Modeling Social Interaction Dynamics in Agent Simulations

Abstract

In this project, I explored how simple social interaction rules among agents can produce complex emergent behaviors in a two-dimensional continuous environment. Using object-oriented programming and a custom linked list structure, I implemented an agent-based simulation in which agents move based on their nearby neighbors within a defined interaction radius. My experiments showed that increasing the number of agents and expanding their interaction radius both significantly increase the time required for the system to stabilize, with larger radii often causing the simulation to reach its iteration limit. These findings highlight the significant impact of small changes in parameters on substantial shifts in system dynamics and equilibrium behavior.

Results

- **Experiment 1**

In this experiment, I explored how the number of agents affects the time it takes for a social system to stabilize. Each simulation was conducted on a 500×500 landscape with a constant interaction radius of 25 units. The number of iterations before all agents stopped moving was recorded as the measure of stabilization time.

I hypothesized that increasing the number of agents would extend the time required for the system to stabilize, as more agents would mean more potential interactions per update cycle, increasing system complexity.

Table 1. Number of Agents vs. Iterations to Stabilization

Number of Agents	Iterations
50	196
100	436
150	599
200	1348
250	1449

As the number of agents increases from 50 to 250, the total number of iterations required for the simulation to stabilize rises from 196 to 1,449. This indicates that denser populations lead to more frequent interactions, which slow the system's ability to reach equilibrium. The original data is saved as a text file in the project.

The results supported this hypothesis: as the number of agents increased, stabilization time grew steadily. The sharp increase between 150 and 200 agents suggests that there may be a density threshold beyond which local movement patterns start to overlap, causing the simulation to require

substantially more iterations to settle. In real-world terms, this implies that in systems with many interacting individuals, higher population densities tend to delay the formation of stable configurations or consensus.

- **Experiment 2**

In the second experiment, I kept the number of agents constant at 150 while varying their interaction radius. The interaction radius determines how far an agent can sense and respond to others nearby. The same 500×500 landscape was used for all trials, and the total number of iterations until agents stopped moving was again recorded.

I expected that increasing the interaction radius would lengthen the stabilization time, since each agent would need to account for more neighboring agents when deciding whether to move.

Table 2. Interaction Radius vs. Iterations to Stabilization

Radius	Iterations
10	517
20	813
30	880
40	1781
50	5000

When the interaction radius increases from 10 to 50, the number of iterations required for stabilization rises from 517 to the 5,000-iteration cap. This means that expanding the radius causes agents to respond to a wider neighborhood, dramatically slowing convergence. The original data is saved as a text file in the project.

This hypothesis was confirmed. For small radii, agents stabilized relatively quickly, but as the radius increased beyond 30, the number of iterations rose dramatically, reaching the 5,000 iteration cap at a radius of 50. This suggests that systems with broader ranges of influence, where agents are aware of or react to many others, tend to exhibit slower and more chaotic convergence. In real-world analogs, this reflects how social systems with long-range interactions may oscillate indefinitely due to widespread dependencies among individuals.

Acknowledgements

I would like to thank Professor Harper and Professor Al Madi for providing the project template and guidance site, as well as the TAs for guiding me through the debugging process. Thanks to Oracle and W3Schools Java Tutorial for the detailed documentation.

Extensions:

- **What did I do and why:**

I designed a new WanderingAgent that explores the landscape with persistent, semi-random direction.

The algorithm of the WanderingAgent is:

1. Check if it's time to change direction (based on step counter)

2. Move in the current direction at constant speed

3. Handle boundary collisions by reflecting direction

4. Reset direction counter on boundary hits

5. Update position within landscape bounds

Unlike Social/AntiSocial agents, it doesn't care about other agents at all. It only explores following the logic above. The goal was to study exploration/coverage behavior vs. socially driven clustering and to see how a neutral explorer changes population dynamics when mixed into the system. I wanted one agent (the WanderingAgent) that explores the map on purpose instead of reacting to people nearby. That way I can compare social crowd behavior vs pure exploration. It also helps break up clumps when mixed with the social ones. To explore this new agent simulation, I added a new method getAgents() in the Landscape class, which returns a LinkedList of agents.

- **Outcome**

WanderingAgents spread out fast and cover a lot of space. When I mix them with the other types, they reduce big clusters and make the whole thing look more stirred. Adding a few wanderers makes the system less stuck and more evenly spread.

Table 3. Impact of Agent Mix on Convergence in Different Iterations

Social	Anti-Social	Wandering	Iterations	Notes
30	0	0	188	Pure social clustering
25	0	5	151	Mixed with exploration
20	0	10	332	Mixed with exploration
15	0	15	116	Mixed with exploration
0	30	0	1	Pure anti-social spreading
0	25	5	61	Mixed with exploration
15	15	0	96	Pure social

				clustering
10	10	10	114	Mixed with exploration

As the mix shifts from purely social (slower, clustering) to purely anti-social (instant dispersion), convergence speeds up. Adding WanderingAgents often reduces iterations by breaking clusters (e.g., 30→25 social: 188→151; 15→15 social: 188→116), but can increase iterations when exploration keeps the system active (20/0/10 → 332). Wanderers trade speed for better coverage and less mega-clumping.

In mixed populations, adding WanderingAgents (green) noticeably changes convergence behavior compared to pure Social (blue) and Anti-Social (red) setups. Pure Social (30/0/0) takes longer to settle (188 iters) while pure Anti-Social (0/30/0) disperses almost instantly (1 iter). Introducing wanderers to social groups generally accelerates stabilization—e.g., 25/0/5 drops to 151 and 15/0/15 to 116—by breaking up sticky clusters, though it's not strictly monotonic: 20/0/10 spikes to 332 when persistent exploration keeps re-energizing the system. Adding wanderers to anti-socials slows the otherwise instant settle (0/25/5 → 61) because explorers keep moving rather than freezing once spaced out. Balanced mixes fall in between (15/15/0 at 96; 10/10/10 at 114) and visually show better coverage with fewer mega-clumps.

- **How to run**

Compile all the java classes under ext folder and run the WanderingAgentSimulation class. Uncomment/Comment the experiment code block if you want to get new data text files.