



## artisoc Cloudレシピブック

### 09. 描画ツール2を使って歩くモデルの高速版

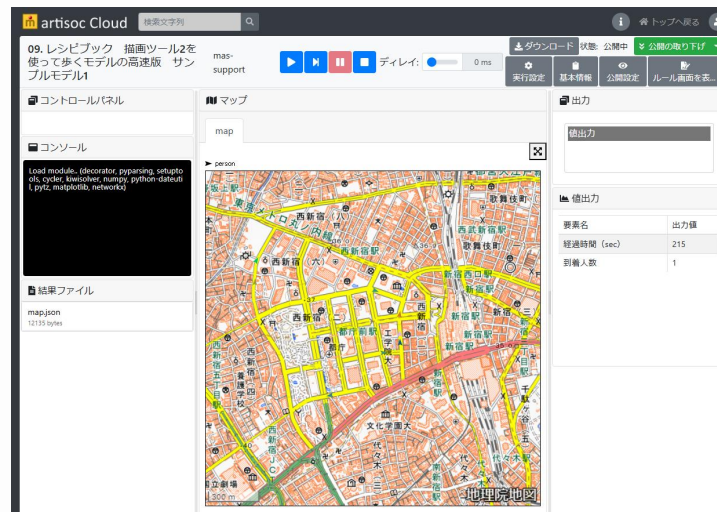
(株) 構造計画研究所  
創造工学部

<https://mas.kke.co.jp>

大規模な道路データでも高速に動作する移動モデルを作成します。

## 主な改良点

- 描画ツール2で作成した道路ネットワークを読み込み、pointエージェントを作成せずに Universe.graph にのみ情報を蓄積します。
- person エージェントが移動する際に、marker エージェントを直近の目的地に置くことで道路に沿って移動します。
- この変更により、ステップ毎に point エージェントのルールを実行しない、マップ出力に point エージェントを表示しないため、大規模な道路データを利用しても高速動作します。



### 09. レシピブック サンプルモデル1

## 地理院地図から背景地図を取得します

- [地理院地図](#)にアクセスし、任意の場所（例えば、新宿都庁）を検索して表示します。
- 右上の 共有 > 画像保存 をクリックします。画像大きさを指定し、ファイルを出力します。  
大きさを固定  
大きさ： 500 x 500  
出力ファイル名： map.png



# 地理院地図から背景画像を取得する（2）

## 地図の縮尺を取得します

- 画像を出力するときに緯度・経度が表示されます。
- 距離と方位角の計算 を開き、入力単位選択を「十進法度単位」に切り替えた後、緯度・経度を入力します。

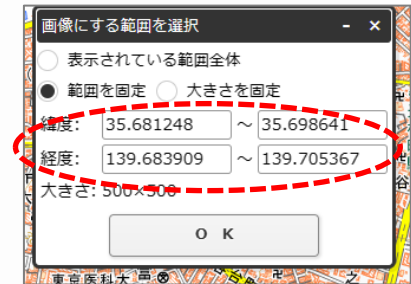
出発点	緯度：	35.681248	
	経度：	139.683909	
到着点	緯度：	35.681248	←出発点と同じ値
	経度：	139.705367	

- 「計算実行」をクリックすると距離が出力されます。

測地線長 1,942.488(m)

- 地図の縮尺を求めます。

地図の縮尺：  $1942.488(\text{m}) / 500(\text{セル}) = 3.88 \text{ (m/セル)}$  ←後述でルールに記述



## 描画ツール2を使って道路を定義します

- [描画ツール2](#)にアクセスします。
- 画像ファイル（map.png）の読み込みます。
- 操作コマンドを選択して、道路上をマウスでクリックしていきます。（次ページ参照）
- すべての線を引き終わったら、JSONファイルをダウンロードします。

ファイル名： map.json



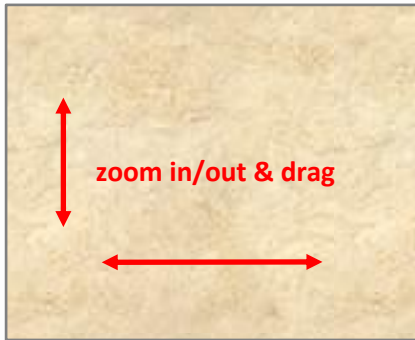
描画ツール2



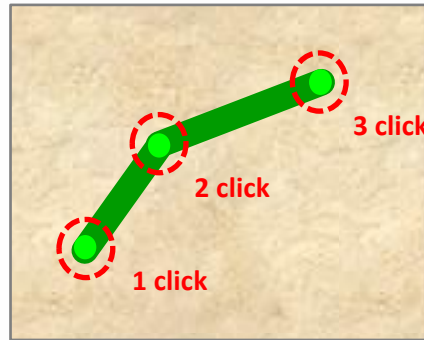
# 描画ツール2を使って道路を定義する（2）

操作コマンドを選択して、マウスをクリックします

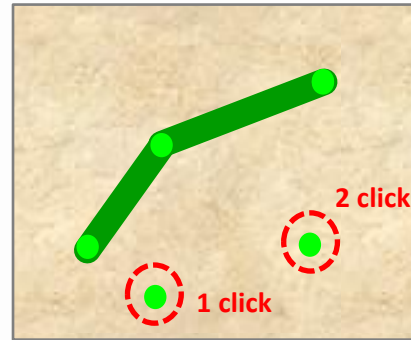
① 画像移動



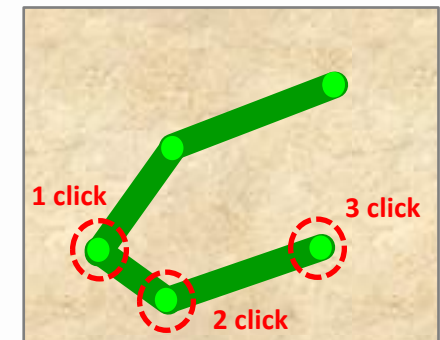
② ポイント&リンク



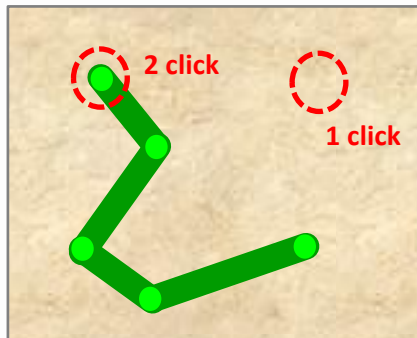
③ ポイント



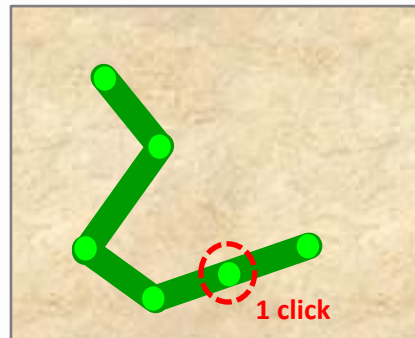
④ リンク



⑤ ポイント移動

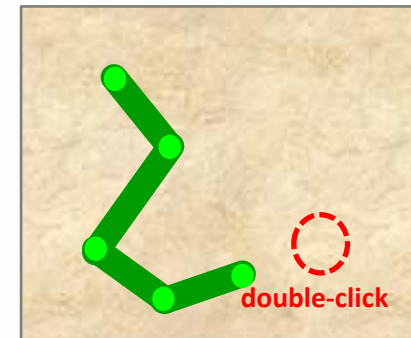


⑥ リンク分割



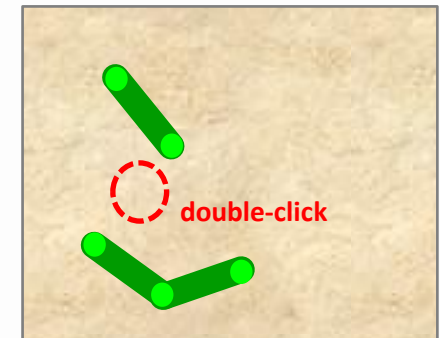
※ 分割するリンクの真ん中をクリック

⑦ ポイント削除



※ ポイントにつながるリンクも削除される

⑧ リンク削除

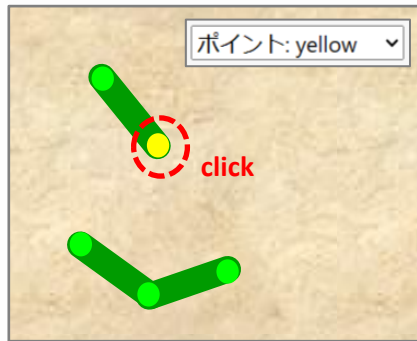


※ 削除するリンクの真ん中をクリック

# 描画ツール2を使って道路を定義する (3)

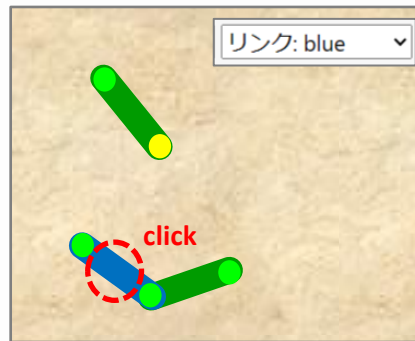
操作コマンドを選択して、マウスをクリックします

⑨ ポイント色変更



※ オプション「ポイント:色」で  
指定した色に切り替わる

⑩ リンク色変更



※ オプション「リンク:色」で  
指定した色に切り替わる

⑪ 一方通行切替



※ オプション「矢印型」で  
指定した矢印に切り替わる

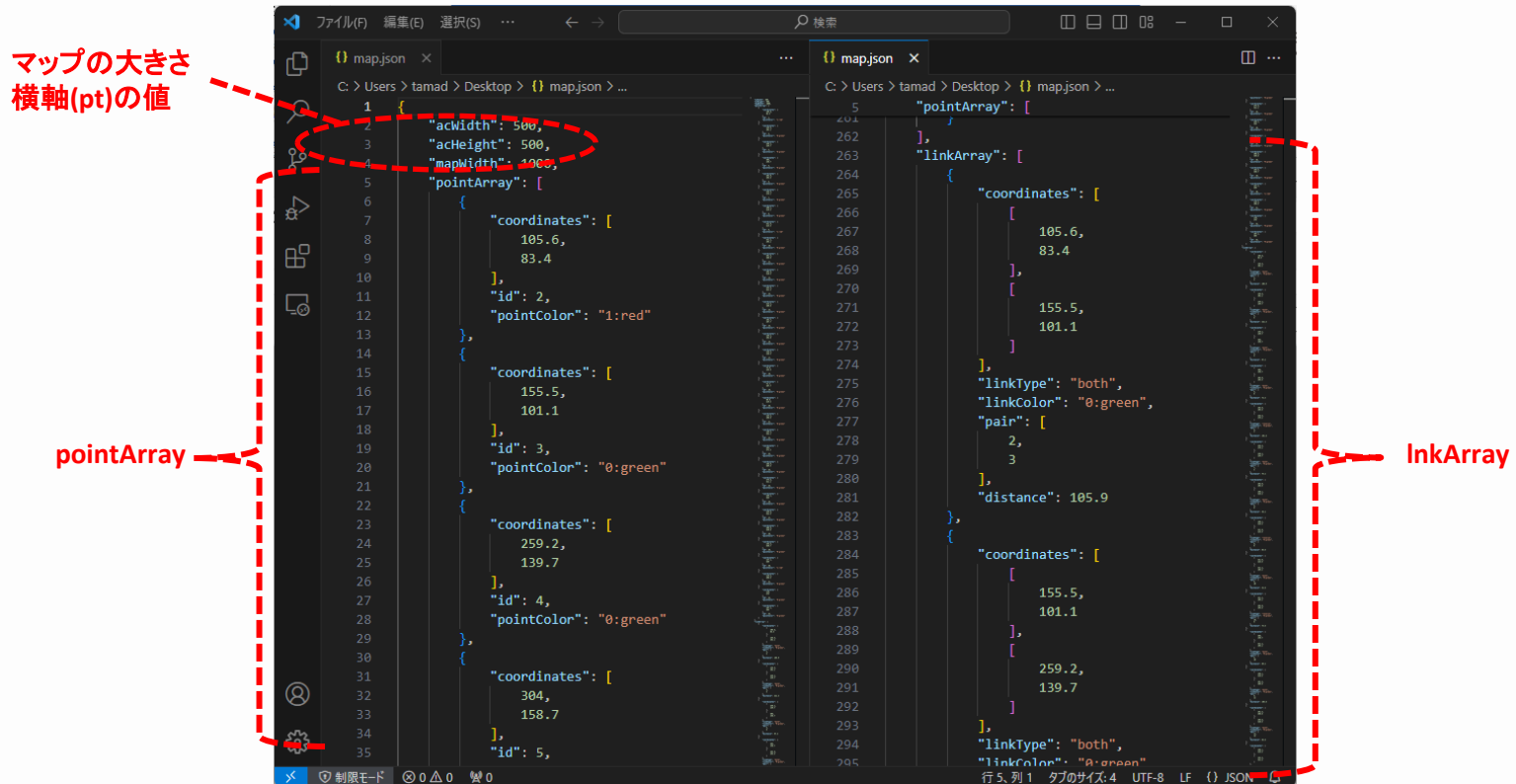
横幅(pt)を変更すると、画面上部の座標の縮尺が変更されます



横幅(pt)はx座標の最大値

作成したjsonファイルは、テキストエディタで編集することができます

- acWidth, acHeight : マップの大きさ（横幅, 縦幅） ※画像ファイルのサイズ
- mapWidth : 横軸(pt)で指定した値
- pointArray : ポイントの配列（座標、id、表示色）
- LinkArray : リンクの配列（2点の座標、色、リンクの太さ）





# 道路に沿って歩くモデルをつくろう

マップ上を与えられた経路で歩くモデルを作成します。  
下記のモデルは [Firefox](#) で実行することをおすすめします。

The screenshot shows the artisoc Cloud web interface. The main area displays a map of Shinjuku, Japan, with a red line indicating a walking route. The interface is divided into several sections:

- Control Panel:** Includes buttons for play, pause, and stop, along with a delay slider set to 0 ms.
- Console:** Displays a message: "Load module.. (decorator, pyparsing, setup, ols, cyclus, kiwisolver, numpy, python-dateutil, pytz, matplotlib, networkx)".
- Output Table:** Shows the results of the simulation.

要素名	出力値
経過時間 (sec)	215
到着人数	1

09. レシピブック サンプルモデル1

# ① モデルを定義する (1)

モデルツリーで「空間」「エージェント」「変数」を定義します

- モデルツリーの「Universe +」をクリックして、「空間を追加」を選択します。

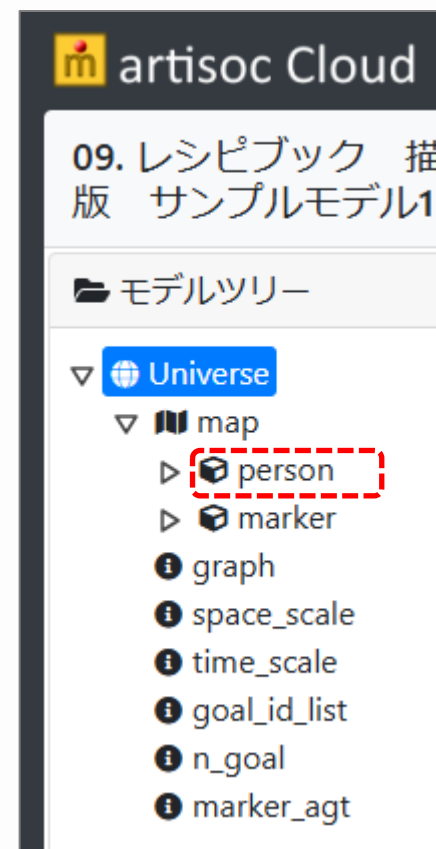
空間名 : map  
空間の大きさ X : 500  
空間の大きさ Y : 500  
ループする : チェックしない

- モデルツリーの「map +」をクリックして、「エージェント種別を追加」を選択します。

エージェント種別名 : person

- モデルツリーの「person +」をクリックします。

変数名 : color  
変数名 : speed  
変数名 : route  
変数名 : route\_count



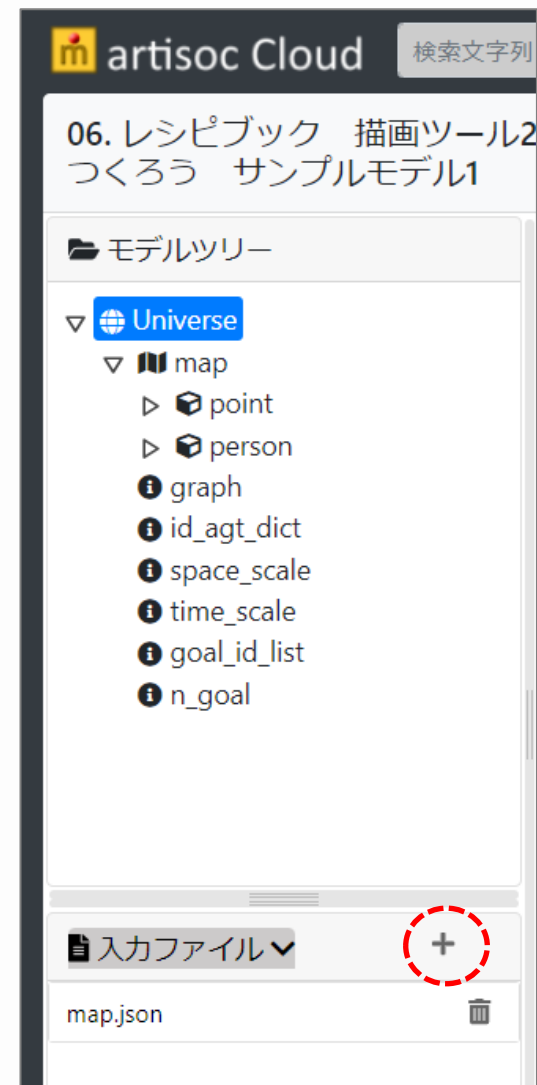
# ① モデルを定義する (2)

- モデルツリーの「map +」をクリックして、「エージェント種別を追加」を選択します。  
エージェント種別名 : marker
- モデルツリーの「Universe +」をクリックして、「変数を追加」を選択します。  
変数名 : graph  
変数名 : space\_scale  
変数名 : time\_scale  
変数名 : goal\_id\_list  
変数名 : n\_goal  
変数名 : marker\_agt



# ① モデルを定義する (3)

- 入力ファイルの「+」をクリックして、  
入力ファイルをインポートします。  
ファイル名： map.json ← **描画ツール2**で作成した  
jsonファイル



## ② エージェントのルールを記述する (1)

ネットワークファイル (map.json) で定義した道路に沿って歩きます

- モデルツリーの「Universe」をクリックしてルールエディタを表示します。

```
1 def univ_init(self):
2
3     Universe.space_scale = 3.88          # 地図の縮尺 (m/セル) ← 地図の縮尺で求めた値
4     Universe.time_scale = 1              # 時間刻み (sec/step)
5     self.file_read_graph('map.json')     # ネットワークの初期化
6
7     self.set_goal_id_list('red')          # ポイント色から目的地を格納
8     create_agt(Universe.map.person, num=10) # personを生成
9
10    Universe.n_goal = 0                   # 到着人数を初期化
11
12    Universe.marker_agt = create_agt(Universe.map.marker) # 移動マーカーの生成 ← 移動マーカーを生成
13
```

```
26 # ネットワークの初期化
27 def file_read_graph(self, one_filename):
28
29     import json
30     import networkx
31
32     # 変数の初期化
33     Universe.graph = networkx.DiGraph()
34
35     # jsonファイル読込
36     f = open(one_filename, 'r')
37     map_json_dict = json.load(f)
38     f.close()
39
40     self.init_point(map_json_dict)        # ポイントの初期化
41     self.init_link(map_json_dict)         # リンクの初期化
42
```



## ② エージェントのルールを記述する (2)

- モデルツリーの「Universe」をクリックしてルールエディタを表示します。

```
43 # ポイントの初期化
44 def init_point(self, map_json_dict):
45
46     import networkx
47
48     h = get_height_space(Universe.map) # 高さの取得
49
50     for one_feature in map_json_dict['pointArray']:
51         one_coordinates = one_feature['coordinates']
52         one_id = one_feature['id']
53         one_color = one_feature['pointColor'].split(':')
54
55         # graphに追加
56         Universe.graph.add_node(str(one_id))
57         Universe.graph.nodes[str(one_id)]['point_id'] = one_id
58         Universe.graph.nodes[str(one_id)]['x'] = float(one_coordinates[0])
59         Universe.graph.nodes[str(one_id)]['y'] = h - float(one_coordinates[1])
60         Universe.graph.nodes[str(one_id)]['color'] = self.get_rgb(one_color[1])
61
```

←ポイント属性をUnivesre.graph.nodesに格納

## ② エージェントのルールを記述する (3)

- モデルツリーの「Universe」をクリックしてルールエディタを表示します。

```
63 # リンクの初期化
64 def init_link(self, map_json_dict):
65
66     import networkx
67
68     for one_feature in map_json_dict['linkArray']:
69         one_coordinates_array = one_feature['coordinates']
70         one_link_type = one_feature['linkType']
71         one_link_color = one_feature['linkColor'].split(':')
72         one_color = self.get_rgb(one_link_color[1])
73         one_pair = one_feature['pair']
74         one_distance = one_feature['distance']
75
76         one_id = one_pair[0]
77         two_id = one_pair[1]
78
79         # graphとlink_listに追加
80         if one_link_type == 'both' or one_link_type == 'forward':
81             Universe.graph.add_edge(str(one_id), str(two_id), weight=one_distance)
82             Universe.graph.edges[str(one_id), str(two_id)]['color'] = one_color
83             Universe.graph.edges[str(one_id), str(two_id)]['distance'] = one_distance
84
85         if one_link_type == 'both' or one_link_type == 'reverse':
86             Universe.graph.add_edge(str(two_id), str(one_id), weight=one_distance)
87             Universe.graph.edges[str(two_id), str(one_id)]['color'] = one_color
88             Universe.graph.edges[str(two_id), str(one_id)]['distance'] = one_distance
89
90
```

←リンク属性をUnivesre.graph.edgesに格納

←両矢印／片矢印に対応

## ② エージェントのルールを記述する（４）

- モデルツリーの「Universe」をクリックしてルールエディタを表示します。

```
92 # 色名からRGB取得
93 def get_rgb(self, one_color):
94
95     if one_color=='green':
96         one_rgb = rgb(0, 128, 0)
97     elif one_color=='red':
98         one_rgb = rgb(255, 0, 0)
99     elif one_color=='blue':
100         one_rgb = rgb(0, 0, 255)
101     elif one_color=='yellow':
102         one_rgb = rgb(255, 255, 0)
103     elif one_color=='cyan':
104         one_rgb = rgb(0, 255, 255)
105     elif one_color=='magenta':
106         one_rgb = rgb(255, 0, 255)
107     else:
108         one_rgb = rgb(0, 0, 0)
109
110     return(one_rgb)
111
112
113 # 色名から goal_id_listを生成
114 def set_goal_id_list(self, one_color):
115
116     Universe.goal_id_list = []
117
118     for k, v in Universe.graph.nodes.items():
119         if v['color'] == self.get_rgb(one_color):
120             Universe.goal_id_list.append(k)
```

←ポイント属性の色名を使って色を塗る

←指定された色名で goal\_id\_listを作成

## ② エージェントのルールを記述する (5)

- モデルツリーの「person」をクリックしてルールエディタを表示します。

```
1 def agt_init(self):
2
3     self.color = rgb(randint(0, 255), randint(0, 255), randint(0, 255)) # 色をランダムに指定
4     self.speed = 1 # 移動速度 (m/s) の指定
5     self.speed = self.speed / Universe.space_scale * Universe.time_scale # 速度変換
6     self.route = []
7
8     # 出発地をランダムに決める
9     start_id = self.get_random_id()
10    one_node = Universe.graph.nodes[str(start_id)]
11    self.x = one_node['x']
12    self.y = one_node['y']
13
14    # 最も近い目的地を決めて経路を設定
15    self.route = self.get_route_from_goal_id_list(start_id)
16    self.route_count = 0
17
18
19 def agt_step(self):
20
21    # 経路に沿って移動
22    target_id = int(self.route[self.route_count])
23    target_node = Universe.graph.nodes[str(target_id)]
24    target_agt = Universe.marker_agt
25    target_agt.x = target_node['x']
26    target_agt.y = target_node['y']
27
28    distance = self.pursue(target_agt, self.speed)
29
30    while(distance > 0):
31        self.route_count += 1
32        if len(self.route) <= self.route_count:
33            # 目的地に到着したら終了
34            Universe.n_goal += 1
35            del_agt(self)
36            return
```

←出発地はランダムに決め  
目的地はgoal\_id\_listから  
最短のポイントを選択

## ② エージェントのルールを記述する（6）

- モデルツリーの「person」をクリックしてルールエディタを表示します。

```
46 # 目的地をランダムに決めて経路を設定
47 def set_random_route(self, start_id):
48
49     while(True):
50         goal_id = self.get_random_id()
51         if start_id != goal_id: # 出発地と目的地が同一でない場合
52             break
53
54     self.route = self.get_route(start_id, goal_id) # 最短経路を取得
55     self.route_count = 0
56
57 # 2点間の最短経路を取得
58 def get_route(self, start_id, goal_id):
59
60     import networkx as nx
61
62     try:
63         # ダイクストラ法を使って最短経路を探索
64         one_route = nx.dijkstra_path(Universe.graph, str(start_id), str(goal_id), weight='distance')
65
66         return(one_route)
67     except:
68         print('[error] get_route: point_id=', start_id, 'と', goal_id, 'がつながっているか確認してください')
69         exit_simulation()
70
71
72 # point_idをランダムに取得（但し、goal_id_list以外）
73 def get_random_id(self):
74
75     r = random.choice(list(Universe.graph.nodes.keys()))
76     while r in Universe.goal_id_list:
77         r = random.choice(list(Universe.graph.nodes.keys()))
78
79     return(r)
80
```

←目的地をランダムに設定  
(但し、出発地を除く)

←ダイクストラ法を使って  
2点間の最短経路を取得

←出発点をランダムに設定  
(但し、goal\_id\_listの  
目的地を除く)



## ② エージェントのルールを記述する（7）

- モデルツリーの「person」をクリックしてルールエディタを表示します。

```
81 # 2点間の最短経路の距離を取得
82 def get_route_length(self, start_id, goal_id):
83
84     import networkx as nx
85
86     try:
87         # ダイクストラ法を使って最短経路を探索
88         one_length = nx.dijkstra_path_length(Universe.graph, str(start_id), str(goal_id), weight='distance')
89
90         return(one_length)
91     except:
92         print('[error] get_route_length: point_id=', start_id, 'と ', goal_id, 'がつながっているか確認してください')
93         exit_simulation()
94
95 # 目的地リストの中から最も近い目的地までの最短経路を取得
96 def get_route_from_goal_id_list(self, start_id):
97
98     distance_dict = {}
99
100     for one_goal_id in Universe.goal_id_list:
101         distance_dict[one_goal_id] = self.get_route_length(start_id, one_goal_id)
102
103     distance_dict2 = sorted(distance_dict.items(), key=lambda x:x[1])
104     one_goal_id, one_length = distance_dict2[0]
105
106     one_route = self.get_route(start_id, one_goal_id)
107
108     return(one_route)
109
```

←ダイクストラ法を使って  
2点間の最短距離を取得

←goal\_id\_listの中から  
最も近い目的地を選択

### ③ 出力画面を定義する (1)

## マップ出力画面を定義します

- 出力画面を表示します。
- 出力パネル > 出力設定 > マップ出力 を選択し、「追加」をクリックします。  
マップ名: map  
空間: map  
固定画像: map.png ← **地理院地図から取得した画像ファイル**
- マップ要素リスト > エージェント「+」をクリックします。  
要素名: person  
出力対象: person  
マーカー  
選択: 矢印  
エージェント表示色:  
変数指定: color  
拡大率:  
固定値: 10



### ③ 出力画面を定義する (2)

- 出力パネル > 出力設定 > 値出力 を選択し、「追加」をクリックします。

出力名: 値出力

- 値出力要素リスト + をクリックします。

要素名: 経過時間 (sec)

出力対象: count\_step()

小数表示: 0 桁

要素名: 到着人数

出力対象: Universe.n\_goal

小数表示: 0 桁

- シミュレーションを実行します。

値出力設定

出力名: 値出力

値出力要素リスト

要素名	出力対象	小数表示
経過時間 (sec)	count_step()	0 桁
到着人数	Universe.n_goal	0 桁

Cancel OK

値出力要素設定

要素名: 経過時間 (sec)

出力値: count\_step()

小数表示: 0 桁

Cancel OK

値出力要素設定

要素名: 到着人数

出力値: Universe.n\_goal

小数表示: 0 桁

Cancel OK

# [Tips] point\_id毎の到着人数を知りたいとき

point\_id毎に到着人数を知りたい場合は、次のようにモデルを修正します。

- モデルツリーの「Universe +」をクリックして「変数を追加」を選択します。

変数名 :        n\_goal\_dict

- Universeルールに追加します。

univ\_init :

```
# point_id毎の到着人数を初期化
Universe.init_n_goal_dict()
```

univ\_step\_end :

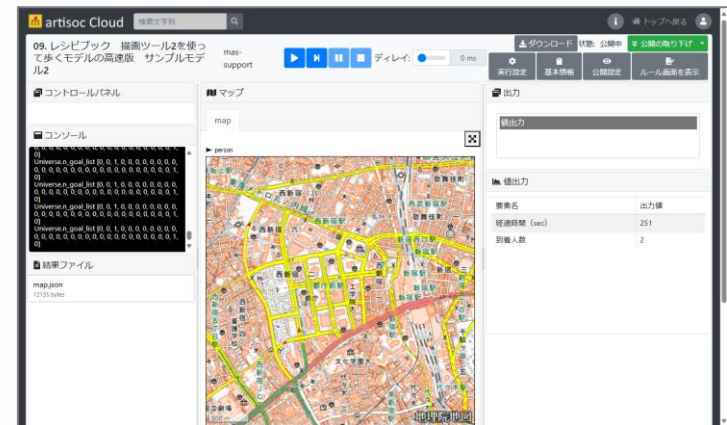
```
# point_id毎の到着人数を表示
print('Universe.n_goal_dict', Universe.n_goal_dict)
```

```
# point_id毎の到着人数を初期化
def init_n_goal_dict(self):
    Universe.n_goal_dict = {}
    for node_id in Universe.goal_id_list:
        Universe.n_goal_dict[int(node_id)] = 0
```

- personルールに追加します。

agt\_step :

```
「#目的地に到着したら終了」の後に追加
# point_id毎の到着人数を加算
Universe.n_goal_dict[target_id] += 1
```

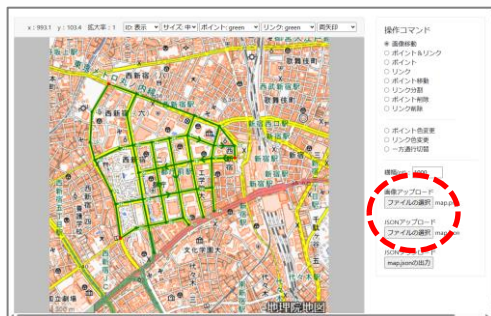


09. レシピブック サンプルモデル2

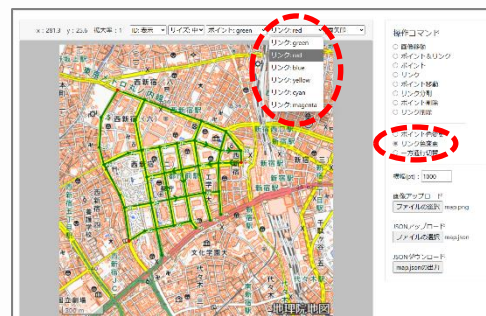
# [Tips] 任意の道路で減速したいとき (1)

任意の道路で減速したい場合は、描画ツールでリンク色を変更したあとに、personルールに減速する処理を追加します。

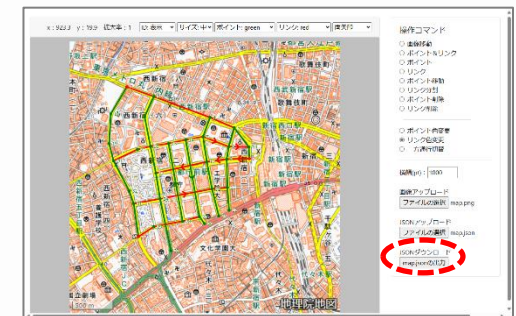
- 描画ツールを表示して、map.png と map.json を読み込みます。
- リンク色の選択プルダウンで「リンク：red」を選択し、操作コマンドの「リンク色変更」を選択します。
- 変更したいリンクの真ん中をクリックして赤色に変わります。
- 修正が完了したら「JSONでダウンロード」をクリックし、map2.json にリネームします。



map.pngとmap.jsonをアップロード



「リンク:red」と「リンク色変更」を選択



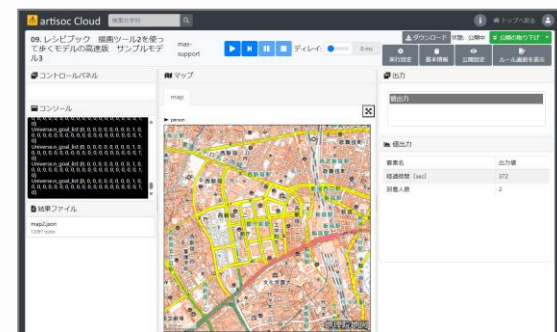
JSONをダウンロードし、map2.jsonにリネーム



# [Tips] 任意の道路で減速したいとき (2)

map2.json を入力ファイルに追加し、ルールを追加・更新します。

- 入力ファイルで「map.json」を削除し、「map2.json」を追加します。
- Universeルールを更新します。  
univ\_init :  
    # ネットワークの初期化  
    self.file\_read\_graph('map2.json')
- personルールを追加します。  
agt\_step :  
    「# リンク色が'red'の場合は減速」以下を追加



09. レシピブック サンプルモデル3

```
18
19 def agt_step(self):
20
21     # 経路に沿って移動
22     target_id = int(self.route[self.route_count])
23     target_node = Universe.graph.nodes[str(target_id)]
24     target_agt = Universe.marker_agt
25     target_agt.x = target_node['x']
26     target_agt.y = target_node['y']
27     # リンク色が'red'の場合は減速
28     if self.route_count > 0:
29         last_id = int(self.route[self.route_count - 1])
30         link_color = Universe.graph.edges[str(last_id), str(target_id)]['color']
31         if link_color == Universe.get_rgb('red'):
32             speed_rate = 0.3
33         else:
34             speed_rate = 1.0
35     else:
36         speed_rate = 1.0
37
38     distance = self.pursue(target_agt, self.speed * speed_rate)
```

リンク色が'red'の場合は、  
speed\_rateを0.3に設定して減速

「person」がポイントから出発するのではなく、リンク上から出発したい場合、経路情報を使って出発地と次のポイントまでのリンク上にランダムに配置します。

- personルールに追加します。

agt\_init :

「# 最も近い目的地を決めて経路を設定」の後に追加

# リンク上に配置

```
target_id = int(self.route[self.route_count + 1])
```

```
target_node = Universe.graph.nodes[str(target_id)]
```

```
target_agt = Universe.marker_agt
```

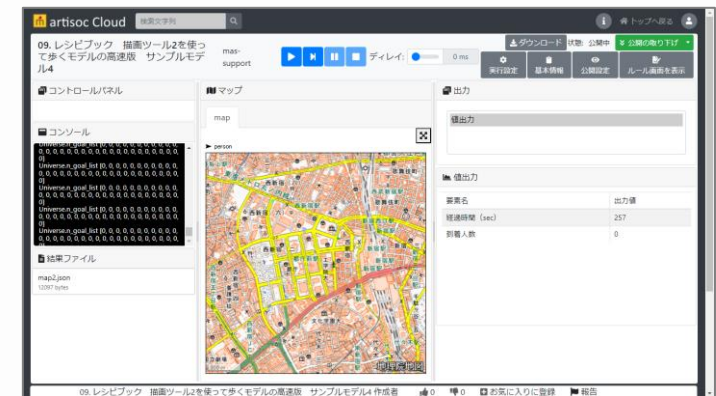
```
target_agt.x = target_node['x']
```

```
target_agt.y = target_node['y']
```

```
d = measure_agt_distance(self, target_agt)
```

```
self.turn_agt(target_agt)
```

```
self.pursue(target_agt, d * rand())
```



09. レシピブック サンプルモデル4