

artisoc Cloud初級チュートリアル

■ イン트로ダクション

■ 基礎編

□ I. モデルの作成手順を学ぶ

- ➡ モデル作成の流れ
- ➡ 表示設定

□ II. 基本的なテクニックを学ぶ

- ➡ 変数設定
- ➡ Universe, エージェントのルール
- ➡ コントロールパネル

□ III. 相互作用を含むモデルを作成する

- ➡ 条件分岐
- ➡ エージェント間の相互作用

■ artisoc Cloudというソフトウェアを使います

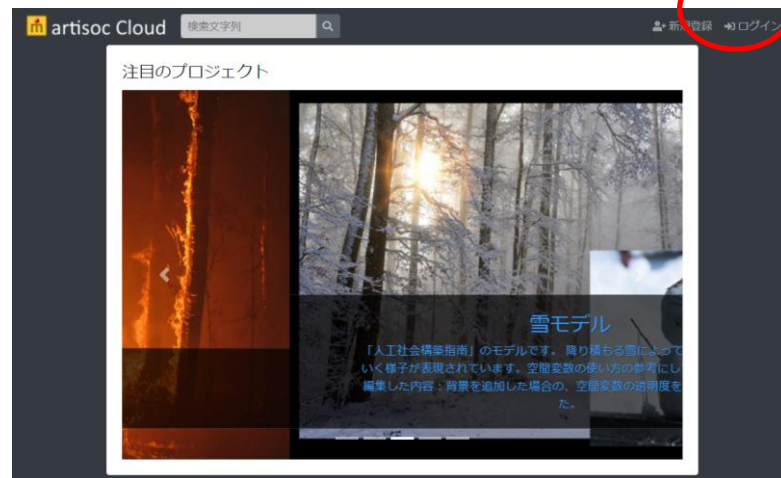
□ <https://artisoc-cloud.kke.co.jp/>

□ 動作環境： Google ChromeもしくはFirefoxブラウザを使ってください

□ 1. 上記URLから「artisoc Cloud」に接続してください。

□ 講習用アカウントでartisoc Cloudにログインしてください。

ログイン



社会は様々な要素が「つながり」、「相互に影響しあう」 複雑なシステムとして成り立っている

社会の 複雑なシステム

多数の主体や
シナリオが絡む
システム

【例：交通システム】

渋滞はどこで起こる？
誰が損をする？
災害時にはどうなる？

創発現象が
存在するシステム

個々の主体の
相互作用によって
起こる複雑な現象

【例：交通渋滞】

個々の自動車の小さな
ブレーキの連鎖が渋滞を
引き起こす

MAS: 個々のエージェントの相互作用によって生じる現象をシミュレーションによって見えるようにして理解する

MAS
【マルチエージェント・シミュレーション】

社会シミュレーションで
読み解く複雑な社会課題

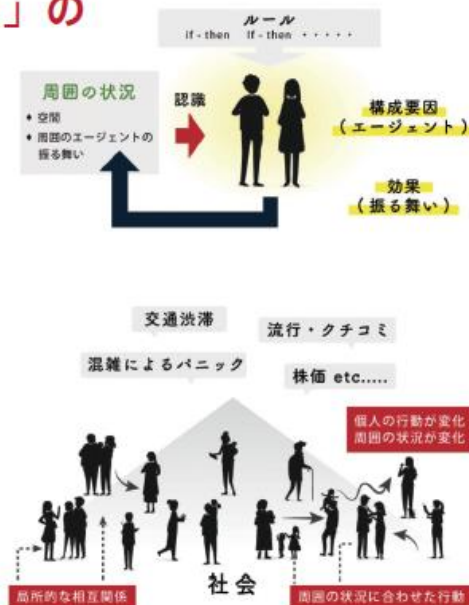
コンピュータのなかに
人工社会を構築

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
社会現象を再現

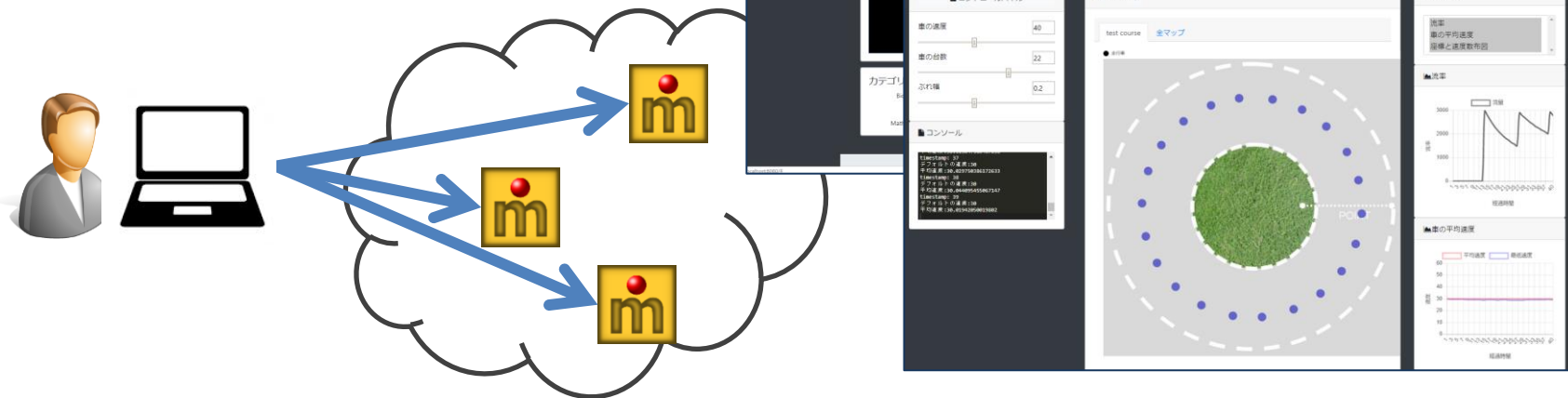
エージェント：自律的に行動する主体



社会シミュレーションのプラットフォーム

- 利用者がWeb上で簡単に社会シミュレーションを使うことができる。その結果を共有・議論することができる
- クラウドのパワーを使ったより高度な解析を簡単に実行できる
- モデルをクラウドでテンプレートとして共有し、派生したモデルを簡単に構築できる

artisoc Cloud



artisocのモデルを動かす

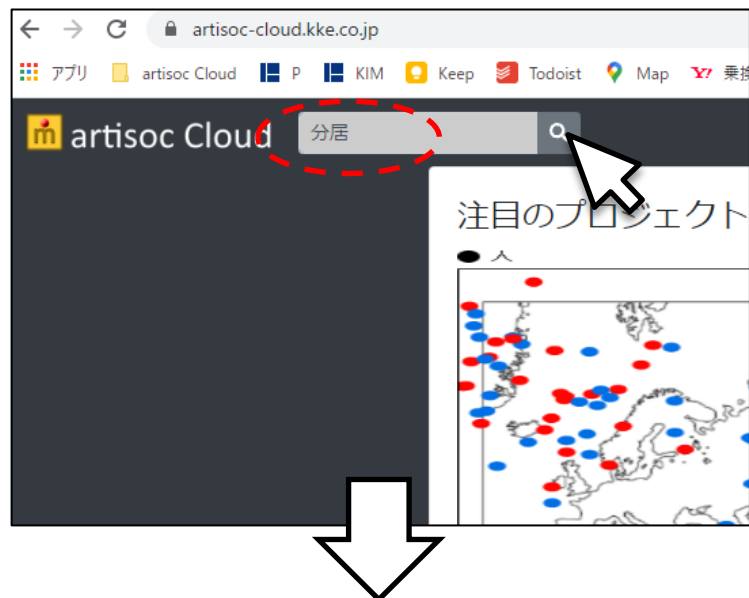
7

- 下記URLにアクセスし、画面右上からログインします

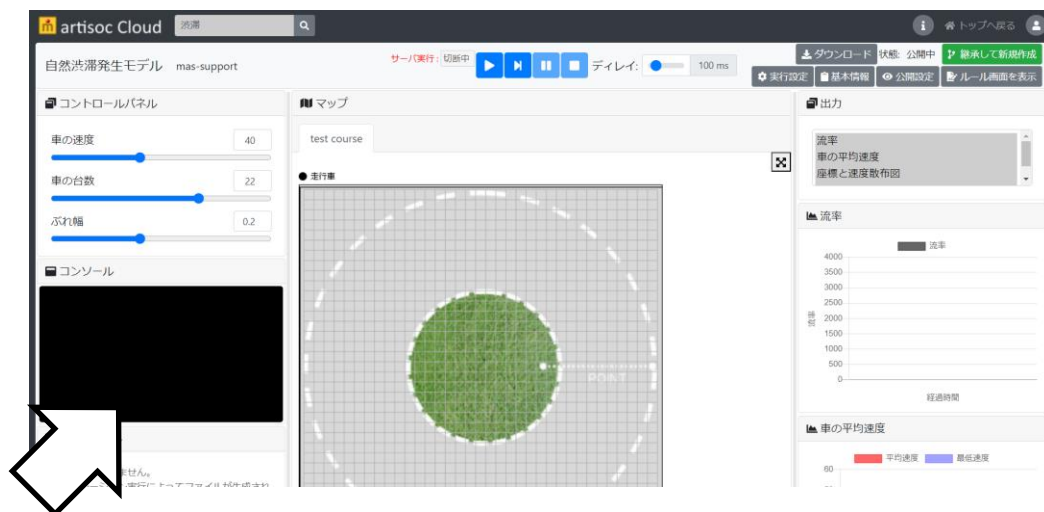
- <https://artisoc-cloud.kke.co.jp/>



- トップページから「渋滞」で検索、「[自然渋滞発生モデル](#)」をクリック



自然渋滞発生モデルが開きます

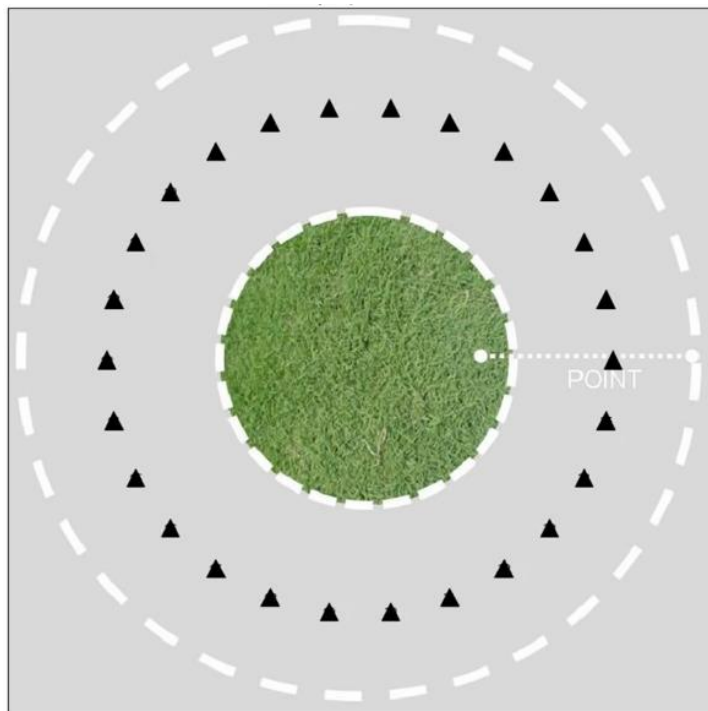


[自然渋滞発生モデル](#)

mas-support 閲覧回数 4 更新日 2021-08-11

高速道路などで発生する自然渋滞を、マルチエージェント・シミュレーションのモデルとして再現したモデルです。詳しい解説はこちら↓

<https://mas.kke.co.jp/model/自然渋滞発生モデル/>



渋滞の写真 (C)ヌンヌン “渋滞”,
<http://www.flickr.com/photos/nunnun/2152291102/>
<http://creativecommons.org/licenses/by/2.0/deed.en>.

ルール

1. 前の車を追従 (ただし速度にはぶれ幅があり)
2. 車間距離が短くなるとブレーキ
3. 車間距離が長くなると加速

シンプルなルールで渋滞の本質を表現

シチュエーションやパラメータは東京大学西成教授の実証実験を参考
参考文献 渋滞学, 西成 活裕著, 新潮選書, 2006

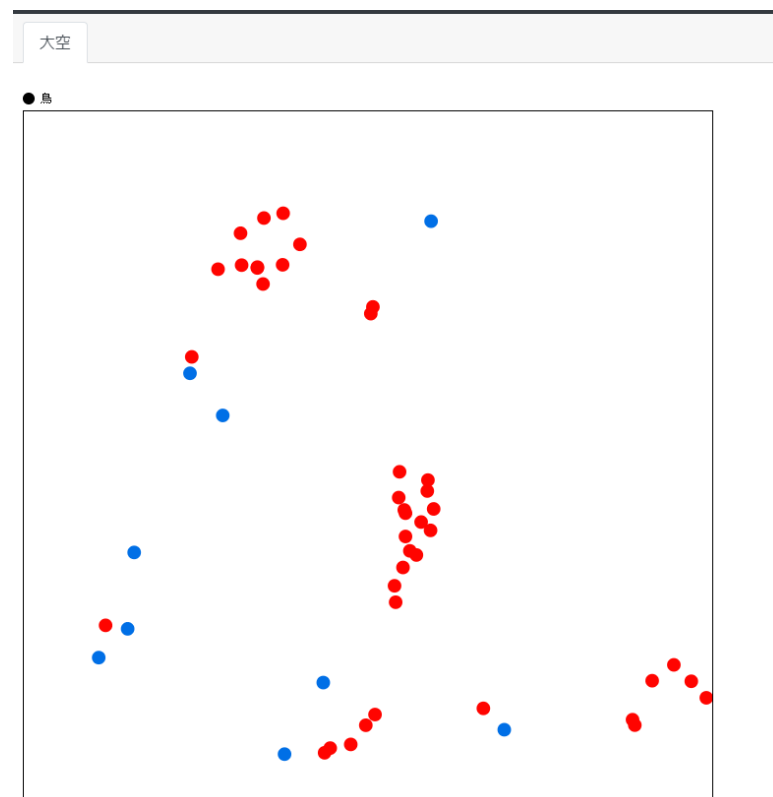
■ 概要

□『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

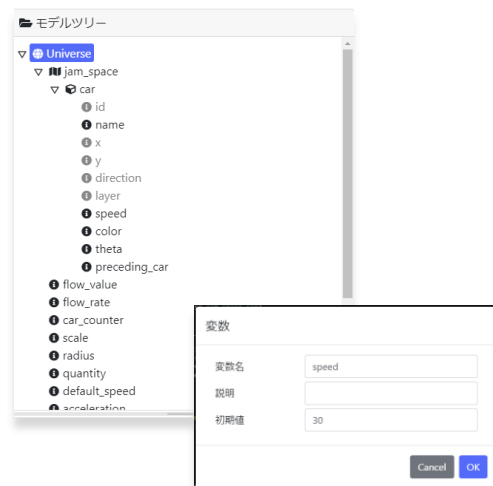


I. モデルの作成手順を学ぶ

- 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

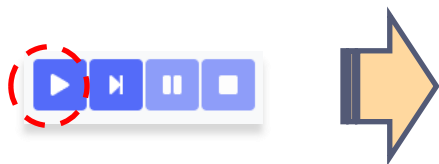
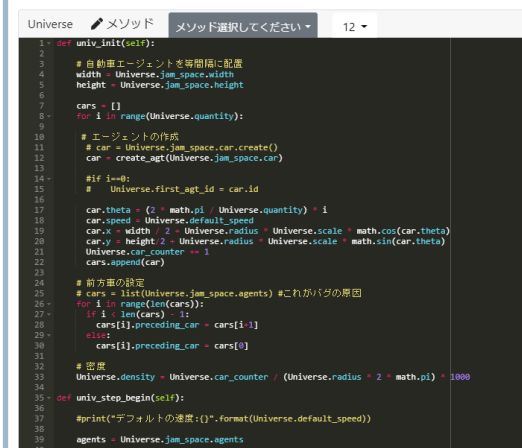
① 空間／エージェントの種類・属性を作成



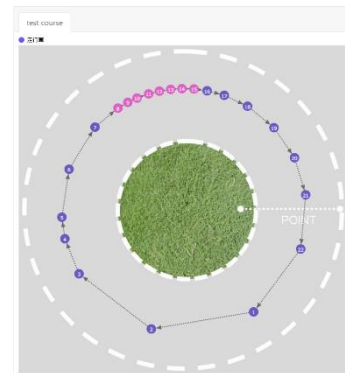
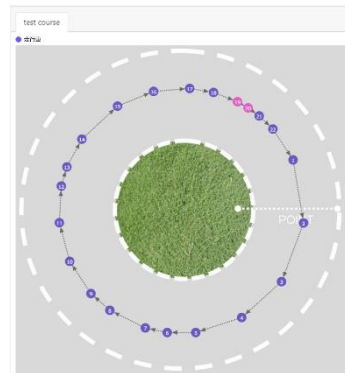
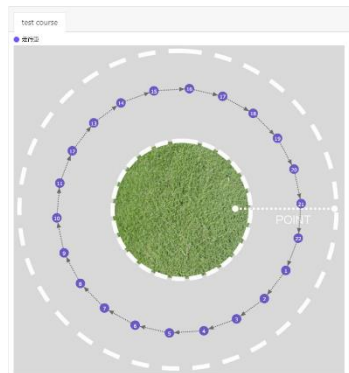
② シミュレーション結果の出力形式を決定



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！



- 下記URLにアクセスし、画面右上からログインします

□ <https://artisoc-cloud.kke.co.jp/>



- 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



モデル基本情報

作成者

mas_admin

モデル名

飛ぶ鳥モデル

概要

説明がありません。

モデルのタグ

新規タグ

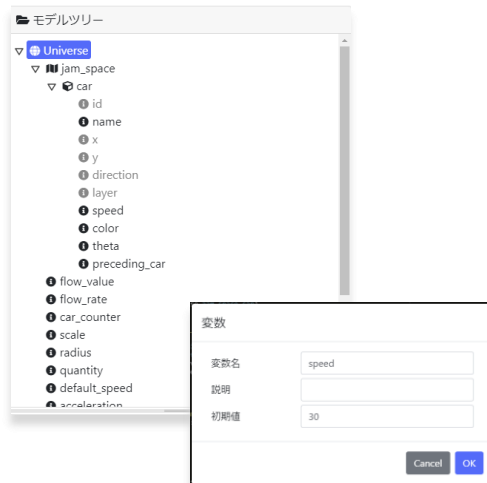
追加

モデルのイメージ画像 :

Cancel

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

```

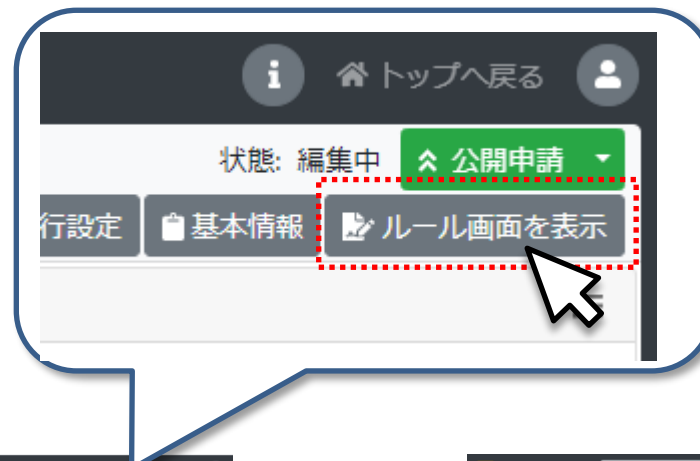
Universe メソッド メソッド選択してください 12
1 def univ_init(self):
2
3
4 # 自動車エージェントをランダムに配置
5 width = Universe.jam_space.width
6 height = Universe.jam_space.height
7
8 cars = []
9 for i in range(Universe.quantity):
10
11 # エージェントの作成
12 # car = Universe.jam_space.car.create()
13 car = create_agent(Universe.jam_space, car)
14
15 # if i==0:
16 # Universe.first_agent_id = car.id
17
18 car.theta = (2 * math.pi / Universe.quantity) * i
19 car.speed = Universe.default_speed
20 car.x = width * 2 * Universe.radius * Universe.scale * math.cos(car.theta)
21 car.y = height * 2 * Universe.radius * Universe.scale * math.sin(car.theta)
22 Universe.car_counter += 1
23 cars.append(car)
24
25 # 前方車の設定
26 # cars = list(Universe.jam_space.agents) #これがバグの原因
27 for i in range(len(cars)):
28 if i != len(cars) - 1:
29 cars[i].preceding_car = cars[i+1]
30 else:
31 cars[i].preceding_car = cars[0]
32
33 # 密度
34 Universe.density = Universe.car_counter / (Universe.radius * 2 * math.pi) * 1000
35
36 def univ_step_begin(self):
37
38 #print("デフォルトの速度: {}".format(Universe.default_speed))
39
40 agents = Universe.jam_space.agents
  
```


■「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

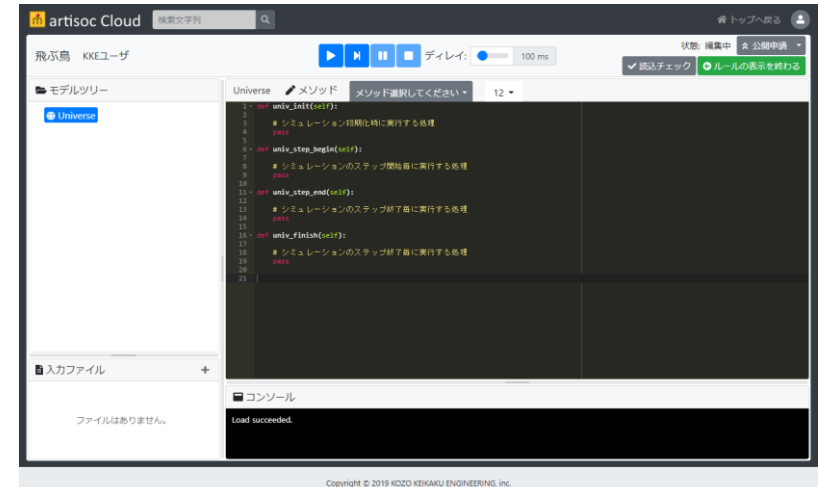
- 最初はモデル全体を表す「Universe」だけがあります
- ここに「空間」「エージェント種別」「変数」などを追加していきます



- 「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



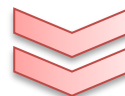
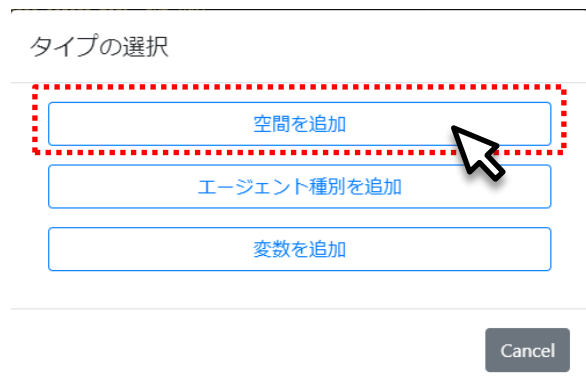
ルール画面

■ 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

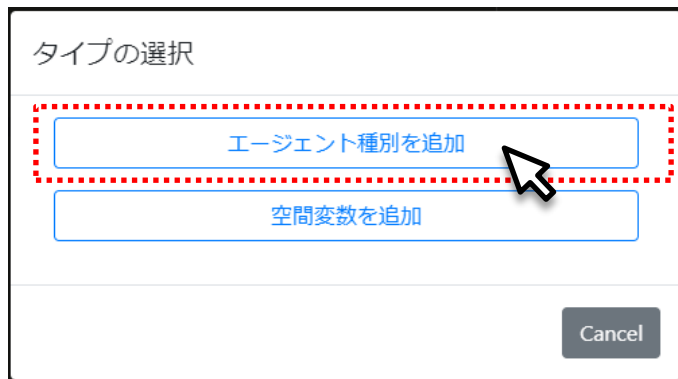


■ 空間の中にエージェント種別を作成

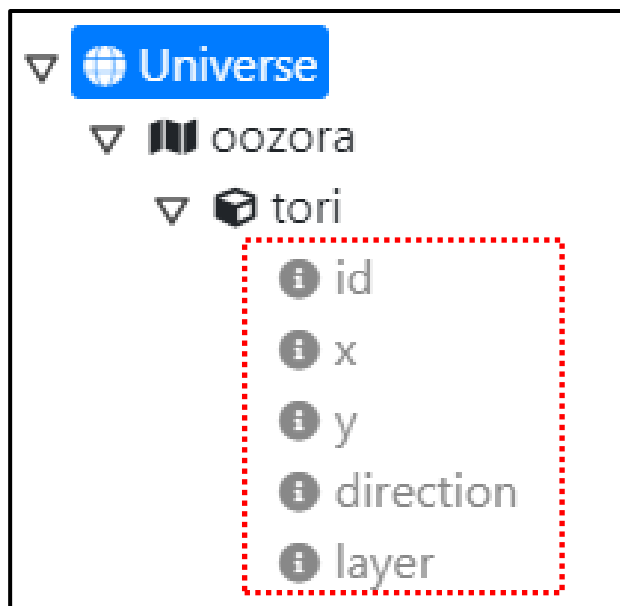
□ エージェント種別名「tori」・・・鳥エージェント

➡「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



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

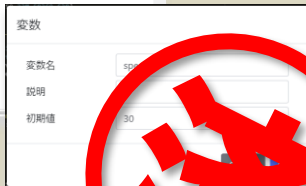
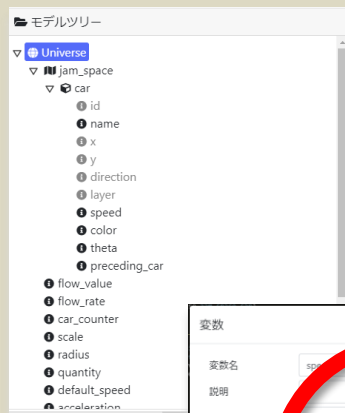
ルールを変更したら
動かなくて、元に戻せな
くなった・・・



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

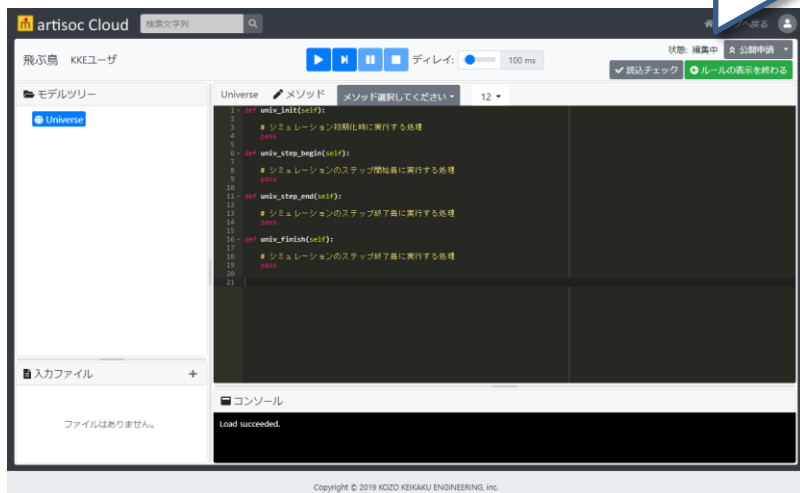


③ エージェントの行動ルールを作成

```

Universe メソッド メソッド選択してください 12
1 def univ_init(self):
2
3
4 # 自動車エージェントをランダムに配置
5 width = Universe.jam_space.width
6 height = Universe.jam_space.height
7
8 cars = []
9 for i in range(Universe.quantity):
10
11 # エージェントの作成
12 # car = Universe.jam_space.car.create()
13 car = create_agent(Universe.jam_space, car)
14
15 # if i==0:
16 #     Universe.first_agent_id = car.id
17
18 car.theta = (2 * math.pi / Universe.quantity) * i
19 car.speed = Universe.default_speed
20 car.x = width * 2 * Universe.radius * Universe.scale * math.cos(car.theta)
21 car.y = height * 2 * Universe.radius * Universe.scale * math.sin(car.theta)
22 Universe.car_counter += 1
23 cars.append(car)
24
25 # 前方車の設定
26 # cars = list(Universe.jam_space.agents) #これがバグの原因
27 for i in range(len(cars)):
28     if i != len(cars) - 1:
29         cars[i].preceding_car = cars[i+1]
30     else:
31         cars[i].preceding_car = cars[0]
32
33 # 密度
34 Universe.density = Universe.car_counter / (Universe.radius * 2 * math.pi) * 1000
35
36 def univ_step_begin(self):
37
38 #print("デフォルトの速度: {}".format(Universe.default_speed))
39
40 agents = Universe.jam_space.agents
  
```

■「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

■ 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



■ 空間とエージェントを「マップ」として出力

マップ出力設定

マップ名: 大空

空間: oozora

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: 50

Y軸設定

最小値: 0

最大値: 50

マップ要素リスト エージェント +

Cancel OK

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



マップ要素設定（エージェント）

要素名: 鳥

エージェント: tori

マーカー

☐ なし

☒ 選択 円

☐ ファイル: 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。

拡大率: 1

エージェント表示色

☒ 固定色 0,0,0

☐ 変数指定

エージェント情報の表示

表示する変数: 指定しない

小数の表示桁数: 0 桁

文字色: 0,0,0

エージェント間に線を引く

対象の変数: 指定しない

線の種類: 実線 (—)

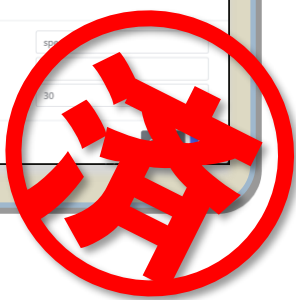
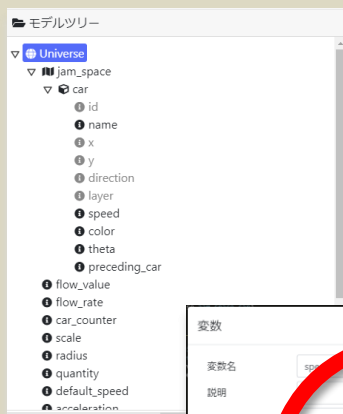
矢印の種類: なし (—)

線の色: 0,0,0

Cancel OK

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

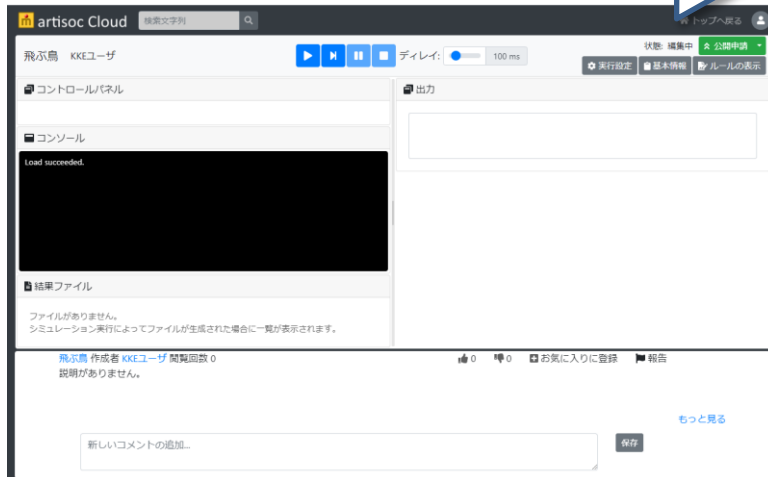
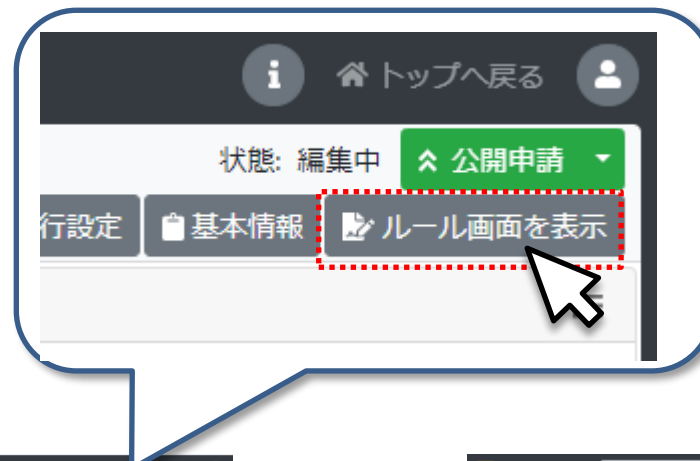
```

Universe メソッド メソッド選択してください 12
1 def univ_init(self):
2
3
4 # 自動車エージェントをランダムに配置
5 width = Universe.jam_space.width
6 height = Universe.jam_space.height
7
8 cars = []
9 for i in range(Universe.quantity):
10
11 # エージェントの作成
12 # car = Universe.jam_space.car.create()
13 car = create_agent(Universe.jam_space, car)
14
15 # if i==0:
16 #     Universe.first_agent_id = car.id
17
18 car.theta = (2 * math.pi / Universe.quantity) * i
19 car.speed = Universe.default_speed
20 car.x = width * 2 * Universe.radius * Universe.scale * math.cos(car.theta)
21 car.y = height * 2 * Universe.radius * Universe.scale * math.sin(car.theta)
22 Universe.car_counter += 1
23 cars.append(car)
24
25 # 前方車の設定
26 # cars = list(Universe.jam_space.agents) #これがバグの原因
27 for i in range(len(cars)):
28     if i != len(cars) - 1:
29         cars[i].preceding_car = cars[i+1]
30     else:
31         cars[i].preceding_car = cars[0]
32
33 # 密度
34 Universe.density = Universe.car_counter / (Universe.radius * 2 * math.pi) * 1000
35
36 def univ_step_begin(self):
37
38 #print("デフォルトの速度: {}".format(Universe.default_speed))
39
40 agents = Universe.jam_space.agents
  
```

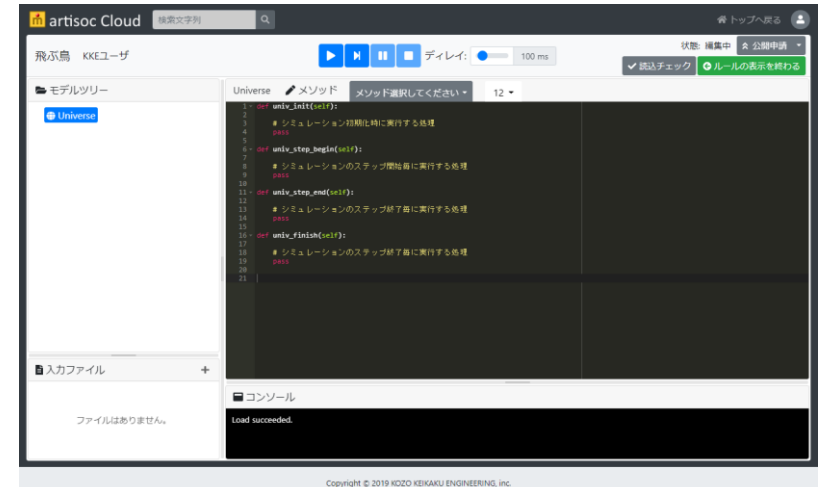
■ 以下のようなモデルのルールを作成します

- 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



- 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



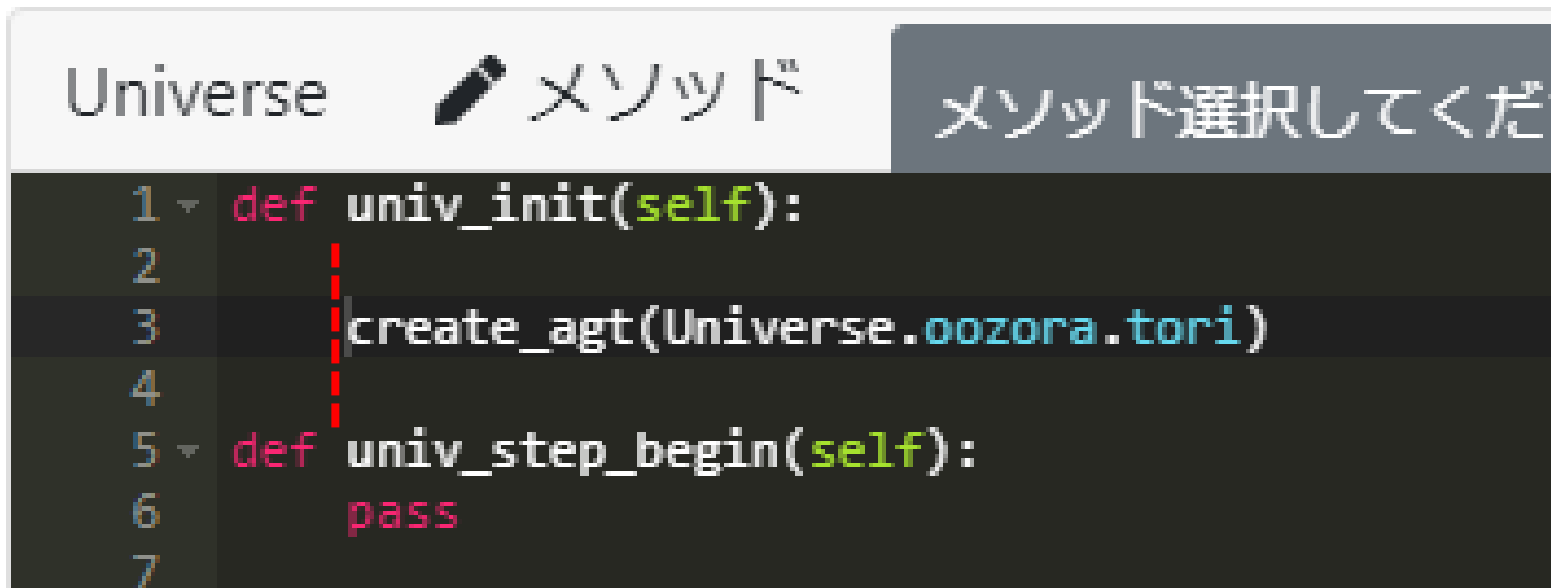
- シミュレーション開始時に鳥エージェントを1体生成
 - univ_initに鳥エージェントを生成するルールを記述します
 - エージェントの生成→create_agt: エージェントを生成する関数
 - ➡ 関数: ルールをひとまとめにしたもの
 - ➡ 「create_agt(Universe.oozora.tori)」でtoriエージェントを生成します
 - 「pass」を消し、下図のように記述します
 - ➡ インデントに注意→次ページ

Universe  メソッド

メソッド選択してください

```
1 def univ_init(self):  
2  
3     create_agt(Universe.oozora.tori)  
4  
5 def univ_step_begin(self):  
6     pass  
7
```

- 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

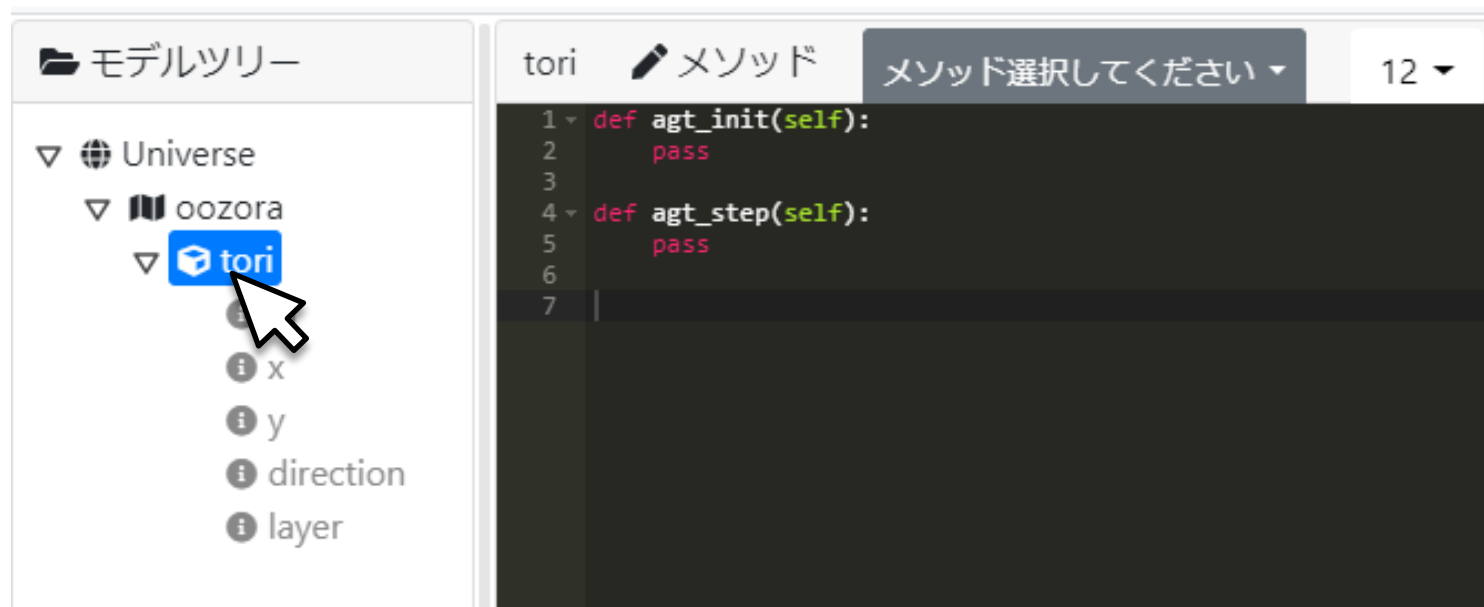


The screenshot shows a code editor interface. At the top, there are tabs labeled 'Universe' and 'メソッド' (Methods). A button labeled 'メソッド選択してください' (Please select a method) is visible. The code is written in Python and demonstrates indentation using tabs. A vertical dashed red line indicates the current indentation level.

```
1 ▾ def univ_init(self):  
2       
3     create_agt(Universe.oozora.tori)  
4       
5 ▾ def univ_step_begin(self):  
6     pass  
7
```


■ 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



■ 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

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 def agt_step(self):  
7     pass  
8  
9
```

モデルツリー

- Universe
 - oozora
 - tori
 - id
 - x
 - y
 - direction
 - layer

■ 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

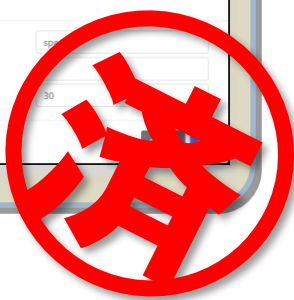
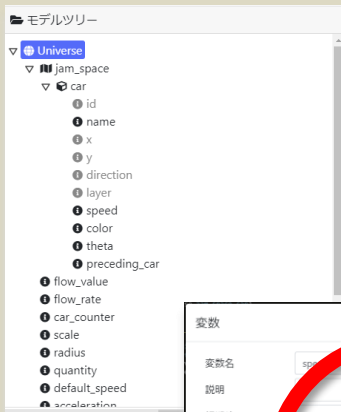
□ forward: 前に進む関数

➡ 「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
```

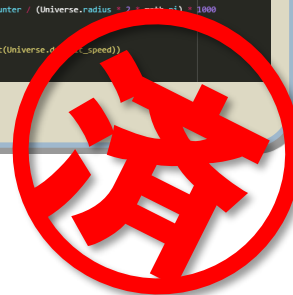
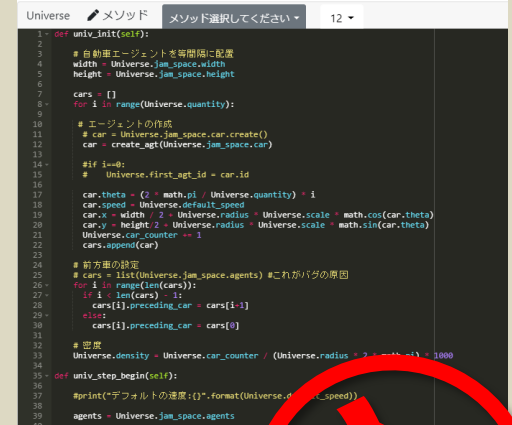
① 空間／エージェントの種類・属性を作成

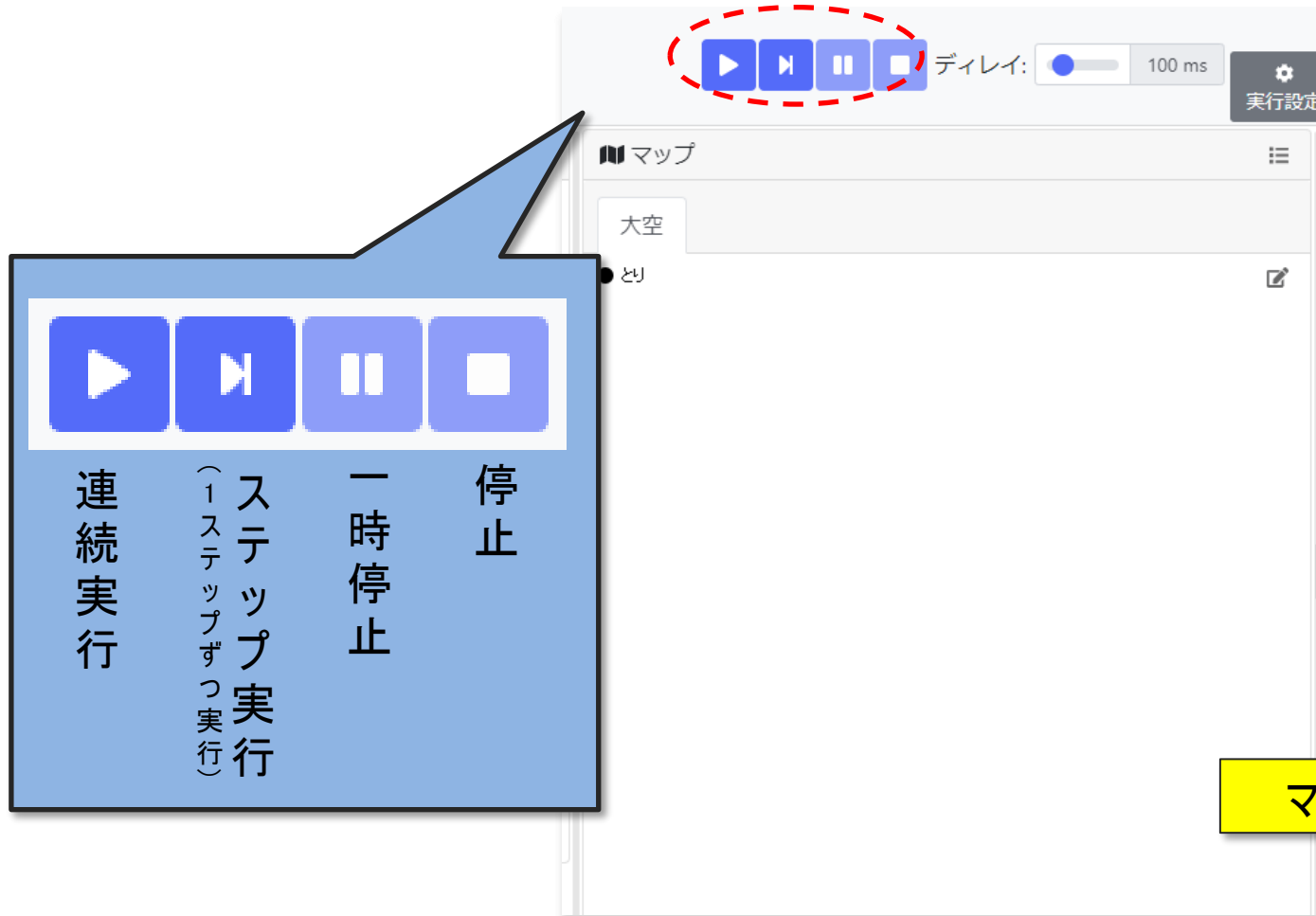


② シミュレーション結果の出力形式を決定



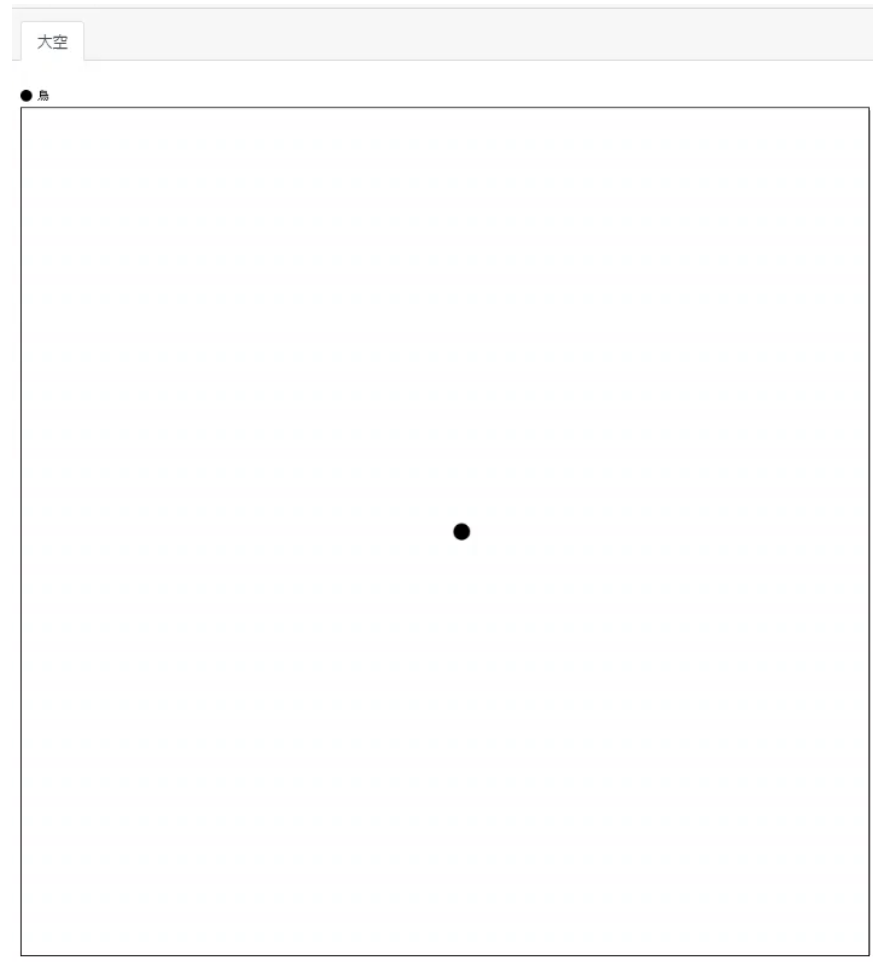
③ エージェントの行動ルールを作成





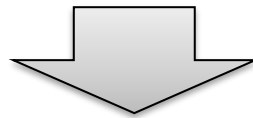
ルールの編集、出力の設定を行うときは「**停止**」状態である必要があります

- 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・・・

スライダーで実行速度を調整する

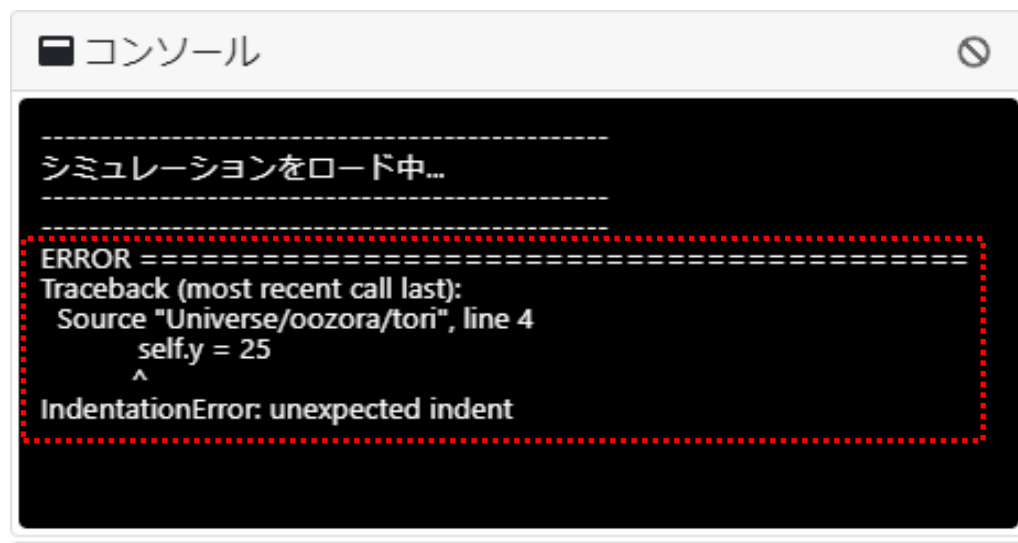


右に動かすと実行速度が遅くなり、左に動かすと速くなる

- うまく動かなければ、「読込チェック」をクリックしてください
 - ソール画面にエラーメッセージが表示されます



- エラーメッセージの意味が分からないときは、Q&Aで質問してください
 - エラーメッセージをQ&Aにコピーしてお知らせください



■ 空間がループ = 上端-下端・左端-右端が繋がった空間

空間

空間名

oozora

空間種別

連続空間

説明

大空を表す空間

記憶数

0

空間の大きさ X

50

空間の大きさ Y

50

レイヤ数

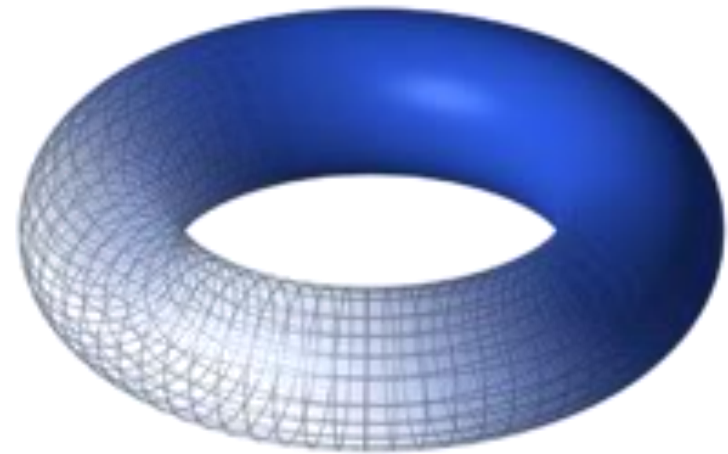
1

ループする

☒

Cancel

OK



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

- 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://artisoc-cloud.kke.co.jp/models/GyXulUvDRbG-mZA4hkwF0g>
 - ➡ 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



- 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

The screenshot displays the 'tori' rule editor. The top bar shows 'tori', a pencil icon, 'メソッド', a dropdown menu with 'メソッド選択してください', and a page number '12'. The main area contains a code editor with the following Python code:

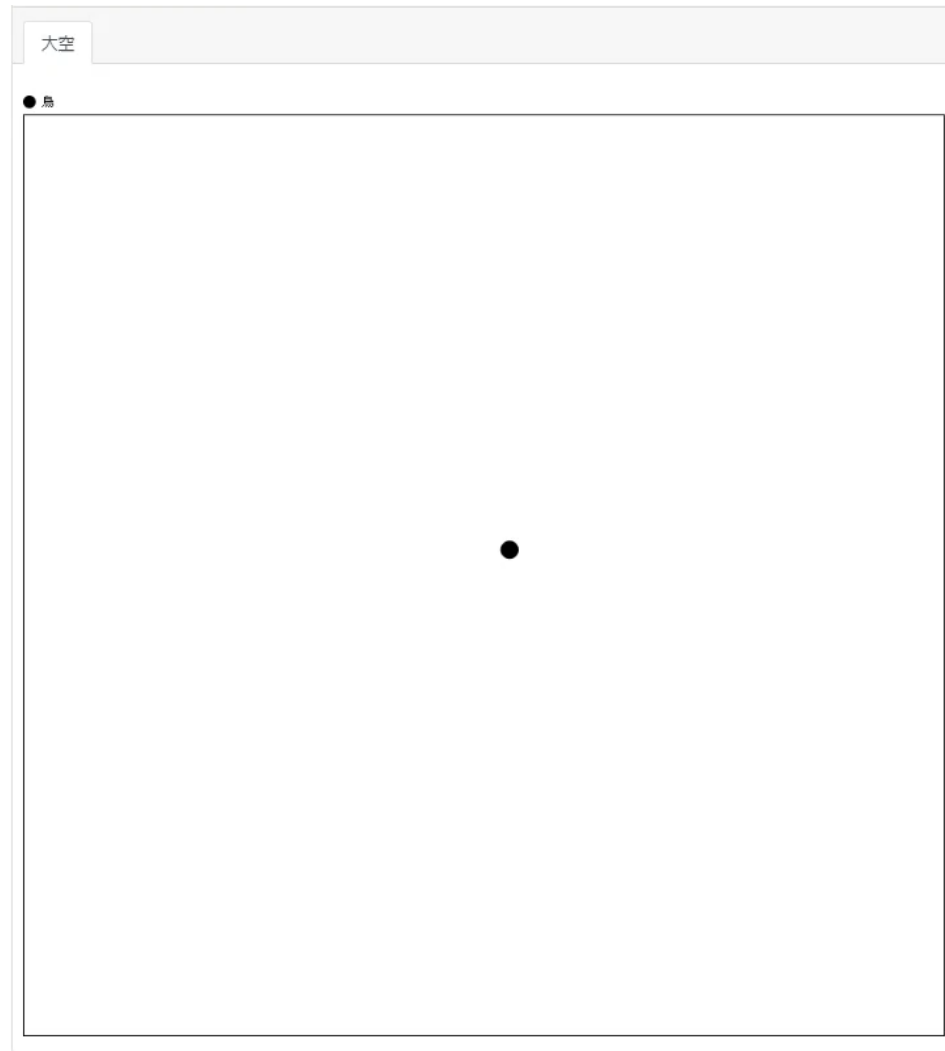
```
1 def agt_init(self):
2
3     self.x = 25
4     self.y = 25
5
6     self.direction = 45
7
8 def agt_step(self):
9
10    self.forward(1)
11
12
```

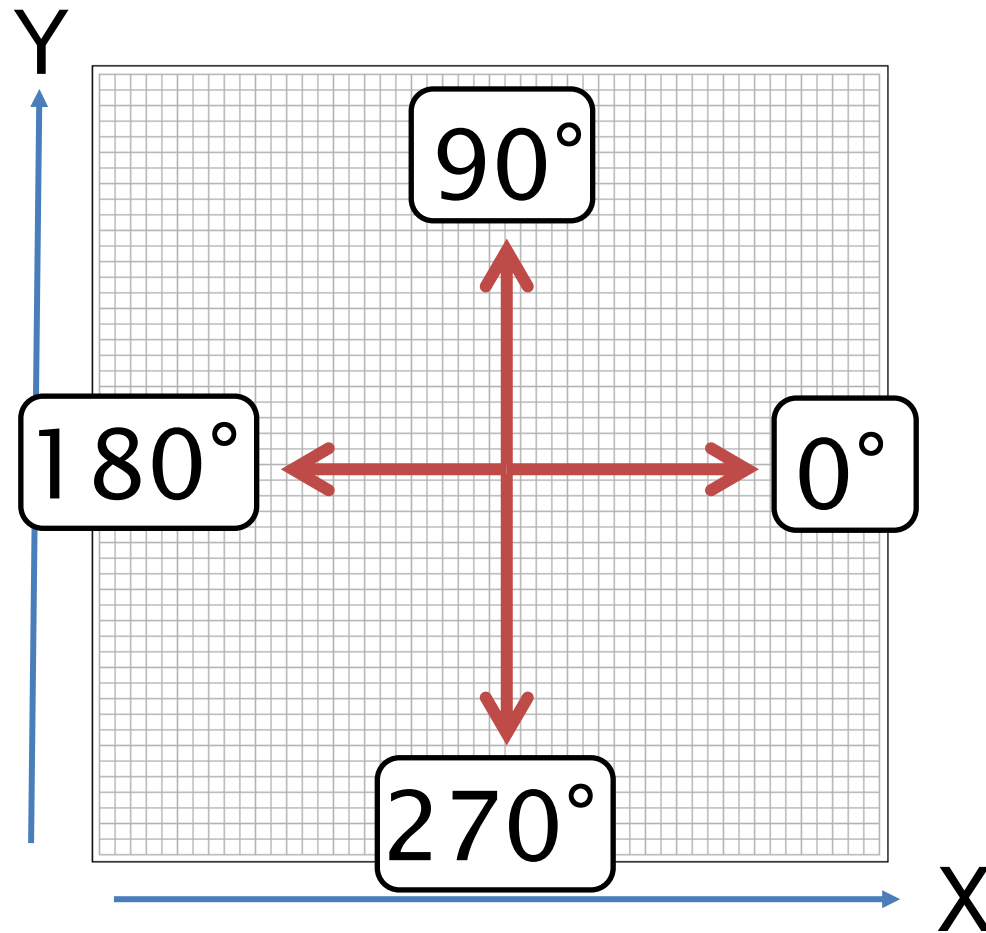
A callout box on the right shows the 'モデルツリー' (Model Tree) structure:

- モデルツリー
 - Universe
 - oozora
 - tori
 - id
 - x
 - y
 - direction
 - layer

The 'direction' property under the 'tori' object is highlighted with a red dashed box. A red dotted line also highlights the assignment 'self.direction = 45' in the code editor, with a blue arrow pointing from the callout box to it.

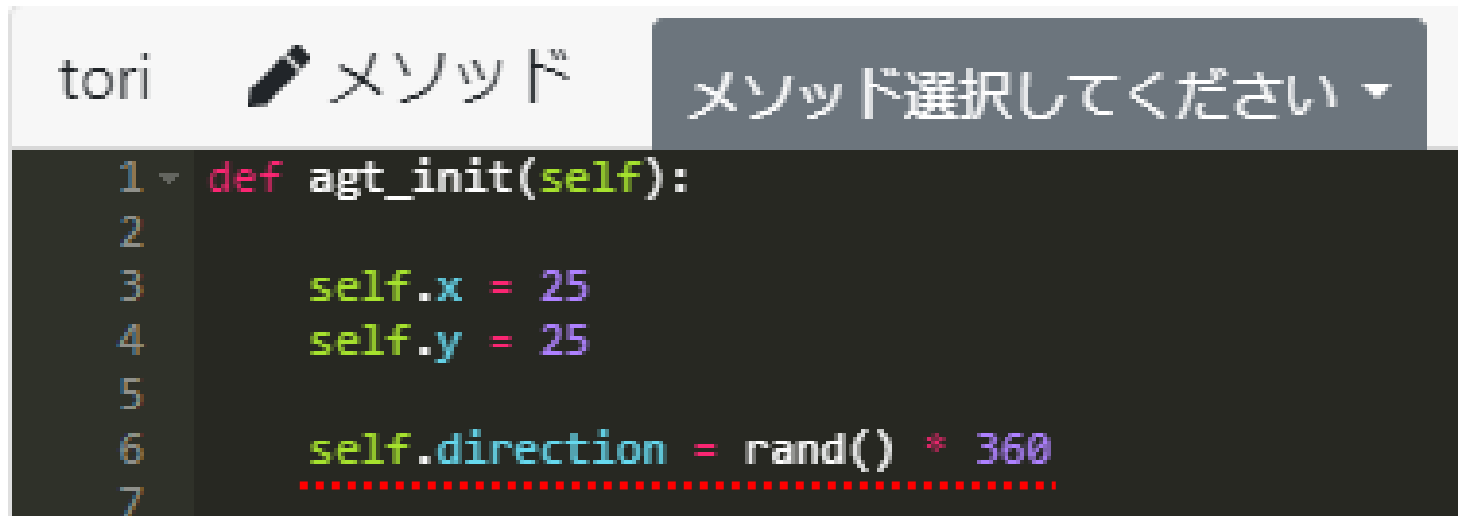
■ 鳥が右上に飛んでいきます





※左下原点の場合

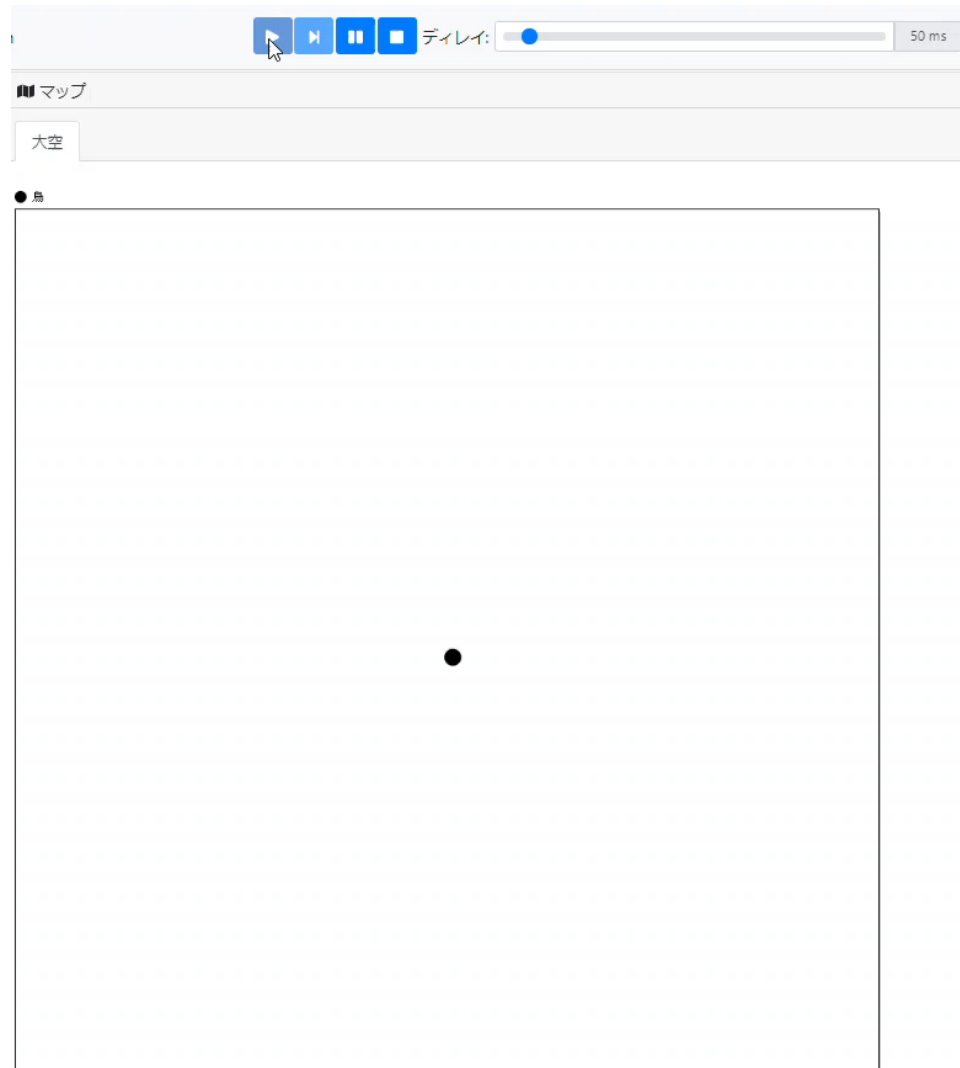
- 属性「direction」に進む方向の値をランダムに設定(0~360度)して実行します
 - rand: 0~1のランダムな値(一様乱数)を取得する関数
 - 「rand() * 360」で0~360のランダムな値を意味します
 - ➡ 「*」は掛け算の記号です
 - ➡ 関数には必ず()がつきます



The screenshot shows a code editor with a dark background. At the top, there is a tab labeled 'tori' and a button with a pencil icon labeled 'メソッド'. To the right of the tab is a dropdown menu labeled 'メソッド選択してください ▾'. Below the tab, the following Python code is displayed:

```
1 def agt_init(self):
2
3     self.x = 25
4     self.y = 25
5
6     self.direction = rand() * 360
7     .....
```

- 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：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
- 返回值（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
 - ※ 返回值がない（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
- 返回值：正常終了時は-1（いまは使っていません）

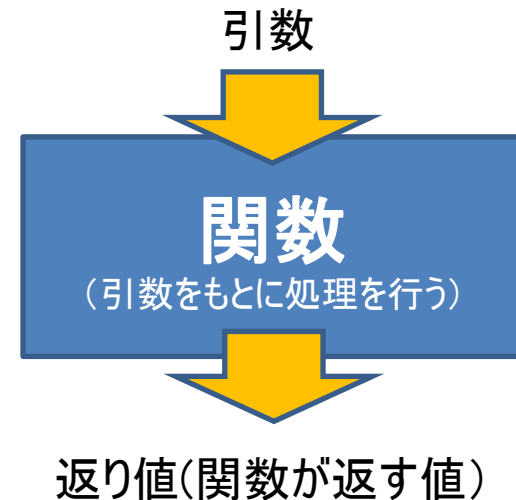
```
self.forward(10) # 引数の数10だけ前に進む
```

■ 関数の例：rand

- 引数：なし
- 処理：0~1の一樣乱数を発生させる
- 返回值：0~1の一樣乱数

```
r = rand() # rand関数の返回值（0~1の一樣乱数）を変数rに代入する
```

※ 引数がないときも()が必要



■ 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

The screenshot displays the software interface for Kozo Keikaku Engineering Inc. On the left, a 'モデルツリー' (Model Tree) panel shows a hierarchy: 'Universe' (selected with a mouse cursor), 'oozora', and 'tori'. On the right, the 'メソッド' (Method) panel shows the Python code for the 'Universe' class. The code defines three methods: 'univ_init(self)', 'univ_step_begin(self)', and 'univ_step_end(self)'. The 'univ_init' method contains the line 'create_agt(Universe.oozora.tori, num=100)', which is highlighted with a red dotted line. A dropdown menu above the code area shows 'メソッド選択してください' (Please select a method).

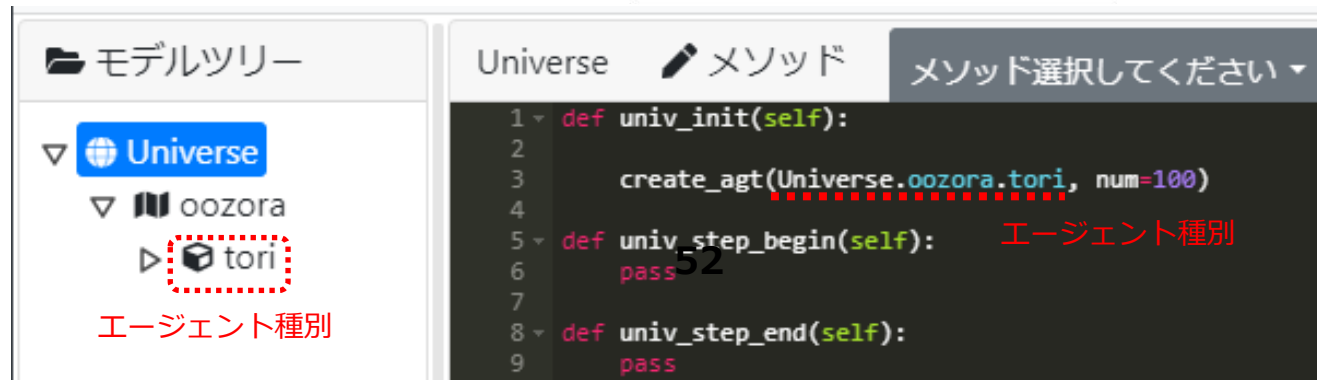
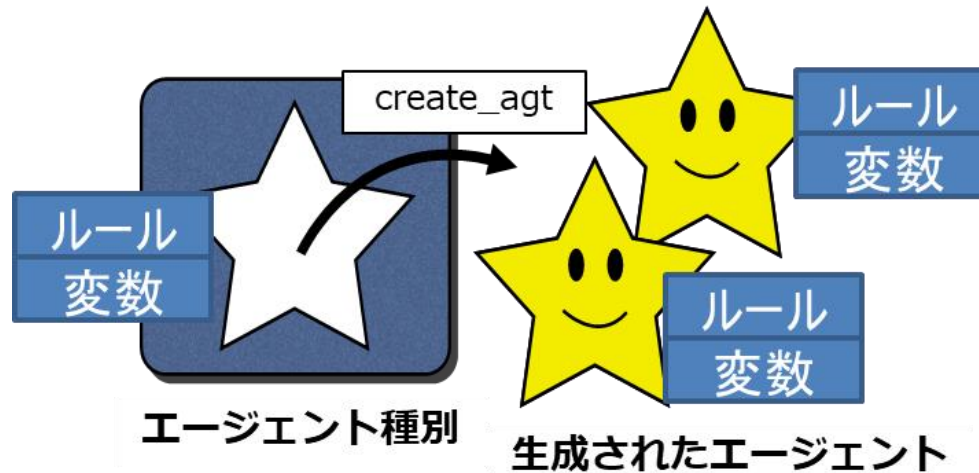
```
1 def univ_init(self):
2
3     create_agt(Universe.oozora.tori, num=100)
4
5 def univ_step_begin(self):
6     pass
7
8 def univ_step_end(self):
9     pass
```

- 100羽の鳥が円形に飛んでいます



□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



■ 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

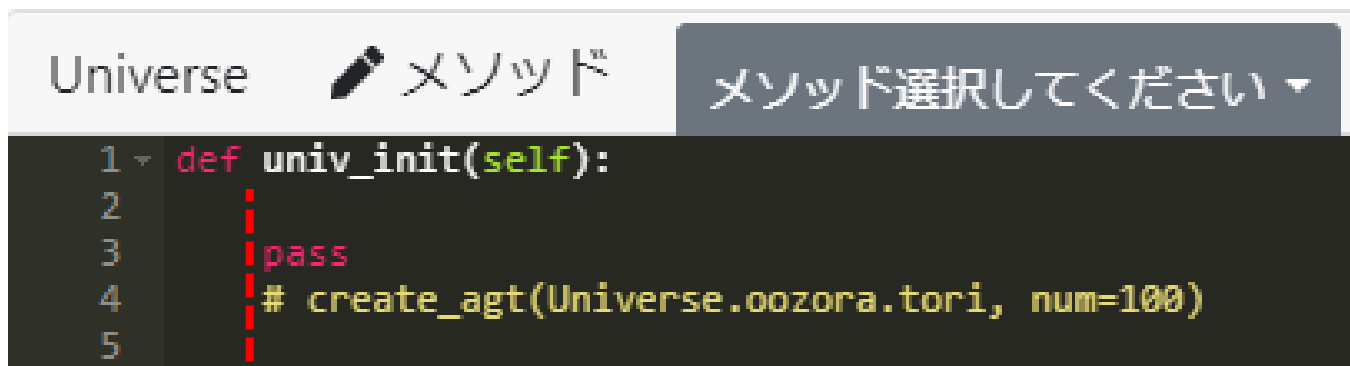
■ passについて

- 「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



The screenshot shows the Artisoc editor interface. At the top, there is a header bar with the text "Universe" on the left, a pencil icon and the word "メソッド" (Method) in the center, and a dropdown menu on the right labeled "メソッド選択してください" (Please select a method). Below the header, the editor area displays a Python code snippet. The code is as follows:

```
1 def univ_init(self):  
2     :  
3     pass  
4     # create_agt(Universe.oozora.tori, num=100)  
5
```

■ 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

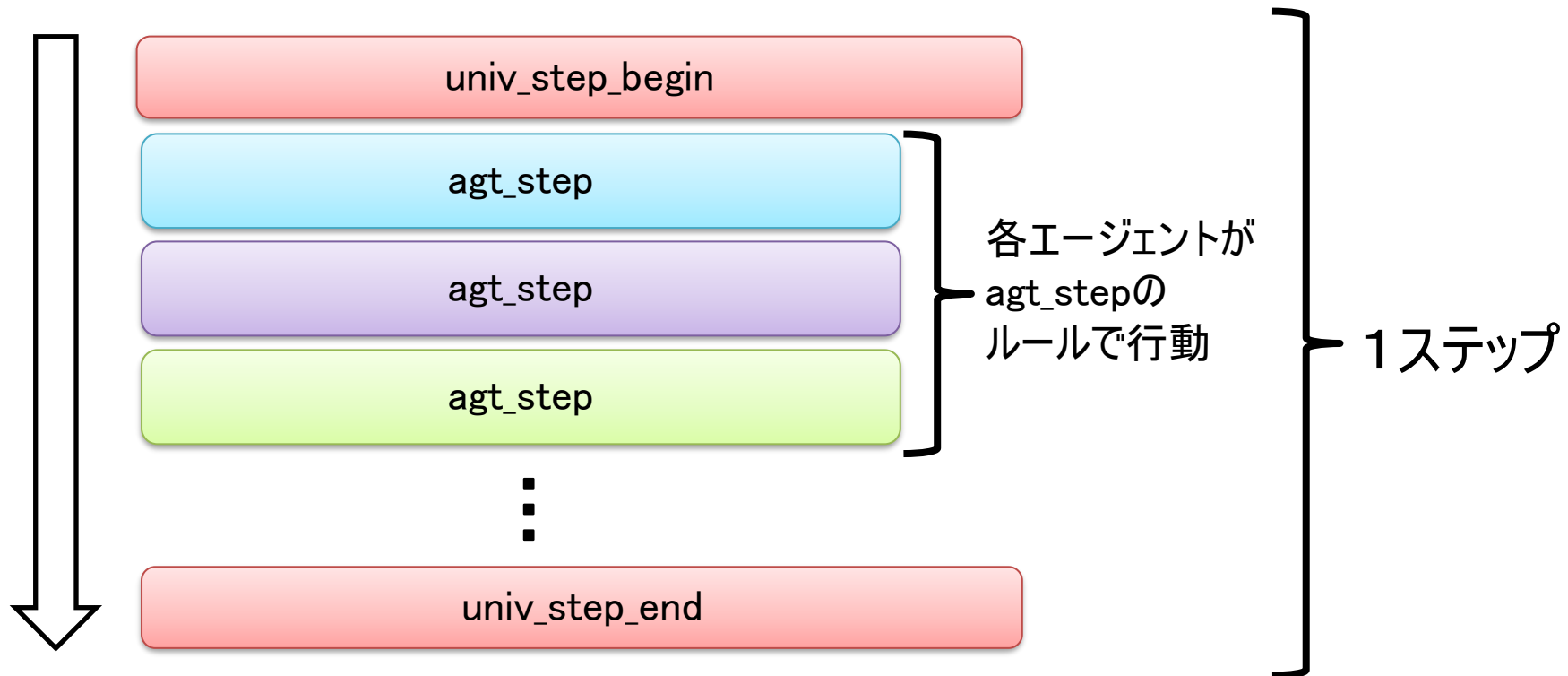
■ Universeのルールエディタ→モデル全体のルール

- 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

- 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

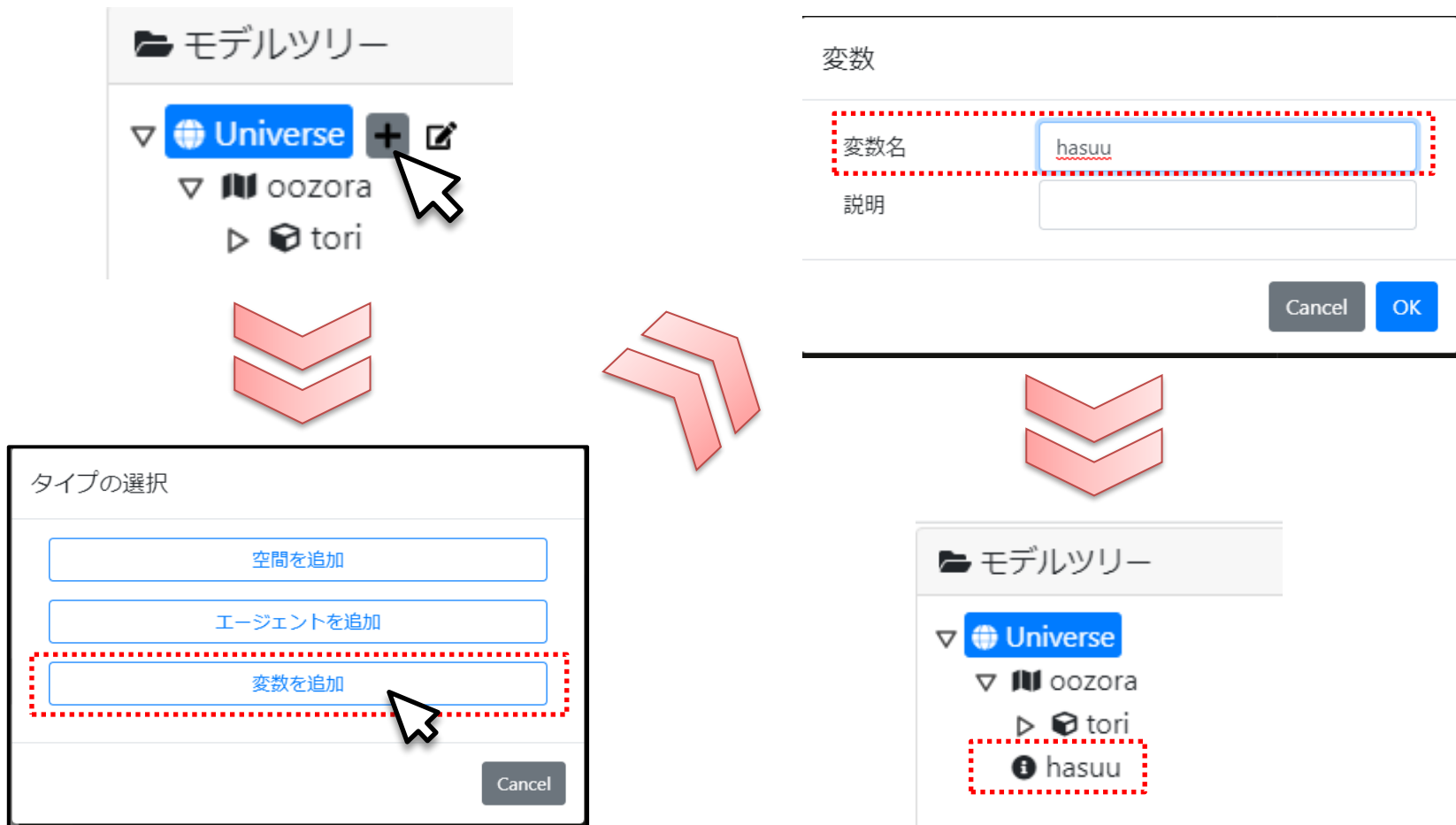
■ 1ステップの流れ



- 鳥の数をスライダーの操作で設定できるようにする



① 鳥の数を表す変数としてUniverseの下に変数「hasuu」を作成



② 変数「hasuu」を鳥の数に設定

Universe



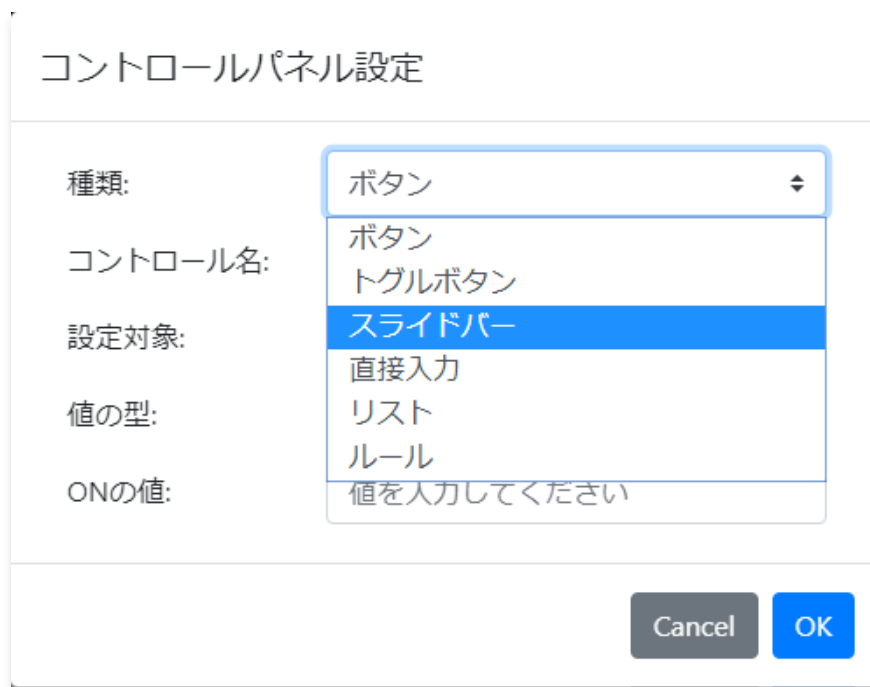
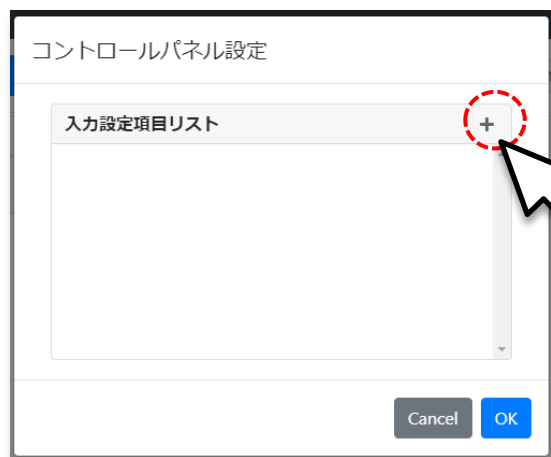
メソッド

メソッド選択してください ▼

12 ▼

```
1 def univ_init(self):  
2     create_agt(Universe.oozora.tori, num=Universe.hasuu)  
3
```

- 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「スライダー」を選択



■ 鳥の数をスライダ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設定

種類:	スライダー
コントロール名:	鳥の数
設定対象:	hasuu
値の型:	整数 (integer)
初期値:	100
範囲:	10 ~ 300
目盛り間隔:	1

Cancel 

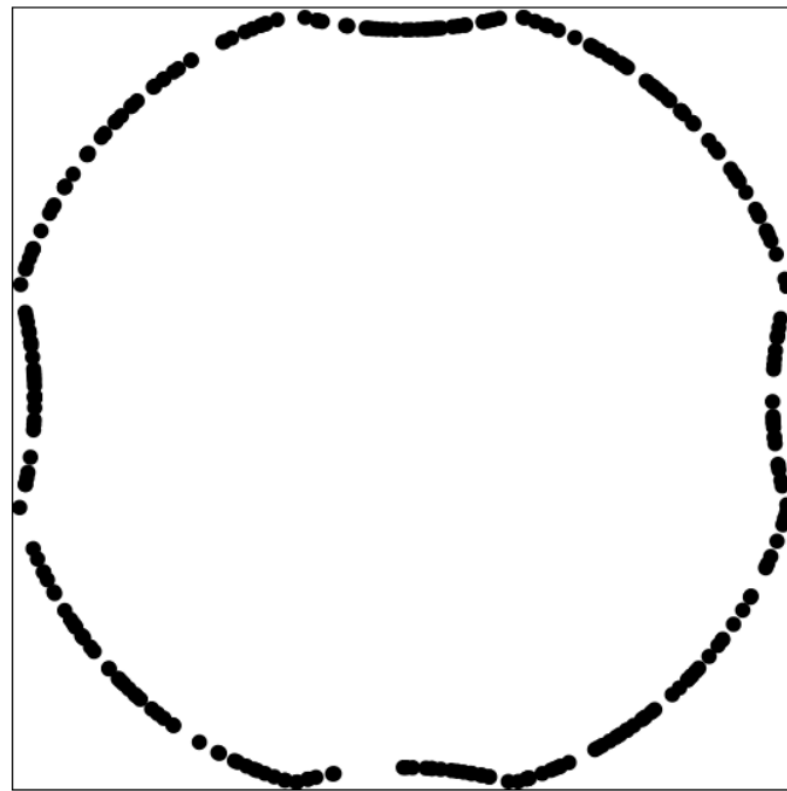
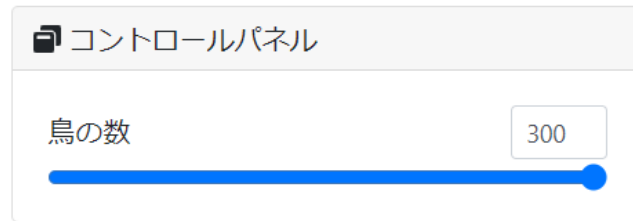
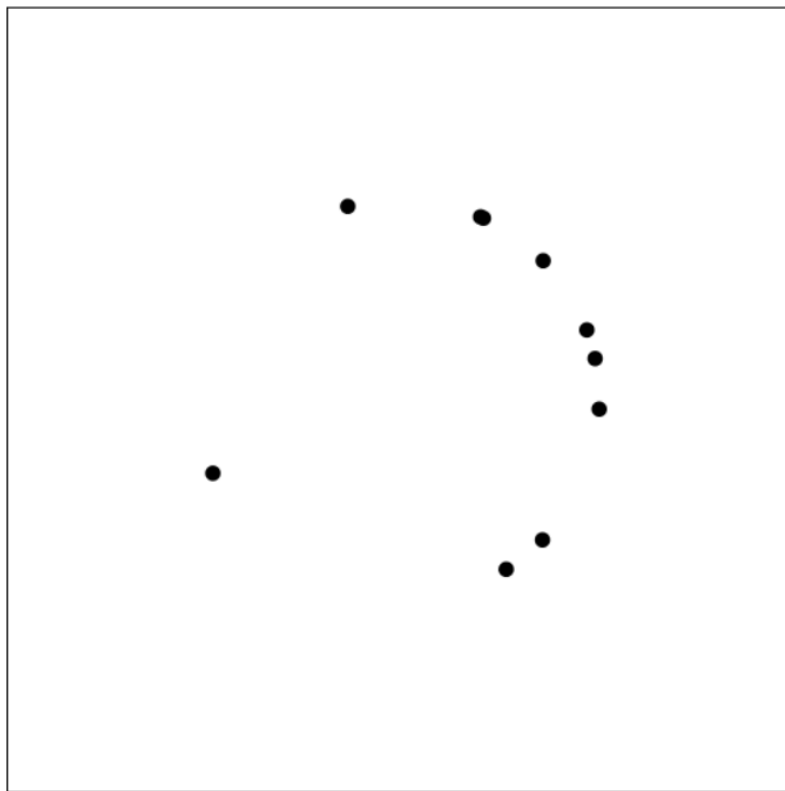
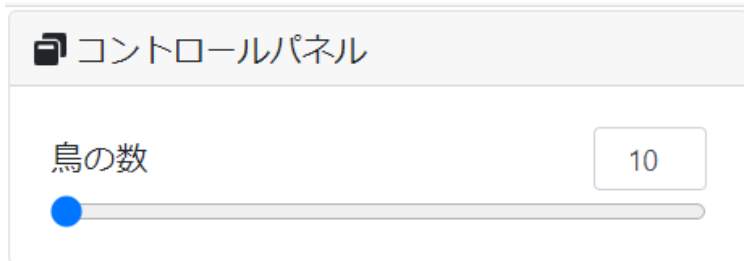


コントロールパネル

鳥の数

100

■ コントロールパネルで鳥の数を設定できます



III. 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

○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 Reynolds, 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



- 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://artisoc-cloud.kke.co.jp/models/oOf0rCNOTU-38dfhIYHfzg>
 - ➡ 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

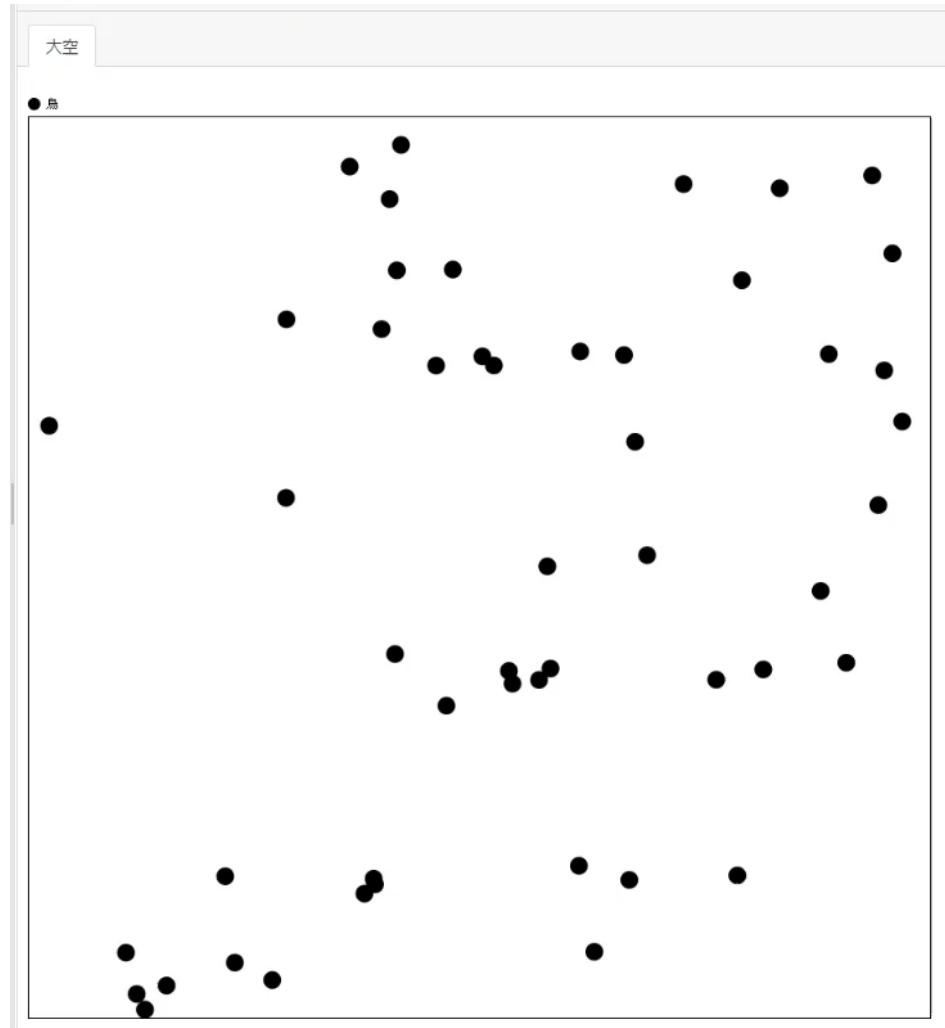
- 鳥のルールを以下のように修正します

- 初期位置をランダムに

→ 「`rand() * 50`」で0~50のランダムな値

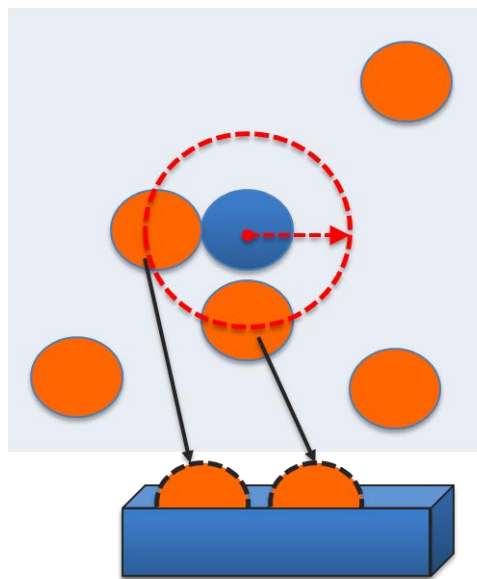
```
1 def agt_init(self):  
2  
3     # 初期位置をランダムで設定  
4     self.x = rand() * 50  
5     self.y = rand() * 50  
6  
7     self.direction = rand() * 360  
8  
9 def agt_step(self):  
10     self.forward(1)  
11  
12
```

■ 鳥がランダムに飛んでいます



■ 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)
```

- 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

```
9 def agt_step(self):
10
11     # 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）
12     tori_set = self.make_agtset_around_own(2, False)
13
14     if count_agtset(tori_set) > 0: # もし周囲に鳥がいれば
15         one = randchoice(tori_set) # ランダムに一羽を選ぶ
16         self.direction = one.direction # 自分の向きをその鳥に合わせる
17
18
19     self.forward(1)
20
```

※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。

- 「条件分岐文」という重要な手法について学びます
 - 条件分岐分: もし ~ ならば ... という処理
 - もし count_agtset(tori_set) の結果が 0 を超えていれば...

条件文が真の場合に実行する処理

if 条件文:
処理

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)

※処理はインデント内に記述

```
9 def agt_step(self):
10
11     # 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)
12     tori_set = self.make_agtset_around_own(2, False)
13
14     if count_agtset(tori_set) > 0: # もし周囲に鳥がいれば
15         one = randchoice(tori_set) # ランダムに一羽を選ぶ
16         self.direction = one.direction # 自分の向きをその鳥に合わせる
17
18     self.forward(1)
19
20
```

■ 鳥が群れを作ります



- 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://artisoc-cloud.kke.co.jp/models/GM0tA8hfTwKvcfp9M3TWJQ>
 - ➡ 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



おわりに



- 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



- 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



The image illustrates the navigation structure of the artisoc Cloud documentation. A speech bubble highlights the 'チュートリアル' (Tutorial), 'マニュアル' (Manual), and '関数仕様' (Function Specification) items in the left sidebar. Below this, a screenshot of the 'artisoc Cloud' interface shows a project titled 'ライフゲームモデル' (Life Game Model). To the right, a large red double arrow points to a detailed view of the 'artisoc Cloud ドキュメント' (artisoc Cloud Documents) page, which lists the contents and indices of the documentation.

artisoc Cloud ドキュメント

Search docs

CONTENTS:

- artisoc Cloudチュートリアル
- artisoc Cloudマニュアル
- artisoc Cloud関数仕様

artisoc Cloud ドキュメント

Docs » artisoc Cloudドキュメント

Contents:

- artisoc Cloudチュートリアル
- artisoc Cloudマニュアル
 - 1. artisoc Cloud 入門
 - 2. リファレンス
 - 3. ルール文法
- artisoc Cloud関数仕様
 - 数値計算
 - 文字列操作
 - データ型変換
 - エージェント操作 (生成・削除)
 - エージェント操作 (その他)
 - 空間エージェント関数 (移動)
 - 空間エージェント関数 (空間操作)
 - 空間エージェント関数 (エージェント集合生成)
 - エージェント集合操作 (基本操作)
 - エージェント集合操作 (集合演算)
 - エージェント集合操作 (生成)
 - エージェント集合操作 (配置)
 - エージェント集合操作 (その他)
 - 空間操作
 - ファイル入出力
 - その他

Indices and tables

- 検索ページ

■ 最新情報はMASコミュニティのほか、SNSでも発信します

MASコミュニティ
https://mas.kke.co.jp



 @artisoc_



 @kkeartisoc



ご受講ありがとうございました