



artisoc Cloudレシピブック

13. 複数のエージェントが連携するモデルを作成しよう

(株) 構造計画研究所

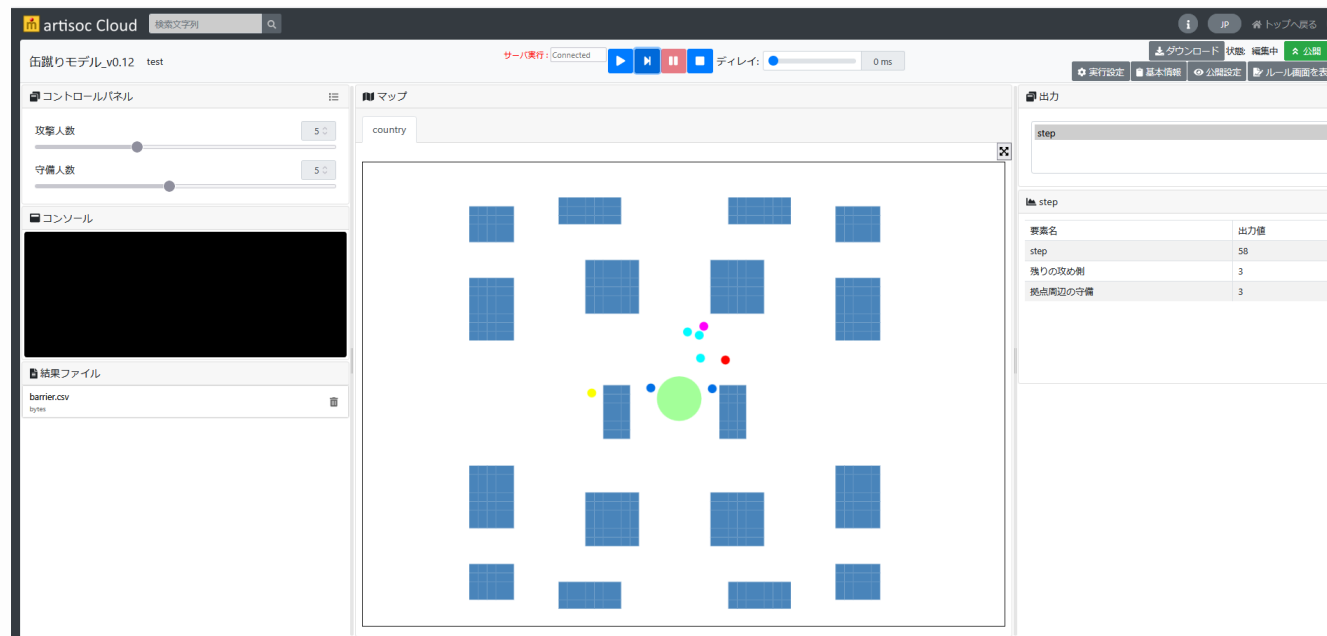
(株) PARA-SOL

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

- 缶蹴りとは？
 - オニは中央の缶を守りながら隠れた人を探し、隠れた人はオニの隙を突いて缶を蹴り、捕まった仲間を救出することを目指す遊びです。
 - 全員の解放を狙う隠れる側と、缶を守り抜くオニ側の戦略的な駆け引きが魅力の伝統的な陣取りゲームです。



1. エージェントは攻め側（隠れる人）と守り側（オニ）の二つのチームに分け、攻め側は守り側を避けながら陣地を目指します。
2. 守り側は攻め側が陣地に入らないように守ります。
3. 攻め側は守り側に振れられたら消えてしまいます。
4. 攻め側が陣地の中に入ったら攻め側の勝利、攻め側の数が1以下になったら守り側の勝利です。



まず動かしてみよう

- 実際にモデルを動かして動きを見てみましょう。

artisoc Cloud

伍蔵りモデル_v0.12 test

サーバー実行: Connected

プレイボタン

デレイ: 160 ms

コントロールパネル

攻撃人数: 5

守備人数: 5

コンソール

守備側の勝利
シミュレーションを終了しました。

結果ファイル

barrier.csv

マップ

country

出力

要素名	出力値
step	59
残りの攻め側	1
拠点周辺の守備	2

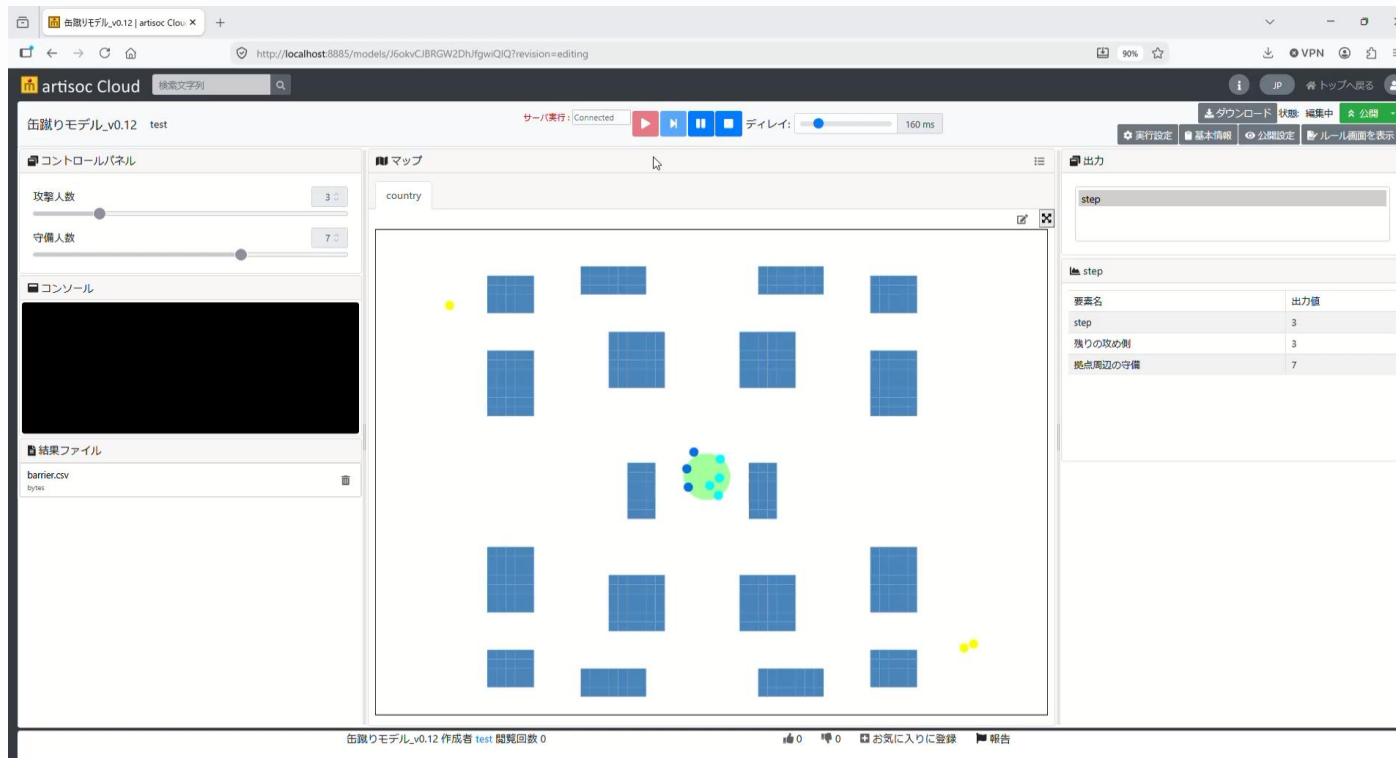
伍蔵りモデル_v0.12 作成者 test 閲覧回数 0

お気に入り登録

報告

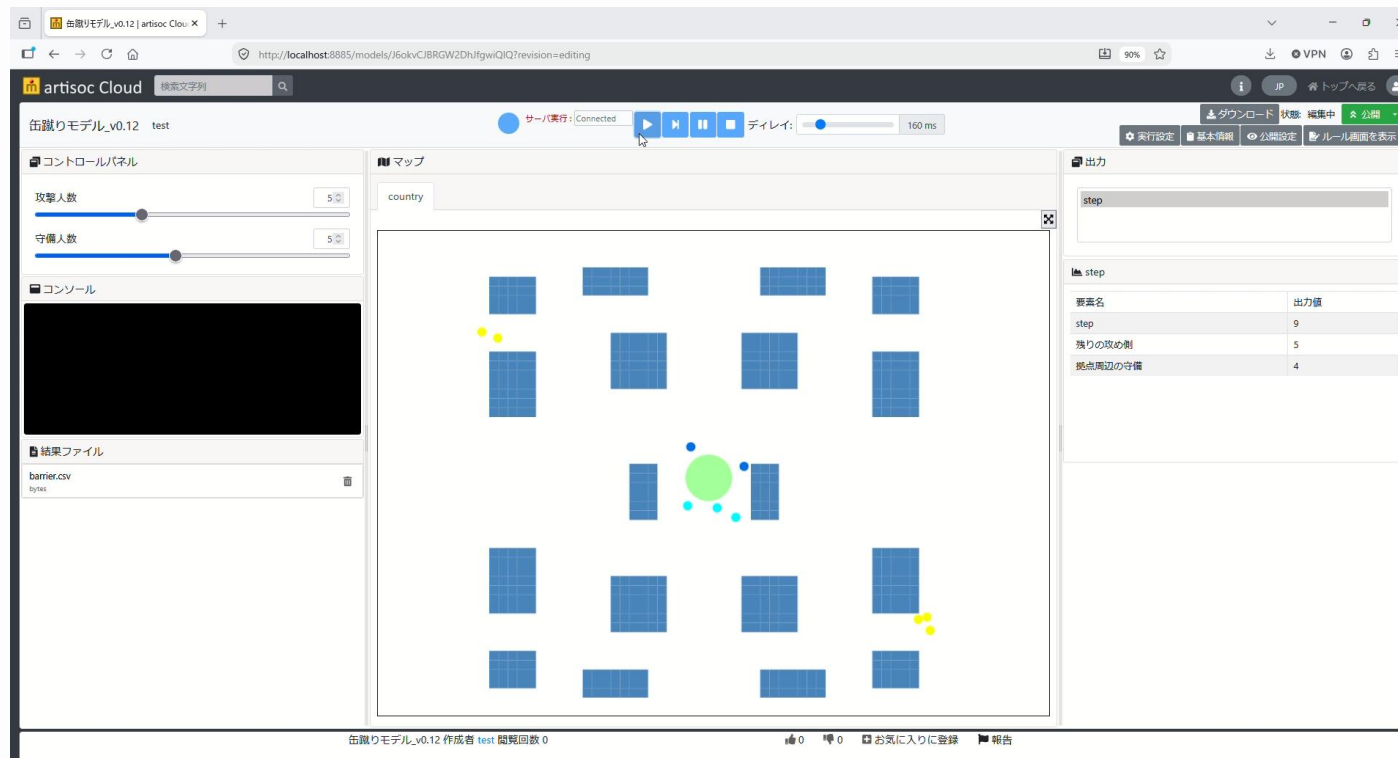
コンパネを操作してみる（1）

- コンパネの値を変更することで攻め側、守り側の数を変更することが出来ます。
- 攻めと守りの数を3:7にした場合
 - 守りの数が多く攻めに小さく見える。



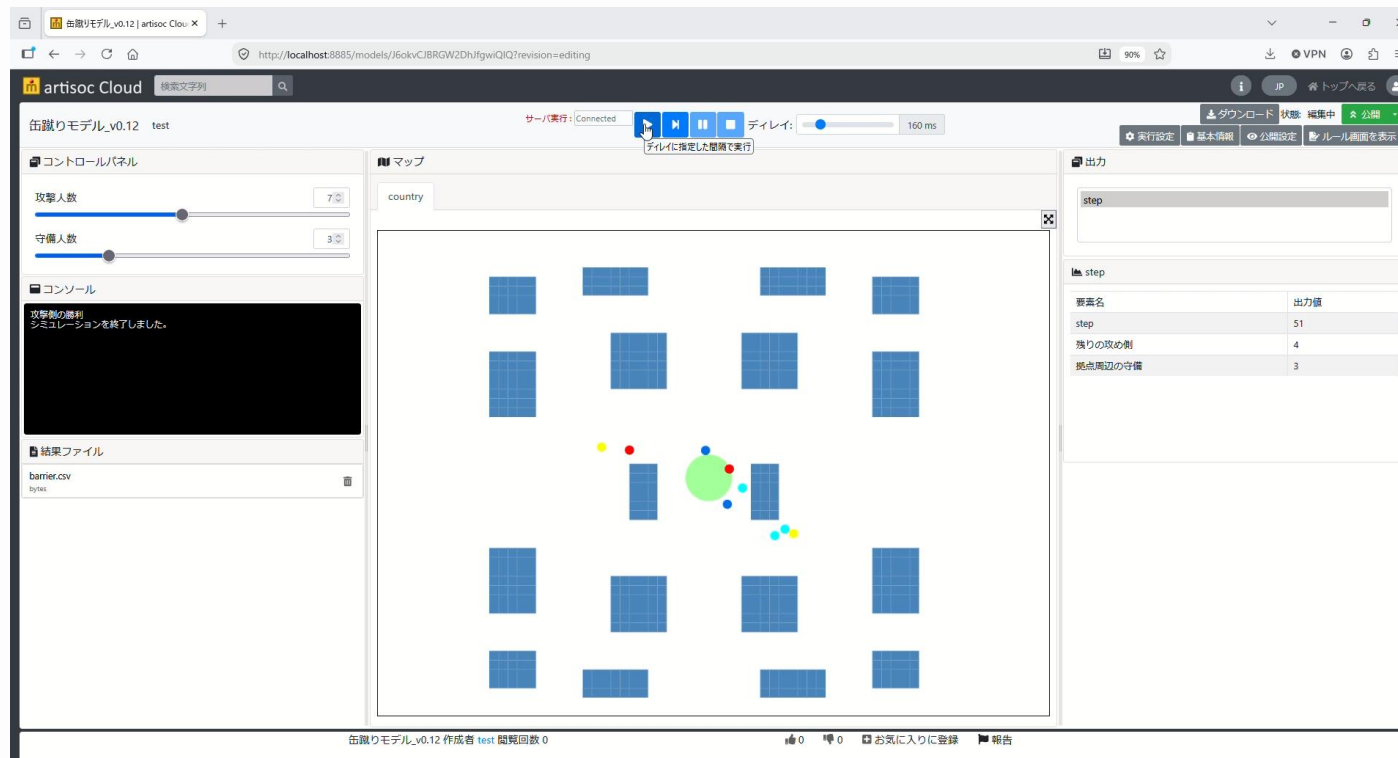
コンパネを操作してみる（2）

- コンパネの値を変更することで攻め側、守り側の数を変更することが出来ます。
- 攻めと守りの数を5:5にした場合
 - お互いが勝ったり負けたりするようになる。



コンパネを操作してみる (3)

- コンパネの値を変更することで攻め側、守り側の数を変更することが出来ます。
- 攻めと守りの数を7:3にした場合
 - 守りの数が足りず、攻めが有利になる。



- 攻めのルール

1. モデル開始時に半数はマップ左上に生成し、残りの半数はマップ右下に生成する。
2. 近くに守りのエージェントがいる場合、それを避けようと動く。
3. 守りにつかまらないように陣地内に入ったら攻めの勝利です。

- 守りのルール
 1. モデル開始時にマップ中心に生成する。
 2. 攻めが陣地内に入らないように守ります。
 3. 陣地の周りに攻めがいはいないか探します。

- その他のルール
 1. 守りが攻めに触れたとき、触れられた攻めを消す
 2. エージェントは障害物、味方にぶつからないように動く

モデルの作り方

1. マップを作成する
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

① マップを作成する

① マップを作成する (1)

Excelを使ってマップを作成します。

- barrier.csvをExcelで開きます。
- マップの進行可能な箇所のプロパティを設定します。
- 0が進行可能、1が進行不可です。

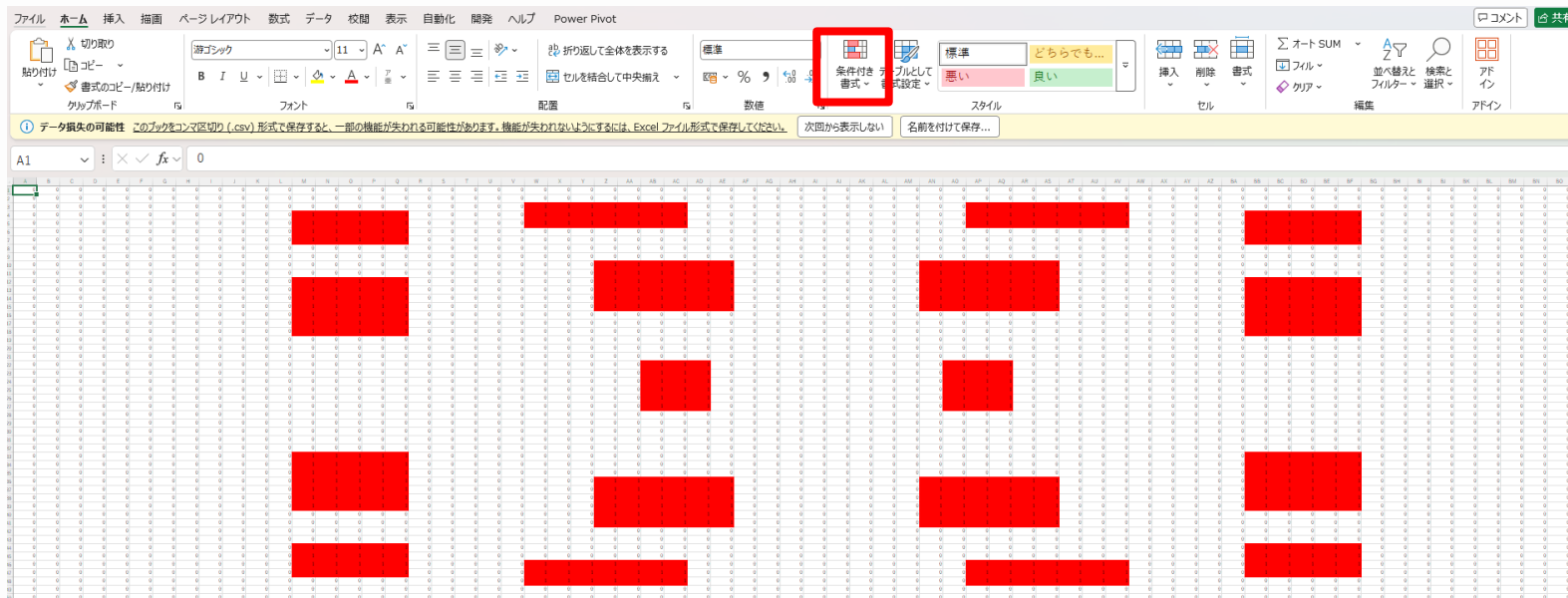
The screenshot shows an Excel spreadsheet with a large grid of cells. The formula bar at the top indicates the active cell is A1, containing the value 0. The grid is filled with a pattern of 0s and 1s, representing a map layout. The columns are labeled with letters from A to SO, and the rows are labeled with numbers from 1 to 50. The data is organized in a way that suggests a path or a set of allowed/disallowed areas on a map.

① マップを作成する (2)

マップのプロパティ値は、以下の手順で作成します。

1. マップの大きさ (70×50) を枠で囲みます。
2. 壁 (進行不可) を 1 で定義します。

※ホームタブ > 条件付き書式 > カラースケール で着色するとマップが見やすくなります。



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

▼ Universe

▼ country

▷ barrier

▷ offence

▷ difference + [edit] [delete]

▷ base

① n_offence

① n_barriers

① noise

① c_avoidance

① v_matching

① f_centering

① barrier_type

① barrier_matrix

① n_diffence

① c_offence

① dif_cnt

① prem_tar

① barrier_pos

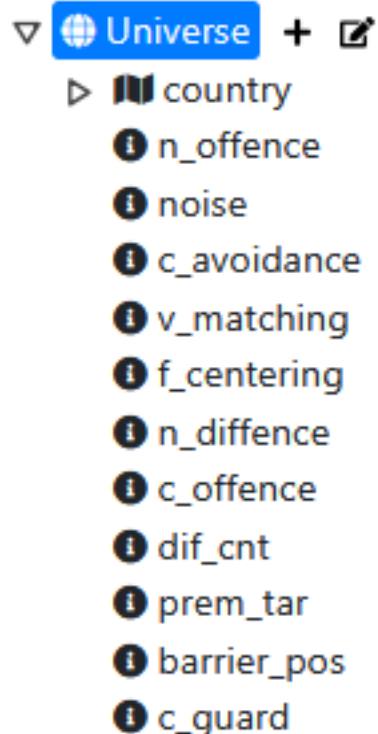
① c_guard

- 空間 (Universe.country)
連続空間、大きさ70×50、レイヤ数1、ループするで作成
- エージェント (Universe.country)
 - barrier: 障害物
 - offence: 陣地に侵入する攻撃エージェント
 - エージェント変数
 - speed: 移動速度
 - role: 役割
 - color: 色
 - distance: 中心までの距離
 - diffence: 陣地を守る守備エージェント
 - エージェント変数
 - speed: 移動速度
 - role: 役割
 - color: 色
 - orbit_dir: 陣地から見た角度
 - target: 標的
 - dif_area: 陣地の距離
 - base: 陣地
 - エージェント変数
 - cnt_dif: 陣地周辺の守備の数

② モデルを定義する (2)

Universeの「変数」を定義します。

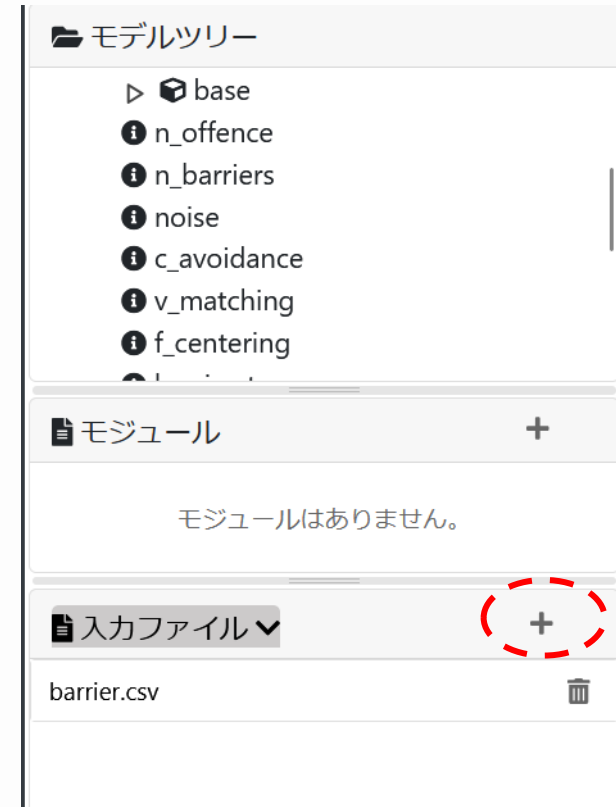
モデルツリー



- n_offence : 攻め側のエージェントの数
- noise : エージェントの動きの揺らぎ
- c_avoidance : 衝突回避
- v_matching : 整列
- f_centering : 追跡
- n_diffence : 守り側のエージェントの数
- c_offence : 陣地付近の攻め側エージェントの数
- dif_cnt : 陣地付近の守り側エージェントの数
- prem_tar : 陣地付近の攻め側エージェント
- barrier_pos : 障害物の座標
- c_guard : 守備役エージェントの数

② 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。
barrier.csv: 障害物をファイル入力



② モデルを定義する (4)

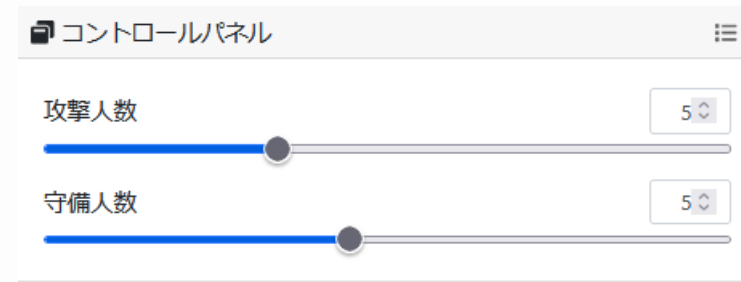
- コントロールパネルに「攻撃人数」、「守備人数」、「noise」を追加し、変数を設定する。

– 攻撃人数

- 種類 : スライダー
- 設定対象 : n_offence
- 値の型 : 整数 (integer)
- 初期値 : 5
- 区間 : 0~20
- 目盛間隔 : 1

– 守備人数

- 種類 : スライダー
- 設定対象 : n_defence
- 値の型 : 整数 (integer)
- 初期値 : 5
- 区間 : 1~10
- 目盛間隔 : 1



③ Universeのルールを記述する

③ Universeのルールを記述する（1）

- 陣地を攻めるエージェントと陣地を守るエージェントに分かれ、他の味方エージェントと相互連携するモデルを作成します。
 - univ_initは、変数の初期化やファイル読み込み、ポテンシャルの計算等を行います。

```
1 def univ_init(self):
2
3     Universe.c_avoidance = 1
4     Universe.f_centering = 1
5     Universe.v_matching = 1
6     Universe.c_guard = 0
7     Universe.noise = 10
8
9
10    # 拠点の生成
11    one = create_agt(Universe.country.base)
12    one.x = 35
13    one.y = 25
14
15    # 障害物の生成
16    self.create_barrier()
17
18    # 攻め側の生成
19    self.create_offence()
20
21    # 守り側の生成
22    self.create_diffence()
23
```

③ Universeのルールを記述する (2)

- create_barrierは、barrier.csvを読み込み、障害物を生成します。

```
40 # 障害物の生成
41 def create_barrier(self):
42     import csv
43
44     try:
45         # 入力ファイルの読み込み
46         with open('barrier.csv', 'r', encoding='utf-8') as fi:
47
48             reader = csv.reader(fi)
49             Universe.barrier_pos = [[0 for _ in range(50)] for _ in range(70) ]
50             for y, row in enumerate(reader):
51                 for x, cell in enumerate(row):
52                     if cell.strip() == '1':
53                         one = create_agt(Universe.country.barrier)
54                         Universe.barrier_pos[x][y] = 1
55                         one.x = x
56                         one.y = y
57
58     except:
59         print(f"[Error] create_barrier : 障害物の生成に問題が発生しました。")
60         exit_simulation()
61
62
```

③ Universeのルールを記述する (3)

- create_offenceは、攻め側のエージェントを生成し、マップの右下と左上を初期位置として交互に配置します。
- 初期の役割として「baiter(様子見)」を設定し、陣地の様子を見ながら陣地に近づこうとする。

```
64 def create_offence(self):
65     right_flg = True
66
67     for i in range(Universe.n_offence):
68         one = create_agt(Universe.country.offence)
69         if right_flg:
70             one.x = 65
71             one.y = 5
72             one.direction = rand() * 360
73             one.speed = 0.2 + rand()*0.3
74             one.direction -= 90
75             one.direction %= 360
76             right_flg = False
77             one.role = 'baiter'
78             one.color = COLOR_YELLOW
79
80         else:
81             one.x = 5
82             one.y = 45
83             one.direction = rand() * 360
84             one.speed = 0.2 + rand()*0.3
85             one.direction -= 90
86             one.direction %= 360
87             right_flg = True
88             one.role = 'baiter'
89             one.color = COLOR_YELLOW
```


③ Universeのルールを記述する（4）

- create_diffenceは、守り側のエージェントを生成し、マップの中心に配置します。
- 役割として「chase(索敵)」と「guard(守備)」を設定し、索敵と陣地の守備を行う。
 - この時、「guard(守備)」のエージェント数は最大3とする。

```
91 def create_diffence(self):
92     role_flg = True
93     for i in range(Universe.n_diffence):
94
95         one = create_agt(Universe.country.diffence)
96         one.x = 35
97         one.y = 25
98         one.speed = 0.2 + rand()*0.3
99         one.direction -= 90
100        one.direction %= 360
101        one.color = COLOR_BLUE
102        if role_flg:
103            one.role = 'chase'
104            if Universe.c_guard < 5:
105                role_flg = False
106                one.color = COLOR_CYAN
107        else:
108            Universe.c_guard += 1
109            one.role = 'guard'
110            role_flg = True
111            one.color = COLOR_BLUE
112            one.dif_area = Universe.c_guard
113
```

③ Universeのルールを記述する (5)

- univ_step_endで、毎ステップ終了時、攻め側の残りのエージェント数をカウントする。連携が出来ない1機以下になった時点で守備側の勝利とする。

```
27 def univ_step_end(self):  
28  
29     Universe.c_offence = count_agt(agttype=Universe.country.offence)  
30     if Universe.c_offence < 2:  
31         print('守備側の勝利')  
32         exit_simulation()  
33  
34  
35
```

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）

- base

- 各ステップごとに陣地周辺の攻めをチェックし、攻めが陣地内に入った場合は、「攻撃側の勝利」と出力し、シミュレーションを終了する。
- 周辺に攻め側のエージェントがいる場合、そのエージェントの情報を守り側のエージェントに情報を渡す。

```
1 def agt_init(self):
2     pass
3
4 def agt_step(self):
5     attcker_flg = self.make_agtset_around_own(2.5, False, agttype=Universe.country.offence) # 拠点内に攻めが入っていないかチェック
6     self.cnt_dif = [] # リストの初期化
7     self.cnt_dif = self.make_agtset_around_own(5, False, agttype=Universe.country.difference) # 拠点周りの守りの数をチェック
8     if len(attcker_flg) > 0: # 拠点内に攻めがいる場合
9         print('攻撃側の勝利')
10        exit_simulation()
11
12    Universe.dif_cnt = len(self.cnt_dif) # 周囲の守りの人数を更新
13
14    # 拠点周りの攻めをチェックし、近くにいる場合そのエージェントの情報を守りに渡す。
15    search_enm = self.make_agtset_around_own(7, False, agttype=Universe.country.offence)
16    if len(search_enm) > 0:
17        Universe.prem_tar = search_enm
18    else:
19        Universe.prem_tar = []
```

④ エージェントのルールを記述する (2)

- offence(攻め)

- role_update: 攻め側の役割の更新
- color_update: 役割に合わせた色の更新
- role_action_update: 役割に合わせた動きの更新
- v_matching: 周囲の攻め側の動きに合わせる
- c_avoidance: 味方にぶつからないようによける
- dif_c_avoidance: 守り側が近くにいる場合よける
- bar_c_avoidance: 障害物がある場合よける

```
3
4 def agt_step(self):
5
6     self.turn(90)
7     self.speed = 1
8     objset = set()
9
10    agt_type = 'Universe.country.offence'
11
12    temp = self.make_agtset_around_own(1, False, agttype=Universe.country.difference)
13    if len(temp) > 0:
14        del_agt(self) # 守りに触れている場合、自身をマップから取り除く。
15
16    # 役割の更新
17    self.role_update()
18
19    # 色の更新
20    self.color_update()
21
22    # 役割毎の動きの更新
23    self.role_action_update()
24
25    # 周囲の攻撃の動きに合わせる
26    self.v_matching(objset, agt_type)
27
28    # 攻撃手を避ける[衝突回避Collision Avoidance]
29    self.c_avoidance(agt_type)
30
31    # 守備手から逃げる[衝突回避Collision Avoidance]
32    self.dif_c_avoidance()
33
34    # 障害物を避ける[衝突回避Collision Avoidance]
35    self.bar_c_avoidance()
36
37    # 少しゆらぎを入れる
38    self.turn((rand() - 0.5) * Universe.noise)
39    self.speed *= 1.0 + (rand() - 0.5) * Universe.noise / 360
40
41    # 前に進む
42    self.forward(self.speed)
43
44    # マーカーの向きを格好よく調整
45    self.turn(-90)
```

④ エージェントのルールを記述する (3)

- offence(攻め)

- 陣地から一定の距離に近づくまで様子を見ながら陣地に近づく。
- 陣地と自身の間に守りのエージェントがいない場合、陣地に向かって攻める。
- 周りに攻めている味方がいる場合、おとりになって守り側のエージェントに近づく。

```
# 役割の更新
def role_update(self):
    objset = set()

    self.distance = measure_distance(self.x, self.y, 35, 25, Universe.country) # 中心との距離を取得
    get_dif = self.make_agtset_around_own(self.distance, False, agttype=Universe.country.diffence) # 周囲の守りの情報を取得
    get_at = self.make_agtset_around_own(10, False, agttype=Universe.country.offence) # 周囲の攻めの情報を取得

    if self.distance < 11:
        for obj in get_dif:
            t = get_direction(self.x, self.y, obj.x, obj.y, Universe.country) # 周囲の守りの角度を取得
            b = get_direction(self.x, self.y, 35, 25, Universe.country) # 自身と陣地の角度を取得
            t = (t - b + 360) % 360 # 角度の差分を出す。
            if 20 > t or 340 < t:
                objset.add(obj) # 自身と陣地の間に守りがあるかどうか判別する
        if 0 < count_agtset(objset):
            self.role = 'baiter'
            for obj in get_at:
                if obj.role == 'attacker':
                    self.role = 'decoy'
                pass
            else:
                self.role = 'attacker'
        else:
            self.role = 'baiter'

# 役割に合わせた色の更新
def color_update(self):
    if self.role == 'decoy':
        self.color = COLOR_MAGENTA
    elif self.role == 'baiter':
        self.color = COLOR_YELLOW
    else:
        self.color = COLOR_RED
```

エージェントの役割を更新した後、
自分の役割に合わせた色に変更する

④ エージェントのルールを記述する (4)

- offence(攻め)

- 陣地に向かって攻める場合、陣地に進行方向を合わせる。
- おとり役である場合、自身に一番近い守りのエージェントに進行方向を合わせる。
- 様子見をする場合、陣地から一定距離の周辺を巡回する。

```
def role_action_update(self):  
  
    # 目的物がある時それに向かう[接近]  
    if self.role == 'attacker':  
        if Universe.f_centering == 1:  
            temp = self.make_agtset_around_own(1000, False, agttype=Universe.country.base)  
            if len(temp) > 0:  
                target = list(temp)[0]  
                self.pursue(target, 0)  
  
    # おとり役の動き[接近]  
    if self.role == 'decoy':  
        if Universe.f_centering == 1:  
            temp = self.make_agtset_around_own(40, False, agttype=Universe.country.diffence)  
            if len(temp) > 0:  
                target = list(temp)[0]  
                self.pursue(target, 0)  
  
    # 様子見ゆさぶりの動き  
    if self.role == 'baiter':  
        if Universe.f_centering == 1:  
            temp = self.make_agtset_around_own(1000, False, agttype=Universe.country.base)  
            if len(temp) > 0:  
                target = list(temp)[0]  
                self.pursue(target, 0)  
                if 10 < measure_distance(self.x, self.y, 35, 25, Universe.country):  
                    pass  
                else:  
                    self.turn(int(rand()*2)*2 - 1 * 90)
```

④ エージェントのルールを記述する (5)

- offence(攻め)

- 自身の周辺に守りのエージェントがいる場合、避ける方向に進行方向を調節する。
- 攻めの役割によって、攻めのの情報を取得する距離を変えている。陣地に向かって攻めている場合、ある程度の距離の守りは無視して陣地に向かうように動く。

```
172 def dif_c_avoidance(self):
173     if Universe.c_avoidance == 1:
174         if self.role == 'attacker':
175             temp = self.make_agtset_around_own(1.5, False, agttype=Universe.country.difference)
176         else:
177             temp = self.make_agtset_around_own(3, False, agttype=Universe.country.difference)
178         if 0 == self.watch_angle(90, Universe.country, temp):
179             pass
180         else:
181             deg = (int(rand()*2)*2 - 1) * 120
182             self.turn(deg)
183         if 0 == self.watch_angle(90, Universe.country, temp):
184             pass
185         else:
186             self.turn(-2 * deg)
187         if 0 < self.watch_angle(90, Universe.country, temp):
188             self.speed /= 2
```


④ エージェントのルールを記述する (6)

• diffence(守り)

- guard_action_update: 守備のエージェントの動きを更新する
- f_centering_offence: 攻め側のエージェントを発見した時の動き
- v_matching: 周囲の守り側のエージェントに動きを合わせる
- c_avoidance: 進行方向の味方にぶつからないようによける
- c_avoidance_chase: 索敵のエージェントが陣地外を移動するように制御
- f_centering: 陣地周辺の守り側エージェントが少ない場合、陣地周辺に集まるように制御
- f_centering_prem_target: 拠点近くの攻め側エージェントを優先して狙うように制御
- c_avoidance_barrier: 障害物をよけて移動する

```
4 = def agt_step(self):
5
6     self.turn(90)
7     self.speed = 1
8     objset = set()
9
10    agt_type = 'Universe.country.diffence'
11
12    # 防衛役は陣地周辺を巡回する
13    self.guard_action_update()
14
15    # 目的物がある時それに向かう[接近]
16    self.f_centering_offence()
17
18    # 周囲の守備手にベクトルを合わせる[整列Velocity Matching]
19    self.v_matching(objset, agt_type)
20
21    # 守備手を避ける[衝突回避Collision Avoidance]
22    self.c_avoidance(agt_type)
23
24
25    # 拠点を避ける[衝突回避Collision Avoidance]chaseのみ
26    self.c_avoidance_chase()
27
28    # 守りが手薄である場合、拠点付近に戻る
29    self.f_centering_base()
30
31    # 拠点近くに敵がいる場合、優先して追う
32    self.f_centering_prem_target(objset)
33
34    # 障害物を避ける[衝突回避Collision Avoidance]
35    self.c_avoidance_barrier()
36
37    # 少しゆらぎを入れる
38    self.turn((rand() - 0.5) * Universe.noise)
39    self.speed *= 1.0 + (rand() - 0.5) * Universe.noise / 360
40
41    # 前に進む
42    self.forward(self.speed)
43
44    # マーカーの向きを格好よく調整
45    self.turn(-90)
46
```

④ エージェントのルールを記述する (7)

- diffence(守り)

- 守備のエージェントそれぞれに守備位置を割り当て、そのエリア内を巡回するように設定。
- エリア外に出た場合は、エリア内に向けて移動するように定義

```
def guard_action_update(self):
    if Universe.f_centering == 1:
        if self.role == 'guard':
            temp = self.make_agtset_around_own(100, False, Universe.country.base)
            temp_dif = self.make_agtset_around_own(5, False, Universe.country.diffence)

            if len(temp):
                target = list(temp)[0]
                dist = measure_distance(self.x, self.y, target.x, target.y, Universe.country)
                self.pursue(target, 0)
                if dist > 4:
                    self.orbit_dir = 0

                elif dist < 2:
                    self.orbit_dir = 180

            else:
                dirc = get_direction(35, 25, self.x, self.y, Universe.country)
                dirc = dirc % 360
                if dirc >= 360 / Universe.c_guard * self.dif_area or dirc < 360 / Universe.c_guard * (self.dif_area - 1):
                    bord = (360 / Universe.c_guard * self.dif_area + dirc < 360 / Universe.c_guard * (self.dif_area - 1) + 180) % 360
                    if dirc > bord:
                        self.orbit_dir = 90
                    else:
                        self.orbit_dir = -90
                else:
                    if rand() > 0.5:
                        self.orbit_dir = 90
                    else:
                        self.orbit_dir = -90
                self.turn(self.orbit_dir)
```

④ エージェントのルールを記述する (8)

- diffence(守り)
 - 周囲に攻め側がいる場合、一番近いエージェントを追いかける。
 - 陣地から一定距離離れた場合、陣地方向に向かって進む。

```
def f_centering_offence(self):
    if Universe.f_centering == 1:
        if self.role == 'chase':
            temp_at = self.make_agtset_around_own(15, False, agttype=Universe.country.offence)
            temp_bs = self.make_agtset_around_own(10, False, agttype=Universe.country.base)
        else:
            temp_at = self.make_agtset_around_own(5, False, agttype=Universe.country.offence)
            temp_bs = self.make_agtset_around_own(5, False, agttype=Universe.country.base)

    if len(temp_bs) > 0:
        if len(temp_at) > 0:
            target = list(temp_at)[0]
            self.pursue(target, 0)
        else:
            temp_bs = self.make_agtset_around_own(1000, False, agttype=Universe.country.base)
            if len(temp_bs) > 0:
                target = list(temp_bs)[0]
                self.pursue(target, 0)
```

④ エージェントのルールを記述する (9)

- diffence(守り)

- 陣地周辺の守り側のエージェントが少ない場合、陣地周辺に戻ることを優先する。
- 陣地周辺に攻め側のエージェントが近づいた場合、最も近い守り側のエージェントが優先して追う。

```
def f_centering_base(self):
    if Universe.f_centering == 1:
        if self.role == 'chase':
            if Universe.dif_cnt < 3:
                temp = self.make_agtset_around_own(1000, False, agttype=Universe.country.base)
                if len(temp) > 0:
                    target = list(temp)[0]
                    self.pursue(target, 0)

def f_centering_prem_target(self):
    if Universe.f_centering == 1:
        if len(Universe.prem_tar) > 0:
            for enm in Universe.prem_tar:
                enm_dis = measure_agt_distance(self, enm)
                get_df = self.make_agtset_around_own(enm_dis, False, agttype=Universe.country.diffence)
                if len(get_df) > 0:
                    for obj in get_df:
                        t = get_direction(self.x, self.y, obj.x, obj.y, Universe.country) # 周囲の守りの角度を取得
                        b = get_direction(self.x, self.y, enm.x, enm.y, Universe.country) # 自身と陣地の角度を取得
                        t = (t - b + 360) % 360 # 角度の差分を出す。
                        if 90 > t or 270 < t:
                            objset.add(obj) # 自身と陣地の間に守りがあるかどうか判別する
                    if 0 < count_agtset(objset):
                        pass
                    else:
                        self.pursue(enm, 0)
                        break
```

④ エージェントのルールを記述する (10)

- offence(攻め)とdiffence(守り)共通
 - 自身の周囲のエージェントに対して、進行方向の視野範囲に存在するエージェントの情報を返す。
 - エージェントの視野角は 関数の呼び出し時に引数として渡す。

```
49 def watch_angle(own, angle, space, tempset):  
50     objset = set()  
51     for obj in tempset:  
52         t = get_direction(own.x, own.y, obj.x, obj.y, space)  
53         t = (t - own.direction + 360) % 360  
54         if angle/2 > t or (360 - angle/2) < t:  
55             objset.add(obj)  
56     n = count_objset(objset)  
57     return n
```

④ エージェントのルールを記述する (11)

- offence(攻め)とdiffernce(守り)共通
 - 周囲に味方のエージェントがいる場合、進行方向をそろえる。

```
126 def v_matching(self, objset, agt_type):
127     # 周囲の攻撃手にベクトルを合わせる[整列Velocity Matching]
128     if Universe.v_matching == 1:
129         temp = self.make_agtset_around_own(1.5, False, agttype=agt_type)
130         for obj in temp:
131             t = get_direction(self.x, self.y, obj.x, obj.y, Universe.country)
132             t = (t - self.direction + 360) % 360
133             if 90 > t or 270 < t:
134                 objset.add(obj)
135         if 0 < count_agtset(objset):
136             total = 0
137             dir_dif = 0 # direction_difference
138             ave_spd = 0 # average_speed
139             for obj in objset:
140                 obj.direction = (obj.direction + 90) % 360 # 相手の方向も調整して観察する
141                 if self.direction < 180:
142                     dir_dif = obj.direction - self.direction
143                     if dir_dif > 180:
144                         dir_dif -= 360
145                 else:
146                     dir_dif = obj.direction - self.direction
147                     if dir_dif < -180:
148                         dir_dif += 360
149             total += dir_dif
150             obj.direction = (obj.direction - 90) % 360
151             ave_spd += obj.speed
152         total /= count_agtset(objset)
153         ave_spd /= count_agtset(objset)
154         self.direction += total
155         self.speed = ave_spd
156
```

④ エージェントのルールを記述する (12)

- offence(攻め)とdiffence(守り)共通
 - 周囲のエージェントを取得して、進行方向に存在する場合、ぶつからないように角度を変更する。
 - 視野角内のエージェントの有無は前述のwatch_angle()で判断している。

```
157 def c_avoidance(self, agt_type):
158     if Universe.c_avoidance == 1:
159         temp = self.make_agtset_around_own(1.5, False, agttype=agt_type)
160         if 0 == self.watch_angle(90, Universe.country, temp):
161             pass
162         else:
163             deg = (int(rand()*2)*2 - 1) * 60
164             self.turn(deg)
165         if 0 == self.watch_angle(90, Universe.country, temp):
166             pass
167         else:
168             self.turn(-2 * deg)
169         if 0 < self.watch_angle(90, Universe.country, temp):
170             self.speed /= 2
```

④ エージェントのルールを記述する (13)

- offence(攻め)とdiffence(守り)共通

- エージェントの進行方向に障害物がある場合、ランダムな方向に角度を変える。
- 進行方向に障害物が無くなるまでチェックを繰り返す。

```
191 def bar_c_avoidance(self):
192     if Universe.c_avoidance == 1:
193         for i in range(100):
194             fw_dir = self.direction % 360 # 自身の進行方向の角度を取得
195             if fw_dir < 22.5 or fw_dir > 337.5:
196                 if Universe.barrier_pos[int(self.x) + 1][int(self.y)] == 1: # 進行方向の座標に障害物がある場合
197                     self.turn((int(rand()*2)*2 - 1) * 90) # エージェントを目標させ、角度を調整する
198                 else:
199                     break
200             elif fw_dir >= 22.5 and fw_dir < 67.5:
201                 if Universe.barrier_pos[int(self.x) + 1][int(self.y) + 1] == 1:
202                     self.turn((int(rand()*2)*2 - 1) * 90)
203                 else:
204                     break
205             elif fw_dir >= 67.5 and fw_dir < 112.5:
206                 if Universe.barrier_pos[int(self.x) + 1][int(self.y) + 1] == 1:
207                     self.turn((int(rand()*2)*2 - 1) * 90)
208                 else:
209                     break
210             elif fw_dir >= 112.5 and fw_dir < 157.5:
211                 if Universe.barrier_pos[int(self.x) + 1][int(self.y) + 1] == 1:
212                     self.turn((int(rand()*2)*2 - 1) * 90)
213                 else:
214                     break
215             elif fw_dir >= 157.5 and fw_dir < 202.5:
216                 if Universe.barrier_pos[int(self.x) + 1][int(self.y)] == 1:
217                     self.turn((int(rand()*2)*2 - 1) * 90)
218                 else:
219                     break
220             elif fw_dir <= 337.5 and fw_dir > 292.5:
221                 if Universe.barrier_pos[int(self.x) + 1][int(self.y) - 1] == 1:
222                     self.turn((int(rand()*2)*2 - 1) * 90)
223                 else:
224                     break
225             elif fw_dir <= 292.5 and fw_dir > 247.5:
226                 if Universe.barrier_pos[int(self.x) + 1][int(self.y) - 1] == 1:
227                     self.turn((int(rand()*2)*2 - 1) * 90)
228                 else:
229                     break
230             elif fw_dir <= 247.5 and fw_dir > 202.5:
231                 if Universe.barrier_pos[int(self.x) - 1][int(self.y) - 1] == 1:
232                     self.turn((int(rand()*2)*2 - 1) * 90)
233                 else:
234                     break
235             else:
236                 print('異常を検知しました。')
237                 if Universe.barrier_pos[int(self.x) - 1][int(self.y)] == 1:
238                     self.turn((int(rand()*2)*2 - 1) * 90)
239                 else:
240                     break
241
```


- エージェントに個性を追加してみよう
(移動速度に差をつける、周囲の警戒度を変えてみる)
- 攻め側の開始位置を変えてみよう
(開始位置を変えてみる、開始位置を増やしてみる)
- 地形を生かした陣地の守り方を考えてみよう。
- エージェントとエージェントの間に障害物がある場合、相手が見えないようにしてみよう。