```
using UnityEngine;
using System.Collections;

public class BoidManager : MonoBehaviour
{


    //解析オブジェクト
    BoidClusterAnalysis ana;

    //ボイドルールマネージャ
    BoidRuleManager rule;

    //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;

    //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）
    static public bool mode_2d = true;
    //解析モード（keyのaで切り替え）
    static public bool mode_analysis = false;
    //結合可視モード（keyのcで切り替え）
    static public bool mode_connectivity = false;

    //視界距離（見渡せる範囲）
    public float vision_space = 35f;
    //接触距離（ぶつかる範囲）
    public float neighbor_space = 10f;

    //ボイドの数
    public int pop = 50;


    //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; //追記 #SIRS

    //結合状態可視化用の各種オブジェクト
    private GameObject[] cball;         //クラスターの可視化
    private GameObject[][] connection; //結合の可視化


    //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）
    private float countmax = 3000f; //計測フレーム
    public float cls_sum = 0f; //クラスターの総計
    public float cls_count = 0f; //現在のフレーム数
    public float cls_mean = 0f; //クラスターの平均値

    //新しい知り合いができたフレーム数
    //（30フレーム以上同一のクラスタにいるペアが新たに生まれた場合）
```

1

```
    public int make_friends_frame = 0;

    /*  集合知解析では以下の編集の修飾子を「public」に変更します。 */
    /*
  public float cls_count = 0f; //現在のフレーム数
  public float cls_mean = 0f; //クラスターの平均値
  public int make_friends_frame = 0;
  */




    void Start()
    {

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

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

        boidobj = new GameObject[pop]; //追記 #SIRS   2

        for (int i = 0; i < pop; i++)
        {
            //GameObject bobj = Instantiate ((GameObject)Resources.Load("Boid"));
            GameObject bobj = Instantiate((GameObject)Resources.Load("Boid"), bpar.transform);
            boid[i] = bobj.GetComponent<SingleBoid>();

            boidobj[i] = bobj; //追記 #SIRS   3
        }



        /* 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） */

        //サイズは、重複のない二次元配列
        connection = new GameObject[pop][];
        GameObject cpar = GameObject.Find("ParentConnection"); //親のゲームオブジェクトを探索

        for (int i = 0; i < pop; i++)
        {
            connection[i] = new GameObject[pop - i - 1];
            for (int j = i + 1; j < pop; j++)
```

```
            for (int j = i + 1; j < pop; j++)
            {
                connection[i][j - i - 1] = Instantiate((GameObject)Resources.Load("Connection"), cpar.transform);
            }
        }


        /* クラスターを包む球（ゲームオブジェクト） */

        //ボイドの数だけ配備しておく
        cball = new GameObject[pop];
        GameObject cbpar = GameObject.Find("ParentClusterBall"); //親のゲームオブジェクトを探索

        for (int i = 0; i < pop; i++)
        {
            cball[i] = Instantiate((GameObject)Resources.Load("ClusterBall"), cbpar.transform);
        }


    }

    void Update()
    {


        //すべてのconnetion・cballを非表示にする.
        ResetActive();

        //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();



        //結合状態の可視化（CでON/OFF）
        if (mode_connectivity)
        {
            ShowConnectivity();
        }

        //SIRSルールの適用 #SIRS
        rule.ApplySIRS();
        ShowInfection();

        if (mode_analysis)
        {

            ana.SetBoid(this);
            //個体数（1/10）をクラスター成立の要件とする。
            ana.SetMinimumPop((int)Mathf.Floor(pop / 10f));
            //クラスタを計算
            ana.CountCluster();
            //新しく「知り合い」が成立したボイドの数
```

4

```
            //新しく「知り合い」が成立したボイドの数
            int new_friends = ana.UpdFriendsHistory(30);
            //クラスタに半透明の球を描画
            ShowClusterBall();

            //カウンタが最大値（初期値3000）となるまで解析を続ける
            if (cls_count < countmax)
            {
                //クラスタのフレーム内の総数
                cls_sum += ana.csum;
                //クラスタの1フレーム平均
                cls_mean = cls_sum / cls_count;
                //計算フレーム数
                cls_count++;

                //新しい知り合いが一件でも成立すれば、
                //「知り合い成立機会」を1つ増やす。
                if (new_friends >= 1)
                {
                    make_friends_frame += 1;
                }


            }









        }



        /* Keyイベント */

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

            /*  集合知解析では以下の行をアンコメントします。 */

            cls_sum = 0f;
            cls_count = 0f;
            make_friends_frame = 0;
            ana.InitFriendsHistory(this);

            /* ボイドの感染状態をリセットする #SIRS */
            ResetBoidInfection();

        }

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

5

```
            this.ReverseVelocity();
        }


        /* 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();
        }


        /* 解析モードの切り替え（A） */
        if (Input.GetKeyDown(KeyCode.A))
        {
            mode_analysis = !mode_analysis;
        }

        /* ボイド間の接続モードの切り替え（C） */
        if (Input.GetKeyDown(KeyCode.C))
        {
            mode_connectivity = !mode_connectivity;
        }

        /* 二次元モード（2） */
        if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            mode_2d = true;
        }

        /* 三次元モード（3） */
        if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            mode_2d = false;
        }

        if (Input.GetMouseButton(0))
        {
            rule.ApplyRuleAttractor(new Vector3(0f, 100f, 0f));
            //rule.ApplyRuleAttractor(boid[0].pos);

        }


    }


    private void ReverseVelocity()
    {
        for (int i = 0; i < pop; i++)
        {
            Vector3 ivel = boid[i].vel;
            boid[i].SetVelocity(-ivel);

        }
    }
```

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                     connection[i][j - i - 1].transform.rotation =
315                         Quaternion.LookRotation(jpos - ipos);
316                     connection[i][j - i - 1].transform.localScale =
317                         new Vector3(0.5f, 0.5f, (dis / 2f) + 10f);
318                 }
319                 else
320                 {
321                     connection[i][j - i - 1].SetActive(false);
```

7

```
                }

            }
        }
    }

    /* クラスター（viosion_space未満の集団）を可視化*/
    private void ShowClusterBall()
    {

        for (int i = 0; i < pop; i++)
        {
            cball[i].SetActive(false);
        }

        for (int g = 0; g < ana.csum; g++)
        {
            cball[g].transform.position = ana.cog[g];
            float d = ana.rad[g] * 2f;
            cball[g].transform.localScale = new Vector3(d, d, d);
            cball[g].SetActive(true);
        }

    }

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

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