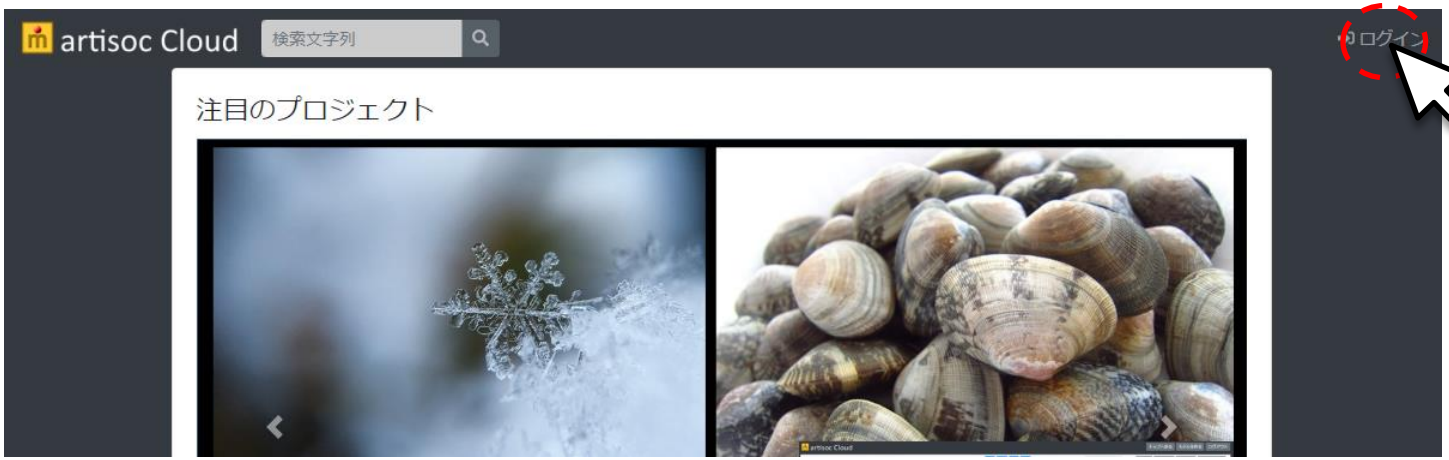
16:35より開始します

- 事前に送付したスライド資料を準備しておいてください
  - 現在映しているこの資料と同じですが、印刷して手元で参照できると便利です
  - 印刷できない方は別ウィンドウで開いておいてください（チャットでURLを送ります）
- ブラウザでartisoc Cloudにアクセスし、ログインを済ませておいてください
  - https://artisoccloud.kke.co.jp/
- p.12の「新規モデルの作成」をやっておいてください（時間があれば）



※zoomが全画面表示になっていてブラウザを開けないときは、Escキーか「全画面表示の終了」で解除できます

artisoc Cloud初級チュートリアル

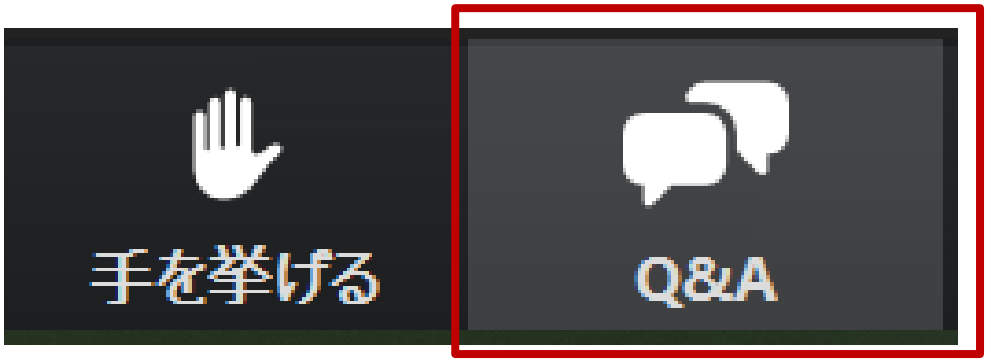
❒受講にあたっての注意

■ 進め方

- artisoc Cloudの操作方法を初心者向けに基礎から解説していきます
  - 操作方法の大筋はartisocと同じです
  - artisoc経験者や、プログラミング経験者にとっては、少し簡単な内容かもしれません
- 操作方法の解説と実際に操作していただく実習を繰り返し行います
- 操作が終わったら「手を挙げる」ボタンで教えてください
  - 指示するまで手を挙げっぱなしにしておいてください
  - なるべく全員が操作を終えるのを待って進めますが、時間の都合上待てない場合もありますのでご了承ください
- 受講中はzoom画面とartisoc Cloudの操作画面（ブラウザ）を行き来することになります。このスライド資料を印刷して手元に置いておくか、別ディスプレイに表示しておくと、操作しながら資料を参照できて便利です

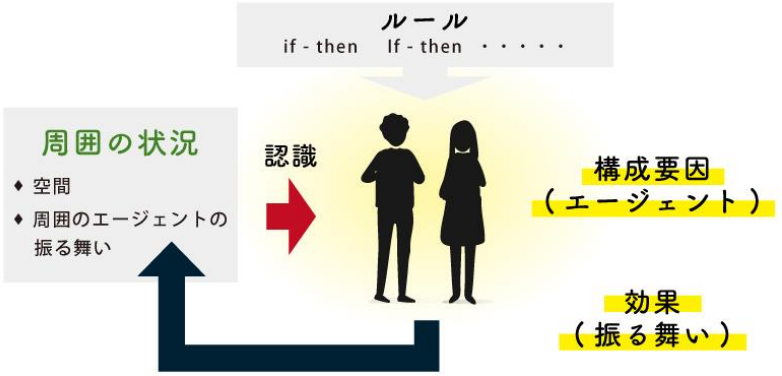
❒受講にあたっての注意

- 質問の仕方
  - 質問があれば、「Q&A」ボタンから投稿してください
    - 基本的には待機スタッフが回答し、他の参加者には共有されません
    - 全員に共有が必要と判断すれば、講師が口頭で回答します
    - 匿名で質問することも可能です
  - 全ての質問には回答することはお約束できませんのでご了承ください
  - 込み入った質問は、後日に質問フォーム等でお願いします
- 画面・音声などのトラブル時には
  - 質問とおなじように「Q&A」ボタンからお問合せください
  - 接続が完全に切れて復旧できないときは、下記番号にお電話ください
    - 03-5342-1273



❑マルチエージェント・シミュレーション(MAS)とは

個々の「エージェント」の振る舞いを定義



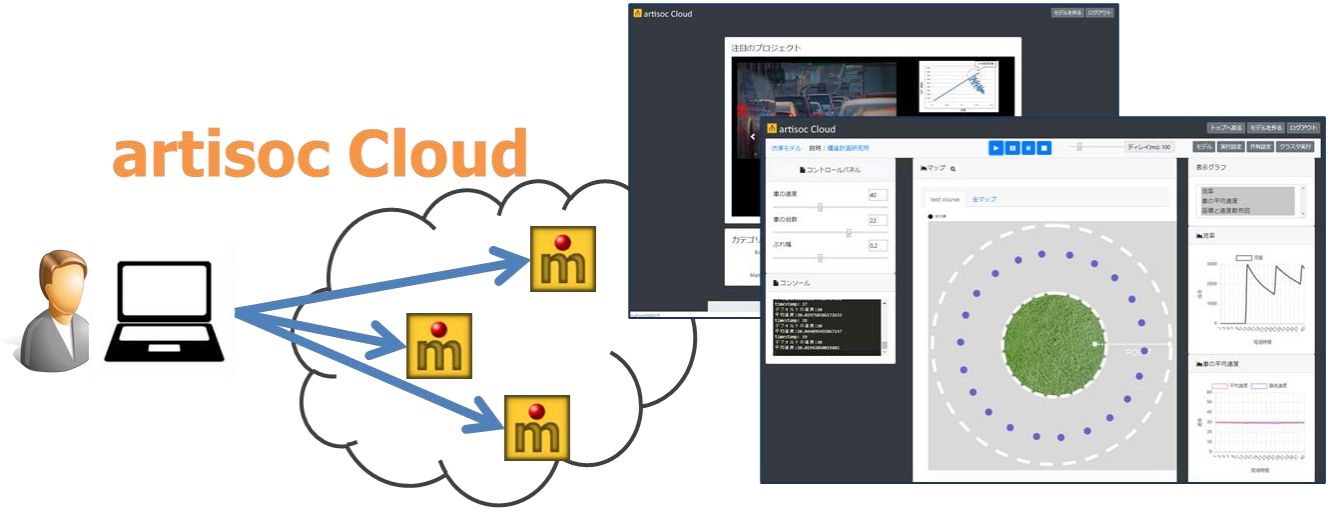
エージェント同士の相互作用による
複雑な社会現象を再現

コンピュータの中に人工社会を構築し、社会現象のメカニズムを理解する

❒artisoc Cloudとは

ユーザフレンドリな
マルチエージェント・シミュレータ

- artisocの次世代版
- 簡単にモデルの作成と可視化が可能
- モデルをWebブラウザ上で作成し共有することが可能
- クラウドのパワーを使った高度な解析が可能



2020年5月より、アカデミックユーザ向けに
機能を限定した試用版を公開

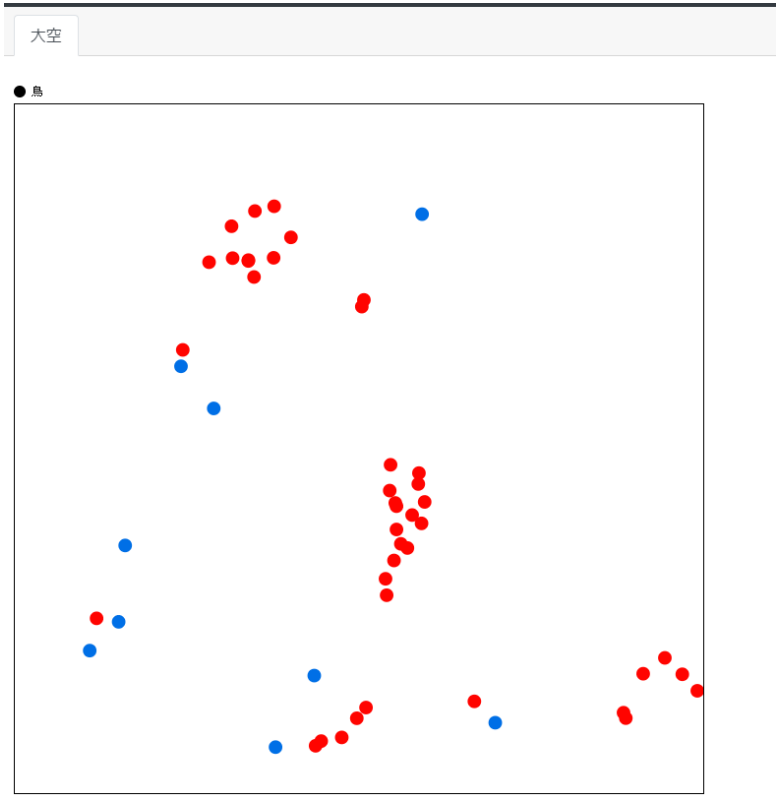
❒この講習について

■ 概要

- ❒『飛ぶ鳥モデル』の作成を通じ、artisoc Cloudの基本的テクニックを習得
  - ➡ 飛ぶ鳥モデル:「大空(oozora)」という空間上に「鳥(tori)」というエージェントが存在し、様々な向きに飛んでいく

■ 講習の流れ

1. モデルの作成手順を学ぶ
2. 基本的なテクニックを学ぶ
3. 相互作用を含むモデルを作成する



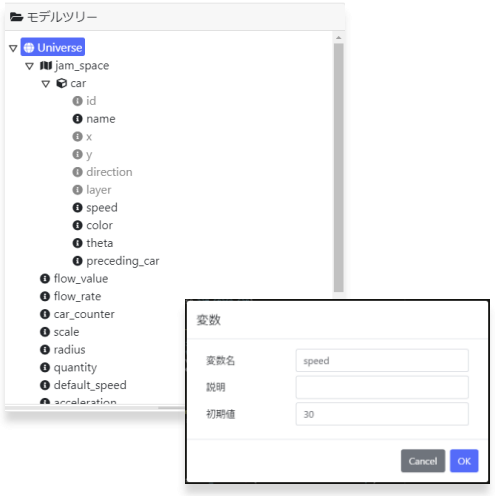
I. モデルの作成手順を学ぶ

❒まずは動かしてみましょう

- 以下のような簡単なモデルを作成し、artisocのモデル作成手順を学びます
  - 「大空(oozora)」という空間上に「鳥(tori)」というエージェントが1体だけ存在
  - 「鳥」は空間の中央から出発して前方にまっすぐ飛んでいく

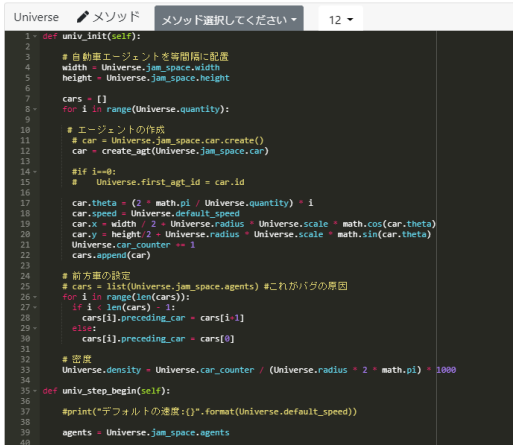
❑モデル構築の流れ

① 空間／エージェントの
種類・属性を作成

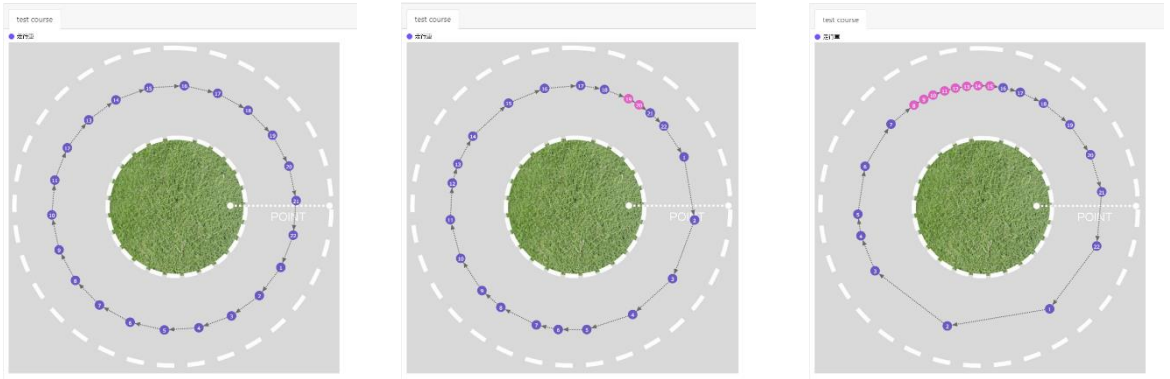


② シミュレーション結果の
出力形式を決定

③ エージェントの
行動ルールを作成

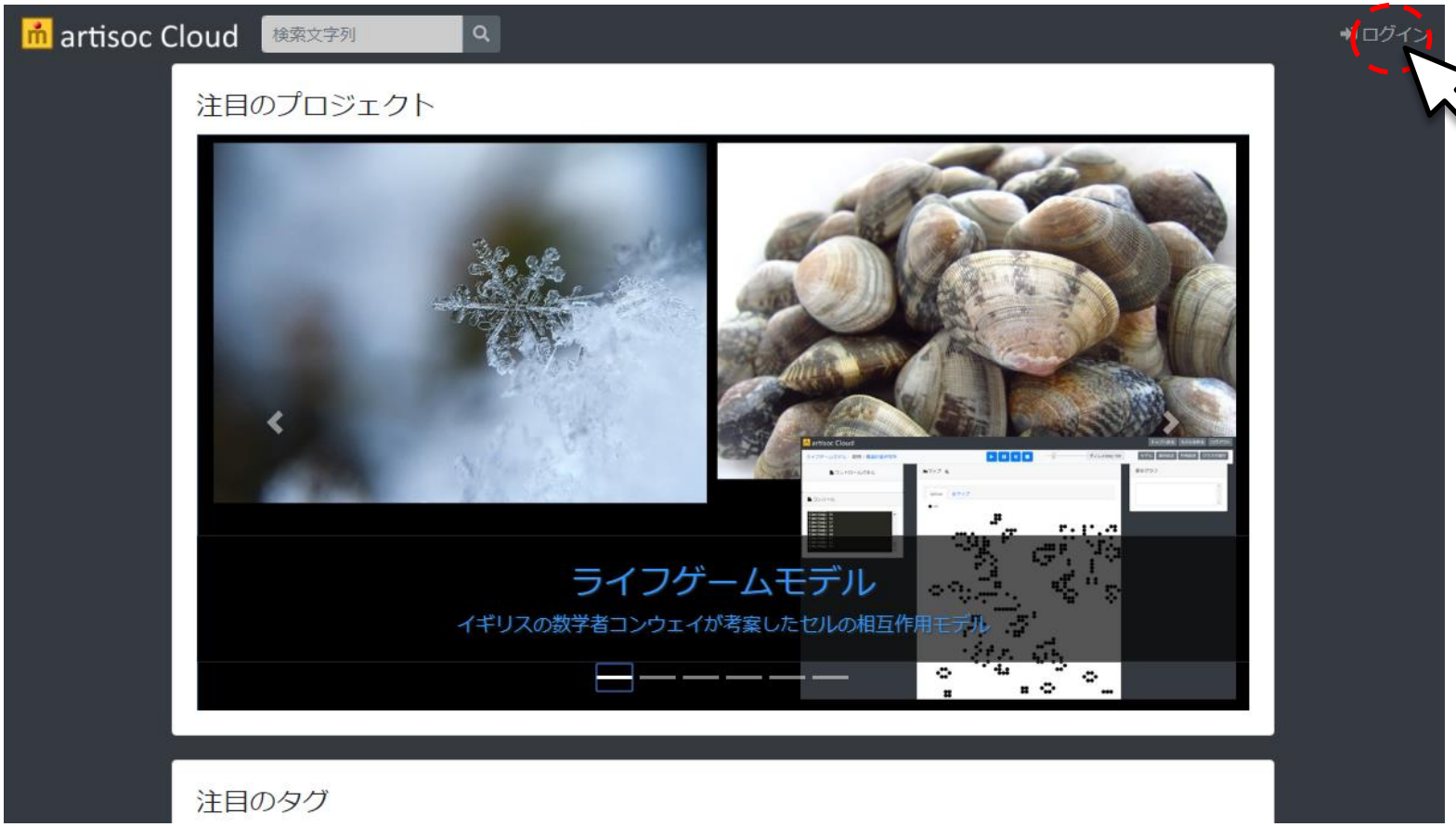


実行ボタンをクリック！



❒準備

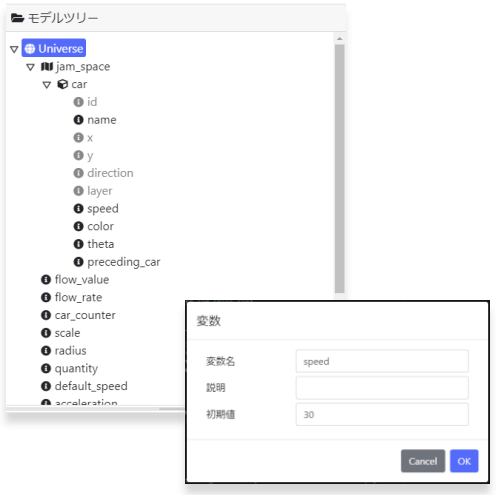
- 下記URLにアクセスし、画面右上からログインします
  - https://artisoccloud.kke.co.jp/



❒準備

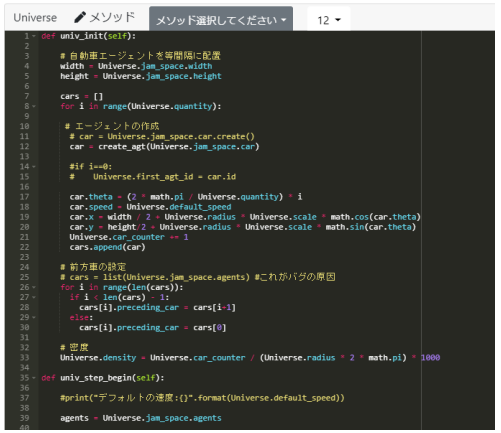
- 右上から「新規モデルの作成」をクリックし、モデル名を入力します
  - 「飛ぶ鳥モデル」とでも名付けましょう

① 空間／エージェントの
種類・属性を作成



② シミュレーション結果の
出力形式を決定

③ エージェントの
行動ルールを作成

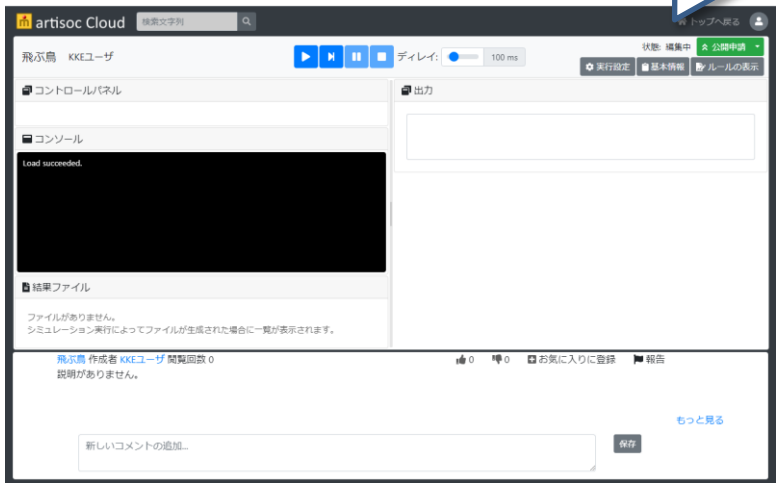


❒空間／エージェントの種類・属性を作成

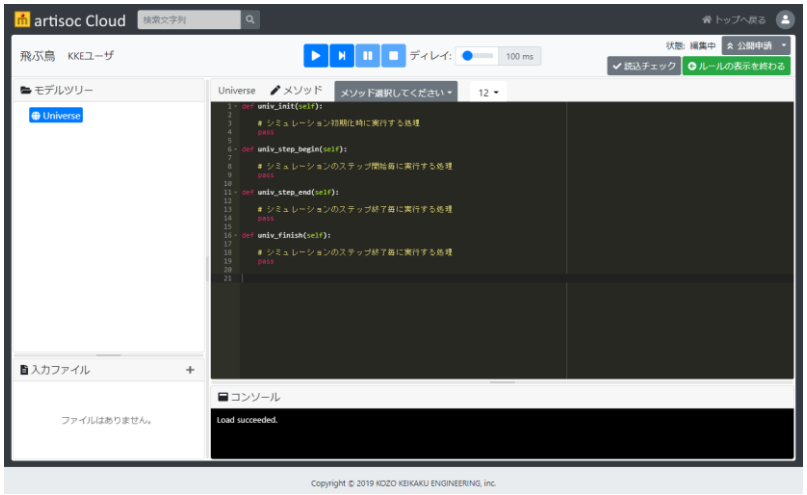
■「モデルツリー」を編集することでモデルの枠組みを構築します

- 最初はモデル全体を表す「Universe」だけがあります
- ここに「空間」「エージェント種別」「変数」などを追加していきます

■「ルール画面を表示」をクリックしてモデルのルール画面に遷移します



出力画面



ルール画面

❒空間の定義

- Universe上に空間を作成
  - 空間名「oozora」・・・・大空
  - その他はデフォルト値
    - ➡ 説明は自由に入力

❒エージェント種別の作成

■ 空間の中にエージェント種別を作成

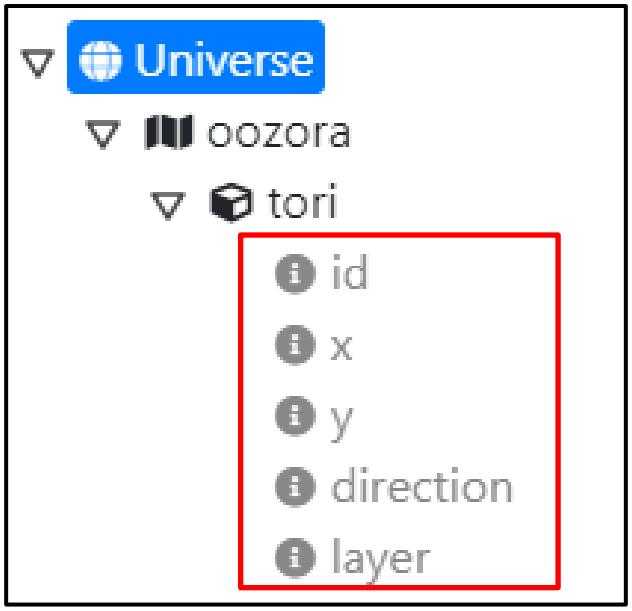
◘ エージェント種別名「tori」・・・鳥エージェント

➡「説明」は自由に、「記憶数」は0のままでOK

※これはエージェントそのものではありません（詳しくはあとで説明します）

❒エージェントの変数

- エージェントは以下の変数を持ちます
  - id … 識別番号
  - x … x座標
  - y … y座標
  - direction … 向き
  - layer … 空間レイヤー（今回は使いません）
- 変数はエージェントの性質を表します



❒TIPS：シミュレーションモデルの保存

シミュレーションモデルに不具合があって、artisocが停止してしまった・・

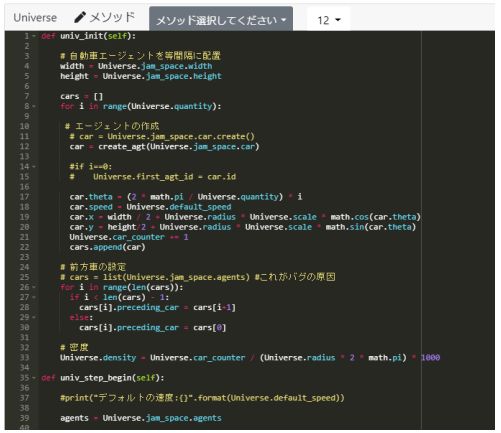
ルールを変更したら動かなくて、元に戻せなくなった・・

こまめに保存

① 空間／エージェントの
種類・属性を作成

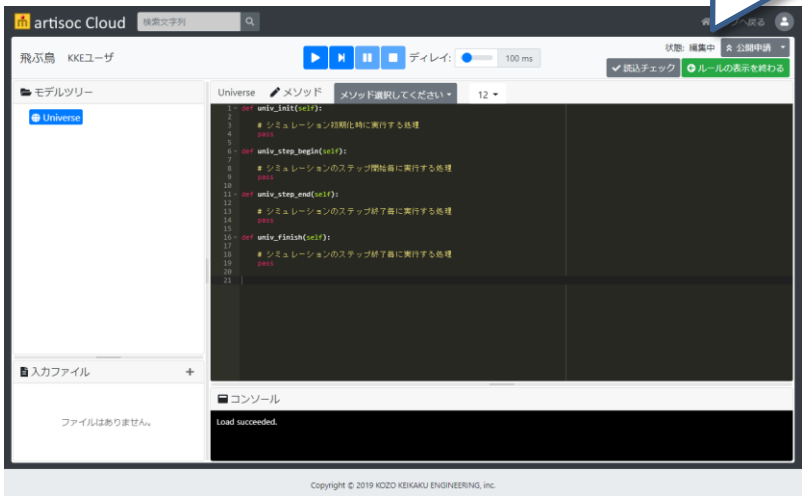
② シミュレーション結果の
出力形式を決定

③ エージェントの
行動ルールを作成

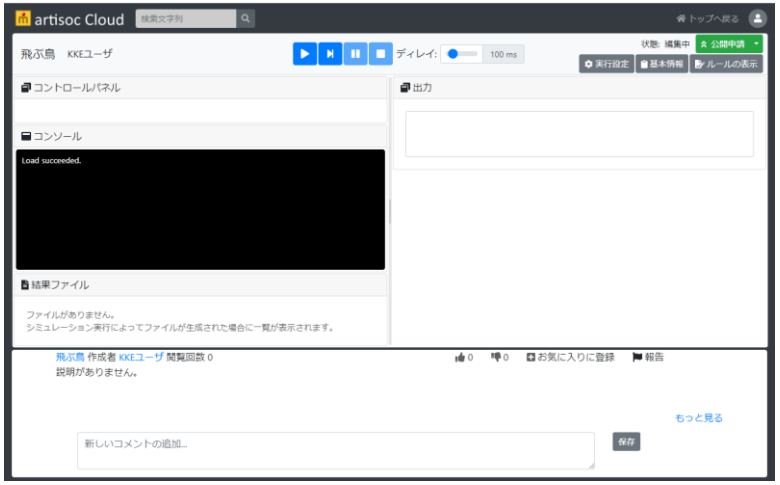


❒出力設定

■「出力画面を表示」をクリックして出力画面へ移動



ルール画面



出力画面

❒出力設定

■ 空間とエージェントを「マップ」として出力

❒ 下図のようにマップ出力を選択

❒出力設定

■ 空間とエージェントを「マップ」として出力

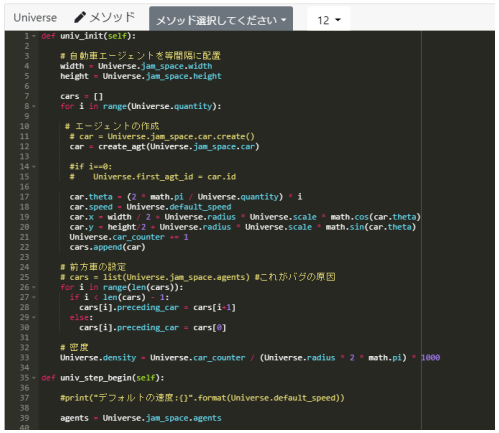
マップ名：大空
空間：oozora
→空間oozoraを「大空」という名前で出力

要素名：鳥
エージェント：tori
→エージェントtoriを「鳥」という名前で出力
※エージェントを表示する形状、色なども設定できるが、デフォルトでOK

① 空間／エージェントの
種類・属性を作成

② シミュレーション結果の
出力形式を決定

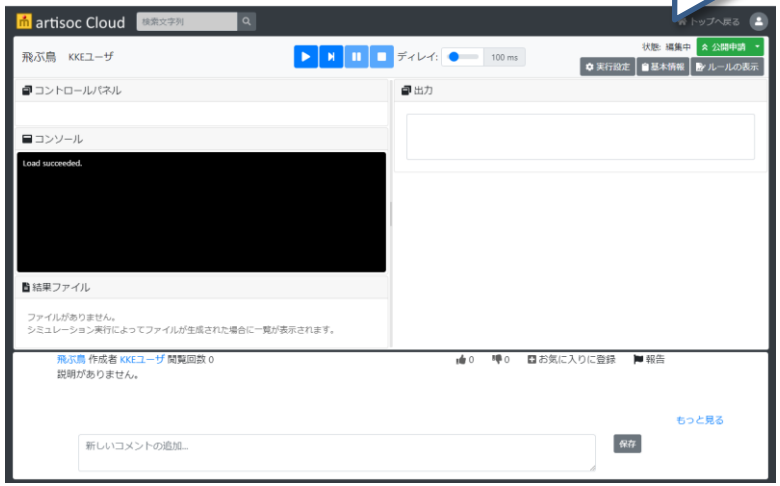
③ エージェントの
行動ルールを作成



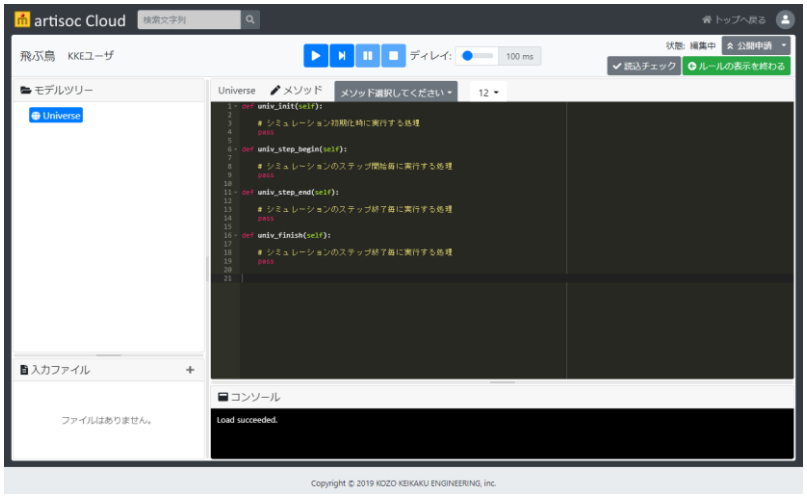
❒ルールを作成

- 以下のようなモデルのルールを作成します
  - シミュレーション開始時に鳥エージェントを1体だけ作成
  - 鳥エージェントの初期位置は空間の中央
  - 鳥エージェントは毎ステップ、前方に1進む

■「ルール画面を表示」をクリックしてモデルのルール画面に遷移します



出力画面



ルール画面

❒エージェントの生成

- シミュレーション開始時に鳥エージェントを1体生成するルールを作成します
  - モデル全体に関わるルール→Universeのルールエディタに記述します
  - モデルツリーのUniverseをクリックするとUniverseのルールエディタが開きます
  - Universeのルールは以下の部分からなります
    - univ_init … シミュレーション開始時に一度だけ実行するルール
    - univ_step_begin … シミュレーションの各ステップ開始時に実行するルール
    - univ_step_end … シミュレーションの各ステップ終了時に実行するルール
    - univ_finish … シミュレーション終了時に一度だけ実行するルール

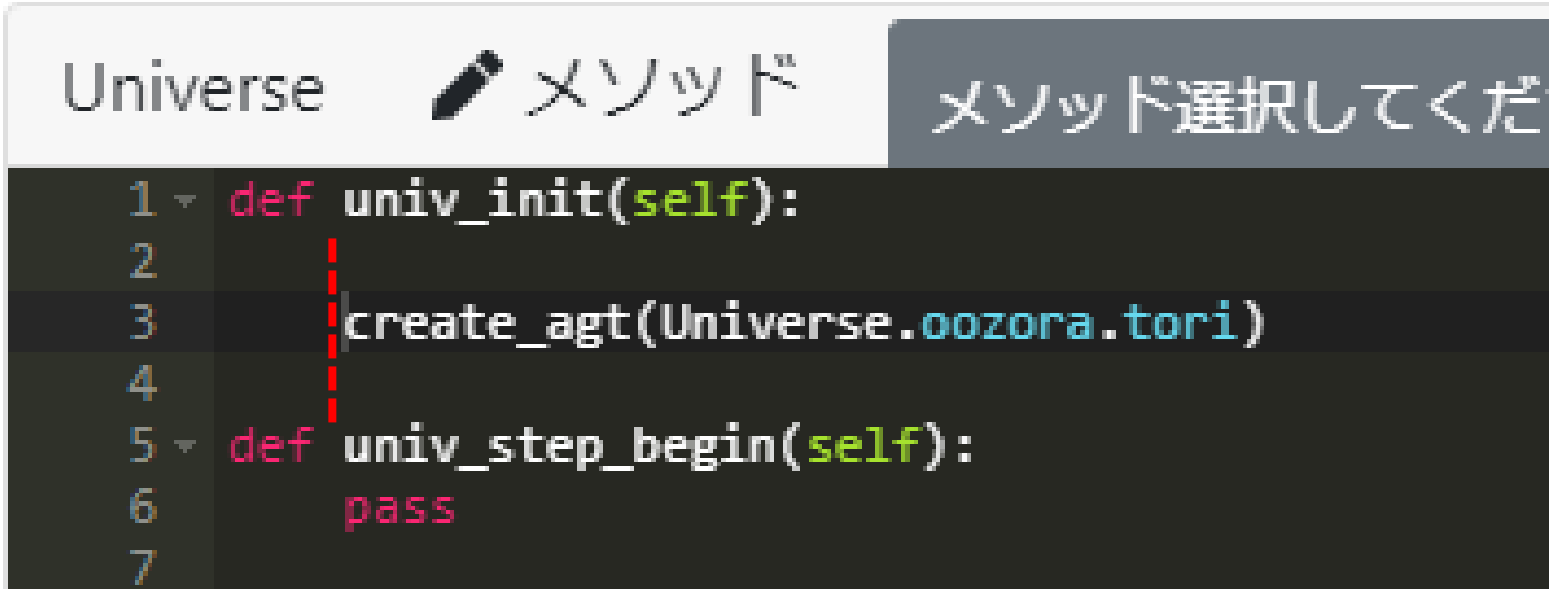
❒エージェントの生成

- シミュレーション開始時に鳥エージェントを1体生成
  - univ_initに鳥エージェントを生成するルールを記述します
  - エージェントの生成→create_agt：エージェントを生成する**関数**
    - 関数：ルールをひとまとめにしたもの
    - 「create_agt(Universe.oozora.tori)」で"toriエージェントを生成します
  - 「pass」を消し、下図のように記述します
    - インデントに注意→次ページ

❒ルール作成時の注意：インデント

- ルールは1段下げた部分に記述することに注意
  - これをインデントといいます
  - インデントをするにはtabキーを使うと便利です
    - 「shift + tab」でインデントが戻ります
  - エンターキーで改行してもインデントは維持されます

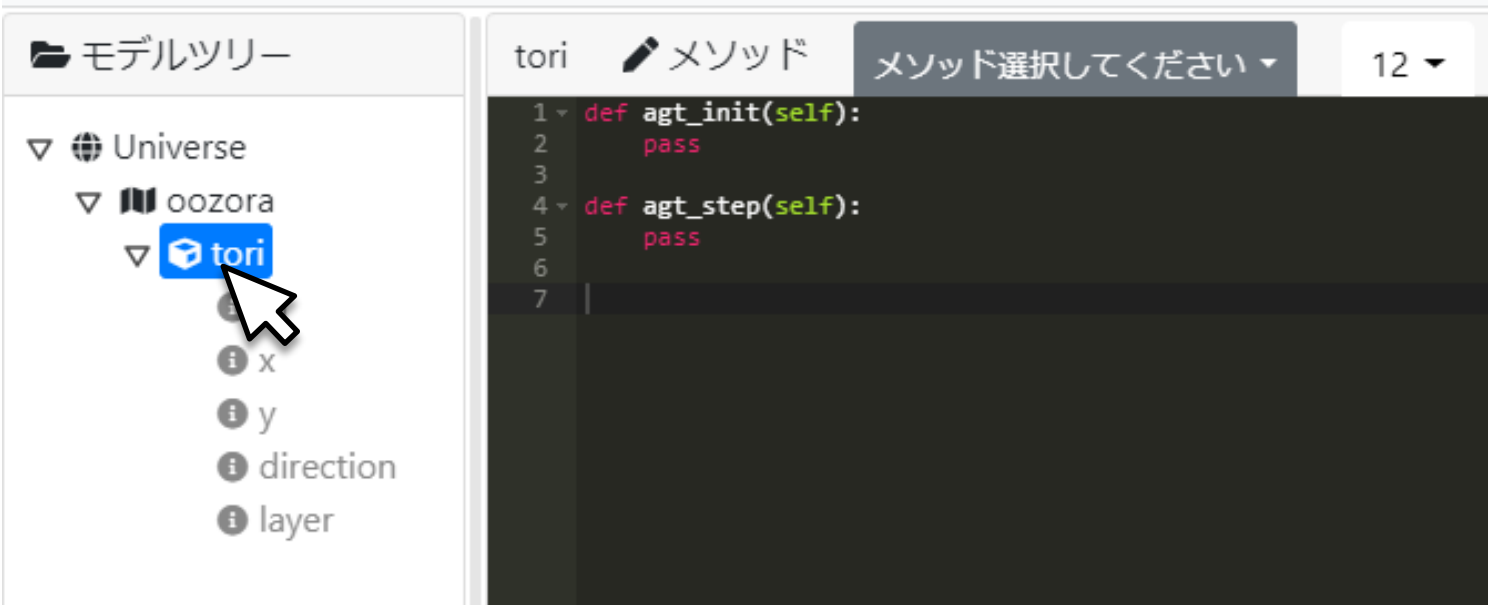
※空白行は自由に挿入してかまいません



❒エージェントのルールを作成

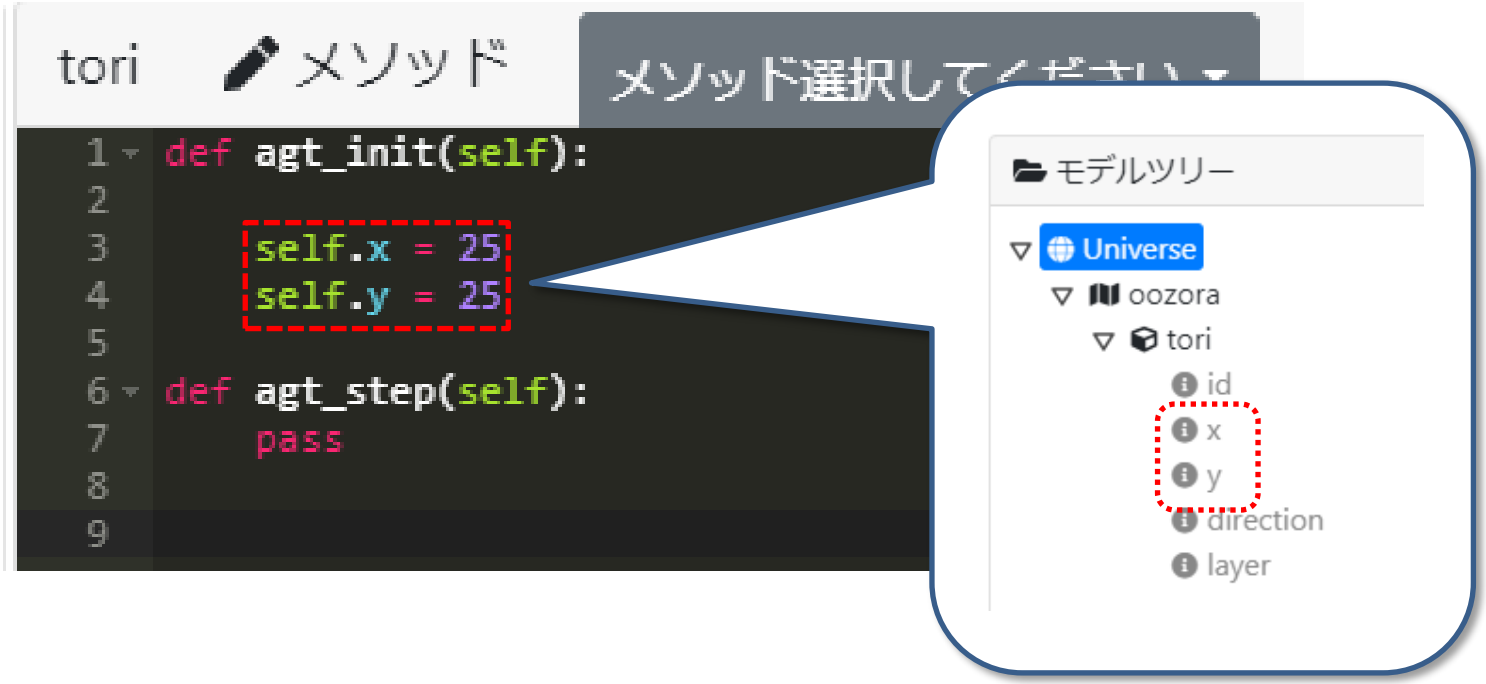
■ エージェントの行動ルールを作成します

- □ エージェントの行動ルール→エージェントのルールエディタに記述します
- □ ツリー上のエージェント種別名をクリックするとエージェントのルールエディタが開きます
- □ エージェントのルールは以下の部分からなります
  - ➡ agt_init … エージェントが生成されたとき一度だけ実行されるルール
  - ➡ agt_step … エージェントが毎ステップ実行するルール



❒エージェントのルールを作成

- 鳥エージェントを初期位置として空間の中央に配置
  - agt_initにルールを書きます
  - 自分自身のx座標とy座標に25を代入します
    - 自分自身の変数を呼び出すときには「self.[変数名]」と記述します
    - 「変数名 = 数値」と記述することで、「変数に数値を代入する」という意味になります
      - ☑「=」前後の半角スペースはなくてもよいが、あったほうが見やすい
  - 「pass」を消し、下図のように記述してください



❒エージェントのルールを作成

- 鳥エージェントは毎ステップ前に進む
  - agt_stepにルールを書きます
  - forward：前に進む関数
    - 「self.forward([数値])」と書くことで数値だけ前に進みます
      - ☑「self」は自分自身なので、自分自身に「進め」と命令するイメージ

tori　メソッド　メソッド選択してください　12

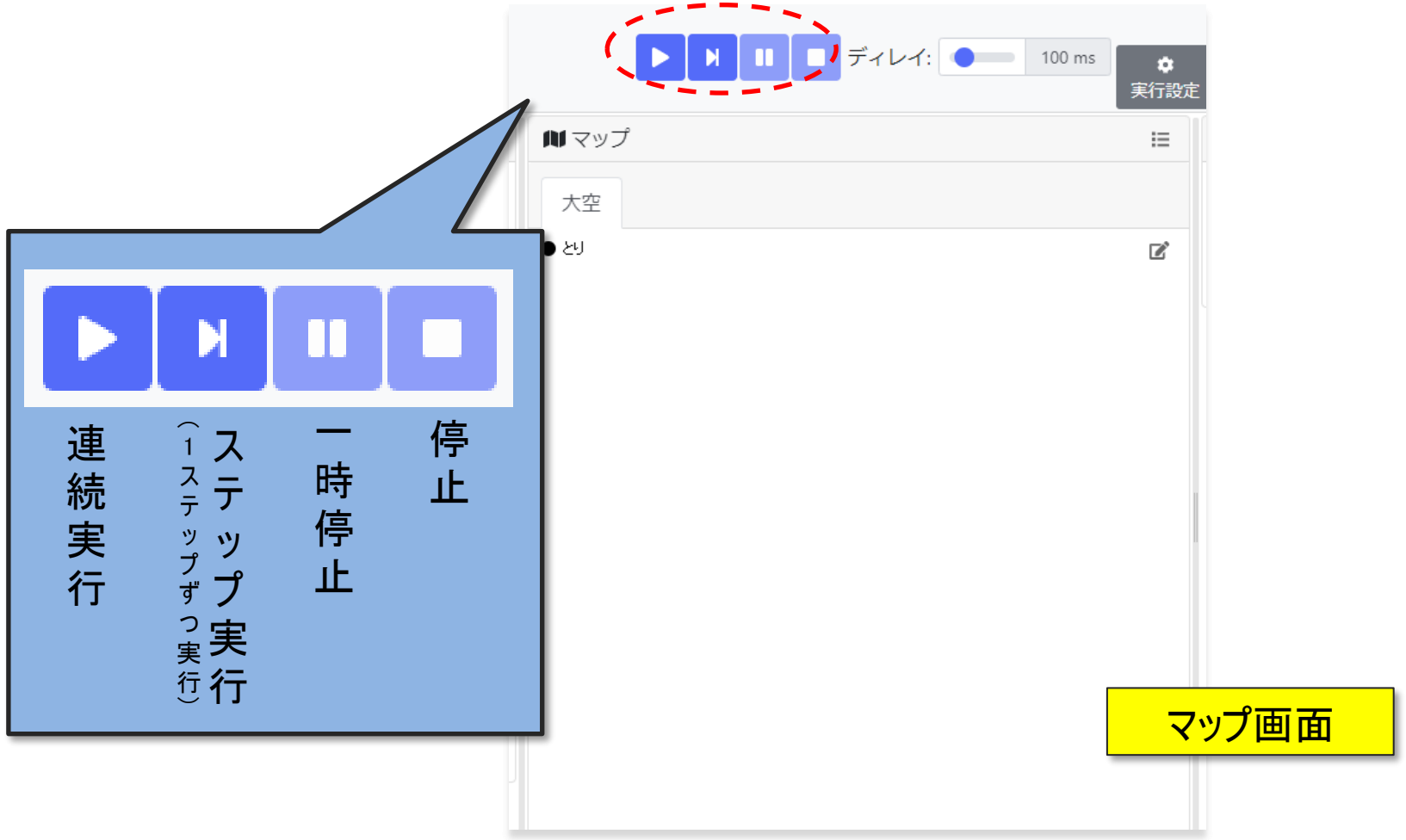
```
1  def agt_init(self):
2
3      self.x = 25
4      self.y = 25
5
6  def agt_step(self):
7
8      self.forward(1)
9
10
```

① 空間／エージェントの**種類・属性**を作成

② シミュレーション結果の**出力形式**を決定

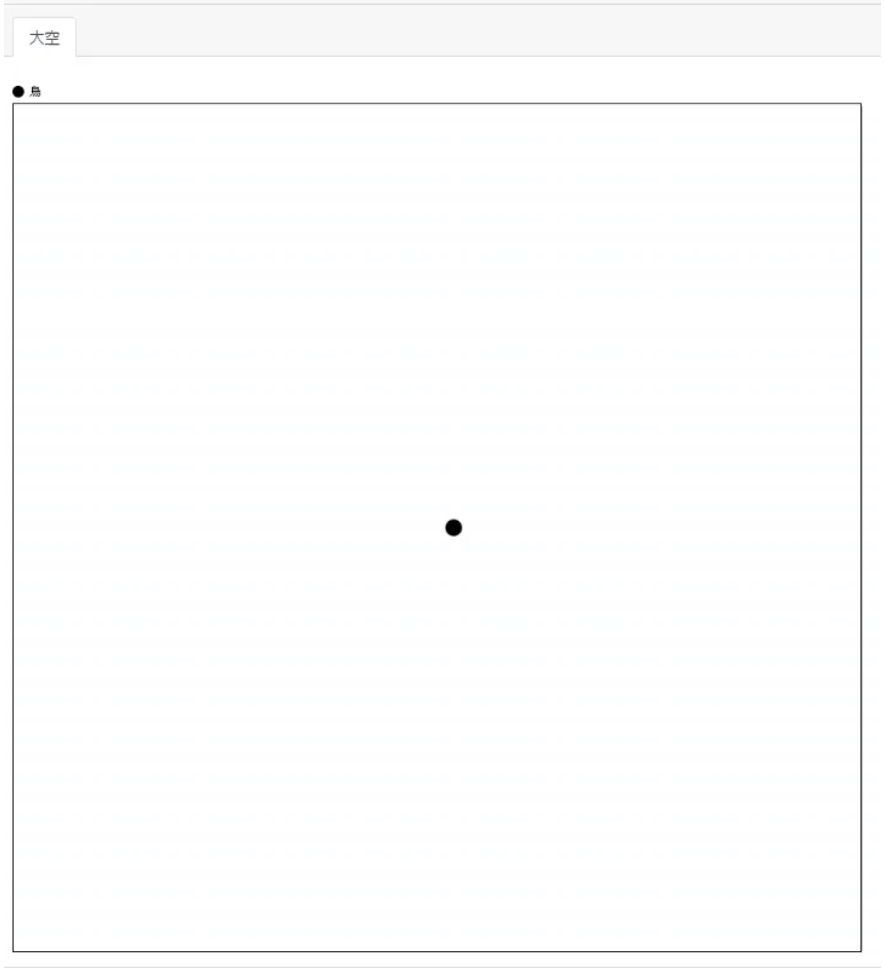
③ エージェントの**行動ルール**を作成

❒シミュレーションを実行



ルールの編集、出力の設定を行うときは「停止」状態である必要があります

- マップ上を鳥が左から右に動いていれば成功です
  - 10000ステップで自動終了するようになっています

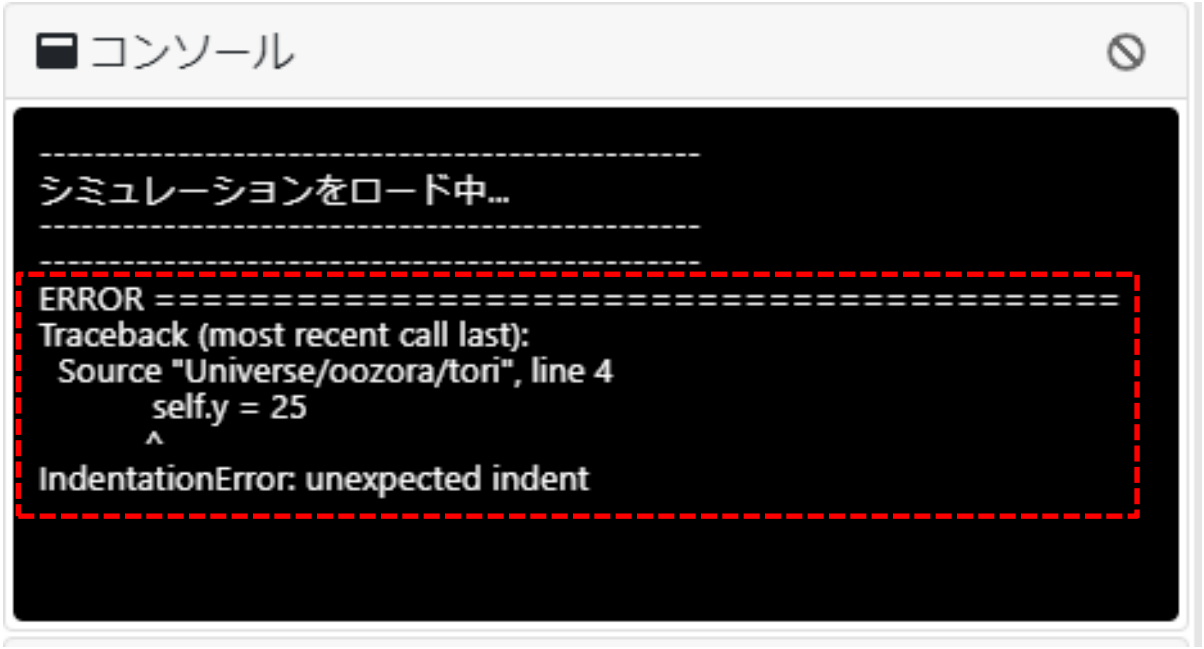


シミュレーション速度を変更したい・・・

スライドバーで実行速度を調整する

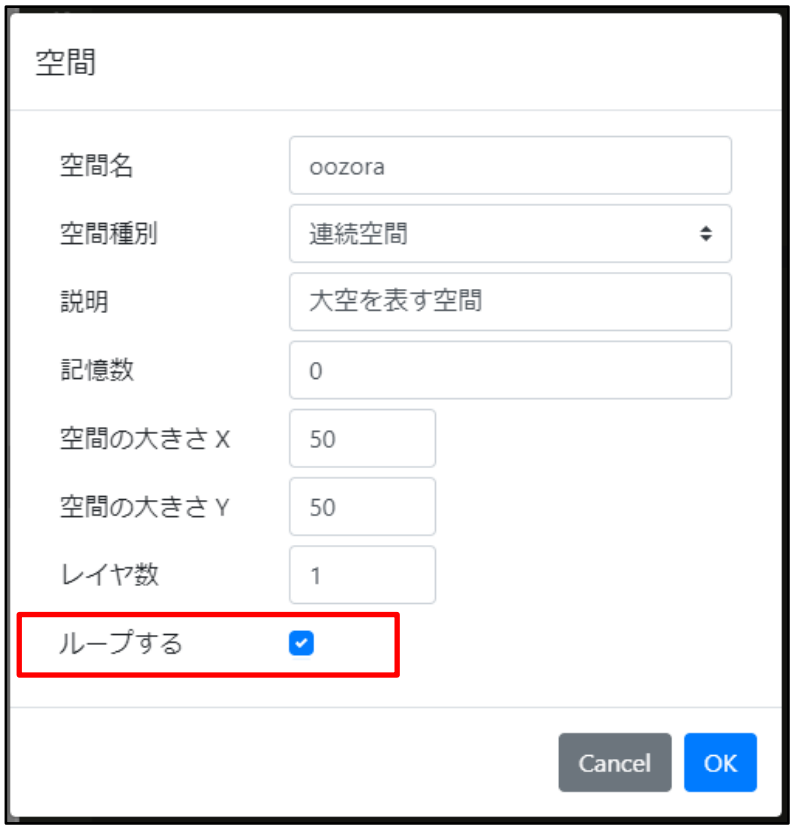
右に動かすと実行速度が遅くなり、左に動かすと速くなる

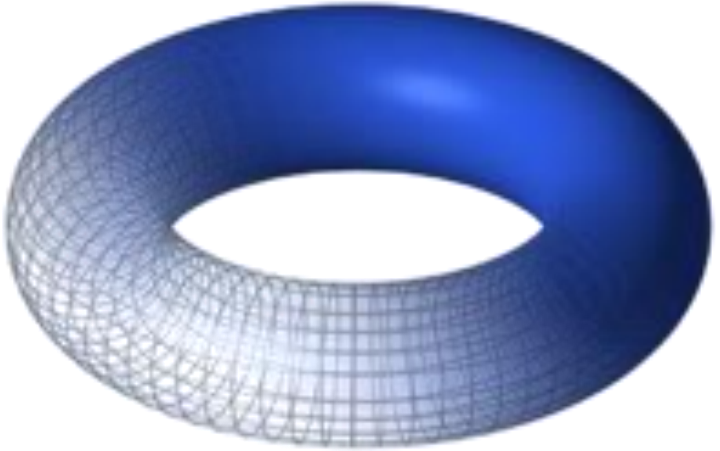
- うまく動かなければ、Q&Aで質問してください
  - コンソール画面にエラーメッセージが出ていたら、コピーしてお知らせください



❒空間のループ

■ 空間がループ =上端-下端・左端-右端が繋がった空間





ループした空間のイメージ

II. 基本的なテクニックを学ぶ

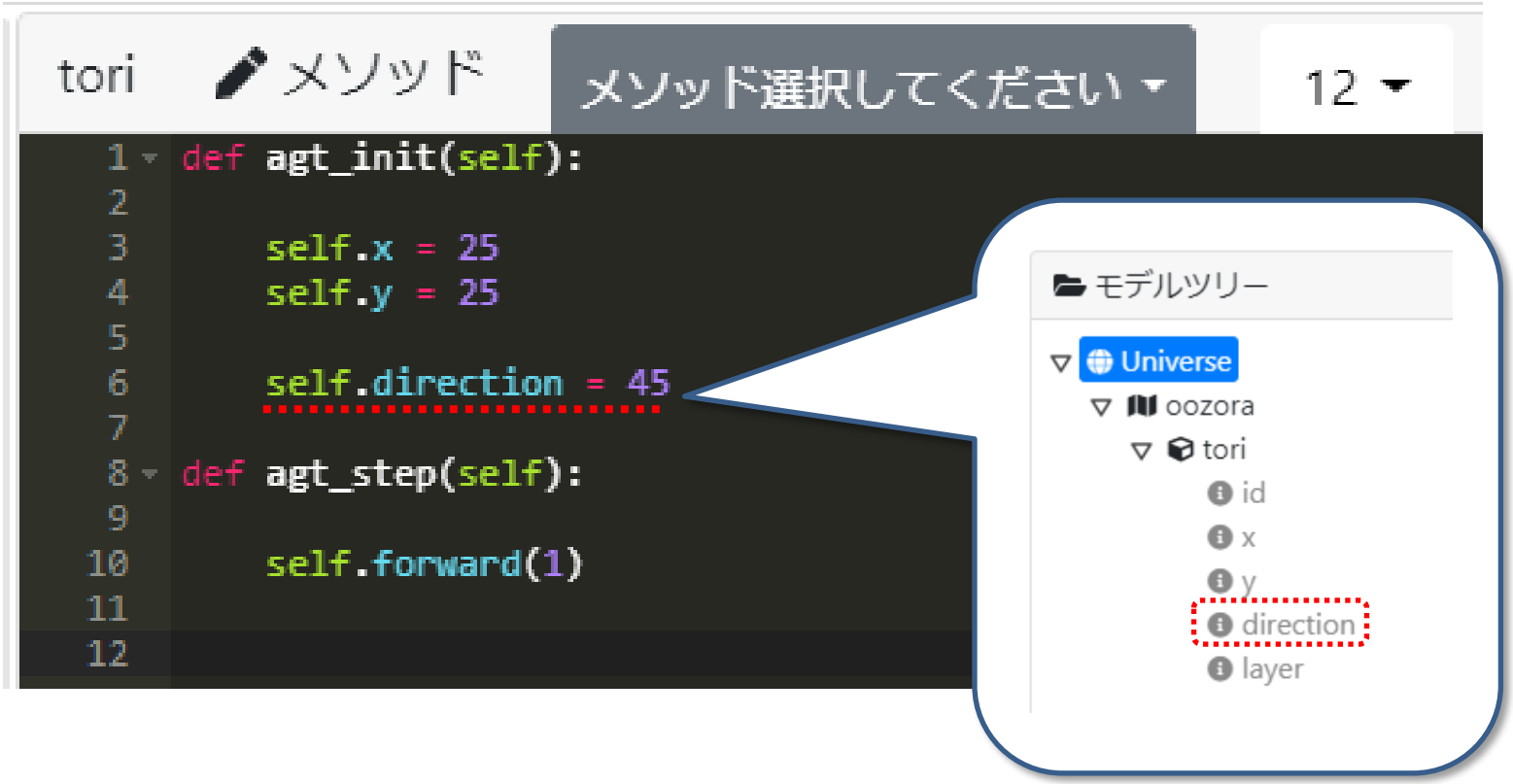
❒基本的なテクニックを覚える

- 飛ぶ鳥モデルを編集しながら、モデル構築の基本的なテクニックを覚えましょう

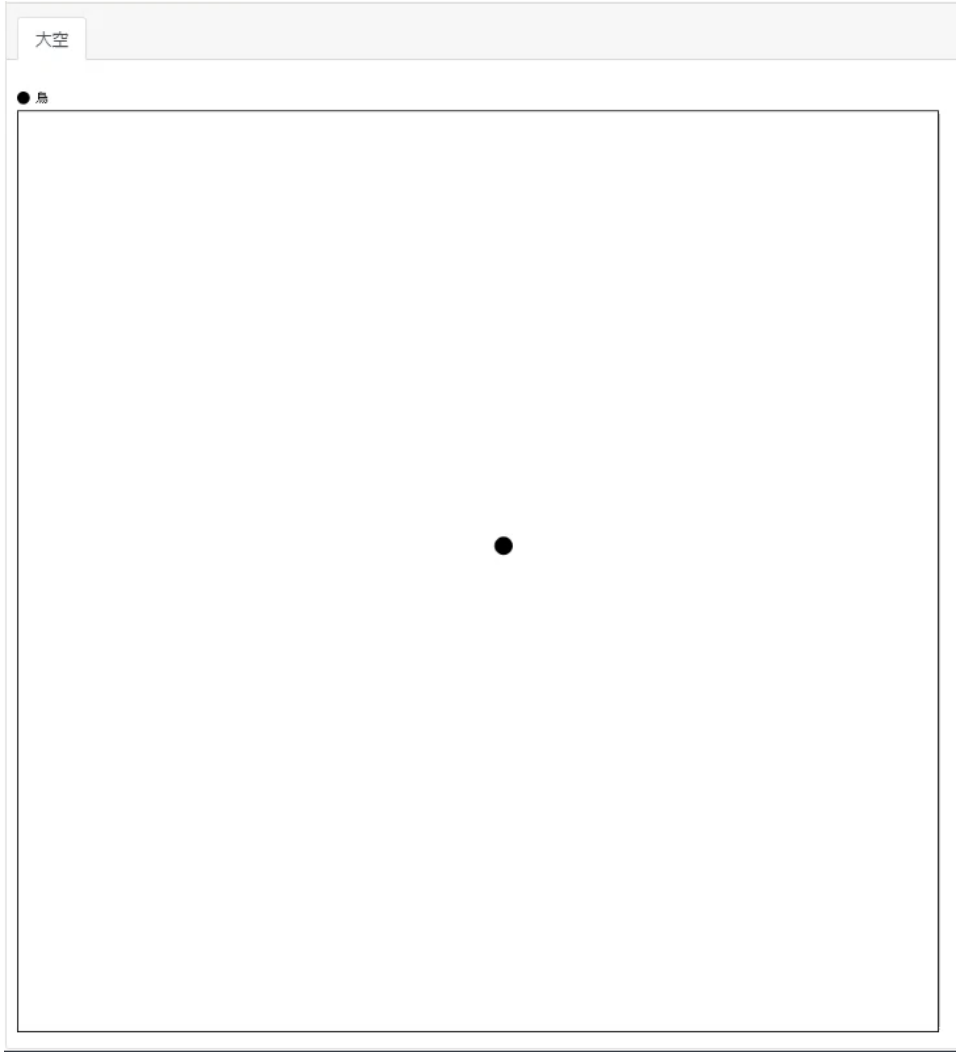
- 途中から来た人や途中でついていけなくなった人は、下記のモデルにアクセスしモデルを継承してください
  - https://artisoccloud.kke.co.jp/models/d4Di76IaTqSCeaMDrbU4TQ
    - チャットでURLを送ります
  - 継承：他のユーザが作ったモデルをコピーして編集する

❒進む方向を変える

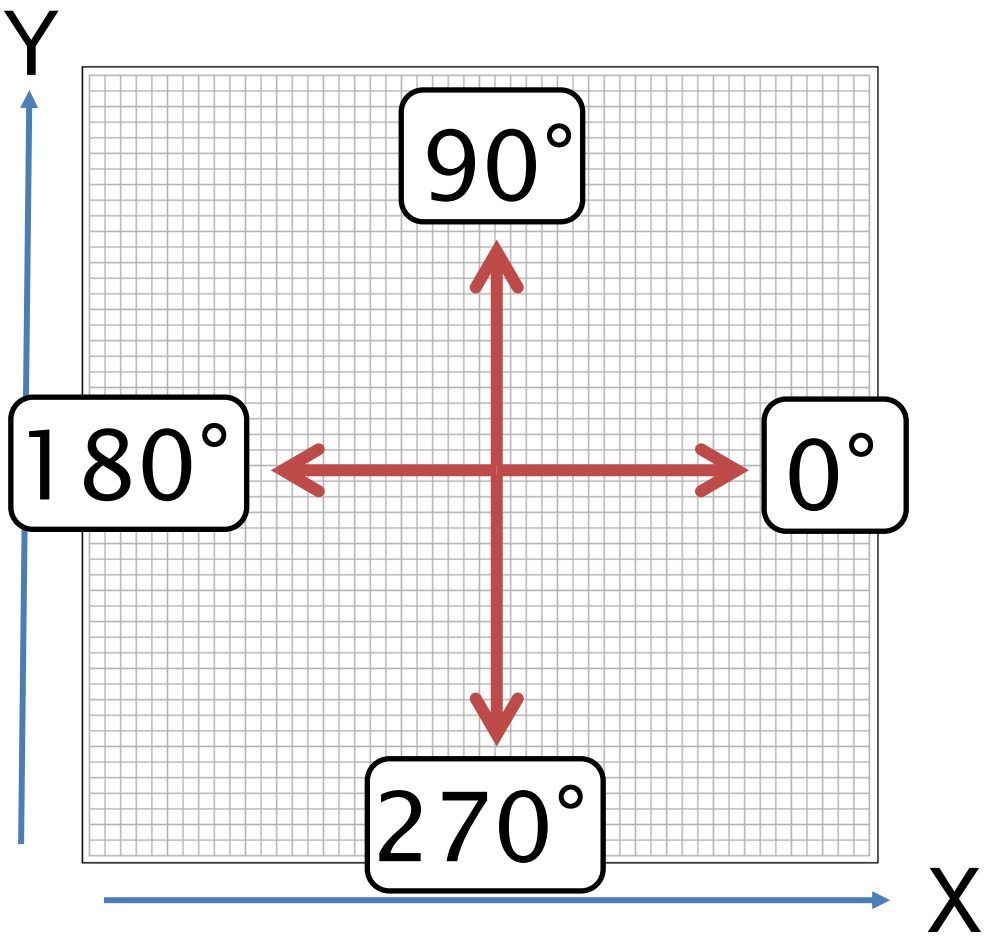
- ルール画面に戻り、ツリー上のtoriをクリックしてルールエディタを開きます
- agt_initで変数「direction」に45を代入して実行します



■ 鳥が右上に飛んでいきます



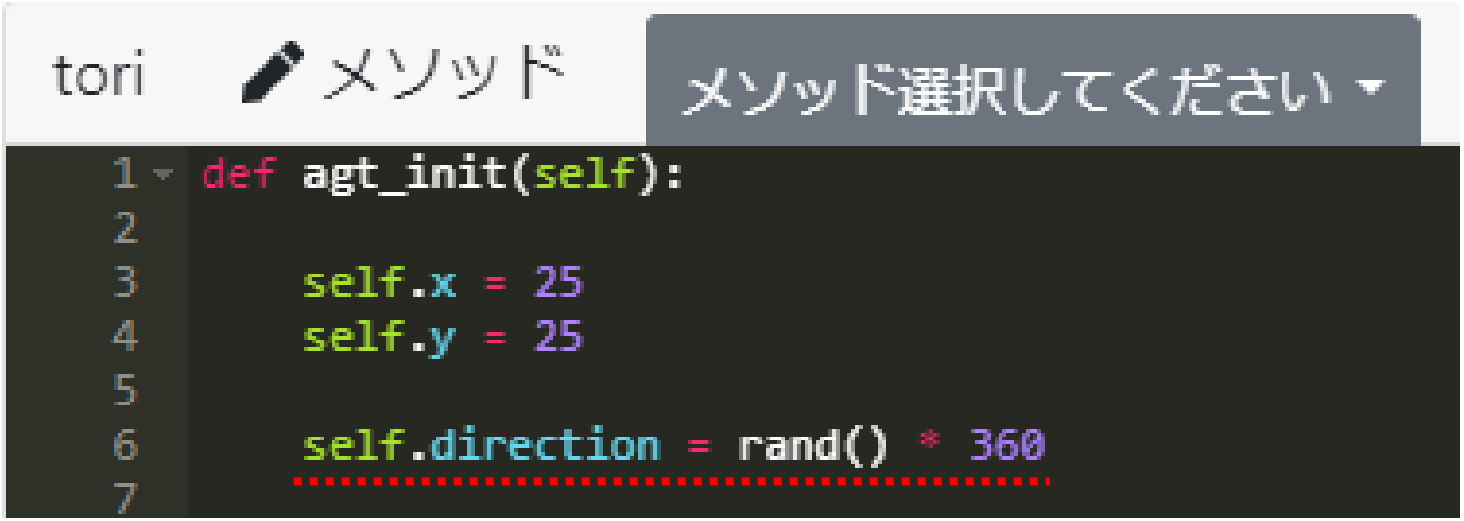
❒artisocにおける方向について



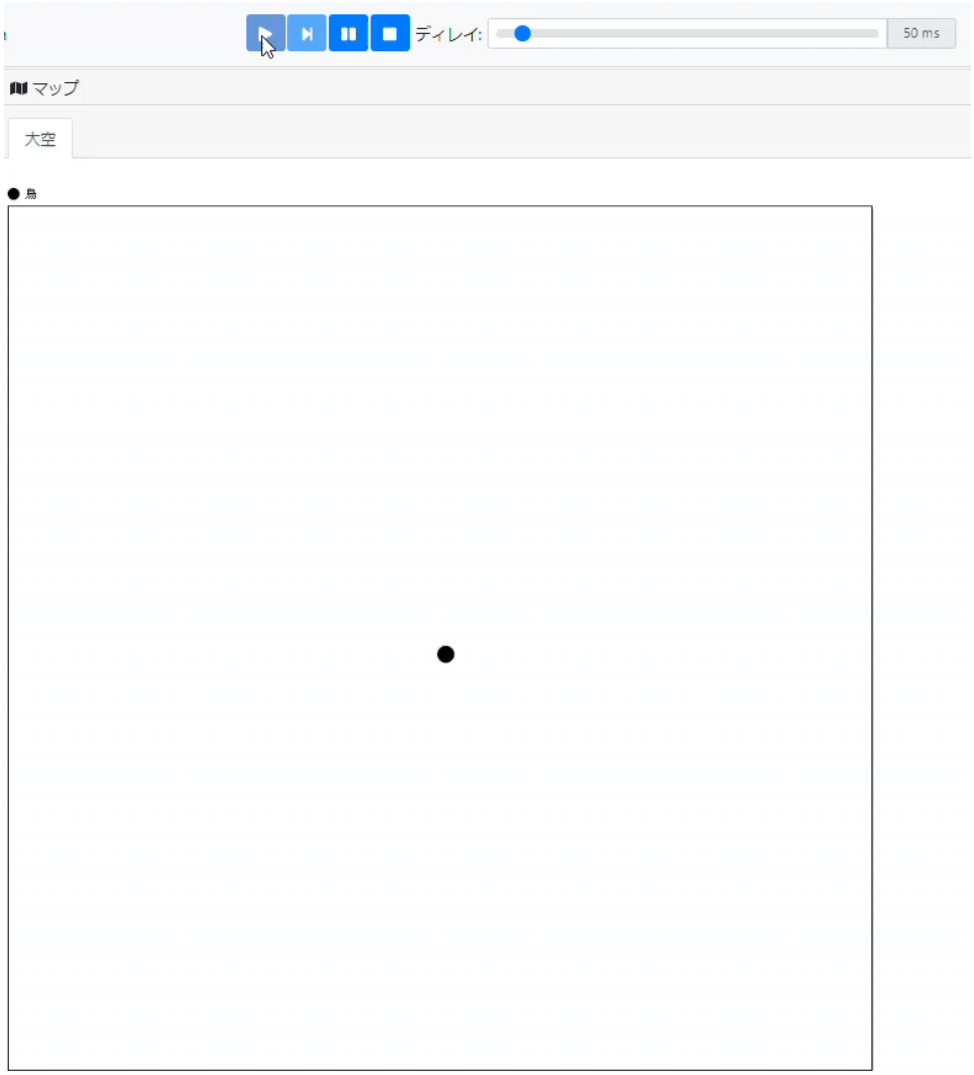
※左下原点の場合

❒進む方向をランダムに変える

- 属性「direction」に進む方向の値をランダムに設定（0~360度）して実行します
  - rand：0~1のランダムな値（一様乱数）を取得する関数
  - 「rand() * 360」で0~360のランダムな値を意味します
    - 「*」は掛け算の記号です
    - 関数には必ず()がつきます



■ 実行と停止を繰り返すと、実行するたびに鳥の飛ぶ向きが変わります



❒関数とは

- 関数：処理をひとまとめにしたもの
  - 引数：関数へ渡す値
  - 処理：引数をもとに関数が行う動作
  - 返り値（戻り値）：処理をもとに関数が返す値

  ※引数は複数あったり、なかったりします

  ※返り値がない（使わない）関数もあります
- 関数の例：forward
  - 引数：進む距離
  - 処理：引数の値だけ前に進む
  - 返り値：正常終了時は-1（いまは使っていません）

```
self.forward(10)  # 引数の数10だけ前に進む
```

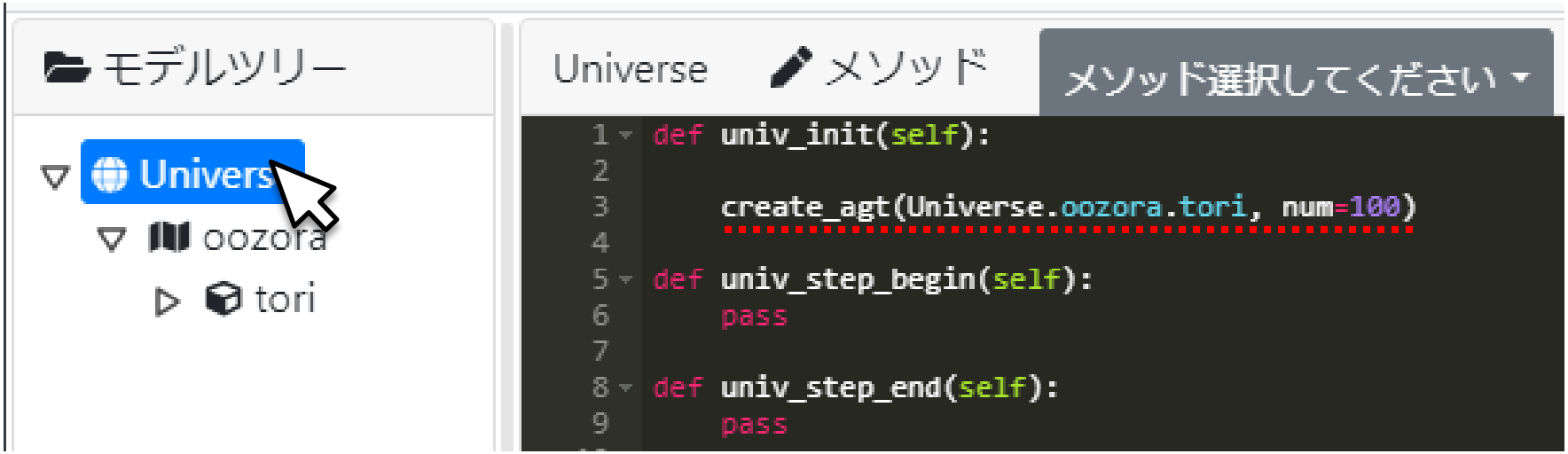
- 関数の例：rand
  - 引数：なし
  - 処理：0~1の一様乱数を発生させる
  - 返り値：0~1の一様乱数

```
r = rand()  # 0~1のランダムな値をrに代入する
```

※引数がないときも()が必要

❒エージェントの生成の仕方を変える

- エージェントを一度に複数生成する
  - 「create_agt([エージェント種別], num=[エージェント数])」で指定した数のエージェントを生成します
  - 100個生成します

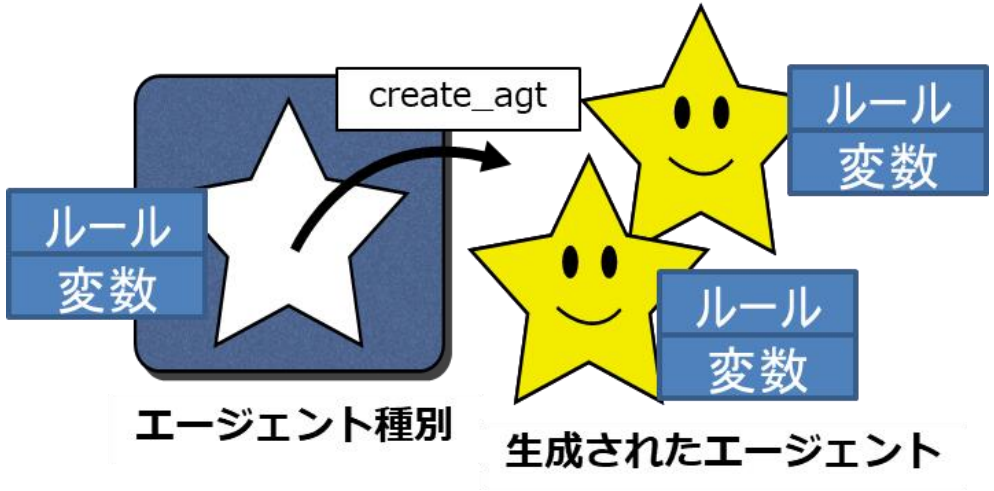


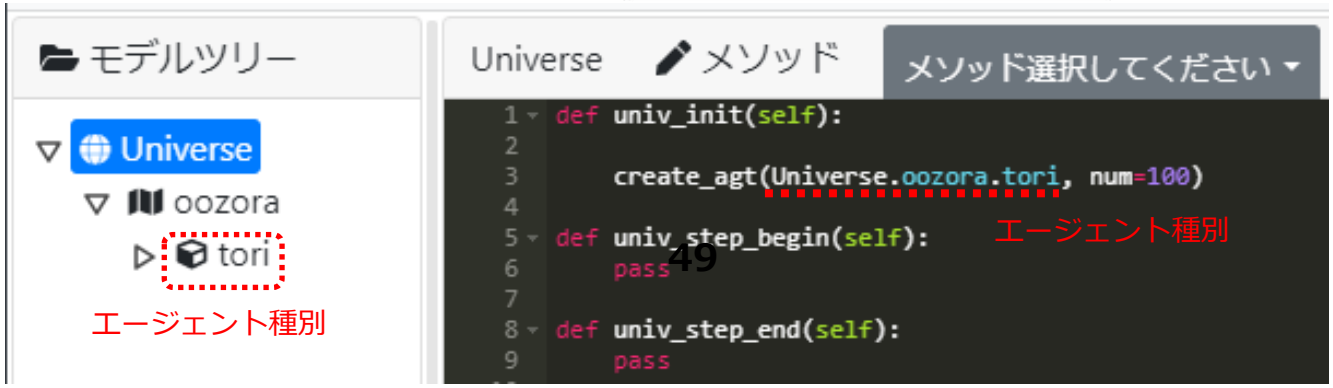
❐エージェントの生成の仕方を変える

- 100羽の鳥が円形に飛んでいきます

❒エージェント種別とエージェント

- エージェント種別：エージェントの「ひながた」のようなもの
  - ツリー上に存在するのはエージェントでなくエージェント種別です
- create_agtはエージェント種別から具体的なエージェントを生成する処理を表します
- 生成されたエージェントは共通のルールと変数を持ち、変数の値はそれぞれのエージェントで異なります
  - 共通のルール：「directionをランダムに設定する」→direction変数の値は個々のエージェントで異なる





❒飛ぶ速さを変える

- 個々のエージェントで飛ぶ速度を変えてみましょう
  - 飛ぶ速さを表す新たなエージェントの変数「speed」を作成します

❒飛ぶ速さを変える

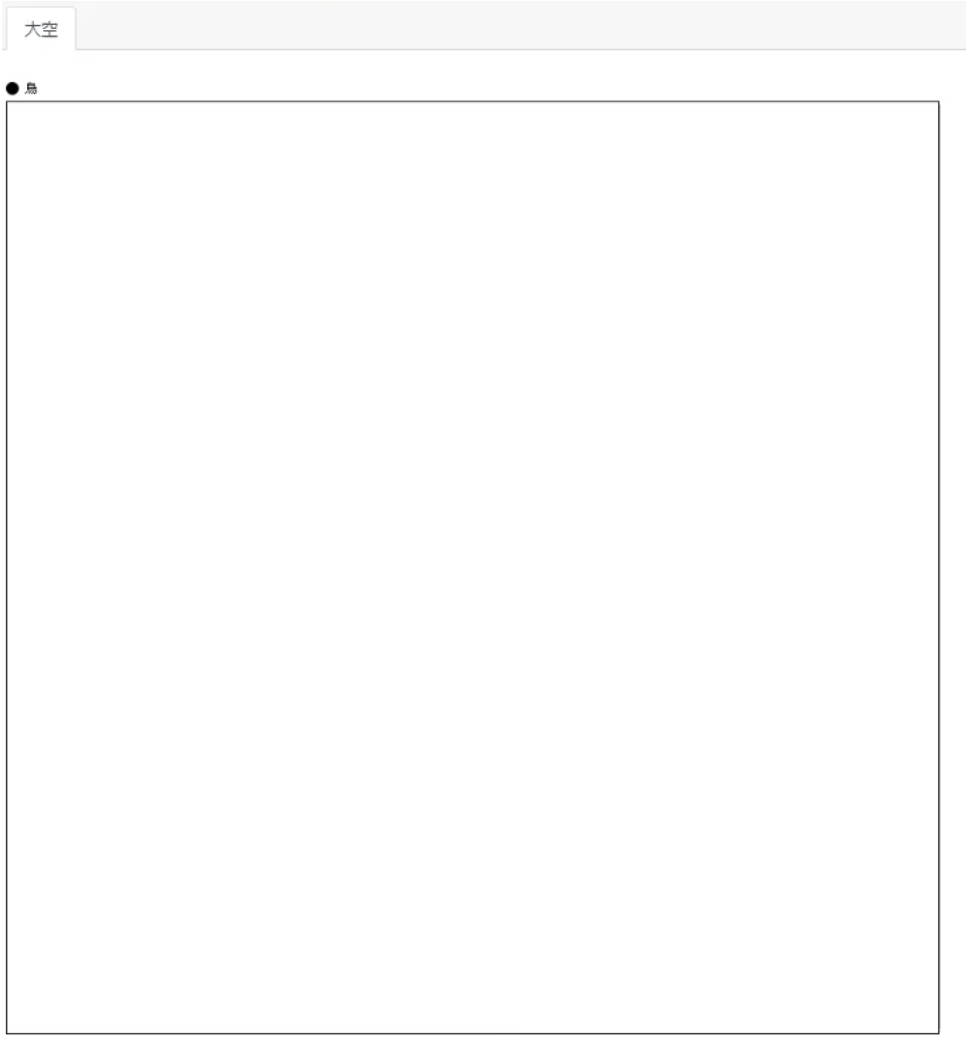
- speedの値をランダムに設定し、飛ぶ速さに指定する
  - agt_initでspeedにランダムな値を指定
    - ➡「1 + rand()」とすることで1~2の間のランダムな値をとる
  - forwardの引数にspeedを指定

tori　メソッド　メソッド選択してください

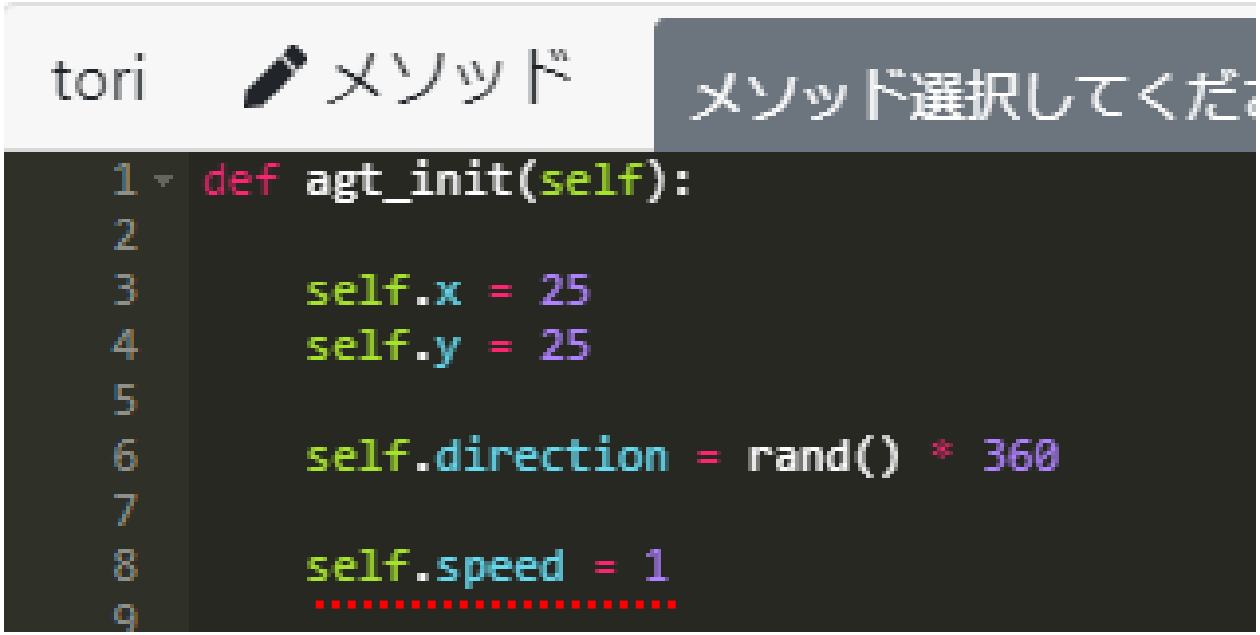
```
1  def agt_init(self):
2
3      self.x = 25
4      self.y = 25
5
6      self.direction = rand() * 360
7
8      self.speed = 1 + rand()
9
10 def agt_step(self):
11
12     self.forward(self.speed)
13
```

❐飛ぶ速さを変える

- 速度が異なるため、散らばって飛んでいきます

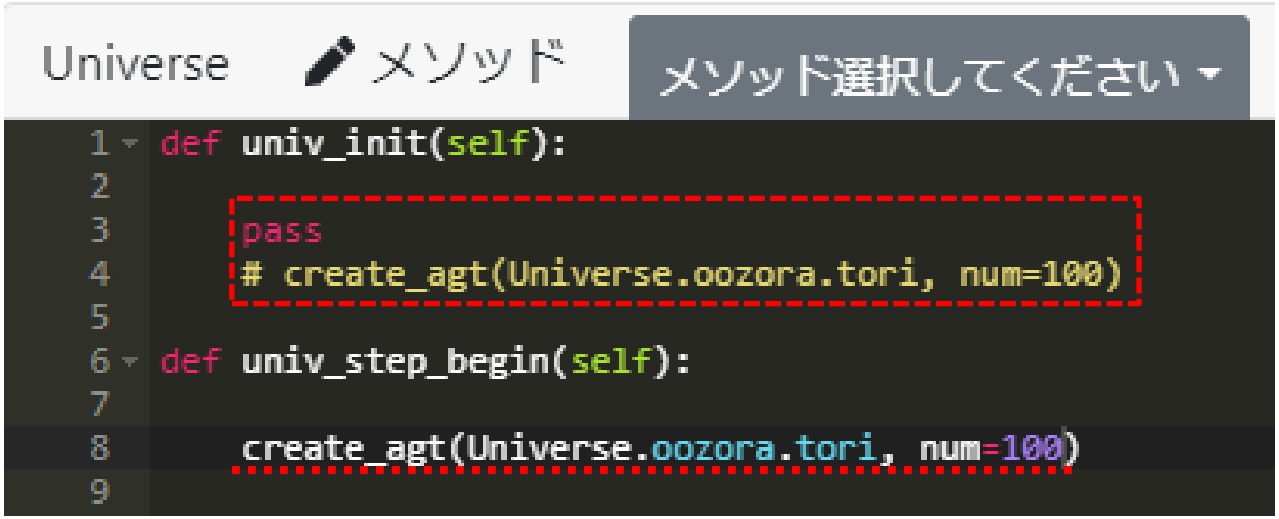


- 速度は1に戻しておきます



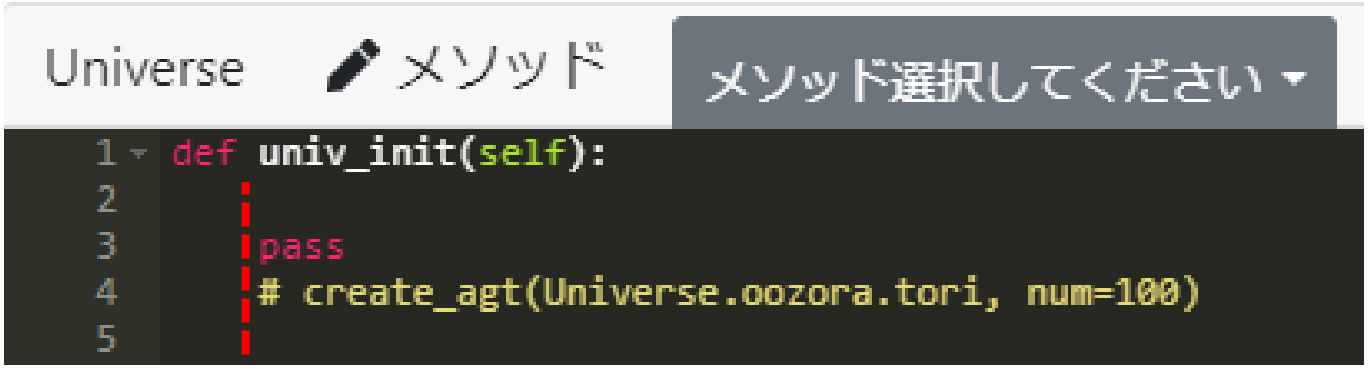
❒エージェントの生成の仕方を変える

- エージェントを毎ステップ生成する
  - create_agtをuniv_step_beginに移します
    - ➡ これで、各ステップの開始時にエージェントを生成します
    - ➡ univ_initに書いてあったルールはコメントアウトし、「pass」を追記します
      - ☑ 次ページ参照



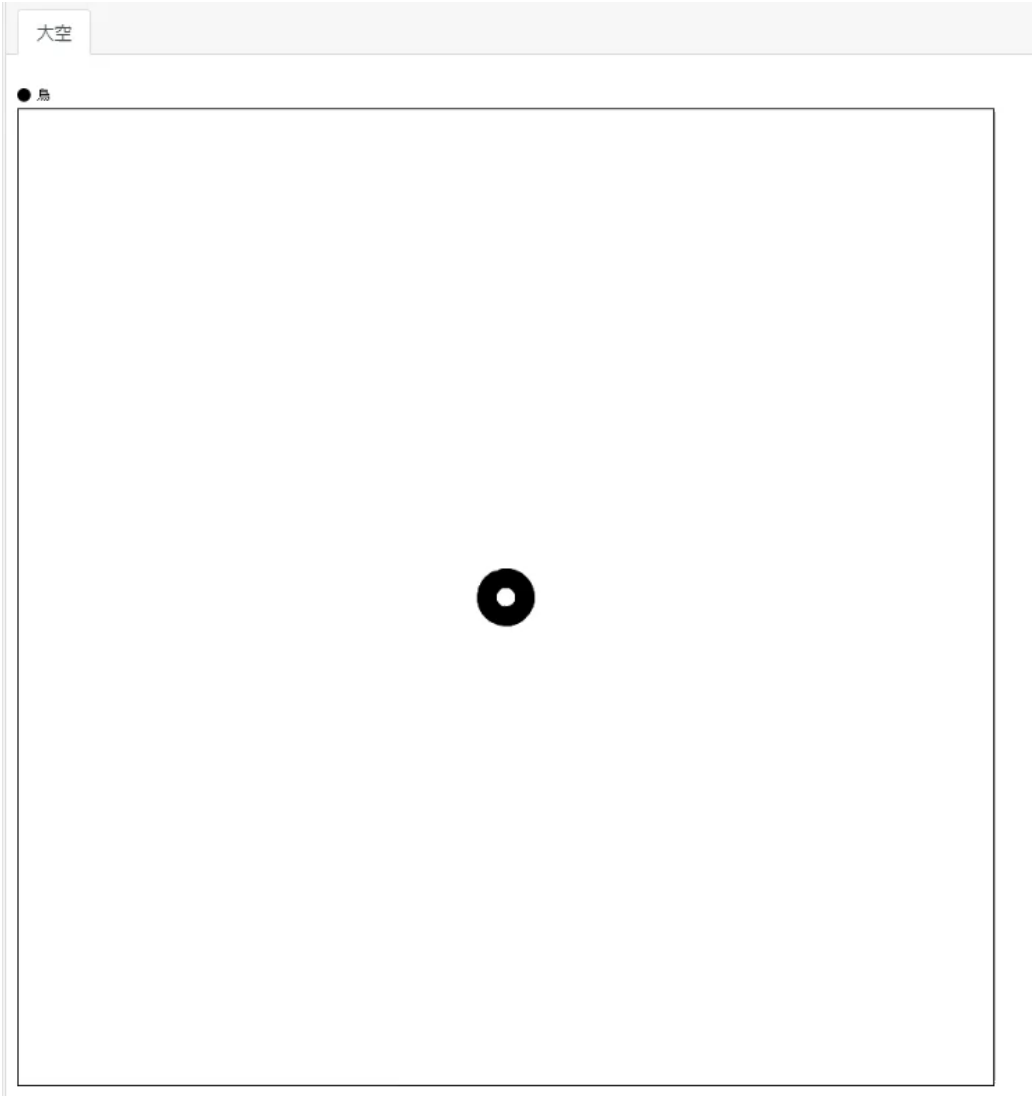
❒artisocヒント_コメントアウトとpass

- コメントアウトについて
  - コードを無効化する機能
  - 行の先頭に「#」
    - 行にカーソルを合わせたり選択した状態で「ctrl + /」でもコメントアウトできます
  - コードを一時的に変更したり、説明を加えるのに使います
- passについて
  - 「何もしない」を意味するルール
  - univ_initなどのルールエディタの要素内では、インデントの中に何かを記述する必要がある
    - でないと、「インデントがない」というエラーになる
  - 何もしないときは、「何もしない」ということをartisocに教えるためにpassを記述します



❐エージェントの生成の仕方を変える

- 毎ステップ、エージェントが生成されます



❒ルールの構成

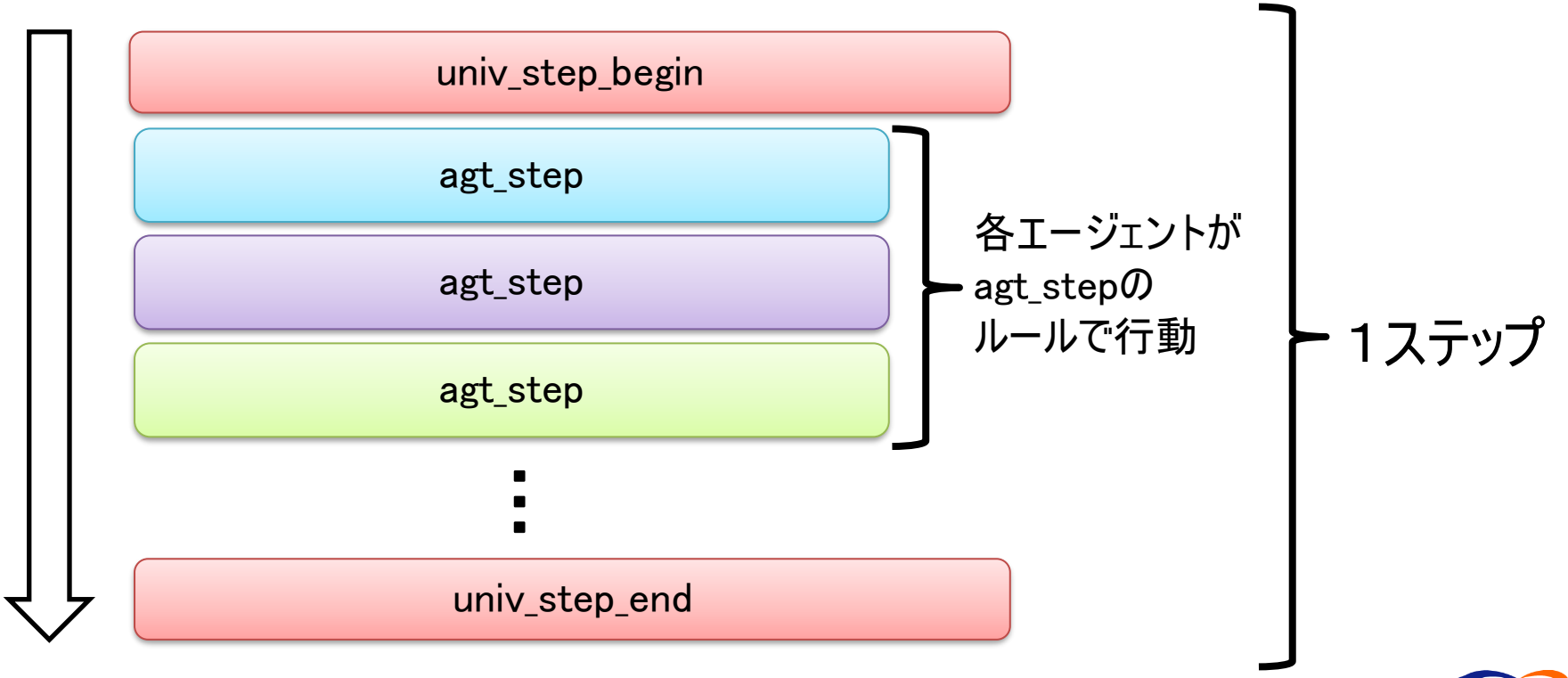
- エージェントのルールエディタ→エージェントの行動ルール
  - agt_init・・エージェント生成時に一度だけ実行
  - agt_step・・各ステップで実行

- Universeのルールエディタ→モデル全体のルール
  - univ_init・・シミュレーション開始時に一度だけ実行
  - univ_step_begin・・各ステップの最初に実行
  - univ_step_end・・・各ステップの最後に実行
  - univ_finish　・・シミュレーション終了時に一度だけ実行

ステップについて

- ステップ
  - artisocの時刻単位
  - 1ステップ=全てのエージェントがルールにしたがって行動

- 1ステップの流れ



❒条件分岐文

- 「条件分岐文」という重要な手法について学びます
- 「10ステップ目までエージェントを毎ステップ生成」というルールを考える
  - 「現在のステップ数が10以下の場合、エージェントを生成」という場合分けのルールを書く
  - 場合分けの処理→**if文**で記述

条件文が真の場合に実行する処理

```
if 条件文:
    処理
```

※処理はインデント内に記述

条件文の例

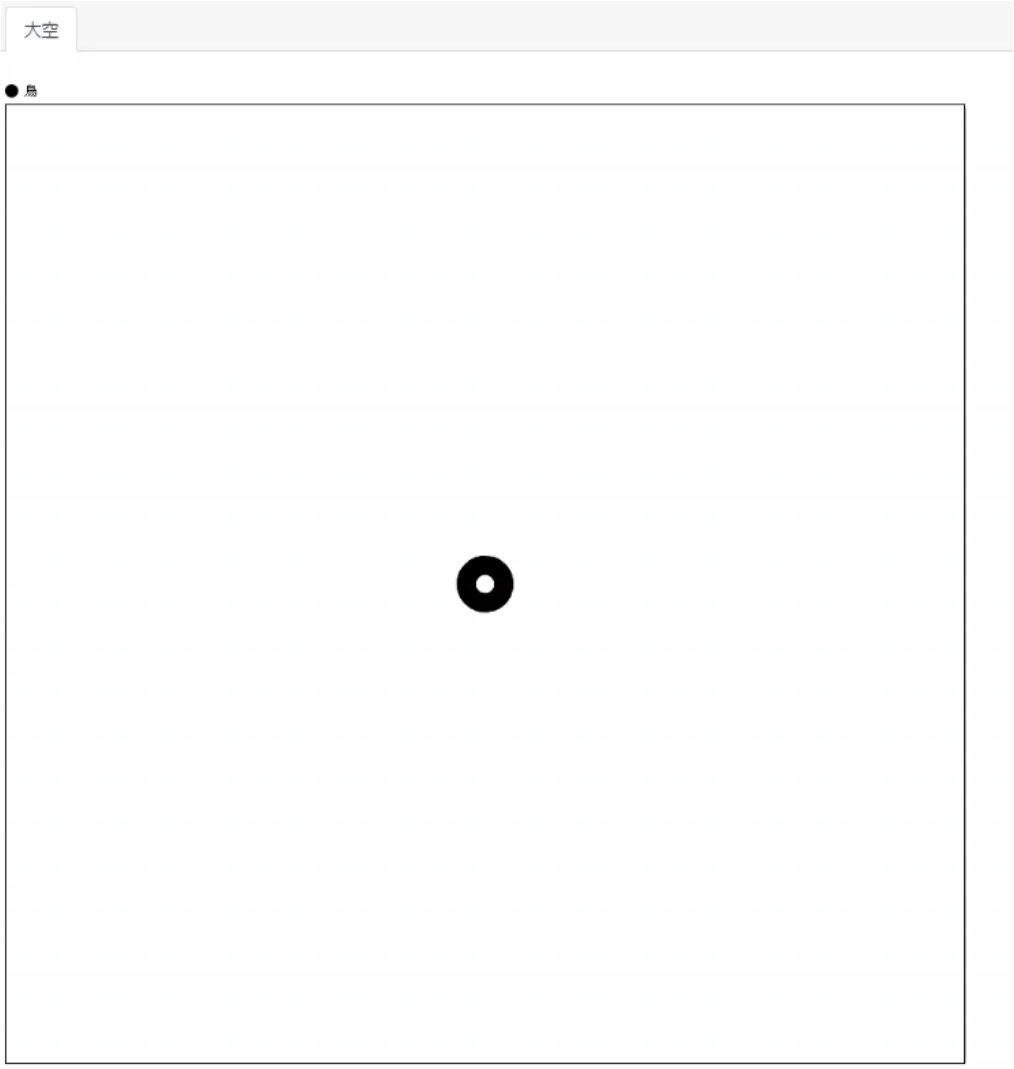
1 < 5　[1は5未満である] ⇒　真(True)
4 >= 8 [4は8以上である] ⇒　偽(False)
3 == 4 [3と4は等しい]　⇒　偽(False)

```
def univ_step_begin(self):

    if count_step() <= 10:
        create_agt(Universe.oozora.tori, num=100)
```

※count_step: 現在のステップ数を取得する関数

❐条件分岐文

- 10ステップ目までエージェントが生成されます



❒条件分岐文

- 毎ステップ0.2の確率でエージェントを生成する
  - rand()の値が0.2未満の場合エージェントを生成

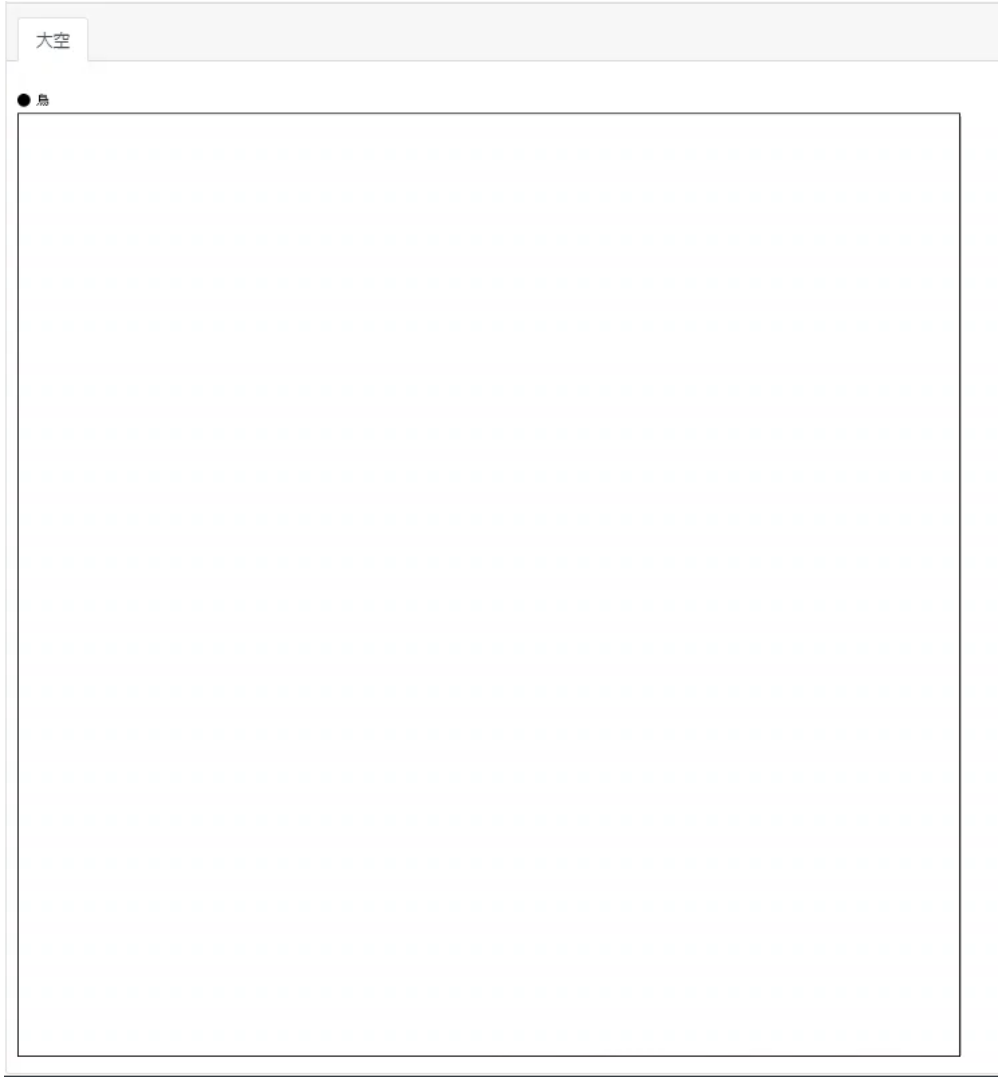
確率rで実行する処理

```
if rand() < r:
    処理
```

```
def univ_step_begin(self):

    if rand() < 0.2:
        create_agt(Universe.oozora.tori, num=100)
```

❐条件分岐文

- 5回に1回の確率でエージェントが生成されます



❒繰り返し文

- 「繰り返し文」について学びます
- 「シミュレーション開始時に100エージェントをまとめて生成する」というルールを考える
  - 先ほどは「num=100」でまとめて生成しましたが、今回はエージェントを1つ作る処理を100回繰り返します
  - 繰り返しの処理→**for文**を用います

n回繰り返す処理

```
for 変数 in range(n):
    処理
```

※処理はインデント内に記述
※変数は何でもかまいませんが、慣習としてiをよく使います

  - univ_initに記述し、univ_step_beginの内容はコメントアウトします

❐繰り返し文

- 鳥が円形に飛んでいきます

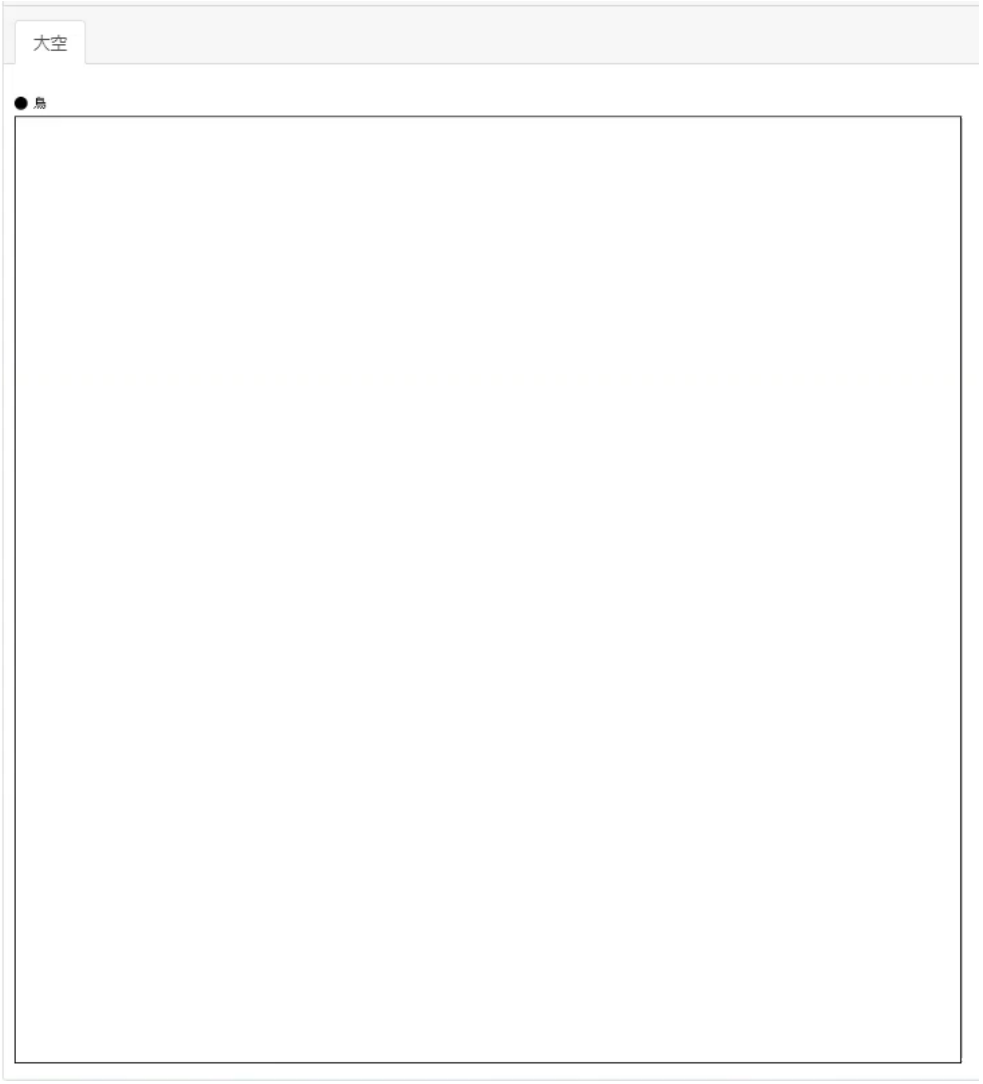
❒繰り返し文

- 「シミュレーション開始時に100エージェントを生成し、それぞれに0度～99度の向きを与える」というルール
  - for文を用いて以下のように書いてみましょう

```
def univ_init(self):

    for i in range(100):
        one = create_agt(Universe.oozora.tori)
        one.direction = i
```

❐繰り返し文

- 鳥が扇形に飛んでいきます



❒繰り返し文

- for文はより一般的には、変数の値を変化させながら繰り返す処理を意味します

変数を0からn−1まで変化させながらn回繰り返す処理

```
for 変数 in range(n):
    変数を用いた処理
```

※処理はインデント内に記述
※変数には慣習としてiをよく使う（integerの頭文字）

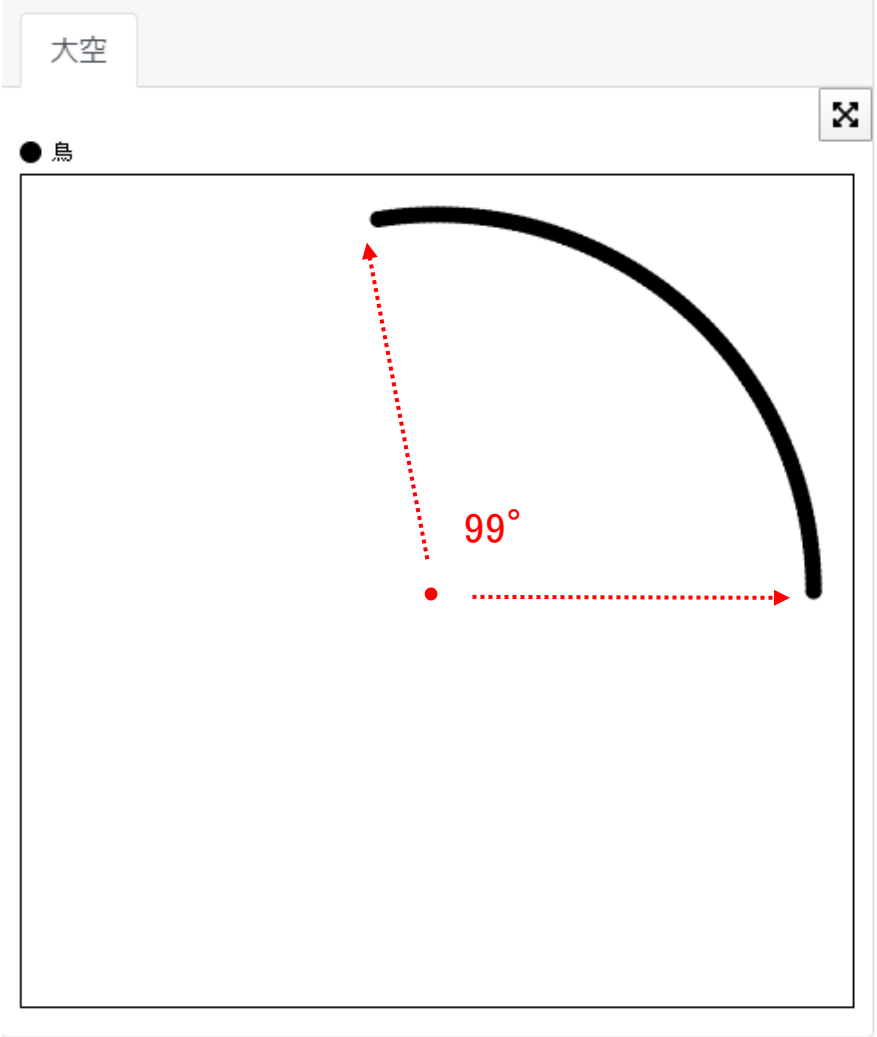
❒繰り返し文

```
def univ_init(self):

  ①for i in range(100):
      ②one = create_agt(Universe.oozora.tori)
      ③one.direction = i
```

①：iの値を0~99まで変えながら②③を100回繰り返す
②：生成したエージェントを変数oneに代入（create_agt関数の戻り値）
③：エージェントoneの変数directionにiを代入

```
one = create_agt(Universe.oozora.tori)
one.direction = 0                          0回目

one = create_agt(Universe.oozora.tori)
one.direction = 1                          1回目

…

one = create_agt(Universe.oozora.tori)
one.direction = 99                         99回目
```

❐繰り返し文



❒変数のデータ型について

- データ型：変数が持つデータの種類のこと
  - 数値だけでなく、様々な種類がある

主なデータ型

- 整数
- 実数（浮動小数点数）
- 文字列
- 論理値 TrueまたはFalseで表す

- エージェント
- エージェント種別
- 空間

etc・・・

データの集まりを扱うための型

- 集合（セット）
- リスト
- タプル
- 辞書

etc・・・

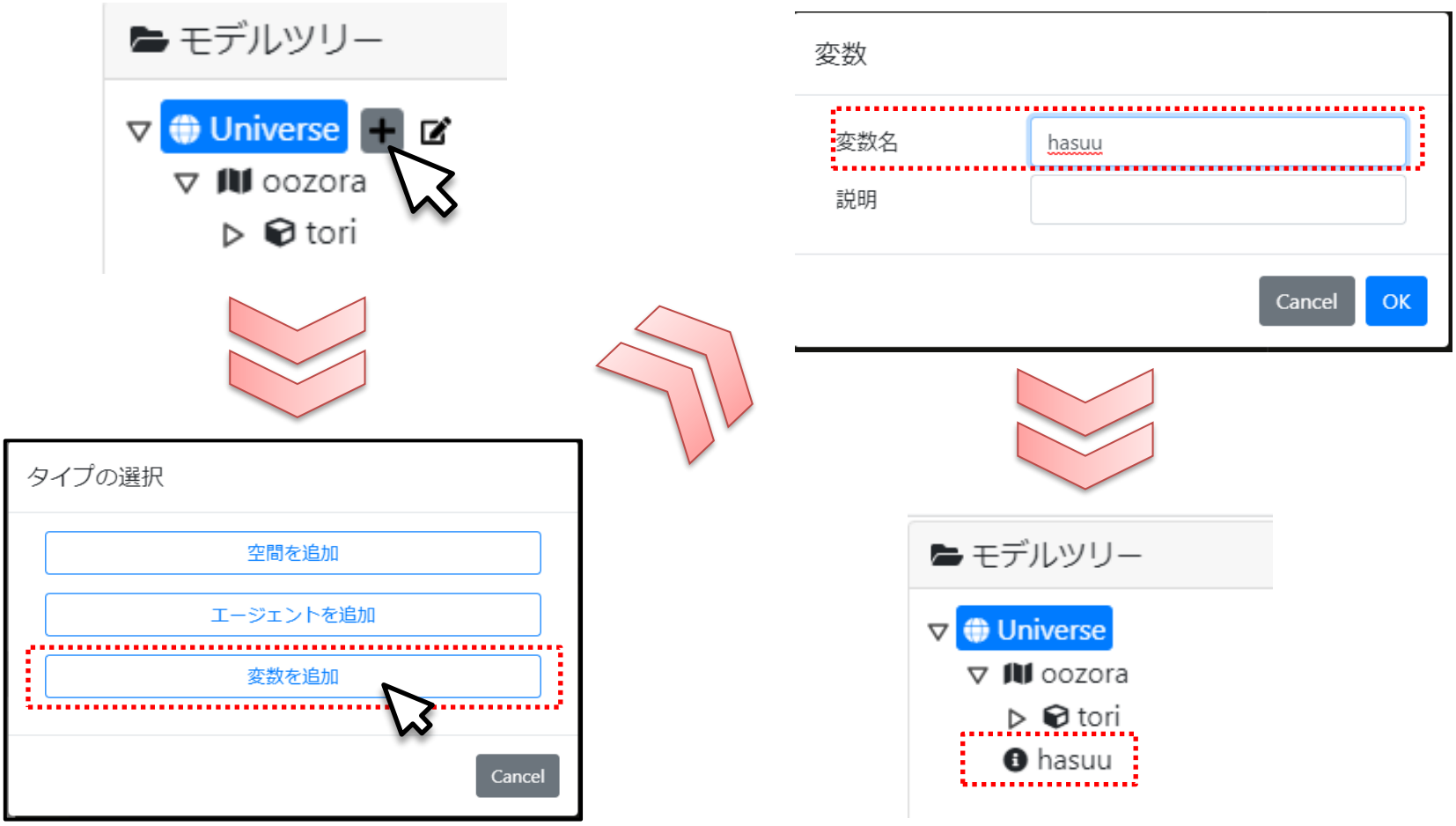
```
a = 3  # 変数aに整数型の値3を代入
b = 4.5  # 変数bに実数型の値4.5を代入
c = “artisoc Cloud”  # 変数cに文字列型の値”artisoc Cloud”を代入
one = create_agt(Universe.oozora.tori)  # 変数oneに生成したtoriエージェント（エージェント型）を代入
```

❐コントロールパネルによる初期値設定

- 鳥の数をスライドバーの操作で設定できるようにする

コントロールパネルによる初期値設定

① 鳥の数を表す変数としてUniverseの下に変数「hasuu」を作成



❒コントロールパネルによる初期値設定

② 変数「hasuu」を鳥の数に設定

```
def univ_init(self):

    for i in range(Universe.hasuu):
        one = create_agt(Universe.oozora.tori)
        one.direction = i
```

❒コントロールパネルによる初期値設定

③ Universe変数「hasuu」をコントロールパネルの操作対象に設定

- 出力画面に移動し、実行画面左上の「コントロールパネル」で設定
- 「種類」に「スライドバー」を選択

❒コントロールパネルによる初期値設定

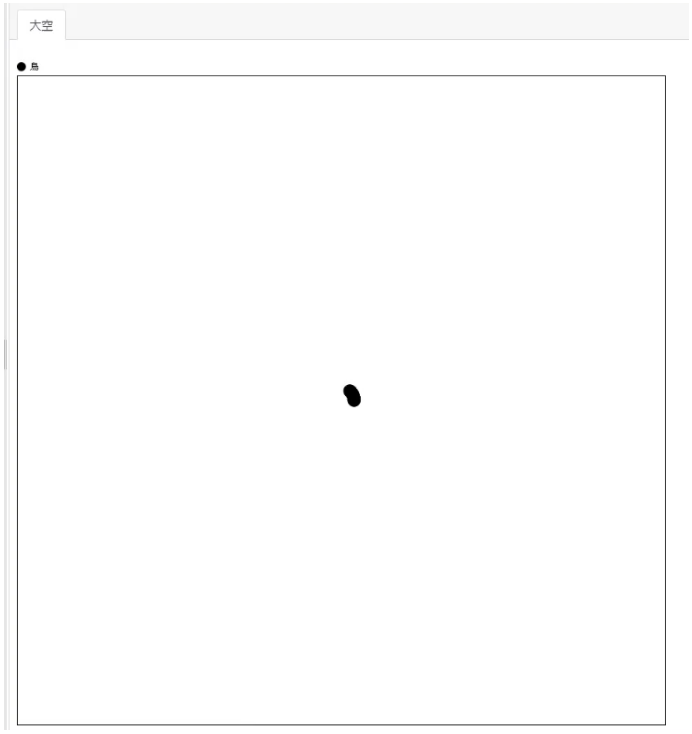
■ 鳥の数をスライドバーの操作で設定できるようにする

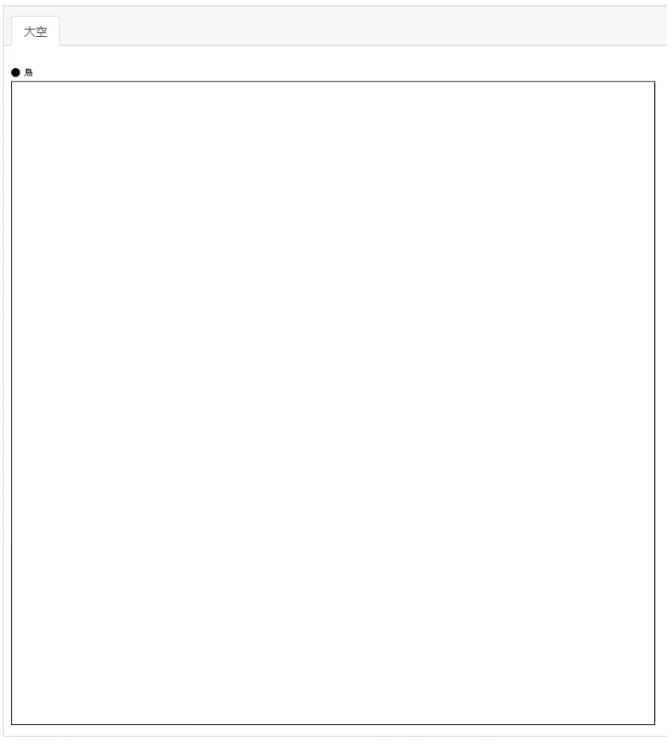
③ Universe変数「hasuu」をコントロールパネルの操作対象に設定

❒ 下図のように項目を設定します

❒コントロールパネルによる初期値設定

■ コントロールパネルで鳥の数を設定できます





❒artisoc Cloudにおける変数の種類

- artisoc Cloudの変数は作成する場所でも分類され、以下のように使い分けます
  - Universe変数　・・・モデル全体の性質を表す（例：鳥の数）
  - エージェント変数・・エージェントの性質を表す（例：鳥の位置、向き、速さ）
  - ローカル変数　・・・ルールエディタ上で一時的に使う変数（例：鳥エージェントone)
  - 空間変数　　・・・・空間の性質を表す(今回は使いません)


```
def univ_init(self):

    for i in range(Universe.hasuu):
        one = create_agt(Universe.oozora.tori)
        one.direction = i
```

ローカル変数

III. 相互作用を含むモデルを作成する

❒相互作用を含むモデルを作成する

鳥が群れながら飛べるのはなぜ？

×リーダーの鳥が指示している

〇個別の鳥が周囲の鳥に飛び方を合わせている

→ボイドモデル（Craig Raynolds, 1987）

ごく単純なボイドモデルを作り、群れながら飛ぶ鳥を表現しましょう

❒相互作用を含むモデルを作成する

- 途中から来た人や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://artisoccloud.kke.co.jp/models/rmJfb1MiRrCPZYh6xPtqAQ
    - チャットでURLを送ります
  - 継承：他のユーザが作ったモデルをコピーして編集する

❒準備

- Universeのルールを以下のように修正します
  - 向きを指定しない（コメントアウトする）
    - 向きはagt_initでランダムに指定される

- 鳥のルールを以下のように修正します
  - 初期位置をランダムに
    - 「rand() * 50」で0~50のランダムな値

Universe　メソッド　メソッド選択してくださ

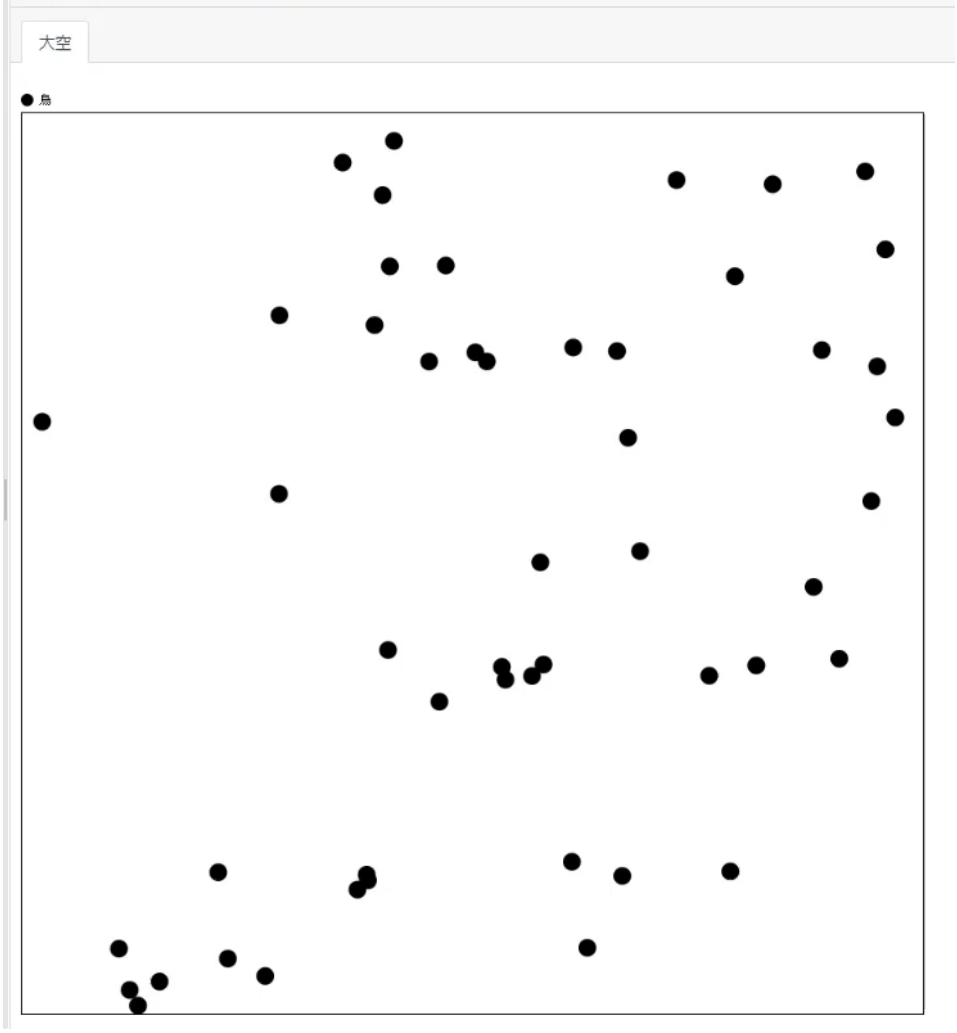
```
1  def univ_init(self):
2
3      for i in range(Universe.hasuu):
4          one = create_agt(Universe.oozora.tori)
5          # one.direction = i
6
```

tori　メソッド　メソッド選択してください

```
1  def agt_init(self):
2
3      self.x = rand() * 50  # ランダムな初期位置
4      self.y = rand() * 50  # ランダムな初期位置
5
6      self.direction = rand() * 360
7
8      self.speed = 1
9
10 def agt_step(self):
11
12     self.forward(self.speed)
13
```

❒準備

■ 鳥がランダムに飛んでいきます（鳥の数=50）



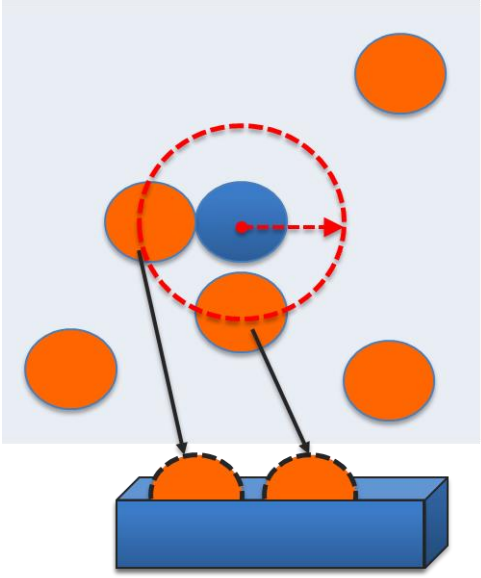
❒周囲の鳥に方向を合わせる

■ 鳥エージェントのルールに、周囲の鳥を認識し飛び方を合わせるルールを追記します

① 周囲のエージェントを探し、**エージェント集合型**の変数に格納する
- エージェント集合型変数：エージェントの集合を値として持つ変数

② エージェント集合型の変数に入っているエージェント数を数える
- 1以上の場合、周囲にエージェントがいる

③ 周囲にエージェントがいる場合、そのうち1つのエージェントを選び、自分の飛ぶ向きをそのエージェントに合わせる



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）

❒周囲の鳥に方向を合わせる

- 周囲のエージェントを取得し、エージェント集合を変数に格納します
  - agt_stepに記述します
  - 用いる関数：make_agtset_around_own
    - 視野の範囲内のエージェント集合を取得する
    - 第1引数：視野
    - 第2引数：自分を含むかどうか（TrueかFalse）

```
def agt_step(self):

    # 自分から距離2以内のエージェントをtori_setに格納（自分自身を含まない）
    tori_set = self.make_agtset_around_own(2, False)

    self.forward(self.speed)
```

❐周囲の鳥に方向を合わせる

- もし周囲に鳥がいれば、そのうち1羽を選び、自分の方向をその鳥に合わせます
  - count_agtset: エージェント集合の要素数を数える関数
  - randchoice: エージェント集合からランダムにエージェントを1つ取得する関数

```
def agt_step(self):

    # 自分から距離2以内のエージェントをtori_setに格納（自分自身を含まない）
    tori_set = self.make_agtset_around_own(2, False)

    if count_agtset(tori_set) > 0:  # もし周囲に鳥がいれば
        one = randchoice(tori_set)  # ランダムに1羽を選ぶ
        self.direction = one.direction  # 自分の向きをその鳥に合わせる

    self.forward(self.speed)
```

※ここで、tori_setは順序を持たない集合型。artisoc4でのAgtsetと異なり、位置を指定してエージェントを取り出すことができないことに注意。

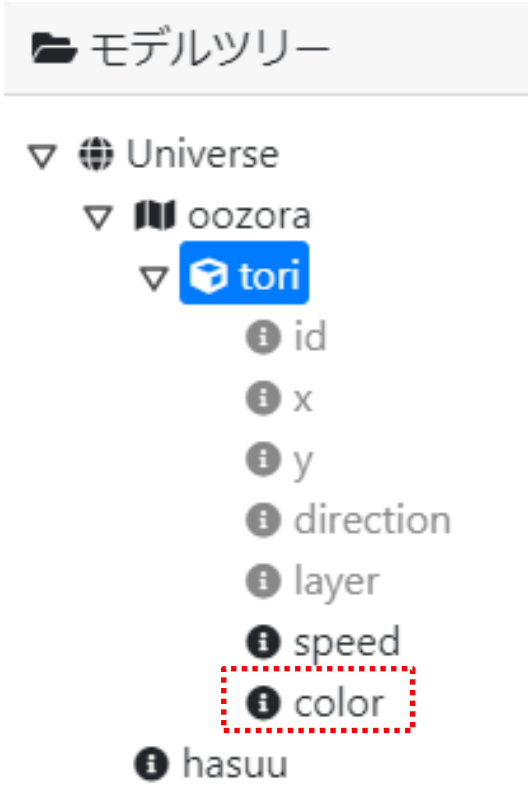
❒周囲の鳥に方向を合わせる

- 鳥が群れを作ります（鳥の数=50）

❒出力を工夫する

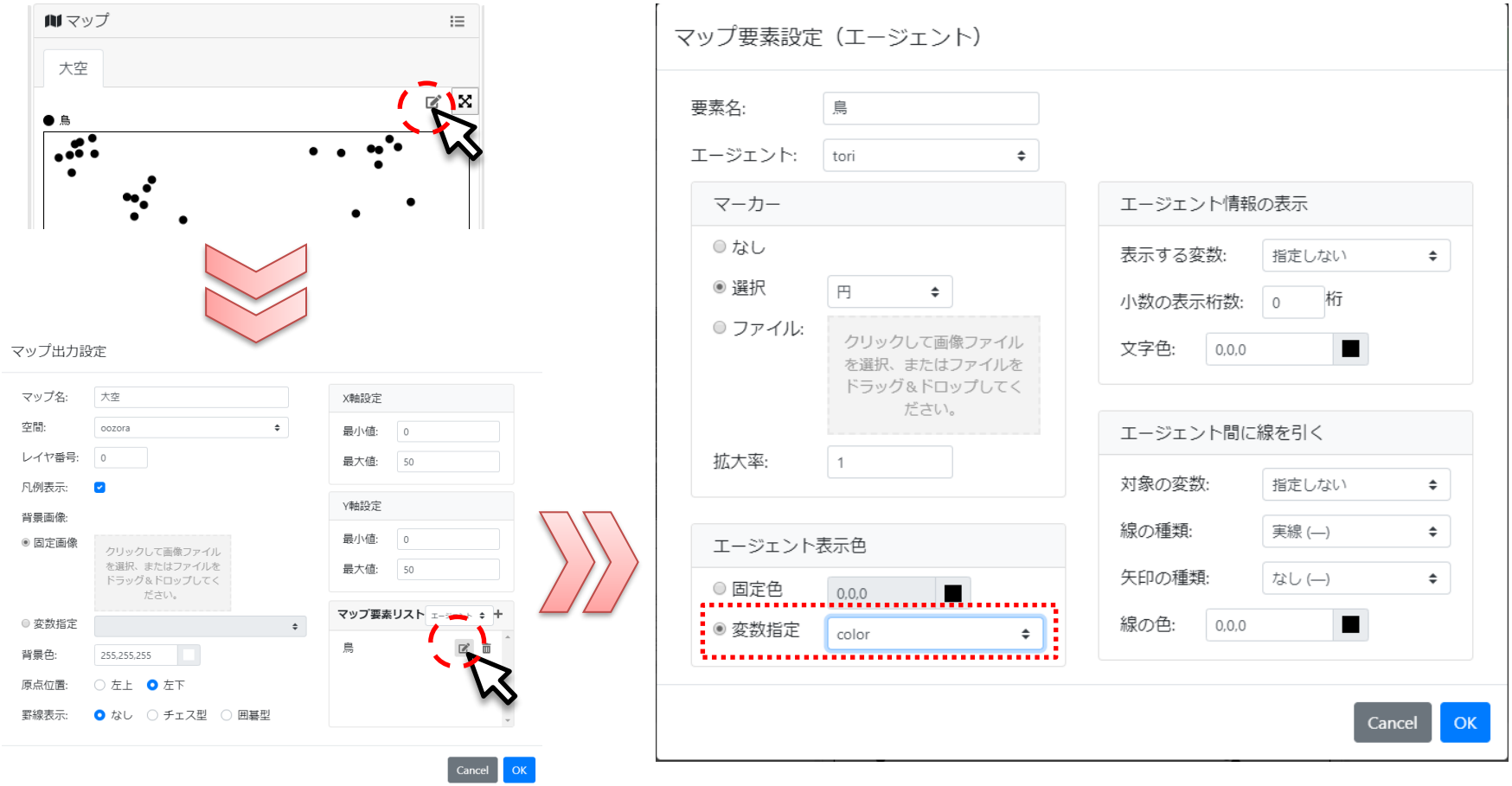
- 以下のようなルールを追記します
  - 群れを形成している場合、エージェントは赤色
  - 群れを形成していない場合、エージェントは青色

- 鳥エージェントに色を表す変数colorを追加します



❒出力を工夫する

- 出力設定で変数colorをエージェントの色に指定します
  - マップの編集→マップ要素リスト（鳥）の編集→エージェント表示色
    - 「固定色」：設定画面で指定した色で表示
    - 「変数指定」：変数に格納された色で表示



❒出力を工夫する

- ルールで変数colorの値を指定します
  - 周囲に鳥がいるとき→COLOR_RED
  - 周囲に鳥がいないとき→COLOR_BLUE

～の場合に実行する処理

```
if（条件文）:
    ～条件に当てはまる場合の処理～
else:
    ～条件に当てはまらない場合の処理～
```

色を表す変数

COLOR_RED	赤色
COLOR_GREEN	緑色
COLOR_BLUE	青色
COLOR_YELLOW	黄色
COLOR_CYAN	水色
COLOR_MAZENTA	紫色
COLOR_BLACK	黒色
COLOR_WHITE	白色

```
def agt_step(self):

    # 自分から距離2以内のエージェントをtori_setに格納（自分自身を含まない）
    tori_set = self.make_agtset_around_own(2, False)

    if count_agtset(tori_set) > 0:  # もし周囲に鳥がいれば、
        one = randchoice(tori_set)  # ランダムに1羽を選ぶ
        self.direction = one.direction  # 自分の向きをその鳥に合わせる
        self.color = COLOR_RED  # 群れを形成するので赤色
    else:                       # 周囲に鳥がいなければ、
        self.color = COLOR_BLUE  # 群れを形成しないので青色

    self.forward(self.speed)
```

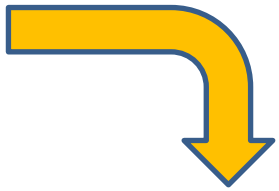
❐出力を工夫する

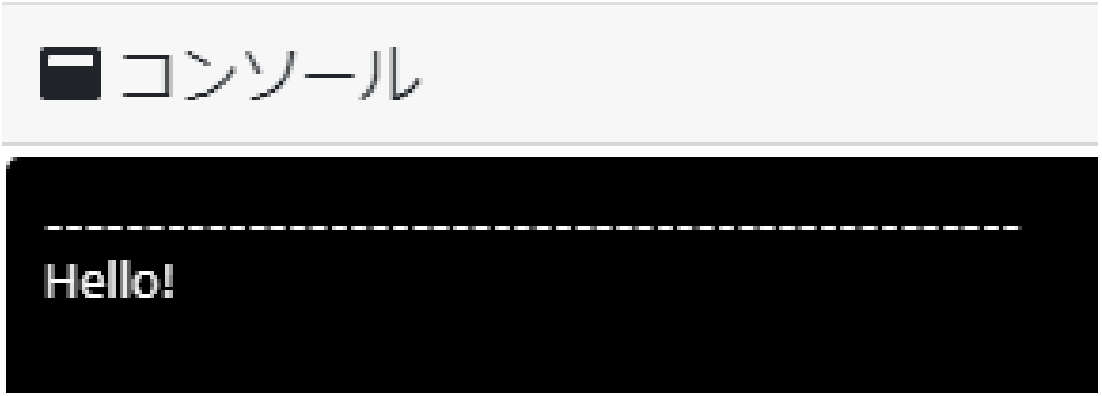
- 群れを形成しているかどうかによって色が変わります（鳥の数=50）

❒コンソール出力機能

- コンソール画面に任意の文字列を表示します
  - print: コンソールに文字を表示する関数
  - シミュレーションの状況把握に役立ちます

```
def univ_init(self):
    print("Hello!")
```





❑コンソール出力機能

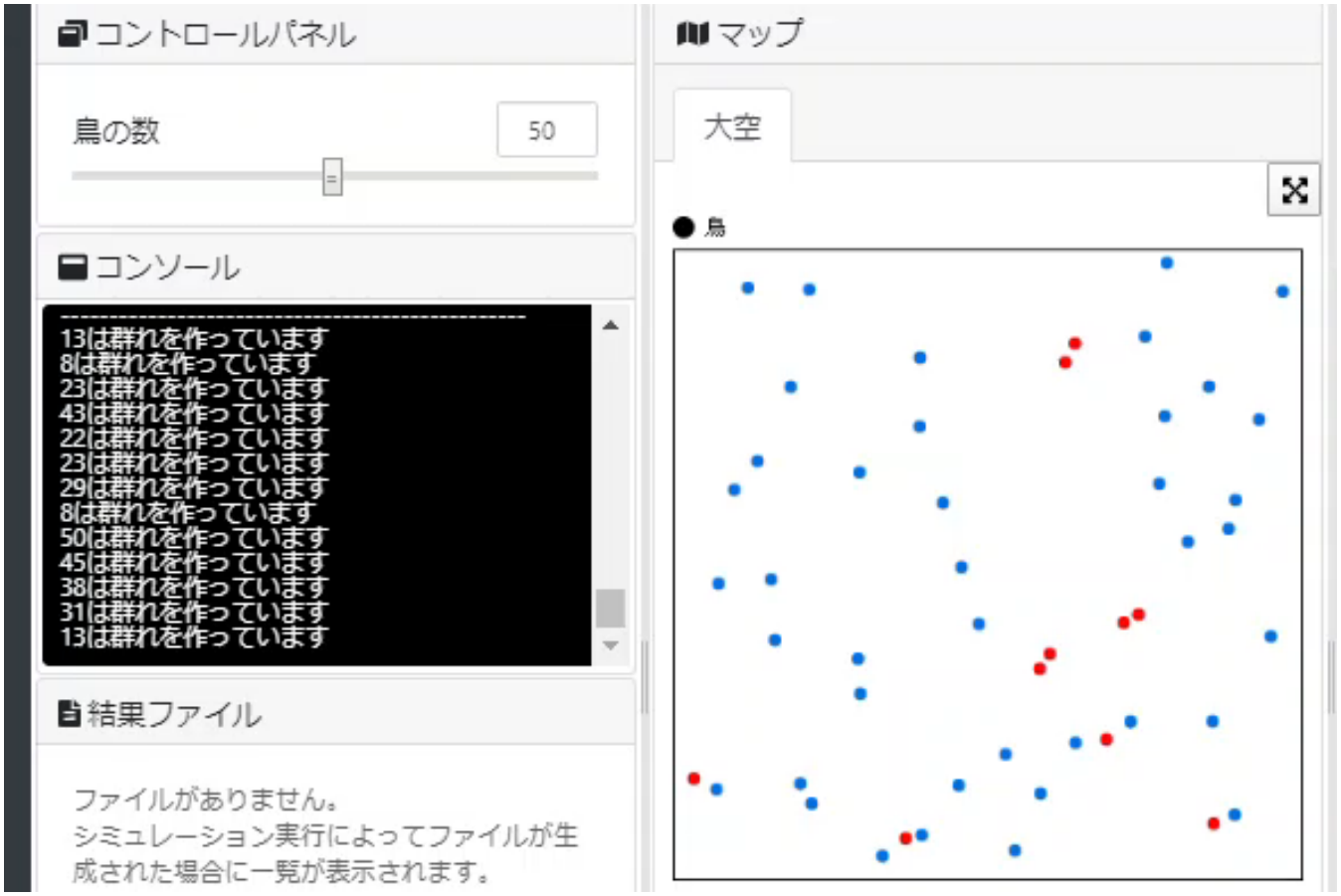
- 群れを作った場合はコンソールに表示する
  - 「[自分のID]は群れを作っています」とコンソール画面に表示

```
def agt_step(self):

    # 自分から距離2以内のエージェントをtori_setに格納（自分自身を含まない）
    tori_set = self.make_agtset_around_own(2, False)

    if count_agtset(tori_set) > 0:  # もし周囲に鳥がいれば
        one = randchoice(tori_set)  # ランダムに1羽を選ぶ
        self.direction = one.direction  # 自分の向きをその鳥に合わせる
        self.color = COLOR_RED  # 群れを形成するので赤色
        print(str(self.id) + "は群れを作っています")
    else:                           # 周囲に鳥がいなければ、
        self.color = COLOR_BLUE  # 群れを形成しないので青色

    self.forward(self.speed)
```

※文字列と数値を続けて表示する場合、数値を関数「str」で文字列に変換し、「+」で繋ぐ
例） Print(“私のIDは” + str(self.id) + “です”)

❒コンソール出力機能

■ コンソール画面に表示されます



❒発展問題

- 向きを完全に合わせるのでなく、少しゆらぎをつけます
  - 「+=」で左辺に右辺を足します
  - 「+= rand() * 10 – 5」で-5~5のランダムな値を足します

```
def agt_step(self):

    # 自分から距離2以内のエージェントをtori_setに格納（自分自身を含まない）
    tori_set = self.make_agtset_around_own(2, False)

    if count_agtset(tori_set) > 0:  # もし周囲に鳥がいれば、
        one = randchoice(tori_set)  # ランダムに1羽を選ぶ
        self.direction = one.direction  # 自分の向きをその鳥に合わせる
        self.direction += rand() * 10 - 5  # 向きにゆらぎをつける
        self.color = COLOR_RED  # 群れを形成するので赤色
    else:                               # 周囲に鳥がいなければ、
        self.color = COLOR_BLUE  # 群れを形成しないので青色

    self.forward(self.speed)
```

❒発展問題

- 鳥の速度の初期値をランダムにし、速度も周囲の鳥に合わせる

- 鳥の認識できる範囲を操作できるようにする
  - 視野の広さを表す変数をUniverseに追加し、コントロールパネルで操作できるようにする
  - make_agtset_around_own関数でそれを用いる

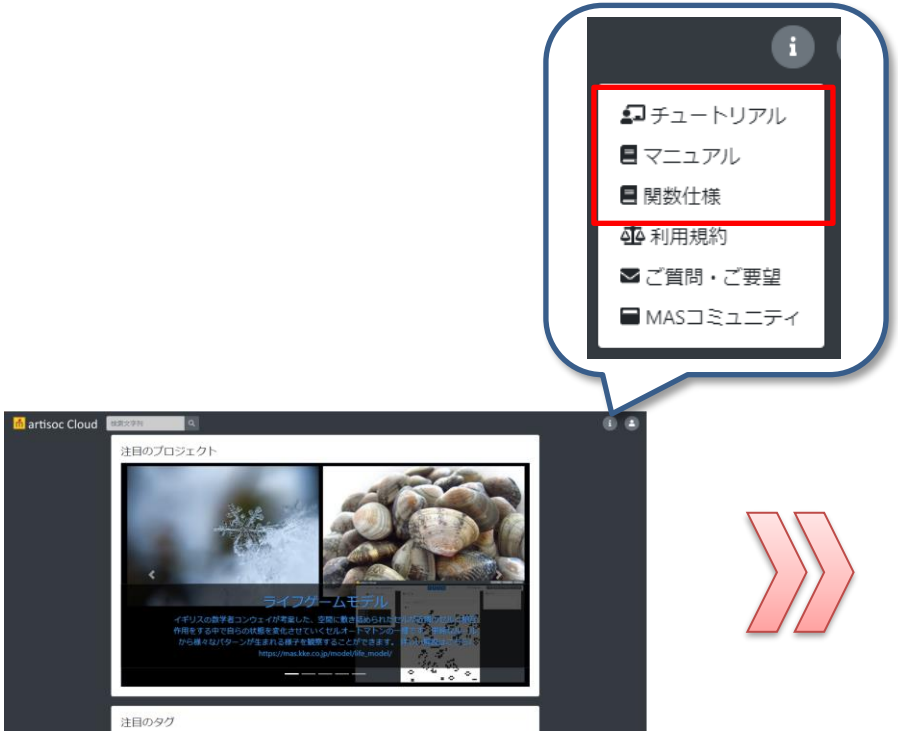
おわりに

❒モデルの公開

- オリジナルのモデルを作成したら、公開しましょう
  - 公開すると、ユーザ登録者なら誰でもモデルを見ることができます
  - コメントなどをつけることも可能です
  - URLを共有すれば、モデルの共有ができます

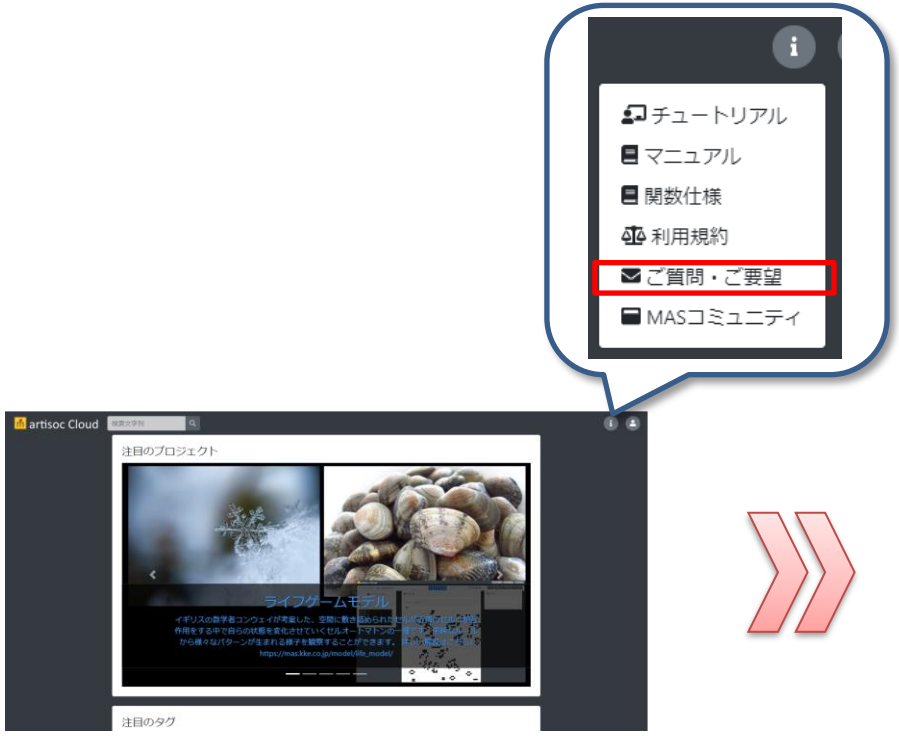
❒参考情報

- 自分でモデルを作成するときは、「チュートリアル」「マニュアル」「関数仕様」を参照してください
  - チュートリアルは現在、この講習と同内容の初級編しか掲載していませんが、徐々に充実させていく予定です



❒参考情報

- ご質問・ご要望がありましたらお気軽にお寄せください
  - まだ試用版という位置づけですので、品質向上のために皆様のご協力をお願いいたします



ご受講ありがとうございました

アンケートにご協力お願いいたします
（チャットでURLを送ります）