演習２：集合の知性を設計する

(05) 05/19　(06) 05/26

A｜Unity環境の整備・簡単なルールの設計

(07) 06/02　(08) 06/09

B｜ボイドルール１・２・３の実装

(09) 06/16

C｜課題１：集合知の解析

(10) 06/23

D１｜SIR（感染モデル）

(11) 06/30　(12) 07/07　(13) 07/14

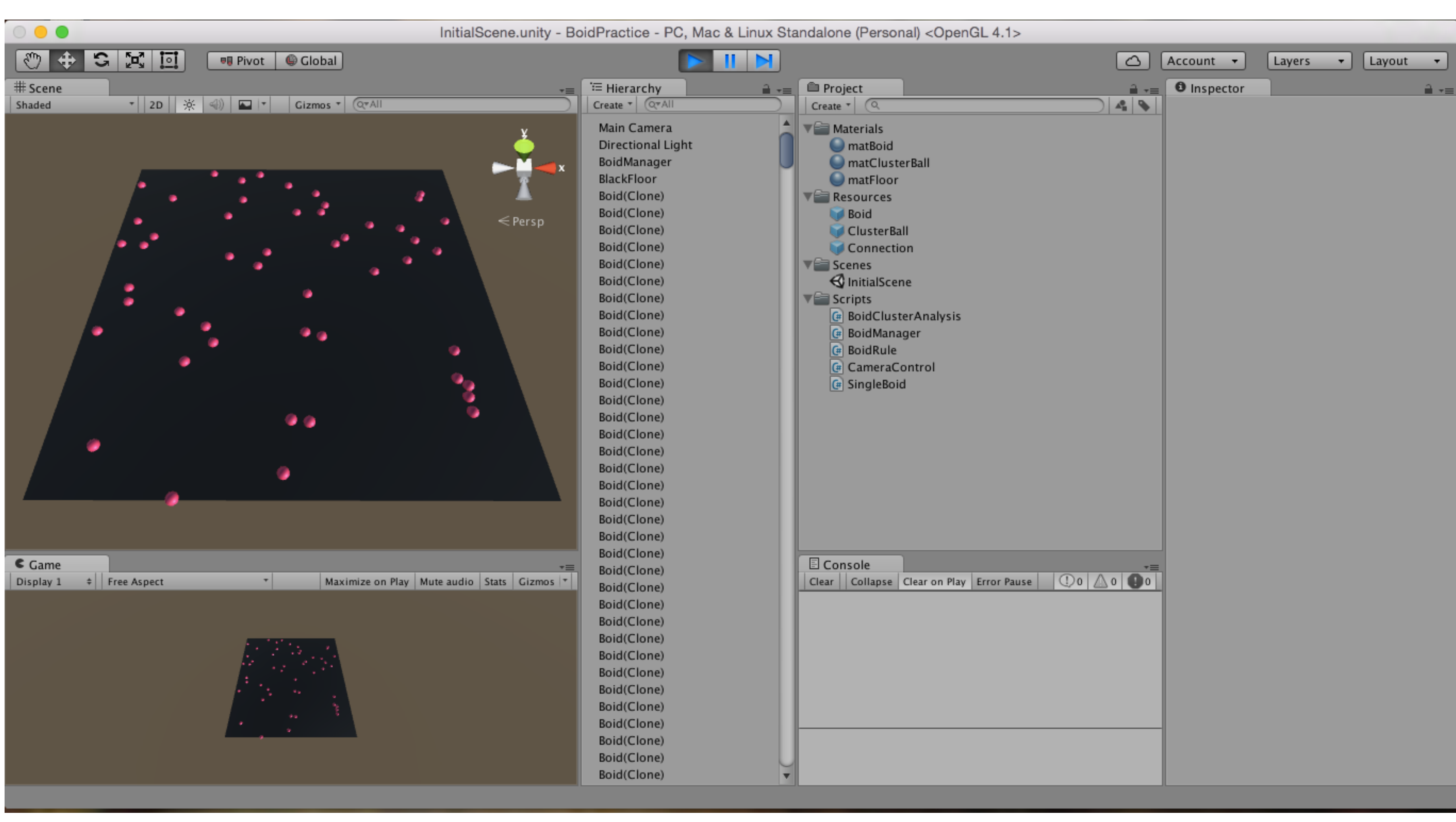
D２｜課題２：マイルール・感染ルール・視点操作

(14-15) 07/21

D３｜発表（One-Minute Movie）

演習２-Ａ

Unity環境の整備・簡単なルールの実装



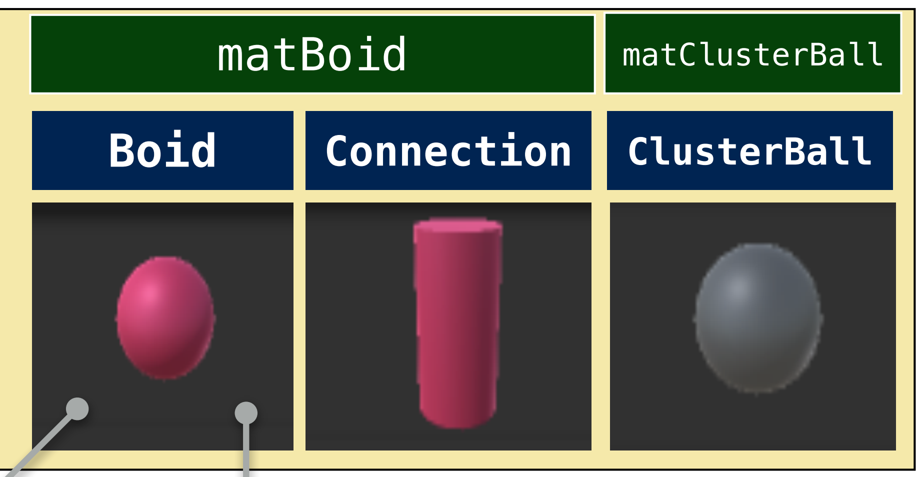
プロジェクト全体の構造

Directional Light

BlackFloor　matFloor

BoidManager

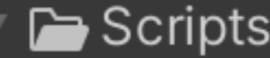
- BoidManager
  描画の実行クラス
- BoidRuleManager
  ボイドのルールの記述用のクラス
- BoidClusterAnalysis
  ボイドのクラスター解析用のクラス



マテリアル

Materials
matBoid
matClusterBall
matFloor

ボイド単体の振る舞いを記述したクラス



- BoidClusterAnalysis
- BoidManager
- BoidRuleManager
- SingleBoid

スクリプト

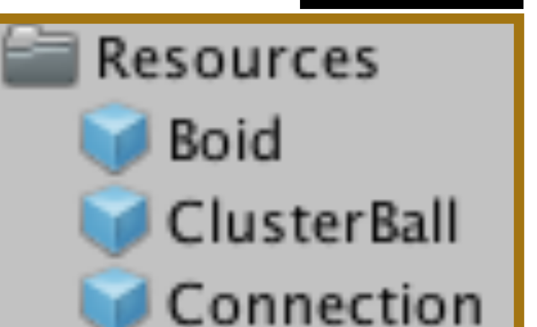
ParentBoid

ParentConnection

ParentClusterBall

プレハブを収納するためのゲームオブジェクト

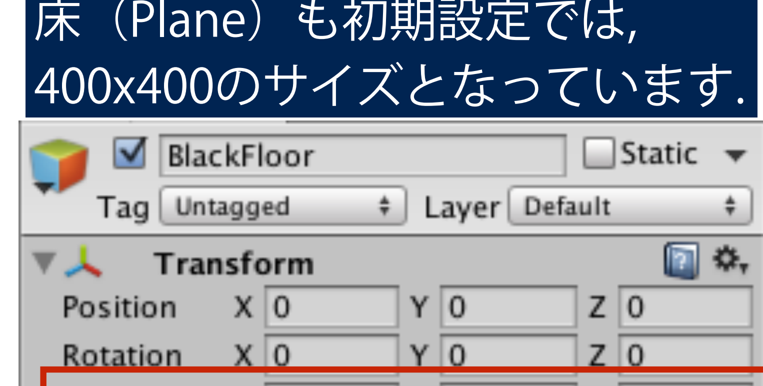
プレハブ



二次元モード・三次元モード

- デフォルトでは, 400 x 400（xz）の空間をボイドが動き回りますが,「3」ボタンを押すと, 三次元モードとなり, 400 x 400 x 200（xzy）の空間を使うことができるようになります.「2」を押すと、元の二次元モードに戻ります。
- 「I」ボタンを押すと, すべてのボイドの位置と速度が初期化されます.

床（Plane）も初期設定では, 400x400のサイズとなっています.



BlackFloor
2次元モード

3次元モード
BlackFloor

キー	機能
2	二次元モード
3	三次元モード
i	ボイドの位置・速度の初期化

BoidManager・BoidRuleManagerクラスのpublic変数

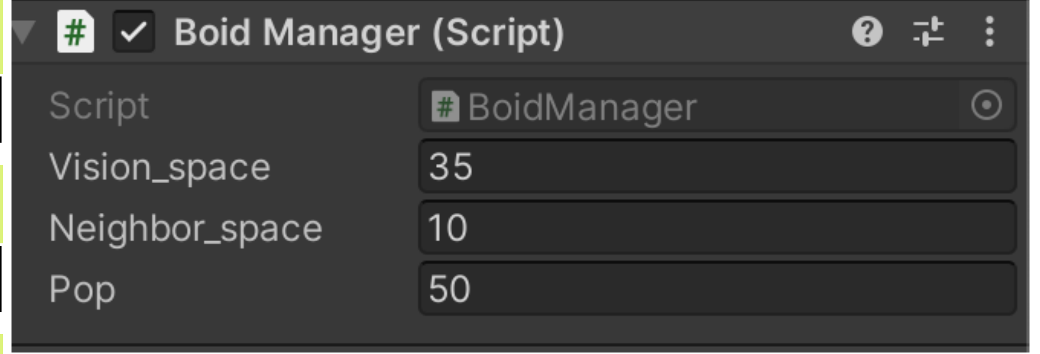
- BoidManager・BoidRuleManagerのいくつかのフィールドについては，インスペクタビューから設定可能な状態となっています．初期状態では，すべてのルールは未設計のため，各ボイドは相互作用をせずに，初期速度を維持したまま空間内を（ビリヤードのように）ただただ動き回ります．

BoidManager

<int> vision_space
各々のボイドの視界距離

<int> neighbor_space
各々のボイドの接触限界距離

<int> pop
ボイドの総数（開始後は変更不可）



BoidRuleManager

<bool> rule1,rule2,rule3
ルール1・2・3の適用の有無

<float> c1,c2,c3
各ルールの影響度（係数）

Boid Rule Manager (Script)
Script BoidRuleManager
C1 0.1
C2 5
C3 0.01
Rule 1
Rule 2
Rule 3

BoidManagerにおけるSingleBoidオブジェクトの呼び出し

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

クラス変数

BoidManager.cs

Start()

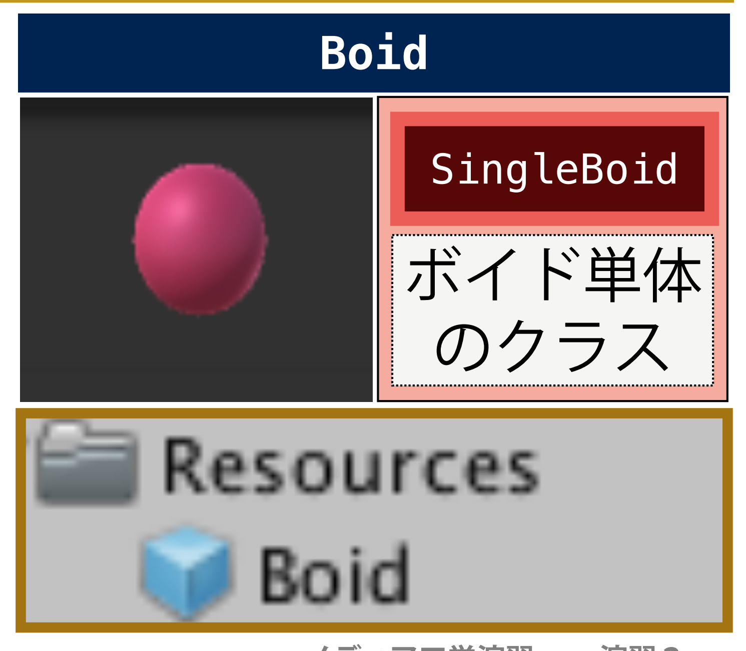
```
/* ボイドオブジェクト（SingleBoidクラス|スクリプト） */

boid = new SingleBoid[bsum];
GameObject bpar = GameObject.Find("ParentBoid"); //親のゲームオブジェクトを探索

for (int i = 0; i < bsum; i++) {
    //GameObject bobj = Instantiate ((GameObject)Resources.Load ("Boid"));
    GameObject bobj = Instantiate((GameObject)Resources.Load("Boid"),bpar.transform);

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

- ボイド単体の基本的な振る舞いは, Boidプレハブのコンポーネントである SingleBoid.cs の中で記述されています.
- BoidManagerは, **start**関数のなかで, まず Boidプレハブのクローンを作成 (**Instantiate**関数) したのちに, Boidのコンポーネントとして, SingleBoid オブジェクトを取り出し, 配列を構成します.



BoidManager・BoidRuleManger・SingleBoidクラスの関係

BoidManagerクラスは, ボイドの集団をフィールドとして管理します。

BoidRuleManagerは, 各種のボイド間の相互作用（ルール）の適用の有無（bool rule1, rule2, rule3）、そしてルールの具体的な内容（void ApplyRuleX）を管理します.

個々のボイドの位置・速度の更新は, SingleBoidクラスで管理されています.

SingleBoid.cs

BoidRule Manager

void ApplyRule1()

void ApplyRule2()

void ApplyRule3()

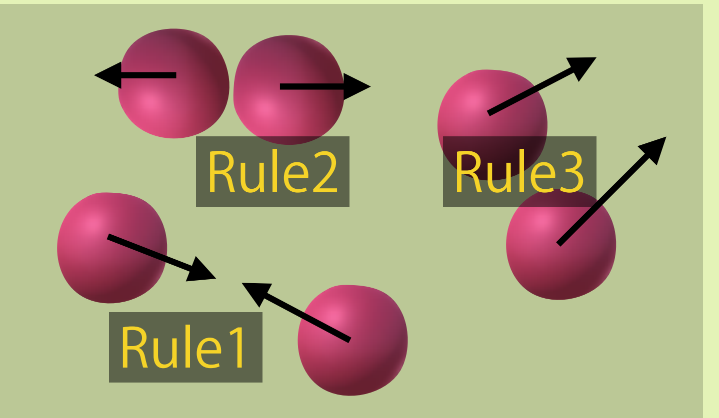
具体的なルールの記述（演習の対象）

ルールの適用の有無

<bool> rule1

<bool> rule2

<bool> rule3



SingleBoidオブジェクトの振る舞い

SingleBoid

SingleBoid.cs

宣言部

```
/* パブリックなフィールド（別クラスから参照可能） */
public Vector3 pos, vel; //位置・速度

/* プライベートなフィールド */
private Rigidbody rb;                           //剛体オブジェクト
private float xmax,xmin,ymax,ymin,zmax,zmin;  //空間の境界
private float speedmax;                         //速さの最大値
```

Start()

```
void Start ()
{
    speedmax = BoidManager.speedmax;
    rb = this.GetComponent<Rigidbody> ();

    this.setBorder ();              //飛翔空間の決定
    this.setRandomPosition ();      //初期位置の決定
    this.setRandomVelocity ();      //初期速度の決定

}
```

Update()

```
void Update ()
{
    //位置と速度の更新
    pos = this.transform.position;
    vel = this.rb.velocity;

    Rebound ();             //境界判定
    LimitVelocity ();       //速度制限
    ConstrainHeight ();     //高さの制限（2Dモード）
}
```

毎フレームの処理（設計者は意識する必要のない処理です．）

Rebound() ▶ LimitVelocity() ▶ ConstrainHeight()

- Rebound(): 境界を越えたら速度を反転
  - xzの初期値は±200, yは0から200
- LimitVelocity(): 速さが最大値を越えたら, 抑制する
  - speedmaxの初期値は100
- ConstrainHeight(): 2Dモードの場合, Y座標を強制的に0とする

SingleBoidオブジェクトの主なパブリック変数・関数

SingleBoid

<Vector3> pos, vel

ボイドの位置と速度

Update()

```
void Update ()
{
    //位置と速度の更新
    pos = this.transform.position;
    vel = this.rb.velocity;

    Rebound ();                //境界判定
    LimitVelocity ();          //速度制限
    ConstrainHeight ();        //高さの制
```

void SetVelocity(Vector3 *vel*)

ボイドの速度を具体的に設定する

void SetRandomPosition()

位置をランダムに設定

void SetRandomVelocity()

位置をランダムに設定

SingleBoid.cs

```
public void setVelocity(Vector3 v){
    this.rb.velocity = v;
    this.vel = this.rb.velocity;
}
```

```
public void setRandomPosition(){
    float rx = xmin + (xmax - xmin) * Random.value;
    float ry = ymin + (ymax - ymin) * Random.value;
    float rz = zmin + (zmax - zmin) * Random.value;

    this.transform.position = new Vector3 (rx, ry, rz);
    this.pos = this.transform.position;

}
```

```
public void setRandomVelocity(){

    float vx = -speedmax + 2f * speedmax * Random.value;
    float vy = -speedmax + 2f * speedmax * Random.value;
    float vz = -speedmax + 2f * speedmax * Random.value;

    rb.velocity = new Vector3 (vx, vy, vz);
    this.vel = this.rb.velocity;

}
```

SingleBoidクラスのオブジェクトの位置と速度は, *boid*.pos, *boid*.velによって取得できます. また, 新たに速度を設定し直す場合は, *boid*.SetVelocity（*vel*）の関数を使います（*boid* は, SingleBoidクラスのインスタンスとします）.

BoidRuleManagerにおけるパブリックメソッド

BoidRuleaManager.cs

void SetBoid(BoidManager m)

現在のボイドの状態（総数・位置・方向・視界距離・接触距離）を、自クラスの同名の変数にコピーする. ルール計算の前に実行する.

```
public void SetBoid(BoidManager m)
{
    boid = m.boid;
    pop = m.pop;

    if (is_vision_space_global)
    {
        for (int i = 0; i < pop; i++)
        {
            boid[i].SetVisionSpace(m.vision_space);
        }
    }

    if (is_neighbor_space_global)
    {
        for (int i = 0; i < pop; i++)
        {
            boid[i].SetNeighborSpace(m.neighbor_space);
        }
    }

}
```

void ApplyRules()

ルールを全て実行する. パブリックなbool変数（rule1・rule2・rule3）の符号によって, 特定のルールのみを実行することが可能. ルールの内容は, ApplyRuleX()の中に記述する.

```
public void ApplyRules()
{

    if (rule1)
    {
        ApplyRule1();
    }

    if (rule2)
    {
        ApplyRule2();
    }

    if (rule3)
    {
        ApplyRule3();
    }

}
```

void ApplyRule1()

void ApplyRule2()

void ApplyRule3()

```
/* ルール１の実行内容*/
1 reference
private void ApplyRule1()
{

}
/* ルール2の実行内容 */
1 reference
private void ApplyRule2()
{

}

/* ルール3の実行内容 */
1 reference
private void ApplyRule3()
{

}
```

BoidManagerとBoidRuleManagerの関係

BoidManager.cs

Start()

```
/* ボイドルールマネージャの生成 */
rule = this.GetComponent<BoidRuleManager> ();
```

Update()

```
//ボイドルールマネージャによるルールの適用
rule.SetBoid(this);
rule.ApplyRules();
```

自分自身のボイド変数を，BoidRuleMangerオブジェクトにコピーした後に，ルールを実行させる．

```
/* 全てのボイドの位置・速度を初期化（I） */
if (Input.GetKeyDown (KeyCode.I)) {
    InitBoidPosition ();
    InitBoidVelocity();
}
```

<I>キーが押されたときに，「InitBoidPosition」と「InitBoidVelocity」を実行する．

BoidRuleManager.cs

BoidRuleManager

void **SetBoid**(BoidManager *m*)

void **ApplyRules**()

InitBoidPosition()

```
/* 全てのボイドの位置をランダムに初期化*/
1 reference
private void InitBoidPosition()
{
    for (int i = 0; i < pop; i++)
    {
        boid[i].SetRandomPosition();
    }
}
```

InitBoidVelocity()

```
/* 全てのボイドの速度をランダムに初期化*/
1 reference
private void InitBoidVelocity()
{
    for (int i = 0; i < pop; i++)
    {
        boid[i].SetRandomVelocity();
    }
}
```

ボイドの位置・速度の初期化．SingleBoidクラスのメソッドを呼び出しています．）

例題

SingleBoid

<Vector3> **pos, vel**

ボイドの位置と速度

```
void setVelocity(Vector3 vel)
```

Voidの速度を更新する.

```
if (Input.GetKeyDown(KeyCode.R)) {
    ReverseVelocity();
}
```

Rキー が押されたら, ReverseVelocityを実行する.

Update()

BoidManager.cs

「R」ボタンで, 全てのボイドの速度が反転（Reverse）するように, BoidMangerクラスのクラスメソッド ReverseVelocityに記述しましょう.

```
void ReverseVelocity()
{
    for(int i = 0; i <  pop ; i++)
    {
        Vector3 ivel = boid[i].vel;
        Vector3 v = new Vector3(-ivel.x, -ivel.y, -ivel.z);

        boid[i].setVelocity(v);
    }
}
```

全てのボイド（**boid[0], boid[1], …boid[bsum-1]**）に対して

ボイド **i** の速度を取得し, **ivel** とし, 新しい速度ベクトル **v** を, **ivel** の全ての速度成分を反転させたものとして定義する.

ボイド **i** の速度を更新する.

BoidManager.cs