CS0011 Introduction to Computing for Scientists
Project 3

Overview

In this project, we will use object-oriented programming to implement a model of microbial growth. When enough nutrients are available, colonies form regular shapes. However, if nutrients are scarce, colonies form irregular shapes due to diffusion-limited growth. We will build a simple model of diffusion-limited growth due to Matsuura [Mat00] and further explored by Tronnolone et al. [TTS+18].

In our model, we represent the environment (say, a Petri dish) as a grid. Both microbes and nutrients are represented as points on a grid. Diffusion of nutrients is modeled using a random walk. At each time step, a nutrient particle takes a randomly-chosen step on the grid. In other words, at each time step, a nutrient particle chooses uniformly at random among the directions up, down, left, and right, and moves a single space in that direction. There can be multiple nutrients in each cell of the grid.

A microbe can take up a nutrient particle that is at the same square of the grid as it is. Once a microbe has a nutrient particle, it may reproduce. A microbe reproduces by choosing a neighboring grid square (up, down, left, or right) that doesn't contain another microbe uniformly at random and placing a new microbe in that grid square. Reproduction uses up the first microbe's nutrients. There can only be 1 microbe in a single cell of the grid. Microbes do not move throughout the grid.

Part 1: Representing nutrients

To start, create a basic `Nutrient` class to represents the nutrient particles that will be moving throughout the grid. Your `Nutrient` class should simply have a single attribute that is `hasMoved`. This attribute should have a Boolean value. Since the simulation will need to run as a series of steps, we need to ensure that each nutrient moves at most once per step. This attribute will help us ensure that. In addition to this attribute, your `Nutrient` class should have the following methods:

- `__init__(self)`: The initializer should set `hasMoved` to `False`.
- `getMoved(self)`: Should return the current value of `hasMoved`.
- `setMoved(self)`: Should set `hasMoved` to `True`.
- `clearMoved(self)`: Should set `hasMoved` to `False`.

Part 2: Representing microbes

Next, create another basic class to track our microbes (aptly named `Microbe`). Since microbes can hold nutrients, we will need an attribute to track whether or not the microbe is holding a nutrient. Your `Microbe` class should have the following methods:

- `__init__(self)`: The initializer should an attribute named `heldNutrient` to `None`.
- `hasNutrient(self)`: Should return `True` if the microbe has a nutrient, `False` otherwise.
- `takeNutrient(self, nutrient)`: Should set assign the argument `nutrient` to `heldNutrient`.
- `consumeNutrient(self)`: Should set `heldNutrient` to `None`. This should be called whenever the nutrient reproduces.

Part 3: Representing the grid

Now that you have nutrients and microbes, you should construct a model of the grid to house them and run the simulation. We will use two different classes to model the grid. The first, `PetriDish`, will represent the entire grid, and will have methods to run the simulation. The second, `PetriCell`, will represent a single cell (space) within the grid. PetriCell will allow us to easily keep track of whether there is a microbe at a given location (cell) in the grid, and if there are one or more nutrients there. You should define the `PetriCell` class first, with the following methods:

- `__init__(self)`: The initializer should set two attributes. `microbe` should be initialized to `None` and `nutrients` should be initialized to an empty list.
- `getMicrobe(self)`: Should return the value of the `microbe` attribute.
- `hasMicrobe(self)`: Should return `True` if the `microbe` attribute is not `None` and `False` if it is `None`.
- `createMicrobe(self)`: Should create a new `Microbe` object and assign it to the `microbe` attribute
- `getNutrient(self)`: Should remove a nutrient the from the `nutrients` attribute and return it. Should return `None` if `nutrients` is empty.
- `getUnmoved(self)`: Should remove all nutrients from `nutrients` that have `hasMoved == False` and return them in a list. All nutrients with `hasMoved == True` should remain in `nutrients`.
- `clearAllMoved(self)`: Should call `clearMoved()` on each `Nutrient` object in `nutrients`.
- `hasNutrients(self)`: Should return `True` if the `nutrients` attribute has any nutrients, and `False` if it is empty.
- `placeNutrient(self, nutrient)`: Should add the argument `nutrient` to the attribute `nutrients`.

- `__str__(self)`: Should give a string representation of the current cell. A cell with a microbe and no nutrients should return `"M0N"`. A cell with no microbe and 3 nutrients should return `"_3N"`. A cell with a microbe and 1 nutrient should return `"M1N"`. Note that a nutrient held by the microbe should not count towards the total. The format is "M" if there is a microbe, "_" if there is not, followed by the length of the `nutrients` attribute followed by an "N".

With the `PetriCell` class finished, you can use it to build the `PetriDish` class with the following methods:

- `__init__(self, x, y, concentration, microbes)`: `x` and `y` should be ints describing the size of the grid. You will represent the grid of a list of lists of `PetriCell` objects. You should store this list of lists in the attribute `grid`. `grid[0][0]` should represent the upper left corner of the grid, while `grid[x - 1][y - 1]` should represent the bottom right. This means that you should initialize `grid` as a list of `x` nested lists, each containing `y` `PetriCell` objects.

  Once you have created the grid, you should add nutrients and microbes. The `concentration` argument should be a float (between `0.0` and `1.0`) specifying the percentage of the grid spaces that should contain a nutrient. Multiplying `concentration`, `x`, and `y` will tell you the number of nutrients to create (truncate the result to an int). You should randomly select valid indices (e.g., from `0 .. x - 1` and `0 .. y - 1`) to place each of these nutrients. Use the `random.randint` function to select the index values. Note that this may place multiple nutrient objects initially in the same cell. This is fine.

  Finally, you should place the microbes. The `microbes` argument should be a list of tuples, where each tuple gives the coordinates where a microbe should be placed (e.g., `[(75, 100), (125, 100)]` would specify microbes to be placed at `grid[75][100]` and `grid[125][100]`).

- `moveNutrients(self)`: This method should move every nutrient in the grid once. For each nutrient, it should randomly choose to move the nutrient up, down, left, or right. For example, a nutrient in `grid[5][5]` could move to any of the following 8 options:

  - `grid[4][4]`
  - `grid[4][5]`
  - `grid[4][6]`
  - `grid[5][4]`
  - `grid[5][6]`
  - `grid[6][4]`
  - `grid[6][5]`
  - `grid[6][6]`

  Note that you must be careful not to move a nutrient "outside" of the grid. For example, a nutrient at `grid[0][0]` only has 3 possible options to move:

  - `grid[0][1]`
  - `grid[1][0]`
  - `grid[1][1]`

After a nutrient is moved, set that nutrient's `hasMoved` attribute to `True` to ensure it is not moved twice during this call to `moveNutrients`.

In order to find all of the nutrients in the grid, you will need to iterate through each `PetriCell` object one at a time, and, for each `PetriCell` object, iterate through its unmoved nutrients.

Once all nutrients have been moved, you should set all of their `hasMoved` attributes back to `False` so that they are ready for the next call to `moveNutrients`.

- `checkMicrobes(self)`: This method should check each microbe in the grid. For each microbe, it should first see if the microbe is not holding a nutrient, but there is a nutrient in that microbe's `PetriCell`. If so, the method should remove that nutrient from the `PetriCell` and pass it to the microbe's `takeNutrient` method. If, after that check, the microbe is holding a nutrient, it should attempt to reproduce. If the cell above, below, to the left, or to the right of the microbe does not have a microbe, the microbe should consume its held nutrient and the method should create a new microbe in that neighboring cell. The choice of creating a new microbe above, below, to the left, or to the right should be randomly for all valid cells (i.e., cells without microbes). If all neighboring cells have microbes, the current microbe cannot reproduce.

  Note that you will not want to actually add the new microbe into the neighboring cell until all microbes have been checked to ensure that we do not create a microbe, and then have it reproduce in the same step. Use the following approach.

  1. Create a list of tuples representing all the microbes to created (as a list of tuples representing coordinates for new microbes).
  2. Every time a microbe would reproduce, add a new tuple to this list.
  3. Once all cells have been checked, create microbes at each coordinate specified in the list of tuples.

- `step(self, iterations)`: Should run through `iterations` steps of the simulation. Each step of the simulation should call `moveNutrients()` and then `checkMicrobes()`.

- `__str__(self)`: This method will be essential for debugging. Should call `__str__()` for each `PetriCell` and return the result as a grid. Each row of the grid should be its own line. All of the cells in a row should be separated by a single space.

Part 4: Running the simulation

With all of this completed, create a `main()` function that creates a grid and run the simulation. You should test that everything is working first. Create a simple 5x5 grid with .96 nutrient concentration and a single starting microbe at (2, 2). Print the state of the grid. Run a single step. Print the grid again. Repeat for 10 steps to make sure everything is working.

Next, attempt a 10x10 grid with .25 nutrient concentration and a single starting microbe at (5, 5). Again, run for 10 steps printing the grid every time.

Once you have completed all parts of the lab, be sure to show your work to the lab instructor.

Going further

Note that more interesting simulations will be larger in scale. For example, a 200x200 grid with .3 nutrient concentration and two starting microbes at (75, 100) and (125, 100). Note that a simulation at this scale will take a few minutes to run.

References

[Mat00] Shu Matsuura. Random growth of fungal colony model on diffusive and non-diffusive media. *Forma*, 15:309–319, 2000. (document)

[TTS+18] Hayden Tronnolone, Alexander Tam, Zoltán Szenczi, J.E.F. Green, Sanjeeva Balasuriya, Ee Lin Tek, Jennifer M. Gardner, Joanna F. Sundstrom, Vladimir Jiranek, Stephen G. Oliver, and Benjamin J. Binder. Diffusion-limited growth of microbial colonies. *Scientific reports*, 8(1):5992, 2018. (document)