演習２：集合の知性を設計する

(05)　05/20

Ａ｜Unity環境の整備・簡単なルールの設計

(06) 05/27　(07)　06/03

Ｂ｜ボイドルール１・２・３の実装

(08-09)　06/10-17

Ｃ｜課題１：集合知の解析

(10)　06/24

Ｄ１｜ＳＩＲ（感染モデル）

(11)　07/01　(12)　07/08　(13)　07/15

Ｄ２｜課題２：マイルール・感染ルール・視点操作

(14-15)　07/22

Ｄ３｜発表（One-Minute Movie）

２ つ の （軽微な）修 正

(1)　RULE2の手続きを一部修正します。

(2)　BoidCluseterAnalysis.csの修正

（１）ＲＵＬＥ２の修正

BoidRuleManager.cs

```
/* ルール2の実行内容 */
private void ApplyRule2()
{

    for(int i = 0; i < pop; i++)
    {
        Vector3 ipos = boid[i].pos;
        Vector3 ivel = boid[i].vel;

        float ineighbor_space = boid[i].neighbor_space;

        Vector3 cvTotal = new Vector3(); //追加

        for(int j = 0; j < pop; j++)
        {

            Vector3 jpos = boid[j].pos;
            float dis = Vector3.Distance(ipos,jpos);
            if (i != j && dis < ineighbor_space && dis != 0)
            {
                cvTotal += c2 * (ipos - jpos)/dis;      //追加
                //Vector3 cv = c2 * (ipos - jpos)/dis; //削除
                //boid[i].SetVelocity(ivel + cv);      //削除
            }

            boid[i].SetVelocity(ivel + cvTotal); //追加
        }
    }
}
```

（２）BoidClusterAnalysis.csの修正

BoidClusterAnalysis.cs

start()

修正前

```
public void Start()
{

    //  あらかじめ、友達履歴記録用の配列を
    //  余裕を持って作成しておく.
    friends_history = new int[9999][];

    for (int i = 0; i < 9999; i++)
    {
        friends_history[i] = new int[9999];

        for (int j = 0; j < 9999; j++)
        {
            friends_history[i][j] = 0;
        }

    }
}
```

修正後

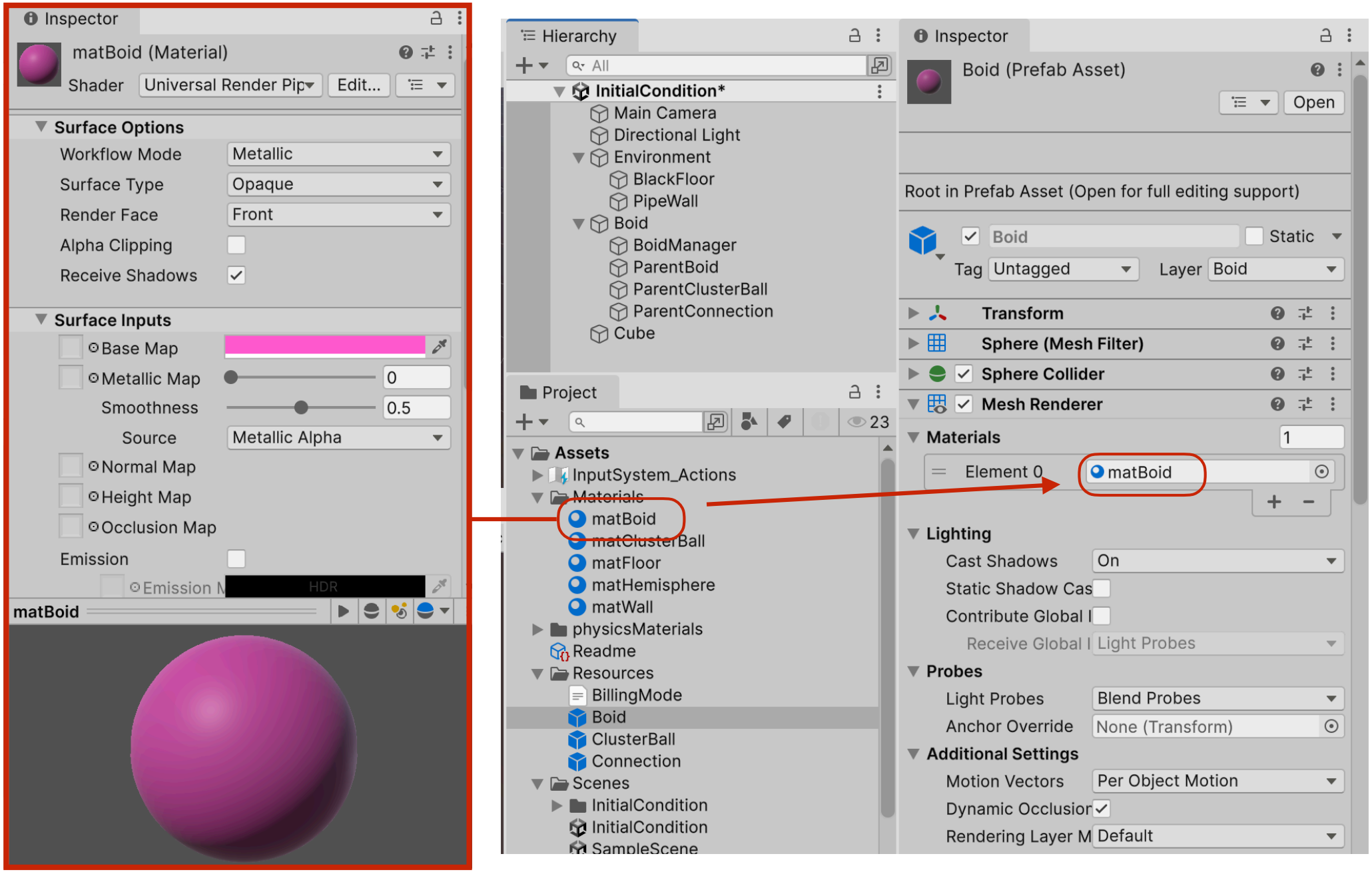
```
public void Start()
{
    // 同じGameObjectのBoidManagerから個体数を取得
    int pop = GetComponent<BoidManager>().pop;

    friends_history = new int[pop][];

    for (int i = 0; i < pop; i++)
    {
        friends_history[i] = new int[pop];

        for (int j = 0; j < pop; j++)
        {
            friends_history[i][j] = 0;
        }

    }
}
```

本題

(1)　SIRSモデルをボイドに統合します。

(2)　壁の両側でルールをキャンセルする仕様を導入します。

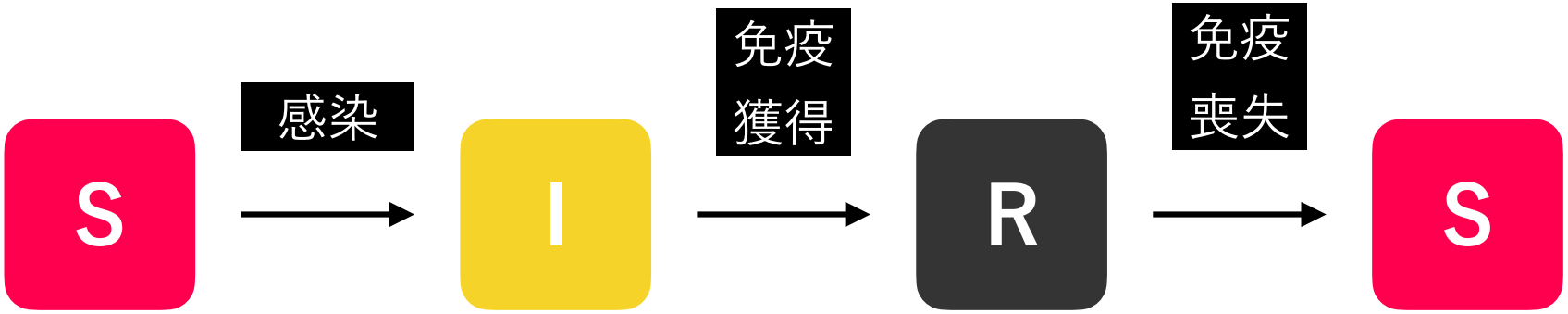
（準備）Boidプレハブの Materialsを変更します



（１）ＳＩＲＳモデル

感染状態

Infected



Susceptible

感受性保持状態

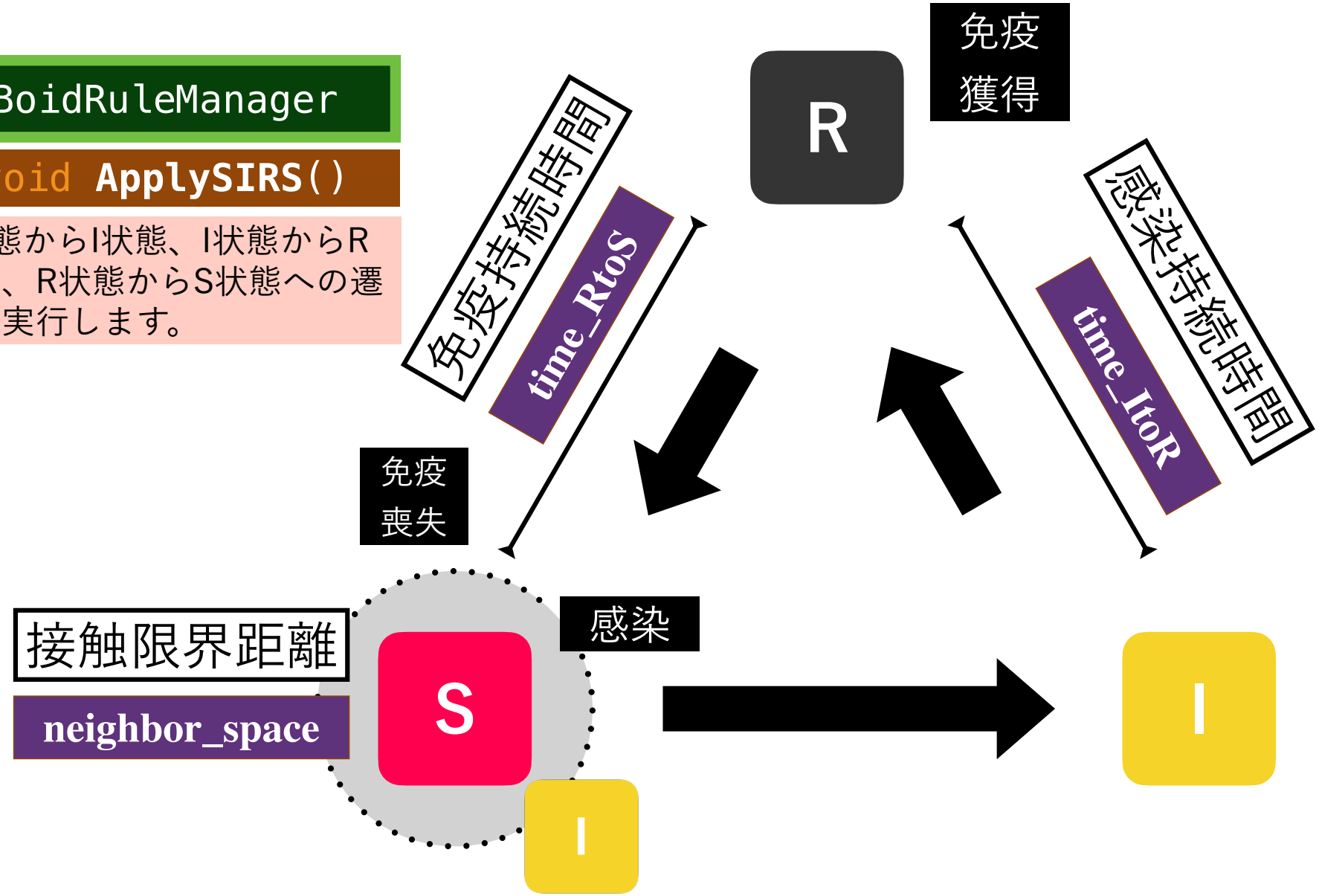
Recovery

回復状態

BoidRuleManager

void ApplySIRS()

S状態からI状態、I状態からR状態、R状態からS状態への遷移を実行します。



BoidManager

<int> neighbor_space

各々のボイドの接触限界距離

BoidRuleManager

<float> time_ItoR,time_RtoS

感染持続時間・免疫持続時間（秒）

SingleBoidクラスの拡張

SingleBoid

<bool> infection

感染の有無

<bool> susceptible

感受性保持状態の有無

<float> timeI, timeR

感染後の経過時間, 回復後の経過時間（秒）

string SIRS()

現在のSIRの状態を返す。S状態の場合”S”を、I状態の場合”I”を, R状態の場合 “R”を返す。

void ResetInfection()

感染状態をリセットする。

個々のボイドが、「S状態」「I状態」「R状態」のうち、どの状態にあるかを調べるために, 本プログラムでは、SingleBoidクラスのSIRSメソッドと、StringクラスのContainsメソッドを組み合わせて使っています。右の例は、i番目のボイドが「I状態」のときの処理を記述したものです。

String

bool Contains(String str)

引数の文字列を含む場合はTRUEを、含まない場合はFALSEを返す。

BoidRuleManager.cs

```
if(boid[i].SIRS().Contains("I")){

    boid[i].timeI += Time.deltaTime;

    if(boid[i].timeI > time_ItoR){
        if(Random.value < 0.05f){
            boid[i].infection = false;
            boid[i].susceptible = false;
            boid[i].timeR = 0f;
        }
    }
}
```

void ApplySIRS()

感染のトリガー（例）

BoidManager.cs

```
/* Sボタンでランダムに感染を発生させる #SIRS */
if (Input.GetKeyDown(KeyCode.S))
{
    this.setRandomInfection();
}
```

Update())

Sボタンを押すと、setRandomInfection関数を実行します。

S ランダムな感染トリガー

```
/* 感染をランダムに発生させる（1%の確率） #SIRS*/
1 reference
private void setRandomInfection()
{
    for (int i = 0; i < pop; i++)
    {
        if (Random.value < 0.01)
        {
            boid[i].infection = true;
        }
    }
}
```

各ボイドについて、1%の確率で、感染状態をTRUE（つまり「I状態」）とします。

同様のフローで、Iボタンで、感染状態をリセットしています。各自で確認してください。

感染状態のVisualizationの例（1/2）

準備として，**BoidManager.cs** に以下を追記します

BoidManager.cs

宣言部

```
//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;
```

Start()

```
void 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[ pop ];
    boidobj = new GameObject[ pop ];

    for (int i = 0; i <  pop; 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.

感染状態のVisualizationの例（2/2）

BoidManager.cs

void ShowInfection()

```
/* 感染状態の可視化 #SIRS*/
private void ShowInfection(){

    for(int i=0;i<pop;i++){

        Renderer r = boidobj[i].GetComponent<Renderer>();

        if(boid[i].SIRS().Contains("S"))
        {
            r.material.color = new Color(1f,0f,0.31f);
            boidobj[i].transform.localScale = new Vector3(1f,1f,1f);
        }

        if(boid[i].SIRS().Contains("I"))
        {
            r.material.color = Color.yellow;
            boidobj[i].transform.localScale = new Vector3(0.8f,0.8f,0.8f);
        }

        if(boid[i].SIRS().Contains("R"))
        {
            r.material.color = new Color(1f,1f,1f);
            boidobj[i].transform.localScale = new Vector3(1f,1f,1f);
        }
    }
}
```

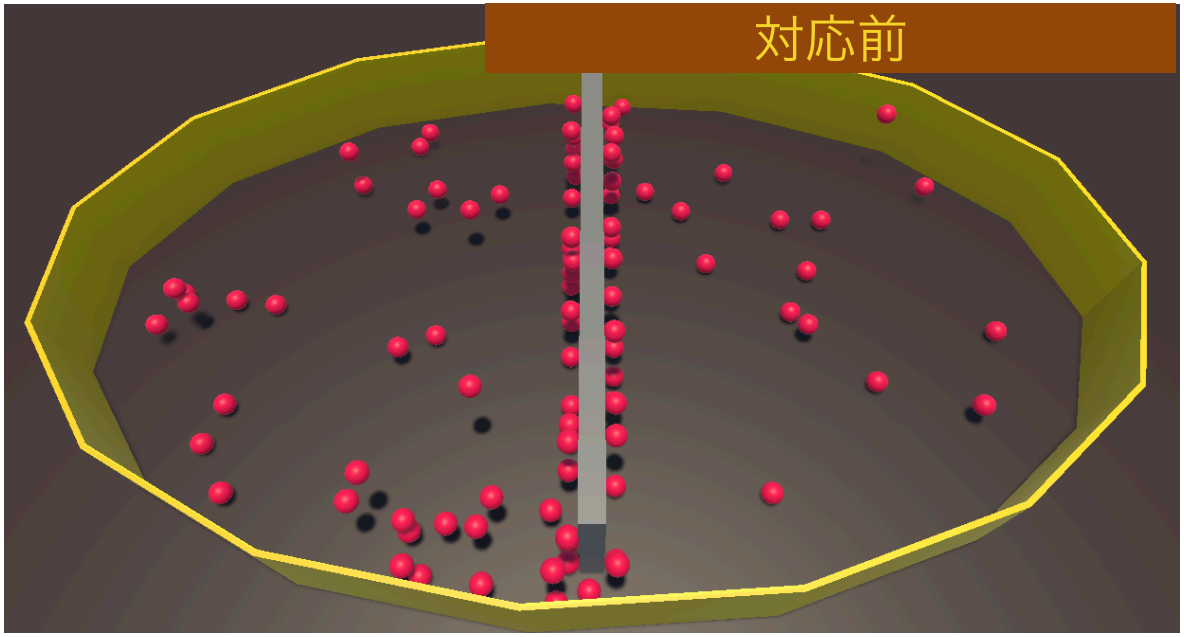
boidobjは、各ボイドのゲームオブジェクトレベルの変数です。ゲームオブジェクトのマテリアルを制御するためにRendererコンポーネントを取り出しておきます。

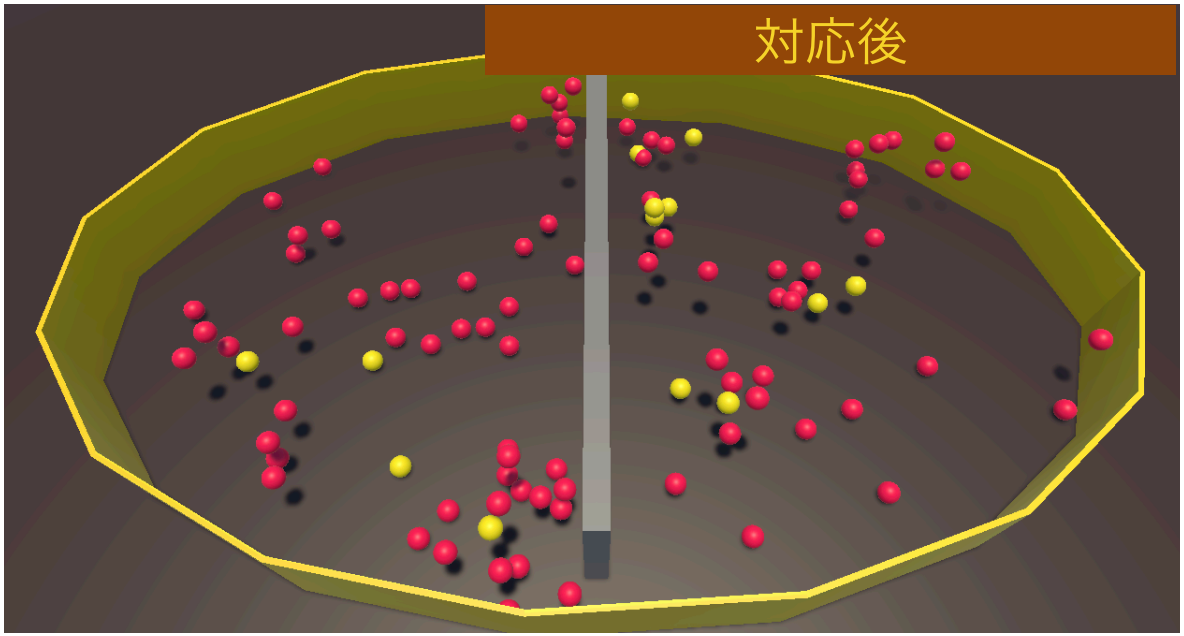
ボイドの感染状態に応じて、色を変えています。
ここでは、感染を黄色で可視化しています。

応用として、感染状態に応じて、大きさを変更することもできます。

実際には、BoidManagerクラスのUpdate関数のなかで、この可視化関数は呼び出されます。

（２）壁の両側で各種のルールをマスク





壁を配置した状況で親近ルールが適用されると，壁の両側のボイドが永久的に引きつけ合うバグが生まれてしまう．

（準備）壁で各種のルールをマスクする方法

手順（方針）

1. 「Wall」Layerを新規作成
Tags and Layers の空きスロット（User Layer 6 など）に Wall を追加。

2. 敷居のCubeを用意
シーンにCubeを置いて敷居にする → **Box Collider**（最初から付いている）を持たせ、そのCubeの **Layer を** Wall に設定。

3. BoidRuleManager に LayerMask を持たせる

```
[SerializeField] private LayerMask wallMask;   // Inspectorで Wall だけにチェック
```

4. ヘルパーは単発 Linecast でOK

```
private bool IsWallBetween(Vector3 a, Vector3 b)
{
    return Physics.Linecast(a, b, wallMask);   // a→b の間に Wall レイヤーのコライダーがあるか
}
```

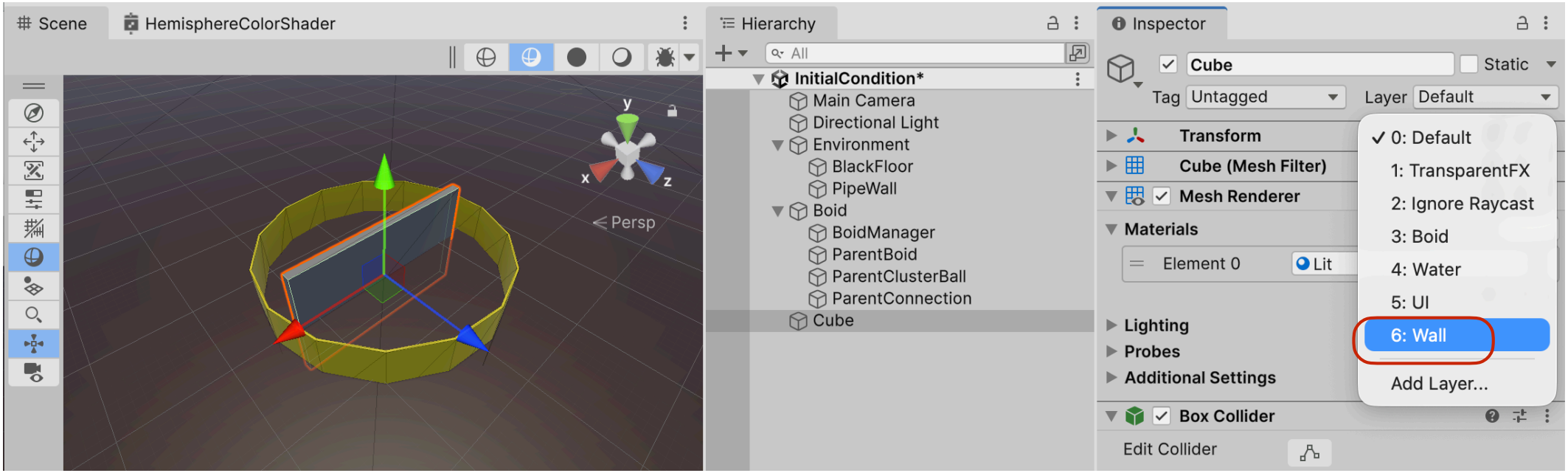
5. 各ルールの判定に追加

```
if (j != i && dis < ivspace && !IsWallBetween(ipos, jpos)) { ... }
```

壁で各種のルールをマスクする方法（1・2）

1. 「Wall」Layerを新規作成
Tags and Layers の空きスロット（User Layer 6 など）に Wall を追加。

2. 敷居のCubeを用意
シーンにCubeを置いて敷居にする → **Box Collider**（最初から付いている）を持たせ、そのCubeの **Layer** を Wall に設定。



適当にCubeを使って壁を配置しましょう（隙間を作って）。ここで、LayerにWallを追加し、自分自身をWallのLayerに割り当てます。加えて、衝突時にルール適用をキャンセルするために、TagもWallに変更しておきます（演習B2の資料を参考にしてください）。

壁で各種のルールをマスクする方法（3・4）

```
3. BoidRuleManager に LayerMask を持たせる
[SerializeField] private LayerMask wallMask;    // Inspectorで Wall
だけにチェック

4. ヘルパーは単発 Linecast でOK
private bool IsWallBetween(Vector3 a, Vector3 b)
{
    return Physics.Linecast(a, b, wallMask);    // a→b の間に Wall
レイヤーのコライダーがあるか
}
```

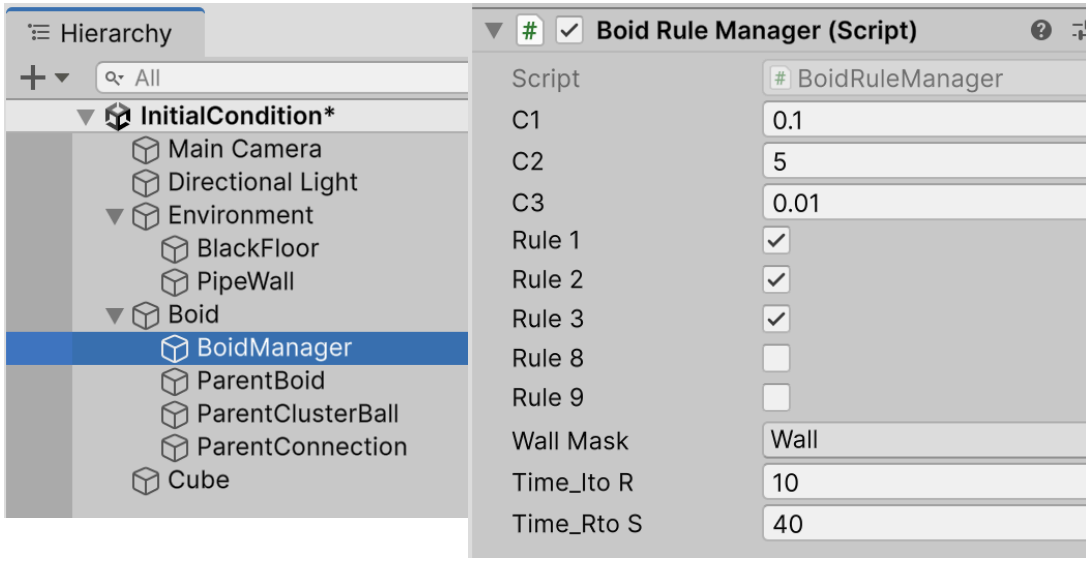
BoidRuleManager.cs

グローバル変数

```
// InspectorでWallだけにチェック
[SerializeField] private LayerMask wallMask;
```

変数にLayerMask型の変数を追加する

1



BoidManagerのインスペクタで、WallMaskにWallを指定します。

3

```
private bool isWallBetween(Vector3 a, Vector3 b)
    {
        return Physics.Linecast(a,b,wallMask);
        //a→bの間にWallレイヤのコライダーがあるか
    }
```

2

壁判定のためのメソッドを用意する。

壁で各種のルールをマスクする方法（５）

５．各ルールの判定に追加

```
if (j != i && dis < ivspace && !IsWallBetween(ipos, jpos)) { ... }
```

BoidRuleManager.cs

```
if (j!=i && dis < ivspace && !isWallBetween(ipos,jpos))
{
    posTotal += jpos;
    count++;
}
```

ApplyRule1()

```
if (j!=i && dis < ivspace && !isWallBetween(ipos,jpos))
{
    velSum += jvel;
    count++;
}
```

ApplyRule2()

```
if (i != j && dis < ineighbor_space
    && dis != 0 && !isWallBetween(ipos,jpos))
{
    cvTotal += c2 * (ipos - jpos)/dis;      //追加
}
```

ApplyRule3()

```
if( i!=j && dis_ij<ineighbor_space
        && boid[j].infection && !isWallBetween(ipos,jpos)){

    if(Random.value < 0.05f){
        boid[i].infection = true;
        boid[i].timeI = 0f;
        break;
    }
}
```

ApplySIRS()

各ルールのメソッドの中で、規則を適用する判定条件に、先に作成した「isWallBetween」を追加します。具体的には「iposとjposの間に壁が無い」という条件が追加されています。