Pandemic Simulation Engine Proposal

Features and Architecture

Features

Overview

The Pandemic Simulation Engine will be a code library designed to take scientific data models and generate game state for simulation based games. The scientific models will be the primary input to the simulation engine, and will be used to drive state changes at each simulation update. The results of the simulation will be recorded on a global blackboard that can be queried by a game running the simulation. The long-term objective of this engine is to be extensible and scalable enough that it can be easily expanded to run multi-faceted simulations based on whatever scientific models are provided. The engine will be designed as a cross-platform C++ library that will be data driven and compatible with the infrastructure of a typical game.

Cross Platform C++ Library

To maximize compatibility with any future game projects that will utilize this engine, the simulation engine will compile down to a platform-agnostic code library.

Model Driven Simulation

This simulation engine will be driven first and foremost by scientific data models and their accompanying inputs. To import these models into the engine, the user can create data files to define a SimProcessor object within the simulation engine.
When a user defines a SimProcessor object, they are defining simulation work to be done within the engine given the simulation's global state, the current states of agents, and existing graph structures. In essence, these objects will specify how the simulation updates itself, defining the state variables that will be stored in simulation agents, how those agent variables are updated, the state variables that will be output to the global blackboard, and how those global variables will be aggregated. In this way, the simulation can be driven by pre-validated scientific models.

Agent-Based Simulation with Graph Structures

Complex model based simulations will be made possible by utilizing an agent-based simulation approach that uses graphs to represent complex real-world relationships and networks. Like SimProcessor objects, SimGraph definitions will be specified with data files and each represent a single SimGraph object within the simulation. When a user defines a SimGraph object, they are defining a structure on which simulation work will be performed by one or more SimProcessors. In graph data files, the user will specify how graphs are generated at startup and potentially restructured during runtime in response to changes in the global simulation blackboard. These data files will also specify which SimProcessors will be attached to the graph, and the dependency order of the SimProcessors, so that requisite computation work can be performed and outputted to the global blackboard before running a SimProcessor that requires it. The SimProcessors attached to a given graph will determine what state the SimAgents of that SimGraph will store.

Scaled and Extendable Simulation

By allowing the specification of models in SimProcessor and SimGraph definition files, and letting users load any number of SimGraphs with any number of SimProcessors attached to them into the simulation, the simulation engine lends itself to being extended and scaled to represent any number of desired scientific models in its simulations.

Simulation Management

There may be cases where it is desirable to manage the simulation state so that very specific scenarios may be simulated by the engine. In that case, the engine will provide methods to control and edit global blackboard data. Since global blackboard data is used to drive a simulation's update, editing global blackboard data is an easy way to initially contrive scenarios to see how they play out.

Queryable Simulation State

After each simulation update, when the simulation's global state has been updated to reflect the state of its SimGraphs, games can query the simulation state to observe how key variables have changed over the course of the simulation. These queries will allow the user to ask for any simulated variable by name. The SimProcessor data files will specify exactly which variables are simulated and what those variables are called. By querying on these variable names, users have full access to any data that was simulated by the engine.

Technical Design

Components:

SimProcessor Definitions

For this simulation, it will be very important to engineer a streamlined way to generically import scientific data models into the simulation engine. All input models must eventually be represented as SimProcessors and SimGraphs within the simulation engine. SimProcessor data files will specify exactly how computational work is performed on SimAgents in a SimGraph and how that work will be aggregated and recorded to the global blackboard.

This information will include:

- SimProcessor String Identifier
- What state variables will be appended onto SimAgents within any SimGraph this SimProcessor is attached to.
- How to compute and update each SimAgent's appended variables.
    - Specifies all required computation inputs, such as the name of a required blackboard variable or SimAgent variable, or the value of a SimAgent's edges in a SimGraph.
    - Specifies the methodology for computing and updating the SimAgent's appended state variables.
- What global variables will be aggregated and appended onto the global blackboard.
    - Specify how these global variables are calculated using the aggregated state of SimAgent variables within an owning SimGraph.

SimProcessor Objects

At startup, all defined SimGraphs will be created along with all the SimProcessor objects attached to them. When created, SimProcessors are stored in a queue on its owning SimGraph. Then, SimProcessors will append all the state variables that it plans to operate on onto all the SimAgents in its owning SimGraph. SimProcessors will also append state variables to the global blackboard where it plans to report its final aggregated computations to. SimProcessor objects will use the command pattern and perform two major roles during simulation update.

- Update owning SimGraphs by taking each SimGraph SimAgent as an input and updating the SimAgent's state variables that are related to said SimProcessor.
- After Updating its owning graph, aggregate agent state variables and report the final results to the global blackboard.

SimGraph Definitions

Like SimProcessors, SimGraphs in the simulation are specified with definition files, with each SimGraph data file corresponding to a single graph that will be generated by the simulation. SimGraph definitions are concerned with encoding the structure of graphs, as well as the SimProcessors that will be used to update them.

This information will include:

- Graph String Identifier
- How to generate the graph (How many agents, how to connect edges)
- How to regenerate the graph in response to global blackboard changes.
- Which SimProcessors are attached to this SimGraph
    - Specify an dependency order of SimProcessor work to be performed, if one SimProcessor requires that another finish its graph agent updates and global blackboard update before another.

SimGraphs and SimAgents

SimGraphs encode structures within which the SimAgent state will be simulated, as well as the SimProcessors that will update and define the graph. The meaning of SimGraphs nodes, edges, and their respective states will all be contextually defined by a SimGraph's definition file. SimAgents are the nodes of a SimGraph, and can represent anything from people, to hospitals, to electrical grid transformers, or even entire populations. SimGraph structures can be as simple as a single node if the SimProcessor does not require a graph structure to simulate its state. SimAgents will have a unique state and use whatever SimProcessors are attached to the SimGraph to update themselves. This update can use any available information that is required to make these updates, including existing agent state, graph structure, edge values, and global blackboard data. All SimGraphs in the simulation will be stored in a queue on the simulation's global state, and will be updated at each update in the simulation. Importantly, SimGraphs will store the dependencies of any SimProcessors it owns, and ensure that SimProcessors do their computations in the proper order in cases where one SimProcessor relies on the outputs of another.

Global Blackboard

The Global Blackboard is a part of the simulation engine's global state, and is the primary location for storing and querying the overall state of the simulation. The Global Blackboard serves as the medium communication between the simulation and game client. It also allows for data to be shared across different SimProcessors and SimGraphs. Global blackboard state variables will be defined and appended by all the SimProcessors across all SimGraphs in the simulation, and these variables will serve as the inputs and outputs of various SimProcessor computations during simulation update.

Simulation Update

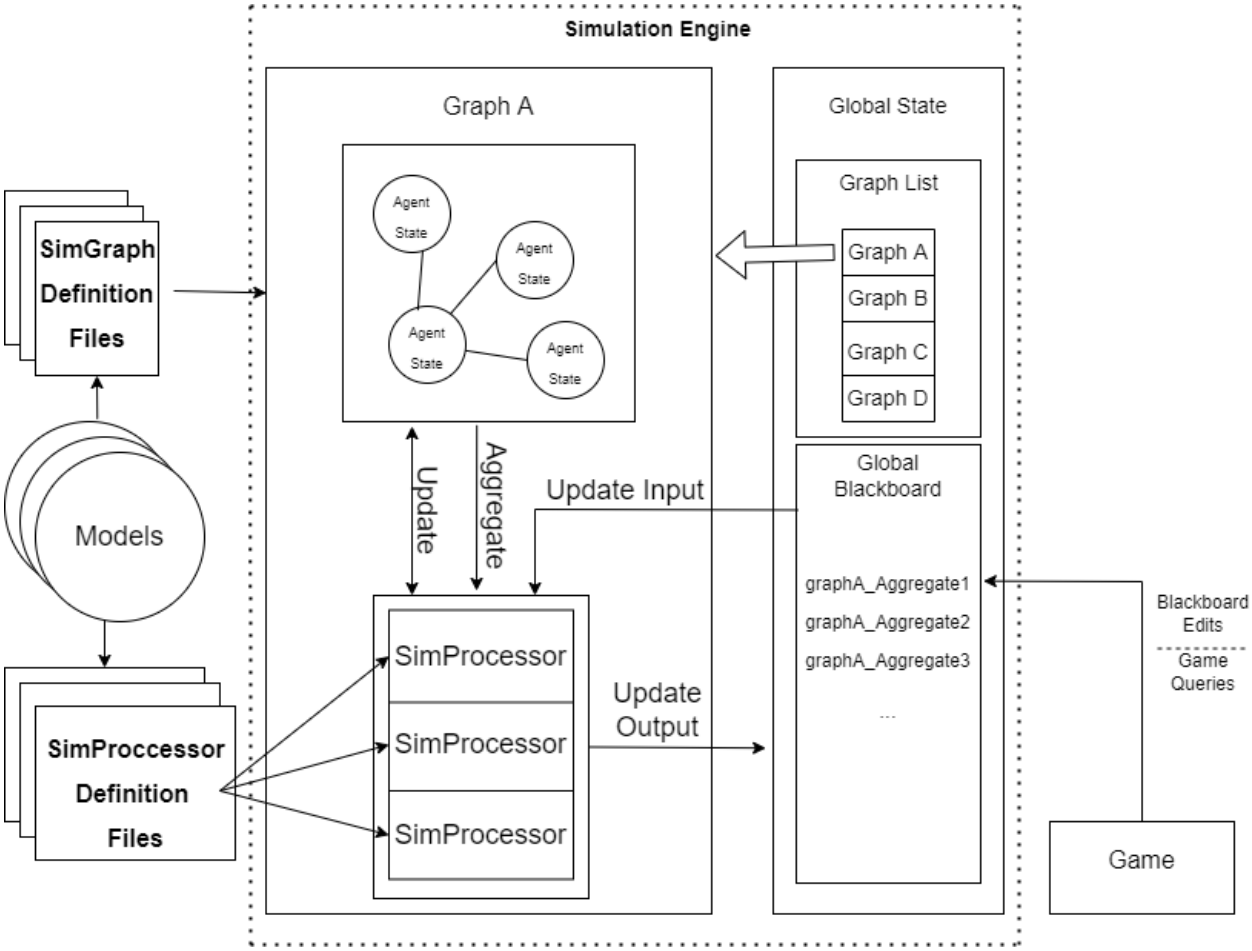
The simulation update step occurs as follows:

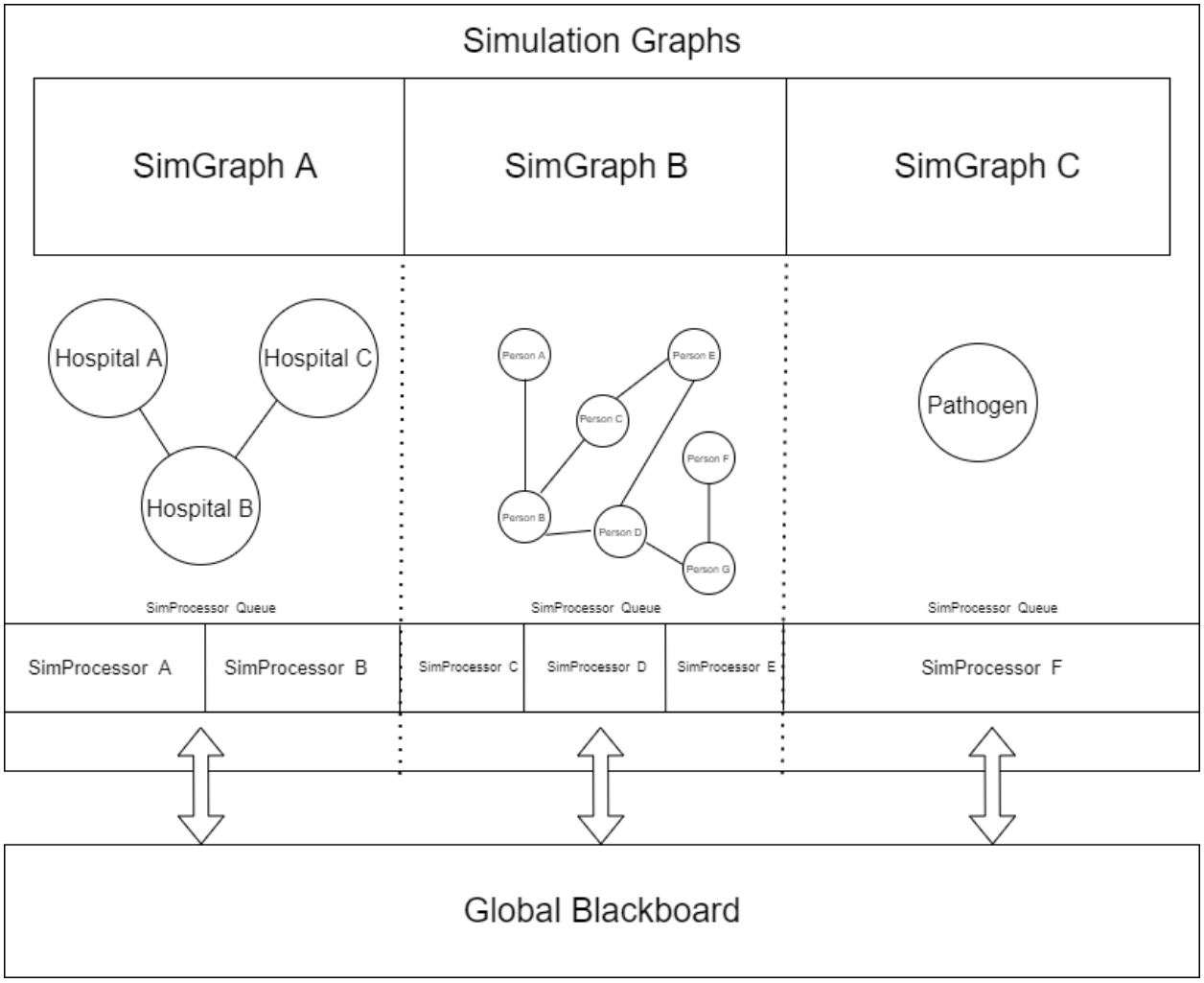
-UpdateGraphStructures: All SimGraph structures are revalidated and updated if necessary. SimGraph structures may be dependent on variables that exist in the global blackboard. If those values change, graph structures may need to be changed as well. Depending on how graph generation and structure is specified in Graph Definition files, SimGraphs with blackboard dependent structures will need to be updated to reflect blackboard changes before any SimAgent updates take place..

-UpdateGraphAgents: Next, the simulation loops over all SimGraphs and updates each of its SimAgent one by one. This step will ensure that all SimProcessor dependencies are honored, so that no SimProcessor does its computation before a SimProcessor it depends on. SimGraphs update their agents by passing them into each of its attached SimProcessors one by one. SimProcessors will then update each SimAgent according to the SimProcessor's definition file.

-AggregateAndUpdateBlackboard: Once all SimAgents are updated, all that remains is to record the results to the global blackboard. The simulation will loop over every SimAgent in every SimGraph again, but this time will use each SimProcessor to collect the computed data and aggregate it according to the methodology specified in the SimProcessor's definition file. When a SimProcessor finishes aggregation, the results will be recorded to the global blackboard.

Architecture Diagrams:




Simulation Graphs
SimGraph A
SimGraph B
SimGraph C
Hospital A
Hospital C
Hospital B
Person A
Person E
Person C
Person F
Person B
Person D
Person G
Pathogen
SimProcessor Queue
SimProcessor Queue
SimProcessor Queue
SimProcessor A
SimProcessor B
SimProcessor C
SimProcessor D
SimProcessor E
SimProcessor F
Global Blackboard