artisoc Cloud

関数仕様

株式会社 構造計画研究所

mas-support@kke.co.jp

目次

数値計算

abs

abs(x)

x の絶対値を返す。

```
a = abs(-3.5)  # -3.5 の絶対値（3.5）を a に代入
```

atan

atan(x)

x の逆正接を、ラジアンで返す。

```
a = atan(1)  # a に 0.7853981633974483（π/4）を代入
```

cos

cos(x)

角度 x（ラジアン）の余弦を返す。

```
a = cos(0)  # a に 1 を代入
```

cosh

cosh(x)

x の逆双曲線余弦を返す。

```
a = cosh(0)  # a に 1 を代入
```

degrees

degrees(x)

角度 x をラジアンから度に変換する。

```
a = degrees(0.7853981633974483)  # a に 45.0 を代入
```

exp

exp(x)

自然対数の低 e の x 上を返す。

```
a = exp(1)  # a に 2.71828182846（e）を代入
```

log

log(x[, base])

引数が 1 つの場合、x の自然対数を返す。

引数が 2 つの場合、2 つ目の引数 base を低とした x の対数を返す。

```
a = log(2.71828182846)  # a に 1.000000000 を代入

a = log(100,10) #a に 2 を代入
```

normalvariate

normalvariate(mu, sigma)

正規分布に従う乱数を返す。

- **mu** - 正規分布の平均。
- **sigma** - 正規分布の標準偏差。

```
a = normalvariate(0,1)  # a に標準正規分布に従う乱数を代入
```

pi

pi()

円周率 π の値を返す。

```
a = pi()  # a に 3.141592653589793 を代入
```

radians

radians(x)

角度 x を度からラジアンに変換する。

```
a = radians(45)  # a に 0.7853981633974483 を代入
```

rand

rand()

0.0 以上 1.0 未満の一様乱数値（ランダムな値）を返す。

```
a = rand()  # a に 0.0~1.0 のランダムな値を代入
```

randint

randint(a, b)

a 以上 b 以下であるようなランダムな整数を返す。

```
a = randint(4,9)  # a に 4~9 の整数いずれかを代入
```

randuniform

randuniform(a, b)

a 以上 b 以下（a>b であれば b 以上 a 以下）の一様乱数値を返す。

```
a = randuniform(3,5)  # a に 3 以上 5 以下のランダムな値を代入
```

round

round(x[, ndigits])

実数 x の値を四捨五入した整数を返す。2 つ目の引数 ndigits が与えられた場合は、小数部を ndigits 桁に丸めた値を返す。

```
a = round(3.5)  # a に 4 を代入
a = round(3.14,1) # a に 3.1 を代入
```

sin

sin(x)

角度 x（ラジアン）の正弦を返す。

```
a = sin(0)  # a に 0 を代入
```

sinh

sinh(x)

x の双曲線正弦を返す。

```
a = sinh(0)  # a に 0 を代入
```

sqrt

sqrt(x)

x の平方根を返す。

```
a = sqrt(2)  # a に 1.4142135623730951 を代入
```

tan

tan(x)

角度 x（ラジアン）の正接を返す。

```
a = tan(0)  # a に 0 を代入
```

tanh

tanh(x)

x の双曲線正接を返す。

```
a = tanh(0)  # a に 0 を代入
```

文字列操作

find

文字列中で検索対象の文字列パターンが最初に現れるインデックスを返す。パターンが見つからなかった場合は-1 を返す。

find(sub[, start[, end]])

- **sub** - 検索対象とする文字列。
- **start** - 検索開始位置。（省略可）
- **end** - 検索終了位置。（省略可で、start が指定されているときのみ有効）

※「[文字列型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#string-methods

```
s = “artisoc Cloud”

# 文字列”artisoc Cloud”で最初に”o”が現れるインデックス（0 から数えている）
print(s.find(“o”))  # => 5

# 文字列”artisoc Cloud”の 6 番目以降で最初に”o”が現れるインデックス
print(s.find(“o”, 6))  # => 10

# 文字列”artisoc Cloud”の 0~4 番目で最初に”o”が現れるインデックス（存在しない）
print(s.find(“o”, 0, 4)) # => -1

# 文字列”artisoc Cloud”で最初に”soc”が現れるインデックス
print(s.find(“soc”))  # => 4
```

len

len(sub)

文字列 sub の文字数を返す。

```
s = “artisoc Cloud”
print(len(s))  # => 13
```

replace

replace(old, new)

文字列をコピーし、現れる部分文字列 *old* 全てを *new* に置換して返す。

※「[文字列型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#string-methods

```
s1 = “artisoc”
s2 = s1.replace(“soc”, “wor”)  # “artisoc”の”soc”を”wor”に入れ替えて s2 に代入
print(s2)  # => artiwor
```

strip

strip()

文字列の前後の空白文字を削除する。

※「[文字列型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#string-methods

```
s1 = "   artisoc Cloud   "
s2 = s1.strip()  # 前後の空白文字を削除
print(s2)  # => artisoc Cloud
```

データ型変換

bool

bool(x)

引数 x をブール値に変換して返す。

```
a = bool(0==1)  # a に False を代入
a = bool(0==0)  # a に True を代入
```

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/functions.html#built-in-functions

float

float(x)

引数 x を実数値（浮動小数点値）に変換して返す。※主にファイルなどから文字列型として読み込んだ数字を変換するのに用いる

```
a = float(3)  # a に 3.0 を代入
```

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/functions.html#built-in-functions

int

int(x)

引数 x を整数値に変換して返す。実数値が与えられた場合は小数点以下を切り捨てる。

```
a = int(3.6)  # a に 3 を代入
```

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/functions.html#built-in-functions

str

str(x)

引数 x を文字列型に変換して返す。

```
a = str(3)  # a に 3 を文字列型として代入
```

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/functions.html#built-in-functions

エージェント操作（生成・削除）

create_agt

エージェントを生成する。生成するエージェント数が 1 のときはエージェントを、複数のときはエージェント集合（set 型）を返す。

create_agt(agttype, num=1)

- **agttype** - 生成するエージェント種別。
- **num** - 生成するエージェント数。初期値は 1。

```
# person エージェントを 1 つ生成する
create_agt(Universe.city.person)

# person エージェントを 1 つ生成して変数 taro に代入する
taro = create_agt(Universe.city.person)

# person エージェントを 10 個生成してその集合を変数 people に代入する
people = create_agt(Universe.city.person, num=10)
```

del_agt

エージェントを削除する。 ※削除するエージェントの step 関数の途中でこれを実行した場合、step 関数自体は最後まで実行される

del_agt(agt)

- **agt** - 削除するエージェント。

```
# エージェント taro を削除する
del_agt(taro)

# （agt_step 内で）ブール型変数 condition が True のとき、自分自身を削除し step 関数の処理を終了
if condition:
    del_agt(self)
    return
```

kill_agt

エージェントを削除する 。※エージェントが実際に削除されるのは、ステップの終了時

kill_agt(agt)

- **agt** - 削除するエージェント。

```
# エージェント taro を削除する（実際に削除されるのはステップ終了時）
kill_agt(taro)
```

エージェント操作（その他）

get_history

指定した過去のステップにおけるユニバースまたは空間またはエージェントを取得する。※取得したユニバースまたは空間またはエージェントの変数を取得できるが、数値型と文字列型の変数に限る※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※空間上に配置したエージェントの場合、自動的に定義される変数（x,y,layer 等）は対象外となるため、別で変数を追加する必要がある。

記憶数を用いたサンプルモデルは「記憶数サンプルモデル」を参照してください。

get_history(object, step)

- **object** - 過去を取得する対象。ユニバースまたは空間またはエージェント。
- **step** - 過去に遡るステップ数。

```
Universe.value = 1                  # Universe に value に 1 を代入


taro = create_agt(Universe.taro)  # エージェントを生成し、変数 taro に代入
taro.x = 10                         # taro が持つ変数 x に 10 を代入
Universe.taro_obj = taro            # 変数 taro をオブジェクトに代入
Universe.value += 1                 # 変数 value に 1 を加算
Universe.taro_obj.x += 1            # 変数 x に 1 を加算
```

```
if count_step() > 1:
    # Universe とエージェントの変数に関して、過去の情報を取得する。
    past_Universe = get_history(Universe, 1)  # 1 ステップ前の Universe の情報を取得する。
    past_taro_obj = get_history(Universe.taro_obj, 1)
```

specify_agttype

指定したエージェントのエージェント種別を取得する。

specify_agttype(agt)

- **agt** - 指定するエージェント

```
taro_type = specify_agttype(taro)  # taro のエージェント種別を取得
create_agt(taro_type)  # taro と同じ種別のエージェントを生成
```

specify_kill_agt

指定したエージェントの kill フラグの値を取得する。※kill フラグは kill_agt 関数によって立てられる

specify_kill_agt(agt)

- **agt** - 指定するエージェント

```
# taro の kill フラグが立っているときのみ処理を行う
if specify_kill_agt(taro):
    …
```

空間エージェント関数（移動）

forward

前方へ進む。正常終了のときは-1、空間がループしていないときで指定した距離を進めなかったときは、進めなかった距離を返す。

forward(distance)

- **distance** - 進む距離

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる ※連続空間でのみ使用可能。

```
# 前方に 2 進む（空間エージェントのルールエディタ内に記述）
self.forward(2)
```

forward_direction_sqgrid

四角格子上を指定した方向へ移動する。四角格子空間または連続空間で使用可能。

forward_direction_sqgrid(direction, distance)

- **direction** - 進む方向。
  - 原点が左下のとき　0:右、1:右上、2:上、3:左上、4:左、5:左下、6:下、7:右下
  - 原点が左上のとき　0:右、1:右下、2:下、3:左下、4:左、5:左上、6:上、7:右上
- **distance** - 進む距離。チェビシェフ距離で定義し、整数で指定する。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# 右上に 2 進む（左下原点のとき、空間エージェントのルールエディタ内に記述）
self.forward_direction_sqgrid(1, 2)
```

move_center

エージェント自身が乗っている空間（Layer）の中央に移動する。※格子空間で空間の幅（高さ）が偶数の場合、小数点切り捨て

move_center()

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# エージェント自身が 50×50 の空間に乗っている時(25,25)に移動
self.move_center()
```

move_space_around_own_sqgrid

四角格子空間で、指定した視野の範囲で空き地（格子）を探し、移動する。検索された領域に複数の空き地が存在するときは、ランダムに移動位置を決定する。

move_space_around_own_sqgrid(view, avoid=None, manhattan=False)

- **view** - 視野。整数で指定する。
- **avoid** - 重ならないエージェント集合。初期値は None で、このときは全てのエージェントと重ならない。
- **manhattan**: False のときは視野をチェビシェフ距離で、True のときはマンハッタン距離で定義する。初期値は False。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# （以下、空間エージェントのルールエディタ内に記述）

# チェビシェフ距離で視野 2 の範囲の格子のうち、エージェントが誰もいない場所へランダムに移動する
self.move_space_around_own_sqgrid(2)

# チェビシェフ距離で視野 3 の範囲の格子のうち、集合 enemies に含まれるエージェントが誰もいない場所へランダムに移動する
self.move_space_around_own_sqgrid(3, avoid=enemies)

# マンハッタン距離で視野 3 の範囲の格子のうち、集合 enemies に含まれるエージェントが誰も
```

```
いない場所へランダムに移動する

self.move_space_around_own_sqgrid(3, avoid=enemies, manhattan=True)
```

move_space_around_position_sqgrid

四角格子空間で、指定された位置を中心に周りの空き地を探し、移動する。検索された領域に複数の空き地が存在するときは、ランダムに移動位置を決定する。

move_space_around_position_sqgrid(view, x, y, layer, avoid=None, manhattan=False)

- **view** - 指定する座標の周囲を探索する視野。
- **x** - 指定する位置の x 座標。
- **y** - 指定する位置の y 座標。
- **layer** - 指定するレイヤ。
- **avoid** - 重ならないエージェント集合。初期値は None で、このとき全てのエージェントと重ならない。
- **manhattan** - False のときは視野をチェビシェフ距離で、True のときはマンハッタン距離で定義する。初期値は False。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# （以下、空間エージェントのルールエディタ内に記述）


# レイヤ 0 の座標(25,30)からチェビシェフ距離で視野 2 の範囲の格子のうち、エージェントが誰もいない場所へランダムに移動する

self.move_space_around_position_sqgrid(2, 25, 30, 0)


# レイヤ 0 の座標(25,30)からチェビシェフ距離で視野 3 の範囲の格子のうち、集合 enemies に
```

```
含まれるエージェントが誰もいない場所へランダムに移動する
self.move_space_around_position_sqgrid(3, 25, 30, 0, avoid=enemies)

# レイヤ 0 の座標(25,30)からマンハッタン距離で視野 3 の範囲の格子のうち、集合 enemies に
含まれるエージェントが誰もいない場所へランダムに移動する
self.move_space_around_position_sqgrid(3, 25, 30, 0, avoid=enemies, manhattan=T
rue)
```

pursue

目標のエージェントへ向かって移動する。正常終了時は-1、進めなかったときは進めなかった距離を返す。

pursue(agt, distance)

- **agt** - 目標とするエージェント。
- **distance** - 移動する距離。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる※連続空間でのみ使用可能。

```
# エージェント taro に向かって距離 3 移動する
self.pursue(taro, 3)

# エージェント taro に向かって距離 0.1 進み、進めなかったとき（taro まで距離 0.1 未満ま
で接近したとき）自分の色を赤くする
if self.pursue(taro, 0.1) > 0:
    self.color = COLOR_RED
```

turn

向きを変える。

turn(degree)

- **degree** - 向きを変える角度。度数法で指定。左下原点のとき、引数が正の値のときは左回り、負の値のときは右回りに向きを変える。左上原点のときは逆。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# 左回りに 30 度向きを変える（左下原点のとき）
self.turn(30)
```

turn_agt

指定したエージェントの方向を向く。

turn_agt(agt)

- **agt** - 指定するエージェント。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# エージェント taro の方向を向く
self.turn_agt(taro)
```

空間エージェント関数（空間操作）

get_height_space_own

自身が乗っている空間の縦幅を取得する。

get_height_space_own()

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# エージェント自身が 50×50 の空間に乗っている時、a に 50 を代入
a=self.get_height_space_own()
```

get_width_space_own

自身が乗っている空間の横幅を取得する。

get_width_space_own()

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# エージェント自身が 50×50 の空間に乗っている時、a に 50 を代入
a=self.get_width_space_own()
```

reverse_direction_sqgrid

四角格子空間上で逆方向を取得する。

reverse_direction_sqgrid(direction)

- **diection** - 方向

– 原点が左下のとき　0:右、1:右上、2:上、3:左上、4:左、5:左下、6:下、7:右下

– 原点が左上のとき　0:右、1:右下、2:下、3:左下、4:左、5:左上、6:上、7:右上

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
a=self.reverse_direction_sqgrid(0)
# エージェント自身が左下原点の空間に乗っている時、a に 4 を代入
```

空間エージェント関数（エージェント集合生成）

make_agtset_around_own

連続空間上で指定した視野の範囲内にいるエージェントの集合を返す。

make_agtset_around_own(distance, include_self, agttype=None)

- **distance** - 視野の広さ。
- **include_self** - False のときエージェント集合に自分を含まず、True のとき含む。
- **agttype** - 集める対象とするエージェント種別。初期値は None で、このとき全てのエージェント種別を対象とする。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる※連続空間でのみ使用可能。

```
# 自分から距離 5 以内のエージェントを全て取得する（自分を含まない）
# （空間エージェントのルールエディタに記述）
neighbors = self.make_agtset_around_own(5, False)

# 自分から距離 3 以内の Universe.hiroba.hito エージェントを取得する（自分を含む）
# （空間エージェントのルールエディタに記述）
neighbors = self.make_agtset_around_own(5, True, agttype=Universe.hiroba.hito)
```

make_agtset_around_own_sqgrid

四角格子空間上で、指定した視野の範囲内にいるエージェントの集合を返す。

make_agtset_around_own_sqgrid(distance, include_self, agttype=None, manhattan=False)

- **distance** - 視野の広さ。
- **include_self** - False のときエージェント集合に自分を含まず、True のとき含む。
- **agttype** - 集める対象とするエージェント種別。初期値は None で、このとき全てのエージェント種別を対象とする。
- **manhattan** - False のときは視野をチェビシェフ距離で、True のときはマンハッタン距離で定義する。初期値は False。

※基本的に、空間エージェントのルールエディタ内で「self.[関数名]」の形で用いる

```
# 自分から距離 5 以内のエージェントを全て取得する（自分を含まない）
# （空間エージェントのルールエディタに記述）
neighbors = self.make_agtset_around_own(5, False)
```

```
# 自分から距離 3 以内の Universe.hiroba.hito エージェントを取得する（自分を含む）
# （空間エージェントのルールエディタに記述）
neighbors = self.make_agtset_around_own(5, True, agttype=Universe.hiroba.hito)
```

エージェント集合操作（基本操作）

add

エージェント集合型変数にエージェントを追加する。

add(agt)

- **agt** - 追加するエージェント。

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people にエージェント taro を追加する
people.add(taro)
```

clear

エージェント集合型変数の中身をクリアする。

clear()

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people の中身をクリアする
people.clear()
```

copy

エージェント集合型変数をコピーする。

copy()

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people をコピーして people2 を作成する
people2 = people.copy()
```

count_agt

引数の指定に当てはまるエージェントの数を取得する。

count_agt(agttype=None, space=None, layer=None, only_alive=False)

- **agttype** - 指定するエージェント種別。指定しない場合は全てのエージェント種別。
- **space** - 指定する空間。指定しない場合は全ての空間。
- **layer** - 指定するレイヤ。指定しない場合は全てのレイヤ。
- **only_alive** - kill フラグが立っていないエージェントのみを指定するかどうか。初期値は False。 ※kill フラグは kill_agt 関数によって立てられる

```
# エージェント種別 person のエージェント数を取得する
n = count_agt(agttype=Universe.city.person)
n = count_agt(agttype=Universe.city.person, only_alive=True)  # kill フラグが立っていないエージェント数を取得する
```

count_agtset

エージェント集合型変数が保持しているエージェント数を取得する。

count_agtset(agtset)

```
# エージェント集合 people のエージェント数を取得する
n = count_agtset(people)
```

discard

エージェント集合型変数から、指定したエージェントを削除する。

discard(agt)

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people からエージェント taro を削除する
people.discard(taro)
```

randchoice

エージェント集合からランダムに 1 つのエージェントを取り出す。

randchoice(agtset)

```
# エージェント集合 people からランダムに 1 つのエージェントを取り出し変数 one に格納する
one = randchoice(people)
```

randsample

エージェント集合からランダムに重複なく複数のエージェントを取り出す

randsample(agtset, num)

- **agtset** - 対象とするエージェント集合
- **num** - 取り出すエージェントの個数

```
# エージェント集合 people からランダムに 2 つのエージェントを取り出し変数 pair に格納する
pair = randsample(people)
```

エージェント集合操作（集合演算）

difference

2 つのエージェント集合の差集合を取得する。

difference(agtset1, agtset2)

- **agtset1** - もととなるエージェント集合
- **agtset2** - agtset1 から取り除くエージェント集合

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people1 から people2 を除いた people3 を取得する
people3 = people1.difference(people2)

# ※以下のように書いても同様
people3 = people1 - people2
```

intersection

2 つのエージェント集合の積集合を取得する。

intersection(agtset1, agtset2)

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people1 と people2 の両方に所属するエージェント集合 people3 を取得する
people3 = people1.intersection(people2)

# ※以下のように書いても同様
people3 = people1 & people2
```

symmetric_difference

2 つのエージェント集合の対称差集合を取得する。

symmetric_difference(agtset1, agtset2)

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people1 と people2 のどちらか一方にだけ含まれているエージェントからなるエージェント集合 people3 を取得する
people3 = people1.symmetric_difference(people2)

# ※以下のように書いても同様
people3 = people1 ^ people2
```

union

2 つのエージェント集合の和集合を取得する。

union(agtset1, agtset2)

※「[集合型変数].[関数名]」の形で用いる※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/stdtypes.html#set

```
# エージェント集合 people1 と people2 のいずれかに含まれているエージェントからなるエージェント集合 people3 を取得する
people3 = people1.union(people2)
```

```
# ※以下のように書いても同様
people3 = people1 | people2
```

エージェント集合操作（生成）

make_agtset

引数の指定に当てはまるエージェントの集合を取得する。

make_agtset(agttype=None, space=None, layer=None, only_alive=False)

- **agttype** - 指定するエージェント種別。指定しない場合は全てのエージェント種別。
- **space** - 指定する空間。指定しない場合は全ての空間。
- **layer** - 指定するレイヤ。指定しない場合は全てのレイヤ。
- **only_alive** - kill フラグが立っていないエージェントのみを指定するかどうか。初期値は False。

```
# エージェント種別 person のエージェント集合を取得する
people = make_agtset(agttype=Universe.city.person)

# エージェント種別 person のエージェントのうち、kill フラグが立っていないエージェントの集合を取得する
people = make_agtset(agttype=Universe.city.person, alive=True)
```

make_agtset_around_position

連続空間上の指定した位置から指定した範囲内にいるエージェントの集合を返す。

make_agtset_around_position(distance, space, x, y, layer, agttype=None)

- **distance** - 指定する範囲の広さ。
- **space** - 指定する空間。
- **x** - 指定する位置の x 座標。
- **y** - 指定する位置の y 座標。
- **layer** - 指定するレイヤ。
- **agttype** - 集める対象とするエージェント種別。初期値は None で、このとき全てのエージェント種別を対象とする。

※連続空間でのみ使用可能。

```
# 空間 Universe.city のレイヤ 0 の座標(25,30)から距離 10 以内に存在するエージェントの集合を取得する
group = make_agtset_around_position(10, Universe.city, 25, 30, 0)

# 空間 Universe.city のレイヤ 0 の座標(25,30)から距離 10 以内に存在するエージェントのうち Universe.city.person 種別のエージェントのみを取得する
people = make_agtset_around_position(10, Universe.city, 25, 30, 0, agttype=Universe.city.person)
```

make_agtset_around_position_sqgrid

四角格子空間上で、指定した位置から指定した範囲内にいるエージェントの集合を返す。

make_agtset_around_position_sqgrid(distance, space, x, y, layer, agttype=None, manhattan=False)

- **distance** - 指定する範囲の広さ。
- **space** - 指定する空間。
- **x** - 指定する位置の x 座標。
- **y** - 指定する位置の y 座標。
- **layer** - 指定するレイヤ。
- **agttype** - 集める対象とするエージェント種別。初期値は None で、このとき全てのエージェント種別を対象とする。
- **manhattan** - False のときは指定範囲をチェビシェフ距離で、True のときはマンハッタン距離で定義する。初期値は False。

```
# 空間 Universe.city のレイヤ 0 の座標(25,30)からチェビシェフ距離で距離 2 以内に存在するエージェントの集合を取得する
group = make_agtset_around_position_sqgrid(2, Universe.city, 25, 30, 0)

# 空間 Universe.city のレイヤ 0 の座標(25,30)からマンハッタン距離で距離 3 以内に存在するエージェントのうち Universe.city.person 種別のエージェントのみを取得する
people = make_agtset_around_position(3, Universe.city, 25, 30, 0, agttype=Universe.city.person, manhattan=True)
```

エージェント集合操作（配置）

random_put_agtset

指定されたエージェント集合を連続空間上にランダムに配置する。

random_put_agtset(agtset)

- **agtset** - 指定するエージェント集合

※連続空間でのみ使用可能。

```
# エージェント集合 people を空間上にランダムに配置する
random_put_agtset(people)
```

random_put_agtset_sqgrid

指定されたエージェント集合を格子空間上にランダムに配置する。

random_put_agtset_sqgrid(agtset, overlap=False, avoid=None)

- **agtset** - 指定するエージェント集合
- **overlap** - agtset を配置する際、同じ座標に重なって配置することを許すかどうか（初期値は False）
- **avoid** - 重ならないエージェント集合。既にマップ上に配置されているエージェント集合を指定する。初期値は None で、このとき重ならないエージェント集合を指定しない。

```
# エージェント集合 people を格子空間上にランダムに配置する（重なって配置しない）
random_put_agtset_sqgrid(people)

# エージェント集合 people を空間上にランダムに配置する（重なって配置することを許す）
random_put_agtset_sqgrid(people, overlap=True)

# エージェント集合 people を、既に配置されているエージェント集合 residents にも重ならないようランダムに配置する
random_put_agtset_sqgrid(people, avoid=residents)
```

エージェント集合操作（その他）

make_agtlist

set 型のエージェント集合（順序なし）を受け取り、list 型（順序あり）にして返す。

make_agtlist(agtset, key=None, reverse=False)

- **agtset** - エージェント集合
- **key** - 並べ替えの基準となるエージェント属性（文字列で指定）（初期値は指定なし）
- **reverse** - 並べ替えの順序。False（初期値）だと昇順、True だと降順。

```
# エージェント集合 people をエージェント属性 tall の昇順に並べたリストを取得する
paople_list = make_agtlist(people, key="tall")
p = people_list[0]  # リストの 0 番目（属性 tall が最も小さいエージェント）を取得する

# エージェント集合 people をエージェント属性 tall の降順に並べたリストを取得する
paople_list = make_agtlist(people, key="tall", reverse=True)
p = people_list[0]  # リストの 0 番目（属性 tall が最も大きいエージェント）を取得する
```

sort_agtlist

list 型のエージェント集合を並べ替えて、list 型で返す。

sort_agtlist(agtlist, key, reverse=False)

- **agtlist** - エージェント集合（list 型）

- **key** - 並べ替えの基準となるエージェント属性（文字列で指定）（初期値は指定なし）
- **reverse** - 並べ替えの順序。False（初期値）だと昇順、True だと降順。

```
# list 型のエージェント集合 people_list をエージェント属性 tall の昇順に並べたリストを取得する
paople_sort = sort_agtlist(people_list, key="tall")
p = people_sort[0]  # リストの 0 番目（属性 tall が最も小さいエージェント）を取得する

# list 型のエージェント集合 people_list をエージェント属性 tall の降順に並べたリストを取得する
paople_sort = sort_agtlist(people_list, key="tall", reverse=True)
p = people_sort[0]  # リストの 0 番目（属性 tall が最も大きいエージェント）を取得する
```

空間操作

get_agt_direction

エージェント 1 から見たエージェント 2 の角度を返す。空間がループする場合は、最短距離として求められる方向の角度を返す。

get_agt_direction(agt1, agt2)

- **agt1** - エージェント 1
- **agt2** - エージェント 2

```
# taro から見た hanako の角度を取得する
d = get_agt_direction(taro, hanako)
```

get_direction

地点 1 から見た地点 2 の角度を返す。空間がループする場合は、最短距離として求められる方向の角度を返す。

get_direction(x1, y1, x2, y2, space)

- **x1** - 地点 1 の x 座標
- **y1** - 地点 1 の y 座標
- **x2** - 地点 2 の x 座標
- **y2** - 地点 2 の y 座標
- **space** - 対象となる空間

```
# 空間 Universe.city において、座標(1,1)から見た座標(10, 15)の角度を取得する
d = get_direction(1, 1, 10, 15, Universe.city)
```

get_height_space

指定された空間の縦幅を取得する。

get_height_space(space)

- **space** - 空間

```
# 空間 Universe.city が 50×50 の時 50 を d に代入
d = get_height_space(Universe.city)
```

get_layer_space

指定された空間のレイヤ r 数を取得する。

get_layer_space(space)

- **space** - 空間

```
# 空間 Universe.city の layer が 1 の時 d に 1 を代入
d = get_layer_space(Universe.city)
```

get_ride_space

指定されたエージェント種別が乗っている空間を取得する。

get_ride_space(agttype)

- **agttype** - エージェント種別

```
# エージェント種別 Universe.city.taro が乗っている空間を d に代入
d = get_ride_space(Universe.city.taro)
```

get_width_space

指定された空間の横幅を取得する。

get_width_space(space)

- **space** - 空間

```
# 空間 Universe.city が 50×50 の時 50 を d に代入
d = get_width_space(Universe.city)
```

measure_agt_distance

連続空間上でエージェント間の距離を取得する。

measure_agt_distance(agt1, agt2)

- **agt1** - 1 つ目のエージェント
- **agt2** - 2 つ目のエージェント

※連続空間でのみ使用可能。

```
# エージェント taro と hanako の距離を取得する
distance = measure_agt_distance(taro, hanako)
```

measure_agt_distance_sqgrid

四角格子空間上でエージェント間の距離を取得する。

measure_agt_distance_sqgrid(agt1, agt2, manhattan=False)

- **agt1** - 1 つ目のエージェント
- **agt2** - 2 つ目のエージェント
- **manhattan** - False のときは距離をチェビシェフ距離で、True のときはマンハッタン距離で定義する。初期値は False。

```
# エージェント taro と hanako の距離をチェビシェフ距離で取得する
distance = measure_agt_distance_sqgrid(taro, hanako)

# エージェント taro と hanako の距離をマンハッタン距離で取得する
distance = measure_agt_distance_sqgrid(taro, hanako, manhattan=True)
```

measure_distance

連続空間上で 2 地点間の距離を取得する。

measure_distance(x1, y1, x2, y2, space)

- **x1** - 1 地点目の x 座標
- **y1** - 1 地点目の y 座標
- **x2** - 2 地点目の x 座標
- **y2** - 2 地点目の y 座標
- **space** - 対象とする空間

※連続空間でのみ使用可能。

```
# 空間 Universe.city 上で座標(1, 1)と(25, 30)の距離を取得する
distance = measure_distance(1, 1, 25, 30, Universe.city)
```

measure_distance_sqgrid

四角格子空間上で 2 地点間の距離を取得する。

measure_distance_sqgrid(x1, y1, x2, y2, space, manhattan=False)

- **x1** - 1 地点目の x 座標
- **y1** - 1 地点目の y 座標
- **x2** - 2 地点目の x 座標
- **y2** - 2 地点目の y 座標
- **space** - 対象とする空間
- **manhattan** - False のときは距離をチェビシェフ距離で、True のときはマンハッタン距離で定義する。初期値は False。

```
# 空間 Universe.city 上で座標(1, 1)と(25, 30)の距離をチェビシェフ距離で取得する
distance = measure_distance(1, 1, 25, 30, Universe.city)

# 空間 Universe.city 上で座標(1, 1)と(25, 30)の距離をマンハッタン距離で取得する
distance = measure_distance(1, 1, 25, 30, Universe.city, manhattan=True)
```

specify_loop

指定した空間のループ設定を取得する。

specify_loop(space)

- **space** - 空間

```
# 空間 Universe.city がループする空間なら、d に True を代入、ループしない空間なら False を代入する
d = specify_loop(Universe.city)
```

specify_space_type

指定した空間の種別を取得する。（連続空間は 0、四角格子空間は 1 を返す）

specify_space_type(space)

- **space** - 空間

```
# 空間 Universe.city が連続空間なら d に 0 を代入、四角格子空間なら 1 を代入する
d = specify_space_type(Universe.city)
```

ファイル入出力

close

ファイルを閉じる。

close()

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/tutorial/inputoutput.html

```
# ファイルの中身の読み込み
f = open("hoge.txt")
data = f.read()
f.close()

# ファイルへの書き込み（上書き）
f = open("hoge.txt", mode='w')
f.write("hogehoge")
f.close()

# ファイルへの書き込み（追記）
f = open("hoge.txt", mode='a')
f.write("hogehoge")
f.close()
```

open

ファイルを開く。

open(file, mode='r')

- **file** - 読み込み対象のファイル名。
- **mode** - 読み込みモード。
  - 'a': 読み込み用に開く（デフォルト）
  - 'w': 書き込み用に開き、ファイルに上書きする
  - 'a': 書き込み用に開き、ファイル末尾に追記する

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/library/functions.html#open

```
# ファイルの中身の読み込み
f = open("hoge.txt")
data = f.read()
f.close()

# ファイルへの書き込み（上書き）
f = open("hoge.txt", mode='w')
f.write("hogehoge")
f.close()

# ファイルへの書き込み（追記）
f = open("hoge.txt", mode='a')
f.write("hogehoge")
f.close()
```

read

ファイルを読み込む。

read()

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/tutorial/inputoutput.html

```
# ファイルの中身の読み込み
f = open("hoge.txt")
data = f.read()
f.close()
```

write

ファイルへ書き込む。

write(string)

- **string** - 書き込む文字列

※Python 標準の関数です。詳しくは Python 公式ドキュメントを参照してください。
https://docs.python.org/ja/3/tutorial/inputoutput.html

```
# ファイルへの書き込み（上書き）
f = open("hoge.txt", mode='w')
f.write("hogehoge")
f.close()

# ファイルへの書き込み（追記）
f = open("hoge.txt", mode='a')
f.write("hogehoge")
f.close()
```

その他

exit_simulation

関数実行後、univ_finish を実行してシミュレーションを終了する。

exit_simulation()

```
exit_simulation() # これを実行したタイミングでシミュレーションを終了する
```

count_step

現在のステップ数を取得する。

count_step()

```
a = count_step() # 50 ステップ目の時、a に 50 を代入
```

gradation

グラデーション色を指定する。

gradation(start, end ,grad)

- **start** - 開始色
- **end** - 終了色
- **grad** - 度合い

```
# 赤と青の中間の色を取得する
c = gradation(COLOR_RED, COLOR_BLUE, 0.5)
```

hsv

「色相(Hue)」、「彩度(Saturation)」、「明度(Value)」の 3 つの引数から、色の RGB 値を返す。

hsv(h, s, v)

- **h** - 色相（0~360）
- **s** - 彩度（0~1）
- **v** - 明度（0~1）

```
c = hsv(0, 1, 1)  # 赤色を取得する
```

pause_simulation

関数を実行したシミュレーションステップ終了後にシミュレーションを一時停止状態にする。再生ボタンを押下することで再開される。 ※univ_init や univ_finish では利用できません。

```
# 10 ステップ目終了時にシミュレーションを一時停止
if count_step() == 10:
    pause_simulation()
```

print

コンソールに出力する。

print(sub)

- **sub** - 出力する対象

```
print("aaa")  #文字列 aaa が出力される
print(2)      #整数値 2 が出力される
```

rgb

RGB 色を取得する。

rgb(r, g, b)

- **r** - R 値
- **g** - G 値
- **b** - B 値

```
c = rgb(255, 0, 0)  # 赤色を取得する
```