演習２：集合の知性を設計する

(05)　05/14

Ａ｜Unity環境の整備・簡単なルールの設計

(06) 05/21　(07)　05/28

Ｂ｜ボイドルール１・２・３の実装

(08)　06/04

Ｃ｜課題１：集合知の解析

(09)　06/11　(10)　06/18

Ｄ１｜ＳＩＲ（感染モデル）

(11)　06/25　(12)　07/02　(13)　07/09

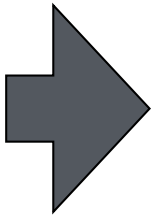
Ｄ２｜課題２：マイルール・感染ルール・視点操作

(14-15)　07/16

Ｄ３｜発表（One-Minute Movie）

SingleBoid.cs

```
28      void Update ()
29      {
30          //位置と速度の更新
31          pos = this.transform.position;
32          vel = this.rb.velocity;
33
34          Rebound ();             //境界判定
35          //LimitVelocity ();     //速度制限
36          setGravity();           //重力の設定
37
38      }
```



```
28      void Update ()
29      {
30          //位置と速度の更新
31          pos = this.transform.position;
32          vel = this.rb.velocity;
33
34          Rebound ();             //境界判定
35          LimitVelocity ();       //速度制限
36          setGravity();           //重力の設定
37
38      }
```

SingleBoid.csのLimitVelocity関数（速度の上限付与）がデフォルトでコメントアウトされているので、コメントアウトを解除してください。前回のバグが解消されます。

演習２- B

ボイドルールの設計（ルール１）

親近行動 を設計します.

BoidManager.cs

Update()

```
if(Input.GetMouseButton(0))
{
    rule.ApplyRuleAttractor(new Vector3(0f,3f,0f));
}
```

マウスボタンを押すと，ApplyRuleAttractor が作動

BoidRuleManager.cs

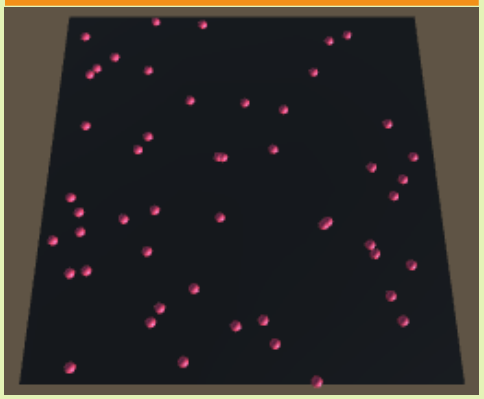
```
//ルールの係数
public float c1 = 0.1f;
public float c2 = 5.0f;
public float c3 = 0.01f;
```

```
//ルール1の適用
if (rule1) {
    ApplyRule1 ();
}
```

ApplyRule()

BoidManager・BoidRuleManger・SingleBoidクラスの関係

BoidManager

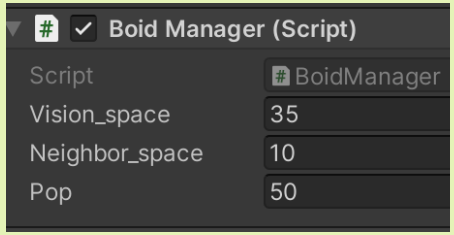


ボイドの描画

ボイド全体の管理

各ルールの適用

解析の実行



BoidManagerクラスは, ボイドの集団をフィールドとして管理します。

BoidRuleManagerは, 各種のボイド間の相互作用（ルール）の適用の有無（bool rule1, rule2, rule3）、そしてルールの具体的な内容（void ApplyRuleX）を管理します.

個々のボイドの位置・速度の更新は, SingleBoidクラスで管理されています.

Single Boid

ボイド単体の振る舞い

SingleBoid.cs

BoidRule Manager

ルールの適用の有無

<bool> rule1

<bool> rule2

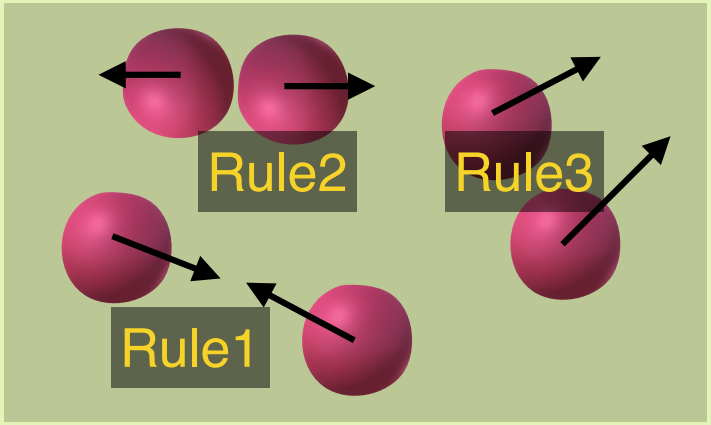
<bool> rule3

void ApplyRule1()

void ApplyRule2()

void ApplyRule3()

具体的なルールの記述（演習の対象）



BoidManager・BoidRuleManagerクラスのpublic変数

- BoidManager・BoidRuleManagerのいくつかのフィールドについては, インスペクタビューから設定可能な状態となっています. 初期状態では, すべてのルールは未設計のため, 各ボイドは相互作用をせずに, 初期速度を維持したまま空間内を（ビリヤードのように）ただただ動き回ります.

BoidManager

<int> vision_space

各々のボイドの視界距離

<int> neighbor_space

各々のボイドの接触限界距離

<int> pop

ボイドの総数（開始後は変更不可）

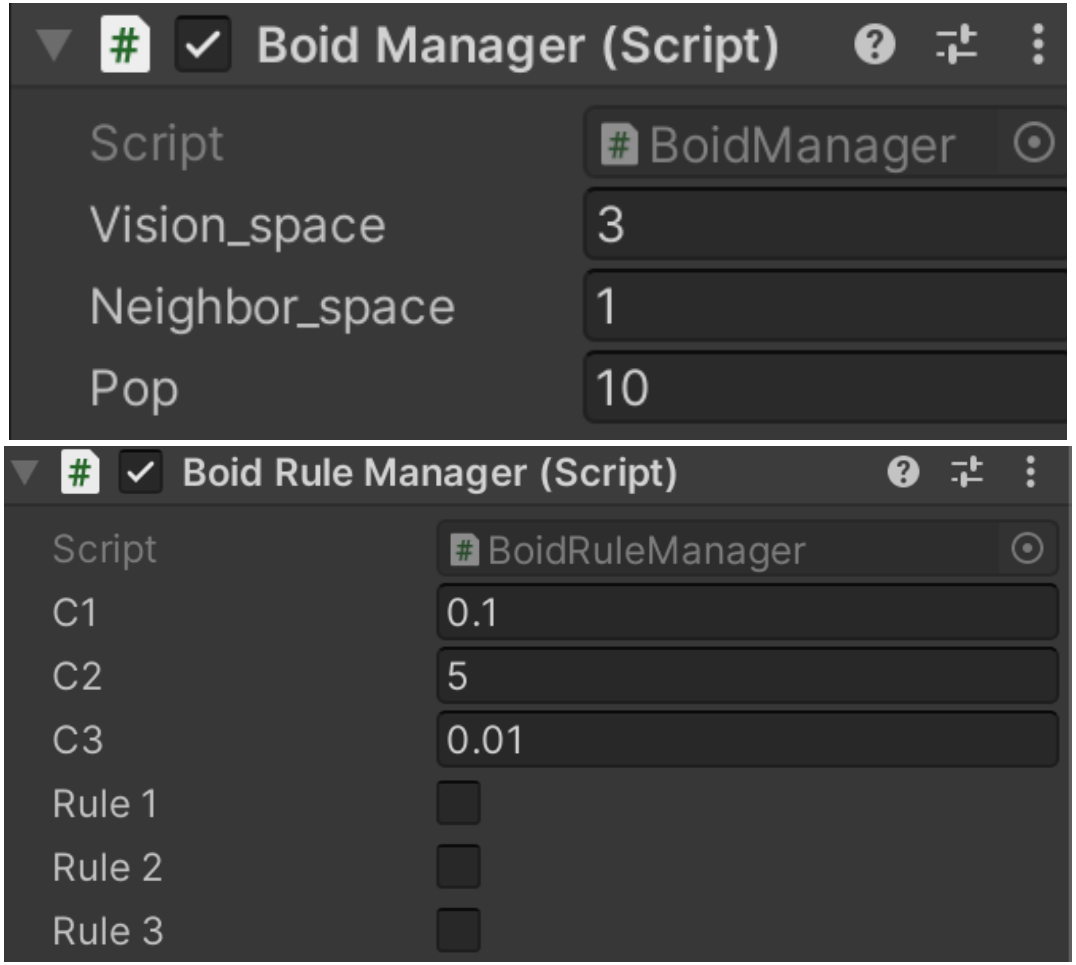
BoidRuleManager

<bool> rule1,rule2,rule3

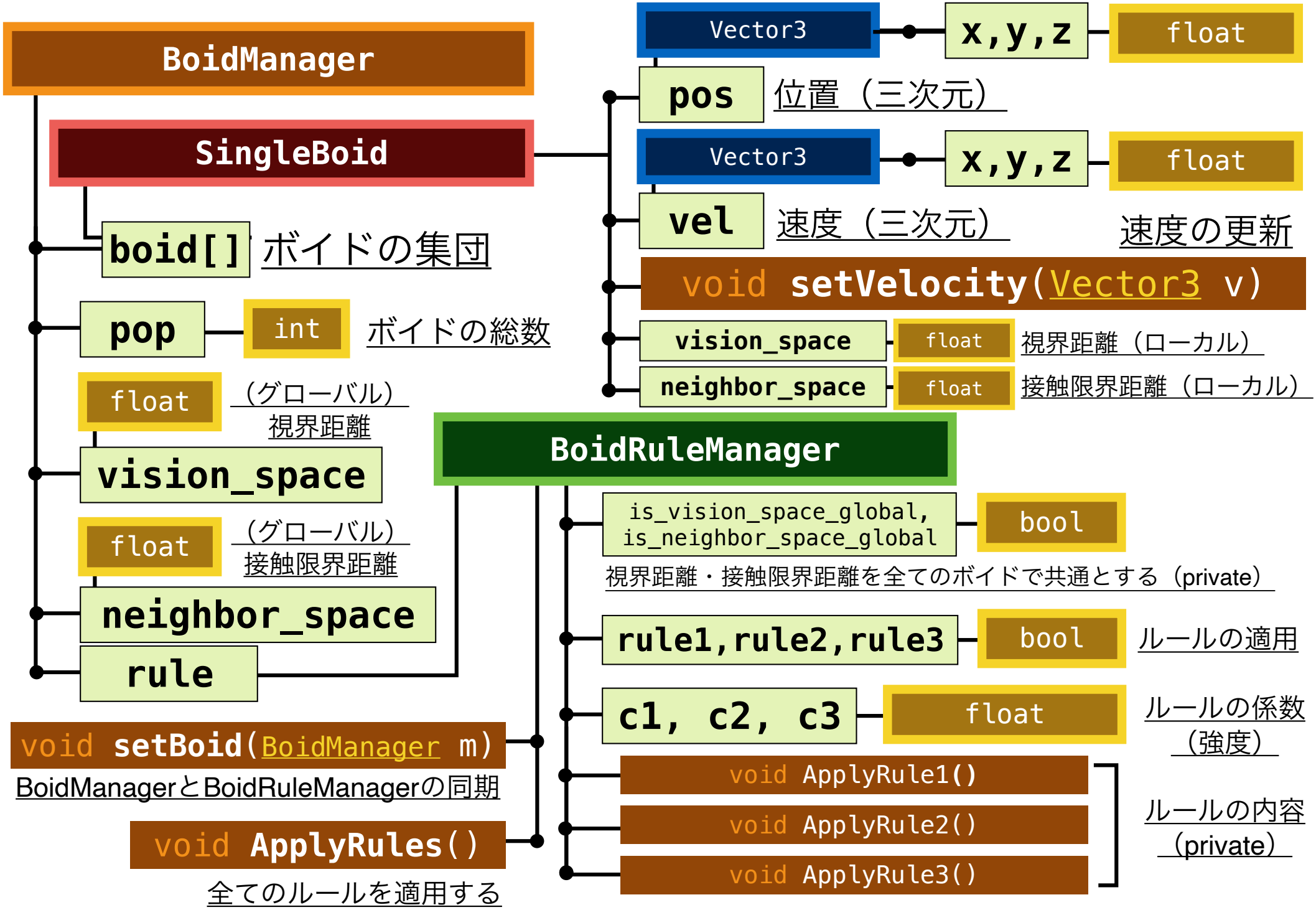
ルール1・2・3の適用の有無

<float> c1,c2,c3

各ルールの影響度（係数）



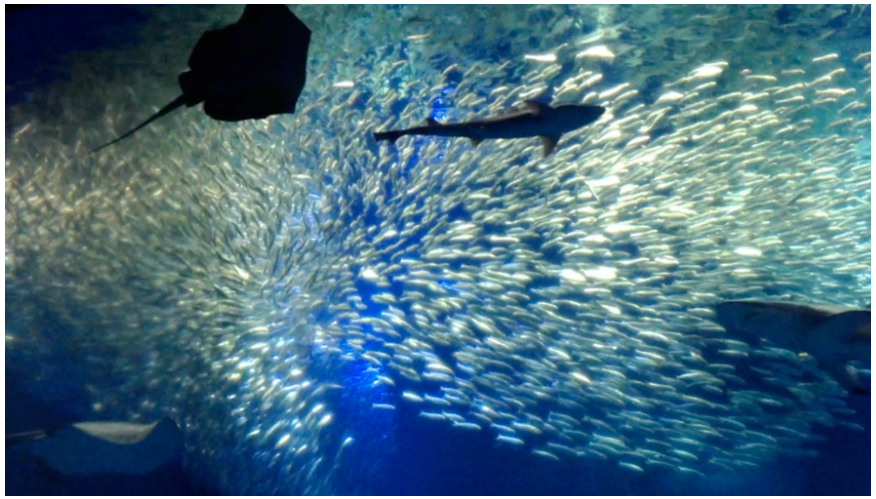
BoidManager / SingleBoid / BoidRuleManagerの関係

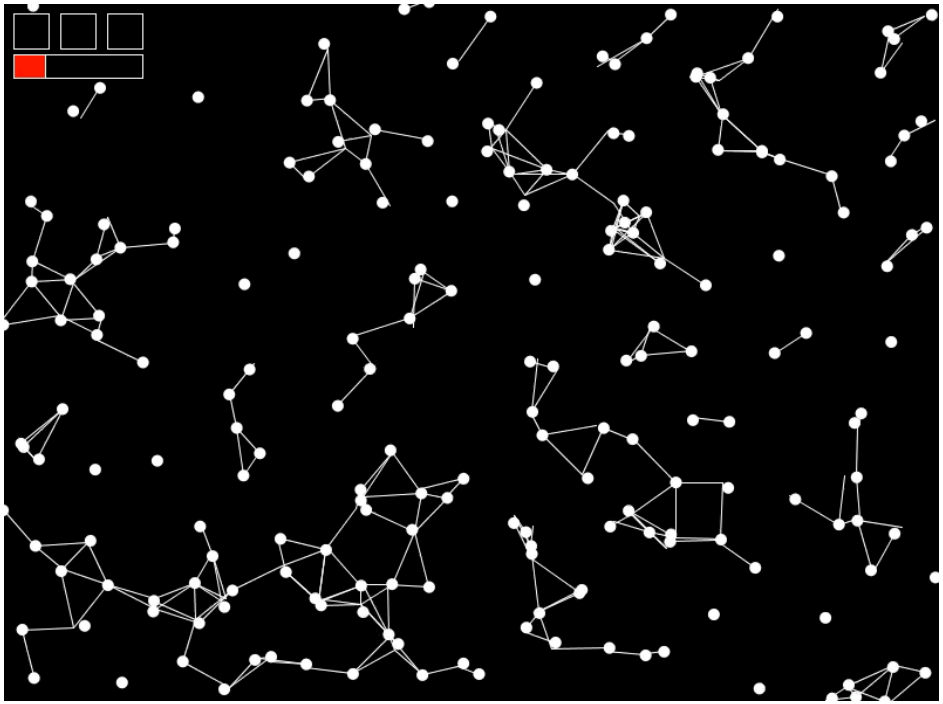


生命と生命のリズム（同調）

集団の同調

Cooperativeness / 群知能





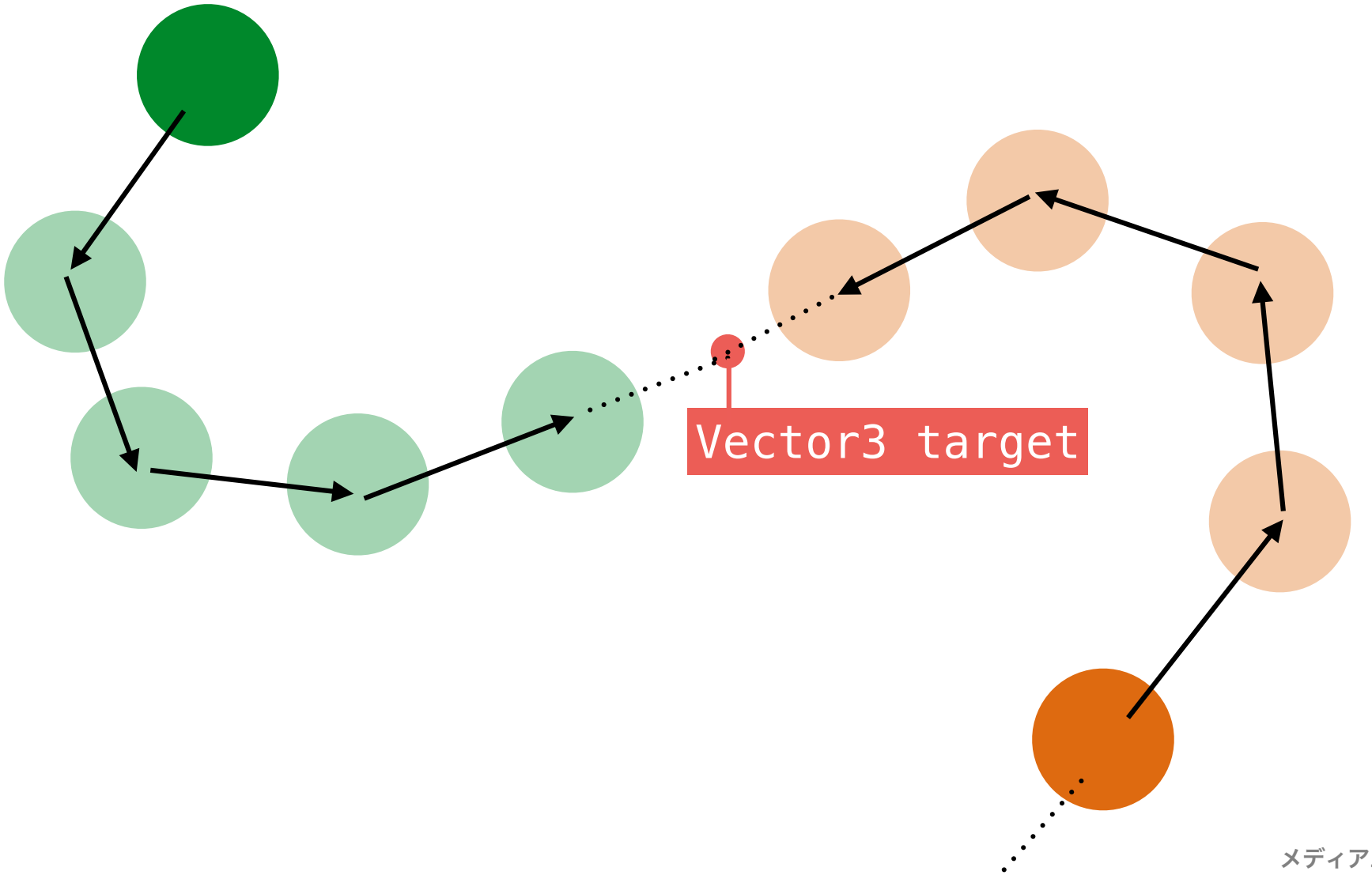
boids（ボイド; bird-like droids）

準備

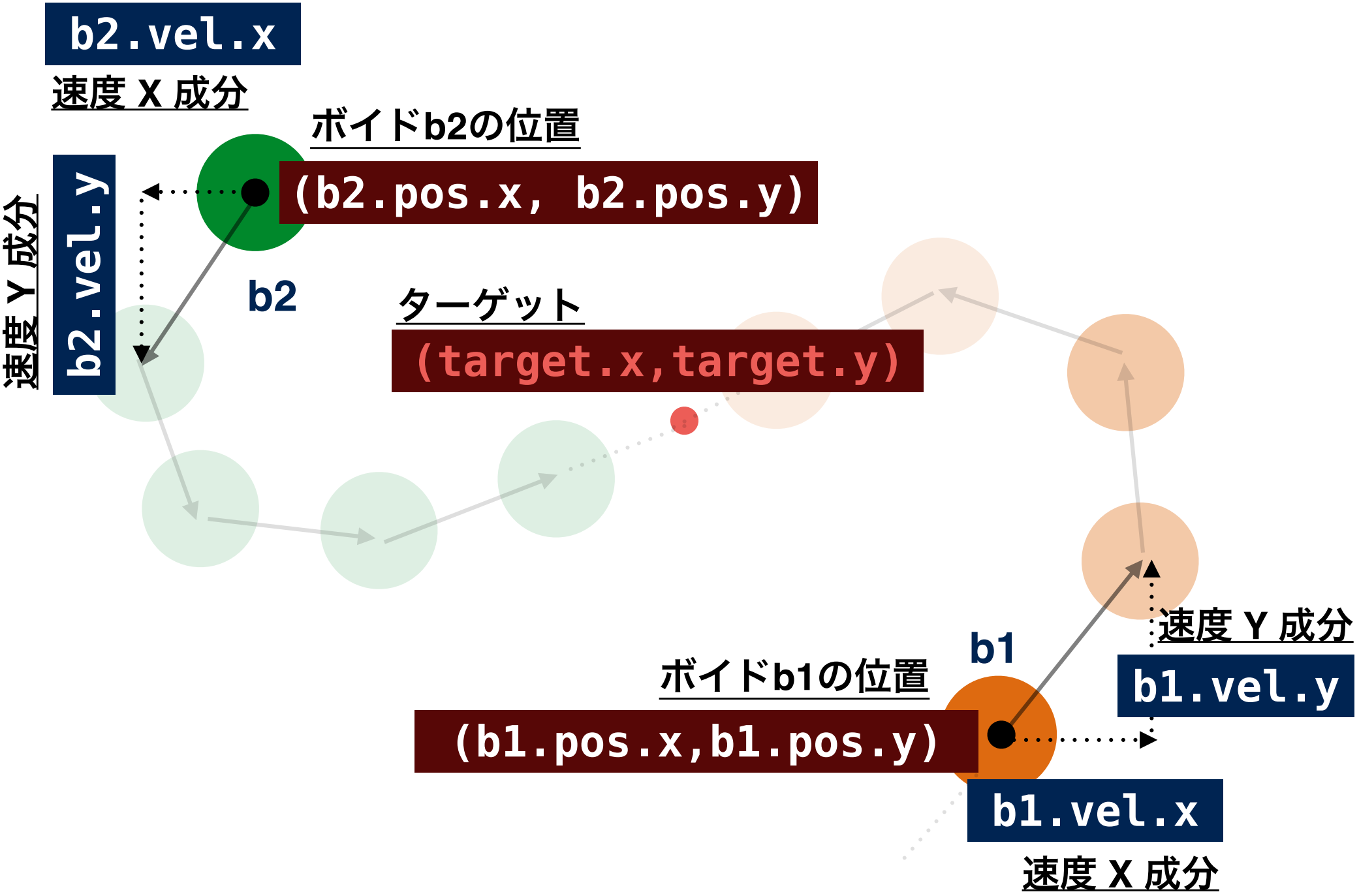
`BoidRuleManager.cs` に、 `void ApplyRuleAttractor(Vector3 target)`

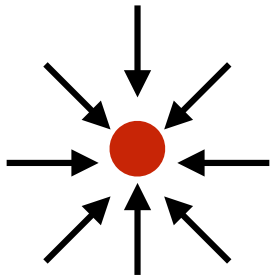
を定義し，

ボイドの速度を，引数で指定された位置へと徐々に修正するルールを追加してください．



準備（二次元で図示します.）



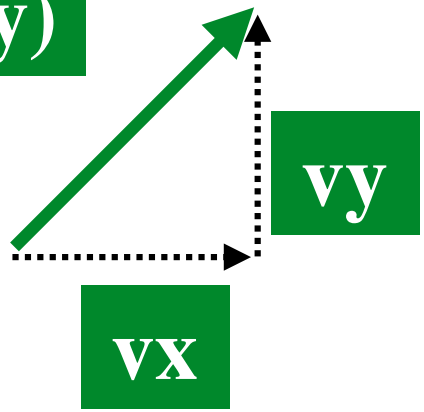


速度を, 指定された点に（徐々に）向けるためのベクトル計算

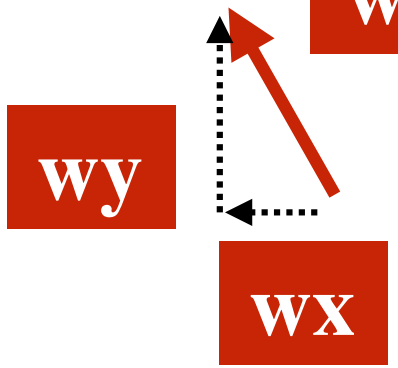
位置による引き込み（引力）

ベクトルの加算

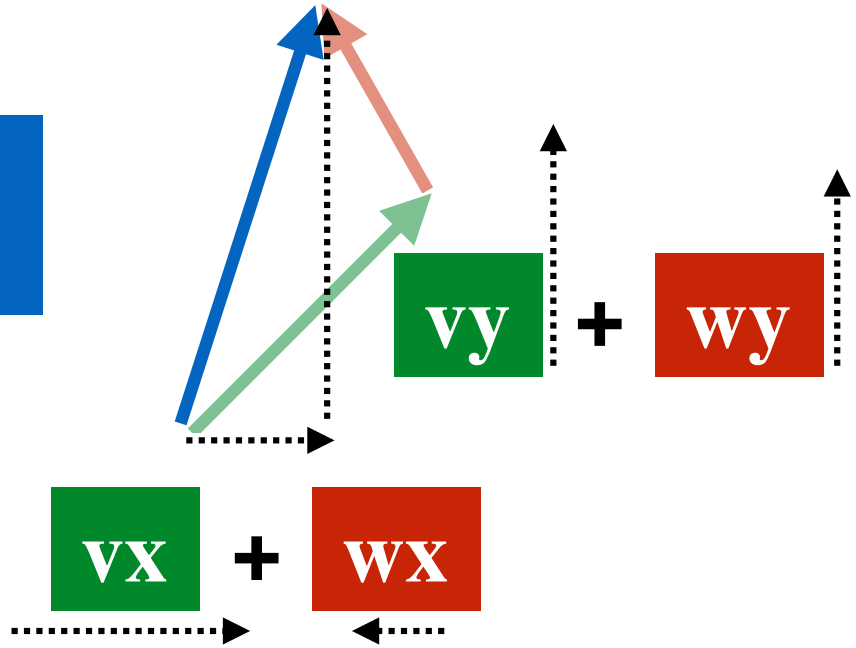
$v' = (vx, vy)$



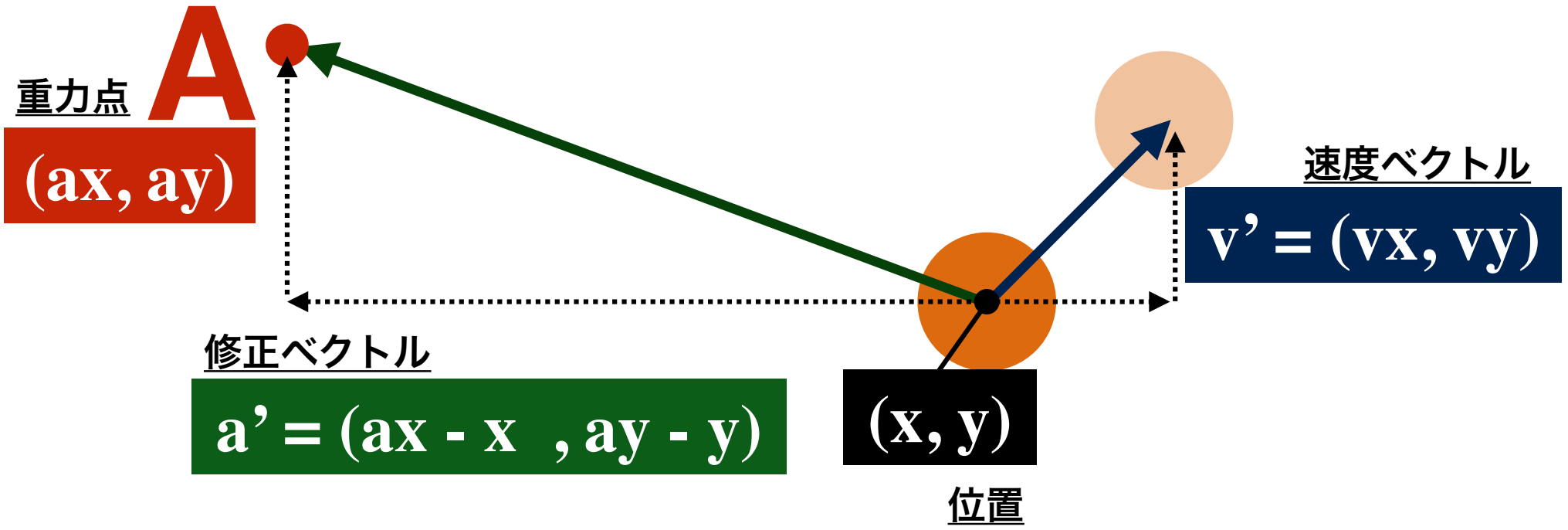
$w' = (wx, wy)$

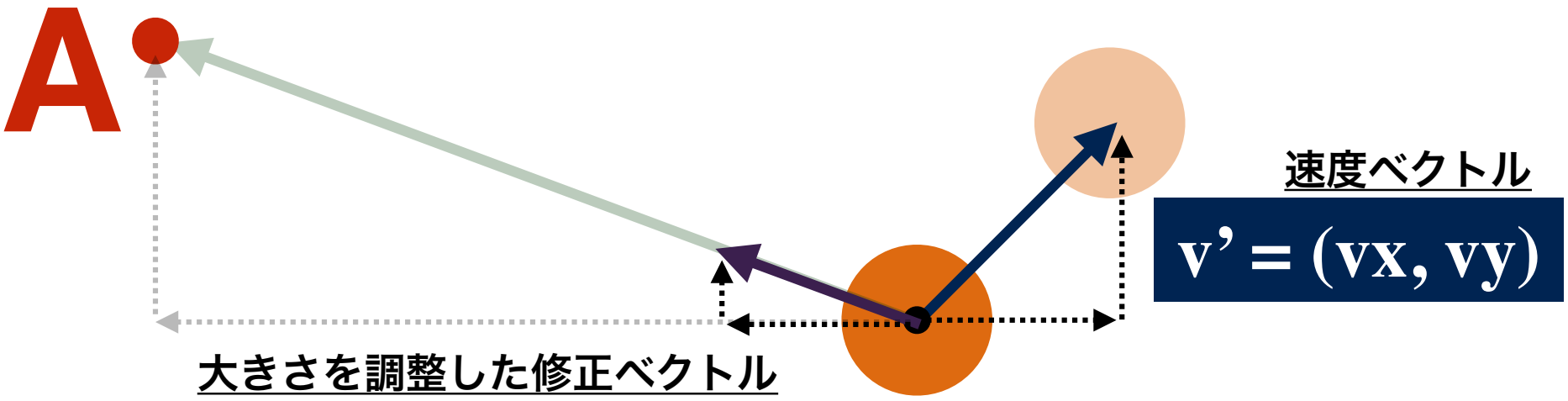


$v' + w' = (vx+wx, vy+wy)$

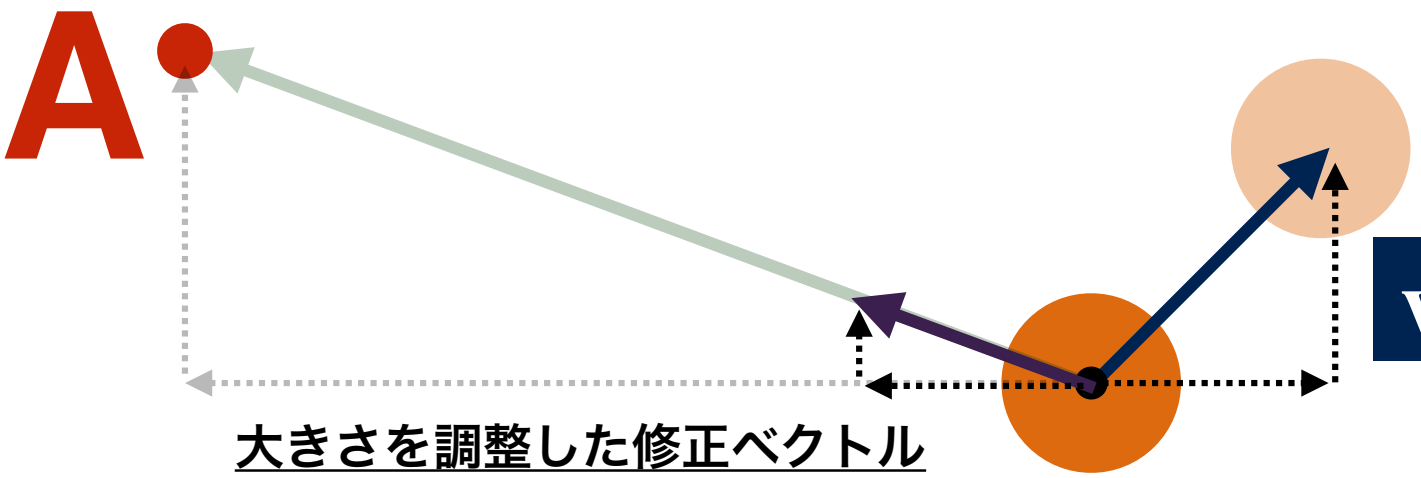


<速度 v'> を<点 A> の方向に修正する





ca' = (c1*(ax - x), c1*(ay - y))

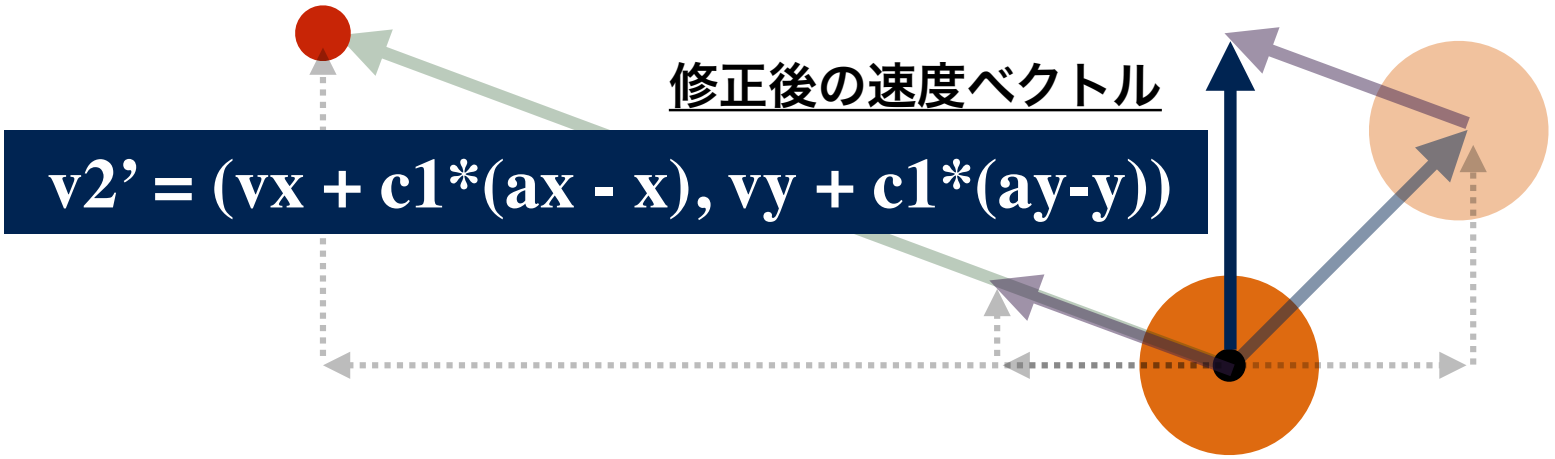


速度ベクトル

$v' = (vx, vy)$

大きさを調整した修正ベクトル

$ca' = (c1*(ax - x), c1*(ay - y))$



修正後の速度ベクトル

$v2' = (vx + c1*(ax - x), vy + c1*(ay-y))$

三次元表現でも同様

$v2' = (vx, vy, vz) + c1*(ax-x, ay-y, az-z)$

ヒント

```
public void ApplyRuleAttractor(Vector3 target){

    for(int i=0;i<pop;i++){
        Vector3 ipos = boid[i].pos;
        Vector3 ivel = boid[i].vel;
        float c = 0.1f;

        Vector3 cv =

        boid[i].SetVelocity(          );
    }
}
```

ボイド i の位置と速度

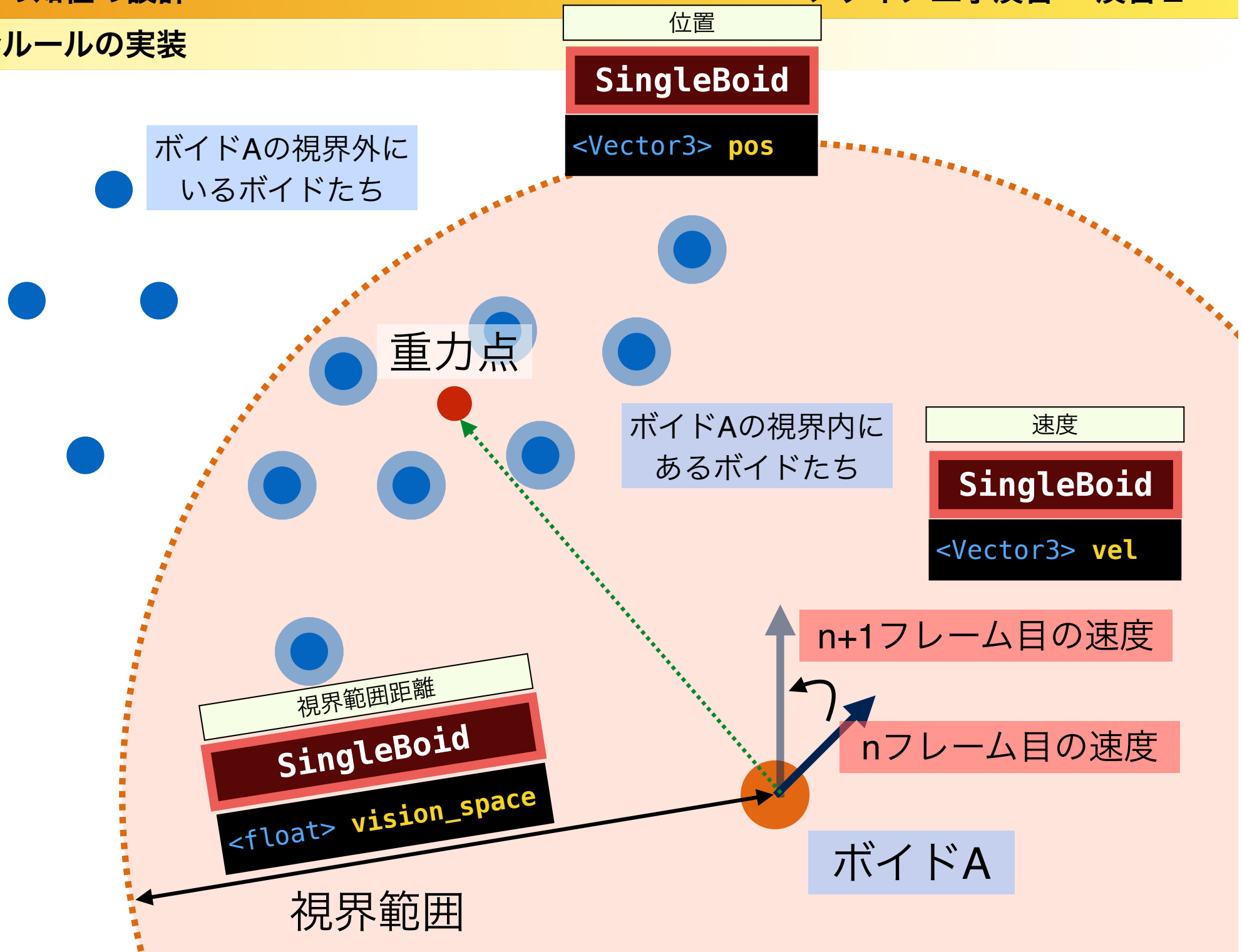
引き込みの程度

修正ベクトルの計算

i 番目のボイドの速度ベクトルを 更新する.

BoidRuleManager.cs

結合ルールの実装


位置
SingleBoid
<Vector3> pos
ボイドAの視界外にいるボイドたち
重力点
ボイドAの視界内にあるボイドたち
速度
SingleBoid
<Vector3> vel
n+1フレーム目の速度
nフレーム目の速度
視界範囲距離
SingleBoid
<float> vision_space
ボイドA
視界範囲

結合ルールの実装

1. 重力点を視界内にある全てのボイドの位置の重心として求めます.

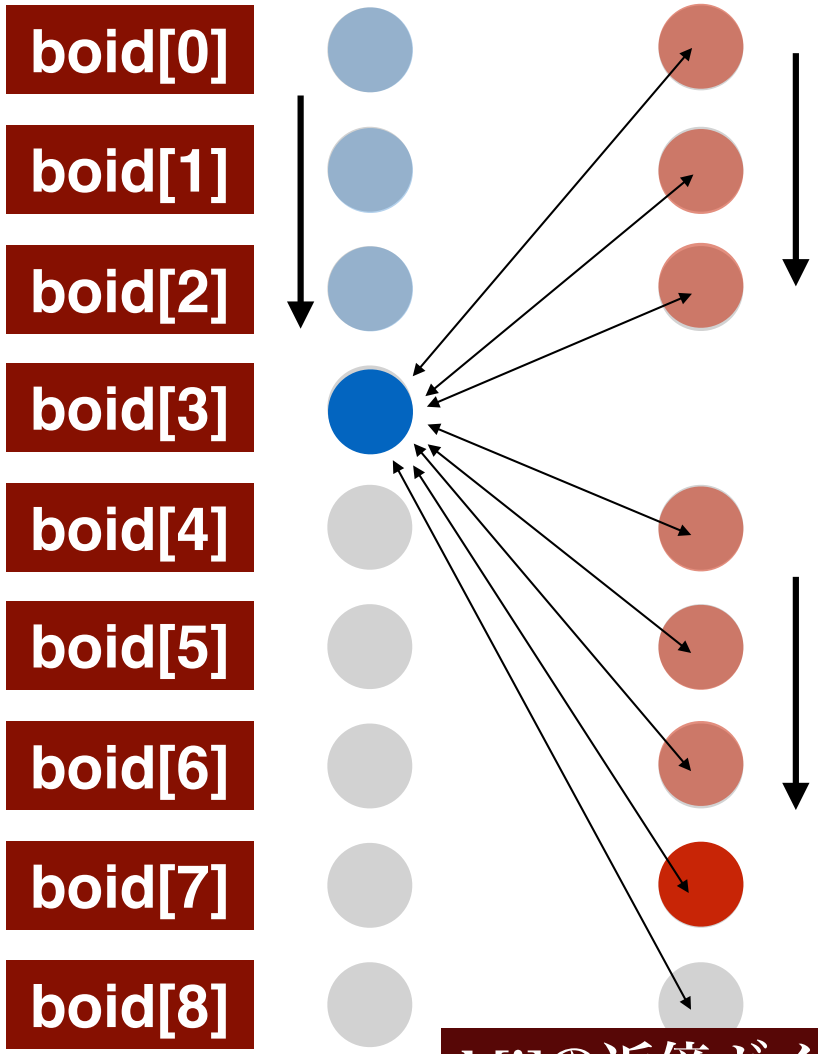
1. 重心の計算

2. 重心に引き込まれるように, ボイドの速度を新しい速度に更新する.

2. 速度の修正

2はapplyRuleAttractorで学習済みですので, ここでは1の解説をします.

b[i]の近傍ボイドの重心計算フロー



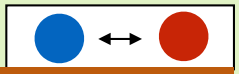

```
for(int I=0;i<pop;i++)
```

初期化

視界内にあるボイドの総数 — `int count = 0;`

視界内にあるボイドの座標の総和 — `Vector3 posTotal= Vector3.zero;`

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



b[i]とb[j]の距離が, vision_space以内であったときの処理

```
count++;
posTotal += boid[1].pos;
count++;
posTotal += boid[4].pos;
count++;
posTotal += boid[6].pos;
count++;
posTotal += boid[7].pos;
```

b[i]の近傍ボイドの重心座標：Vector3 posMean = posTotal / count

結合ルール関数内部の記述例 BoidRuleManager.cs

```
/* ルール１の実行内容*/
private void ApplyRule1()
{
    for(int i=0;i<pop;i++){
        Vector3 ipos = boid[i].pos;
        Vector3 ivel = boid[i].vel;
        float ivspace = boid[i].vision_space;

        float count = 0f;
        Vector3 posTotal = new Vector3();

        for(int j=0;j<pop;j++){
            Vector3 jpos = boid[j].pos;
            float dis =

            if(                                ){
                posTotal +=
                count++;
            }
        }
        if(count > 0){
            Vector3 target =
            Vector3 cv =
            boid[i].SetVelocity
        }
    }
}
```

全てのボイド（i=0…pop−1）について，
1） 視界内にあるボイドの重心位置 target を求め，
2） 速度を重心に引き込まれるように更新します．

i 番目のボイドの位置・速度をipos, ivelとする.

i 番目のボイドの視界範囲をivspaceとする

j番目のボイドの位置をjposとする.

近傍判定

ボイド i とボイド j の距離を disとする.

ボイド i の近傍ボイドの位置の平均値をtargetとする.

1. 重心の計算

係数「c1」を用いて修正ベクトルの計算

ボイド i の速度を更新.

2. 速度の修正