Agent Simulation Using LinkedLists

Abstract

In this project, we designed and implemented an agent-based simulation using linked lists as a fundamental data structure. The core purpose of this project was to explore the behavior of the agents and to implement our LinkedLists, particularly how they interact with their surroundings and with other agents. We created two distinct types of agents: SocialAgents, which move towards other agents, and AntiSocialAgents, which attempt to move away from other agents. By running simulations with these agents, we aimed to understand how their behaviors and interactions affect the overall dynamics of the landscape. In our simulations we saw that the social agents would tend to form small groupings with variance distance between each other dependent on their radius. Whereas the antisocial agents would spread out relatively evenly forming exclusive halos around the social agent clusters.

Results

AgentSimulation

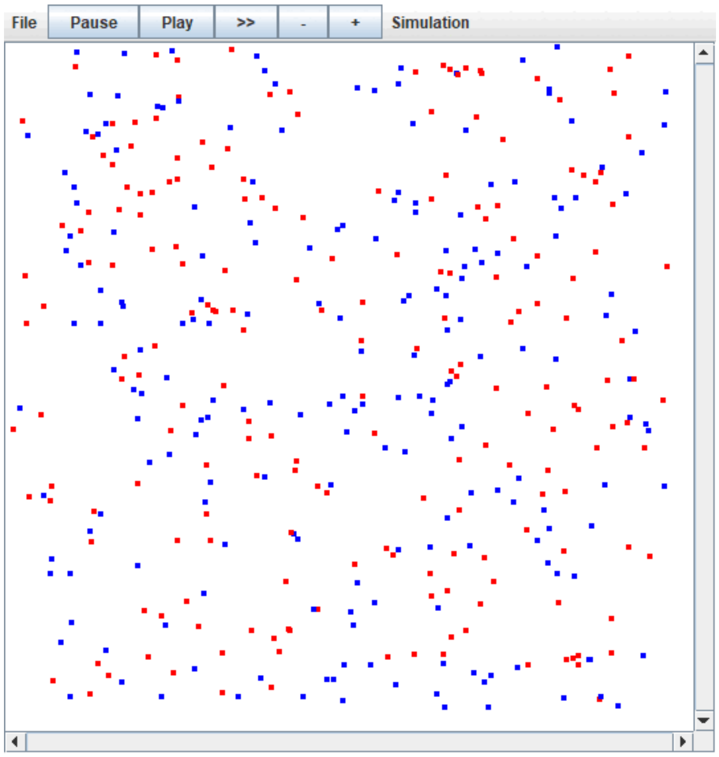
The `AgentSimulation` class implements a basic agent simulation as required by the project. It utilizes methods from `LandscapeFrame` to create a cleaner and more straightforward implementation. The simulation consists of a 2D landscape populated by 100 SocialAgents and 100 AntiSocialAgents. The agents are randomly placed within the landscape, and the simulation starts, allowing them to interact.

The key elements of the `AgentSimulation` class are:

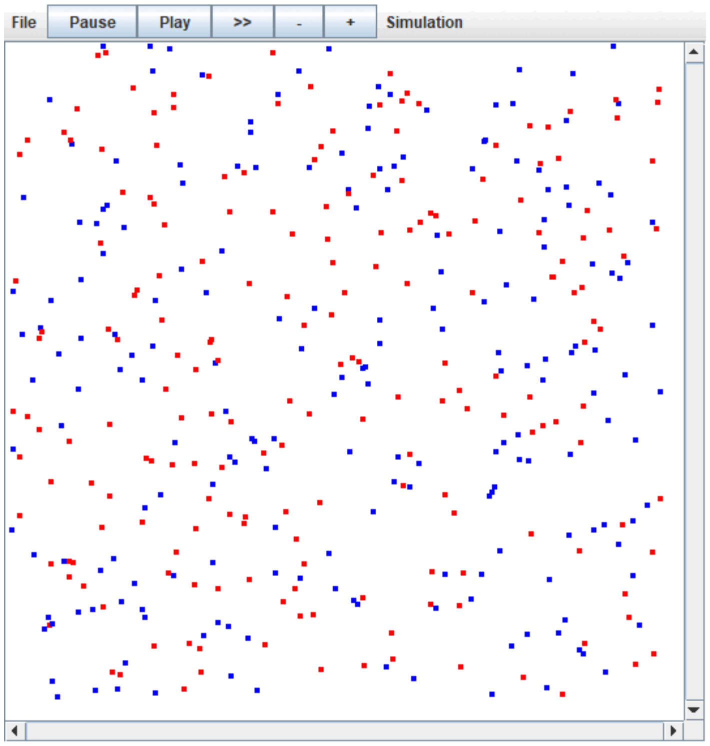
- Initialization of a 2D landscape named `scape` with a size of 500x500.
- Creation of 100 SocialAgents and 100 AntiSocialAgents, randomly positioned within the landscape.
- Initialization of a game frame (`LandscapeFrame`) for visualization.
- Commencement of the simulation.
- Saving images of the simulation frames at regular intervals.
- The simulation continues until all agents have stopped moving.

The `AgentSimulation` class advances the simulation by simulating a button press of the advance button defined in our `LandscapeFrame`.

This is the animation generated with Social and AntiSocial radii of 15 and 11 respectively:



This is the animation generated with Social and AntiSocial radii of 30 and 22 respectively:



As is visible from the figures the simulations using lower radii resulted in more tightly grouped social clusters and less sparsely scattered AntiSocialAgents. This quite closely aligns with expectations as this is to be expected with higher and lower thresholds for neighbor nearness.

An interesting corollary of this is the fact that when radii for social agents were small the simulation took longer, when radii for anti-social agents were small the simulation went quicker.

Similarly, higher density of agents would result in shorter or longer runtimes depending on the social/antisocial agent ratio.

All of these factors are a result of this simulation being effectively an optimization problem for space distribution between two different actor types.

Finally, there is a functional upper limit to the density of agents relative to radius wherein the simulation will never be able to end as the antisocial agents won't be able to fulfil all their conditions. On the other hand, Social Agents have no upper limit on density and will in fact resolve quicker as density goes up. However, they have a strong lower bound on agent count that will allow for a resolution to the game. All of which are easy to test via the GUI in the `AdjustableSimulation` class.

Extensions

1. **Sector Map for Efficient Computation:**
    - **Implementation**: In this extension, I introduced a sector map to partition the landscape into smaller regions or sectors. Each agent is placed in a specific sector based on its coordinates. This approach allows for more efficient neighbor searching by narrowing down the search space to nearby sectors, reducing the number of agents to consider. It is effectively an implementation of a hashSet. Consider $\mathbb{Z}_n \times \mathbb{Z}_m$ via the cartesian product. This will represent our sector map for a $n(10) \times m(10)$ pixel landscape. Now consider, via an ultra-power construction, if we were to insert for each integer pair $(x, y) \in \mathbb{Z}_n \times \mathbb{Z}_m$, a space $\mathbb{Z}_{10} \times \mathbb{Z}_{10}$. Now if we were again, via an ultra-power construction, to insert for each integer pair $(x', y') \in \mathbb{Z}_{10} \times \mathbb{Z}_{10}$, the interval $([0,1) \times [0,1)) \in \mathbb{R}^2$, we would have a construction equivalent to some segment of the reals. This is the principle under which this method functions. We map each agent to a sector corresponding to their location and use this to limit the number of agents necessary to poll to find neighbors.
    - **Effect**: The implementation of a sector map significantly improves the efficiency of neighbor searches in your agent-based simulation. Without a sector map, searching for neighboring agents would require iterating through the entire list of agents. With the sector map, you can limit the search to agents within the same or adjacent sectors, resulting in faster and more optimized simulations, especially as the number of agents increases. This results in stable simulations with upwards of 20,000 agents in a 500×500 landscape. The upper limit being a function of

agent density makes it such that as long as density stays below some threshold considerably higher numbers are possible.

2. **Graphical User Interface (GUI) with Optional Settings:**
    - **Implementation**: I added a graphical user interface (GUI) to my simulation, providing users with a more interactive and user-friendly experience. This GUI includes various optional settings that allow users to customize the simulation's behavior. Some of the optional settings could include parameters like the number of agents, the size of the landscape, the radius to check for neighbors (individually controllable), the sleep time between updates, and more.
    - **Effect**: The GUI enhances the usability of the simulation. Users can interact with the simulation, modify parameters, and observe how changes affect the agents' behavior and interactions. This feature makes the simulation more accessible and provides a way to experiment with different scenarios without altering the code. It also facilitates experimentation and analysis by enabling users to explore the impact of different settings on the agents' behaviors and patterns.

Reflection

In this project, we used linked lists as the primary data structure to manage the agents' positions and interactions. Linked lists were chosen over alternatives like arrays or ArrayLists due to their efficiency in handling dynamic collections of data. Linked lists allow for easy insertions and removals of elements, which are essential for managing agents' positions and movements in a 2D landscape. They also provide more flexibility in memory allocation and do not require a fixed array size, making them suitable for this simulation. The LinkedList structure also allowed us to easily manipulate and organize the agents, something which was essential in the implementation of my hashSet type sector map.

References/Acknowledgements

I worked on this project individually; all code is my own. Most JavaDoc is handwritten, some is trimmed down versions of migrated Javadoc (ie in implementing LinkedList I copied in the declarations and Javadoc of things I needed to make/implement and then wrote up the code myself before trimming down and rewriting JavaDoc.) Also, in the linked list implementation I looked at Javas own implementation to make sure I implemented some methods I might need as I wanted to make my code able to function on either implementation. Specifically, I originally wrote it for Javas's implementation and coded in my implementation afterwards. However, all code is mine and written by me at the end of the day.