Problem Statement

This project will simulate disease spread between agents running on individual threads.

Agents have the following possible states:

- Vulnerable, Sick, Immune, Dead
- Vulnerable -> Sick -> (Immune | Dead)

Initially, most agents start out vulnerable, with a small subset being sick. Sick agents will notify their neighbors within exposure distance that they have been exposed.
Vulnerable agents become sick after the incubation period, and after another period of being sick, the agent either recovers (becomes immune) or dies.

The configuration file is a plain text file w/configuration options specified one per line, in any order
Things specified in config file:

- Dimension of board
- Exposure distance constant
- Incubation time constant
- Sickness time constant
- Recovery probability constant
- Grid Type
    - Grid - agents spaced evenly, filling up an n*m grid
    - Random - agents placed randomly on the board space
    - RandomGrid - agents placed randomly on n*m grid, but may not fill it fully
- # of initially sick agents

The GUI will draw the agents on the screen w/various colors to indicate their status.
Each agent must run on its own thread, and communicate via its neighbors.

Implementation

At the start of the program, we will read and parse the configuration file, returning a ConfigInfo object that stores all of the fields required (w/ default values as needed), passing this object to most likely the AgentManager.

AgentInitializer - generates however many agents the simulation requires, assigning to each a position, state, and possibly a list of neighbors.

After the agents are initialized, they will be started on their own threads and be able to run.

Agent Implementation:

The agents will communicate through messages sent through thread-safe blocking queues. The main function of the agent will be to read messages from its own queue to process and to send messages to the queues of its neighbors. The agent will have a special thread inside of it (might not be thread, but just functionality) that sleeps most of the time, but wakes up to check the state of the agent and do various logic according to the state it sees. For example

- If StateThread checks state and state is vulnerable/immune/dead, do nothing (may just shut off the thread once agent is dead)
- If awoken to find state is “exposed”, start the IncubationThread to send sickness message into queue when it wakes up.
- If awoken to find “sick” state - send message to neighbors of current agent to become exposed

This should be all that an agent is required to do, obviously glossing over some of the details as to how to parse and handle every message that it reads, and the exact standard of communication.

AgentManager Implementation

The AgentManager class will hold some sort of data structure of all of the agents. This will be what the GUI communicates with, likely asking the AgentManager for some sort of collection representing the things it needs to draw (i.e. a collection of colors and points).

It may also be convenient for this class to be able to compute neighbors dynamically for any agent that asks. This would most likely only be really necessary if movement is involved, and would likely require a non-naive solution to neighbor finding.

All in all, this should be all that is necessary for the project, at least from a high-level point of view. It is unlikely that implementing the additional required features would change this proposal too much.

Rough Design Diagram