Heat map cellular simulation

In the cellular approaches we have seen so far, we have spread a property from one cell to its neighbours based on a set of simple rules. There are also situations where a grid might contain actors which move around according to individual behaviour.

One example is evacuation. We can adapt our approach to model evacuation by using a heat map.

Our model is again a grid of cells, this time representing a space from which people wish to evacuate. We populate this with a number of people and some exits. The behaviour we wish to simulate is that people move to the nearest exit.

One way to do this is to have a separate grid which assigned a score to each cell based on how far it is from the exit. Then we can instruct people to move to an adjacent cell with a lower exit score than the current cell.

First, let's import packages.

```
from tkinter import *
from tkinter.ttk import *
from time import sleep
from random import randint
```

Now we can set up a list to store the positions of our people. We'll store `0` for an empty cell and `1` for a person. To start, let's place around 50 people in the middle of the grid.

```
grid = []
for r in range(50):
    thisrow = []
    for c in range(70):
        thisrow.append(0)
    grid.append(thisrow)
for person in range(50):
    grid[randint(15,35)][randint(15,55)] = 1
```

Now we need a list to store our heat map. To start, let's set all cells to have a score of `-1`, which just means they haven't been scored yet.

```
heatmap = []
for r in range(50):
    thisrow = []
    for c in range(70):
        thisrow.append(-1)
    heatmap.append(thisrow)
```

Now, let's position an exit with score `0` in the middle of the right hand end of the grid.

```
heatmap[25][69] = 0
```

To fill in our heatmap requires a series of loops. We use `count` to work out how far away from an exit we currently are. We look through the grid for any cells which are next to a cell with score `count`, and set those cells to `count+1`. We keep doing this until there are no `-1` scores left in the heat map, which we track using a variable called `heatmapIncomplete`.

```
count = 0
heatmapIncomplete = True
while heatmapIncomplete:
    for r in range(50):
        for c in range(70):
            if heatmap[r][c] == -1:
                if r-1>0 and heatmap[r-1][c] == count:
                    heatmap[r][c] = count+1
                elif c-1>0 and heatmap[r][c-1] == count:
                    heatmap[r][c] = count+1
                elif r+1<50 and heatmap[r+1][c] == count:
                    heatmap[r][c] = count+1
                elif c+1<70 and heatmap[r][c+1] == count:
                    heatmap[r][c] = count+1
    count += 1

    heatmapIncomplete = False
    for row in heatmap:
        if -1 in row:
            heatmapIncomplete = True
```

Now we can draw this in the same way as we did for the Game of Life and forest fires. We'll use white for an empty cell, blue for a person, and red for the exit.

```
window = Tk()
window.geometry('1400x1000')
window.title("Evacuation")
canvas = Canvas(window, width=1400, height=1000, bg='white')
canvas.pack(anchor=CENTER, expand=True)

def handler():
    global run
    run = False
    window.destroy()
window.protocol("WM_DELETE_WINDOW", handler)

run = True

while run:
    canvas.delete("all")

    for r in range(len(grid)):
        for c in range(len(grid[0])):
            if heatmap[r][c] == 0:
                colour = "red"
            elif grid[r][c] == 1:
                colour = "blue"
            else:
                colour = "white"
            canvas.create_rectangle((20*c,20*r), (20*(c+1), 20*(r+1)), fill=co
```

```
    window.update()
    sleep(0.2)

window.mainloop()
```

This should draw a room with some people and an exit. After drawing the grid and before `window.update()`, we need to tell our people to move. This is different from the other situations because we don't want to destroy people (like putting out fires) and we don't want to move two people to the same cell.

To do this, we set up `nextgrid` and initially fill it with blank spaces.

```
for r in range(len(grid)):
    thisrow = []
    for c in range(len(grid[0])):
        thisrow.append(0)
    nextgrid.append(thisrow)
```

Now we loop across the grid and, for each person not by an exit, we search for the neighbouring cell with the lowest score on the heatmap. If this cell is empty, we put our person there. If not, we leave them where they were in the current iteration. (Note that this automatically removes anyone adjacent to an exit.)

```
for r in range(len(grid)):
    for c in range(len(grid[0])):
        if grid[r][c] == 1 and heatmap[r][c] > 1: # person not by exit
            min_r = r
            min_c = c
            if r-1>0 and heatmap[r-1][c] < heatmap[r][c]:
                min_r = r-1
                min_c = c
            if c-1>0 and heatmap[r][c-1] < heatmap[r][c]:
                min_r = r
                min_c = c-1
            if r+1<50 and heatmap[r+1][c] < heatmap[r][c]:
                min_r = r+1
                min_c = c
            if c+1<70 and heatmap[r][c+1] < heatmap[r][c]:
                min_r = r
                min_c = c+1
            if nextgrid[min_r][min_c] == 0:
                nextgrid[min_r][min_c] = 1
            else:
                nextgrid[r][c] = 1
```

If you run this, you should see a grid with some people who move towards the exit.

Some things to try:

- Can you add a second exit?
- Do you notice any strange behaviour? Can you adjust the simulation to work around this?
- You can place objects in the room by giving a cell a very high score on the heatmap so no one ever walks there. For example, running this code just after you have placed your exit will set up a wall which the people must move around. Try colouring cells which are `1000` on the heatmap so you can see this wall.

```
for r in range(20,30):
    heatmap[r][67] = 1000
```

- What other improvements might make this simulation more realistic?
- What other situations could you simulation in this way?