Kennesaw State University

Department of Computer Science

Object Oriented Modeling and Simulation CS4491 Section 2

Project: Object Oriented Traffic Simulation

Jordan White, Nicholas Powers jwhit144@students.kennesaw.edu

npowers2@students.kennesaw.edu

Contents

[Introduction 4](#_Toc418371360)

[The Model 4](#_Toc418371361)

[Internet packet and vehicle analogy 5](#_Toc418371362)

[Vehicle routing 6](#_Toc418371363)

[Preventing deadlock 6](#_Toc418371364)

[Simulation GUI 7](#_Toc418371365)

[Figure 1: Illustration of the simulation model with LaneQueus (Blue), and Intersections. 5](#_Toc418369792)

# Introduction

The project in question was to create an accurate traffic-based simulation involving intersections, roads, and points of interest. What we had to do was think hard about how to simulate traffic accurately in the most simplistic manner that we could think of. At first I thought that we were going to do this project in OOSimL, but then Nick showed me a program someone made that looked a lot like a basic 3D video game simulation along the lines of the Sim City games (or many Sim City influenced games). After seeing the simulation, we then decided to develop the program with a process based simulation engine compared to discrete-event simulation, which the assignments were built upon. To accomplish the development process, we had to draw models and outlines of how real traffic works, and the program was written in C# using Visual Basic and a game engine called Unity 3D Personal Edition. Development time, along with several delays, took us a month to accomplish.

# The Model

The model contains three types of objects. Those are **LaneQueues**, Vehicles, and Intersections.

The **LaneQueue** is a passive object that represents a section of a road, specifically a single lane. It functions similar to a standard queue data structure. The capacity of the internal queue is dependent on the length of the section of roadway. Also the **LaneQueue** has a *CurrentLength* field that represents the sum of the length of the contained vehicles and the space between them.

The Vehicle is a passive object that represents a road vehicle. At any given time a vehicle is either stored inside a **LaneQueue** or temporarily within an intersection. In the latter case the vehicle is currently turning or switching lanes. When a vehicle is not turning it is either moving or stationary within a lane, in this case the vehicle is stored within a **LaneQueue**.

The **Intersection** is the only active object in the simulation. An **Intersection** can represent any kind of transition in the roadway structure, for example an extra lane is added or an intersection. An intersection represents a node in the user generated street network graph. When the user places a point of interest along the

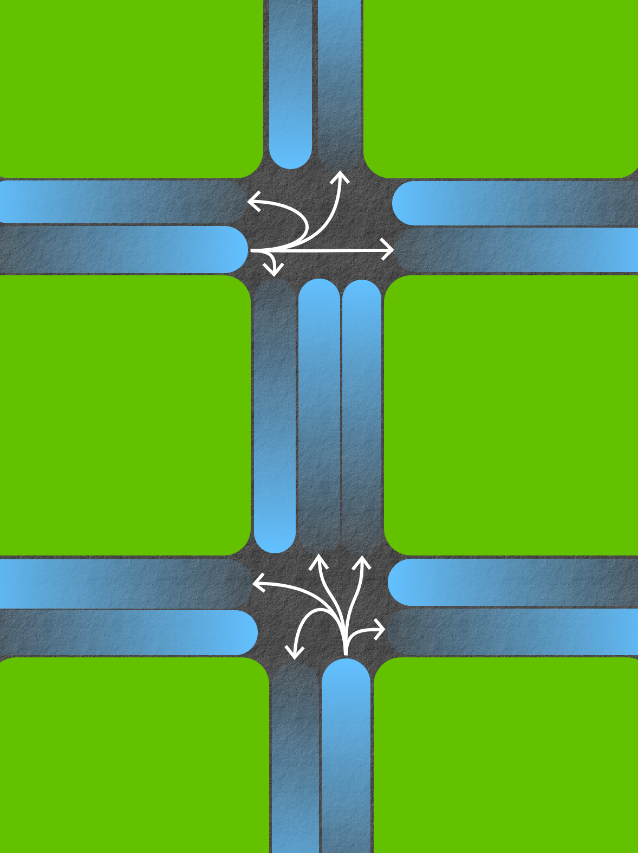


Figure : Illustration of two intersection nodes in a simulation model with LaneQueus (Blue), and Intersections.The white arrows represent the paths vehicles can make

# Internet packet and vehicle analogy

In order to accurately model traffic, vehicles need to navigate as they would in reality. A vehicle’s destination depends on where the vehicle starts. To most efficiently get to its destination a vehicle needs to make a specific set of turns along its way. Similar to how packets in the internet make a specific set of hops between routers, vehicles are routed from one intersection to another.

# Vehicle routing

Finite Automata is used to route vehicles to their destination. Before running the simulation a transition matrix is constructed. Vehicles are routed to minimize the distance travelled. To begin constructing the transition matrix the shortest path between every pair of nodes is first found using the Bellman Ford algorithm. Using this information the transition matrix is constructed.

The columns of the matrix represent the destination, while the rows represent the starting point. To figure out the next node to visit using this matrix simply query the cell at the row that is your current node index, and the column that is the destination’s index. A jagged array is used to store this matrix since not all nodes in the graph will be a destination. To find the value of the ith component in the jth column simply compare the shortest path value of all adjacent nodes to the ith node and pick the one with the smallest value for destination j. The transition function uses this matrix to determine the next node the vehicle will go to.

# Preventing deadlock

In situations where cycles exist within the graph it is possible for vehicles to get deadlocked. To prevent this from happening, a second transition matrix is used to provide a detour. To construct this matrix an algorithm similar to the one used to construct the optimal transition matrix is used, however instead of choosing the node with the smallest shortest path it instead picks the one with the second smallest shortest path.

When a vehicle waits at an intersection for too long the next node is chosen from the second transition matrix. Currently the simulation only supports nodes with a maximum out-degree of 4. That means a total of 4 transition matrices can exist each one representing a better option.

# Simulation GUI

To reduce the complexity of the model construction and modification process a GUI is used to provide tools the user can use to modify the street network. After the user has constructed a high level representation of the model an algorithm will construct the final model. To construct the final model the user’s graph of streets and intersections is transformed into a new graph consisting of **laneQueues**. The new graph is much larger in terms of node and edge count. Each edge contains length information.

A vehicle’s movement is influenced by physics. Exchanging vehicles between intersections needs to be done carefully to accurately model the movement of a vehicle.