Alt_Game_of_Life

September 9, 2020

[1]:
```
import numpy as np
import unittest
import inspect
```

[5]:
```
def update_state(matrix):
    padded_matrix = np.pad(matrix, pad_width=(1, 1), mode='constant',␣
 ↪constant_values=0)
    output_matrix = np.zeros_like(matrix)

    row, col = padded_matrix.shape
    for i in range(1, row-1):
        for j in range(1, col-1):
            neighbor_sum = padded_matrix[i-1:i+2, j-1:j+2].sum()
            if padded_matrix[i][j] == 1:
                # Any live cell with fewer than two live neighbours dies, as if␣
 ↪by underpopulation.
                # Any live cell with more than three live neighbours dies, as if␣
 ↪by overpopulation.
                if not neighbor_sum-1 in range(2, 4):
                    output_matrix[i-1, j-1] = 0
               # Any live cell with two or three live neighbours lives on to the␣
 ↪next generation.
                else:
                    output_matrix[i-1, j-1] = 1
            else:
                # Any dead cell with exactly three live neighbours becomes a␣
 ↪live cell, as if by reproduction.
                if neighbor_sum == 3:
                    output_matrix[i-1, j-1] = 1
                else:
                    output_matrix[i-1, j-1] = 0
    return output_matrix
```

[ ]:
```
def validate(expected, output, msg):
    try:
        np.testing.assert_array_equal(expected, output)
        print ("Test Case Successful: %s" % msg)
```

```
    except ex:
        print (ex)
        print ("Test Case Failed: %s" % msg)
    return

def test_case_1():
    msg = inspect.currentframe().f_code.co_name
    input_state = np.array([[1, 0, 0, 0], [0, 1, 1, 0], [0, 1, 1, 0], [0, 0, 0,␣
 ↪0]])
    expected_output_state = np.array([[0, 1, 0, 0], [1, 0, 1, 0], [0, 1, 1, 0],␣
 ↪[0, 0, 0, 0]])
    validate(expected_output_state, update_state(input_state), msg)

def test_case_2():
    msg = inspect.currentframe().f_code.co_name
    input_state = np.array([[0, 1, 1, 0], [1, 1, 1, 0], [0, 1, 1, 0], [0, 0, 0,␣
 ↪0]])
    expected_output_state = np.array([[1, 0, 1, 0], [1, 0, 0, 1], [1, 0, 1, 0],␣
 ↪[0, 0, 0, 0]])
    validate(expected_output_state, update_state(input_state), msg)

def execute_test_cases():
    test_case_1()
    test_case_2()
```

[12]:
```
execute_test_cases()
```

```
Test Case Successful: test_case_1
Test Case Successful: test_case_2
```