FlowLogic: Traffic Flow Simulator

Design Document

Team 18

Colin Lappin

Dominic DeLuca

Isaac Hallman

Dylan Mitchell

Index

- **Purpose** **2**
  - Functional Requirements
  - Non-Functional Requirements
- **Design Outline** **6**
  - High Level Overview
  - State Diagram Overview
- **Design Issues** **9**
  - Functional Issues
  - Non-Functional Issues
- **Design Details** **12**
  - Class Diagram
  - Description of Classes
  - Sequence Diagram
  - Navigation Flow Map
  - State Diagram
  - UI Mockup

Purpose

Urban road design is an ever-important field as cities grow and new ones appear. The process of designing new layouts for these roads can be difficult, though, especially considering that the existing software available to assist in this task is hard to use.

There are existing software solutions such as Vissim, SUMO, and SimTraffic that provide traffic simulation capabilities. These solutions lack, however, in their difficulty-of-use. SUMO even requires coding knowledge, while Vissim and SimTraffic require experience and knowledge of their hard-to-use UI just to get things done with any efficiency.

We are designing a traffic flow simulator that will be used by traffic professionals and enthusiasts alike. The goal of the program is to provide a useful, easy-to-use and accessible design solution for creating urban traffic layouts that allows for efficient simulation and visualization. This will be accomplished by providing a user-friendly drag-and-drop style system for layout design, and providing visual feedback after traffic is simulated.

Functional Requirements

As a designer,

1. I would like to have a grid-style canvas that I can build on
2. I would like to drag and drop roads onto the grid
3. I would like to adjust intersection types so that I can control traffic flow.
4. I would like to be able to adjust the amount of people going to each building so that I can simulate a realistic environment.
5. I would like to place parking areas with different parking capacities.

6. I would like to configure traffic light timings so that I can test different strategies.
7. I would like to be able to simulate traffic flow so that I can view how my layout would work in a real world scenario.
8. I would like to be able to import saved layouts so that I can see other's work.
9. I would like to be able to export saved layouts so that I can share my work with others.
10. I would like to see statistics from the traffic simulation so that I can perform an analysis of my layout.
11. I would like to receive suggestions on my layouts so that I can improve on them.
12. I would like to adjust speed limits on roads.
13. I would like to be presented with a menu when I open the app.
14. I would like to be able to adjust the width (# of lanes) on roads.
15. I would like to adjust the direction from where traffic flows in the simulation.
16. I would like an adjustable-sized map for different size layouts.
17. I would like to graphically view statistics.
18. I would like to be able to edit and remove roads.
19. I would like to be able to edit and remove buildings.
20. I would like roads to automatically snap together into intersections when they meet.
21. I would like to be able to save layouts to my computer.
22. I would like to be able to load my saved layouts to the simulation.
23. I would like to view my saved layouts all in one place
24. I would like to delete and rename saved layouts.

25. I would like to name roads and buildings within a layout, so that I can refer to them more easily.
26. I would like to assign different types of vehicles to the simulation.
27. I would like to add pedestrian walkways and crosswalks if time allows.
28. I would like to be able to add construction zones that impact traffic flow.
29. I would like to be able to add one-way streets
30. I would like to be able to simulate rush hour periods.
31. I would like to be able to compare statistics between two saved layouts if time allows.
32. I would like to be able to set accident probabilities if time allows.
33. I would like to simulate special events (concerts, games, etc) if time allows.
34. I would like to be able to add roundabouts.
35. I would like to be able to factor in emergency vehicles if time allows.
36. I would like to be able to add public transportation if time allows.
37. I would like an undo button.

As a viewer,

1. I would like to adjust the speed of the simulation so that I can run real-time or accelerated scenarios as needed.
2. I would like to be able to simulate traffic flow so that I can view how my layout would work in a real world scenario.
3. I would like to be able to import saved layouts so that I can see other's work.
4. I would like to see statistics from the traffic simulation so that I can perform an analysis of my layout.
5. I would like to be presented with a menu when I open the app.
6. I would like to view my saved layouts all in one place
7. I would like to delete and rename saved layouts.

Non-Functional Requirements

Architecture & Performance

As a developer, I want to

1. Use the Java class system for backend development.
2. Use JavaFX for frontend development.
3. Use the Model-View-Controller Architecture, which separates the UI completely from the backend.
4. Handle a simulation of 20,000 cars in 5 minutes or less.

Usability & Accessibility

As a developer, I want to

1. I want the grid objects to snap immediately to the grid when placed
2. Anyone to be able to learn the UI within 15 minutes
3. Run my application on any major OS (MacOS, Linux, Windows)

Security

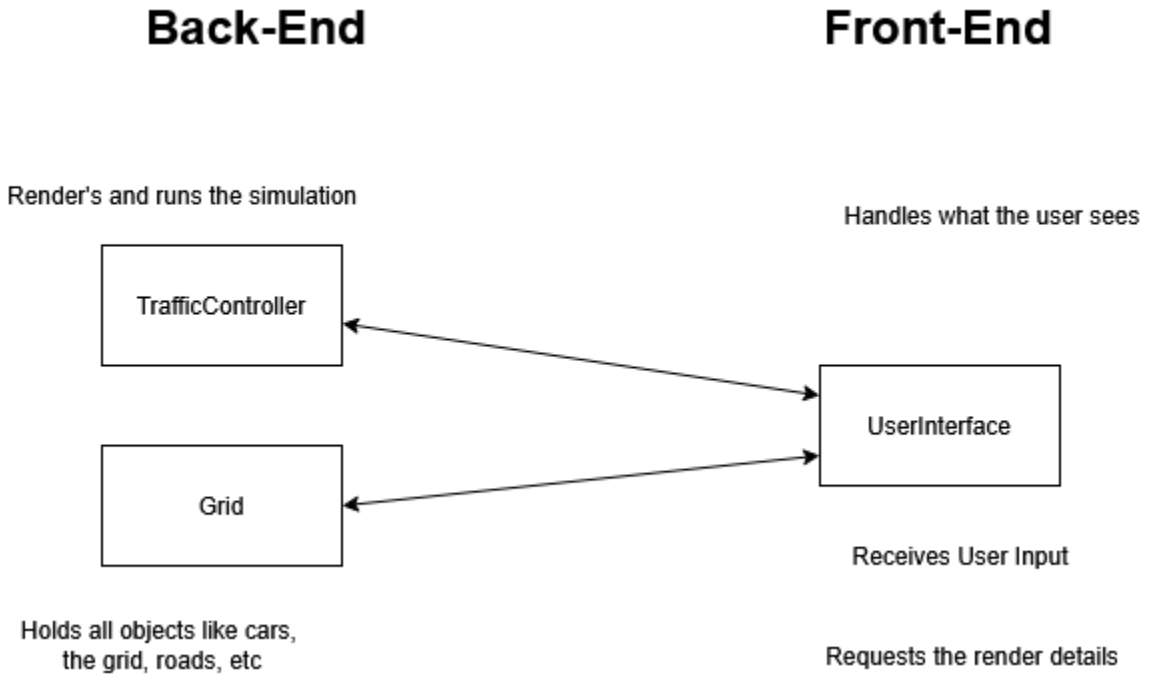
As a developer, I want to

1. Decrease my liability for security by keeping this app offline
2. Allow the user to have full control over file sharing

Design Outline

High-Level Overview

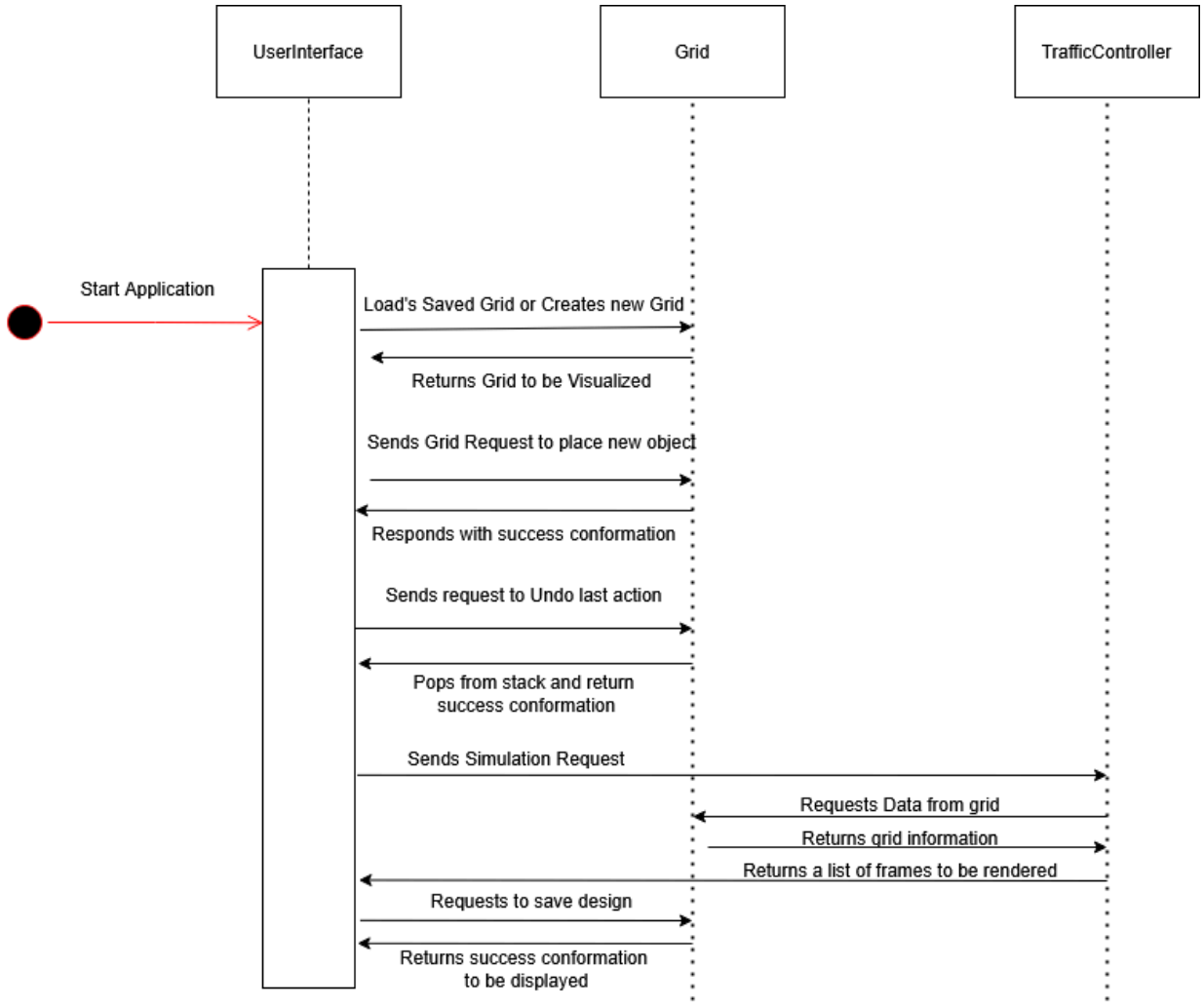
The traffic simulation system is divided into two main components: the back end, which handles data storage and simulation processing, and the front end, which manages user input and rendering. The back-end consists of a TrafficController, responsible for running the simulation and calculating vehicle movements, and a Grid, which stores all relevant objects like roads, intersections, and vehicles. The front end is represented by the UserInterface, which displays the simulation and processes user input. The UserInterface communicates with both back-end components. It communicates with the Grid to store changes the user makes and visualize the Grid. It communicates with the Traffic controller to compute the simulation and then render the cars and their paths.



1. **UserInterface**
    a. What the user sees and interacts with
    b. Sends requests to Grid to save changes made by the User
    c. Renders new information to the screen
2. **TrafficController**
    a. Computes the path driven by each car
    b. Stores the location and speed of each vehicle in each time step, allowing for replaying and changing of speed in the simulation
    c. Accessed by UserInterface whenever the User wants to run a simulation
3. **Grid**
    a. Stores all GridObjects (Roads, Building, etc)
    b. Stores a stack of changes, allowing for an undo
    c. Accessed by UserInteface whenever the User requests a change

Sequence of Events Overview

The sequence of events overview shows the interaction among UserInterface, Grid, and TrafficController. When the program is started, the UserInterface will either load a saved grid or create a new grid. The user can send requests to the grid to do things such as place a new object. If the operation is successful, the grid will return a success confirmation. Additionally, the user can request to undo the last action which will have the grid pop off the changes stack and undo the action before returning a success confirmation. To simulate traffic flow, the UserInterface sends a simulation request to the TrafficController. The controller retrieves data from the grid and then processes it into frames, which is returned to the UserInterface to be visualized.



Design Issues

Functional Issues

- **Do users need to login to use the system?**
    - Option 1: Make an account and allow users to store their own designs as a “designer” and view others as a “viewer”
    - Option 2: Allow users to select their desired “mode” whenever. They can pick whether they want to design or view a model or switch between the two.

Decision: Option 2

Justification: Since there are not a lot of security concerns involved with this project, there is really no need to create accounts. All the files being stored locally means that designers can switch to a “view mode” to show off their work. The view mode just makes it easier to understand what's happening in a simulation by removing options for editing the layout.

- **How do users share their design?**
    - Option 1: Create a networking system by which the users can send files back and forth via a stream.
    - Option 2: Allow the storage of files to be local, and users can simply email the files to one another.

Decision: Option 2

Justification: This solution is much simpler to implement. If there was more time in the semester, perhaps this is something that would be implemented, but allowing users to simply email files back and forth will give us time to focus on the harder parts of this project.

- **How do we handle cars entering the model?**
    - Option 1: Have them generate offscreen, then drive onto the Grid of the Simulation from a random side of the screen.
    - Option 2: Have them appear at random throughout the layout

    - Option 3: Have designated “in roads” (that must connect to the side of the screen)where users can specify which roads the vehicles will enter the layout from

Decision: Option 3

Justification: This option will allow the user to have more control over the simulation, therefore allowing them to make layouts that are more accurate to real life scenarios.

- **How should we handle intersections?**
    - Option 1: Users must physically place intersections on their own.
    - Option 2: Intersection automatically is created when two opposite facing roads are placed on the same grid spot.

Decision: Option 2

Justification: Option two allows the user to place roads where they need to go and worry about the intersection type later. It’s also much less work for the user when the intersection is created automatically. This will only be a bit more work for us, so the tradeoff is worth it.

Non-Functional Issues

- **What service should we use to store data?**
    - Option 1: MongoDB (Online)
    - Option 2: mySQL (Online)
    - Option 3: JSON Data (Local)

Decision: Option 3

Justification: This option takes the application offline, making it more secure. Additionally the JSON data will work well to include metadata along with necessary data for saving each file.

- **What language is appropriate for implementing the backend services?**
    - Option 1: Java

    - Option 2: JavaScript
    - Option 3: Python
    - Option 4: C++

Decision: Java

Justification: A Traffic simulation involves a lot of real-time processing of multiple elements such as vehicles, intersections, traffic signals, etc. which all must operate concurrently. Java provides efficient multithreading mechanisms (like Thread) that make it ideal for running all of these at the same time. Additionally, since JavaFX is going to be used for the frontend, choosing Java for the backend allows efficient communication between frontend and backend components.

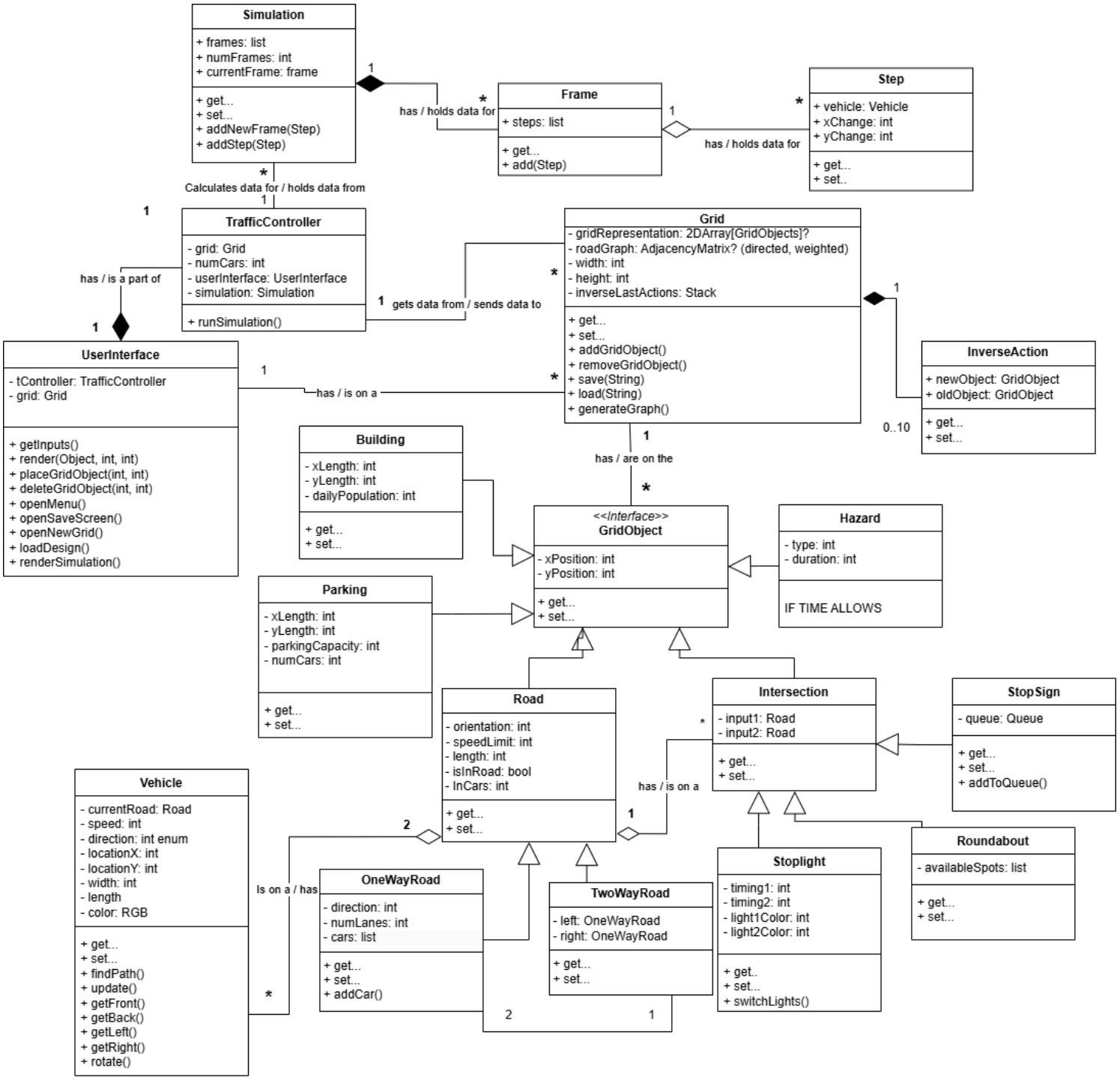
- **What language is appropriate for implementing the frontend services?**
    - Option 1: React
    - Option 2: JavaFX
    - Option 3: WebGL

Decision: JavaFX

Justification: Our simulator requires an interactive grid-based canvas where users can drag and drop roads, intersections, and other infrastructure. JavaFX happens to provide built-in support for drag-and-drop elements (such as DragEvent) along with resizable, interactive nodes (such as Canvas, Pane, etc). These make JavaFX suitable for graphical layout editing. Additionally, since our backend will be utilizing Java, JavaFX will ensure smooth integration with the Java-based system.

Design Details

Class Design



Description of Classes

- **Simulation**
    - A Simulation is what displays to the user when they select the simulate option. It displays each frame of the simulation as it runs, showing the user what traffic might be like with the system they built.
    - A Simulation holds a list of Frames that it will iterate through as the simulation runs. Simulations hold the data for Frames.
    - A Simulation also stores the number of frames it will run through.
    - A Simulation will also keep track of which frame in the list it is currently on.
- **Frame**
    - A Frame object is a container of Step objects and holds their data. Each Frame will be iterated through by a Simulation object to create the traffic simulation.
    - Frame stores a list of Step objects that it will iterate through to build the Frame.
- **Step**
    - A Step object is representative of a change in the Simulation. Each Step object will be used to build a Frame that a Simulation will iterate through.
    - Each Step object stores a Vehicle that it will change the location of. It will also store the Vehicle's change in X and Y coordinates for the Grid.
- **TrafficController**
    - An instance of the TrafficController class will be used by the UI to run calculations for and generate new simulations that it can render.
    - TrafficController holds an instance of the grid for the layout, as well as a simulation object that it creates and fills.

- **UserInterface**
    - This class handles all that the user will see and interact with, including generating new windows, displaying a simulation, and dragging and dropping roads and buildings.
    - This class will make calls to the back-end classes “Grid” and “TrafficController”.
    - This class stores a grid object that it displays, as well is a traffic controller that it uses to run simulations.
- **Grid**
    - The grid holds all of the grid objects (roads, buildings, etc.) and is referenced by the UI for display. It is also referenced by the TrafficController for simulation.
    - A Grid object will store its length and width to determine how big the Grid will be for the Simulation.
- **InverseAction**
    - This class stores an action that the user has completed like placing a building or removing a road. It will be used for the “undo” button
    - InverseAction stores OldObject, which is the grid object with its old attributes, and NewObject, which is the grid object with its new attributes
- **GridObject**
    - A GridObject is the “parent” to everything that can appear on the map during the simulation. Objects like Roads, Hazards, Buildings, etc. are all GridObjects and will be placed on the grid accordingly during a simulation.
    - All GridObjects will store an X and Y coordinate to keep track of their location on the grid.
- **Building**

    - This is what people's "destinations" are. It will be placed on the grid and the user can specify how many people will use this building throughout the day.
    - This stores the maximum number of people that will go to this building as well as the size attributes
- **Parking**
    - This is a grid object where cars, during the simulation will be able to park
    - It stores the maximum capacity of the parking area and the number of cars currently inside it for simulation time, as well as the size attributes
- **Road**
    - The road class is the parent class to both OneWayRoad and TwoWayRoad
    - The Road parent class stores whether or not the road is an input road from the side of the map, the speed limit of the road, the orientation, and the length.
- **Intersection**
    - An Intersection object will appear whenever two Roads cross each other, forming one.
    - An Intersection will take two inputs–the Road objects feeding into it to form an Intersection.
- **Hazard**
    - A Hazard can appear randomly on a road to represent a potential hazard one may encounter while driving. It will slow down/block traffic within the simulation.
    - A Hazard will keep track of what type of Hazard it can be.
    - A Hazard will only appear on the road for a limited amount of time before disappearing.

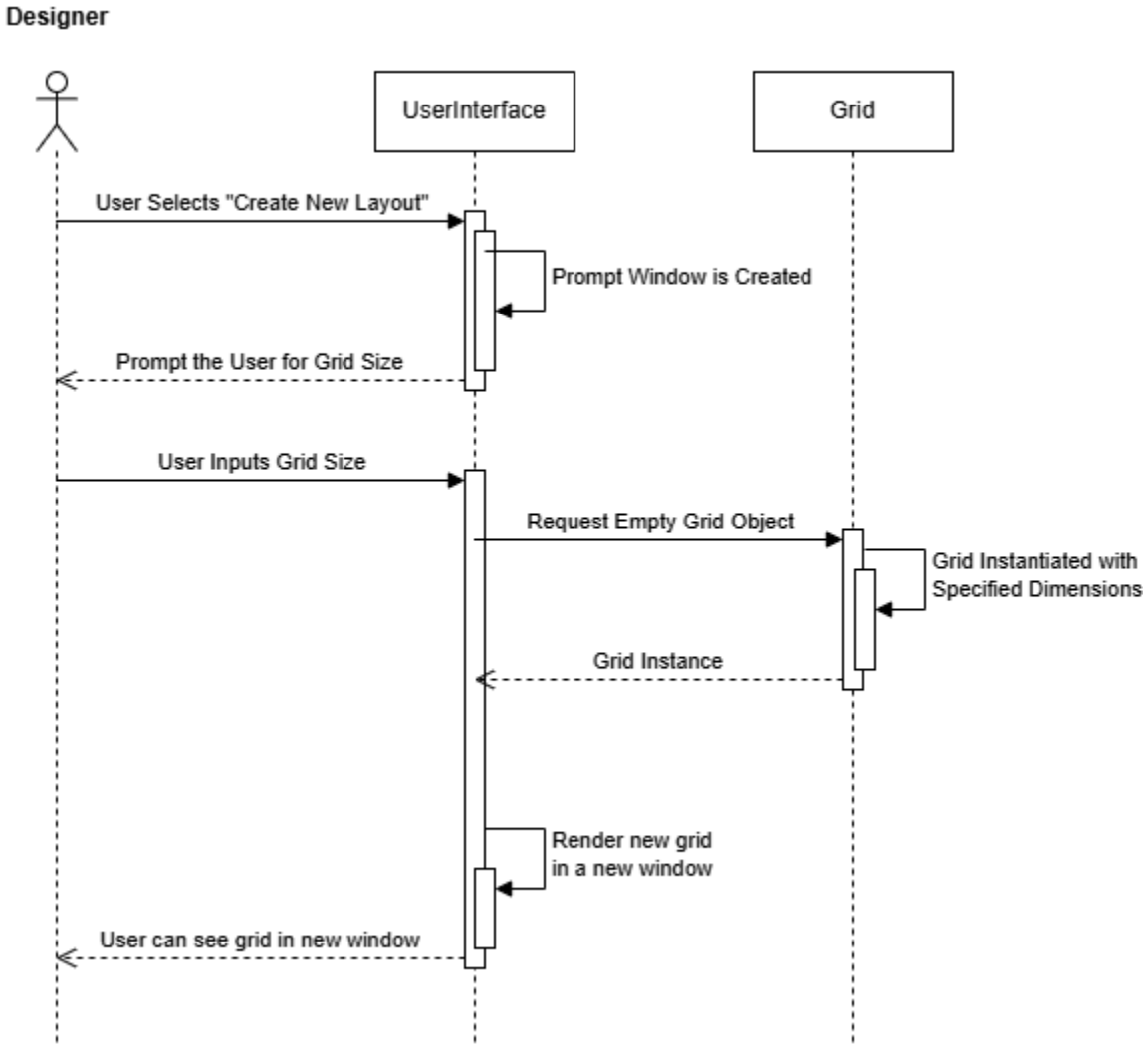
- **StopSign**
    - A StopSign is a type of intersection object that will appear along Roads where a stop is necessary to prevent collisions.
    - A StopSign object will hold a queue of Vehicles that are currently waiting on the way to be cleared before they can go.
- **Stoplight**
    - A Stoplight will record whether or not Vehicles pulling up to an Intersection will be allowed to move forward.
    - A Stoplight will store two colors–one allowing them to move forward (green) and one forcing them to stop (red).
    - A Stoplight will also store 2 timers. Each timer will indicate how long the light will stay on its corresponding color before switching to the opposite color.
- **Roundabout**
    - A Roundabout will move a lot smoother than an intersection. Rather than saving timers for Stop Lights, they will allow only a certain number of cars in before forcing every other Vehicle to wait for a spot to open.
    - A Roundabout will utilize a list of available spots. If this list has space, a Vehicle may enter the Roundabout. If full, the Vehicle must wait for a spot to open.
- **OneWayRoad**
    - A OneWayRoad will be utilized by Vehicle objects while they're moving within the simulation.
    - A OneWayRoad object will have a designated direction. A Vehicle driving on this road will only be able to move that direction.
    - A OneWayRoad will also save the number of lanes. This will determine how many Vehicles are able to move on it at once.

    - A OneWayRoad will also keep track of the Vehicles currently moving on it.
- **TwoWayRoad**
    - A TwoWayRoad will be, at its core, two OneWayRoad objects right next to each other. They will hold the same values as a OneWayRoad on its own.
    - A TwoWayRoad will keep track of a OneWayRoad on the right, and a OneWayRoad on the left.
- **Vehicle**
    - Each car that is shown in a simulation run by the user will be represented/tied to a Vehicle object.
    - Each Vehicle will keep track of the current road it is driving on.
    - Each Vehicle will keep track of its speed during the simulation.
    - Each Vehicle will keep track of the direction it is moving in for the simulation.
    - Each Vehicle will keep track of its location on screen using X and Y coordinates via a grid.
    - A Vehicle's size will be represented using a width and length modifier.
    - A Vehicle's color will be determined with an RGB value.

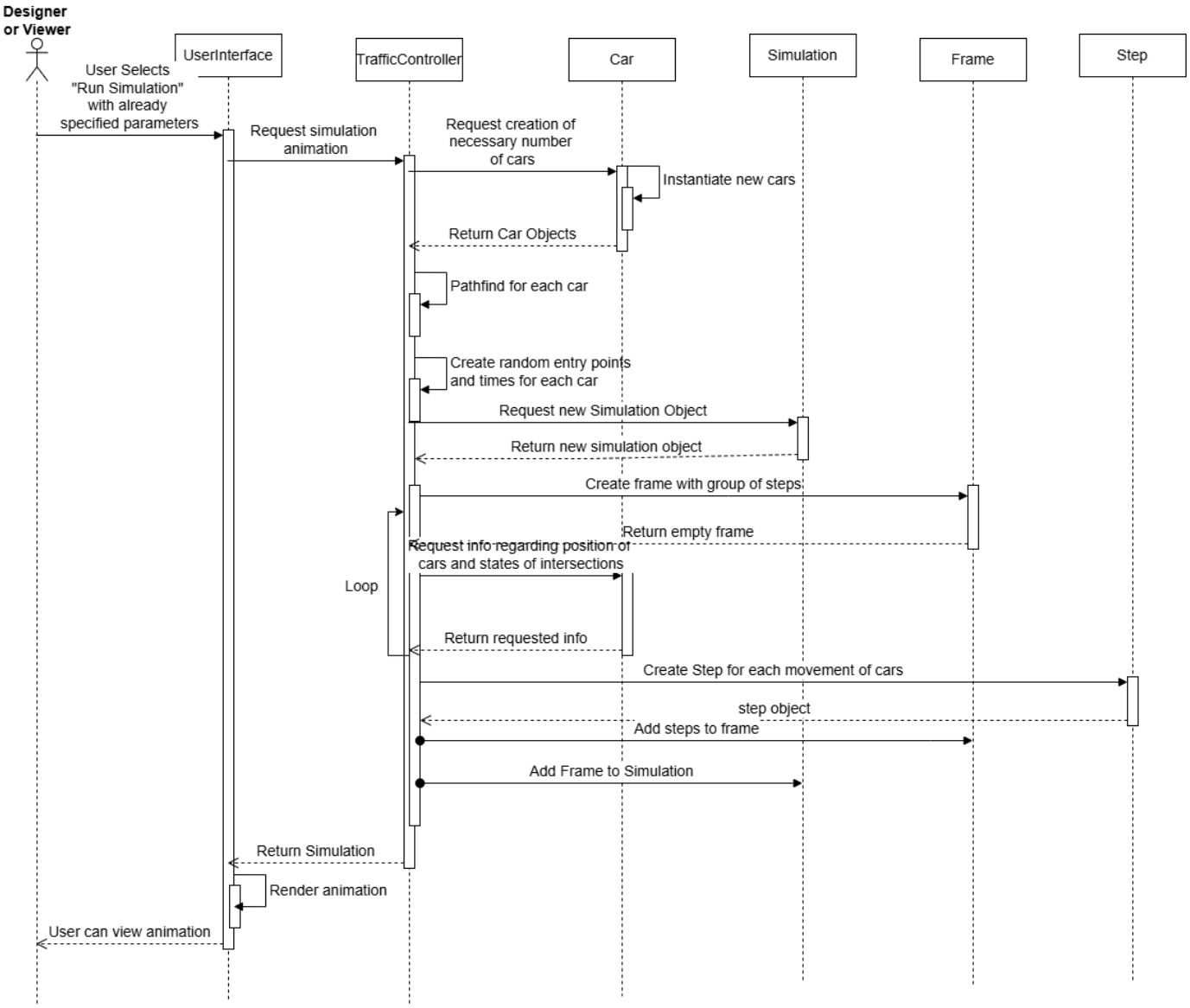
Sequence Diagram

The following diagrams depict a typical user request, and then the sequence of logical events and interactions between objects in the back end and front end of the application to ensure that the user's request is met.

1. Sequence of events when a designer creates a new layout

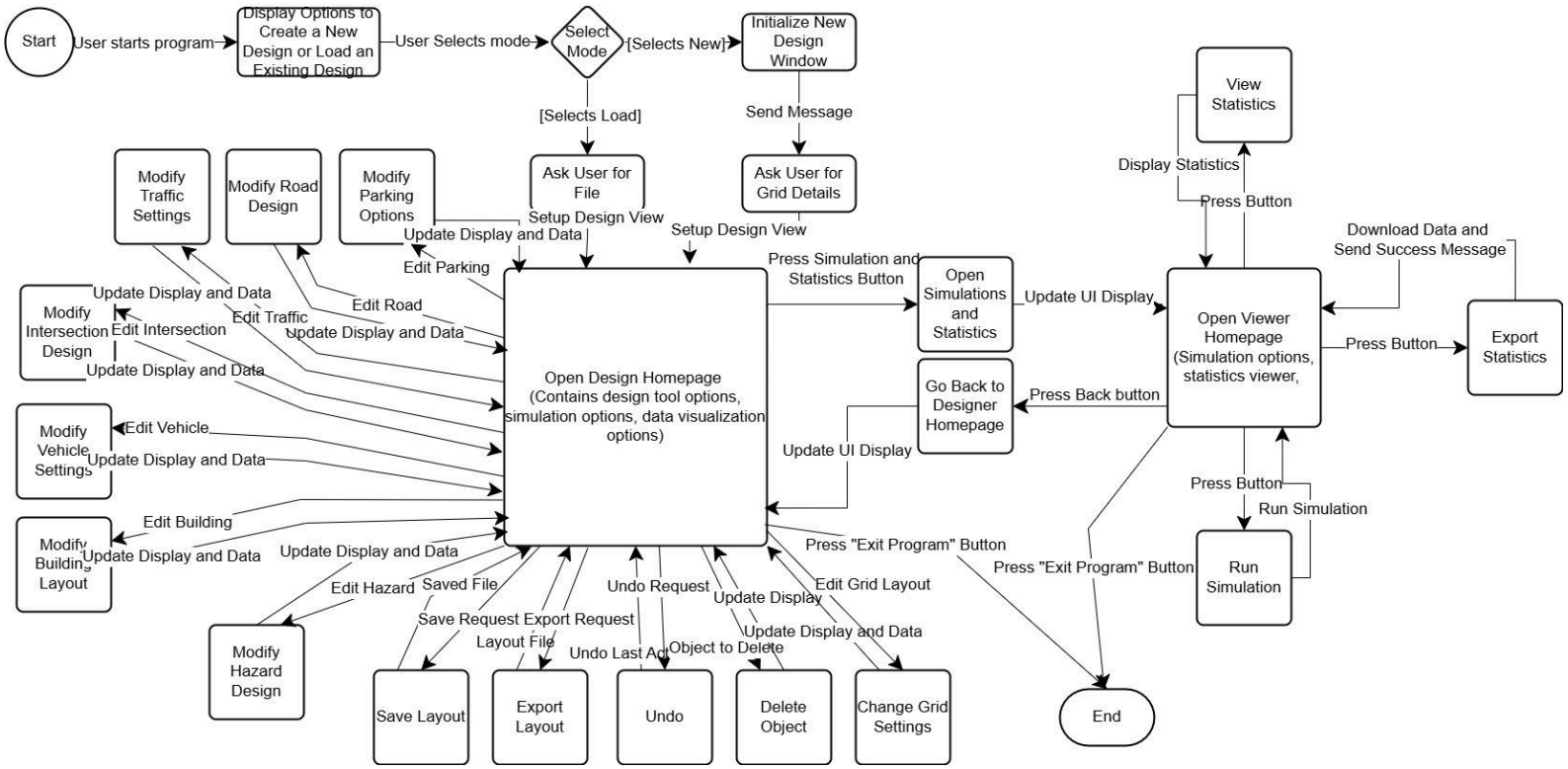


2. Sequence of events when a user runs a simulation



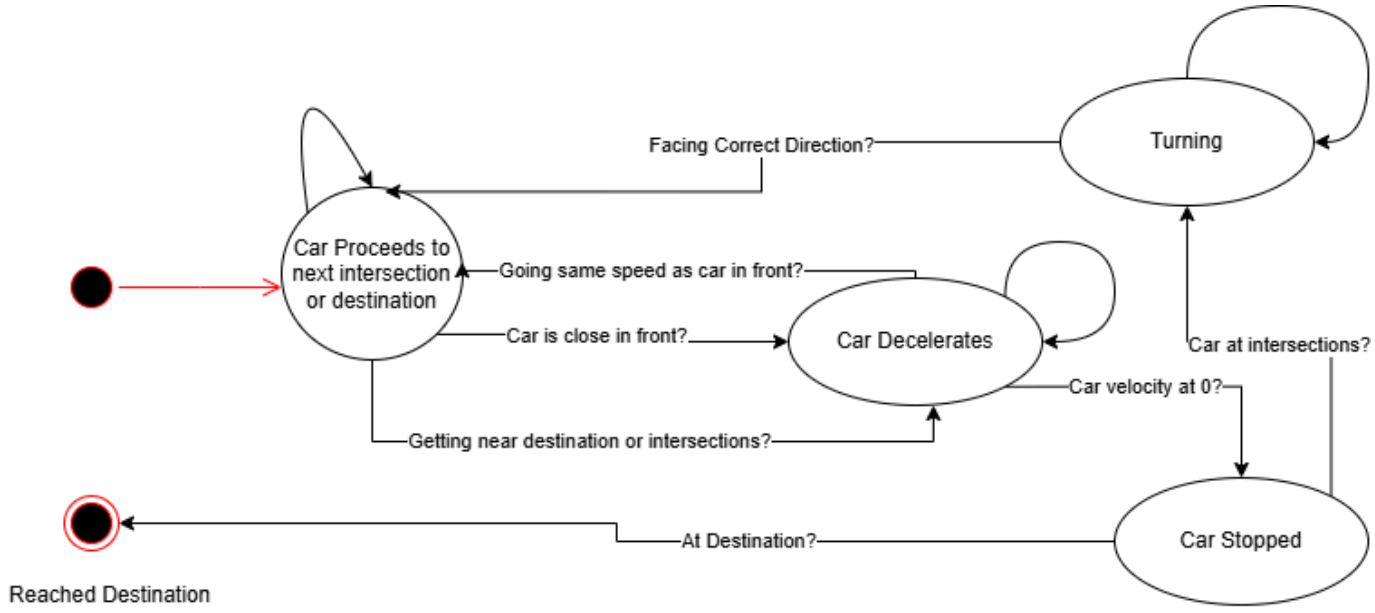
Navigation Flow Map

The design of our navigation emphasizes high functionality with a variety of options for the user. The user is encouraged to spend most of their time on the Design Homepage, customizing and designing components on the main grid object. This homepage contains all of the tools and editing options. The user can add Roads, Intersections, Buildings, Hazards, and Vehicles to the grid. The user can also edit or delete an object by clicking on it and viewing its edit menu. Once the user has completed their design, they can run traffic simulations and view statistics through the Viewer Homepage. On this homepage, the user can select to run a simulation, view statistics, and export the statistics. The user can navigate between homepages by clicking the homepage buttons.



State Diagram

This state diagram shows the different states in which a car can be during the course of a simple simulation.



For most of the simulation, the car proceeds to the next intersection as normal. If the car gets close to the one in front of it then it decelerates. If it is nearing its destination or the next intersection then it also decelerates. If the car decelerates all the way down to 0 and is at an intersection, then it turns onto the next road. If it is at the destination then we have reached the end. If not, then we return to the "continuing to next intersection" state.

UI Mockup

Main Menu

This is the main menu for our program; it allows the user to select whether to load an old simulation or begin with a new one.

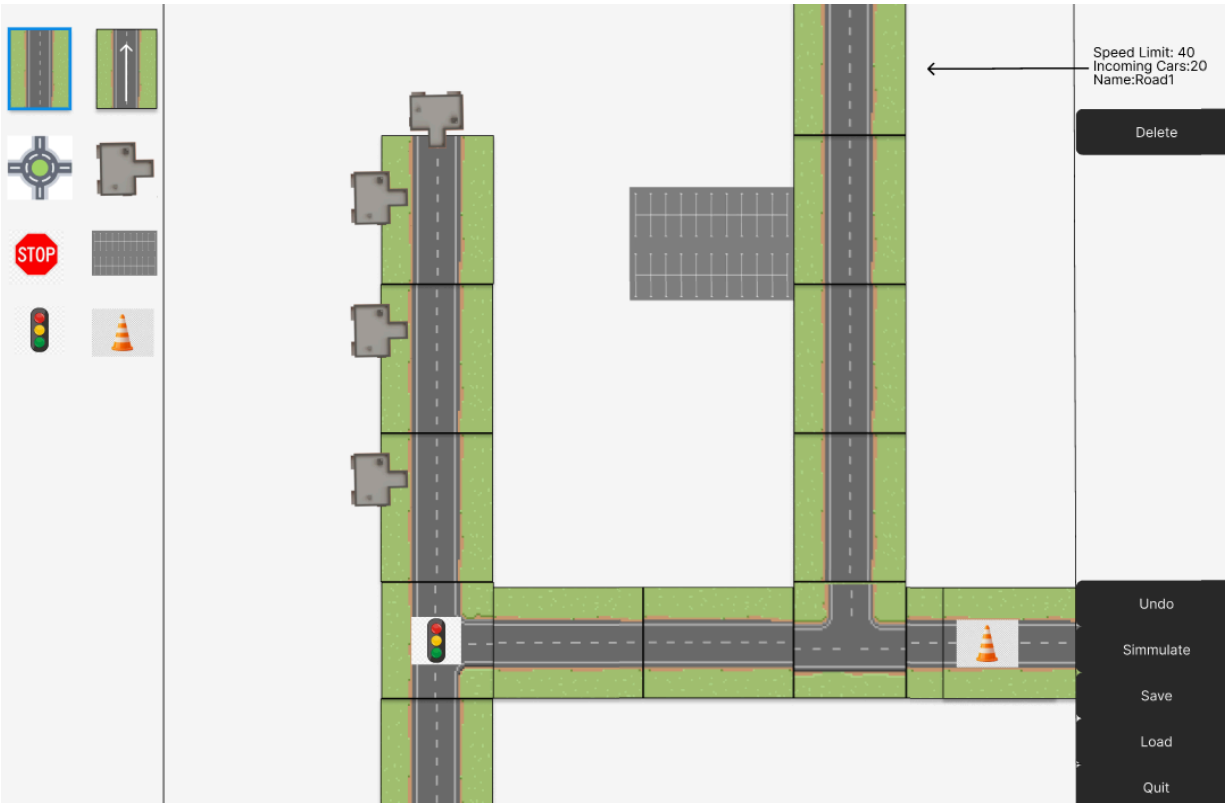
Load Menu

If the user selects Load, this menu is pulled up to allow the user to select which of their saved designs they would like to load.

Name	Date
MyDesign1	2/10/2025
MyDesign2	2/14/2025

Design

This is the main design page. It shows placeable objects on the left bar and allows users to drag and drop them onto the map. When an object is selected, options for that object are pulled up on the right bar, allowing the user to change things like speed limits or stoplight timing. On the bottom right there are buttons to undo an action, start the simulation, save the design, load a design, or quit the program.



Simulate

This is the simulate page. This is where the user goes once the simulate button is pressed. On the left, the program displays live stats such as the number of cars and the average speed of the cars. On the right, we have options such as the speed of the program; stats, which provides the user with statistics regarding the entire simulation; a replay button which restarts the simulation; a back button which takes the user back to the design menu, and a quit button which exits the program.