

演習 2 : 集合の知性を設計する

(05) 05/14

A | Unity環境の整備・簡単なルール設計

(06) 05/21 (07) 05/28

B | ボイドルール 1・2・3の実装

(08) 06/04 (09) 06/11

C | 課題 1 : 集合知の解析

(10) 06/18

D 1 | SIR (感染モデル)

(11) 06/25 (12) 07/02 (13) 07/09

D 2 | 課題 2 : マイルール・感染ルール・視点操作

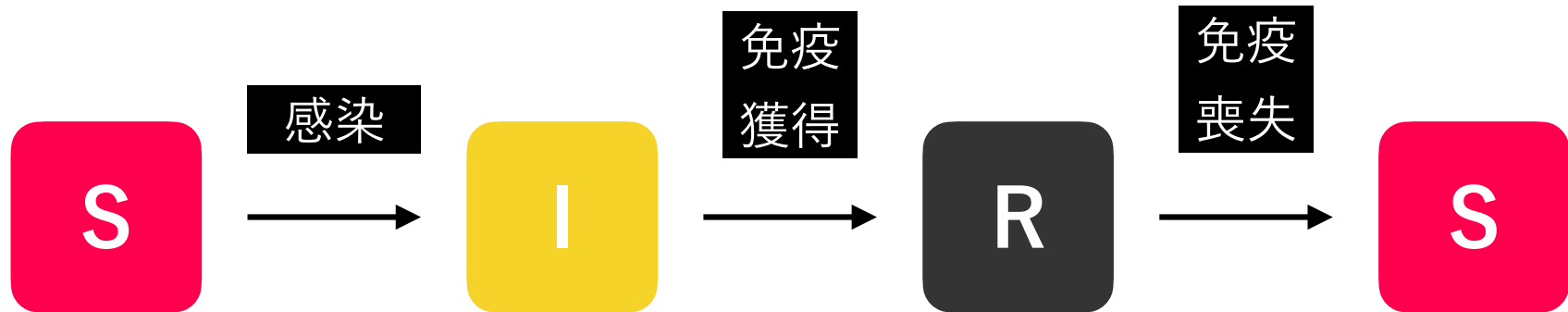
(14-15) 07/16

D 3 | 発表 (One-Minute Movie)

SIRSモデル

感染状態

Infected



Susceptible

感受性保持状態

Recovery

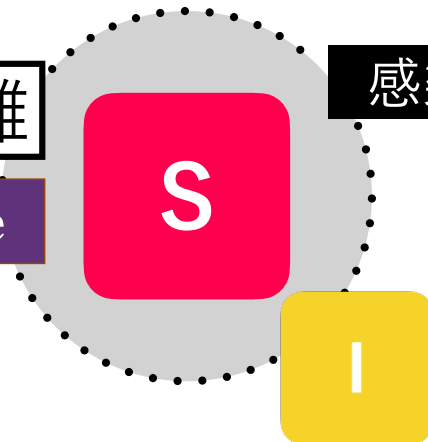
回復状態

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。

接触限界距離
neighbor_space



免疫喪失

免疫持続時間
time_RtoS



免疫獲得

感染持続時間
time_ItoR



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。

```
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()

BoidRuleManager.cs

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

```
//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[ 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)

```
/* 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);
```

```
        }
```

```
    }
```

```
}
```

BoidManager.cs

void ShowInfection()

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 MATERIAL 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。