



artisoc Cloudレシピブック

14. ひとつのモデルを複数回実行するモデルを作ろう

(株) 構造計画研究所

(株) PARA-SOL

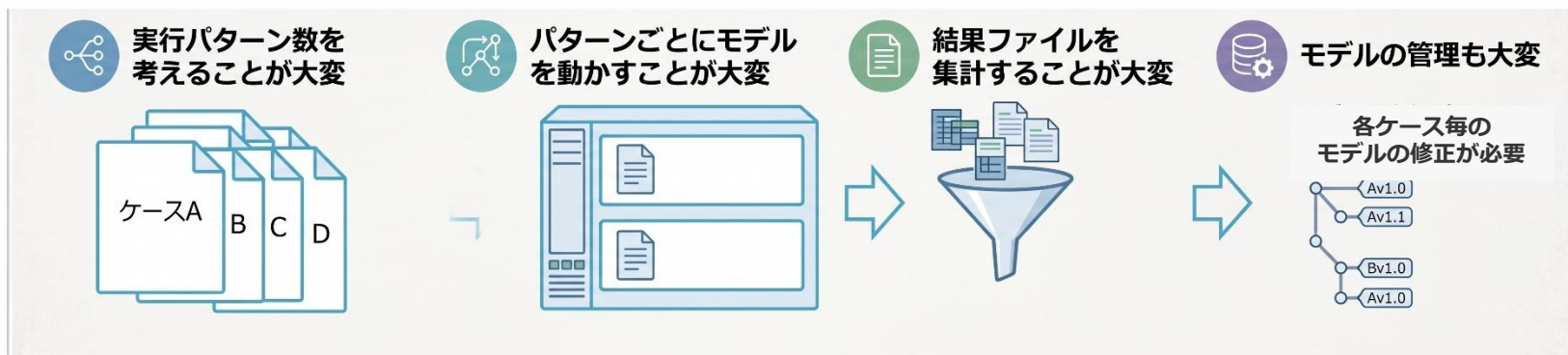
<https://mas.kke.co.jp>

こんなことはありませんか？

モデルを作って色々な事象のシミュレーションを行いたい...しかし、

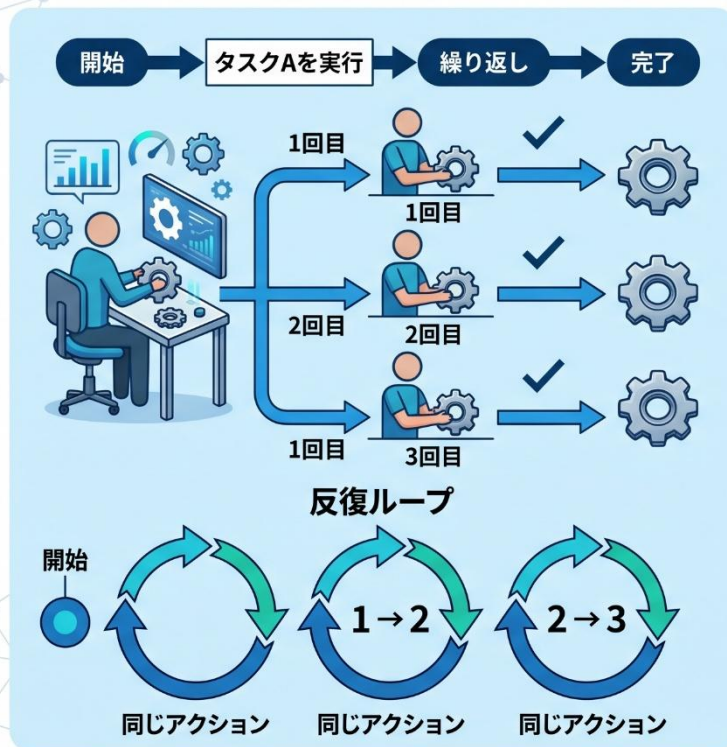
- 実行パターン数を考えることが大変。
- パターンごとにモデルを動かすことが大変。
- 結果ファイルを集計することが大変。

パターンごとにモデルを分けることもあるので、
モデルの管理も大変だったりします。

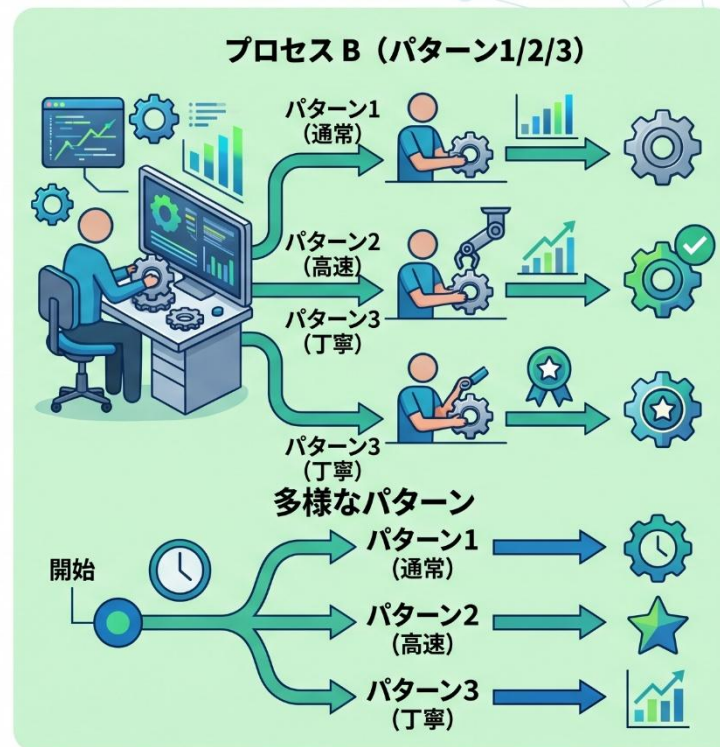


色々なパターンのシミュレーションをまとめて実行できたらいいな...。
ということで、今回は連続で複数回シミュレーションを実行するモデルを作ってみました。

同じことを繰り返す

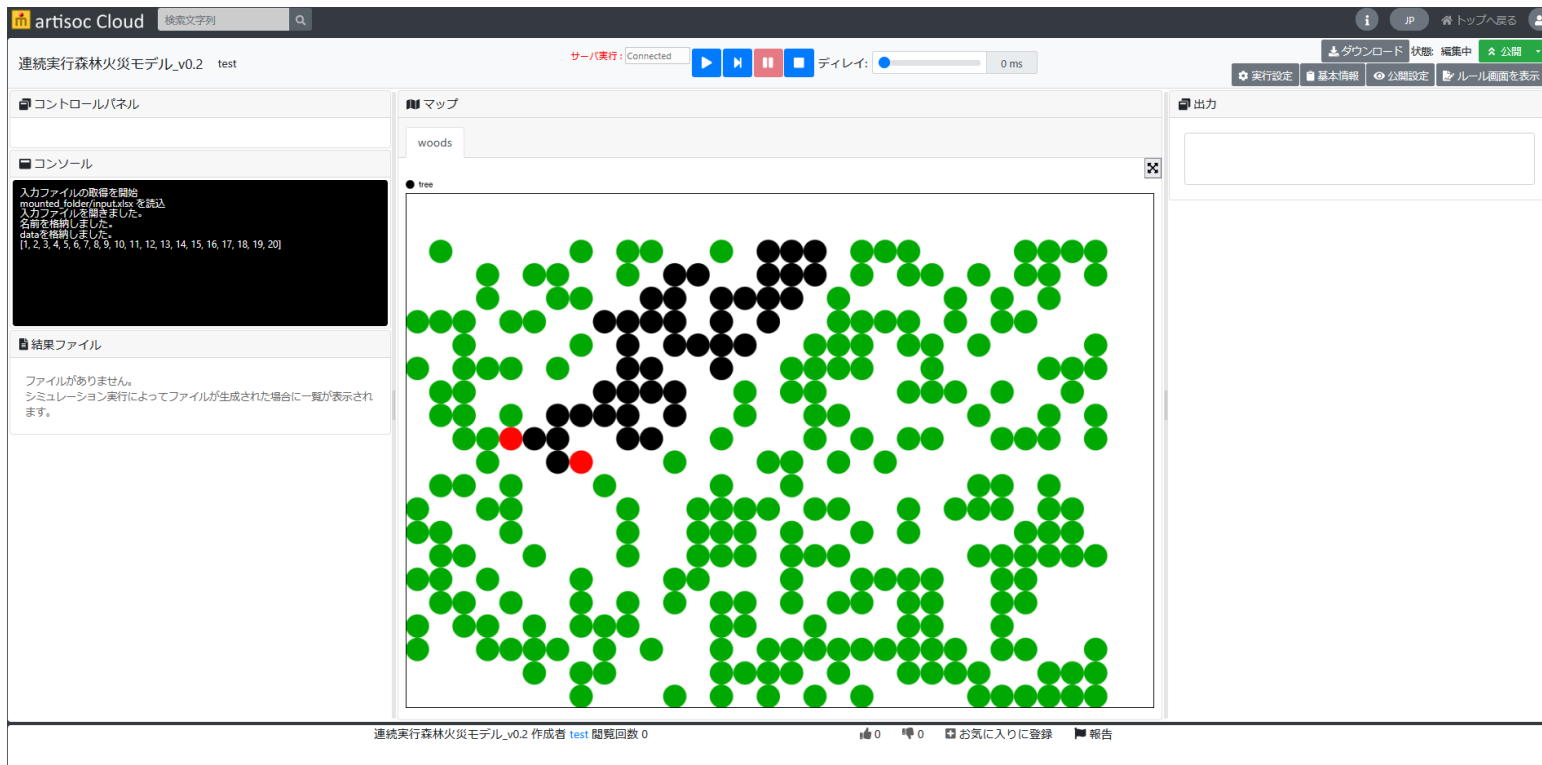


様々なパターンで実行



連続実行モデルを作ろう。

- 今回は「森林火災モデル」を基に、シミュレーションを複数回実行するモデルを作成します。
- このモデルを基に、様々なモデルを複数回実行可能なモデルに改造可能です。



- 木エージェントに火が燃え移り、燃え尽きていく様子を表しています。
- 木と空地の割合に応じて、どれくらい火が燃え広がるか複数のパターンで検証していきます。

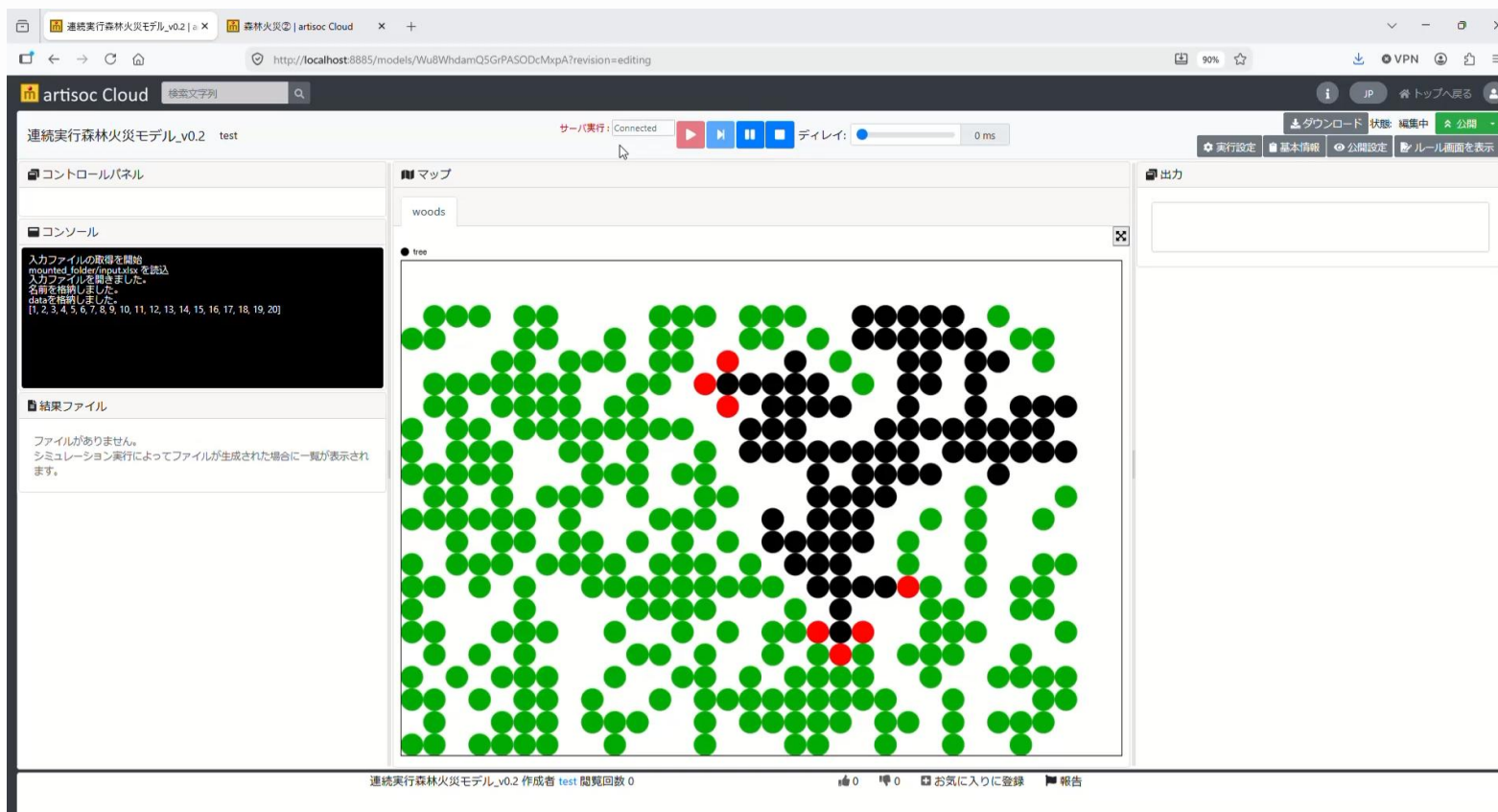


モデルの解説は「[第2部：人工社会の発想と技法に慣れる](#)」第19章：同期問題に注意する」を参照ください。
ベースのモデルは「[森林火災 過去の状態を参照](#)」のモデルを参照ください。

まず動かしてみよう

- 決められた木の割合を基に、マップのランダムな位置にエージェントを配置します。
- エージェントの配置完了後、ランダムな一つのエージェントから発火します。
- 一度のシミュレーションは、発火したエージェントの縦横隣のエージェントに火が燃え移り、隣接する木が全て燃えてしまうまで続きます。
- 入力ファイルから複数のパターンの実行を設定し、連続で実行しています。

※サンプルモデルは[こちら](#)



- 木のルール

- シミュレーション開始時にランダムな位置にエージェントを配置します。
- 配置後、ランダムな一本の木が発火します。
- 発火した木の色は赤に変化し、次のステップで黒に変化します。
- 前のステップで赤色のエージェントが縦横に隣接している場合、そのエージェントも発火し、赤色に変化する。
- 隣接するすべての木を焼き尽くし、赤いエージェントがいなくなった時点でシミュレーションを終了する。

モデルの作り方

1. 入力ファイルを定義する
2. モデルを定義する
3. Universeのルールを記述する
4. エージェントのルールを記述する
5. オンプレミス版artisoc Cloudで作成する

①入力ファイルを定義する

① 入力ファイルを定義する

- モデルの変化をつけて、試したい項目を基に入力ファイルに定義する値を設定します。
- 新規csvファイル「input.csv」を作成します。
- 今回は、マップに配置する「木の割合」のみを定義してみます。

※複数の項目を指定し、項目ごとに考えられる組み合わせ全てシミュレーションを実行することも可能です。

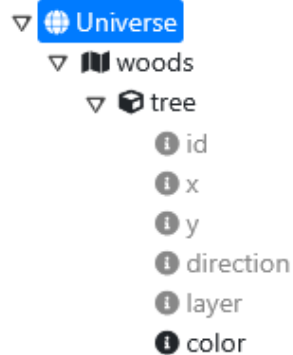
	A	B	C
1	木の割合		
2	0.05		
3	0.1		
4	0.15		
5	0.2		
6	0.25		
7	0.3		
8	0.35		
9	0.4		
10	0.45		
11	0.5		
12	0.55		
13	0.6		
14	0.65		
15	0.7		
16	0.75		
17	0.8		
18	0.85		
19	0.9		
20	0.95		
21	1		

②モデルを定義する

② 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



- 空間 (Universe.woods)
連続空間、大きさ30×20、レイヤ数1、ループしないで作成
- エージェント (Universe.woods.trees)
エージェントの記憶数を1に設定ください。
 - エージェント変数
color: エージェントの色を定義します。

※他モデルで移植する場合は、元のモデルのエージェント、空間、変数を定義ください。

② モデルを定義する (2)

Universeの「変数」を定義します。

モデルツリー

Universe

woods

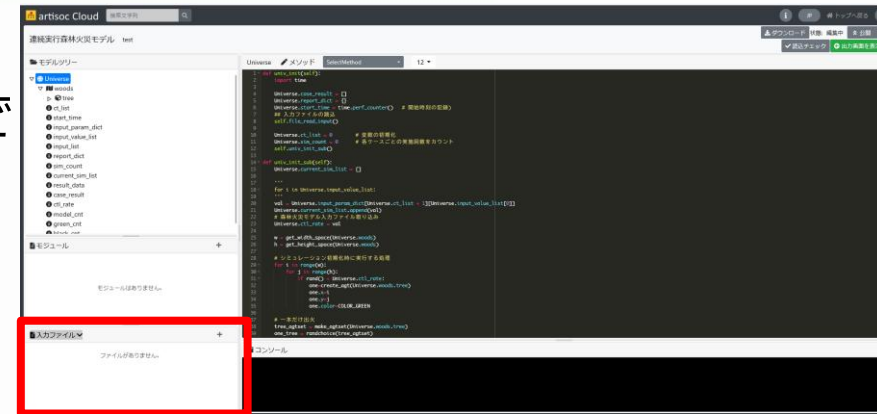
- ct_list
- start_time
- input_param_dict
- input_value_list
- input_list
- report_dict
- sim_count
- current_sim_list
- result_data
- case_result
- ctl_rate
- green_cnt
- black_cnt

- ct_list :実行中のケースを管理
- start_time :モデルの開始時刻を取得
- input_param_dict :ケースごとの変数を管理
- input_value_dict :各ケース名を管理
- input_list :ケースの実行順を管理
- report_dict :レポート出力用の結果
- sim_count :シミュレーション実行回数
- current_sim_list :実行中ケースの変数を保存
- result_data :シミュレーション実行結果
- case_result :ケース毎の実行結果の保存
- ctl_rate :木の延焼率
- green_cnt :燃えていない木の数
- black_cnt :燃えた木の数

こちらの3つの変数は今回の森林火災モデルでのみ使用する変数です。

② モデルを定義する (3)

- クラウド版artisoc Cloud(academicライセンス、standardライセンス)である場合、モデルのルール編集画面の入力ファイルに「input.csv」をアップロードください。



③ Universeのルールを記述する

③ Universeのルールを記述する（1）

- univ_init

- シミュレーションの初期化と入力ファイルの取得を行います。
- シミュレーション全体で一度だけ実行し、他のモデルに流用する際も基本的に変更する必要はありません。（シミュレーションの始めに初期化したいリストや辞書がある場合は追加ください）

```
1 def univ_init(self):
2     import time
3
4     Universe.case_result = {} # 各ケースごとの結果を初期化する
5     Universe.report_dict = {} # レポート出力用のリストを初期化する。
6     Universe.start_time = time.perf_counter() # (開始時刻の記録)
7     ## 入力ファイルの読込
8     self.file_read_input()
9
10    Universe.ct_list = 0 # 変数の初期化
11    Universe.sim_count = 0 # 各ケースごとの実施回数をカウント
12    self.univ_init_sub() # 各ケース毎の初期化を行う
```

③ Universeのルールを記述する (2)

- file_read_input
 - input.xlsxの値を取得し、input_param_dictに取り込みます。
 - 複数の項目を設定していた場合、各項目に対して、すべての組み合わせに関してパターンを作成します。
 - 入力ファイルの一行目には項目名が入りその値をモデルでも使用しています。「input_value_list」に保存されており、出力時やシミュレーション実行中に使用することが可能です。

```
141 def file_read_input(self):
142     import pandas as pd
143     import itertools
144
145     Universe.input_param_dict = {}
146     Universe.input_value_list = []
147     Universe.input_list = []
148
149     print("入力ファイルの取得を開始")
150     input_filename = "input.csv"
151
152     try:
153         print(f"{input_filename} を読み込")
154
155         # CSVの読み込み (エンコーディングは環境に応じて utf-8 や shift_jis に変更してください)
156         df = pd.read_csv(input_filename)
157
158         print("入力ファイルを開きました。")
159
160         # 列名 (項目名) を取得
161         Universe.input_value_list = df.columns.tolist()
162         print("名前を格納しました。")
163
164         # 各列のデータ (None / NaN を除外) を辞書に格納
165         data_by_col = {}
166         for col in df.columns:
167             # dropna() で空セルを除外してリスト化
168             val_list = df[col].dropna().tolist()
169             if val_list:
170                 data_by_col[col] = val_list
171
172         print("dataを格納しました。")
173
174         # itertools.product で全パターンの組み合わせを自動生成し、Dictに格納
175         keys = list(data_by_col.keys())
176         values = list(data_by_col.values())
177
178         Universe.input_param_dict = {}
179         Universe.input_list = []
180
181         for key, combination in enumerate(itertools.product(*values), start=1):
182             Universe.input_param_dict[key] = dict(zip(keys, combination))
183             Universe.input_list.append(key)
184
185         print(Universe.input_list)
186         # print(Universe.input_param_dict)
187
188     except Exception as e:
189         print(f"[Error] file_read_input: {input_filename}")
190         print(f"詳細: {e}")
191         exit_simulation()
192
```

③ Universeのルールを記述する (3)

- univ_init_sub
 - 一度シミュレーションが終了するごとに初期化を行う。
 - 他モデルからの移植を行う場合、univ_initに記載している内容をここで行うとよい。

```
14 def univ_init_sub(self):
15     Universe.current_sim_list = []
16
17     val = Universe.input_param_dict[Universe.ct_list + 1][Universe.input_value_list[0]]
18     Universe.current_sim_list.append(val)
19     # 森林火災モデル入力ファイル取り込み
20     Universe.ctl_rate = val
21
22     w = get_width_space(Universe.woods)
23     h = get_height_space(Universe.woods)
24
25     # シミュレーション初期化時に実行する処理
26     for i in range(w):
27         for j in range(h):
28             if rand() < Universe.ctl_rate:
29                 one=create_agt(Universe.woods.tree)
30                 one.x=i
31                 one.y=j
32                 one.color=COLOR_GREEN
33
34     # 一本だけ出火
35     tree_agtset = make_agtset(Universe.woods.tree)
36     one_tree = randchoice(tree_agtset)
37     one_tree.color=COLOR_RED
```

③ Universeのルールを記述する（4）

- univ_step_begin, univ_step_end
 - 各ステップ毎に実行する処理を記述する。
 - 他モデルから移植する場合、基本的には同様の処理を記述する。
 - モデルの終了条件を満たし、exit_simulation()を呼び出している箇所があれば、そこを63,64行目を参照し、「univ_finish_sub」, 「univ_init_sub」を呼び出すように修正する。

```
39 def univ_step_begin(self):
40     pass
41
42
43 def univ_step_end(self):
44     # シミュレーションのステップ終了毎に実行する処理
45
46     # 火のチェック
47     red_flag = False
48     tree_agtset = make_agtset(Universe.woods.tree)
49     for one_tree in tree_agtset:
50         if one_tree.color == COLOR_RED:
51             red_flag = True
52             break
53
54     if red_flag == False:
55         # 火元がない
56         for one_tree in tree_agtset:
57             if one_tree.color == COLOR_BLACK:
58                 Universe.black_cnt += 1
59             else:
60                 Universe.green_cnt += 1
61     Universe.case_result.append(Universe.black_cnt)
62     Universe.case_result.append(Universe.green_cnt)
63     self.univ_finish_sub()
64     self.univ_init_sub() # シミュレーション結果の初期化
```

③ Universeのルールを記述する (5)

- univ_finish_sub
 - シミュレーションが一度終了した場合、ログの出力と画面の初期化を行う。
 - 同じモデルを複数回実行し、結果の分布を調査したい場合、sim_countが実行回数を満たすまで次のケースに入らないようにする。
(今回は10回実行するようにしている)
 - マップ上のエージェントを取得し、それらすべてを削除します。

今回はログの出力をケースごとに実行しています。
この処理を実行するタイミングをずらすことで、一度の試行ごとにログの出力を行うことも可能です。

```
80 - def univ_finish_sub(self):  
    if Universe.sim_count > 10:  
        self.output_log()      # ログ出力処理 (csv)  
        Universe.case_result = □  
        Universe.ct_list += 1   # カウンタ + 1  
        Universe.sim_count = 0  
        if len(Universe.input_list) <= Universe.ct_list:  
            exit_simulation()   # 全ケース実行したら終了  
        Universe.sim_count += 1  
    tree_agtset = make_agtset(Universe.woods.tree)  
    for one_tree in tree_agtset:  
        del_agt(one_tree)  
    Universe.green_cnt = 0  
    Universe.black_cnt = 0  
    96
```

③ Universeのルールを記述する (6)

- output_log
 - 1つのケースのシミュレーションに対して、実行結果を出力しています。
 - 115行目にて、ログのヘッダー部分を設定しています。
 - データの書き込みは、今回1列目に木の割合、2列目に結果を出力している。他モデルで使用する場合は、任意の出力に変更ください。

```
101 def output_log(self): # ログ出力処理 (csv & ZIP)
102     import csv
103     import io
104     import zipfile
105
106     case_str = f"{Universe.ct_list:02d}"
107     zip_filename = f"case{case_str}_output.zip"
108     inner_csv_name = f"output.csv"
109
110     # 1. メモリ上で CSV データを作成
111     csv_buffer = io.StringIO()
112     writer = csv.writer(csv_buffer)
113
114     # case_name = f"case{case_str}"
115     case_name = f"{round(Universe.current_sim_list[0] * 100)}%" # カラム名を%で出力
116
117     # ヘッダー書き込み (リスト形式にする)
118     writer.writerow(['燃えた', '燃えてない'])
119
120     # 10個の試行結果をそのままリストとして保存
121     # (report_dict には case_name: [val1, val2, ...] の形で格納)
122     Universe.report_dict[case_name] = list(Universe.case_result)
123
124     # データ行書き込み
125     for i in range(0, len(Universe.case_result), 2):
126         writer.writerow([Universe.case_result[i], Universe.case_result[i+1]])
127
128     # 2. メモリ上の CSV 文字列を取り出して ZIP に直接書き込む
129     zip_buffer = io.BytesIO()
130     with zipfile.ZipFile(
131         zip_buffer, mode="w", compression=zipfile.ZIP_DEFLATED
132     ) as zf:
133         zf.writestr(inner_csv_name, csv_buffer.getvalue().encode("utf-8-sig"))
134
135     # 3. 完成した ZIP ファイルのみを出力
136     with open(zip_filename, mode="wb") as f:
137         f.write(zip_buffer.getvalue())
138
```

③ Universeのルールを記述する (7)

- univ_finish
 - レポートを作成・出力後、モデル実行時間を出力し終了。

```
66 ▾ def univ_finish(self):  
67     import time  
68  
69     self.output_report()  
70  
71     # 終了時間を記録して差分を計算  
72     end_time = time.perf_counter()  
73     execution_time = end_time - Universe.start_time  
74  
75     out_str = f"実行時間: {execution_time:.1f} 秒"  
76     print(out_str)  
77
```

③ Universeのルールを記述する (8)

- output_report
 - 215,216行で各項目の名前を設定しています。
 - 223~232行目にレポートを出力しています。
今回は、1列目に各シミュレーションの木の割合、2列目に、木の燃焼率を全ての実行結果について縦向きに出力しています。
 - レポートの出力形式を変更する場合はこちらを修正ください。

```
194 def output_report(self):
195     import csv
196
197     summary_path = "report.csv"
198
199     # データが空でないかチェック
200     if not Universe.report_dict:
201         print("レポート対象のデータがありません。")
202         return
203
204     # CSVファイルの書き込み (utf-8-sig を指定すると Excel で開いても文字化けしません)
205     with open(summary_path, mode="w", newline="", encoding="utf-8-sig") as f:
206         writer = csv.writer(f)
207
208         # ヘッダーの書き込み
209         writer.writerow(["木の割合", "燃焼率"])
210
211         # データの書き込み
212         for case_name, result_list in Universe.report_dict.items():
213             for i in range(0, len(result_list), 2):
214                 burned = result_list[i]
215                 not_burned = result_list[i + 1]
216
217                 # ゼロ除算エラーを防ぐための安全処理 (合計が0の場合は0%とする)
218                 total = burned + not_burned
219                 burnedrate = (burned / total * 100) if total > 0 else 0
220
221                 # 1行分を追加
222                 writer.writerow([case_name, burnedrate])
223
224     print("レポート (report.csv) の出力が完了しました。")
```

④ エージェントのルールを記述する

④ エージェントのルールを記述する (1)

- tree

- 2ステップ目以降、自身の隣接するエージェントの色が赤い場合、自身の色を赤くほんこうする。
- 1ステップ前に自身の色が赤である場合、自身の色を黒に変更する。

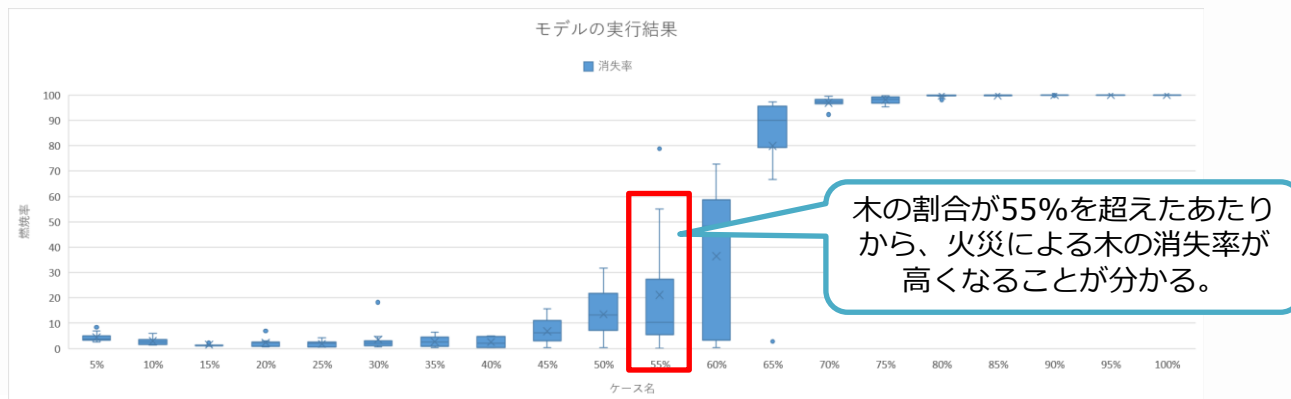
※他のモデルを移植する場合、こちらはそのままエージェントのルールを記述ください

```
1 def agt_init(self):
2
3     # エージェントの初期化处理
4     pass
5
6 def agt_step(self):
7
8     # シミュレーション開始前のステップを指定するとエラーになるので
9     # 最初のステップは何もせずに関数を抜ける
10    if count_step() <= 1:
11        return
12
13    # 2ステップ目以降であれば
14    past1=get_history(self, 1) # 1ステップ前の自分をpast1にセット
15
16    if past1.color==COLOR_GREEN: # 1ステップ前の色が緑であれば
17
18        neighbor=self.make_agtset_around_own(1,False)
19
20        for one in neighbor:
21
22            # 周囲のエージェントの1ステップ前の状態をpast2にセット
23            past2=get_history(one,1)
24
25            # 1ステップ前が赤であれば、自分を赤に
26            if past2.color==COLOR_RED:
27                self.color=COLOR_RED
28                break
29        else:
30            self.color=COLOR_BLACK
```

- モデルを実行してみよう
 - モデルの実行画面を開き、再生ボタンをクリックし、シミュレーションを実行します。
- シミュレーション終了後
 - 実行画面左下「結果ファイル」にファイルが出力します。必要なファイルのファイル名をクリックすることでダウンロードできます。



- モデルを実行後、report.csvを基に、箱ひげ図を作ってみましょう。
 - 「ログ集計分析.xlsx」を開き、「input」シートを開きます。
※「ログ集計分析.xlsx」は「ファイルのダウンロード」よりダウンロードください
 - モデル実行後に取得した「report.csv」を開き、項目名も含めデータをすべて選択しコピーします。
 - 「ログ集計分析.xlsx」の「input」シートの「A1」セルに貼り付けます。
 - データを貼り付け後、「ログ集計分析.xlsx」の「output」シートを開きます。
 - 貼り付けたデータを基に、箱ひげ図が作られます。



⑤ オンプレミス版artisoc Cloudで作成する

⑤ オンプレミス版artisoc Cloudで作成する

• オンプレミス版artisoc Cloud

- 通常artisoc Cloudのモデルの入力ファイル・結果ファイルはブラウザの画面上で設定・保存を行いますが、オンプレミス版artisoc Cloudでは、ファイルの出力先を任意のフォルダに変更することが可能です。
- クラウド版に関しては、standardライセンスのみ、有償で共有フォルダにファイルを出力することが可能です。
- オンプレミス版artisoc Cloudを使用し、任意のフォルダにファイルを出力する際は、p18,23,25で記載している入出力ファイルの参照先のパスを変更ください。

P18,150行目

```
148 print('入力ファイルの取得を開始')
149 input_filename = 'input.xlsx'
150 key = 1
```



```
input_filename = "mounted_folder/input.csv"
```

P22,107行目

```
106 case_str = f"{Universe.ct_list:02d}"
107 zip_filename = f"case{case_str}_output.zip"
108 inner_csv_name = f"output.csv"
109
```



```
103 case_str = f"{Universe.ct_list:02d}"
104 zip_filename = f"mounted_folder/output-folder/case{case_str}_output.zip"
105 inner_csv_name = f"output.csv"
```

P24,197行目

```
197 summary_path = "report.csv"
198
```

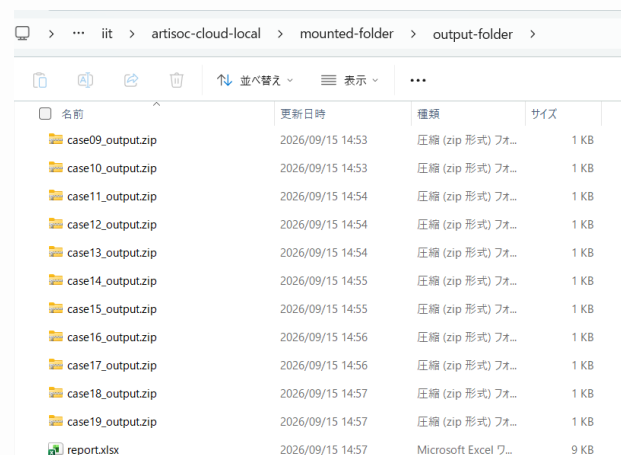
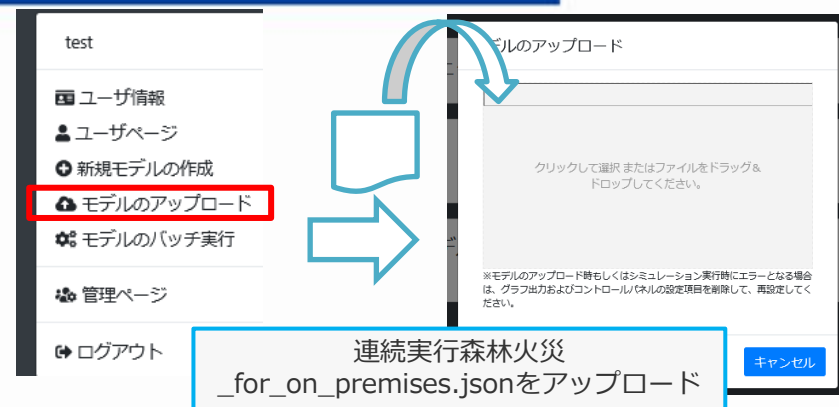


```
= "mounted_folder/output-folder/report.csv"
```

※出力されるファイルの中身に違いはありません。

実行してみよう

- モデルをアップロードする。
 - オンプレミス版artisoc Cloudを起動し、メニューボタンの「モデルのアップロード」からモデルファイルをアップロード
- モデルを実行してみよう
 - モデルの実行画面を開き、再生ボタンをクリックし、シミュレーションを実行します。
- シミュレーション終了後
 - 指定したフォルダ内に、結果ファイルが出力されます。



- 入力ファイルで設定する項目数を増やしてみよう
(燃焼率や、木の燃焼範囲)
- 任意のモデルを移植し、複数回実行してみよう。
- 入力ファイルで考えられる値を設定し、様々なパターンでモデルを実行してみよう。