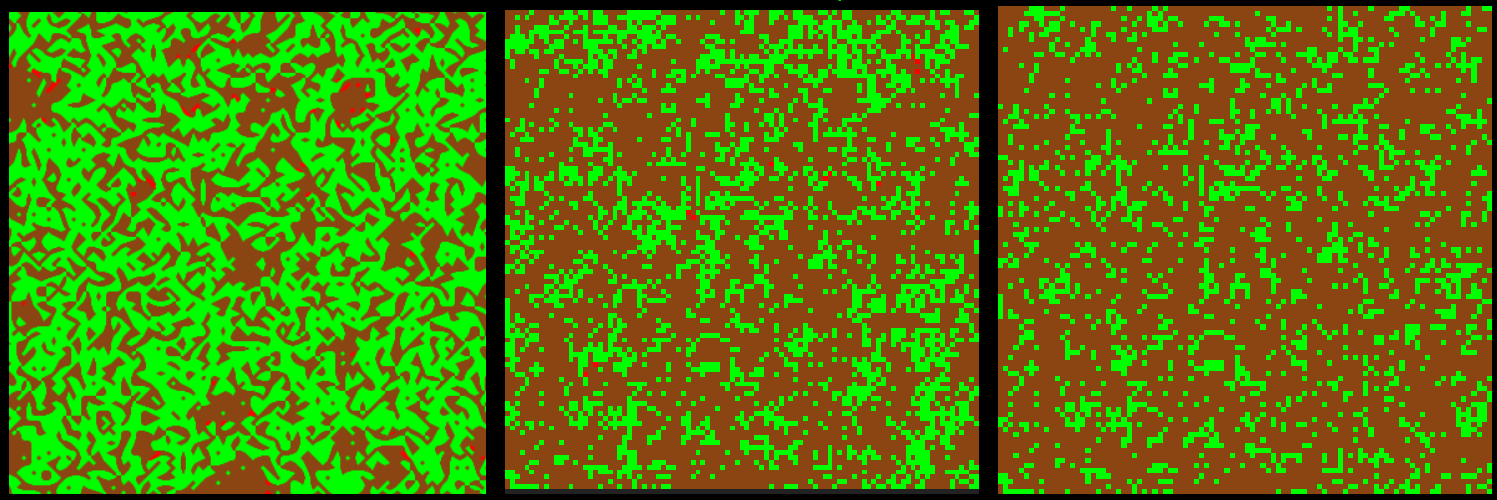
Wildfire Simulation

Marcas Anderson, Pat Hogan, Julianna Osborne

December 5, 2019



Description of Project

- Model a variety of wildfire scenarios.
- Alter code so that there are insulating boundary conditions (no trees around the border).
- Repeat the simulation with multiple probabilities, including random cells initially burning, chance of lightning, and chance of immunity.
- Explain the good and bad parts of the given cellular automata problem.

Assumptions

- The outer boundary of the forest does not have any trees.
- The probability of trees burning in the beginning will be relatively low.
- A probability of lightning will eventually cause trees to burn. A larger probability will cause the trees to catch on fire faster.
- A higher probability of immunity will cause less of the forest to be destroyed.
- A fire starting directly in the middle will be able to cover more land than a fire randomly placed in the grid. Multiple random fires can potentially burn more trees faster than a fire in the middle cell.
- Any non-immune trees will burn if burning trees are adjacent.

Insulating Boundary Conditions

The boundary of the forest must have insulating conditions, meaning that the border must have no trees. In order to translate this into Python, we had to create two for loops, one for the x-direction and one for y. A grid containing 0 represents a section with no trees, therefore the entire border is compiled of 0 grids.

```
for i in range(terrX):
    states[0,i,0] = 0
    states[0,terrY-1,i] = 0
for i in range(terrY):
    states[0,0,i] = 0
    states[0,terrX-1,i] = 0
```

Random Grid

When deciding if we want a random grid, a 0 or 1 is passed. If 1, the random grid code will be run.
A random x and y is generated between 1 and the length/width of the grid and is rounded. These values assign states[0,x,y]=2, the random cell that initially starts on fire.

```
# set the middle cell on fire!!!
if randomFire==0:
    states[0,terrain_size[0]//2,terrain_size[1]//2] = 2
elif randomFire==1:
    #RANDOM GRID
    x = round(random.uniform(1,terrain_size[0]-2))
    y = round(random.uniform(1,terrain_size[0]-2))
    states[0,x,y]=2
else:
#random chance of multiple trees catching fire
    for i in range(1,terrain_size[0]-2):
        for g in range(1,terrain_size[0]-2):
            if states[0,i,g]==1:
                if random.uniform(0,1)<randomFire:
                    states[0,i,g]=1
```

Probability of Lightning

The chance of lightning is occurring throughout the time-steps. There is a very small chance lightning will strike a given cell for a given time-step throughout.
If there is lightning (probLightning!=0), it checks if states[0,f,g] is a tree. If it's a tree, if the random number generated between (0,1) is under the probLightning value, the tree will catch fire.

```
for t in range(1,total_time):
    # Make a copy of the original states
    states[t] = states[t-1].copy()
    if probLightning!=0:
        for f in range(terrain_size[0]):
            for mii in range(terrain_size[1]):
                if states[t,f,mii]==1:
                    xrand=random.uniform(0,1)
                    if xrand<=probLightning:
                        states[t,f,mii]=2
```

Probability of Immunity

The chance of immunity means each tree has a % chance of becoming immune, preventing fire from passing through it. When the program starts, percentImmunity is a decimal value between 0 and 1. If the value is greater than 0, the for loop will run. A random x will be generated and, if x is less than or equal to percentImmunity, the tree will be invincible with value 3.

```
#If percentImmunity has a value
if percentImmunity!=0:
    for i in range(terrain_size[0]):
        for g in range(terrain_size[1]):
            if states[0,i,g]==1:
                xrand=random.uniform(0,1)
                if xrand<=0.2:
                    states[0,i,g]=3
```

Conclusion

- All requirements for this project were in the correct order so that there was no conflict with the code or logic.
- This was a basic template for cellular automata. Some limitations included having to manually code and check every condition.
- Improvements include making the function and the descriptive comments more descriptive.