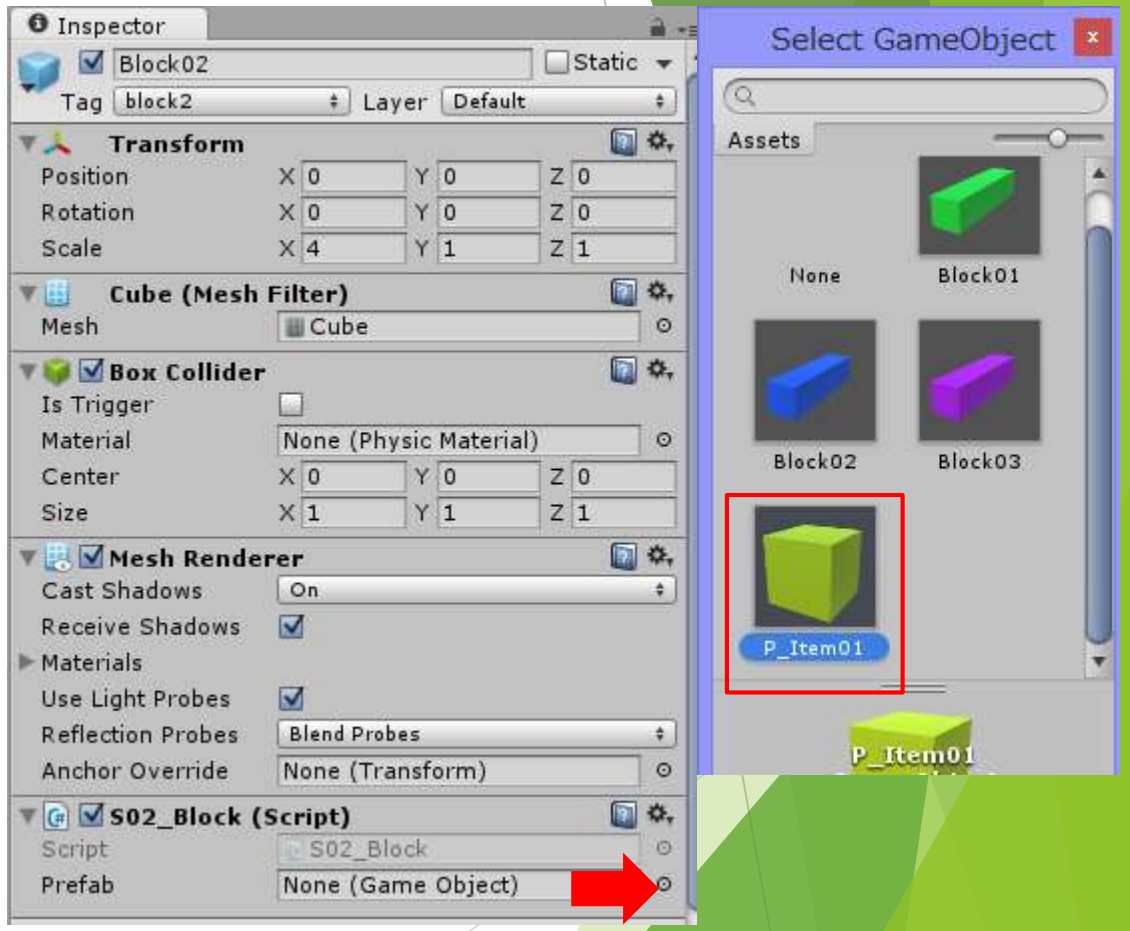
- アイテムはプレハブ化して使う
- ここまで出来た人は「P_Item01」をAssetsにドラッグ＆ドロップしてプレハブ化
- Assetsへの追加が確認できたらHierarchyの「P_Item01」は削除



3-3. アイテムの生成・効果

- アイテムの生成は「Block02」破壊時に一定確率で
  効果は「一定時間バーを長くする」

- 「Block02」で「P_Item01」を使えるようにする
- Assets内の「Block02」のInspectorを見ると
  「S02_Block」に黒い円形のボタンがあるのでクリック
- 使用するオブジェクトの選択画面が出る
- 「P_Item01」を選択



- アイテムを生成するプログラムを書く
- 「S02_Block」を開く
- 赤枠変更

- 破壊時に0~100の整数を生成
- その数が50未満であればアイテム生成

- これでゲーム実行すると
  真ん中のブロック破壊時にランダムで
  アイテムが生成される

```
void OnCollisionEnter(Collision other)
{
    hp--;
    if (hp == 0)
    {
        switch (this.gameObject.tag)
        {
            case "block1":
                s02_score.Additional_score(1);
                Debug.Log("ブロック1です");
                break;
            case "block2":
                s02_score.Additional_score(2);
                Debug.Log("ブロック2です");
                int rnd = Random.Range(0, 100);
                if (rnd < 50)
                {
                    Instantiate(prefab, transform.position, Quaternion.identity);
                }
                break;
            case "block3":
                s02_score.Additional_score(3);
                Instantiate(prefab, transform.position, Quaternion.identity);
                Debug.Log("ブロック3です");
                break;
        }
        GameObject.Destroy(this.gameObject);
    }
```

- 次に効果を付ける
- バーとアイテム両方のプログラムをいじる

- まず「S02_Bar」を開く
- setSpeed()の下あたりに赤枠追加

- 出来たら保存
- 次は「S02_Item」を開く
- OnTriggerEnter()に1行追加

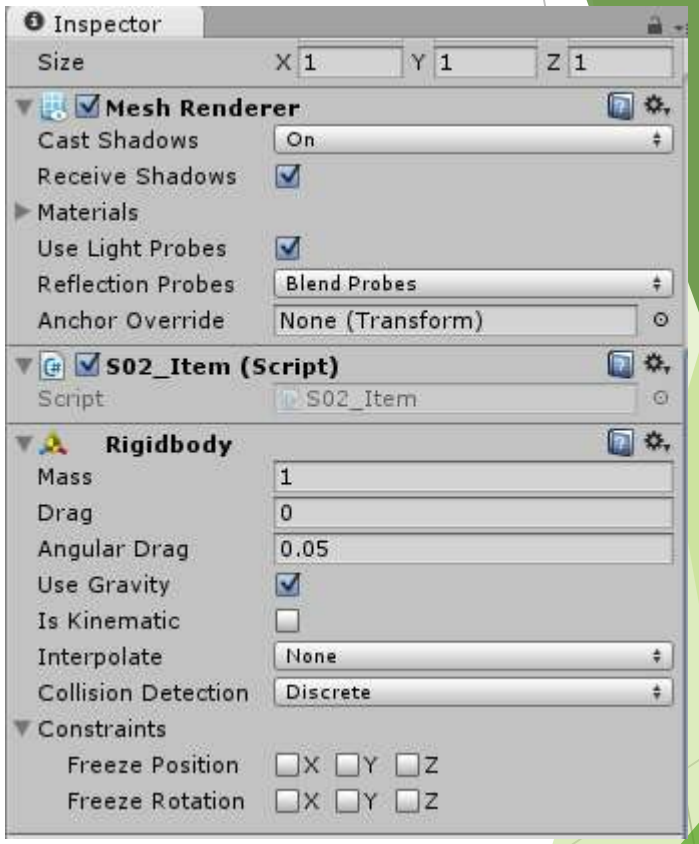
```
private void setSpeed()
{
    if (!key.moveUP)
    {
        speed = 10;
    }
    else
    {
        speed = 20;
    }
}
```

```
private void getItem01()
{
    StartCoroutine("item01");
}

IEnumerator item01()
{
    transform.localScale += Vector3.up;
    yield return new WaitForSeconds(10.0f);
    transform.localScale -= Vector3.up;
}
```

```
void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "Player")
    {
        other.transform.root.SendMessage("getItem01");
        Destroy(gameObject);
    }
}
```

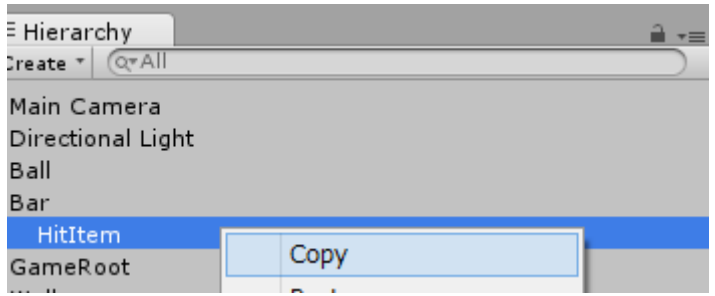
- ここまで出来たらAssetsの「P_Item01」にRigidbodyを追加
- 追加だけで特にいじる必要なし（重力を使いたいだけ）

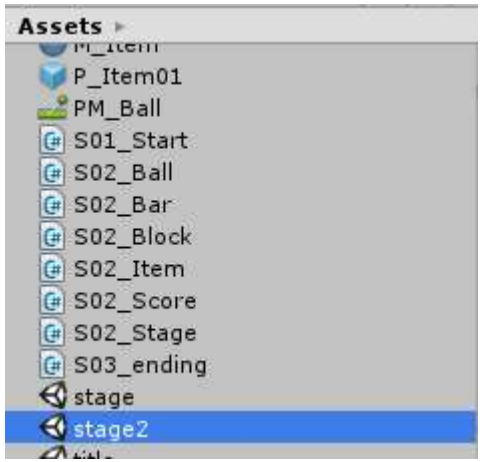
- これでゲーム実行
- 真ん中のブロック破壊→ランダムでアイテム生成
  →バーに接触でバー長くなる→10秒で元に戻る



- 今までのアイテムの設定追加を「stage」で行っていたら「stage2」を「stage2」で行っていたら「stage」を変更する

- Hierarchyの「HitItem」をコピー
- 使ってなかった方のシーンを開く





- 開いたシーンのHierarchy内「Bar」にコピーした「HitItem」を追加
- これで2つのステージでアイテムが使用できる

4. ゲームオーバー処理

4. ゲームオーバー処理

- 今まではデバッグをしやすいように下にも壁を作っていました
- 実際のブロック崩しでは下の壁はなく、下まで行ったらゲームオーバー
- そのゲームオーバー処理を追加する
- 「S02_Stage」を開く
- 赤枠追加
- 次ページに続く

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class S02_Stage : MonoBehaviour {
    public GameObject[] prefab;
    private bool gameClear = false;
    public GUIStyle gui_gameClear;
    static private int stage_no = 2;
    private S02_Score s02_score;
    public bool gameOver = false;

    void Start()
    {
        s02_score = GetComponent<S02_Score>();
        switch (stage_no)
        {
            case 1:
                blockSetting1();
                break;
            case 2:
                blockSetting2();
                break;
        }
    }
```

- Update()の中→

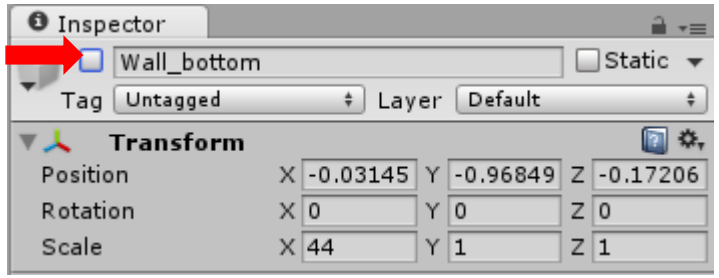
```
void Update()
{
    if (gameClear)
    {
        if (Input.GetMouseButtonDown(0))
        {
            stage_no++;
            if (stage_no < 3)
            {
                SceneManager.LoadScene(stage_no);
            }
            else
            {
                SceneManager.LoadScene("ending");
            }
        }
    }
    if (gameOver)
    {
        if (Input.GetMouseButtonDown(0))
        {
            stage_no = 1;
            s02_score.Reset_score();
            SceneManager.LoadScene(0);
        }
    }
}
```

- ゲームオーバー時

  画面に表示するようGUIを設定↓

```
void OnGUI()
{
    if (gameClear)
    {
        GUI.Label(new Rect(0, 0, Screen.width, Screen.height), "GameClear", gui_gameClear);
    }
    if (gameOver)
    {
        GUI.Label(new Rect(0, 0, Screen.width, Screen.height), "GameOver", gui_gameClear);
    }
}
```

- 次に下の壁を取り除く
- 削除でもいいが、デバッグでまた使うかもしれないので今回は削除しない
- Hierarchyの「Wall」内「Wall_bottom」のInspectorを見る
- 名前の左のレ点チェックを外す
- これで画面内から下の壁が消える
- また使うときはレ点にチェックを入れる
- （stage, stage2両方で行う）



- あとはゲームオーバーになったことを判定する部分を作るだけ
- 「S02_Ball」を開く

- Start()内とその下に関数を追加

```
void Start () {
    audioSource = gameObject.AddComponent<AudioSource>();
    audioSource.clip = sound;
    audioSource.loop = false;
    s02_Stage = GameObject.Find("GameRoot").GetComponent<S02_Stage>();

    transform.GetComponent<Rigidbody>().velocity = new Vector3(10.0f, 10.0f, 0.0f);
    StartCoroutine("checkPos");
}

IEnumerator checkPos()
{
    while (true)
    {
        yield return new WaitForSeconds(1.0f);
        if (transform.position.y < -1.0f)
        {
            s02_Stage.gameOver = true;
            transform.GetComponent<Rigidbody>().velocity = Vector3.zero;
            break;
        }
    }
}
```

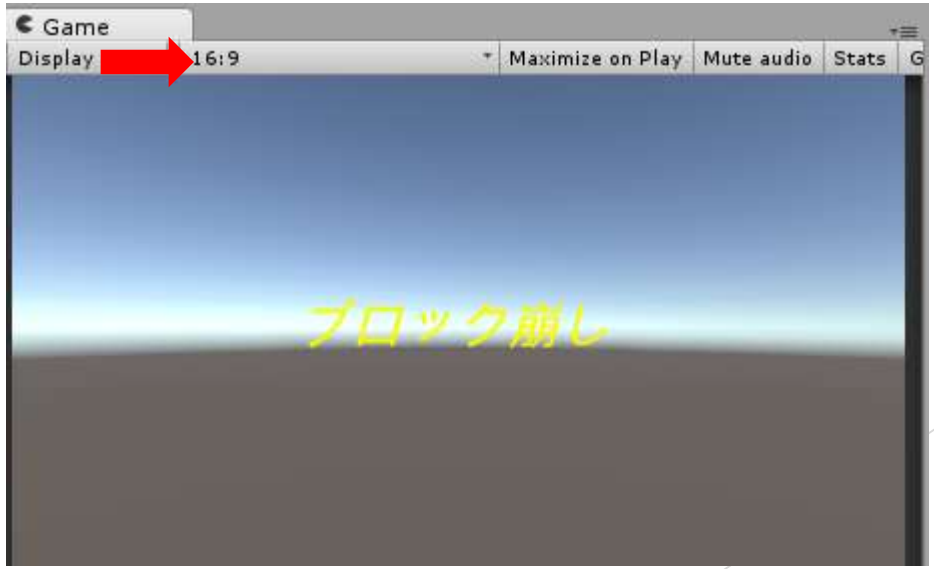
- 書けたらゲーム実行

5. 実行ファイル作成

5. ゲームの実行ファイル作成

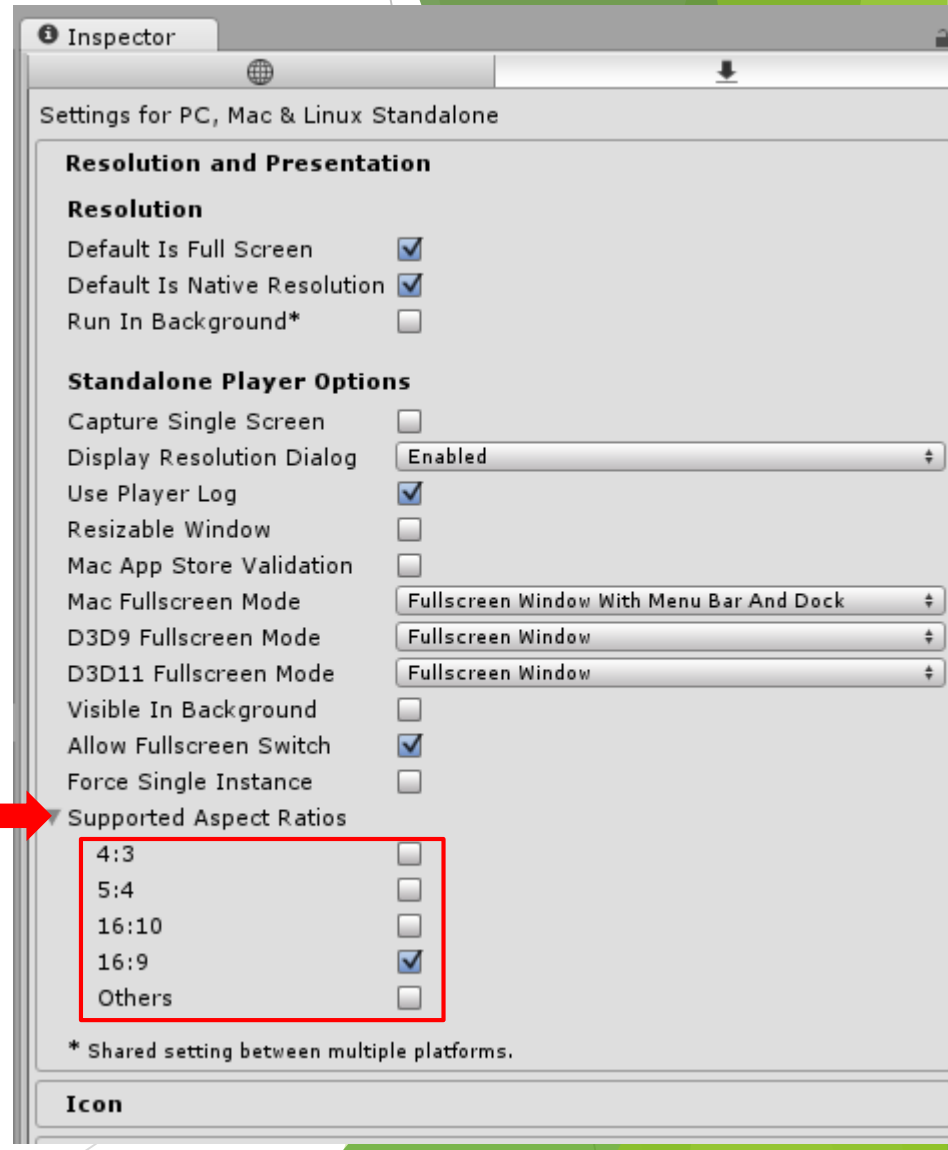
- 以上で最低限ゲームっぽいものに仕上がりました
- 今までUnity上のゲーム画面で操作をしてきた
- これをexeファイルとして書き出して扱えるようにする

- まずアスペクト比を設定
- Gameビューの比を16:9に



- 「File」→「Build & Setting」

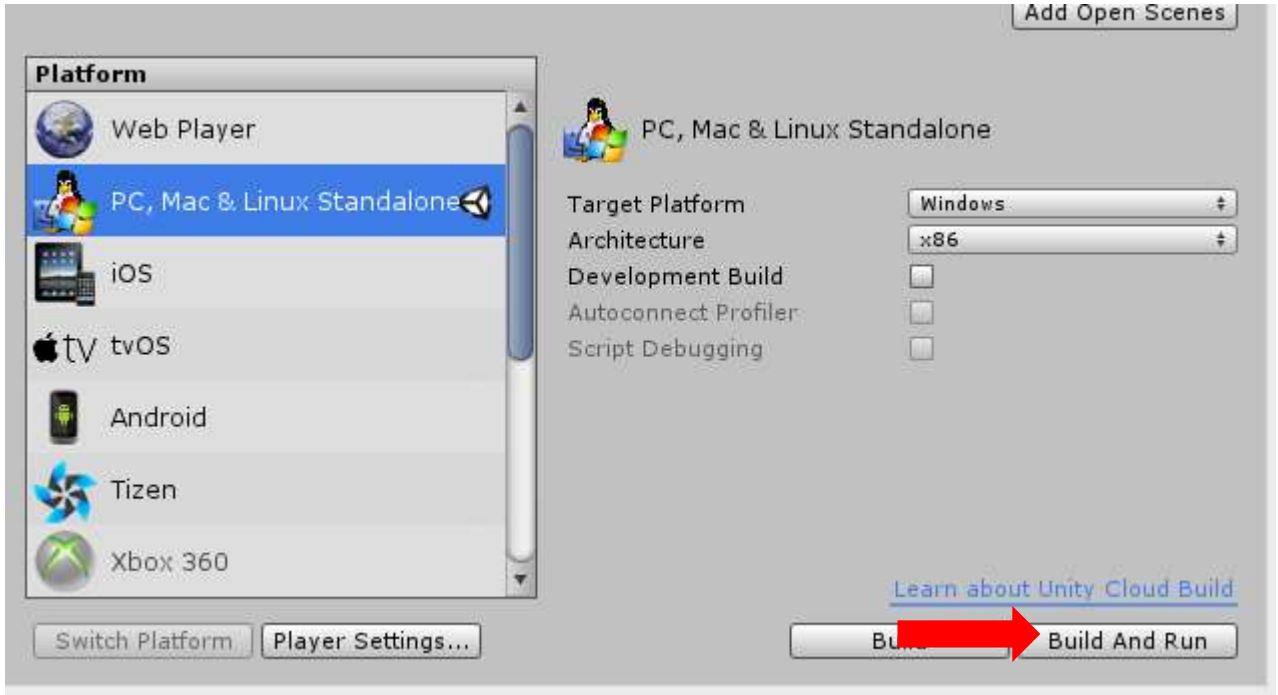
- 「Player Settings」をクリック
- Inspectorを見る
- 「Supported Aspect Ratios」の「16:9」以外にチェックを外す



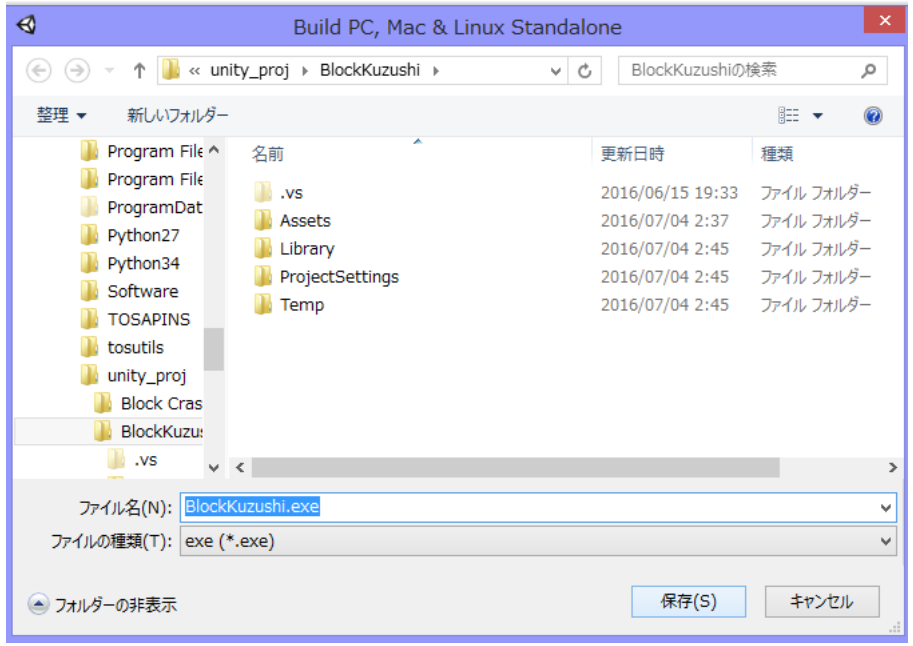
- このままビルドしてもいいがこのゲーム、エスケープキーを押しても終了しない
- ゲーム終了が厄介なのでエスケープキーで終了できるようにする
- 「S01_Start」を開く
- 赤枠追加
- これでタイトル画面時にエスケープキーで終了できる

```
public class S01_Start : MonoBehaviour {

    void Update()
    {
        if (Input.GetKey(KeyCode.Escape))
        {
            Application.Quit();
        }
    }

    void OnGUI()
    {
        if (GUI.Button(new Rect(Screen.width - 120, Screen.height - 40, 100, 20), "スタートボタン"))
        {
            SceneManager.LoadScene("stage");
        }
    }
}
```

- 「Build & Setting」右下の「Build And Run」をクリック

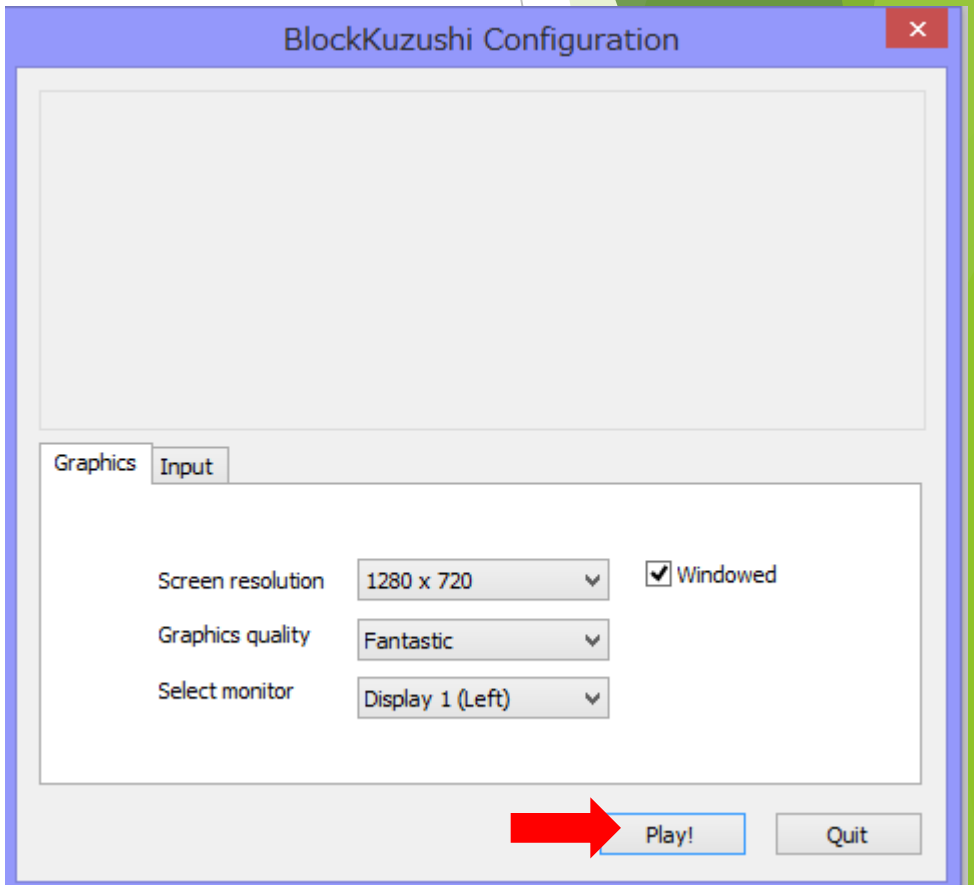


- 保存場所とファイル名を指定（ご自由に）

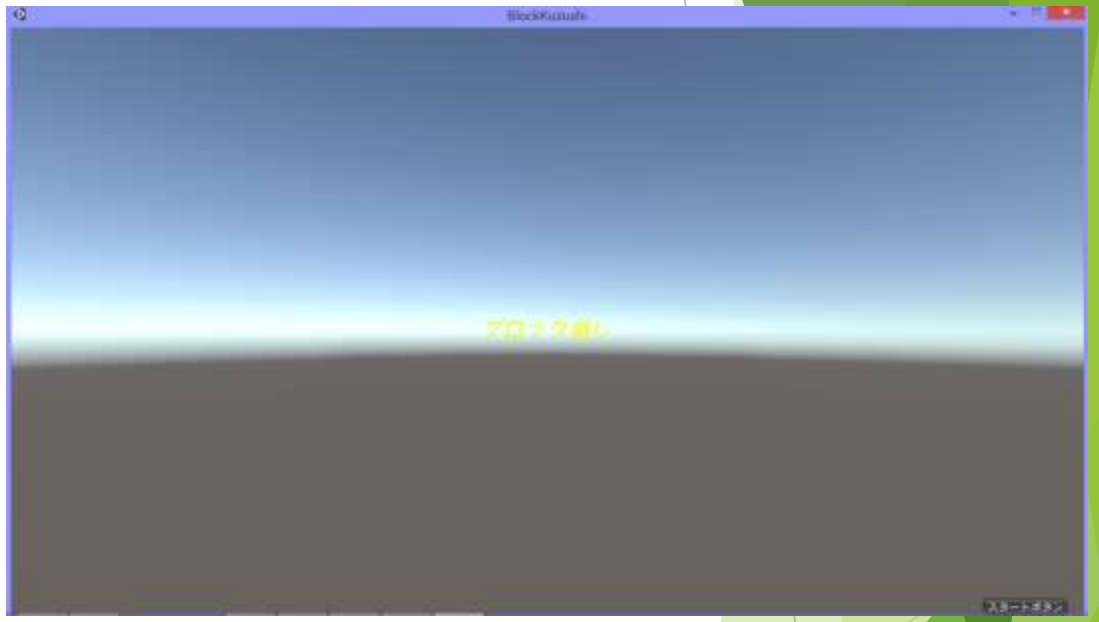


- 作成が終わると→の画面が出るので

  「Windowed」にチェック後、Playをクリック



- Unityのロゴが表示されタイトルに進むはず



- あとは普通に遊ぶだけ

6. 最後に

6. ゲーム完成

- 以上でブロック崩しの作成終わり
- exeで書き出したので自分のPCでなくても遊んでもらえます
- その場合、Buildで新しいフォルダを用意してそのフォルダごとコピーしましょう

- 細かいところで修正すべきところがいくつかあるので探して直してみてもいいかも
- その他にもボールをどんどん早くするとか新たなアイテム追加とか
- やれることはたくさんあります

- 今回のUnity講座はこれで終了！

みなさん、お疲れ様でした！！