

演習 2 : 集合の知性を設計する

(05) 05/19 (06) 05/26

A | Unity環境の整備・簡単なルール設計

(07) 06/02 (08) 06/09

B | ボイドルール 1・2・3 の実装

(09) 06/16

C | 課題 1 : 集合知の解析

(10) 06/23

D 1 | SIR (感染モデル)

(11) 06/30 (12) 07/07 (13) 07/14

D 2 | 課題 2 : マイルール・感染ルール・視点操作

(14-15) 07/21

D 3 | 発表 (One-Minute Movie)

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

```
if (Input.GetMouseButton(0))
{
    rule.ApplyRuleAttractor(new Vector3(0f, 100f, 0f));
}
```

Update()

マウスボタンを押すと, ApplyRuleAttractor が作動

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

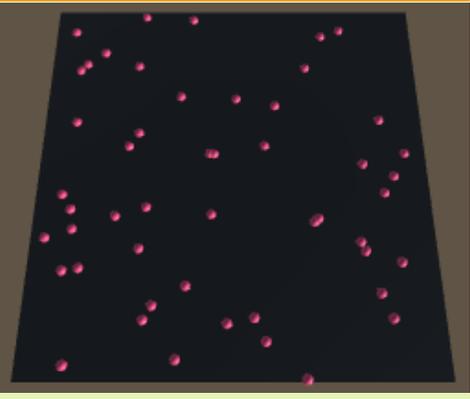
BoidRuleManager.cs

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

rule1 が true のときに, ApplyRule1 を実行

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

```
# [x] Boid Manager (Script)
Script # BoidManager
Vision_space 35
Neighbor_space 10
Pop 50
```

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

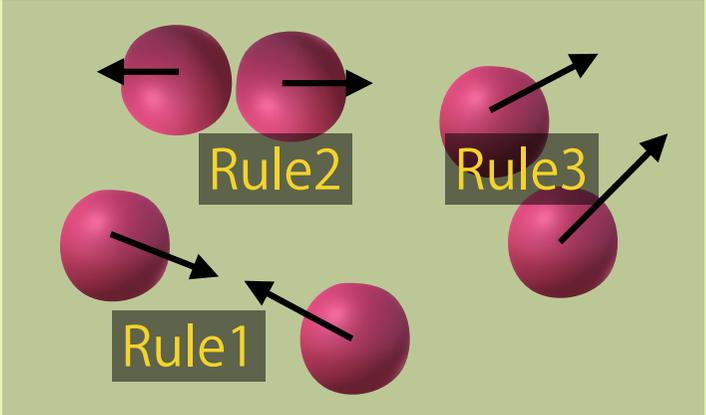
BoidRule Manager

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）

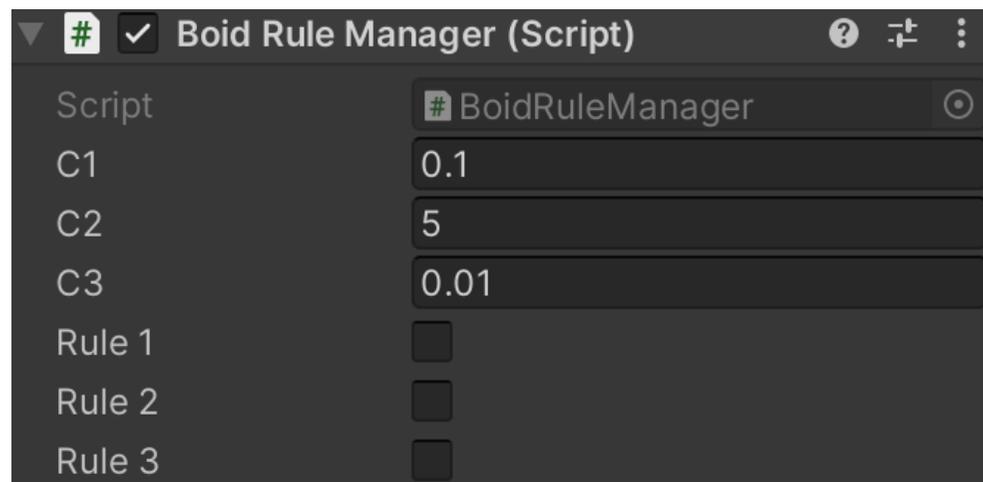
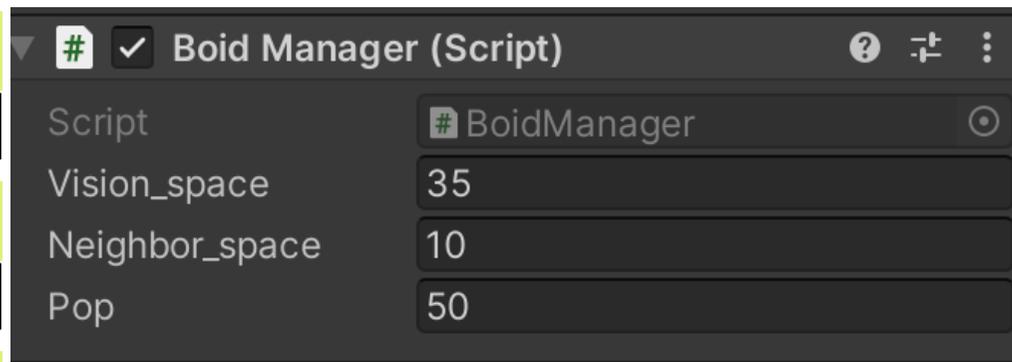
BoidRuleManager

<bool> rule1, rule2, rule3

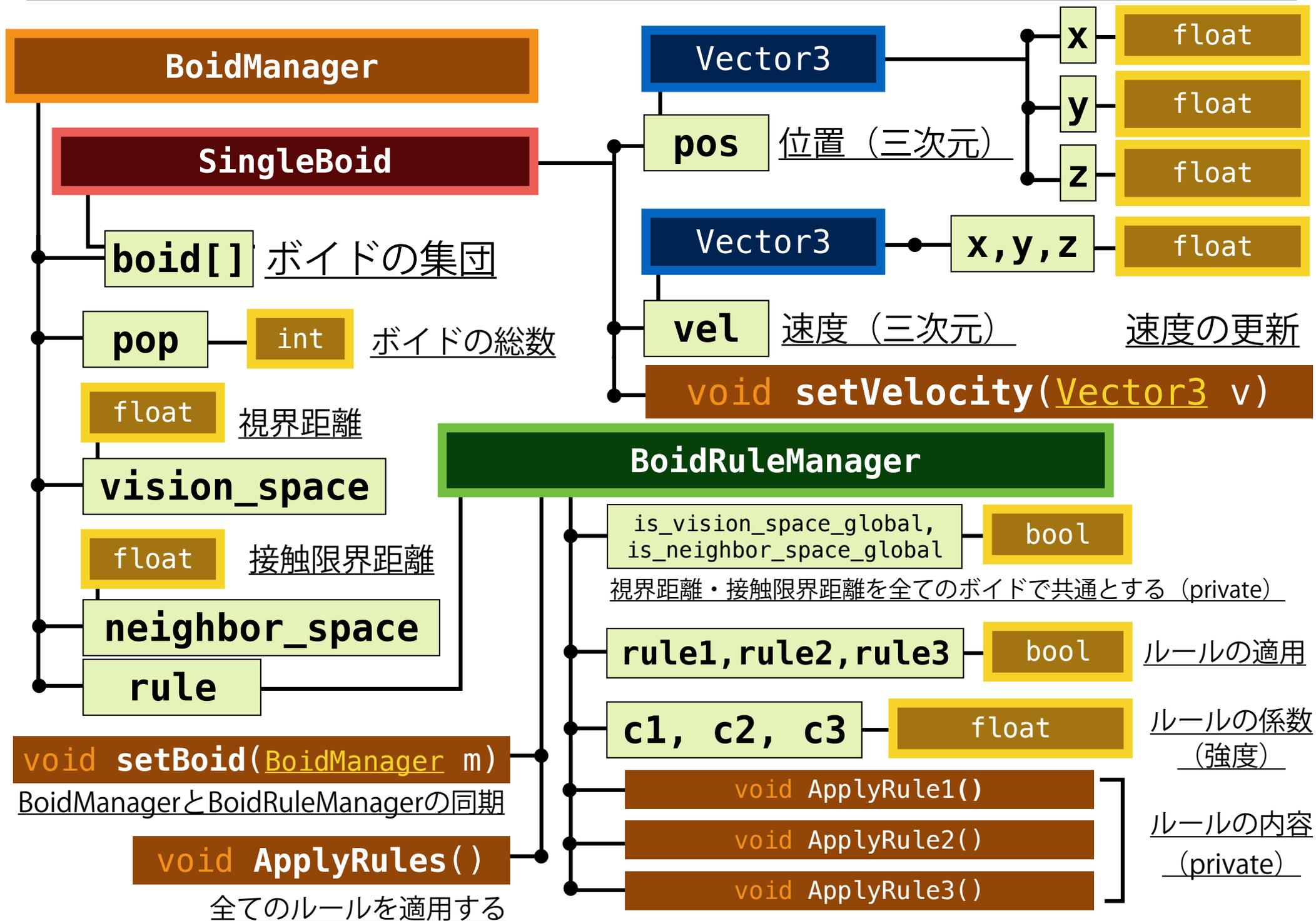
ルール1・2・3の適用の有無

<float> c1, c2, c3

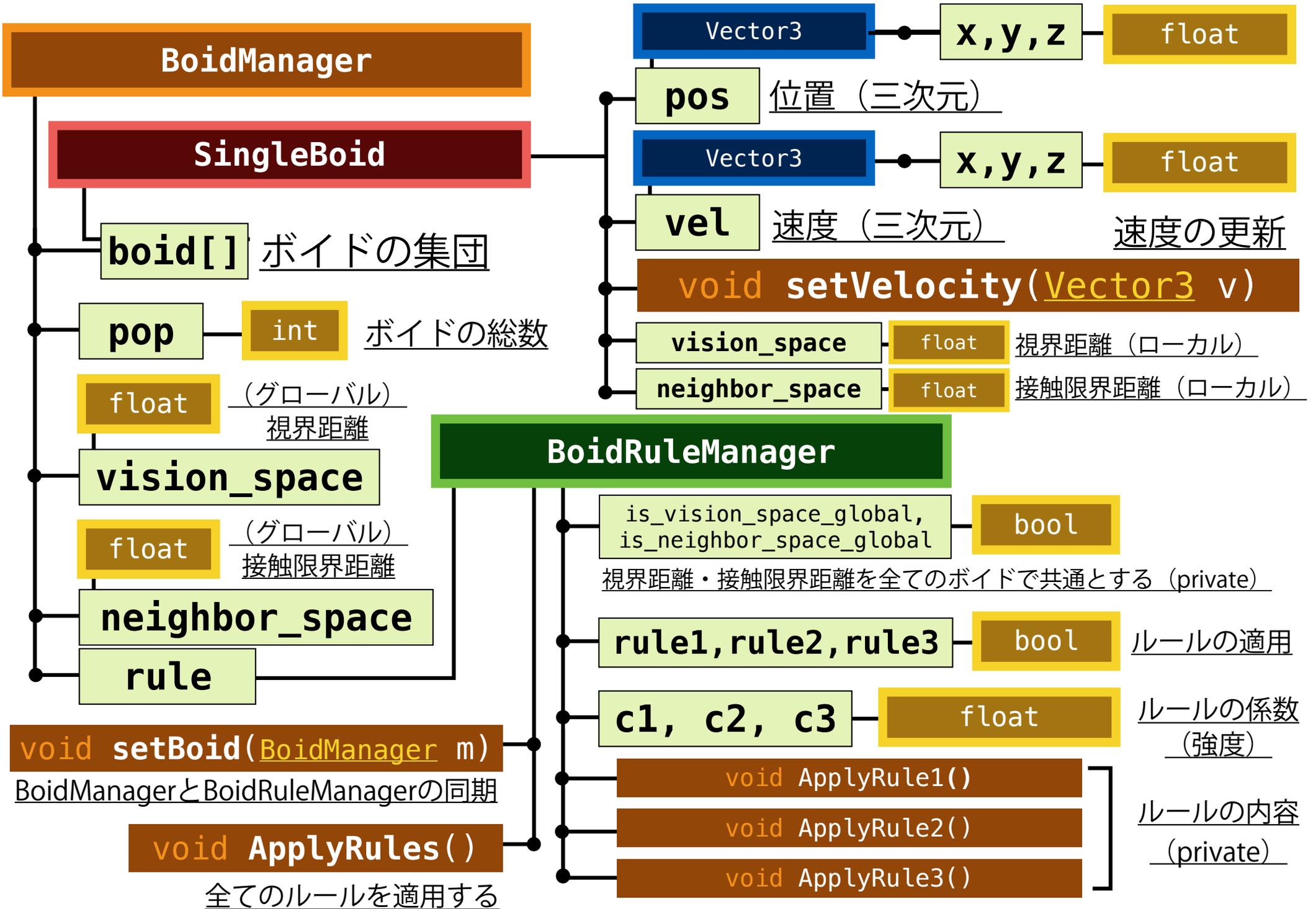
各ルールの影響度（係数）



BoidManager / SingleBoid / BoidRuleManagerの関係 (復習)

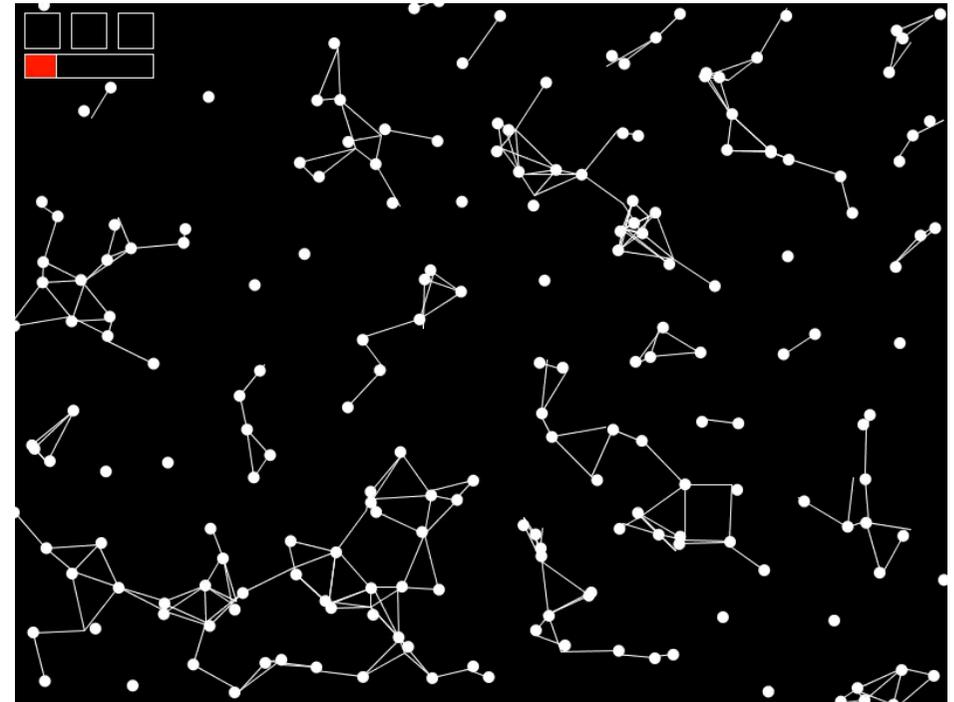
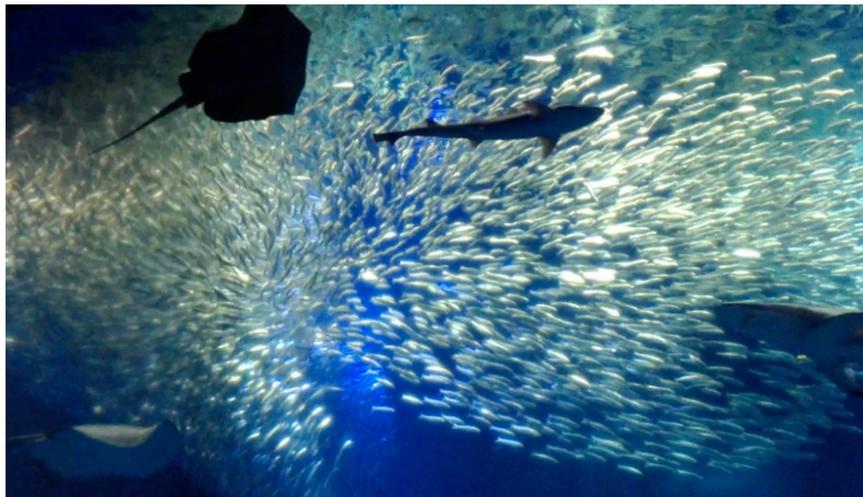


BoidManager / SingleBoid / BoidRuleManagerの関係 (修正版)



集団の同調

Cooperativeness / 群知能



boids (ボイド; bird-like droids)

親近行動

整列行動

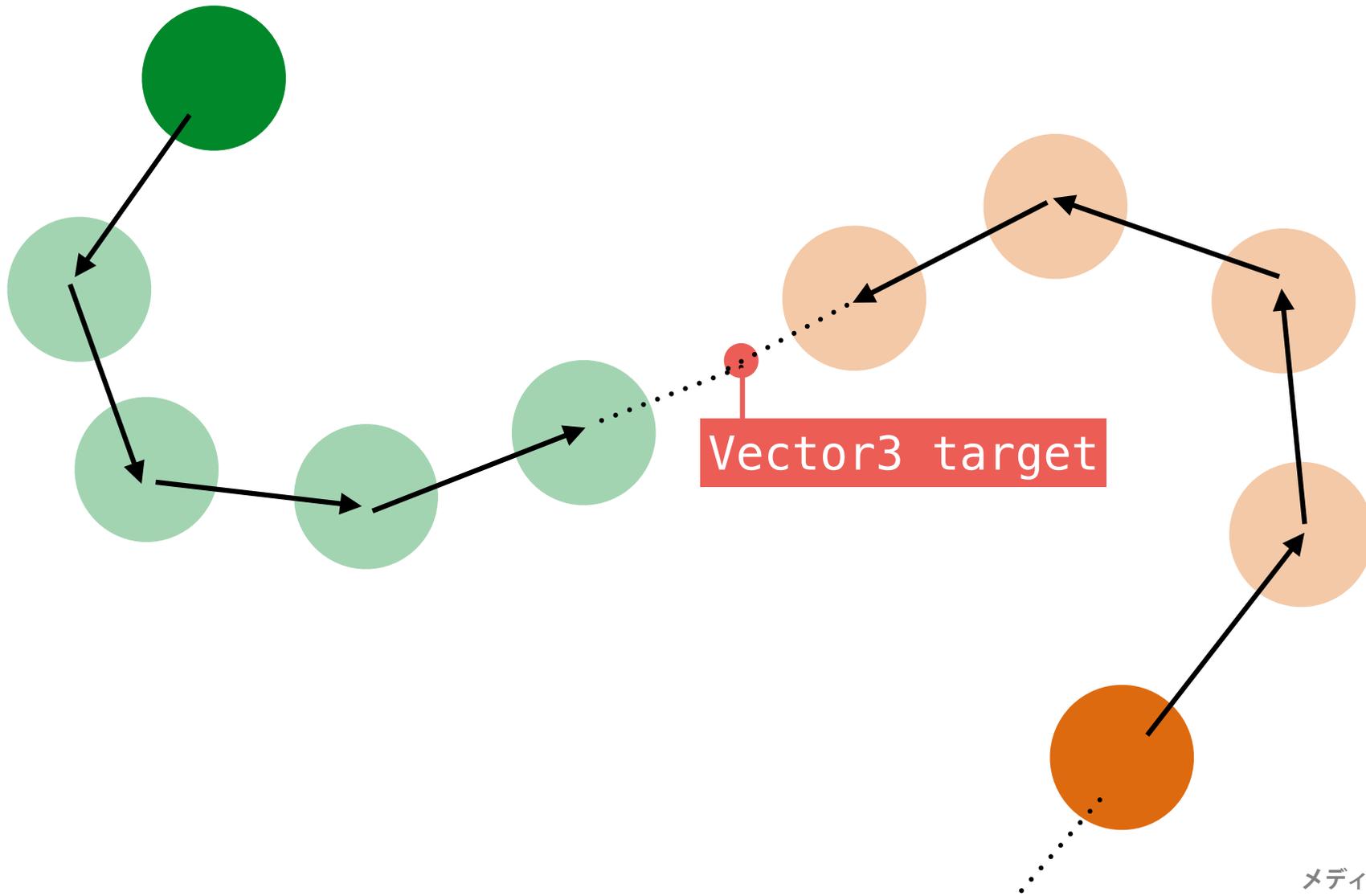
衝突回避

準備

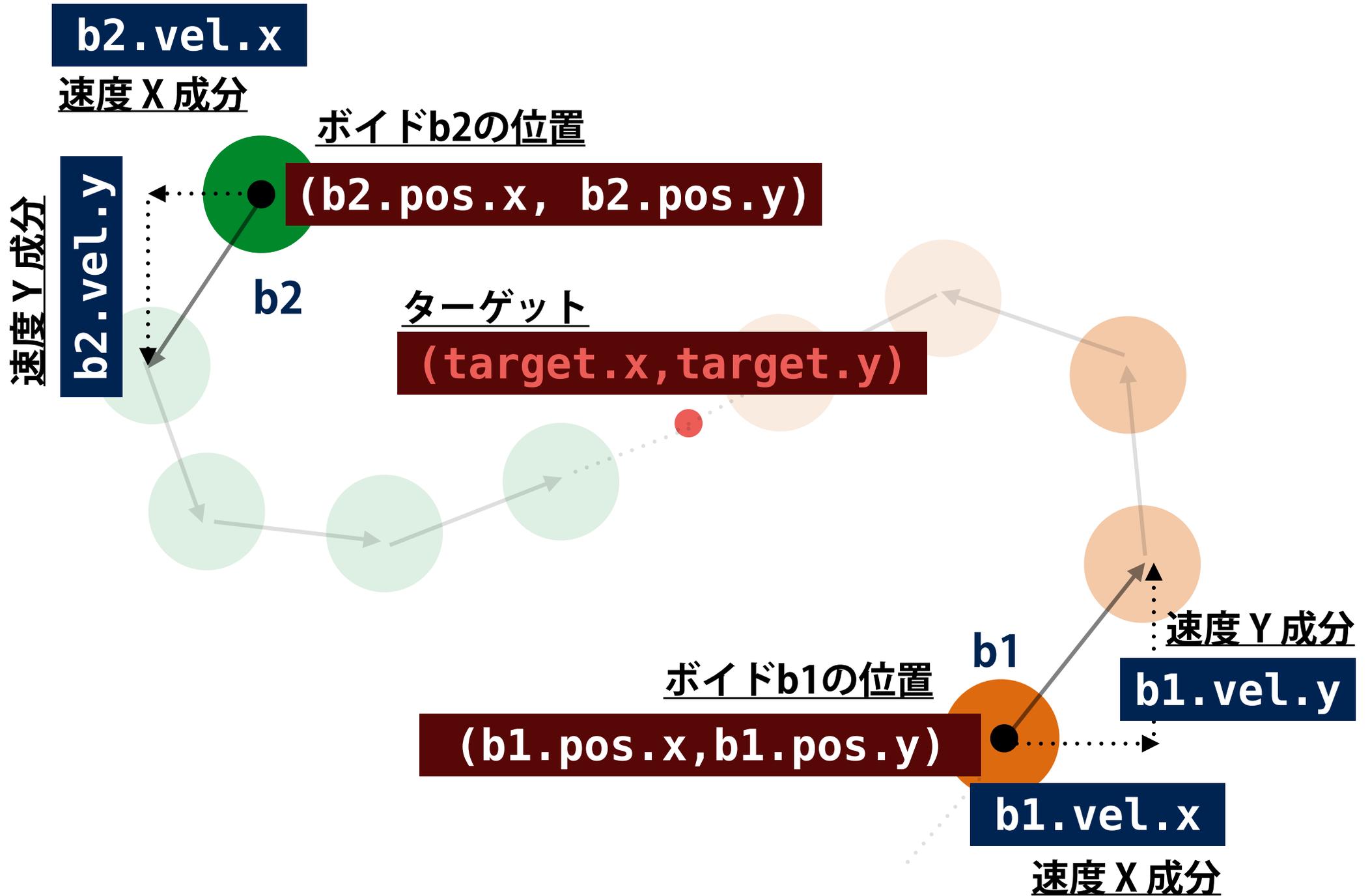
BoidRuleManager.cs に、`void ApplyRuleAttractor(Vector3 target)`

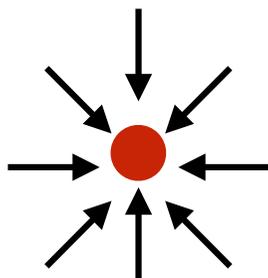
を定義し、

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.)



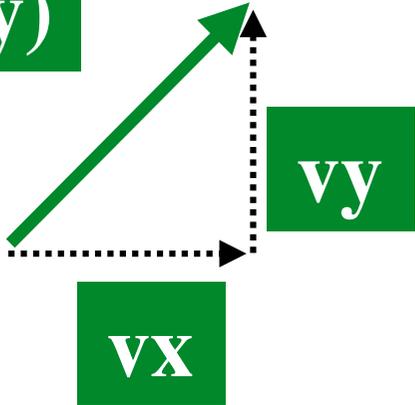


速度を, 指定された点に (徐々に) 向け
るためのベクトル計算

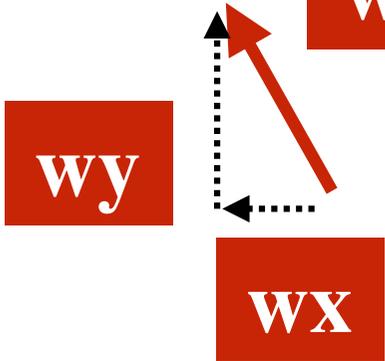
位置による引き込み (引力)

ベクトルの加算

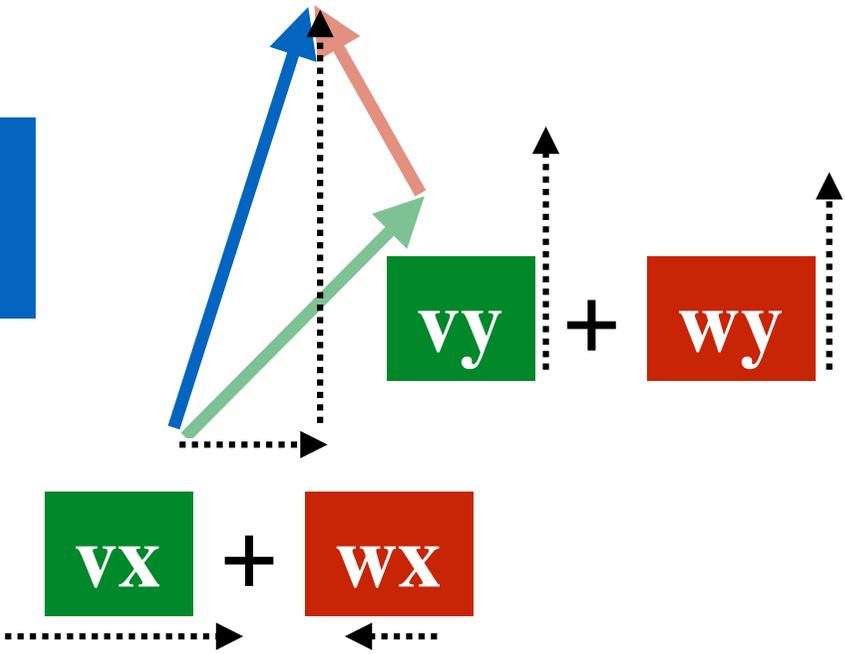
$$\mathbf{v}' = (v_x, v_y)$$



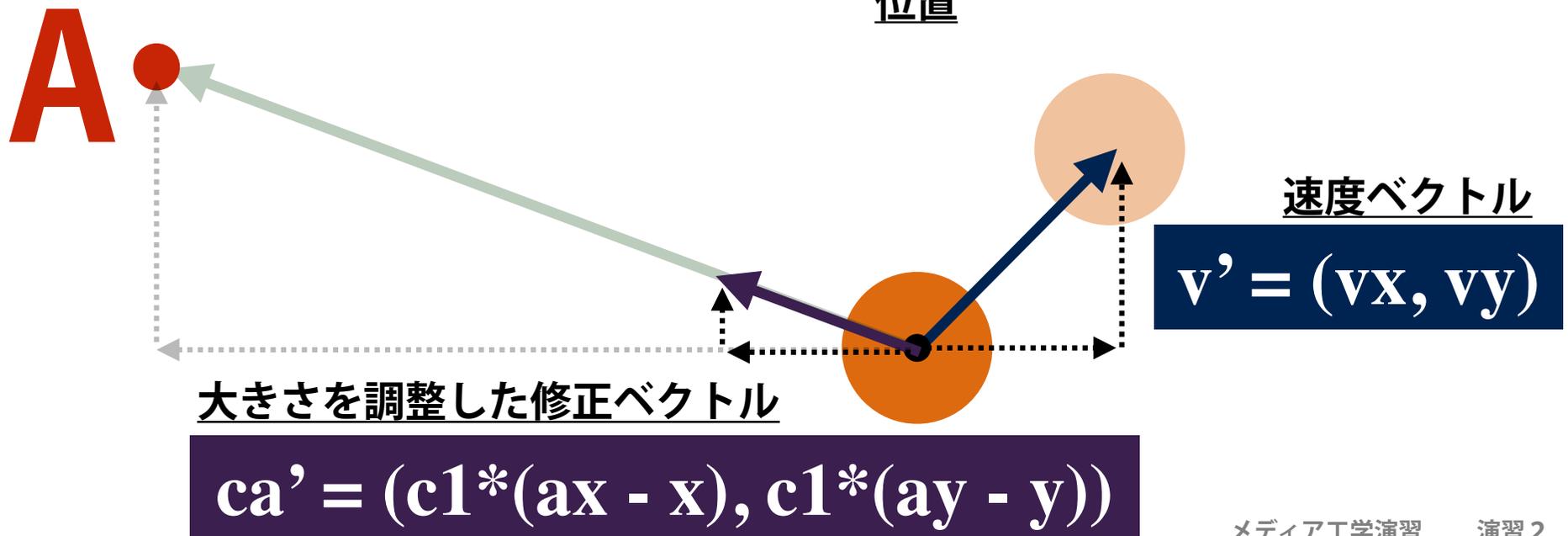
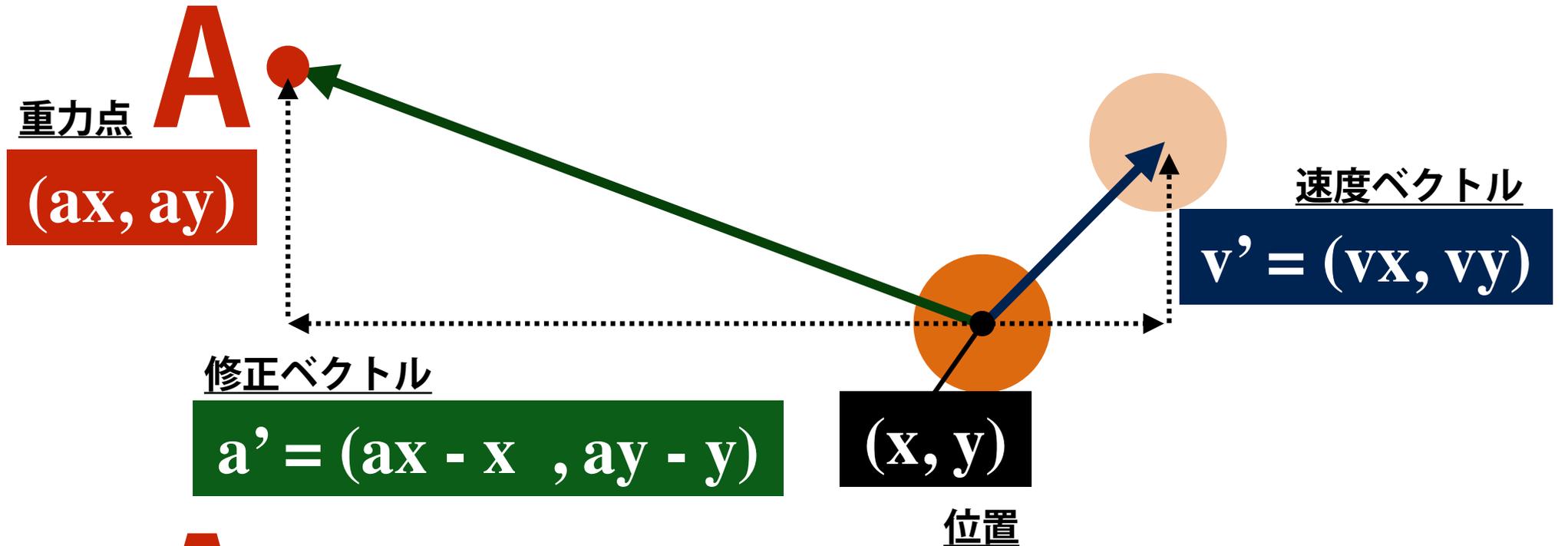
$$\mathbf{w}' = (w_x, w_y)$$



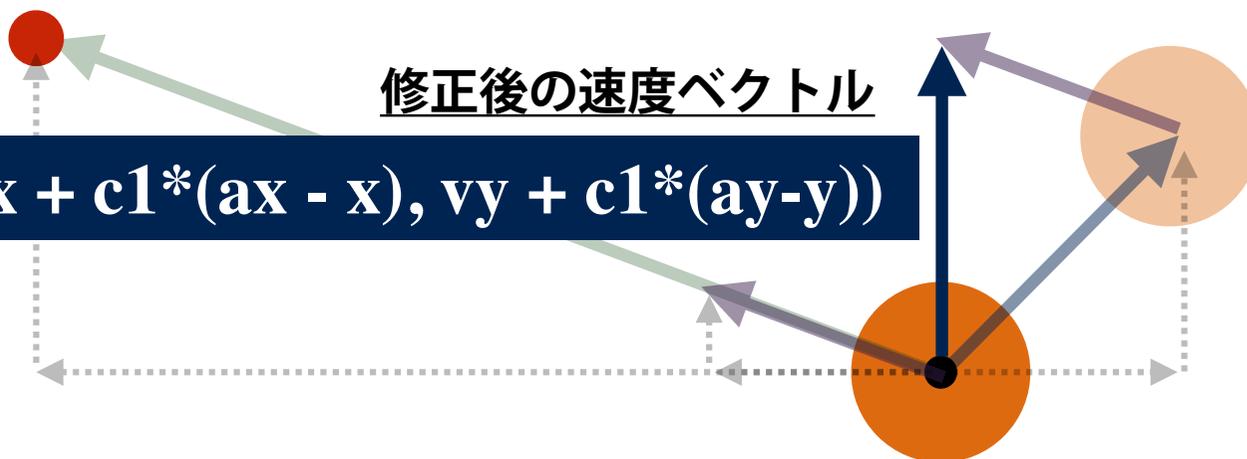
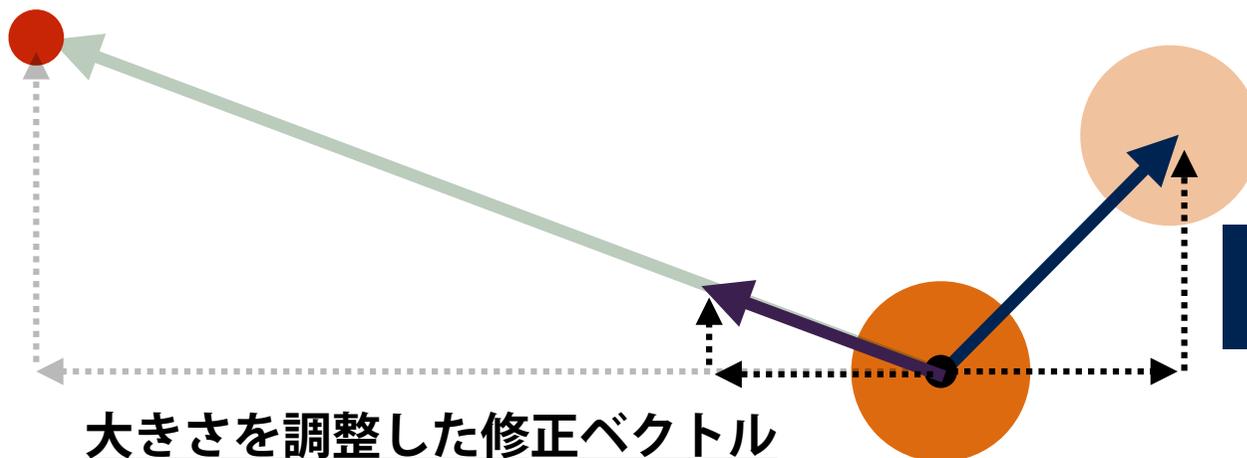
$$\mathbf{v}' + \mathbf{w}' = (v_x + w_x, v_y + w_y)$$



<速度 v' > を<点 A> の方向に修正する



A



三次元表現でも同様

$$v2' = (v_x, v_y, v_z) + c1 * (ax - x, ay - y, az - z)$$

ヒント

```
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 = new Vector3 ();  
  
        cv.x  
        cv.y  
        cv.z  
  
        boid [i].setVelocity ( );  
    }  
}
```

ボイド i の位置
と速度

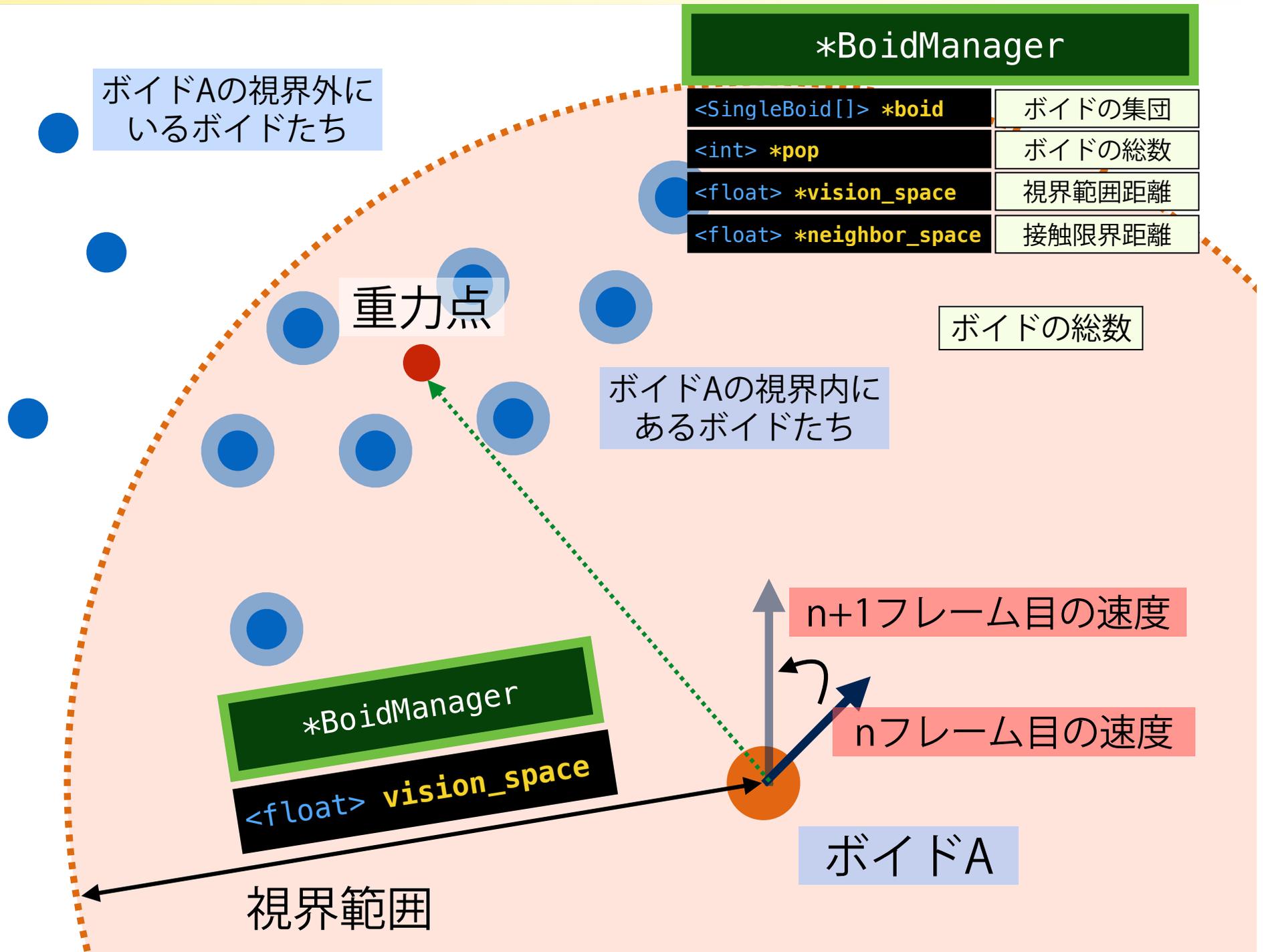
引き込みの程度

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。

BoidRule.cs

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.

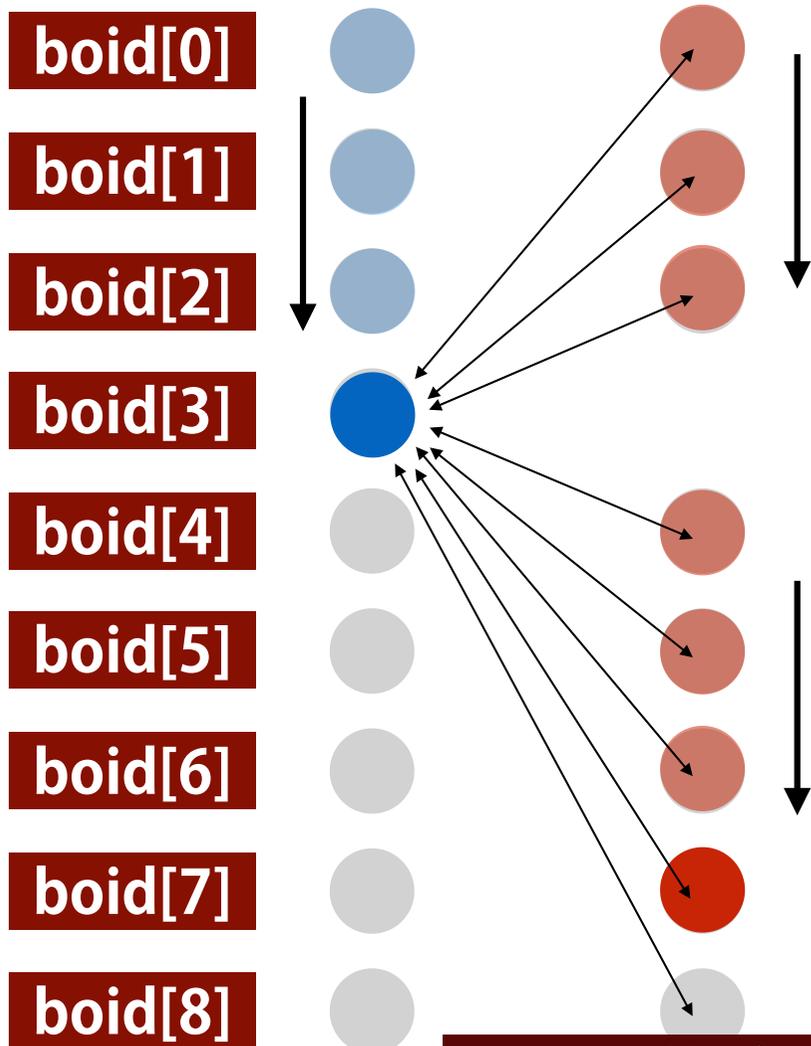
1. 重心の計算

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： $\text{Vector3 posMean} = \text{posTotal} / \text{count}$

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.

```
void ApplyRule1()
{
    for (int i = 0; i < pop; i++) {

        Vector3 ipos = boid [i].pos;
        Vector3 ivel = boid [i].vel;

        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].c (

            }
        }
    }
}
```

i 番目のボイドの位置・
速度を ipos, ivel 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