Kotlinで愉(たの)しむクリエイティブコーディング

畠山 創太

Kotlin Fest 2024
JKUG Presents

学生にプログラミングの授業

https://prtimes.jp/main/html/rd/p/000000237.000005362.html

プログラミングの学習は
楽しくなければならない

Processing

Processing Foundation

Processing

p5.js

Processing Android

Processing Python

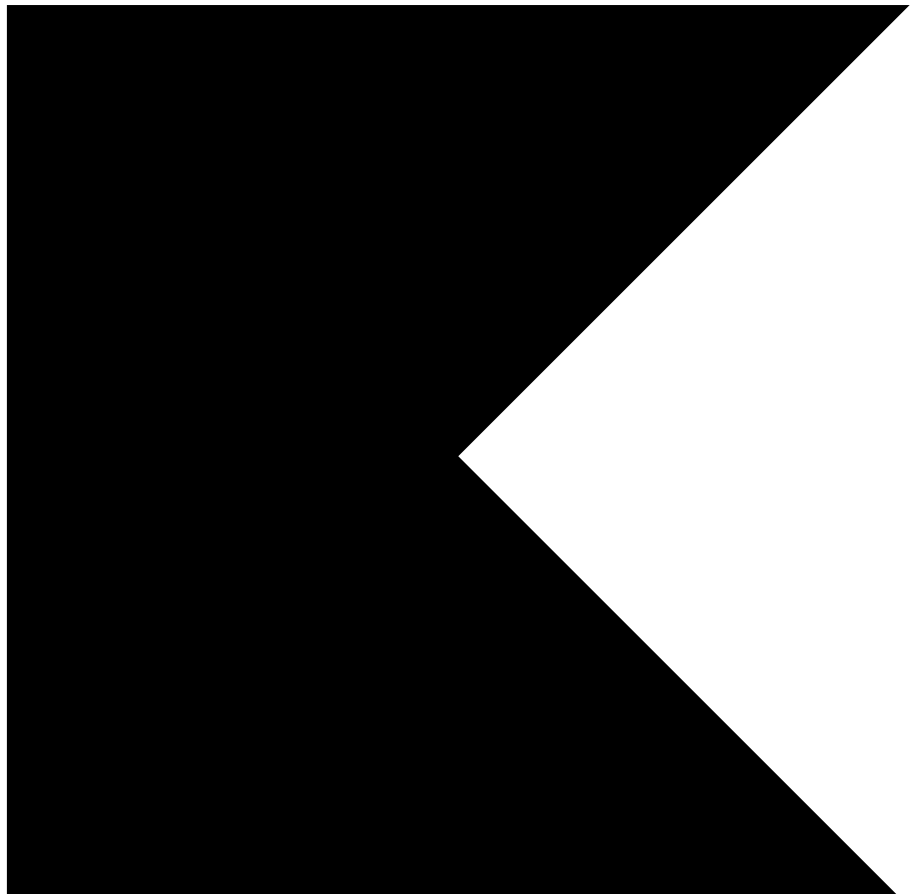
クリエイティブコーディング🤔

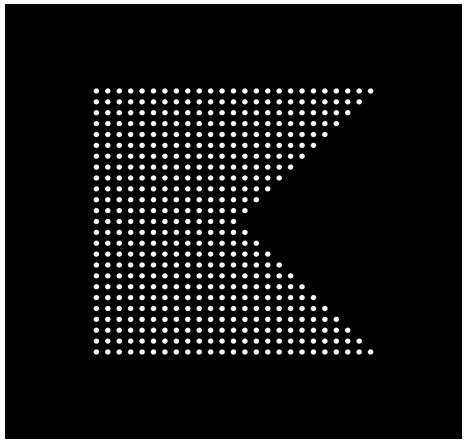
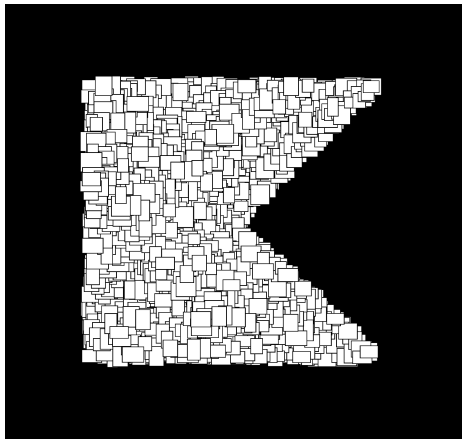
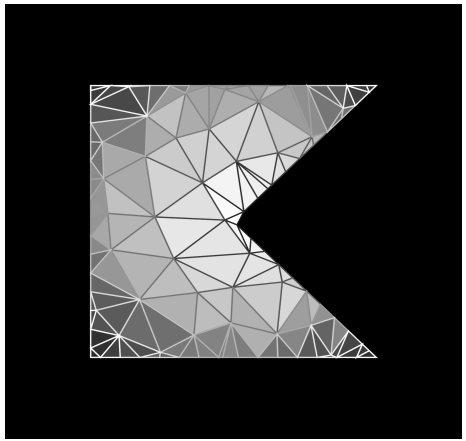
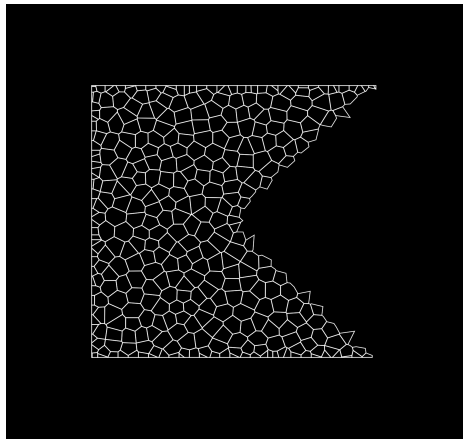
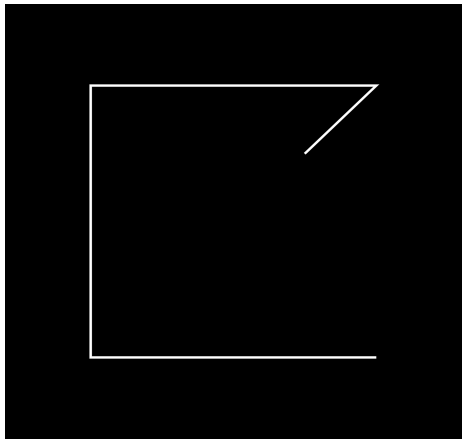
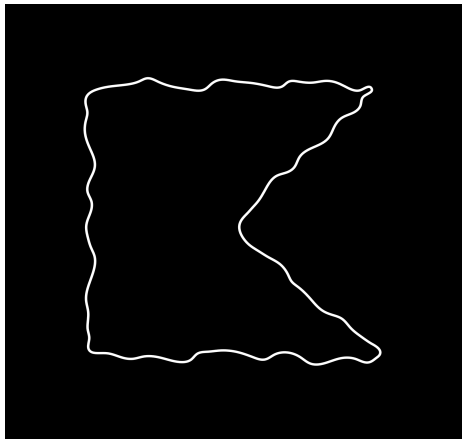
コードで

- 絵を描いたり
- 音楽や映像
- インタラクティブメディア

などの作品を生み出す活動や、その方法のこと

コードで絵を描く



クリエイティブコーディングのいいところ

- ルールが無く、自由。誰でも簡単に始められる
- コードを書いてアイデアを具現化する習慣
- 「ひらめき」と「実験」を繰り返す楽しさ

×

クリエイティブコーディング

OPENRNDR

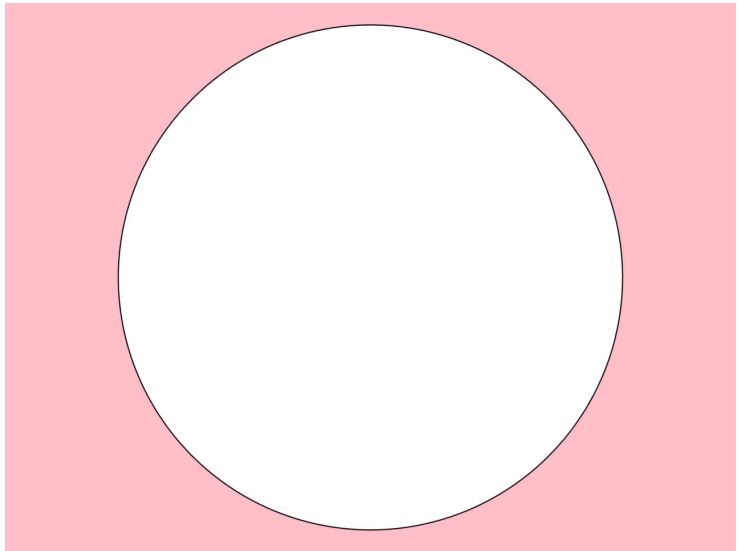
OPENRNDRのはじめ方

1. IntelliJ IDEAをインストール
2. OPENRNDR IntelliJプラグインをインストール
3. `openrndr/openrndr-template` でプロジェクト作成
4. 実行 🚀

https://guide.openrndr.org/setUpYourFirstProgram.html

```
fun main() = application {
  program {
    extend {
      drawer.clear(ColorRGBa.PINK)
      drawer.fill = ColorRGBa.WHITE

      val center = drawer.bounds.center
      val radius = abs(cos(seconds)) * height * 0.51
      drawer.circle(center, radius)
    }
  }
}
```



```
fun main() = application {
  program {
    extend {
      drawer.clear(ColorRGBa.PINK)
      drawer.fill = ColorRGBa.WHITE

      val center = drawer.bounds.center
      val radius = abs(cos(seconds)) * height * 0.51
      drawer.circle(center, radius)
    }
  }
}
```

```
fun main() = application {
  configure {
    // -- アプリケーションの設定を記述
  }

  program {
    // -- ここは一度だけ実行される
    extend {
      // -- ここは繰り返し実行される
    }
  }
}
```

ORX (OPENRNDR Extras)

https://github.com/openrndr/orx

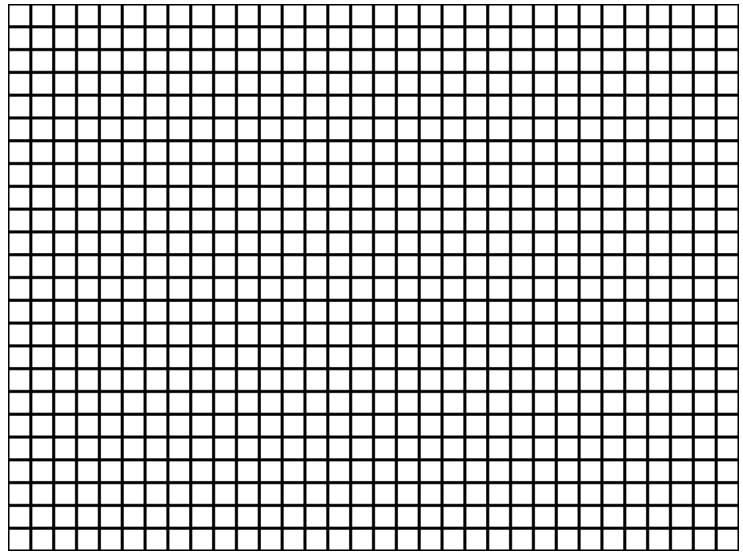
OPENRNDRをさらに豊かにする拡張ライブラリ群

Noise

ノイズ

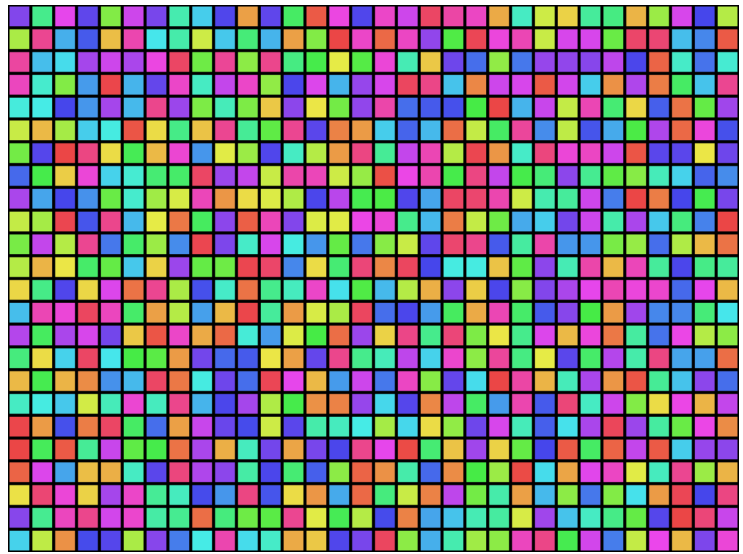
二次元格子

```
fun main() = application {
  program {
    val cells = drawer.bounds.grid(
      cellWidth = 20.0,
      cellHeight = 20.0
    )
    extend {
      drawer.rectangles(cells.flatten())
    }
  }
}
```



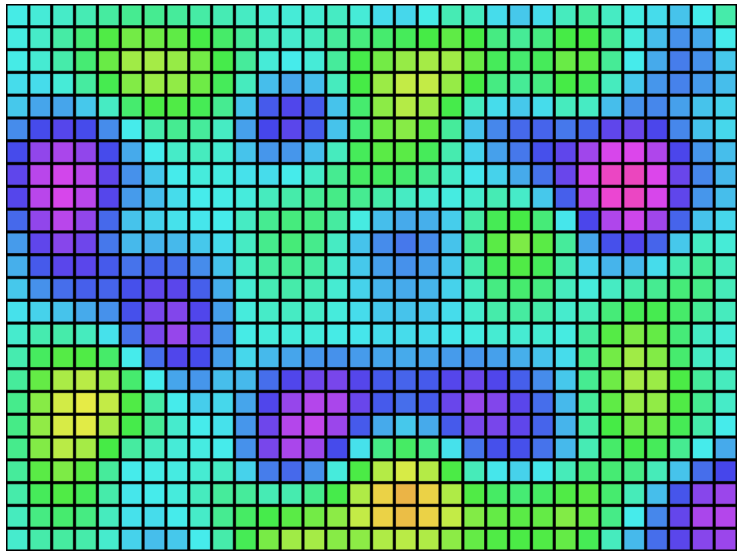
一様乱数

```
fun main() = application {
  program {
    val cells = drawer.bounds.grid(
      cellWidth = 20.0,
      cellHeight = 20.0
    )
    extend {
      Random.resetState()
      drawer.rectangles {
        cells.flatten().forEach {
          val hue = Double.uniform(0.0, 360.0, Random.rnd)
          fill = hsl(hue, 0.8, 0.6).toRGBa()
          rectangle(it)
        }
      }
    }
  }
}
```



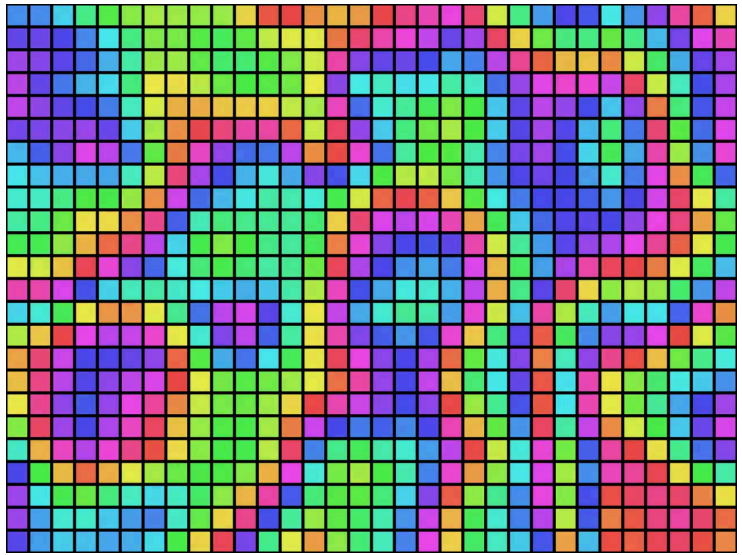
パーリンノイズ

```
fun main() = application {
  program {
    val cells = drawer.bounds.grid(
      cellWidth = 20.0,
      cellHeight = 20.0
    )
    extend {
      drawer.rectangles {
        cells.flatten().forEach {
          val p = it.center * 0.01
          val n = perlin(42, p) * 0.5 + 0.5
          val hue = 360.0 * n
          fill = hsl(hue, 0.8, 0.6).toRGBa()
          rectangle(it)
        }
      }
    }
  }
}
```

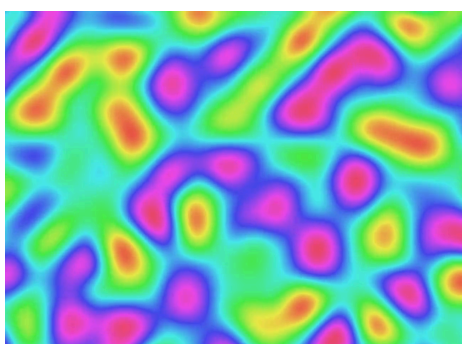


パーリンノイズ（時間発展）

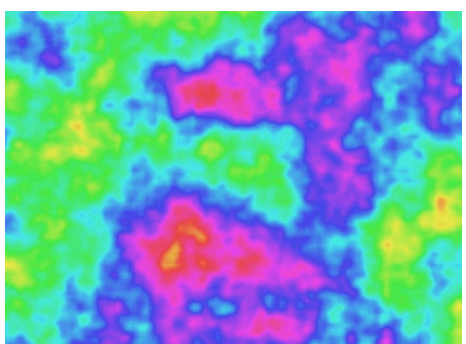
```
fun main() = application {
  program {
    val cells = drawer.bounds.grid(
      cellWidth = 20.0,
      cellHeight = 20.0
    )
    extend {
      drawer.rectangles {
        cells.flatten().forEach {
          val p = it.center * 0.01
          val time = seconds * 0.8
          val n = perlin(42, p.x, p.y, time) * 0.5 + 0.5
          val hue = 360.0 * n
          fill = hsl(hue, 0.8, 0.6).toRGBa()
          rectangle(it)
        }
      }
    }
  }
}
```



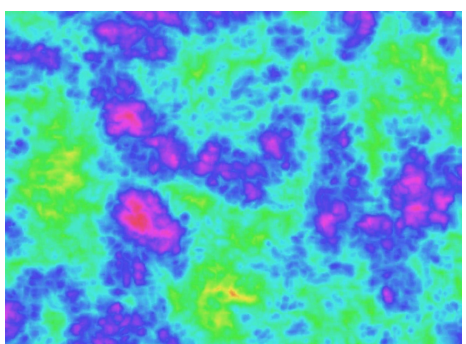
simplex()



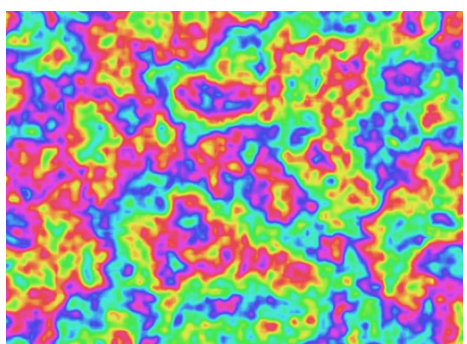
fbm()



rigid()



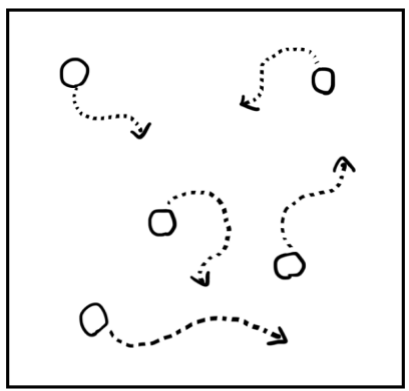
billow()



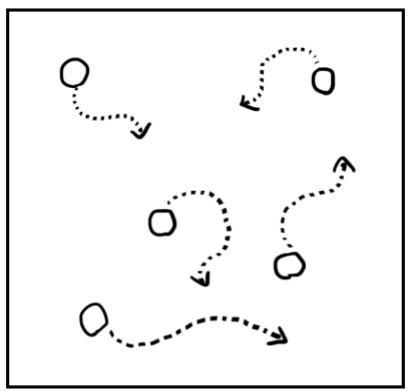
Simulation

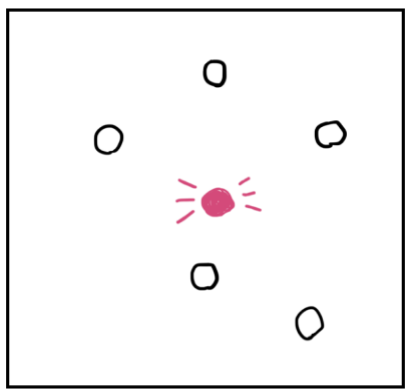
シミュレーション

拡散律速凝集（DLA）
かくさんりっそくぎょうしゅう

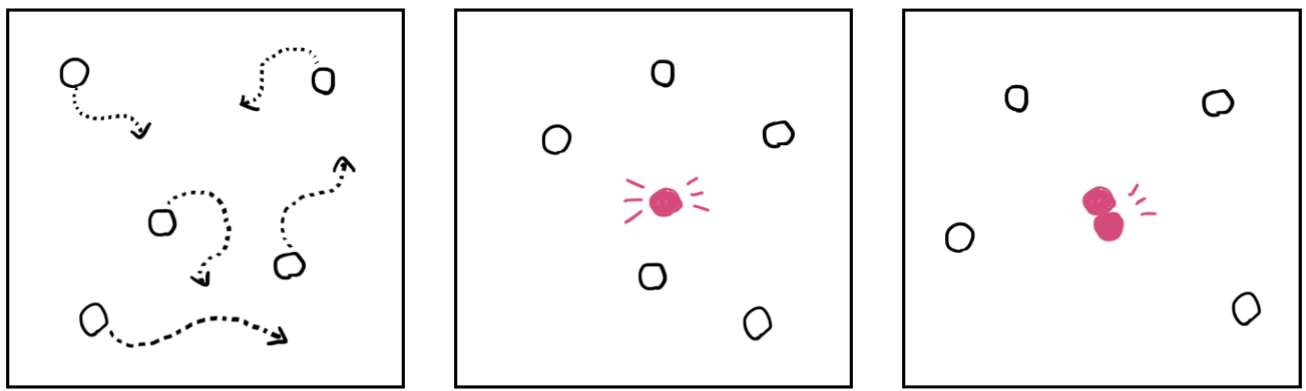


フィールド上をランダムに動き回るたくさんの粒子がある

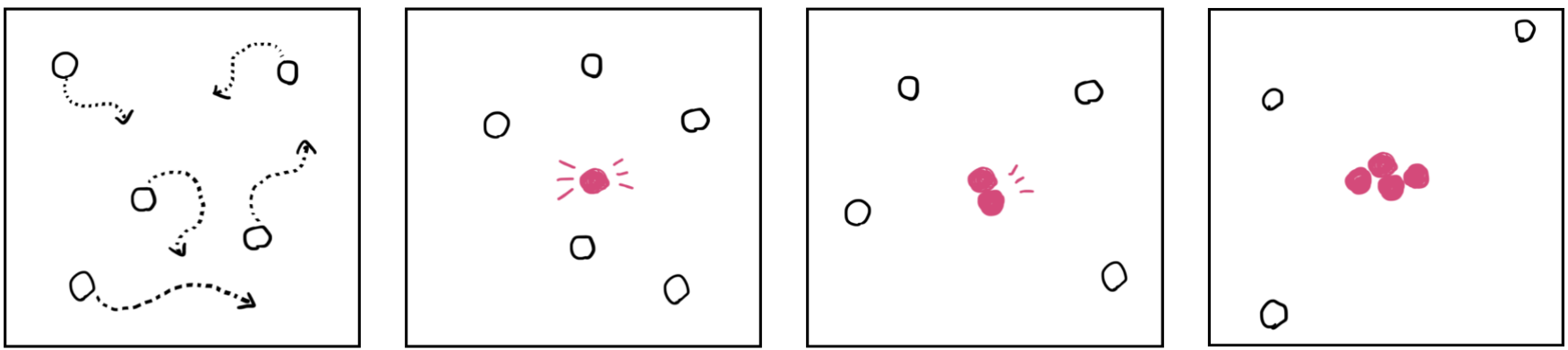




真ん中に核を配置する。



粒子は核にぶつかると吸着して核を拡張する。



時間経過とともに核が成長していく。

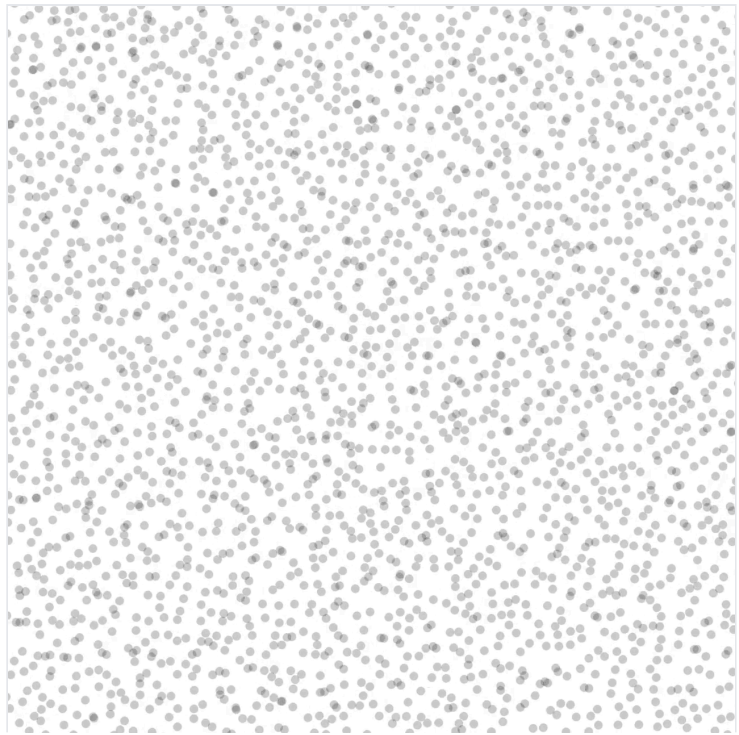
粒子を描く

```
fun main() = application {
  program {
    val points = drawer.bounds.scatter(5.0)
    extend {
      drawer.clear(ColorRGBa.WHITE)
      drawer.stroke = null
      drawer.fill = ColorRGBa.BLACK.opacify(0.2)
      drawer.circles(points, 3.0)
    }
  }
}
```

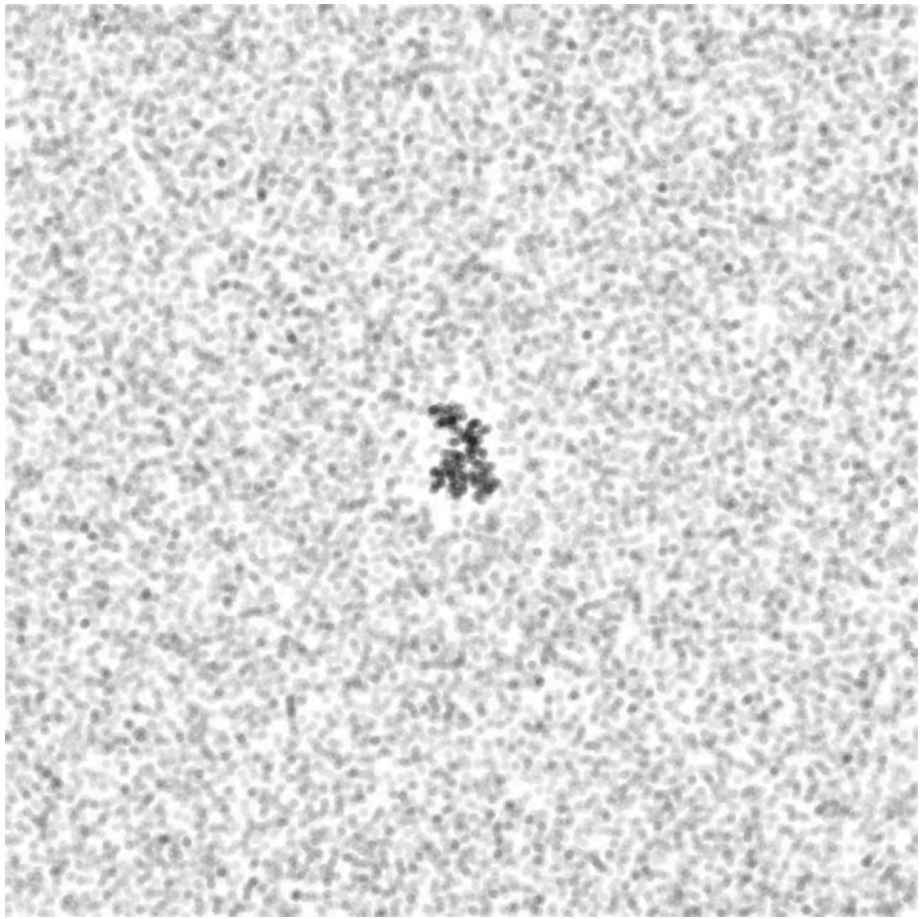
ランダムに動かす（ブラウン運動）

```
fun main() = application {
  program {
    var points = drawer.bounds.scatter(5.0)
    val step = 1.0
    extend {
      drawer.clear(ColorRGBa.WHITE)
      drawer.stroke = null
      drawer.fill = ColorRGBa.BLACK.opacify(0.2)
      drawer.circles(points, 3.0)

      points = points.map {
        val u = Vector2.uniform(-1.0, 1.0)
        val velocity = u.normalized * step
        it + velocity
      }
    }
  }
}
```



真ん中に核を配置してシミュレーション（20倍速）



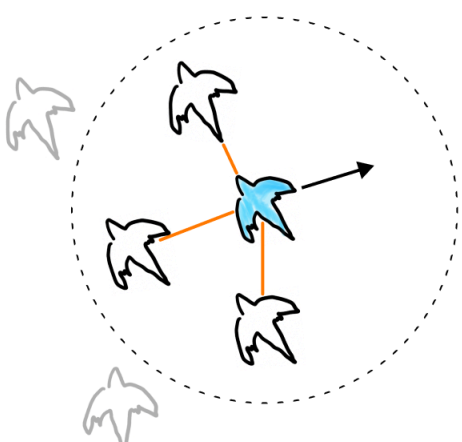

Kotlin Fest 2024
JKUG Presents

Boids

鳥の群れをモデル化する

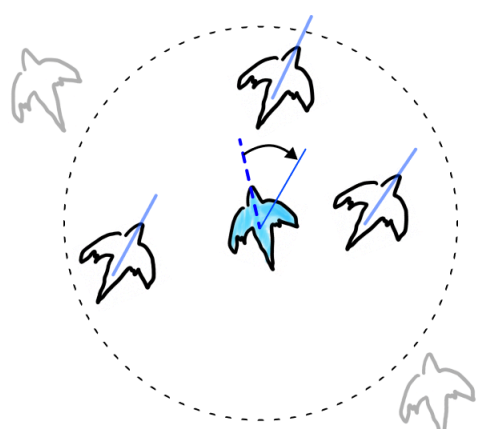
① 分離

近隣の鳥から遠のく



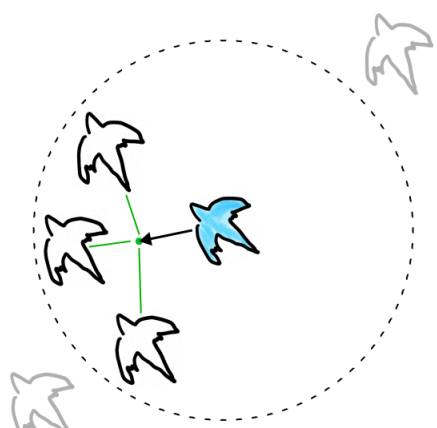
② 整列

近隣の鳥と同じ方を向く

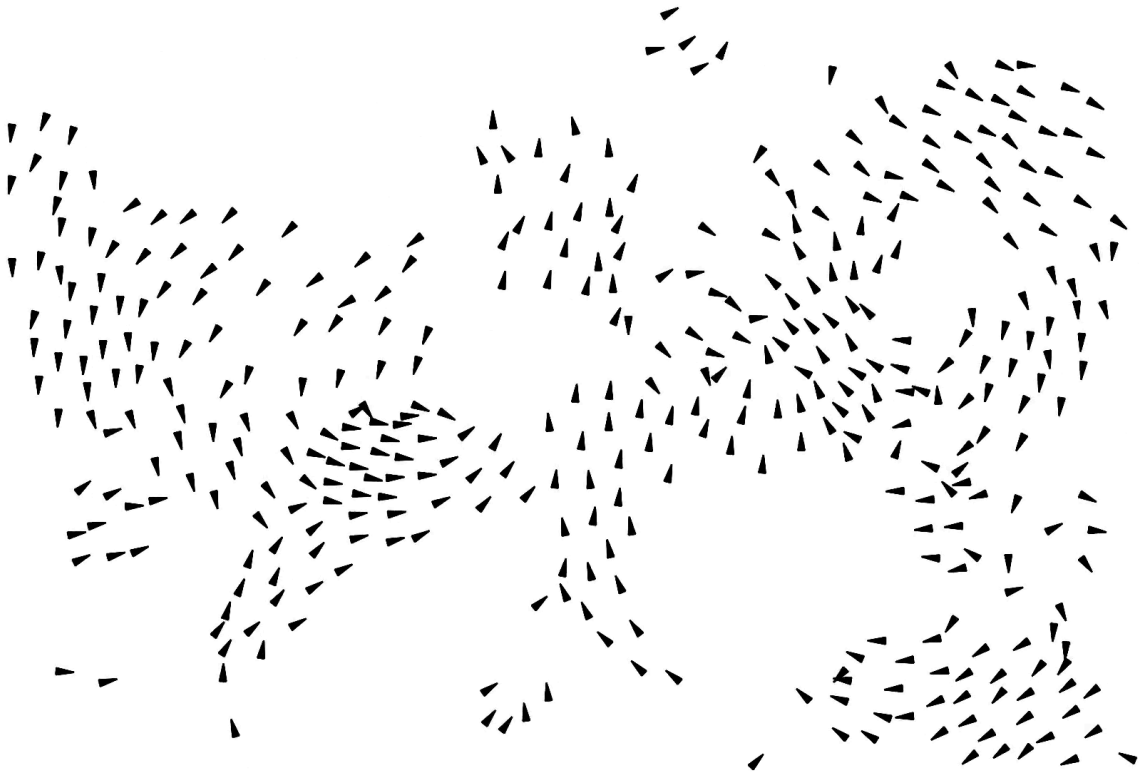


③ 結合

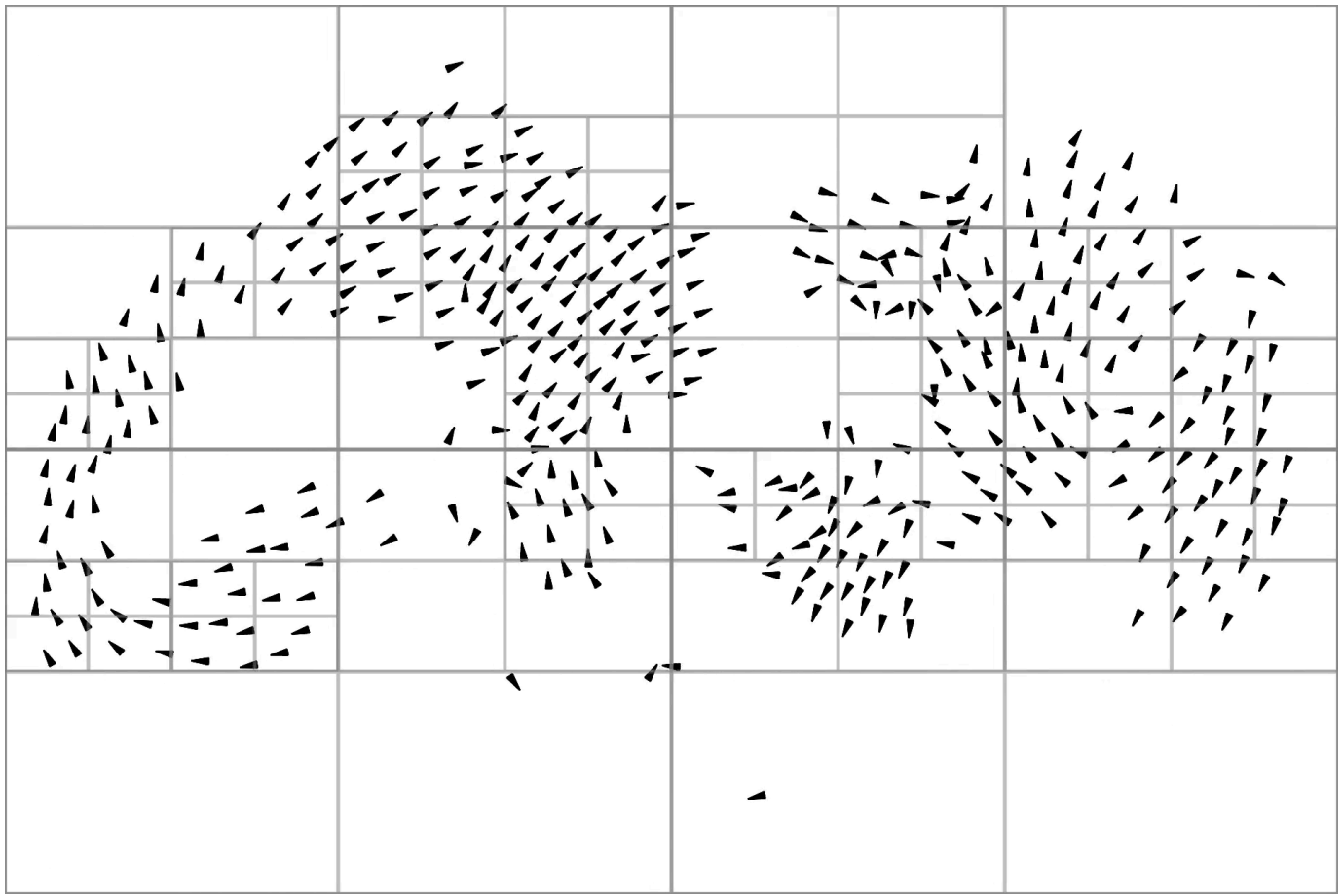
近隣の鳥に合流する



Boidsのシミュレーション

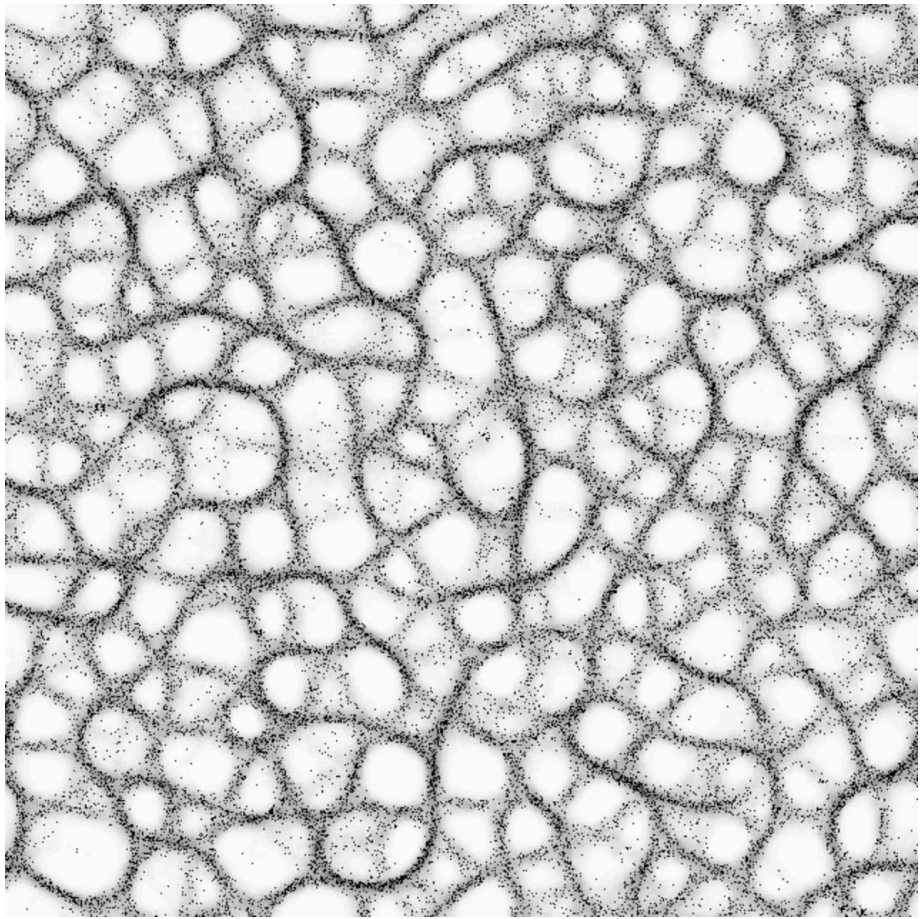


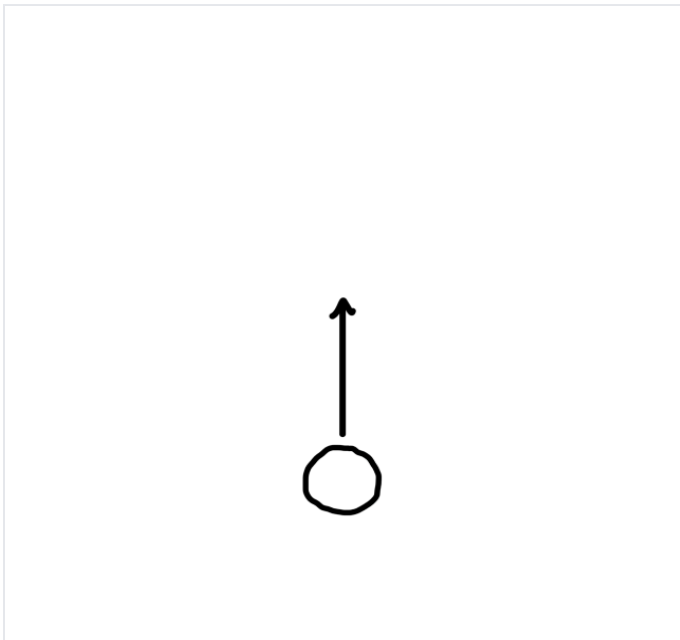
orx-quadtree



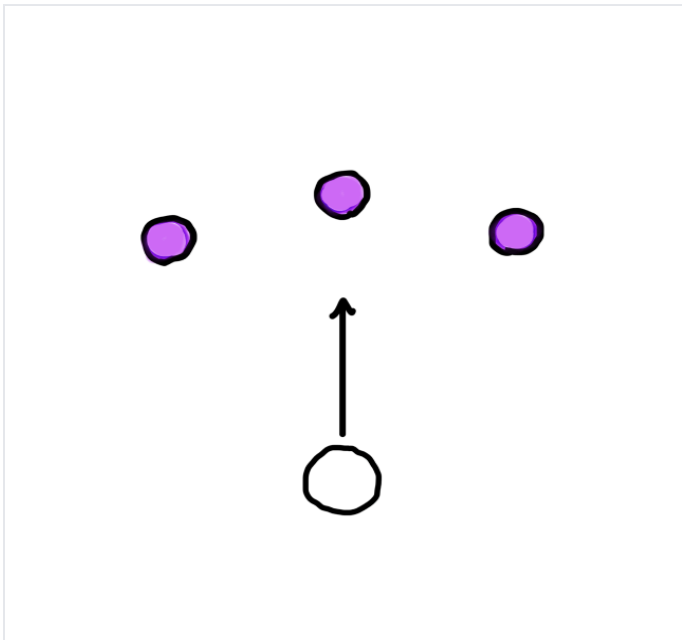
Physarum

Physarumのシミュレーション

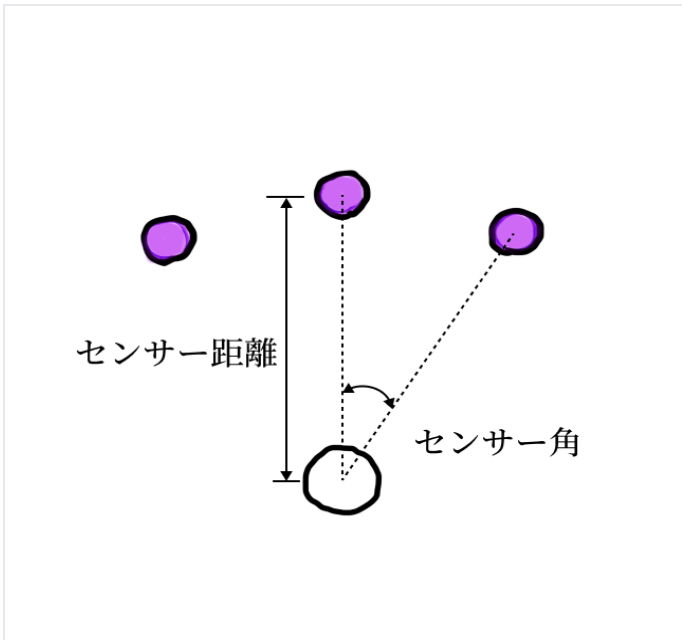




- 粒子は「位置」と「向き」を持っている

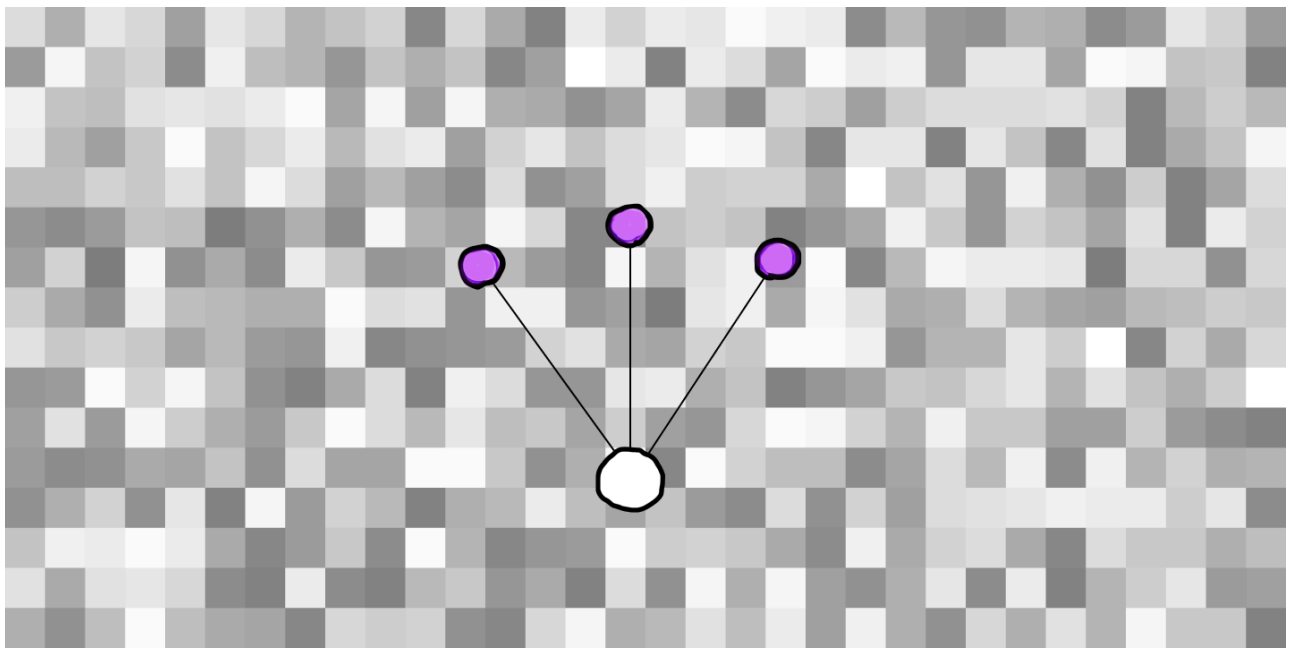


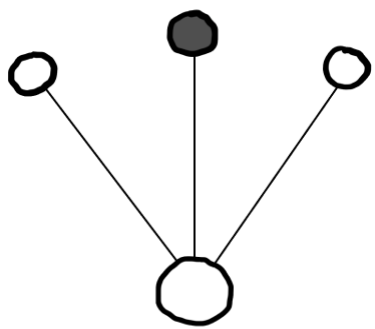
- 粒子は「位置」と「向き」を持っている
- 正面には3つの「センサー」がある



- 粒子は「位置」と「向き」を持っている
- 正面には3つの「センサー」がある
- センサー距離、センサー角はパラメータ

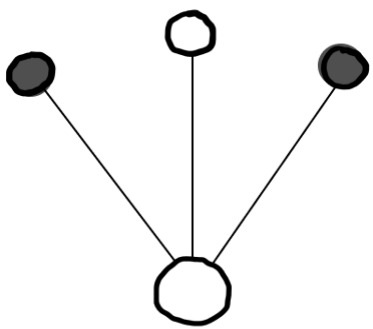
センサーはフィールドを検知





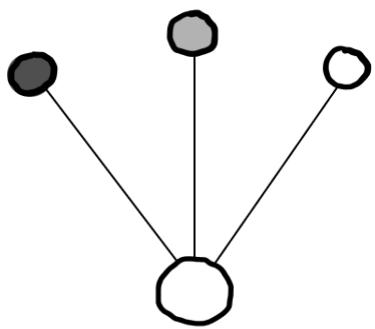
「前 > 左」 且つ 「前 > 右」の場合

→ **方向を変えない**



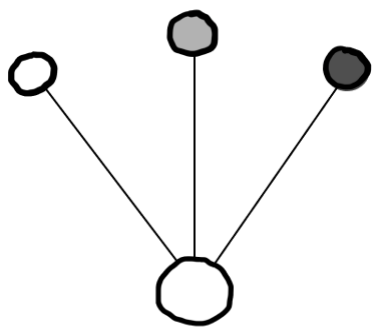
「前＜左」 且つ 「前＜右」の場合

→ 右か左をランダムに向く



「左 > 前 > 右」の場合

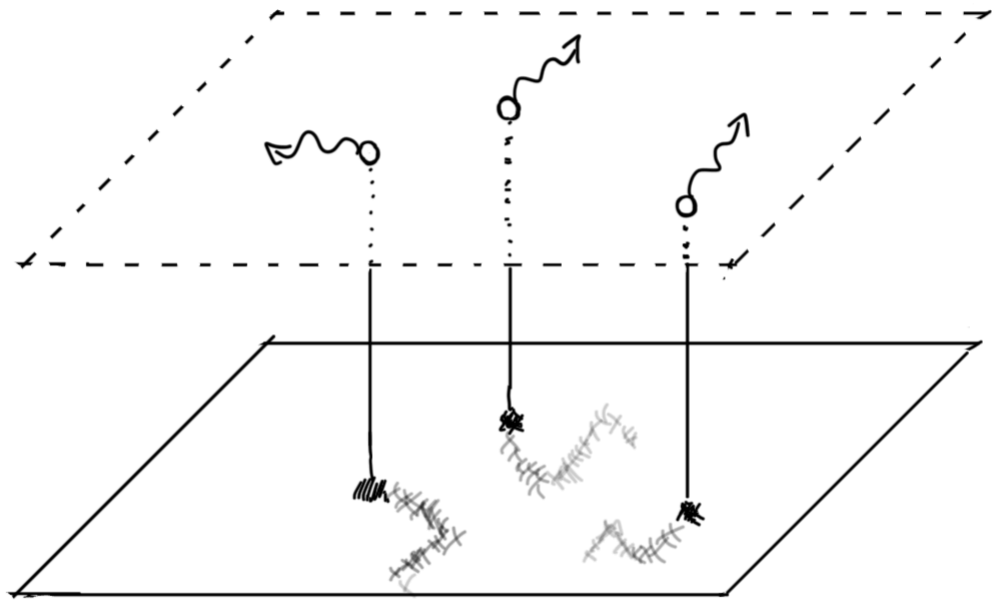
→ **左を向く**



「左 < 前 < 右」の場合

→ **右を向く**

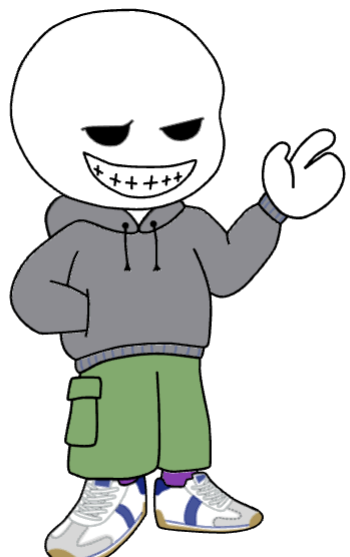
粒子が動き回る



軌跡がフィールドに残る

デモ

ありがとうございました！



@chooblarin