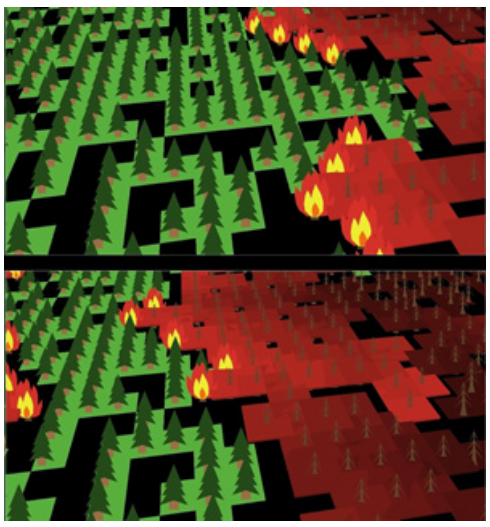
Day 19 Pre-class: Agent-based models and forest fires, Part 2



This pre-class is not an assignment. You don't have to hand anything in. Use this as a review of the code used in the forest fire model from Day 18. You will continue working on your forest fire model during the Day 19 In-class Assignment.

To Do

✅ Review the code below and compare each of the functions to the code you wrote last class. Make a note of any changes you notice. You will be asked about these differences at the beginning of the in-class assignment on Day 19.

```
import numpy as np
import numpy.random as rand
%matplotlib inline
import matplotlib.pyplot as plt

from IPython.display import display, clear_output
import time
```

Functions from Last Class

Last class, we built a function that plots the array, a function that initializes the board, and a function that advances the board based on a set of rules.

Plotting the Board

The `plot_grid` function takes in a given array, finds the positions on the board that are a tree or fire, and displays them as such.

```
def plotgrid(myarray):

    # First create two vectors based on the x and y sizes of the grid
    x_range = np.linspace(0, myarray.shape[1]-1, myarray.shape[1])
    y_range = np.linspace(0, myarray.shape[0]-1, myarray.shape[0])

    # Use the numpy meshgrid function to create two matrices
    # of the same size as myarray with x and y indexes
    x_indices, y_indices = np.meshgrid(x_range, y_range)

    # Make a list of all the x and y indexes that are either trees or fire.
    tree_x = x_indices[myarray == 1];
    tree_y = y_indices[myarray == 1];
    fire_x = x_indices[myarray == 2];
    fire_y = y_indices[myarray == 2];

    # Plot the trees and fire.
    plt.plot(tree_x, myarray.shape[0] - tree_y - 1, 'gs',markersize=10)
    plt.plot(fire_x, myarray.shape[0] - fire_y - 1, 'rs',markersize=10)

    # Set the x and y limits so we don't cut off the shapes
    plt.xlim([-1,myarray.shape[1]])
    plt.ylim([-1,myarray.shape[0]])

    # Removing tick marks
    plt.tick_params(axis='both', which='both',
                    bottom=False, top=False, left=False, right=False,
                    labelbottom=False, labelleft=False)
```

Initializing the Forest

The `set_board` function creates the initial state of the board based on the provided density, setting the leftmost edge as 'on fire.'

```
def set_board(board_size=50,f_trees_start=0.5):
    '''
    Creates the initial game board.

    Inputs:
        board_size: length of one edge of the board
        f_trees_start: probability that a given cell is a tree
                       (effectively the tree density)

    Outputs a 2D numpy array with values set to either 0, 1, or 2
        (empty, tree, or fire)
    '''

    # all cells initialized to 'empty' (0) by default
    game_board = np.zeros((board_size,board_size),dtype='int64')

    # loop over board and roll the dice; if the random number is less
    # than f_trees_start, make it a tree.
    for i in range(board_size):
        for j in range(board_size):
            if rand.random() <= f_trees_start:
                game_board[i,j] = 1

    # set the whole left edge of the board on fire. We're arsonists!
    game_board[:,0] = 2

    return game_board
```

Evolving the Board

The `advance_board` function:

- Takes in the *current* board,
- Defines a *new* board, and then
- Returns that new board where the positions of the fire have been updated.

Based on these rules:

> Each cell has a "neighborhood" that is composed of its four neighbors to the left, right, above, and below it. (Note: not the diagonal elements!) If a cell is along one of the edges of the array, only consider the neighbors that it has, and don't try to go out of the bounds of the array!
>
> The model takes steps forward in time, where every cell is modified based on the previous step. The model evolves as follows:
>
> - If the cell was empty last turn, it stays empty this turn.
> - If the cell is a tree and any of its neighbors were on fire last turn, it catches on fire.
> - If the cell was on fire last turn, the fire has consumed all of the trees and it is now empty.

```
def onBoard(i,j,image):
    if i <= image.shape[0]-1 and i >= 0 and j <= image.shape[1]-1 and j >= 0:
        return True
    else:
        return False

def getNeighborValues(i,j, board):
    neighborhood = [(i-1, j), (i, j-1), (i+1, j), (i, j+1)]

    neighbor_values = []
    for neighbor in neighborhood:
        if onBoard(neighbor[0], neighbor[1], board):
            neighbor_values.append(board[neighbor[0], neighbor[1]])

    return neighbor_values

def advance_board(game_board):
    '''
    Advances the game board using the given rules.
    Input: the initial game board.
    Output: the advanced game board
    '''

    # create a new array that's just like the original one, but initially
    # set to all zeros (i.e., totally empty)
    new_board = np.zeros_like(game_board)

    # loop over each cell in the board and decide what to do.
    for i in range(game_board.shape[0]):
        for j in range(game_board.shape[1]):

            # if the cell was empty last turn, it's still empty.
            # if it was on fire last turn, it's now empty.
            if game_board[i,j] == 0 or game_board[i,j] == 2:
                new_board[i,j] = 0

            # now, if it's a tree we have to decide what to do.
            if game_board[i,j] == 1:

                # initially make it a tree
                new_board[i,j] = 1

                # If one of the neighboring cells was on fire last turn,
                # this cell is now on fire!
                if 2 in getNeighborValues(i,j,game_board):
                    new_board[i,j] = 2

    # return the new board
    return new_board
```

By MSU CMSE