Introduction to Programming in Python

Assignment 5 (Atomic Nature of Matter) Discussion

Introduction

Goal: track the motion of particles undergoing Brownian motion, fit this data to Einstein's model, and estimate Avogadro's constant

Problem 1 (Particle Representation)

Define a data type called `Blob` in `blob.py` to represent a particle (aka blob). The data type must support the following API:

`Blob()`	constructs an empty blob `b`
`b.add(x, y)`	adds a pixel `(x, y)` to *b*
`b.mass()`	returns the mass of `b`, ie, the number of pixels in it
`b.distanceTo(c)`	returns the Euclidean distance between the center of mass of `b` and the center of mass of blob `c`
`str(b)`	returns a string representation of `b`

```
× ~/workspace/atomic_nature_of_matter
```

```
$ python3 blob.py
1 0 0 1 -1 0 0 -1
<ctrl-d>
a           = 1 (0.0000, 0.0000)
b           = 4 (0.0000, 0.0000)
dist(a, b) = 0.0
```

Problem 1 (Particle Representation)

Instance variables:

- x-coordinate of center of mass, `_x` (float)
- y-coordinate of center of mass, `_y` (float)
- Number of pixels, `_pixels` (int)

`Blob()`

- Initialize the instance variables appropriately

`b.add(x, y)`

- Use the idea of *running average*[1] to update the center of mass of blob `b`
- Increment the number of pixels in blob `b` by 1

`b.mass()`

- Return the number of pixels in the blob `b`

`b.distanceTo(c)`

- Return the Euclidean distance between the center of mass of blob `b` and the center of mass of blob `c`

---

[1] If $\bar{x}_{n-1}$ is the average value of $n-1$ points $x_1, x_2, \ldots, x_{n-1}$, then the average value $\bar{x}_n$ of n points $x_1, x_2, \ldots, x_{n-1}, x_n$ is $\frac{\bar{x}_{n-1} \cdot (n-1) + x_n}{n}$.

Problem 2 (Particle Identification)

Define a data type called `BlobFinder` in `blob_finder.py` that supports the following API. Use depth-first search to efficiently identify the blobs.

`BlobFinder(pic, tau)`	constructs a blob finder `bf` to find blobs in the picture `pic` using a luminance threshold `tau`
`bf.getBeads(pixels)`	returns a list of all blobs with mass $\geq$ `pixels`, ie, a list of beads

Problem 2 (Particle Identification)

```
× ~/workspace/atomic_nature_of_matter
$ python3 blob_finder.py 25 180.0 data/run_1/frame00001.jpg
13 Beads:
29 (214.7241, 82.8276)
36 (223.6111, 116.6667)
42 (260.2381, 234.8571)
35 (266.0286, 315.7143)
31 (286.5806, 355.4516)
37 (299.0541, 399.1351)
35 (310.5143, 214.6000)
...
15 Blobs:
29 (214.7241, 82.8276)
36 (223.6111, 116.6667)
1 (254.0000, 223.0000)
42 (260.2381, 234.8571)
35 (266.0286, 315.7143)
31 (286.5806, 355.4516)
37 (299.0541, 399.1351)
35 (310.5143, 214.6000)
...
```

Problem 2 (Particle Identification)

Instance variable:

- Blobs identified by this blob finder, `_blobs` (list of `Blob` objects).

`BlobFinder()`

- Initialize `blobs` to an empty list.
- Create a 2D list of booleans called `marked`, having the same dimensions as `pic`.
- Enumerate the pixels of `pic`, and for each pixel `(i, j)`:
  - create a `Blob` object called `blob`;
  - call `_findBlob()` with the appropriate arguments; and
  - add `blob` to `blobs` if it has a non-zero mass.

`bf._findBlob()`

- Base case: return if pixel `(i, j)` is out of bounds, or if it is marked, or if its luminance (use the `luminance()` method from `Color` for this) is less than `tau`.
- Mark the pixel `(i, j)`.
- Add the pixel `(i, j)` to the blob `blob`.
- Recursively call `_findBlob()` on the N, E, W, and S pixels.

`bf.getBeads(pixels)`

- Return a list of blobs from `blobs` that have a mass $\geq$ `pixels`.

Problem 3 (Particle Tracking)

Implement a program called `bead_tracker.py` that accepts *p* (int), *tau* (float), *delta* (float), and a sequence of JPEG filenames as command-line arguments; identifies the beads in each JPEG image using `BlobFinder`; and writes to standard output (one per line, formatted with 4 decimal places to the right of decimal point) the radial distance that each bead moves from one frame to the next (assuming it is no more than *delta*)

```
× ~/workspace/atomic_nature_of_matter
```

```
$ python3 bead_tracker.py 25 180.0 25.0 data/run_1/frame00000.jpg \
data/run_1/frame00001.jpg
7.1833
4.7932
2.1693
5.5287
5.4292
4.3962
```

Problem 3 (Particle Tracking)

Accept command-line arguments `pixels` (int), `tau` (float), and `delta` (float)

Construct a `BlobFinder` object for the frame `sys.argv[4]` and from it get a list of beads `prevBeads` that have at least `pixels` pixels

For each frame starting at `sys.argv[5]`:

- Construct a `BlobFinder` object and from it get a list of beads `currBeads` that have at least `pixels` pixels
- For each bead `currBead` in `currBeads`, find a bead `prevBead` from `prevBeads` that is no further than `delta` and is closest to `currBead`, and if such a bead is found, write its distance (using format string `'%.4f\n'`) to `currBead`
- Write a newline character
- Set `prevBeads` to `currBeads`

Problem 4 (Estimating Avogadro's Constant)

Implement a program called `avogadro.py` that accepts the displacements (output of `bead_tracker.py`) from standard input; computes an estimate of Avogadro's constant using the formulae described above; and writes the value to standard output

```
× ~/workspace/atomic_nature_of_matter
$ python3 bead_tracker.py 25 180.0 25.0 data/run_1/* | python3 avogadro.py
6.633037e+23
```

Problem 4 (Estimating Avogadro's Constant)

Initialize `ETA`, `RHO`, `T`, and `R` to appropriate values

Calculate `var` as the sum of the squares of the `n` displacements (each converted from pixels to meters) read from standard input

Divide `var` by `2 * n`

Estimate Boltzmann's constant as `6 * math.pi * var * ETA * RHO / T`

Estimate Avogadro's constant as `R / k`

Write to standard output the Avogadro constant in scientific notation (using the format string `"%e"`)