Rahul Myana (rm62647)
Joseph Turcios (jat5874)
Alex Yoon (ay7583)

Final Project Report

1. Background

Physarum polycephalum, or slime mold, is a unicellular organism capable of solving complex spatial and optimization problems despite lacking a nervous system. It can find the shortest paths between food sources efficiently and dynamically adapt to changes in its environment. These behaviors come from simple decentralized rules and local interactions, making it a compelling subject for studying emergent intelligence and computing inspired off of biology.

Our project is based on the paper:
Jones, Jeff (2010). *Characteristics of pattern formation and evolution in approximations of Physarum transport networks* (https://uwe-repository.worktribe.com/output/980579)

The paper describes an agent-based system where each agent represents a small part of the slime mold, guided by simple movement and sensing rules. Agents deposit chemical trails (pheromones) into a shared grid and sense these trails to determine future movement. Over time, this decentralized system shows similarities to complex, network-like structures that efficiently connect food sources.

2. Accomplishments

We implemented a dynamic, interactive slime mold simulation in **Javascript** using the **WebGPU API**. Our simulation uses compute shaders to run **250,000** agents in parallel entirely on the GPU.

1. **Agent-based simulation**

An agent is a data structure with a position and an orientation that represents a small part of the slime mold. Agents move through the environment, draw trails, and move towards trail concentrations and food sources.

Sensor positions are calculated by using the angle of the sensor and its distance to the current agent position. Each sensor position has trail values and food values.

```
let front = agent.pos + vec2f(cos(agent.angle), sin(agent.angle)) *
sensorDist;
```

```
let left = agent.pos + vec2f(cos(agent.angle - sensorAngle),
sin(agent.angle - sensorAngle)) * sensorDist;
let right = agent.pos + vec2f(cos(agent.angle + sensorAngle),
sin(agent.angle + sensorAngle)) * sensorDist;
```

Food values are given a greater weight than trail values since it is useful to the mold for survival (this weight may be modified). The agent updates its position based on whichever value is the strongest. The position is updated using the new orientation. If an agent falls of an edge it wraps around to be on the opposite edge of the environment.

```
let foodWeight = min(1.0, max(foodF, max(foodL, foodR)) * 5.0);
let fVal = mix(trailF, foodF, foodWeight);
let lVal = mix(trailL, foodL, foodWeight);
let rVal = mix(trailR, foodR, foodWeight);

// Deciding new orientation
if (lVal > fVal && lVal > rVal) {
     agent.angle -= turnSpeed;
} else if (rVal > fVal && rVal > lVal) {
     agent.angle += turnSpeed;
} else if (fVal < lVal && fVal < rVal) {
     let rand = random(agent.pos.x + agent.pos.y + f32(i));
     if (rand > 0.5) {
     agent.angle += turnSpeed;
     } else {
     agent.angle -= turnSpeed;
     }
}

// Update position
agent.pos += vec2f(cos(agent.angle), sin(agent.angle)) * moveSpeed;
```

2. **Trail diffusion and decay**

Agents leave behind trails which represent pheromones. The diffuse shader samples a 3x3 area around each pixel and computes an average:

```
var sum = 0.0;
for (var dy = -1; dy <= 1; dy++) {
     for (var dx = -1; dx <= 1; dx++) {
     var sampleCoord = coord + vec2i(dx, dy);
     sum += textureLoad(inputTex, sampleCoord, 0).x;
     }
}
```

We combine the average with the current value of the trail and the decay and diffusion parameters to simulate pheromone spreading and losing strength.

```
let decayed = center * decay;
let diffused = mix(decayed, sum / 9.0, diffuseRate);
```

3. **Food interaction and decay**

Food decay is represented by the foodDecayShader

```
var r = max(0.0, food.r - 0.002);        // fade from red
var g = min(0.15, food.g + 0.001);       // brown blending
var b = min(0.03, food.b + 0.0005);
var a = max(0.0, food.a - 0.002);        // alpha fade
```

4. **Shader Pipeline**

We have several shaders:

1. Agent Shader: calculates and updates agent position and orientation
2. Diffuse Shader: calculates diffusion and decay of the trail
3. Food Shader: calculates decay of food
4. Destroy Shaders: clear trail data based on mouse click and reposition agents that are destroyed to be at the edges of the simulation

Our rendering pipeline consists of:

1. Vertex Shader: draws the environment
2. Fragment Shader: colors the environment

5. **User Interface**

The user can adjust several parameters to control agent and trail behavior. SENSOR_ANGLE is the angular separation between sensors, SENSOR_DIST controls how far agents can sense, MOVE_SPEED and TURN_SPEED control agent movement and rotation speed, DECAY AND DIFFUSE control trail decay and diffusion, COLOR allows the user to change color, SHAPE allows the user to change the initial distribution of agents.

The system supports two interaction modes. Left-clicking creates a circular area that clears trails and repositions agents (destruction). Ctrl-clicking places a food source that attracts agents.

```
canvas.addEventListener("mousedown", (e) => {
  if (!e.ctrlKey) { // Normal click - destroy trails
      const rect = canvas.getBoundingClientRect();
      const mouseX = e.clientX - rect.left;
      const mouseY = e.clientY - rect.top;
      const simX = (mouseX * WIDTH) / canvas.width;
      const simy = (mouseY * HEIGHT) / canvas.height;
      const radius = 150;
      destroy.set([simX, simy, radius]);
      device.queue.writeBuffer(destroyBuffer, 0, destroy);
      window.destroy = true;
  }
});

if (e.ctrlKey) { // Hold Ctrl to place food
  const rect = canvas.getBoundingClientRect();
  const mouseX = e.clientX - rect.left;
  const mouseY = e.clientY - rect.top;
  const simX = (mouseX * WIDTH) / canvas.width;
  const simY = (mouseY * HEIGHT) / canvas.height;
  drawFood(simX, simY, 20);
}
```

6. Miscellaneous

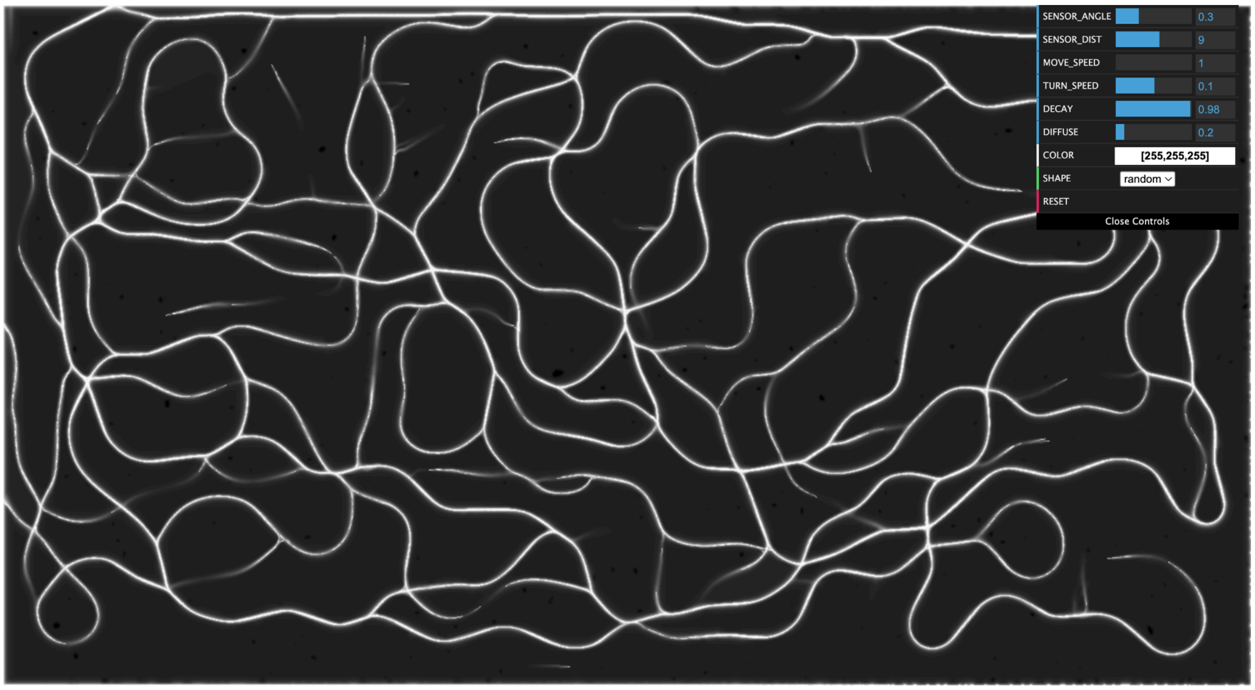
We use double buffering for both trail and food textures. That is, in each frame the system reads from the current texture while writing the next texture and swaps frames.

```
const trailTextures = [
      device.createTexture(trailTextureDesc),
      device.createTexture(trailTextureDesc),
];
const foodTextures = [
      device.createTexture({ parameters }),
      device.createTexture({ parameters }),
];
```

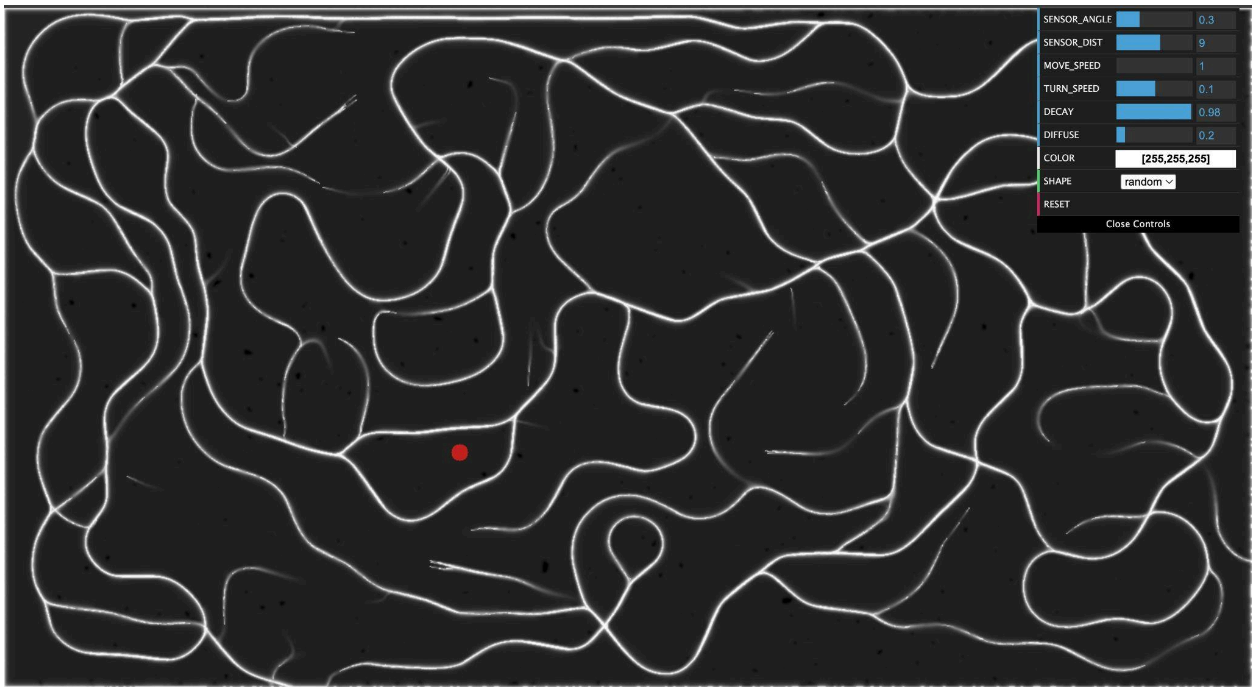
Agent updates and diffusion/food decay are handled in separate passes.

We create buffers for agent data, parameters, and destroyed agents. We use bind groups so that shaders can access data.

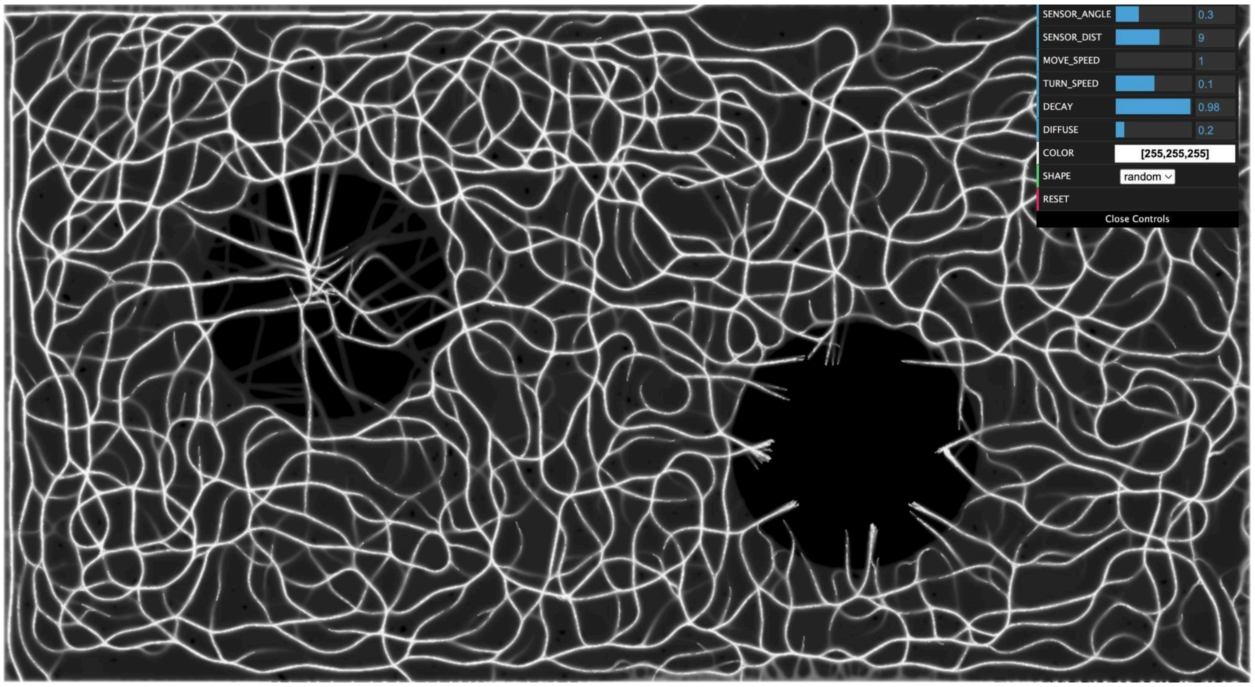
3. Artifacts Produced


SENSOR_ANGLE 0.3
SENSOR_DIST 9
MOVE_SPEED 1
TURN_SPEED 0.1
DECAY 0.98
DIFFUSE 0.2
COLOR [255,255,255]
SHAPE random
RESET

Normal Canvas

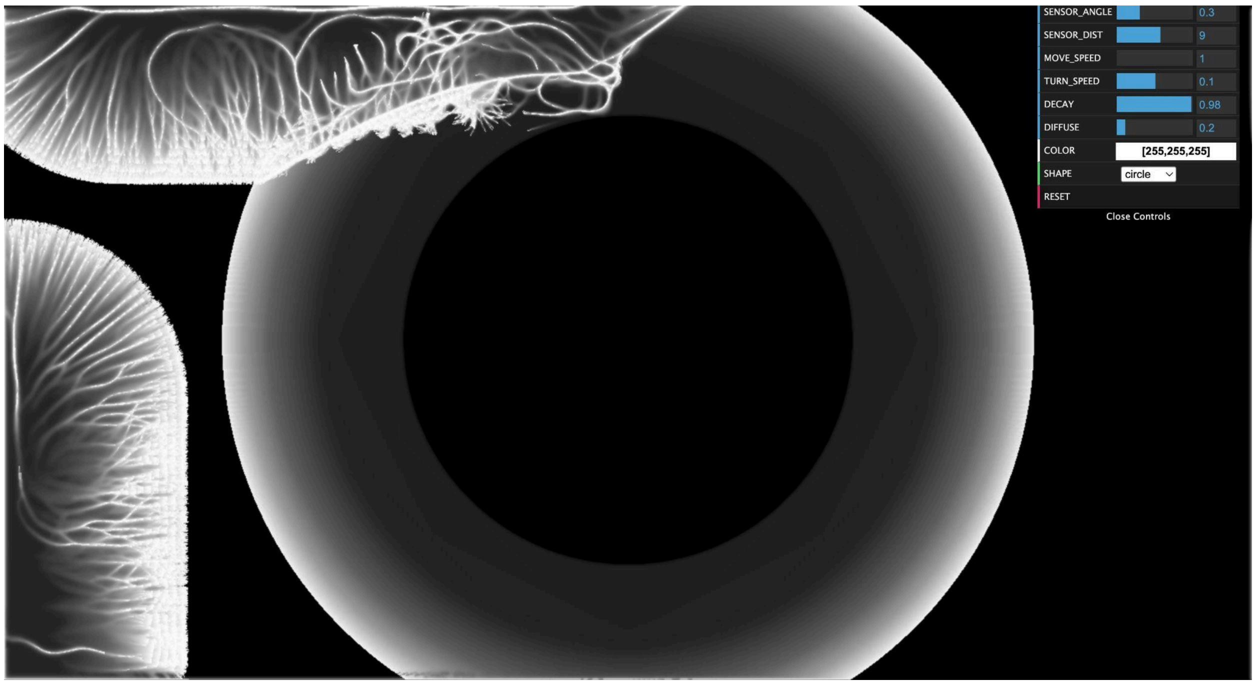

SENSOR_ANGLE 0.3
SENSOR_DIST 9
MOVE_SPEED 1
TURN_SPEED 0.1
DECAY 0.98
DIFFUSE 0.2
COLOR [255,255,255]
SHAPE random
RESET

Food Present



```
SENSOR_ANGLE   0.3
SENSOR_DIST    9
MOVE_SPEED     1
TURN_SPEED     0.1
DECAY          0.98
DIFFUSE        0.2
COLOR          [255,255,255]
SHAPE          random
RESET
Close Controls
```

Decay Effect



```
SENSOR_ANGLE   0.3
SENSOR_DIST    9
MOVE_SPEED     1
TURN_SPEED     0.1
DECAY          0.98
DIFFUSE        0.2
COLOR          [255,255,255]
SHAPE          circle
RESET
Close Controls
```

Circle Slime Mold

4. List of References

- **Main Paper:** Jones, Jeff(2010). *Characteristics of pattern formation and evolution in approximations of Physarum transport networks*.
  - (https://uwe-repository.worktribe.com/output/980579)

Link to gitlab repo:

https://gitlab.com/rahulmyana42/physarum-demo/-/tree/code-freeze?ref_type=heads