

IDD Media lab. 2015

Drawing Machine - openFrameworksで生成的な形を描く

多摩美術大学情報デザイン学科メディア芸術コース

2015年4月28日

田所淳

生成的な形態を作る

- ▶ Drawing Machineのヒントとして…
- ▶ 生成的な形を描いてみる
 - ▶ 生成的 (Generative) - 数学的、機械的、あるいは無作為な過程によって自律的に生成されること

生成的な表現を利用した作品例

- ▶ Gallery of Computation, Jared Tarbell
- ▶ <http://www.complexification.net/gallery/>



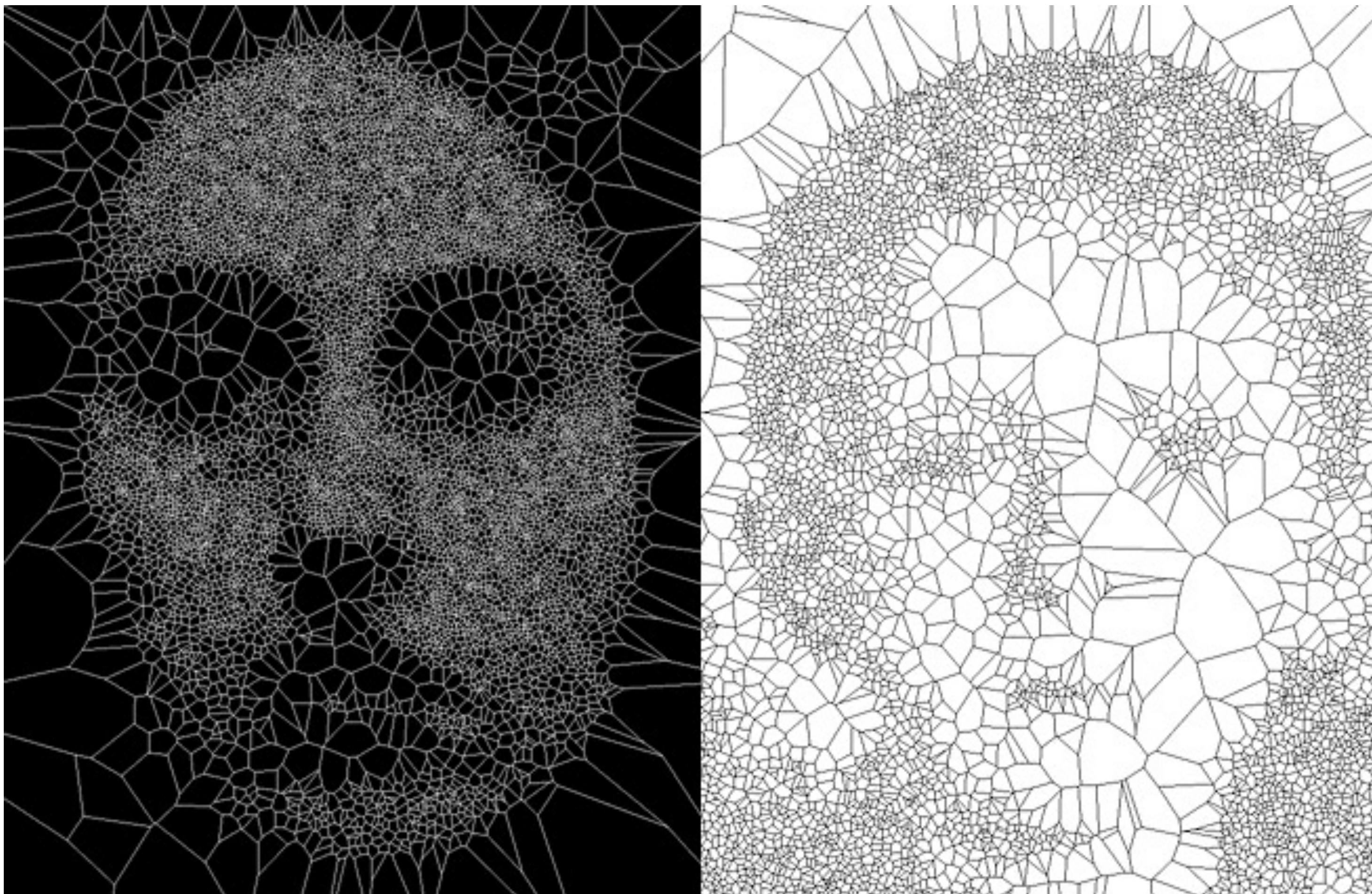
生成的な表現を利用した作品例

- ▶ Ambushes, Eno Henze
- ▶ <http://www.enohenze.de/>



生成的な表現を利用した作品例

- ▶ Segmentation and Symptom, Golan Levin
- ▶ <http://www.flong.com/projects/zoo/>



生成的な表現を利用した作品例

- ▶ Superformula, David Dessens
- ▶ <http://www.sanchtv.com/>



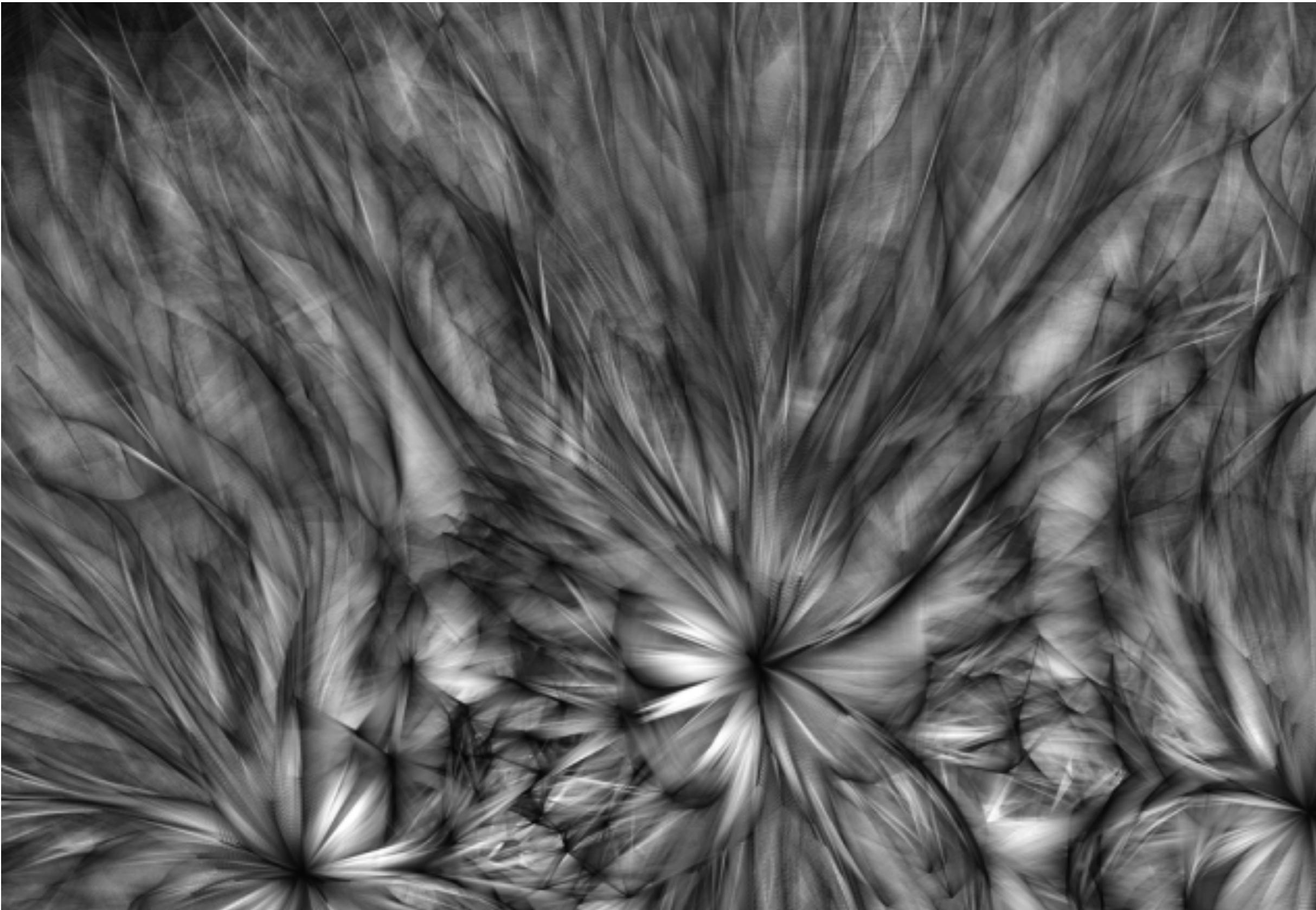
生成的な表現を利用した作品例

- ▶ Arcs 21, Lia
- ▶ <http://liaworks.com/projects/arcs21/>
- ▶ <http://itunes.apple.com/us/app/arcs-21/id338741179?mt=8>



生成的な表現を利用した作品例

- ▶ Process 1 - Process 18, Casey Reas
- ▶ <http://reas.com/texts/processdrawing-ad.html>
- ▶ <http://vimeo.com/22955812>



生成的な表現を利用した作品例

- ▶ Casey Reas “Process” のルール http://reas.com/compendium_text/

Forms

F1: Circle

F2: Line

Behaviors

B1: Move in a straight line

B2: Constrain to surface

B3: Change direction while touching another Element

B4: Move away from an overlapping Element

B5: Enter from the opposite edge after moving off the surface

B6: Orient toward the direction of an Element that is touching

B7: Deviate from the current direction

生成的な表現を利用した作品例

- ▶ Casey Reas “Process” のルール http://reas.com/compendium_text/

Elements

E1: F1 + B1 + B2 + B3 + B4

E2: F1 + B1 + B5

E3: F2 + B1 + B3 + B5

E4: F1 + B1 + B2 + B3

E5: F2 + B1 + B5 + B6 + B7

生成的な表現をとりあつかった書籍

- ▶ Generative Gestaltung
- ▶ Hartmut Bohnacker (編集), Benedikt Gross (編集), Julia Laub (編集), Claudio Lazzeroni (編集)
- ▶ <http://www.amazon.co.jp/dp/3874397599/>



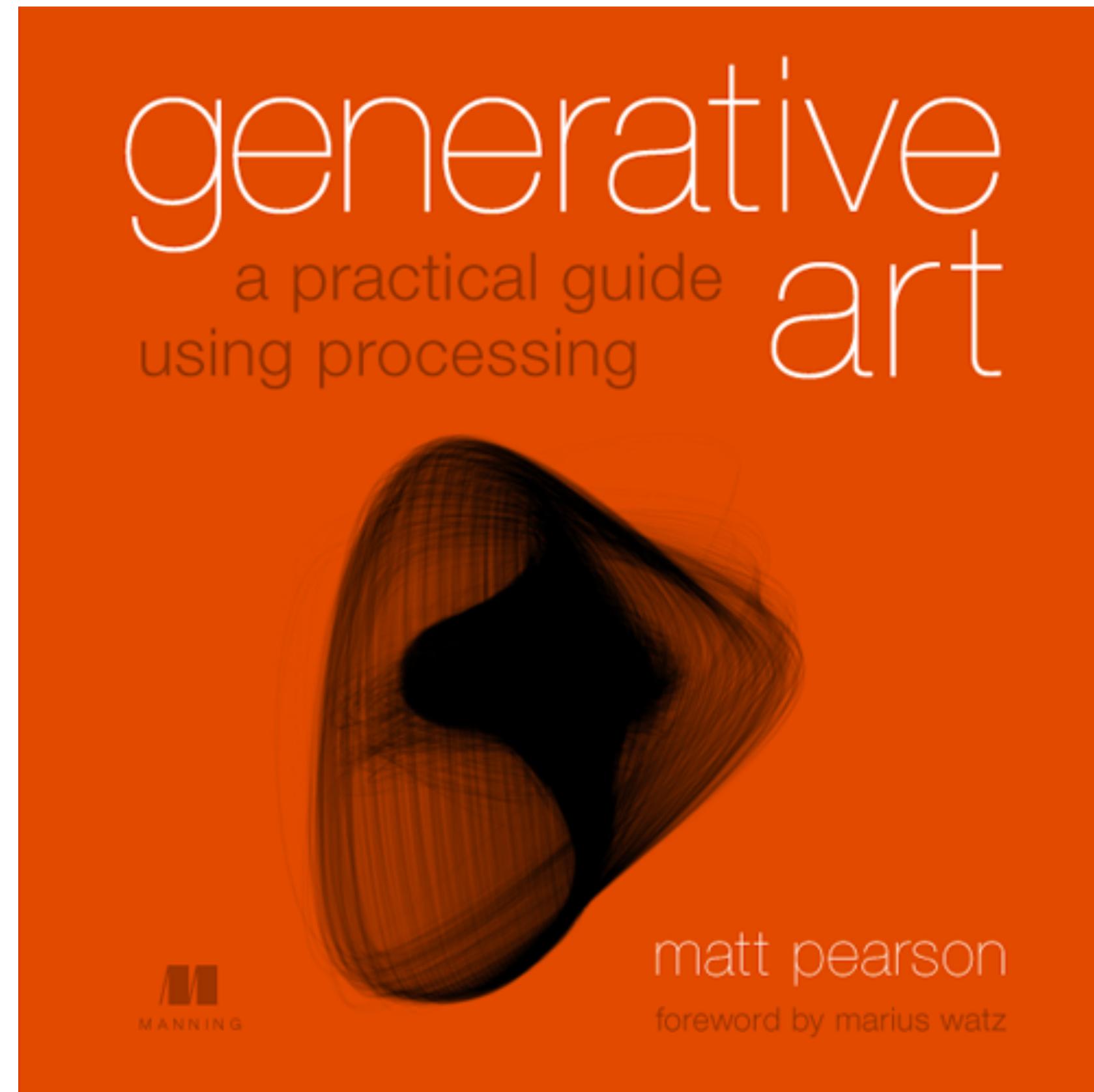
生成的な表現をとりあつかった書籍

- ▶ FORM+CODE -デザイン／アート／建築における、かたちとコード
- ▶ 久保田 晃弘 (監修), 吉村 マサテル (翻訳)
- ▶ <http://www.amazon.co.jp/dp/4861007518/>



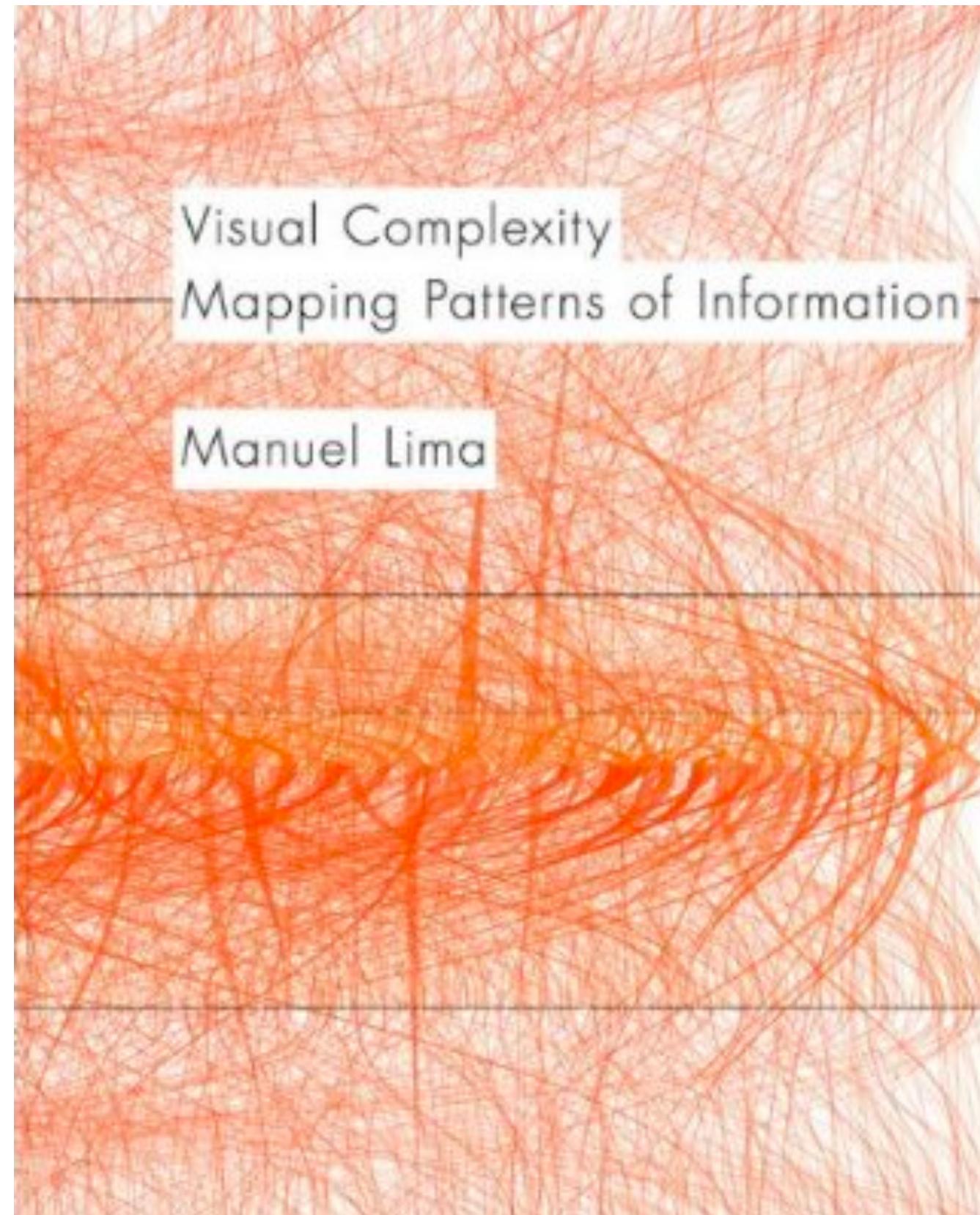
生成的な表現をとりあつかった書籍

- ▶ Generative Art: A Practical Guide Using Processing
- ▶ Matt Pearson (著)
- ▶ <http://www.amazon.co.jp/dp/1935182625/>



生成的な表現をとりあつかった書籍

- ▶ Visual Complexity: Mapping Patterns of Information
- ▶ Manuel Lima (著)
- ▶ <http://www.amazon.co.jp/dp/1568989369/>



生成的な表現をとりあつかった書籍

- ▶ Written Images
- ▶ <http://writtenimages.net/>



openFrameworksで
生成的な表現に挑戦!!

ランダムウォークする点を描いてみる

openFrameworksによる生成的表現

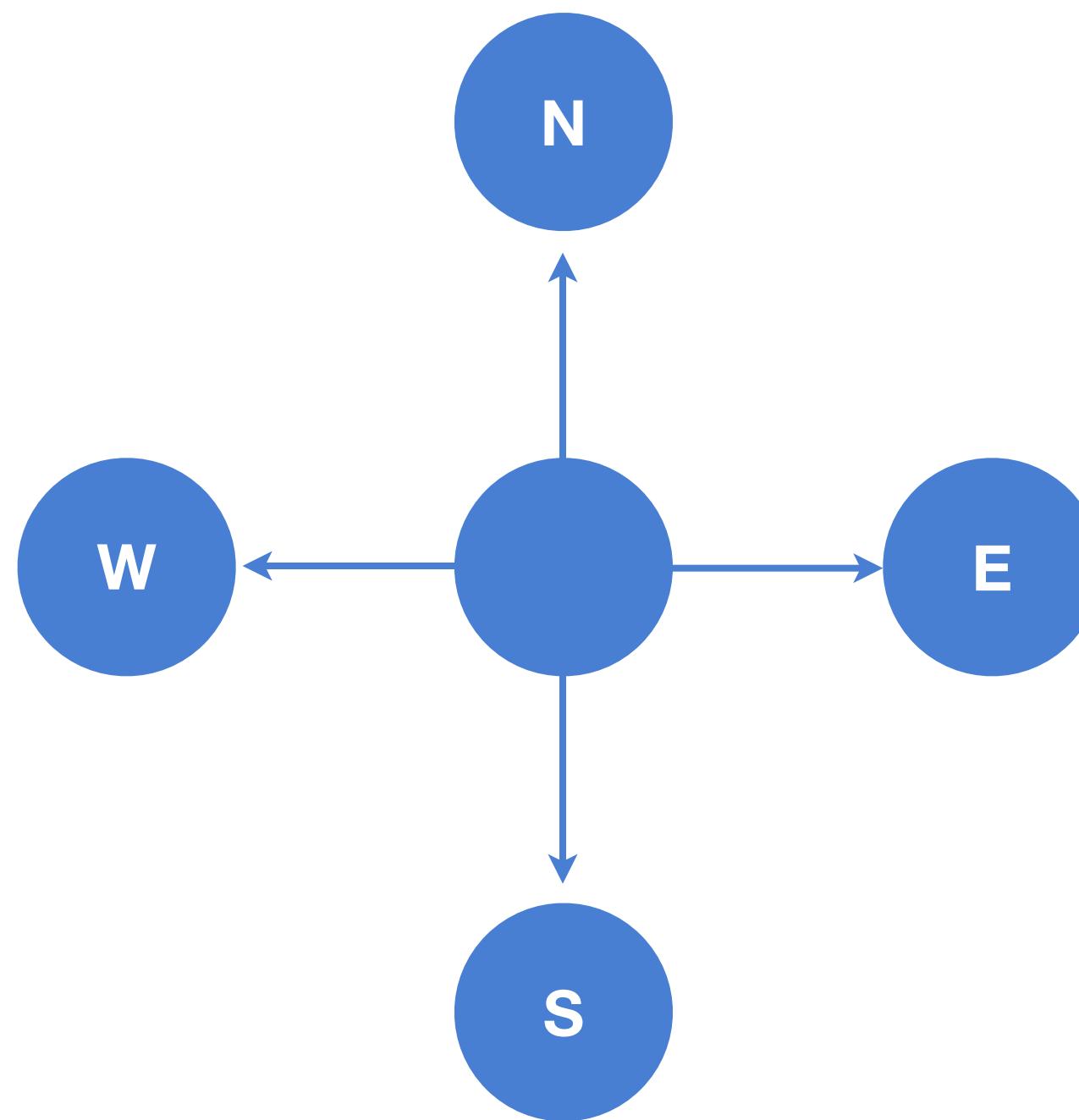
- ▶ openFrameworksによる生成的表現
- ▶ 手元にPCのある方は、 openFrameworksをダウンロード
- ▶ <http://www.openframeworks.cc/>

- ▶ 開発環境
 - ▶ OSX → XCode
 - ▶ Windows, Linux → Code::Blocks

- ▶ 今回は、 OSX環境で説明していきます。

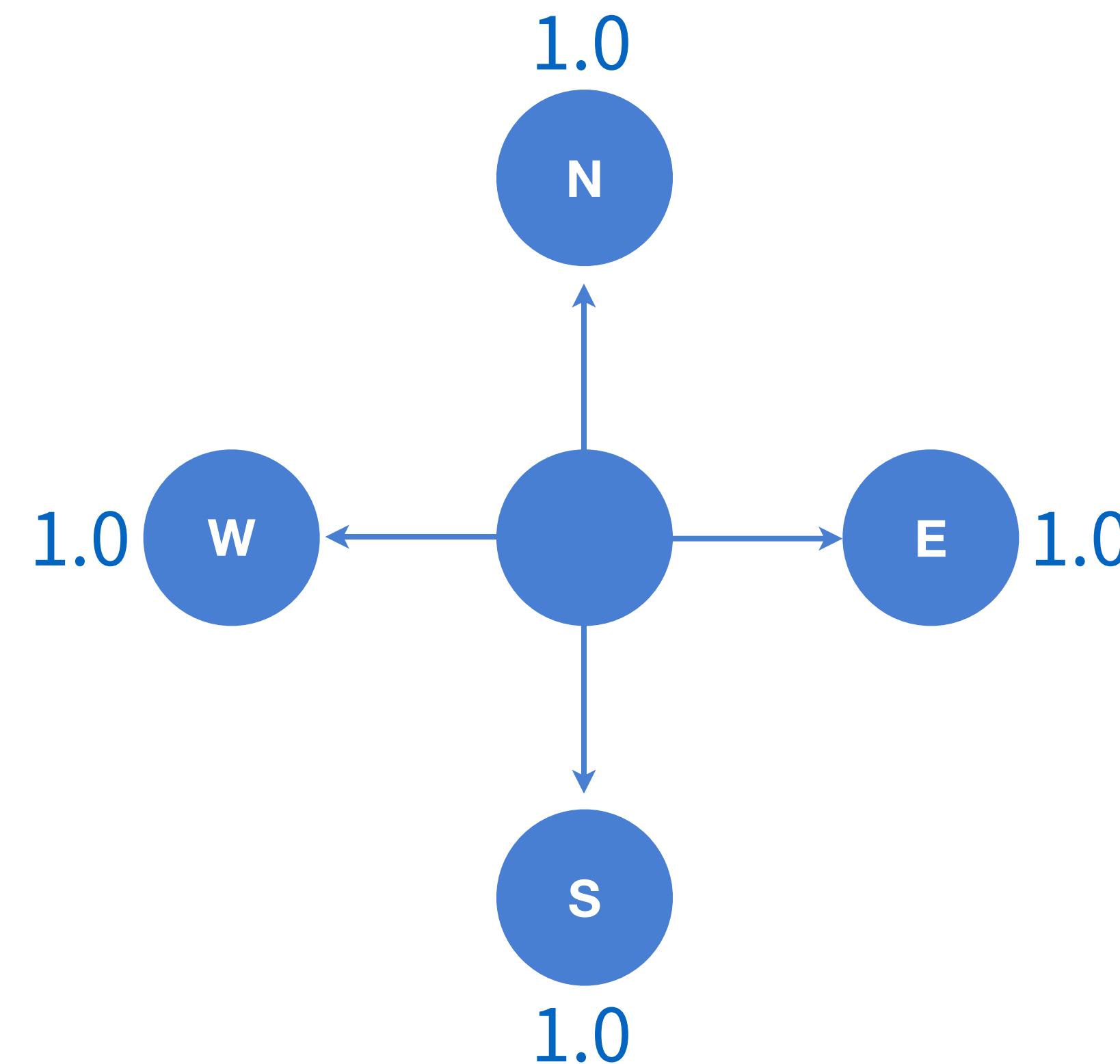
ランダムウォーク

- ▶ シンプルかつ予測が困難な動きをつくってみたい
- ▶ 画面上の1ピクセルが移動する
- ▶ 移動の選択肢は、上下左右の4つ



ランダムウォーク

- ▶ 上下左右(N, S, W, E)それぞれ、同じ確率で移動する
- ▶ 一度移動したら、またその場所から移動をくりかえす
- ▶ ランダムウォーク

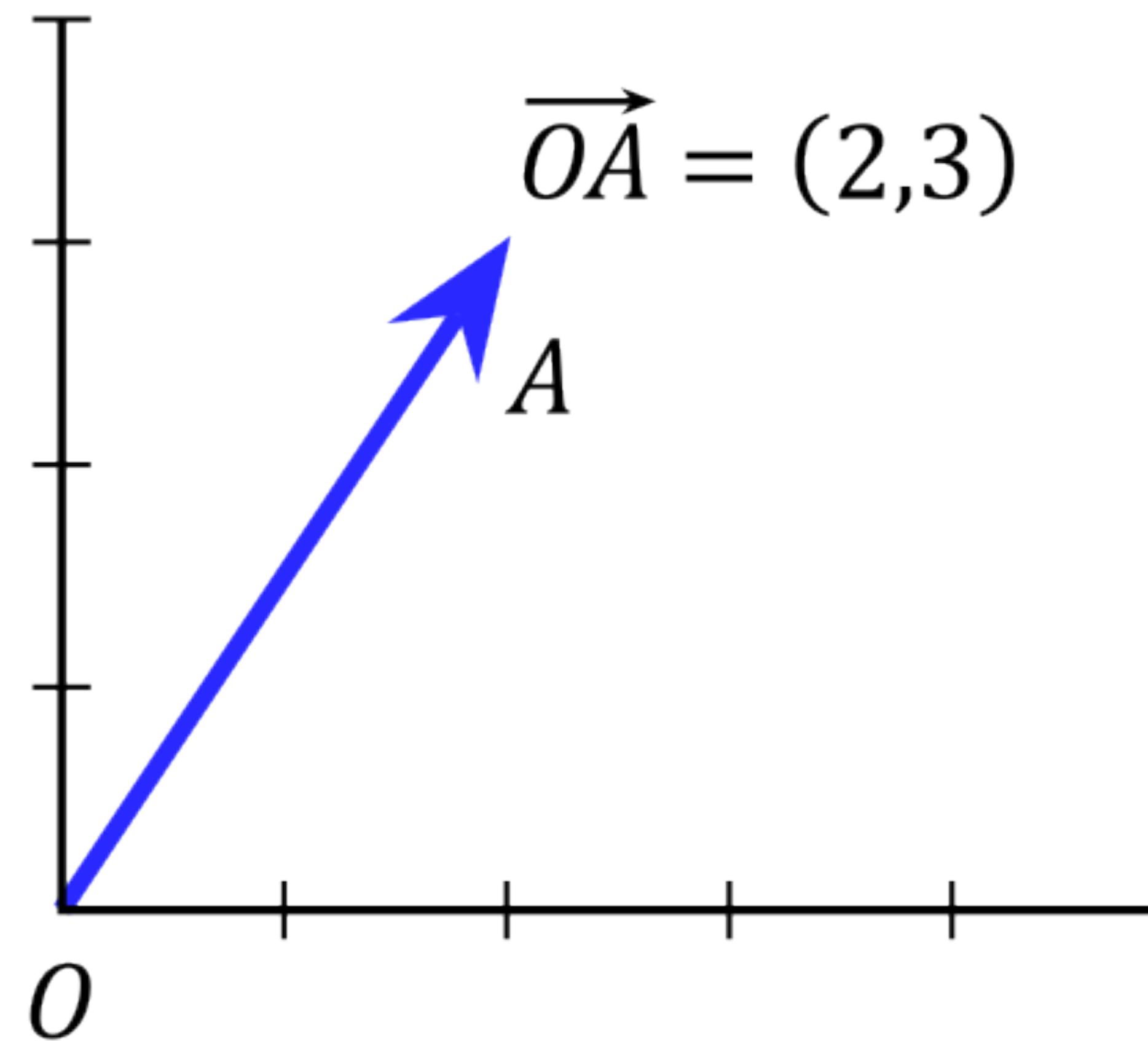


ランダムウォーク

- ▶ ランダムウォークをする点を定義する
- ▶ まずは点の位置を記憶する仕組み

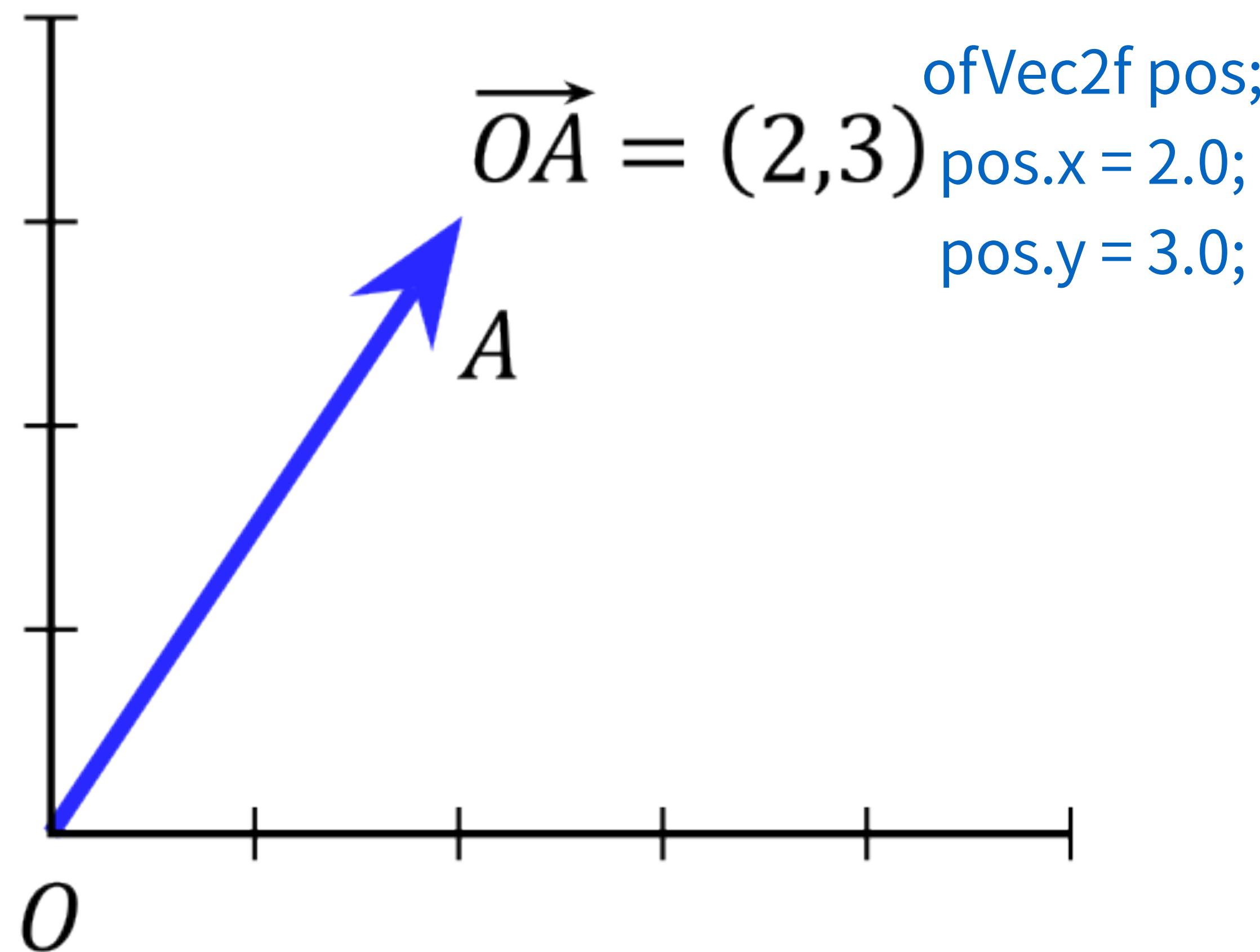
ofVec2fについて

- ▶ ofVec2fとは?
- ▶ ベクトルを表現するための、oFのクラス（型みたいな感じ）



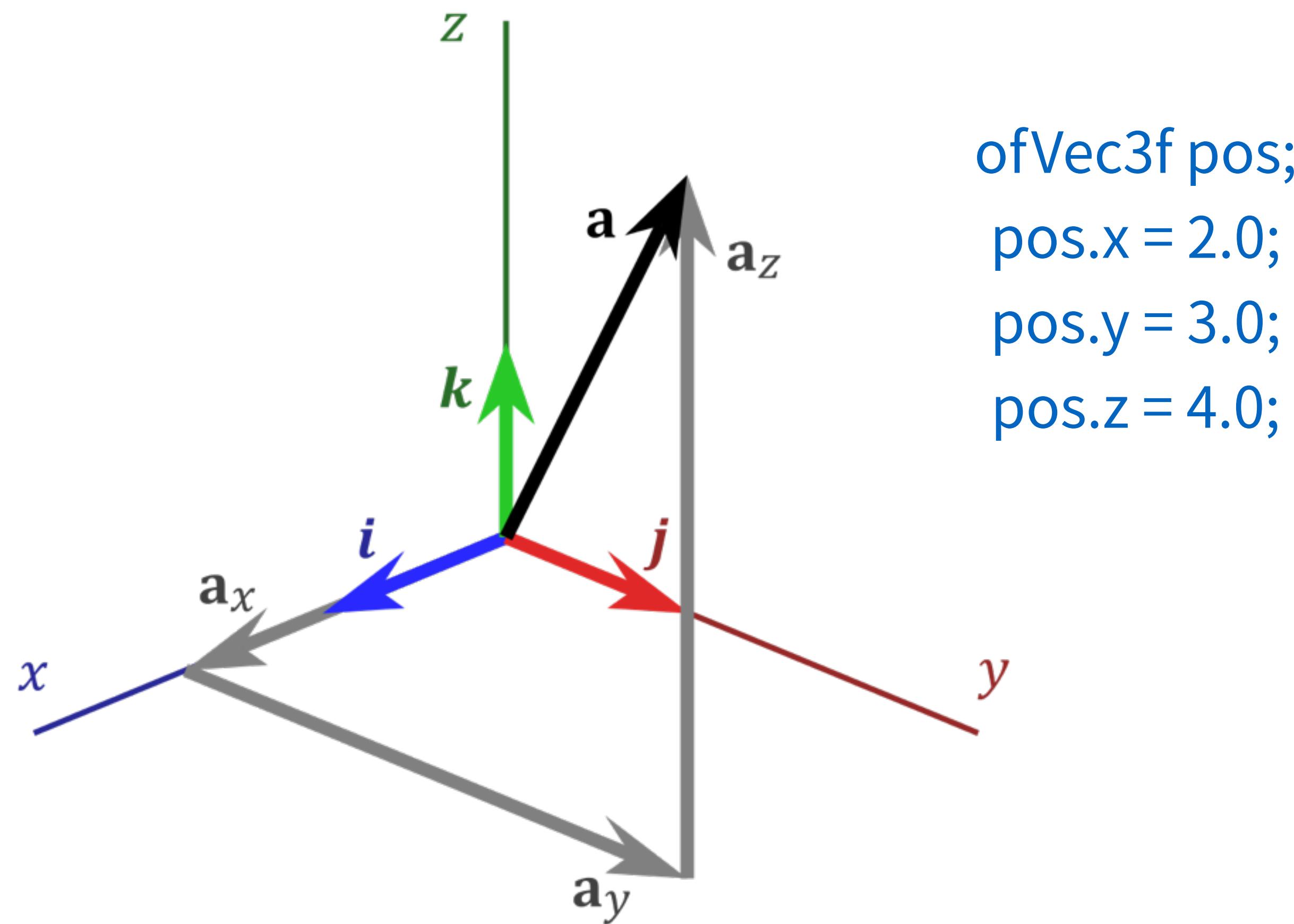
ofVec2fについて

- ▶ ofVec2fで、二次元の位置ベクトルを1つの箱で管理できる
- ▶ ofVec2f pos;
- ▶ X座標 : pos.x , Y座標 : pos.y



ofVec2fについて

- ちなみに、3次元空間のベクトル、ofVec3fもあります



ランダムウォーク

- ▶ testApp.hに、場所を記録するofVec2f型の変数を追加

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{

public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    ...
    void gotMessage(ofMessage msg);

    ofVec2f pos; // 現在位置 ← ここ
};
```

ランダムウォーク

- ▶ testApp.cpp - setup(): 画面の初期設定と、点の位置の初期設定

```
#include "ofApp.h"

void ofApp::setup(){

    //画面初期設定
    ofSetFrameRate(60);
    ofBackground(0);

    //初期位置を画面の中心に
    pos.x = width/2;
    pos.y = width/2;
}
```

ランダムウォーク

- ▶ testApp.cpp - draw() : 位置を更新して、その場所に点を描く

```
void ofApp::draw(){  
  
    //上下左右同じ確率でランダムに移動  
    pos.x += round(ofRandom(-1, 1));  
    pos.y += round(ofRandom(-1, 1));  
  
    //円(点)を描く  
    ofSetColor(255);  
    ofCircle(pos.x, pos.y, 2);  
}
```

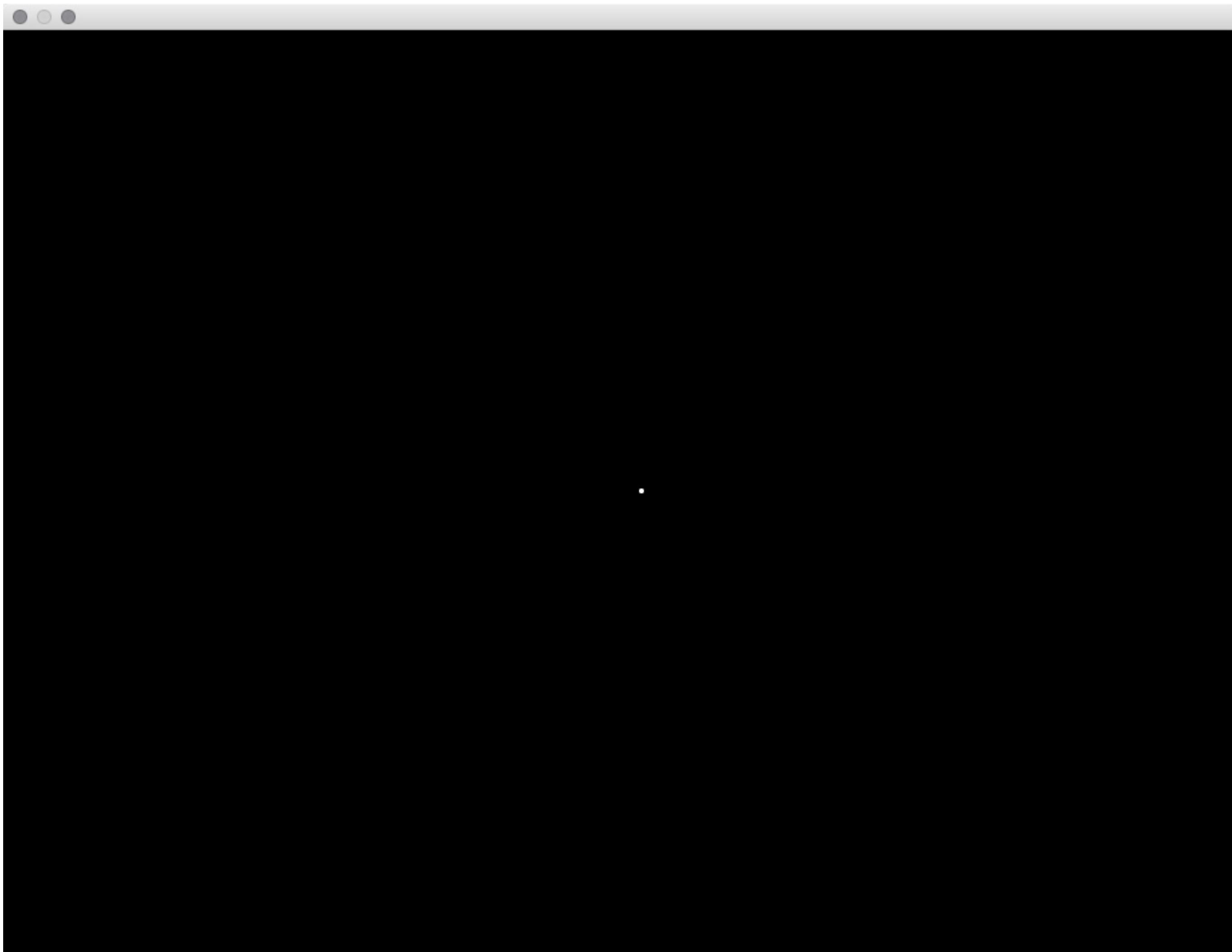
ランダムウォーク

- ▶ testApp.cpp - draw() : 位置を更新して、その場所に点を描く

```
void ofApp::draw(){  
  
    //上下左右同じ確率でランダムに移動  
    pos.x += round(ofRandom(-1, 1));    ] ← ここがポイント  
    pos.y += round(ofRandom(-1, 1));  
  
    //円(点)を描く  
    ofSetColor(255);  
    ofCircle(pos.x, pos.y, 2);  
}
```

ランダムウォーク

- ▶ プルプルと震えるような動き?



ランダムウォーク

- ▶ ランダムに動いた軌跡を見られるようにする
- ▶ 移動の軌跡を残す方法はいろいろ
 - ▶ 座標を記録して、線でつなぐ
 - ▶ ビットマップファイルを用意して、ピクセル操作して画像に軌跡を残す
- ▶ 今回は、さらにシンプルな方法で
 - ▶ 画面の更新を止めて、全てのフレームをそのまま残す

ランダムウォーク

- ▶ testApp.cpp - setup(): 画面の更新をしないように、ちょっと変更

```
#include "ofApp.h"

void ofApp::setup(){

    //画面初期設定
    ofSetFrameRate(60);
    ofBackground(0);

    //画面を更新せず、加算合成していく
    ofSetBackgroundAuto(false);
    ofEnableBlendMode(OF_BLENDMODE_ADD); ] ← 追加

    //初期位置を画面の中心に
    pos.x = ofGetWidth()/2;
    pos.y = ofGetHeight()/2;
}
```

ランダムウォーク

- ▶ testApp.cpp - draw(): 点の色を、ごく薄い白に

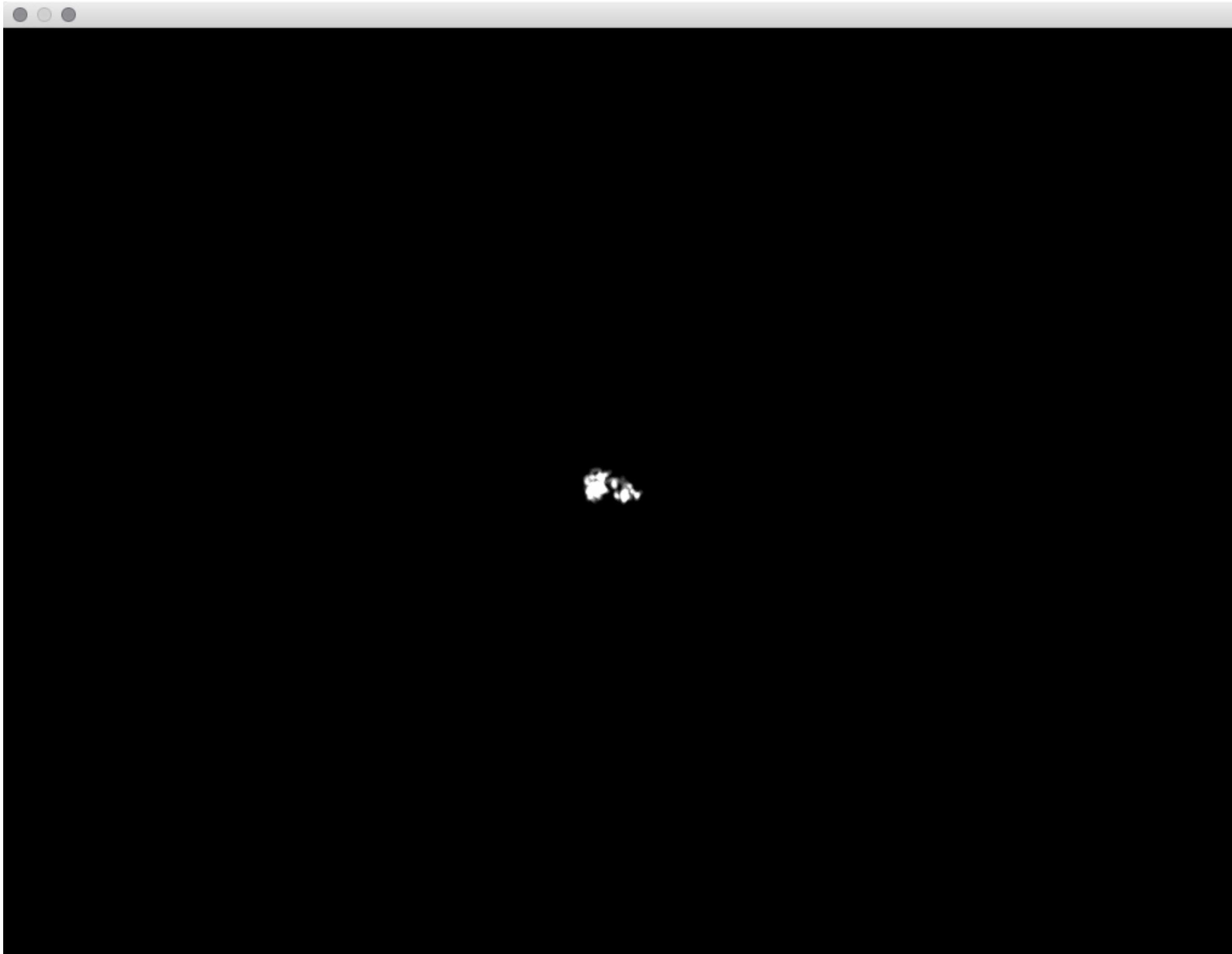
```
void ofApp::draw(){
    ofSetColor(255, 255, 255, 5);

    //上下左右同じ確率でランダムに移動
    pos.x += ofRandom(-1, 1);
    pos.y += ofRandom(-1, 1);

    //円を描く          ← 変更
    ofCircle(pos.x, pos.y, 2);
}
```

ランダムウォーク

- ▶ 移動した軌跡が、ぼんやりと浮かび上がる。…ちょっとまどろっこしい



ランダムウォーク

- ▶ testApp.cpp - draw():くりかえし(for文)をつかって10倍にスピードアップ

```
void ofApp::draw(){
    ofSetColor(255, 255, 255, 5);

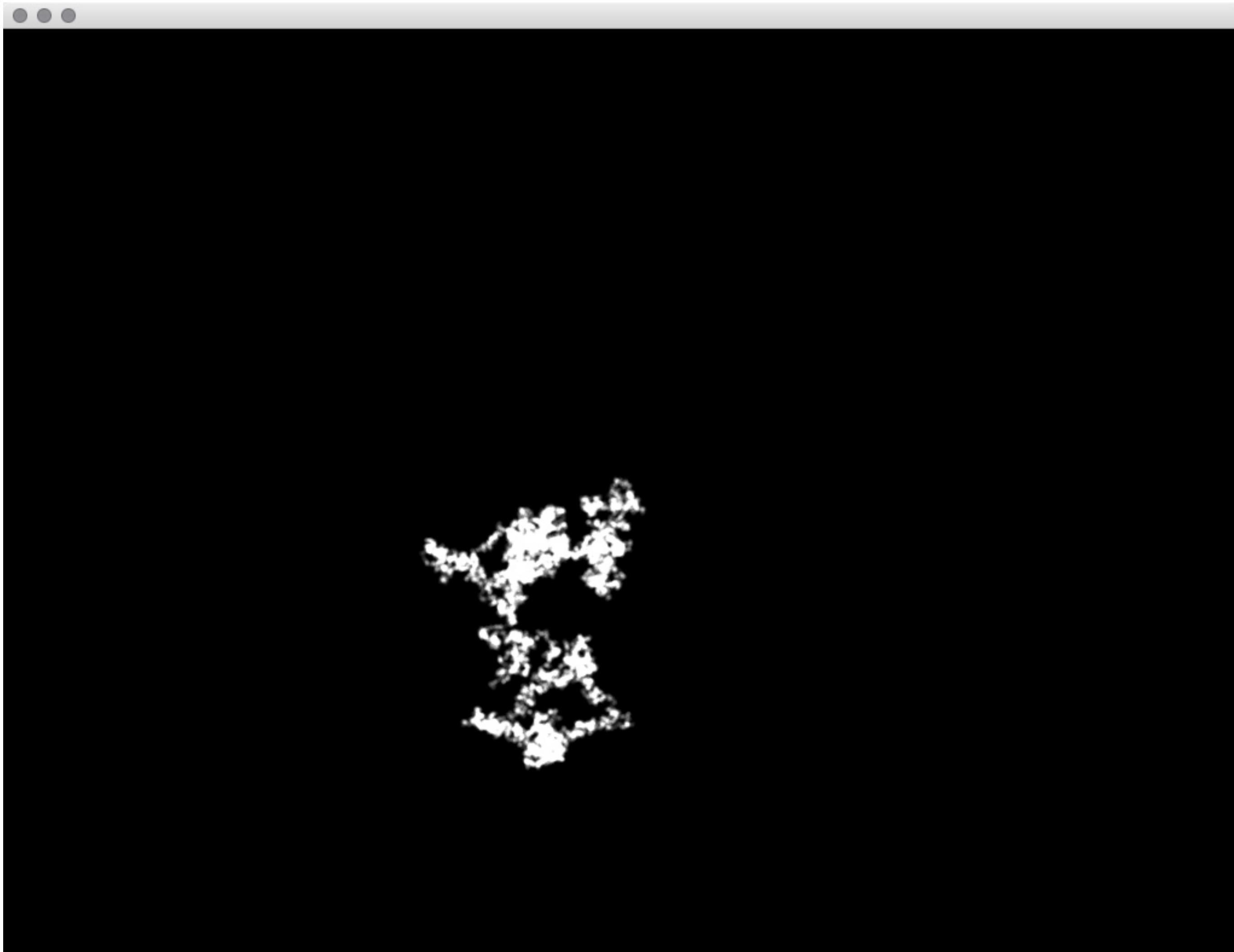
    // 100回くりかえし
    for (int i = 0; i < 10; i++) {

        //上下左右同じ確率でランダムに移動
        pos.x += ofRandom(-1, 1);
        pos.y += ofRandom(-1, 1);

        //円を描く
        ofCircle(pos.x, pos.y, 2);
    }
}
```

ランダムウォーク

- ▶ スピードアップ!! ランダムウォークの軌跡が出現



ランダムウォークを増殖
プログラムを構造化する

プログラムを構造化する

- ▶ このランダムに移動する点を、いちどに沢山表示してみたい
- ▶ どうすれば良いか?

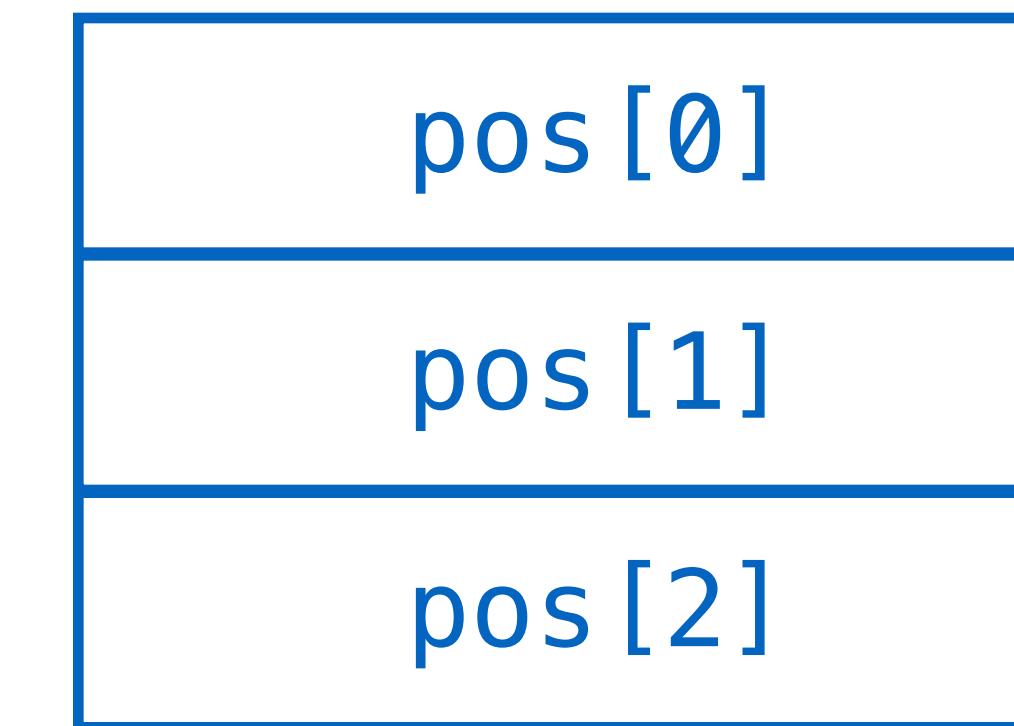
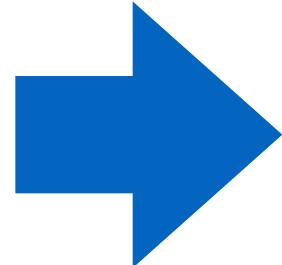
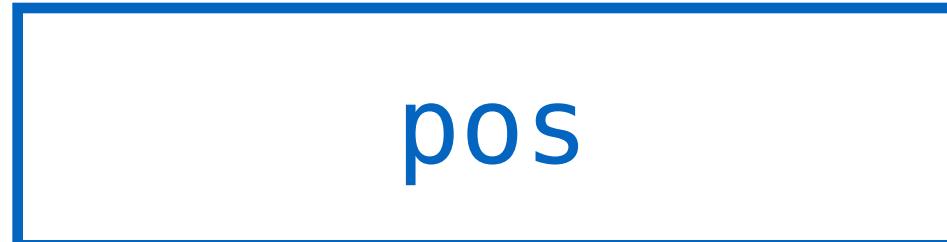
プログラムを構造化する

- ▶ 方法1: 位置をたくさん記憶する
- ▶ ofVec2f で記憶した変数「pos」を大量に用意したい → 配列にする

ofVec2f pos[100];

→ たくさん並んだロッカー

ofVec2f pos;
→ ひとつの箱

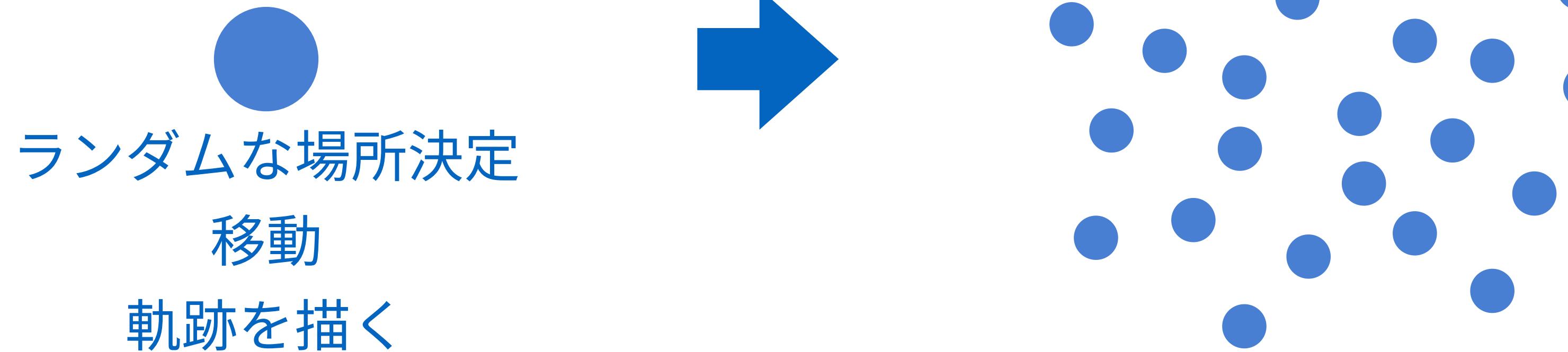


:

pos [99]

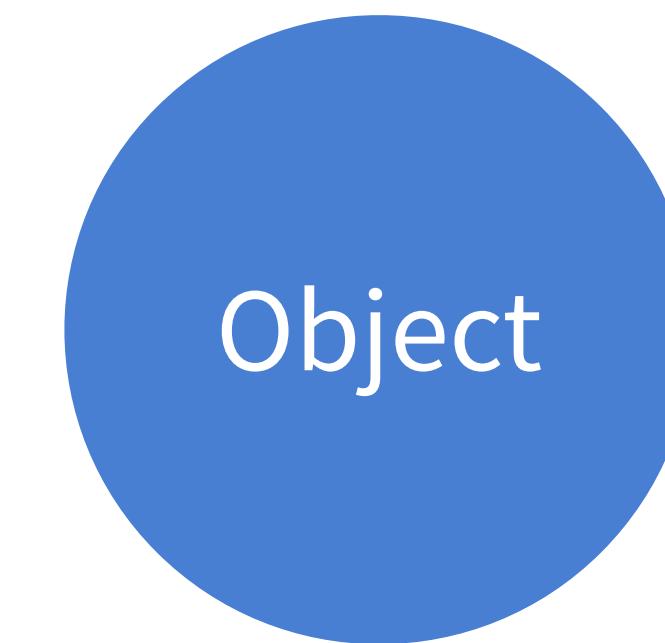
プログラムを構造化する

- ▶ もう1つ別の方法：ランダムに移動する点自身を、大量に生成する
- ▶ 「ランダムな場所を決めて、移動して、軌跡を描く」という機能を大量生成
- ▶ 今回はこの方法を採用



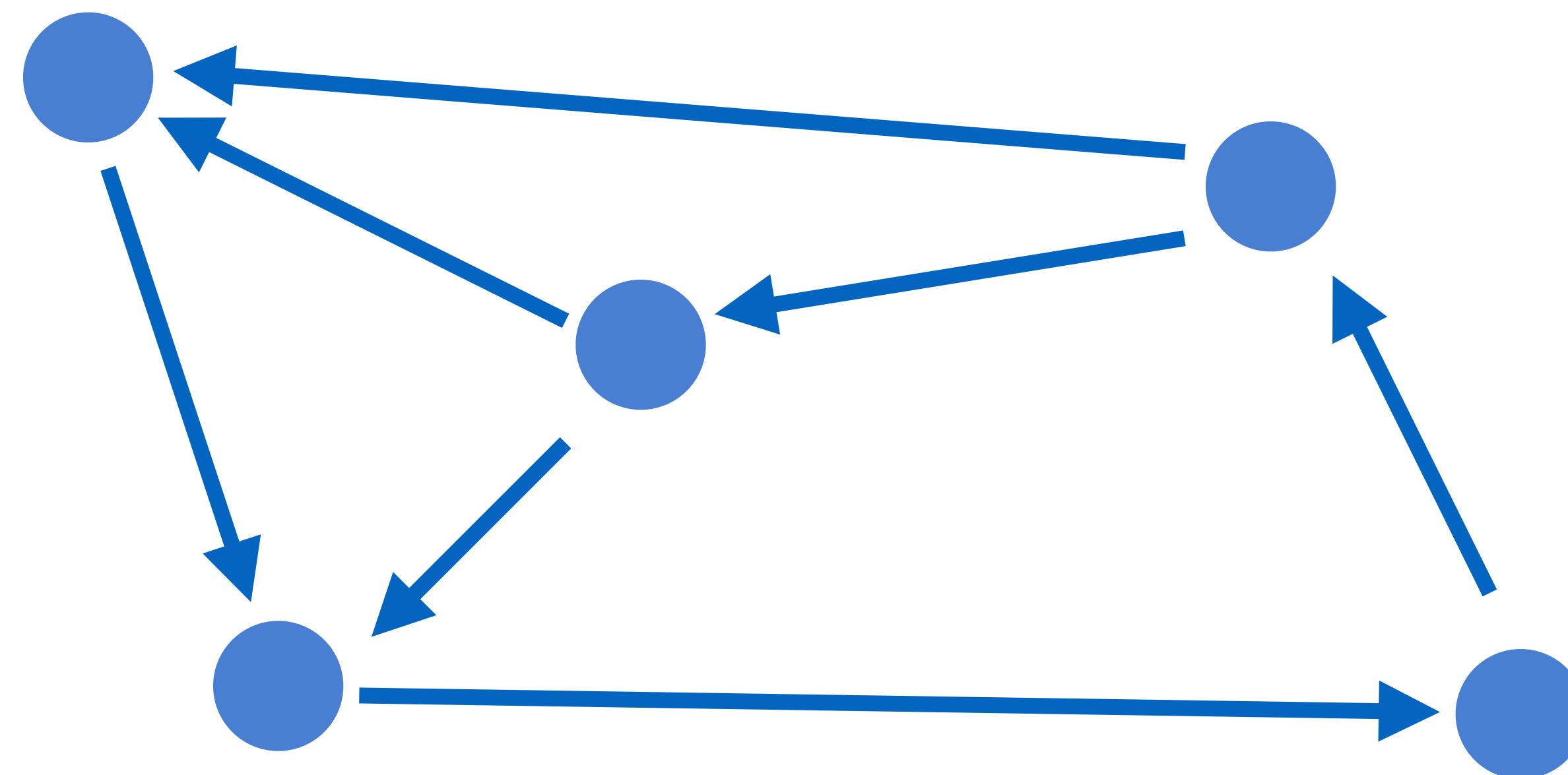
プログラムを構造化する

- ▶ 実は、この「機能の単位」を独立させるという発想は、プログラムの構造化の重要なコンセプトの一つ
- ▶ プログラムの中の独立した機能の単位 → 「オブジェクト（Object）」と呼ぶ



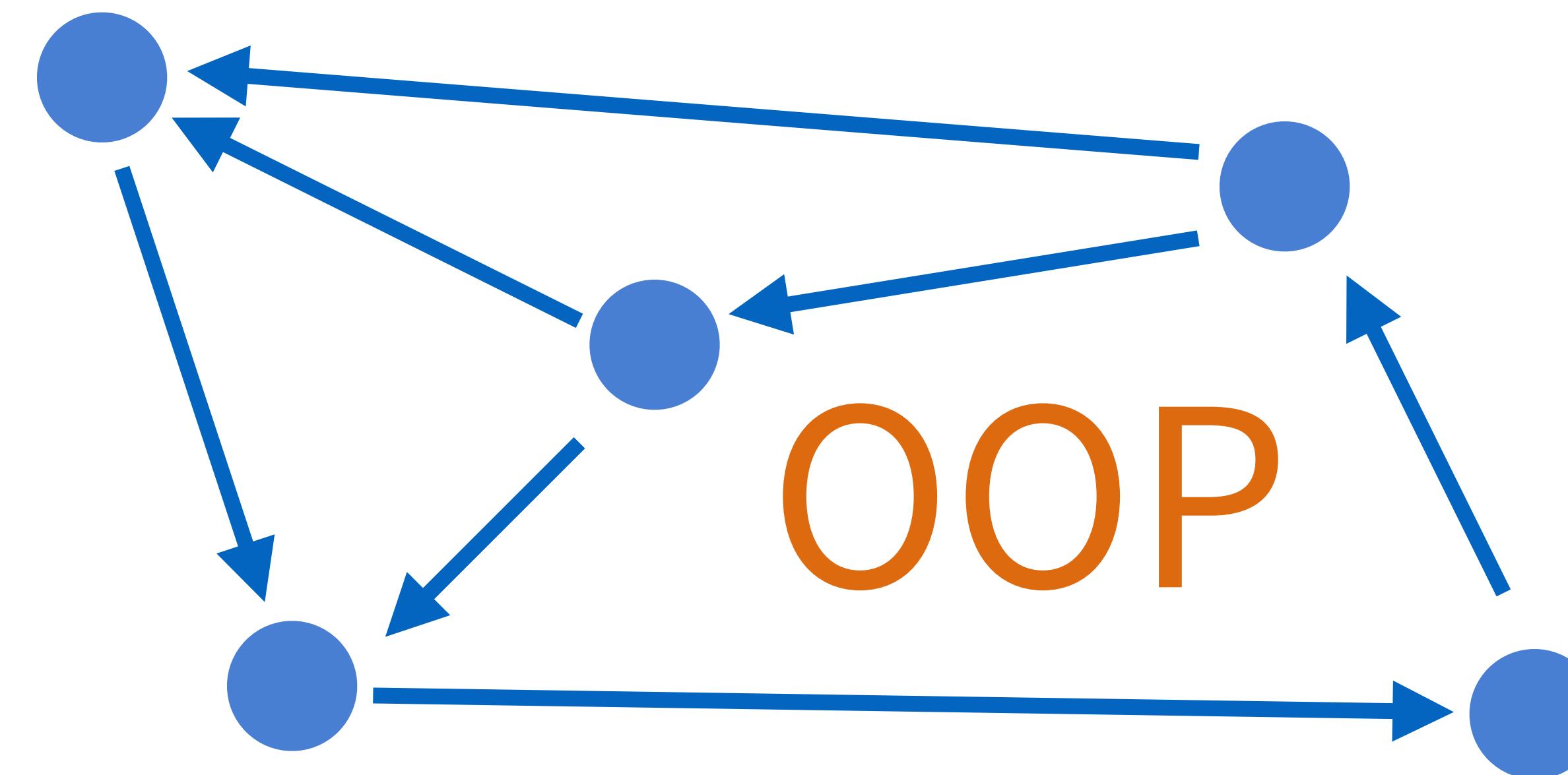
プログラムを構造化する

- ▶ オブジェクトは、それぞれ自律していて、相互にメッセージを送りあう



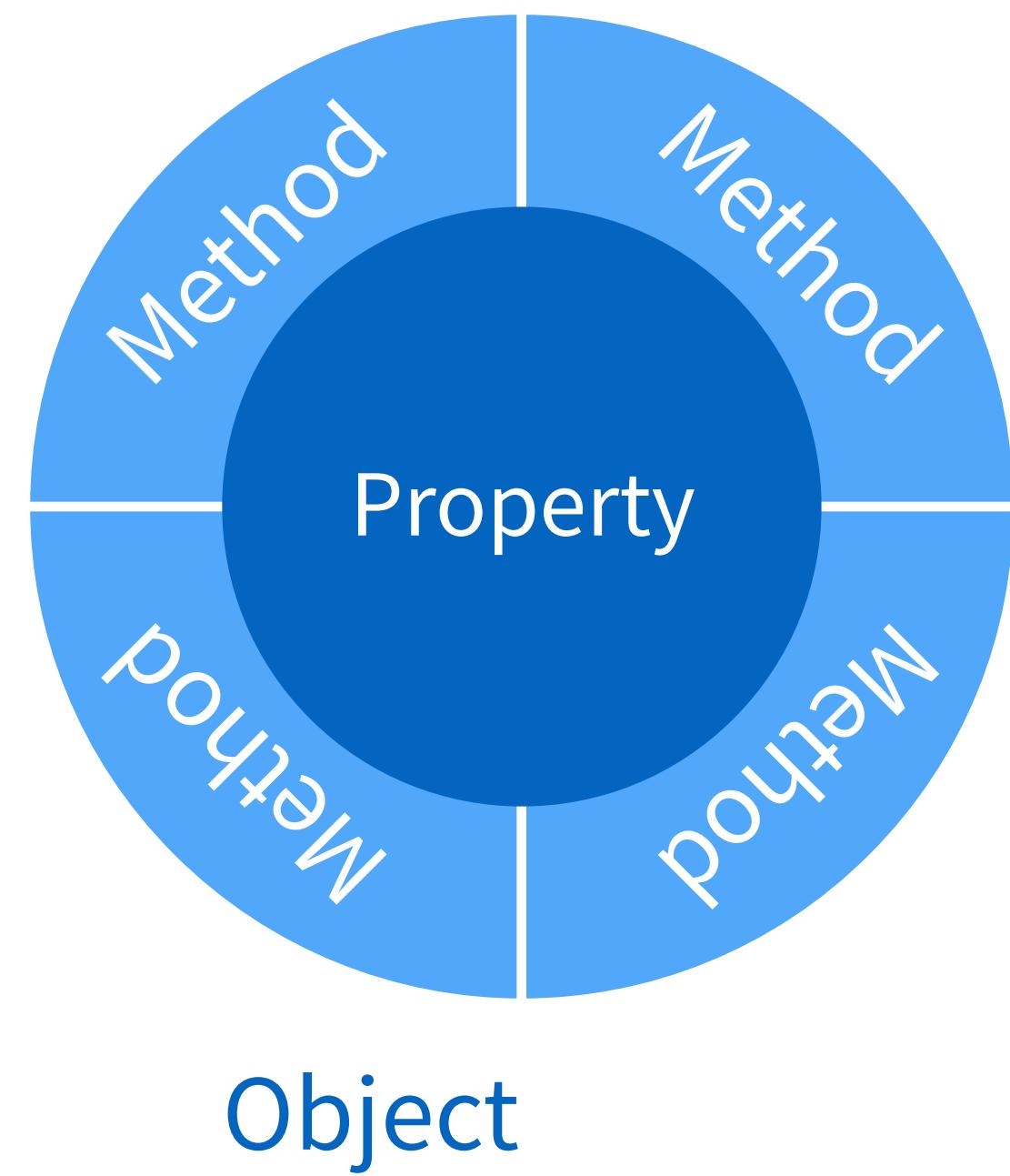
プログラムを構造化する

- こうした、オブジェクトをプログラムの構成単位とするプログラム手法
- オブジェクト指向プログラミング（Object Oriented Programming）と呼ぶ



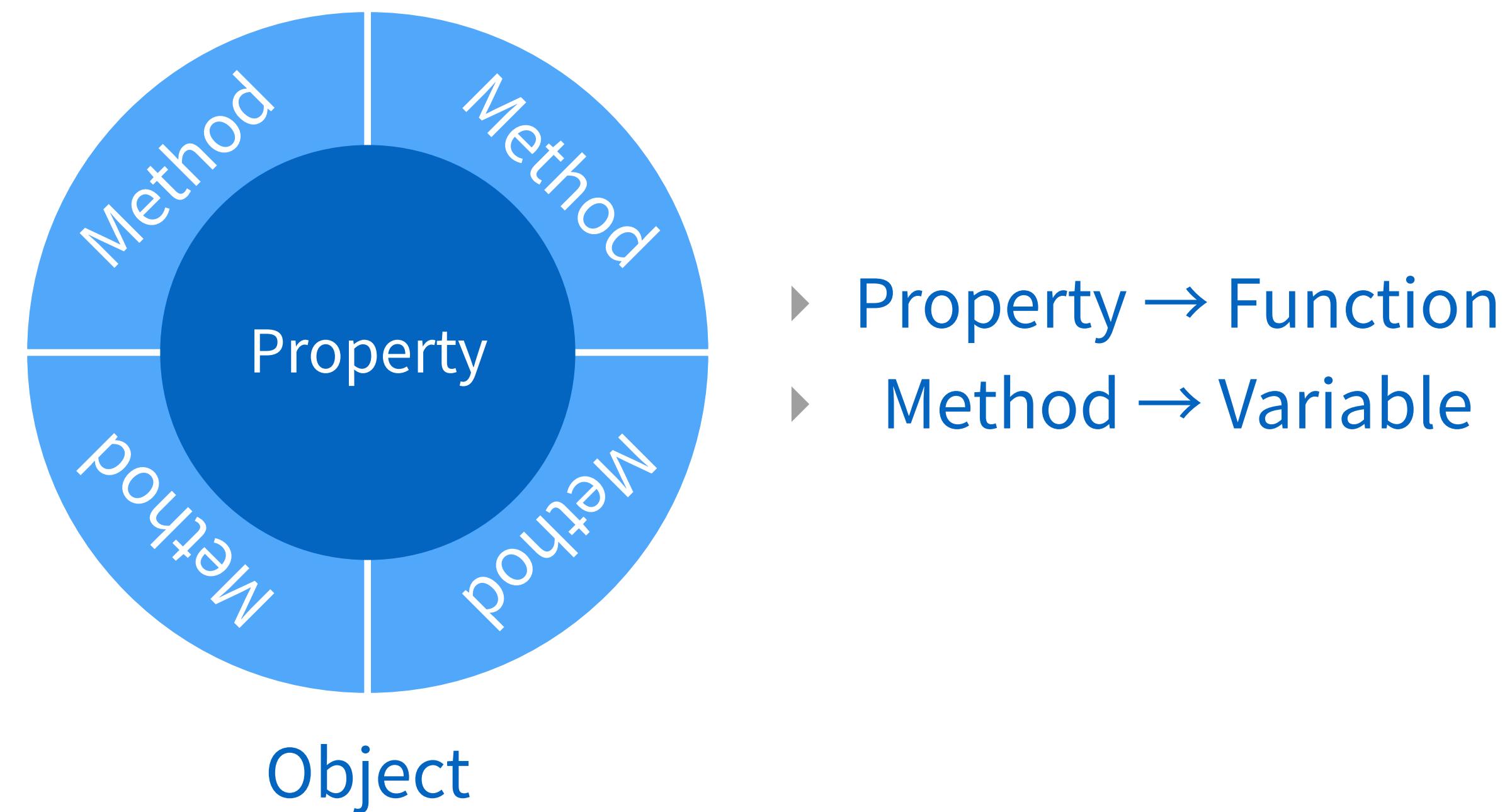
プログラムを構造化する

- ▶ OOPを構成する単位「Object」に注目
- ▶ Objectは、「状態（プロパティー）」と「動作（メソッド）」で特徴を記録する



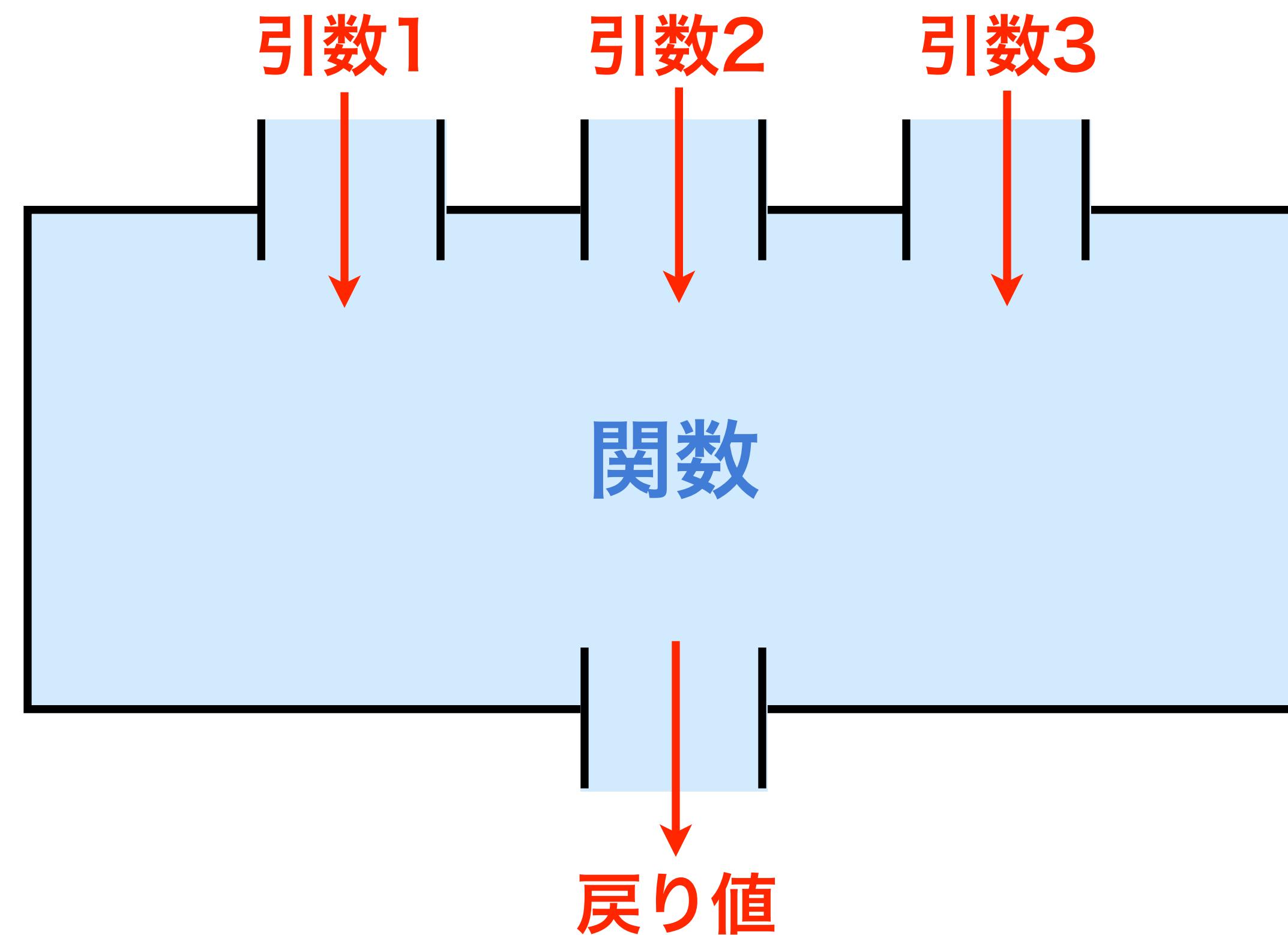
プログラムを構造化する

- ▶ 状態 → 値を記録する = 変数 (Variables)
- ▶ 動作 → 一連の処理をまとめ = 関数 (Functions)



プログラムを構造化する

- ▶ 補足説明: 関数 (Function) とは?
- ▶ 関数 (function) - プログラム内の処理のかたまり
 - ▶ 引数 (ひきすう, argument) - 関数に渡す値 (入力)
- ▶ 返り値 (return value) - 関数が返す値 (出力)



プログラムを構造化する

- ▶ ofAppの、void setup()、void update()、void draw() などは関数

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{

public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);
};
```

← 関数 (Functions)

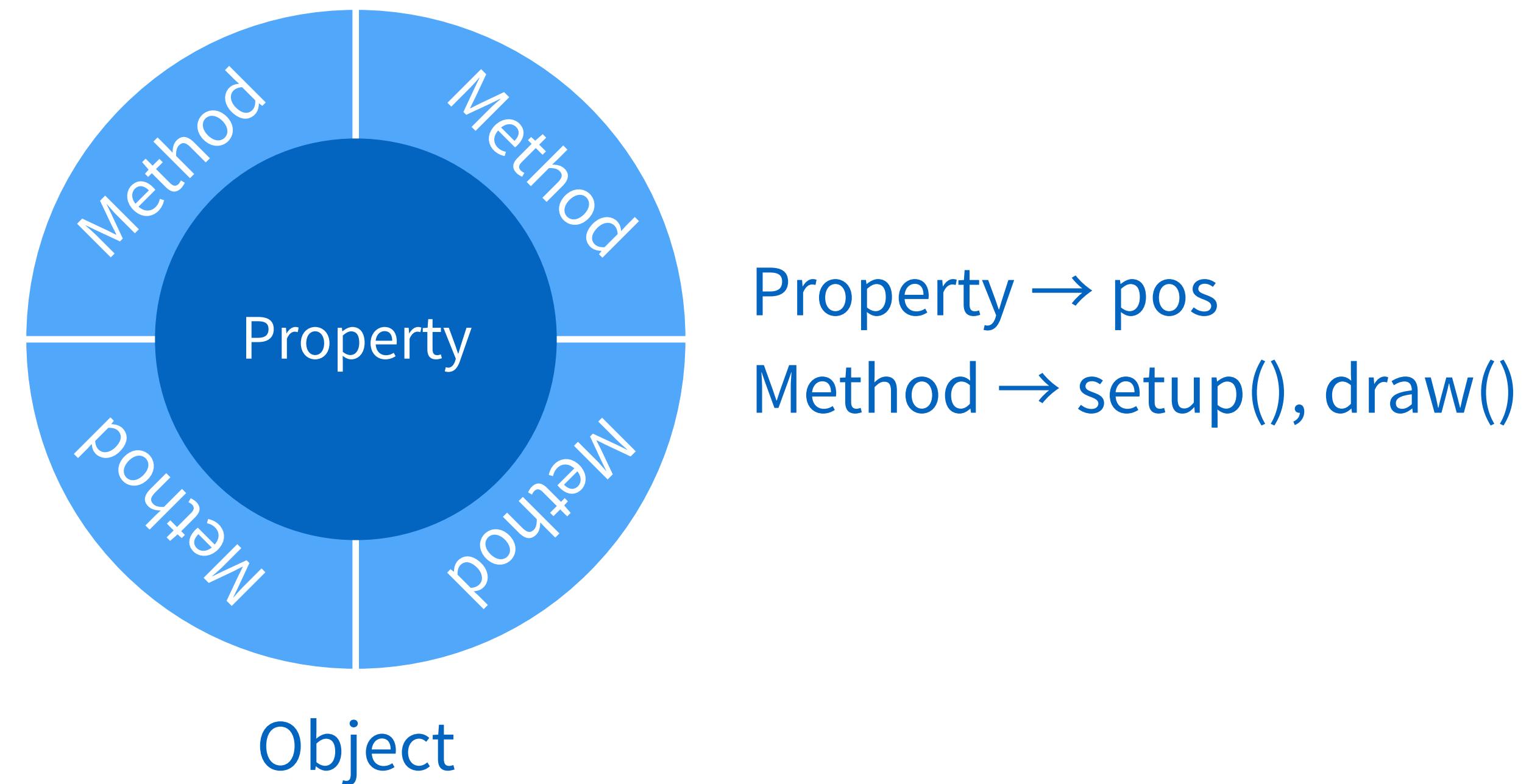
プログラムを構造化する

- 戻り値の型、関数名、引数

戻り値の型	関数名	引数1	引数2
void	mouseMoved	(int x, int y) ;	

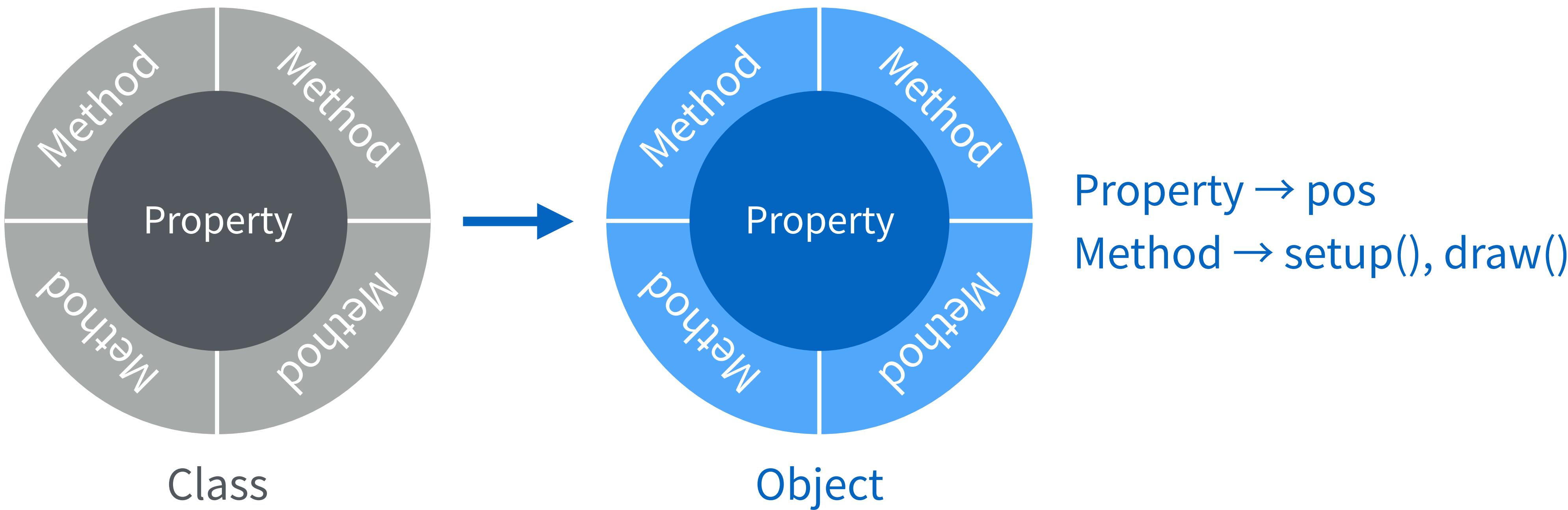
プログラムを構造化する

- ▶ 例えば、ランダムに動く点で考えると
- ▶ 状態（変数） → 点の位置
- ▶ 動作（関数） → 初期設定、描画



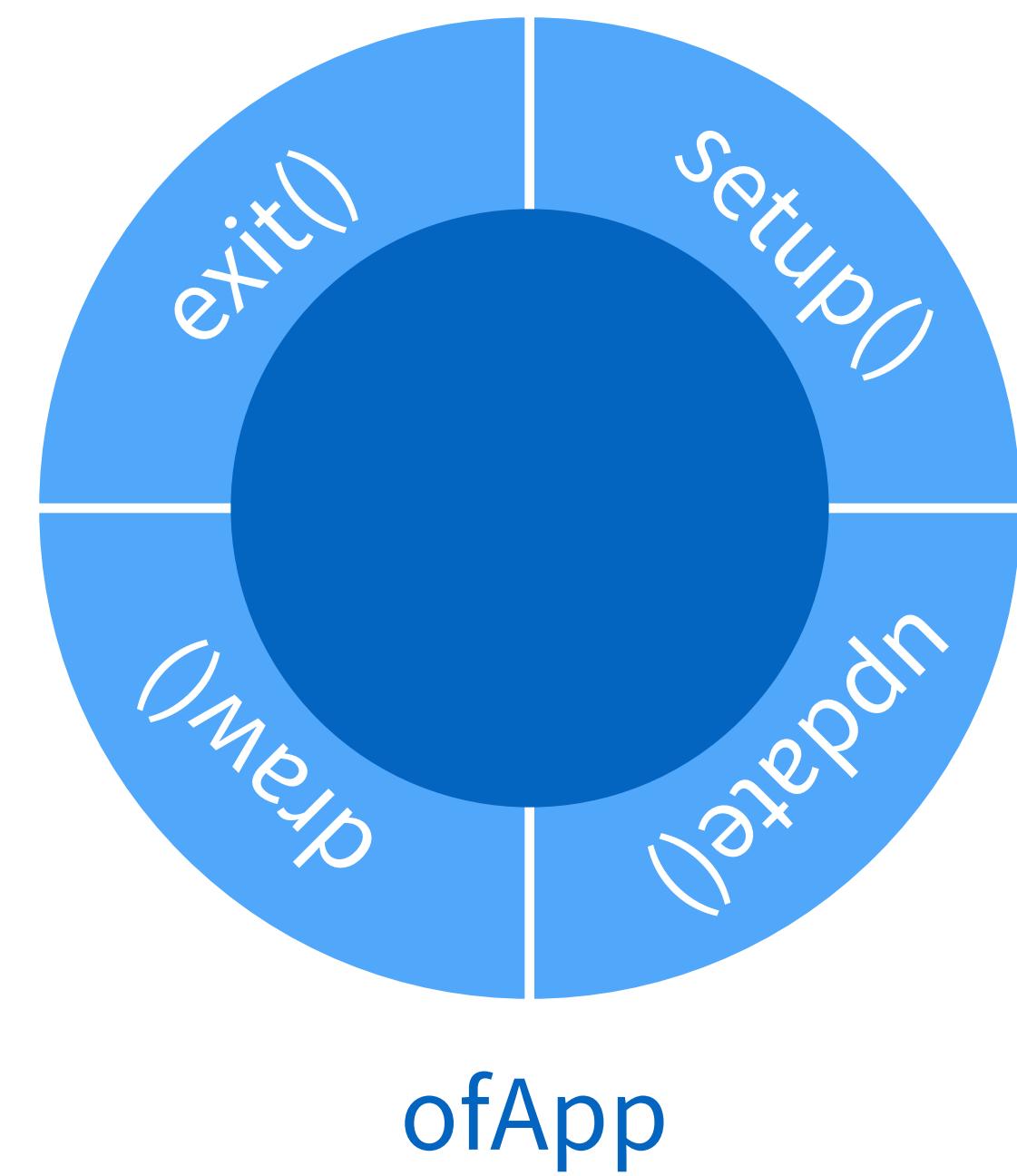
プログラムを構造化する

- ▶ openFrameworksは、C++をベースにしている
- ▶ C++などの「クラスベース」のOOPでは、オブジェクトの設計図をまず書く
- ▶ オブジェクトの設計図のことをクラス (Class) と呼ぶ



プログラムを構造化する

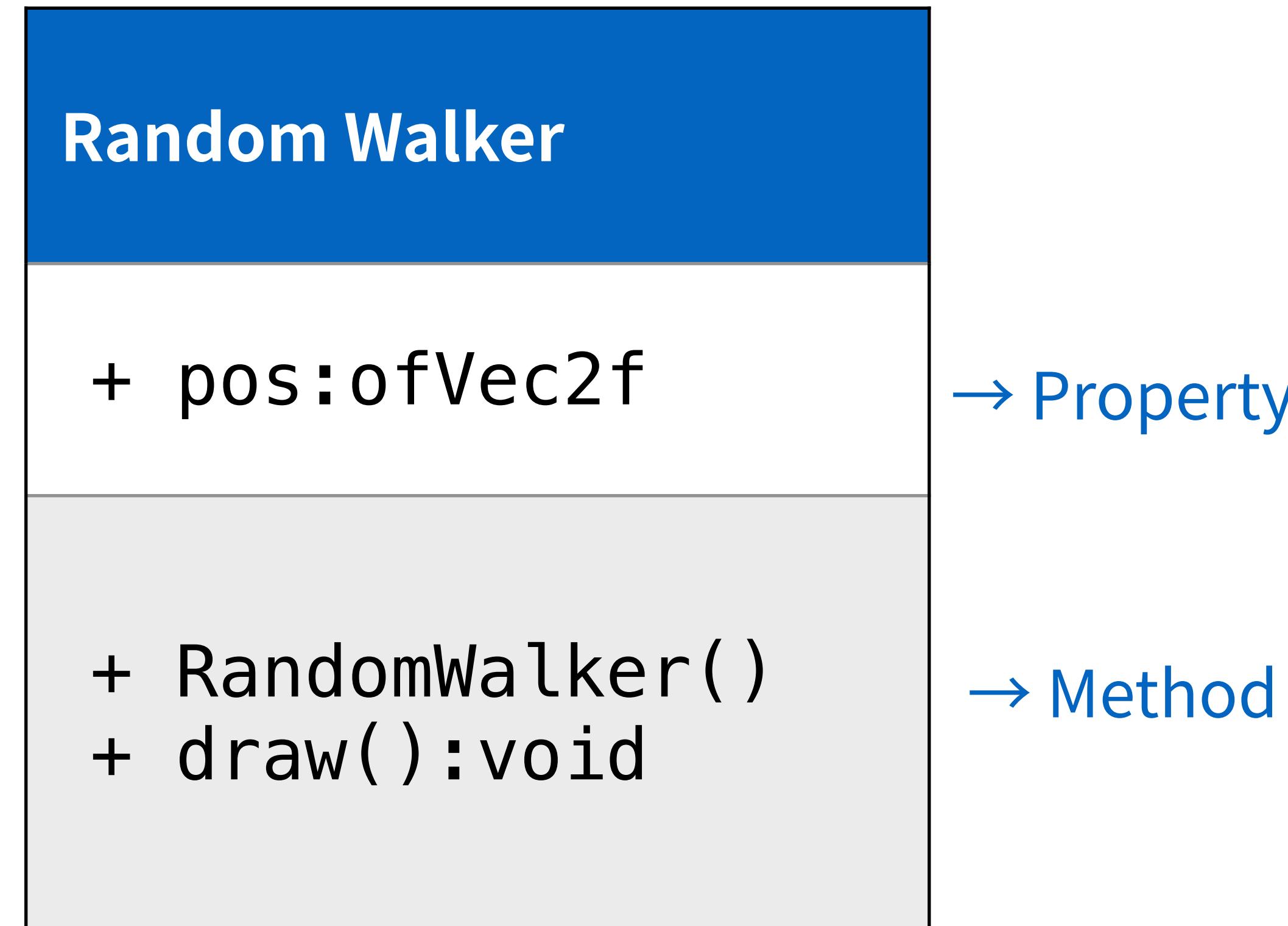
- ▶ 今まで扱ってきた、ofApp も一つのクラス
- ▶ メソッド - setup(), update(), draw() ...etc.
- ▶ プロパティ - ofApp全体で使用する変数



プログラムを構造化する
クラスを作る

クラスを作る

- ▶ 早速クラスを書いてみましょう！
- ▶ ランダムウォークする点のクラス
- ▶ クラス名を「RandomWalker」に設定、機能を表で整理

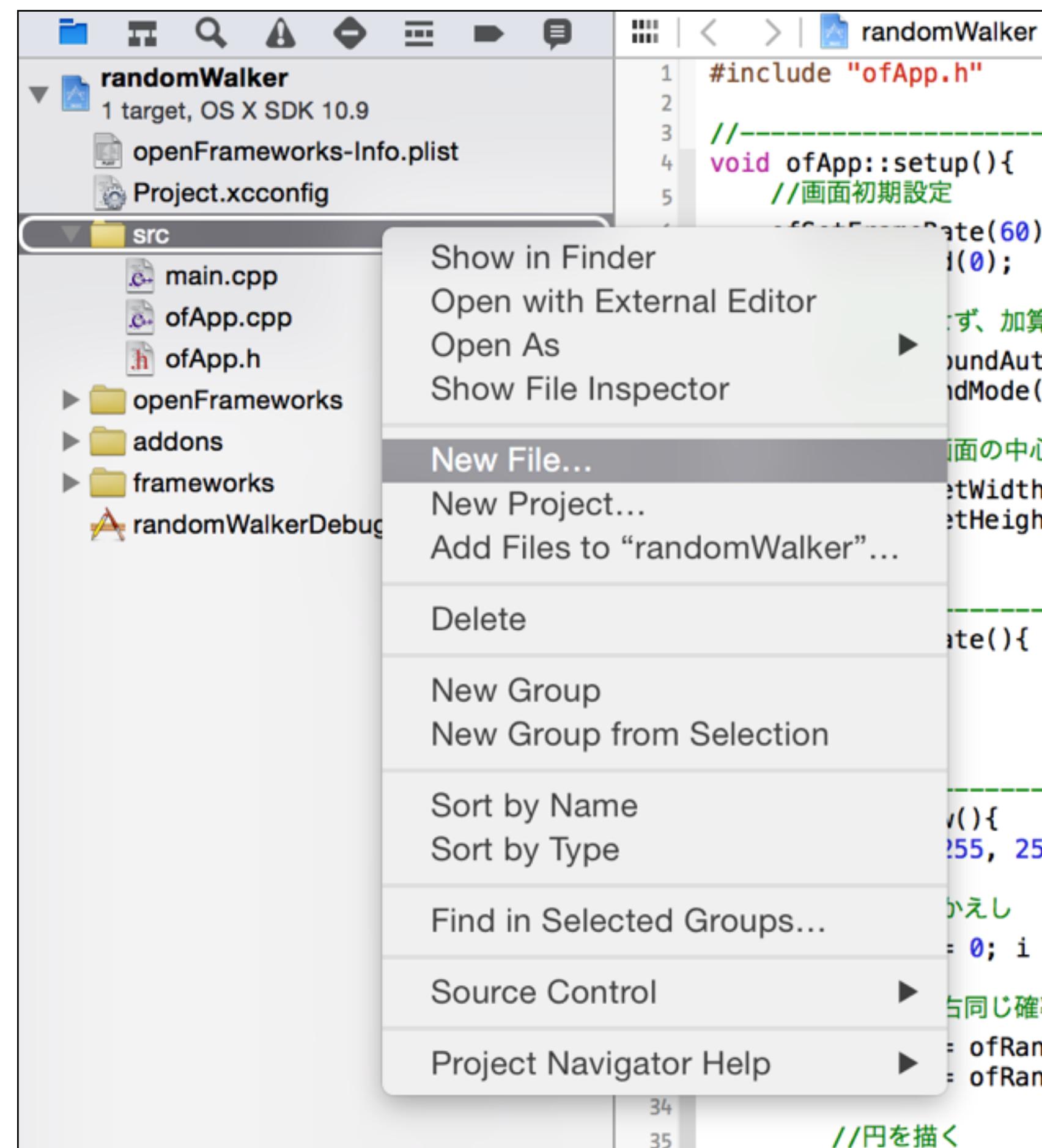


クラスを作る

- ▶ Xcodeを操作して、プロジェクトにクラスのためのファイルを追加します!

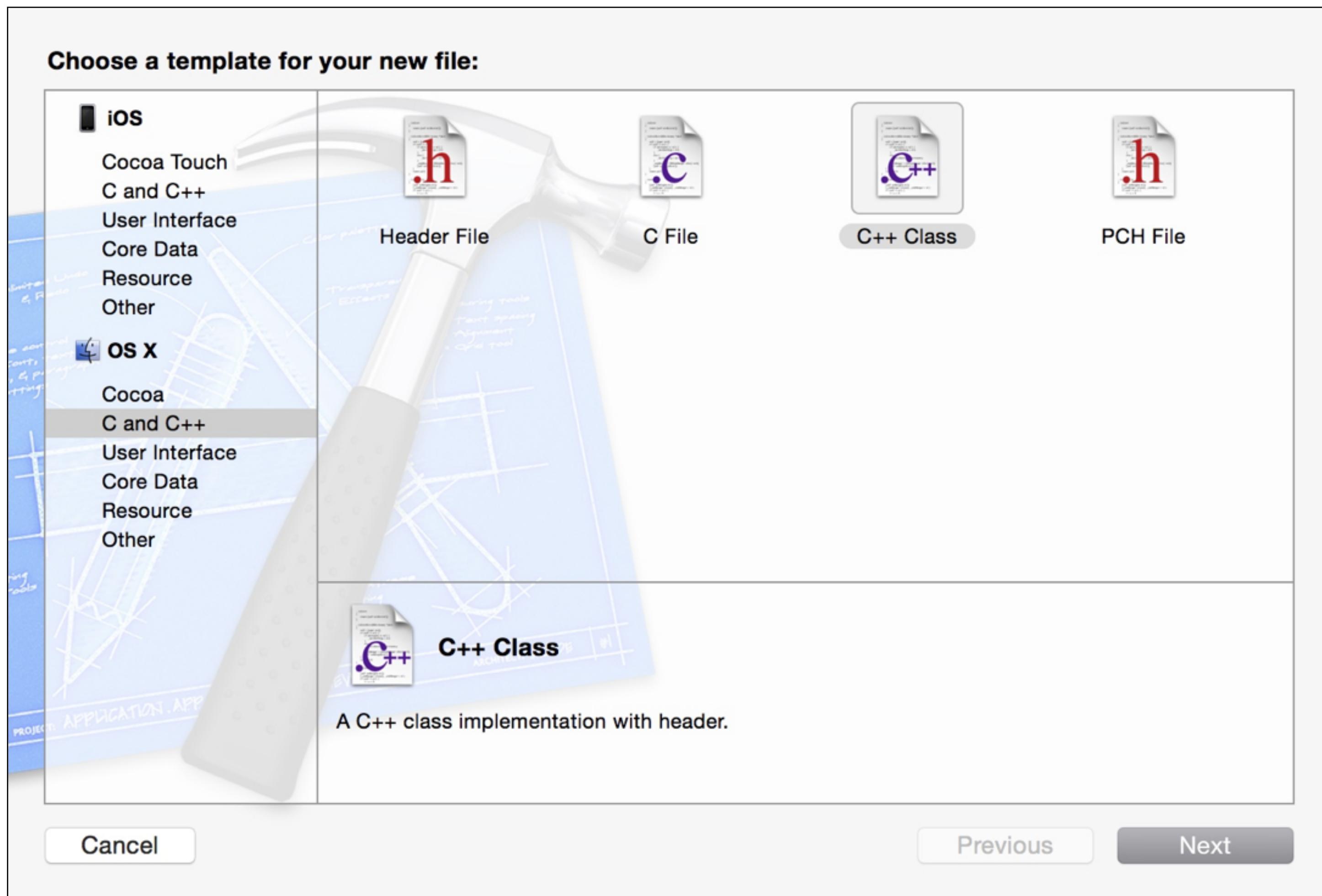
クラスを作る

- ▶ ファイルのリストの「src」 フォルダを右クリック
- ▶ リストから 「New File...」 を選択



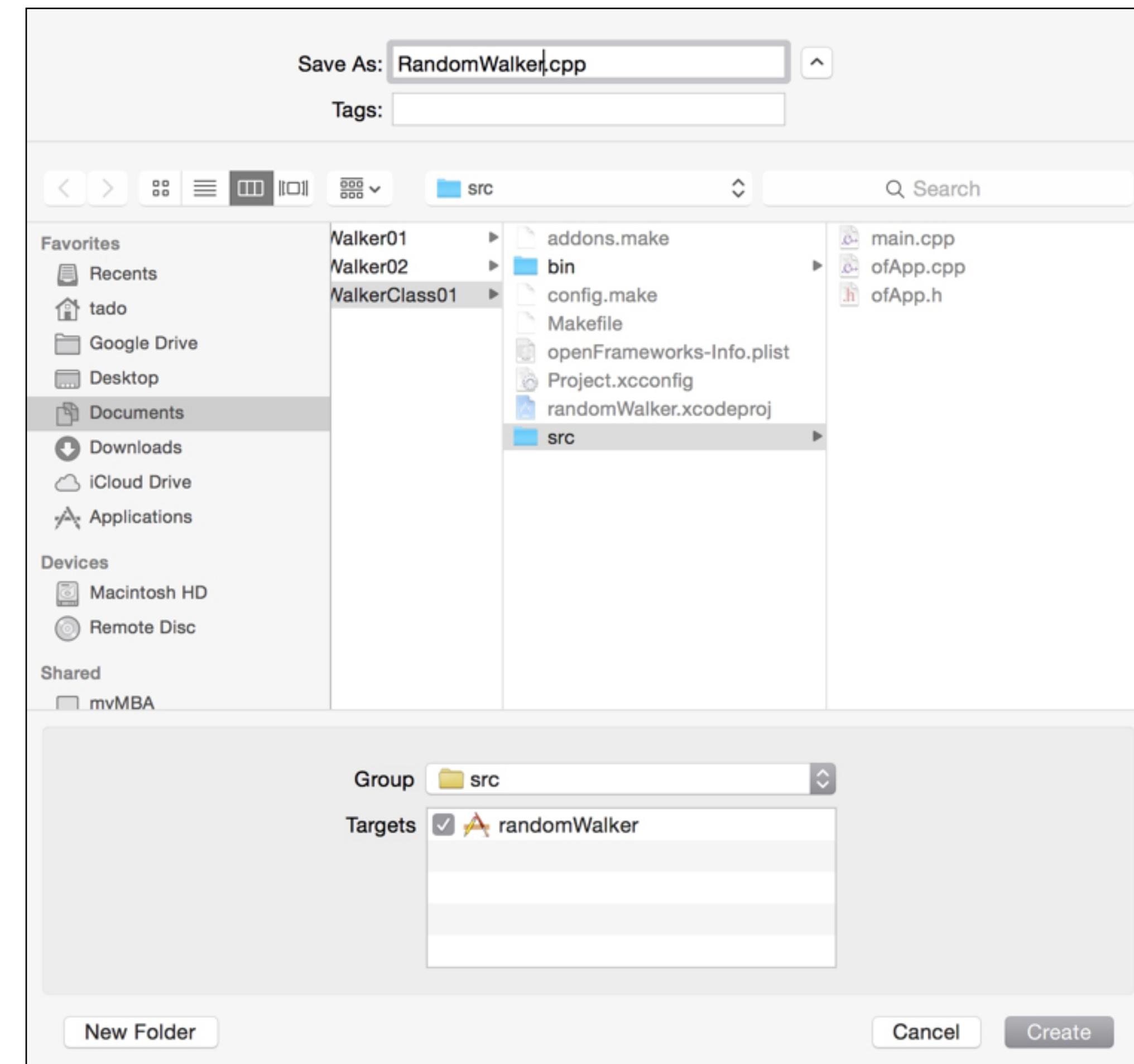
クラスを作る

- ▶ Mac OS X > C and C++ > C++ File を選択



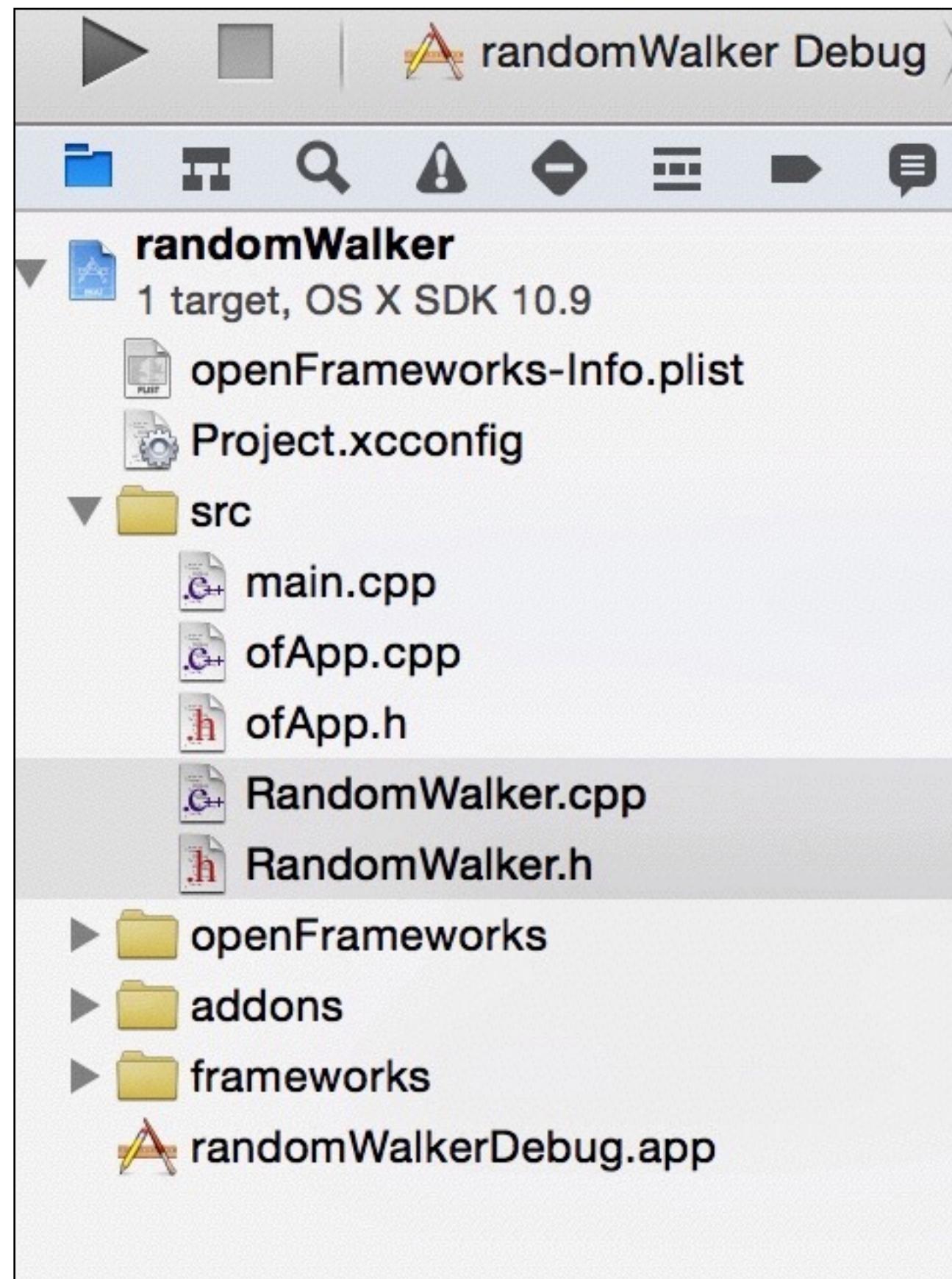
クラスを作る

- ▶ 「RandomWalker」という名前で「src」フォルダに保存
- ▶ 各種設定は、そのままで



クラスを作る

- ▶ ファイルリストは以下のようになるはず
- ▶ あとは、それぞれのファイルにコーディングしていく



クラスを作る

- ▶ RandomWalkerクラスの設計
- ▶ RandomWalker.h (ヘッダーファイル)
 - ▶ 位置を記録する変数 position (ofVec2f) 追加
- ▶ RandomWalker.cpp (実装ファイル)
 - ▶ RandomWalker() - コンストラクタ (後述)
 - ▶ 位置をランダム初期化、ランダムな場所に
 - ▶ void draw()
 - ▶ 位置を更新
 - ▶ 軌跡を描画

クラスを作る

- ▶ コンストラクタ (Constructor) とは?
 - ▶ クラスが初期化される際に呼びだされる特別な関数
 - ▶ 関数名は、クラス名と同じ名前にする
 - ▶ 戻り値はない
-
- ▶ RandomWalkerクラスの場合は、RandomWalker()



クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h"

class RandomWalker {

public:

    RandomWalker();
    void draw();

    ofVec2f position;
};
```

クラスを作る

▶ RandomWalker.h

```
#pragma once ← インクルードガード
#include "ofMain.h" Buildの際に複数回読みこまれないためのしくみ

class RandomWalker {
public:
    RandomWalker();
    void draw();

    ofVec2f position;
};
```

クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h" ← openFrameworksの機能を使うためのライブラリを読み込む

class RandomWalker {

public:

    RandomWalker();
    void draw();

    ofVec2f position;
};
```

クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h"

class RandomWalker { ← クラス名

public:

    RandomWalker();
    void draw();

    ofVec2f position;
};
```

クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h"

class RandomWalker {

public:
    RandomWalker();
    void draw();
    ofVec2f position;
};
```

← “public:” 以下に書いた内容は、外部に公開される

クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h"

class RandomWalker {

public:

    RandomWalker(); ← コンストラクター
    void draw();

    ofVec2f position;
};
```

クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h"

class RandomWalker {

public:

    RandomWalker();
    void draw();      ← 位置の更新と、軌跡の描画

    ofVec2f position;
};
```

クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h"

class RandomWalker {

public:

    RandomWalker();
    void draw();

    ofVec2f position; ← 位置
};
```

クラスを作る

▶ RandomWalker.h

```
#pragma once

#include "ofMain.h"

class RandomWalker {

public:

    RandomWalker();
    void draw();

    ofVec2f position;
}; ← 最後に必ずセミコロン「;」を入れる
```

クラスを作る

- ▶ 次に実装ファイル (RandomWalker.cpp)
- ▶ ヘッダーで設計したとおりに、実装していく

クラスを作る

▶ RandomWalker.cpp - コンストラクター

```
#include "RandomWalker.h"

RandomWalker::RandomWalker(){

    //初期位置を、画面内にランダムに設定
    position.x = ofRandom(ofGetWidth());
    position.y = ofRandom(ofGetHeight());
}
```

クラスを作る

▶ RandomWalker.cpp - draw()

```
void RandomWalker::draw(){
    // 10回くりかえし
    for (int i = 0; i < 10; i++) {

        //上下左右同じ確率でランダムに移動
        position.x += ofRandom(-1, 1);
        position.y += ofRandom(-1, 1);

        //円を描く
        ofCircle(position.x, position.y, 2);
    }
}
```

クラスを作る

- ▶ RandomWalker.cpp - draw() - 画面の端からはみ出さないように

```
void RandomWalker::draw(){
    // 10回くりかえし
    for (int i = 0; i < 10; i++) {

        //上下左右同じ確率でランダムに移動
        position.x += ofRandom(-1, 1);
        position.y += ofRandom(-1, 1);

        // 画面からはみ出たら、反対側から出現
        if (position.x < 0) {
            position.x = ofGetWidth();
        }
        if (position.x > ofGetWidth()) {
            position.x = 0;
        }
        if (position.y < 0) {
            position.y = ofGetHeight();
        }
        if (position.y > ofGetHeight()) {
            position.y = 0;
        }

        //円を描く
        ofCircle(position.x, position.y, 2);
    }
}
```



← 追加

クラスを作る

- ▶ RandomWalkerクラスは完成!!
 - ▶ これをプロジェクト全体に組込む
-
- ▶ ofApp.h と ofApp.cpp を変更して、RandomWalkerを生成して描画する

クラスを作る

- ▶ openFrameworks (C++) のプログラム
- ▶ まず始めに「main.cpp」が実行される
- ▶ openFrameworksでは、main.cppから、ofAppクラスが実行される
- ▶ ofAppをハブとして、様々なクラスを呼び出す



クラスを作る

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "RandomWalker.h"

class ofApp : public ofBaseApp{

public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    RandomWalker walker;
};

};
```

クラスを作る

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "RandomWalker.h" ← 追加するクラスのヘッダーを読み込む

class ofApp : public ofBaseApp{

public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    RandomWalker walker;
};

};
```

クラスを作る

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "RandomWalker.h"

class ofApp : public ofBaseApp{

public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    RandomWalker walker; ← クラスからオブジェクトを生成
};
```

クラスを作る

▶ ofApp.cpp - setup()

```
void ofApp::setup(){
    //画面初期設定
    ofSetFrameRate(60);
    ofBackground(0);

    //画面を更新せず、加算合成していく
    ofSetBackgroundAuto(false);
    ofEnableBlendMode(OF_BLENDMODE_ADD);
}
```

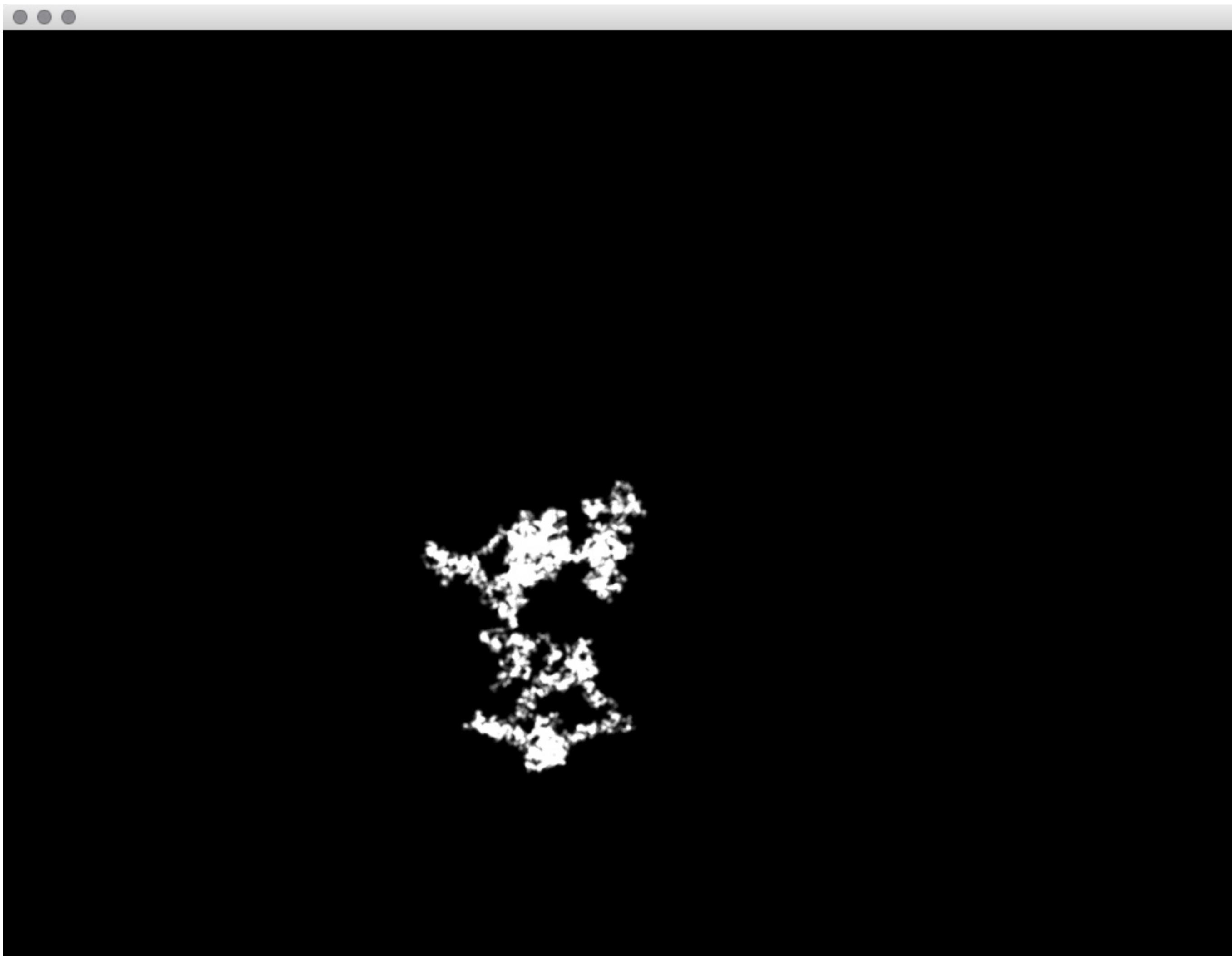
クラスを作る

▶ ofApp.cpp - draw()

```
void ofApp::draw(){  
  
    // 色を設定  
    ofSetColor(255, 255, 255, 5);  
  
    // RandomWalkerを表示  
    walker.draw();  
}
```

クラスを作る

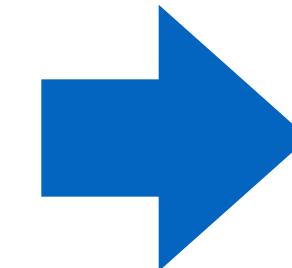
- ▶ ランダムウォークの軌跡が出現（ただし、まだ1つ）



クラスを作る

- ▶ いよいよ、RandomWalkerを増殖させてみましょう！
- ▶ 方法は簡単、RandomWalkerを配列にすればOK。

RandomWalker walker;
1つのwalker



RandomWalker walker[100];
100個のwalker

クラスを作る

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "RandomWalker.h"

class ofApp : public ofBaseApp{

public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    RandomWalker walker[100]; ← 100個の配列に
};
```

クラスを作る

▶ ofApp.cpp - draw()

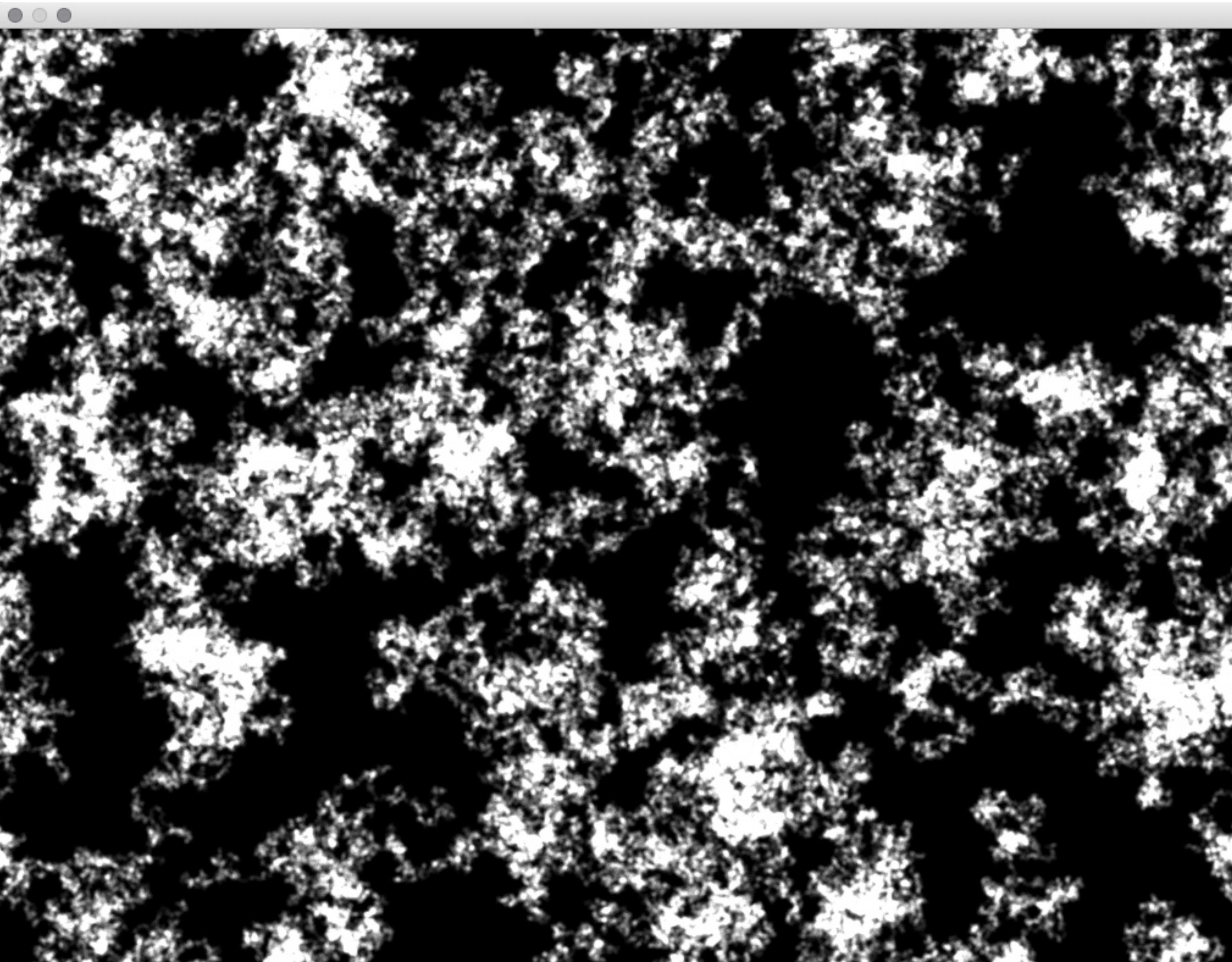
```
void ofApp::draw(){

    // 色を設定
    ofSetColor(255, 255, 255, 5);

    // RandomWalkerを100個表示
    for (int i = 0; i < 100; i++) {
        walker[i].draw();
    }
}
```

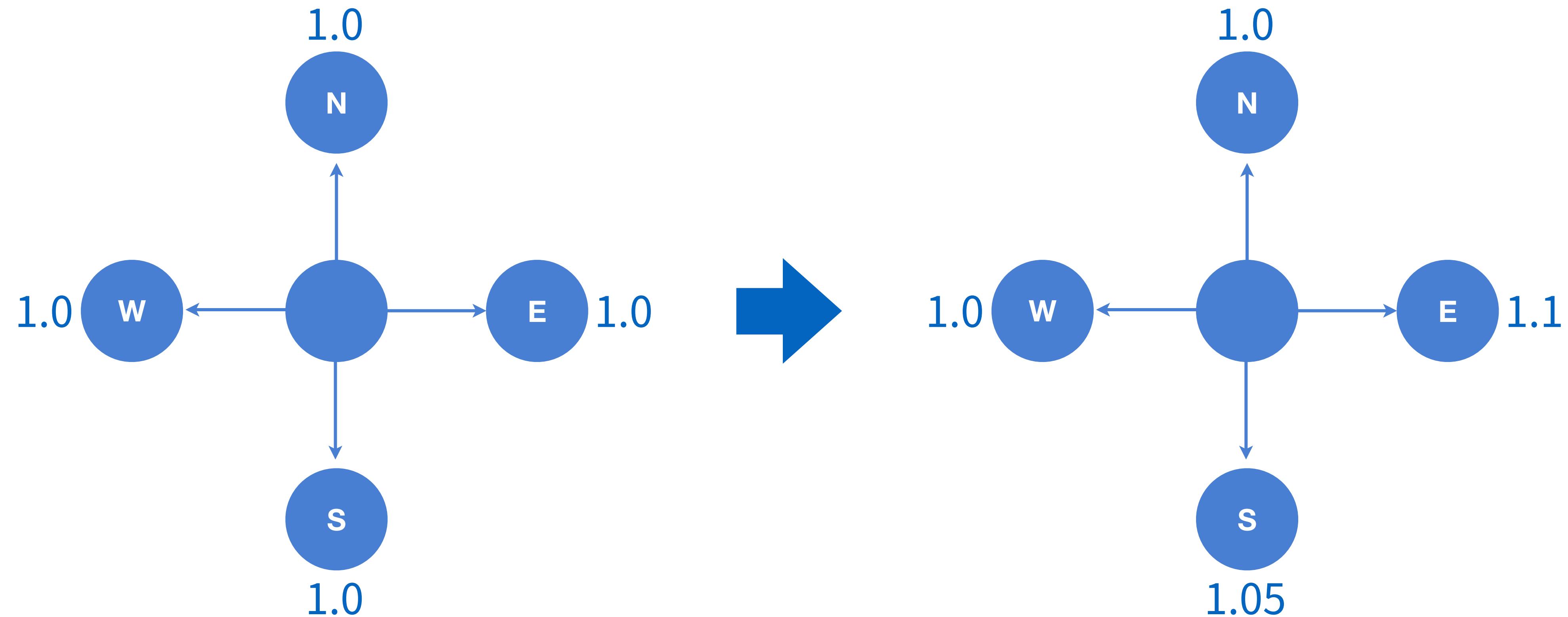
クラスを作る

- ▶ 100個のランダムウォークが完成!!



クラスを作る

- ▶ 現在の動きは、上下、左右同じ確率
- ▶ この確率を微妙に操作したら、どうなるのか？



クラスを作る

▶ RandomWalker.cpp - 確率を変更

```
void RandomWalker::draw(){
    // 10回くりかえし
    for (int i = 0; i < 10; i++) {

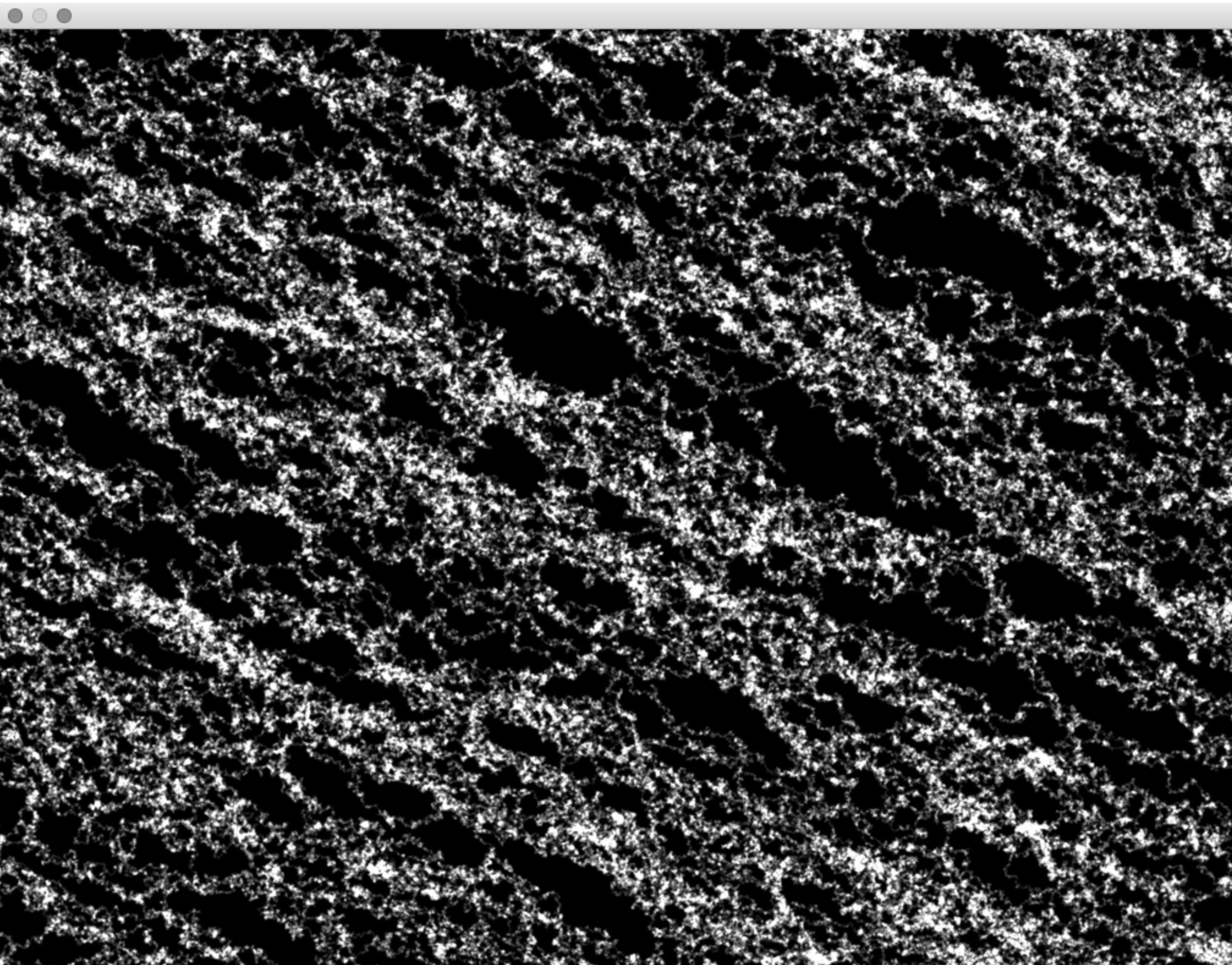
        //上下左右同じ確率でランダムに移動
        position.x += ofRandom(-1, 1.1);
        position.y += ofRandom(-1, 1.01); } ← ここを変更

        // 画面からはみ出たら、反対側から出現
        if (position.x < 0) {
            position.x = ofGetWidth();
        }
        if (position.x > ofGetWidth()) {
            position.x = 0;
        }
        if (position.y < 0) {
            position.y = ofGetHeight();
        }
        if (position.y > ofGetHeight()) {
            position.y = 0;
        }

        //円を描く
        ofCircle(position.x, position.y, 2);
    }
}
```

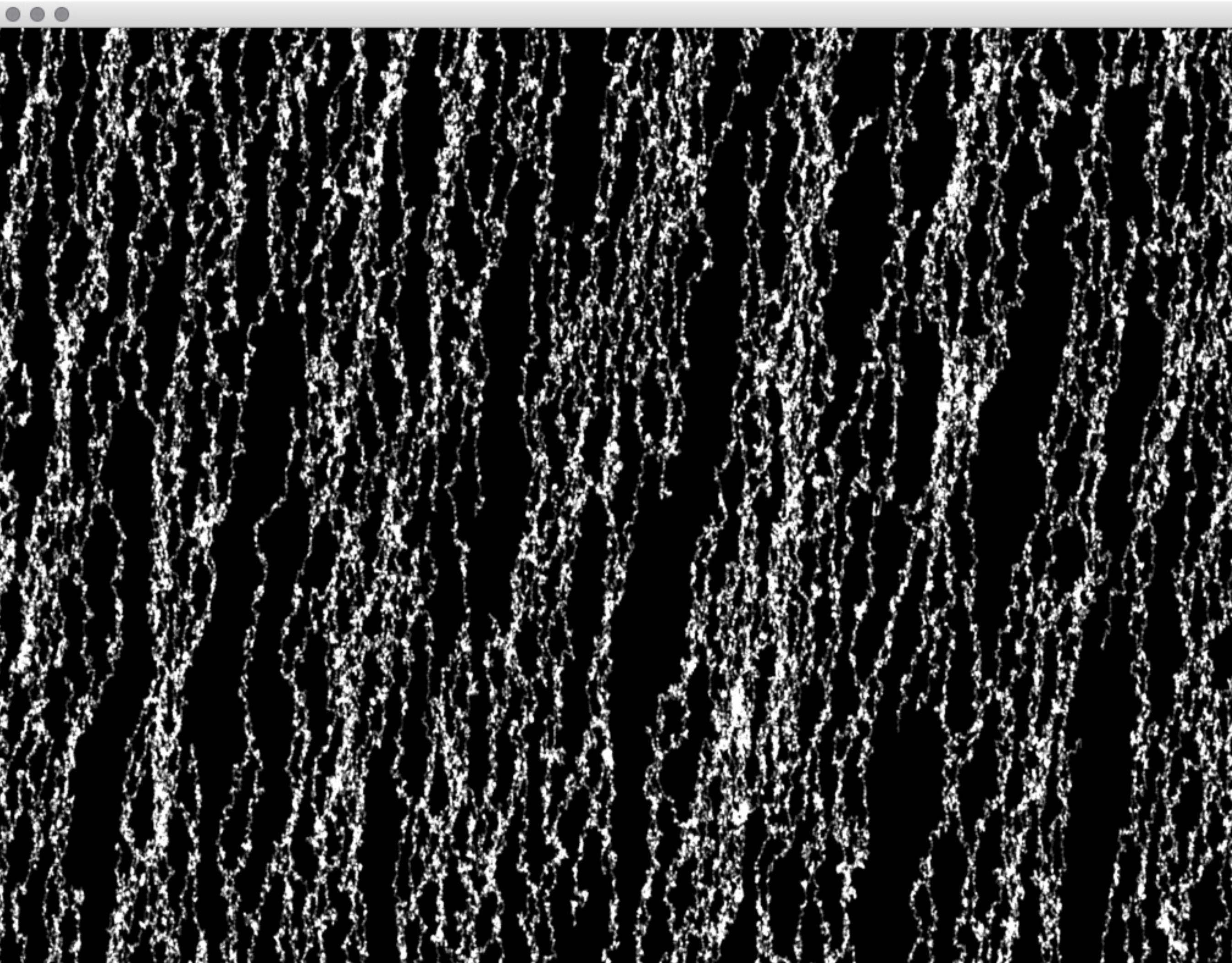
クラスを作る

- ▶ 例: 左 1、右 1.1、上 1、下 1.05



クラスを作る

- ▶ 例: 左 0.5、右0.51、上 1.0、下 0.9



ランダムウォーク応用

ランダムウォーク応用

- ▶ このRandomWalkerを応用してみる
- ▶ サンプルをみながら、解説します

ゆっくりとフェードアウト

- ▶ ゆっくりとフェードアウトするように
- ▶ 毎回画面更新の際に、ごく薄い透明度の黒い四角を描く
- ▶ 薄いセロファンを重ねていくようなイメージ

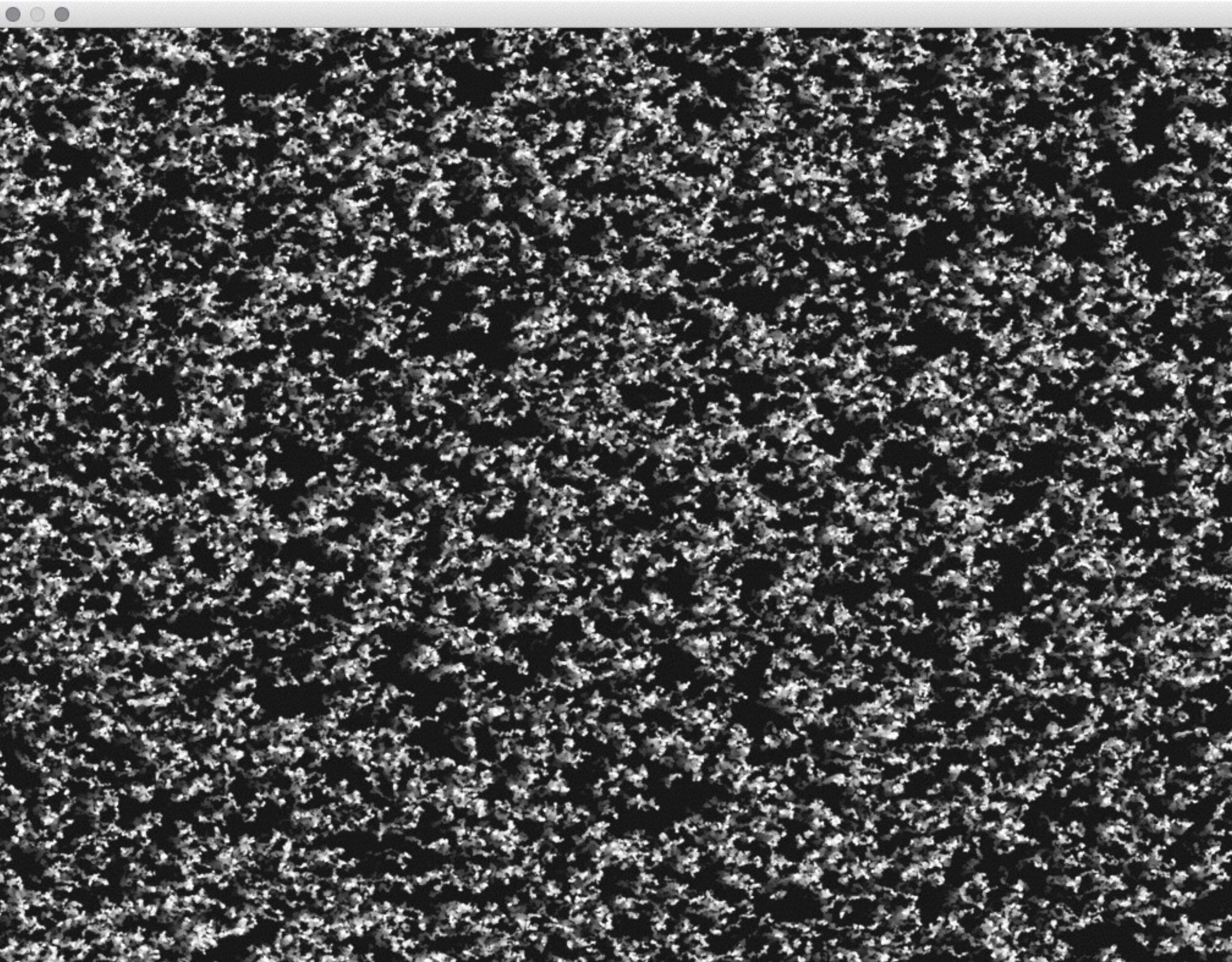
ゆっくりとフェードアウト

- ▶ ofApp - draw() を変更

```
void ofApp::draw(){  
  
    // フェードアウト  
    ofEnableBlendMode(OF_BLENDMODE_ALPHA);  
    ofSetColor(0, 0, 0, 3);  
    ofRect(0, 0, ofGetWidth(), ofGetHeight());  
  
    // 色を設定  
    ofEnableBlendMode(OF_BLENDMODE_ADD);  
    ofSetColor(255, 255, 255, 63);  
  
    // RandomWalkerを1000個表示  
    for (int i = 0; i < 1000; i++) {  
        walker[i].draw();  
    }  
}
```

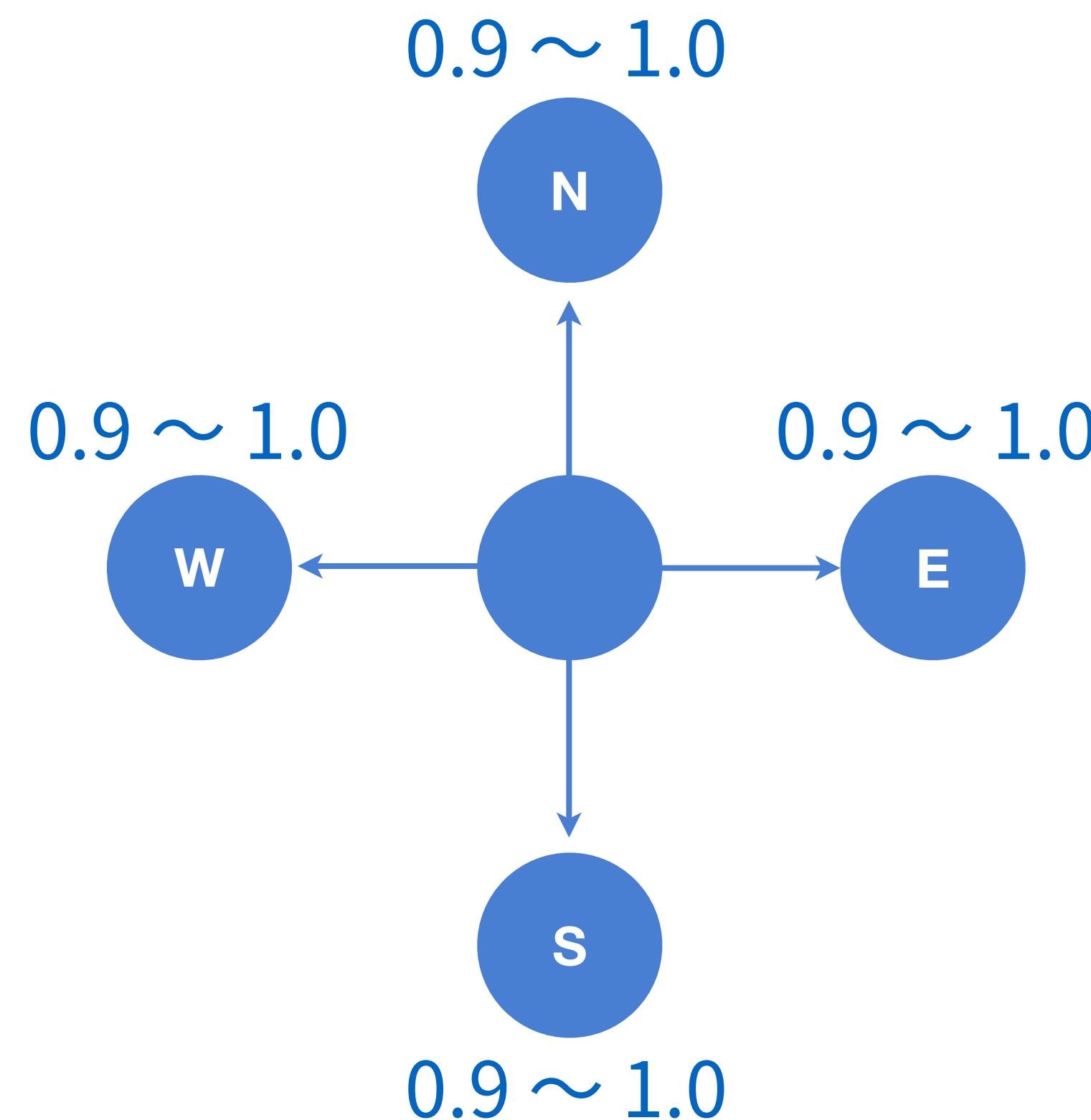
ゆっくりとフェードアウト

- ▶ 徐々にフェードアウトしていく



移動確率をランダムに

- ▶ 上下左右に移動する確率をランダムにしてみる
- ▶ ランダムな確率のランダムウォークを、1つの点から始めるとどうなるか？



移動確率をランダムに

▶ RandomWalker.h

```
#pragma once
#include "ofMain.h"

class RandomWalker {
public:
    RandomWalker();
    void draw();

    ofVec2f position;
    float left, right, top, bottom; ← 追加
};
```

移動確率をランダムに

▶ RandomWalker.cpp

```
#include "RandomWalker.h"

RandomWalker::RandomWalker(){

    //初期位置を画面中心に
    position.x = ofGetWidth()/2.0;
    position.y = ofGetHeight()/2.0; ← 変更

    //移動確率をランダムに
    left = ofRandom(0.9, 1.0);
    right = ofRandom(0.9, 1.0);
    top = ofRandom(0.9, 1.0);
    bottom = ofRandom(0.9, 1.0);
}
```

← 追加

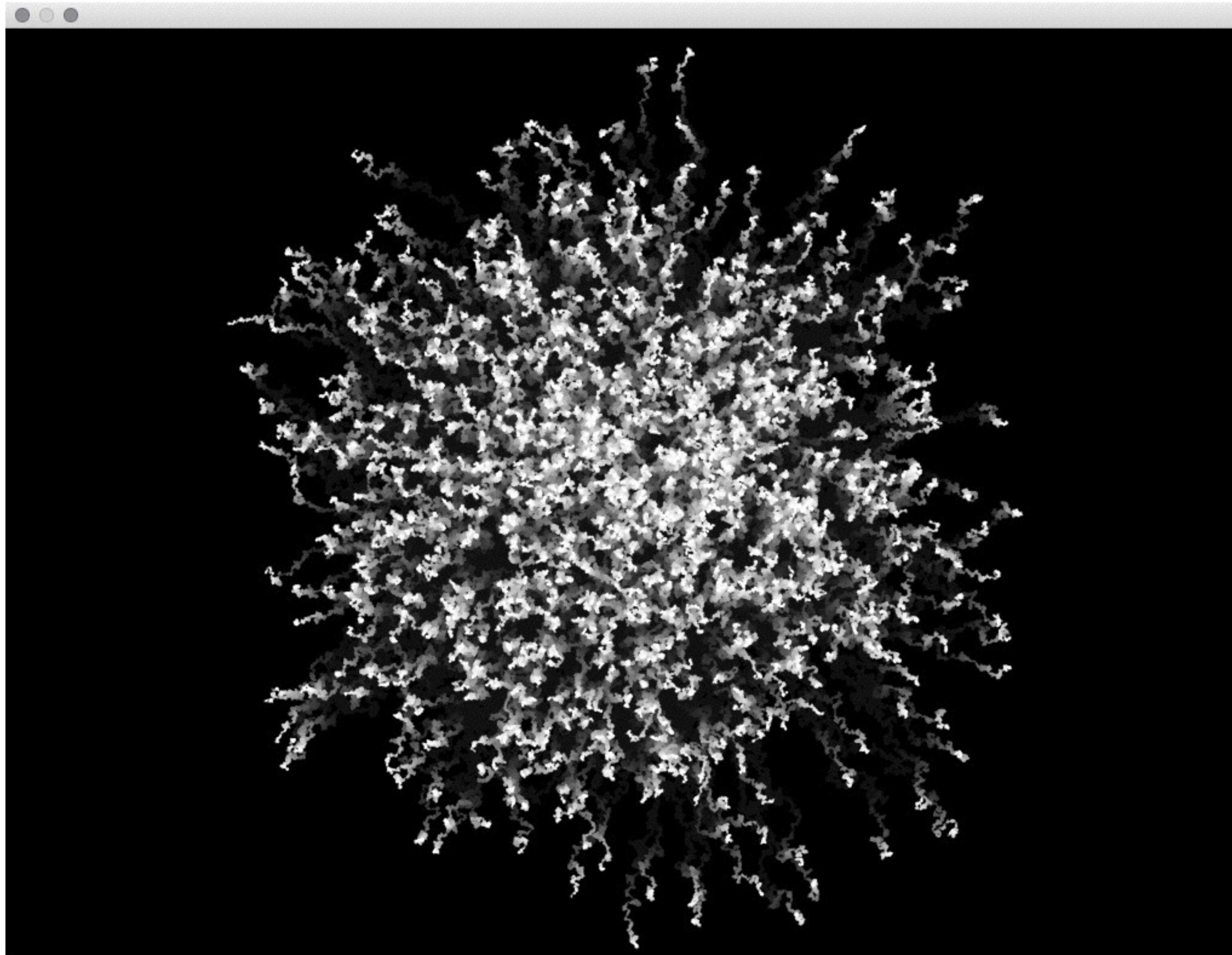
移動確率をランダムに

▶ RandomWalker.cpp

```
void RandomWalker::draw(){
    // 10回くりかえし
    for (int i = 0; i < 10; i++) {
        //上下左右同じ確率でランダムに移動
        position.x += ofRandom(-left, right);
        position.y += ofRandom(-top, bottom); ← 変更
        ...
        ofCircle(position.x, position.y, 1);
    }
}
```

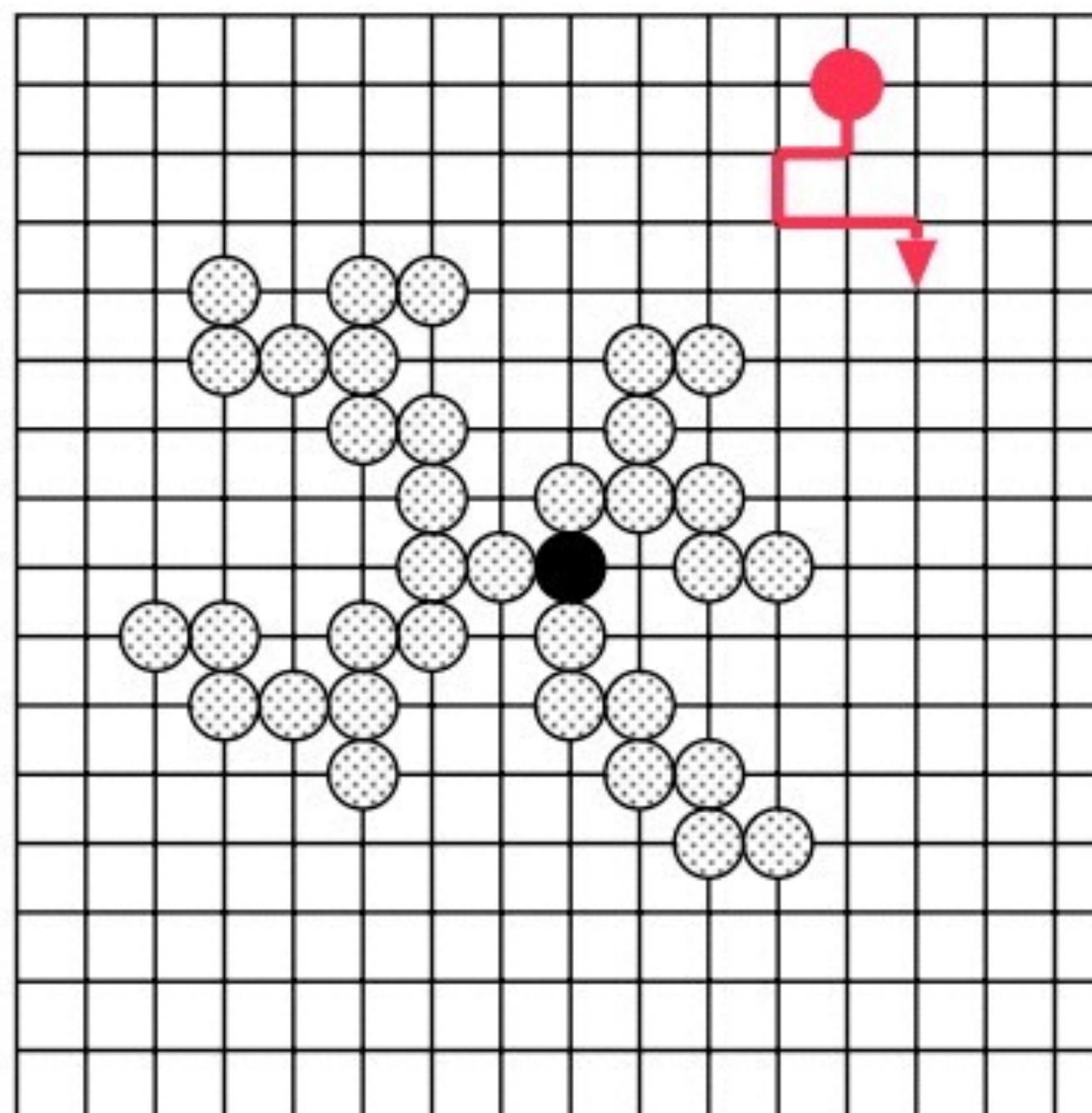
移動確率をランダムに

- ▶ 画面中央から、湧き出てくる



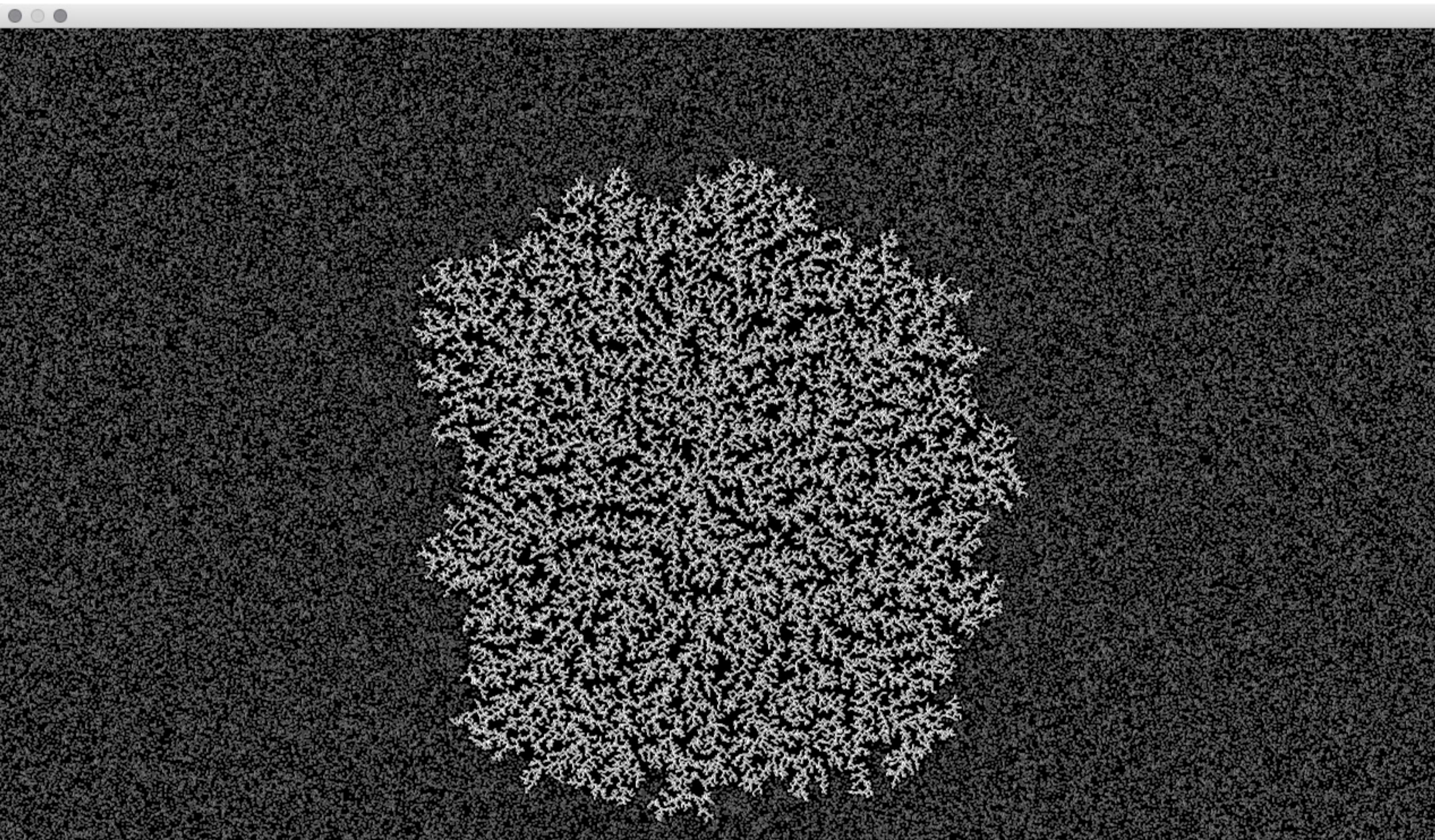
Diffusion-limited aggregation

- ▶ ランダムウォークを利用した興味深い現象を紹介
- ▶ Diffusion-limited aggregation (DLA)
- ▶ 拡散に支配された凝集過程
- ▶ 中心に置かれた粒子の周囲を、ランダムウォークしながらやって来る粒子が付着する



Diffusion-limited aggregation

- ▶ 氷の結晶のようなパターンが生成される!!



Diffusion-limited aggregation

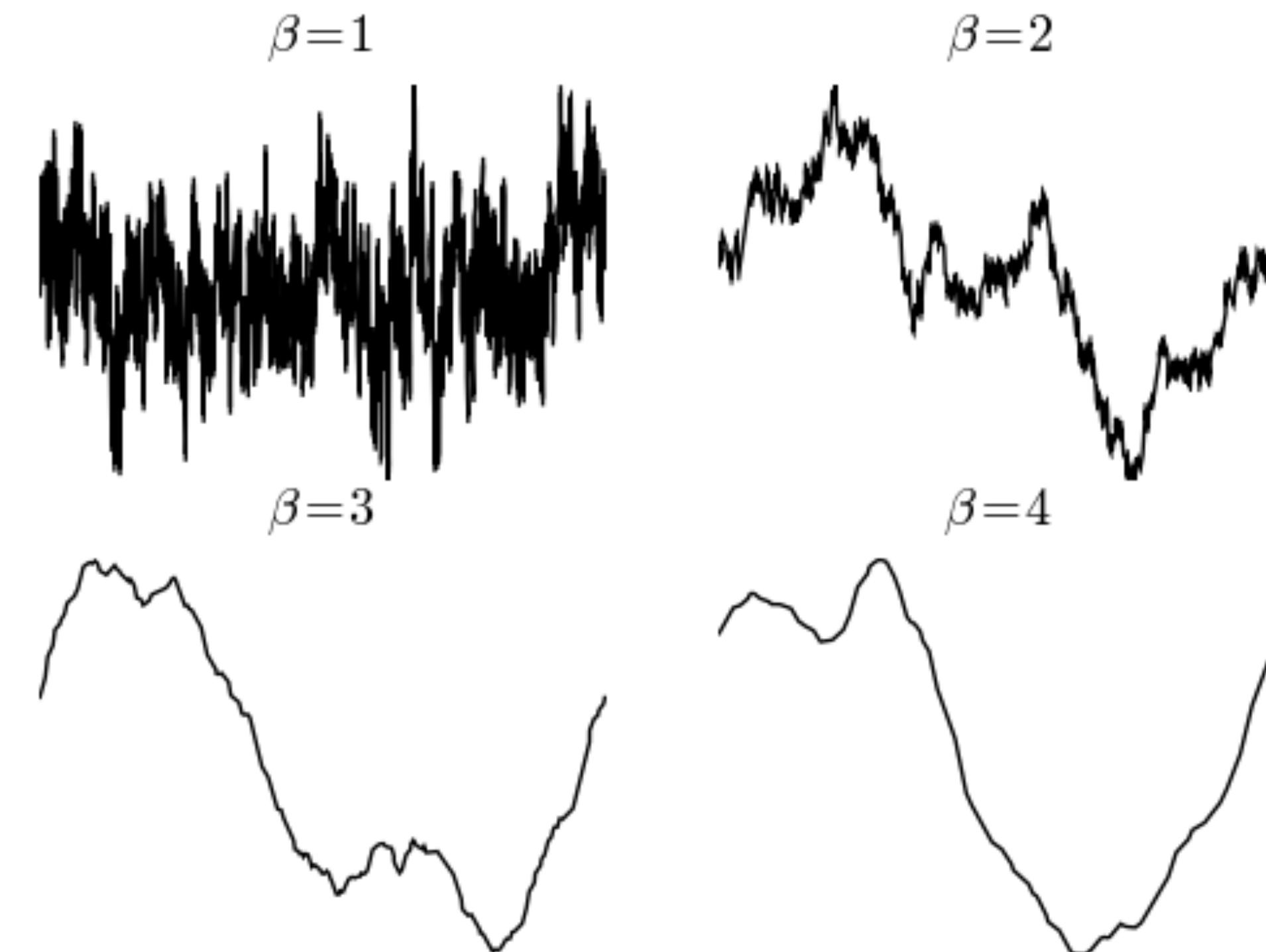
- ▶ 稲妻の放電パターンや樹氷の成長などと類似
- ▶ ランダムな運動のくりかえしの中から生成される秩序
- ▶ Lichtenberg figures
- ▶ <http://www.youtube.com/watch?v=FWOst4VwwEU>



Perlin noise による運動

Perlin noiseによる運動

- ▶ ランダムのもう少し高度な活用
- ▶ Perlin noise - ランダムな値の変化が急激にならないよう、スムースに補完されたランダム関数
- ▶ Ken Perlinにより開発



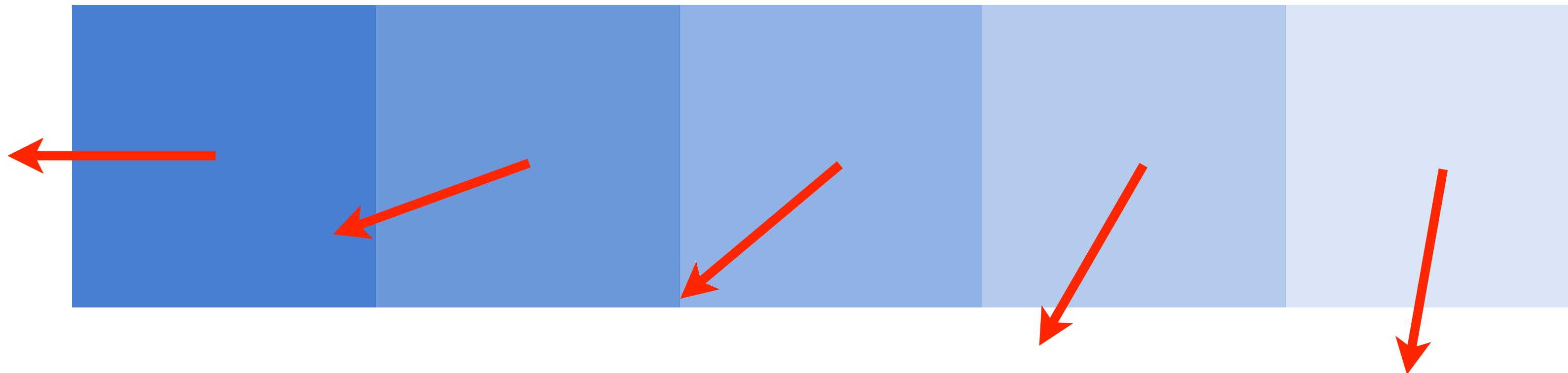
Perlin noiseによる運動

- ▶ Perlin noiseで平面の濃淡を描く



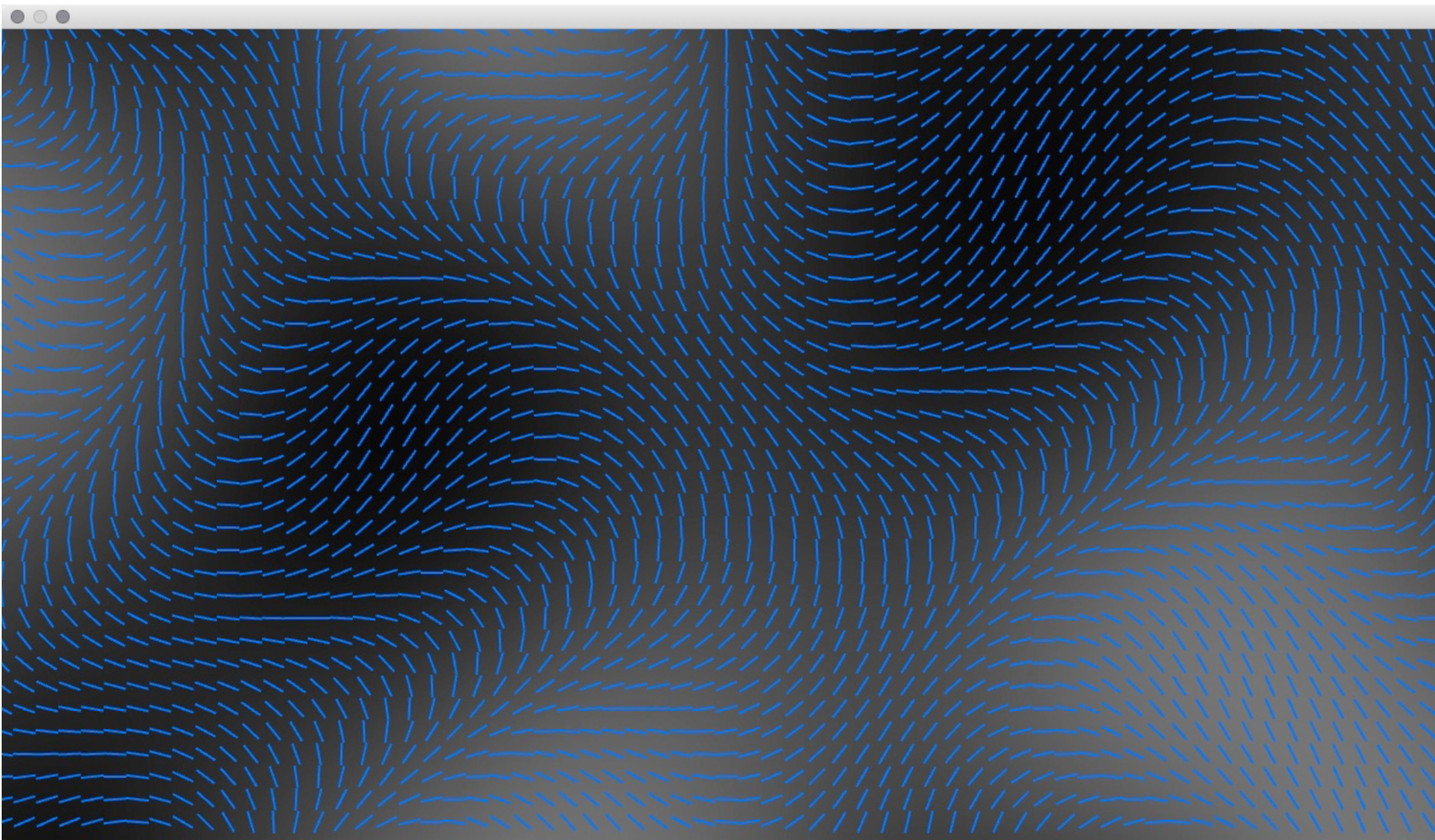
Perlin noiseによる運動

- ▶ 2D平面状のPerlin noiseの値(濃淡)を、角度に変換したらどうなるだろうか？



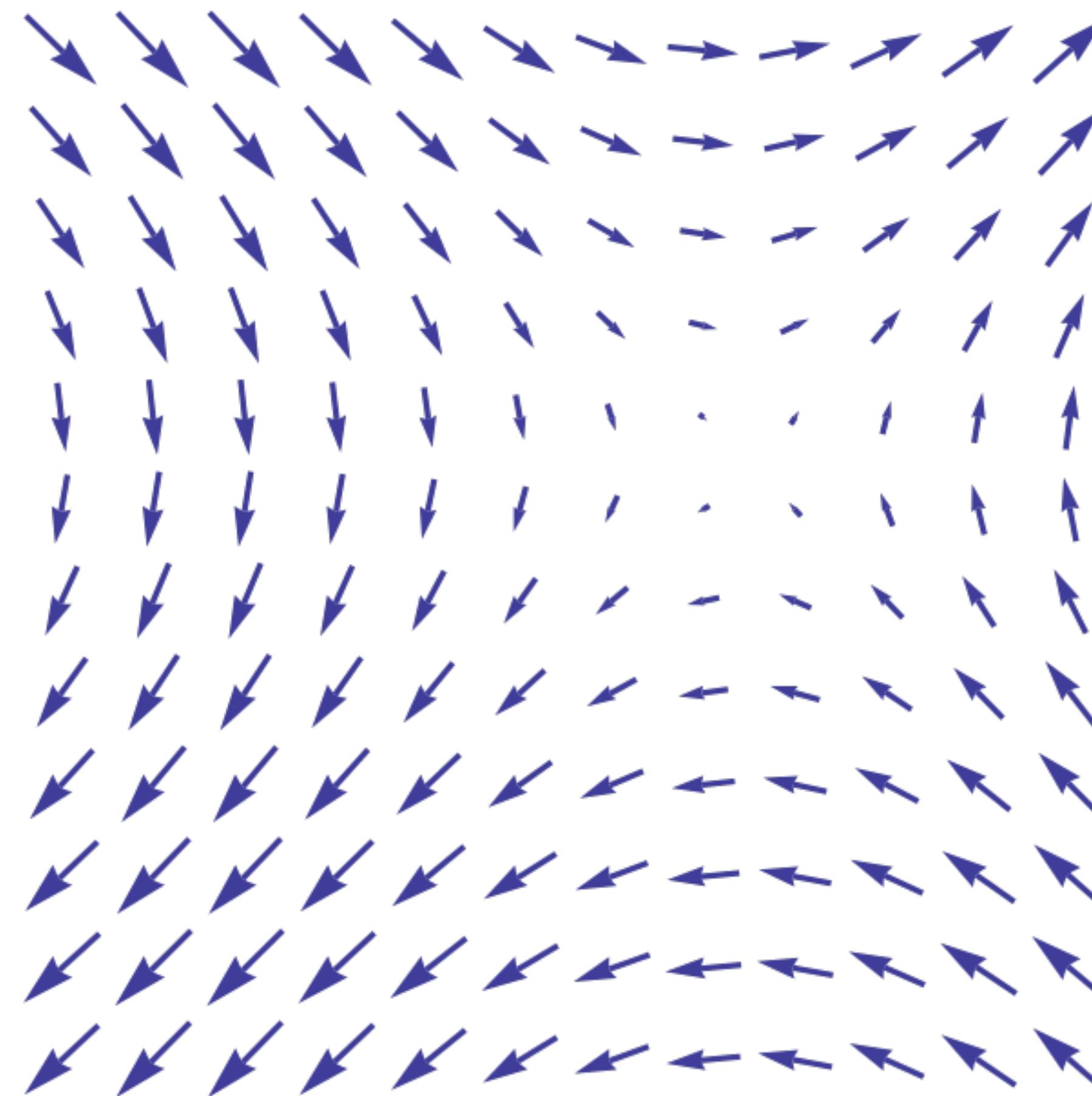
Perlin noiseによる運動

- ▶ 角度に変換されたPerlin noise



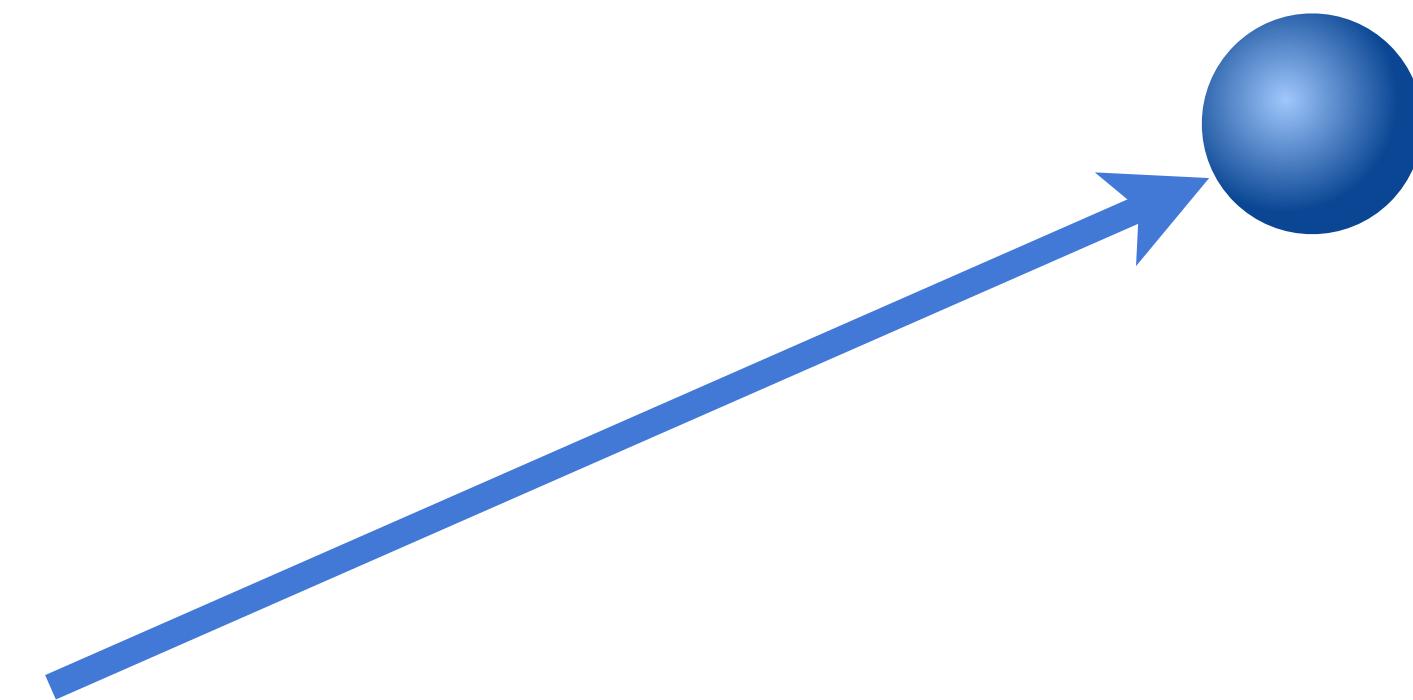
Perlin noiseによる運動

- ▶ Perlin noiseをベクトル場としてとらえる
- ▶ 向きと方向をもった力が、グリッド上に整列している



Perlin noiseによる運動

- ▶ ベクトル場の上に粒(Perticle)を配置するとどうなるか?
- ▶ ベクトル場にはたらく力を計算してPerticleを運動させてみる



Perlin noise による運動

- ▶ Perlin noise による運動が実現!!



Perlin noise による運動

- ▶ パラメータの調整で、様々なパターンに変化する

