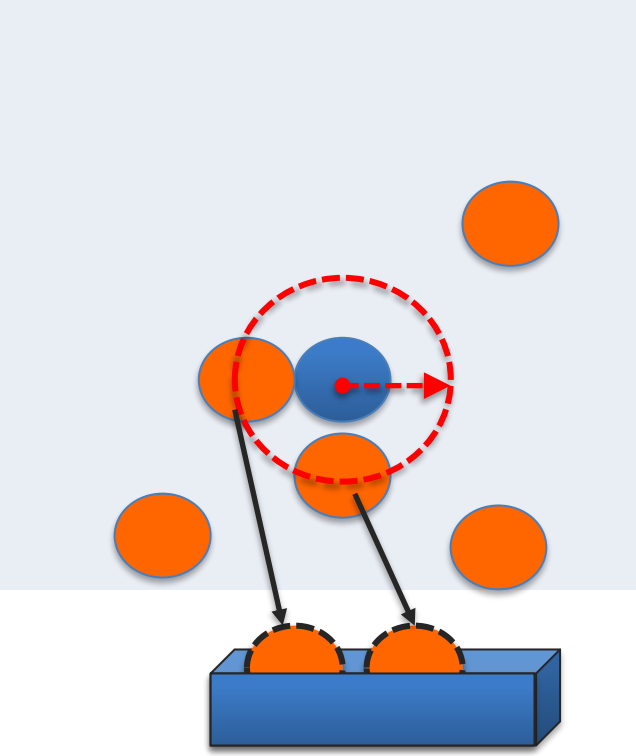
チュートリアル

チュートリアル1:
エージェントを複雑な条件で選別する

❒エージェントを複雑な条件で選別する

- マルチエージェント・シミュレーションでは、エージェントどうしに相互作用させる際に、ある条件を満たした一部のエージェントを対象に特定の行動ルールを適用することが必要な場合が生じます。
- つまり、エージェントをまとめたり選別したりして、さまざまな「エージェントの集まり」を思いどおりに作れることが重要になります。



エージェント集合型変数
(複数のエージェントを格納するための変数)

❒既に学んできた集合

- neighbors = self.make_agtset_around_own(2, False)
  - neighborというエージェント集合型変数の中に、自分を中心にして視野2の範囲に存在しているエージェントを全てリストアップする操作。
  - 特定の条件を満たした「エージェントの集まり」を作っている。

- one = randchoice(neighbors)
  - neighborにリストアップされているエージェントの集まりから、ランダムにエージェントを1つ選び出す。

- 「自分との距離が2よりも遠く4以内のエージェント」をどうやって選び出せばよいか？
  - ①視野4の範囲のエージェントを選び出す。
  - ②視野2の範囲のエージェントを選び出す。
  - ①から②を差し引く。

  - 「エージェントの集まり」を利用して特定の条件に当てはまるエージェントを選別するさまざまな技法を学ぶ。

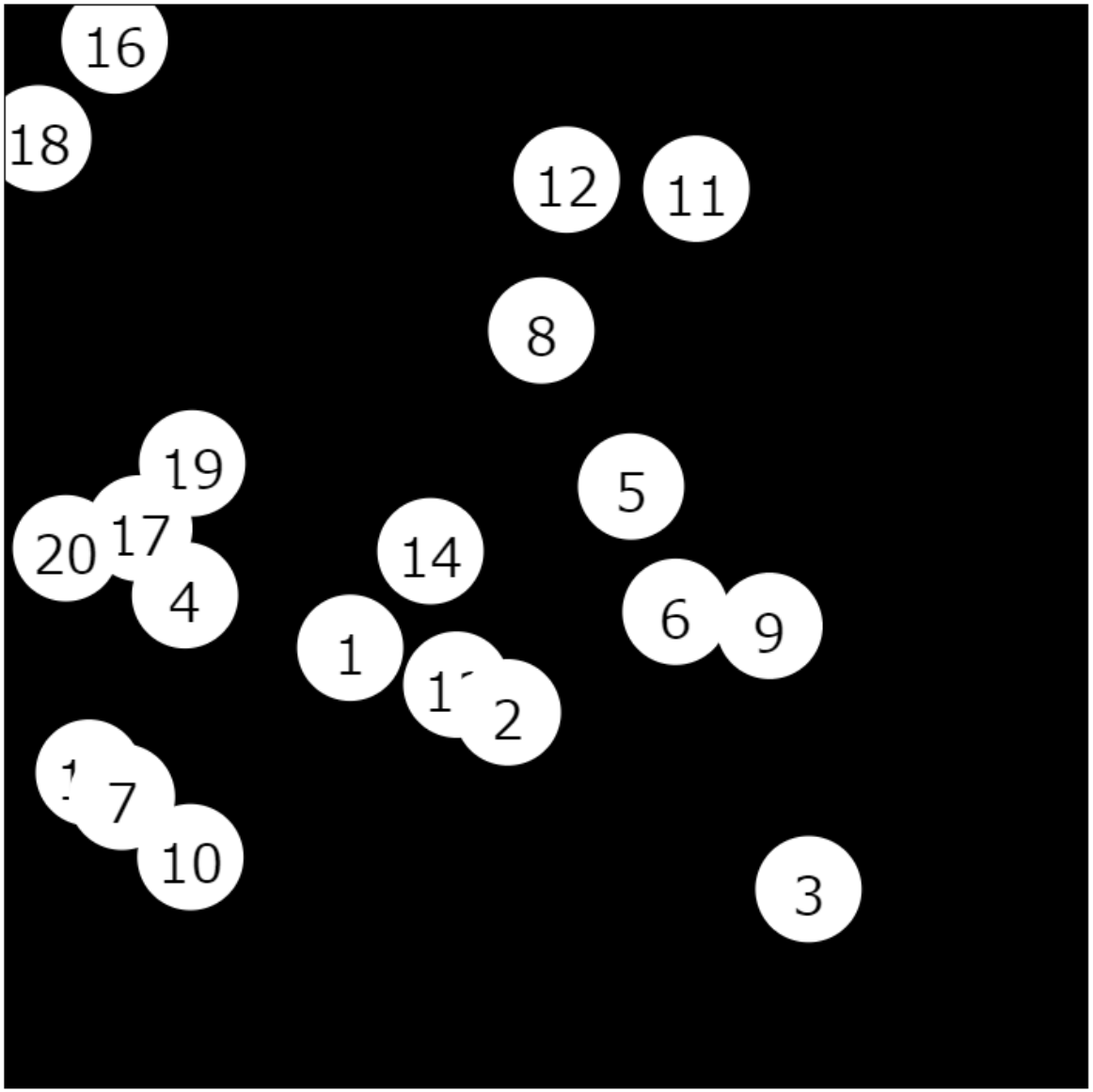
❐「エージェントを複雑な条件で選別する」のモデル

- 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/rUS-jVmORQW6UdRDVHSf9Q
  - モデル名「エージェントを複雑な条件で選別する」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

□マップ、ルール等の編集

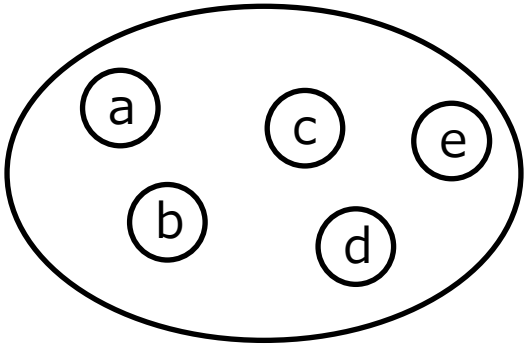
- hirobaマップを編集
  - 背景色を黒に設定する。（右上図）
    - ➡ マップ出力設定>「背景色」を「0,0,0」に設定
  - マップ要素リスト hitoを編集する。（右下図）
    - ➡ マップ出力設定＞ マップ要素リスト＞ hito＞ 拡大率＞ 固定値 「 5 」に設定
→マップ上のアイコンを大きくする。
    - ➡ マップ出力設定＞ マップ要素リスト＞ hito＞ エージェント情報の表示＞ 表示する変数 「number」を選択
→hitoのnumberを表示する。
- hitoエージェントのルールを編集
  - エージェントの色を白色にする。
    - ➡ 「agt_init」で、エージェントの色をCOLOR_WHITE に設定
- 実行時のディレイを500msぐらいにする。

❑実行時の画面

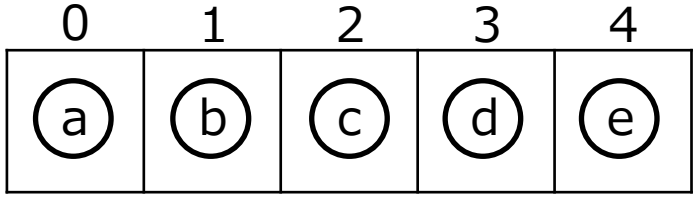


❑いろいろな集合型

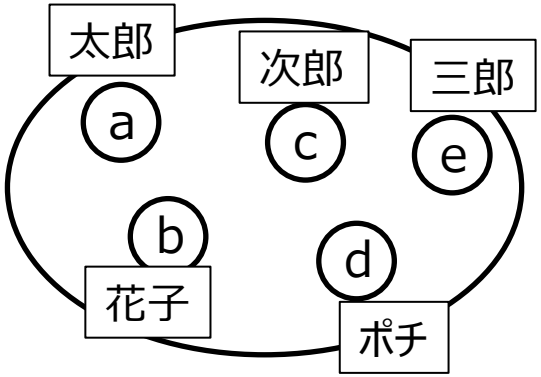
- set型
  - 順番のないユニークな値の集まり。
  - 集合演算が可能。速い。
  - 取り出しの順序が保証されない。
  - エージェント集合はset型。



- list型
  - 順番を付けて並べる値の集まり。
  - インデックスを用いて要素を抽出できる。
  - 並べ替えができる。
  - 要素の追加などは遅い。



- dict型
  - キーと値の組で表されるデータを格納する。
  - 要素は順序を持たない。
  - キーを指定して値を高速に取り出せる。

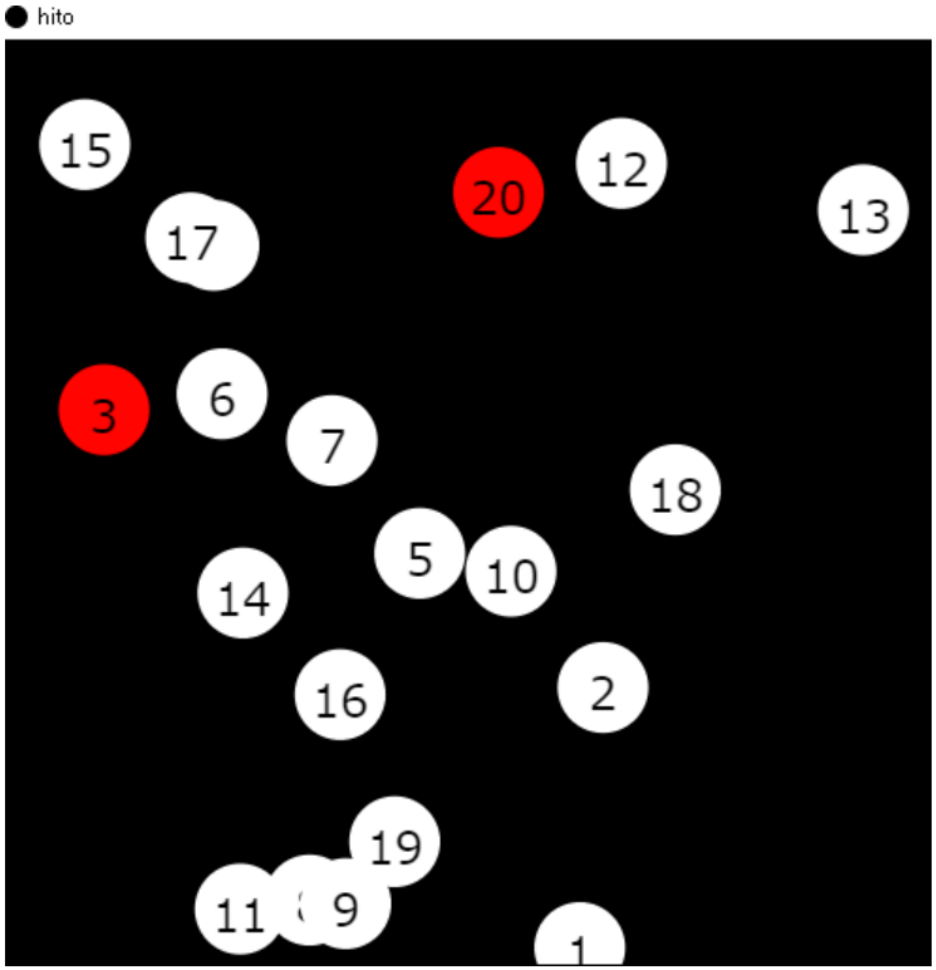


❑setからlistを作る

- make_agtlist：エージェント集合（set型）をlist型に変換する。
- people_list[0]：listの0番目の要素を参照する（0番目のスロットに入っているエージェント）。

```
def univ_init(self):

    for i in range(Universe.ninzu):
        one = create_agt(Universe.hiroba.hito)
        one.number = i


    # エージェント集合（set型）
    people_set = make_agtset(space=Universe.hiroba)

    # エージェント集合 set型をlist型に変換する
    people_list = make_agtlist(people_set)

    # 0番と3番を赤にする
    people_list[0].color = COLOR_RED
    people_list[3].color = COLOR_RED
```

❑実行してみる



0番と3番を赤にしたはずだが…
→変換前のsetには順序が無いので、
　変換後のlistでも順番に整列できていない。

❒listを特定の変数で並べ替える

- sort_agtlist（並べ替えるlist, “並べ替えのキーとなる変数”）
  - list型のエージェント集合を並べ替えて、list型で返す。

```
def univ_init(self):

    for i in range(Universe.ninzu):
        one = create_agt(Universe.hiroba.hito)
        one.number = i


    # エージェント集合（set型）
    people_set = make_agtset(space=Universe.hiroba)

    # エージェント集合 set型をlist型に変換する
    people_list = make_agtlist(people_set)

    # id順で並び替える
    people_list = sort_agtlist(people_list, "id")

    # 0番と3番を赤にする
    people_list[0].color = COLOR_RED
    people_list[3].color = COLOR_RED
```

❑新しく関数を作る(1)

関数の定義

```
def 関数名(self, 引数, 引数, ...):
    関数で実行する処理
    ～～～ ～～～
    return ●●　#戻り値（無くても良い）
```

呼び出し

```
# Universeの関数呼び出し
hogehoge = Universe.関数名（引数）

# エージェント自身の関数呼び出し
hogehoge = self.関数名（引数,引数, ...）

# 他のエージェントの関数呼び出し
hogehoge = one.関数名（引数,引数, ... ）
```

※戻り値の受取、引数は無くても良い
※関数定義の引数名と、呼び出しの引数名が異なっていても良い

❑新しく関数を作る(2)

- def univ_finish(self)の後に新しい関数を追加する。
  - エージェントの色を初期化する関数を作ってみる。

```
31 def reset_color(self):
32
33     # エージェントの色をすべてリセットする
34     people = make_agtset(space=Universe.hiroba)
35     for one in people:
36         one.color = COLOR_WHITE
37
38     return "エージェントの色リセット完了!"
39
```

  - 引数のset内のエージェントの色を赤にする関数を作ってみる。

```
41 def set_colorred(self, people_set):
42
43     # 引数のSet内のエージェントの色をすべて赤に
44     for one in people_set:
45         one.color = COLOR_RED
46
```

❑setの操作　基本(1)

- univ_initの最後に追加して実行してみてください。

① 要素の追加

```
# 要素の追加
Universe.reset_color()

people = set()  #空のsetを作成
people.add( people_list[5] )
people.add( people_list[6] )
people.add( people_list[7] )

Universe.set_colorred(people) #setの中身のエージェントの色を赤に設定
```

② 要素の削除

```
# 要素の削除
Universe.reset_color()

people = set()  #空のsetを作成
people.add( people_list[5] )
people.add( people_list[6] )
people.add( people_list[7] )
people.discard( people_list[6] ) # 6番目の要素を削除

Universe.set_colorred(people) #setの中身のエージェントの色を赤に設定
```

❑setの操作　基本(2)

- univ_initの最後に追加して実行してみてください。

③ 要素を全削除

```
# 要素の全削除
Universe.reset_color()

people = set() #空のsetを作成
people.add( people_list[5] )
people.add( people_list[6] )
people.add( people_list[7] )
people.clear() #全削除

Universe.set_colorred(people) #setの中身のエージェントの色を赤に設定
```

④ setのコピー

```
# セットのコピー
Universe.reset_color()

people = set()  #空のsetを作成
people.add( people_list[5] )
people.add( people_list[6] )
people.add( people_list[7] )
people2 = people.copy()

Universe.set_colorred(people2) #setの中身のエージェントの色を赤に設定
```

❐setの操作　基本(3)

- univ_initの最後に追加して実行してみてください。

⑤ ランダムで一つ取り出す

```
# ランダムで一つ取り出す
Universe.reset_color()

one = randchoice(people_set)
one.color = COLOR_RED
```

⑥ランダムでサンプルを取り出す

- randsample(対象とするエージェント集合, 取り出すエージェントの個数)
  - エージェント集合からランダムに重複なく複数のエージェントを取り出す。

```
# ランダムでサンプルを取り出す
Universe.reset_color()

people = randsample(people_set, 5)
Universe.set_colorred(people)
```

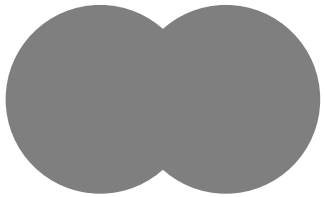
❑setの操作　集合操作(1)

- univ_initの最後に追加して実行してみてください。

① union: 2つのエージェント集合の和集合を取得する。

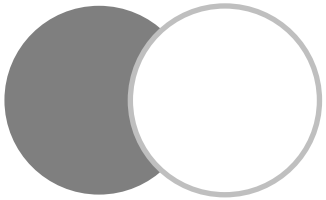
- { }の中に初期要素を指定してsetを作ることができます。

```
# union: 2つのエージェント集合の和集合を取得する
Universe.reset_color()

people1 = {people_list[5], people_list[6], people_list[7]}  #5,6,7のset
people2 = {people_list[6], people_list[7], people_list[8]}  #6,7,8のset
people3 = people1.union(people2) # 和集合を取得する
Universe.set_colorred(people3)
```



② difference: 2つのエージェント集合の差集合を取得する。

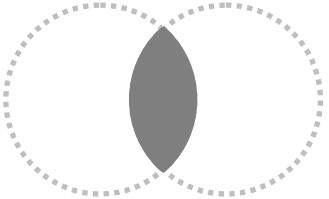
```
# difference: 2つのエージェント集合の差集合を取得する
Universe.reset_color()

people1 = {people_list[5], people_list[6], people_list[7]}  #5,6,7のset
people2 = {people_list[6], people_list[7], people_list[8]}  #6,7,8のset
people3 = people1.difference(people2) # 差集合を取得する
Universe.set_colorred(people3)
```



❑setの操作　集合操作(2)

- univ_initの最後に追加して実行してみてください。

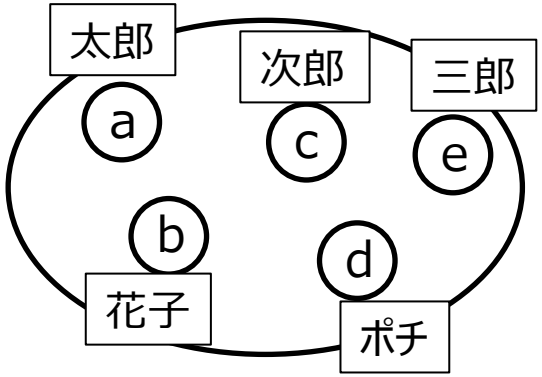
③ intersection: 2つのエージェント集合の積集合を取得する



```
# intersection: 2つのエージェント集合の積集合を取得する
Universe.reset_color()

people1 = {people_list[5], people_list[6], people_list[7]}  #5,6,7のset
people2 = {people_list[6], people_list[7], people_list[8]}  #6,7,8のset
people3 = people1.intersection(people2) # 積集合を取得する
Universe.set_colorred(people3)
```

❒dict型

- dict型の特徴
  - キーと値の組で表されるデータを格納する。
  - 要素は順序を持たない。
  - キーを指定して値を高速に取り出せる。



- dictの新規作成
  - {key:value, key:value, key:value, ...}
    - keyとvalueはどのようなオブジェクトでも良い。

※univ_initの最後に追加してください

```
101     # Dict 新規作成
102
103     Universe.reset_color()
104     people_dict1 = {"taro":people_list[5], "jiro":people_list[6], "saburo":people_list[7]}
105
106
107
```

❒dict型の操作(1)

- 値をすべて取り出す。
  - dict変数.items() で、すべてのkeyとvalueのペア集合を取り出すことができます。
  - forと組み合わせることで、すべてのkeyとvalueを参照することができます。

※univ_initの最後に追加してください

```
# Dictから値をすべて取り出す
Universe.reset_color()
people_dict1 = {"taro":people_list[5], "jiro":people_list[6], "saburo":people_list[7]}

for k,v in people_dict1.items():
    print("key:" + k + " value:" + str(v.number))
    v.color = COLOR_RED
```

❐dict型の操作(2)

■ 値を取り出す。　※univ_initの最後に追加してください

□ dict変数[key]で値を取り出すことができます。

➡ ※keyの要素が無ければエラーがでるので注意する。

```
Universe.reset_color()
people_dict1 = {"taro":people_list[5], "jiro":people_list[6], "saburo":people_list[7]}

# dictからキーが"taro"の要素を取り出し　→ 5番目の要素が返る
taro_value = people_dict1["taro"]
taro_value.color = COLOR_RED # 5番目の要素を赤に設定
```

■ キーと値の追加・更新

□ dict変数[key] = value　で新しい要素を追加できる。

```
# dictに要素追加
Universe.reset_color()
people_dict1 = {"taro":people_list[5], "jiro":people_list[6], "saburo":people_list[7]}

people_dict1["hanako"] = people_list[8] # 花子を追加

hanako_value = people_dict1["hanako"]
hanako_value.color = COLOR_RED
```

❒dict型の操作(3)

- 要素の削除

※univ_initの最後に追加してください

  - dict変数.pop(key)でkeyの要素を削除する。
    - ※keyの要素が無ければエラーがでるので注意する。

```
# dictから要素を削除
Universe.reset_color()
people_dict1 = {"taro":people_list[5], "jiro":people_list[6], "saburo":people_list[7]}

people_dict1.pop("taro")

for k,v in people_dict1.items():
    print("key:" + k + " value:" + str(v.number))
    v.color = COLOR_RED
```

- 要素の存在確認
  - key in dict変数でkeyが含まれるか確認できる。

```
# keyの存在確認
Universe.reset_color()
people_dict1 = {"taro":people_list[5], "jiro":people_list[6], "saburo":people_list[7]}

if "jiro" in people_dict1:
    print("次郎はdict内に存在")
```

❒まとめ

- いろいろな集合型
  - set, list, dict
  - setからlistを作る。
- 新しい関数を作る。
- setの操作を学ぶ　基本編
  - 要素の追加、要素の削除、要素を全削除、setのコピー
  - ランダムで一つ取り出す、ランダムでサンプルを取り出す。
- setの操作を学ぶ　集合操作
  - union、difference、intersection
- dictの使い方を学ぶ
  - dictの作成、値を取り出す、キーと値の追加・更新、要素の削除、要素の存在確認

説明したデータ型はすべてPython標準です
詳しくはPython公式ドキュメントを参照してください。
set:https://docs.python.org/ja/3/library/stdtypes.html#set
list: https://docs.python.org/ja/3/library/stdtypes.html#list
dict: https://docs.python.org/ja/3/library/stdtypes.html#dict

❐「エージェントを複雑な条件で選別する」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/x5zmE47LS_SLctCknlnb-g
  - モデル名「エージェントを複雑な条件で選別する 完成」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

チュートリアル2:
人のネットワーク上の情報伝搬モデルを作成する

□ネットワークについて

- ネットワーク→ものごとの関係性を示したシステム

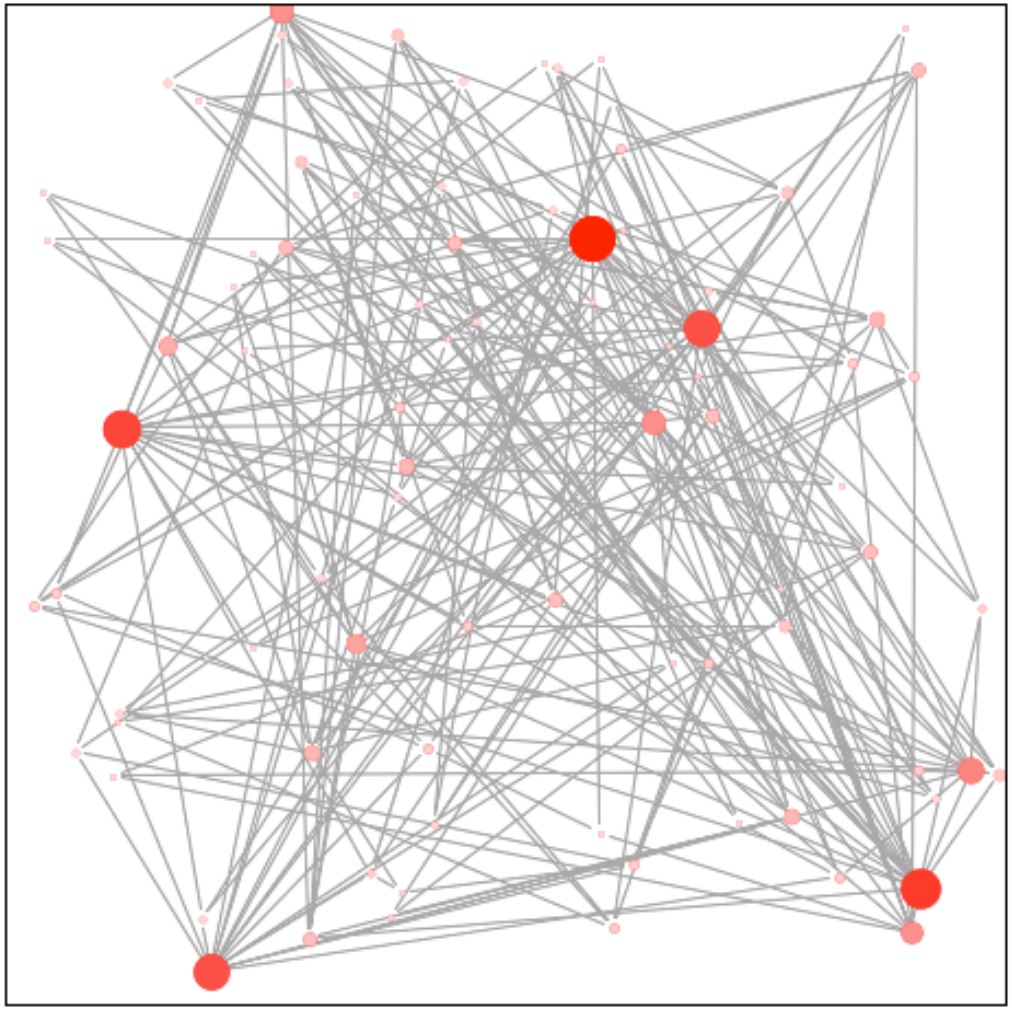
  例）
  - ある場所で一緒になった他人が実は知り合いの知り合いだった…。
  - 世間は一見巨大かつ複雑に見えて、案外狭いもの。

- 社会的なネットワーク分析：人同士の繋がりを分析
  - 現在ではインターネット、食物連鎖、さらには感染症の流行といった様々な現象に分析が適用されている。

- 理工的なネットワーク分析：人以外の繋がりを幾何学的に分析
  - ネットワーク分析の基礎にはグラフ理論があり、研究の歴史はこちらの方が長い。
  - 現在は脳科学や物理学、都市の交通網などに適用。

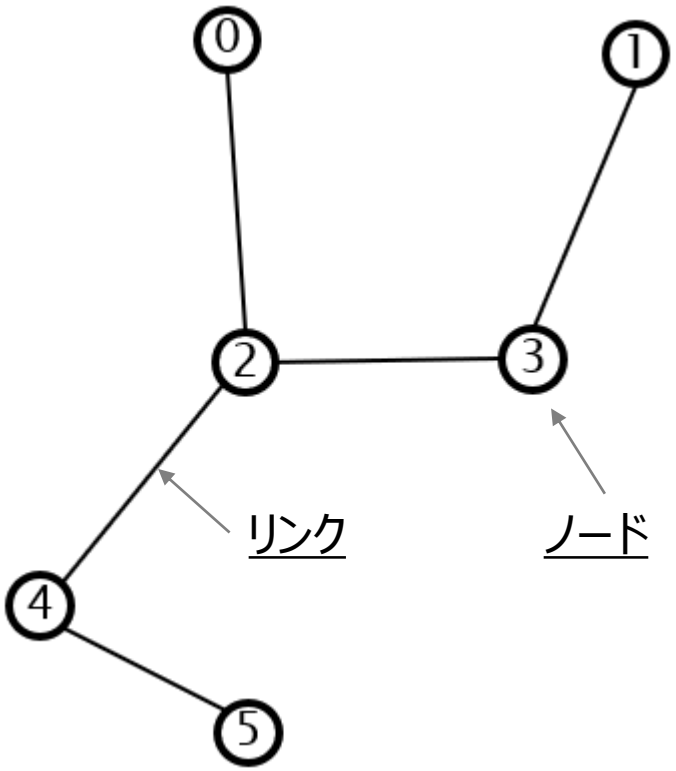
❒ 人のネットワーク上の情報伝搬モデルを作成する

- 人同士の繋がりを想定したネットワーク上における情報伝搬モデルを作成し、社会的ネットワークの性質の一端に触れる。



□ネットワークの考え方（1）

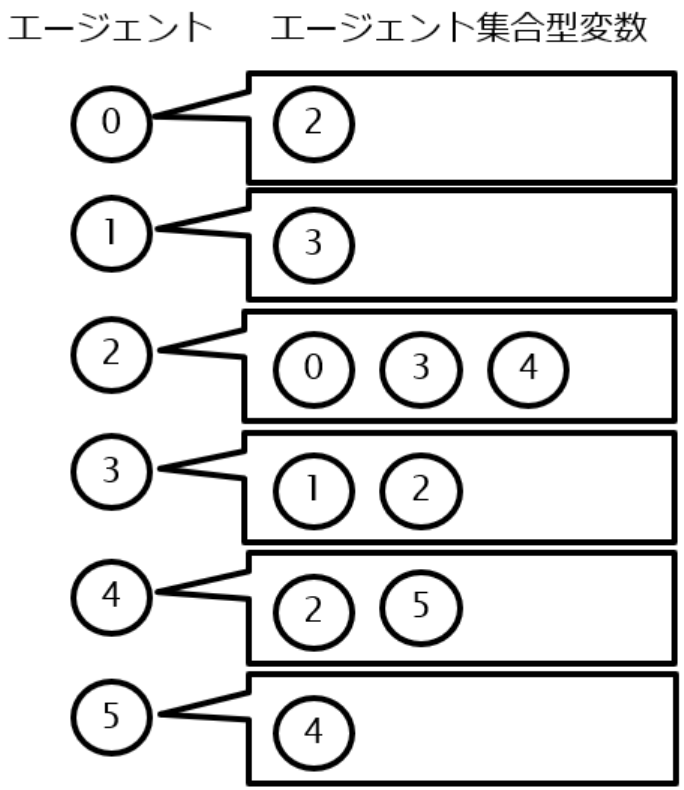
- たとえば、下図のようなネットワークを考えましょう。数字の入った丸が人を、線が人同士の繋がりを表しています。たとえば0さんと2さんは線で繋がっているので、2人は知り合いであるという想定です。



ネットワーク科学の用語では、
この図における人を**ノード**、
繋がりを**リンク**（または**エッジ**）と呼ぶことが多い。
道路ネットワークではノードを交差点と呼ぶ。

❑ネットワークの考え方（2）

- artisoc Cloudでこれを表現するには、ノードをエージェント、リンクをエージェントの持つエージェント集合型変数として定義します。
- 例えば、エージェント2が持つエージェント集合型変数には、エージェント0, 3, 4が格納されています。
- こうすることで、エージェント2がエージェント0, 3, 4と繋がっていることが表現できます。



□ネットワークの作成（1）

- 「新規モデルの作成」 モデル名「人のネットワーク」
- Universeの下に空間を追加
  - 空間名：city（他はデフォルト）
- cityの下に「person」エージェントを追加
  - エージェントには「color」と「link」の変数を追加
    （後ほど、繋がっているエージェントをこの「link」に格納）
- ユニバース変数「num」を作成
- 出力設定
  - マップ出力を追加
    - 空間：「まち」という名前でcityを選択
    - マップ要素リスト：「人」という名前でpersonを選択
      - ☑ エージェントの表示色は変数指定の「color」
- コントロールパネル
  - 「人数」という名前でnumを表示
    - 種類はスライドバー、値の型は整数、範囲は1～50、初期値は20
    - 目盛り間隔 1

□ネットワークの作成（2）

- ネットワークを見やすくするため人を円周上に均等配置
  - 円周のサイズはマップサイズの0.4倍
    - ➡ r = 0.4 * get_width_space(Universe.city)
      - ☑ get_width_space(Universe.city)：cityの横幅
      - ☑ get_height_space(Universe.city)：cityの縦幅

```
def univ_init(self):

    # マップの中心の座標の取得
    cx = get_width_space(Universe.city) / 2
    cy = get_height_space(Universe.city) / 2
    # 演習の半径の定義
    r = 0.4 * get_width_space(Universe.city)

    # Universe.numの数だけ、personを生成し、円周上に均等配置する
    for i in range(Universe.num):
        p = create_agt(Universe.city.person)
        theta = radians(360 * i / Universe.num)
        p.x = cx + r * cos(theta)
        p.y = cy + r * sin(theta)

```

❑ネットワークの作成（3）

- personのlink属性を初期化

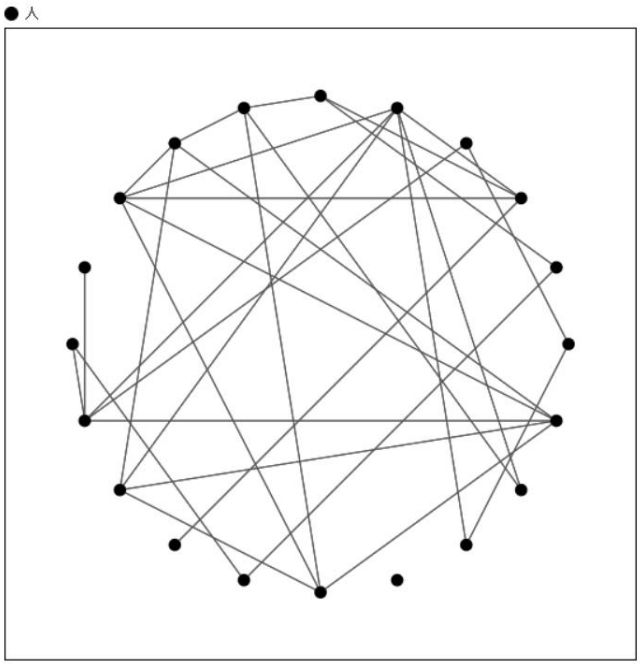
```
1 def agt_init(self):
2     self.link = set()
3
```

空の集合型変数setをセット
空箱を用意したイメージ

- link属性にはエージェントと繋がりのある他のエージェントを格納する。
- ここではself.link = set() として、linkが集合型変数であることを宣言。

□ネットワークの作成（4）

- 人をつなぐ
  - 各エージェントのlinkに繋がりのある人をランダムに追加する。
  - Universe変数link_rateを追加。
  - コントロールパネルを作成。
    - 名前：「リンク確率」 種類：スライドバー　対象：link_rate
    - 値の型：実数　範囲：0～1　初期値：0.1、目盛り間隔は0.01



□ネットワークの作成（5）

personをランダムにつなぐ

```
def univ_init(self):

    # マップの中心の座標の取得
    cx = get_width_space(Universe.city) / 2
    cy = get_height_space(Universe.city) / 2
    # 演習の半径の定義
    r = 0.4 * get_width_space(Universe.city)

    # Universe.numの数だけ、personを生成し、円周上に均等配置する
    for i in range(Universe.num):
        p = create_agt(Universe.city.person)
        theta = radians(360 * i / Universe.num)
        p.x = cx + r * cos(theta)
        p.y = cy + r * sin(theta)


    # 人の繋がりを定義
    people = make_agtset()
    for p1 in people:
        for p2 in people:
            if rand() < Universe.link_rate:
                p1.link.add(p2)
                p2.link.add(p1)

```

peopleの組み合わせで
forループをまわす

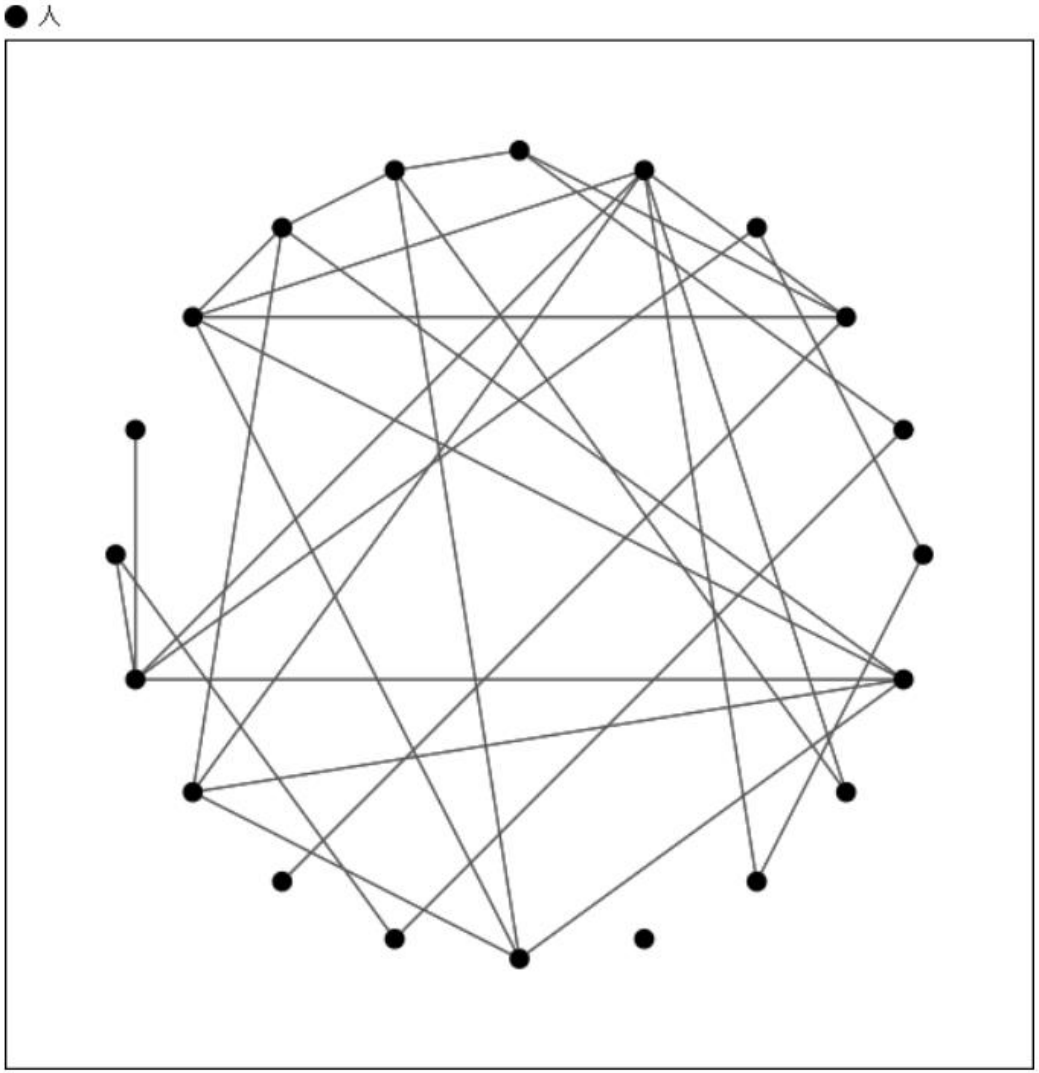
コントロールパネルで設定する
確率で、二人をつなぐ

□ネットワークの作成（6）

- マップ上にネットワークを表示
  - マップの出力設定を開く。
  - マップ要素リストにある「人」の設定画面を開く。
    - ➡「エージェント間に線を引く」の欄。
      - ☑「対象の変数」に「link」を選択。
      - ☑「OK」を押します。
  - モデルを保存後、実行してエージェント間にネットワークが作成されていることを確認。

□ネットワークの作成（7）

マップ上にネットワークを表示



❑「人のネットワーク 1.1完了時点」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/C9hNCGHzQai0xDLphC9BMw
  - モデル名「人のネットワーク 1.1完了時点」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

❑情報が伝わるルールを定義する（1）

- 最初に、3人が情報を知っているというルールをuniv_initに追加する。
- マップ上では、情報を知っている人を赤、知らない人を青で表示する。

```
def univ_init(self):

    # マップの中心の座標の取得
    cx = get_width_space(Universe.city) / 2
    cy = get_height_space(Universe.city) / 2
    # 演習の半径の定義
    r = 0.4 * get_width_space(Universe.city)

    # Universe.numの数だけ、personを生成し、円周上に均等配置する
    for i in range(Universe.num):
        p = create_agt(Universe.city.person)
        theta = radians(360 * i / Universe.num)
        p.x = cx + r * cos(theta)
        p.y = cy + r * sin(theta)

        if i < 3:
            p.color = COLOR_RED  # 最初の3人は、情報を知っている
        else:
            p.color = COLOR_BLUE  # それ以外の人は、情報を知らない

    # 人の繋がりを定義
    people = make_agtset()
    for p1 in people:
        for p2 in people:
            if rand() < Universe.link_rate:
                p1.link.add(p2)
                p2.link.add(p1)

```

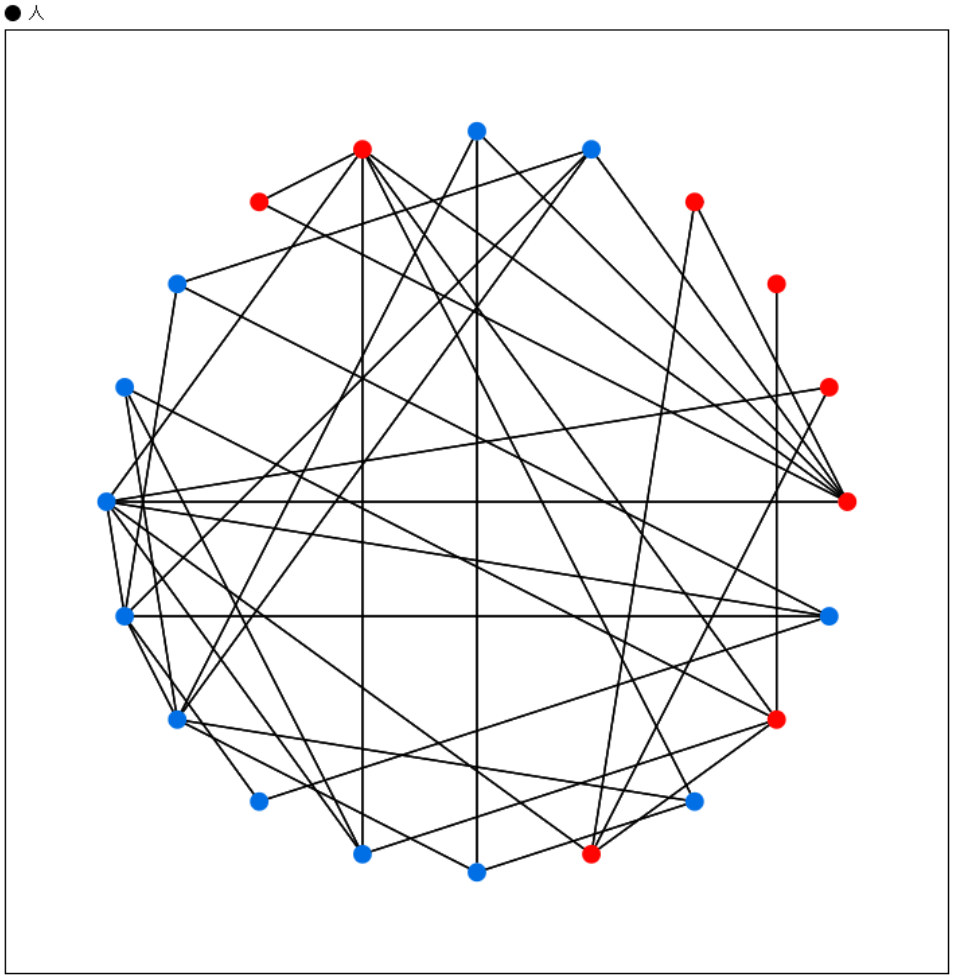
❑情報が伝わるルールを定義する（2）

- エージェントは毎ステップ任意の確率（ここでは0.1）で自分と繋がっている人に情報を伝える。

```
1  def agt_init(self):
2      self.link = set()
3
4  def agt_step(self):
5
6      if self.color == COLOR_RED:      # 自分が情報を知っていれば
7          for p in self.link:
8              if rand() < 0.1:     # 毎ステップ10%の確率でつながっている人に伝える
9                  p.color = COLOR_RED
10
```

❑情報が伝わるルールを定義する（3）

- 実行してみる



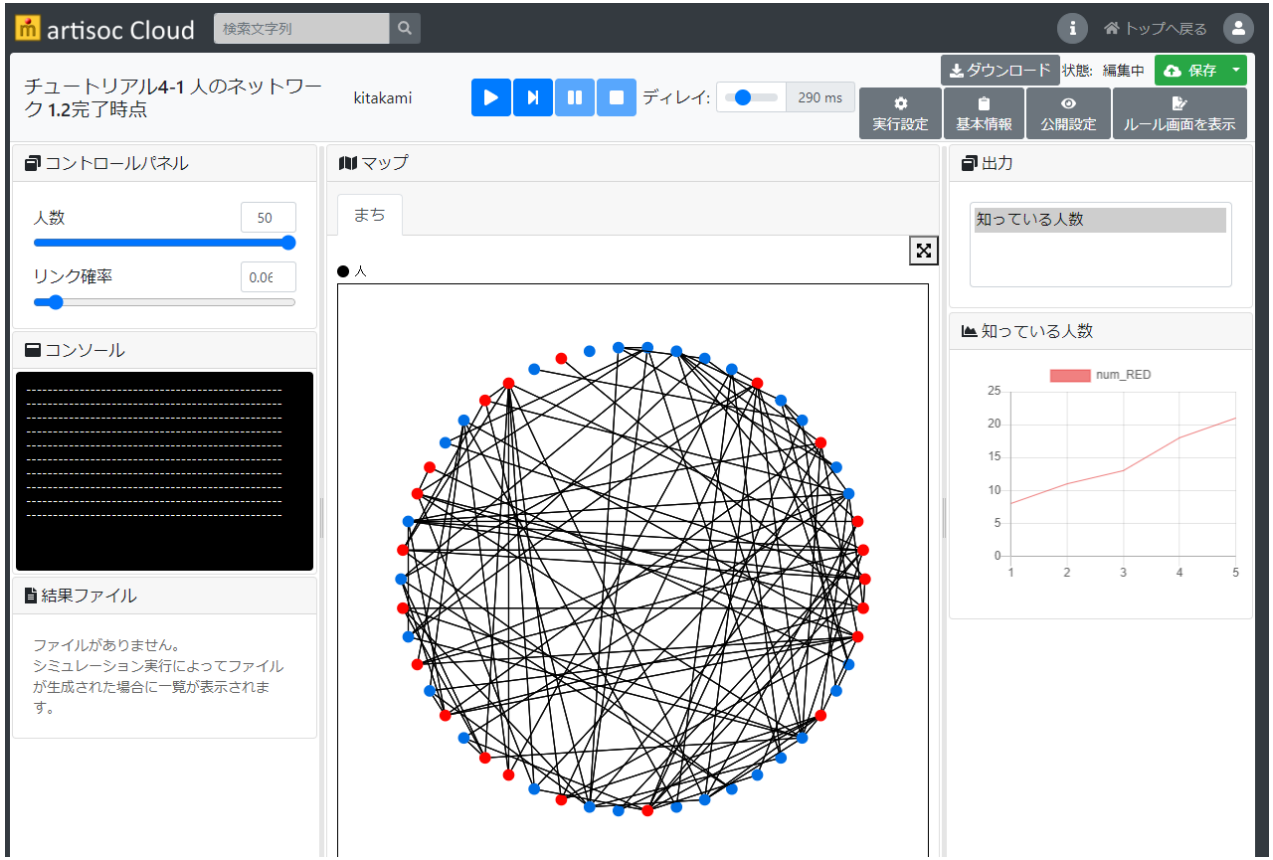
❑情報の伝わり方を可視化する（1）

- 情報を知っている人の数をグラフに出力する。
  - モデルの情報を可視化することで、新たな発見や間違いに気づくことができる。
  - モデルツリーに、新たなUniverse変数num_REDをつくる。
    - この変数に、情報を知っている人の数（つまり赤の数）を記録。
  - 毎ステップ、変数の初期化と赤の数え上げを行う。

```
30 def univ_step_begin(self):
31
32     # ステップの最初にカウントを初期化
33     Universe.num_RED = 0
34
35 def univ_step_end(self):
36
37     # ステップの最後に数をカウント
38     people = make_agtset()
39     for p1 in people:
40         if p1.color == COLOR_RED:
41             Universe.num_RED += 1
42
```

❑情報の伝わり方を可視化する（2）

- 情報を知っている人の数をグラフに出力する。
  - 出力画面から出力設定。
  - 時系列グラフを追加し、時系列グラフ要素リストにUniverse.num_REDを登録する。



□「人のネットワーク 1.2完了時点」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/4oLY4O-fQbuq3aYQ41OpcQ
  - モデル名「人のネットワーク 1.2完了時点」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

- リンク確率と伝わる速度の関係を調べる。
  - リンク確率Universe.link_rateを変化させたときの、情報伝搬の速度を調べてみましょう。全員に伝わるまでのステップ数は、リンク確率とどのような関係にあるでしょうか。

- 情報伝播の確率と伝わる速度の関係を調べる。
  - 本節のモデルでは、自分と繋がっている人に情報を伝える確率を0.1としました。この確率をUniverse変数とし、スライドバーで操作できるようにしましょう。そして、この確率と情報伝播の速度の関係を調べてみましょう。

- 同期のルールを入れる。
  - agt_stepで情報を伝えるルールについて、第2部19章で学んだ同期問題を考えてみましょう。
  - 本節のモデルの「自分が情報を知っているとき、繋がっている人に情報を伝える」というルールは、第2部19章の森林火災モデルでの「自分が燃えているとき、隣り合う木を燃やす」というルールと対応します。したがって、第2部19章で扱った同期問題がここでも発生します。つまり、エージェントのルールが実行される順番によって、情報の伝わり方が変わってしまうのです。
  - これを防ぐため、エージェントの行動を同期させるルールを導入してみましょう。
  - また、エージェントの行動を同期させるべきなのはどのような場合か、考えてみましょう。