Agent-Based Simulation on a 2D Landscape with Java

Abstract:

The context of the real-world problem I explored in this project was agent-based simulation with self-moving agents and interactions. I explored behaviors with levels of sociability. To better understand the problem, I used SocialAgents and AntiSocialAgents, where SocialAgents would move towards other agents and AntiSocialAgents would move away. The core CS concepts I used in this project were LinkedLists, lists, inheritance, and I simulated interactions with these. My key findings from this project involved computational modeling and how to work with that when the agents ended up being unpredictable and not following the programmed framework.

Results:

The experiments I ran were testing the outcomes with a different number of agents and different radius sizes. The metrics I used to report to understand the outcomes of the experiments were the number of iterations it took for the simulation to stop running, and then if they timed out or not. Only one experiment timed out and that was when there were 250 agents and 5000+ iterations, so the simulation timed out.

Element 1:

Agents	Iterations	Timed out
50	15	no
100	7	no
150	13	no
200	11	no
250	5000	yes

Trends: If there are more agents, the interactions occurring happen much more frequently and the system stabilizes quicker, so the iterations lessen. However, when there are 250 agents, the system becomes overpowered and is unable to stabilize anymore.

Element 2:

Radius	Iterations	Timed out
10	15	no
20	10	no
30	8	no

40	12	no
50	14	no

The experiment here was to test the impact of the radius size on the number of iterations in the simulation. The trend here seems like a larger radius is more optimal for the simulation to stop. It also looks like a radius of 30 had the best results as both a radius smaller and larger need more iterations to end.

The outcome of the experiments showed that it seems like a greater number of agents caused the simulation to have a lesser number of iterations and finish running faster, maybe because of the increased number of interactions the agents were having. However, this is only true until 200 agents because at 250 agents, the simulation timed out. Another outcome showed me that a larger radius didn't necessarily cause less iterations in the simulation. Up until a radius of 30, it was better to have a larger radius but after that it wasn't as effective.

Extensions:

I didn't do any extensions.

Reflection:

1. Public E removeLast() is the method which needs a DoublyLinkedList in order to achieve a constant time because the doubly linked list allows us to access the next and previous node so we can update the list using the previous node without going through the entire list again. It now has a constant runtime because as stated above the next and previous node are directly accessible without going through the entire list again.
2. In LinkedList, each node is probably around 12 bytes so the list will be 12n. For DoublyLinkedList, each node is probably 20 bytes so the list will be 20n. Both LinkedLists and ArrayLists have pros and cons relative to each other. LinkedLists don't require a set size of the array while ArrayLists do. Alternatively, ArrayLists are better for memory because they take up less than LinkedLists do. It is also easier to access specific elements in ArrayLists than in LinkedLists because in LinkedLists you have to go through the whole list rather than directly access.

Acknowledgements:

I went to TA hours on Wednesday and Sunday for help on this project. I also worked with Kamalani for parts of the project.