

# 3D Traffic Modeling in Unity

Group 2:

**Isaiah Martinez**

CSUN

Computer Science Department

isaiah.martinez.891@my.csun.edu

**Jae Molina**

CSUN

Computer Science Department

jae.molina.499@my.csun.edu

**Anastasia Naydina**

CSUN

Computer Science Department

anastasia.naydina.947@my.csun.edu

05/13/2024

# Contents

<b>1</b>	<b>Change History</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Background . . . . .	4
2.2	Related Works . . . . .	4
2.3	Stakeholders . . . . .	4
2.4	Document Structure . . . . .	4
2.5	Responsibilities . . . . .	5
<b>3</b>	<b>Design Goals</b>	<b>5</b>
3.1	Purpose . . . . .	5
3.2	Objectives . . . . .	5
<b>4</b>	<b>System Behavior</b>	<b>5</b>
4.1	General Overview . . . . .	5
4.2	Key Features and Functionality . . . . .	5
<b>5</b>	<b>Logical View</b>	<b>5</b>
5.1	High-Level Design (Architecture) . . . . .	6
5.1.1	TomTom.py . . . . .	6
5.2	Detailed Class Design . . . . .	6
5.2.1	TomTom.py . . . . .	7
<b>6</b>	<b>Scenario View</b>	<b>8</b>
6.1	Use Cases . . . . .	9

# 1 Change History

Version: 0.42

Modifier: Isaiah Martinez

Date: 4/21/24

Description of Change: Finished TomTom API using Python. Made Python Script accessible via Command line.

---

Version: 0.19

Modifier: Isaiah Martinez

Date: 3/29/24

Description of Change: Added API for TomTom to obtain Images of Traffic Flow. Demo car scene implemented. Pathfinding added.

---

Version: 0.11

Modifier: Isaiah Martinez, Jae Molina, Anastasia Naydina

Date: 2/26/24

Description of Change: Simple Car model made. Looked at Related Works for process in utilizing Unity for traffic modeling. Looking for additional related works.

---

Version: 0.09

Modifier: Isaiah Martinez, Jae Molina, Anastasia Naydina

Date: 2/19/24

Description of Change: Discussed High Level Architecture of the project: Unity for modeling, C#/Python for helper script, Python for ML training, and JS for API connectivity. Added template to follow for documentation. Structured git repo directories. First, we will be working on a set amount of locations with small amount of available traffic data. Later, we hope to implement API connectivity to obtain traffic info and map data with more locations.

---

Version: 0.05

Modifier: Isaiah Martinez, Jae Molina, Anastasia Naydina

Date: 2/12/24

Description of Change: Made Git repository. Looked at scholarly articles for related works. Uploaded sample scholarly article to view. Laid out big ideas for project. Began work on models to be used in Unity.

---

# 2 Introduction

This document describes the work put in to create a program that runs in Unity to model traffic in 3D. It allows for users to determine a start position, as well as a desired ending position which will follow traffic rules to ensure a fast and legal guide.

The code is able to be viewed on Github at this [Link](#).

## 2.1 Background

Traffic congestion is a very common experience for those who live in Los Angeles [1]. In order to address this problem many apps such as Waze and Google Maps were created [2]. Following in the style of these applications, we wanted to create software that follows a similar purpose: to provide a good model for traffic flow and path to a destination with this information.

## 2.2 Related Works

We were interested in trying to model a similar outcome as outlined in this paper [3]. In order to achieve this, we would utilize Unity, a 3rd party package called EasyRoads [6], MLAgents [5], and an external source for the traffic data. To obtain the traffic data, we would utilize real-time traffic information from TomTom [7]. We initially were hoping to utilize ML Learning Agents to do something similar to what was done in this Youtube video [4]. In the end we were decided against implementing ML Learning Agents for our project, as we would utilize Unity's built-in pathfinding capabilities.

## 2.3 Stakeholders

The Major Stakeholders are:

1. Users: they want a product that will work by giving accurate and detailed information of traffic. Additionally, they want the data to be useful for determining the route to take from their source destination to the desired ending destination.
2. Developers: they want to have a simplistic model for creating, modifying, and debugging the project across its various languages, scripts, and so forth. Development would be made easy by utilizing professional tools, technologies, and features.

## 2.4 Document Structure

This document contains a series of 3 main sections:

1. Design Goals: This section outlines the main objectives and guiding principles that drive the design of the software project. It provides a high-level vision of what the software aims to achieve in terms of functionality, usability, performance, and other key aspects.
2. System Behavior: This section describes how the system behaves as a whole and its overall functionality. It should give readers a clear understanding of the system's expected actions and reactions in different scenarios.
3. Logical View: The logical view provides a conceptual model of the software's architecture and its components. This includes how the software is organized and how its different parts work together.

4. Scenario View: The scenario view illustrates how the system behaves in specific use cases or scenarios. It helps to show how the system meets the requirements in practice.

We will explain the project utilizing UML diagrams, User-Interaction Diagrams, Class Diagrams, and other commonly used professional Software Development Models. This is to ensure that our project is explained with the utmost clarity, and conciseness.

## 2.5 Responsibilities

Isaiah: API connectivity. Obtained the Traffic Data and Map Data through the use of a Python Script and TomTom's developer API. Added ML Agents to the unity project.

Jae:

Anastasia:

## 3 Design Goals

This section outlines the main objectives and guiding principles that drive the design of the software project. It provides a high-level vision of what the software aims to achieve in terms of functionality, usability, performance, and other key aspects.

### 3.1 Purpose

### 3.2 Objectives

what were our initial goals for the project? were they met? why/why not. how does our finished project compare to that of our initial vision?

## 4 System Behavior

This section describes how the system behaves as a whole and its overall functionality. It should give readers a clear understanding of the system's expected actions and reactions in different scenarios.

### 4.1 General Overview

what is the project? what is the core of the project?

### 4.2 Key Features and Functionality

what are some cool things that we made/implemented?

## 5 Logical View

The logical view provides a conceptual model of the software's architecture and its components. This includes how the software is organized and how its different parts work together. This will be the largest section of the document.

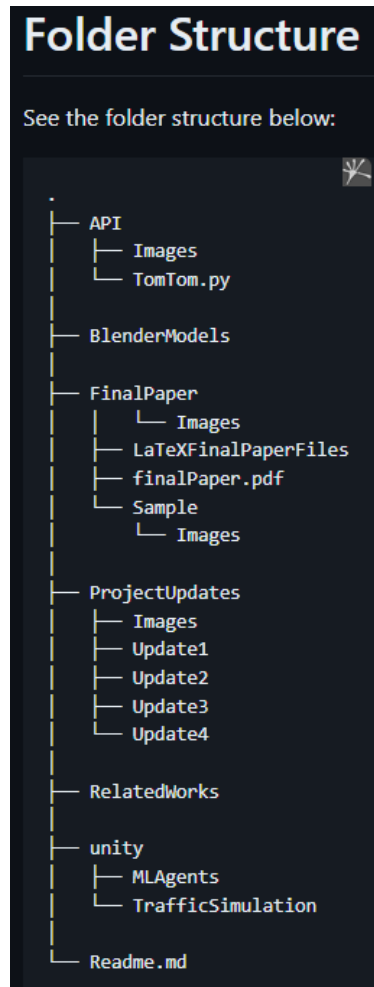


Figure 1: Folder Structure of the project.

## 5.1 High-Level Design (Architecture)

Another diagram be a layered image for each section of the project from unity to connecting to the python script, pathfinding, etc. each layer should be explained as well.

### 5.1.1 TomTom.py

Below is a class diagram of the TomTom API.

## 5.2 Detailed Class Design

should show a list of all functions and classes used to create the project  
all classes/functions should have: -Inputs -Purpose -Output

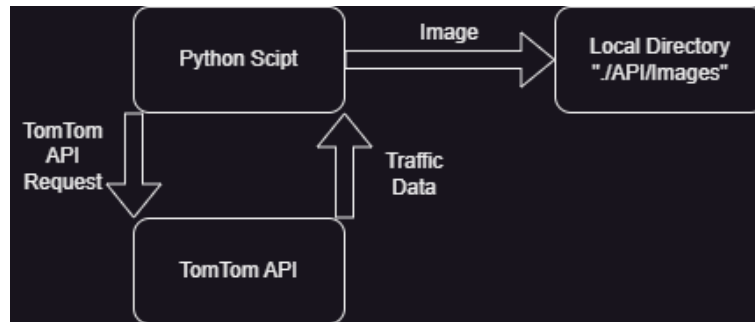


Figure 2: Hierarchy of the script.

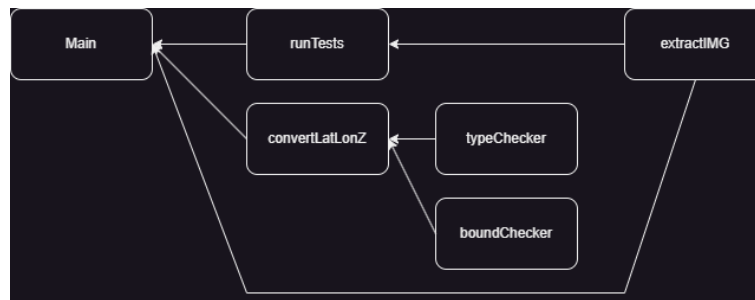


Figure 3: Hierarchy of the script.

### 5.2.1 TomTom.py

- extractIMG
  - **Inputs:** (*String*) HTML link, (*Boolean*) Save the image if true
  - **Purpose:** given the HTML link, connects to the TomTom API to obtain the desired image. If instructed, will also save the image in the local folder: './API/Images'
  - **Output:** N/A
- boundChecker
  - **Inputs:** (*Float*) Latitude, (*Float*) Longitude, (*Int*) Zoom level, (*String*) Style of image, (*Boolean*) Verbose option. If true prints to console about checking bounds for given inputs
  - **Purpose:** Ensures proper bounds on the given inputs
  - **Output:** (*Boolean*) True means that the inputs are properly within the bounds
- typeChecker
  - **Inputs:** (*Float*) Latitude, (*Float*) Longitude, (*Int*) Zoom level, (*String*) Style of image, (*Boolean*) Verbose option. If true prints to console about checking types for given inputs

- **Purpose:** Checks the types for all parameters given
- **Output:** (*Boolean*) True means that the inputs are of the proper type
- convertLatLonZ
  - **Inputs:** (*Float*) Latitude, (*Float*) Longitude, (*Int*) Zoom level, (*String*) Style of image, (*Boolean*) Verbose option. If true prints to console about converting parameters to proper format
  - **Purpose:** Determines the coordinates to use according to the zoom level. This information is needed when accessing the API to obtain the correct location information [8].
  - **Output:** (*Tuple*) Tuple of 3 elements: X, Y, Zoom level
- heightCalc
  - **Inputs:** (*Int*) Red value of pixel, (*Int*) Green value of pixel, (*Int*) Blue value of pixel
  - **Purpose:** Determines the height of a given pixel given the RGB values of the image. This formula is from TomTom [9]. Unused in the final product
  - **Output:** (*Float*) Height of the region described by the pixel as feet above sea level
- runTests
  - **Inputs:** (*Boolean*) Save the images if true, (*Boolean*) Verbose option. Print to the console about each step
  - **Purpose:** Will run a series of tests with default locations to show the different options available for this script
  - **Output:** N/A
- main
  - **Inputs:** (*String*) Latitude, (*String*) Longitude, (*String*) Zoom Level, (*String*) Style of Image to produce, (*Boolean*) Save image created if true, (*String*) Run demo options to showcase all options from this script if true, (*String*) Verbose. If true, describes the steps occurring in the program as they occur
  - **Purpose:** Parses the given arguments, runs through each of the above described functions to achieve the goal of obtaining the image with the given arguments
  - **Output:** N/A

## 6 Scenario View

The scenario view illustrates how the system behaves in specific use cases or scenarios. It helps to show how the system meets the requirements in practice.



## 6.1 Use Cases

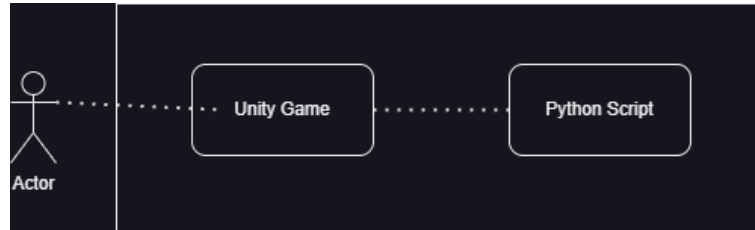


Figure 4: Hierarchy of the entire project.

## References

- [1] A. L. CNN Illustrations by Natalie Leung, “Los Angeles’ traffic problem in graphics,” CNN, Feb. 27, 2018. [Link](#)
- [2] “The 7 Best Traffic Apps of 2023,” Lifewire. [Link](#)
- [3] Teo Niemirepo, J. Toivonen, Marko Viitanen, and J. Vanne, “Open-Source CiThruS Simulation Environment for Real-Time 360-Degree Traffic Imaging,” Nov. 2019, doi: <https://doi.org/10.1109/iccve45908.2019.8965242>.
- [4] “Training an unbeatable AI in Trackmania,” [www.youtube.com](http://www.youtube.com). Youtube [Link](#)
- [5] U. Technologies, “Make a more engaging game w/ ML-Agents — Machine learning bots for game development — Reinforcement learning — Unity,” [unity.com](http://unity.com). [Link](#)
- [6] “EasyRoads3D Free v3 — 3D Characters — Unity Asset Store,” [asset-store.unity.com](http://asset-store.unity.com). Unity Store [Link](#)
- [7] “Traffic APIs,” TomTom. [Link](#).
- [8] “Zoom Levels and Tile Grid — Map Display API,” [developer.tomtom.com](http://developer.tomtom.com). [Link](#).
- [9] “Hillshade Tile — Map Display API,” [developer.tomtom.com](http://developer.tomtom.com). [Link](#).