説明書

実行方法

- Pythonインタープリタから実行する場合

  付属の.batファイルを実行することで実行できます. 自身の環境に, Pythonをインストールする必要があります. Python 3.8.10 で作成した為, 同様のPython を使用する事を推奨します. 下記のサイトにPythonの環境構築方法が記されています.
  https://www.python.jp/install/windows/install.html

  コマンドプロンプトやPowerShellの様なコマンドラインから実行する場合は, cdを用いてsrcディレクトリを移動した後, 下記のコマンドを実行する事で実行可能です. "ファイル名"の部分を実際のファイル名

  ```
  Python MasterWindow.py
  ```

ディレクトリ構成

```
.
├── Run.bat                 -> 実行補助のbatファイル. ※1
├── UserFile                -> 自身で作成したファイルを置くディレクトリ
│   ├── __pycache__         -> ※2
│   │   ├── sample1.cpython-38.pyc
│   │   ├── sample2.cpython-38.pyc
│   │   ├── sample3.cpython-38.pyc
│   │   ├── sample4.cpython-38.pyc
│   │   └── sample5.cpython-38.pyc
│   ├── sample1.py          -> サンプルファイル
│   ├── sample2.py
│   ├── sample3.py
│   ├── sample4.py
│   └── sample5.py
├── readme.html     -> 説明書 pdf と内容は同じです.
├── readme.md
├── readme.pdf      -> 説明書 htmlと内容は同じです.
└── src                     -> 今回作成したソースコードが置かれたディレクトリ
    ├── MasterWindow.py     -> メインプログラム
    └── lib                 -> サブプログラムの置かれたディレクトリ
        ├── __pycache__     -> -> ※2
        │   ├── move.cpython-38.pyc
        │   ├── normalize_path.cpython-38.pyc
        │   └── object.cpython-38.pyc
        ├── move.py         -> 亀や餌の動き方について記述されたファイル
        ├── normalize_path.py
        └── object.py       -> -> 画面, 亀, 餌に関するファイル
```

※1 ディレクトリを移動すると動きません. 移動したい場合は, ショートカットを作成してください.
※2 Pythonが実行時に作成したファイル群. 各種ファイルのコンパイルされたものが格納されている.

記述方法

まず, 関連ファイルのimportを行います. 下記の6行は必ず先頭に記述してください.

```
import sys
import os
tmppath = os.path.normpath(os.path.join(os.path.dirname(__file__), "../src/lib"))
sys.path.append(tmppath)
from lib.object import Feald, INF
from lib.move import Move
```

次にmain関数を記述します. 作成したファイルはmain関数が実行されます. その為, main関数は必ず記述する必要があります. main関数には, 以下の要素を記述する必要があります.

- 餌の座標のリスト
  - 座標は(x, y)の2次元です.
- 亀の座標のリスト
  - 餌と同様です.
- オプション
  - 餌のアルゴリズムの指定 -> Feeds_algo
  - 亀のアルゴリズムの指定 -> Turtle_algo
  - 距離関数の指定 -> DistanceFunction(str)
  - 射程の指定 -> sight
  - 移動速度の指定 -> Speed
  - 遅延時間の指定 -> Delay
  - Gridの有無の指定 -> Grid
  - 最大行動回数の指定 -> max-loop
- Feald オブジェクトの作成
  - 以下の引数を渡す必要があります.
  - feeds -> 餌の座標のリスト
  - turtles -> 亀の座標のリスト
  - options -> オプション
- Startメソッドの実行

```
def main():
    feed = [(50, 50)]
    kame = [(10, 10), (90, 90)]

    options = {
        # 移動アルゴリズム
        "Feeds_algo" : Move.RandomWalk,
        "Turtle_algo" : Move.HaveNose,

        # 距離関係
```

```
        "DistanceFunction" : "Euclidean",
        "sight" : 5,

        # 亀の移動速度
        "Speed" : 0,
        "Delay" : 0,        # 1 (s) = 1000 (ms)

        # グリッドの描写
        "Grid" : True,

        # 行動回数
        "max-loop" : INF,      # 整数 or INF
    }

    x = Feald(feeds=feed, turtles=kame, options=options)
    x.start()
```

ソースコードの解説

MasterWindow.py

メインプログラムです.

/lib/object.py

- Class Law -> 空間の広さを規定するクラスです.
    - クラス変数
        - x_lim -> xの範囲を指定します.
        - y_lim -> yの範囲を指定します.

- Class Feald -> 描写範囲とその内部を規定するクラスです. Lawを継承しています.
    - クラス変数
        - mode -> 未使用
        - turn -> 現在のターン数を記録
    - インスタンス変数
        - Feeds -> 餌のオブジェクトを格納したリスト
        - Turtles -> 亀のオブジェクトを格納したリスト
        - FeedsAlgo -> 餌の移動に用いる関数の参照
        - TurtleAlgo -> 亀の移動に用いる関数の参照
        - delay -> 1ターンごとの遅延時間を記録. (ms)
        - Options -> 入力したOptionへの参照
    - メソッド　※先頭に_がついているメソッドは内部に秘匿されている為, 外部からアクセス出来ません.

        - __init__ -> Fealdクラスのコンストラクタを呼びます. Fealdを作成した際に最初に必ず実行されるメソッドです.
            - 入力：
              feeds -> 餌の座標を格納したリスト
              turtles -> 亀の座標を格納したリスト
              options -> 移動の関数や遅延時間などを格納した辞書
            - 出力：なし
        - __wall -> 壁を描写するメソッドです.
            - 入力：なし
            - 出力：なし
        - __grid -> Gridを描写します. x, y共に100分割します.
            - 入力：なし
            - 出力：なし
        - init_colors -> 亀に使用する色のリストを初期化します.
            - 入力：なし
            - 出力：なし
        - start -> 指定ターン数若しくは餌が無くなるまで実行します.
            - 入力：なし
            - 出力：なし
        - play -> 1ターンの処理を行います.
            - 入力：なし
            - 出力：なし
        - eat_check -> 餌と亀が重なった際の判定と処理を行います.
            - 入力：なし
            - 出力：なし
        - end_check -> 餌が残っているかをboolで返します. 終了判定に用います.
            - 入力：なし
            - 出力：bool

- Class Feed -> 餌のクラス. Lawを継承しています.
    - インスタンス変数
    - メソッド
        - __init__ -> Feedクラスのコンストラクタを呼びます. 座標が範囲内に収まっているか等の確認を行います.

        - 入力：
          x -> int
          y -> int
          op -> 移動の関数や遅延時間などを格納した辞書.

    - draw_border -> 境界線を描写するメソッド.

        - 入力：なし
        - 出力：なし

    - run -> 移動を行います.

        - 入力：
          algo -> Moveクラスのメソッドの参照.
          options -> 移動の関数や遅延時間などを格納した辞書.
        - 出力：なし

- Class Turtle -> 亀のクラス. Lawを継承しています.
    - メソッド

        - __init__ -> Turtleクラスのコンストラクタを呼びます. Feed同様に座標が範囲内に収まっているかを確認します.

            - 入力：
              x -> int
              y -> int
              op -> 移動の関数や遅延時間などを格納した辞書.
            - 出力：なし

        - _individual_colors -> 亀に個々に別々の色を割り振ります. 規定の配色が無くなった場合は, ランダムに色が設定されます.

            - 入力：なし
            - 出力：str or (float, float, float) -> 色の文字列 or RGB

        - run -> 移動を行います.

            - 入力：
              algo -> Moveクラスのメソッドの参照 option -> 移動の関数や遅延時間などを格納した辞書.
            - 出力：なし

/lib/move.py

- Class Move -> 移動に関するアルゴリズムを管理するクラスです.
  外部からアクセス可能なメソッドの入出力は全て共通です.
  入力： T -> turtleオブジェクト option -> 移動の関数や遅延時間などを格納した辞書.
  出力：int (0〜3)

    - stay -> 移動しない

- RandomWalk -> ランダムに行動します.
- HaveNose -> 一方が匂いを放ち, 一方がそれを検知する鼻を持っている想定の関数です. 餌が匂いを放ち, 亀が検知する事を想定しています.
- _euclidean -> 2点間のユークリッド距離を計算します.
    - 入力：p1 -> (x:int, y:int), p2 -> (x:int, y:int)
    - 出力：float
- _manhattan -> 2点間のマンハッタン距離を計算します.
    - 入力：p1 -> (x:int, y:int), p2 -> (x:int, y:int)
    - 出力：float