```
using UnityEngine;
using System.Collections;

public class SingleBoid : MonoBehaviour
{
  /* 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
  public float vision_space;  //視界距離
  public float neighbor_space; //接触距離

  /* 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


  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
  }

  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）

  }

  //ボイドが境界を出たら，速度を反転させる.
  private void Rebound(){
    if ((pos.x > xmax && vel.x > 0) || (pos.x < xmin && vel.x < 0)) {
      rb.velocity = new Vector3 (-vel.x, vel.y, vel.z);
      this.vel = this.rb.velocity;
    }
    if ((pos.y > ymax && vel.y>0) || (pos.y < ymin && vel.y<0)){
      rb.velocity = new Vector3 (vel.x, -vel.y, vel.z);
      this.vel = this.rb.velocity;

    }
    if ((pos.z > zmax && vel.z>0) || (pos.z < zmin && vel.z<0)) {
      rb.velocity = new Vector3 (vel.x, vel.y, -vel.z);
      this.vel = this.rb.velocity;
    }
  }

```

```
  //速さが規定値を超えている場合，抑制する.
  private void LimitVelocity(){

    float speed = Vector3.Magnitude (vel);

    if (speed > speedmax) {
      rb.velocity *= speedmax / speed;
    }
    this.vel = this.rb.velocity;

  }

  //高さを0とする（二次元モードのとき）.
  private void ConstrainHeight(){
    if (BoidManager.mode_2d){
      pos = new Vector3 (pos.x, 0, pos.z);
      this.transform.position = pos;
    }
  }

  //境界線の決定
  public void SetBorder(){

    xmax = 0.5f * BoidManager.flyspace;
    xmin = -0.5f *  BoidManager.flyspace;
    ymin = 0f;
    ymax = BoidManager.flyheight;
    zmax = 0.5f *  BoidManager.flyspace;
    zmin = -0.5f *  BoidManager.flyspace;

  }

  //位置をランダムに決定
  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;

  }
  //速度をランダムに決定
  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;

  }
  //速度の設定
  public void SetVelocity(Vector3 v){
```

```
  public void SetVelocity(Vector3 v){
    this.rb.velocity = v;
    this.vel = this.rb.velocity;
  }
  //視界距離の設定
  public void SetVisionSpace(float vs){
    this.vision_space = vs;
  }
  //接触距離の設定
  public void SetNeighborSpace(float ns){
    this.neighbor_space = ns;
  }

  /* SIRS関係の変数 （6月23日）*/
  public bool infection = false;  //感染の有無
  public bool susceptible = true; //感染能力の有無
  public float timeI = 0f;  //感染後の経過時間（秒）
  public float timeR = 0f;  //回復後の経過時間（秒）

  // 感染状態をリセットする
  public void ResetInfection(){

    infection = false;  //感染を無くし、
    susceptible = true; //免疫も無くす
    timeI = 0f;
    timeR = 0f;

  }

  // 現在の感染状態を返す関数
  public string SIRS(){

    /* （免疫を失い）感染能力の有る状態（Susceptible） */
    if(!infection && susceptible){
      return "S";
    }
    /* 感染している状態（Infection） */
    if(infection){
      return "I";
    }
    /* 感染から回復し免疫のある状態（Recovery） */
    if(!infection && !susceptible){
      return "R";
    }

    return "";
  }

}
```

1

2

3