```
#!/usr/bin/env python3
# this solution by Abbas Masoumzadeh Tork
from paint import paint
import sys
from time import sleep
import numpy as np
"""
# Sample format to answer pattern questions
# assuming the pattern would be frag0:
..*
**.
.**
#############################################
# Reread the instructions for assignment 1: make sure
# that you have the version with due date SUNDAY.
# Every student submits their own assignment.
* Delete the names and ccids below, and put
# the names and ccids of all members of your group, including you.
# name                         ccid
#############################################
# Your answer to question 1-a:
Glider:
........................................
........................................
........................................
.....................*..................
......................*.................
....................***.................
........................................
........................................
........................................
#############################################
# Your answer to question 1-b:
Glider gun:
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
............................**.....................................
...........................*.......................................
..........................*.............*..........................
.................*........*............**..........................
................**........*...............**.......................
...........................*..............***......*...............
............................**............**......**...............
.......................................**..........................
........................................*..........................
...................................................................
...................................................................
...................................................................
...................................................................
#############################################
# Your answer to question 2:
Although both programs address the same task, they use a different
approach to represent the board. For life.py a row-major order is
used to represent a 2D board which also utilizes guard cells.
But life-np.py uses a 2D array for the board and does not consider
guard cells.
- Part 1:
In life.py, checking for the number of alive neighbors is done
using a separate function called num_nbrs. However, in life-np.py
the process is integrated within the next_state function.
- Part 2:
```

```
life.np utilizes a function called pad. It checks whether there
is any alive cell on the borders. If so, then it adds a free
row or column there to avoid collision with the guard cells.
This potentially results in having an infinite grid, if needed.
- Part 3:
Using a guarded board, along with padding, eases the hassle
of counting number of alive neighbors as there is no need to
worry about the cell being on the boundaries of the board.
#############################################
# Follow the assignment 1 instructions and
# make the changes requested in question 3.
# Then come back and fill in the answer to
# question 3-c:
Assuming a secret number of 100:
...........*.
............*
..........***
.............
.............
#############################################
"""
"""
based on life-np.py from course repo
"""


PTS = '.*#'
DEAD, ALIVE, WALL = 0, 1, 2
DCH, ACH, GCH = PTS[DEAD], PTS[ALIVE], PTS[WALL]


def point(r, c, cols): return c + r*cols

"""
board functions
  * represent board as 2-dimensional array
"""


def get_board():
    B = []
    print(sys.argv[1])
    with open(sys.argv[1]) as f:
        for line in f:
            B.append(line.rstrip().replace(' ', ''))
        rows, cols = len(B), len(B[0])
        for j in range(1, rows):
            assert(len(B[j]) == cols)
        return B, rows, cols


def convert_board(B, r, c):  # from string to numpy array
    A = np.zeros((r, c), dtype=np.int8)
    for j in range(r):
        for k in range(c):
            if B[j][k] == ACH:
                A[j, k] = ALIVE
    return A


def expand_grid(A, r, c, t):  # add t empty rows and columns on each side
    N = np.zeros((r+2*t, c+2*t), dtype=np.int8)
    for j in range(r):
        for k in range(c):
            if A[j][k] == ALIVE:
```

```
                N[j+t, k+t] = ALIVE
    return N, r+2*t, c+2*t


def print_array(A, r, c):
    print('')
    for j in range(r):
        out = ''
        for k in range(c):
            out += ACH if A[j, k] == ALIVE else DCH
        print(out)


def show_array(A, r, c):
    for j in range(r):
        line = ''
        for k in range(c):
            line += str(A[j, k])
        print(line)
    print('')


"""
Conway's next-state formula
"""


def next_state(A, r, c):
    N = np.zeros((r, c), dtype=np.int8)
    changed = False
    for j in range(r):
        for k in range(c):
            num = 0
            if j > 0 and k > 0 and A[j-1, k-1] == ALIVE:
                num += 1
            if j > 0 and A[j-1, k] == ALIVE:
                num += 1
            if j > 0 and k < c-1 and A[j-1, k+1] == ALIVE:
                num += 1
            if k > 0 and A[j, k-1] == ALIVE:
                num += 1
            if k < c-1 and A[j, k+1] == ALIVE:
                num += 1
            if j < r-1 and k > 0 and A[j+1, k-1] == ALIVE:
                num += 1
            if j < r-1 and A[j+1, k] == ALIVE:
                num += 1
            if j < r-1 and k < c-1 and A[j+1, k+1] == ALIVE:
                num += 1
            if A[j, k] == ALIVE:
                if num > 1 and num < 4:
                    N[j, k] = ALIVE
                else:
                    N[j, k] = DEAD
                    changed = True
            else:
                if num == 3:
                    N[j, k] = ALIVE
                    changed = True
                else:
                    N[j, k] = DEAD
    return N, changed


#############################################
```

```
"""
Provide your code for the function
next_state2 that (for the usual bounded
rectangular grid) calls the function num_nbrs2,
and delete the raise error statement:
"""
def next_state2(A, r, c):
    A2 = np.zeros((np.shape(A)[0]+2, np.shape(A)[1]+2))
    A2[1:r+1, 1:c+1] = A

    N = np.zeros((r, c), dtype=np.int8)
    changed = False
    for j in range(r):
        for k in range(c):
            num = num_nbrs2(A2, j+1, k+1)
            if A[j, k] == ALIVE:
                if num > 1 and num < 4:
                    N[j, k] = ALIVE
                else:
                    N[j, k] = DEAD
                    changed = True
            else:
                if num == 3:
                    N[j, k] = ALIVE
                    changed = True
                else:
                    N[j, k] = DEAD
    return N, changed
#############################################


#############################################
"""
Provide your code for the function
num_nbrs2 here and delete the raise error
statement:
"""
def num_nbrs2(A2, j, k):
    return np.sum(A2[j-1:j+2, k-1:k+2]) - A2[j,k]
#############################################


#############################################
"""
Provide your code for the function
next_state_torus here and delete the raise
error statement:
"""
def next_state_torus(A, r, c):
    A2 = np.zeros((np.shape(A)[0]+2, np.shape(A)[1]+2))
    A2[1:r+1, 1:c+1] = A
    A2[0, 1:c+1] = A[r-1, :]
    A2[r+1, 1:c+1] = A[0, :]
    A2[1:r+1, 0] = A[:, c-1]
    A2[1:r+1, c+1] = A[:, 0]
    A2[0,0] = A[r-1, c-1]
    A2[r+1,0] = A[0, c-1]
    A2[0,c+1] = A[r-1, 0]
    A2[r+1,c+1] = A[0,0]

    N = np.zeros((r, c), dtype=np.int8)
    changed = False
    for j in range(r):
        for k in range(c):
            num = num_nbrs2(A2, j+1, k+1)
```

```
            if A[j, k] == ALIVE:
                if num > 1 and num < 4:
                    N[j, k] = ALIVE
                else:
                    N[j, k] = DEAD
                    changed = True
            else:
                if num == 3:
                    N[j, k] = ALIVE
                    changed = True
                else:
                    N[j, k] = DEAD
    return N, changed
#############################################


#############################################
"""
Provide your code for the function
num_nbrs_torus here and delete the raise
error statement:
"""
def num_nbrs_torus(A2, j, k):
    return num_nbrs2(A2, j, k)
#############################################


"""
input, output
"""

pause = 0.2

#############################################
"""
Modify interact as necessary to run the code:
"""
#############################################


def interact(max_itn):
    itn = 0
    B, r, c = get_board()
    print(B)
    X = convert_board(B, r, c)
    A, r, c = expand_grid(X, r, c, 0)
    print_array(A, r, c)
    while itn <= max_itn:
        sleep(pause)
        newA, delta = next_state_torus(A, r, c)
        if not delta:
            break
        itn += 1
        A = newA
        print_array(A, r, c)
    print('\niterations', itn)


def main():
    interact(100-1)


if __name__ == '__main__':
    main()
```