

artisoc Cloud初級チュートリアル

(株) 構造計画研究所
創造工学部

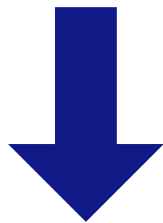
□本日のチュートリアル

- 0. イントロダクション
- 1. モデルの作成手順を学ぶ
 - 空間・エージェントを作成
 - 出力形式を決定
 - エージェントの行動ルールを作成
- 2. 基本的なテクニックを学ぶ
 - 変数設定の変更
 - 関数の使用
 - エージェントの生成の仕方の変更
 - コントロールパネルの設定
- 3. 相互作用を含むモデルを作成する
 - エージェント集合型の変数の活用
 - 条件分岐文の活用
 - エージェント間の相互作用
- おわりに

0. イントロダクション

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

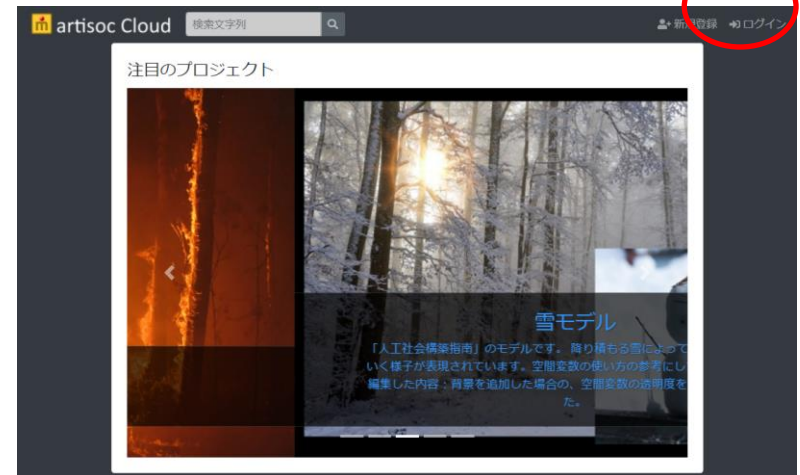


コンピューターの中に人工社会を構築し、社会現象のメカニズムを理解する

□受講に向けた準備

- artisoc Cloudというソフトウェアを使います。
 - <https://artisoc-cloud.kke.co.jp/> から「artisoc Cloud」に接続してください。
 - 動作環境は、Google ChromeもしくはFirefoxブラウザを使ってください。

ログイン

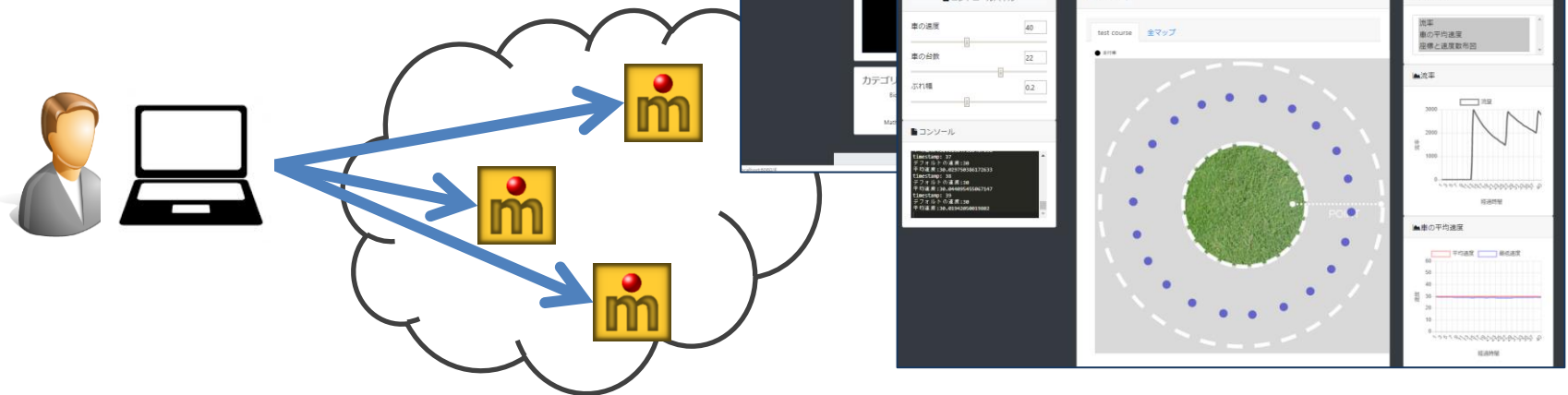


□MASプラットフォーム artisoc Cloud

■ 社会シミュレーションのプラットフォーム

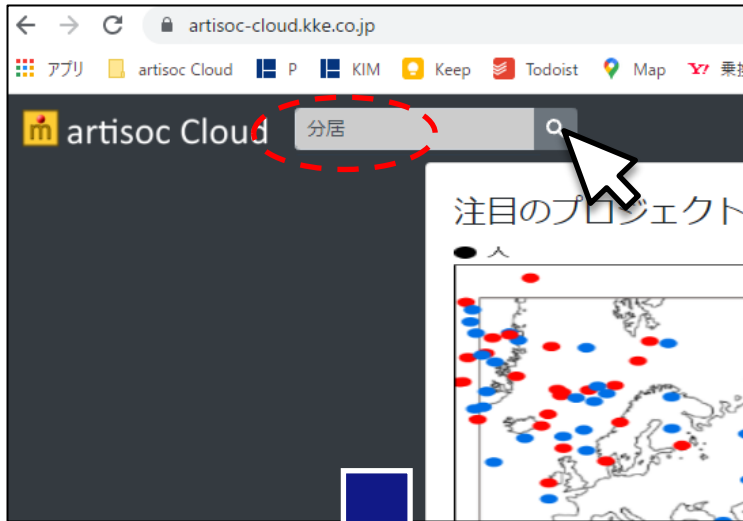
- 利用者がWeb上で簡単に社会シミュレーションを使うことができる。その結果を共有・議論することができる。
- クラウドのパワーを使ったより高度な解析を簡単に実行できる。
- モデルをクラウドでテンプレートとして共有し、派生したモデルを簡単に構築できる。

artisoc Cloud

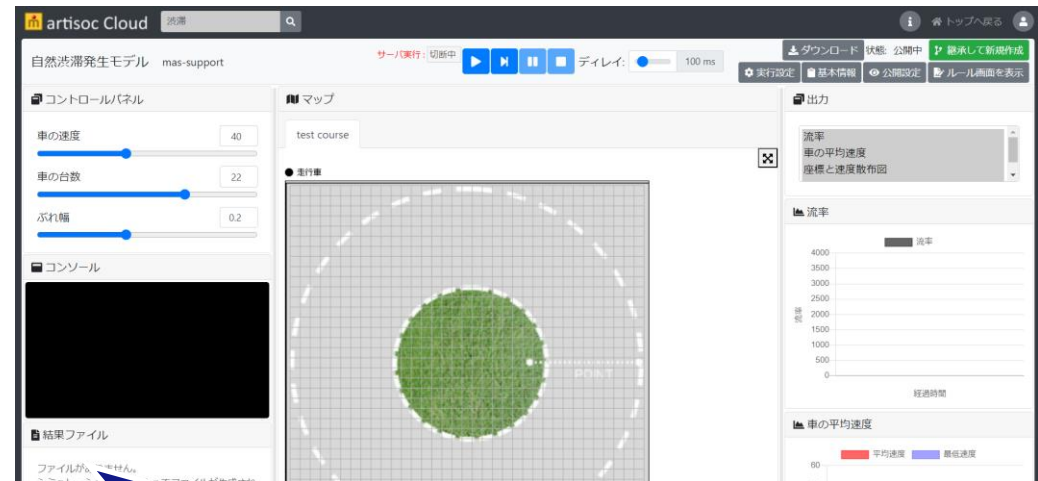


Artisoc Cloudのモデルを動かす

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



自然渋滞発生モデルが開きます



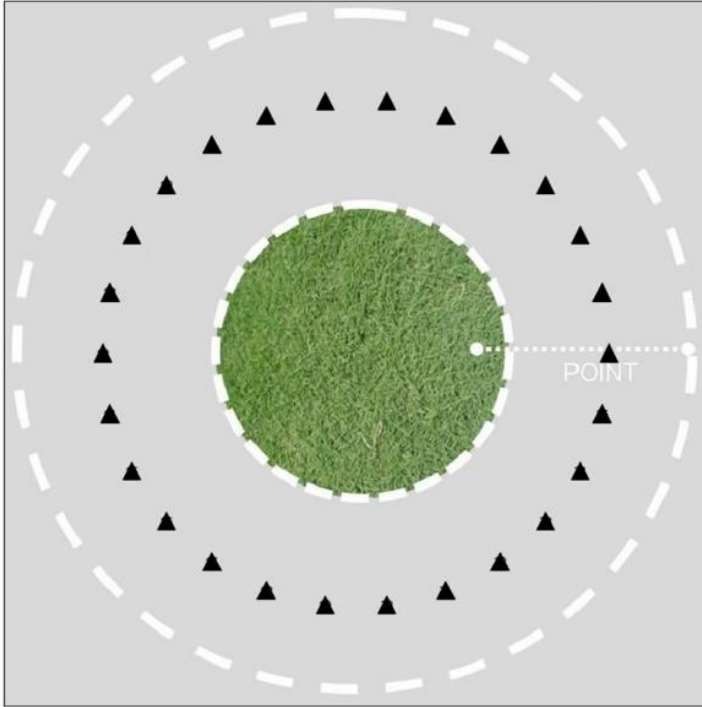
[自然渋滞発生モデル](#)

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/>

□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/>

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

□本日のチュートリアルについて

■ 概要

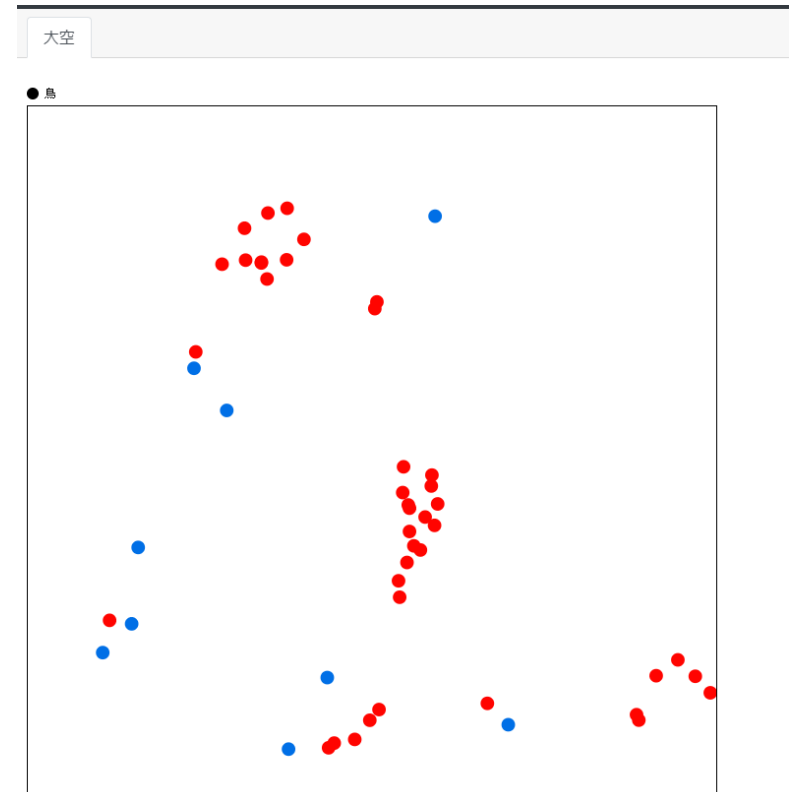
□ 『飛ぶ鳥モデル』の作成を通じ、artisoc Cloudの基本的テクニックを習得する。

➡ 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



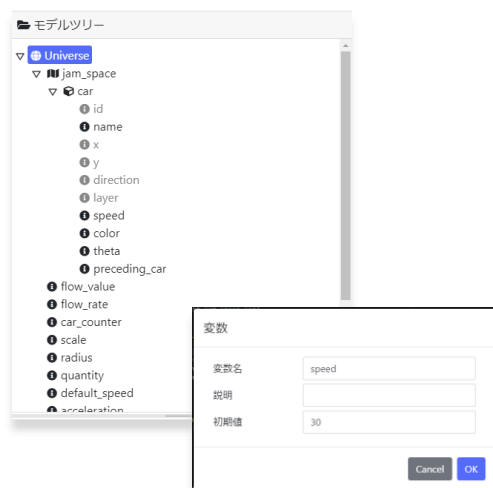
I. モデルの作成手順を学ぶ

□まずは動かしてみよう

- 以下のような簡単なモデルを作成し、artisoc Cloud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体だけ存在する。
 - 「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

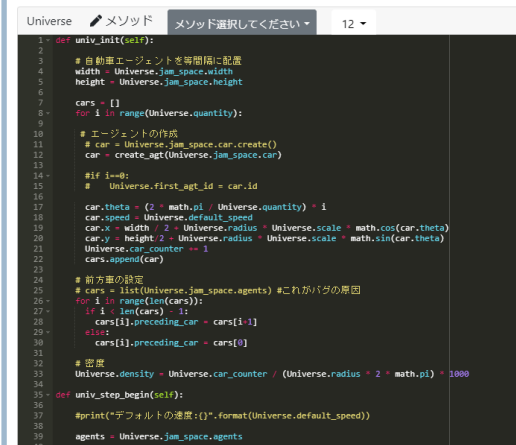
① 空間／エージェントの種類・属性を作成



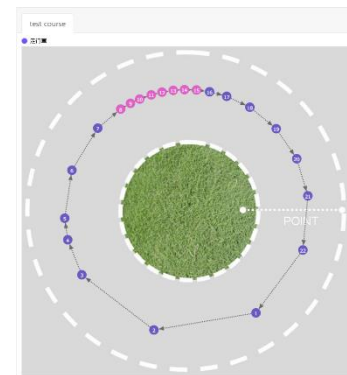
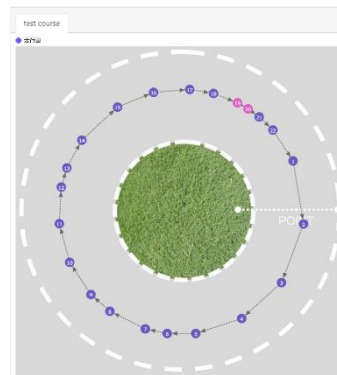
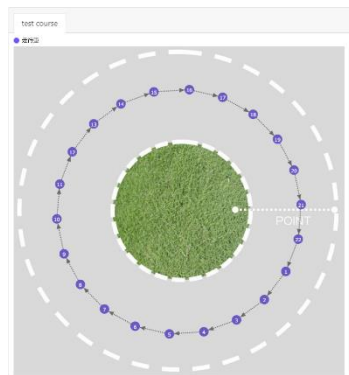
② シミュレーション結果の出力形式を決定



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

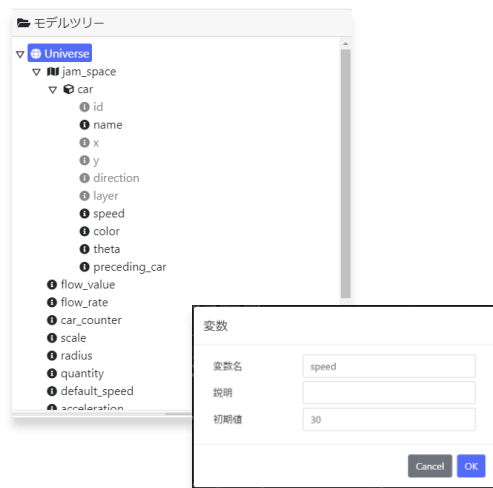
- 右上のアイコンマークから「新規モデルの作成」をクリックし、モデル名を入力します。
 - 「飛ぶ鳥モデル」と名付けましょう。

The image illustrates the steps to create a new model in the software. It is divided into three main sections:

- Left Panel (Sidebar):** A vertical menu with the user name 'mas-support' at the top. The menu items are: ユーザ情報, ユーザページ, **新規モデルの作成** (highlighted with a red dashed box and a mouse cursor), モデルのアップロー, 管理ページ, and ログアウト.
- Top Panel (Navigation):** A dark header bar with an information icon, a home icon labeled 'トップへ戻る', and a user profile icon. Below this is a row of buttons: ダウンロード, 状態: 編集中, and 公開. A second row contains: 実行設定, **基本情報** (highlighted with a red dashed box and a mouse cursor), 公開設定, and ルール画面を表示.
- Right Panel (Form):** A form titled 'モデル基本情報'. It contains the following fields:
 - 作成者: mas-support
 - モデル名: 飛ぶ鳥モデル** (highlighted with a red dashed box)
 - 概要: 説明がありません。 (Text area with a note: 概要欄は10,000文字以内で入力してください。)
 - モデルのタグ: 新規タグ (input field) and 追加 (button)
 - モデルのイメージ画像: A blue placeholder image with a plus icon in the top right corner.At the bottom right of the form are 'Cancel' and 'OK' buttons, with a mouse cursor pointing at the 'OK' button.

□空間／エージェントの種類・属性を作成

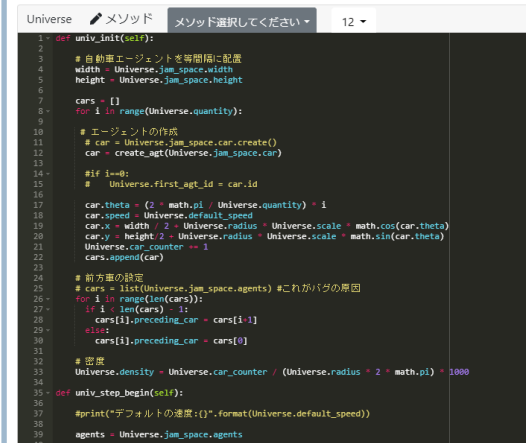
① 空間／エージェントの種類・属性を作成



② シミュレーション結果の出力形式を決定

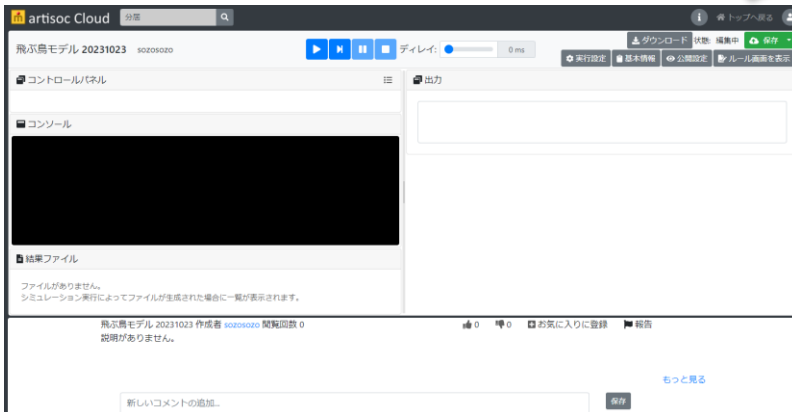


③ エージェントの行動ルールを作成

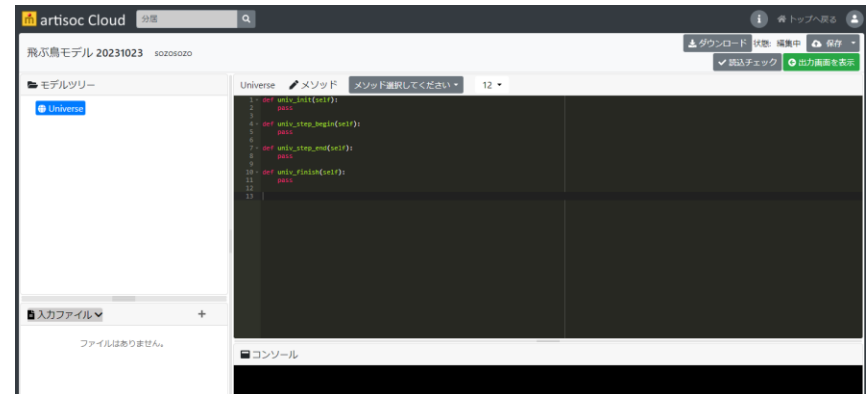


□ルール画面の表示

- 「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。



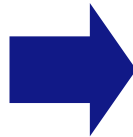
出力画面



ルール画面

□空間／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。



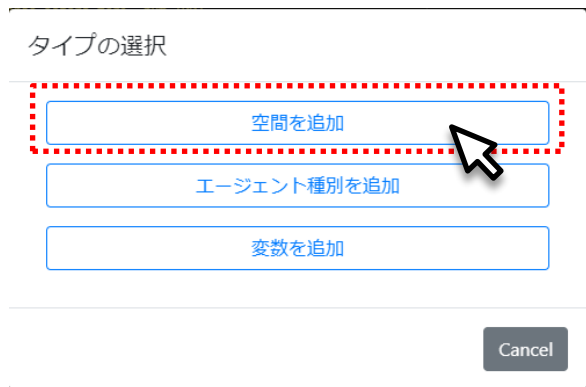
□空間の定義

■ Universe右に表示される「+」をクリックして空間を作成します。

□ 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（oozora）右に表示される「+」をクリックしてエージェント種別を作成します。

- エージェント種別名「tori」・・・鳥エージェント

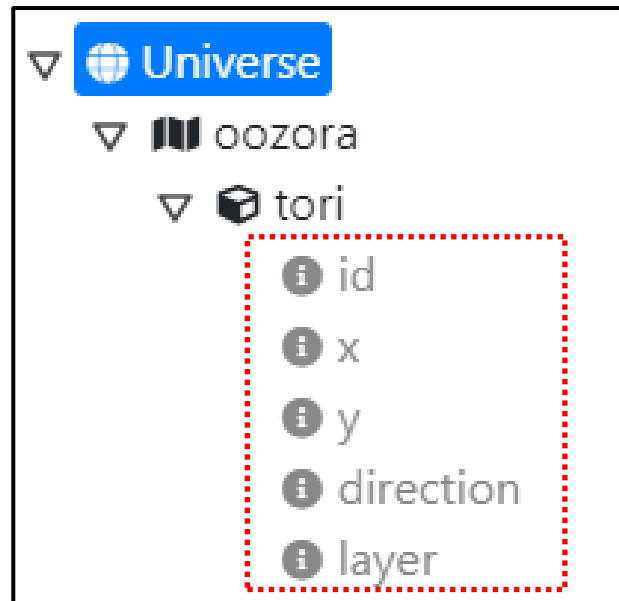
➡ 「説明」は自由に、「記憶数」は0のままでOK。

※この定義はエージェント種別でエージェントそのものではありません。



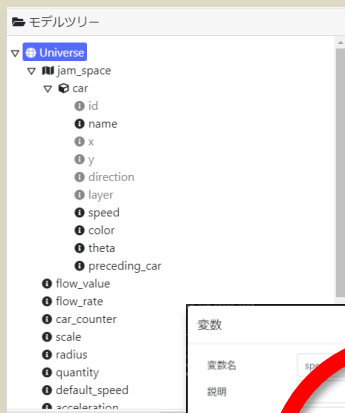
□エージェントの変数

- 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。



□シミュレーション結果の出力形式を決定

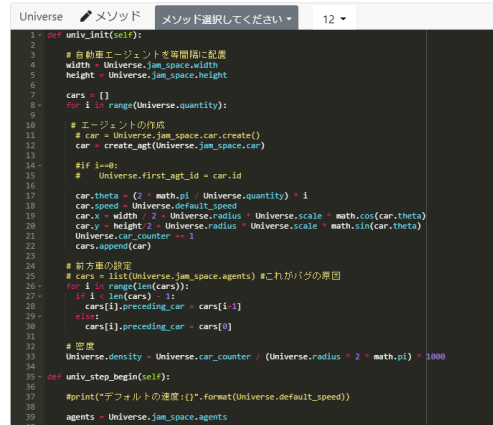
① 空間／エージェントの種類・属性を作成



② シミュレーション結果の出力形式を決定



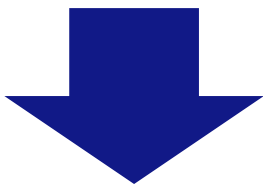
③ エージェントの行動ルールを作成



□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 Cloud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・・

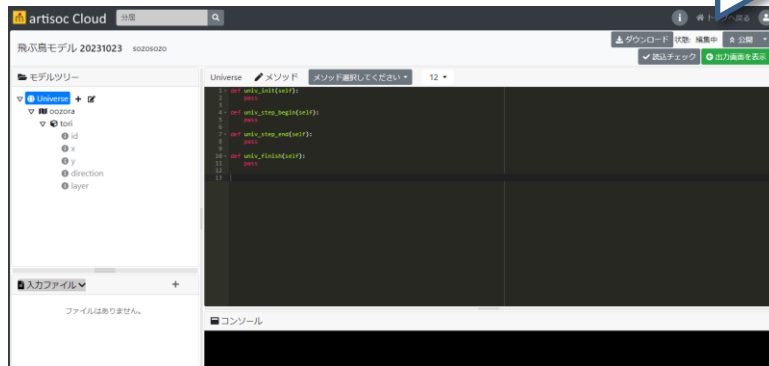
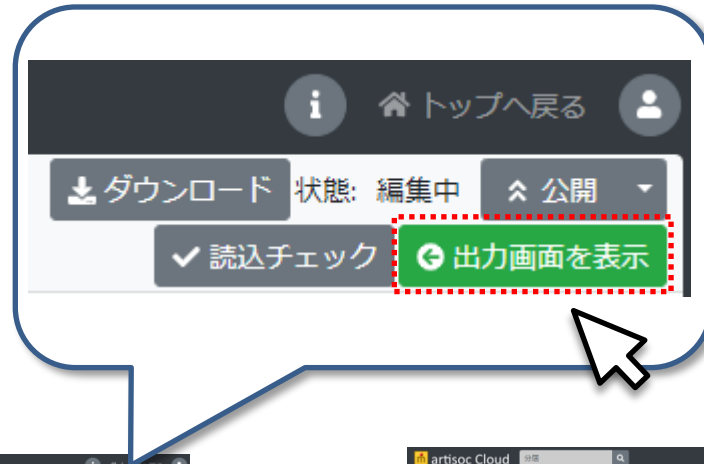


こまめに保存

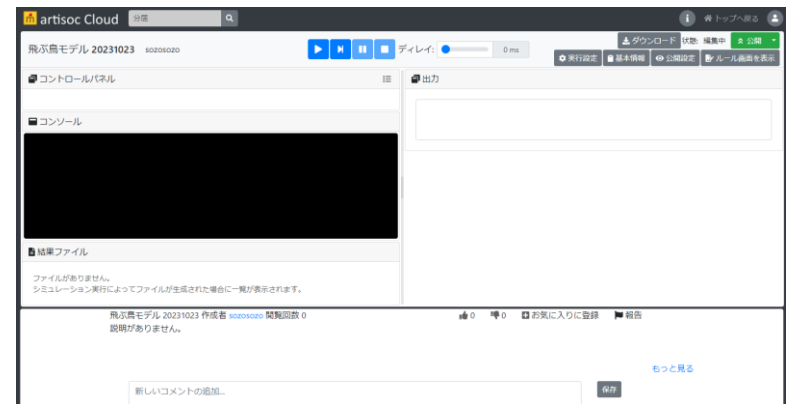


□出力設定

- 「出力画面を表示」をクリックして出力画面へ移動します。



ルール画面



出力画面

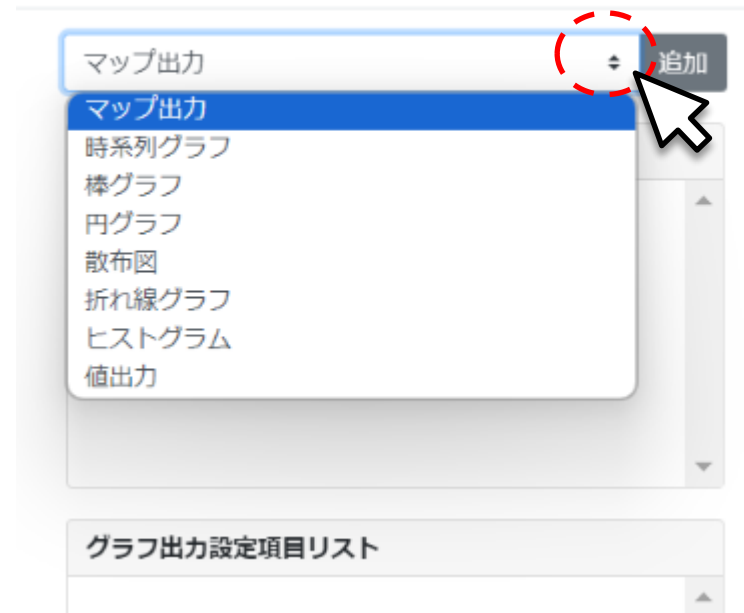
□出力設定

- 空間とエージェントを「マップ」として出力します。
 - 下図のようにマップ出力を選択します。



結果の出力の仕方を
設定する画面を表示

出力設定



結果を「マップ」とする
出力項目を追加

□出力設定

- 空間とエージェントを「マップ」として出力します。

マップ出力設定

マップ名:

空間:

レイヤ番号:

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:

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:

最大値:

Y軸設定

最小値:

最大値:

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:

エージェント:

マーカー

なし

選択

ファイル:

拡大率:

エージェント表示色

固定色

変数指定

エージェント情報の表示

表示する変数:

小数の表示桁数: 桁

文字色:

エージェント間に線を引く

対象の変数:

線の種類:

矢印の種類:

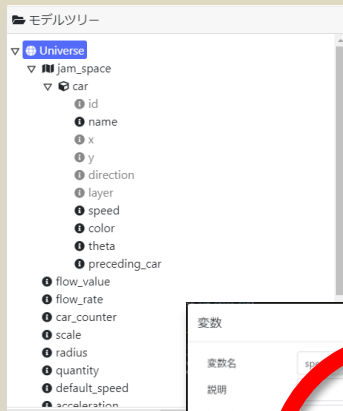
線の色:

Cancel OK

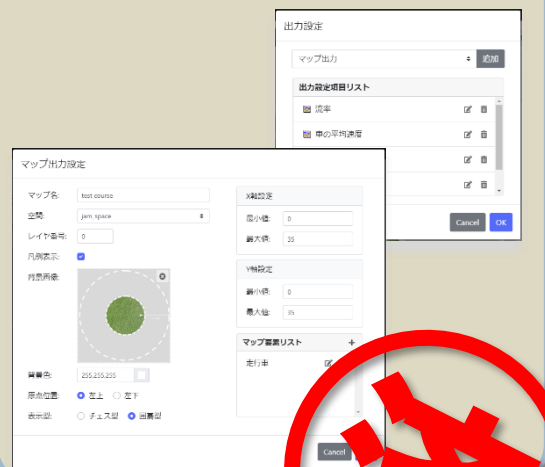
要素名 : 鳥
エージェント : tori
→エージェントtoriを「鳥」という名前で出力
※エージェントを表示する形状、色なども設定できるが、デフォルトでOK

□エージェントの行動ルールを作成

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

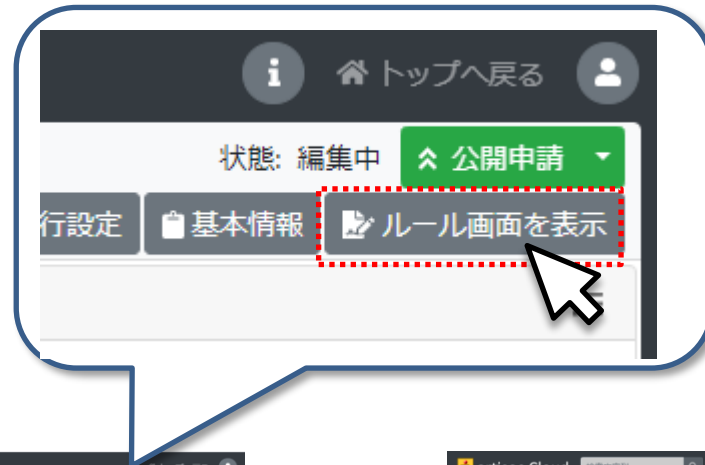
```
Universe メソッド メソッド選択してください 12 -
1 def univ_init(self):
2
3 # 自動車エージェントを等間隔に配置
4 width = Universe.jam_space.width
5 height = Universe.jam_space.height
6
7 cars = []
8 for i in range(Universe.quantity):
9
10 # エージェントの作成
11 # car = Universe.jam_space.car.create()
12 car = create_agt(Universe.jam_space.car)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

□ルールを作成

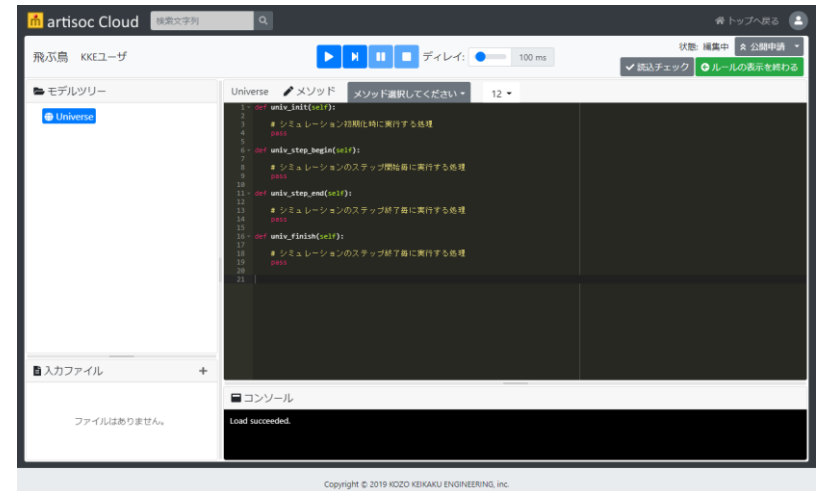
- 以下のようなモデルのルールを作成します。
 - シミュレーション開始時に鳥エージェントを1体だけ作成
 - 鳥エージェントの初期位置は空間の中央
 - 鳥エージェントは毎ステップ、前方に1進む

□ルールを作成

- 「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。



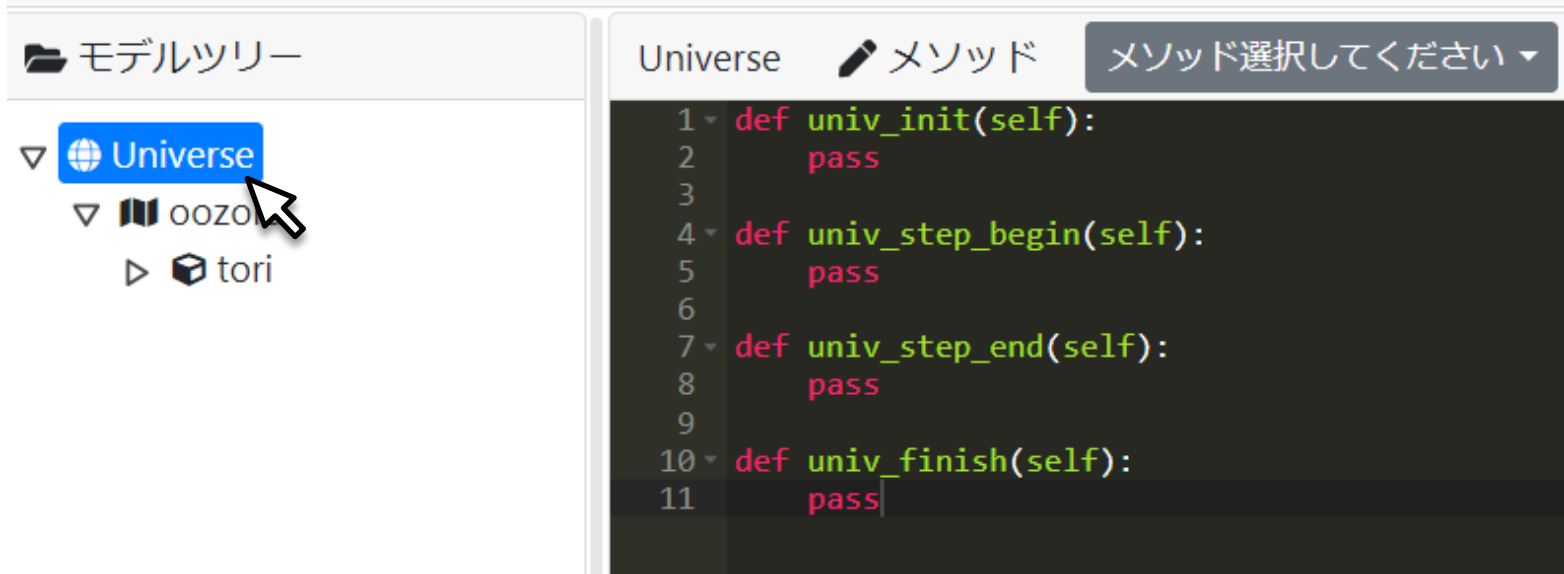
出力画面



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のルールは以下の構成です。
 - 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



モデルツリー


- Universe
 - OOZO
 - tori

Universe メソッド メソッド選択してください ▾

```
1 def univ_init(self):  
2     pass  
3  
4 def univ_step_begin(self):  
5     pass  
6  
7 def univ_step_end(self):  
8     pass  
9  
10 def univ_finish(self):  
11     pass
```

□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体生成します。
 - 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     create_agt(Universe.oozora.tori)  
3  
4  
5 def univ_step_begin(self):  
6     pass  
7  
8 def univ_step_end(self):  
9     pass  
10  
11 def univ_finish(self):  
12     pass
```

□TIPS : ルール作成時の注意 : インデント

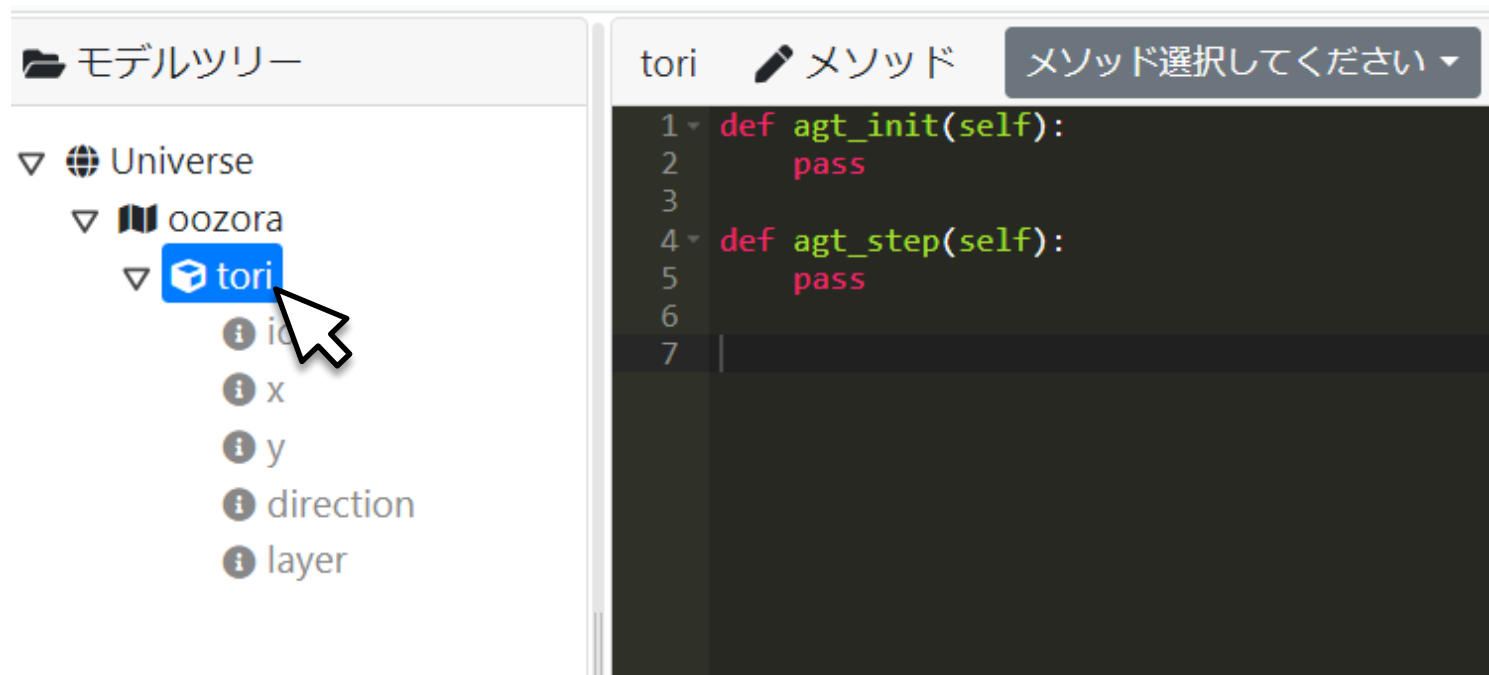
- ルールは、tabキーを使って1段下げた部分に記述することに注意しましょう。
 - これをインデントといいます。
 - インデントをするにはtabキーを使うと便利です。
 - ➡ 「shift + tab」でインデントが戻ります。
 - エンターキーで改行してもインデントは維持されます。
- ※空白行は自由に挿入してかまいません。

```
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       
8 ▾ def univ_step_end(self):  
9     pass  
10      
11 ▾ def univ_finish(self):  
12    pass
```

□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構成です。
 - 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します。
 - 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」は、「変数に数値を代入する」という意味です。
 - 「pass」を消し、下図のように記述してください
 - ➔ pass (パス) は何もしないという意味


```
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

□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みます。
 - 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です。

```
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
8     self.forward(1)
9
```

□ルールの構成

■ 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

toriエージェントの生成

■ 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

toriエージェントが
X=25,y=25に配置

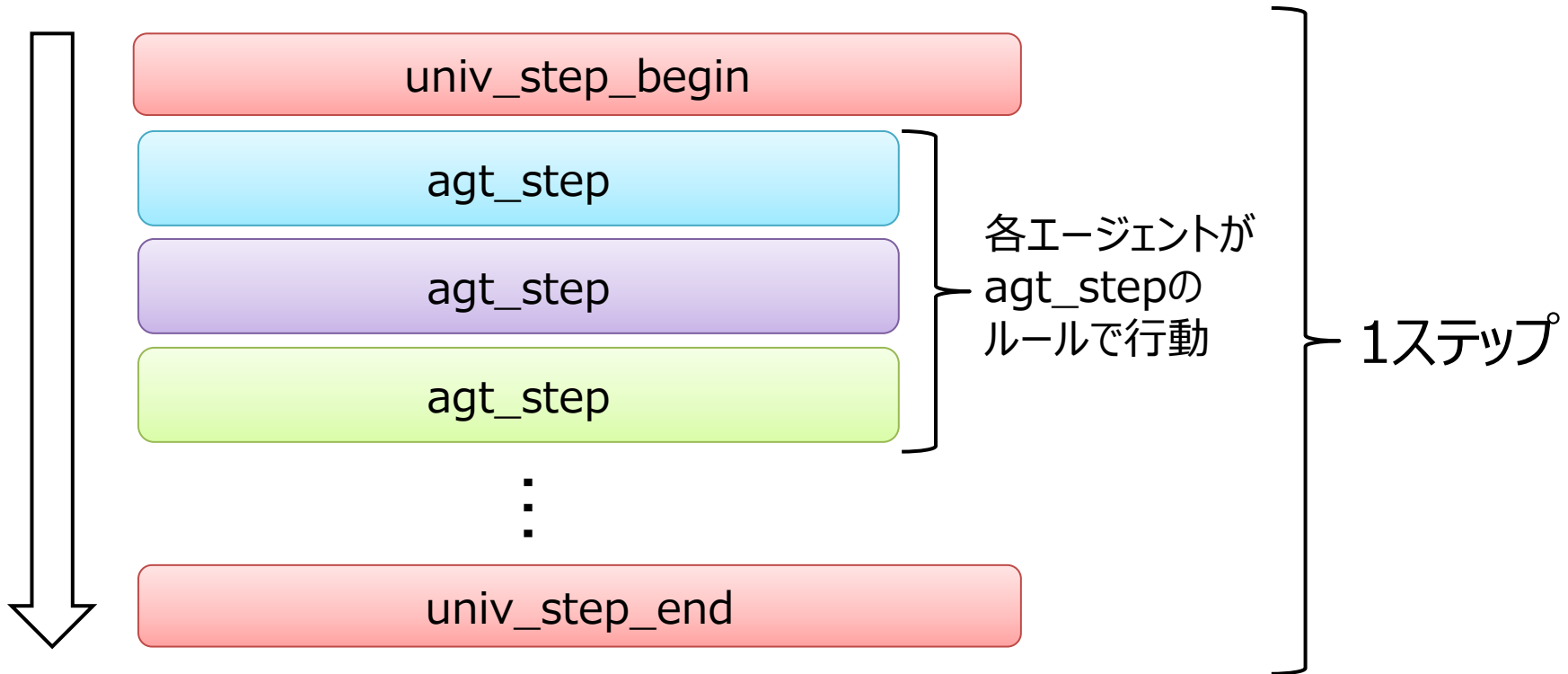
toriエージェントが「1」前進

□ステップについて

■ ステップ

- artisoc Cloudの時刻単位
- 1ステップの間に、全てのエージェントがルールにしたがって行動する。

■ 1ステップの流れ



□モデル構築完了

① 空間／エージェントの種類・属性を作成

モデルツリー

- Universe
 - jam_space
 - car
 - id
 - name
 - x
 - y
 - direction
 - layer
 - speed
 - color
 - theta
 - preceding_car
 - flow_value
 - flow_rate
 - car_counter
 - scale
 - radius
 - quantity
 - default_speed
 - acceleration

変数

変数名	初期値
scale	30

済

② シミュレーション結果の出力形式を決定

出力設定

マップ出力

出力設定項目リスト

マップ出力設定

マップ名: test ovato

種類: jam_space

レイアウト番号: 0

内訳表示:

内訳画像:

X軸設定

最小値: 0

最大値: 35

Y軸設定

最小値: 0

最大値: 35

マップ要素リスト

走行法

済

③ エージェントの行動ルールを作成

```
1 def univ_init(self):
2
3 # 自動車エージェントを等間隔に配置
4 width = Universe.jam_space.width
5 height = Universe.jam_space.height
6
7 cars = []
8 for i in range(Universe.quantity):
9
10 # エージェントの作成
11 # car = Universe.jam_space.car.create()
12 car = create_agt(Universe.jam_space.car)
13
14 # Universe.first_agt_id = car.id
15
16 car.theta = (2 * math.pi / Universe.quantity) * i
17 car.speed = Universe.default_speed
18 cars.append(car)
19 # car.x = Universe.radius * Universe.scale * math.cos(car.theta)
20 car.y = height / 2 + Universe.radius * Universe.scale * math.sin(car.theta)
21 Universe.car_counter += 1
22 cars.append(car)
23
24 # 前方車の設定
25 # cars = list(Universe.jam_space.agents) #これがバグの原因
26 for i in range(len(cars)-1):
27     cars[i].preceding_car = cars[i+1]
28
29 # 密度
30 cars[i].preceding_car = cars[0]
31
32 Universe.density = Universe.car_counter / (Universe.radius * 2 * math.pi) * 1000
33
34 def univ_step_begin(self):
35 #print("エージェントの速度: {}".format(Universe.agents[0].speed))
36
37 agents = Universe.jam_space.agents
```

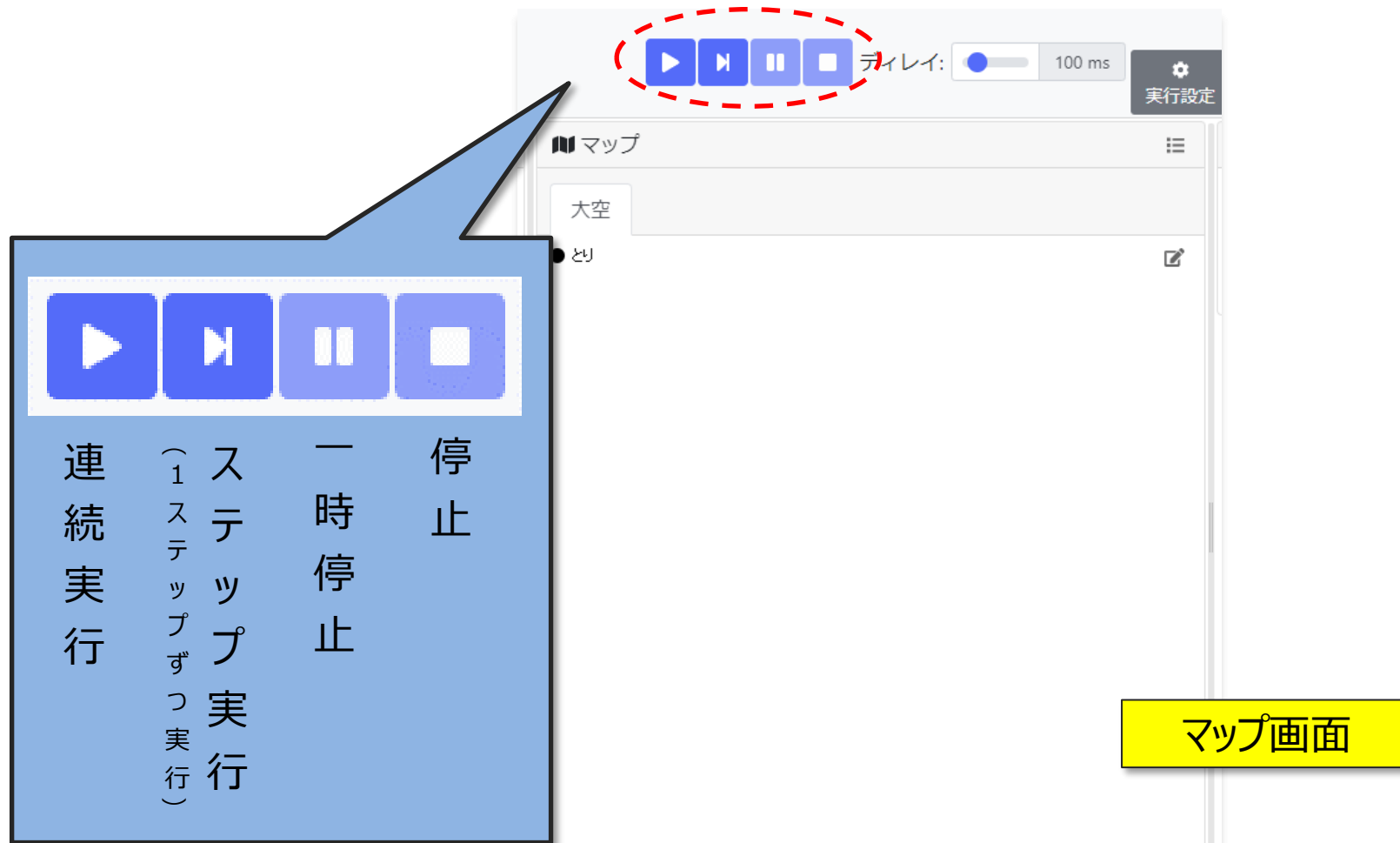
済

□ここまでのモデル

- 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/GyXuUvDRbG-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



□シミュレーションを実行



The screenshot shows a simulation control panel. At the top, there are four blue buttons: a play button, a next button, a pause button, and a stop button. These buttons are circled in red. To the right of the buttons is a slider labeled '遅延: 100 ms' and a gear icon labeled '実行設定'. Below the buttons, there is a section titled 'マップ' with a list of items: '大空' and 'とり'. A yellow callout box points to the buttons and contains the following text:

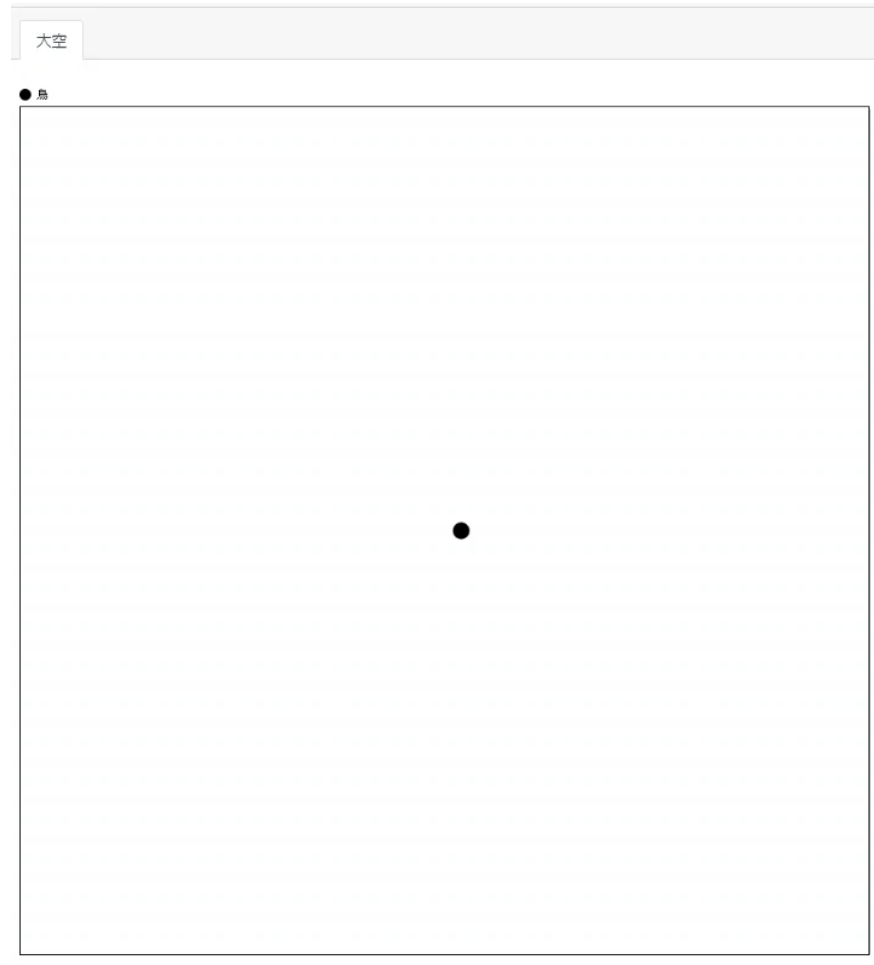
連続実行	(1ステップずつ実行)	ステップ実行	一時停止	停止
------	-------------	--------	------	----

A yellow box at the bottom right of the screenshot is labeled 'マップ画面'.

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。

□モデルの動作を確認する

- 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。



□TIPS : シミュレーション速度を調整する

- 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



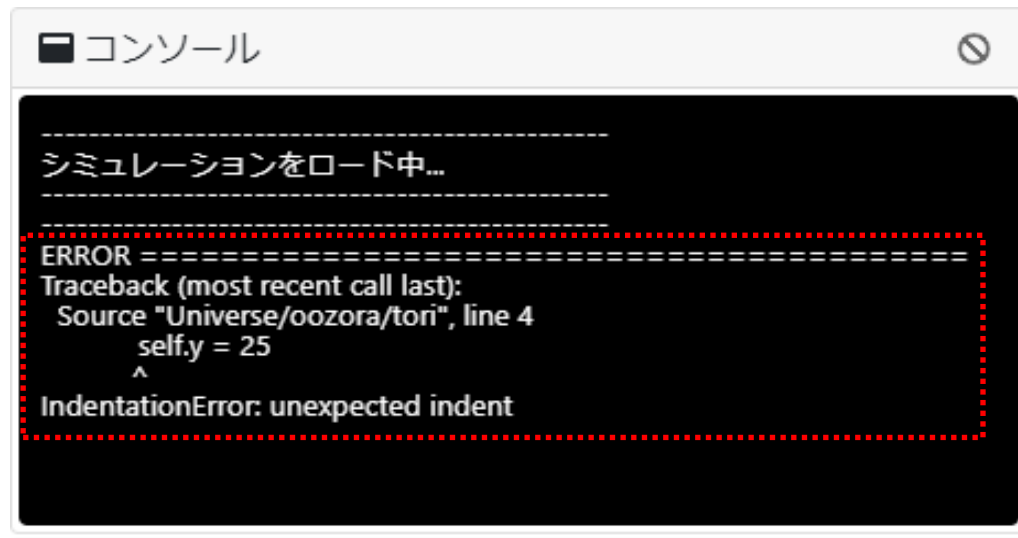
右に動かすと実行速度が遅くなり、左に動かすと速くなる

□TIPS : シミュレーションが実行できないとき

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



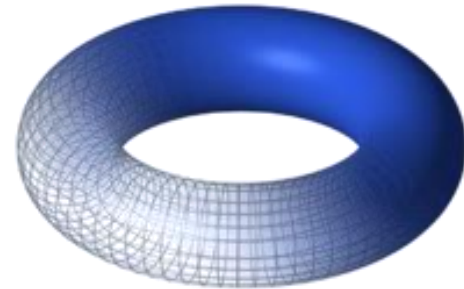
- エラーメッセージの意味が分からないときは、質問してください。
 - 遠隔参加の方は、エラーメッセージをQ&Aにコピーしてお知らせください。



□TIPS : 空間のループを変更する

- 空間がループ = 上端-下端・左端-右端が繋がった空間

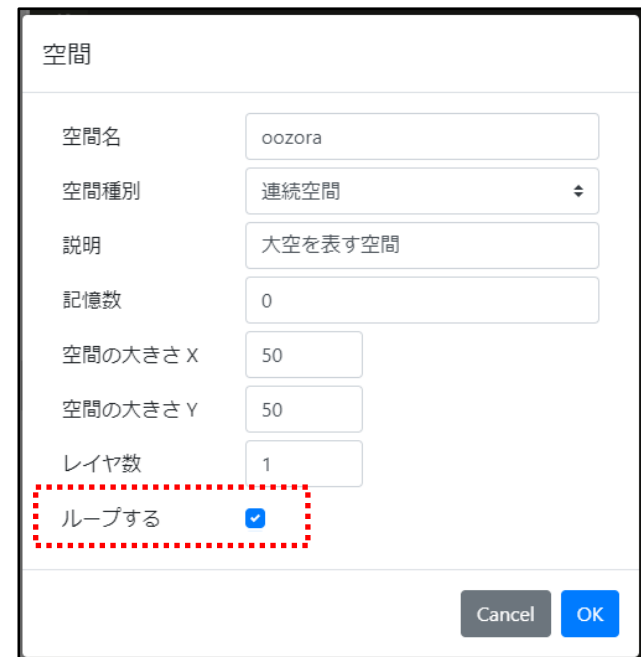
ループした空間のイメージ



「ルール画面を表示」をクリック



「Edit」をクリック



2. 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

- 途中から来た人は、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/GyXuIUvDRbG-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



□変数設定の変更（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。
 - self.direction = 45

モデルツリー

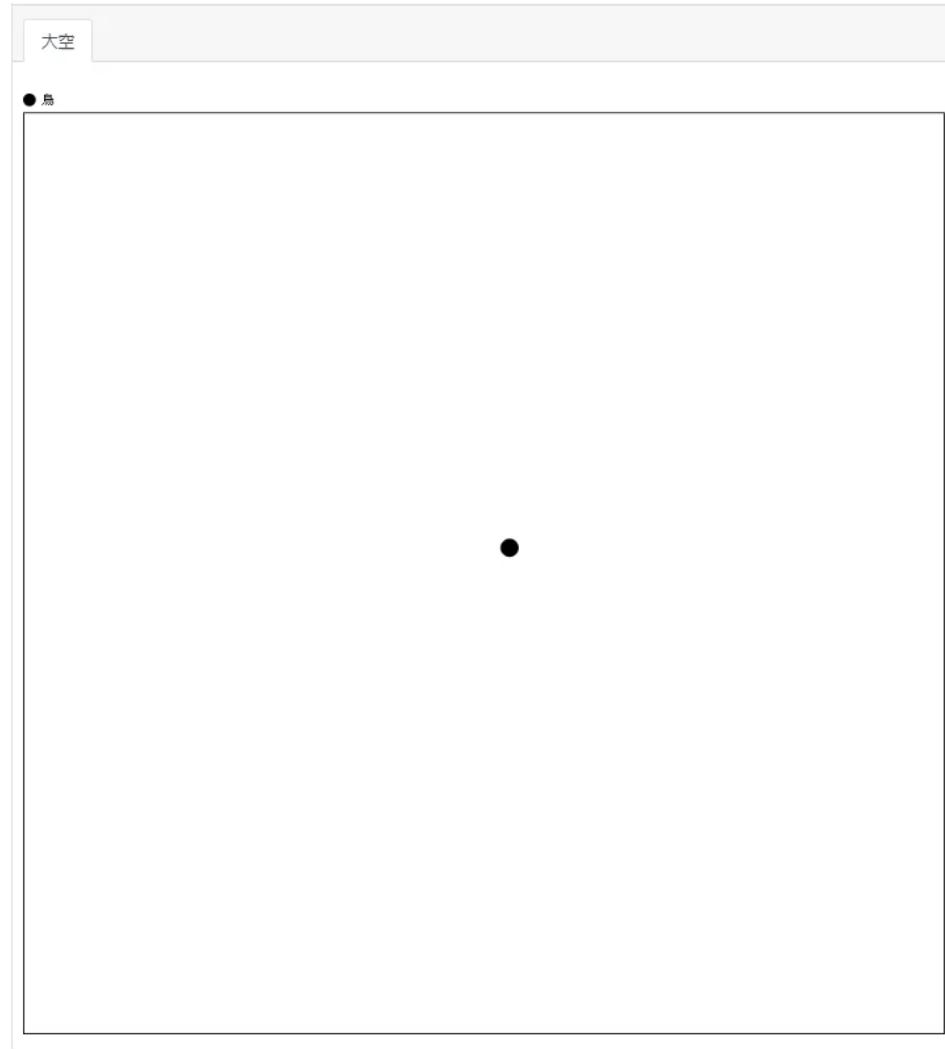
- Universe
 - oozora
 - tori
 - id
 - x
 - y
 - direction
 - layer

tori メソッド メソッド選択してください ▾

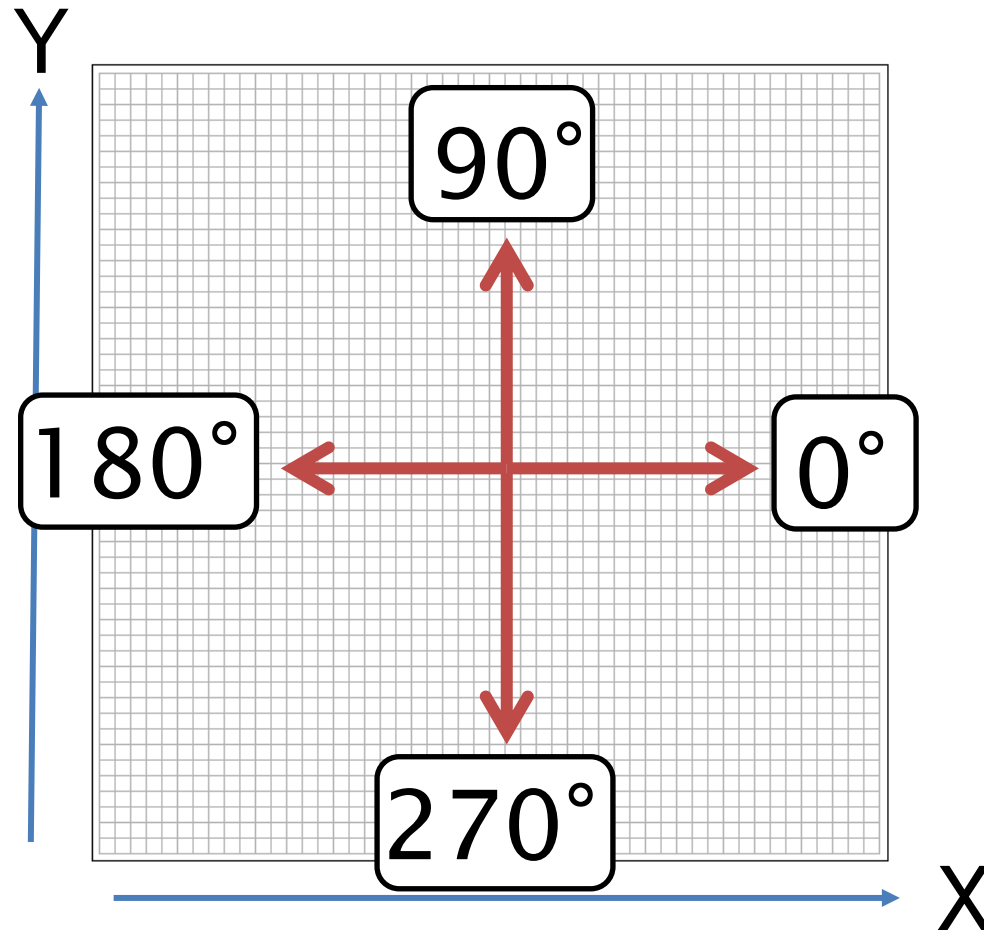
```
1 def agt_init(self):
2
3     self.x = 25
4     self.y = 25
5
6     self.direction = 45
7
8 def agt_step(self):
9
10    self.forward(1)
11
12
```

□モデルの動作を確認する

- 鳥が右上に飛んでいきます。




□artisoc Cloudにおける方向



※左下原点の場合

□関数の使用（進む方向をランダムに変える）

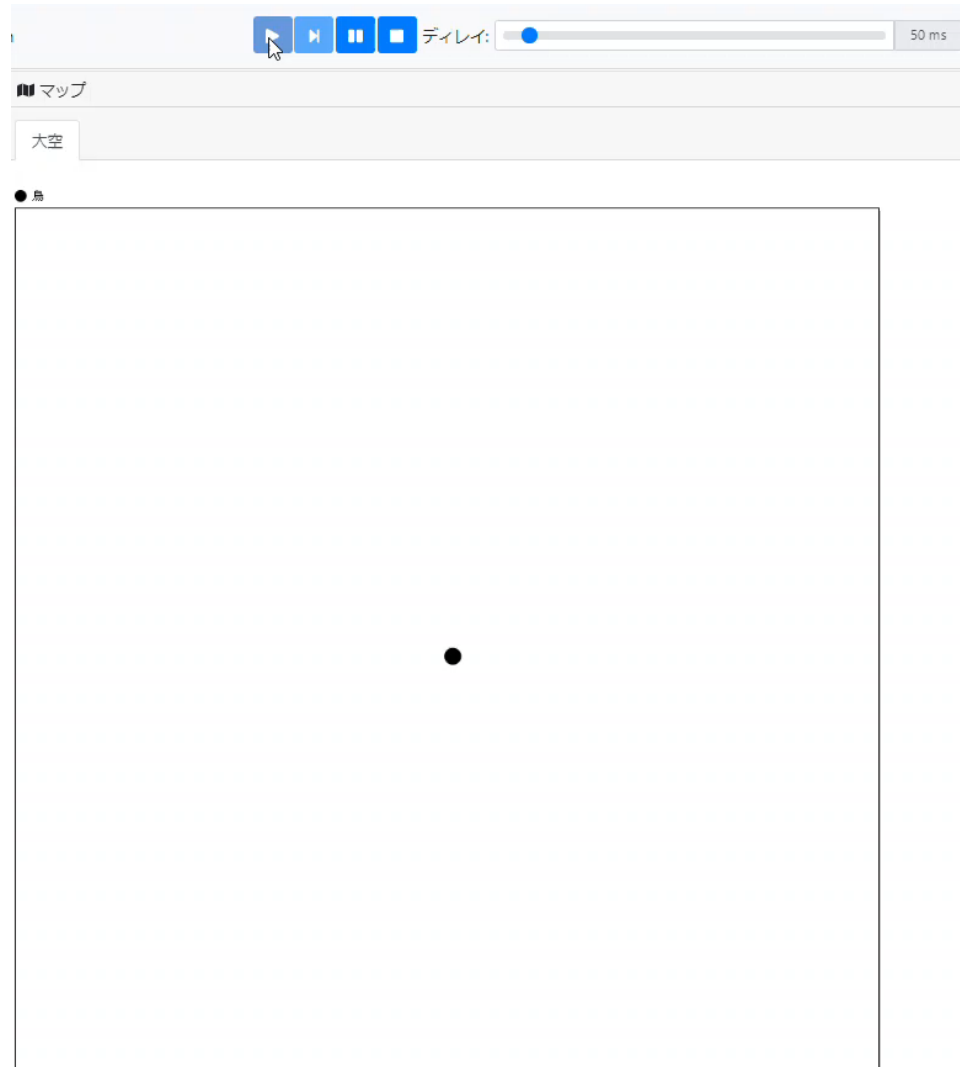
- 属性「direction」に進む方向の値をランダムに設定（0~360度）して実行します。
 - **rand** : 0~1のランダムな値（一様乱数）を取得する関数
 - 「rand() * 360」で0~360のランダムな値を意味します。
 - ➡ 「*」は掛け算の記号です。
 - ➡ 関数には必ず()がつきます。
 - ➡ 「=」は右の値を左の変数の値に代入するという操作を示す。←要注意

```
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
```


□モデルの動作を確認する

- 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
 - 戻り値（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
※戻り値がない（使わない）関数もあります

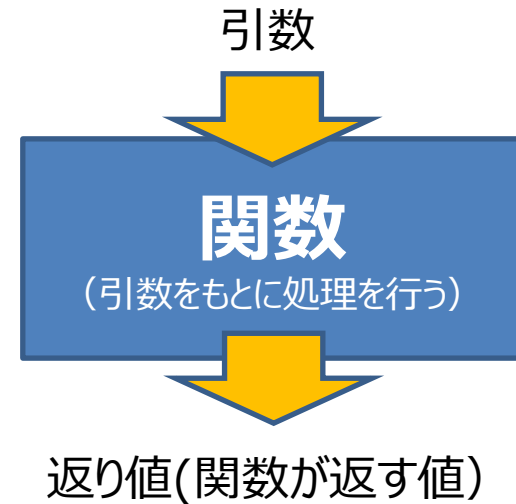
- 関数の例：forward
 - 引数：進む距離
 - 処理：引数の値だけ前に進む
 - 戻り値：正常終了時は-1（今は使っていません）

```
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
 - 戻り値：0~1の一樣乱数

```
r = rand() # rand関数の戻り値（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



□エージェントの生成の仕方の変更

- 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。

モデルツリー

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

```
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, num = 100)  
4     .....  
5 def univ_step_begin(self):  
6     pass  
7  
8 def univ_step_end(self):  
9     pass  
10  
11 def univ_finish(self):  
12     pass
```

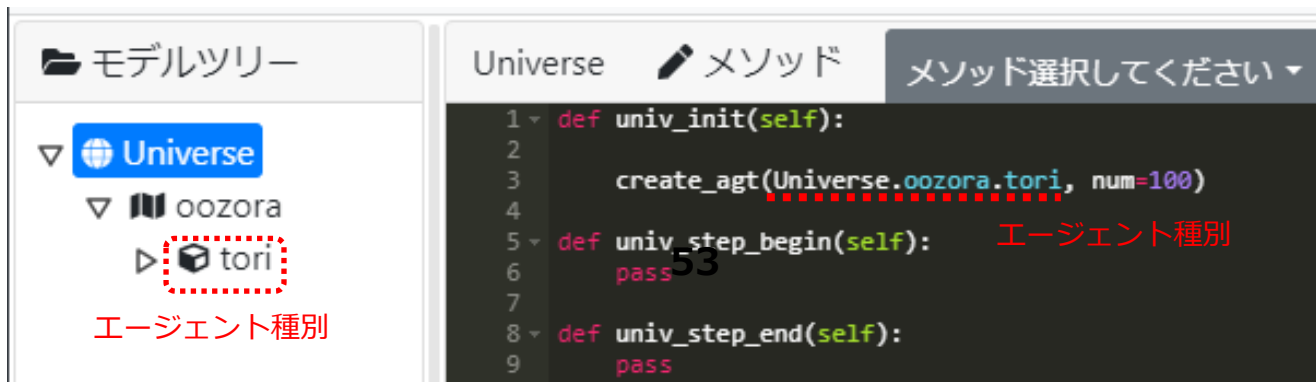
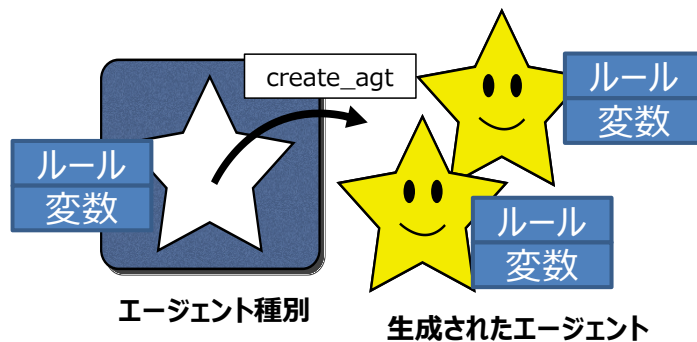
□モデルの動作を確認する

- 100羽の鳥が円形に飛んでいきます。



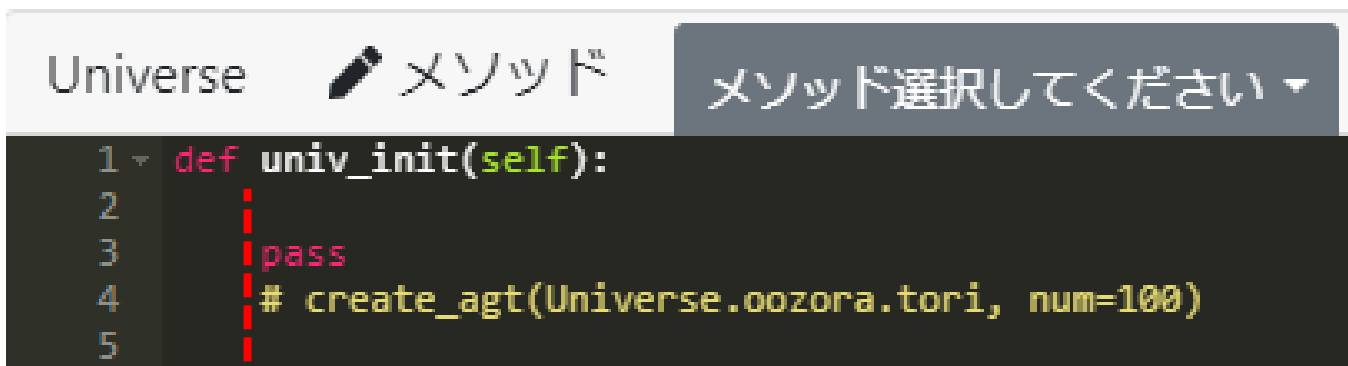
□エージェント種別とエージェント


- エージェント種別：エージェントの「ひながた」のようなものです。
 - 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を指定して、エージェントを生成します。
- 生成されたエージェントは共通のルールと変数を持ちますが、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。



□TIPS : コメントアウトとpass

- コメントアウトは、コードを無効化する機能です。
 - 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は、「何もしない」を意味するルールです。
 - 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 Cloudに教えるためにpassを記述します。



```
Universe  メソッド メソッド選択してください ▾
```

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

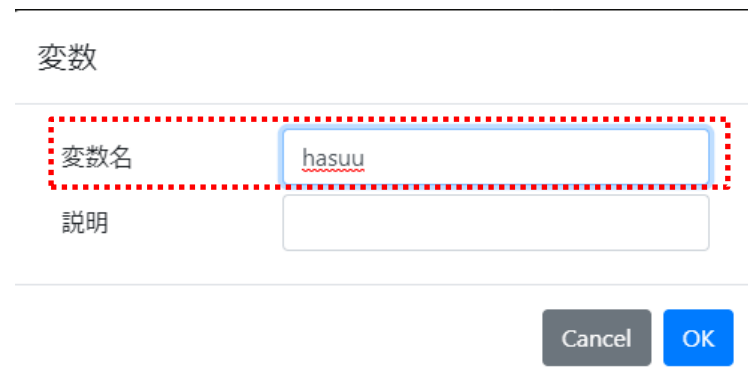
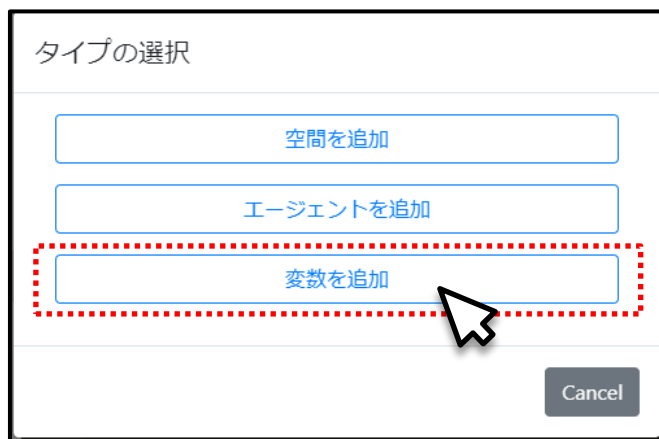
コントロールパネルの設定

- 鳥の数をスライダーの操作で設定できるようにする。



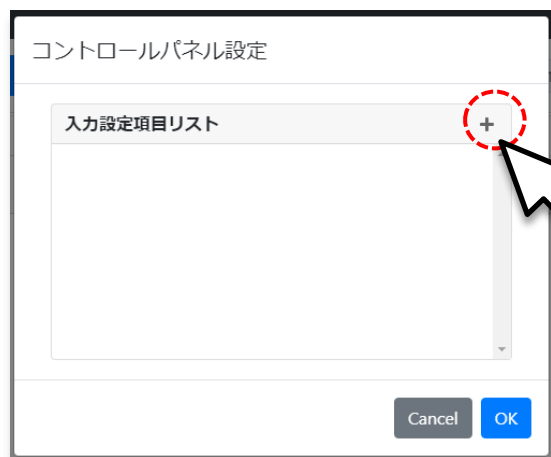
コントロールパネルの設定

- 鳥の数を表す変数としてUniverseの下に変数「hasuu」を作成する。



コントロールパネルの設定

- Universe変数「hasuu」をコントロールパネルの操作対象に設定する。
 - 出力画面に移動し、実行画面左上の「コントロールパネル」で設定する。
 - 「種類」で「スライダー」を選択する。



コントロールパネル設定

種類:	ボタン
コントロール名:	ボタン トグルボタン スライダー
設定対象:	直接入力 リスト ルール
値の型:	値を入力してください
ONの値:	

Cancel OK

コントロールパネルの設定

- 鳥の数をスライダーの操作で設定できるようにする。
 - Universe変数「hasuu」をコントロールパネルの操作対象に設定する。
 - ➡ 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:	50
範囲:	1 ~ 100
目盛り間隔:	1

Cancel OK



コントロールパネルで鳥の数を設定できます。

□コントロールパネルの設定

- 変数「hasuu」の数だけ、鳥を生成する。
 - for i in range(Universe.hasuu):
 one = create_agt(Universe.oozora.tori)
- 生成した順番に応じて、進む方向を決める。
 - (上記の続き)
 one.direction = i

```
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
```

□ここまでのモデル

- 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



3. 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。
- ○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。



□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。

（復習）コメントアウトの方法

- 行の先頭に「#」
- 行にカーソルを選択した状態で「ctrl + /」

```
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         .....
```

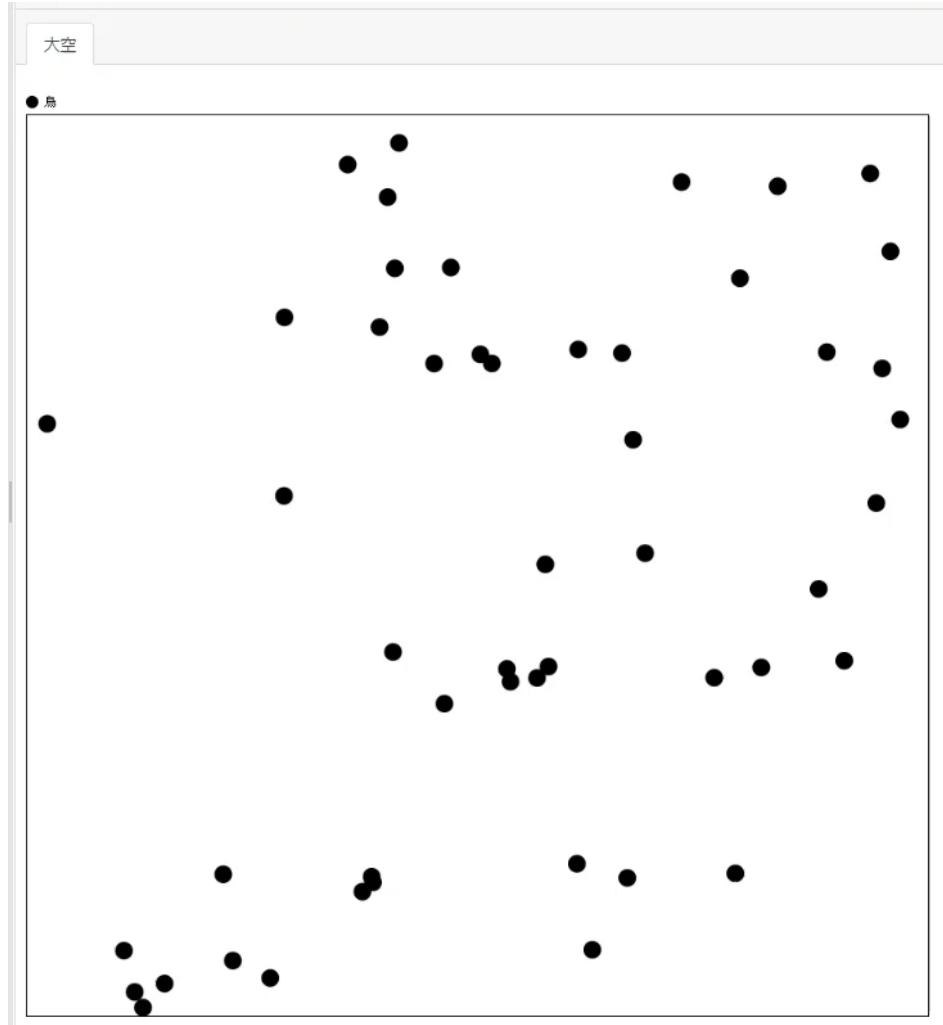
■ 鳥のルールを以下のように修正します。

- 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     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 def agt_step(self):
9
10    self.forward(1)
11
```

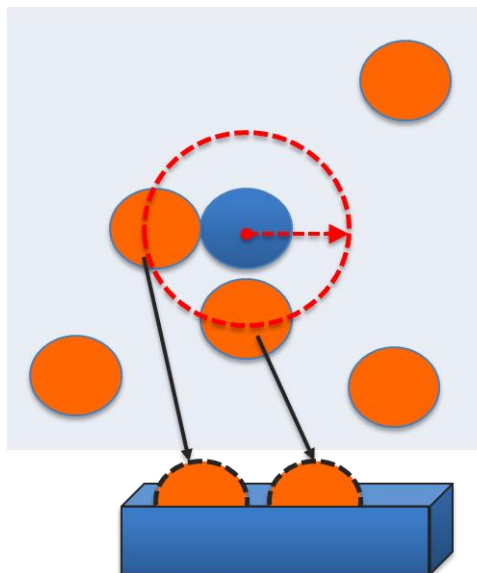
□モデルの動きを確認する

- 鳥がランダムに飛んでいきます（鳥の数=50）。



□エージェント集合型の変数の活用（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。
 - 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
 - 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。
 - 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)

□エージェント集合型の変数の活用（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。
 - **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）

```
8 def agt_step(self):
9
10     # 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）
11     tori_set = self.make_agtset_around_own(2, False)
12
13     self.forward(1)
14
```

□条件分岐文の活用（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てます。

- 「条件分岐文」はif文で記述します。

- ➔ 条件分岐分：もし ~ ならば … という処理

```
if 条件分岐文 :  
    処理
```

※処理はインデント内に記述

条件分岐文の例

$1 < 5$ [1は5未満である] ⇒ 真(True)

$4 \geq 8$ [4は8以上である] ⇒ 偽(False)

$3 == 4$ [3と4は等しい] ⇒ 偽(False)

- `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

```
8 def agt_step(self):  
9  
10 # 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）  
11 tori_set = self.make_agtset_around_own(2, False)  
12  
13 if count_agtset(tori_set) > 0: # もし周囲に鳥がいれば  
14     one = randchoice(tori_set) # ランダムに一を選ぶ  
15     self.direction = one.direction # 自分の向きをその鳥に合わせてる  
16  
17 self.forward(1)  
18
```

※ここで、tori_setは順序を持たない集合型。artisc4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。

□条件分岐文の活用（周囲の鳥に方向を合わせる）

- if count_agtset(tori_set) > 0: # もし周囲に鳥がいれば
 one = randchoice(tori_set) # ランダムに一羽を選ぶ
 self.direction = one.direction # 自分の向きをその鳥に合わせる

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

□モデルの動作を確認する

- 鳥が群れを作ります。



□本チュートリアル終了時点のモデル

- 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



おわりに



□初級チュートリアル、おつかれさまでした！

■ 初級チュートリアルで学んだこと

1. artisoc Cloudの起動方法
2. 新しくモデルを作る方法
3. エージェントの作り方

...

これから好きなモデルを好きなだけ作成できます！



→次は何をすればよいのだろう？

□Step1: artisoc Cloudをはじめ

■ 初級チュートリアルに参加する

- まずはartisoc Cloudを動かしてみよう。
- コンピュータのなかの人工社会を体感しよう。

■ いろんなモデルを見てみる

- 身の回りの複雑系には、読み物として身近な事例を紹介しています。
- 渋滞、人のつながり、流行、エスカレーター問題など

■ 手を動かしてみる

- MASの教材にいろんなコンテンツがアップされています。
- 興味のあるコンテンツをダウンロードしてお試ください。



飛ぶ鳥モデルからボイドモデルへ

難易度 ★★☆☆☆



立ち話モデル

難易度 ★★☆☆☆



ターミナル駅の通勤客の流れ

難易度 ★★★☆☆



病気の流行

難易度 ★★★☆☆



水槽の中のプランクトン

難易度 ★★★☆☆



シェリングの分层モデル

難易度 ★★☆☆☆

□Step2: artisoc Cloudの使い方を学ぶ

■ [artisoc Cloud教科書](#)を読んで勉強する

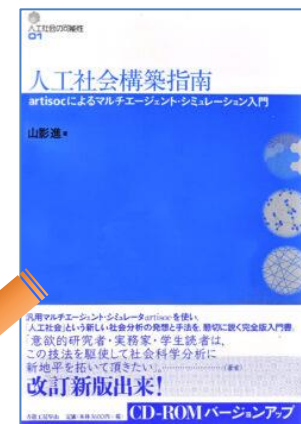
- 書籍『人工社会構築指南』をベースにartisoc Cloud向けに加筆
✓ **講義や研究室ゼミのテキストとしてご利用ください**

■ もっと本格的なモデルをつくりたいとき

- [モデル作成のレシピブック](#)には、ポテンシャル法やネットワークモデルによる経路探索など、モデル作成で参考になる手法が学べます。

■ 分からないことがあったら[質問掲示板](#)

- モデルを作成していてつまづいたとき
- モデル作成のヒントが欲しいとき



□Step3: 勉強会に参加して仲間をつくらう

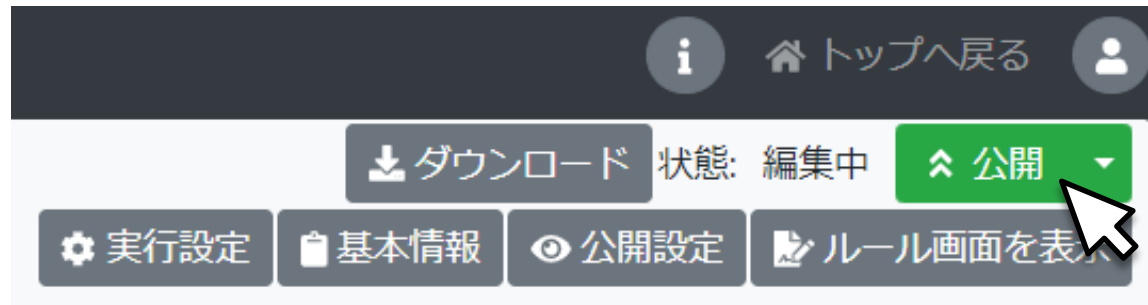
■ artisoc Cloud勉強会が目指すべきところ

1. 複雑系やMAS(マルチエージェント・シミュレーション)を学ぶための仲間づくりを行う場を提供したい。
2. 経験や分野の異なる多様な方々が集まり、相互作用することで、新しい発見や問題解決の場に育てたい。
3. 誰もが簡単に artisoc Cloud を利用して、社会課題について自分事として考え、行動する社会をつくりたい。



□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。



- 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>

