Armand Lala & Davide Vinci

Urban Mycelium

ARCH 4342 ΠΛΗΡΟΦΟΡΙΚΗ: Υπολογιστικές Προσεγγίσεις στις Τέχνες και Επιστήμες

Project introduction

Urban Mycelium is an interactive simulation that explores urban behavior through the lens of biological intelligence. Inspired by the decentralized logic of *Physarum polycephalum* (a slime mold), this project applies agent-based modeling to a map of Rome, reimagining the city as a living substrate for exploration, settlement, and transformation.

The simulation begins with a single user interaction: a click spawns a colony of mold-like agents onto the darkest paths of the city map, which represent accessible urban channels. These agents move organically, combining random motion with attraction to specific urban districts based on dynamic values: income, services, and toxicity. As they move and cluster, they affect these values, creating feedback loops that reflect both growth and stress in different parts of the city.

Each district functions as a responsive node. Agents are drawn to areas with high utility but can also cause degradation (e.g. increased toxicity) when overcrowded. Through this interaction, the simulation visualizes how individual actions influence urban systems over time.

A heatmap and trail system reveal the most traversed paths, while a dynamic interface allows users to control agent speed and explore node data in real time. The result is a living, data-driven network, a digital mycelium that grows, adapts, and reshapes the map of Rome in response to collective movement.

Urban Mycelium is both a technical and conceptual project, using code (p5.js) not only to simulate behavior, but to reflect on the complex, emergent patterns that define cities.

Inspirations

The core logic of Urban Mycelium draws inspiration from bio-inspired computation, specifically the behavior of the slime mold *Physarum polycephalum*. This organism, as shown in the research of Atsushi Nakagaki and his team, can solve complex spatial problems: such as finding the most efficient path between food sources through decentralized, local

decision-making. In one famous experiment, *Physarum* replicated the Tokyo rail network, optimizing connections in a way that mirrored human-engineered systems.

In order to find a visually appealing effect of this mold we also found other great projects, such as:

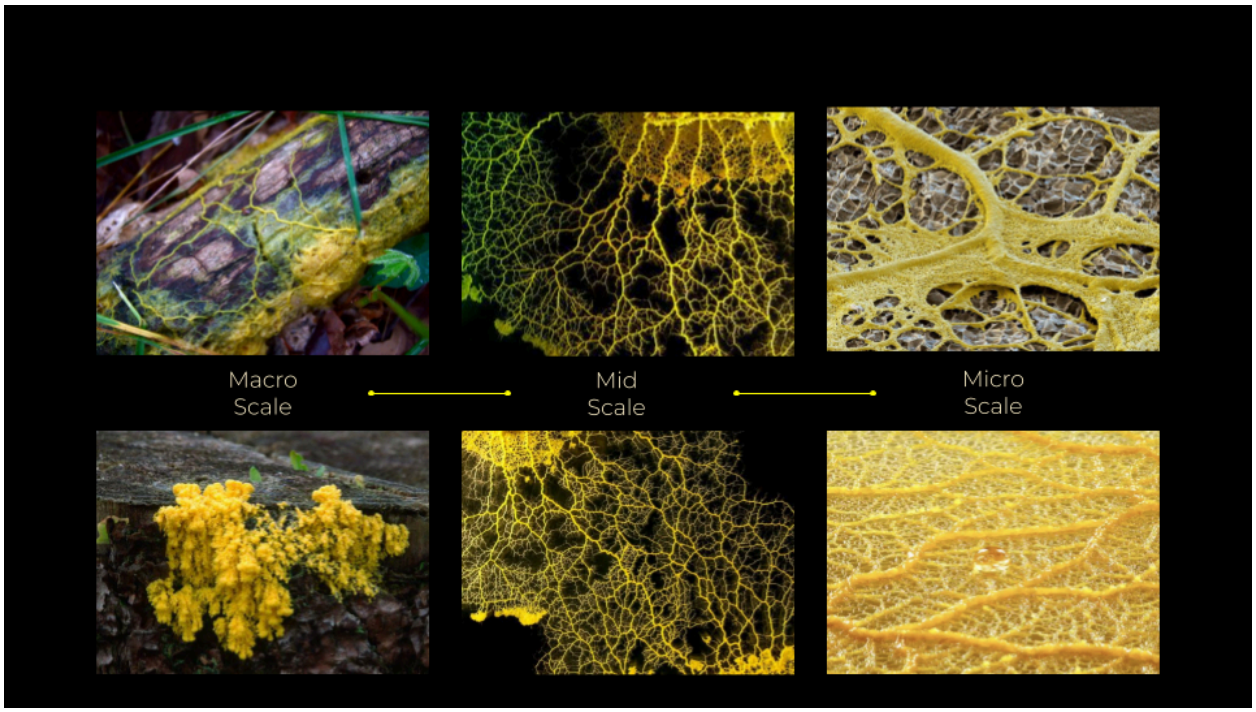
- Physarum Poly-xylem City
- 'living screens' proposes slime mould as and architectural application
- Pompidou exhibition shows how slime mould can help with city planning | RIBAJ
- Slime Molds Simulation - IAAC BLOG
- Slime Mould – Building Construction

Also a great Youtube video showing how to obtain a great visual effect for slime molds in p5.js:

Physarum Poly-xylem City

What is the *Physarum polycephalum?*

Physarum polycephalum is a slime mold protist known for its large, single-celled, multinucleate form called a plasmodium. Despite lacking a brain, it can solve mazes, optimize networks, and even learn from experience, making it a fascinating subject in biology and unconventional computing.



First applications

The first step was to achieve a visually appealing slime effect on p5. And these are some of the first applications:

- p5.js Web Editor | BIOLUMINA_FIN_BIANCA
- p5.js Web Editor | BIOLUMINA_FIN
- p5.js Web Editor | BIOLUMINA_FIN copy

There is also a connection in these previous projects with the urban environments. The molds run through the road system of 5 big world famous cities.

Why Rome?

Rome was chosen as the case study for *Urban Mycelium* because of its unique urban complexity, historical layering, and recognizable spatial patterns. Unlike cities with more rigid or grid-based layouts, Rome is characterized by organic growth, winding paths, and infrastructural networks that have evolved over centuries. This makes it an ideal terrain for simulating behavior inspired by natural systems like mycelium or slime mold.

Moreover, Rome's diversity in socio-economic conditions across its districts: from high-income central neighborhoods to more peripheral areas with varying public service access and environmental challenges; provides rich data for modeling interactions between agents and urban zones. Each neighborhood can function as a distinct node in the simulation, with its own attractors and deterrents, making the dynamics more realistic and meaningful.

Where did we get the data?

The values for income, public services, and toxicity assigned to each Roman district in *Urban Mycelium* are not based on direct official datasets, but were instead constructed and normalized manually to simulate realistic urban conditions and make the model function effectively.

The district coordinates (e.g., Parioli, San Lorenzo, EUR) were selected based on real geographic locations in Rome, and their associated values were inspired by general knowledge and demographic patterns of the city. For example:

- Parioli is known to be an affluent area with good services and low pollution, so it was assigned high income and service scores and low toxicity.

- San Lorenzo, being more student-heavy and urbanized, received moderate service scores and higher toxicity.

- Peripheral zones like Cecchignola or Centocelle were modeled with lower income/service levels and higher toxicity to reflect environmental or infrastructural marginalization.

Each value (income, services, toxicity) was normalized between 0 and 1, to be easily used in the simulation's utility function. The values are qualitative approximations — their purpose is not statistical accuracy but to create meaningful contrasts that mold agents can respond to. This allows the simulation to demonstrate emergent behavior and urban feedback dynamics in a simplified but believable way.

Description of the code

The code for *Urban Mycelium* builds an interactive simulation based on the behavior of autonomous agents (the "mold") moving through a digital map of Rome. The structure is divided into several main parts that handle graphics, logic, and interaction.

When the sketch starts, the program loads an image of Rome (`roma_1.png`) which has been processed to highlight dark paths. These paths represent viable routes that the mold can travel. The image is resized and turned into pixel data so that the program can check whether a specific point on the map is walkable.

Two off-screen canvases are used in parallel:

- `trailCanvas` shows the short-term movement of the mold as glowing trails.

- `heatmapCanvas` builds up a long-term trace of where the mold has passed, using color to show "hotspots" of activity.

The mold agents are created when the user clicks on the canvas for the first time. They spawn at the nearest valid location (a thick channel) and begin exploring the map. Each mold moves based on a combination of random wandering and attraction to nearby “nodes,” which represent districts of Rome. These nodes have three properties: income, public services, and toxicity.

The mold chooses which node to approach based on a calculated “utility” score (income + services − toxicity), but with a bit of randomness added. If it finds a node with high utility and stays near it long enough, it can decide to "settle" there. Once settled, the mold stops moving but still influences the data.

Each node is a point on the map representing a real Roman neighborhood. The values for income, services, and toxicity are manually set based on general knowledge of the city. These values change during the simulation: income and services go up slightly when mold gathers near a node, but toxicity increases if the population becomes too dense. This models a feedback loop, where too much growth can lead to environmental stress.

The user can hover over any node to see a popup with information: the district’s name, current values, and how many mold agents are interacting with it.

Visually, the simulation shows agents as pulsating blue circles. As they move, they leave behind soft trails on the `trailCanvas`, and their movement adds color to the `heatmapCanvas`, making it easy to see which parts of the city are most active.

The simulation also includes a slider that allows the user to control the speed of the agents in real time, offering a way to slow down or accelerate the spread.

The code creates a system where simple agents, following local rules and responding to their environment, generate complex, organic-looking patterns. It reflects how individuals can collectively shape the dynamics of a city over time, just like mycelium growing through a landscape.

Use of AI?

To support the documentation of our final project, we used ChatGPT as a tool to help us clearly organize our ideas, summarize the structure of our code, and improve how we described the project conceptually.

Code structure and explanation

[Urban Mycelium]

```
let moldImage;
let trailCanvas;
let heatmapCanvas;
```

Global variables are declared to hold the main image, drawing layers, and arrays for agents

```
let molds = [];
let nodes = [];
```

(molds) and city nodes. These are used throughout the simulation.

```
function preload() {
  moldImage = loadImage("roma_1.png");
}
```

The preload function ensures the city map image is fully loaded before the rest of the sketch begins. This is necessary for image-based logic used later.

```
function setup() {
  moldImage.resize(...);
  createCanvas(...);
  ...
  generateRomeNodes();
}
```

Sets up the environment: resizes the image, initializes canvases and UI, and generates district data for Rome. Also prepares image pixels for simulation.

```
function draw() {
  background(255);
  updateMolds();
  updateNodes();
  renderVisuals();
}
```

This is the main loop. It updates the agents, calculates node values, and handles rendering — including trails, heatmap, and tooltips.

```
class Mold {
  constructor(x, y) { ... }
  update() { ... }
  display(pg) { ... }
}
```

Defines mold agent behavior. Agents move based on random walk and utility-driven attraction, can settle, and leave trails or heat.

```
class Node {
  constructor(...) { ... }
  updateAttributes() { ... }
  display() { ... }
}
```

Represents city districts with dynamic data. Nodes react to mold presence by adjusting income, services, and toxicity.

```
function mousePressed() {
  if (!firstClickDone) { ... }
}
```

Handles user interaction. The first click triggers mold spawning at the closest valid channel point in the image.

```
function isOnChannel(x, y) {
  let i = (x + y * width) * 4;
  return pixelBrightness < threshold;
}
```

Checks if a pixel belongs to a dark path (channel) in the image. Used to restrict mold movement to valid areas.