Aedan Wells
Summer Bronson
Wesley Camphouse
Brenton Candelaria
Jonathan Carsten
Samuel Nix

vCAVS: Viral Cellular Automata Visualization Simulator

Interim Report

1. Project Overview

In the last few years, COVID19 has affected the lives of people across the world. Our goal with this project is to provide visualization for people to further understand the consequences of the deadly pandemic. We are achieving this through a cellular automata model that is extended to be more realistic and representative of the real world. This automata is guided by user input involving mask mandates, vaccination rate, age range, population date and more factors. We have completed a working UI and grid simulation prototype representing people who catch COVID19 and either recover or die. In the near future, we plan on adding buildings to replicate real life scenarios of entering a building and being closer to people as well as have an after simulation information debrief. This debrief will show numerically and graphically the statistics shown in the simulation.

1.1 Scope and Objectives

The objectives we hope to satisfy by this project are:

- Provide a robust UI where the user can"
  - Configure settings related to infection rate, virality, population, map layout, and simulation environment
  - Generate custom buildings with different mandates and sizes and viruses with differing infectivity and severity to be simulated
- Simulate the spread of COVID19 through a populated environment with a cellular automata model, where spread is influenced by the various factors such as vaccination rate, mask usage, social distancing.
- Present an elegant grid visualization of the above simulation where buildings, people movement, peoples disease status, and ongoing statistics are presented in an aesthetic manner.
- Implement Buildings that run on separate threads to do computations within the simulation
- Output data for the results of each simulation and display it to the user in a readable and graphical format.

In terms of scope, we will complete the above objectives but are limited to time and resources. We do not have a large enough server to have the simulation scale to hundreds of thousands to millions of people nor thousands of buildings running on their own threads. Time does not allow for being able to click on individual buildings, render a simulation, and see the number of people moving within. Time can possibly reduce the amount of objectives we can complete, however, we believe we can complete them in the given time.

1.2 Supplementary Requirements

Our supplementary requirements encompass aesthetics, extensibility, reusability, reliability, and documentation. Our product is not just a mathematical simulator, it needs to be able to visualize the movement of people and the buildings they move around. To this end, we need a user interface that has visual appeal so users can enjoy using the product. Aesthetically, we want the people and their colors to be appealing against the background, interesting looking buildings, well presented data and a settings UI that feels good to use.

Our product strives to be extensible in multiple ways. At its core, it is a cellular automata that is mathematically supported by disease metrics particularly about COVID19. In this way, the severity and transmissibility can be modified to simulate any disease a user specifies. As well, buildings can be added with different attributes such as capacity, disease rules, and opening times. New factors that can affect disease or behavior can be added such as allergies, hobbies, favorite places or even favorite people can be extended. The software can even be extended to the outer bounds of size: millions of people in whole graphical regions spreading COVID19. The software is at its core extendable which lent to our use of a prototyping/agile software development life cycle.

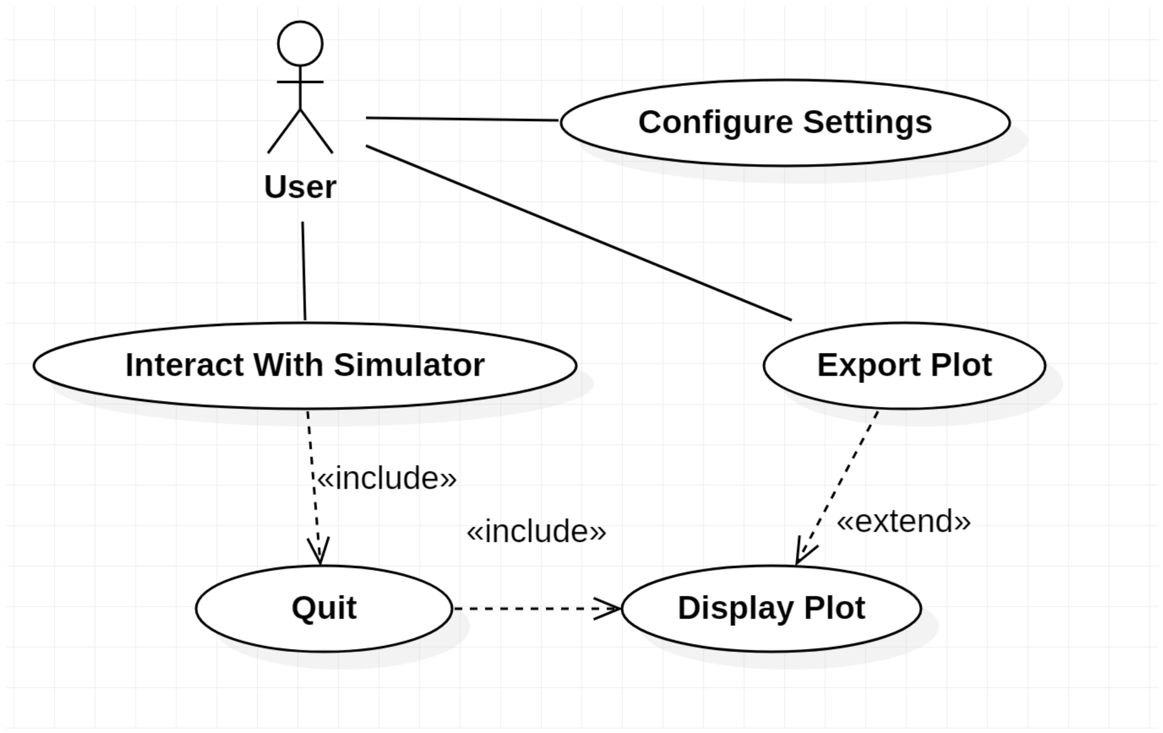
Modules within our product are reusable in other pieces of software. The grid generator to visualize a list of people can be used in games and other automata. Our current user interface may be specific to our project, however, the names and what settings they change can be modified and applied to other projects. The automata as well is geared for disease simulation, however, the values and state changes can be edited based on the new project's specification. Lastly, the Plot Generator at the end will be created in python due to pythons power in creating graphs. This module is reusable in any situation where graphs are needed to be generated and exported to a user.

Our software is reliable for the user. The settings in the UI directly change the SimSetting classes values which directly affect the automata's results. The visualization needs to be reliable as it is the main deliverable. Information is stored and combined with the plot producer to make the plots and display information. Reliability is at the forefront of this project as a lack of reliability will lose user confidence in the simulation we provide.

Documentation has been a major aspect of the project since its inception. We have created diagrams for all aspects of the project at the brainstorming stage and iteratively as the source code grows. Because of the extensibility of our project, the documentation needs to be constantly updated to accurately reflect the code created. The code is also documented using the Doxygen comment style so that any member can approach a file and understand exactly what it is doing. File headers, function headers, and in line commenting allows for easier readability and understanding from outside users.

2. Customer Requirements

See below for the Use case diagram for the simulation user.



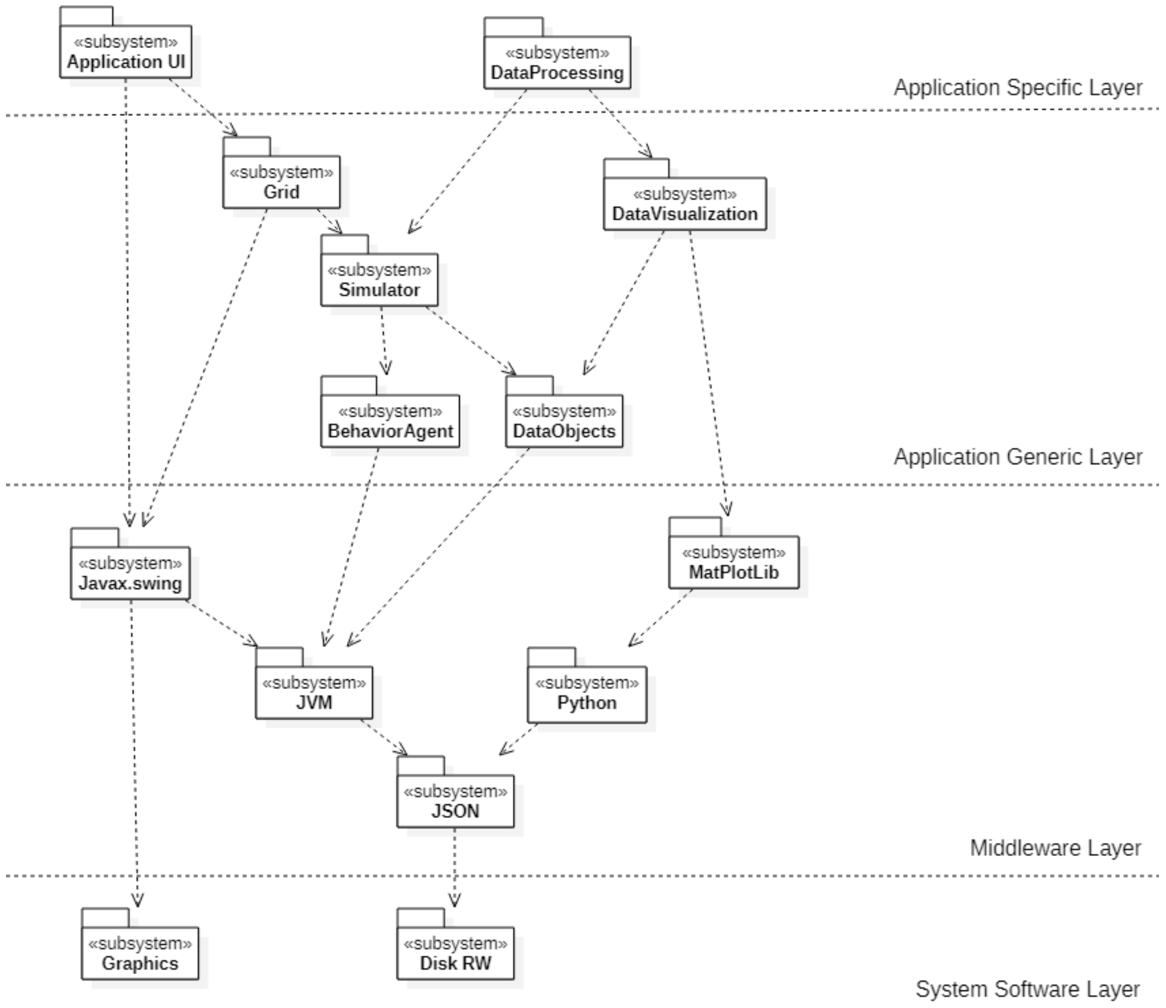
- Actors
  - User : Users are the people who interact with the UI and run the simulation. They can adjust the simulation settings and interact with post-simulation data.
- Use Case Descriptions
  - Configure Settings : Users can adjust the population, initial infection rate, duration of simulation, mask/vaccination/social distance percentages. Later prototypes will provide more options for users.
  - Interact with Simulation : Users can start the simulation after configuring the settings. From there the user has the option to play, speed up, slow down, and pause.
  - Display Plot : After the simulation is ended, the display plot use case is included to display the statistics of the simulation results.
  - Export Plot : Users will have the option to export the plot displayed after the simulation ends, to an image file.

3. Architectural Design

During the design process, several possible environments and architectures were considered for the software to run on. Most of these other options were eliminated due to time constraints or due to unnecessary complexity. As an example, a database was considered that could be used to store records of many viral strains and their configuration data, as well as complete records for each "person" in the simulation for a particular run. This added unnecessary hardware and software overhead, as it was determined that the software does not need or generate enough data to merit a full database system. Another consideration was making the software a web-based application, but this unnecessarily complicates the project further, and offers very little to the design goals. Thus, it was decided that an application run locally on the users machine would be best, as it offers the least complexity and still meets the design requirements.

In regards to the subsystem design, our team elected to stick somewhat closely to a closed subsystem architecture, so that the highest subsystem (the UI) need not access the lowest subsystem (the DataObjects) directly. This enables lower coupling between the subsystems and provides for ease of maintenance.

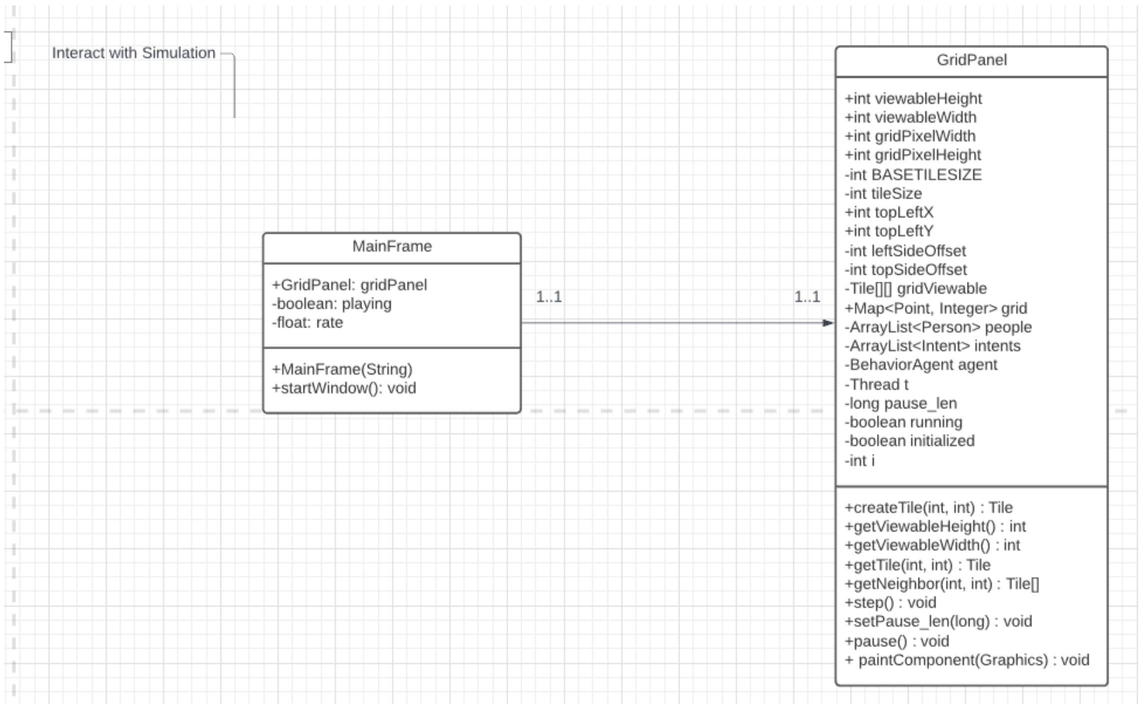
3.1 Subsystem Architecture

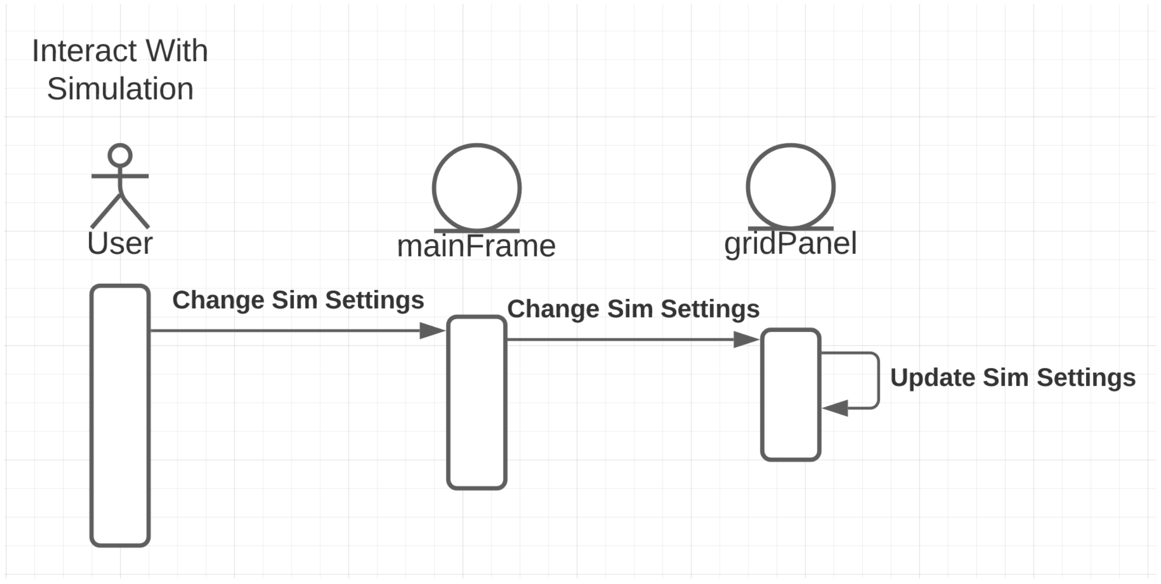


The primary subsystems are shown in the diagram above. The UI subsystem, which handles user interactions and input, relies on the Grid for drawing the cells. The DataProcessing component relies on the Simulator as well as the DataVisualization component. Each of these rely on the DataObjects package for additional information and some aggregate data. The BehaviorAgent gives intelligent behavior to the Simulator. Most of these packages rely on the Java VM to run, though the DataVisualization will rely on a separate python script and MatPlotLib. These two components will be able to interact via JSON files that are written to disk. There is very little to do with the System Software layer, as the entire software is meant to be run locally on the host machine, so only disk accesses and graphics processing is utilized by the software.

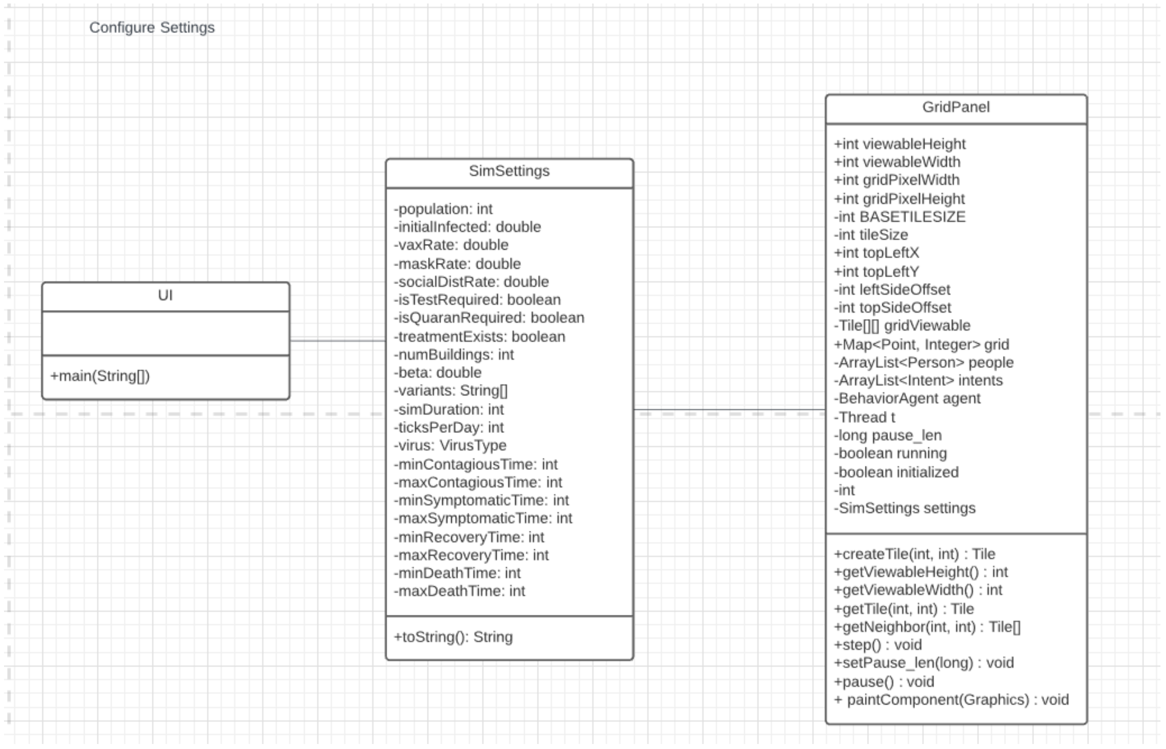
4. Use Case Realization Design

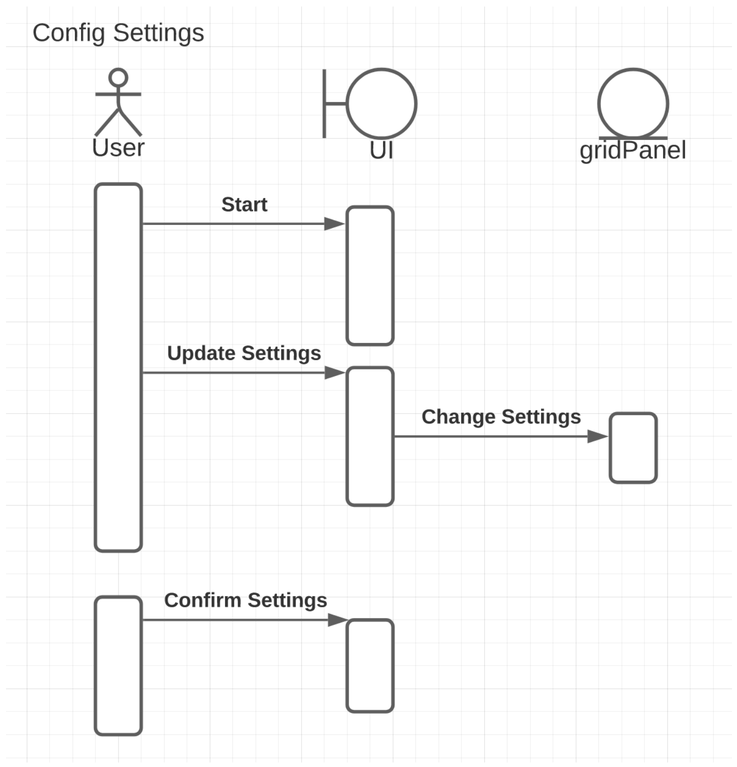
Interact With Simulation Use Case:



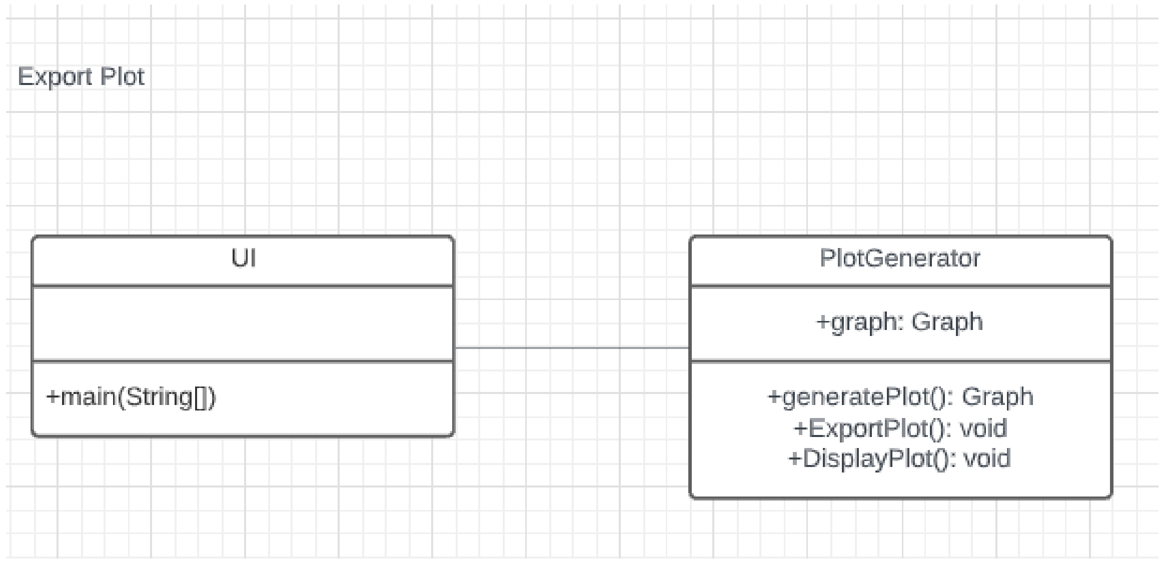


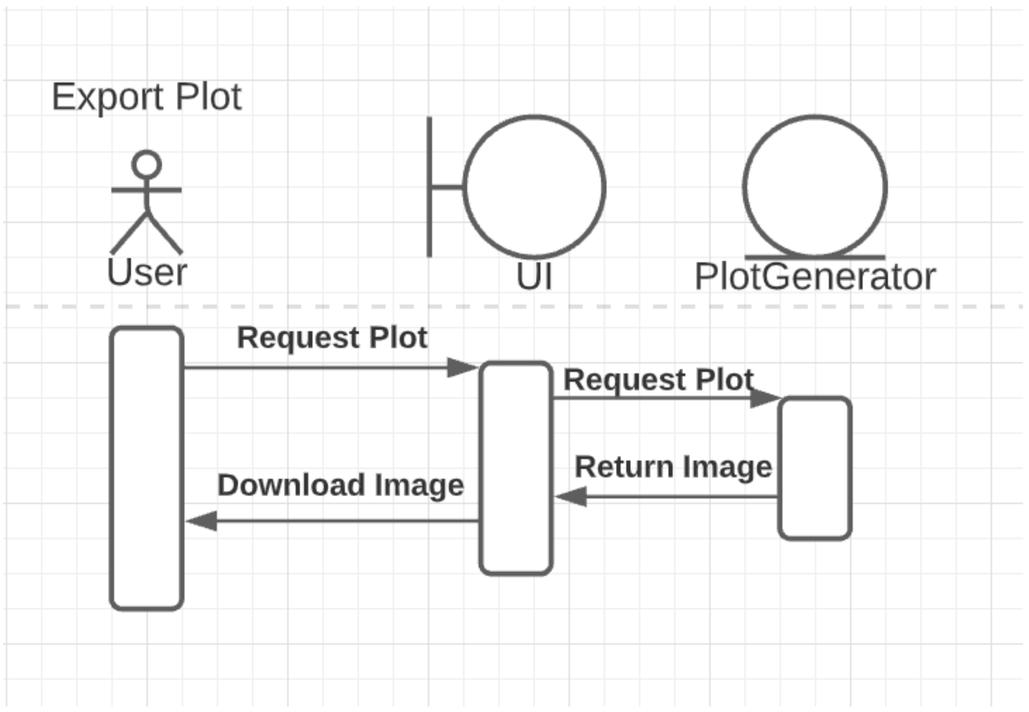
Configure Settings Use Case:



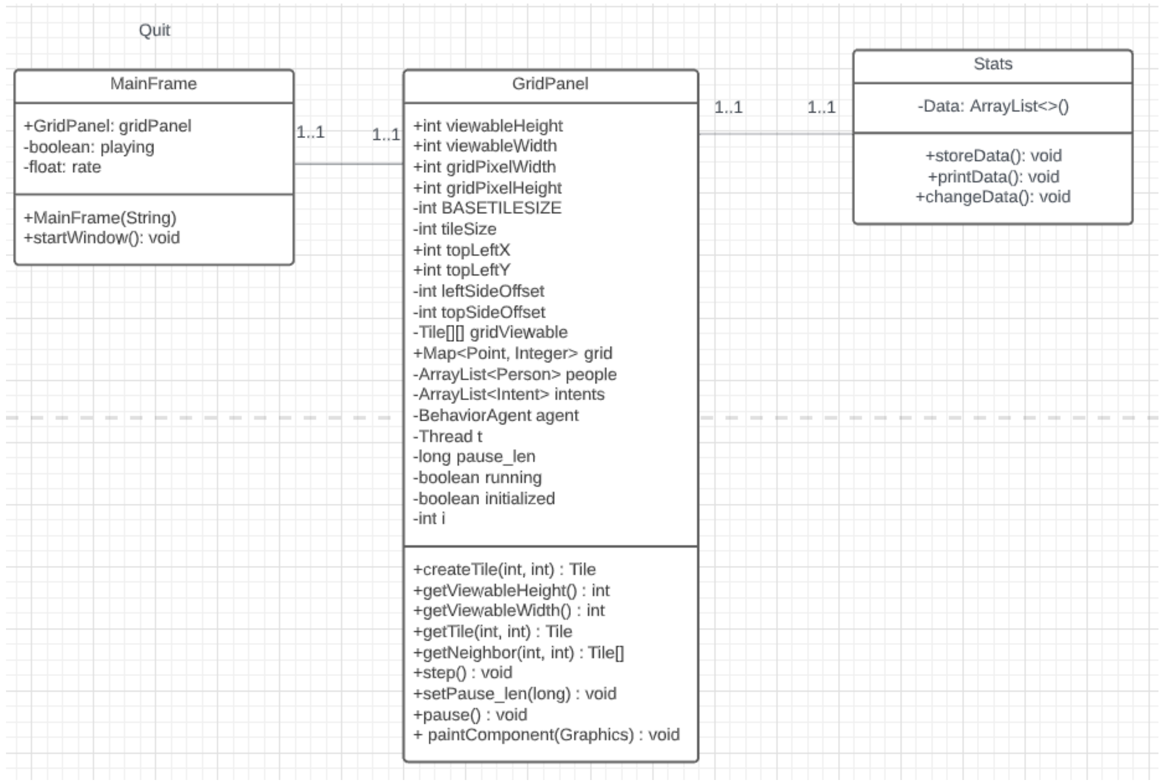


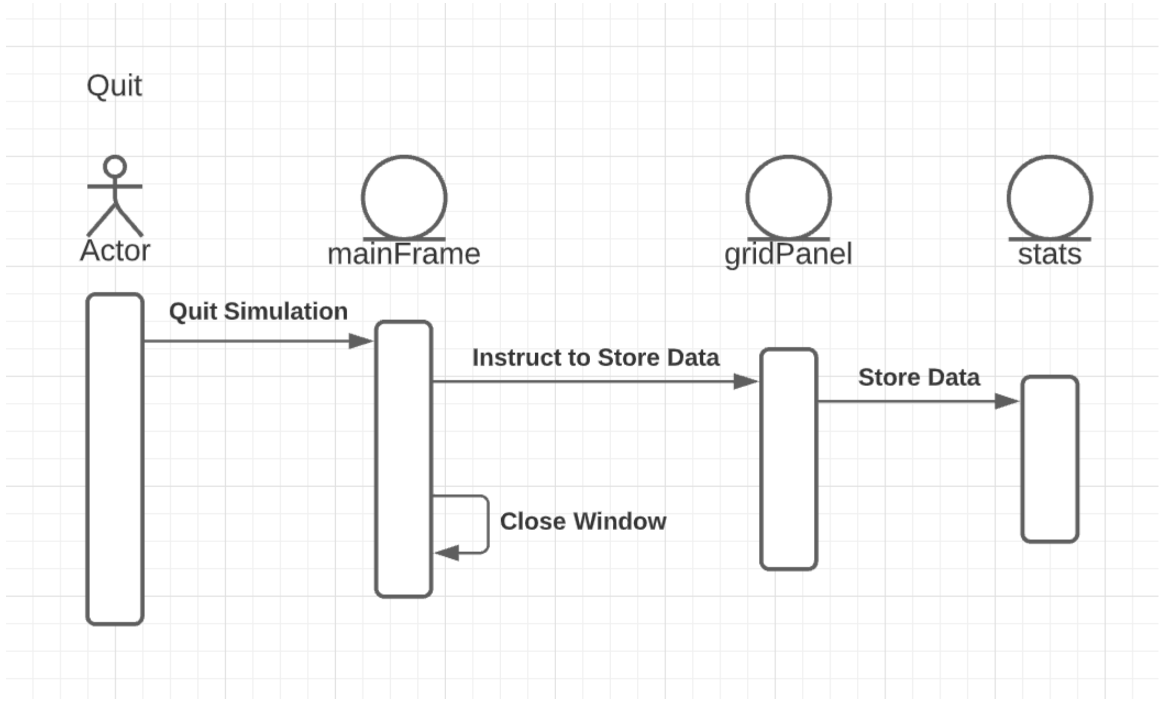
Export Plot use case:



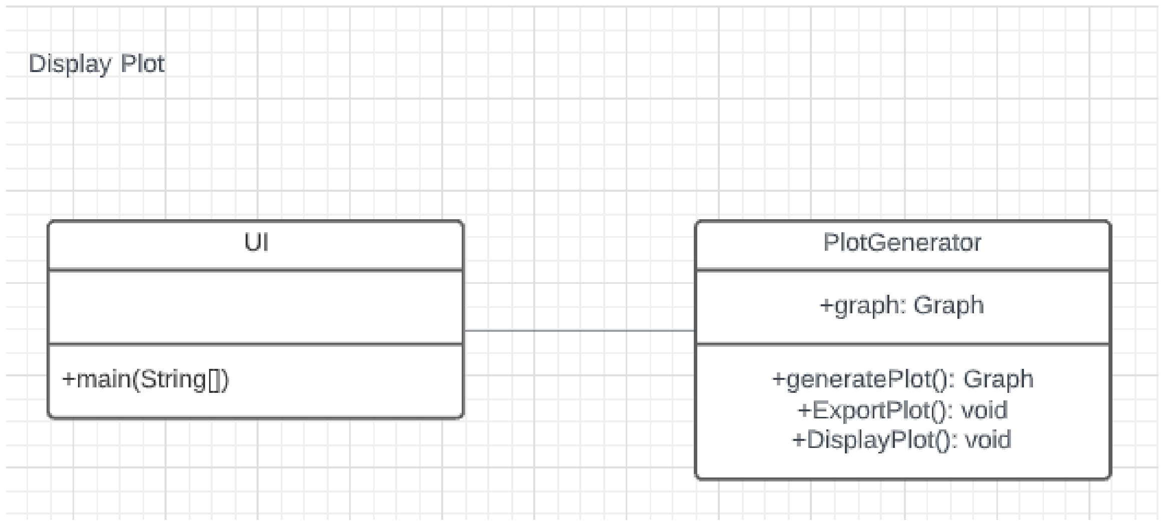


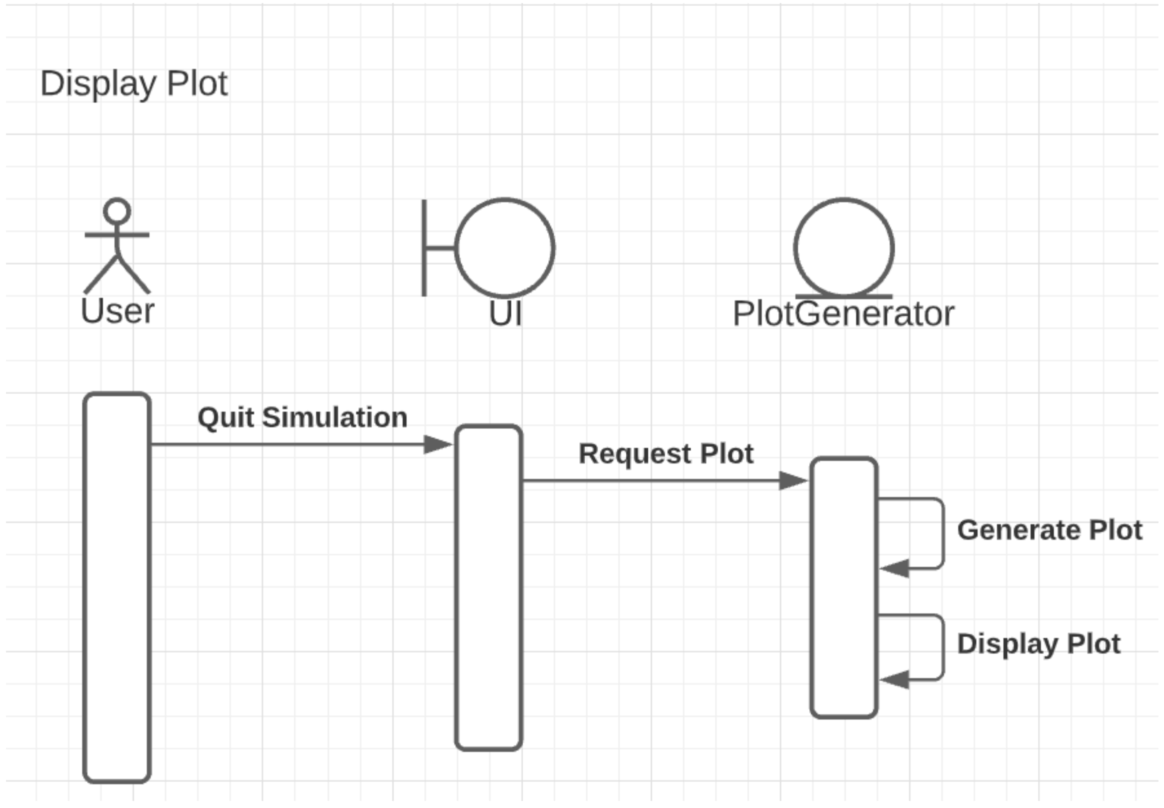
Quit use case:


Quit
MainFrame
+GridPanel: gridPanel
-boolean: playing
-float: rate
+MainFrame(String)
+startWindow(): void
1..1
1..1
GridPanel
+int viewableHeight
+int viewableWidth
+int gridPixelWidth
+int gridPixelHeight
-int BASETILESIZE
-int tileSize
+int topLeftX
+int topLeftY
-int leftSideOffset
-int topSideOffset
-Tile[][] gridViewable
+Map<Point, Integer> grid
-ArrayList<Person> people
-ArrayList<Intent> intents
-BehaviorAgent agent
-Thread t
-long pause_len
-boolean running
-boolean initialized
-int i
+createTile(int, int) : Tile
+getViewableHeight() : int
+getViewableWidth() : int
+getTile(int, int) : Tile
+getNeighbor(int, int) : Tile[]
+step() : void
+setPause_len(long) : void
+pause() : void
+ paintComponent(Graphics) : void
1..1
1..1
Stats
-Data: ArrayList<>()
+storeData(): void
+printData(): void
+changeData(): void


Quit
Actor
mainFrame
gridPanel
stats
Quit Simulation
Instruct to Store Data
Store Data
Close Window

Display Plot use case:





5. Subsystem Design

5.1 Grid

The grid subsystem has minimal interfaces open to other subsystems. The only other subsystem that calls methods present in the grid subsystem is Simulator, as it controls movements on the grid.

5.2 Data Objects

This subsystem has interfaces available to the Grid and Simulator subsystems. It allows the grid to access objects from it so that they can be displayed and acted on by other nearby objects. It allows the simulator to access and modify objects from this subsystem in order to dictate their behaviors.

5.3 Simulator

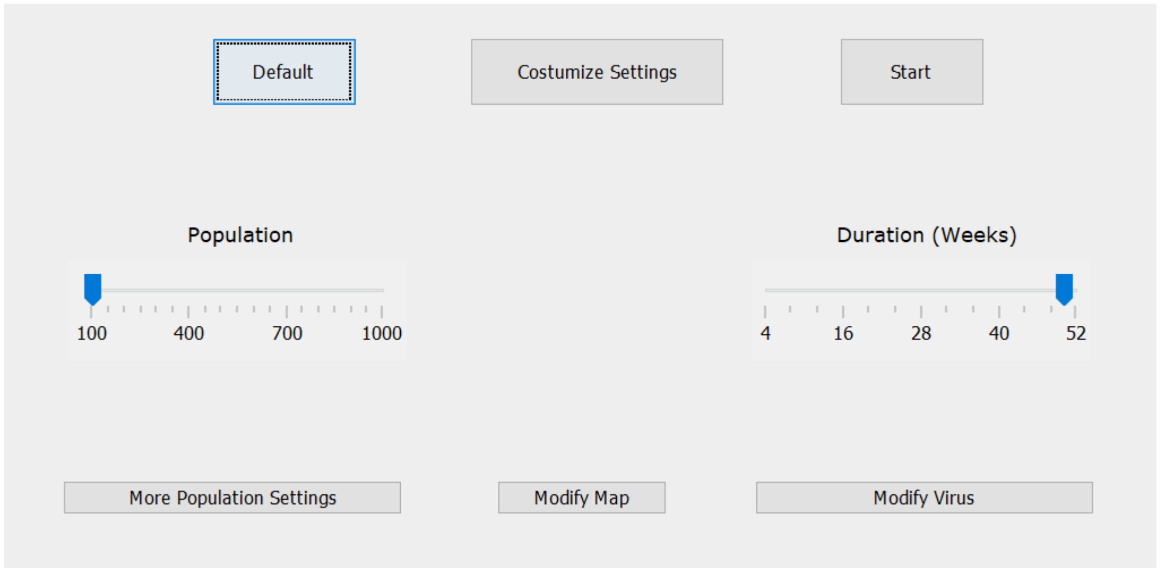
The simulator subsystem has interfaces available to all other subclasses. The interface available to UI allows the UI elements to change aspects of the simulation. The interfaces available to the other two subsystems simply facilitate the ability of the Simulator subsystem to act on those subsystems.

5.4 UI

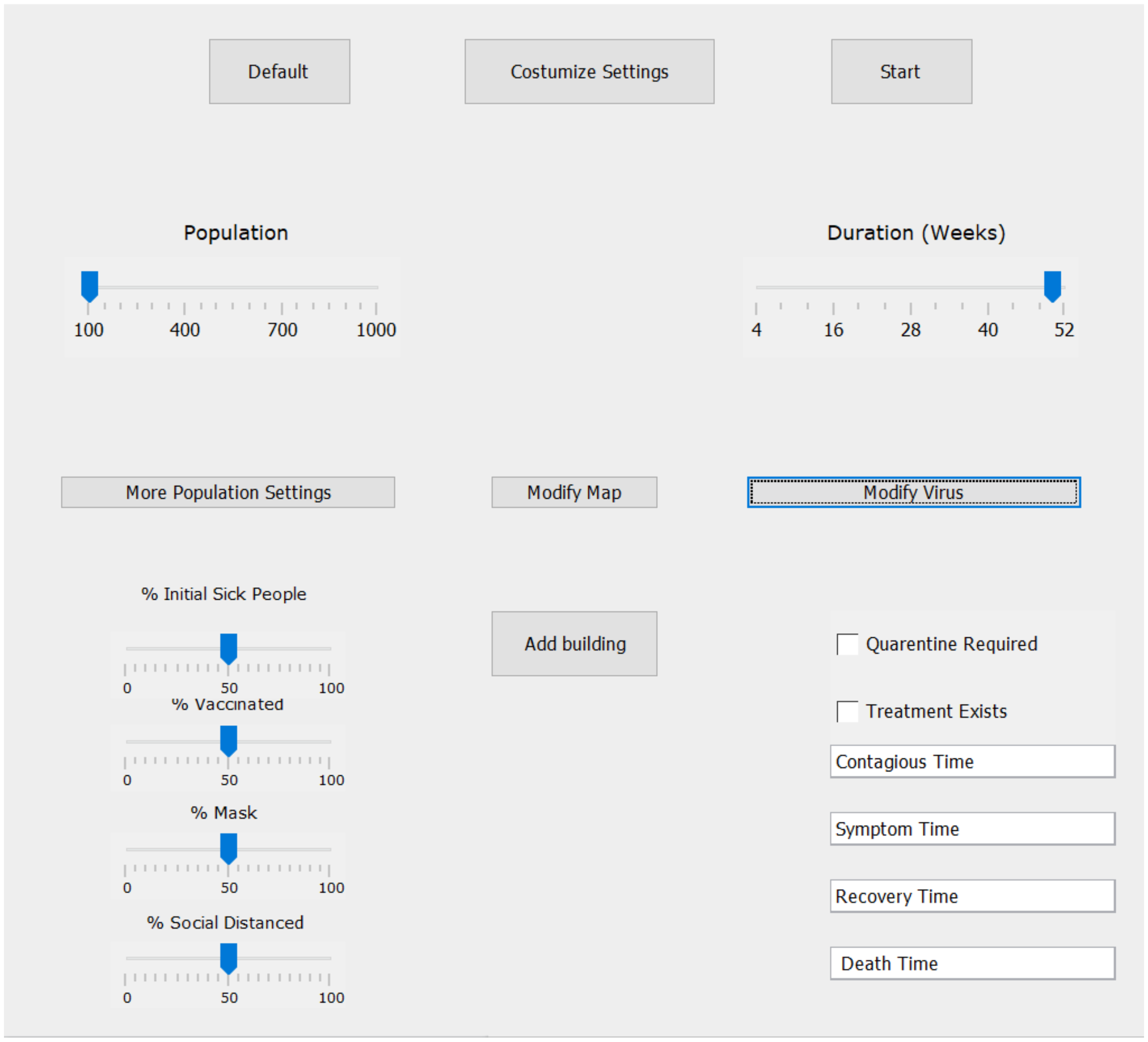
The UI subsystem has no interfaces available to any of the other subsystems.

6. Human Interfaces

When the program is run, the initial UI currently looks like this:



When the “Customize Settings”, “More Population Settings”, “Modify Map”, or “Modify Virus” buttons are pressed, it expands into



In the future, the UI will have buttons to add different types buildings to the simulation grid,

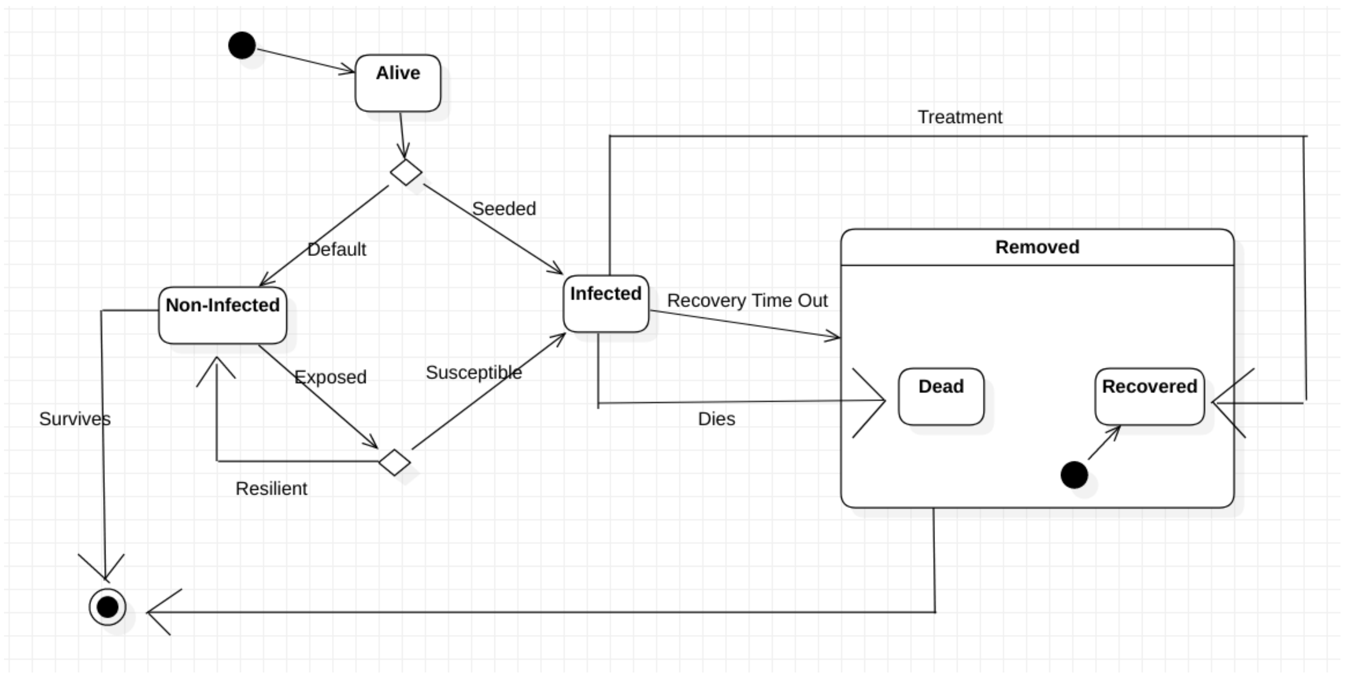
7. Testing Plan

Since our software does not have a measurable way to test if it works. We will run the simulation 10 times with the default settings, and then change one attribute 10 times. The average of the data of each simulated will be compared to see if the output has the expected changes. It is necessary to run each simulation multiple times and then find the average because each run has a factor of randomness.

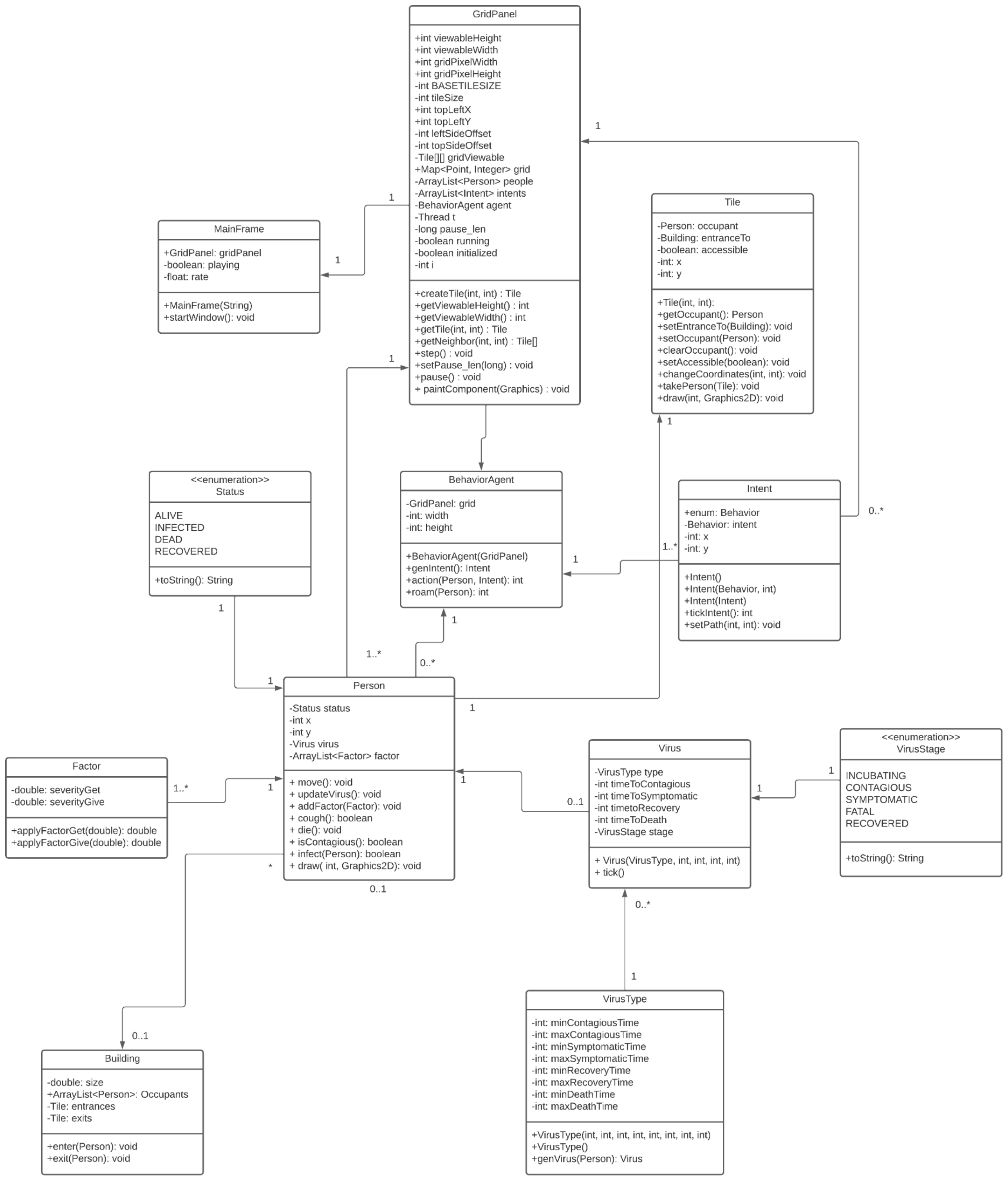
8. Appendices

See below for additional diagrams we have created in brainstorming and constructing code.

Person Status Statechart diagram:


Alive
Treatment
Seeded
Default
Removed
Non-Infected
Infected
Recovery Time Out
Exposed
Susceptible
Dead
Recovered
Survives
Dies
Resilient

Full Class Diagram for the project:



8.1 Project Status

Our project currently has a UI, grid visualization/simulator, and data objects. The UI has functioning sliders and buttons that affect the values in the data objects. The grid visualization currently shows a grid containing multiple objects. It can be run to show the objects moving, along with making the simulation stop, slow down, and speed up. The objects change color to show status. The data objects are able to take in data, complete calculations.

We have met every week as a whole group on every Sunday for a couple of hours to either process code, do diagrams, or complete reports. Individual members have met in small groups to complete tasks such as UI creation, diagrams, and automata manipulation.

Our project requires a few more elements to meet our objectives. Buildings need to be integrated, data needs to be collected during the simulation, and the data needs to be passed to another module for processing and generation for graphs. The settings need to have the ability to create custom viruses and buildings. We expect to roll these changes out before the final software is due.