Background:

Slime molds are an essential decomposer in many ecosystems, and demonstrate interesting and complex behaviors. They begin as single individual cells similar to amoebas. Under certain conditions, like when lots of food is close, cells can aggregate to form the visible slime molds people are familiar with. One type of mold consists of many different cells clustered together. Another is essentially one enormous, visible cell with many nuclei. These molds can travel at around 1mm/hour, and they leave behind a trail of chemicals. Other cells can sense these chemicals, and travel along these chemical gradients, resulting in the visible slime mold 'colony' known as a 'pseudoplasmodium.'

In a similar fashion, slime molds can detect food chemicals as well. A piece of food will leech off chemicals over time, resulting in a field of chemicals around it in a gradient. There will be more leeched chemicals the closer to the food source. Slime molds detect these chemicals and travel up the gradient, searching for stronger densities. In this way, they can forage around their environment eating food. This foraging behavior is what this simulation focuses on. Researchers have found that slime molds travel between food very efficiently, displaying something that seems like intelligence, without actually having a brain or nervous system.

Agents and rules:

1) Foods
    - Spawned at beginning of sim, none added after it has started
    - The number spawned can be changed by the user
    - Stationary, do not move
    - 'Leech' chemicals onto surrounding patches, changing their 'strength' variable
        - The radius in which chemicals are leeched can be changed by the user
    - Are eaten by cells and consequently disappear
2) Cells
    - Represent a cell of a slime mold
    - Do not move
    - There is one 'active cell' at a time
    - The active cell spawns a new cell on a neighboring patch with the highest strength (with some randomness), and this cell becomes the new active cell
    - When all neighboring patches have 0 strength, an active cell will spawn cells in all directions called 'searchers'
        - Searchers spawn a cell on a neighboring patches if the cell has less than 4 neighboring cells
        - Once a searcher is spawned on a patch with nonzero strength, the cells resume normal activity and all searchers become inactive
3) Nodes
    - Represent the location that foods were during the first part of the simulation
        - One will also represent the starting place of the first cell
    - Have no movement, cannot be spawned after they are initially placed, essentially remain stationary and unchanged
    - Are linked by links
    - Display a label showing the order in which the mold ate the food that was once there
4) Patches
    - Each has a strength variable which is set at the beginning, and each time a food is eaten
        - Strength is determined by the patch's distance from the closet food source

- Each has a variable to remember whether it had a food on it, so nodes can be spawned on these patches
- Each has a visited-queue variable, which if the patch had food, stores the order in which it was visited by the mold

User's Guide:

This simulation represents the foraging behavior of slime molds, and examines how the path taken relates to the shortest possible path between food sources. On setup, the user can determine how many food sources are available. Each piece of food has a chemical gradient around it, which the mold can sense. The user can change the radius around a food in which this gradient will appear. A single mold cell is spawned. When the simulation is run, the cell will look at each of its neighboring patches searching for food molecules. If the cell finds a gradient, it spawns a new cell at the strongest location. In this manner, cells appear to travel up the gradient towards a food source. Once found, the food source is eaten and disappears. The landscape and gradients are redrawn to reflect this change. If a cell cannot find a gradient, it causes the mold to enter a new mode. The mold will expand in all directions until it finds some food chemicals, after which it will return to it's normal behavior of travelling up the gradient. The last parameter that the user can change is the efficiency with which the mold forages for food. When a cell senses a gradient, it finds the location that has the highest concentration of food chemicals and spawns a new cell there, if search efficiency is 0. As the number is increased, cells will find the area to spawn a new cell, then randomly put it up to +/- (search-efficiency/2) units away. This can cause the mold's path to meander and potentially miss food, or go to a food that is not the closest.

After the mold has eaten all the food, the next phase begins. The locations of where the food once was are marked again, this time with the order in which they were eaten. The path that the mold took (only when it was actively moving towards food, and not when searching in a big radius around itself) is marked. Each food location and cell starting point are linked to each other, so that each node is linked directly with every other node. The user can then click the find-shortest-path button. Starting at the cell starting point, it finds the shortest link between that point and a food source. Then, the next shortest link is highlighted until all the food nodes have been connected by one link path. As this process occurs, the user can see which node is reached next, and if this sequence matches the order in which the mold ate the food. At the end, the simulation will display the shortest possible path that links all the nodes and compare it to the path that the mold took while foraging.

Some assumptions were made in the making of this simulation. One is that food sources always have a perfectly circular radius of chemicals around them. Also, it assumes that the gradient field changes immediately when food is eaten. The simulation also takes some liberty with the way that the mold travels, and ignores available energy (it has infinite) and reproduction. There is no cell aggregation into a mold, and instead cells are produced in new areas and do not move themselves. Another assumption is in how the mold will behave when no food is nearby. Also, the molds move in a directed line up a gradient. In real life, the mold is an amorphous mass, not a defined line, that expands towards a food source. Watching videos of slime mold movement will help demonstrate the differences between real life and simulated movement.

One important thing to keep in mind is how the mold and shortest patch algorithms differ. Once a food source is eaten, the mold finds the cell that is in the strongest chemical area. This can be any one of its cells. In the shortest patch algorithm, linkages can only be made between nodes. There cannot be a path from the middle of a link to a node, only node to node.

Code documentation and validation:

The overview of the code will go through each procedure, explaining what they do. Important ones will go into detail. Low-level documentation is in the code.

- setup: Sets up the simulation are some global variables. Set patch vars, create foods and starting cell
- go: Ticks through the simulation. if global var go-type is 0, cells forage like normal. If 1, cells enter search mode and search in a radius around the last active cell.
- setup-patches: sets all variables to defaults, determines strength based on the distance it is from the closest food agent
- setup-foods: gives food a random location, sets default values to vars, asks the patch it is on to set var had-food to true
- setup-cells: gives a random location, sets default values to vars
- setup-nodes: give random location, set default values to variables, but calls itself again if the random location is not on a patch with had-food = true
- cells-forage: Determines cell behavior for when in foraging mode. Frist checks that there is an active-cell, and if so begins. The active-cell is asked to check the strength of the patch it is on. If 0, go-type is set to 1 (to enter searching mode). All cells become searcher cells and are added to the list of searchers. It spawns a new searcher cell itself. If the patch strength is not 0, then foraging behavior is continued. The active cell looks at all the neighboring patches and finds the one with the highest strength. It then hatches a new cell here. If search-efficiency > 0 as set by the user, the x and y coordinated at which this cell would have been spawned are subject to some randomness. As long as this new location is on the map, the new cell is spawned and becomes the active-cell.
- new-cell [px py]: Called when creating a new cell agent. Given x and y coordinates, sets the new cell's position and makes it a forager.
- eat-food: Asks all cells if it occupies the same patch as a food. If so, it asks that food to die. The patches are setup again to represent the loss of one of the foods and its gradient. It updates the visited-queue var of the patch to show it had food and in which order that food was eaten, then increased the global visited counter.
- new-active-cell: Sets a new cell as the global var active-cell. Is called after food is eaten or when mold is searching for food. Checks all the cells to find the cell that is on the patch with the highest nonzero strength. This cell becomes the new active-cell. If the new active cell is that same as it was when the procedure was called, it returns false. If there is a new active cell, it returns true.
- new-searcher-cell: Called when a new searcher cell is to be created. First, it checks if the current cell has more than 4 neighboring cells. If so, this cell is no longer a searcher and is removed from the list of searchers. Otherwise, a neighboring patch with no cells on it is chosen, and a new cell is hatched there. This new cell becomes a searcher cell and adds itself to the list of searchers.
- cells-radial-search [who-active-cell]: First checks if a new active-cell can be found. If so, that means a cell has found a chemical gradient and has become a new active-cell. go-type is set to 0 to resume foraging mode, and all cells set their searcher variables to false and removed themselves from the list of searchers. Otherwise, the list of searchers is iterated through, asking each cell in the list to call new-searcher-cell.
- djikstra: Note – doesn't actually do Djikstra's algorithm, does something similar but visits all nodes along the way, name is legacy, easier to leave unchanged. At the end of the first phase once all foods have been eaten, all forager cells ask the patches they are on to set their color to orange to show the path of forager cells. Then nodes are created where the foods once were and sets their color. Another node is created at the first cell's starting position, and is given a different color and labelled "Staring point." Then all cells are killed. All nodes create links to every other node, the next-node variable is set to the node at the cell's starting point, and links are formatted.

- find-shortest-path: Progresses on tick and is iterated through by user button. Sets a max length to 1000, then asks links that connect one visited node to one unvisited node and that are not yellow (already confirmed links) to compare their length to the max length. If it is shorter is set its length as the max length and set itself in a keeper var. This way the shortest link meeting the conditions is found. Then this shortest link is reformatted (becomes yellow) and set the nodes it connects as visited. Once every node is visited all other links are hidden, to display the yellow links only.
- Validation: There was no specific way this code was validated. After each feature was added, the sim was run to ensure that the feature was behaving as intended. The feature was also reviewed carefully by hand to make sure edge cases were accounted for.

Hypothesis testing/data analysis:

The main hypothesis that this simulation intended to test was whether or not slime molds are capable of finding the shortest/most efficient path between their starting location and each piece of food. To test this, the simulation was run multiple times with independent variables of gradient-radius and search-efficiency. The number of foods was 10 each trial. The path that the mold took was compares to the result of the shortest path algorithm.

First, with a gradient-radius set to 100, six trials were run, with search-efficiency set to 0, 1, 2, 3, 4, and 5.

Search-efficiency	
0	All in order
1	All in order
2	All in order
3	One out of order
4	One out of order
5	Two out of order

Then, the gradient radius was set to 300 and another six trails were run.

Search-efficiency	
0	All in order
1	One out of order
2	All in order
3	All in order
4	All in order
5	All in order

These results show that when working with a good searching efficiency, the shortest possible path between food nodes is usually found. When the gradients are smaller, molds have to search more, and can often find foods that are not necessarily the closest to the most recently visited food, but closest to the path which the mold has taken and grown across. these results can also be affected by the randomness in the map. Foods are spawned very close together can lead to a random placement of a cell using high search-efficiency closer to another food, and the path may change.

Sources:

http://bioweb.uwlax.edu/bio203/2010/renner_brad/reproduction.htm
https://www.quantamagazine.org/slime-molds-remember-but-do-they-learn-20180709/

https://bio.libretexts.org/Bookshelves/Introductory_and_General_Biology/Book%3A_General_Biology_(Boundless)/23%3A_Protists/23.2%3A_Characteristics_of_Protists/23.2B%3A_Protist_Life_Cycles_and_Habitats
https://www4.stat.ncsu.edu/~reich/CSUMS/presentations/Slime.pdf
https://www.sciencedirect.com/science/article/pii/S0925231214009278?via%3Dihub
https://www-sciencedirect-com.ezproxy.wpi.edu/science/article/pii/S0925231214009278#bib19
https://doi.org/10.1016/j.neucom.2012.10.044