CSc 110, Spring 2018
Programming Assignment #10: Fire (20 points)
Due Tuesday, April 17th, 7:00 PM

Thanks to Angela B. Shiflet from Wofford College and Marty Stepp from the University of Washington.

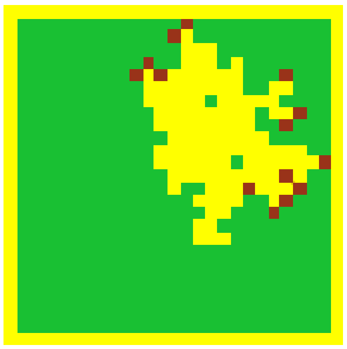
This assignment focuses on lists of lists. Turn in a file named `fire.py`. You will need `DrawingPanel.py` and `one.txt`. Place these files in the same directory as your program.

Background Information:

As you have doubtless seen, computers are commonly used to simulate real life scenarios. One such common scenario is how fire spreads. This is super common in computer gaming to make games more realistic and interesting. However, it is also has many applications in preventing and controlling real life fires. Fire simulations are used for training for fire fighters. Trainers can't go out and create a huge fire in order to train fire fighters but they also don't want to send them into a fire with no practical training. Simulations are a great way to solve this problem. Fire simulations are also used to predict how fires will spread so that people in danger can be evacuated.

In this assignment, you will create a simulation of fire spreading.

How the program works:



For this program, your world consists of a grid of squares, each representing a square meter of ground. Some of these squares will contain trees (or other burnable vegetation), some nothing and some fire. We will represent trees as green, nothing as yellow and fire as red.

You can assume that all the squares on the edge of the grid will always contain nothing. Think of them as a firebreak (a section of land that has been cleared to prevent the further spread of fire).

As your program runs, you will animate how the fire spreads from burning areas to areas with vegetation. Your simulation should continue until all fire has gone out.

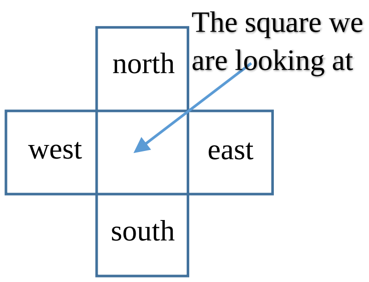
Program Behavior and Development Strategy:

Step 1: populate the simulation grid

When your program starts it should prompt the user for a file name. It should then read in the initial state of the land from this file. This file will contain a grid of 0, 1 and 2 characters separated by whitespace. You program should create a list of lists with the inner lists as long as the rows are wide and the outer list as long as the number of rows in the file.

We have provided one sample input file, `one.txt`. You will need to create other input files to test your code.

Step 2: spreading the fire



The next step is to determine how the fire spreads. If a square is empty (0), it will stay empty no matter what. If a square is on fire (2), on the next time step it will become empty (0). If a square has a tree in it (1) you will need to look at its neighbors to determine its chances of catching fire.

For this simulation we will be considering the neighbor right above (north), right below (south), to the right (east) and to the left (west). We will not be considering diagonal neighbors. If any of the north, south, east or west neighbors are on fire, generate a random number between 1 and 100 and if it is below 75 (the propagation percent) make that square on fire (2), otherwise leave it as a tree (1).

Note that as you are doing this you will be changing your list of lists. This means that if you were to alter your original list, you would be comparing each square to some squares that were already updated and some that weren't. To avoid this problem, create a new list of lists to store all of your changes in. When you finish a pass through the grid replace your original list of lists with this new list of lists.

Step 4: Visualization

In order to watch the simulation you will need to create a visualization on a drawing panel. Draw a grid on a drawing panel that is 10 times the length of each inner list and 10 times the height of the outer list. Make each square 10 by 10 pixels. Color the squares differently depending on whether they are fire, empty or tree. In this document we have used green, yellow and red but you may use whatever colors you would like.

Step 5: Simulating

Animate the simulation by updating the fire's progress and redrawing every 100 milliseconds. Your simulation should continue until all fire has gone out. Remember, you can make your program pause by calling `panel.sleep(`**number**`)` where **number** is the number of milliseconds you want it to pause.

Style Guidelines:

For this assignment you are required to have a constant for the propagation percentage. The propagation percentage is the percentage chance that the fire spreads (a combination of dryness, wind, temperature, etc). In your program it should initially be set to 75. For full credit it should be possible to change this constant's value and cause your program to change its behavior. You may use additional constants if they make your code clearer.

We will grade your function structure strictly on this assignment. Use at least **three nontrivial functions** besides `main`. These functions should use parameters and returns, including lists of lists, as appropriate. The functions should be well-structured and avoid redundancy. No one function should do too large a share of the overall task. You may not nest these functions inside each other or in main.

Your `main` function should be a concise summary of the overall program. It is okay for `main` to contain some code but `main` should not perform too large a share of the overall work itself, such as traversing over the list of lists. Also avoid "chaining," when many functions call each other without ever returning to `main`.

We will check strictly for redundancy on this assignment. If you have a very similar piece of code that is repeated several times in your program, eliminate the redundancy such as by creating a function, by using `for` loops over the elements of lists, and/or by factoring `if/else` code.

Since lists of lists are a key component of this assignment, part of your grade comes from using lists of lists properly. Carefully consider how they should be passed as parameters and/or returned from functions as you are decomposing your program.

You are limited to features in lectures 1 - 32. Follow past style guidelines such as names, variables, line lengths, and comments (at the beginning of your program, on each function, and on complex sections of code). You may not have any global variables (except constants) or code outside of functions.