Probability cellular automata

We can build models similarly to the Game of Life cellular automata, to simulate situations where activity can be modelled as movement or spread on a grid.

One example is forest fires. We can model a geographic area as a grid and designate each cell on the grid as either

1. trees (can be set on fire)
2. on fire
3. burnt out (so can't be set on fire)

To do this, we'll use a setup similar to Game of Life, except instead of rules considering 8 neighbours we will apply a simple rule: if a neighbouring cell immediately north, south, east or west is on fire, this cell catched fire. A cell burns for one iteration and then cannot be set on fire again.

First, let's set up the simulation by importing packages.

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

We need to track the grid, so we'll use a list variable. Let's use `0` for trees, `1` for fire and `9` for burnt out. We'll start with a grid of trees.

```
grid = []
for r in range(50):
    thisrow = []
    for c in range(70):
        thisrow.append(0)
    grid.append(thisrow)
```

In order to have something to simulate, let's start a single cell on fire (`1`) at a random point somewhere near the middle of the grid.

```
grid[randint(15,35)][randint(15,55)] = 1
```

Now let's draw our grid onto a window `canvas`, exactly as we did with the Game of Life except there are three colours - green for trees, red for fire, and grey for burnt out (not computer systems often use the American spelling 'gray').

```
window = Tk()
window.geometry('1400x1000')
window.title("Forest fire")
```

```
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 grid[r][c] == 9:
                colour = "gray"
            elif grid[r][c] == 1:
                colour = "red"
            else:
                colour = "green"
            canvas.create_rectangle((20*c,20*r), (20*(c+1), 20*(r+1)), fill=co

    window.update()
    sleep(0.1)

window.mainloop()
```

Now all we have to do is update the process of spreading the fire. If a cell is on fire (`1`) it should change to burnt out (`9`). Any tree cells (`0`) that have a neighbour on fire should switch to on fire.

So after drawing the grid and before updating the window, we must apply this rule. As with the Game of Life, we need a second list to store what we will change to.

```
nextgrid = []
```

A cell on our grid is indexed as `grid[r][c]` where `r` is the row number and `c` is the column number. We are interested in all the cells around this one and want to count how many are currently alive. The neighbouring cells we are interested in are:

- `grid[r-1][c]`
- `grid[r][c-1]`
- `grid[r][c+1]`
- `grid[r+1][c]`

```
for r in range(len(grid)):
    thisrow = []
    for c in range(len(grid[0])):
        thiscell = grid[r][c]
        if thiscell == 1:
            thiscell = 9 # burnt out
        elif thiscell == 0 and r-1>=0 and c-1>=0 and r+1<50 and c+1<70:
            if grid[r-1][c] == 1 or grid[r][c-1] == 1 or grid[r][c+1] == 1 or
```

```
                thiscell = 1 # caught fire
        thisrow.append(thiscell)
    nextgrid.append(thisrow)

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

What happens when you run this? Does it look like a fire spreading naturally through a forest?

To me, it appears too regular. This is because our simulation is entirely deterministic - fire always spreads to adjacent cells in all directions. We can improve the simulation by adding probability.

We checked whether any neighbour was on fire using `thissum > 0`. One way to introduce randomness is to simply toss a coin if a neighbouring cell is on fire.

To do this, instead of just setting `thiscell = 1 # caught fire`, first toss a coin:

```
if random()<0.5:
    thiscell = 1 # caught fire
```

If you run this, you will hopefully see something less regular that looks more believable as a fire.

Ideas for improvement that you might like to try:

- The outline of the grid never catches fire. Can you fix this?
- Can you arrange a multi-stage fire where, say, a cell stays on fire for three iterations?
- Can you make it more likely a cell will catch fire the more neighbours it has that are on fire?
- Can you add buildings to your simulation? Can you track whether these have burned? This might be the basis for a simulation that could run many times to evaluate the risk to a nearby settlement.
- What other improvements might make this simulation more realistic?
- What other situations could you simulation in this way?