**Physarum-Inspired Pathfinding Simulation**

**Overview**

This simulation mimics the behavior of Physarum polycephalum (slime mold) for pathfinding. The organisms explore their environment, deposit chemical trails, and reinforce successful paths while weaker trails decay over time.

**Dependencies and Installation**

**Required Libraries**

pip install pygame numpy

**System Requirements**

* Python 3.7+
* At least 1GB RAM
* Display resolution of 800x600 or higher

**How to Run**

1. **Save the code** as physarum\_simulation.py
2. **Install dependencies** using the pip command above
3. **Run the simulation**:
4. python physarum\_simulation.py

**Controls**

* **R**: Reset the simulation with new random walls
* **ESC**: Exit the simulation

**How It Works**

**Core Algorithms**

**1. Agent Movement**

* **Greedy Component**: Agents prefer moves closer to the goal (Euclidean distance)
* **Trail Following**: Agents are attracted to areas with stronger pheromone trails
* **Stochastic Selection**: Top 3 candidates are chosen with weighted probability to avoid local minima

**2. Trail Dynamics**

* **Deposition**: Agents deposit trails at their current position
* **Decay**: All trails gradually weaken over time (simulating volatility)
* **Reinforcement**: Successful paths get additional reinforcement

**3. Obstacle Avoidance**

* Walls block movement completely
* Agents can get "stuck" and will reset to start after 50 failed moves

**Key Parameters (Tunable)**

# In PhysarumSimulation.\_\_init\_\_()

self.trail\_deposit = 0.1 # Trail strength deposited per step

self.trail\_decay = 0.995 # Trail decay rate (0.995 = 0.5% loss per frame)

self.num\_agents = 5 # Number of simultaneous agents

# In PhysarumAgent.calculate\_attractiveness()

distance\_weight = 0.7 # Goal-seeking behavior strength

trail\_weight = 0.3 # Trail-following behavior strength

**Visual Legend**

* **Blue**: Walls/obstacles
* **Green**: Start position
* **Red**: Goal position
* **Yellow**: Chemical trails (intensity indicates strength)
* **Black dots**: Active agents
* **White**: Empty space

**Extension Ideas for Adaptive NPC Systems**

**1. Multi-Goal Pathfinding**

# Modify agent to handle multiple goals

class MultiGoalAgent(PhysarumAgent):

def \_\_init\_\_(self, x, y, agent\_id, goals):

super().\_\_init\_\_(x, y, agent\_id)

self.goals = goals # List of goal positions

self.current\_goal\_idx = 0

self.completed\_goals = []

**2. Dynamic Environment**

# Add moving obstacles or changing terrain

class DynamicEnvironment:

def update\_walls(self, time\_step):

# Move walls, add/remove obstacles

pass

def add\_new\_goals(self, positions):

# Dynamically add new objectives

pass

**3. Agent Specialization**

# Different agent types with unique behaviors

class ScoutAgent(PhysarumAgent):

"""Fast, low trail deposit - explores new areas"""

def \_\_init\_\_(self, x, y, agent\_id):

super().\_\_init\_\_(x, y, agent\_id)

self.speed\_multiplier = 1.5

self.trail\_deposit\_rate = 0.05

class WorkerAgent(PhysarumAgent):

"""Slow, high trail deposit - reinforces good paths"""

def \_\_init\_\_(self, x, y, agent\_id):

super().\_\_init\_\_(x, y, agent\_id)

self.speed\_multiplier = 0.8

self.trail\_deposit\_rate = 0.2

**4. Hierarchical Pathfinding**

# Combine with A\* for long-distance planning

class HierarchicalAgent(PhysarumAgent):

def \_\_init\_\_(self, x, y, agent\_id):

super().\_\_init\_\_(x, y, agent\_id)

self.waypoints = [] # A\* generated waypoints

self.current\_waypoint = 0

def plan\_route(self, start, goal):

# Use A\* to generate waypoints

# Use Physarum behavior between waypoints

pass

**5. Collective Intelligence**

# Agents share information and coordinate

class CollectiveIntelligence:

def \_\_init\_\_(self):

self.global\_trail\_map = {}

self.discovered\_paths = []

self.danger\_zones = []

def update\_global\_knowledge(self, agent\_experiences):

# Aggregate learning from all agents

pass

def broadcast\_information(self, agents):

# Share discovered paths and dangers

pass

**6. Game Integration Examples**

**Real-Time Strategy (RTS)**

* **Unit Pathfinding**: Multiple units naturally form efficient paths
* **Resource Gathering**: NPCs find optimal routes to resources
* **Territory Control**: Establish patrol routes and defensive positions

**Role-Playing Game (RPG)**

* **NPC Wandering**: Natural, organic movement patterns
* **Quest Pathfinding**: Dynamic route finding for objectives
* **Group Behavior**: Party members coordinate movement

**Survival Games**

* **Resource Discovery**: NPCs explore and map resources
* **Danger Avoidance**: Learn to avoid dangerous areas
* **Base Building**: Find optimal locations for structures

**Performance Optimization**

**For Large Grids**

# Use spatial partitioning for large environments

class SpatialGrid:

def \_\_init\_\_(self, width, height, cell\_size):

self.sectors = {}

self.cell\_size = cell\_size

def get\_nearby\_agents(self, x, y, radius):

# Only check nearby sectors

pass

**For Many Agents**

# Implement agent pooling and LOD

class AgentManager:

def \_\_init\_\_(self):

self.active\_agents = []

self.inactive\_agents = []

self.max\_active = 100

def update\_agents(self, player\_position):

# Only update agents near player

pass

**Research Applications**

This simulation demonstrates several AI concepts:

* **Swarm Intelligence**: Emergent behavior from simple rules
* **Reinforcement Learning**: Trail reinforcement mimics reward learning
* **Multi-Agent Systems**: Coordination without central control
* **Bio-Inspired Computing**: Natural algorithms for artificial systems

The code is designed to be modular and extensible, making it perfect for experimenting with different bio-inspired algorithms and game AI systems.