

演習 2 : 集合の知性を設計する

(05) 05/20 (06) 05/27

A | Unity環境の整備・簡単なルール設計

(07) 06/10 (08) 06/17

B | ボイドルール 1・2・3 の実装

(09) 06/24 (10) 07/01

C 1 | 集合知の解析

(12) 07/08 (13) 07/15

C 2 | マイルール (ルール 4) の実装・視点の操作

(14-15) 07/22

C 2 | 発表 (One-Minute Movie)

演習 2 - C2

マイルール（ルール4）の実装・視点の操作

+ 課題内容

+ カメラの視点制御の例（サンプルの解説）

+ 特定のボイドのマテリアルを変える方法

+ クラスタに関する情報にアクセスするには

課題内容

課題

新しいルール4を設計し、そのインタラクションを効果的に体感するためのカメラ視点を獲得せよ。

ルール1から3を設計した際と同様に、**BoidManager.cs** に追記するかたちで、ルール4を作成します。

新しいルールはいくつ作ってもらっても構いません（しかし映像は指定時間におさめてください）。

提出物

課題を満足する90秒以内（最低60秒）の映像ファイル

QuickTimeの画面収録機能を使って、Game Viewをキャプチャし、適当な部分をトリミングしてください（編集もOK）。

ストーリー性のあるものには高い評価を与えます。

新しいゲームオブジェクトを参加させるのもOK。いろいろOKです。

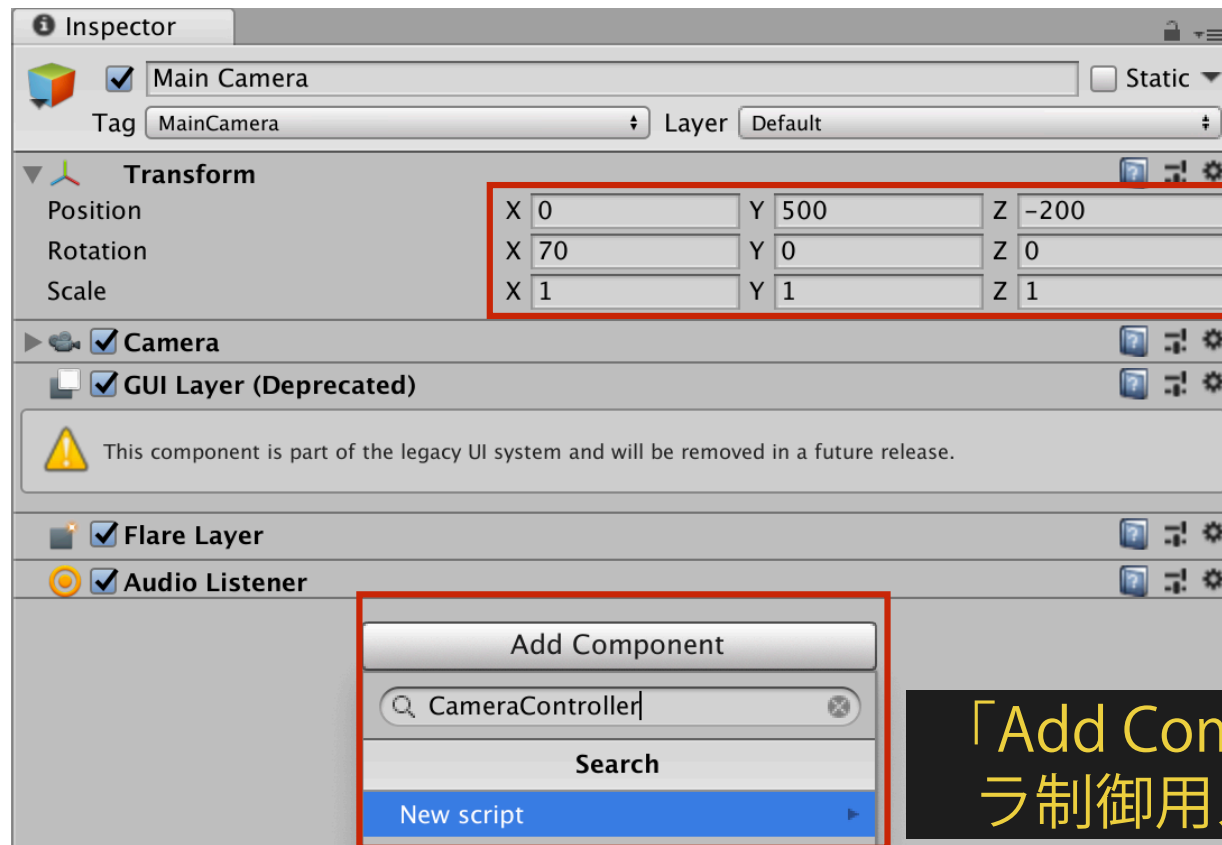
発表日

7月22日の1限・2限（一人3分ほど）

原則、最初の1分でスライド（時間節約のためPDF）を使ってコンセプトを発表し、残りの90秒で映像を流してもらいます。

あらかじめ、所定のフォルダに映像ファイルを提出してください。

カメラ制御用スクリプトの追加



カメラの位置
と方向

「Add Component」からカメラ制御用スクリプトを追加。

カメラの位置と方向は, Main Camera オブジェクトの Position と Rotation によって変更します.

カメラの視点を「動的」に変更するためには, Main Camera オブジェクトにカメラ制御用のスクリプトをインクルードします.

カメラ視点制御用のスクリプトのイグザンプルとして、まずは「CameraControl.cs」という名前のスクリプトを加えます.

カメラの視点制御の例 (サンプルの解説) 2 / 3

//カメラの視点を操作するスクリプト

```
public class CameraControl : MonoBehaviour {
```

```
//ボイドマネージャ (ゲームオブジェクト)
```

```
public GameObject boidmanager_obj;
```

```
BoidManager b; //ボイドマネージャ (オブジェクト)
```

```
void Start () {
```

```
//空のゲームオブジェクトBoidMangerから
```

```
//コンポーネントBoidManagerを取り出す。
```

```
b = boidmanager_obj.GetComponent<BoidManager>();
```

```
}
```

```
void Update () {
```

```
//ボイドの配列を取り出す。
```

```
SingleBoid[] boid = b.boid;
```

```
//ボイド人口が2以上の場合、以下を実行。
```

```
if (b.bsum > 2) {
```

```
//0番目のボイドの位置にカメラを置く
```

```
transform.position = boid [0].pos;
```

```
//1番目のボイドを常に見る。
```

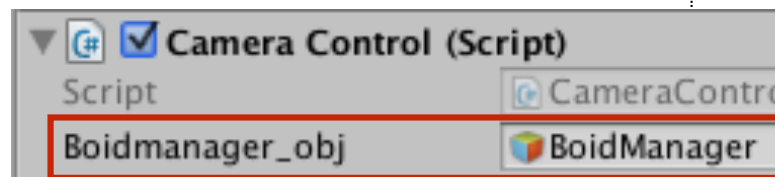
```
transform.rotation = Quaternion.LookRotation (boid [1].pos - boid [0].pos);
```

```
}
```

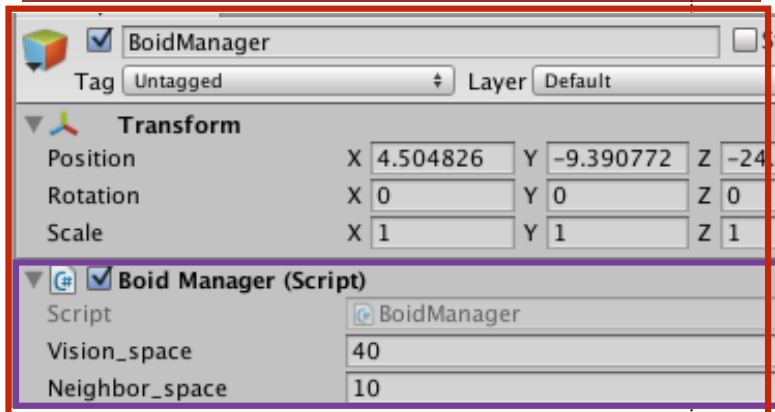
```
}
```

```
}
```

CameraControl.cs



ゲームオブジェクト
(Hierarchyからドラッグ&ドロップしてください)



スクリプトコンポーネント

個々のボイドの位置情報は、BoidManager (ゲームオブジェクト) の中に含まれる BoidManger.csによって記述されたコンポーネントの中に入っていますので、初期の処理として、まず、このスクリプトコンポーネントにアクセスしています。

カメラの視点制御の例 (サンプルの解説) 3 / 3

CameraControl.cs

```
//ボイドマネージャ (ゲームオブジェクト)  
public GameObject boidmanager_obj;  
BoidManager b; //ボイドマネージャ (オブジェクト)  
  
void Start () {  
    //空のゲームオブジェクトBoidMangerから  
    //コンポーネントBoidManagerを取り出す。  
    b = boidmanager_obj.GetComponent<BoidManager>();  
}  
  
void Update () {
```

ボイドの配列を
取り出すまでの
定型処理の領域

1

```
//ボイドの配列を取り出す。  
SingleBoid[] boid = b.boid;
```

毎回, 最新の空間情報に更
新されたボイドの配列にア
クセスします。

カメラの視点を制
御する処理領域

2

```
//ボイド人口が2以上の場合  
if (b.bsum > 2) {  
    //0番目のボイドの位置にカメラを置く  
    transform.position = boid [0].pos;  
    //1番目のボイドを常に見る。  
    transform.rotation = Quaternion.LookRotation (boid [1].pos - boid [0].pos);  
}  
}
```

b.bsum で ボイドの総数にアクセスできます。

カメラの位置を添え字<0>の
ボイドの現在の位置にセット

カメラの方向を, 常に添え字<1>のボイドを向くようにセットします。

boid[i].pos, boid[i].vel

によって, i番目のボイドの位置と速度を取得
できます. これを使って, カメラの transform
を制御します.

SingleBoid

<Vector3> pos, vel

ボイドの位置と速度

特定のボイドのマテリアルを変える方法 (1/3)

準備として,

BoidManager.cs

に以下を追記します

```
//ボイドの配列 (インスペクタには非表示)
[HideInInspector]
public SingleBoid[] boid;
[HideInInspector]
public GameObject[] boidobj;
```

宣言部

BoidManager.cs

```
void Start () {
```

Start()

```
/* 解析オブジェクトの生成 */
```

```
ana = this.GetComponent<BoidClusterAnalysis> ();
```

```
/* ボイドオブジェクト (SingleBoidクラス | スクリプト) */
```

```
boid = new SingleBoid[bsum];
```

```
boidobj = new GameObject[bsum];
```

```
for (int i = 0; i < bsum; i++) {
```

```
    GameObject bobj = Instantiate ((GameObject)Resources.Load ("Boid"));
```

```
    boid [i] = bobj.GetComponent<SingleBoid> ();
```

```
    boidobj[i] = bobj;
```

```
}
```

プレハブの boid に対応するゲームオブジェクトの配列をパブリックなフィールドにします。

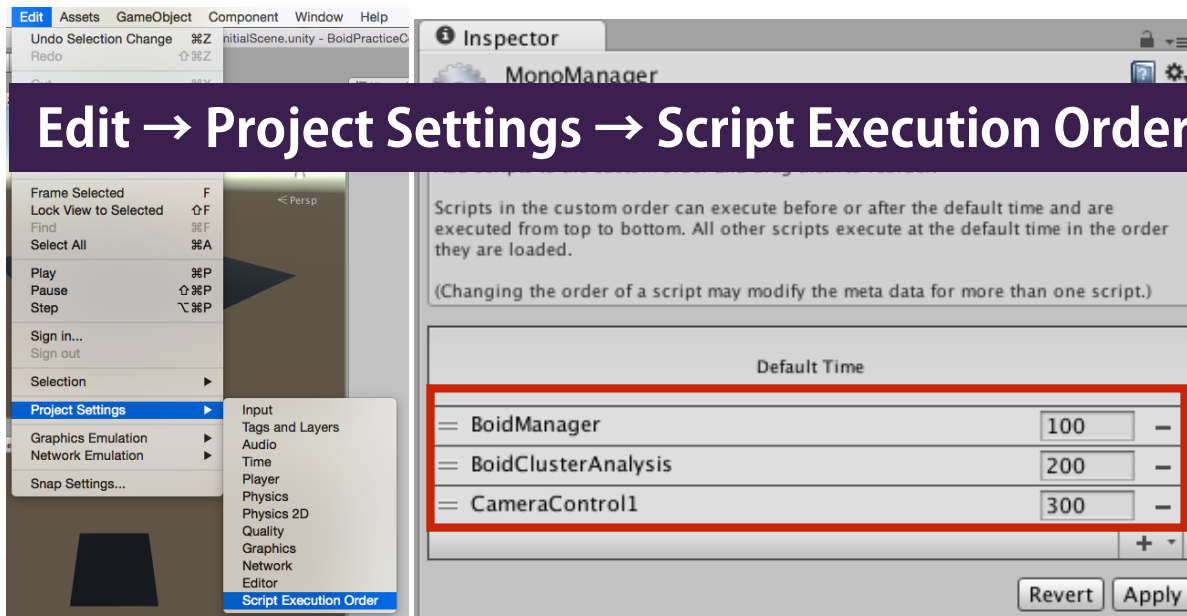
特定のボイドのマテリアルを変える方法 (2/3)

カメラコントロール用サンプルに以下を追記して、ボイドの色を変えています。

CameraControl1.cs

```
void Start () {  
    //空のゲームオブジェクトBoidMangerから  
    //コンポーネントBoidManagerを取り出す。  
    b = boidmanager_obj.GetComponent<BoidManager>();  
  
    b.roidobj [0].GetComponent<Renderer> ().material.color = new Color (1f, 1f, 1f);  
    b.roidobj [1].GetComponent<Renderer> ().material.color = new Color (1f, 1f, 0f);  
}
```

添字<0>のボイドを白に、添字<1>のボイドを黄色にします。



bのフィールドを正確に参照するためには、CameraControl関連のスク립トの実行順序をBoidManagerよりも後で実行されるように調整する必要があります。

特定のボイドのマテリアルを変える方法 (3 / 3)

ボイド<1>



クラスタに関する情報にアクセスするには (1/3)

準備として, **BoidManager.cs** に以下の修正をします.

```
public class BoidManager : MonoBehaviour {  
  
    //解析オブジェクト  
    public BoidClusterAnalysis ana;  
  
    //飛翔スペース  
    static public int flyspace = 400; //一辺の長さ  
    static public int flyheight = 200; //高さ (3Dモード)  
    //速さの最大値  
    static public float speedmax = 100f;  
}
```

BoidClusterAnalysis 型のフィールド ana のアクセス修飾子を public として, 他のクラスからアクセス可能な状態とします.

宣言部

BoidManager.cs

BoidClusterAnalysisクラスの ana フィールドは, a の key を押すことでクラスタ計算を行い, クラスタに関する様々な情報を保持します.

実際, BoidMangerクラスでは, ana を使って, クラスタを可視化しています.

BoidClusterAnalysis

| | |
|---|---------------------------------|
| <code><int> csum</code> | <code><int[]> cpop</code> |
| クラスタの総数 | 各クラスタの個体数 |
| <code><Vector3[]> cog</code> | |
| 各クラスタの重心 | |
| <code><Vector3[]> rad</code> | |
| 各クラスタの半径 (重心から最遠方ボイドまでの距離) | |
| <code><int[]> cluster_id</code> | |
| 各ボイドが所属するクラスタの id (0, 1, 2, ...) . どのクラスタにも属さない場合 -1. | |

クラスタに関する情報にアクセスするには (2 / 3)

```
void Update () {  
  
    int csum = b.ana.csum; //クラスタの数  
  
    for (var i = 0; i <b.bsum ; i++) {  
  
        Color col; //ポイドに設定する色  
        int cid = b.ana.cluster_id [i]; //iポイドが属しているクラスタのid  
  
        switch (cid) {  
        case -1:    col = new Color (1f, 1f, 1f);    break;  
        case 0:    col = new Color (1f, 1f, 0f);    break;  
        case 1:    col = new Color (0f, 0f, 1f);    break;  
        default:   col = new Color (1f, 0.5f, 0f);  break;  
        }  
  
        b.boidobj [i].GetComponent<Renderer>().material.color = col; //色の設定  
  
        //クラスタ0の重心からクラスタ1の重心の方向を見る (csum>=2の場合)  
        if (csum >= 2) {  
            transform.position = b.ana.cog [0];  
            transform.rotation = Quaternion.LookRotation (b.ana.cog [1] - b.ana.cog [0]);  
        } else if (csum == 1) {  
            transform.position = new Vector3 (200f, 200f, 200f);  
            transform.rotation = Quaternion.LookRotation (b.ana.cog [0] - transform.position);  
        } else{  
            transform.position = new Vector3 (200f, 200f, 200f);  
            transform.rotation = Quaternion.LookRotation (b.boid [0].pos - transform.position);  
        }  
    }  
}
```

Update()

所属しているクラスタのidに従って色を振り分けています。

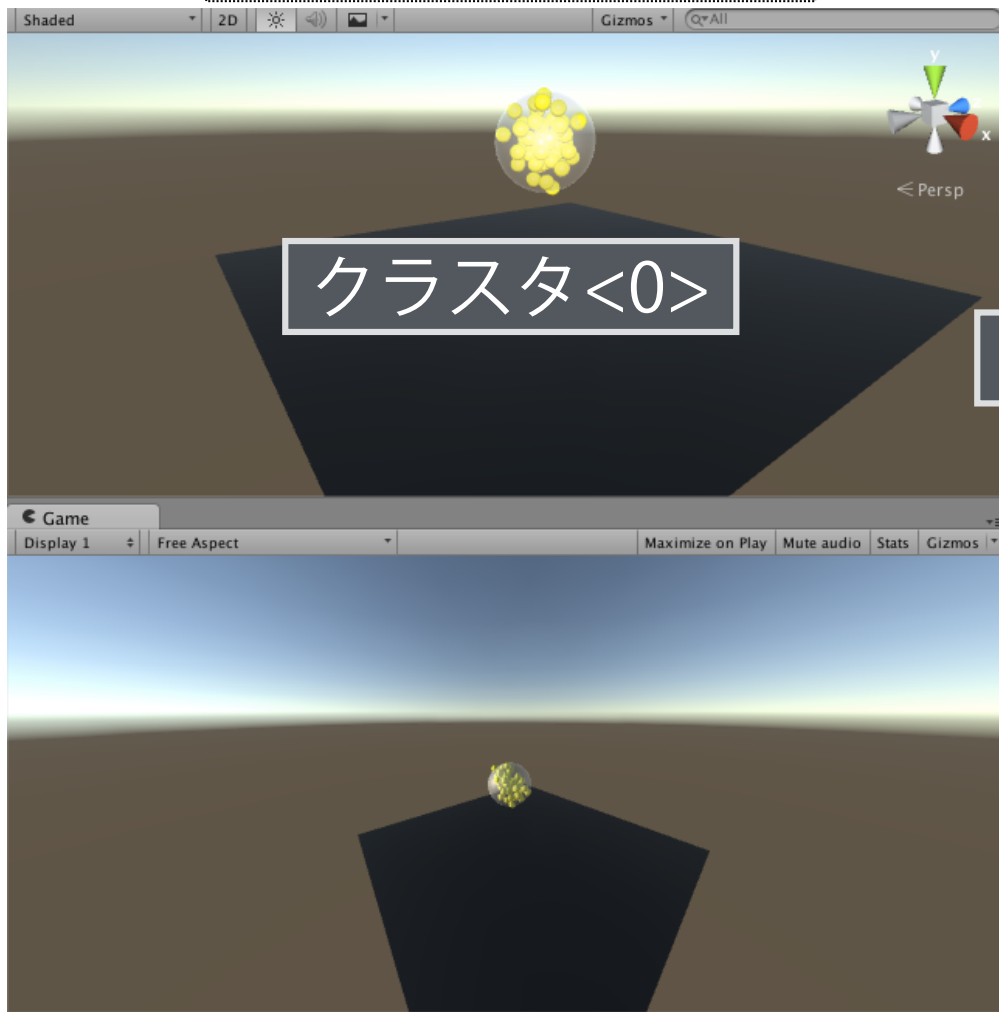
クラスタの重心位置を使って、カメラの視点・方向を決めています。

その他の部分は、サンプル (CameraControl.cs) と同様です。

A 解析モードの ON / OFF の切り替え

正確にコードを実行するには、解析モードをONとすることがあります。

クラスタが1つの場合



クラスタが2つの場合

