
FUEL
ICE

Smart Traffic Grid

Technical Report

by Glenn Cullen - x14729249

Declaration Cover Sheet for Project Submission

SECTION 1 - Student to complete

Name: Glenn Cullen

Student ID: x14729249

Supervisor: Lisa Murphy

SECTION 2 - Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: Glenn Cullen Date: 13/05/2018

Table Of Contents

1. Executive Summary

As we touch upon a world where computers interact with all real world processes ubiquitously and seamlessly, it is time to take the first step in that direction.

We have the ability, using the internet and incredible processing speeds, not to mention incredible algorithms, to give the world a brain. To shit the world to a computer and say, “hey, make that work better.”

It is with this future utopia in mind that I embarked upon this project as a way towards improving traffic conditions during emergency situations making the process safer for all, in a way that would have previously been unheard of.

Today, with the advent of of tools which allow systems to speak over the internet, we are no longer limited to computer languages and hardware and particular systems, but we are moving to a more homogenized network where, even though each node in the system might have different DNA, they have a common language which they can speak to each other with.

I believe the process of using computers ubiquitously should begin with issues of safety so that we can start off utilising the IoT in a positive way.

This Technical Report will show you how this proposed system works.

2. Introduction

2.1. Background

As we begin to truly utilise the power of the internet, there is real opportunity to allow previously ‘dumb’ systems to ‘wake up’ and take control of their own environment and in using modern technology, become experts within their own niche.

In an emergency situation, response time is essential above all else. However, it is largely determined by factors outside the control of the emergency vehicle driver, the control lies with the flow of traffic. It is for this reason that I wanted to apply the principles of the Internet of Things to a traffic light grid spanning an entire environment and managing it when there is an emergency.

The idea is that traffic will be monitored in terms of congestion on the adjacent road and when an emergency takes place, a cloud based AI agent will determine the best route for the emergency vehicle to take and control the traffic lights in order to keep traffic moving continually in the direction of the emergency.

2.2. Aims

The purpose of this implementation is to demonstrate the efficacy of such a system and to prove that it is not only safe, but has a negative correlative effect on traffic congestion overall when the opportunity to save a life is weighed against a the minor inconvenience of your light staying red for longer than usual.

I aimed to create a functioning city simulation where a road network is able to be filled with autonomous vehicles which will be able to drive safely throughout, stop at traffic lights, avoid hitting into the back of each other, know when it’s safe to turn, and stop when an emergency vehicle is en route to an emergency.

I aimed to create an AI in the cloud which will be able to take in information about the state of the road network, as well as the current location of the emergency vehicle and the location of the emergency in progress, and then

use that information to determine the most optimal route from the emergency vehicle to take.

I aimed that the road network itself would update the cloud AI, and that the cloud AI would control the traffic lights during an emergency. Also that the cloud AI would receive information directly from the emergency vehicle as to its position en route to an emergency, and then control the Fire Person facing app to steer the emergency worker in the right direction.

I aimed to have an android app that will be able to display the route to the Fire Person.

2.3. Technologies

2.3.1. Software

- *Unity*
  - Game Engine used to create the simulation of the city in action.
- *Android Studio*
  - Development Environment for developing Android Apps and is used to create the Fire Person facing application to display route to emergency as determined by the cloud AI.
- *Jetbrains Rider Unity IDE*
  - Development Environment for use with C# and is integrated with Unity specific functionality.
- *Jetbrains Pycharm IDE*
  - Development Environment for use with python

2.3.2. Languages

- *C#*
- *Python*
- *Java*

2.3.3. Cloud Communication

- *Amazon Web Services IoT*
  - MQTT communications via the cloud between ‘things’, specifically the Fire Person facing app, and the cloud AI.
- *Pubnub*
  - MQTT communications via the cloud between ‘things’, used for unity to communicate with the cloud AI because AWS does not yet have integration with unity.

2.3.4. Testing

- *Pytest*
  - Test library for python.
- *JUnit*
  - Test library for android.
- *Unity environment*

- Given the nature of objects within the unity game engine, there is no way to carry out unit or integration tests and therefore the environment itself becomes the test suite. This will be explained further below.

3. System Requirements

3.1. Functional requirements

3.1.1. Requirement 1: Initiate Emergency

3.1.1.1. Description

This use case describes the how an emergency situation is initiated in the simulation.

3.1.1.2. Use Case

Scope

The scope of this use case is to initiate an emergency within the simulation.

Flow Description

Precondition

The simulation is running, the cloud AI is connected to the IoT communication software and listening on the appropriate channels and there is no fire currently in progress.

Activation

This use case is activated when the user uses the mouse to click on a building within the simulation and setting it on fire.

Main Flow

1. The simulation is running
2. The cloud AI is connected to comms
3. The cloud AI is subscribed to relevant channel
4. The user selects a building with the mouse
5. The building ignites (see E1)

Exceptional Flow

E1 : Building already on fire

1. There is already a building on fire
2. This building is already on fire
3. Another building is not ignited

Termination

The system terminates the process when a building has been ignited.

Post Condition
The system is informed that there is an emergency taking place.

3.1.2. Requirement 2: Inform Systems of Route

3.1.2.1. Description

This use case describes how the cloud AI informs the other systems of the route to the emergency.

3.1.2.2. Use Case

Scope

The scope of this use case is to inform the other systems of the route that the cloud AI has calculated.

Flow Description

Precondition

The simulation is running, the cloud AI is connected to the IoT communication software and listening on the appropriate channels, the Fire Person facing app is connected to the IoT communication software and listening on the appropriate channels, and there is a fire currently in progress.

Activation

This use case is activate when the cloud AI has determined the best route for the vehicle to take.

Main Flow

1. The cloud AI receives communication containing the location of the emergency and the Fire Brigade
2. The cloud AI calculates the most optimal route
3. The cloud AI publishes the route as a json to the appropriate channels

Termination

This use case is terminated once the routes have been published to the appropriate channels

Post Condition

The cloud AI knows there is a fire and will not accept traffic information until it is resolved, prioritising the emergency vehicle location and knowing when emergency has been resolved.

3.1.3. Requirement 3: Inform of Traffic Information

3.1.3.1. Description

This use case describes how the simulation sends traffic information to the cloud AI.

3.1.3.2. Use Case

Scope

The scope of this use case is to send traffic information from the simulation to the cloud AI.

Flow Description

Precondition

The simulation is running, the cloud AI is connected to the IoT communication software and listening on the appropriate channels, the Fire Person facing app is connected to the IoT communication software and listening on the appropriate channels, and a fire has just been initiated.

Activation

This use case is activated when a fire is initiated.

Main Flow

1. A fire is initiated in the simulation
2. The simulation gathers all the congestion information throughout the whole cityscape
3. The simulation sends this information to the cloud AI through the IoT communication channels

Termination

This use case is terminated when the information is sent to the cloud.

Post Condition

The simulation sends the location information to the cloud; emergency vehicle and emergency.

3.2. Data requirements

3.2.1. Performance / Response Time

The cloud based web service needed to be able to communicate with the system in as fast a time as possible as the AI needs to be able to make changes to the simulation's traffic lights in as close to real time as possible given that a lapse could lead to a failure of the emergency vehicle to reach its destination safely. The AI uses an A* style searching algorithm to determine the best possible route for the emergency vehicle which is an extremely fast path finding search algorithm that can also take in heuristical data. The A* method significantly reduces the scope of the problem as the AI does not consider routes that will definitely not yield the best result, thereby increasing performance.

3.2.2. Maintainability

The system uses two methods of communicating in the cloud, Pubnub and AWS IoT. Although not implemented in this software, in the real world this gives us the opportunity to switch between communication strategies if there is downtime in one tool. Alternatively, we could take advantage of 'shadows' in AWS which continue to receive information when any element of the system goes down, and then brings them up to speed when the element comes back online.

3.3. User Requirements

3.3.1. Security

In any type of communication online, there is a need for data protection, and this includes road traffic information. Access to such information could be considered sensitive. AWS and Pubnub can be setup to require stringent credential information so that only the system with the correct information can subscribe or publish to a channel.

3.4. Environmental Requirements

3.4.1. Cloud IoT Web Services

In order to process data in the cloud, the system communicates using Pubnub and Amazon Web Services. The reason for using a web service such as this is that they offer scalable processing power as well as storage. This will be used to optimise the AI so that the system can receive updates from the AI in the shortest time possible. Another factor is that there can be multiple instances of the AI available for other uses, for testing or for updating safely.

3.4.2. Unity Game Engine

The simulation is build using the unity game engine, this is able to handle many integral processes such as gravity, collision detection, torque and other parts of the physical sciences. This is necessary as it gives us a better and more accurate simulation of autonomous drivers in the simulation and therefore the results we get will tend towards similar to a real world setting.

3.4.3. Extensibility

The Unity engine uses a component based design principle. This differs from OOP insofar as objects are defined by components added to them, and components are interchangeable between objects. In terms of extensibility, if the system were to be updated to include faster cars, all that would need to be done is to take an already existing object and manipulate the current components or write simple components to add to these object, thus making extensibility in the simulated environment easy considering that it is unlikely to break the system.

The A* searching algorithm takes in heuristical data to make its calculation. Adding to the heuristics that the AI is considering to make its decision would be easy and fast. This will make extending the AI's functionality a relatively painless process.

3.4.4. Portability

All systems within the project communicate through the cloud with messages comprised of json. For this reason, adding any other software to the system is as easy as interpreting the json. For this reason, adding other systems to this one would not be dependent on

being able to cross communicate between coding languages or frameworks, and makes this system very portable.

3.5. Usability Requirements

3.5.1. Availability

The availability of the AI is of paramount importance to the running of the system in general, however, if this system were applied to the real world, any downtime would simply default to the current system we have, an imperfect system, but one that will still function.

3.5.2. Recovery

The source code for the system and for the AI is backed up with version control in an online repository. If the connection between the system and the web service is lost in the real world, there should be a system in place to immediately switch from one host to another without the user's knowledge. To try and reduce the downtime to 0, the system could be spread out over a number of cloud Web Service instances which can jump between processors and servers whenever necessary, especially in the case of downtime.

3.5.3. Robustness

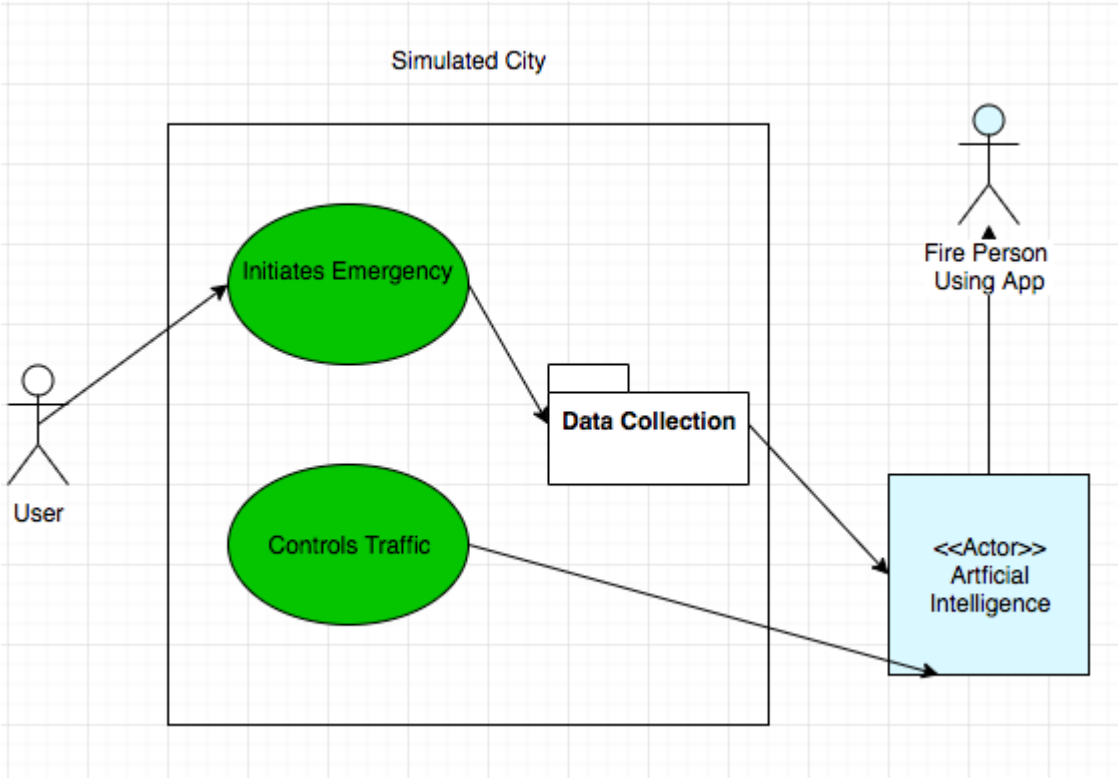
The system is designed in such a way that the user's input is practically negligible, insofar as their actual input is being data that gets picked up by the road sensors. Given the complexity of both the simulated environment and the AI controlling it, the less complex variables in the mix, the better. Having the user as minimally active as possible makes the system more robust as a whole as the AI will be working in an environment entirely familiar to it, and predictable to an extent. The main thing that needs to remain robust is the connection between the system and the AI.

3.5.4. Reusability

The goal of the project was partly to develop an AI that can take in traffic information and be able to understand what is happening in the city as a whole. This could be applicable to any system that needs to understand traffic in depth in order to be functional; city planning being an obvious example.

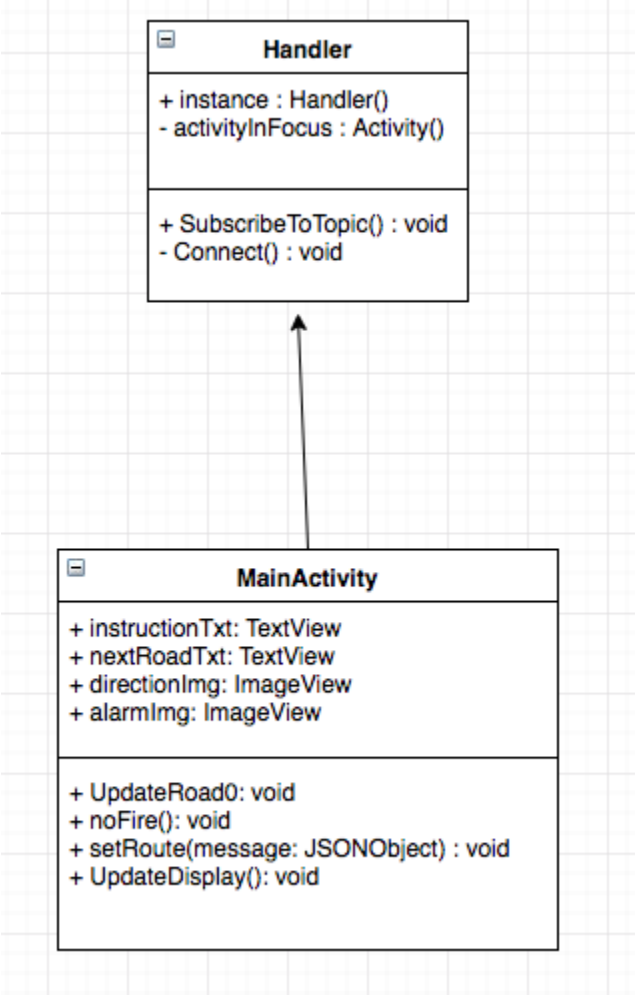
4. Design and Architecture

4.1. System Use Case Diagram



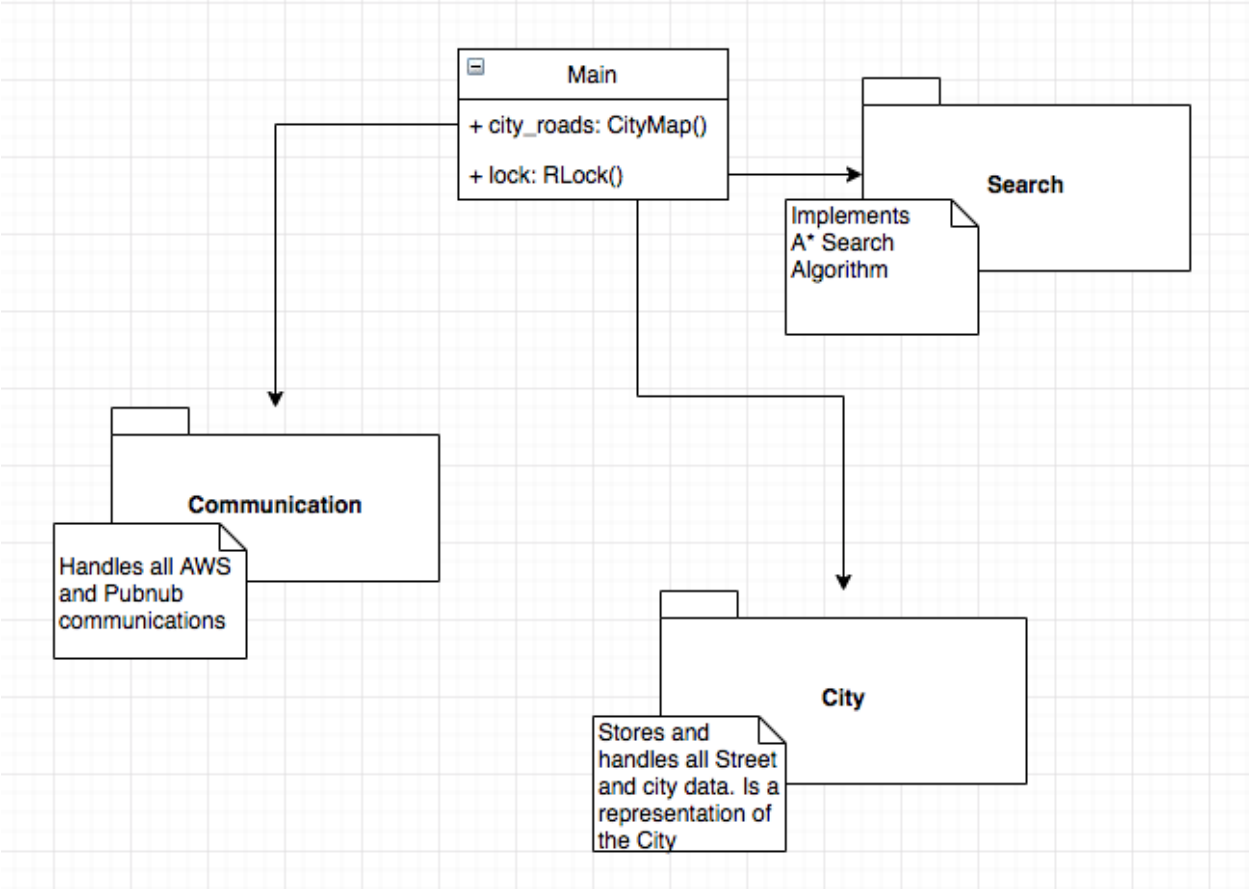
4.1.1. figure

4.2. Android Class Diagram



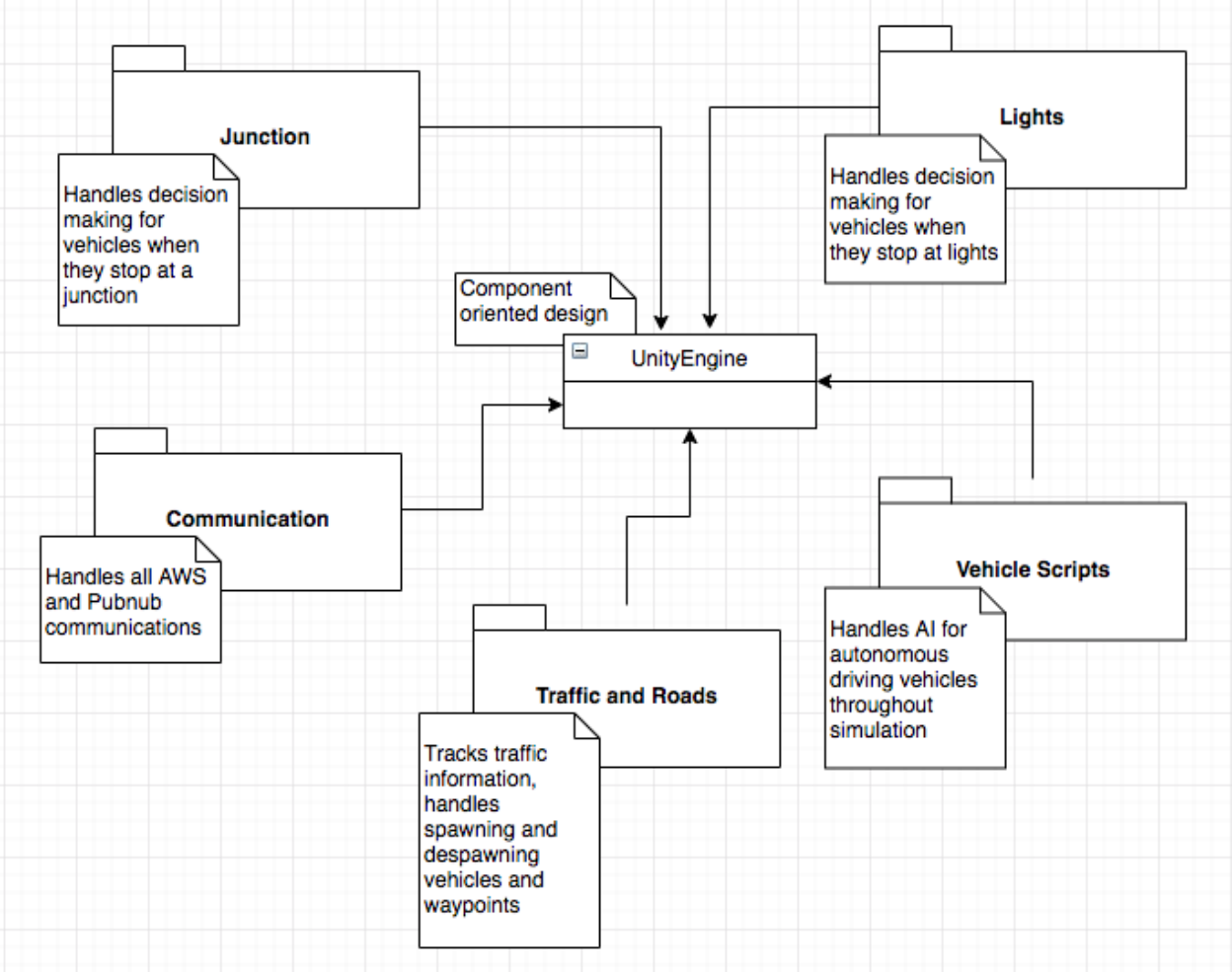
4.2.1. figure

4.3. Cloud AI Class Diagram



4.3.1. figure

4.4. Simulation Class Diagram



4.4.1. figure

5. Implementation

5.1. Python

The implementation of the cloud AI uses a connection with AWS to communicate with the Fire Brigade, and Pubnub to connect with the simulation:

Aws.py

```
myMQTTClient = AWSIoTMQTTClient(str(uuid.uuid1()))  # Add client ID

def connect():
    # set up AWS IoT MQTT
    myMQTTClient.configureEndpoint("a3oazwlb9g85vu.iot.us-east-2.amazonaws.com", 8883)  #
endpoint
    myMQTTClient.configureCredentials(
        "/Users/glennncullen/PycharmProjects/ShmartCity/app/communication/credentials/root.pem",

"/Users/glennncullen/PycharmProjects/ShmartCity/app/communication/credentials/cdbd424344-
private.pem.key",

"/Users/glennncullen/PycharmProjects/ShmartCity/app/communication/credentials/cdbd424344-
certificate.pem.crt")  # Add paths (CA, private key, cert)
    myMQTTClient.configureOfflinePublishQueueing(-1)  # Infinite offline publish queueing
    myMQTTClient.configureDrainingFrequency(2)  # Draining: 2 Hz
    myMQTTClient.configureConnectDisconnectTimeout(10)  # Disconnect at 10 seconds
    myMQTTClient.configureMQTTOperationTimeout(5)  # Operation timeout 5 seconds
    # connect to AWS IoT MQTT
    try:
        return myMQTTClient.connect()
    except AWSIoTExceptions:
        print("unable to connect MQTT")
        traceback.print_exc()
```

Pubnub_handler.py

```
class PubNubHandler:
    def __init__(self):
        self.pnconfig = PNConfiguration()
        self.pnconfig.subscribe_key = 'sub-c-ec33873a-53d1-11e8-84ad-b20235bcb09b'
        self.pnconfig.publish_key = 'pub-c-56bfd71d-e6e9-479d-9c08-b2c719d6a4c7'
        self.pnconfig.secret_key = 'sec-c-
OWY1ZDU0NGUtN2IyZC00YmJmLWFmNTEtOTc3NDFkYWE0YjUw'
        self.pubnub = PubNub(self.pnconfig)
        self.my_channels = [
            'all-roads',
            'update-congestion',
            'fire-in-progress',
            'update-position',
            'fire-extinguished'
        ]
        self.connected = False
        self.Subscribe()

    def Subscribe(self):
        callback = SCSubscribeCallback()
        self.pubnub.add_listener(callback)
        self.pubnub.subscribe().channels(self.my_channels).execute()
        while not callback.subscribed:
            if callback.failed:
                break
        self.connected = callback.subscribed

    def Publish(self, message, channel):
        if self.connected:
            if self.pubnub.publish().channel(channel).message(message).async(my_publish_callback):
                print("pubbed %s to %s" % message, channel)
        else:
            print("cannot publish while not connection -- trying to reconnect")
            self.pubnub.reconnect()


class SCSubscribeCallback(SubscribeCallback):

    def __init__(self):
        self.subscribed = False
        self.failed = False
        self.fire_in_progress = False

    def presence(self, pubnub, presence):
        pass  # must implement abstract method

    def status(self, pubnub, status):
        if status.category == PNStatusCategory.PNUnexpectedDisconnectCategory:
            self.failed = True
            self.subscribed = False
```

```
        elif status.category == PNStatusCategory.PNConnectedCategory:
            self.subscribed = True
            self.failed = False
        elif status.category == PNStatusCategory.PNReconnectedCategory:
            self.subscribed = True
            self.failed = False

    def message(self, pubnub, message):
        # receive all roads
        if message.channel == 'all-roads':  # {num: {road details}, num {road details} ... }
            print message.channel, ": ", message.message
            main.build_city(message.message)
        # receive updated congestion
        if message.channel == 'update-congestion':  # message: {road: name, congestion:
congestion}
            if not self.fire_in_progress:
                print message.channel, ": ", message.message
                main.update_congestion(message.message)
        # alert that a fire is in progress
        if message.channel == 'fire-in-progress':  # message: {start: road, end: road}
            print message.channel, ": ", message.message
            self.fire_in_progress = True
            response = main.calculate_best_route(message.message)
            for element in response:
                print element
            pubnub.publish().channel('route-to-fire').message(response).async(my_publish_callback)
            aws.publish(response, '/shmartcity/route/')
        # ambulance position has been updated
        if message.channel == 'update-position':  # message: {next: true}
            print message.channel, ": ", message.message
            aws.publish(message.message, '/shmartcity/nextroad/')
        # fire has been extinguished
        if message.channel == 'fire-extinguished':  # message: {extinguished: true}
            print message.channel, ": ", message.message
            aws.publish(message.message, '/shmartcity/extinguished/')
            self.fire_in_progress = False


PubNubHandler()
```

This code defines what will happen when the channel receives a message. In some cases, it will build the entire city of Street Nodes:

```
from math import sqrt


class StreetNode:
    def __init__(self, name, lights, position, congestion):
        self.name = name
        self.straight = None
        self.left = None
        self.right = None
        self.lights = lights
        self.congestion = congestion
        self.position = position
        self.max_travel_distance = 586.1679

    def to_a_string(self):
        s1 = "name: %s" % self.name
        if self.straight is not None:
            s2 = "straight: %s" % self.straight.name
        else:
            s2 = "None"
        if self.left is not None:
            s3 = "left: %s" % self.left.name
        else:
            s3 = "None"
        if self.right is not None:
            s4 = "right: %s" % self.right.__class__.__name__
        else:
            s4 = "None"
        s5 = "lights: %s" % str(self.lights)
        s6 = "congestion: %s" % str(self.congestion)
        s7 = "position: %s" % self.position
        return s1, s2, s3, s4, s5, s6, s7

    def get_connected(self):
        connected = []
        if self.straight is not None:
            connected.append(self.straight)
        if self.left is not None:
            connected.append(self.left)
        if self.right is not None:
            connected.append(self.right)
        return connected

    def cost_to_road(self, road):
        return int(
            (road.congestion / 7 * 200)
            + (self.calculate_distance(self.position, road.position) * 1000)
        )

    def calculate_distance(self, pos1, pos2):
```

```
        # distance between two vectors
        return sqrt((pow((pos1["x"] - pos2["x"]), 2) + pow((pos1["y"] - pos2["y"]), 2))) /
self.max_travel_distance
        # expressed as a number between 0 and 1
        # print((sqrt((pow((pos1["x"] - pos2["x"]), 2) + pow((pos1["y"] - pos2["y"]), 2)))) /
main.max_travel_distance)
```

Or update the congestion in a particular Street Node:

```
def update_congestion(self, road, congestion):
    if road not in self.city_roads:
        return;
    self.city_roads[road].congestion = congestion
```

Or call the A* search to calculate the best route:

```
def a_star_search(start, end):
    open_list = [start]
    came_from = {start: None}
    total_running_cost = {start: 0}
    path = []

    while len(open_list) > 0:
        current_road = open_list[0]
        open_list.remove(current_road)
        if current_road == end:
            break
        for next_road in current_road.get_connected():
            cost = total_running_cost[current_road] + current_road.cost_to_road(next_road)
            current_road.cost_to_goal = cost
            if next_road not in total_running_cost or cost < total_running_cost[next_road]:
                total_running_cost[next_road] = cost
                open_list.append(next_road)
                open_list.sort(key=lambda x: x.cost_to_road, reverse=True)
                came_from[next_road] = current_road

    path_node = end
    while path_node != start:
        path.append(path_node.name)
        path_node = came_from[path_node]
    path.append(start.name)
    path.reverse()

    return path
```

5.2. Android

There are only two classes in the android app, a singleton that handles communication with AWS and MainActivity.

The Handler method SubscribeToTopic defines what to do when a particular message is received:

```
private static void subscribeToTopic(String topic){
  try {
     myMQTTManager.subscribeToTopic(topic, AWSIotMqttQos.QOS0, new
AWSIotMqttNewMessageCallback() {
        @Override
        public void onMessageArrived(final String topic, final byte[] data) {
           activityInFocus.runOnUiThread(new Runnable() {
              @Override
              public void run() {
                 try {
                    JSONObject receivedJson = null;
                    try {
                       receivedJson = new JSONObject(new String(data, "UTF-8"));
                       if(receivedJson.has("path")){
                          ((MainActivity) activityInFocus).setRoute(receivedJson);
                       }
                       else if (receivedJson.has("next")){
                          ((MainActivity) activityInFocus).updateRoad();
                       }else if(receivedJson.has("extinguished")){
                          ((MainActivity) activityInFocus).noFire();
                       }
                       String jsonString = receivedJson.toString();
                       Log.i(LOG_TAG, "Received:\t" + jsonString);
                    } catch (JSONException e) {
                       Log.e(LOG_TAG, "Error creating Json for topic: " + topic, e);
                    }
                 } catch (UnsupportedEncodingException e) {
                    Log.e(LOG_TAG, "Received message encoding error", e);
                 }
              }
           });
        }
     });
  } catch (Exception e) {
     Log.e(LOG_TAG, "unable to subscribe to: " + topic, e);
  }
}
```

In the MainActivity, there is a method to set the route:

```
public void setRoute(JSONObject message){
  try {
     route = (JSONArray) message.get("path");
     String currentRoad = (String) route.get(0);
     String nextRoad = (String) route.get(1);
     String nextDirection = getNextDirection(currentRoad.charAt(currentRoad.length()-2),
                                    nextRoad.charAt(nextRoad.length()-2));

alarmImg.setImageDrawable(getApplicationContext().getDrawable(R.drawable.alarm_green));
     updateDisplay(nextRoad, nextDirection);
     route.remove(0);
     Log.i(LOG_TAG, route.get(0).getClass().getSimpleName());
  } catch (JSONException e) {
     Log.e(LOG_TAG, "Unable to decode path json object");
  }
}
```

To update the route to the next Road:

```
public void updateRoad(){
  try {
     String currentRoad = (String) route.get(0);
     String nextRoad = (String) route.get(1);
     String nextDirection = getNextDirection(currentRoad.charAt(currentRoad.length()-2),
nextRoad.charAt(nextRoad.length()-2));
     route.remove(0);
     updateDisplay(nextRoad, nextDirection);
  } catch (JSONException e) {
     e.printStackTrace();
  }
}
```

To figure out what direction to take at the next turn:

```
public String getNextDirection(char from, char to){
  switch (from){
    case 'N':
      if(to == 'N') return "straight";
      if(to == 'E') return "right";
      if(to == 'W') return "left";
      break;
    case 'S':
      if(to == 'S') return "straight";
      if(to == 'W') return "right";
      if(to == 'E') return "left";
      break;
    case 'E':
      if(to == 'E') return "straight";
      if(to == 'N') return "left";
      if(to == 'S') return "right";
      break;
    case 'W':
      if(to == 'W') return "straight";
      if(to == 'S') return "left";
      if(to == 'N') return "right";
      break;
  }
  return "";
}
```

And to set it back to default with No fire:

```
public void noFire(){
  instructionTxt.setText("Waiting for something to go on fire");
  nextRoadTxt.setText("");
  directionImg.setImageDrawable(null);
  alarmImg.setImageDrawable(getApplicationContext().getDrawable(R.drawable.alarm_red));
}
```

5.3. Unity

The class VehicleBehaviour.cs is the main class that helps the car drive. In order to make the car move forward or to stop, we must do things like apply Torque:

```
private void Move()
{
 _currentSpeed = 2 * Mathf.PI * WheelFrontLeft.radius * WheelFrontLeft.rpm * 60 / 1000;
  if (_currentSpeed < _speedConstant && (!_isBraking || !StopVehicle || !EmergencyBrake))
  {
    WheelFrontLeft.motorTorque = MaxTorque;
    WheelFrontRight.motorTorque = MaxTorque;
  }
   else
  {
    WheelFrontLeft.motorTorque = 0f;
    WheelFrontRight.motorTorque = 0f;
  }
}
```

Or apply Brake Torque:

```
private void CheckBraking()
{
  WheelBackLeft.brakeTorque = _brakeTorqueConstant;
  WheelBackRight.brakeTorque = _brakeTorqueConstant;
  WheelFrontLeft.brakeTorque = _brakeTorqueConstant;
  WheelFrontRight.brakeTorque = _brakeTorqueConstant;
}
```

And to check which way to angle the wheels we can use:

```
private void CheckSteerAngle()
{
 Vector3 relativeVector =
transform.InverseTransformPoint(_pathNodes[_currentPathNode].position);
 relativeVector /= relativeVector.magnitude;
 float angle = MaxSteerAngle + Vector3.Distance(transform.position,
 _pathNodes[_currentPathNode].position);
 float turnAngle = (relativeVector.x / relativeVector.magnitude) * MaxSteerAngle;
 _targetSteerAngle = turnAngle;
}
```

In order to reduce speed we can combine the move and brake functions:

```
private void ReduceSpeed()
{
 if (!CompareTag("firebrigade") || Handler.IsSomethingOnFire) return;
 if (!(_speedConstant > MaxSpeed + 5)) return;
 _isBraking = true;
 _brakeTorqueConstant = MaxBrakeTorque;
}
```

In order for the vehicle to know where it’s going next, we set the next road which isn’t built until the lights turn green, so that if there are any changes in circumstance, the vehicle will stay put:

```
public void SetNextRoad()
{
 Dictionary<String, Transform> dict;
 dict = CompareTag("firebrigade") ?
_currentRoad.GetComponent<WaypointPath>().GetNextRandomWaypointPathForFirebrigade() :
_currentRoad.GetComponent<WaypointPath>().GetNextRandomWaypointPath();
 String[] roadChoice = dict.Keys.ToArray();
 NextRoad = dict[roadChoice[0]];
 LeftCross = false;
 RightCross = false;
 LeftJunctionJoin = false;
 RightJunctionJoin = false;
 LeftJunctionLeave = false;
 RightJunctionCrossing = false;
 IsGoingStraightAtCross = false;
 IsGoingStraightAtJunction = false;
 IsUnableToMove = false;
 switch (roadChoice[0])
 {
    case "straight-cross":
        IsGoingStraightAtCross = true;
        break;
    case "left-cross":
        LeftCross = true;
```

```
            break;
        case "right-cross":
            RightCross = true;
            break;
        case "straight-junction":
            IsGoingStraightAtJunction = true;
            break;
        case "left-junction-join":
            LeftJunctionJoin = true;
            break;
        case "right-junction-join":
            RightJunctionJoin = true;
            break;
        case "left-junction-leave":
            LeftJunctionLeave = true;
            break;
        case "right-junction-crossing":
            RightJunctionCrossing = true;
            break;
        case "despawn":
            GetComponentInParent<TrafficDensity>().Despawn(gameObject);
            break;
        case "cant-move":
            IsUnableToMove = true;
            break;
        default:
            print("SetNextRoad switch statement:\t" + gameObject.name);
            Debug.Break();
            break;
    }
}
```

This method also calculates which way the vehicle will travel be it straight, left or right, turning right will require vastly different observations that going straight or taking a left, and depending on whether we're at a lights, or a junction will matter too.

When we're ready to move, we Build the next path and the vehicle heads towards the waypoint:

```
public void BuildNextPath()
{
 _previousRoad = _currentRoad;
 _previousRoad.GetComponent<WaypointPath>().DecreaseCongestion();
 _currentRoad = NextRoad;
 _currentRoad.GetComponent<WaypointPath>().IncreaseCongestion();
  NextRoad = null;
  Transform[] pathTransforms = _currentRoad.GetComponentsInChildren<Transform>();
 _pathNodes = new List<Transform>();
  foreach(Transform waypoint in pathTransforms){
     if (_currentRoad.transform != waypoint)
     {
         _pathNodes.Add(waypoint);
     }
  }
 _currentPathNode = 0;
}
```

Sometimes, when the Fire Brigade is travelling very fast, it needs to calculate a brake torque specific to the distance it needs to go. I wasn't able to find code for this, but I did find the actual real world equation in physics and was able to code it in using some of the information available through Unity's Physics engine and it worked exceptionally:

```
private float CalculateBrakeTorque(float distance)
{
  return
     0.5f *
     _rigidbodyComponent.mass *
     (
         (float) Math.Pow(_rigidbodyComponent.velocity.x, 2) +
         (float) Math.Pow(_rigidbodyComponent.velocity.y, 2) +
         (float) Math.Pow(_rigidbodyComponent.velocity.z, 2)
     ) /
     distance;
}
```

A set of special instructions are necessary for when the Fire Brigade is trying to drive towards an emergency which tell it how fast to travel and whether or not it's safe to move forward:

```
private void EmergencyDriving()
{
  if (IsGoingStraightAtCross)
  {
     if (_previousRoad != null)
     {
        if (_previousRoad.gameObject.name.Substring(0, 5) ==
_currentRoad.gameObject.name.Substring(0, 5) &&
            _currentRoad.GetComponent<WaypointPath>().Congestion == 1)
        {
           _speedConstant = MaxSpeed * 2;
        }
        else
        {
           _speedConstant = MaxSpeed;
        }
     }
     else
     {
        _speedConstant = MaxSpeed;
     }
  }
  else if(_currentRoad.GetComponent<WaypointPath>().Congestion == 1)
  {
     _speedConstant = MaxSpeed * 2;
  }
  else
  {
     _speedConstant = MaxSpeed;
  }
  if (Vector3.Distance(transform.position, _pathNodes[_currentPathNode].position) < 15 &&
        Vector3.Distance(transform.position, _pathNodes[_currentPathNode].position) > 5)
  {
     if(_currentSpeed > 50 &&
        _pathNodes[_currentPathNode].GetComponent<Waypoint>().IsLastOnRoad &&
        Handler.Instance.LookAhead().Substring(0, 5) !=
_currentRoad.gameObject.name.Substring(0, 5))
     {
        _brakeTorqueConstant = CalculateBrakeTorque(Vector3.Distance(transform.position,
 pathNodes[_currentPathNode].position));
     }

     if (_currentRoad.GetComponent<WaypointPath>().TrafficLights == null) return;
     if (_currentRoad.GetComponent<WaypointPath>().TrafficLights.GetAllRed())
     {
        _brakeTorqueConstant = CalculateBrakeTorque(Vector3.Distance(transform.position,
 pathNodes[_currentPathNode].position));
     }
  }
```

```
  else
  {
      _brakeTorqueConstant = _isBraking ? MaxBrakeTorque : 0f;
  }
}
```

There are many collider Triggers in Unity which react when they are entered, normally the presence of a vehicle can tell the simulation that it is unsafe to turn, or it may stop the vehicle until it is safe to move. One such of these triggers is the siren, which surrounds the Fire Brigade and in a sphere and is only active when there is a fire. It tells other cars to stop in their tracks, unless they are in the middle of a turn, or on the same path as the Fire Brigade:

```
private void OnTriggerEnter(Collider other)
{
  if (!Handler.IsSomethingOnFire) return;
  if (other.gameObject.GetComponent<CarFrontCollider>() == null) return;
  VehicleBehaviour vehicle = other.gameObject.GetComponentInParent<VehicleBehaviour>();
  if(vehicle == null) return;
  if (Handler.Path.Contains(vehicle._currentRoad.GetComponent<WaypointPath>())) return;
  if(vehicle._isBraking) return;
  if (vehicle.IsGoingStraightAtJunction || vehicle.RightJunctionCrossing ||
      vehicle.RightJunctionJoin || vehicle.LeftJunctionJoin ||
      vehicle.LeftJunctionLeave || vehicle.LeftCross || vehicle.RightCross ||
      vehicle.IsGoingStraightAtCross) return;
  vehicle.EmergencyBrake = true;
  vehicle._brakeTorqueConstant = vehicle.MaxBrakeTorque;
  foreach (JunctionLane lane in vehicle.JunctionLanes)
  {
      lane.TrafficInLane = false;
  }
}

private void OnTriggerExit(Collider other)
{
  if (!Handler.IsSomethingOnFire) return;
  VehicleBehaviour vehicle = other.gameObject.GetComponentInParent<VehicleBehaviour>();
  if(vehicle == null) return;
  if (Handler.Path.Contains(vehicle._currentRoad.GetComponent<WaypointPath>())) return;
  if(vehicle._isBraking) return;
  vehicle.EmergencyBrake = false;
  vehicle._brakeTorqueConstant = 0;
  foreach (JunctionLane lane in vehicle.JunctionLanes)
  {
      lane.TrafficInLane = true;
  }
}
```

When a car is at a junction, it gets added to a list and the list is checked each frame. If it is safe for the vehicle to progress, it is set to vehicle.Continue() or else it will be set to vehilcle.Stop()

```
private void MoveVehicles()
{
  for(int i = _vehiclesTurning.Count-1; i >= 0; i--)
  {
    VehicleBehaviour vehicle = _vehiclesTurning[i];
    if (Handler.IsSomethingOnFire && vehicle.CompareTag("firebrigade"))
    {
      vehicle.SetNextRoad();
      vehicle.Continue();
      _vehiclesTurning.Remove(vehicle);
      continue;
    }
    if (vehicle.NextRoad == null)
    {
      vehicle.SetNextRoad();
    }

    if (vehicle.NextRoad != null)
    {
      if (vehicle.NextRoad.GetComponent<WaypointPath>().GetCongestion() >
          vehicle.NextRoad.GetComponent<WaypointPath>().CongestionThreshold)
      {
        vehicle.SetNextRoad();
      }
    }

    if (vehicle.IsUnableToMove)
    {
      vehicle.Stop();
    }
    else if (vehicle.LeftJunctionLeave || vehicle.IsGoingStraightAtJunction)
    {
      _vehiclesTurning.Remove(vehicle);
    }
    else if (vehicle.RightJunctionCrossing)
    {
      if (_rightLane.TrafficInLane)
      {
        vehicle.Stop();
      }
      else
      {
        vehicle.Continue();
        _vehiclesTurning.Remove(vehicle);
      }
    }
    else if (vehicle.RightJunctionJoin)
    {
      if (_rightLane.TrafficInLane || _leftLane.TrafficInLane)
```

```
            {
                vehicle.Stop();
            }
            else
            {
                vehicle.Continue();
                _vehiclesTurning.Remove(vehicle);
            }
        }
        else if (vehicle.LeftJunctionJoin)
        {
            if (_rightLane.TrafficInLane)
            {
                vehicle.Stop();
            }
            else
            {
                vehicle.Continue();
                _vehiclesTurning.Remove(vehicle);
            }
        }
    }
}
```

Similarly, with the Lights cross sections, the vehicles are added to a list and checked each frame. They will not be checked if the lights are red and will only be able to continue based on certain conditions:

```
private void MoveTrafficOnX()
{
  int vehiclesTurningRight = 0;
  for(int i = _vehiclesOnX.Count-1; i >= 0; i--)
  {
     VehicleBehaviour vehicle = _vehiclesOnX[i];
     if (vehicle.NextRoad == null && !vehicle.IsUnableToMove)
     {
        vehicle.SetNextRoad();
     }
     if (vehicle.NextRoad != null)
     {
        if (vehicle.NextRoad.GetComponent<WaypointPath>().GetCongestion() >
            vehicle.NextRoad.GetComponent<WaypointPath>().CongestionThreshold)
        {
           vehicle.SetNextRoad();
        }
     }
     if (vehicle.IsUnableToMove)
     {
        vehicle.Stop();
     }
     if (!vehicle.RightCross)
     {
        foreach (LightStopX lightStop in GetComponentsInChildren<LightStopX>())
        {
           if (!ReferenceEquals(lightStop.VehicleAtLight, vehicle)) continue;
           if ((vehicle.IsGoingStraightAtCross && !lightStop.StraightOn.TrafficInLane &&
!lightStop.Front.TrafficInLane)
              || (vehicle.LeftCross && !lightStop.LeftTurn.TrafficInLane &&
!lightStop.Front.TrafficInLane)
              || (vehicle.LeftJunctionJoin && !lightStop.LeftTurn.TrafficInLane)
              || (vehicle.RightJunctionJoin && !lightStop.RightTurn.TrafficInLane)
           )
           {
              vehicle.Continue();
              _vehiclesOnX.Remove(vehicle);
           }
           else
           {
              vehicle.Stop();
           }
        }
     }
     else
     {
        foreach (LightStopX lightStop in GetComponentsInChildren<LightStopX>())
        {
```

```
            if (ReferenceEquals(lightStop.VehicleAtLight, vehicle) &&
!lightStop.CrossLane.TrafficInLane
                && !lightStop.RightTurn.TrafficInLane)
            {
                vehicle.Continue();
                _vehiclesOnX.Remove(vehicle);
                break;
            }
            if (ReferenceEquals(lightStop.VehicleAtLight, vehicle) && lightStop.CrossLane.TrafficInLane
                                                && !lightStop.RightTurn.TrafficInLane)
            {
                vehiclesTurningRight++;
                vehicle.Stop();
            }
            else
            {
                vehicle.Stop();
            }
        }
    }
}

  if (vehiclesTurningRight != 2) return;
  for(int i = _vehiclesOnX.Count-1; i >= 0; i--)
  {
     VehicleBehaviour vehicle = _vehiclesOnX[i];
     vehicle.Continue();
     _vehiclesOnX.Remove(vehicle);
  }
}
```

When checking if a vehicle can move during an emergency, only the exact lane is checked against the light it's at, rather than the direction:

```
private void CheckMoveInEmergency()
{
  for (int i = _vehiclesOnZ.Count - 1; i >= 0; i--)
  {
    VehicleBehaviour vehicle = _vehiclesOnZ[i];
    foreach (LightStopZ lightStop in GetComponentsInChildren<LightStopZ>())
    {
      if (ReferenceEquals(lightStop.VehicleAtLight, vehicle) &&
          lightStop.gameObject.name.Equals("East to West") &&
          zEast.greenLight.activeSelf)
      {
        vehicle.SetNextRoad();
        vehicle.Continue();
        _vehiclesOnZ.Remove(vehicle);
        break;
      }
      if (ReferenceEquals(lightStop.VehicleAtLight, vehicle) &&
          lightStop.gameObject.name.Equals("West to East") &&
          zWest.greenLight.activeSelf)
      {
        vehicle.SetNextRoad();
        vehicle.Continue();
        _vehiclesOnZ.Remove(vehicle);
        break;
      }
      vehicle.Stop();
    }
  }
```

In the Building class, when a building is selected, the fire objects are set to go off, the location of the Fire Brigade and the building are also sent via PubNub to the cloud AI:

```
private void OnMouseDown()
{
  if(Handler.IsSomethingOnFire) return;
  bool checkIsNull = true;
  foreach (FireBaseScript fire in _fires)
  {
    if (fire != null)
    {
      checkIsNull = false;
      break;
    }
  }
  if (checkIsNull) return;
  foreach (WaypointPath path in
GameObject.Find("Roads").GetComponentsInChildren<WaypointPath>())
```

```
  {
    path.NotifyCongestionChange();
  }
  Dictionary<string, object> message = new Dictionary<string, object>();
  message.Add("start",
GameObject.Find("Firebrigade").GetComponent<VehicleBehaviour>()._currentRoad.gameObject.n
ame);
  message.Add("end", ConnectedRoad.gameObject.name);
  Handler.Instance.PublishMessage("fire-in-progress", message);
  Handler.BuildingOnFire = this;

  foreach (FireBaseScript fire in _fires)
  {
    Instantiate(Explosion, fire.transform.position, Quaternion.LookRotation(fire.transform.forward),
fire.transform);
    fire.StartParticleSystems();
  }
}
```

A singleton is used for all communications with pubnub:

```
public static Handler Instance
{
  get
  {
    if (_instance != null) return _instance;
    Connect();
    _instance = new Handler();
    return _instance;
  }
}

private static void Connect()
{
  _pnConfiguration.SubscribeKey = "sub-c-ec33873a-53d1-11e8-84ad-b20235bcb09b";
  _pnConfiguration.PublishKey = "pub-c-56bfd71d-e6e9-479d-9c08-b2c719d6a4c7";
  _pnConfiguration.SecretKey = "sec-c-
OWY1ZDU0NGUtN2IyZC00YmJmLWFmNTEtOTc3NDFkYWE0YjUw";
  _pnConfiguration.LogVerbosity = PNLogVerbosity.BODY;
  _pnConfiguration.UUID = "shmart-city-unity";

  _pubnub = new PubNub(_pnConfiguration);

  _pubnub.SusbcribeCallback += Callback;

  // subscribe to this channels
  _channels = new List<string>()
  {
    "route-to-fire"
  };
  _pubnub.Subscribe().Channels(_channels).Execute();
}
```

The Spawn Point object checks whether the current traffic density is below the wanted traffic density every few seconds, and if it is, it will spawn a car on one of the random spawn points:

```
private IEnumerator SpawnCars()
{
  while (true)
  {
    if (_currentDensity < Density)
    {
      List<SpawnPoint> shuffledRoads = SpawnPoints.ToList();
      while (_currentDensity < Density && shuffledRoads.Count > 0)
      {
        SpawnPoint spawn = shuffledRoads[Random.Range(0, shuffledRoads.Count - 1)];
        if (spawn.IsOccupied)
        {
          shuffledRoads.Remove(spawn);
          continue;
        }
        Transform vehicle = VehiclesToSpawn[Random.Range(0, VehiclesToSpawn.Count)];
        vehicle.GetComponent<VehicleBehaviour>().StartingRoad = spawn.Road;
        spawn.Road.GetComponent<WaypointPath>().IncreaseCongestion();
        vehicle.name = "ID: " + _name;
        _name++;
        Transform[] waypoints = spawn.Road.GetComponent<WaypointPath>().GetWaypoints();
        Vector3 rotation = waypoints[waypoints.Length - 1].transform.position -
waypoints[0].transform.position;
        Instantiate(vehicle, spawn.transform.position, Quaternion.LookRotation(rotation), transform);
        _currentDensity++;
        shuffledRoads.Remove(spawn);
      }
    }
    yield return new WaitForSeconds(SpawnRate);
  }
```

Colliders on the front of the vehicle will make it stop if it enters the collider on the back of another vehicle:

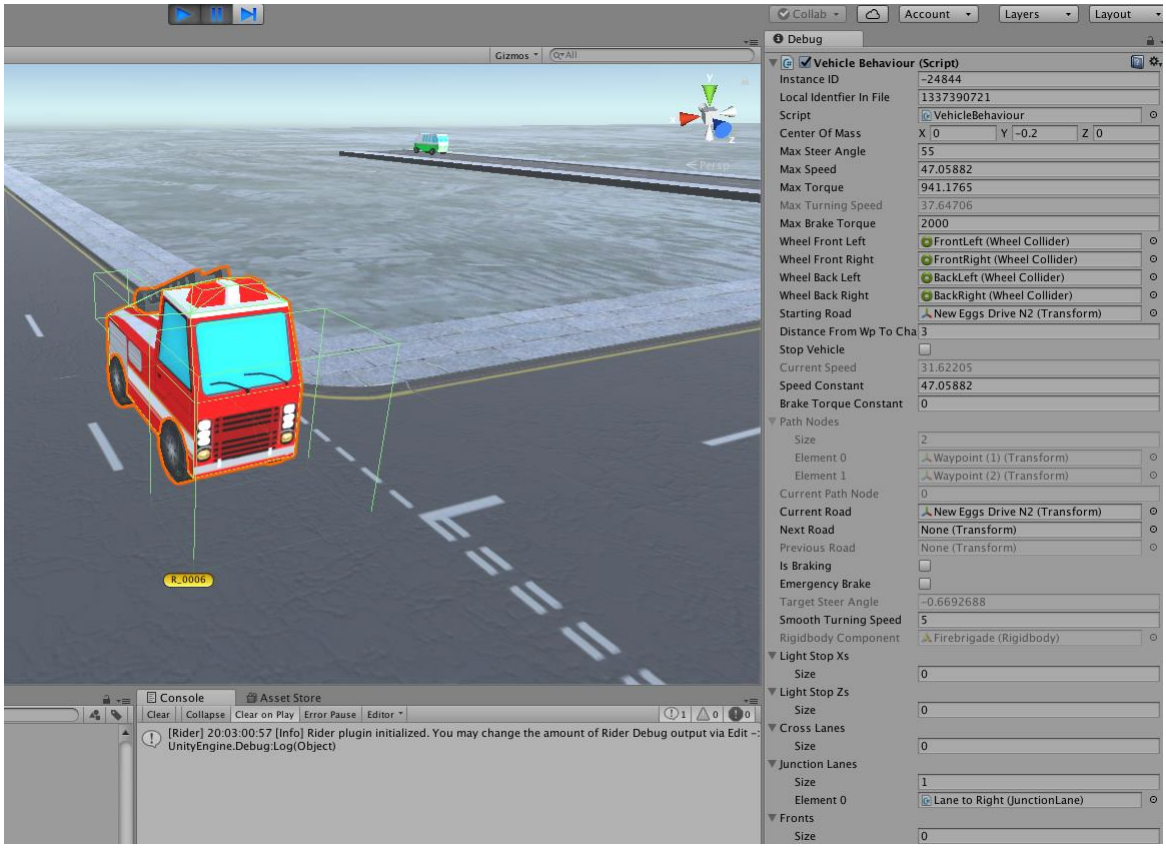
```
private void OnTriggerEnter(Collider other)
{
  if (other.gameObject.GetComponent<CarBackCollider>() == null) return;
  VehicleBehaviour vehicle = other.gameObject.GetComponentInParent<VehicleBehaviour>();
  if(vehicle == null) return;
  if (ReferenceEquals(vehicle, _parent)) return;
  _parent.Stop();
}

private void OnTriggerExit(Collider other)
{
  if (other.gameObject.GetComponent<CarBackCollider>() == null) return;
  _parent.Continue();
}
```

6. Testing

6.1. Unity

Given the nature of objects within the unity game engine, there is no way to carry out unit or integration tests and therefore the environment itself becomes the test suite. In order to do this we can set the ‘inspector’ window to debug mode and see all normally inaccessible variable and watch them change in real time. We can introduce anything we want at run time to see how it reacts, we can pause and move the scene frame by frame watching the variables change as we do. We can also see the outlines of any object such as triggers or colliders.

With this we can then create test scenes, little mock ups of various situations, and test them extensively.



6.1.1. figure

6.2. Android

Although the android code is very short and consists of only one class, I created two unit tests that check the creation of the MainActivity class and test the return of a method which determines the next direction the user is travelling.

```
@Test
public void testNextDirection(){
    assertTrue(main.getNextDirection('N', 'N').equals("straight"));
}
```

6.2.1. figure

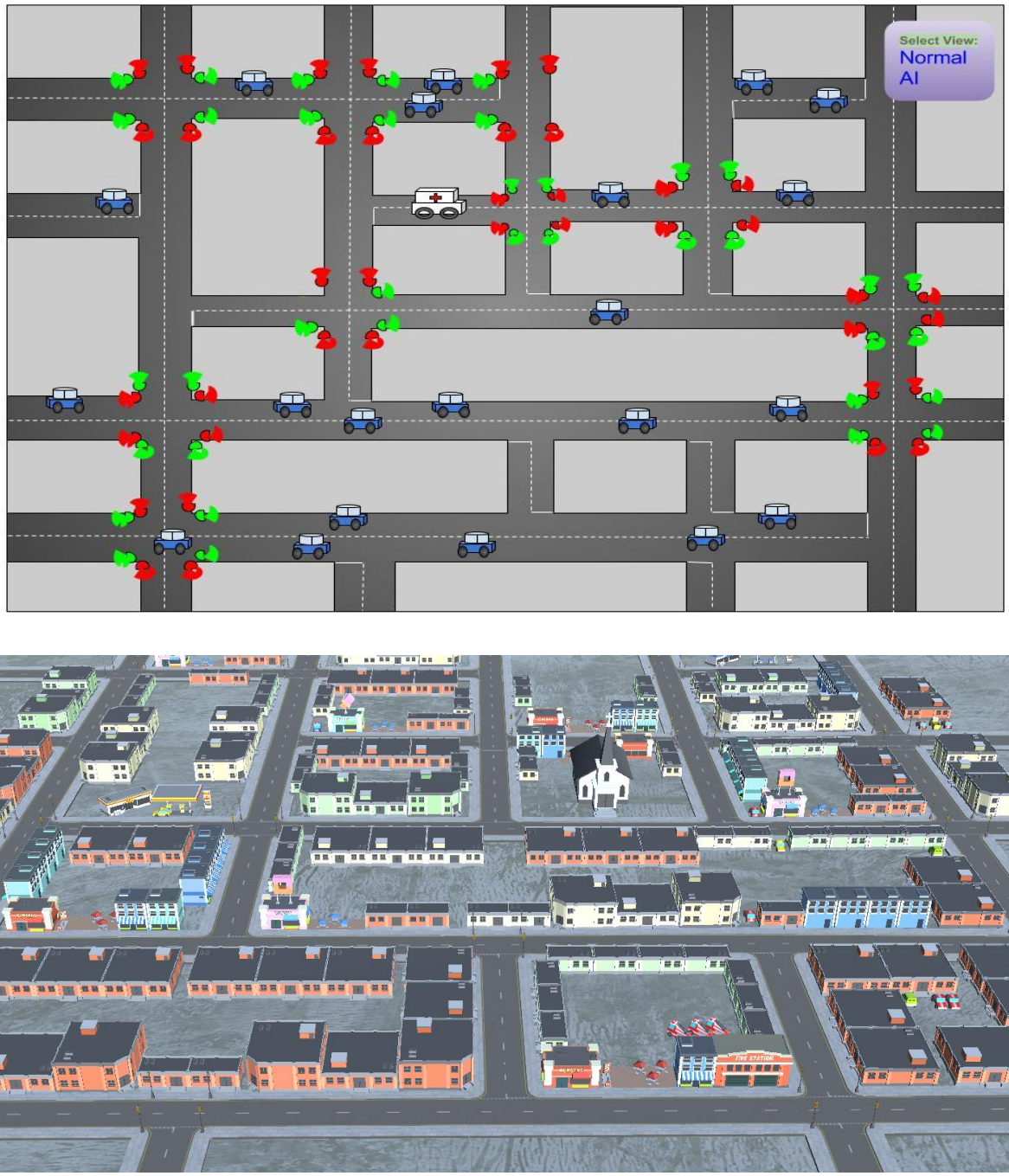
6.4. Python

I used pytest to create a system test that ensures the AI is working correctly. This uses test data stored in test_data.txt, which is a basic dictionary containing all of the roads in the city. It asserts that all methods are returning the expected results and most optimal route.

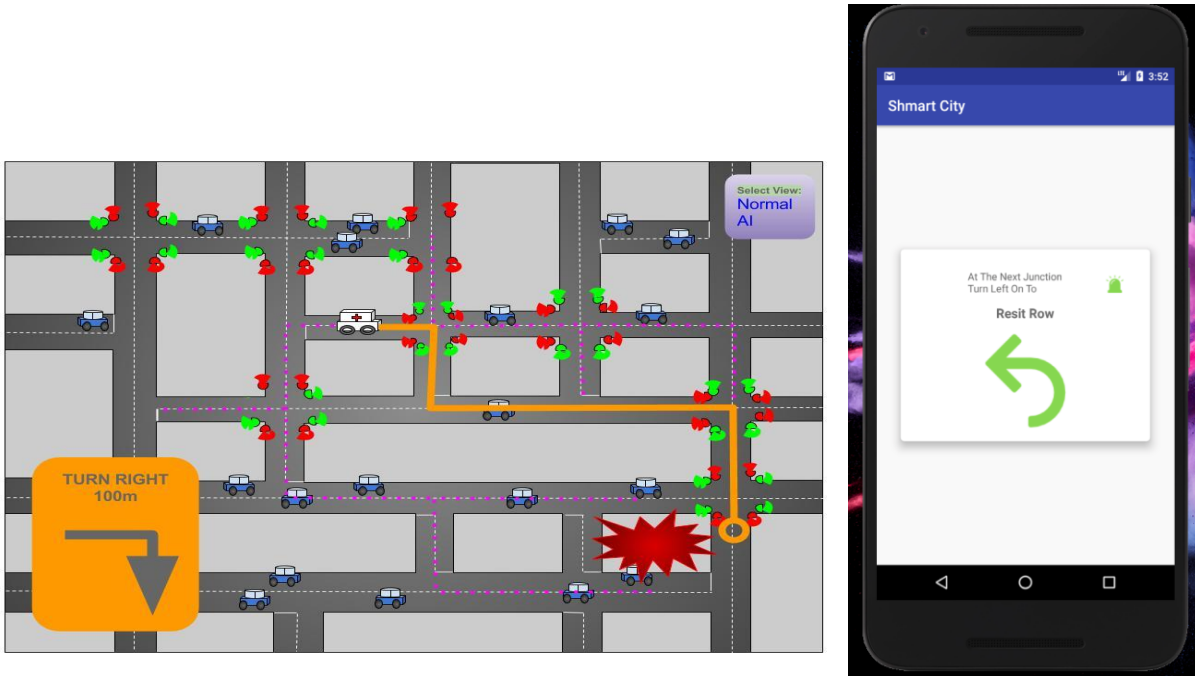
```python
import ast

from app import main

with open('tests/test_data.txt', 'r') as file:
    test_roads = ast.literal_eval(file.read())


# TESTS INCLUDE TESTING FOR city_map.py

def test_build_city():
    main.build_city(test_roads)
    assert main.city_roads.city_roads["Gorgeous Grove W1"].name == test_roads["24"]["name"]


def test_update_congestion():
    main.update_congestion({"road": "Gorgeous Grove W1",
                            "congestion": 1})
    assert main.city_roads.city_roads["Gorgeous Grove W1"].congestion == 1


def test_calculate_best_route():
    assert main.calculate_best_route({
        "start": "Gorgeous Grove W1",
        "end": "Gorgeous Grove W2"
    }) == {"path": [
        "Gorgeous Grove W1",
        "Gorgeous Grove W2"
    ]}

```

6.5. figure

7. GUI



7.1. Figure: GUI City View from proposal vs GUI City View in system

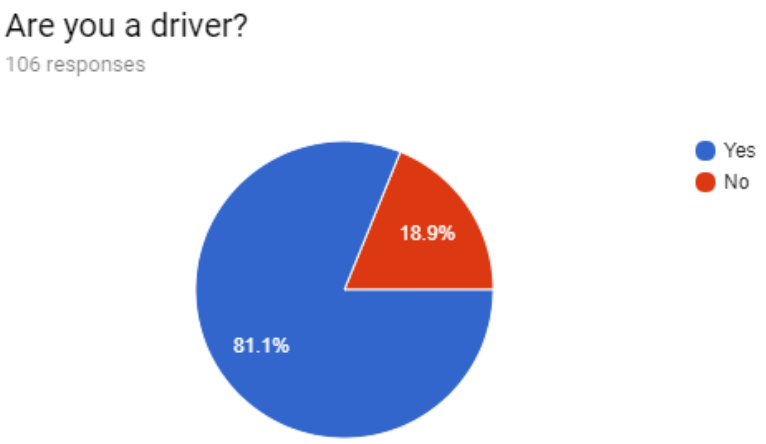


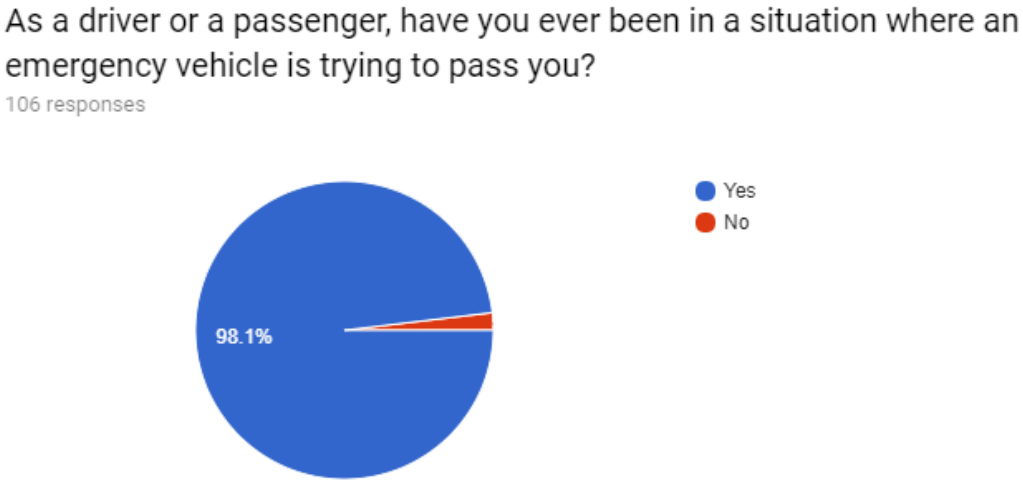
8. *Figure: GUI for directions in proposal vs GUI for directions in system*

9. Customer Testing

At the beginning of the project, I did a survey with random members of the public to gauge the need for such a system as this. These were the results.

There were 106 responses to the public survey, 81.1% of which were drivers. Findings from the survey were very revealing in terms of how road users feel about being in a situation where an emergency vehicle is trying to pass them, how safe they feel in these situations, and above all, how they feel that more can be done using current technology to cater for all road users during these unavoidable situations.

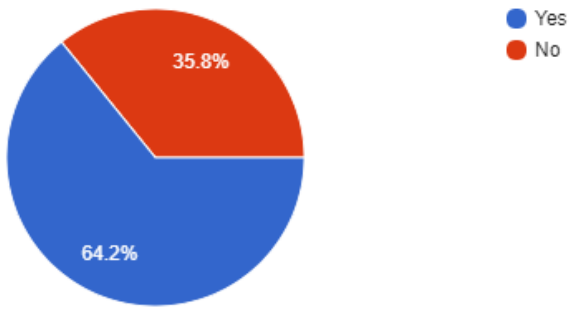




We can see that the overwhelming majority of people have encountered emergency vehicles.

As a driver or a passenger, have you ever experienced a situation where an emergency vehicle was not able to bypas...fic until the traffic light went green?

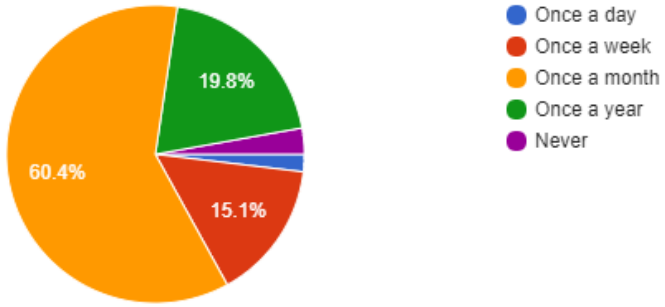
106 responses



A third of all respondents have been in a situation where the emergency vehicle was unable to bypass traffic, pointing to a very real concern given that response time in an emergency situation is often the most important factor.

On average how often do you have to move out of the way for emergency vehicles?

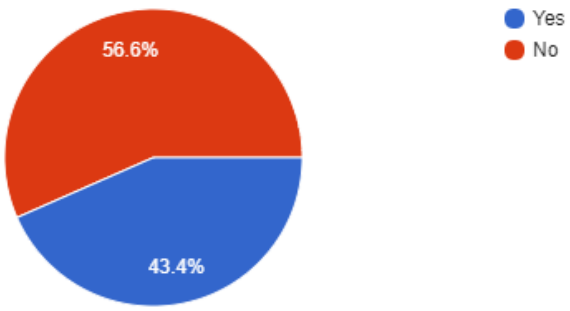
106 responses



The vast majority of people encounter emergency vehicles at least once a month. Given the amount of road users, this is a demonstration of the frequency at which emergency vehicles are trying to get through traffic.

Do you feel that it is always safe to move out of the way for emergency vehicles?

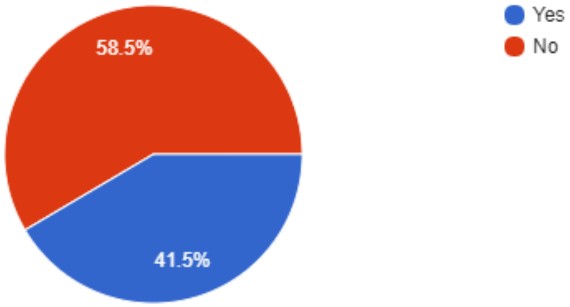
106 responses



On the whole, people who encounter emergency vehicles don't always feel safe giving them passage. The importance of this becomes clear in the results of the next questions.

If you weren't sure it was safe to move for an emergency vehicle, would you still move?

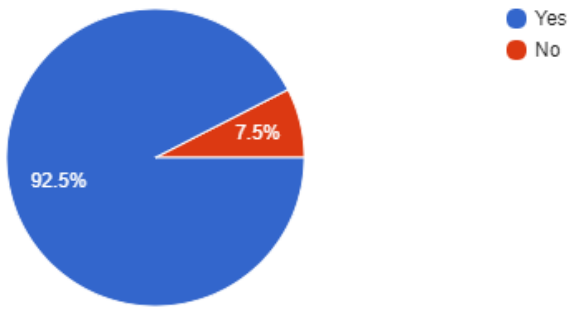
106 responses



Although in the minority, the amount of people who still move for emergency vehicles when it doesn't seem safe to do so is significant. Each "yes" response indicates a time when individuals and road users in general are being put at risk in order to help out whoever the emergency vehicle is responding to.

Do you think the traffic system grid should be able to respond to an emergency and control the flow of traffic ie. changing traffic light colours?

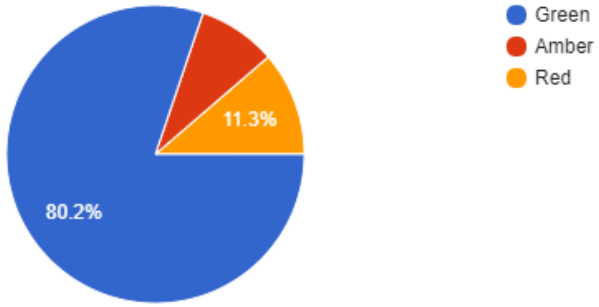
106 responses



In 2017 technology has reached a point where systems in general are able to make autonomous decisions without human intervention, and has become ubiquitous in our daily lives. One of the possible concerns of a smart traffic control system such as this is that it will not sit well with the public in general, however, we can see from the responses that people are comfortable with this idea if it is to increase safety for everyone.

In your opinion, what colour traffic light is easiest to be at when manoeuvring for an emergency vehicle?

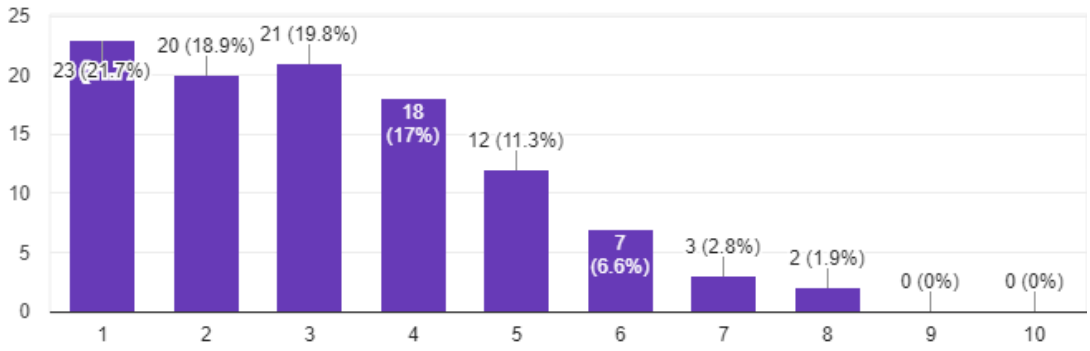
106 responses



People say that accommodating emergency vehicles is easiest when traffic lights are green, changing the signals is an integral part of this software.

How would you rate how easy it is for emergency vehicles to pass traffic DURING rush hour in Dublin?

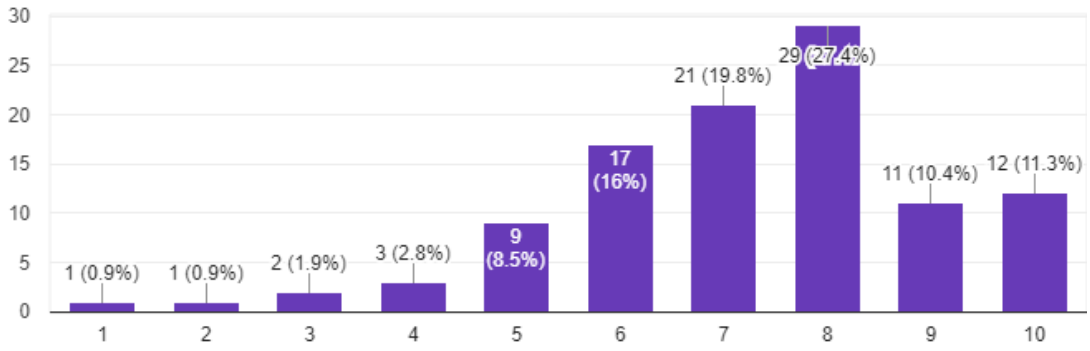
106 responses



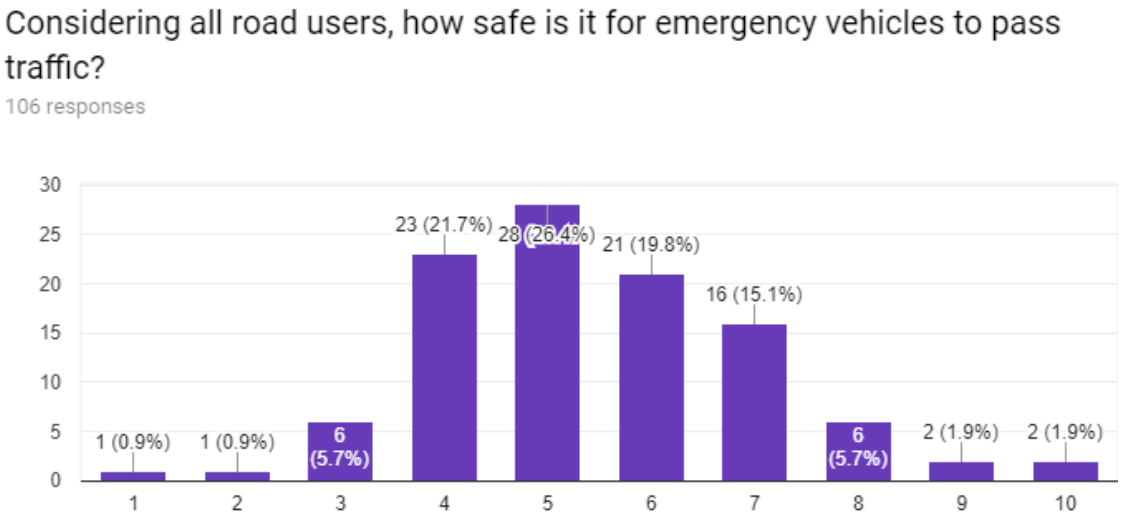
With the current system, people on the whole think that during rush hour it is difficult for emergency vehicles to bypass traffic. With this system, the flow of traffic itself will respond to the emergency and the route taken by the emergency services will be cleared; also, because the system will be monitoring all traffic, it will be able to identify routes with less traffic and obstructions.

How would you rate how easy it is for emergency vehicles to pass traffic OUTSIDE OF rush hour in Dublin?

106 responses



The responses to this question show that at times when the flow of traffic is at its most manageable, it is easier for emergency vehicles to bypass traffic, however, people on the whole still don't think it is completely easy, which indicates that more can be done.



When taking into consideration all road users such as cyclists, busses, trucks, vans, cars etc. it is obvious that there is serious room for improvement.

10. Conclusions

There is no doubt that even with this relatively simple implementation of an AI controlling the traffic grid during an emergency, the safety of road user's is increased. There are no situations where a road user is in danger at a junction as only one lane cane move at a time, and therefore road users need not worry about a siren.

As a proof of concept, there is no doubt in my mind that this is something that should be researched further, and not just because of the perceived benefits that this simulation shows, but because of the countless other applications that an AI controlling the traffic grid could yield.

11. Further Application and Development

The model could be applied to normal traffic situations. An AI with a complete understanding of traffic in an entire city at all times, would be extremely successful in managing traffic during busy periods such as rush hour and would be able to dramatically reduce the amount of congestion, which would positively impact the environment and positively impact ordinary people's lives.

Alongside this, using an AI to intuitively control traffic would also be of benefit to road users during times when the road network is not busy. There would no longer be any requirement for someone to sit at a red light when there is no traffic passing the green light on the other side. This is akin to sensors currently used to change traffic lights, however, having an AI which can interpret the entire grid would mean that the system itself could prepare for the arrival of a single car at a junction, well in advance of its arrival, taking all other road users into consideration. What this means is that the driver wouldn't even be aware of the change in lights, they would simply have an unobstructed path when it is not busy.

Introducing a Convolutional Neural Network (CNN) to this system could yield extraordinary results. A CNN is the type of AI that Google DeepMind is famous for. Its has the ability to evaluate the data it receives on its own terms, rather than what the developer deems to be important. This would mean that this type of AI would be able to see the traffic grid in a completely new way and perceive things never possible to humans, thus improving road travel, congestion, safety etc. like never before. Indeed, theoretically, this could predict accidents before they happen, and try to direct traffic away from potential danger.

This system could be used to see the effects of changes to the road network. Taking the Luas going through College Green as an example, a further developed version of this software could help city planners route the Luas and affected traffic in such a way that the effect on congestion is reduced dramatically.

12. Scope Changes From Proposal

12.1. Neural Network

Originally I wanted to implement some kind of machine learning into the AI, but this proved to be out of scope once the complexity of building a working traffic system with hundreds of independent autonomous vehicles was realised. A simpler AI exists in this project, but there is room for a neural network in a further implementation.

12.2. GUI Switch View

Originally the user was going to have the ability to switch to a view that shows what the sensors are seeing, but once I realised that the only heuristic needed according to this particular implementation was traffic congestion on a street by street basis rather than traffic speed etc, there was no need to show that in particular to the user as they are always able to see the relative congestion by simply looking at the GUI.

12.3. Place Emergency vehicle on Map

I decided that having the Fire Brigade drive around the streets in the same way as all other autonomous vehicles made for a more interesting experience and scrapped this idea.

13. References

Certain assets used were purchased in the unity asset store and modified by me. All of the code in the Assets/scripts folder is entirely my own, and code in other folders from assets has been modified where needed. This will be evident in the github commits. Therefore, the assets I used were mostly just for aesthetics, cars, buildings, road network etc. Here are the 3rd party assets I used:

- Concrete Texture
  - https://assetstore.unity.com/packages/2d/textures-materials/concrete/yughues-free-concrete-materials-12951
- Road Builder
  - https://assetstore.unity.com/packages/tools/modeling/road-builder-73065
- Traffic Lights
  - https://assetstore.unity.com/packages/3d/props/exterior/dynamic-street-props-game-ready-86377
- Vehicles
  - https://assetstore.unity.com/packages/3d/vehicles/land/m-lowpoly-cars-83703
- Buildings
  - https://assetstore.unity.com/packages/3d/environments/simple-buildings-cartoon-city-29003
- Fire Effects
  - https://assetstore.unity.com/packages/vfx/particles/fire-explosions/fire-spell-effects-36825

14. Appendix

14.1. Project Proposal

Objectives

The objective of this project is to create a simulated city that can monitor traffic conditions and, based on this information, control traffic lights and send travel instructions to emergency vehicles in order that the emergency services can get to where they need to be as quickly and as safely as possible.

Background

As we start to utilise the internet in new ways, there is a real opportunity to allow previously ‘dumb’ systems to wake up and take control of their own environment and in using modern technology, become experts within their own niche.

In an emergency situation, response time is essential, but it is largely determined by factors outside the control of the emergency vehicle driver. It is for this reason that I want to apply the principles of the Internet of Things to a traffic light network as a whole.

The idea is that traffic lights, along with any other applicable sensor, will monitor traffic conditions themselves and be able to, as a network, decide and implement the most optimal use of what’s available to them, green, amber and red, to get an emergency vehicle from point A to point B quickly and safely.

The obvious outcome of a successful implementation is in majorly reducing the response times of emergency vehicles. Another outcome is the safety of all road users when emergency vehicles are trying to pass.

Technical Approach

The system will work by monitoring traffic in multiple ways, these sensors will be simulated:

- Vehicle Speed
- Vehicle Density
- Traffic density
- State of the road (roadworks, closures etc.)
- Pedestrian behaviour

The system will be trained using machine learning to work out the best possible route for the emergency vehicles, while assessing the impact on the wider traffic network. It will control the flow of traffic to this end, and it will evaluate its decisions based on the immediate results of that decision using Temporal Difference Learning.

The system will be presented as a city simulation. The cityscape will be littered with traffic, bad road conditions and emergency vehicles which will be randomly placed, as will the emergencies they are trying to get to. We will also be able to see what is being communicated to the driver of the emergency vehicle by the network.

The simulation will be viewable in two ways:

- Human Perspective: A top down view of the cityscape presented as we would see it naturally from this position.
- System Perspective: A top down view of the cityscape that shows us how the system is seeing the world which will include, line of sight for sensors, colours representing traffic density, calculations of the speed of the emergency vehicles and other road users including pedestrians.

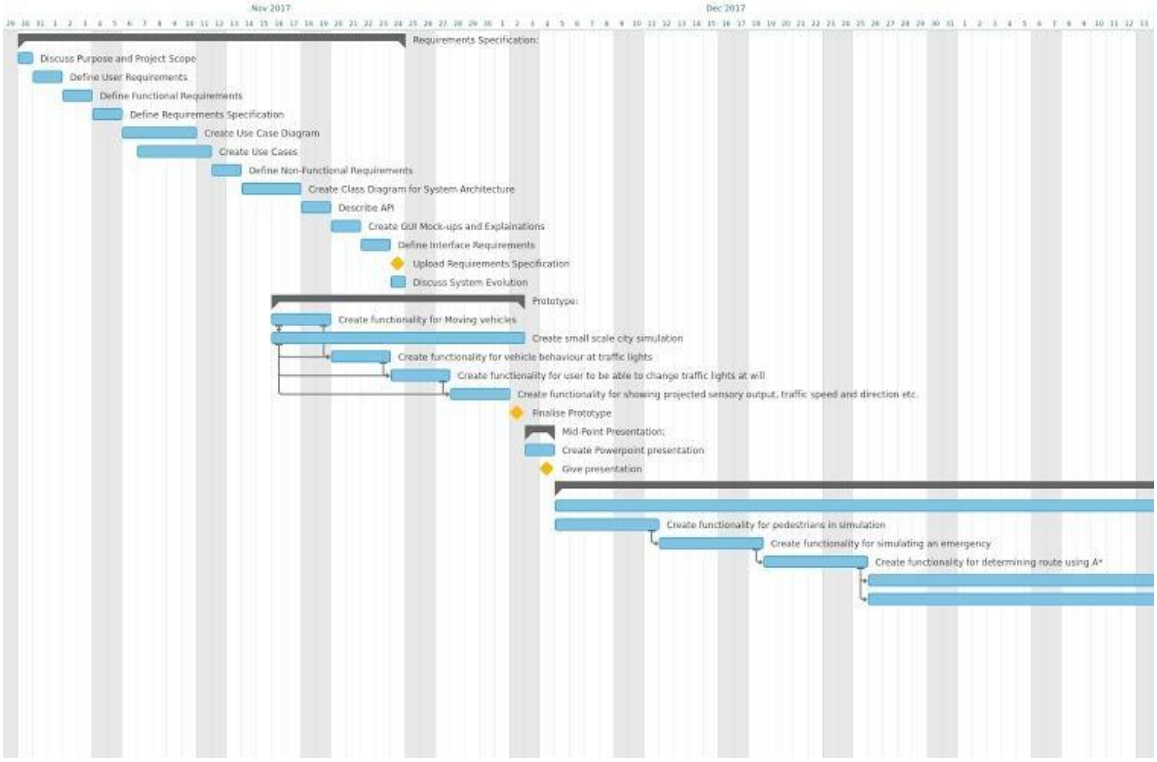
Every so often the system will create a random obstacle such as a pedestrian walking out onto the road, or a road user breaking the red light and it will be up to the system itself to decide the best approach and alert the emergency vehicle.

All the collected data will be sent to a cloud processor which will make decisions, control the traffic lights and alert the emergency vehicle.

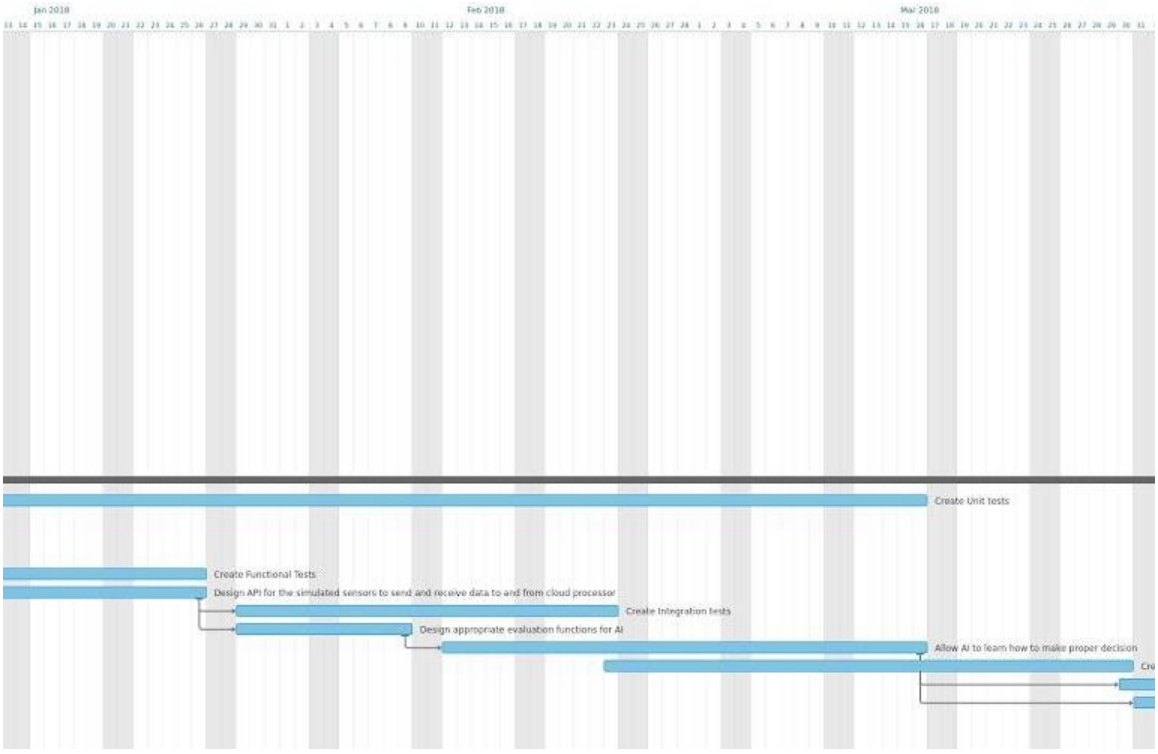
Technical Details

- Unity: This is a game engine I will be using to create the simulation of the city
- Cloud Processing: All of the processing will take place in the cloud using data collected from various sources
- A*: This is an algorithm for working out a route from one place to another. The difference between this and other algorithms is that it can take into consideration other factors; in this case, traffic density, roadworks etc.
- Neural Network: Trained with Temporal Difference Learning: A neural network will be trained to consider the best route for the emergency vehicle to take, this will happen continually on the fly as the route may need to change based on unforeseen scenarios. Temporal Difference Learning is a technique which doesn't rely on the final outcome to determine how good or bad a decision was, for example, if we only used whether or not the vehicle made it to its destination as an evaluation function, this will disregard good decision made during a journey which didn't successfully complete, and will bolster bad decisions made in a journey which did complete. The works by determining functions which will evaluate success after each decision is completed.
- Cloud Processing: All collected data will be send to a cloud processor (Microsoft Azure, AWS etc.) where the decision will be made.

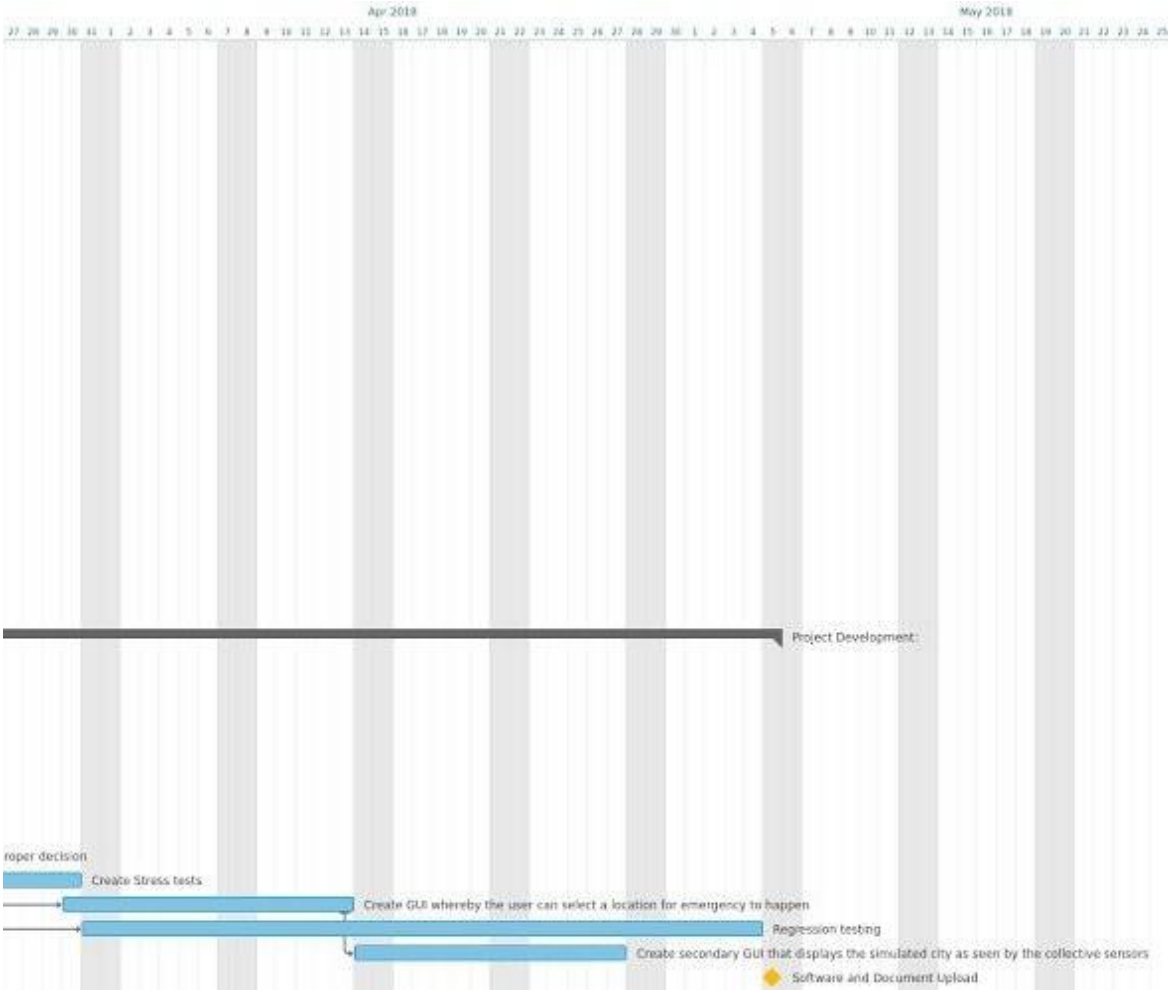
14.2. Project Plan



14.3. Figure part 1



14.4. Figure part 2



14.5. Figure part 3

14.6. Monthly Journals

14.6.1. September

In September I finalised which idea I was going to use for my Software project and started to prepare some notes for the project pitch.

I did the project pitch and it was accepted first time.

Given that there is a machine learning element to the project, I began to research different types of learning algorithms to see which one will suit my project best.

14.6.2. October

In October, I wrote the Project Proposal. For this I fleshed out my idea in terms of implementation, what the user will be able to see, and how the system will work behind the scenes.

In order to do this, I researched the different technologies I will be using, and also the types of theoretical implementations I will be using for the Artificial Intelligence. I also began to consider how the AI will work in terms of how the system will see the world and begin to learn what it should and shouldn't be doing.

Created a project plan and a gantt chart for the entire project.

Considered the types of testing that I will be using for the project.

14.6.3. November

Downloaded Unity and started a project. Began by creating a 2D project as 2D seems like it would be easier to handle. Downloaded some sprites for testing in 2D and was able to get the vehicle moving by itself around a road, albeit with difficulty.

14.6.4. December

Looked into doing the project in 3D and saw that there is a lot more available to me in terms of how the physics would work. For example, in order to turn an object in 2d, you simply add a force and rotate. It's a nightmare to get it to run smoothly like a normal vehicle. By comparison, in 3D there is already Physics object which will do all the necessary forces for you. You literally just need to work out which way to point the wheels and then point them.

14.6.5. January

Looked for assets to use in the project and found some good vehicle assets as well as an automatic road builder. Have the vehicles moving along a waypoint from node to node, but they still don't look out for one another very well. Using a thing called RayCasting that

sends like a laser from the object and returns whether it's hit anything and what that thing is etc. Seems to be workable.

14.6.6. February

Got a new roadbuilder that actually works, last one was a nightmare. With this one, you just drag a load of lines, connect them together and when you're done, it creates the road network along the lines allowing for corners and junctions etc.

14.6.7. March

RayCasting is extremely expensive on the engine, learned about box colliders. You just check the isTrigger button and then you can use the OnTriggerEnter and OnTriggerExit methods. This way I can control when a vehicle reaches the traffic lights.

14.6.8. April

Implemented traffic lights, connected traffic lights to triggers so that if the light is red the traffic will stop at the trigger until it changes. Having a nightmare trying to get the cars to turn right at a junction without smashing into one another.

Set up the android and python projects and began structuring them.

14.6.9. May

Massive final push. Pretty much implemented the lot, what I had done previously became something of the studying part. I learned a lot from it but it was not entirely usable. Tidied everything and put in some nice features.

Done.

14.7. Mid-Point Requirements Specification Document

Executive Summary

As we begin to move towards a world where computers interact with all real world processes ubiquitously and seamlessly, it is time to take the first steps in that direction.

We have the ability using the internet and incredible processing speeds, not to mention incredible algorithm, to put ‘eyes’ on the world, show it to a computer and let that computer tell us what we have right and what we have wrong.

It is with this in mind that I decided upon this project; a way of improving the flow of traffic during emergencies so that emergency vehicles can better reach their destination in a safer and quicker way.

I believe the process of using computers ubiquitously should begin with issues of safety so that we can start off utilising this power in a positive way.

This technical report will show you how the system will work and the impact it would have on the ability of emergency vehicles to get where they need to go.

Introduction

1.1 Purpose

The purpose of this document is to set out the requirements for the development of a simulated system which monitors and controls traffic and feeds emergency vehicles with information in order that they will arrive at their destination in the fastest and safest way possible.

1.2 Project Scope

The scope of the project is to develop a simulation to prove how the use of widely available sensors, traffic data, and Artificial intelligence can be used to monitor and control traffic systems in order that emergency vehicles can get to their destination in the shortest time possible.

I conducted a public survey in order to establish road users experiences when emergency vehicles are trying to pass traffic and whether they think more can be done to improve the process, the results of which are expanded upon below.

In brief, this system will essentially be a simulated city with a fully functioning traffic system whereby the traffic system as a whole is monitored by simulated sensors, and controlled by an AI with the purpose of improving the safety and reliability of emergency response vehicles.

1.3 Definitions, Acronyms, and Abbreviations

AI Artificial Intelligence

1.4 Background

As we start to utilise the internet in new ways, there is a real opportunity to allow previously ‘dumb’ systems to wake up and take control of their own environment and in using modern technology, become experts within their own niche.

In an emergency situation, response time is essential, but it is largely determined by factors outside the control of the emergency vehicle driver. It is for this reason that I want to apply the principles of the Internet of Things to a traffic light network as a whole.

The idea is that traffic lights, along with any other applicable sensor, will monitor traffic conditions themselves and be able to, as a network, decide and implement

the most optimal use of what's available to them, green, amber and red, to get an emergency vehicle from point A to point B quickly and safely.

1.5 Aims

The obvious outcome of a successful implementation is in majorly reducing the response times of emergency vehicles. Another outcome is the safety of all road users when emergency vehicles are trying to pass.

1.6 Technical Approach

The system will work by monitoring traffic in multiple ways, these sensors will be simulated:

- Vehicle Speed
- Vehicle Density
- Traffic density
- State of the road (roadworks, closures etc.)
- Pedestrian behaviour

The system will be trained using machine learning to work out the best possible route for the emergency vehicles, while assessing the impact on the wider traffic network. It will control the flow of traffic to this end, and it will evaluate its decisions based on the immediate results of that decision using Temporal Difference Learning.

The system will be presented as a city simulation. The cityscape will be littered with traffic, bad road conditions and emergency vehicles which will be randomly placed, as will the emergencies they are trying to get to. We will also be able to see what is being communicated to the driver of the emergency vehicle by the network.

The simulation will be viewable in two ways:

- Human Perspective: A top down view of the cityscape presented as we would see it naturally from this position.
- System Perspective: A top down view of the cityscape that shows us how the system is seeing the world which will include, line of sight for sensors, colours representing traffic density, calculations of the speed of the emergency vehicles and other road users including pedestrians.

Every so often the system will create a random obstacle such as a pedestrian walking out onto the road, or a road user breaking the red light and it will be up to the system itself to decide the best approach and alert the emergency vehicle.

All the collected data will be sent to a cloud processor which will make decisions, control the traffic lights and alert the emergency vehicle.

1.7 Technologies

- Unity: This is a game engine I will be using to create the simulation of the city
- Cloud Processing: All of the processing will take place in the cloud using data collected from various sources
- A*: This is an algorithm for working out a route from one place to another. The difference between this and other algorithms is that it can take into consideration other factors; in this case, traffic density, roadworks etc.
- Neural Network: Trained with Temporal Difference Learning: A neural network will be trained to consider the best route for the emergency vehicle to take, this will happen continually on the fly as the route may need to change based on unforeseen scenarios. Temporal Difference Learning is a technique which doesn't rely on the final outcome to determine how good or bad a decision was, for example, if we only used whether or not the vehicle made it to its destination as an evaluation function, this will disregard good decision made during a journey which didn't successfully complete, and will bolster bad decisions made in a journey which did complete. The works by determining functions which will evaluate success after each decision is completed.
- Cloud Processing: All collected data will be send to a cloud processor (Microsoft Azure, AWS etc.) where the decision will be made.

2 System Requirements

2.1 Functional requirements

2.1.1 Requirement 2: Place Emergency Vehicle

2.1.1.1 Description & Priority

This functionality allows the user to select a point anywhere on a road in the city simulation and place an emergency vehicle at this point.

2.1.1.2 Use Case

Scope

The scope of this use case is to allow users to place an emergency vehicle in the city.

Description

This use case describes the steps a user will take to place an emergency vehicle on the map.

Flow Description

Precondition

The system is running the simulation of the city and being controlled by the AI.

Activation

This use case starts when the user selects a point on a road in the simulation.

Main flow

1. The system runs the simulation
2. The simulation displays the city
3. The AI controls the traffic with data from the simulation
4. The user selects a point on a road
5. The system places an emergency vehicle at the selected point

Termination

The system terminates the process when the emergency vehicle is placed in the simulation

Post condition

The system continues to control the simulation and the emergency vehicle travels around the map like any other vehicle.

2.1.2 Requirement 1: Place Emergency

2.1.2.1 Description & Priority

This functionality allows the user to select a point anywhere except a road in the city simulation and set it as an emergency to which the emergency vehicle must travel.

2.1.2.2 Use Case

Scope

The scope of this use case is to allow users to initiate an emergency.

Description

This use case describes the steps a user will take to place an emergency on the map.

Flow Description

Precondition

The system is running the simulation of the city and being controlled by the AI; an emergency vehicle has already been placed by the user.

Activation

This use case starts when the user selects a point anywhere except a road in the simulation.

Main flow

1. The system runs the simulation
2. The simulation displays the city
3. The AI controls the traffic with data from the simulation
4. The user selects a point not on a road
5. The system places an emergency at the selected point

Termination

The system terminates the process when the emergency is placed in the simulation

Post condition

The AI calculates the best route, sends information to the emergency vehicle, and controls the flow of traffic to benefit the emergency vehicle.

2.1.3 Requirement 2: Switch View

2.1.4 Description & Priority

This use case describes how a user can select different ways to view the simulation.

2.1.5 Use Case

Scope

The scope of this use case is to allow a user to choose which view of the simulation they would like to see.

Description

This use case describes the steps a user will take to change the view of the simulation.

Flow Description

Precondition

The system is running the simulation of the city and the AI is being sent data.

Activation

This use case starts when the user selects a view for the simulation to display.

Main flow

1. The system runs the simulation
2. The simulation displays the city
3. The AI controls the traffic with data from the simulation
4. The user selects the normal view of the simulation (see A1)
5. The user selects the AI view of the simulation (see A2)
6. The system changes the UI to display appropriately

Alternate flow

A1 : Normal View

1. The system shows a bird's eye view of the city how it would be seen from above by the naked eye.

A2 : AI View

2. The system shows a bird's eye view of the city how it is seen from the perspective of the AI, for example how the AI calculates the speed of vehicles, traffic density, sensor input etc.

Termination

The system terminates the process when the view has been switched.

Post condition

The system displays information to the user based on their selected view.

2.1.6 Requirement 2: Control Traffic Flow

2.1.6.1 Description & Priority

This use case describes how the AI controls the flow of traffic.

2.1.6.2 Use Case

Scope

The scope of this use case is to allow the AI to control the flow of traffic.

Description

This use case describes the steps the AI will take to determine the best way to control traffic.

Flow Description

Precondition

The system is running the simulation of the city and the AI is being sent data.

Activation

This use case starts when the system is running and sending data to the AI.

Main flow

1. The system runs the simulation
2. The simulation displays the city
3. The system sends data to the AI from the simulation
4. The AI feeds data into its neural network
5. The AI's neural network asses what changes to traffic lights needs to happen to maintain or improve traffic conditions in a non-emergency state (see A1)
6. The AI's neural network asses what changes to traffic lights needs to happen to benefit the emergency vehicle in an emergency state (see A2)
7. The AI makes changes

A1 : Non-emergency State

1. AI calculates the current state of traffic
2. AI identifies areas for improvement

A2 : Emergency State

1. The user places an emergency vehicle in the simulation
2. The user places an emergency in the simulation
3. AI calculates the current state of traffic

4. AI identifies area where it can manipulate traffic to benefit the emergency vehicle

Termination

The system terminates the process when the AI has made a change.

Post condition

The traffic light system is changed and the system feeds more data to the AI.

2.1.7 Requirement 2: Calculate Emergency Vehicle Route

2.1.7.1 Description & Priority

This use case describes how the AI sends route information to the emergency vehicle to get it to its destination as quickly as possible.

2.1.7.2 Use Case

Scope

The scope of this use case is to allow the AI to calculate the best route during an emergency.

Description

This use case describes the steps the AI will take to determine the best route for the emergency vehicle.

Flow Description

Precondition

The system is running the simulation of the city, the AI is being sent data, an emergency vehicle has been placed and an emergency has been placed.

Activation

This use case starts when the user places an emergency.

Main flow

1. The system sends data to the AI from the simulation
2. The AI feeds data into its neural network
3. The AI calculates the best route for the emergency vehicle
4. The AI sends route information to the emergency vehicle

Termination

The system terminates the process when the emergency vehicle has reached its destination.

Post condition

The system returns to a non-emergency state.

2.2 Data requirements

2.2.1 Performance/Response time

The cloud based web service will need to be able to communicate with the system in as fast a time as possible as the AI will need to be able to make changes to the system in as close to real time as possible given that a momentary lapse could lead to a failure of the emergency vehicle to reach its destination safely. In order for the AI to work as efficiently as possible it will be able to control the traffic flow during its decision making process, and will make retrospective changes if necessary. During an emergency, the AI will pay particular attention to the route that has been calculated for the emergency vehicle to travel. The AI will use an A* style searching algorithm to determine the best possible route for the emergency vehicle, and will continue to update this as the vehicle travels. The A* method will significantly reduce the scope of the problem so that the AI is not considering routes that will definitely not yield the best result, thereby increasing performance.

2.2.2 Maintainability

Given that the processing power should be spread throughout multiple cloud based web services, there is no need for downtime when applying a patch to the server as it can be applied and tested in an isolated but fully functional instance before the system starts communicating with the updated patch. If there is an error with the patch that isn't spotted until it goes to production and subsequently negatively impacts the performance of the system, the server can revert back to a previous state using the version control logs.

2.3 User requirements

2.3.1 Security

If this were applied to the public domain and used in a real city, the communication between the traffic grid and the AI would need to be secured by end-to-end encryption; not only to stop intruders in the network from seeing the data, but also to ensure that malicious input is not sent to the AI in order to manipulate the system as a whole. At the same time, the source code for the system would have to be entirely accessible by the public at all times, and any changes made would have

to have public oversight to ensure the integrity of the system from individuals, terrorist organisations, and from government surveillance.

2.3.2 Reliability

The system should not crash due to any errors in the simulation, or due to any errors in communicating with the AI or if it receives an error from the web service. Errors should be handled at the system level to retry these processes when an error occurs. The system can not operate functionally without the internet, but the system should not crash when the connection drops, rather it should pause and wait for the connection to be reestablished.

2.4 Environmental requirements

2.4.1 Amazon Web Service

In order to process the data in the cloud the system will communicate with a web service like amazon web service. The reason for using a web service such as this is that they offer scalable processing power as well as storage. This will be used to optimise the AI so that the system can receive updates from the AI in the shortest time possible. Another factor is that there can be multiple instances of the AI available for other uses, for testing or for updating safely.

2.4.2 Unity Game Engine

The simulation itself will be built using the unity game engine. This will be able to handle many integral processes such as collision detection so that the AI can calculate its success rate and make changes to its neural network accordingly.

2.4.3 Extensibility

The Unity engine uses a component based design principle. This differs from OOP insofar as objects are defined by components added to them, and components are interchangeable between objects. In terms of extensibility, if the system were to be updated to include faster cars, or pedestrians, all that would need to be done is to take an already existing object and manipulate the current components or write simple components to add to these object, thus making extensibility in the simulated environment easy considering that it is unlikely to break the system. The AI will be built in such a way that it is always learning, and as such, new heuristics, or inputs, can be added to the neural network so that it will take new or more relevant information into consideration when deciding the best approach. This will make extending the AI's functionality a relatively painless process. Extensive System and unit testing will also be written during development ensuring that and extensions to the software will not compromise the system.

2.4.4 Portability

The simulation and system in general will be built in unity, and the AI will exist in the cloud and will not communicate with the system in such a way that it is

necessary for it to also be built in unity, therefore the system can be rebuilt using any engine and can still interact with the AI directly.

2.5 Usability requirements

2.5.1 Availability

The availability of the AI is of paramount importance to the running of the system in general, however, if this system were applied to the real world, any downtime would simply default to the current system we have, an imperfect system, but one that will still function.

2.5.2 Recovery

The source code for the system and for the AI will be backed up with version control in an online repository. If the connection between the system and the web service is lost, there should be a system in place to immediately switch from one host to another without the user's knowledge. To try and reduce the downtime to 0, the system could be spread out over a number of cloud Web Service instances which can jump between processors and servers whenever necessary, especially in the case of downtime.

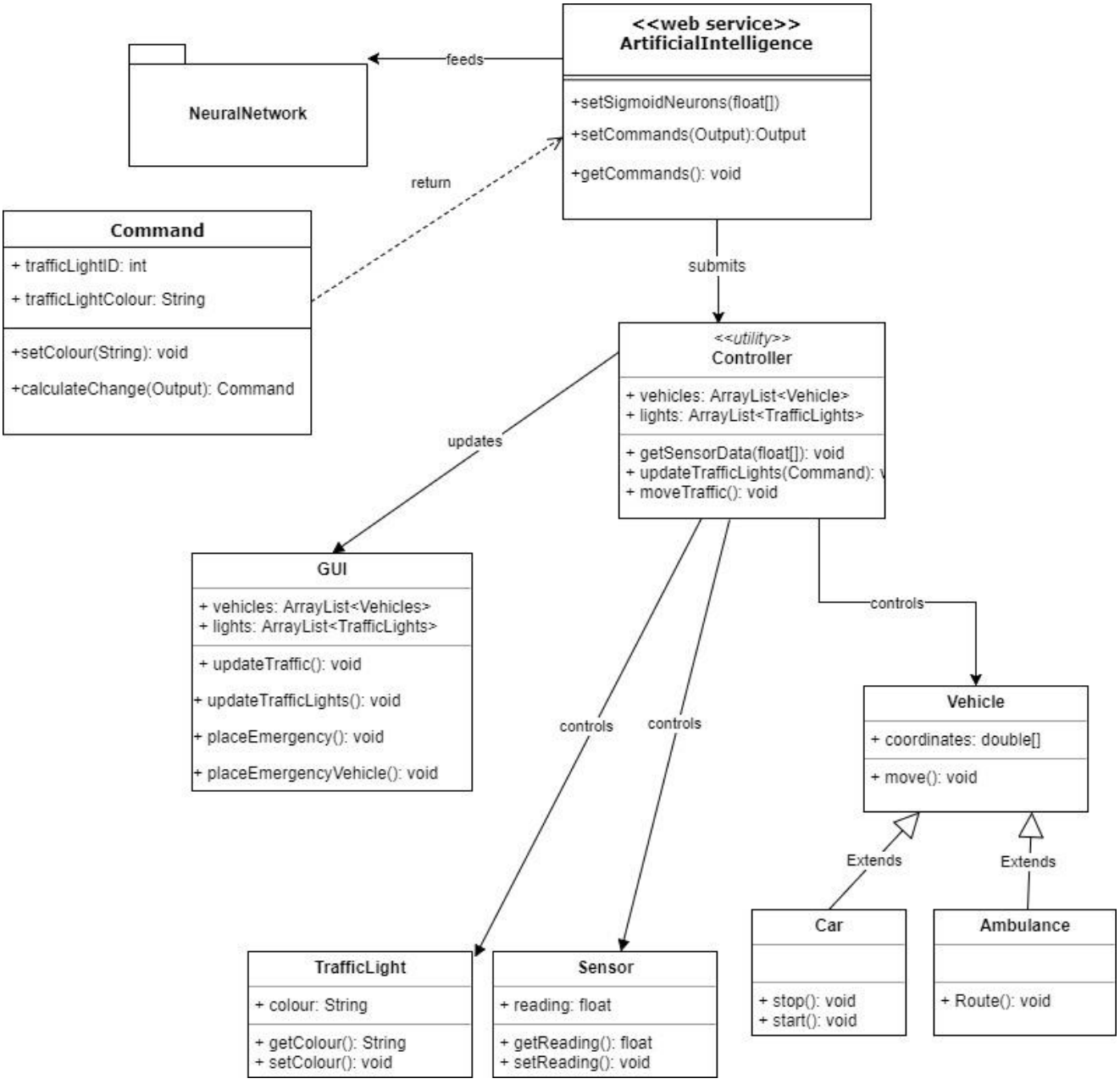
2.5.3 Robustness

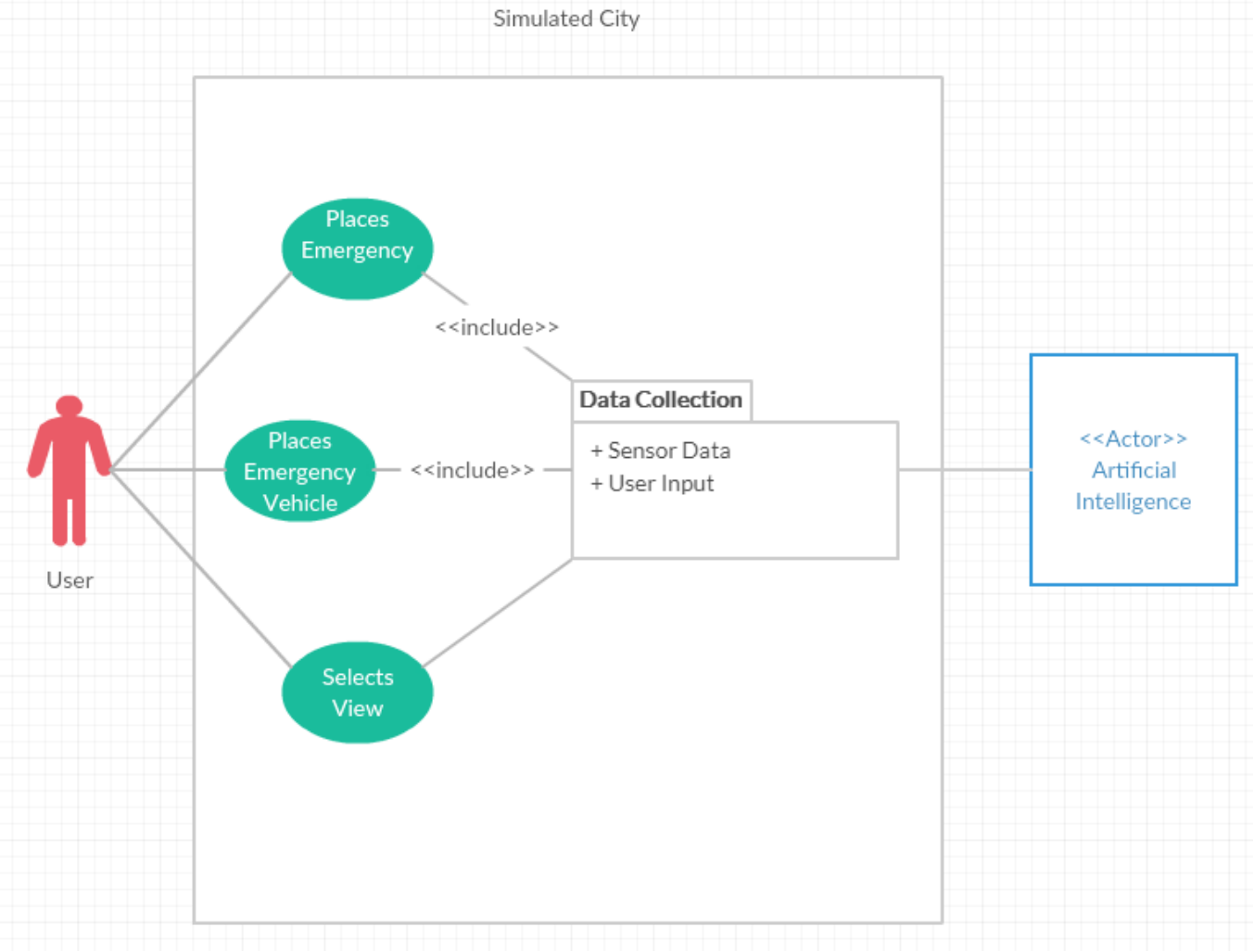
The system will be designed in such a way that user input is minimal. Given the complexity of both the simulated environment and the AI controlling it, the less complex variables in the mix, the better. Having the user as minimally active as possible makes the system more robust as a whole as the AI will be working in an environment entirely familiar to it, and predictable to an extent. The main thing that needs to remain robust is the connection between the system and the AI. If the AI makes a mistake, that is not a failure in robustness, it is in fact a learning opportunity for the neural network; however if there is an error in the data sent or received by the system, then the whole process could be compromised.

2.5.4 Reusability

The goal of the project is partly to develop an AI that can take in traffic information and be able to understand what is happening in the city as a whole. This could be applicable to any system that needs to understand traffic in depth in order to be functional; city planning being an obvious example.

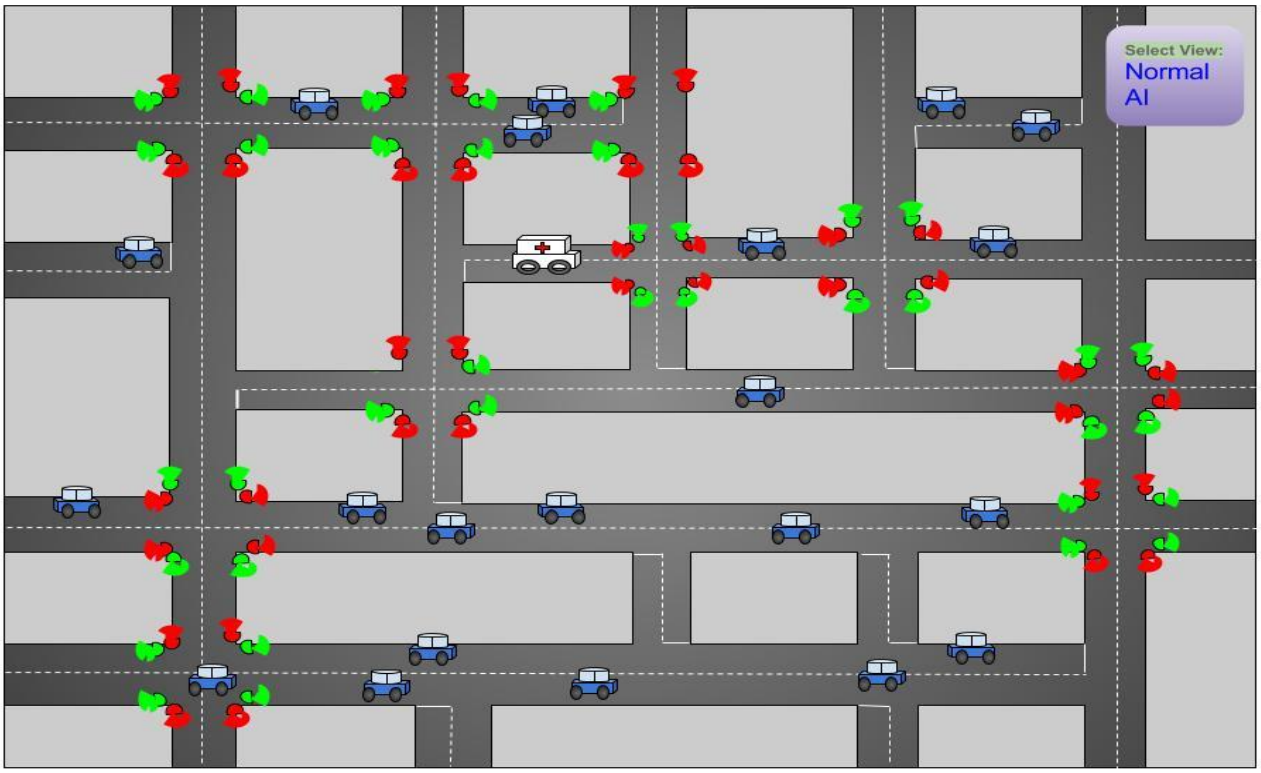
3 Design and Architecture



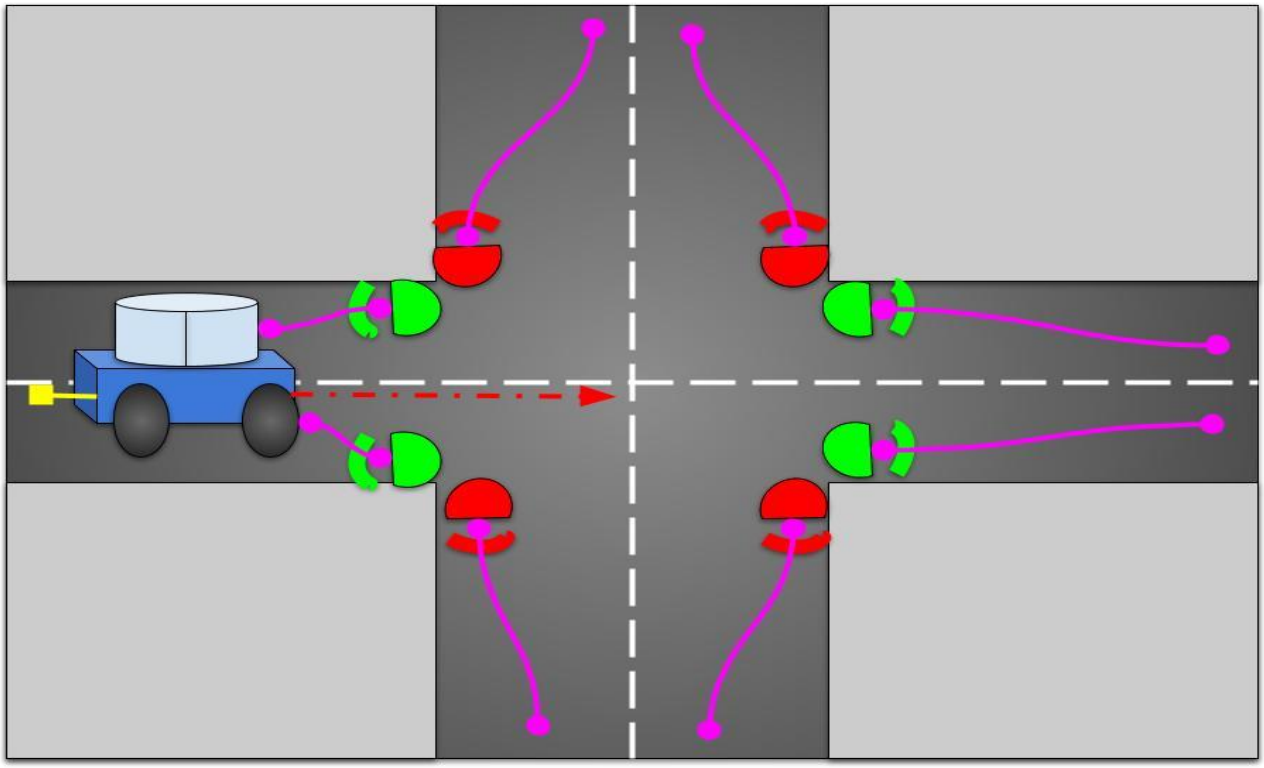


3.1 Implementation

3.2 Graphical User Interface (GUI) Layout



City Simulation running in normal view with no emergency in progress

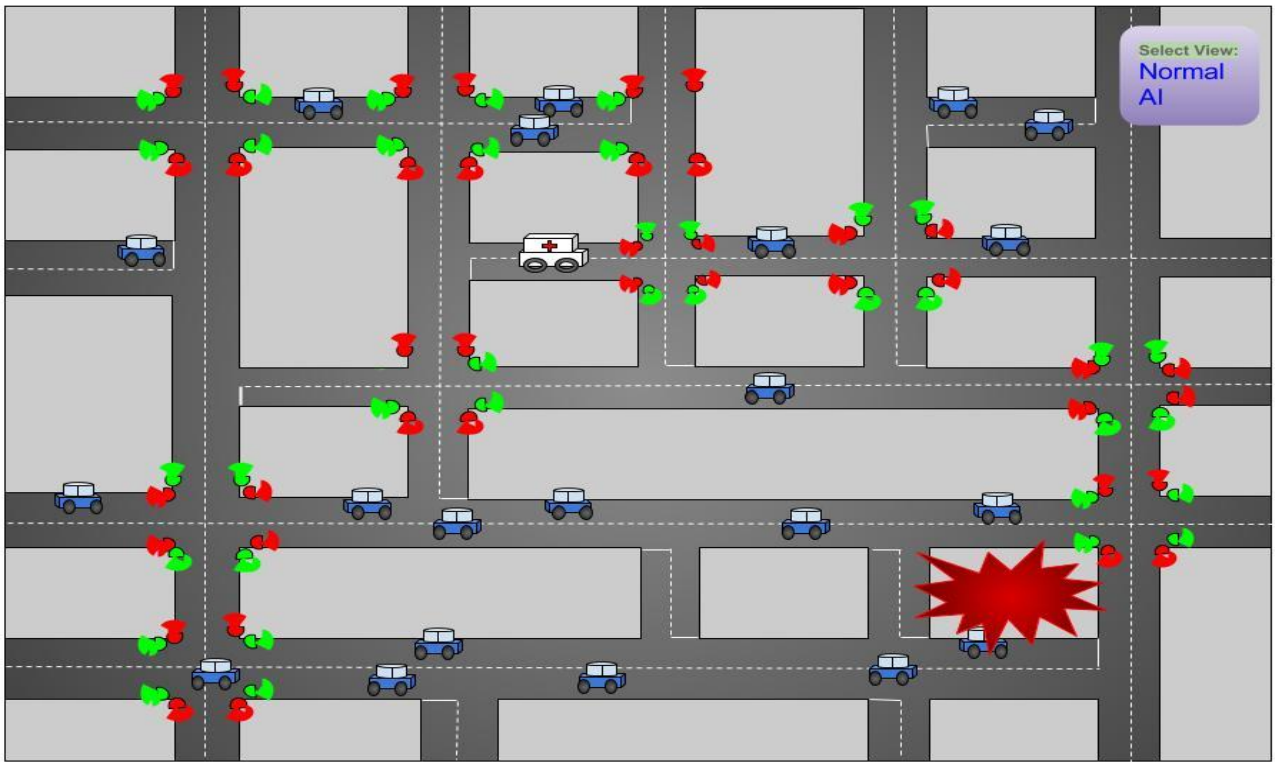


Showing sensory input for the AI View

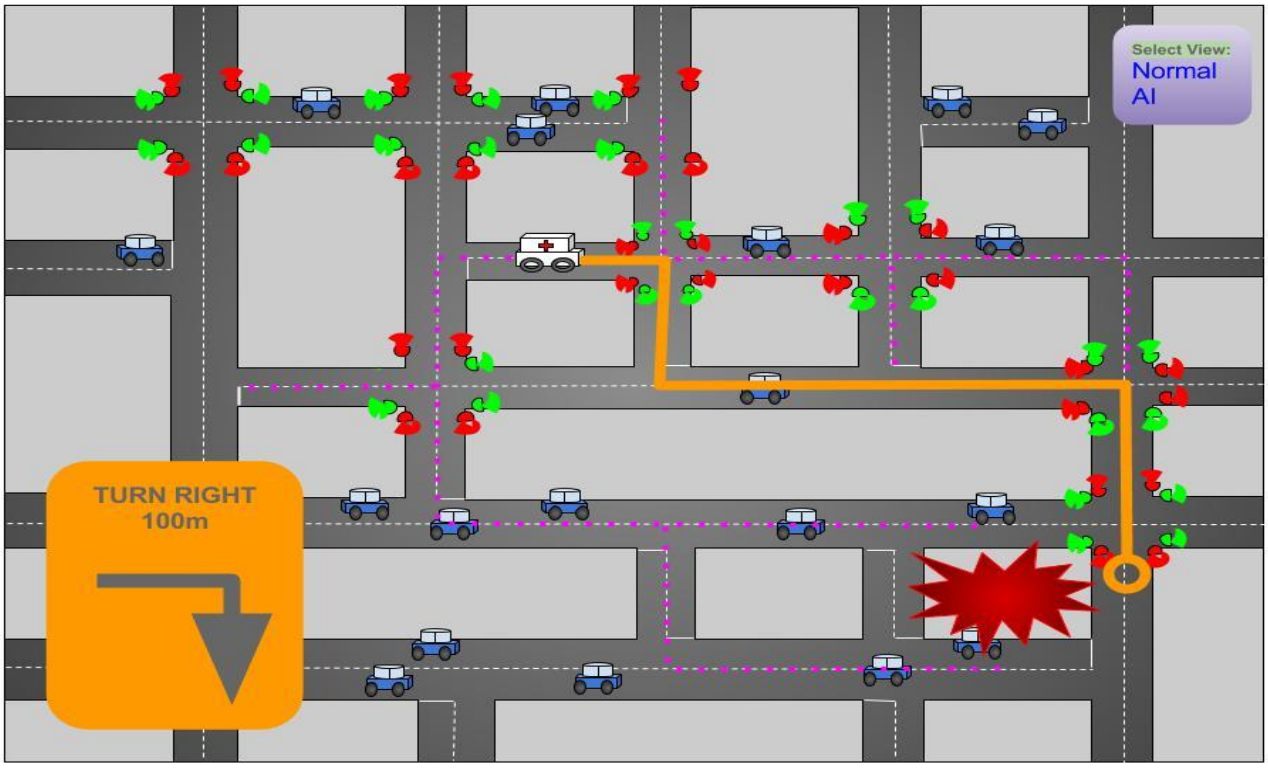
The Red arrow indicates the trajectory of the vehicle

The Yellow line indicates the calculated speed of the vehicle

The purple lines are proximity sensors indicating the distance to the next vehicle



User places an emergency on the map shown here as an explosion



The AI Calculates the most optimal route and begins to control the flow of traffic

The route that is currently being taken is referenced by the orange line

Sat Nav type information for this route is displayed to the user

The possible routes are referenced by the purple dotted lines

The AI may switch routes on the fly if it finds more optimal conditions