```
# -*- coding: utf-8 -*-
"""
Langton's Ant Simulation

Created on 2021-04-25

@author: Richard Wainwright, 40126812

Langton's Ant is Turing complete because the ant can move back and forth
and can read and write.  Using this behaviour you can set up an initial
state for the ant to simulate logic gates and have the ant run any program
for you.
"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation


class Ant:
    """
    Ant object to interact with lattice
    """
    def __init__(self, x, y, direction):
        self.direction = direction
        self.x = x
        self.y = y

    def turn_right(self):
        self.direction = {
            'N': 'E',
            'E': 'S',
            'S': 'W',
            'W': 'N'
        }[self.direction]

    def turn_left(self):
        self.direction = {
            'N': 'W',
            'E': 'N',
            'S': 'E',
            'W': 'S'
        }[self.direction]

    def move(self):
        if self.direction == 'N':
            self.y += 1
        elif self.direction == 'E':
```

```
            self.x += 1
        elif self.direction == 'S':
            self.y -= 1
        else:
            self.x -= 1


class LangAnt:
    """
    Object for lattice to demonstrate Langton's Ant
    """
    def __init__(self, ant, rules, N=256, finite=False, fastMode=False):
        self.grid = np.zeros((N, N), np.uint)
        self.finite = finite
        self.fastMode = fastMode
        self.N = N
        self.ant = ant
        self.rules = rules

    def getGrid(self):
        return self.grid

    def step(self):
        """
        Have the ant turn and move according to the rules
        """
        if self.ant.x < 0 or self.ant.y < 0 or \
                self.ant.x >= self.N or self.ant.y >= self.N:
            return self.grid

        new_grid = self.getGrid()

        if self.rules[new_grid[self.ant.x][self.ant.y]] == 'R':
            self.ant.turn_right()
        else:
            self.ant.turn_left()

        new_grid[self.ant.x][self.ant.y] = \
            (new_grid[self.ant.x][self.ant.y] + 1) % len(self.rules)

        self.ant.move()


"""
rules strings: LR = basic ant
               RLR = Chaos
               LRRRRRLLR = fills grid
               RLRLRLRLRLR = quicker highway
```

```
"""
# get user input to create dictionary of ant movement rules
accept = "RL"
rule_string = ""
while rule_string == "" or not all(c in accept for c in rule_string):
    rule_string = input("Enter rule string: ")

rules = dict((k, v) for k, v in enumerate(rule_string))
num_rules = len(rule_string)

# create ant and lattice
N = 128
midpoint = N // 2
ant = Ant(midpoint, midpoint, 'W')
lattice = LangAnt(ant, rules, N)

cells = lattice.getGrid()  # initial state

# plot cells
fig = plt.figure()

img = plt.imshow(cells, cmap="gnuplot2", vmin=0, vmax=(num_rules - 1),
                 animated=True)


def animate(i):
    """perform animation step"""
    global lattice

    lattice.step()
    cells_updated = lattice.getGrid()

    img.set_array(cells_updated)

    return img,


interval = 0  # ms

# animate 24 frames with interval, calling animate function at each frame
ani = animation.FuncAnimation(fig, animate, frames=24, interval=interval,
                              blit=True)

plt.show()
```