Assignment 10

Final Challenge

Date Due: 5:00pm Friday Dec 5 **Total Marks: 19**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.5+
- You may not, in part or in whole, submit any code that was written by generative AI tools, such as ChatGPT
- **Assignments must be submitted to Canvas (the course website).**
- **Canvas will not let you submit work after the assignment hand-in closes.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (14 points):

It's time to put all of your Python skills to work. For the first time we are giving you a non-trivial problem, but only describing the problem that needs to be solved, and giving you very little guidance on **how** to solve it. You are allowed to use any of the Python language features and problem solving methods that we have discussed in class or used on assignments or in labs in order to solve the problem.

This assignment is intended only for students who have mastered all the basic concepts up to this point in the course. If you struggled with your Codelab exercises and haven't completed some of them, go back and do those instead! You will find it a better use of time to develop completely solid basics rather than spending time on this problem.

Problem Description

Pokemon are fantastic creatures that people can catch and train, and many Pokemon trainers aspire to catch as many as possible [1]. Imagine that you have caught so many Pokemon that you have decided to acquire your own farm to feed them. Your farm is an NxN plot of fruit trees, and there are potentially five different types of fruits that can grow on the farm: **firefruits, waterfruits, grassfruits, joltfruits** and the very rare **stellarfruit**.

Initially, your farm is mostly empty, with just a few fruit trees placed here and there. However, each **spring**, the fruit trees will spread to adjacent locations in the farm (using a process described further below). Each **fall**, you will harvest the different types of fruit from all of the trees on your farm.

Your task for this assignment is to write a program that will simulate the growth of your farm over time and report the total fruit yield by the time the farm is done growing.

Problem Details

Initial conditions

We will visualize the farm as an NxN grid of characters, where an underscore (_) represents an empty space for a fruit tree, and a letter represents a fruit tree of the corresponding type (**F** for **firefruit** tree, **W** for **waterfruit** tree, etc...). The initial conditions for the farm will come from a file so you will have to do some file input to set up your farm. Be careful: the file format for the input file will NOT look like the visualization of the farm (just described), but will instead be much more compact. It will look like this:

```
4
0,0 1,0
2,0 3,0
0,3 1,3
2,3 3,3
```

The first line of the file is a single integer indicating the dimensions of the farm (the farm is always square). This is followed by exactly 4 more lines, one line for each type of basic fruit tree **in this order**: firefruit, waterfruit, grassfruit, joltfruit. On each line will be one or more sets of (row,column) coordinate pairs representing the intial locations of the fruit trees. The coordinate pairs on the same line are separated from each other by spaces and the row and column for each pair are separated by a comma. Below is a visualization of the farm that is represented by the input file above. Make sure you understand how the input file matches up to the grid picture before going any further!

[1]Some might even say they gotta catch them all

	0	1	2	3
0	F	_	_	G
1	F	_	_	G
2	W	_	_	J
3	W	_	_	J

Visualization of the initial conditions of a 4x4 farm

Spring Spreading

Once you have loaded the initial farm, you will simulate the farm's behaviour over a period of multiple years. During the spring of each year, every fruit tree on the farm has the potential to spread to adjacent **empty** spaces. Spaces that already contain a fruit tree will never change, only the empty ones. The rules for how the trees can spread are as follows.

If an empty space is:

- adjacent to only one type of tree: then that tree type will spread to the space
- adjacent to ALL FOUR types of basic trees: then a stellarfruit tree will grow in that space
- adjacent to two or three types of basic trees: then the **dominant** tree type will spread. Firefruit dominates grass; grassfruit dominates water; waterfruit dominates fire. Joltfruit dominates ALL 3 other basic fruits (because Joltfruit is Pikachu's favorite and Pikachu is the best!!). Stellarfruit dominates everything, even Joltfruit.
- adjacent to exactly three types and none are dominant: then nothing spreads.

For the spreading above, we will use only **orthogonal** adjacency (i.e. diagonals don't count). The following is an example of the trees spreading on a small farm over 2 years.

	0	1	2
0	F	_	G
1	_	_	_
2	W	_	J

Initial Farm

	0	1	2
0	F	F	G
1	W	_	J
2	W	J	J

After Year 1

	0	1	2
0	F	F	G
1	W	J	J
2	W	J	J

After Year 2

Your simulation should end once the farm has reached a state that will **no longer change** so make sure you can detect this. Be careful! The farm will NOT necessarily be full of trees, as some spaces may be permanently empty, as in this example:

	0	1	2
0	-	-	-
1	-	F	-
2	W	-	G

Initial Farm

	0	1	2
0	-	F	-
1	W	F	F
2	W	-	G

After Year 1

	0	1	2
0	W	F	F
1	W	F	F
2	W	-	G

After Year 2; no change after this point

Fall harvest

In fall, after the trees in your farm spread during the spring, you will harvest the fruit from the farm. Each tree produces 1 fruit of the matching type. Your task is to keep track of how much fruit is produced by the farm in total during all the years while the farm is still growing.

Output

Once the farm has reached the point where no more trees can spread, your program should print to the console a summary report. This report should consist of:

- The number of years it took for the farm to fill up with trees to its current point.
- The total amount of each type of fruit produced since the initial state of the farm. This total should include exactly one harvest from the final, stabilized state of the farm, as well as all the harvests prior to that point.
- The amount of each type of fruit produced during ONLY the FINAL year (since the farm has now stopped growing, this should also be the same as the amount that will be consistently produced by the farm from now on if the simulation were to continue for subsequent years)

For the `palletfarm` input file that we are providing as a sample input, your output will look something like this:

```
Fruit yield from final year:
***************************
Grassfruit :  4
Joltfruit :  4
Firefruit :  4
Waterfruit :  4
Stellarfruit :  0

Total farm yield after 2 years:
*********************************
Grassfruit :  7
Joltfruit :  7
Firefruit :  7
Waterfruit :  7
Stellarfruit :  0
```

We are providing you with **five input files** for five different initial farms. In a **separate document for each farm**, copy/paste your console output for that farm. Note that the output above is correct for `palletfarm`, but we are not telling you in advance what the correct output will be for the other farms!

Testing

Because you do not have the answers in advance for most of the input farm files, you will have to employ your testing skills to ensure the correctness of your program. However you do not need to hand in any testing beyond your output for the provided files

Hints

- Divide up the work you need to do to solve this problem into subtasks and determine the best order in which to solve them.
- Make sure the code for each sub-task is working before you proceed to the next one (using testing).
- The Spring-time propagation is the trickiest part of this problem. Keep in mind that only the trees that are already present at the START of spring will spread to adjacent tiles. Newly grown trees will NOT spread during the same spring-time in which they first appeared. Make sure you take this account when designing your algorithms.
- Try not to overthink the difficulty of determining whether or not the farm has changed after a spring-time growth. It's not actually very difficult to determine whether something has changed when YOU are the one changing it! Remember, any time you need to remember ANYTHING in a computer program, you can add extra variables if you need them.

What to Hand In

- Submit your solution to the main problem in a file called a10q1.py
- Submit FIVE different output files, one for each input farm, named in the format `farmname_`output.txt.
- You do not have to submit the provided input files themselves.

Evaluation

A detailed grading rubric is found at the end of this document. Evaluation criteria include:

- Appropriate organization of the program using functions and/or modules
- Use of appropriate data types
- Correctness of program

Detailed Grading Rubric

3 Marks: Organization of the program using functions and/or modules	
Grade	Criteria
3	Excellent: functions and/or modules break the program into cohesive units.
2	Good: some good functions are used, others are less good or more breakdown was needed
1	Weak: Some functions are used, but many have poor design or cohesion
0	None: Functions are not used or everything is in one function.

3 Marks: Use of Mature Techniques	
Grade	Criteria
3	Excellent: code shows maturity in data structure choices and use of programming elements.
2	Good: Several good choices are made, but there are a few instances of immaturity (e.g. using 8 if-statements when a simple loop would do).
1	Weak: There are a few instances of maturity, but many poor choices are made.
0	None: Only the very simplest data types and techniques are used with no evidence of maturity.

8 Marks: Program correctness.	
Grade	Criteria
8	Excellent: Final outputs are provided and correct for all farm files. The code itself appears perfectly correct.
6	Good: A minor error causes small instances of incorrect output. Any errors in code are non-obvious.
4	Moderate: Most or all of the outputs are not perfectly correct, but appear to be on the right track.
2	Some output is produced, but it is not really very close OR the program crashes on any but the simplest files.
0	Nothing seems correct.

Question 2 (3 points):

All programs that you submit for this assignment will be assessed with regard to their **style and readability**. Make sure that all of your code:

- includes your name, nsid, and student number as comments
- uses informative variable names
- makes effective use of formatting and white space
- includes a `docstring` for all **function definitions** in the program
- for longer programs, uses helpful and concise single-line comments where appropriate

What to Hand In

There is nothing to hand in for this question. It's simply describing how the rest of your work will be evaluated with regard to style and readability. It is possible to earn these marks even if you don't hand in every other question in this assignment, but you must hand in enough work such that your style and readability can be adequately assessed.

Evaluation

- 3 marks for consistently following the principles listed above

Question 3 (2 points):

In your submission for this assignment, you have an opportunity to earn something we'll call "the wow factor". To earn the wow factor, you must do something creative that goes beyond the expectations laid out in the assignment. This could be an extra feature for one of the previous questions, or it could be an extra little program that you submit that builds on the skills used in the assignment.

You should not aim to do a lot of work for this, but should instead demonstrate in a simple and effective way the depth of your understanding. In short, take ownership of your work and "impress us".

Telling you exactly what to do to earn "the wow factor" would defeat the entire purpose, but here are some very general ideas.

- improve the aesthetic appeal of your program(s)
- improve the user-friendliness of your program(s)
- remove a simplifying assumption and handle the resulting added complexity
- compare two different ways of doing the same task, and submitting data/analysis on which was better
- just doing something generally cool

One quick warning: using fancy extra code libraries in order to AVOID a key learning objective from the assignment is not impressive, it's actually anti-impressive. Focus instead on solving problems using simple clean foundational programming skills. Using extra libraries is fine if they let you do something BEYOND what the assignment asked, but not if they're replacing a core assigned task.

What to Hand In

A plain text file called `wow.txt` describing what you did that you think merits the wow factor. One or two sentences is enough. If you create any additional code files, hand those in too.

Evaluation

- 2 marks for something impressive