Name: "'Bosheng Zhang"'
SID: "'s4500483"'

```python
"""
Langton's ant is a two-dimensional universal Turing machine with a very simple set of rules
but complex emergent behavior.
In simplest term, it's just a cellular automaton like Conway's Game of Life.
The ant moves in the grid(cells) following very simple rules.
The ant has certain orientation in the cell: up, down, left, right which is used for turning the direction of the ant.

IMPROVEMENT ON DIFFERENT COLORS
The ant might go back to the square 1, and the square shouldn't change from 1 -> 0,
it should change from 1 -> 2, as same as 2 -> 3... and so on.
Each number should represent a different colour.
"""

import numpy as np
import random

WHITE = 0
BLACK = 1

# LR: Langton's ant has the name "RL" in this naming scheme.
SIMPLEST = {
    0: (1, 'R'),
    1: (0, 'L'),
}

# RLR: Grows chaotically. It is not known whether this ant ever produces a highway.
CHAOTIC = {
    0: (1, 'R'),
    1: (2, 'L'),
    2: (0, 'R')
}
# LLRR: Grows symmetrically.
SYMMETRIC = {
    0: (1, 'L'),
    1: (2, 'L'),
    2: (3, 'R'),
    3: (0, 'R')
}
# LRRRRRLLR: Fills space in a square around itself.
SQUARE = {
    0: (1, 'L'),
    1: (2, 'R'),
    2: (3, 'R'),
    3: (4, 'R'),
    4: (5, 'R'),
    5: (6, 'R'),
    6: (7, 'L'),
    7: (8, 'L'),
```

```python
    8: (0, 'R')
}
# LLRRRLRLRLLR: Creates a convoluted highway.
CONVOLUTED_HIGHWAY = {
    0: (1, 'L'),
    1: (2, 'L'),
    2: (3, 'R'),
    3: (4, 'R'),
    4: (5, 'R'),
    5: (6, 'L'),
    6: (7, 'R'),
    7: (8, 'L'),
    8: (9, 'R'),
    9: (10, 'L'),
    10: (11, 'L'),
    11: (0, 'R'),
}
# RRLLLRLLLRRR: Creates a filled triangle shape that grows and moves.
FILLED_TRIANGLE = {
    0: (1, 'R'),
    1: (2, 'R'),
    2: (3, 'L'),
    3: (4, 'L'),
    4: (5, 'L'),
    5: (6, 'R'),
    6: (7, 'L'),
    7: (8, 'L'),
    8: (9, 'L'),
    9: (10, 'R'),
    10: (11, 'R'),
    11: (0, 'R'),
}
# direction index set
DIRECTIONS = {
    'U': (0, 1),
    'D': (0, -1),
    'L': (-1, 0),
    'R': (1, 0)
}
# here we initial the direction as up->right->down->left, which is clockwise
directions = 'URDL'


class LangtonAnt:
    """
    Object for computing langton's ant cellular automation
    """
    def __init__(self, N, ant_position, rules):
        self.grid = np.zeros((N, N), np.uint)
        self.rules = rules
        self.ant_position = ant_position
        self.ant_direction = random.choice(directions)
        print(self.ant_direction)

    def get_states(self):
```

```python
        """Returns the current states of the cells"""
        return self.grid

    def get_current_position(self):
        """Returns the ant current position"""
        return self.grid[self.ant_position]

    def set_current_position(self, num):
        """set ant current position base on input num"""
        self.grid[self.ant_position] = num

    def rotate(self, direc):
        """rotate the ant dependents on the direc: L->90 clockwise; R->90 anti-clockwise"""
        # At a white square, turn 90   clockwise
        if direc == 'R':
            index = 1
        # At a black square, turn 90   counter-clockwise
        if direc == 'L':
            index = -1
        self.ant_direction = directions[(directions.find(self.ant_direction) + index) % len(directions)]

    def move(self):
        # move forward one unit
        index = DIRECTIONS[self.ant_direction]
        self.ant_position = (
            self.ant_position[0] + index[0],
            self.ant_position[1] + index[1]
        )

    def update(self):
        """update one epoch"""
        current_position = self.get_current_position()
        # locate current position and read from input transition table
        transition = self.rules[current_position]
        # get next position index and direction: L OR R
        new_position, direc = transition

        # flip the color of the square
        self.set_current_position(new_position)
        self.rotate(direc=direc)
        self.move()


# ----------------------------------------------------------------------

N = 256

n = int(input("choose the ruleset of ant: (0 - 5)"))
ruleset = {
    0: SIMPLEST,
    1: CHAOTIC,
    2: SYMMETRIC,
```

```
    3: SQUARE,
    4: CONVOLUTED_HIGHWAY,
    5: FILLED_TRIANGLE
}[n]

# create the langton ant object
ant = LangtonAnt(N, ant_position=(int(N / 2), int(N / 2)), rules=ruleset)
cells = ant.get_states()  # initial state

# ----------------------------------------------------------------------
# plot cells
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig = plt.figure()

plt.gray()

img = plt.imshow(cells, animated=True, cmap='tab20c', vmin=0, vmax=(len(ruleset) - 1))


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

    ant.update()
    cells_updated = ant.get_states()

    img.set_array(cells_updated)

    return img,


interval = 0.1  # ms

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

plt.show()
```