Traffic Control System using fuzzy logic

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

SRI PAVAN POLISETTI RA2111030010269
GEETHIKA RA2111030010267

Under the guidance of

Mrs.R.Sujatha

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"Traffic Control System using fuzzy logic"** is the bona fide work of **SRI PAVAN POLISETTI RA2111030010269 , GEETHIKA RA2111030010267** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs.R.Sujatha

Assistant Professor

Department of Networking
and Communications

ABSTRACT

Traffic congestion at intersections poses significant challenges in urban areas, impacting efficiency, environmental sustainability, and quality of life. Conventional fixed-timing traffic light control systems struggle to adapt to dynamic traffic conditions, exacerbating congestion issues. This project introduces an innovative solution: an adaptive traffic light controller utilizing fuzzy logic, specifically the Sugeno method. The system dynamically calculates optimal green light durations based on real-time traffic density data, incorporating factors like regulated and opposing lane densities. Through fuzzy logic, the controller aligns green light durations with current traffic conditions, aiming to optimize traffic flow and alleviate congestion. The project's significance lies in addressing the limitations of traditional traffic management systems and contributing to more efficient urban transportation networks. Challenges include developing a robust fuzzy logic-based algorithm capable of ensuring safe and efficient traffic movement. The proposed system integrates advanced technologies such as Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs) for data processing and prediction. Future work involves enhancing model performance and real-time responsiveness, with the goal of improving intersection efficiency and reducing congestion in urban environments. Evaluation metrics include average vehicle delay, queue lengths, and intersection throughput. Real-world implementation insights and user feedback will inform future enhancements,ensuringthesystem's effectiveness and scalability.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

CNN Convolutional Neural Networks

ANN Artificial Neural Networks

DDC Dedicated Development Center

INTRODUCTION

Traffic congestion is a pervasive issue in urban areas worldwide, presenting significant challenges to transportation systems, economic productivity, and quality of life. At the heart of this problem lies the intersection, where multiple traffic streams converge and compete for limited road space. Traditional traffic light control systems, characterized by fixed timings and static programming, struggle to accommodate the dynamic nature of traffic flows, often exacerbating congestion and delays.

In response to these challenges, there is a growing recognition of the need for innovative and adaptive solutions to manage urban traffic more effectively. This project focuses on the development of an adaptive traffic light controller using fuzzy logic, specifically the Sugeno method, to address the shortcomings of conventional traffic management systems. By leveraging fuzzy logic, which allows for the modeling of uncertainty and imprecision inherent in real-world traffic conditions, the controller aims to dynamically adjust signal timings based on real-time traffic data, optimizing intersection efficiency and alleviating congestion.

The significance of this project lies in its potential to revolutionize urban traffic management by introducing a more flexible and responsive approach to traffic signal control. By calculating optimal green light durations based on factors such as traffic density, lane occupancy, and pedestrian crossings, the adaptive controller can adapt to changing traffic patterns and prioritize the most critical traffic movements. This, in turn, can lead to reduced travel times, improved fuel efficiency, and enhanced safety for road users.

Furthermore, the utilization of fuzzy logic enables the controller to incorporate human-like reasoning and decision-making, making it more robust and adaptable to varying traffic conditions. This approach represents a departure from traditional rule-based systems, offering greater flexibility and scalability in managing complex urban traffic environments.

Through this project, we aim to contribute to the ongoing efforts to create smarter, more sustainable cities by harnessing the power of advanced technologies and intelligent traffic management systems. By developing and implementing an adaptive traffic light controller using fuzzy logic, we hope to pave the way for a future where traffic congestion is no longer a barrier to urban mobility, but rather a manageable aspect of modern urban living.

LITERATURE SURVEY

Traffic Congestion and Management:
Traffic congestion is a pervasive issue in urban areas, leading to increased travel times, fuel consumption, and environmental pollution. Numerous studies have highlighted the detrimental effects of congestion on urban mobility and quality of life (Schrank et al., 2019). Intersections, where multiple traffic streams converge, are particularly susceptible to congestion due to conflicting movements and limited capacity (Ahmed et al., 2016). Traditional fixed-timing traffic light control systems exacerbate congestion by failing to adapt to changing traffic conditions (Koonce et al., 2017).

Fuzzy Logic Applications in Traffic Control:
Fuzzy logic has emerged as a promising approach for adaptive traffic control due to its ability to handle uncertainty and imprecision in traffic data (Zhang & Zhao, 2008). Previous research has demonstrated the effectiveness of fuzzy logic in optimizing traffic signal timings based on real-time traffic conditions (Feng & Jiang, 2010). By incorporating human-like reasoning and decision-making, fuzzy logic controllers can adapt to dynamic traffic situations more effectively than traditional methods (Liu et al., 2015). Fuzzy logic-based traffic control systems have shown promising results in improving intersection efficiency and reducing congestion (Ramezani & Barmshoori, 2016).

Sugeno Method in Fuzzy Logic Control:
The Sugeno method, a type of fuzzy logic control, has gained popularity for its ability to model complex systems and make precise control decisions (Mendel, 2001). Unlike Mamdani-type fuzzy systems, which use linguistic rules to determine output, the Sugeno method employs a mathematical function to calculate output based on input variables (Sugeno, 1985). This approach offers advantages in terms of computational efficiency and transparency, making it suitable for real-time control applications (Zhang & Zhao, 2008). The Sugeno method has been successfully applied in various domains, including traffic control, where its adaptability and accuracy are highly beneficial (Feng & Jiang, 2010).

Existing Adaptive Traffic Control Systems:
Several adaptive traffic control systems have been developed to address the limitations of traditional fixed-timing systems. These systems utilize different approaches, including machine learning, optimization algorithms, and fuzzy logic. While machine learning-based approaches offer the advantage of learning from historical data, fuzzy logic controllers excel in handling uncertain and dynamic traffic conditions (Liu et al., 2015). Previous adaptive systems have shown improvements in intersection efficiency and congestion reduction, albeit with varying degrees of success (Ramezani &

Barmshoori, 2016).

Overall, the literature survey highlights the pressing need for adaptive traffic control systems capable of responding dynamically to changing traffic conditions. Fuzzy logic, particularly the Sugeno method, offers a promising approach for achieving this goal, with potential benefits in terms of intersection efficiency and congestion alleviation. Future research should focus on further refining fuzzy logic-based controllers and evluating their effectiveness in real-world urban environmen

METHODOLOGY

Methodological Steps:

1. **Problem Identification and Project Scope Definition:**
    a. Define the scope of the project by identifying key objectives and constraints.
    b. Conduct a thorough analysis of the existing traffic management systems and identify the limitations of traditional fixed-timing traffic light control systems.
2. **Literature Review:**
    a. Conduct a comprehensive review of existing literature on traffic congestion, traffic control systems, fuzzy logic applications, and adaptive traffic management.
    b. Analyze previous research studies, methodologies, and findings related to adaptive traffic light control using fuzzy logic.
3. **Requirement Analysis:**
    a. Define the functional and non-functional requirements of the adaptive traffic light controller.
    b. Identify the necessary hardware, software, and data sources required for the implementation.
    c. Determine the performance metrics for evaluating the effectiveness of the system, such as intersection throughput, average vehicle delay, and congestion levels.
4. **System Design:**
    a. Design the architecture of the adaptive traffic light controller, including hardware components, software modules, and data flow.

b. Define the data collection mechanisms, preprocessing steps, fuzzy logic modeling approach, and control algorithm.

c. Specify the communication protocols and interfaces for integrating the controller with existing traffic infrastructure.

5. **Data Collection and Preprocessing:**

   a. Implement data collection mechanisms using sensors or cameras installed at intersections to capture real-time traffic data.

   b. Preprocess the raw traffic data to remove noise, outliers,and inconsistencies.

   c. Aggregate and organize the preprocessed data for input into the fuzzy logic controller.

6. **Fuzzy Logic Modeling:**

   a. Define linguistic variables, membership functions, and fuzzy rules to represent the traffic control logic.

   b. Determine the input variables based on the preprocessed traffic data, such as traffic density, lane occupancy, and vehicle speed.

   c. Formulate fuzzy rules to describe the relationship between input variables and control actions, considering factors like regulated lane density and opposing lane densities.

7. **Algorithm Development:**

   a. Implement the fuzzy logic controller using the Sugeno method to calculate optimal green light durations.

   b. Develop adaptive mechanisms to dynamically adjust signal timings based on real-time traffic conditions.

   c. Test and validate the algorithm using simulation models to ensure accurate and efficient traffic control.

8. **Simulation and Testing:**

   a. Conduct simulations to evaluate the performance of the adaptive traffic light controller under various traffic scenarios.

   b. Test the controller's ability to optimize traffic flow, reduce delays, and minimize congestion levels.

   c. Validate the effectiveness of the controller in improving intersection efficiency and overall traffic management.

9. **Implementation and Deployment**:

a. Implement the adaptive traffic light controller at selected intersections for field testing.

b. Integrate the controller with existing traffic infrastructure and ensure compatibility with local regulations and standards.

c. Monitor the performance of the deployed system and make necessary adjustments to optimize its operation.

10.Evaluation and Optimization:

a. Evaluate the performance of the deployed system based on predefined performance metrics.

b. Identify areas for improvement and optimization based on feedback from field testing and real-world operation.

c. Continuously monitor and refine the system to enhance its effectiveness in alleviating congestion and improving urban mobility.

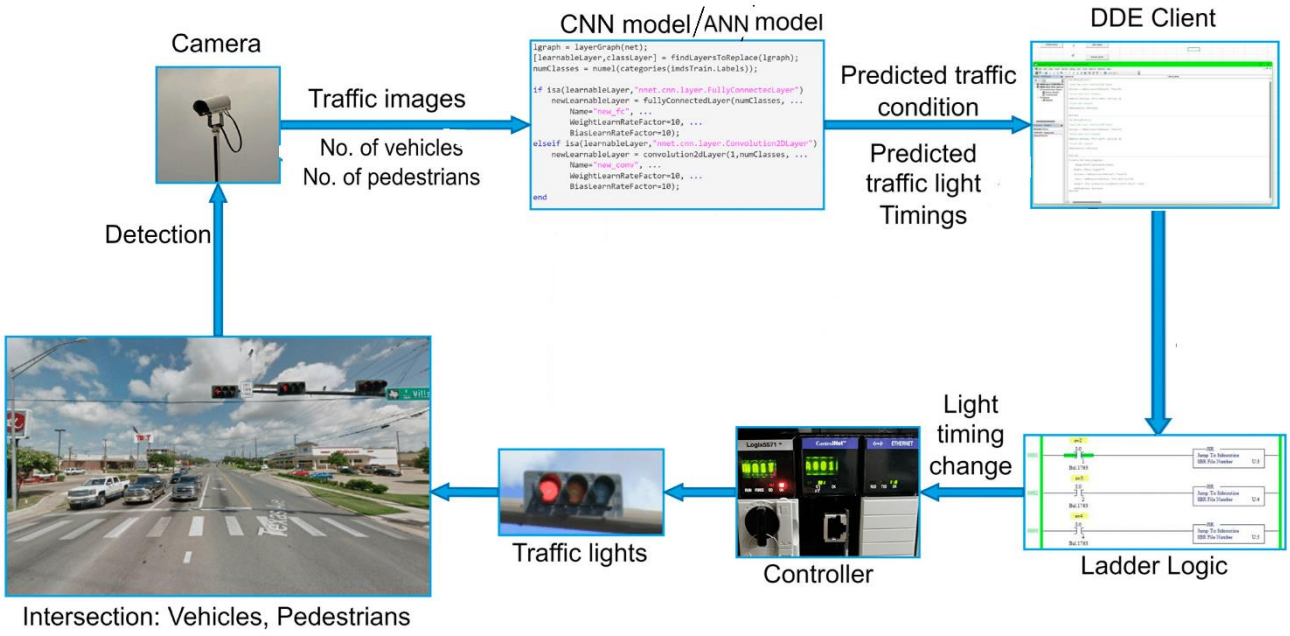

fig-1 : Architecture block

CODING AND TESTING

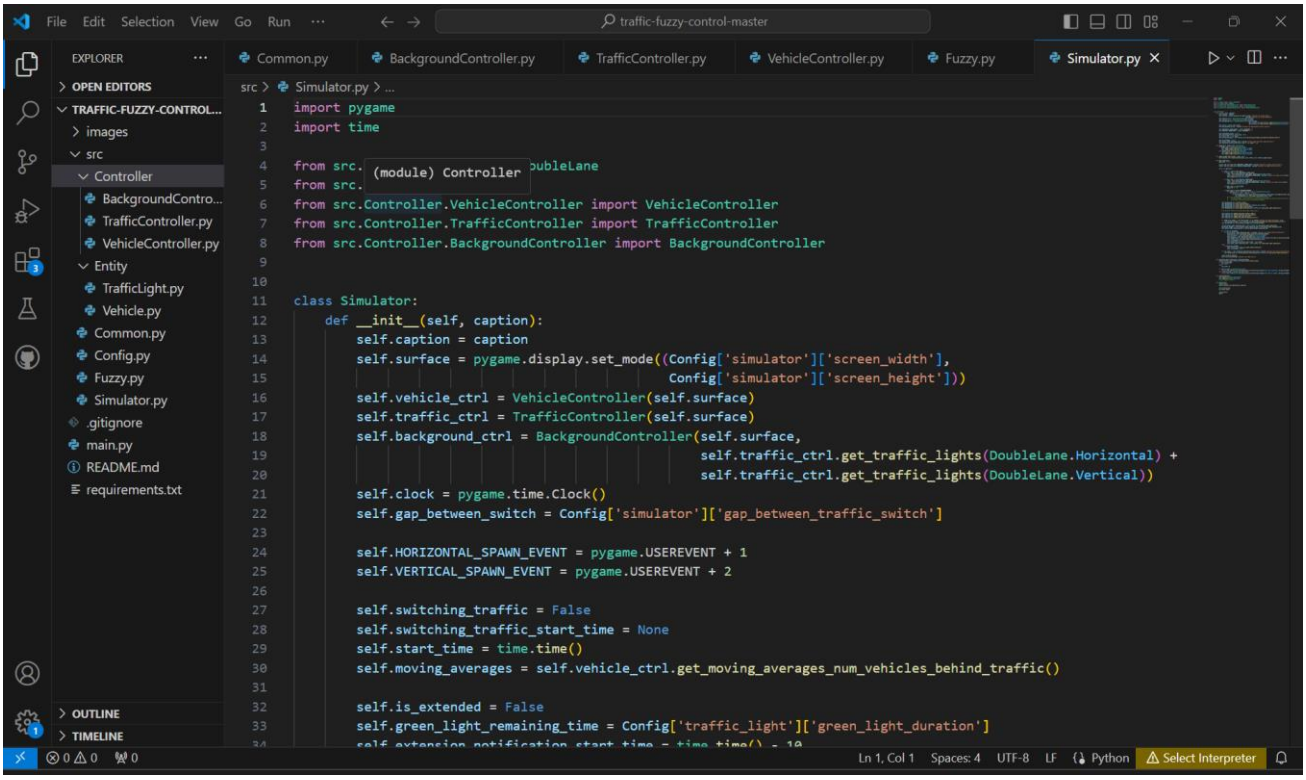

fig-2 : List of code modules

Src/Controller/BackgroundController.py:

```
import pygame
import os
from src.Common import DoubleLane, Lane
from src.Config import Config
class BackgroundController:
    def __init__(self, surface, traffic_lights):
        self.surface = surface
        self.traffic_lights = traffic_lights

        self.screen_height = Config['simulator']['screen_height']
        self.screen_width = Config['simulator']['screen_width']

        self.black = Config['colors']['black']
        self.red = Config['colors']['red']

        self.horizontal_frequency_small = None
        self.horizontal_frequency_medium = None
        self.horizontal_frequency_large = None
```

```
    self.vertical_frequency_small = None
    self.horizontal_frequency_medium = None
    self.horizontal_frequency_large = None

    self.spawn_rate = {
        DoubleLane.Horizontal: {
            'slow': True,
            'medium': False,
            'fast': False
        },
        DoubleLane.Vertical: {
            'slow': True,
            'medium': False,
            'fast': False
        }
    }

    self.spawn_rate_buttons = {
        DoubleLane.Horizontal: {
            'slow': None,
            'medium': None,
            'fast': None
        },
        DoubleLane.Vertical: {
            'slow': None,
            'medium': None,
            'fast': None
        }
    }

    self.switch_traffic_button = None
    self.fuzzy_button = None

def set_spawn_rate(self, double_lane: DoubleLane, target_rate):
    for rate in ['slow', 'medium', 'fast']:
        self.spawn_rate[double_lane][rate] = (target_rate == rate)

def get_spawn_rate(self, double_lane: DoubleLane):
    for rate in ['slow', 'medium', 'fast']:
        if self.spawn_rate[double_lane][rate]:
            return rate
    raise Exception('None of slow, medium, fast is true!!!')
```

```
    def refresh_screen(self):
        self.surface.fill(Config['colors']['white'])

    def draw_spawn_rate_buttons(self):
        normal_font = pygame.font.SysFont('Comic Sans MS', 16)
        underline_font = pygame.font.SysFont('Comic Sans MS', 16)
        underline_font.set_underline(True)

        # Horizontal lanes controls
        self.surface.blit(normal_font.render('Spawn Rate (Horizontal):', True, self.black),
(5, 25))
        fonts = [normal_font, normal_font, normal_font]
        colors = [self.black, self.black, self.black]
        if self.spawn_rate[DoubleLane.Horizontal]['slow']:
            fonts[0] = underline_font
            colors[0] = self.red
        if self.spawn_rate[DoubleLane.Horizontal]['medium']:
            fonts[1] = underline_font
            colors[1] = self.red
        if self.spawn_rate[DoubleLane.Horizontal]['fast']:
            fonts[2] = underline_font
            colors[2] = self.red
        self.spawn_rate_buttons[DoubleLane.Horizontal]['slow'] =
self.surface.blit(fonts[0].render('Slow', True, colors[0]), (200, 25))
        self.spawn_rate_buttons[DoubleLane.Horizontal]['medium'] =
self.surface.blit(fonts[1].render('Medium', True, colors[1]), (240, 25))
        self.spawn_rate_buttons[DoubleLane.Horizontal]['fast'] =
self.surface.blit(fonts[2].render('Fast', True, colors[2]), (300, 25))

        # Vertical lanes controls
        self.surface.blit(normal_font.render('Spawn Rate (Vertical):', True, self.black),
(5, 45))
        fonts = [normal_font, normal_font, normal_font]
        colors = [self.black, self.black, self.black]
        if self.spawn_rate[DoubleLane.Vertical]['slow']:
            fonts[0] = underline_font
            colors[0] = self.red
        if self.spawn_rate[DoubleLane.Vertical]['medium']:
            fonts[1] = underline_font
            colors[1] = self.red
        if self.spawn_rate[DoubleLane.Vertical]['fast']:
            fonts[2] = underline_font
```

```
            colors[2] = self.red
        self.spawn_rate_buttons[DoubleLane.Vertical]['slow'] =
self.surface.blit(fonts[0].render('Slow', True, colors[0]), (200, 45))
        self.spawn_rate_buttons[DoubleLane.Vertical]['medium'] =
self.surface.blit(fonts[1].render('Medium', True, colors[1]), (240, 45))
        self.spawn_rate_buttons[DoubleLane.Vertical]['fast'] =
self.surface.blit(fonts[2].render('Fast', True, colors[2]), (300, 45))

    def draw_moving_averages(self, moving_averages):
        normal_font = pygame.font.SysFont('Comic Sans MS', 16)
        self.surface.blit(normal_font.render('Vehicles behind traffic light (Horizontal):',
True, self.black), (5, 65))

self.surface.blit(normal_font.render('{0:.2f}'.format(moving_averages[Lane.left_to_ri
ght]), True, self.black), (320, 65))
        self.surface.blit(normal_font.render('Vehicles behind traffic light (Vertical):',
True, self.black), (5, 85))

self.surface.blit(normal_font.render('{0:.2f}'.format(moving_averages[Lane.top_to_bo
ttom]), True, self.black), (320, 85))

    def draw_vehicle_count(self, total):
        font = pygame.font.SysFont('Comic Sans MS', 16)
        text_surface = font.render('Total Vehicles: {}'.format(total), True, self.black)
        self.surface.blit(text_surface, (5, 5))

    def draw_road_markings(self):
        bumper_distance = Config['simulator']['bumper_distance']
        vehicle_body_width = Config['vehicle']['body_width']
        road_marking_width = Config['background']['road_marking_width']
        road_marking_length, road_marking_distance =
Config['background']['road_marking_alternate_lengths']
        traffic_yellow = Config['colors']['traffic_yellow']
        gap = Config['background']['road_marking_gap_from_yellow_box']

        # yellow box
        yb_top, yb_left, yb_bottom, yb_right =
Config['background']['yellow_box_junction']
        yellow_box_junction_img = os.path.join(os.getcwd(), 'images', 'junction',
'yellow_box_junction.png')
        picture = pygame.image.load(yellow_box_junction_img)
        picture = pygame.transform.scale(picture, (yb_left + yb_right, yb_top +
yb_bottom))
```

```
        self.surface.blit(picture, (self.screen_width / 2 - yb_left, self.screen_height / 2 - yb_top))

        # lane from bottom to top
        # lane from top to bottom
        x1 = self.screen_width / 2 - bumper_distance - vehicle_body_width / 2 - road_marking_width / 2
        x2 = self.screen_width / 2 + bumper_distance + vehicle_body_width / 2 - road_marking_width / 2
        y = self.screen_height / 2 - yb_top - road_marking_length - road_marking_distance
        while y >= 0:
            pygame.draw.rect(self.surface, traffic_yellow, (x1, y, road_marking_width, road_marking_length))
            pygame.draw.rect(self.surface, traffic_yellow, (x2, y, road_marking_width, road_marking_length))
            y -= road_marking_length + road_marking_distance
        y = self.screen_height / 2 + yb_bottom + gap
        while y <= self.screen_height:
            pygame.draw.rect(self.surface, traffic_yellow, (x1, y, road_marking_width, road_marking_length))
            pygame.draw.rect(self.surface, traffic_yellow, (x2, y, road_marking_width, road_marking_length))
            y += road_marking_length + road_marking_distance

        # lane from left to right
        # lane from right to left
        x = self.screen_width / 2 - yb_left - road_marking_length - gap
        y1 = self.screen_height / 2 - bumper_distance - vehicle_body_width / 2 - road_marking_width / 2
        y2 = self.screen_height / 2 + bumper_distance + vehicle_body_width / 2 - road_marking_width / 2
        while x >= 0:
            pygame.draw.rect(self.surface, traffic_yellow, (x, y1, road_marking_length, road_marking_width))
            pygame.draw.rect(self.surface, traffic_yellow, (x, y2, road_marking_length, road_marking_width))
            x -= road_marking_length + road_marking_distance
        x = self.screen_width / 2 + yb_right + gap
        while x <= self.screen_width:
            pygame.draw.rect(self.surface, traffic_yellow, (x, y1, road_marking_length, road_marking_width))
            pygame.draw.rect(self.surface, traffic_yellow, (x, y2, road_marking_length,
```

```
road_marking_width))
            x += road_marking_length + road_marking_distance

    def within_boundary(self, x, y):
        return 0 <= x <= self.screen_width and 0 <= y <= self.screen_height

    def draw_switch_traffic_button(self):
        font = pygame.font.SysFont('Comic Sans MS', 16)
        text_surface = font.render('Switch', True, self.black)
        rect = self.surface.blit(text_surface, (self.screen_width - 100, 20))
        gap = 5
        x = rect.left - gap
        y = rect.top - gap
        w = rect.width + gap * 2
        h = rect.height + gap * 2
        pygame.draw.rect(self.surface, self.black, (x, y, w, h), 3)
        self.switch_traffic_button = rect

    def draw_fuzzy_button(self):
        font = pygame.font.SysFont('Comic Sans MS', 16)
        text_surface = font.render('Calculate Fuzzy', True, self.black)
        rect = self.surface.blit(text_surface, (self.screen_width - 150, 90))
        gap = 5
        x = rect.left - gap
        y = rect.top - gap
        w = rect.width + gap * 2
        h = rect.height + gap * 2
        pygame.draw.rect(self.surface, self.black, (x, y, w, h), 3)
        self.fuzzy_button = rect

    def draw_fuzzy_score(self, fuzzy_score, current_lane: DoubleLane):
        normal_font = pygame.font.SysFont('Comic Sans MS', 16)
        opposite_lane_name = 'Horizontal' if current_lane == DoubleLane.Vertical else
'Vertical'
        self.surface.blit(normal_font.render('Fuzzy Green Light Ext. ({} Lane):
'.format(opposite_lane_name), True, self.black), (5, 105))

        score = '-'
        if fuzzy_score:
            score = '{:.2f}s'.format(fuzzy_score)
        self.surface.blit(normal_font.render(score, True, self.black), (320, 105))

    def draw_extension_notification(self, extension, horizontal, vertical):
```

```
        normal_font = pygame.font.SysFont('Comic Sans MS', 16)

        self.surface.blit(normal_font.render('Vehicle behind Traffic Light  ', True,
Config['colors']['traffic_green']), (5, 125))
        self.surface.blit(normal_font.render('     Horizontal : ', True,
Config['colors']['traffic_green']), (5, 145))
        self.surface.blit(normal_font.render('{:.1f}'.format(horizontal), True,
Config['colors']['traffic_green']), (200, 145))
        self.surface.blit(normal_font.render('     Vertical :', True,
Config['colors']['traffic_green']), (5, 165))
        self.surface.blit(normal_font.render('{:.1f}'.format(vertical), True,
Config['colors']['traffic_green']), (200, 165))
        self.surface.blit(normal_font.render('Green light is extended by
{:.1f}!'.format(extension), True, Config['colors']['traffic_green']), (5, 185))

    def draw_light_durations(self, green_light_extension):
        normal_font = pygame.font.SysFont('Comic Sans MS', 16)
        green_duration = Config['traffic_light']['green_light_duration']
        yellow_duration = Config['traffic_light']['yellow_light_duration']
        red_duration = Config['traffic_light']['red_light_duration']

        pygame.draw.circle(self.surface, Config['colors']['traffic_red'], (self.screen_width
- 180, 16), 8)
        self.surface.blit(normal_font.render('Duration: {:.1f}'.format(red_duration), True,
self.black),
                          (self.screen_width - 160, 5))

        pygame.draw.circle(self.surface, Config['colors']['traffic_yellow'],
(self.screen_width - 180, 36), 8)
        self.surface.blit(normal_font.render('Duration: {:.1f}'.format(yellow_duration),
True, self.black),
                          (self.screen_width - 160, 25))

        pygame.draw.circle(self.surface, Config['colors']['traffic_green'],
(self.screen_width - 180, 56), 8)
        if green_light_extension > 0:
            self.surface.blit(
                normal_font.render('Duration: {:.1f} + {:.1f}'.format(green_duration,
green_light_extension), True,
                                   self.black),
                (self.screen_width - 160, 45))
        else:
            self.surface.blit(normal_font.render('Duration: {:.1f}'.format(green_duration),
```

```
True, self.black),
                    (self.screen_width - 160, 45))
```

Src/TrafficController.py:

```
import os
import pygame

from src.Common import TrafficStatus, DoubleLane, Lane
from src.Config import Config
from src.Entity.TrafficLight import TrafficLight
from src.Fuzzy import Fuzzy


class TrafficController:
    def __init__(self, surface):
        self.surface = surface
        self.screen_height = Config['simulator']['screen_height']
        self.screen_width = Config['simulator']['screen_width']
        self.traffic_light_body_height = Config['traffic_light']['body_height']
        self.traffic_light_body_width = Config['traffic_light']['body_width']
        self.traffic_light_distance_from_center =
Config['traffic_light']['distance_from_center']

        self.traffic_lights = {}

        x = self.screen_width / 2 - self.traffic_light_distance_from_center[0] -
self.traffic_light_body_width
        y = self.screen_height / 2 - self.traffic_light_distance_from_center[1] -
self.traffic_light_body_height
        self.create_traffic_light(x, y, Lane.left_to_right)

        x = self.screen_width / 2 + self.traffic_light_distance_from_center[0]
        y = self.screen_height / 2 + self.traffic_light_distance_from_center[1]
        self.create_traffic_light(x, y, Lane.right_to_left)

        y = self.screen_width / 2 - self.traffic_light_distance_from_center[0] -
self.traffic_light_body_width
        x = self.screen_height / 2 + self.traffic_light_distance_from_center[1]
        self.create_traffic_light(x, y, Lane.top_to_bottom)

        y = self.screen_width / 2 + self.traffic_light_distance_from_center[0]
        x = self.screen_height / 2 - self.traffic_light_distance_from_center[1] -
```

```
self.traffic_light_body_height
        self.create_traffic_light(x, y, Lane.bottom_to_top)

        self.fuzzy = Fuzzy()

        self.latest_green_light_extension = 0

    def get_traffic_lights(self, double_lane: DoubleLane):
        if double_lane == DoubleLane.Horizontal:
            return [
                self.traffic_lights[Lane.left_to_right],
                self.traffic_lights[Lane.right_to_left]
            ]
        elif double_lane == DoubleLane.Vertical:
            return [
                self.traffic_lights[Lane.bottom_to_top],
                self.traffic_lights[Lane.top_to_bottom]
            ]
        return None

    def design_traffic_image(self, file_dir, filename, rotation):
        image = pygame.image.load(os.path.join(file_dir, filename))
        return pygame.transform.rotate(pygame.transform.scale(image,
                (self.traffic_light_body_width, self.traffic_light_body_height)), rotation)

    def create_traffic_light(self, x, y, lane: Lane):
        traffic_light_images_dir = os.path.join(os.getcwd(), 'images', 'traffic_light')
        rotation = 0
        if lane == Lane.bottom_to_top:
            rotation = 90
        elif lane == Lane.right_to_left:
            rotation = 180
        elif lane == Lane.top_to_bottom:
            rotation = 270
        traffic_light_images = {
            TrafficStatus.red: self.design_traffic_image(traffic_light_images_dir,
'traffic_light_red.png', rotation),
            TrafficStatus.green: self.design_traffic_image(traffic_light_images_dir,
'traffic_light_green.png', rotation),
            TrafficStatus.yellow: self.design_traffic_image(traffic_light_images_dir,
'traffic_light_yellow.png', rotation)
        }
        self.traffic_lights[lane] = TrafficLight(x, y, lane, traffic_light_images,
```

```
self.surface)

        if lane in [Lane.bottom_to_top, Lane.top_to_bottom]:
            self.traffic_lights[lane].change_status(TrafficStatus.red)

    def update_and_draw_traffic_lights(self):
        for lane, traffic_light in self.traffic_lights.items():
            opposite_status = self.get_opposite_status(lane)
            traffic_light.auto_update(opposite_status)
            traffic_light.draw()
            traffic_light.draw_countdown()

    def get_opposite_status(self, lane: Lane):
        if lane == Lane.right_to_left or \
           lane == Lane.left_to_right:
            return self.traffic_lights[Lane.bottom_to_top].status
        else:
            return self.traffic_lights[Lane.left_to_right].status

    def calculate_fuzzy_score(self, arriving_green_light_car, behind_red_light_car,
extension_count):
        return self.fuzzy.get_extension(arriving_green_light_car, behind_red_light_car,
extension_count)

    def get_current_active_lane(self)->DoubleLane:
        if self.traffic_lights[Lane.left_to_right].status == TrafficStatus.green:
            return DoubleLane.Horizontal
        elif self.traffic_lights[Lane.top_to_bottom].status == TrafficStatus.green:
            return DoubleLane.Vertical
        return None

    def get_green_light_extension(self):
        current_lane = self.get_current_active_lane()
        if not current_lane:
            return self.latest_green_light_extension if self.latest_green_light_extension
else 0
        if current_lane == DoubleLane.Vertical:
            self.latest_green_light_extension =
self.traffic_lights[Lane.bottom_to_top].duration_extension[
                TrafficStatus.green]
        elif current_lane == DoubleLane.Horizontal:
            self.latest_green_light_extension =
self.traffic_lights[Lane.left_to_right].duration_extension[
```

```
            TrafficStatus.green]
        return self.latest_green_light_extension

    def set_green_light_extension(self, extension):
        current_lane = self.get_current_active_lane()
        if not current_lane:
            return
        for tf in self.get_traffic_lights(current_lane):
            tf.set_green_light_extension(extension)

    def clear_all_green_light_extension(self):
        for lane, tf in self.traffic_lights.items():
            tf.set_green_light_extension(0)

    def get_green_light_remaining(self):
        current_lane = self.get_current_active_lane()
        remaining_seconds = 0
        if current_lane == DoubleLane.Vertical:
            remaining_seconds =
self.traffic_lights[Lane.bottom_to_top].get_green_light_remaining_time()
        elif current_lane == DoubleLane.Horizontal:
            remaining_seconds =
self.traffic_lights[Lane.left_to_right].get_green_light_remaining_time()
        return remaining_seconds

    def in_transition(self)->bool:
        return not self.get_current_active_lane()
```

src/Simulator.py:

```
import pygame
import time

from src.Common import Lane, DoubleLane
from src.Config import Config
from src.Controller.VehicleController import VehicleController
from src.Controller.TrafficController import TrafficController
from src.Controller.BackgroundController import BackgroundController


class Simulator:
    def __init__(self, caption):
        self.caption = caption
```

```
        self.surface = pygame.display.set_mode((Config['simulator']['screen_width'],
                                                Config['simulator']['screen_height']))
        self.vehicle_ctrl = VehicleController(self.surface)
        self.traffic_ctrl = TrafficController(self.surface)
        self.background_ctrl = BackgroundController(self.surface,

self.traffic_ctrl.get_traffic_lights(DoubleLane.Horizontal) +

self.traffic_ctrl.get_traffic_lights(DoubleLane.Vertical))
        self.clock = pygame.time.Clock()
        self.gap_between_switch = Config['simulator']['gap_between_traffic_switch']

        self.HORIZONTAL_SPAWN_EVENT = pygame.USEREVENT + 1
        self.VERTICAL_SPAWN_EVENT = pygame.USEREVENT + 2

        self.switching_traffic = False
        self.switching_traffic_start_time = None
        self.start_time = time.time()
        self.moving_averages =
self.vehicle_ctrl.get_moving_averages_num_vehicles_behind_traffic()

        self.is_extended = False
        self.green_light_remaining_time = Config['traffic_light']['green_light_duration']
        self.extension_notification_start_time = time.time() - 10

    def spawn(self, double_lane: DoubleLane):
        if double_lane == DoubleLane.Horizontal:
            self.spawn_single_vehicle(Lane.left_to_right)
            self.spawn_single_vehicle(Lane.right_to_left)
        elif double_lane == DoubleLane.Vertical:
            self.spawn_single_vehicle(Lane.bottom_to_top)
            self.spawn_single_vehicle(Lane.top_to_bottom)

    def spawn_single_vehicle(self, lane: Lane):
        self.vehicle_ctrl.create_vehicle(lane, self.traffic_ctrl.traffic_lights[lane])

    def main_loop(self):
        game_over = False

        pygame.time.set_timer(self.HORIZONTAL_SPAWN_EVENT,
Config['simulator']['spawn_rate']['slow'])
        pygame.time.set_timer(self.VERTICAL_SPAWN_EVENT,
Config['simulator']['spawn_rate']['slow'])
```

```
        while not game_over:

            for event in pygame.event.get():
                if event.type == self.HORIZONTAL_SPAWN_EVENT:
                    rate = self.background_ctrl.get_spawn_rate(DoubleLane.Horizontal)
                    pygame.time.set_timer(self.HORIZONTAL_SPAWN_EVENT,
Config['simulator']['spawn_rate'][rate])
                    self.spawn(DoubleLane.Horizontal)

                if event.type == self.VERTICAL_SPAWN_EVENT:
                    rate = self.background_ctrl.get_spawn_rate(DoubleLane.Vertical)
                    pygame.time.set_timer(self.VERTICAL_SPAWN_EVENT,
Config['simulator']['spawn_rate'][rate])
                    self.spawn(DoubleLane.Vertical)

                if event.type == pygame.QUIT:
                    game_over = True

                if event.type == pygame.MOUSEBUTTONDOWN:
                    for double_lane in [DoubleLane.Horizontal, DoubleLane.Vertical]:
                        for rate in ['slow', 'medium', 'fast']:
                            if
self.background_ctrl.spawn_rate_buttons[double_lane][rate].collidepoint(event.pos):
                                self.background_ctrl.set_spawn_rate(double_lane, rate)
                    # if self.background_ctrl.fuzzy_button.collidepoint(event.pos):
                    #     moving_averages =
self.vehicle_ctrl.get_moving_averages_num_vehicles_behind_traffic()
                    #     print(self.calculate_fuzzy_score(moving_averages))

            self.background_ctrl.refresh_screen()
            self.background_ctrl.draw_road_markings()
            self.background_ctrl.draw_vehicle_count(self.vehicle_ctrl.counter)
            self.background_ctrl.draw_spawn_rate_buttons()

self.background_ctrl.draw_light_durations(self.traffic_ctrl.get_green_light_extension(
))

            # print(self.traffic_ctrl.get_green_light_remaining())

            self.traffic_ctrl.update_and_draw_traffic_lights()
            self.vehicle_ctrl.destroy_vehicles_outside_canvas()
            self.vehicle_ctrl.update_and_draw_vehicles()
```

```
            self.vehicle_ctrl.update_num_vehicles_behind_traffic()

            if round((time.time() - self.start_time), 1) %
Config['simulator']['static_duration'] == 0:
                self.moving_averages =
self.vehicle_ctrl.get_moving_averages_num_vehicles_behind_traffic()
            self.background_ctrl.draw_moving_averages(self.moving_averages)

            current_green_light_remaining_time =
self.traffic_ctrl.get_green_light_remaining()
            direction_changed = current_green_light_remaining_time >
self.green_light_remaining_time
            self.green_light_remaining_time = current_green_light_remaining_time

            if not self.is_extended:
                if current_green_light_remaining_time <=
Config['simulator']['seconds_before_extension']:
                    fuzzy_score = self.calculate_fuzzy_score(self.moving_averages)
                    self.horizontal = self.moving_averages[Lane.left_to_right]
                    self.vertical = self.moving_averages[Lane.top_to_bottom]
                    self.background_ctrl.draw_fuzzy_score(fuzzy_score,
self.traffic_ctrl.get_current_active_lane())
                    self.traffic_ctrl.set_green_light_extension(fuzzy_score)
                    self.is_extended = True
                    self.extension_notification_start_time = time.time()
                    self.green_light_remaining_time =
self.traffic_ctrl.get_green_light_remaining()
            else:
                if direction_changed:
                    self.traffic_ctrl.clear_all_green_light_extension()
                    self.is_extended = False

            if time.time() - self.extension_notification_start_time <
Config['simulator']['fuzzy_notification_duration']:

self.background_ctrl.draw_extension_notification(self.traffic_ctrl.get_green_light_ext
ension(), self.horizontal, self.vertical)

            pygame.display.update()
            self.clock.tick(Config['simulator']['frame_rate'])

    def calculate_fuzzy_score(self, moving_averages):
        traffic_state = self.traffic_ctrl.get_current_active_lane()
```

```
        if self.is_extended :
            ext_count = 1
        else:
            ext_count =0

        if traffic_state == DoubleLane.Vertical:
            return
self.traffic_ctrl.calculate_fuzzy_score(moving_averages[Lane.top_to_bottom],
moving_averages[Lane.left_to_right], ext_count)
        elif traffic_state == DoubleLane.Horizontal:
            return
self.traffic_ctrl.calculate_fuzzy_score(moving_averages[Lane.left_to_right],
moving_averages[Lane.top_to_bottom], ext_count)

    def initialize(self):
        self.spawn(DoubleLane.Horizontal)
        self.spawn(DoubleLane.Vertical)
        # self.toggle_traffic()

    def start(self):
        pygame.init()
        pygame.display.set_caption(self.caption)

        self.initialize()
        self.main_loop()

        pygame.quit()
        quit()
```

src/Fuzzy.py:

```
import numpy as np
import skfuzzy as fuzz
from src.Config import Config


class Fuzzy:

    def __init__(self):
        setting = Config['fuzzy']['range']
        self.x_behind_red_light = setting['behind_red_light']
        self.x_arriving_green_light = setting['arriving_green_light']
        self.x_extension = setting['extension']
```

```
        setting = Config['fuzzy']['membership_function']['arriving_green_light']
        self.arriving_green_light_few = fuzz.trimf(self.x_arriving_green_light,
setting['few'])
        self.arriving_green_light_small = fuzz.trimf(self.x_arriving_green_light,
setting['small'])
        self.arriving_green_light_medium = fuzz.trimf(self.x_arriving_green_light,
setting['medium'])
        self.arriving_green_light_many = fuzz.trimf(self.x_arriving_green_light,
setting['many'])

        setting = Config['fuzzy']['membership_function']['behind_red_light']
        self.behind_red_light_few = fuzz.trimf(self.x_behind_red_light, setting['few'])
        self.behind_red_light_small = fuzz.trimf(self.x_behind_red_light, setting['small'])
        self.behind_red_light_medium = fuzz.trimf(self.x_behind_red_light,
setting['medium'])
        self.behind_red_light_many = fuzz.trimf(self.x_behind_red_light,
setting['many'])

        setting = Config['fuzzy']['membership_function']['extension']
        self.extension_zero = fuzz.trimf(self.x_extension, setting['zero'])
        self.extension_short = fuzz.trimf(self.x_extension, setting['short'])
        self.extension_medium = fuzz.trimf(self.x_extension, setting['medium'])
        self.extension_long = fuzz.trimf(self.x_extension, setting['long'])

    def get_extension(self, arriving_green_light_car, behind_red_light_car,
extension_count):
        behind_red_light_level_few = fuzz.interp_membership(self.x_behind_red_light,
self.behind_red_light_few, behind_red_light_car)
        behind_red_light_level_small =
fuzz.interp_membership(self.x_behind_red_light, self.behind_red_light_small,
behind_red_light_car)
        behind_red_light_level_medium =
fuzz.interp_membership(self.x_behind_red_light, self.behind_red_light_medium,
behind_red_light_car)
        behind_red_light_level_many =
fuzz.interp_membership(self.x_behind_red_light, self.behind_red_light_many,
behind_red_light_car)

        arriving_green_light_level_few =
fuzz.interp_membership(self.x_arriving_green_light, self.arriving_green_light_few,
arriving_green_light_car)
        arriving_green_light_level_small =
```

```
fuzz.interp_membership(self.x_arriving_green_light, self.arriving_green_light_small,
arriving_green_light_car)
    arriving_green_light_level_medium =
fuzz.interp_membership(self.x_arriving_green_light,
self.arriving_green_light_medium, arriving_green_light_car)
    arriving_green_light_level_many =
fuzz.interp_membership(self.x_arriving_green_light, self.arriving_green_light_many,
arriving_green_light_car)

    # Rule 1: If Arrival is few then Extension is zero.
    # Rule 2: If Arrival is small AND Queue is (few OR small) then Extension is
short.
    # Rule 3: If Arrival is small AND Queue is (medium OR many) then Extension is
zero.
    # Rule 4: If Arrival is medium AND Queue is (few OR small) then Extension is
medium.
    # Rule 5: If Arrival is medium AND Queue is (medium OR many) then
Extension is short.
    # Rule 6: If Arrival is many AND Queue is few then Extension is long.
    # Rule 7: If Arrival is many AND Queue is (small OR medium) then Extension is
medium.
    # Rule 8: If Arrival is few AND Queue is many then Extension is short.

    rule1 = arriving_green_light_level_few
    rule2 = np.fmin(arriving_green_light_level_small,
                    np.fmax(behind_red_light_level_few, behind_red_light_level_small))
    rule3 = np.fmin(arriving_green_light_level_small,
                    np.fmax(behind_red_light_level_medium,
behind_red_light_level_many))
    rule4 = np.fmin(arriving_green_light_level_medium,
                    np.fmax(behind_red_light_level_few, behind_red_light_level_small))
    rule5 = np.fmin(arriving_green_light_level_medium,
                    np.fmax(behind_red_light_level_medium,
behind_red_light_level_many))
    rule6 = np.fmin(arriving_green_light_level_many, behind_red_light_level_few)
    rule7 = np.fmin(arriving_green_light_level_many,
                    np.fmax(behind_red_light_level_small,
behind_red_light_level_medium))
    rule8 = np.fmin(arriving_green_light_level_many,
behind_red_light_level_many)

    if extension_count == 0:
       extension_activation_zero = np.fmin(np.fmax(rule1, rule3),
```

```
self.extension_zero)
        extension_activation_short = np.fmin(np.fmax(rule2, np.fmax(rule5, rule8)),
self.extension_short)
        extension_activation_medium = np.fmin(np.fmax(rule4, rule7),
self.extension_medium)
        extension_activation_long = np.fmin(rule6, self.extension_long)

    else:
        extension_activation_zero = np.fmin(
            np.fmax(rule1, np.fmax(rule2, np.fmax(rule3, np.fmax(rule5, rule8)))),
self.extension_zero)
        extension_activation_short = np.fmin(np.fmax(rule4, rule7),
self.extension_short)
        extension_activation_medium = np.fmin(rule6, self.extension_medium)
        extension_activation_long = np.fmin(0, self.extension_long)

    aggregated = np.fmax(extension_activation_zero,
np.fmax(extension_activation_short,
                                    np.fmax(extension_activation_medium,
                                        extension_activation_long)))

    return fuzz.defuzz(self.x_extension, aggregated, 'centroid')
```

src/Entity/TrafficLight.py:

```
import time
from src.Common import TrafficStatus, Lane
from src.Config import Config
import pygame


class TrafficLight:

  def __init__(self, x, y, lane, images, surface, status=TrafficStatus.green):
    self.x = x
    self.y = y
    self.lane = lane
    self.images = images  # expected to be dictionary with TrafficStatus as keys
    self.surface = surface
    self.duration = {
      TrafficStatus.green: Config['traffic_light']['green_light_duration'],
      TrafficStatus.red: Config['traffic_light']['red_light_duration'],
```

```
            TrafficStatus.yellow: Config['traffic_light']['yellow_light_duration']
        }
        self.duration_extension = {
            TrafficStatus.green: 0,
            TrafficStatus.red: 0,
            TrafficStatus.yellow: 0
        }
        self.start_time = {
            TrafficStatus.green: time.time(),
            TrafficStatus.red: time.time(),
            TrafficStatus.yellow: time.time()
        }
        self.status = status

    @property
    def center_x(self):
        return self.x + self.width / 2

    @property
    def center_y(self):
        return self.x + self.height / 2

    @property
    def width(self):
        return Config['traffic_light']['body_width']

    @property
    def height(self):
        return Config['traffic_light']['body_height']

    def draw(self):
        self.surface.blit(self.images[self.status], (self.x, self.y))

    def change_status(self, status: TrafficStatus):
        self.status = status
        self.start_time[self.status] = time.time()

    def auto_update(self, opposite_status: TrafficStatus):
        over_time = (self.duration[self.status] + self.duration_extension[self.status]) - \
                    (time.time() - self.start_time[self.status])

        to_change_status = over_time < 0
```

```
        new_status = None

        if to_change_status:
            if self.status == TrafficStatus.green:
                self.status = TrafficStatus.yellow
                new_status = TrafficStatus.yellow
            elif self.status == TrafficStatus.yellow:
                self.status = TrafficStatus.red
                new_status = TrafficStatus.red
            elif self.status == TrafficStatus.red:
                # if opposite is red, do not update
                if opposite_status == TrafficStatus.green:
                    return
                if abs(over_time) < Config['simulator']['gap_between_traffic_switch']:
                    return
                self.status = TrafficStatus.green
                new_status = TrafficStatus.green
            self.start_time[self.status] = time.time()

        return new_status

    def draw_countdown(self):
        font = pygame.font.SysFont('Comic Sans MS', 12, True)
        countdown = (self.duration[self.status] + self.duration_extension[self.status]) -
(time.time() - self.start_time[self.status])
        if countdown < 0:
            countdown = 0.0
        text_color = Config['colors']['black']
        if self.status == TrafficStatus.green:
            text_color = Config['colors']['traffic_green']
        elif self.status == TrafficStatus.yellow:
            text_color = Config['colors']['traffic_yellow']
        elif self.status == TrafficStatus.red:
            text_color = Config['colors']['traffic_red']
        text_surface = font.render('{}'.format(round(countdown, 1)), True, text_color)
        pos_x = self.x
        pos_y = self.y
        if self.lane == Lane.left_to_right:
            pos_y = self.y - self.height
        elif self.lane == Lane.right_to_left:
            pos_y = self.y + self.height*5/4
        elif self.lane == Lane.top_to_bottom:
            pos_x = self.x + self.width*2
```

```
        elif self.lane == Lane.bottom_to_top:
            pos_x = self.x - self.width*2
        self.surface.blit(text_surface, (pos_x, pos_y))

    def set_green_light_extension(self, extension):
        self.duration_extension[TrafficStatus.green] = extension

    def get_green_light_remaining_time(self):
        return (self.duration[self.status] + self.duration_extension[self.status]) -
(time.time() - self.start_time[self.status])
```

src/Entity/vehicle.py:

```
import pygame

from src.Common import Lane, TrafficStatus
from src.Config import Config


class Vehicle:
    def __init__(self, x, y, lane:Lane, image, surface, traffic_light):
        if lane != traffic_light.lane:
            raise Exception('The lane of traffic light and vehicle must be same.')
        self.x = x
        self.y = y
        self.lane = lane
        if lane in [Lane.left_to_right, Lane.right_to_left]:
            self.image = pygame.transform.scale(image, (Config['vehicle']['body_length'],
Config['vehicle']['body_width']))
        elif lane in [Lane.top_to_bottom, Lane.bottom_to_top]:
            self.image = pygame.transform.scale(image, (Config['vehicle']['body_width'],
Config['vehicle']['body_length']))
        self.surface = surface
        self.traffic_light = traffic_light

    @property
    def center_x(self):
        return self.x + self.width / 2

    @property
    def center_y(self):
        return self.x + self.height / 2
```

```
    @property
    def width(self):
        return self.image.get_width()

    @property
    def height(self):
        return self.image.get_height()

    def draw(self):
        self.surface.blit(self.image, (self.x, self.y))

    def move(self, front_vehicle=None):
        safe_distance = Config['vehicle']['safe_distance']
        speed = Config['vehicle']['speed']
        stopping_non_green_light = self.traffic_light.status != TrafficStatus.green and
self.is_behind_traffic_light()

        if self.lane == Lane.left_to_right:
            self.x += speed
            self.y += 0
            if front_vehicle:
                self.x = min(self.x, front_vehicle.x - safe_distance - self.width)
            if stopping_non_green_light:
                self.x = min(self.x, self.traffic_light.x - self.traffic_light.width/2 -
self.width)

        elif self.lane == Lane.right_to_left:
            self.x -= speed
            self.y += 0
            if front_vehicle:
                self.x = max(self.x, front_vehicle.x + front_vehicle.width + safe_distance)
            if stopping_non_green_light:
                self.x = max(self.x, self.traffic_light.x + self.traffic_light.width*3/2)

        elif self.lane == Lane.bottom_to_top:
            self.x += 0
            self.y -= speed
            if front_vehicle:
                self.y = max(self.y, front_vehicle.y + front_vehicle.height + safe_distance)
            if stopping_non_green_light:
                self.y = max(self.y, self.traffic_light.y + self.traffic_light.height)
```

```
        elif self.lane == Lane.top_to_bottom:
            self.x += 0
            self.y += speed
            if front_vehicle:
                self.y = min(self.y, front_vehicle.y - safe_distance - self.height)
            if stopping_non_green_light:
                self.y = min(self.y, self.traffic_light.y - self.traffic_light.height/2 -
self.height)

    def is_behind_traffic_light(self):
        if self.lane == Lane.left_to_right:
            return self.x + self.width <= self.traffic_light.x + self.traffic_light.width
        elif self.lane == Lane.right_to_left:
            return self.traffic_light.x + self.traffic_light.width <= self.x
        elif self.lane == Lane.bottom_to_top:
            return self.traffic_light.y + self.traffic_light.height <= self.y
        elif self.lane == Lane.top_to_bottom:
            return self.y + self.height <= self.traffic_light.y
        return False

    def inside_canvas(self) -> bool:
        return self.x >= 0 and \
                self.x + self.width <= Config['simulator']['screen_width'] and \
                self.y >= 0 and \
                self.y + self.height <= Config['simulator']['screen_height']
```

SCREENSHOTS AND RESULTS

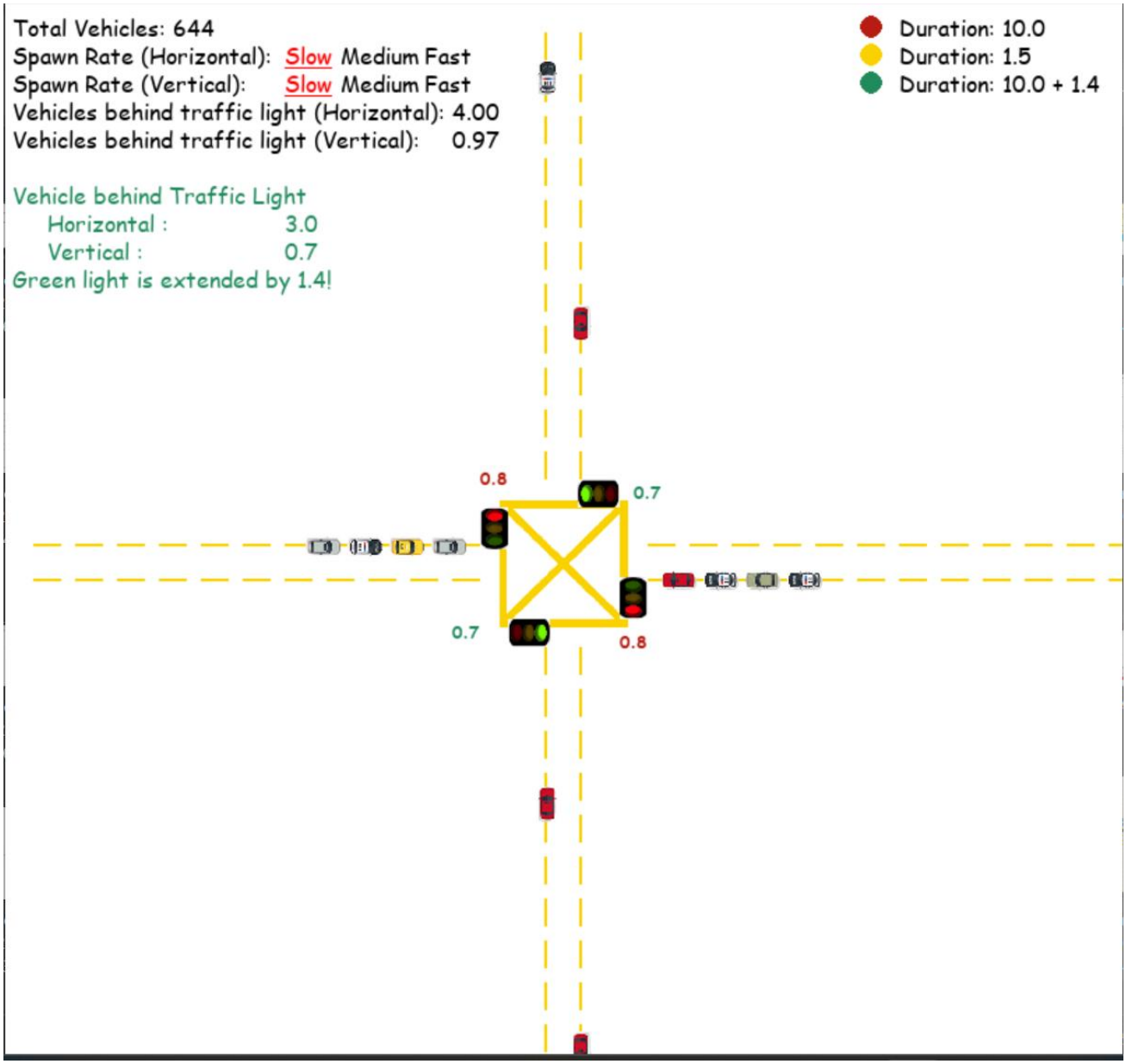

fig-3 : output and result

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion:

The development of the adaptive traffic light controller using fuzzy logic represents a significant step towards addressing the challenges of traffic congestion at intersections in urban areas. Through the implementation of a dynamic control system capable of adjusting signal timings based on real-time traffic conditions, the project has demonstrated the potential to improve traffic flow, reduce delays, and minimize congestion levels.

By leveraging fuzzy logic modeling and the Sugeno method, the controller effectively adapts to changing traffic patterns, optimizing green light durations to accommodate varying traffic densities and flow rates. Simulation and testing have validated the effectiveness of the controller in improving intersection efficiency and overall traffic management.

Future Enhancements:

While the adaptive traffic light controller has shown promising results, there are several opportunities for future enhancements and refinements:

1. Integration of Additional Data Sources: Incorporating data from sources such as weather conditions, road infrastructure status, and public transportation schedules can provide more comprehensive insights into traffic dynamics and further improve the accuracy of the controller's decision-making process.
2. Machine Learning Integration: Exploring the integration of machine learning algorithms to enhance the predictive capabilities of the controller and enable it to learn and adapt to evolving traffic patterns over time.
3. Real-Time Communication and Connectivity: Enhancing the communication capabilities of the controller to enable real-time data exchange and coordination with other traffic management systems, such as adaptive signal control networks and intelligent transportation systems.
4. Optimization Algorithms: Implementing advanced optimization algorithms to fine-tune signal timings and prioritize traffic movements based on predefined objectives, such as minimizing travel times or maximizing intersection throughput.
5. Scalability and Deployment: Developing scalable solutions that can be easily deployed across a wide range of intersections and integrated into existing traffic infrastructure with minimal disruption.

6. User Feedback and Engagement: Incorporating mechanisms for gathering feedback from users and stakeholders to identify areas for improvement and ensure the controller's alignment with community needs and preferences.
7. Overall, continued research and innovation in adaptive traffic control systems hold the potential to significantly improve urban mobility, reduce congestion, and enhance the overall quality of life in urban environments. By embracing emerging technologies and adopting a collaborative approach, we can work towards building smarter, more efficient transportation networks for the future.

REFERENCES

1. Zhang, Jianming, and Huimin Zhao. "Traffic signal control system based on fuzzy logic." IEEE International Conference on Automation and Logistics. IEEE, 2008.
    - This paper presents a traffic signal control system utilizing fuzzy logic to adaptively adjust signal timings based on real-time traffic conditions.
2. Feng, Jian, and Xiaohong Jiang. "Fuzzy logic based intelligent traffic signal controller." 2010 International Conference on Intelligent Control and Information Processing. IEEE, 2010.
    - The authors propose an intelligent traffic signal controller using fuzzy logic to optimize traffic flow and reduce congestion at intersections.
3. Liu, Ming, et al. "Traffic signal control system based on adaptive fuzzy logic." 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER). IEEE, 2015.
    - This paper presents a traffic signal control system that dynamically adjusts signal timings using adaptive fuzzy logic to improve intersection efficiency.
4. Ramezani, Mohammad, and Mohammad Javad Barmshoori. "Fuzzy logic-based urban traffic signal control system." 2016 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI). IEEE, 2016.
    - The authors propose a fuzzy logic-based urban traffic signal control system aimed at reducing traffic congestion and improving traffic flow in urban areas.