City Traffic Simulation in Javascript and WebGL

Aven Bross

5/7/2015

Abstract

In this paper I explore a simulation of streetlights and city traffic. The goal is to create a small section of city with cars passing through that can be manipulated and examined. Traffic density, where cars are going, and where they enter from will vary. By examining build up and statistics, the hope is to be able to establish optimum lane layouts and streetlight timings to minimize traffic halting and clogging.

1 Introduction

The goal of this project is to create a sandbox system to simulate a subsection of city traffic. Cars will enter the subsection with a desired destination and progress through the traffic lights and stop signs. By examining where and how traffic builds up under different conditions, I hope to allow optimization of traffic light timing and city layout within the subsection. The simulation should allow varying where traffic enters, where popular destinations are and the density of traffic. By observing these changing variables over time, optimum timings and layouts can be found to fit traffic pattern changes based on time of day as well.

2 Model

The traffic model I have decided on utilizes discrete grid cells for car movement and routing decisions. Speed and movement will not be appear discrete or jumpy but the underlying system will be ”turn-based” in nature to allow for a simple decsion making system and remove the need for collsion detection.

2.1 City Layout

The city layout shall be simulated as a grid system. Each square shall have a value *direction* and a value *empty*. The *direction* value will be an integer represented cardinal direction indicating whether the given square is a lane going a certain direction, or a null value to represent a piece of an intersection or other non-directioned road. The *empty* value will simply indicate whether the square is open to be moved into. Squares may also be subject to street lights, preventing their occupants from moving on until a reasonable value is reached.

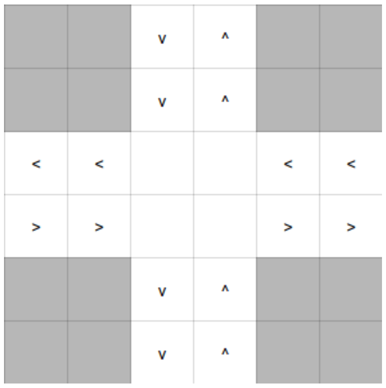
2.2 Cars

Cars in the simulation will be simple constructions that have a *position* in floating point x,y coordinates, a *direction* as an integer in the same format as above, a floating point *speed* value and a possible *destination* in x,y coordinates. Upon a move call the car will move in its forward direction unless obstructed or faced with an intersection or turn possibility.

For the current simulation, destination pathing was ommitted. Since it would be a fairly complicated user interface to allow manipulation of popular destinations and starting points, cars will simply make random decisions as they are faced with movement options.
It should also be noted that decisions to move forward are weighted higher, for example as a car moves into the first squuare of an intersection, it has a $\frac{2}{3}$ chance to go forward, with only $\frac{1}{3}$ chance of making a right turn.

2.3 Simple System

Here is a diagram for a simple intersection. Arrows indicate lane direction, grey squares indicate impassable.



The algorithm for determining possible decisions for a car is outlined in pseudocode below. This will be refined and added to for multiple lanes, stop lights, and many other situations.

```
if forwardSquare.empty
    add possibility MOVE_FORWARD
if rightSquare.direction is right
    add possibility TURN_RIGHT
if leftSquare.direction is left
    add possibility TURN_LEFT
else if leftSquare.direction is none
    while leftSquare.direction is none
        set leftSqare to the next left square over
    if leftSquare.direction is left
        add possibility TURN_LEFT
```

The car would then look at the possibilities and make a decision based upon its desired *destination*.

2.4 Movement

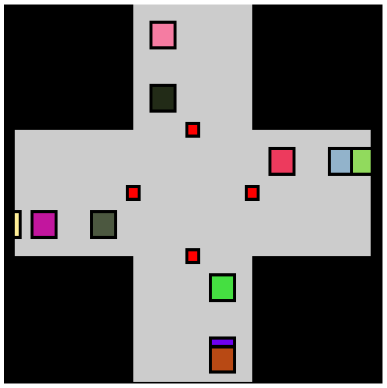
Each car has a speed which will determine speed of movement. When a car makes a movement decision, they choose a destination square and set it as occupied. Over time they will progress to their destination square and will then not make a further decision until such time that they reach their destination state. Upon reaching the quarter-way point, they will mark their previous square as empty. We will assume cars will be maintaining low city speeds and speed will not vary between cars enough to cause collision detection problems with this mehtod.

2.5 Streetlights

Stoplights are modeled as separate entities each with an *x,y* position, an integer *state* and three timing variables *redtime*, *yellowtime* and *greentime*. The *position* denotes the top left corner of the intersection the street light will be managing. There are six possible *state* values that are cycled through in order: green vertical, yellow vertical, red all, green horizontal, yellow horizontal, red all. The amount spent in each state is determined by the timing variables above.
Streetlights manage traffic by marking squares with which direction movement is allowed. Cars in turn incorporate this into their logic by checking the square ahead does not have a stoplight car for its current direction

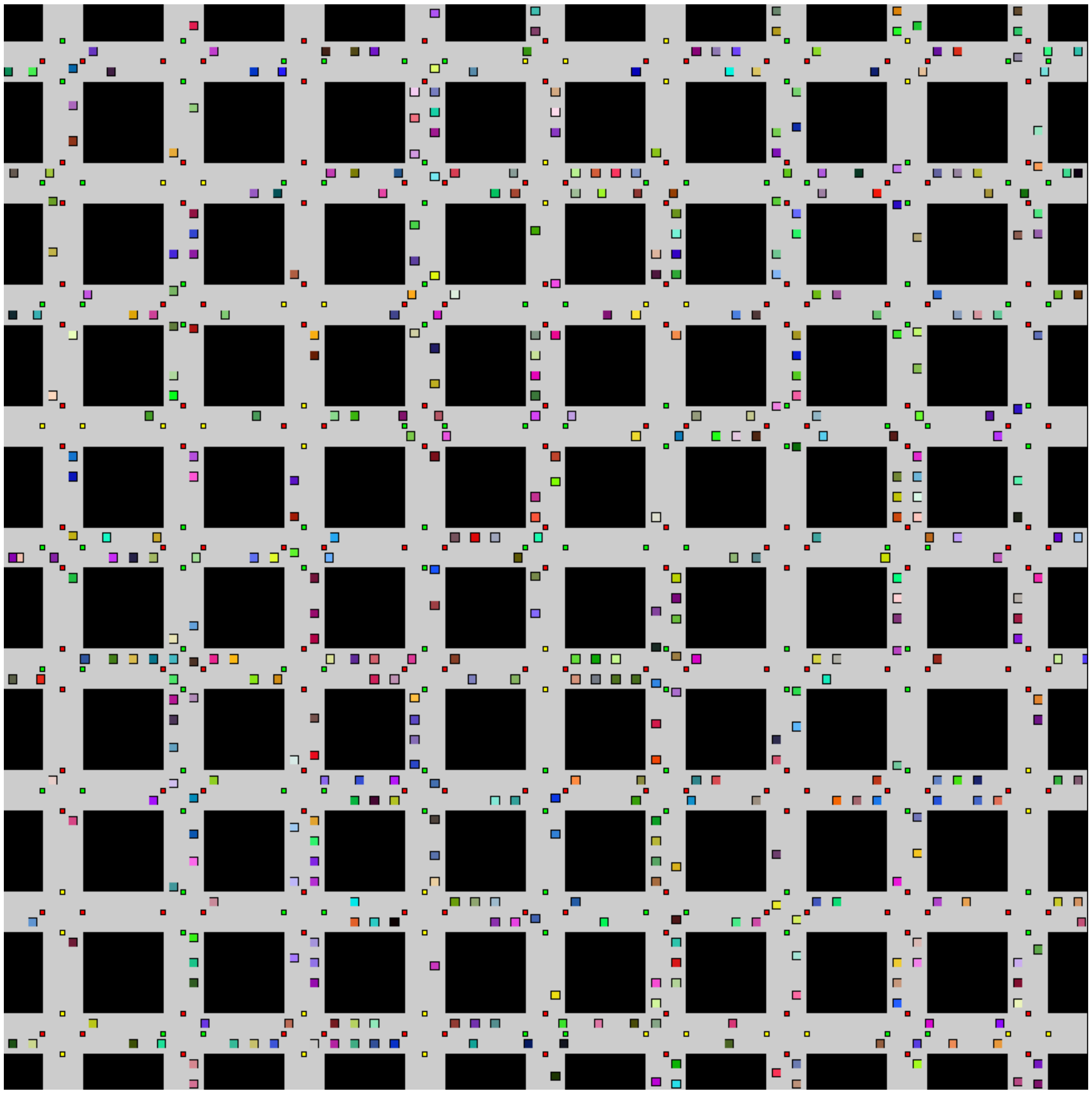
3 Simulations

The city map constructed for the simulation was ultimately decided to be a uniform grid pattern. Each intersection is constructed as per the image below, as a 6 by 6 section of grid cells. The middle four form the intersection, with a stoplight managing them. Directed lanes enter and exit at the proper sides and impassible squares occupying the corners.



By placing multiple of these sections next to each other we create something much like a section of city blocks. Since the user interface to control all stoplights and car destinations would be quite large and a bit daunting to set up, the simulation generates the system randomly. Each stoplight was set to begin at a random one of its six possible states and have it's *greentime* value set between 5.0 and 15.0 seconds. The valeus for *yellowtime* and *redtime* were locked at 3.0 and 1.0 seconds respectively.

Cars are set to randomly enter the system from the edges, with a new car entering every 0.5 seconds. Moving out of the grid elminates a car from the simulation.



4 Conclusions

The current simulation provides quite a nice representation of city traffic, cars make decisions, don't hit one another and pass through roads and intersections in a realistic way. An interesting observation from the simulation is the fact that while cars only enter from the edges and make decisions randomly, the density of cars remains rather uniform accross the entire simulation. Intersections in the center get clogged just as often as those around the edges.

Unfortunately, the simulation does not provide the original goal of a sandbox for manipulating and observing conditions. As a future development, it would be nice to have a user interface for maniuplating all of the variablles in the simulation, as well as a readout of car statistics such as how many lights they hit and the amount of time they stop at lights. It would also be nice to have an editor system to allow building of custom city layouts, to simulate specific situations.

References

[1] Nover, H. (2005). Algebraic Cryptanalysis of AES: An Overview. University of Wisconsin, USA.