An agent-based model of with environment-agent interaction using Python

The Keller-Segel Model of Slime Mold Aggregation

Patrick Vincent N. Lubenia

10 May 2020

Contents

Agent-Based Models

Agent-based modeling is a framework for creating and simulating models of complex systems. It is a mindset wherein a system is described from the point of view of the units constituting it [1]. Agent-based models are computational simulation models that involve numerous discrete agents. They show a system's emergent collective behavior resulting from the interactions of the agents. In contrast to equation-based models, each agent's behaviors in an agent-based model are described in an algorithmic fashion by rules rather than equations. Agents in the model do not typically perform actions together at constant time-steps [2]. Their decisions follow discrete-event cues or a series of interactions.

Depending on one's objectives, agents: are discrete entities, may have internal states, may be spatially localized, may perceive and interact with the environment, may behave based on predefined rules, may be able to learn and adapt, and may interact with other agents. Generally, agent-based models: often lack central supervisors/controllers and may produce nontrivial "collective behavior" as a whole.

The following scientific method-based approach must be kept in mind when designing an agent-based model:

1. Specific problem to be solved by the model
2. Availability of data
3. Method of model validation.

In order to be scientifically meaningful, an agent-based model must be:

1. Built using empirically-derived assumptions, then simulate to produce emergent behavior: for predictions; or
2. Built using hypothetical assumptions, then simulate to reproduce observed behavior: for explanations.

Once a code has been programmed, its basic implementation structure has 3 parts:

1. Initialization
2. Updating
3. Visualization.

Agents are initially placed in the model's environment. The system is then updated according to rules that govern the behavior of the environment and/or agents. Finally, states are visualized in order to appreciate the changes in the system.

The agent-based modeling framework is open-ended and flexible. It may be tempting to be detailed to make a model more realistic. But it must be remembered that increased complexity leads to increased difficulty in analysis. Moreover, the open-endedness of the framework makes it code-intense as lots of details of the simulation must be manually taken care of. Thus, codes must be kept simple and organized.

Empty `class`

Sometimes, a `class` is created when objects are needed to be flexible enough to be given desired attributes. In

```
1 class perfomer:
2   pass
```

the `pass` on line 2 allows one to create an empty class. This class can now be used to create a performer:

```
1 performer_ = performer()
```

It is simple to add attributes to the object called performer, say its location in terms of coordinates, name, and age:

```
1  # Assign attribute x to performer_
2  performer_.x = 3
3
4  # Assign attribute y to performer_
5  performer_.y = 4
6
7  # Assign attribute name to performer_
8  performer_.name = 'Luca'
9
10 # Assign attribute age to performer_
11 performer_.age = 16
```

This use of `class` is utilized in the model below.

Discretization

The Keller-Segel model of slime mold aggregation is modeled using partial differential equations. In order to implement the equations in Python, the continuous equations must be discretized. This allows time-step updating of solutions to the equation.

Consider a function of 1 variable: $y = f(x)$. Recall that for a small change in x, say Δx, the derivative of y with respect to x is approximately the slope:

$$\begin{aligned} y'(x) = \frac{dy}{dx} &\approx \frac{\Delta y}{\Delta x} \\ &= \frac{f(x+\Delta x) - f(x)}{\Delta x}. \end{aligned}$$

So,

$$f(x + \Delta x) \approx f(x) + y'(x)\Delta x.$$

Let $y_n = f(x_n)$. Thus, the next time-step is $y_{n+1} = f(x_n + \Delta x)$. Hence,

$$y_{n+1} = y_n + y'(x_n)\Delta x. \tag{1}$$

Note: Equality is used in Equation (1) as Δx can be made as small as possible to get an "approximately exact" solution.

This result can easily be extended to a function of 2 variables. Consider now $y = f(x, t)$. For a small change in t, say Δt:

$$\begin{aligned} y'(x,t) = \frac{\partial y}{\partial t} &\approx \frac{\Delta y}{\Delta t} \\ &= \frac{f(x,t+\Delta t) - f(x,t)}{\Delta t}. \end{aligned}$$

So,

$$f(x, t + \Delta t) \approx f(x, t) + y'(x, t)\Delta t.$$

Let $y_n = f(x_n, t_n)$. Thus, $y_{n+1} = f(x_n, t_n + \Delta t)$. Hence,

$$y_{n+1} = y_n + y'(x_n, t_n)\Delta t.$$

Now, consider the Taylor series expansion of a function f of a single variable x around $x = x_0$:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots.$$

Extending Equation (1) to include the second derivative:

$$y_{n+1} = y_n + y'(x_n)\Delta x + \frac{y''(x_n)}{2}(\Delta x)^2.$$

It is easy to show that the previous time-step is

$$y_{n-1} = y_n - y'(x_n)\Delta x + \frac{y''(x_n)}{2}(\Delta x)^2.$$

And so,

$$y_{n+1} + y_{n-1} = 2y_n + y''(x_n)(\Delta x)^2.$$

This gives

$$y''(x_n) = \frac{y_{n+1} - 2y_n + y_{n-1}}{(\Delta x)^2}.$$

This result is exactly the same for the 2-variable case if differentiating twice with respect to the same variable. So, to be clear, explicitly state the differentiation. Consider $y = f(x, t)$.

$$\begin{aligned} & y''(x_n) = \frac{y_{n+1} - 2y_n + y_{n-1}}{(\Delta x)^2} \\ \Rightarrow \quad & \frac{\partial^2}{\partial x^2}[f(x_n, t_n)] = \frac{f(x_n + \Delta x, t_n) - 2f(x_n, t_n) + f(x_n - \Delta x, t_n)}{(\Delta x)^2}. \end{aligned} \tag{2}$$

As an application of Equation (2), consider the function $z = f(x, y)$. The Laplacian of f is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

To discretize the Laplacian of f, use Equation (2):

$$\nabla^2 f = \frac{f(x_n + \Delta x, y_n) - 2f(x_n, y_n) + f(x_n - \Delta x, y_n)}{(\Delta x)^2} + \frac{f(x_n, y_n + \Delta y) - 2f(x_n, y_n) + f(x_n, y_n - \Delta y)}{(\Delta y)^2}.$$

Letting $\Delta y = \Delta x$:

$$\nabla^2 f = \frac{f(x_n + \Delta x, y_n) + f(x_n - \Delta x, y_n) + f(x_n, y_n + \Delta y) + f(x_n, y_n - \Delta y) - 4f(x_n, y_n)}{(\Delta x)^2}. \tag{3}$$

The Keller-Segel Model of Slime Mold Aggregation

Overview

Evelyn Keller and Lee Segel studied the development of cellular slime mold [4]. Upon germination, cells disperse. When a source of food (bacteria) is present, the cells move toward it. They described this aggregation of cells "as a breakdown of stability caused by intrinsic changes in the basic parameters which characterize the system". Simply put, the cells react to changes in its environment. This aggregation is called *chemotaxis*. Chemotaxis is a biological process involved in the development of multicellularorganisms, immune response, and cancer metastasis [3]. It is the mechanism where cells follow chemical cues in their native environments, moving towards a desired goal, e.g., source of nutrients.

Keller and Segel studied the cellular slime mold species *Dictyostelium discoideum*. They discovered that their aggregation was caused by a substance called cyclic adenosine monophosphate (cAMP), a nucleotide that mediates numerous biological responses [6].

The model presented is simulated to visualize emergent behavior from empirically-derived assumptions. The agents are slime mold cells interacting with its environment defined by cAMP concentrations everywhere.

This model follows an agent-based modeling framework with the following tasks that must be undertaken [7] [8]:

1. Design the data structure to store the:
   (a) Attributes of the agents
   (b) States of the environment
2. Describe the rules for how:
   (a) The environment behaves on its own
   (b) Agents interact with the environment
   (c) Agents behave on their own
   (d) Agents interact with each other.

Not all these tasks are needed in every agent-based model.

The Code

Needed Libraries

To start the code, the following libraries are imported:

```
# For assigning a random location to agents
import random as rd

# For initializing the environment
import numpy as np

# For visualizing the environment using plot
import matplotlib.pyplot as plt

# For checking if a filename already exists, to avoid overwriting files
import os
```

Modeling Task 1a and 1b: Attributes of the agents and state of the environment

The first task is to "design the data structure to store the attributes of the agents". The agents in this model are slime mold cells. They are not differentiated by any characteristic except for their location. Hence, their only attributes are the x- and y-coordinates describing their spatial location.

The second task is to "design the data structure to store the states of the environment". The environment is a discrete grid system where each cell in the grid is defined by a number representing the cAMP concentration in that location.

The class called `agent` is initialized:

```
class agent:
  pass
```

A function called `create_agents` is defined to create the residents of the community:

```
def create_agents():

  # Allow list of slime mold cells, the environment, and updated environment to be accessed outside
    the function
  global agents_list, env, next_env

  # Initialize list
  agents_list = []

  # Create specified number of slime mold cells
  for each_agent in range(n_agents):

    # Create an agent
    agent_ = agent()

    # Assign to a random location
    agent_.x = rd.randint(0, width-1)
    agent_.y = rd.randint(0, width-1)

    # Place slime mold cell on the list
    agents_list.append(agent_)

  # Initialize environment and updated environment
  env = np.zeros([width, width])
  next_env = np.zeros([width, width])
```

`global` allows variables to be accessible outside the function. It is automatically created when the function is run. Note that this function needs 2 variables to be predefined: n_agents (total number of slime mold cells) and width (number of rows and columns of the grid).

Modeling Task 2a: Behavior of the environment

The third task is to "describe the rules for how the environment behaves on its own". cAMP concentration changes in 2 ways:

1. Diffusion
2. Decay.

Let $c(x, y, t)$ be the concentration of cAMP at position (x, y) at time t. c changes by diffusion over space. Using the diffusion equation:

$$\frac{\partial c}{\partial t} = D\nabla^2 c$$

where D is the diffusion constant

In order to implement the Laplacian of c in Python, use Equation (3) where $f(x_n, y_n)$ is represented by `env[x,y]`. Since the environment is a discrete grid system, diffusion happens by the movement of cAMP concentration to the right, to the left, upward, and downward from a grid cell:

$$\nabla^2 c = \frac{\texttt{env[x+1,y]} + \texttt{env[x-1,y]} + \texttt{env[x,y+1]} + \texttt{env[x,y-1]} - 4 * \texttt{env[x,y]}}{(\Delta h)^2} \tag{4}$$

where Δh is the spatial step-size.

c changes also by decaying. Include a term to describe this:

$$\frac{\partial c}{\partial t} = D\nabla^2 c - kc$$

where k is the rate of cAMP decay.

The implementation of the decay of c in Python is simple:

$$kc = k * \texttt{env[x, y]}. \tag{5}$$

Create a function called `update` using Equations (4)-(5):

```
def update():

  # Allow list of slime mold cells, the environment, and updated environment to be accessed outside
    the function
  global agents_list, env, next_env

  # cAMP concentration in each grid cell diffuses (based on diffusion equation) and decays
  for x in range(width):
    for y in range(width):
      next_env[x, y] = env[x, y] + D*((env[(x+1)%width, y] + env[(x-1)%width, y] + env[x, (y+1)%width]
     + env[x, (y-1)%width] - 4*env[x, y])/(dh)**2)*dt - k*env[x, y]*dt

  # Variables for environment and updated environment are updated
  env, next_env = next_env, env
```

% is the modulo operator, ensuring that coordinates on an edge do not produce errors. The variable dt also appears as an additional factor to indicate updating per time-step. Note that this function needs 4 variables to be predefined: D (cAMP diffusion constant), dh (spatial step-size), dt (time step-size), and k (cAMP decay constant).

Modeling Task 2b: Agent interaction with the environment

The fourth task is to "describe the rules for how agents interact with the environment". Slime mold cells increase cAMP concentration through cAMP secretion. Include a term to the equation describing the change in c:

$$\frac{\partial c}{\partial t} = D\nabla^2 c - kc + fa$$

where f is the rate of cAMP secretion and $a(x, y, t)$ is a slime mold cell at position (x, y) at time t. This is easily implemented by adding the following lines to the `update` function:

```
  # Each agent secretes cAMP
  for agent_ in agents_list:
    env[agent_.x, agent_.y] += f*dt
```

Note that the function needs an additional variable to be predefined: f (slime mold cells' secretion rate of cAMP).

Modeling Task 2c: Behavior of agents

The fifth task is to "describe the rules for how agents behave on their own". As mentioned in the Overview, slime mold cells aggregate due to cAMP. So slime mold cells do not exactly behave on their own. But for organization purposes, the following behavior is described here. Keller and Segel modeled chemotaxis using partial differential equations [5].

In this paper, to model chemotaxis, a slime mold cell moves to a random location in its neighborhood with probability

$$\Pr[\text{move}] = \frac{e^{\frac{\Delta c}{c_0}}}{1 + e^{\frac{\Delta c}{c_0}}} \tag{6}$$

which is a sigmoid function. $\Delta c = \text{new } c - \text{old } c$ is the difference between the cAMP concentrations of the cell's randomly chosen location and its current location, and c_0 is the sensitivity of the probability to Δc. Observe that if $\Delta c = 0$, i.e., then the probability of moving to a new location is 50%. If $\Delta c > 0$, i.e., the randomly chosen cell has higher cAMP concentration, then the probability of moving increases; otherwise, if $\Delta c < 0$, the probability decreases. Since probabilities are random, to model this decision, Pr[move] is compared with a random number. Movement happens if Pr[move] is bigger than this random number (the higher Pr[move], the higher the chances that a movement will occur given any random number).

Add the following lines to the function `update`:

```
  # Chemotaxis of slime mold cells with probability based on a sigmoid function
  for agent_ in agents_list:
    new_x, new_y = (agent_.x + rd.randint(-1,2))%width, (agent_.y + rd.randint(-1,2))%width
    if np.exp((env[new_x, new_y] - env[agent_.x, agent_.y])/0.1)/(1 + np.exp((env[new_x, new_y] - env[
    agent_.x, agent_.y])/0.1)) > rd.random():
      agent_.x, agent_.y = new_x, new_y
```

Each slime mold cell is checked. A random grid cell in the slime mold cell's neighborhood is chosen. Equation (6) is used, and the decision to move is as explained above. Note that c_0 in this model is set at 0.1.

Note: Slime mold cells do not interact with each other in this model so Modeling Task 2d ("describe the rules for how agents interact with each other") is skipped.

Visualization

To visualize the states of the system, the function `visualize` is created:

```
def visualize():

  # Initialize subplots
  fig, ax = plt.subplots()

  # Plot the environment using grayscale gradient: light color for low cAMP concentration, dark color
    for high cAMP concentration
  ax.imshow(env, cmap = plt.cm.binary, vmin = 0, vmax = 1)

  # Coordinates of the slime mold cells
  x = [agent_.x for agent_ in agents_list]
  y = [agent_.y for agent_ in agents_list]

  # Plot the reversed coordinates (to be consistent with the plotting of plt.imshow)
  ax.plot(y, x, 'b.')

  # Title
  ax.set_title('Slime Mold Aggregation' + '\n' + 'Time: ' + str(time))

  # Horizontal axis label
  ax.set_xlabel(str(n_agents) + ' Cells || ' + 'cAMP Diffusion: ' + str(D) + '\n' + 'cAMP Decay: ' +
    str(k) + ' || cAMP Secretion: ' + str(f))

  # Remove extra tick marks on the axes
  ax.set_xticks([])
  ax.set_yticks([])

  # Prepare format of file name
  filename = 'Aggregation'

  # Starting filename count
  i = 1

```

```
  # Check if filename already exists; add 1 if it does
  while os.path.exists('{}{:d}.png'.format(filename, i)):
    i += 1

  # Save figure
  plt.savefig('{}{:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)
```

Subplots are used instead of the simpler `plt.plot` so that when the file is run, successive runs of plots do not overlap into one figure. The function plots the 2-dimensional environment using grayscale (atribute cmap): from white (for grid cells with value 0; atribute vmin) to black (for grid cells with value 1; atribute vmax). Thus, the darker the grid cell, the higher the concentration of cAMP. The x- and y-coordinates of slime mold cells are plotted in reverse coordinates to match the way `imshow` graphs the grid: the vertical axis goes from last value to first value. The tight specification in `savefig` removes extra white spaces around the figure.

Implementation

The simulation follows the basic code implementation structure mentioned in the introduction: agents are created (Initialization), then they are allowed to move according to their defined behavior (Updating). Plots are used to visualize (Visualization) what happens to the system before and after the agents move. The model simulates a 1,000 slime mold cells on a 100×100 grid. The cAMP diffusion constant, decay constant, and secretion rate are 0.001, 0.1, and 2, respectively. Spatial and time step-size are both 0.01.

```
# Number of slime mold cells
n_agents = 1000

# Number of rows/columns in spatial array
width = 100

# cAMP diffusion constant
D = 0.001

# Spatial step size
dh = 0.01

# Time step size
dt = 0.01

# cAMP decay constant
k = 0.1

# cAMP secretion rate by a slime mold cell
f = 1.5

# Create slime mold cells
create_agents()

# Needed for label of initial state
time = 0

# Visualize initial state
visualize()

# Update the model 500 times
for i in range(5):
  for j in range(100):
    update()

  # Compute number of iterations
  time = (i+1)*(j+1)

  # Visualize the state every 100interations
  visualize()
```

The system is updated 500 times, showing the graph 5 times (every 100th iteration).

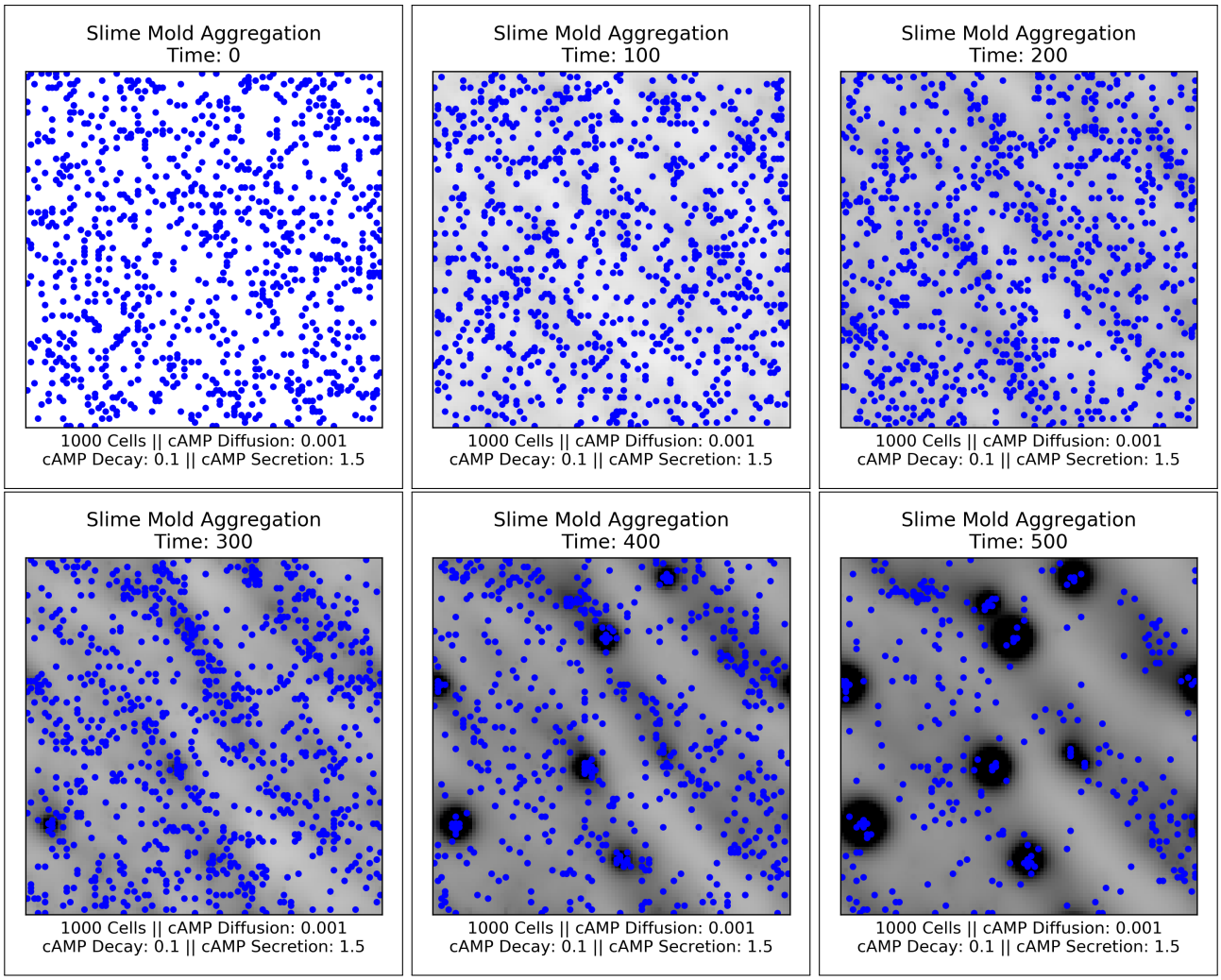

Figure 1: Simulation of Slime Mold Cell Aggregation. Slime mold cells move toward areas of high cAMP concentration.

Initially dispersed, Figure 1 shows that slime mold cells exhibit chemotaxis mediated by cAMP concentration over time, as explained by Keller and Segel in their paper. Dark patches signify increasing concentration of cAMP.

Appendix: Full Code

```
### Needed libraries ###

# For assigning a random location to agents
import random as rd

# For initializing the environment
import numpy as np

# For visualizing the environment using plot
import matplotlib.pyplot as plt

# For checking if a filename already exists, to avoid overwriting files
import os



### Initialize class to create slime mold cells ###

class agent:
  pass



### Create slime mold cells ###

def create_agents():

  # Allow list of slime mold cells, the environment, and updated environment to be accessed outside
    the function
  global agents_list, env, next_env

  # Initialize list
  agents_list = []

  # Create specified number of slime mold cells
  for each_agent in range(n_agents):

    # Create an agent
    agent_ = agent()

    # Assign to a random location
    agent_.x = rd.randint(0, width-1)
    agent_.y = rd.randint(0, width-1)

    # Place slime mold cell on the list
    agents_list.append(agent_)

  # Initialize environment and updated environment
  env = np.zeros([width, width])
  next_env = np.zeros([width, width])



### Simulate change in cAMP concentration and chemotaxis of slime mold cells ###

def update():

  # Allow list of slime mold cells, the environment, and updated environment to be accessed outside
    the function
  global agents_list, env, next_env

  # cAMP concentration in each grid cell diffuses (based on diffusion equation) and decays
  for x in range(width):
    for y in range(width):
      next_env[x, y] = env[x, y] + D*((env[(x+1)%width, y] + env[(x-1)%width, y] + env[x, (y+1)%width]
     + env[x, (y-1)%width] - 4*env[x, y])/(dh)**2)*dt - k*env[x, y]*dt

  # Variables for environment and updated environment are updated
  env, next_env = next_env, env

  # Each agent secretes cAMP
  for agent_ in agents_list:
    env[agent_.x, agent_.y] += f*dt

  # Chemotaxis of slime mold cells with probability based on a sigmoid function
```

```
  for agent_ in agents_list:
    new_x, new_y = (agent_.x + rd.randint(-1,2))%width, (agent_.y + rd.randint(-1,2))%width
    if np.exp((env[new_x, new_y] - env[agent_.x, agent_.y])/0.1)/(1 + np.exp((env[new_x, new_y] - env[agent_.x, agent_.y])/0.1)) > rd.random():
      agent_.x, agent_.y = new_x, new_y



### Visualize the state ###

def visualize():

  # Initialize subplots
  fig, ax = plt.subplots()

  # Plot the environment using grayscale gradient: light color for low cAMP concentration, dark color for high cAMP concentration
  ax.imshow(env, cmap = plt.cm.binary, vmin = 0, vmax = 1)

  # Coordinates of the slime mold cells
  x = [agent_.x for agent_ in agents_list]
  y = [agent_.y for agent_ in agents_list]

  # Plot the reversed coordinates (to be consistent with the plotting of plt.imshow)
  ax.plot(y, x, 'b.')

  # Title
  ax.set_title('Slime Mold Aggregation' + '\n' + 'Time: ' + str(time))

  # Horizontal axis label
  ax.set_xlabel(str(n_agents) + ' Cells || ' + 'cAMP Diffusion: ' + str(D) + '\n' + 'cAMP Decay: ' + str(k) + ' || cAMP Secretion: ' + str(f))

  # Remove extra tick marks on the axes
  ax.set_xticks([])
  ax.set_yticks([])

  # Prepare format of file name
  filename = 'Aggregation'

  # Starting filename count
  i = 1

  # Check if filename already exists; add 1 if it does
  while os.path.exists('{}{:d}.png'.format(filename, i)):
    i += 1

  # Save figure
  plt.savefig('{}{:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)



### Simulation ###

# Number of slime mold cells
n_agents = 1000

# Number of rows/columns in spatial array
width = 100

# cAMP diffusion constant
D = 0.001

# Spatial step size
dh = 0.01

# Time step size
dt = 0.01

# cAMP decay constant
k = 0.1

# cAMP secretion rate by a slime mold cell
f = 1.5

# Create slime mold cells
create_agents()
```

```
147
148 # Needed for label of initial state
149 time = 0
150
151 # Visualize initial state
152 visualize()
153
154 # Update the model 500 times
155 for i in range(5):
156   for j in range(100):
157     update()
158
159   # Compute number of iterations
160   time = (i+1)*(j+1)
161
162   # Visualize the state every 100interations
163   visualize()
164
165 ####### End of Code #######
```

References

[1] Bonabeau, Eric. "Agent-Based Modeling: Methods and Techniques for Simulating Human Systems." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, 14 Mar. 2002, pp. 7280-7287.

[2] Castiglione, Filippo. "Agent Based Modeling." *Scholarpedia*, Brain Corporation, 29 Sep. 2006, www.scholarpedia.org/article/Agent_based_modeling.

[3] Clark, Andrew G, et al. "3D Cell Migration in The Presence of Chemical Gradients Using Microfluidics." *Methods in Cell Biology*, vol. 147, Elsevier, 2018, pp. 133-147.

[4] Keller, Evelyn F, and Lee A Segel. "Initiation of Slime Mold Aggregation Viewed as an Instability." Journal of Theoretical Biology, vol. 26, 1970, pp. 399-415.

[5] Keller, Evelyn F, and Lee A Segel. "Model for Chemotaxis." *Journal of Theoretical Biology*, vol. 30, 1971, pp. 225-234.

[6] Mathiesen, Jesper Mosolff, et al. "cAMP Biosensors Applied in Molecular Pharmacological Studies of G Protein-Coupled Receptors." *Methods in Enzymology*, vol. 522, Elsevier, 2013, pp. 191-207.

[7] Sayama, Hiroki. "Agent-Based Models." *LibreTexts*, 23 June 2019, https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book%3A_Introduction_to_the_Modeling_and_Analysis_of_Complex_Systems_(Sayama)/19%3A_Agent-Based_Models/19.02%3A_Building_an_Agent-Based_Model.

[8] Sayama, Hiroki. "Agent-Environment Interaction." *LibreTexts*, 10 November 2019, https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book%3A_Introduction_to_the_Modeling_and_Analysis_of_Complex_Systems_(Sayama)/19%3A_Agent-Based_Models/19.03%3A_Agent-Environment_Interaction.