

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class BoidManager : MonoBehaviour
5 {
6
7
8     //解析オブジェクト
9     BoidClusterAnalysis ana;
10
11     //ボイドルールマネージャ
12     BoidRuleManager rule;
13
14     //飛翔スペース
15     static public int flyspace = 400; //一辺の長さ
16     static public int flyheight = 200; //高さ (3Dモード)
17                                     //速さの最大値
18     static public float speedmax = 100f;
19
20     //二次元モードと三次元モードの切り替え (keyの2と3で切り替え)
21     static public bool mode_2d = true;
22     //解析モード (keyのaで切り替え)
23     static public bool mode_analysis = false;
24     //結合可視モード (keyのcで切り替え)
25     static public bool mode_connectivity = false;
26
27     //視界距離 (見渡せる範囲)
28     public float vision_space = 35f;
29     //接触距離 (ぶつかる範囲)
30     public float neighbor_space = 10f;
31
32     //ボイドの数
33     public int pop = 50;
34
35
36     //ボイドの配列 (インスペクタには非表示)
37     [HideInInspector]
38     public SingleBoid[] boid;
39     [HideInInspector]
40     public GameObject[] boidobj; //追記 #SIRS
41
42
43     //結合状態可視化用の各種オブジェクト
44     private GameObject[] cball; //クラスターの可視化
45     private GameObject[][] connection; //結合の可視化
46
47
48     //ボイド解析用の変数 (ルール作成の演習では全てprivateとする)
49     private float countmax = 3000f; //計測フレーム
50     public float cls_sum = 0f; //クラスターの総計
51     public float cls_count = 0f; //現在のフレーム数
52     public float cls_mean = 0f; //クラスターの平均値
53
54     //新しい知り合いができたフレーム数
55     // (30フレーム以上同一のクラスターにいるペアが新たに生まれた場合)
```

1

```

56     public int make_friends_frame = 0;
57
58     /* 集合知解析では以下の編集の修飾子を「public」に変更します。 */
59     /*
60     public float cls_count = 0f; //現在のフレーム数
61     public float cls_mean = 0f; //クラスターの平均値
62     public int make_friends_frame = 0;
63     */
64
65
66
67
68     void Start()
69     {
70
71         /* ボイドルールマネージャの生成 */
72         rule = this.GetComponent<BoidRuleManager>();
73
74         /* 解析オブジェクトの生成 */
75         ana = this.GetComponent<BoidClusterAnalysis>();
76
77
78         /* ボイドオブジェクト (SingleBoidクラス | スクリプト) */
79         boid = new SingleBoid[pop];
80         GameObject bpar = GameObject.Find("ParentBoid"); //親のゲームオブジェクト
81
82         boidobj = new GameObject[pop]; //追記 #SIRS 2
83
84
85
86         for (int i = 0; i < pop; i++)
87         {
88             //GameObject bobj = Instantiate ((GameObject)Resources.Load
89             ("Boid"));
90             GameObject bobj = Instantiate((GameObject)Resources.Load("Boid"),
91             bpar.transform);
92             boid[i] = bobj.GetComponent<SingleBoid>();
93             boidobj[i] = bobj; //追記 #SIRS 3
94         }
95
96
97         /* 視界範囲内にあるボイドを結ぶ結合子 (ゲームオブジェクト) */
98
99         //サイズは、重複のない二次元配列
100        connection = new GameObject[pop][];
101        GameObject cpar = GameObject.Find("ParentConnection"); //親のゲームオブ
102        ジェクトを探索
103
104        for (int i = 0; i < pop; i++)
105        {
106            connection[i] = new GameObject[pop - i - 1];
107            for (int j = i + 1; j < pop; j++)

```

```

100         for (int j = i + 1; j < pop; j++)
107         {
108             connection[i][j - i - 1] =
Instantiate((GameObject)Resources.Load("Connection"), cpar.transform);
109         }
110     }
111
112
113     /* クラスターを包む球 (ゲームオブジェクト) */
114
115     //ボイドの数だけ配備しておく
116     cball = new GameObject[pop];
117     GameObject cbpar = GameObject.Find("ParentClusterBall"); //親のゲームオ
オブジェクトを探索
118
119     for (int i = 0; i < pop; i++)
120     {
121         cball[i] = Instantiate((GameObject)Resources.Load("ClusterBall"),
cbpar.transform);
122     }
123
124
125 }
126
127 void Update()
128 {
129
130
131     //すべてのconnetion・cballを非表示にする。
132     ResetActive();
133
134     //ボイドルールマネージャによるルールの適用
135     rule.SetBoid(this);
136     rule.ApplyRules();
137
138
139
140     //結合状態の可視化 (CでON/OFF)
141     if (mode_connectivity)
142     {
143         ShowConnectivity();
144     }
145
146     //SIRSルールの適用 #SIRS
147     rule.ApplySIRS();
148     ShowInfection();
149
150     if (mode_analysis)
151     {
152
153         ana.SetBoid(this);
154         //個体数 (1/10) をクラスター成立の要件とする。
155         ana.SetMinimumPop((int)Mathf.Floor(pop / 10f));
156         //クラスタを計算
157         ana.CountCluster();
158         //新しく「知り合い」が成立したボイドの数

```

4

```

150 //新しい「知り合い」が成立したボイドの数
159 int new_friends = ana.UpdFriendsHistory(30);
160 //クラスタに半透明の球を描画
161 ShowClusterBall();
162
163 //カウンタが最大値（初期値3000）となるまで解析を続ける
164 if (cls_count < countmax)
165 {
166     //クラスタのフレーム内の総数
167     cls_sum += ana.csum;
168     //クラスタの1フレーム平均
169     cls_mean = cls_sum / cls_count;
170     //計算フレーム数
171     cls_count++;
172
173     //新しい知り合いが一件でも成立すれば、
174     //「知り合い成立機会」を1つ増やす。
175     if (new_friends >= 1)
176     {
177         make_friends_frame += 1;
178     }
179
180
181 }
182
183
184
185
186
187 }
188
189
190
191 /* Keyイベント */
192
193 /* 全てのボイドの位置・速度を初期化 (I) */
194 if (Input.GetKeyDown(KeyCode.I))
195 {
196     InitBoidPosition();
197     InitBoidVelocity();
198
199     /* 集合知解析では以下の行をアンコメントします。 */
200
201     cls_sum = 0f;
202     cls_count = 0f;
203     make_friends_frame = 0;
204     ana.InitFriendsHistory(this);
205
206     /* ボイドの感染状態をリセットする #SIRS */
207     ResetBoidInfection();
208
209 }
210
211 if (Input.GetKeyDown(KeyCode.R))
212 {
213     this.RegenerateVelocity();

```

5

```
213         this.ReverseVelocity();
214     }
215
216     /* Sボタンでランダムに感染を発生させる #SIRS */
217     if (Input.GetKeyDown(KeyCode.S))
218     {
219         this.setRandomInfection();
220     }
221
222
223     /* 解析モードの切り替え (A) */
224     if (Input.GetKeyDown(KeyCode.A))
225     {
226         mode_analysis = !mode_analysis;
227     }
228
229     /* ボイド間の接続モードの切り替え (C) */
230     if (Input.GetKeyDown(KeyCode.C))
231     {
232         mode_connectivity = !mode_connectivity;
233     }
234
235     /* 二次元モード (2) */
236     if (Input.GetKeyDown(KeyCode.Alpha2))
237     {
238         mode_2d = true;
239     }
240
241     /* 三次元モード (3) */
242     if (Input.GetKeyDown(KeyCode.Alpha3))
243     {
244         mode_2d = false;
245     }
246
247     if (Input.GetMouseButton(0))
248     {
249         rule.ApplyRuleAttractor(new Vector3(0f, 100f, 0f));
250         //rule.ApplyRuleAttractor(boid[0].pos);
251     }
252
253 }
254
255
256
257
258
259 private void ReverseVelocity()
260 {
261     for (int i = 0; i < pop; i++)
262     {
263         Vector3 ivel = boid[i].vel;
264         boid[i].SetVelocity(-ivel);
265     }
266 }
267
```

6

```
268
269
270  /* 全てのボイドの位置をランダムに初期化*/
271  private void InitBoidPosition()
272  {
273      for (int i = 0; i < pop; i++)
274      {
275          boid[i].SetRandomPosition();
276      }
277  }
278
279  /* 全てのボイドの速度をランダムに初期化*/
280  private void InitBoidVelocity()
281  {
282      for (int i = 0; i < pop; i++)
283      {
284          boid[i].SetRandomVelocity();
285      }
286  }
287
288  /* 全てのボイドの感染状態をリセットする #SIRS */
289  private void ResetBoidInfection()
290  {
291      for (int i = 0; i < pop; i++)
292      {
293          boid[i].ResetInfection();
294      }
295  }
296
297  /* ボイド同志の結合状態 (vision_space未満の距離) を可視化 */
298  private void ShowConnectivity()
299  {
300
301      for (int i = 0; i < pop; i++)
302      {
303          Vector3 ipos = boid[i].pos;
304
305          for (int j = i + 1; j < pop; j++)
306          {
307              Vector3 jpos = boid[j].pos;
308              float dis = Vector3.Distance(ipos, jpos);
309
310              if (dis < vision_space)
311              {
312                  connection[i][j - i - 1].SetActive(true);
313                  connection[i][j - i - 1].transform.position = (ipos +
jpos) / 2;
314
315                  connection[i][j - i - 1].transform.rotation =
Quaternion.LookRotation(jpos - ipos);
316                  connection[i][j - i - 1].transform.localScale =
new Vector3(0.5f, 0.5f, (dis / 2f) + 10f);
317              }
318              else
319              {
320                  connection[i][j - i - 1].SetActive(false);
321              }

```

7

```
322     }
323
324     }
325 }
326 }
327
328 /* クラスター (viosion_space未満の集団) を可視化*/
329 private void ShowClusterBall()
330 {
331     for (int i = 0; i < pop; i++)
332     {
333         cball[i].SetActive(false);
334     }
335
336     for (int g = 0; g < ana.csum; g++)
337     {
338         cball[g].transform.position = ana.cog[g];
339         float d = ana.rad[g] * 2f;
340         cball[g].transform.localScale = new Vector3(d, d, d);
341         cball[g].SetActive(true);
342     }
343 }
344
345 }
```

```
347
348 /* 感染状態の可視化 #SIRS */
349 private void ShowInfection()
350 {
351     for (int i = 0; i < pop; i++)
352     {
353         Renderer r = boidobj[i].GetComponent<Renderer>();
354
355         if (boid[i].SIRS().Contains("S"))
356         {
357             r.material.color = new Color(1f, 0f, 0.31f);
358             boidobj[i].transform.localScale = new Vector3(10f, 10f, 10f);
359         }
360         if (boid[i].SIRS().Contains("I"))
361         {
362             r.material.color = Color.yellow;
363             boidobj[i].transform.localScale = new Vector3(8f, 8f, 8f);
364         }
365         if (boid[i].SIRS().Contains("R"))
366         {
367             r.material.color = new Color(0.2f, 0.2f, 0.2f);
368             boidobj[i].transform.localScale = new Vector3(10f, 10f, 10f);
369         }
370     }
371 }
372
373
374
375
376
```

8

```
377
378
379
380
381  /* 感染をランダムに発生させる (1%の確率) #SIRS*/
382  private void setRandomInfection()
383  {
384      for (int i = 0; i < pop; i++)
385      {
386          if (Random.value < 0.01)
387          {
388              boid[i].infection = true;
389          }
390      }
391  }
392
393
394  private void ResetActive()
395  {
396
397      for (int i = 0; i < pop; i++)
398      {
399          cball[i].SetActive(false);
400      }
401
402
403      for (int i = 0; i < pop; i++)
404      {
405          for (int j = i + 1; j < pop; j++)
406          {
407              connection[i][j - i - 1].SetActive(false);
408          }
409      }
410
411  }
412
413
414
415 }
416
```

9