第2部　人工社会の発想と技法に慣れる

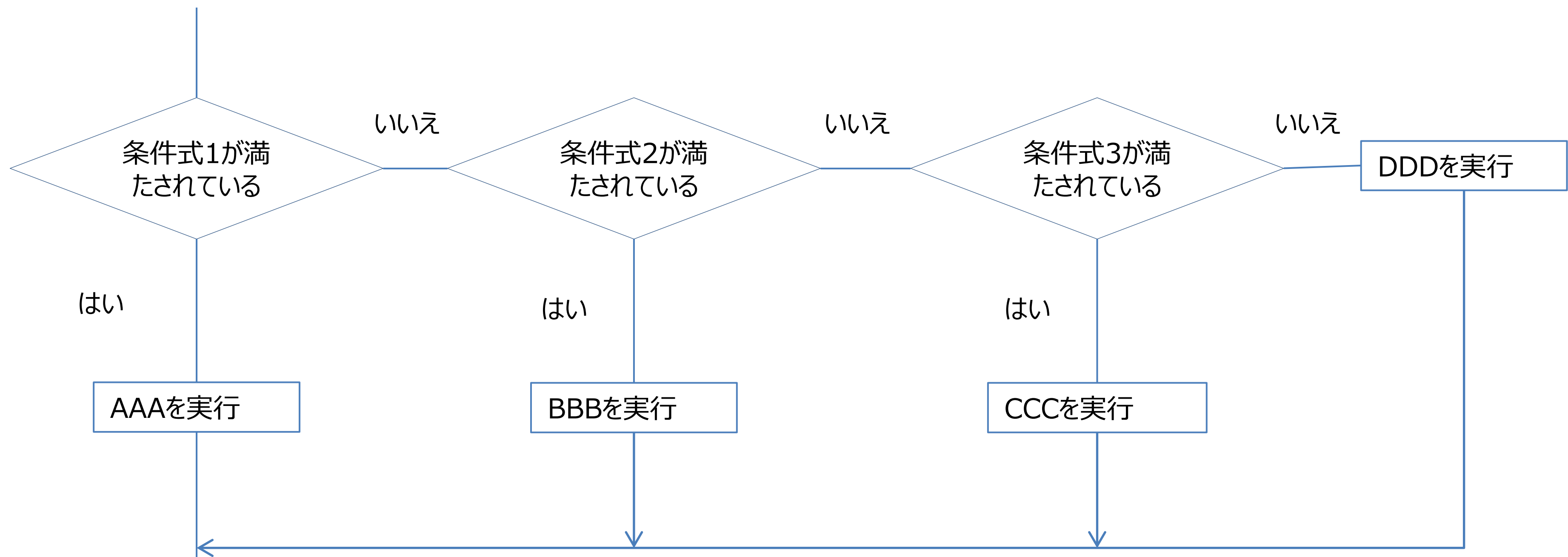
第11章：状況に応じた行動の選択肢を増やす

❒11.0 if文の使い方をマスターしよう

- 複雑な場合分けをルール化するためには、if文の構造を複雑にします。
- フローチャートをきちんと書くことが重要です。
- ルールを見やすく書くことも大事です。
- 混雑する駅で人の流れがスムーズになる現象をモデル化しましょう。
- 周囲の状況を調べる技巧的な方法です。

❒11.1 複雑な場合分け(1)

■ 複雑な行動をルール化するための「条件分岐」のフローチャート



❐11.1 複雑な場合分け(2)

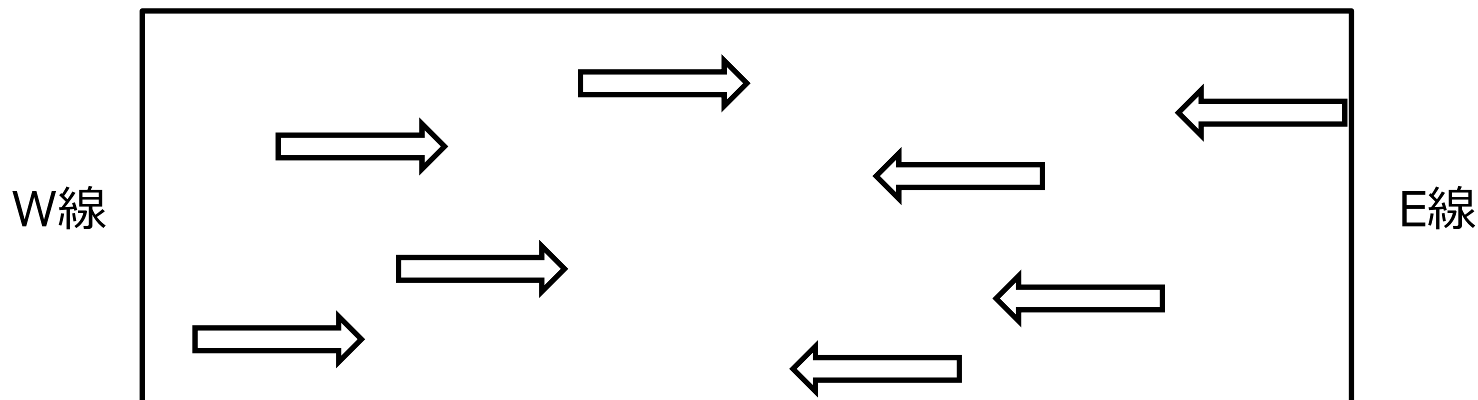
- artisoc Cloudのルール表記

```
if 条件式1:
        AAA
elif 条件式2:
        BBB
elif 条件式3:
        CCC
else:
        DDD
```

- 条件式を上から順番にチェックしていき、該当するものを実行する。
- 「elif 条件式 :」のセットはいくつあってもOK。
- AAAやBBBの実行ルールは、それ自体がif文となる、“入れ子構造”でもOK。
- 条件PとQの両方が成立している場合、どちらかが成立している場合などを論理演算子を使って複雑化することも可能。

❑11.2 ターミナル駅の通勤客の流れ(1)

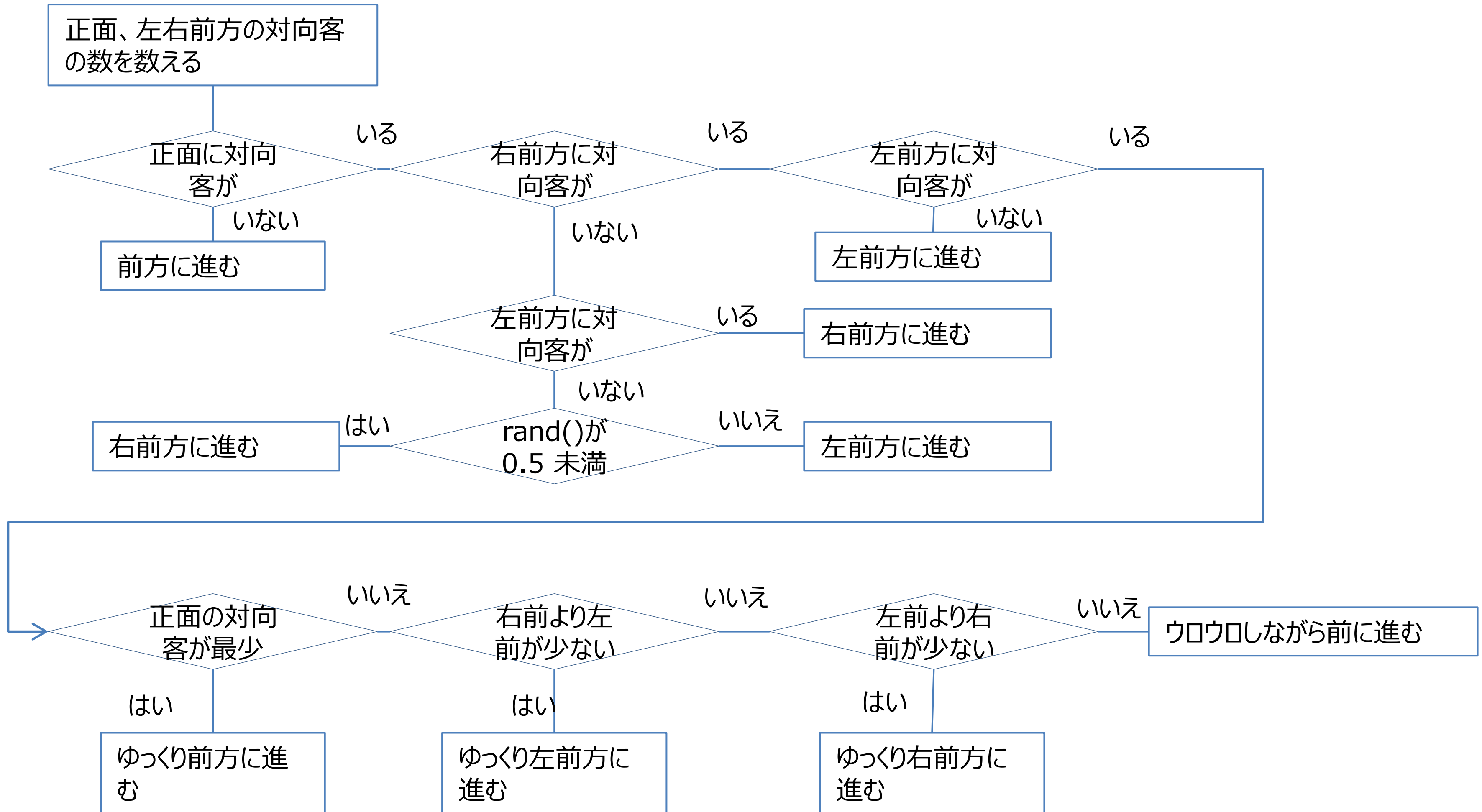
- ラッシュアワーでは、乗降客が互いに別の電車に乗るために混雑する状況になる。
- しかし、すれ違う流れが生じて、スムーズに歩ける場合もある。
- 各人はどのように判断し、行動しているのか？
- スムーズな人の流れはひとりでにできるのか？
  - W線の改札口が西側に、
    E線の改札口が東側にある。
  - W線の降車客全員がE線の改札口に向かい、
    E線の降車客全員がW線の改札口に向かうと仮定する。



❒11.2 ターミナル駅の通勤客の流れ(2)

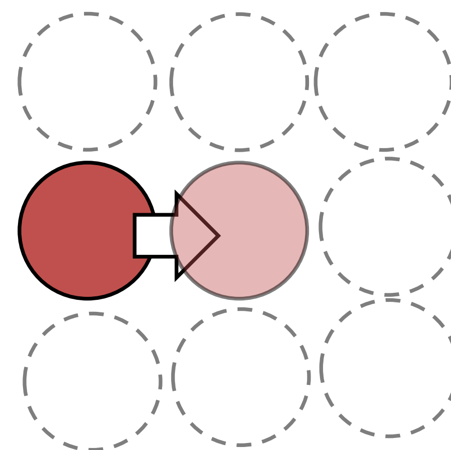
- 各人は反対方向から来る乗降客となるべくぶつからないようにするのと同時に、なるべく早く目的の改札口に向かおうとする。
  - 1)正面に対向客がいなければ、まっすぐ、速いペースで進む。
  - 2)正面には対向客がいるが、左前方か右前方にいなければ、いない方に速いペースで進む（どちらにもいなければ、ランダムに進む方向を決める）。
  - 3)正面も左右前方にも人がいる場合、一番空いている方向に進む。
  - 4)前方がどこも同じように対向客で混んでいるなら、うろうろしながら前に進む。

❐11.2 ターミナル駅の通勤客の流れ(3)



❑11.2 ターミナル駅の通勤客の流れ(4)

- 前方に対向客が何人いるか調べる。
  - 「調べたい方向を向く→調べたい範囲の中心に前進する→周囲を調べる→元の位置に戻る
    →向きを戻す」
    - self.turn(), self.forward(), self. make_agtset_around_own(), count_agtset()、self.forward（）、self.turn()とうまく組み合わせる。
    - 1ステップの間に行って戻る ＝ 動いていない。



- 状況合わせて、進む方向と速さを設定する。
  - self.turn()とself.forward()を使用する。
- 駅のコンコースの全体的な様子を把握する。
  - 乗降客の速さの平均を調べる。
    - 歩く速さではなく、向かっている改札口の方向に近づく速さを求める。

❐11.3 モデルの枠組みを作る

- 新規モデルを作成 「rush-hour」として保存する。
- 1)Universeの下に空間terminalを20×20の大きさで作る（ループあり）。
- 2)Universeの下に、各改札口からの乗降客数を表す整数型変数peopleと駅方向の速さの平均を求めるための実数型変数advanceを追加する。
- 3)terminalの下に、eastwardとwestwardの2種類のエージェント種を作る。
- 4)2種類のエージェントの下に目的改札口方向への移動速度を表す実数型変数advanceを追加する。
- 5)peopleを10から200まで変化させるコントロールパネルを設定する。
  - 種類: スライドバー, コントロール名: 乗降客数, 設定対象: people
  - 値の型: 整数（integer）, 初期値: 10, 範囲: 10-200, 目盛間隔:10
- 6)terminalと2種類のエージェントをマップ出力するように設定する。
- 7)平均速度を時系列グラフに示すように設定する
  - Y軸を最小0、最大1.2、目盛間隔0.2
  - 出力値は Universe.advance/(2*Universe.people)

❐11.4 ルールを書き込む(その1)

- peopleの数だけ東向きと西向きのエージェントを生成して、ターミナル駅通路にランダムに配置

```
1 def univ_init(self):
2     create_agt(Universe.terminal.eastward, num = Universe.people)
3     create_agt(Universe.terminal.westward, num = Universe.people)
4     one = make_agtset(space=Universe.terminal)
5     random_put_agtset(one)
6
```

- 集計値を初期化

```
7 def univ_step_begin(self):
8     # シミュレーションのステップ開始毎に実行する処理
9     Universe.advance=0
10
```

- スムーズな人の流れが発生したらシミュレーション終了
  - 最高速度1.2なので、平均1.19になったら終了する。

```
11
12 def univ_step_end(self):
13     # シミュレーションのステップ終了毎に実行する処理
14      if Universe.advance/(2*Universe.people)>=1.19:
15         print("Simulation completed after" + str(count_step()) + "steps")
16         exit_simulation()
17
```

- 保存・実行

❐11.5 ルールを書き込む(その2)(1)

- 東行きエージェントのルール
- 1)東に向かせる。

```
1  def agt_init(self):
2
3      # エージェントの初期化処理
4      self.direction=0
```

- 2)正面・右・左の前方の対向エージェントの状況を調べる。

```
7      # シミュレーションのステップ処理
8      #正面前方
9      self.forward(1)
10     crowd=self.make_agtset_around_own(1, False, agttype=Universe.terminal.westward)
11     CNo=count_agtset(crowd)
12     self.forward(-1)
13
14
15     #右前方
16     self.turn(-45)
17     self.forward(1)
18     crowd=self.make_agtset_around_own(1, False, agttype=Universe.terminal.westward)
19     RNo=count_agtset(crowd)
20     self.forward(-1)
21     self.turn(45)
22
23      #左前方
24     self.turn(45)
25     self.forward(1)
26     crowd=self.make_agtset_around_own(1, False, agttype=Universe.terminal.westward)
27     LNo=count_agtset(crowd)
28     self.forward(-1)
29     self.turn(-45)
30
```

❐11.5 ルールを書き込む(その2)(2)

- 3）状況に応じて前方に進み、そしてまた東を向く。
  - フローチャートに忠実に従ってルールを書き込む。

```
31
32     if CNo==0:  # 正面前方に誰もいなければ
33         self.forward(1.2)
34         self.advance=1.2
35
36     elif RNo==0:    # 右前方に誰もいなくて
37         if LNo==0:  # 左前方にもいなければ
38             if rand()<0.5:  # ランダムで方向を決める
39                 self.turn(30)
40
41             else:
42                 self.turn(-30)
43
44         else:   # 左前方には人がいる
45             self.turn(-30)
46
47         self.forward(1.2)
48         self.advance=1
49
50     elif LNo==0:    #  （正面や右にはいるが）左にはいなければ
51         self.turn(30)
52         self.forward(1.2)
53         self.advance=1
54
```

```
54
55     # ここから前方にはどの方向にも対向客がいる場合
56     elif RNo>CNo and LNo>CNo:   # 正面が一番空いていれば
57         self.forward(0.8)
58         self.advance=0.8
59
60     elif RNo>LNo:               # 左の方が空いていれば
61         self.turn(30)
62         self.forward(0.8)
63         self.advance=0.7
64     elif RNo<LNo:               # 右の方が空いていれば
65         self.turn(-30)
66         self.forward(0.8)
67         self.advance=0.7
68     else:                       # どちらも混んでいれば
69         self.turn(rand()*30-15)
70         self.forward(0.5)
71         self.advance=0.5
72
73     self.direction=0
74     Universe.advance += self.advance
75
```

❐11.5 ルールを書き込む(その2)(3)

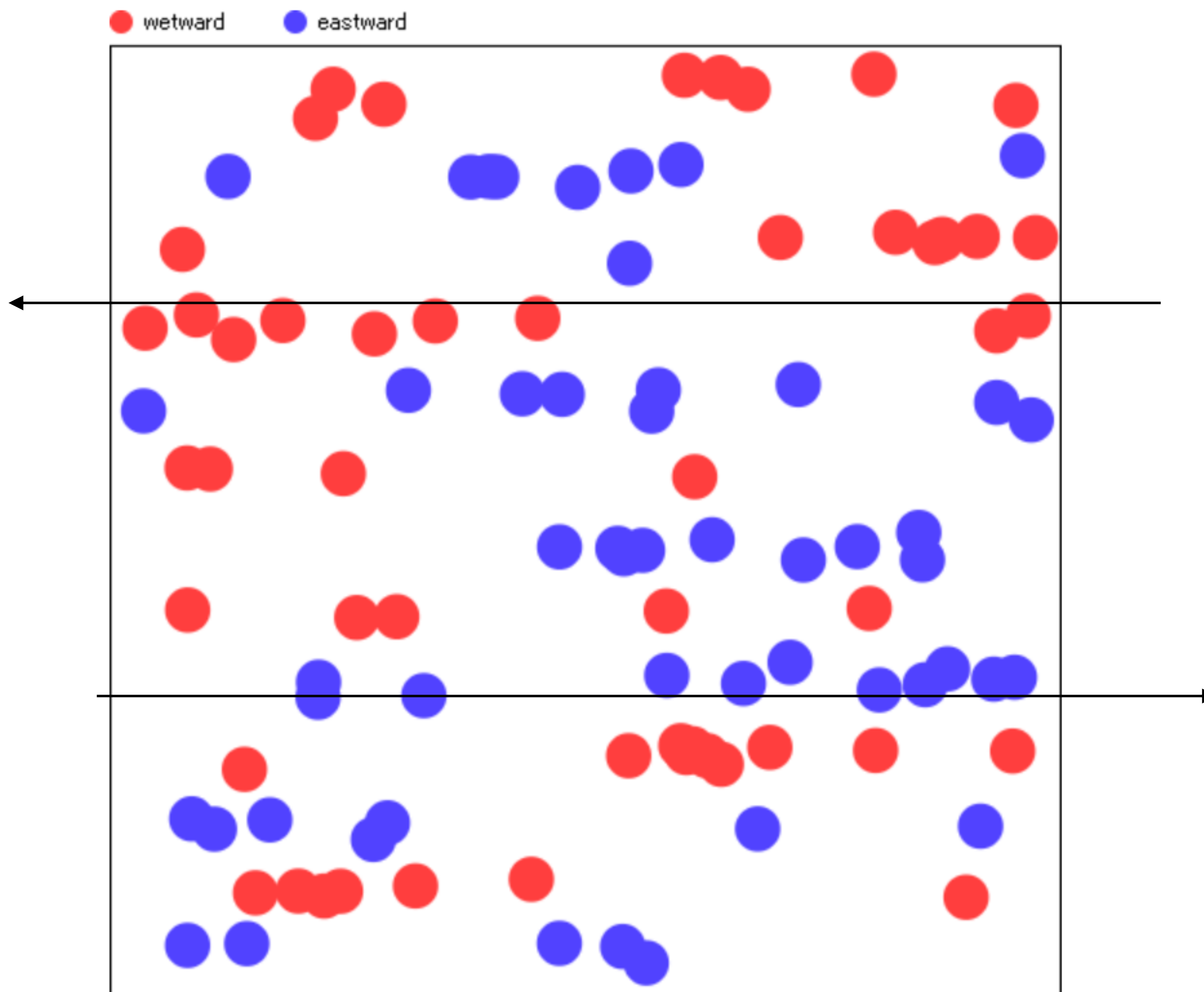
- if文の「入れ子構造」を繰り返すことで複雑な場合分けを表現する。
- 移動距離について
  - 直進の場合1.2
  - 斜め30度1.2進むと、正面方向には約1進む（厳密ではないが）
- Universe.advance += self.advance
  - エージェントの行動ルールではなく、シミュレーションの過程で全体状況を調べるための作業
    - ➡ エージェントがモデル作成者に協力してくれている。
  - Universe.advance = Universe.advance + self.advanceと同じ意味

❐11.5 ルールを書き込む(その2)(4)

- westward（西向き）のエージェントのルールも忘れずに記入する。
  - コピペする
  - ただし、
    - self.direction = 180にする必要（agt_initとagt_stepの最後）
    - self.make_agtset_around_own()で調べるエージェント種別はeastward

- 保存・実行

□実行結果



❐新しく学んだ事項

- if文を複雑に組み合わせる。
- 周囲の様子を調べる技巧的な方法。
- コメントを付してルールを読み取りやすくする。

❐「rush-hour 11.5終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 説明したモデルは以下のURLで共有しています。
  - https://artisoc-cloud.kke.co.jp/models/cUsDqkoxTXOGd4GLWYDEgw
  - モデル名「rush-hour 11.5終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒練習問題

- ①11.5のエージェントのルール表記で登場したif文で、下の３重の入れ子構造の部分を２重に単純化してみる。
  - 複数の条件式を論理演算子（and, or 等）でつなぐ。
    - if 条件式1 and 条件式2: どちらの条件も満たした場合 Trueになる
    - if 条件式1 or 条件式2： どちらかの条件を満たした場合 Trueになる

```
elif RNo==0:    # 右前方に誰もいなくて
    if LNo==0:  # 左前方にもいなければ
        if rand()<0.5:  # ランダムで方向を決める
            self.turn(30)

        else:
            self.turn(-30)

    else:   # 左前方には人がいる
        self.turn(-30)
```

- ②この章で作ったモデルは、対向客を避けるルールが基本だった。これを同方向の客に追従するルールに変えてみよう。
  - 乗り換え客は、自分と同方向に向かう客が多い方向に異動しながら進む。
  - 果たして、スムーズな人の流れは現れるだろうか？

第12章：エージェントの属性を豊富にする

❐12.0 「量より質」の豊かさを求めて

- エージェントの空間座標値（X, Y）に別の属性を与えます。
- artisoc Cloudの空間をグラフとして利用しましょう。
- 個々のエージェントの属性として「色」も追加できます。
- エージェントの「色」は出力マップに表せます。
- シミュレーション実行中に属性（色）が変わってもマップに現れます。
- エージェントに色をつける方法を学びましょう。

❐12.1 属性の「一目瞭然」化

- 空間の下にエージェント種別を定義すると5つの変数が自動で生成。
  - id、x、y、layer、direction
    - マップを横軸X、縦軸Yのグラフと見なすことも可能。
      （空間を動き回るエージェントというとらえ方をしないことも？）

- この章の目的
  - エージェントの属性として色を指定する手法を学ぶこと
    - 出力マップの上で個々のエージェントに色を自由につけることができるだけでなく、シミュレーションの過程で属性＝色を変えることも可能。

□12.2 人工国家の人口と経済(1)

- 世界の全100ヶ国は、初期状態では、全て、人口も1単位、経済規模も1単位だが、毎ステップ、人口は2%前後で増加し、経済規模は2%前後で成長する。
  - エージェント：国
  - x：人口、y：経済規模
  - 出力マップ：100ヶ国の人口と経済規模を示すグラフ
    - 新規モデルの作成　モデル名：「 national-size 」
    - graph（ループせず）という空間に、
nationエージェント（100カ国）を作り、
人口増加率（PGR）と経済成長率（EGR）の2変数（実数型）を追加する。
    - マップ出力の設定をする。

❐12.2 人工国家の人口と経済(2)

- nationのルールの書き込み
  - 人口（経済）の成長は、前の期の値に増加率（成長率）を掛けることで、当期の値を求める。
  - self.x と self.y には1ステップ前の値が入っていることに留意する。
  - 「or」を使ったif文：A or B どちらかが真の場合には～

```
1 def agt_init(self):
2     # エージェントの初期化処理
3     self.x=1
4     self.y=1
5     self.PGR=0.02+(rand()-0.5)/50
6     self.EGR=0.02+(rand()-0.5)/50
```

```
8  def agt_step(self):
9
10     # シミュレーションのステップ処理
11     self.x=self.x*(self.PGR+1)
12     self.y=self.y*(self.EGR+1)
13
14     if self.x>=50 or self.y>=50:
15         print("completed after"+str(count_step())+"steps")
16         exit_simulation()
17
```

❐12.2 人工国家の人口と経済(3)

- 保存、実行
  - 最初のうちはほぼ変化がないが、やがて急速に成長する。
  - 133～134ステップで終了する。

- artisoc Cloudの「空間」は必ずしも物理的な空間を意味しない。
  - 人口・経済規模の分布図
    - x軸が人口、y軸が経済規模
    - 経済水準(経済規模／人口）は45度線が同一水準
左上ほど高水準、右下ほど低水準

❐12.3 経済水準を色で表す(1)

- 経済水準をエージェントに計算させて、グラフ上に色で表示させる。
  - エージェントに特定の色を属性として与える変数を追加する。
  - その変数値をマップ上に出力するための設定。

- nationエージェントにecolevelという変数を追加する。
  - artisoc Cloudでは「色」を値に持つ変数は整数型

- ecolevelをマップ上に出力する。
  - 出力設定 > マップ出力・編集 > マップ要素リスト＞ nationの編集
    - ➡ エージェント表示色を「変数指定」にし、ecolevelを選択する。

❏12.3 経済水準を色で表す(2)

- national-sizeを継承して、 national-size Bとして保存する。
- 経済規模が人口より5単位以上大きくなった国→赤、
  人口が経済規模より5単位以上大きくなった国→青、
  どちらにも当てはまらない場合→緑

```
 8  def agt_step(self):
 9
10      # シミュレーションのステップ処理
11      self.x=self.x*(self.PGR+1)
12      self.y=self.y*(self.EGR+1)
13
14      # 色を設定する
15      if self.y >= self.x + 5:
16          self.ecolevel = COLOR_RED
17      elif self.x >= self.y + 5:
18          self.ecolevel = COLOR_BLUE
19      else:
20          self.ecolevel = COLOR_GREEN
21
22      if self.x>=50 or self.y>=50:
23          print("completed after"+str(count_step())+"steps")
24          exit_simulation()
```

❒12.3 経済水準を色で表す(3)

- COLOR_XXX
  - 8色を指定。文字で書かれているが“整数値”
    - COLOR_RED →赤
    - COLOR_GREEN →緑
    - COLOR_BLUE →青
    - COLOR_YELLOW →黄
    - COLOR_CYAN →水色
    - COLOR_MAZENTA →紫
    - COLOR_BLACK →黒
    - COLOR_WHITE →白

❑12.4 もっと微妙な色で属性を表す(1)

- rgb関数
  - 赤（Red）、緑（Green）、青（Blue）をミックスすることで様々な色を作る。
  - rgb（xxx, yyy, zzz）の形　（xxx, yyy, zzzはいずれも0から255の整数値）
    - 白→ rgb （255, 255, 255）
    - 赤→ rgb （255, 0, 0）
    - 緑→ rgb （0, 255, 0）
    - 青→ rgb （0, 0, 255）
    - 黒→ rgb （0, 0, 0）

❑12.4 もっと微妙な色で属性を表す(2)

- 「nation-size B」を継承して、「nation-size C」として保存する。
- 初めは黒っぽいが、経済成長が相対的に高ければ赤っぽく、人口増加が相対的に高ければ青っぽくする。
  - 12.3で書いた、特定の色をself.ecolevelに代入するif文のかわりに、以下の代入文を1行書き込む。

```
# 色を設定する
self.ecolevel = rgb(5 * int(self.y), 0, 5 * int(self.x))
```

- ➡ 初期状態はrgb（5, 0, 5）なのでほぼ黒
- ➡ 経済だけ成長するとrgb （250, 0, 5）となりほぼ赤
- ➡ 人口が増えるだけだとrgb （5, 0, 250）となりほぼ青
- ➡ 経済と人口の成長が同程度では緑

- 経済水準を（self.y - self.x）/（self.x + self.y）で表すと、経済水準が高いほどプラス1、経済水準が低いほどマイナス1に近づく。
  - いろいろなルールの書き方があるが、ここでは、(My.Y – My.X) / (My.X +My .Y)というプラス1からマイナス1の範囲の値をとる指標(el)で経済水準を表す。

❐12.4 もっと微妙な色で属性を表す(3)

- 「nation-size C」を継承して、「nation-size D」として保存する。

- 下記のルールを書き込む。
  - 人口増がゼロで最速経済成長だと、左上隅（1, 50）に到達し、elが約0.96なので、Int（265*el）は255 → RGB（255, 0, 0）
  - 逆だと、右下隅（50, 1） → RGB（0, 0, 255）

```
13
14      # 色を設定する
15      el=(self.y-self.x)/(self.x+self.y)
16      if self.y>=self.x:
17        self.ecolevel=rgb(int(265*el),255-int(265*el),0)
18      else:
19        self.ecolevel=rgb(0,255-int(265*(-el)),int(265*(-el)))
20
```

❐12.5 エージェント種別が異なるのか、属性が異なるのか

- これまで
  - 異なるエージェントは別種のエージェント種別として設定。
    - 東に向かう客と西に向かう客、1セント硬貨と10セント硬貨、人間とペット

- この章で学んだことを活かすと
  - エージェントの種類の違いを属性の違いと見なすことも可能。
    - 乗降客、コイン、生き物というエージェント種別で、属性が異なるものがいるという考え方。

- どちらが正しいというわけではなく、
  モデルを作る際の柔軟性やルールの類似性に合わせて考えるべき。

□新しく学んだ事項

- マップを2次元グラフとして利用する。
- 定数（整数型）としての色の指定（COLOR_REDなど）とマップ出力。
- 実行中にエージェントの色属性を変える。
- rgb関数

❐「national-size C 12.5終了時点」のモデル

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/QhjMTP59TuWVd40OuO47Wg
  - モデル名「national-size C 12.5終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

第13章：周囲のエージェントから影響を受ける

❐13.0 周囲の影響から逃れられません

- 病気の流行をモデル化しましょう。
- 周囲のエージェントの状態が重要です。
- 他のエージェントの属性を調べさせます。
- エージェントを取り出して調べる「for～in文」を学びます。

❐13.1 周囲のエージェントを詳しく調べる

- 第6章では
  - 周囲に他のエージェントがどれくらいいるかで自分の行動を選ぶ。

- この章では
  - より一般化
  - 周囲にいる他のエージェントの属性の状態に依存して、自分自身の行動や属性に変化が生じる。

- 具体的には
  - 1)周囲を認識させて、周囲にいる他のエージェント（複数でも）をリストアップする。
    - 第6章で学習済み
  - 2)リストにあるエージェントたちの属性を認識する。
    - 新規に学ぶこと
  - 3)認識した属性に応じて、自分の行動や属性を変化させる。
    - 「場合分け」の応用

❐13.2 病気の流行をモデル化する(1)

- 病気の流行のモデル
  - 確率過程モデル、ネットワーク理論など昔からのホットトピック。

- 社会に風邪をひいている人（患者、赤色のエージェント）が少数いる。
- 健康な人（水色のエージェント）は、自分の周囲に風邪をひいている患者が多いと、自分も風邪をひく（赤くなる）。
  - 単一のエージェント種別に、風邪（「赤色」の属性）か健康（「水色」の属性）かを区別する属性変数を持たせてモデル化する。

❐13.2 病気の流行をモデル化する(2)

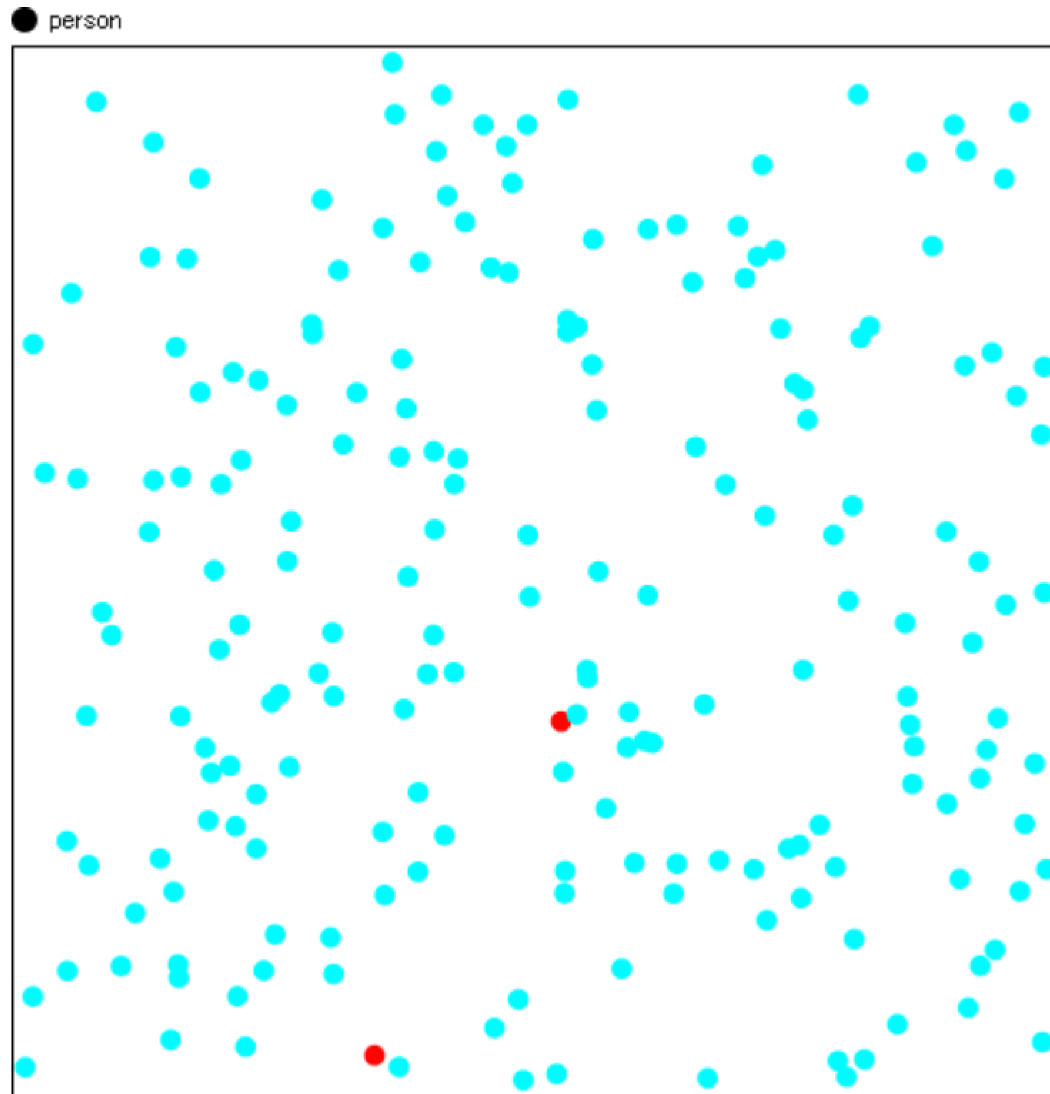
- モデルを新規作成　モデル名：infection
- Universeの下に空間society（デフォルト、連続空間）と変数pop（人口）を生成する。
- societyの下にエージェントperson（デフォルト）を生成。健康状態を表す整数型変数conditionを追加。
- 出力設定（マップ出力）
  - 色をconditionによる「変数指定」にする。
- コントロールパネル設定
  - popを対象。初期値:200、100から2000まで100刻みでコントロール。

❐13.2 病気の流行をモデル化する(3)

- personをpopで指定された数だけ生成し、societyにばらまく。
  - 風邪をひいている人は1%
- one ＝ エージェント型の一時的変数
  - create_agt()で生成したエージェントをoneに代入。
    → 1%の確率で風邪をひくように設定。
- one.condition
  - oneという仮の名前のエージェント（実体はpersonというエージェント種別のエージェント）が持っているconditionという変数を意味している。
- 上書き保存して実行。

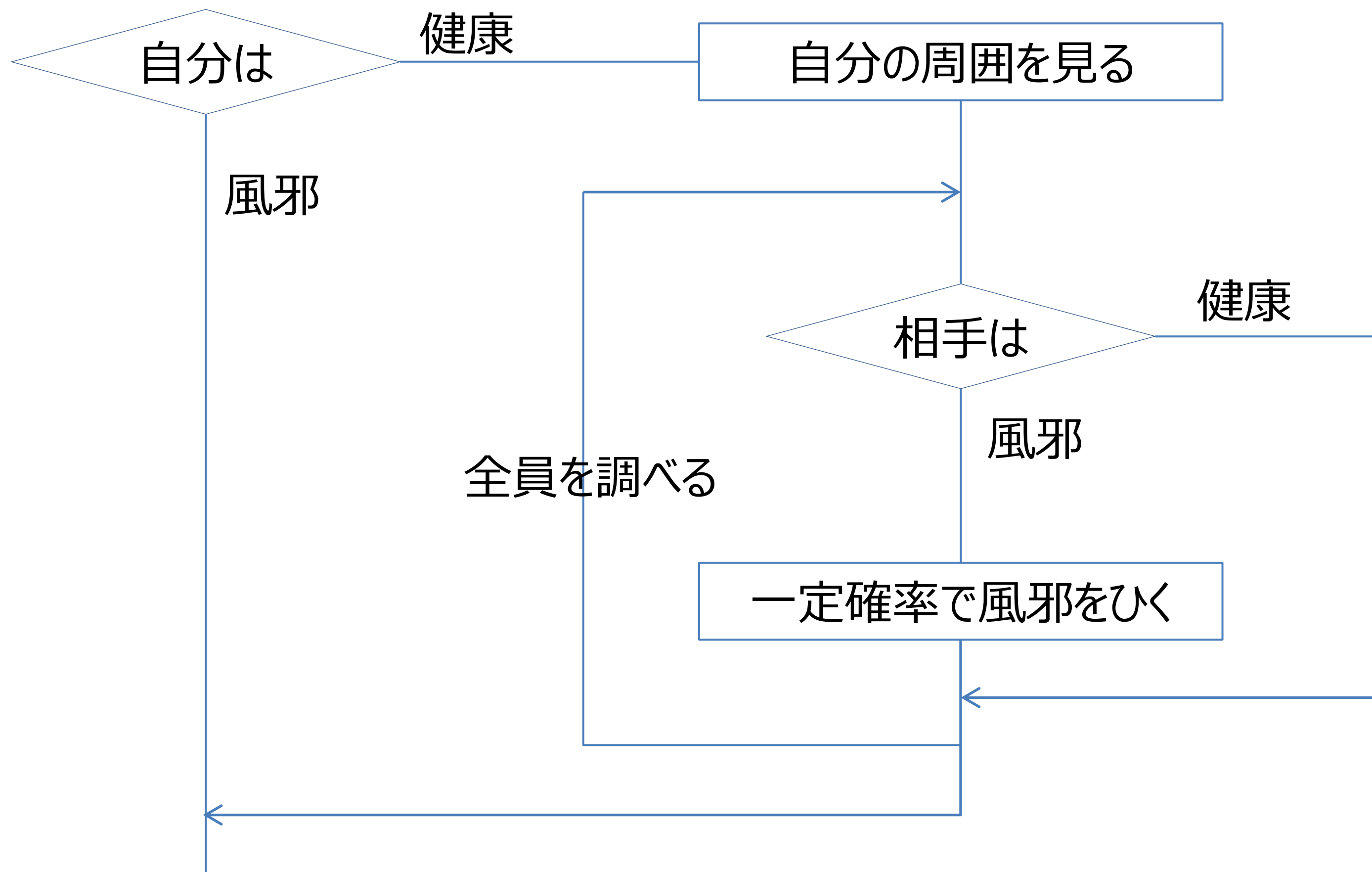
```
1  def univ_init(self):
2  
3      for i in range(Universe.pop):
4          one = create_agt(Universe.society.person)
5          if rand() < 0.01:
6              one.condition = COLOR_RED
7          else:
8              one.condition = COLOR_CYAN
9  
10     people = make_agtset(agttype=Universe.society.person)
11     random_put_agtset(people)
12 
```

❒13.2 病気の流行をモデル化する(4)



❒13.3 「for～in文」の登場(1)

■ personのルールはフローチャートの通り



❒13.3 「for～in文」の登場(2)

■ 周囲に風邪をひいている人が1人いると、3割の確率で自分に風邪がうつる。

Universe.society.personのルール

```
def agt_init(self):
    pass

def agt_step(self):
    if self.condition == COLOR_RED:
        # 風邪をひいているならなにもしない
        pass
    else:
        # 自分のまわりのエージェントを調べる
        neighbor = self.make_agtset_around_own(2, False)

        for one in neighbor:    # neighboの中から順次取り出す
            if one.condition == COLOR_RED:  # 風邪をひいていれば
                if rand() < 0.3:
                    self.condition = COLOR_RED
```

❐13.3 「for～in文」の登場(3)

- for one in neighbor:
  QQQQQ

  - neighborというエージェント集合型変数にしまわれているエージェントをoneという仮の名称（変数型はエージェント）で1つずつ順番に全て取り出し、個々のoneについて、QQQQQというルールを実行する。
  - neighborの中には、self.make_agtset_around_own(2, False)により、視野2の範囲にある自分以外のエージェントが全て含まれる。
  - それらをoneという仮の名称で1つずつ取り出してきて、QQQQQというルールに当てはめる。
  - 全てのエージェントを取り出し尽くしたら終了する。

❐13.3 「for～in文」の登場(4)

- QQQQQの箇所の if 文について
  - 周囲にいる人たちが病気かどうか調べて、病気ならば0.3の確率で自分も病気になる。
  - 周囲に患者が一人だけだと風邪をひく確率は30%だが、人数が多いと増加する。

- 一旦保存、継承後、モデル名: ”infection B”で保存する。

❐「infection 13.3終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/06NFuqGhTYSgPV8mCJtr5w
  - モデル名「infection 13.3終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒13.4 病気の流行を把握する(1)

- for～in文を使い、Universeのルールで全エージェントについて一括して集計してみる。
  1. Universeに患者をカウントするnumber（整数型）を追加。

```
13
14 def univ_step_begin(self):
15     Universe.number = 0
16
```

  2. 次のようなルールを書き込む。

```
17 def univ_step_end(self):
18     people = make_agtset(space=Universe.society)
19     for one in people:
20         if one.condition == COLOR_RED:
21             Universe.number = Universe.number + 1
22
```

❐13.4 病気の流行を把握する(2)

- 時系列グラフの出力設定をする。
  - 出力値は、患者数そのものではなく、全人口に占める病人の割合
  - 100 * Universe.number / Universe.pop

- S字型カーブに近いことを確認する。
  - 初期状態の感染率を集計していないので、立ち上がりは実際よりも少しきつめになる。

❒「infection B 13.4終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/5YQV7Se6ToWecPH2FyP-_A
  - モデル名「infection B 13.4終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❏13.5 現実の流行現象に近づける(1)

- モデルを保存、継承して、モデル名：infection Cとして保存する。

- 初期状態の感染率を操作できるようにする。
  - Universeに実数型変数initratioを追加し、コンパネに設定する。
    - 0と0.1の範囲で0.01間隔

- 風邪をひいても7ステップ経つと健康になる。
  - personにステップをカウントするための整数型変数cureを追加する。
  - 風邪をひいている場合にcureが増えるルール。

- 人々は動き回る。
  - 健康な人だけでなく、風邪をひいている人もいる。
    - direction、turn()、forward()を活用

❑13.5 現実の流行現象に近づける(2)

- Universeに実数型変数initratioを追加し、コンパネに設定する。
  - 0と0.1の範囲で0.01間隔
- personにステップをカウントするための整数型変数cureを追加。
- Universeの初期設定への追加。

```
def univ_init(self):

    for i in range(Universe.pop):
        one = create_agt(Universe.society.person)

        one.direction = rand() * 360    #ランダムな方向を向く
        one.cure = 0    # 治癒状況を示すカウンタ cureに初期値を設定

        if rand() < Universe.initratio: # 感染率はコンパネで与える

            one.condition = COLOR_RED
        else:
            one.condition = COLOR_CYAN

    people = make_agtset(agttype=Universe.society.person)
    random_put_agtset(people)

```

❐13.5 現実の流行現象に近づける(3)

■ エージェントのルールへの追加

```
 4 def agt_step(self):
 5     if self.condition == COLOR_RED:
 6         self.cure = self.cure + 1
 7     else:
 8         # 自分のまわりのエージェントを調べる
 9         neighbor = self.make_agtset_around_own(2, False)
10 
11         for one in neighbor:    # neighboの中から順次取り出す
12             if one.condition == COLOR_RED:  # 風邪をひいていれば
13                 if rand() < 0.3:
14                     self.condition = COLOR_RED
15 
16     # +10 ~ -10で向きを変えて移動
17     self.turn(rand() * 20 - 10)
18     self.forward(1)
19 
20     # 7ステップ経過したら全快
21     if self.cure >= 7:
22         self.condition = COLOR_CYAN
23         self.cure = 0
24 
25 
```

■ 上書きして実行

- ❑ コンパネ設定を変更しながら試してみる。

□新しく学んだ事項

- Universeのルールの中で、生成したエージェントの初期設定をする。
- 周囲にいるエージェントを全て調べる。
- for～in文
- エージェント型変数とその使い方
- for～in文を利用して、Universeのルールで一括集計する。
- ステップをカウントして、何ステップおきに実行するルール書く。

□「infection C 13.5終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/spz4IvU3Skm7ZFeK8lmImA
  - モデル名「infection C 13.5終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

第14章：特定のエージェントから影響を受ける

❒14.0 誰でもよいわけではありません

- 結婚相手を探すチョウをモデル化します。
- 個別の相手の属性が重要です。
- エージェントの集合からエージェントを１体だけ選ぶ技法を学びます。

❐14.1 エージェントをピックアップする

- for～in文
  - エージェント集合型の変数の中にリストアップされた全エージェントを順番に1つずつ取り出してきて、その内部状態（属性）を調べ、その結果に応じて、自分の状態や行動を変える。

- どれか1つのエージェントだけを取り出し、その属性を調べたい場合は？

❐14.2 結婚相手をさがすモデルを作る(1)

- チョウがキャベツ畑で交際相手を求めて飛び回っている。
- 雌はキャベツの葉にとまっている。近くに雄が来てとまると、つがいになる。
- 雄はフラフラと飛び回っているが、近くにチョウを見つけると接近する。
  - しかし、そのチョウが雄だったり、既につがいになっている雌だったりすると、他のチョウを求めて飛び去る。
  - もし独身の雌の場合には、そのチョウの交際相手としてそこにとどまる。

❐14.2 結婚相手をさがすモデルを作る(2)

- 新規モデル作成　モデル名：結婚モデル

- Universeの下にcabbagefield空間を生成（デフォルト）
その下にbutterflyエージェントを設定。

- butterflyに整数型変数conditionを作る。

- univ_initでbutterflyエージェントを60体生成。

- cabbagefirldをマップ出力するように設定。
  - マップ要素にbutterflyを追加。
  - 色を変数指定（condition）

❐14.3 ルールの大枠(1)

- 雄と雌の2種類のエージェントを作るのではなく、
  種類のエージェントを、conditionという変数を利用して、
  「雄か雌か」「未婚か既婚か」という属性を用いて区別する。
  - 独身雄をRED、既婚雄をYELLOW
  - 独身雌をMAGENTA、既婚雌をCYAN

- 雌は性別に反応、
  雄は性別＋雌の状態（既婚・未婚）に反応。

- 独身雌を見つけたかどうかで、雄の行動パターンが変わる。

❒14.3 ルールの大枠(2)

- チョウをランダムに配置。未婚雄と未婚雌を半々にする。

```
1  def agt_init(self):
2      self.x = rand() * 50
3      self.y = rand() * 50
4
5      if rand() < 0.5:
6          self.direction = rand() * 360
7          self.condition = COLOR_RED
8      else:
9          self.condition = COLOR_MAGENTA
10
```

- 上書き保存・実行

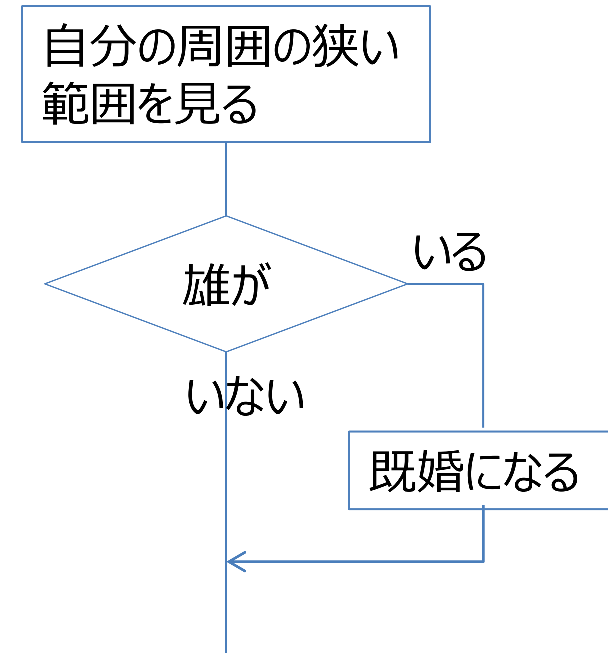
❑14.3 ルールの大枠(3)

■ 大枠は下記の通り

```
11  def agt_step(self):
12
13      if self.condition == COLOR_MAGENTA:
14          # 雌のルール
15          pass
16      elif self.condition == COLOR_RED:
17          # 雄のルール
18          pass
19
```

❐14.4 メスのルール

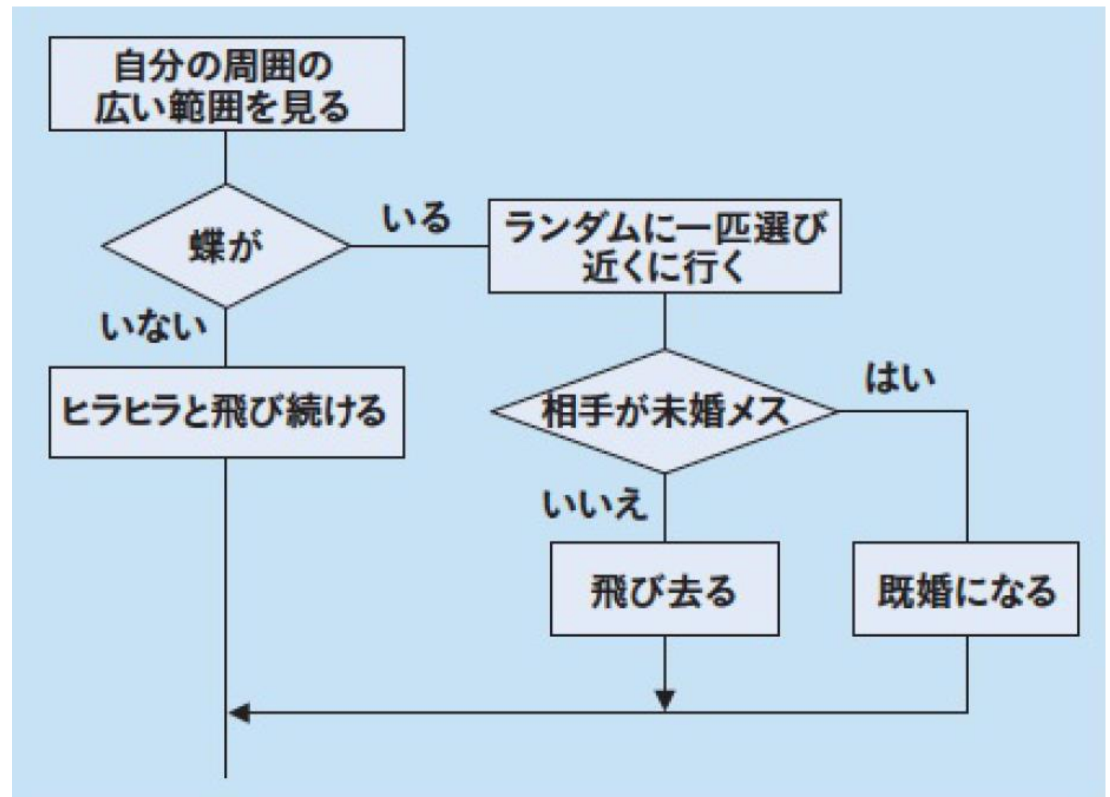
■ 未婚雌は雄がすぐそばに寄ってくれば既婚状態になる。




```
13      if self.condition == COLOR_MAGENTA:
14
15          # 雌のルール
16          neighbor = self.make_agtset_around_own(1, False)
17
18          for one in neighbor:
19              # もしまわりに雄（既婚、未婚問わず）がいれば
20              if one.condition == COLOR_RED or one.condition == COLOR_YELLOW :
21                  # 既婚状態に変化
22                  self.condition = COLOR_BLUE
```

❐14.5 オスのルール(1)

- 未婚雄は周囲を広く調べて、チョウが全くいなければ、ひらひらと飛び回る。
- チョウが見つかれば（2匹以上見つかると、ランダムに1匹だけを選び出して）、そこに飛んでいき、未婚雌かどうかを調べる。
- もし未婚雌ならそこに留まって結婚。
  それ以外なら飛び去る。



❏14.5 オスのルール(2)

```
26          # 雄のルール
27          neighbor = self.make_agtset_around_own(2, False)
28          if count_agtset(neighbor) == 0: # チョウがいない場合
29              self.turn(rand() * 60 - 30)
30              self.forward(0.6)
31          else:
32              # チョウがいる場合
33              one = randchoice(neighbor)
34              self.x = one.x + 0.5
35              self.y = one.y + 0.5
36
37              if one.condition == COLOR_MAGENTA:
38                  self.condition = COLOR_YELLOW
39              else:
40                  self.turn(rand() * 60 - 30)
41                  self.forward(2)
42
```

❒14.5 オスのルール(3)

- one = randchoice(neighbor)
  - neighborの中のエージェントの中からランダムに1匹を選び出す。
    - 選ばれたエージェントは、one（エージェント型変数）に代入。
    - neighborに1匹のエージェントも格納されていない場合、を実行してしまうとエラーが生じる。
    - → 必ず0の場合を除外するようなクセをつける。
      - ☑ if文で0となる場合を除外するなど

□新しく学んだ事項

- エージェント集合型変数の中からエージェント1匹（1体）を選び出す。
- randchoice

❐「結婚モデル 14.5終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/3ztUI-4yQ0SzI_Z6ZUOj0A
  - モデル名「結婚モデル 14.5終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒練習問題の説明(1)

- 次のような事例を考えましょう。
  - 広告メディアが互いに異なる製品を宣伝している。宣伝に接した消費者の一部はその製品を使うようになる。
  - まず、Universeの下に空間marketを作り（設定はデフォルト値のまま）、その下にmediaエージェントを10、consumerエージェントを100作ります。mediaにはproduct、consumerにはfavoriteをどちらも整数型の変数として作ります。マップ出力では、両タイプのエージェントも追加した変数を指定する色表示とします。

❐練習問題の説明(2)

- 広告メディアは最初に、アカ製品を宣伝するメディア（赤色表示）とアオ製品を宣伝するメディア（青色表示）を半々の確率で存在するようにします。

```
def agt_init(self):
    self.x = rand() * 50
    self.y = rand() * 50

    if rand() >= 0.5:
        self.product = COLOR_RED
    else:
        self.product = COLOR_BLUE

```

- 他方、消費者も最初、marketにランダムに存在しています。

```
def agt_init(self):
    self.x = rand() * 50
    self.y = rand() * 50

    self.direction = rand() * 360

```

- 毎ステップ、メディアは場所を変えずに宣伝し続けるのに対し、消費者はmarketの中を動き回ります。

❐練習問題

■ 14.1

消費者は、身近な（視野範囲が2）メディアから影響を受け、3割の確率でそのメディアが宣伝している製品を好むようになる。

視野内に複数のメディアがいる場合、自分が影響を受けるメディアはランダムに選ばれることにしましょう。

■ 14.2

消費者は、身近な（視野範囲が2）メディア（複数いる場合はランダムに選択）が宣伝している製品と同じ製品を使っている他の消費者が身近（視野範囲が2）に複数いる場合にかぎり、その製品を好むようになる。

第15章：他のエージェントに働きかける

❒15.0 他者の状態を変える

- 周囲のエージェントの属性（内部状態）や行動を変えられます。
- 他のエージェントの変数値を操作する方法を学びます。
- エージェントの内部状態を論理的な真偽で区別します。
- 牧羊犬のモデルを作ります。

❐15.1 積極的な働きかけ

- 第13章の流行現象のモデル
  - 意図的に他人に病気をうつそうとしてるわけではない
    - ここでは、意図的に働きかける、モデル

- モデルの基本
  - 働きかける側
    - 周囲を見回して、周囲にいるエージェントを認識する
    - 彼ら（の全員または一部）の属性を変える
  - 働きかけられる側
    - 属性に応じた行動のレパートリーを持っている
    - 他のエージェントに変えられた属性に対応する行動をとる

❐15.2 牧羊犬の仕事をモデル化する(1)

- 牧場では夕方、飼っている羊を柵に入れなければならない
  牧羊犬が羊を追い込む仕事をする
  - 新規モデル作成　モデル名：Shepherd
  - Universeの下に空間meadowを生成（デフォルト）
  - meadowの下にエージェント種 Sheep、Dogをつくる
  - univ_initでSheepを100匹、Dogを10匹生成
  - Sheepには牧羊犬に捕まったかどうかを表す変数 caught を作成
    - つかまったらTrue、そうでなければFalseとする（←ブール値については後述）
  - 出力設定でマップ出力の設定

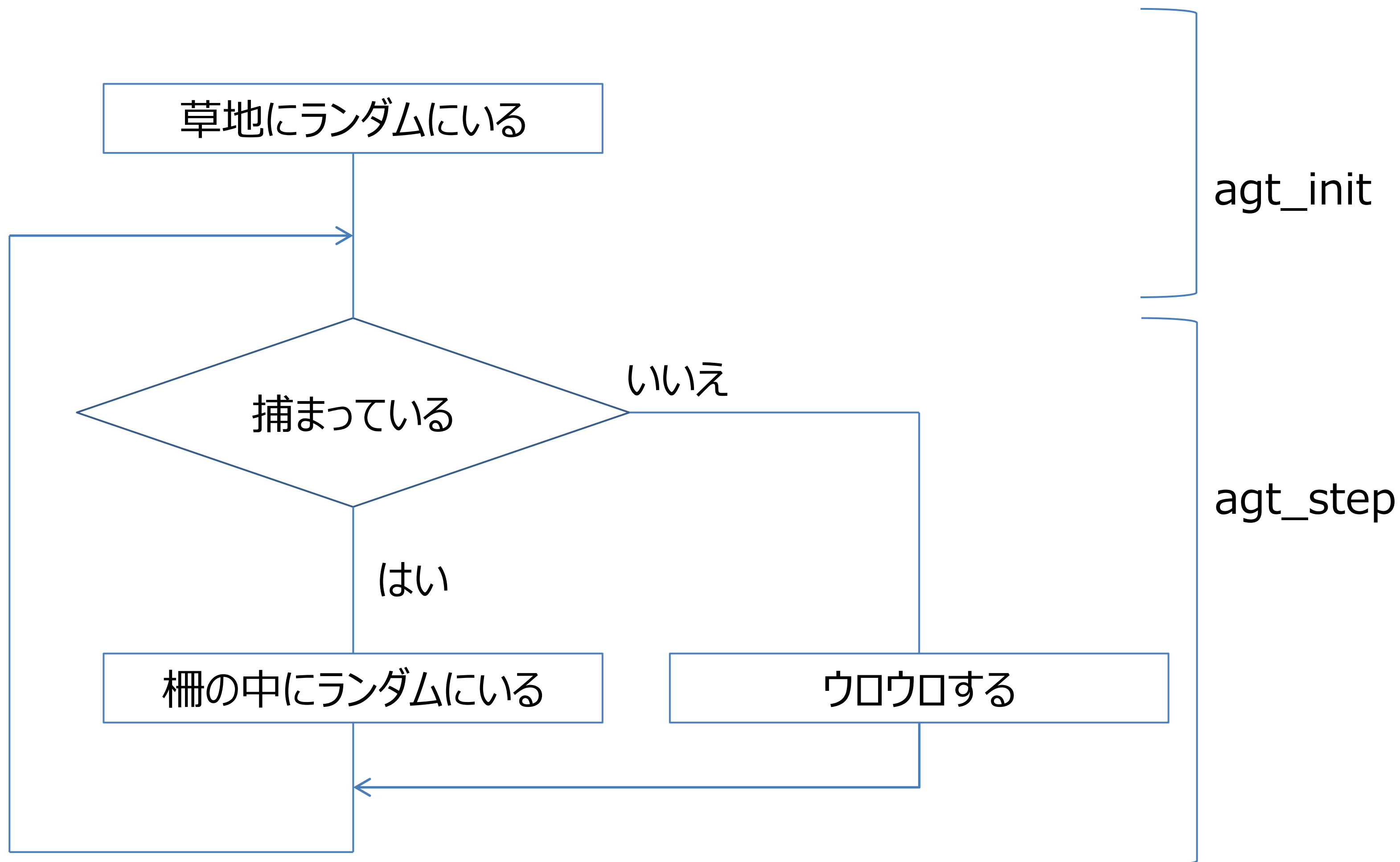
❒15.2 牧羊犬の仕事をモデル化する(2)

- エージェントの行動の本質的な部分は？
  - 羊は、柵の外ではうろうろと歩き回る
    - 柵の内側はmeadowのX座標が0から10、Y座標が0から10の範囲の正方形とし、柵の外はそれ以外とする
  - 牧羊犬に捕まった羊は、柵に追い込まれる

    （ルールを単純にするため、瞬間移動で柵の中に入る）
  - 牧羊犬は牧場を走り回り、羊がそばにいると、羊を1頭ずつ捕まえる

❑15.3 羊たちの行動(1)

- フローチャート



❐15.3 羊たちの行動(2)

■ 1) 最初は、まだ牧羊犬に捕まっておらず、草地にランダムにいる。

```
1 def agt_init(self):
2     self.x = rand() * 50
3     self.y = rand() * 50
4 
5     self.direction = rand() * 360
6     self.caught = False
7 
8 
```

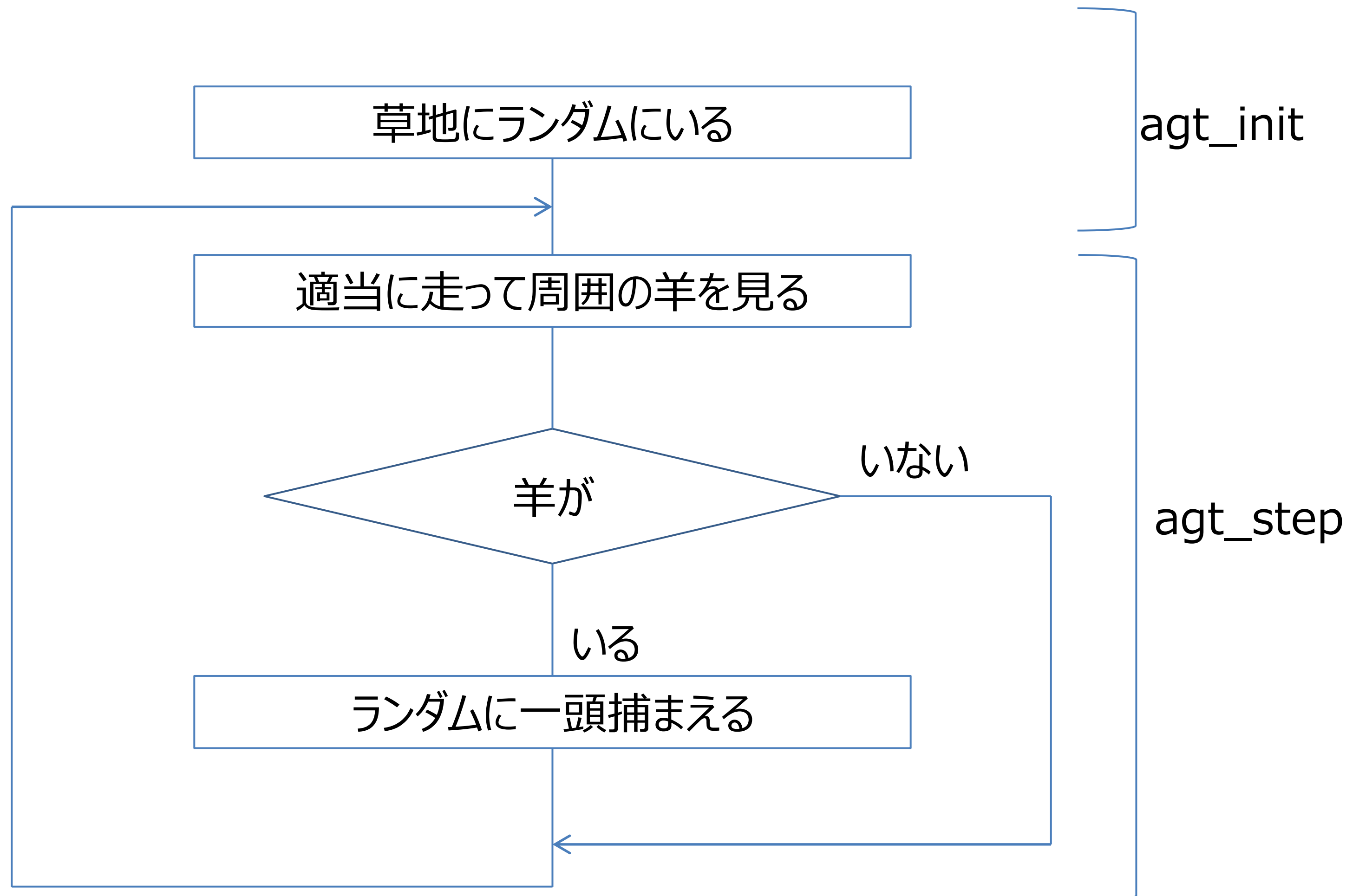
❐15.3 羊たちの行動(3)

- 2) 毎ステップ、捕まったかそうでないかで、異なる行動をとる。

```
 8
 9  def agt_step(self):
10
11      if self.caught == True:
12          self.x = rand() * 10
13          self.y = rand() * 10
14
15          self.caught = False
16
17      elif 10 <= self.x or 10 <= self.y:
18          self.forward(0.5)
19          self.turn(rand() * 40 - 20)
20
21
```

❐15.4 牧羊犬の行動(1)

■ フローチャート



❐15.4 牧羊犬の行動(2)

- 1) 初期状態は、羊と同様。
- 2) 牧羊犬は毎ステップ走り回って、羊を探し、近くにいると、1頭を捕まえる。

```
def agt_init(self):

    #初期状態は羊と同様
    self.x = rand() * 50
    self.y = rand() * 50

    self.direction = rand() * 360

def agt_step(self):

    # 牧羊犬は毎ステップ走り回って、
    self.forward(1)
    self.turn(rand() * 40 - 20)

    # ヒツジを探し
    neighbor = self.make_agtset_around_own( 2, False, agttype=Universe.meadow.Sheep)
    num = count_agtset(neighbor)

    if 1 <= num: # ヒツジが近くにいると
        target = randchoice(neighbor)    # 1頭を捕まえる
        target.caught = True

```

❐15.4 牧羊犬の行動(3)

- self.make_agtset_around_ownのところ
  - 視野2の範囲に羊がいるかを調べる
- count_agtset()のところ
  - 視野2の範囲の羊の数を調べる
- そして、もし1頭以上いれば、ランダムに1頭選んで、targetという一時的なエージェント型変数に格納
- 格納されたその1頭を、捕まった状態に変える
  - target.caught = True
    - TrueまたはFalseの2値を取る値を「ブール値」と呼びます

❐「Shepherd 15.4終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 途中でついていけなくなった人は、下記のモデルにアクセスし、モデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/qsXnymdGRoKImYIcyMP0Gg
  - モデル名「Shepherd 15.4終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒15.5 作業を完了する時間を調べる(1)

- Shepherdモデルを継承、モデル名:Shepherd B で保存する

- 羊を飼うには牧羊犬が必要だが、牧羊犬には優秀なのもいれば、駄犬もいる。それでは効果で優秀な牧羊犬を少数入手すべきなのか、それとも廉価で平凡な牧羊犬を多く入手すべきなのか。
  - Dogの数を変化させる
  - self.make_agtset_around_ownの視野の広さを変えてみる
  - self.forward()の数値（足の速さ）を変えてみる
  - → コントロールパネルで変えられるようにしてみる
    - 第7章で学んだことをベースに自分でやってみる

❐15.5 作業を完了する時間を調べる(2)

- 柵の中にいる羊を数える作業をUnvierse内で行う
  - Universeの下に整数型変数infenceを作成する

```
 8 def univ_step_end(self):
 9     Universe.infence = 0    # 初期化
10     total = make_agtset(agttype=Universe.meadow.Sheep)  #すべてのヒツジをtotalに格納
11 
12     for one in total:
13         if one.x < 10 and one.y < 10:   # ヒツジが柵内にいればカウントする
14             Universe.infence += 1
15 
16     if Universe.infence >= 100:
17         print("Completed after" + str(count_step()) + " steps")
18         exit_simulation()
19 
```

❒15.6 モデル作りを工夫する

- 牧羊犬は柵の外だけを走り回る
  - これまでのモデルは、牧羊犬は柵の中にも入って作業
  - そこで、次のルールを加える（加える場所は各自で考える）

```
# 策の中に入らない
if self.x < 10 and self.y < 10:
    if 0.5 <= rand():
        self.x = 11
        self.y = rand() * 11
    else:
        self.x = rand() * 11
        self.y = 11
```

☐新しく学んだ事項

- 他エージェントの変数を変えることにより、そのエージェントに自分の行動を変えさせる。
- ブール値とその使い方

❐「Shepherd B 15.6終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/YdyWjrAbRBGZ4BIt_AdIow
  - モデル名「Shepherd B 15.6終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐練習問題

- 広告メディアが互いに異なる製品を宣伝している。宣伝に接した消費者の一定割合にその製品を使うようにさせる。

- 15.1
  - メディアは、近くにいる消費者の中からランダムに選ばれた一人の好みを宣伝している商品に変える。
    - ヒント: 牧羊犬モデルと同じ構造　target.favorite = self.product

- 15.2
  - メディアは近くにいる全ての消費者に影響を及ぼし、3 割の確率で消費者は宣伝している製品を好むようになる。
    - ヒント: for～in文　target.favorite = self.product

第16章：エージェントへの究極の働きかけ

❐16.0 エージェントそのものに作用できます

- エージェントに対する究極の働きかけは、エージェントの生成と抹消です。
- プランクトンの食物連鎖をモデル化しましょう。
- シミュレーションの最中にエージェントを作ります。
- シミュレーションの最中にエージェントを消します。
- 新しく生成されたエージェントの変数に値を設定します。
- 自分で自分を消すことも可能です。

❑16.1 エージェント自体への働きかけが可能です

- artisoc Cloudでは、シミュレーションの途中でもエージェントを作ることが可能
  - 単にエージェントが動かなくなる、他から影響を受けなくなるだけでなく、モデルから“抹消”することも可能
  - 自分で自分を消すことも可能

❐16.2 「殺す」は「消す」(1)

- 水槽に植物プランクトンphytoと動物プランクトンzooが浮遊
  - 後者が前者を餌にしている

1. 新規モデル作成　モデル名: plankton
2. 空間aquarium（デフォルト）を生成
3. その下に、エージェント種別　植物プランクトンphyto、動物プランクトンzooを作成
4. univ_initで植物プランクトンphytoを200、動物プランクトンzooを20生成
5. マップ出力の設定: 各エージェントを表示（エージェント毎に色を変える）

❑16.2「殺す」は「消す」(2)

- 植物プランクトンphytoのルール
  - ランダムな位置と向きに初期配置
  - プラスマイナス60度の範囲で向きを変えながら、毎ステップ0.5進む

```
def agt_init(self):

    # ランダムな位置と向きに初期配置
    self.x = rand() * 50
    self.y = rand() * 50

    self.direction = rand() * 360

def agt_step(self):

    # プラスマイナス60度の範囲で向きを変えながら、
    self.turn(rand() * 120 - 60)

    # 毎ステップ0.5進む
    self.forward(0.5)

```

❐16.2「殺す」は「消す」(3)

- 動物プランクトンzooプランクトンのルール
  - 植物プランクトンphytoと初期設定は同じ
  - 行動ルールには微妙な違い
    - プラスマイナス30度で向きを変える
    - 毎ステップ2進む
  - また、視野2の範囲に植物プランクトンphytoがいれば、1ステップに1匹食べる（消す）

- ルール（次スライド参照）
  - del_agt(food)：エージェントを消すルール
  - エージェント集合型変数surroundと、エージェント型変数foodの定義

- 上書き保存して実行

❐16.2 「殺す」は「消す」(4)

```
def agt_init(self):

    # ランダムな位置と向きに初期配置
    self.x = rand() * 50
    self.y = rand() * 50

    self.direction = rand() * 360

def agt_step(self):

    # プラスマイナス30度の範囲で向きを変えながら、
    self.turn(rand() * 60 - 30)

    # 毎ステップ2進む
    self.forward(2)

    # 餌をさがして、あれば食べる
    surround = self.make_agtset_around_own(2, False, agttype=Universe.aquarium.phyto)
    if 0 < count_agtset(surround):
        food = randchoice(surround) # 1つ選ぶ
        del_agt(food) # 食べる


```

❐「plankton 16.2終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/n_vOOKvkSB-IICEhpQAwPg
  - モデル名「plankton 16.2終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐16.3 「産む」は「創る」(1)

- planktonモデルを継承、モデル名plankton Bとして保存

- 植物プランクトンphytoが増殖するルールを付加
  - 5%の確率で、create_agt()で植物プランクトンphytoエージェント1個をつくる
  - そのまま一時的なエージェント型変数daughterに格納
  - できたてのエージェントに位置や向きを設定
    - 親と同じ場所
    - 向きはランダム

  - ※ create_agt() で生成されるとすぐにagt_init(self) が実行
    - 位置・向きともにランダムに設定
    - 上記ルールにより改めて設定し直している

- 保存、実行

❒16.3 「産む」は「創る」(2)

```
 9  def agt_step(self):
10
11      # プラスマイナス60度の範囲で向きを変えながら、
12      self.turn(rand() * 120 - 60)
13
14      # 毎ステップ0.5進む
15      self.forward(0.5)
16
17      # 一定確率で増殖
18      if rand() < 0.05:
19          daughter = create_agt(Universe.aquarium.phyto)
20          daughter.x = self.x
21          daughter.y = self.y
22          daughter.direction = rand() * 360
23
24
```

❐16.4 個体数変動を調べる(1)

- plankton Bを継承、モデル名 plankton Cとして保存
- プランクトン数を時系列グラフに出力
  - Universeの下に整数型変数numphytoとnumzooを追加
  - numphyto/10とnumzooを出力
    - 数が違うので桁を合わせる
- どちらかのプランクトンがいなくなったらシミュレーション終了

```
 8 def univ_step_end(self):
 9 
10     # phyto, zooの数を数える
11     Universe.numphyto = count_agt(agttype=Universe.aquarium.phyto)
12     Universe.numzoo = count_agt(agttype=Universe.aquarium.zoo)
13 
14     # どちらかの数が0になればシミュレーション終了
15     if Universe.numphyto <= 0 or Universe.numzoo <= 0:
16         print("Extinct after " + str(count_step()) + " steps")
17         exit_simulation()
18 
19 
```

❐16.4 個体数変動を調べる(2)

- count_agt()とcount_agtset()の違いに注意
  - count_agt()は、当該エージェントの数をカウント

- del_agt()の効果
  - エージェントを消しているので、 count_agt()でカウントしても間違いない
  - 仮に、捕食された植物プランクトンphytoを、赤色から黒色に変えるというルールにしていると、count_agt()では、赤色も黒色も一緒にカウントしてしまう

❐16.5 過ぎたるは及ばざるがごとし

- plankton Cを継承し、モデル名plankton Dとして保存
- 植物プランクトンが周囲に増えすぎると環境が劣化して、自分自身が死んでしまう
  - 周りに4体以上植物プランクトンがいれば、自分を消す
  - del_agt(self)：自分を消すルール

```
 9  def agt_step(self):
10
11      # プラスマイナス60度の範囲で向きを変えながら、
12      self.turn(rand() * 120 - 60)
13
14      # 毎ステップ0.5進む
15      self.forward(0.5)
16
17      # 一定確率で増殖
18      if rand() < 0.05:
19          daughter = create_agt(Universe.aquarium.phyto)
20          daughter.x = self.x
21          daughter.y = self.y
22          daughter.direction = rand() * 360
23
24      # 増えすぎたら死滅
25      surround = self.make_agtset_around_own(1, False, agttype=Universe.aquarium.phyto)
26      if 4 <= count_agtset(surround):
27          del_agt(self)
28
```

❐「plankton D 16.5終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/q3oghH6wQbyLjMy-9lmuHg
  - モデル名「plankton D 16.5終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒16.6 自然の生態系に近いモデルをめざして(1)

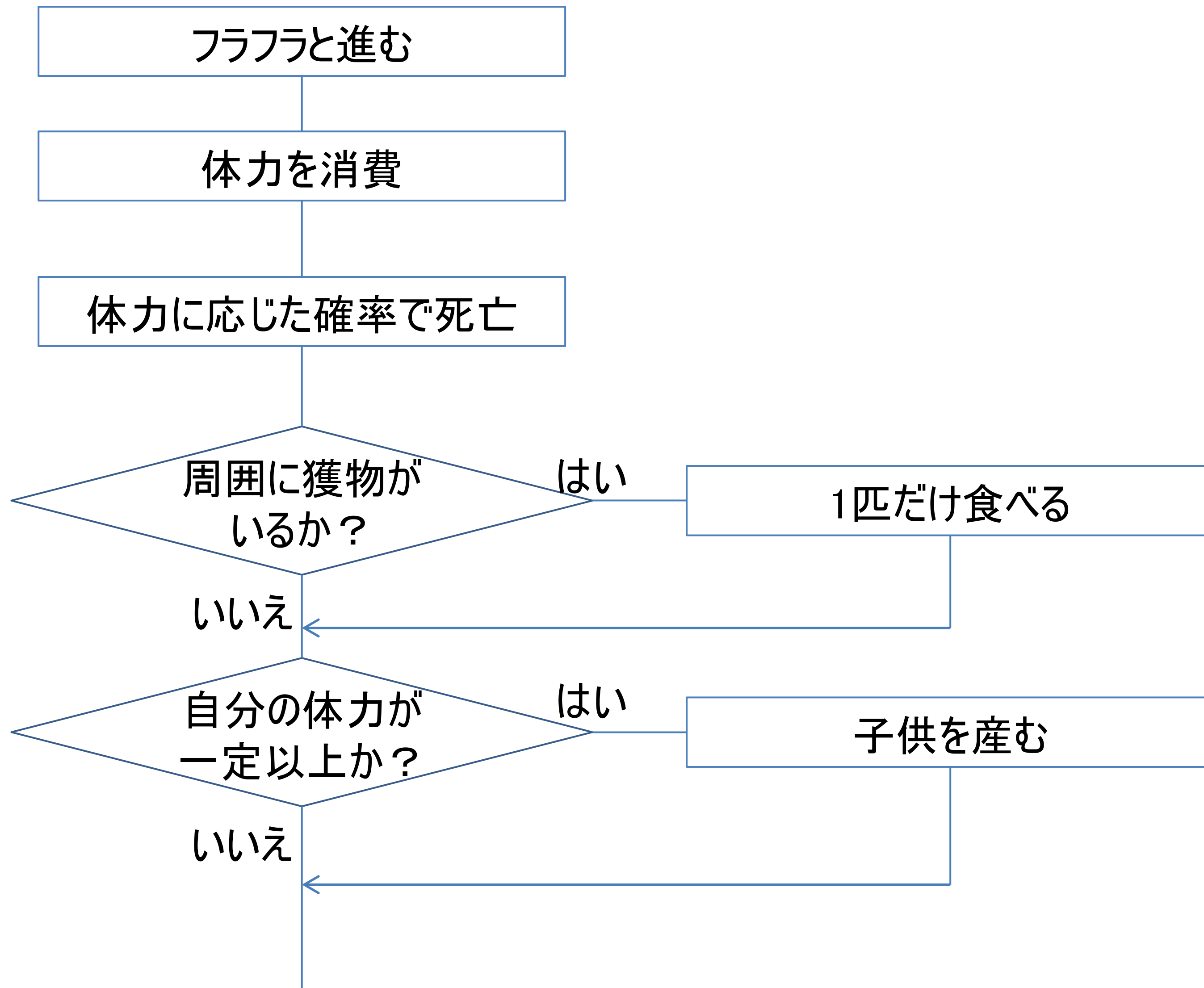
- 海洋に植物プランクトンと動物プランクトンが浮遊
  - 後者が前者を餌に
- 植物プランクトンは増殖、しかし増殖しすぎると死ぬ
- 動物プランクトンは、周囲の植物プランクトンを食べる
  - 食べないと体力が消耗
  - たくさん食べると増殖

→ 動物プランクトンのルールについて追加

❐16.6 自然の生態系に近いモデルをめざして(2)

- plankton Dを継承し、モデル名plankton Eとして保存
- 動物プランクトンzooのルール
  1. 動物プランクトンzooのエージェント変数powerを追加（体力を表す）
  2. 最初4から9の間の体力（power）をランダムで与える
  3. 体力は毎ステップ1ずつ減少
植物プランクトンを食べると3上昇
  4. 体力が減るとそれに応じて死ぬ確率は高くなる
     - ゼロなら100%、1なら20%、10なら2%
  5. 体力が10を超えると増殖
     - 親プランクトンは体力を4減らす
     - 子プランクトンは体力4与えられて産まれる
  6. 子プランクトンは、親と同じ位置。向きはランダム

❐16.6 自然の生態系に近いモデルをめざして(3)



❑16.6 自然の生態系に近いモデルをめざして(4)

動物プランクトンzooのルール

```
1  def agt_init(self):
2
3      # ランダムな位置と向きに初期配置
4      self.x = rand() * 50
5      self.y = rand() * 50
6
7      self.direction = rand() * 360
8
9      # パワーの初期値は4～9
10     self.power = 4 + rand() * 5
11
12 def agt_step(self):
13
14     # プラスマイナス30度の範囲で向きを変えながら、
15     self.turn(rand() * 60 - 30)
16
17     # 毎ステップ2進む
18     self.forward(2)
19
20     # パワーが無くなると死
21     self.power = self.power - 1
22     if rand() * self.power <= 0.2:
23         del_agt(self)
24
```

❐16.6 自然の生態系に近いモデルをめざして(5)

動物プランクトンzooのルール（続き）

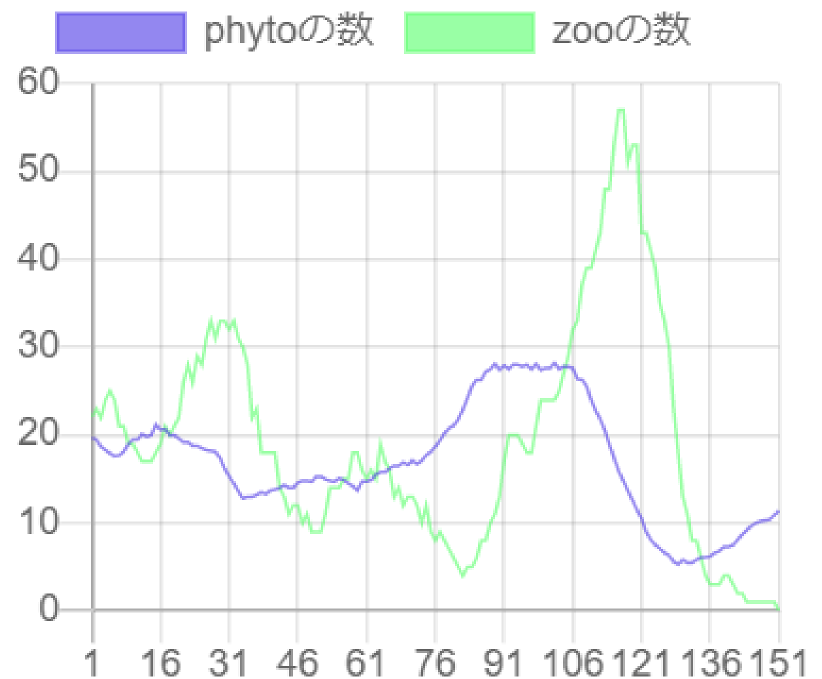
```
25      # 餌をさがして、あれば食べる
26      surround = self.make_agtset_around_own(2, False, agttype=Universe.aquarium.phyto)
27      if 0 < count_agtset(surround):
28          food = randchoice(surround) # 1つ選ぶ
29          del_agt(food) # 食べる
30
31          self.power = self.power + 3 # 食べるとパワーが回復
32
33      # パワーが10以上になると増殖
34      if 10 <= self.power:
35          daughter = create_agt(Universe.aquarium.zoo)
36          daughter.x = self.x
37          daughter.y = self.y
38          daughter.direction = rand() * 360
39          daughter.power = 4
40          self.power = self.power - 4
41
```

❒16.6 自然の生態系に近いモデルをめざして(6)

- 本質的には捕食者と被捕食者との相互作用のモデル
  - 餌が多いと捕食者が増え、その結果餌が減って捕食者も減り、すると餌が増え始める、という循環的な特徴を持っているはず。
  - しかし、なかなか安定的に循環しない。

- 餌も捕食者も空間的に散らばっているために、空間を考慮しない数理モデルのようにはきれいな結果にならない。
  - 餌が遠くにあっても食べられない。餌がなくなる前に、捕食者が絶滅してしまう可能性の方がずっと大きい。

- 捕食者と被捕食者との間に循環的な関係が形成されるのは、それほど簡単なわけではない。



□新しく学んだ事項

- create_agt()をエージェントの行動ルールとして用いる。
- count_agt()
- del_agt()
- del_agt(self)
- エージェント数をあまり多くしない。

❐「plankton E」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/hTkZUtqdStqg_arsh40jBA
  - モデル名「plankton E」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐練習問題

- 1. 牧場に生えている草を羊が食べる。
  - kusaharaという空間にkusaとhitsujiエージェントを追加し、kusaが増殖するルールと羊が草を食べるルールを書きましょう。

- 2. planktonモデルに、動物プランクトンzooを食べる魚エージェントfishを追加する。

- 3. plankton Eモデルには，植物プランクトンphytoが増えたり死んだりする条件、動物プランクトンzooが植物プランクトンphytoを食べて増える体力、毎ステップの体力消耗度、体力と死亡率との関係，増殖する際の体力の親から子への移動量など、たくさんのパラメータがある。それらを適当に調整して、安定的な（周期的にプランクトン数が変動する）状態を作り出してみよう。

第17章：非対称的な相互作用を複雑化する

❐17.0 そう簡単には従わない

- 思いのままにならないところが面白いのです。
- 影響を及ぼす側と及ぼされる側との関係をモデル化します。
- 誘導のモデルと取り締まりのモデルを作ります。

❐17.1 一方的関係から非対称的かつ双方的な関係へ

- これまでの手法を組合せながら、少し複雑な影響力行使の方法を学ぶ。
- 相手に特定の行動をさせたり、現在取っている行動を禁止することは比較的単純です。
- 次のような状況をモデル化します。
- 相手に特定の行動をさせようとするが、なかなか言うことを聞かない。
- 特定の行動を禁止しようとするが相手がある程度逆らって言うことを聞かない。

❐17.2 幼稚園の先生の苦労をモデル化する(1)

- 先生は、みんなを東向きに歩かせようとして、あちこち走り回りながら、自分の周囲の子供たちを東に向かわせる。
- しかし、子供たちは方向を適当に変えてしまう。

- 新規モデル作成　モデル名：kindergarten
- Universeの下にplayground空間（デフォルト）を生成
- playground空間にエージェントteacher（20名）とkid（200名）を生成
- マップ出力設定（エージェントを●で表示、それぞれ色を変える）
- 大きな幼稚園で、子供たちが歩き回っている

❐17.2 幼稚園の先生の苦労をモデル化する(2)

- 初期設定
  - teacherもkidもランダムに配置、向きもランダム

```
def agt_init(self):
    self.x = rand() * 50
    self.y = rand() * 50

    self.direction = rand() * 360

```

- kidの行動ルール

```
def agt_step(self):
    self.turn(rand() * 60 - 30)
    self.forward(0.5)


```

❒17.2 幼稚園の先生の苦労をモデル化する(3)

- teacherの行動ルール

```
 7 def agt_step(self):
 8 
 9     #ランダムで動き回る
10     self.turn(rand() * 60 - 30)
11     self.forward(1)
12 
13     # 周りにいるkidを捕まえて東を向かせる
14     neighbor = self.make_agtset_around_own(2, False, agttype=Universe.playground.kid)
15     for one in neighbor:
16         one.direction = 0
17 
```

❐17.3 状況の把握を容易にする(1)

- 先生の言いつけを守っている子供＝水色
  先生の言いつけを守っていない子供＝紫
  - 整数型変数colorをkidに追加
  - マップ出力を変更（エージェントの色の変数指定）
  - 以下のルールをkidに付加

```
 7  def agt_step(self):
 8      self.turn(rand() * 60 - 30)
 9      self.forward(0.5)
10
11      if self.direction  < 30 or 330 < self.direction:
12          self.color = COLOR_CYAN     # 東を向いていればシアン
13      else:
14          self.color = COLOR_MAGENTA      # そうでなければマゼンタ
15
```

❐17.3 状況の把握を容易にする(2)

- 先生の言うことを聞いている子供たちの割合を集計
  → 時系列グラフに出力
  - Universeに整数型変数followを追加
  - Universeのルールに下記を書込
  - 時系列グラフの出力設定
    - 100 * Universe.follow / 200

```
 8  def univ_step_begin(self):
 9      pass
10
11  def univ_step_end(self):
12      Universe.follow = 0
13
14      # 先生の指示に従って東を向いている生徒の数を数える
15      total = make_agtset(Universe.playground.kid)
16      for one in total:
17          if one.direction < 30 or 330 < one.direction:
18              Universe.follow = Universe.follow + 1
19
20  def univ_finish(self):
21      pass
22
```

❒「kindergarten」のモデル

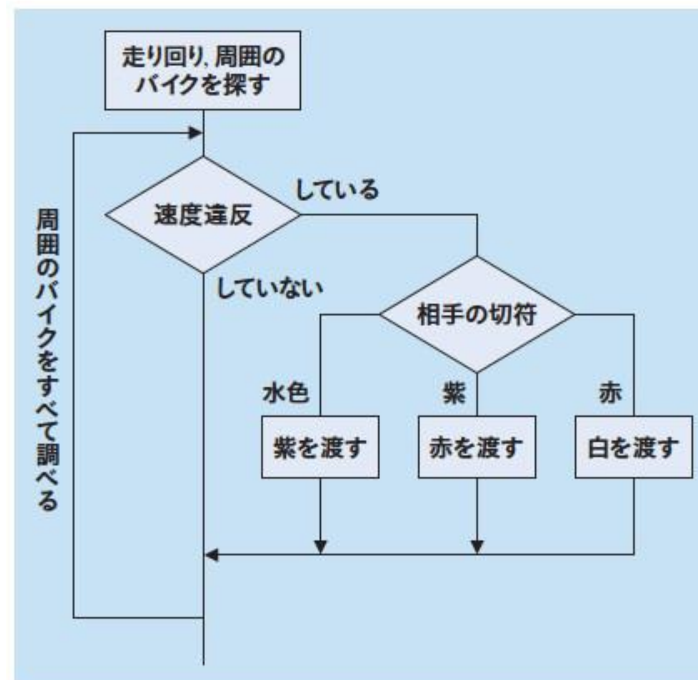
途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/USSPSYC4QiiT_PO3m5J5Tw
  - モデル名「kindergarten」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐17.4 暴走族の取り締まりをモデル化する（1）

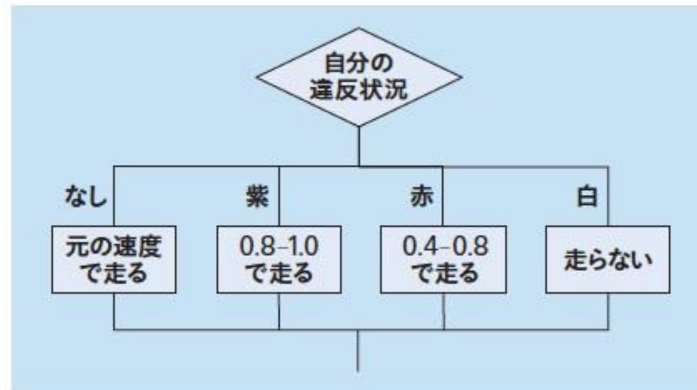
- 禁止（と不服従）の例です
  - 町中を暴走族が我が物顔にバイクを乗り回しているので、それを警官が取り締まろうとしている。
  - スピード違反のバイクを捕まえて、反則切符を渡す。
  - 罰則はだんだん厳しくなるが、暴走族はなかなかめげない。

- 警官のフローチャート
  - パトロールしながら、スピード違反のバイクを捕まえる。
  - 違反者に応じた反則切符を切る。



❐17.4 暴走族の取り締まりをモデル化する（2）

- 暴走族のフローチャート
  - 反則切符の厳しさに応じて、バイクのスピードを落とす。
  - 素直に制限速度以下にはしない。



- 新規モデル作成　モデル名：crackdown
- Universeの下にstreets空間（デフォルト）を生成
- streets空間にエージェントbiker（100人）とpolice（20人）を生成
- マップ出力設定（エージェントを矢印で表示、それぞれ色を変える）
  - bikerの色表示はfineによる変数指定とする

❐17.4 暴走族の取り締まりをモデル化する（3）

- 警官のルール
  - 空間全体にランダムに配置、動く方向もランダム
  - 動く速さ：0.1（各ステップ1動くことが時速100kmに相当）
  - 制限速度は60km/h（つまり0.6）
  - 反則の段階：水色（ゼロ）、紫色（1回）、赤色（2回）、白色（3回）
    - ➡ 3度めの違反で白色になり、マップ上から見えなくさせる

```
def agt_init(self):
    self.x = rand() * 50
    self.y = rand() * 50
    self.direction = rand() * 360

def agt_step(self):
    self.forward(0.1)
    self.turn(rand() * 60 - 30)
    gang = self.make_agtset_around_own(0.5, False, agttype=Universe.streets.biker)
    for kid in gang:
        if kid.speed > 0.6:
            kid.speed = 0
            if kid.fine == COLOR_CYAN:
                kid.fine = COLOR_MAGENTA
            elif kid.fine == COLOR_MAGENTA:
                kid.fine = COLOR_RED
            elif kid.fine == COLOR_RED:
                kid.fine = COLOR_WHITE

```

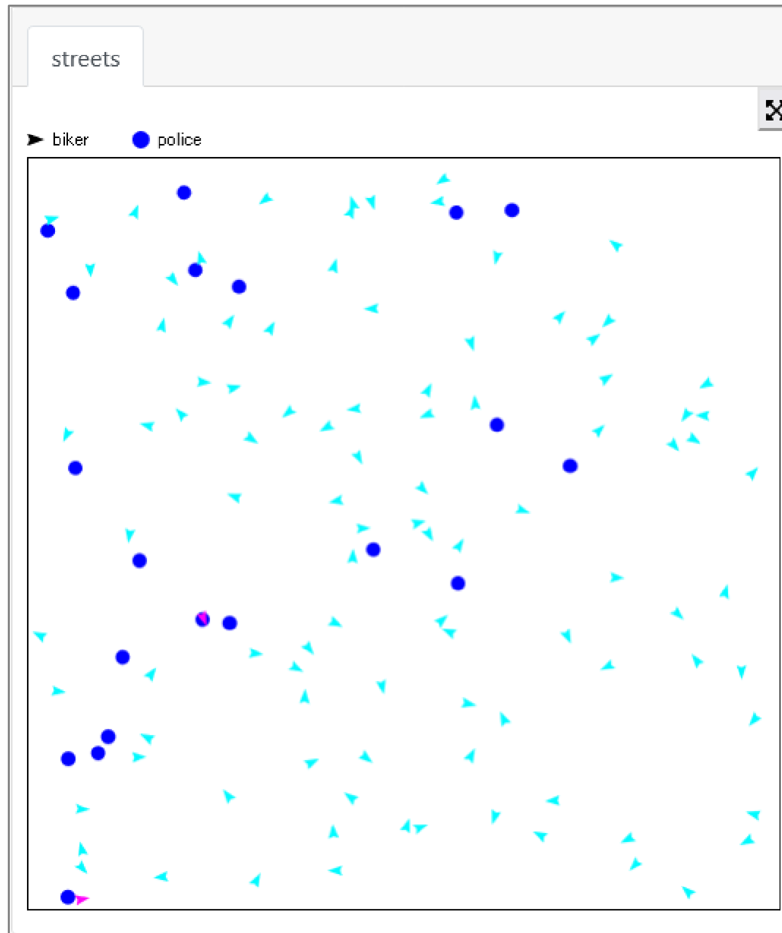
❐17.4 暴走族の取り締まりをモデル化する（4）

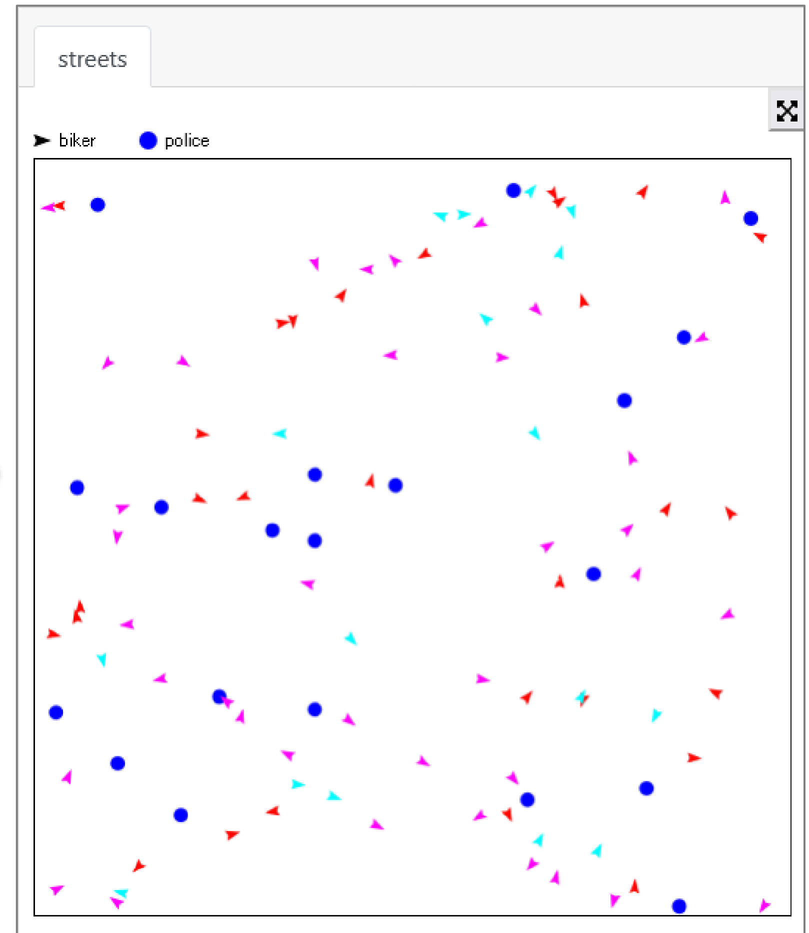
- 暴走族のルール
  - 自分の属性の変化に応じて自分の行動を変える
  - 空間全体にランダムに配置、動く方向もランダム
  - 動く速さ：1

```
def agt_init(self):
    self.x = rand() * 50
    self.y = rand() * 50
    self.direction = rand() * 360
    self.speed = 1
    self.fine = COLOR_CYAN

def agt_step(self):
    if self.speed == 0:
        if self.fine == COLOR_MAGENTA:
            self.speed = 1 - rand() * 0.2
        elif self.fine == COLOR_RED:
            self.speed = 0.8 - rand() * 0.4

    self.forward(self.speed)

```

❑17.4 暴走族の取り締まりをモデル化する（5）

- 実行してみましょう
  - 水色が徐々に赤色に変わっていきます





❐「crackdown」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/PBHCTERURLu7My8vtNuPBg
  - モデル名「crackdown」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐17.5 取り締まりの効果を調べる

- 暴走族の取り締まり状況をリアルタイムで見てみましょう
  - モデル名：crackdownを継承して新規作成し、crackdown(B)にリネーム
  - Universeの下に追加し、時系列グラフで出力
    - 平均時速：avspeed
    - 暴走族の人数：active
  - 暴走族がいなくなった段階で、シミュレーションを終了

```
 8  def univ_step_end(self):
 9      Universe.avspeed = 0
10      Universe.active = 0
11  
12      gang = make_agtset(agttye=Universe.streets.biker)
13      for kid in gang:
14          if kid.fine != COLOR_WHITE:
15              Universe.avspeed += kid.speed
16              Universe.active += 1
17  
18      if Universe.active == 0:
19          print('No More Active Bikers')
20          exit_simulation()
21      else:
22          Universe.avspeed = 100 * Universe.avspeed / Universe.active
23  
```

❐「crackdown(B)」のモデル

<u>途中でついていけなくなった人・モデルがうまく動かない人</u>

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/V9imXm3BQ4uQVR_Vmgsjkw
  - モデル名「crackdown(B)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐17.6 命令よりは同調が重要？

- 他のエージェントに働きかける問題と他のエージェントから影響を受ける問題
  - モデル名：kindergarten(B)を継承して新規作成し、kindergarten(C)にリネーム
  - teacherの行動ルールはそのまま
  - kidの行動ルールを記述
    - 1. 周囲の仲間を調べる（視野の広さ1）
    - 2. 周囲に仲間がいて、そのほとんど（たとえば8割）が東の方を向いていれば、自分もそうする

```
 7 def agt_step(self):
 8 
 9     neighbor = self.make_agtset_around_own(1, False, agttype=Universe.playground.kid)
10     num = count_agtset(neighbor)
11     if num == 0:
12         self.turn(rand() * 60 - 30)
13     else:   # 周囲に子供がいれば
14         count = 0   # 念のために初期化する
15         for one in neighbor:    # 周囲の子供の状態を調べる
16             if one.color == COLOR_CYAN:
17                 count += 1
18 
19         if count / num >= 0.8:  # 従順な子供が多ければ
20             self.direction = 0
21             self.turn(rand() * 30 - 15)
22         else:   # さもなければ
23             self.turn(rand() * 60 - 30)
24 
25     self.forward(0.5)
26 
27     if self.direction  < 30 or 330 < self.direction:
28         self.color = COLOR_CYAN     # 東を向いていればシアン
29     else:
30         self.color = COLOR_MAGENTA      # そうでなければマゼンタ
31 
```

❒「kindergarten(B)」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/MpsDPyJHRW6FgOvfgu9UkQ
  - モデル名「kindergarten(B)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐新しく学んだ事項

- 文法的には新しく登場したものはありません
- 他のエージェントの属性の複雑な変え方

第18章：空間を「場」として利用する

❐18.0 空間には意味があります

- 空間に役割や機能を持たせることができます。
- 空間変数という新しいタイプの変数を設定します。
- 空間上の「立ち位置」でエージェントが影響を受けます。
- 「場」や「勾配」をマップ出力で色の違いで可視化できます。
- 空間変数の値は、シミュレーションの過程で変化可能です。
- 深海魚のモデルは小手調べです。
- 空気感染をモデル化しましょう。

❒18.1 「場」や「勾配」は難しくない

- これまで
  - 空間は単なる2次元の座標系

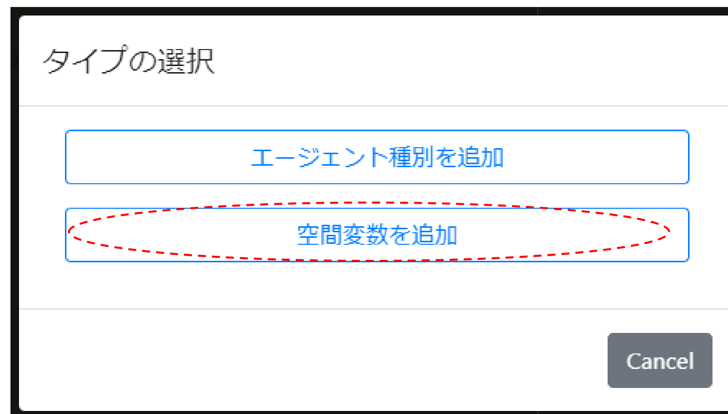
  - 空間は「場」や「勾配」という性質を持つことも
    - 肥沃な農地、地価の高い都心、フェロモンの濃い部分、近隣地点との値の差（勾配） etc.

- 「場」や「勾配」の情報を2次元空間上に表現可能
  - e.g., 地形図で等高線を引く

❐18.2 空間変数を利用する(1)

- 空間に変数を設定し、その変数値を利用してマップ上に色で表現

- 海がだんだん深くなる、を表現
  - 新規モデル作成　モデル名：ocean
  - Universeの下にsea空間（デフォルト）を生成
  - 空間に空間変数depthを追加
    - ➡「空間変数」= 空間に直接追加する変数



❐18.2 空間変数を利用する(2)

- マップ出力
  - マップ出力設定>マップ要素リストで空間変数を追加
  - 要素名: depth
  - 表示色: グラデーション指定, 対象変数: depth
  - 変数範囲 0<= x <= 100, 対応色を水色から青色に変化するように設定

マップ出力設定

マップ名: ocean
空間: ocean
レイヤ番号: 0
凡例表示:
背景画像:
固定画像　クリックして画像ファイルを選択、またはファイルをドラッグ＆ドロップしてください。
変数指定
背景色: 255,255,255
原点位置: 左上　左下
罫線表示: なし　チェス型　囲碁型

X軸設定
最小値: 0
最大値: 51

Y軸設定
最小値: 0
最大値: 51

※ 連続空間モデルの場合、マップの出力サイズは[空間のサイズ+1]となります

マップ要素リスト　エージェント　+
エージェント
空間変数

Cancel　OK

マップ要素設定（空間変数）

要素名: depth

表示色
グラデーション指定
対象変数　depth
変数範囲　0　≦ X ≦　100
対応色　選択　選択
固定色　0,0,0
変数指定　指定しない

透明度
固定値　0
変数指定　指定しない

変数表示
変数値を表示する
対象変数　指定しない
小数の表示桁数　0　桁
表示色　0,0,0

Cancel　OK

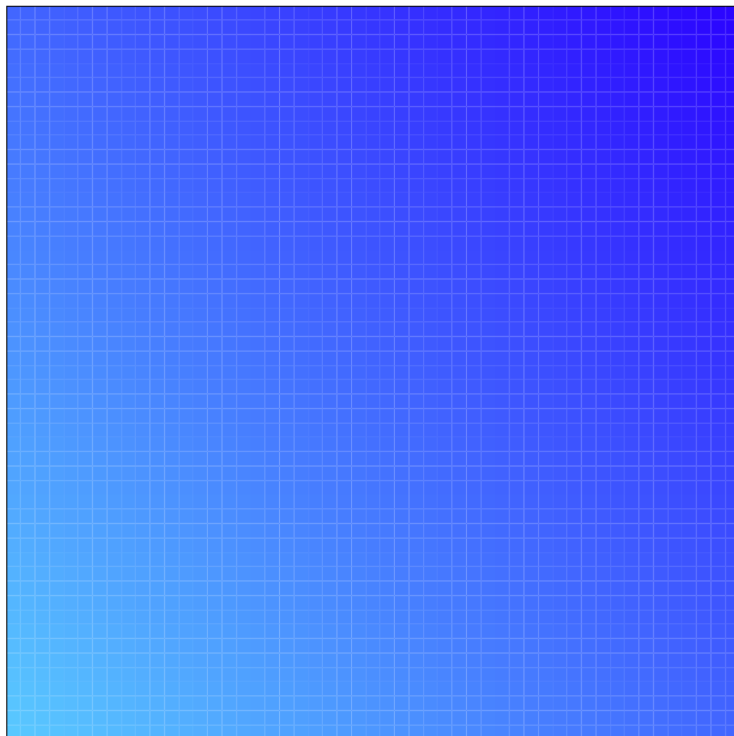
❒18.2 空間変数を利用する(3)

■ ルール記述

```
def univ_init(self):
    for x in range(51):
        for y in range(51):
            Universe.sea.depth[x, y, 0] = x + y

```

左下が0、
右上に向けて100に
なるまでなだらかに変化

■ 保存・実行

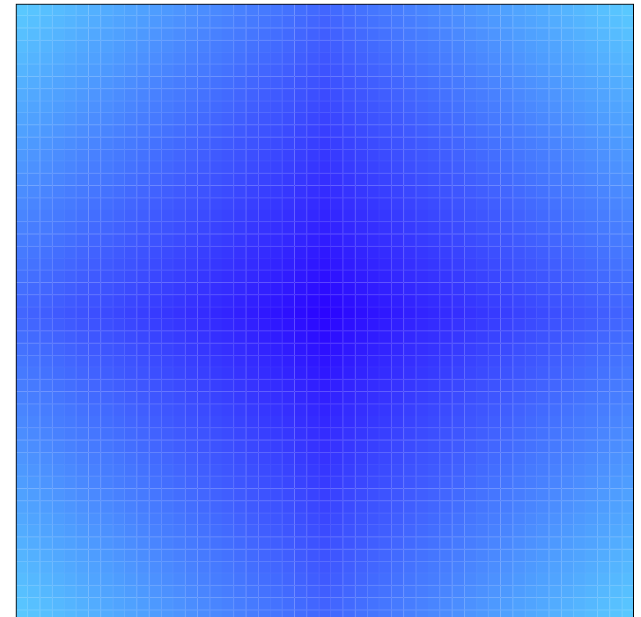


❐18.3 深海魚の泳ぐ海

- モデルを継承、モデル名「ocean B」で保存
- 海の深さを中央部分が最も深く、周辺に行くに従って浅くなるように設定する

```
def univ_init(self):
    for x in range(51):
        for y in range(51):

            k = abs(x - 25) * 2
            l = abs(y - 25) * 2
            Universe.sea.depth[x, y, 0] = 100 - k - l

```

- abs()は絶対値をとるというルール
- k は x が25の時に最小値
- l は y が25の時に最小値
- 100 – k – l は x と y が25の時に最大値100



❐18.4 深海魚の行動(1)

- 空間seaにエージェントfishを追加
  - univ_initで10匹のfishエージェントを作成
  - マップ出力にfishを追加（▲にする）
    - fishを前面に出すように順番をfish→depthとする

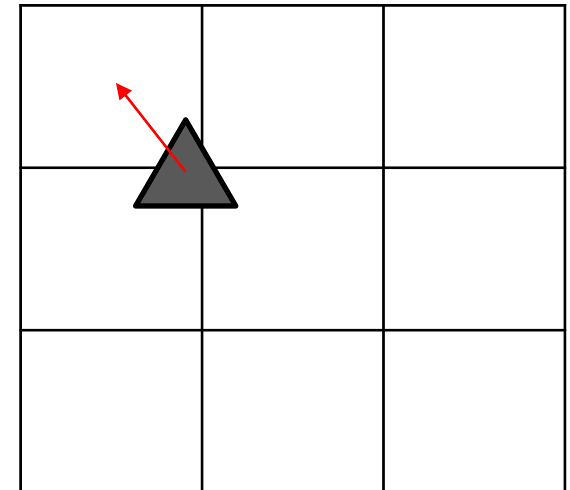
- 深いところで生活する魚（深海魚）は、浅い方へ泳いでいくと反転する
  - 初期設定
    - 中央に配置、ランダムな方向を向く

```
1 def agt_init(self):
2 
3     # 中央に配置
4     self.move_center()
5 
6     # ランダムな方向を向く
7     self.direction = rand() * 360
8 
```

self.move_center()
空間の中央にエージェントを配置する関数

❒18.4 深海魚の行動(2)

```
 9 def agt_step(self):
10
11     xi = int(self.x)
12     yi = int(self.y)
13
14     if Universe.sea.depth[xi, yi, 0] <= 60:
15         self.turn(180)
16     else:
17         self.turn(rand() * 40 - 20)
18
19     self.forward(0.5)
20
21
```



- 魚のいる海の深さは近くの空間座標で近似
  - ➡ fishの位置座標は実数型だが、空間変数depthの位置座標は整数型
  - ➡ int() 関数で、実数を整数に変換（切り捨て）して参照

❐「ocean B」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/dSji0LJITgqIEjGTkj6OLA
  - モデル名「ocean B」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒18.5 空気感染をモデル化する(1)

- 空間変数の値がシミュレーション中に変化するモデルを作る

- 患者は自分の近くに病原体を残す
  - 健康人は自分の近くに病原体が多いと病気になる
  - 病原体は徐々に減っていく
    - ➡ 第13章の「Infection C 13.5終了時点」をもとに作成　influenzaモデル

❐「influenza」のモデル

※「infection C 13.5終了時点」を「influenza」とリネーム

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/sMB9p7nBT3CdniW_7HVGKQ
  - モデル名「influenza」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐18.5 空気感染をモデル化する(2)

- influenzaモデルの改造
  - 空間societyに空間変数virusを追加
  - Universeとpersonの初期設定
    - univ_Initとagt_initをそのまま利用
  - マップ出力を変える
    - virus空間変数を出力
    - 最小値0で白色、最大値5で紫色にする

❐18.5 空気感染をモデル化する(3)

- 患者は病原体を空間に残し、健康人は病原体の数によって病気になったりならなかったりする

```
 4  def agt_step(self):
 5  
 6      xi = int(self.x)
 7      yi = int(self.y)
 8  
 9      if self.condition == COLOR_RED:
10          Universe.society.virus[xi, yi, 0] = Universe.society.virus[xi, yi, 0] + 3
11          self.cure = self.cure + 1
12      else:
13          if Universe.society.virus[xi, yi, 0] * rand() > 2:
14              self.condition = COLOR_RED
15  
16      # +10 ～ -10で向きを変えて移動
17      self.turn(rand() * 20 - 10)
18      self.forward(1)
19  
20      # 7ステップ経過したら全快
21      if self.cure >= 7:
22          self.condition = COLOR_CYAN
23          self.cure = 0
24  
```

❐18.5 空気感染をモデル化する(4)

- 空間上の病原体が徐々に減っていく

```
22 def univ_step_end(self):
23
24     # 空間変数のウイルスを減少させる
25     for x in range(get_width_space(Universe.society)):
26         for y in range(get_height_space(Universe.society)):
27             if Universe.society.virus[x, y, 0] > 0:
28                 Universe.society.virus[x, y, 0] = Universe.society.virus[x, y, 0] - 1
29
30     Universe.number = 0
31     people = make_agtset(space=Universe.society)
32     for one in people:
33         if one.condition == COLOR_RED:
34             Universe.number = Universe.number + 1
35
```

❐「influenza 18.5終了時点」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/1neivodnQW-SGyCri2OZOQ
  - モデル名「influenza 18.5終了時点」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

□新しく学んだ事項

- 直接、空間に変数を追加する（空間変数の追加）。
- 空間変数は空間に格子上に設定されている（整数座標しか取れない）　。
- ループしている空間の勾配をなめらかに設定する技巧的な方法
- シミュレーションの実行中に空間変数の値を変える。
- 空間変数の値を色でマップ出力する。
- マップ要素リストの順番の意味と順番の変更法
- 空間変数のいくつかの利用方法
- move_center()
- abs()

第19章：同期問題に注意する

❒19.0 シミュレーションの中の時刻と時間を意識しましょう

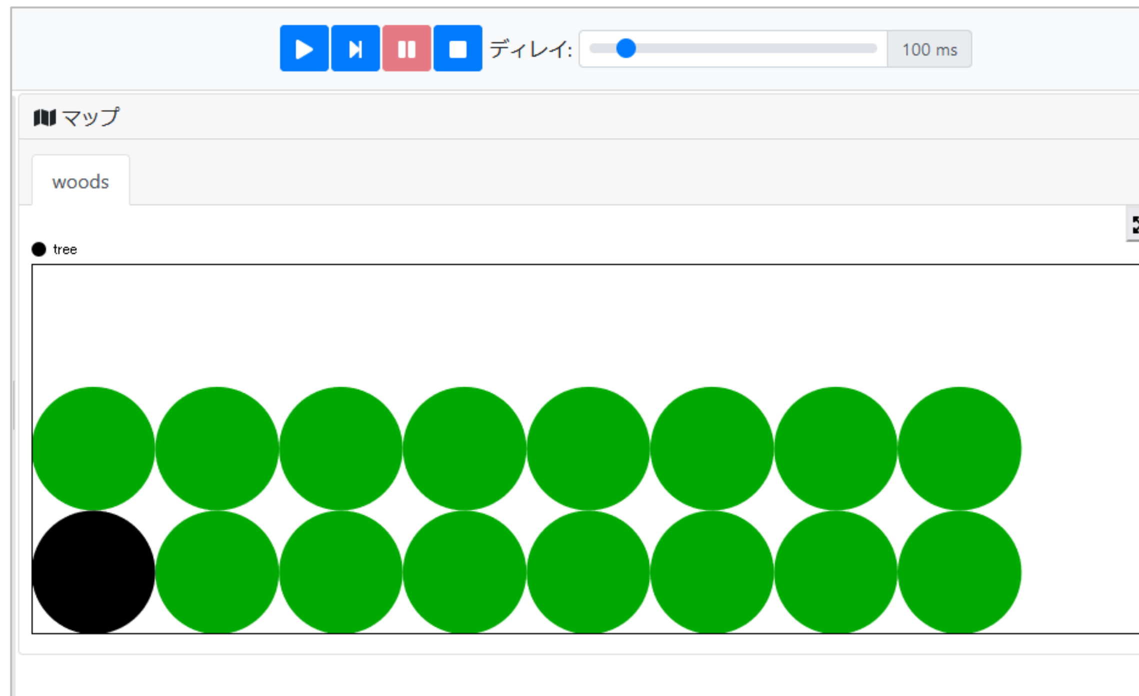
- シミュレーションの中の時刻と時間に注意する必要のある場合があります。
- 注意するポイントは「同期」という問題です。
- 沢山いるエージェントの「実行順序」の問題でもあります。
- 過去を現在に（現在を未来に）つなげるエージェントの「記憶」の問題でもあります。
- 簡単なモデルで、問題の所在をはっきりさせます。
- 問題を解決するには2つのやり方があります。

❐19.1 シミュレーションの中の時刻と時間

- MASの「同期問題」
  - MASでは、エージェントが「勝手に」どんどん相互作用
    →各ステップで生じる相互作用は複雑化
  - 同時に行動することが実際にどのような現象になっているかを理解する必要がある
  - さらに、多くのエージェントがホントに「同時」に行動しているようにルールを書かなければならない

❐「森林火災 初期サンプルモデル」のモデル

- 下記のモデルにアクセスしモデルを継承してください
  - ❑ https://artisoc-cloud.kke.co.jp/models/dsKv7a7pTpqSEeZoCTwgEw
  - ❑ モデル名「森林火災 初期サンプルモデル」で公開しています
  - ❑ 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る



❐19.2 なぜ「同期」が重要なのか(1)

- 山に木が生えている
- 山火事が起こると、燃えている木に隣接する木にも火が燃え移り、木が次々と燃えていく
  - Universeの下に空間woods（8×2、ループせず）を生成
  - woodsの下にエージェントtreeを生成（エージェント数0）
  - treeに整数型変数colorを追加
  - マップ出力設定
    - ➡ treeの色指定をcolorにする

❒19.2 なぜ「同期」が重要なのか(2)

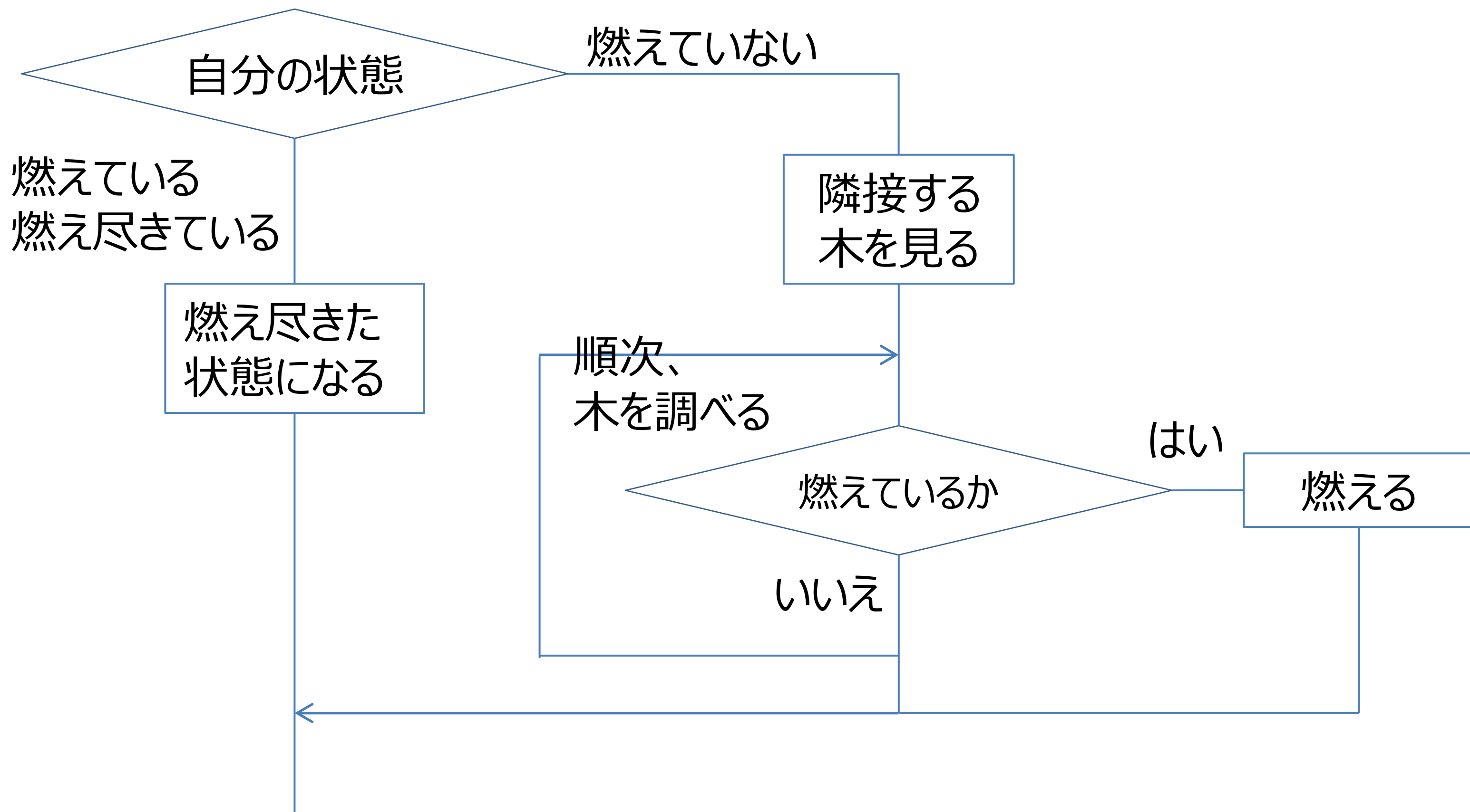
```python
def univ_init(self):

    # シミュレーション初期化時に実行する処理
    for i in range(8):
        for j in range(2):
            one=create_agt(Universe.woods.tree)
            one.x=i
            one.y=j
            if i==0 and j==0:
                one.color=COLOR_RED
            else:
                one.color=COLOR_GREEN

def univ_step_begin(self):

    # シミュレーションのステップ開始毎に実行する処理
    pass

def univ_step_end(self):

    # シミュレーションのステップ終了毎に実行する処理
    pass

def univ_finish(self):

    # シミュレーションのステップ終了毎に実行する処理
    pass

```

8×2の空間にぎっしりと木が生えている状況

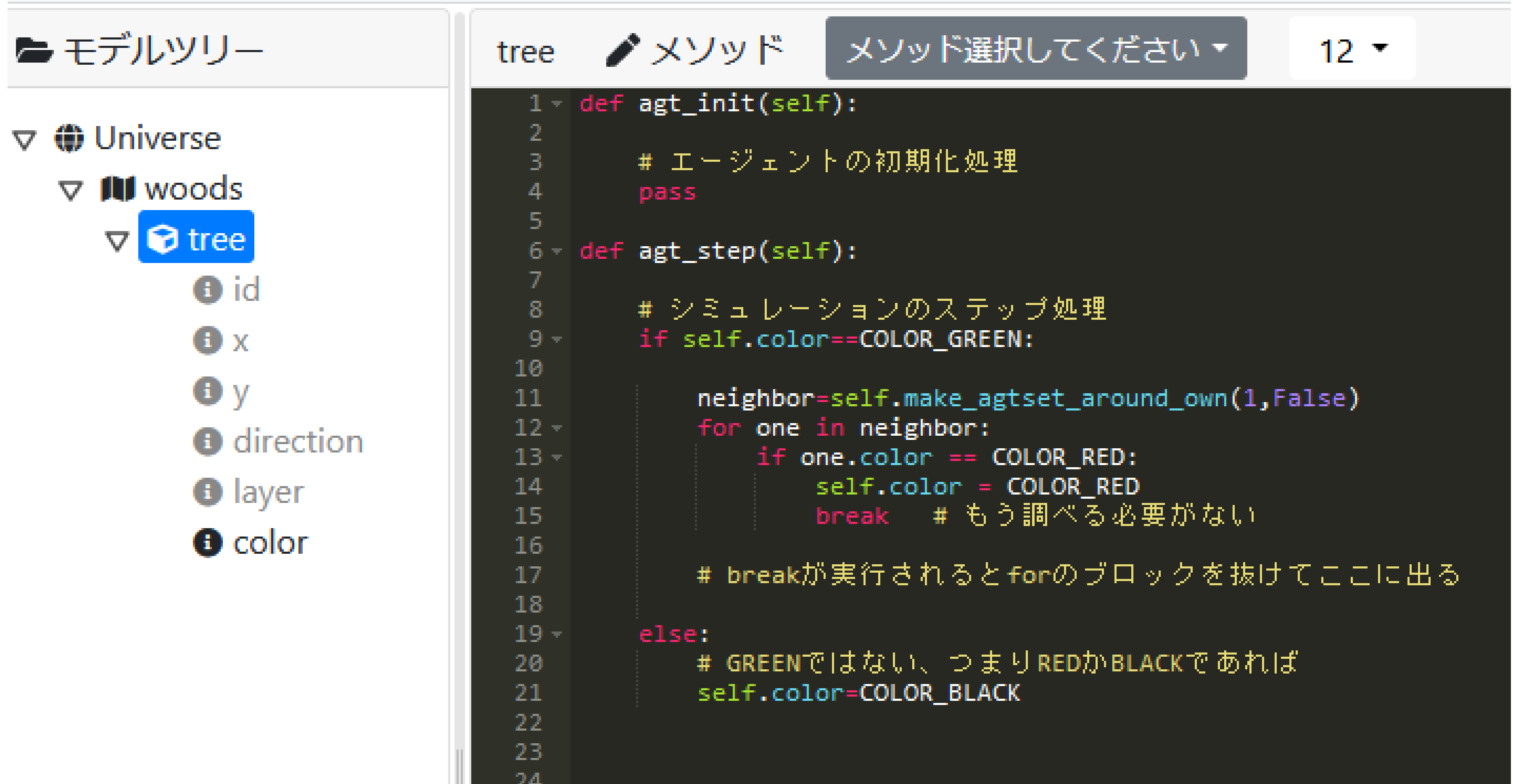
左下の木が一本だけ燃えている状態

❑19.2 なぜ「同期」が重要なのか(3)

■ 隣接している木が燃えていれば火が燃え移り、
燃えている木は燃え尽きる



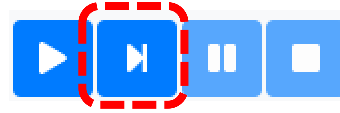
❐19.2 なぜ「同期」が重要なのか(4)



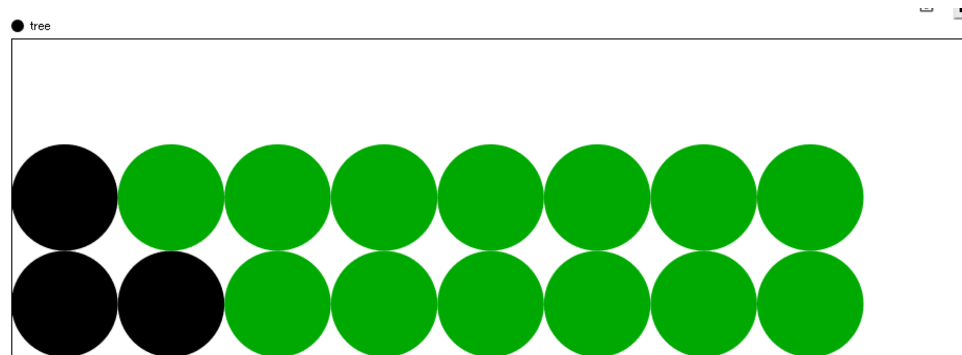
break
　ただちに繰り返し文による繰り返し作業を中止するルール

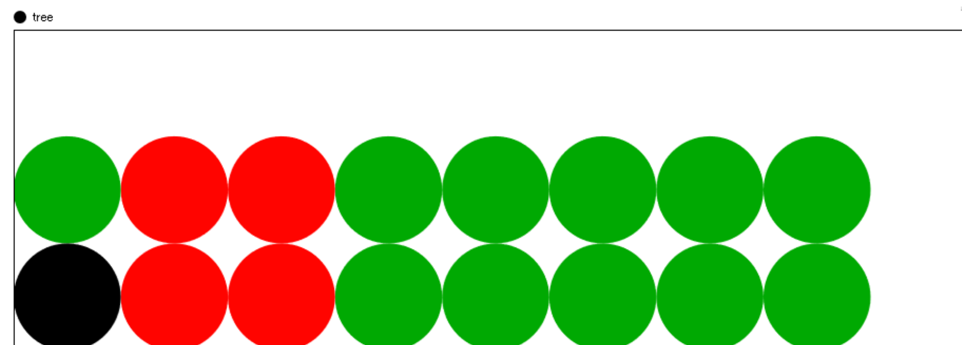
❐19.2 なぜ「同期」が重要なのか(5)

- ステップ実行ボタンを1回ずつ押してみる。



- 実際のシミュレーション結果は、「燃えている木に隣接する木に火が移り、次々と木が燃えていく」という想定とは違う結果に。
- なぜ、こうなるのか考えてみましょう。





□「森林火災 ステップの最後に一括実行」のモデル

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/fOryKS_6TKO6WsccnqKlqg
  - モデル名「森林火災 ステップの最後に一括実行」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐19.3 artisoc Cloudはなにをやっているのか

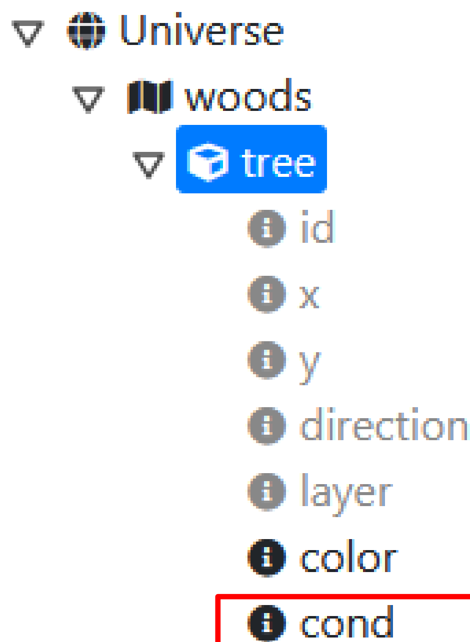
- artisoc Cloudは、ルールを実行するエージェントをランダムに決める
  - agt_stepは⓪～⑮までのエージェントが、ランダムに決められた順番で実行
    - ⓪よりも先に①⑧が実行→①と⑧は赤くなり燃え移っていく
    - ①⑧よりも先に⓪が実行→⓪は燃え尽きて①⑧には燃え移らない
    - ①から⑮、そして⓪の順に実行→一瞬で全部燃え尽きてしまう

- →　同期をとる必要（以下の2つのやり方）
  - A. 状態の変化を直ちに実行せず、ステップの最後に一括して実行
  - B. 現在の状態ではなく、1ステップ前の状態に応じて、状態を変化

⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮
⓪	①	②	③	④	⑤	⑥	⑦

❐19.4 最後にまとめて状態を変える（1）

- treeに、「次の瞬間」にどのような状態になるのかを示す整数型変数condを追加
- ステップの最後にすべてのエージェントの状態を一括で変更するため、直接colorを変えずに、変数condに状態を設定




```
def agt_init(self):

    # エージェントの初期化処理
    pass

def agt_step(self):

    # シミュレーションのステップ処理
    if self.color==COLOR_GREEN:
        self.cond=COLOR_GREEN
        neighbor=self.make_agtset_around_own(1,False)
        for one in neighbor:
            if one.color==COLOR_RED:
                self.cond=COLOR_RED
                break
    else:
        self.cond=COLOR_BLACK

```

❒19.4 最後にまとめて状態を変える（2）

- Universeのステップの最後に、すべてのエージェントのcondをcolorに設定し、すべてのエージェントの状態を一括で変更

```
18 def univ_step_end(self):
19 
20     # シミュレーションのステップ終了毎に実行する処理
21     total=make_agtset(Universe.woods.tree)
22     for one in total:
23         one.color=one.cond
24 
```

❐19.4 最後にまとめて状態を変える（3）

- 状態変化先延ばし法・一括置換法

1. エージェントに次の瞬間に変わる状態を指定しておく変数（「バックアップ変数」）を追加し、行動ルールとしてはその変数を変化させて、現在の状態を表す変数は変化させない。

2. 最後に、一括してバックアップ変数を、状態を表す変数に代入する

❐「森林火災 過去の状態を参照」のモデル

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/EE0OvU7NRs6YCTsrl4Nwhg
  - モデル名「森林火災 過去の状態を参照」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❐19.5 過去の状態を参照する（1）

- 「一瞬前」の周囲や「一瞬前」の自分の状態をもとにして「現在」の自分の状態が決まる、という方法
  - 「過去参照」法、「因果的状態変化」法

- get_history()という関数を用いて、過去のエージェントの状態を認識する

❒19.5 過去の状態を参照する（2）

- Universe
  - woods
    - tree
      - id
      - x
      - y
      - direction
      - layer
      - color

```
def agt_init(self):

    # エージェントの初期化処理
    pass

def agt_step(self):

    # シミュレーション開始前のステップを指定するとエラーになるので
    # 最初のステップは何もせずに関数を抜ける
    if count_step() <= 1:
        return

    # 2ステップ目以降であれば
    past1=get_history(self, 1) # 1ステップ前の自分をpast1にセット

    if past1.color==COLOR_GREEN: # 1ステップ前の色が緑であれば

        neighbor=self.make_agtset_around_own(1,False)

        for one in neighbor:

            # 周囲のエージェントの1ステップ前の状態をpast2にセット
            past2=get_history(one,1)

            # 1ステップ前が赤であれば、自分を赤に
            if past2.color==COLOR_RED:
                self.color=COLOR_RED
                break
    else:
        self.color=COLOR_BLACK

```

❑新しく学んだ事項

- 同期問題
- 最後に一括置換する解決法
- バックアップ変数
- 過去を参照する解決法
- 変数に過去の値を記憶させておく方法
- 過去の変数の価を取り出す方法
- get_history()
- 繰り返し文を中止させるbreak

第20章：コンウェイの「ライフゲーム」をつくる

❒20.0 artisoc Cloudのパワーを実感できます

- こんなに簡単にできてしまってよいのでしょうか。
- 同期と簡単な場合分けを使うだけです。
- 新しく学ぶ技法は全然ありません。
- 少し工夫を加えましょう。

□20.1 ライフゲームとは

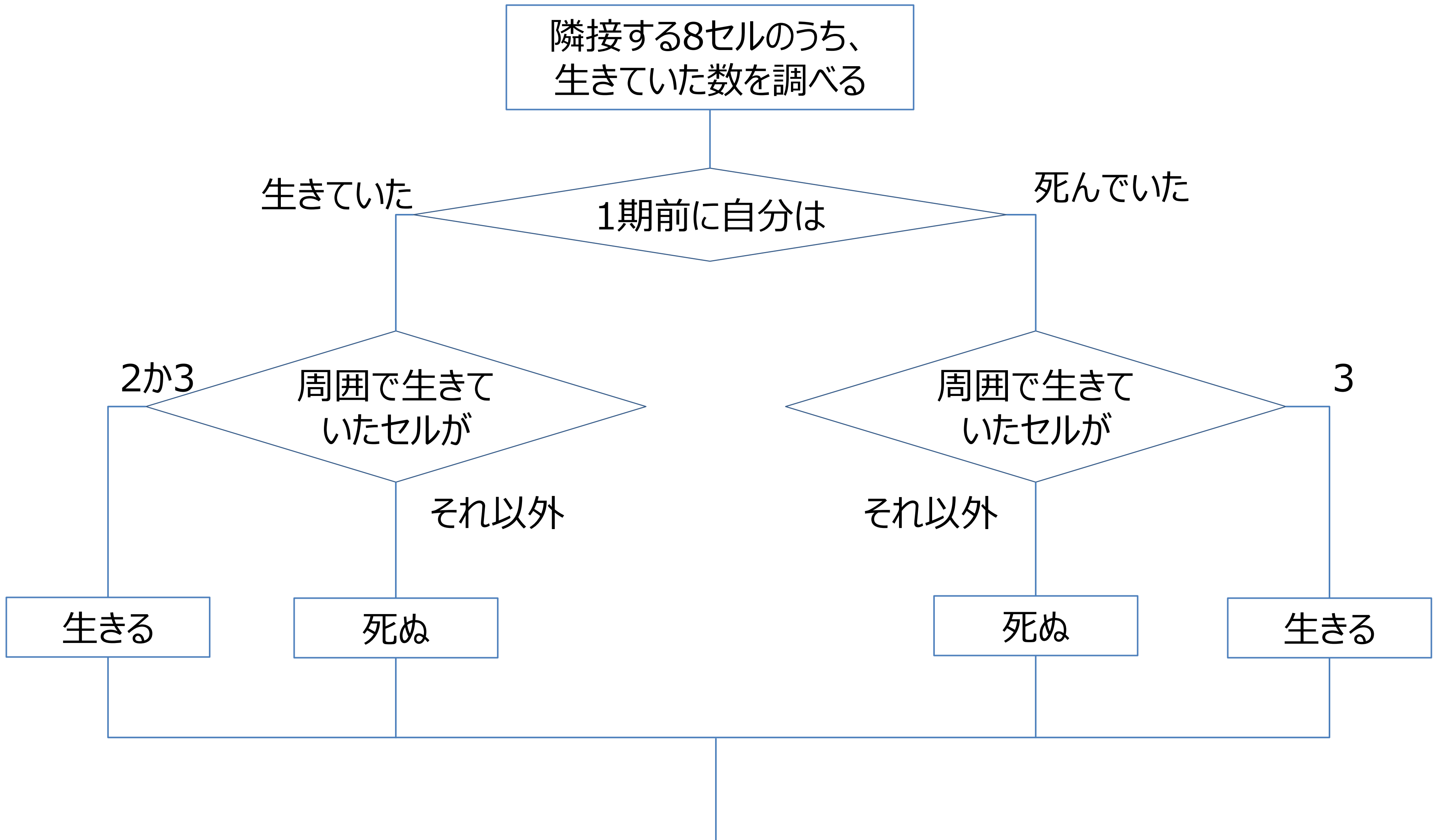
- 数学者ジョン・コンウェイによって考案された著名な人工生命モデル
  - 1960年代末に考案

- 各セルは0か1の2つの状態（生か死か）しかとらず、
  状態が変化するかどうかは隣接する8セルで決まるモデル
  - ➡ セルは混雑していても、過疎でも死んでしまい、
    適度な密度の時にしか生きられないという考え方。
  - ➡ 結果、1辺が2の正方形で安定したり、
    セルが3つ並んだ長方形が縦の状態と横の状態を繰り返したり、
    様々な形状と変化を見せることに。
  - ➡ セルが沢山生き残っていてもすぐに全滅したり、
    少数のセルしか生き残っていないのにしぶとく生き残ったりすることも。

□20.2 ライフゲームのモデル化(1)

- 平面にびっしりと格子型（セル型）のエージェント（以下、セル）を並べる
- 各セルは生と死という2つの状態をとる
- 各セルは、周囲（ムーア近傍）の8セルの状態に応じて、自分の状態を変える
- 今、生きているセルは、周囲のセルのうち2または3個が生きていれば生き続けるが、それ以外の場合は死ぬ
- 今、死んでいるセルは、周囲のセルのうち3個が生きているときのみ、生きる状態になる（誕生する）が、それ以外の場合は死んだままである

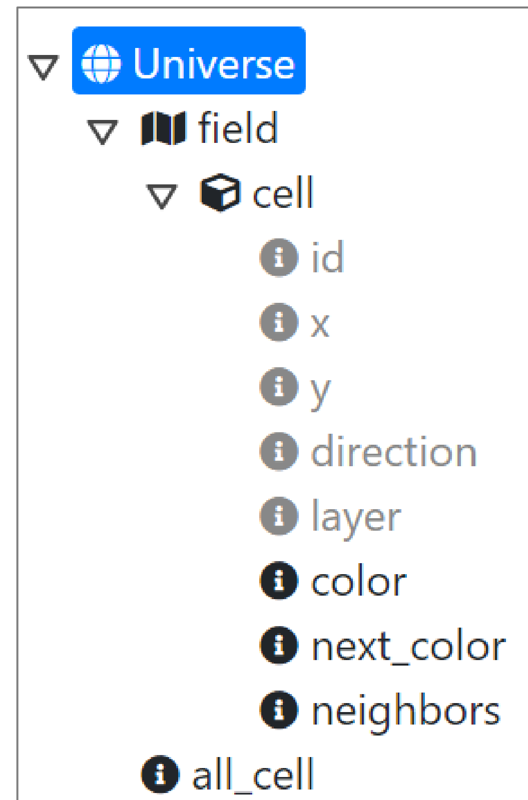
□20.2 ライフゲームのモデル化(2)



□20.2 ライフゲームのモデル化(3)

- 新規モデル作成　モデル名： game-of-life
- Universeの下に空間field（四角格子空間, 25×25、ループあり）を生成
- Universeの下にall_cellを追加（全cell）
- fieldにエージェントcellを生成
  - 変数colorを追加
    - ※生きていたら水色、死んでいれば白色とする
  - 変数next_colorを追加（同期一括実行用）
  - 変数neighborsを追加（お隣さんを記憶）

- マップ出力の設定
  - cellの出力
    - 変数指定でcolorを選択
    - 形を四角■にする



❐20.3 セルを貼り付ける

```
def univ_init(self):
    for x in range(25):
        for y in range(25):
            one = create_agt(Universe.field.cell)
            one.x = x
            one.y = y

            if rand() < 0.2: # セルを敷き詰める。20%で生セル
                one.color = COLOR_CYAN
            else:
                one.color = COLOR_WHITE


    # 自分の周りのセルをセットする
    self.all_cell = make_agtset()
    for one in self.all_cell:
        one.neighbors = one.make_agtset_around_own_sqgrid(1, False)

```

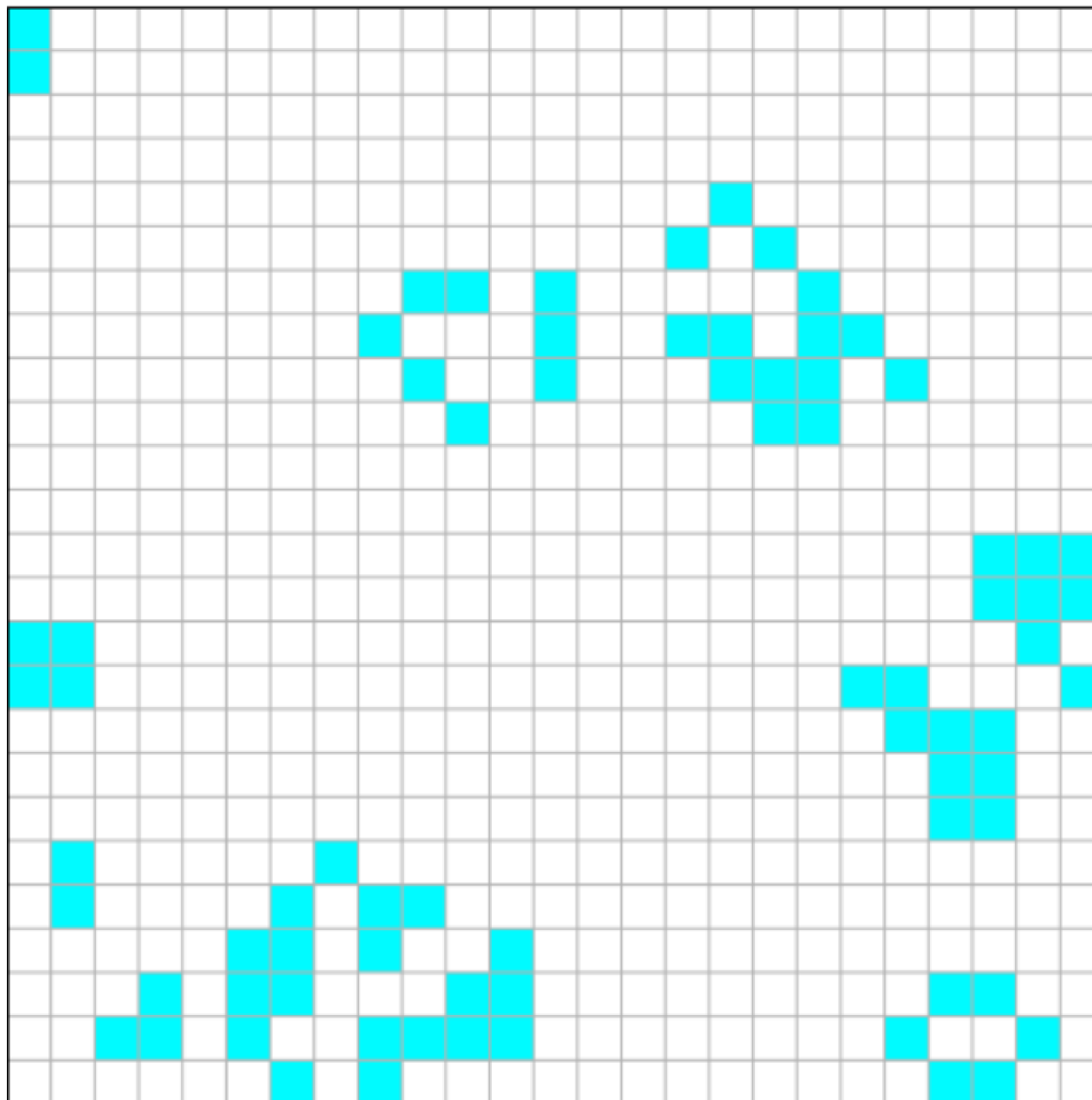
□20.4 ライフゲームのロジック

```
1  def agt_init(self):
2      pass
3
4  def agt_step(self):
5      if count_step() > 1:
6          alive = 0
7          for one in self.neighbors:
8              if one.color == COLOR_CYAN:
9                  alive += 1  #周りの生きているセルをカウント
10
11         if self.color == COLOR_CYAN: # 自分が生きている場合
12             if alive == 2 or alive == 3:
13                 self.next_color = COLOR_CYAN # 生きたまま
14             else:
15                 self.next_color = COLOR_WHITE # 死ぬ
16         else: # 1期前に自分が死んでいる場合
17             if alive == 3:
18                 self.next_color = COLOR_CYAN # 生きる（誕生する）
19             else:
20                 self.next_color = COLOR_WHITE # 死んだまま
21
```

❐20.5 自動的に終了させる

```
19  def univ_step_begin(self):
20      pass
21
22  def univ_step_end(self):
23      if count_step() > 1:
24          stop = True # 変化を検出するフラグ Trueなら変化なし
25
26          # step_endで全てのセルを更新して同期
27          for one in self.all_cell:
28              if stop and one.color != one.next_color:
29                  stop = False
30              one.color = one.next_color
31
32          if stop:
33              print(count_step(), "ステップで終了しました。")
34              exit_simulation()
35
36  def univ_finish(self):
37      pass
38
```

□実行してみよう



❐「game-of-life」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/zPzVgQKyRpW49ybcUU2x5A
  - モデル名「game-of-life」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る