day11

December 12, 2020

1 Day 11 Like a game of life

This is a much simplified version of Conways game of life, a coding pattern that we used for years at the Guardian when interviewing individuals. This is a little easier, because we have fewer rules, and no concerns around infinite or wrapping maps or anything.

The hardest part here, from experience, is working out how to count the neighbours of a cell. There's lots of ways to do this, but since I've done it before it shouldn't be too bad.

The key for finding the count of the neighbours is to work out how we are going to address the cells. My games experience makes me tend to address cells by cartesian coordinates, x and y. That means to find neighbours we want to find the cells addressed by x-1, y-1 through to x+1, y+1.

Secondly, we need to handle cells at the edge of the map. Instead of reaching directly into the data structure, we call getCell(x,y) which if we address outside the board, will just always return false.

Our third oddity here compared to Conways Game of Life is that not all cells can be occupied. Some cells have a seat, and can either be occupied, or empty, or floor. We'll represent those as a tristate variable, 0 = floor, 1=emptyseat, 2=occupied.

Lets go

[20]:
```
import ipytest
ipytest.autoconfig()

from collections import defaultdict

FLOOR=0
EMPTY=1
OCCUPIED=2

def read_map(data):
    seats = defaultdict(int)
    seats["height"]=len(data)
    for y,line in enumerate(data):
        for x,c in enumerate(line.strip()):
            if c == "L":
                seats[(x,y)]=EMPTY
            if c == "#":
                seats[(x,y)]=OCCUPIED
```

```
    seats["width"]=len(line.strip())
    return seats

def get_cell(seats, x,y):
    if x < 0 or x >= seats["width"] or y < 0 or y >= seats["height"]:
        return FLOOR
    return seats[(x,y)]

testdata = """L.LL.LL.LL
LLLLLLL.LL
L.L.L..L..
LLLL.LL.LL
L.LL.LL.LL
L.LLLLL.LL
..L.L.....
LLLLLLLLLL
L.LLLLLL.L
L.LLLLL.LL"""

tests = [
    [(0,0),EMPTY],
    [(0,1),EMPTY],
    [(0,2),EMPTY],
    [(1,0),FLOOR],
    [(2,0),EMPTY],
    [(-1,0),FLOOR],
    [(0,-1),FLOOR]
]

testseats = read_map(testdata.split("\n"))
for test in tests:
    assert get_cell(testseats, test[0][0], test[0][1]) == test[1],␣
 ↪f"get_cell({test[0]}) should equal {test[1]}"
```

Great, we've got a map, and we can look at it. Let's start working out if we can count neighbours

[25]:
```
def count_neighbours(seats, x, y):
    neighbours = [(-1,-1), (0,-1), (1,-1), (-1,0), (1,0), (-1,1), (0,1), (1,1)]
    count = 0
    for (dx,dy) in neighbours:
        if get_cell(seats,x+dx,y+dy) == OCCUPIED:
            count += 1
    return count

testdata = """L.LL.LL.LL
LLLLLLL.LL
L.L.L..L..
```

```
LLLL.LL.LL
L.LL.LL.LL
L.L###L.LL
..L.L.....
LLLLLLLLLL
L.LLLLLL.L
L.LLLLL.LL"""
testseats = read_map(testdata.split("\n"))

assert count_neighbours(testseats, 0, 0) == 0
assert count_neighbours(testseats, 3, 5) == 1
assert count_neighbours(testseats, 4, 5) == 2
assert count_neighbours(testseats, 5, 5) == 1
```

Ok, time to handle swapping seats around. The key thing here is to not modify the map as we go, instead we build a new map. That way we don't count neighbours that are half from the previous iteration and half from the next iteration.

To work out the next step, our iterate function should return not just the new map, but should return a true/false to say whether or not any thing changed.

[37]:
```
def iterate(seats):
    changed = False
    newseats = seats.copy()
    for y in range(seats["height"]):
        for x in range(seats["width"]):
            neighbours = count_neighbours(seats, x, y)
            if neighbours == 0 and seats[(x,y)] == EMPTY:
                newseats[(x,y)] = OCCUPIED
                changed = True
            if neighbours >= 4 and seats[(x,y)] == OCCUPIED:
                newseats[(x,y)] = EMPTY
                changed = True
    return newseats, changed

def countseats(seats):
    return len([k for k in seats.keys() if seats[k] == OCCUPIED])

testdata = """L.LL.LL.LL
LLLLLLL.LL
L.L.L..L..
LLLL.LL.LL
L.LL.LL.LL
L.LLLLL.LL
..L.L.....
LLLLLLLLLL
L.LLLLLL.L
L.LLLLL.LL"""
```

```
testseats = read_map(testdata.split("\n"))

assert get_cell(testseats, 0,0) == EMPTY
assert get_cell(testseats, 1,1) == EMPTY

testseats,changed = iterate(testseats)
assert changed
assert get_cell(testseats, 0,0) == OCCUPIED
assert get_cell(testseats, 1,1) == OCCUPIED

testseats,changed = iterate(testseats)
assert changed
assert get_cell(testseats, 0,0) == OCCUPIED
assert get_cell(testseats, 1,1) == EMPTY

testseats,changed = iterate(testseats)
assert changed
testseats,changed = iterate(testseats)
assert changed
testseats,changed = iterate(testseats)
assert changed

testseats,changed = iterate(testseats)
assert not changed
assert 37 == countseats(testseats)
```

Now we can try on real data and see how long it takes until the iterations stop

[38]:
```
seats = read_map(open("day11.txt").readlines())
changed = True
iterations = 0
while changed:
    seats,changed = iterate(seats)
    iterations += 1
print(iterations)
print(countseats(seats))
```

```
76
2265
```

1.1 Part 2 - Line of sight

Ok, so this should be pretty simple from what we've done so far, but instead of count_neighbours only caring about the range -1 to 1 in each direction, we just need to look for a seat, occupied or not in the direction.

We'll now need to know if we went off hte edge of the map, so get_cell will change to return a special value if we're out of bounds as well

A simple change to count neighbours should allow us to replicate the tests, and then a change to the rules in iterate and we should be able to go again

[47]:
```
VOID = 3

def get_cell(seats, x,y):
    if x < 0 or x >= seats["width"] or y < 0 or y >= seats["height"]:
        return VOID
    return seats[(x,y)]

def count_neighbours(seats, x, y):
    neighbours = [(-1,-1), (0,-1), (1,-1), (-1,0), (1,0), (-1,1), (0,1), (1,1)]
    count = 0
    for (dx,dy) in neighbours:
        tx,ty = x,y
        found = False
        while not found:
            tx,ty = tx+dx, ty+dy
            c = get_cell(seats,tx,ty)
            if c == OCCUPIED:
                count += 1
                found = True
            if c == VOID or c == EMPTY:
                found = True
    return count

testdata = """.......#.
...#.....
.#.......
.........
..#L....#
....#....
.........
#........
...#....."""
testseats = read_map(testdata.split("\n"))

assert count_neighbours(testseats, 3, 4) == 8

testdata = """.............
.L.L.#.#.#.#.
............."""
testseats = read_map(testdata.split("\n"))

assert count_neighbours(testseats, 2, 1) == 0

testdata = """.##.##.
```

```
#.#.#.#
##...##
...L...
##...##
#.#.#.#
.##.##."""
testseats = read_map(testdata.split("\n"))

assert count_neighbours(testseats, 3, 3) == 0
```

Ok, so our neighbour check works, let's tweak iterate and go again on our original test data

[49]:
```
def iterate(seats):
    changed = False
    newseats = seats.copy()
    for y in range(seats["height"]):
        for x in range(seats["width"]):
            neighbours = count_neighbours(seats, x, y)
            if neighbours == 0 and seats[(x,y)] == EMPTY:
                newseats[(x,y)] = OCCUPIED
                changed = True
            if neighbours >= 5 and seats[(x,y)] == OCCUPIED:
                newseats[(x,y)] = EMPTY
                changed = True
    return newseats, changed

def countseats(seats):
    return len([k for k in seats.keys() if seats[k] == OCCUPIED])

testdata = """L.LL.LL.LL
LLLLLL.LL
L.L.L..L..
LLLL.LL.LL
L.LL.LL.LL
L.LLLLL.LL
..L.L.....
LLLLLLLLLL
L.LLLLL.L
L.LLLLL.LL"""
testseats = read_map(testdata.split("\n"))

assert get_cell(testseats, 0,0) == EMPTY
assert get_cell(testseats, 2,0) == EMPTY
assert get_cell(testseats, 3,0) == EMPTY

testseats,changed = iterate(testseats)
assert changed
```

```
assert get_cell(testseats, 0,0) == OCCUPIED
assert get_cell(testseats, 2,0) == OCCUPIED
assert get_cell(testseats, 3,0) == OCCUPIED

testseats,changed = iterate(testseats)
assert changed
assert get_cell(testseats, 0,0) == OCCUPIED
assert get_cell(testseats, 2,0) == EMPTY
assert get_cell(testseats, 3,0) == EMPTY

testseats,changed = iterate(testseats)
assert changed
assert get_cell(testseats, 0,0) == OCCUPIED
assert get_cell(testseats, 2,0) == EMPTY
assert get_cell(testseats, 3,0) == OCCUPIED

testseats,changed = iterate(testseats)
assert changed
testseats,changed = iterate(testseats)
assert changed
testseats,changed = iterate(testseats)
assert changed

testseats,changed = iterate(testseats)
assert not changed
assert 26 == countseats(testseats)
```

Ok, let's try on real data!

[50]:
```
seats = read_map(open("day11.txt").readlines())
changed = True
iterations = 0
while changed:
    seats,changed = iterate(seats)
    iterations += 1
print(iterations)
print(countseats(seats))
```

```
86
2045
```