```
 1: from functools import partial
 2: from typing import List
 3: from time import sleep
 4: import sys
 5:
 6: class Board(object):
 7:     def __init__(self, h: int, w: int):
 8:         super().__init__()
 9:         self.h = h
10:         self.w = w
11:         self.m = [[Cell(i, j) for j in range(h)] for i in range(w) ]
12:
13:     def populate(self):
14:         self.m = [[cell.link(self.m) for cell in row] for row in self.m]
15:         return self
16:
17:     def __str__(self) -> str:
18:
19:         s = ""
20:         for row in self.m:
21:             l = []
22:             for cell in row:
23:                 l.append(str(cell.v))
24:
25:             s += " ".join(l) + "\n"
26:
27:         return s
28:
29:
30: class Cell:
31:     def __init__(self, x: int, y: int):
32:         self.x = x
33:         self.y = y
34:         self.v = 0
35:
36:     def link(self, board: List):
37:         nix = partial(neg_index, len(board[0]))
38:         niy = partial(neg_index, len(board))
39:         self.north = board[self.y - 1][self.x]
40:         self.south = board[niy(self.y + 1)][self.x]
41:         self.east = board[self.y][nix(self.x + 1)]
42:         self.west = board[self.y][self.x - 1]
43:
44:         return self
45:
46: class Vector2(object):
47:     def __init__(self, x, y) -> None:
48:         super().__init__()
```

```
49:         self.x = x
50:         self.y = y
51:
52: def neg_index(offset: int, n: int):
53:     return (n - offset)
54:
55: # %%
56: def apply_rules(cell: Cell):
57:     cell.v -= 1
58:     cell.north.v += 1
59:     cell.south.v += 1
60:     cell.east.v += 1
61:     cell.west.v += 1
62:
63:     if cell.v > 21:
64:         cell.v = 0
65:
66: def select_cells(board: Board):
67:     to_apply = []
68:     for row in board.m:
69:         for cell in row:
70:             if cell.v > 0:
71:                 to_apply.append(Vector2(cell.x, cell.y))
72:     return to_apply
73:
74:
75: # %%
76: def simulate(gen: int, board: Board):
77:     nbm = board.m.copy()
78:
79:     for i in range(gen):
80:         changes = select_cells(board)
81:
82:
83:     for cell in changes:
84:         apply_rules(nbm[cell.x][cell.y])
85:
86:     nb = Board(board.h, board.w)
87:     nb.m = nbm
88:     return nb
89:
90: # %%
91: RESET = '\033[0m'
92:
93: def get_color_escape(r, g, b, background=True):
94:     return '\033[{};2;{};{};{}m'.format(48 if background else 38, r, g, b)
95:
96: def val_to_color(v):
```

```
    color = ""
    if v < 0:
        color = get_color_escape(0, 0, 0) + " "
    elif v < 3:
        color = get_color_escape(0, 0, v * 20) + " "
    elif v < 6:
        color = get_color_escape(0, v*10, v*20) + " "
    elif v < 9:
        color = get_color_escape(0, v*20, v * 10) + " "
    elif v < 12:
        color = get_color_escape(v*10, v*20 ,0) + " "
    elif v < 15:
        color = get_color_escape(v*20, v*10, 0) + " "
    elif v < 18:
        color = get_color_escape(v*20, 0, v*100) + " "
    elif v < 21:
        color = get_color_escape(v*10, v*10, v*10) + " "
    elif v > 21:
        color = get_color_escape(255, 0, 255) + " "

    color += RESET

    return color

def color_matrix(matrix):
    colored = [[0 for cell in row] for row in matrix]
    i = 0
    for row in matrix:
        j = 0
        for cell in row:
            colored[i][j] = val_to_color(cell.v)
            j+=1
        i+=1

    return colored
def text_render(board: Board):
    for line in color_matrix(board.m):
        print(''.join(map(str,line)))

def life(gen):
    matrix = Board(51, 51).populate()
    matrix.m[20][30].v = 21

    for i in range(gen):
        text_render(simulate(1, matrix))
        sleep(0.2)
        print("\033[2J\033[1;1H")

```

```
145: def main(argv):
146:     if len(argv) != 2:
147:         print(len(argv))
148:         print("Usage: ", argv[0], " <number-of-generations>")
149:         exit(1)
150:     else:
151:         life(int(argv[1]))
152:         exit(0)
153:
154: if __name__ == "__main__":
155:     main(sys.argv)
```