Web-based interactive simulation of mycelial growth

Introduction:

For my project I made an interactive simulation of mycelial growth. This was inspired by an interest in the network intelligence displayed by fungal mycelium, and was designed with the intention of modelling an approximation of their foraging behaviour. The behaviour I wanted to model could be summarised as the ability to explore the environment, discover nutrients, remodel the network, then explore the environment again.

This was something I found fascinating in general, but the topic is useful to explore for a few reasons. Amongst organisms displaying network intelligence, fungi are understudied and appreciated [1], meaning a user-friendly simulation of their growth could be an introduction to thinking about fungal intelligence. There are also some biological mechanisms underlying long distance communication throughout fungal networks that are still not well understood, and so a more robust extension of my program could be used to explore these mycological questions [1, 2, 3].

The process of writing this program was a fun creative project that involved giving names to unknown biological functions, and constructing a semi-decentralised network that displays emergent intelligence, but could also be used as a foundation for further mycological study.

Overview of relevant mycelial behaviour:

Mycelium is the root-like structure of a fungus, which is itself composed of a mass of strands called hyphae. Hyphae are made of hyphal cells, and at the tip of each branch of hyphae, is a hyphal tip. This root network demonstrates intelligence similar to that observed in organisms such as slime moulds, but is also comparable to the swarm intelligence of ant colonies. In this comparison, each hyphal tip plays the role of an individual agent, because despite being connected to every other individual, they always make decisions locally. This makes fungal mycelium a decentralised network, with no central planning to its overall behaviour. Instead, its complex and well adapted behaviour emerges from the interaction between each of its hyphal tips, acting on a few basic rules within its local environment.

The specific kind of fungi that I've modelled my simulation on are saprotrophic cord forming fungi, that have been heavily studied by Lynne Boddy for their foraging behaviour [1, 2, 3, 4, 5]. These are non resource-restricted fungi, meaning that they don't spread spores, but instead grow out long thick cords, made from parallel strands of hyphae, that seek new nutrient sources to colonise [2, 3]. This results in "long-lived systems which interconnect discrete nutrient resources." [3]

Different species will be differently adapted to their environments. For example, a species that is adapted to colonising leaf litter is likely to find nutrient sources all around it, but in small quantities, making a highly branching, but short ranged foraging strategy more efficient. Another species may be adapted to stretch for metres between decaying fallen trees, which it will then spend a long time colonising before growing out again [3]. This species would waste resources spreading out too much over a short distance, and can instead rely on the size of its target to make sure its long and relatively unbranched hyphae can find it. Some species even "switch between slow-dense and fast-effuse growth which effectively provides them with two foraging strategies" [2].

Once this fungus reaches a new nutrient resource, responses include strengthening of mycelial interconnections between resources, regression and recycling of non-connected mycelium, finally followed by renewed searching [1].

Strengthening of connected cords, and the regression of unconnected ones, is an important function in fungi that allows them to efficiently reallocate biomass around the network. Biomass re-allocation may take place via fluid flow, from 'sources' where nutrients are less utilised, to 'sinks', where the nutrients are being rapidly used, such as for growth. This makes the fungal network a hydraulic system utilising "growth-induced mass flow" [4]. High current cords can be preferentially thickened, reducing the friction of fluid flow which offsets the expense of this adjustment [4].

Implementation

Overview:

On opening the web page, the user will see some instructions for the user interface, and a few sliders and buttons alongside the canvas. The user can place food objects around the canvas by scrolling, to change the size of the object, and then left clicking to place the object at the mouse's location. The sliders control parameters that adjust the branching behaviour of the fungus. The fungus is then placed with those parameters, and the simulation can be started. The fungus will begin growing, branching, and hopefully detect food. Once it does, it will be attracted to it until it is inside the food object, at which point it will begin colonizing it. Once it has spent an amount of time colonizing the resource that is proportional to its size, it will create a new fungus object, and begin growing out from this resource in the same way as it started. As the fungus grows, it will eventually grow over 5000 cells, which I've chosen as a checkpoint. At this point, some hyphal tips that havent found food will receive a signal to be removed recursively down to the next junction. This simulates a reallocation of biomass, as these removed cells allow cells to be grown elsewhere, hopefully in more productive directions. If no cells can be removed, and the fungus is at 5000 cells, the program will halt to prevent crashing the browser by creating too many objects.

Code details

My program is accessible via a url that points to my home folder on raptor. It's contained in a single html page that links all the Javascript files that make up the bulk of the code. The code is written in Javascript using the P5.JS library. I had some experience using this library, and so I used it in favour of other tools, like Three.js for example, despite the fact that P5 may offer less performance. P5 utilises HTML's canvas object which is a cartesian plane, and so my program is based on vectors. For example, the fungus starts at a X/Y location, every branch has a root cell at that location, and each child cell is placed a set distance away from its parent cell in whatever direction it's growing. It is visualised using only a few basic P5 functions, such as line() to create the strands of hyphae, and rect() to create food objects.

The code is broken up into eight Javascript files, and one index.html file that links them all together. Four files are dedicated to the different classes that construct the fungus organism itself: *spore.js, fungus.js*, *branch.js* and *cell.js*.

Main classes:

spore is very simple, and is essentially a container for other objects. All it has in its constructor is a position, and a children array.

fungus creates a spore object, and adds a user specified amount of branches to the spore's children array. The class also contains two functions that update the entire organism by calling functions on each branch to grow, branch, detect food, and to actually draw the

resulting behaviour. *updateAndDrawTree(cell)* is a recursive function that moves in a pre order depth first search, and draws each cell. Two more functions are contained in the fungus class, which control regression. These move recursively in the other direction, from the tip down towards the root.

branch is a slightly more complex class. It is technically a tree data structure, where each node in the tree is a cell object. Its properties include a root cell, an attractor, a direction vector, and an array of hyphal tips. *grow()* iterates over every hyphal tip, and calls *addChild(tip, index)* with each cell as a parameter. The corresponding function within the cell class is where most of the code is contained. The cell returns a new cell to the branch class, which replaces the original cell that the function was called on in the hyphal tips array. *branch()* iterates over every hyphal tip, and if it passes a check, calls cell.branch() on each tip. *detectFood()* is the final function, which just iterates over every hyphal tip and calls cell.detectFood(), much like *branch().*

cell is where the most code is contained, mirroring the fact that the hyphal tips are the site of decision making. It has seventeen properties and seven functions that control the bulk of the fungi's behaviour. *attract(attractor, vector)* takes two vectors as arguments, and returns a vector that is set to a standardised size, and points from the vector to the attractor, or in the opposite direction of the attractor, depending on if the cell is being attracted or repelled from the attractor. *addChild()* creates a vector that is slightly rotated in a random direction from this cells' movement direction, and is tugged back towards its attractor by adding the vector returned from the attract function. This causes it to explore semi-randomly, while still moving generally away from its source, as well as towards food objects once they've been detected. *branch(branch, n)* causes the cell to grow once as usual, before creating two other child cells which grow out at an angle from the original direction. Due to a bug in my code, branches tend to be messier, creating four side branches instead of just two; this will be discussed in the evaluation. *detectFood(branch)* iterates over every food object, and checks if this cell is near, or inside it. If it is within a range proportional to the size of the food object, representing fungi's rudimentary sensory perception [1], it will be attracted to the food. Once it is inside, it will begin colonizing the food, again, for a time that is proportional to the size of the food resource. It will also call *foundFood(cell, food, branch),* which is a recursive function that sends a signal from this tip, down to the root. It is visualised by highlighting it in white, and this highlighted branch should never be pruned. Due to another bug, however, sometimes it is.

Other files:

gui.js contains global variables, and the code that manages the GUI.

cursorSquare.js follows the mouse and visualises the size of the food they'd like to place, and *food.js* contains code for the interaction between food objects placed by the user, and hyphal tips.

sketch.js contains the driver code. It contains global functions that are used in *setup()* as well as others like *checkEndState(),* which halts the program when too many objects are created. It also contains two P5 functions:
- *setup(),* which is called once when the page is loaded. This initialises a canvas object, and generates the GUI for the user.
- *draw()* is called at a default frame rate of 30 fps, and enables easy drawing to the canvas. This function contains a loop over all fungi objects to update, which leads to all of the behaviour we see. It also increments a *tick* variable which tracks time, and allows certain functions to be called incrementally.

Evaluation

Positives:

I managed to produce a web-based interactive simulation of mycelial growth. I satisfactorily implemented growth, branching, food detection, attraction, colonisation, regression, and re-exploration, although there are some changes I would still make which will be discussed later.

The fact that the program is usable with default settings means it only requires pressing a few buttons to start seeing something interesting. This low barrier to entry means it's potentially accessible to anyone with any kind of interest in the project. It produces an interesting animation that can be fun to watch regardless of any deeper interest in the mycological aspects, which again makes it very accessible. The added ability to adjust certain parameters of the fungi's branching however, means that not only is it slightly more engaging to play with, but it can also represent a wider variety of mycelial behaviours. By placing food resources of different sizes, and in different locations, different environments can be created, and the user can observe how different branching parameters will change the effectiveness of a fungus' foraging strategy.

Negatives:

I didn't leave enough distinct time after coding my project and the deadline to fix bugs that emerged towards the end of the project. There are two key bugs that I've mentioned already. One means that when branching occurs, instead of two child branches being created, there are instead four. From what I can tell however, it's still only two child cells that are contained in the data structure, so the additional two branches don't have any further functions called on them. They don't break the simulation, or appear out of place, but it doesn't accurately portray any real fungus, nor anything I intended to simulate. The other bug means that hyphal tips which have found food are pruned. I intended my program to leave behind a network of interconnected resources, but because of this bug, sometimes there will be colonised resources that are disconnected from the rest of the network. It seems to occur when too much branching happens all at once; if I knew any more then I would be able to fix it. Given more time I would have also worked a bit more on the GUI to make it more user friendly and engaging.

I limited the growth of the fungus to a set amount of cell objects, and used this as a halting condition to the program to prevent it growing out of control. This allows the fungus to continue to grow for a long time, while keeping the biomass of the organism restrained. Ideally this wouldnt be a restriction, and I would instead model the growth of the fungus to have a limited amount of resources, which is then replenished by feeding; some species of fungi can grow their mycelium *indefinitely* under the right conditions, but this isnt feasible for a browser based simulation if I want each cell to be represented by an object in memory, which is required to communicate back and forth along the branches.

As mentioned before, a fungus may encounter vastly different sizes of nutrients, and in reality would change its resulting behaviour. For simplicity, my simulation assumes a favourable reaction, which would perhaps in reality be saved for specifically when the new resource is considerably larger than the original "food base". That being said, there are small differences in my code to account for nutrient size, such as the time spent colonising it, the amount of branches that spread from it, and the speed with which the connected branches to that food source thicken. [2, 3] The distance between the original food base and the new resource would also impact its behaviour, which is not accounted for in my code [2]

I would have liked to have implemented a version of the “sit and wait” foraging strategy, where food can be detected along the length of its network rather than only in the tips. It could be added easily into the update function that is called on each cell before being drawn to the screen [5].

I’d have also liked to have implemented a model of fluid flow that's more accurate to real regression behaviour. As it is, cells are removed which keeps the fungus under 5000 cells and allows it to continue growing, but there is no direct relationship between the regressed cells and the ones which are produced.

Conclusion

The project was my first self-directed creative coding project. It taught me the basics of designing and implementing a system from scratch, and was also good practice coding in a more object-oriented manner than I’ve had to in previous modules.

I used git for the first time when developing the code for this project. I don’t think I used it in a very efficient way, but it got me over the initial hump that prevented me from using it in the past.

To implement the branches I had to practice implementing a few different data structures before settling on trees, which then required practice in recursively navigating trees using different basic algorithms, which were all useful coding skills to learn.

I’m happy with the result of my project, although there are a few things I would change to make it a more realistic simulation, and to present it in a more aesthetically pleasing way.

Bibliography

1 - Boddy, L. 1999, "Saprotrophic cord-forming fungi: Meeting the challenge of heterogeneous environments", *Mycologia,* vol. 91, no. 1, pp. 13-32.

2 - Boddy, Lynne. “Saprotrophic cord-forming fungi: warfare strategies and other ecological aspects.” *Fungal Biology* 97 (1993): 641-655.

3 - Kristin Aleklett, Lynne Boddy, *Fungal behaviour: a new frontier in behavioural ecology*, Trends in Ecology & Evolution, Volume 36, Issue 9, 2021, Pages 787-796

4 **-** Luke Heaton, Boguslaw Obara, Vincente Grau, Nick Jones, Toshiyuki Nakagaki, Lynne Boddy, Mark D. Fricker, *Analysis of fungal networks*, Fungal Biology Reviews, Volume 26, Issue 1, 2012, Pages 12-29

5 - Mark D. Fricker, Dan Bebber, Lynne Boddy, *Chapter 1 Mycelial networks: Structure and dynamics*, Editor(s): Lynne Boddy, Juliet C. Frankland, Pieter van West, British Mycological Society Symposia Series, Academic Press, Volume 28, 2008, Pages 3-18