See I making a Traffic Schedule Manager with

**Project Goal**

• The goal of this project is to simulate and optimize traffic light management using OS scheduling algorithms, ensuring efficient, fair, and real-time traffic control. The system will be built with a Python backend and a React frontend, allowing users to visualize and interact with the traffic simulation than Simulating Traffic Management by implementing a four-way intersection where traffic flow is controlled using OS scheduling algorithms.

**Intial Milestones**

• Develop a lightweight Python API using Flask or FastAPI to handle vehicle input, process traffic rules, and update signals • Create an interactive UI where users can add vehicles, see real-time traffic light updates, and visualize vehicle movement. • Ensure traffic lights change based on vehicle density and scheduling logic, improving efficiency over fixed timer system. • Implement a priority system that allows ambulances, fire trucks, and police cars to pass first.

With libraries including

1 Backend (Python + Flask/FastAPI)

• Handles traffic logic & scheduling algorithms (FCFS, Round Robin, Priority Scheduling).

• Receives vehicle input from different lanes via API.

• Processes real-time signal changes and sends updates to the frontend.

1.1 Libraries Used:

• Flask – For API development • Threading – To simulate real-time updates

• collections (deque) – For queue-based vehicle management

• Time, Random – For scheduling and test data

2 Frontend (React +CSS)

• Interactive UI for a four-way intersection where users can add vehicles.

• Displays real-time traffic lights based on backend responses.

• Animates vehicle movement when lights turn green

2.1 Libraries Used:

• React.js – For UI development

• Axios – For API communication

• CSS – For styling

• Framer Motion (optional) – For smooth animations

3 Running Locally

3.1 Backend

• Runs on localhost:5000 using Flask

• Vehicles are added via API calls and processed using scheduling algorithms

3.2 Frontend

• Runs on localhost:3000 using React.and CSS

• Fetches real-time traffic updates from the backend and displays animations.

I have made little front end in react with

App.js

import React, { useState, useEffect, useRef } from 'react';

import './App.css';

import { motion } from 'framer-motion';

function App() {

const [vehicles, setVehicles] = useState([]);

const [inputValues, setInputValues] = useState({

north: '',

south: '',

east: '',

west: ''

});

const [isSimulating, setIsSimulating] = useState(false);

const [lightStates, setLightStates] = useState({

north: 'red',

south: 'red',

east: 'green',

west: 'green'

});

const vehiclePositions = useRef({});

// Traffic light cycle (10 second total, 8s green + 2s yellow)

useEffect(() => {

if (!isSimulating) return;

const cycleDuration = 10000; // 10 seconds total

const greenDuration = 8000; // 8 seconds green

const yellowDuration = 2000; // 2 seconds yellow

let cycleStartTime = Date.now();

let isNorthSouthGreen = true;

const updateLights = () => {

const elapsed = Date.now() - cycleStartTime;

if (elapsed >= cycleDuration) {

// Switch directions

isNorthSouthGreen = !isNorthSouthGreen;

cycleStartTime = Date.now();

}

const inYellowPhase = elapsed >= greenDuration && elapsed < greenDuration + yellowDuration;

setLightStates({

north: isNorthSouthGreen ? (inYellowPhase ? 'yellow' : 'green') : 'red',

south: isNorthSouthGreen ? (inYellowPhase ? 'yellow' : 'green') : 'red',

east: !isNorthSouthGreen ? (inYellowPhase ? 'yellow' : 'green') : 'red',

west: !isNorthSouthGreen ? (inYellowPhase ? 'yellow' : 'green') : 'red'

});

};

const interval = setInterval(updateLights, 100);

updateLights(); // Initial update

return () => clearInterval(interval);

}, [isSimulating]);

const handleInputChange = (direction, value) => {

setInputValues(prev => ({

...prev,

[direction]: value

}));

};

const startSimulation = () => {

setIsSimulating(true);

setVehicles([]);

const newVehicles = [];

const directions = {

north: 'down',

south: 'up',

east: 'left',

west: 'right'

};

Object.entries(inputValues).forEach(([direction, value]) => {

const vehicleString = value.replace(/\s/g, '');

const vehicleArray = vehicleString.split(',').filter(Boolean);

vehicleArray.forEach((char, index) => {

let type;

switch(char.toLowerCase()) {

case 'c': type = 'car'; break;

case 'b': type = 'bike'; break;

case 't': type = 'bus'; break;

case 'a': type = 'ambulance'; break;

default: return;

}

const id = `${direction}-${index}-${Date.now()}`;

newVehicles.push({

id,

type,

direction: directions[direction],

originDirection: direction,

delay: index \* 0.5,

hasStarted: false

});

vehiclePositions.current[id] = getInitialPosition(directions[direction]);

});

});

setVehicles(newVehicles);

};

const getEmoji = (type) => {

switch (type) {

case 'car': return '🚗';

case 'bike': return '🚲';

case 'ambulance': return '🚑';

case 'bus': return '🚌';

default: return '❓';

}

};

const getInitialPosition = (direction) => {

// Position vehicles further back from intersection

switch (direction) {

case 'left': return { x: '120vw', y: '50vh' };

case 'right': return { x: '-20vw', y: '50vh' };

case 'up': return { x: '50vw', y: '120vh' };

case 'down': return { x: '50vw', y: '-20vh' };

default: return { x: '50vw', y: '50vh' };

}

};

const getStopPosition = (direction) => {

// Position where vehicles stop before intersection

switch (direction) {

case 'left': return { x: '70vw', y: '50vh' };

case 'right': return { x: '30vw', y: '50vh' };

case 'up': return { x: '50vw', y: '70vh' };

case 'down': return { x: '50vw', y: '30vh' };

default: return { x: '50vw', y: '50vh' };

}

};

const getTargetPosition = (direction) => {

switch (direction) {

case 'left': return { x: '-20vw', y: '50vh' };

case 'right': return { x: '120vw', y: '50vh' };

case 'up': return { x: '50vw', y: '-20vh' };

case 'down': return { x: '50vw', y: '120vh' };

default: return { x: '50vw', y: '50vh' };

}

};

const shouldMove = (vehicle) => {

const currentLight = lightStates[vehicle.originDirection];

const isAmbulance = vehicle.type === 'ambulance';

if (isAmbulance) return true;

if (currentLight === 'green') return true;

if (currentLight === 'yellow' && vehicle.hasStarted) return true;

return false;

};

return (

<div className="app">

<div className="intersection">

{/\* Roads \*/}

<div className="road horizontal"></div>

<div className="road vertical"></div>

{/\* Lane markings \*/}

<div className="lane-markings horizontal"></div>

<div className="lane-markings vertical"></div>

{/\* Zebra crossings \*/}

<div className="zebra zebra-top"></div>

<div className="zebra zebra-bottom"></div>

<div className="zebra zebra-left"></div>

<div className="zebra zebra-right"></div>

{/\* Traffic lights - one for each direction \*/}

<div className="traffic-light north">

<div className={`light ${lightStates.north === 'red' ? 'red active' : 'off'}`}></div>

<div className={`light ${lightStates.north === 'yellow' ? 'yellow active' : 'off'}`}></div>

<div className={`light ${lightStates.north === 'green' ? 'green active' : 'off'}`}></div>

</div>

<div className="traffic-light south">

<div className={`light ${lightStates.south === 'red' ? 'red active' : 'off'}`}></div>

<div className={`light ${lightStates.south === 'yellow' ? 'yellow active' : 'off'}`}></div>

<div className={`light ${lightStates.south === 'green' ? 'green active' : 'off'}`}></div>

</div>

<div className="traffic-light east">

<div className={`light ${lightStates.east === 'red' ? 'red active' : 'off'}`}></div>

<div className={`light ${lightStates.east === 'yellow' ? 'yellow active' : 'off'}`}></div>

<div className={`light ${lightStates.east === 'green' ? 'green active' : 'off'}`}></div>

</div>

<div className="traffic-light west">

<div className={`light ${lightStates.west === 'red' ? 'red active' : 'off'}`}></div>

<div className={`light ${lightStates.west === 'yellow' ? 'yellow active' : 'off'}`}></div>

<div className={`light ${lightStates.west === 'green' ? 'green active' : 'off'}`}></div>

</div>

{/\* Direction inputs \*/}

<div className="direction-input north">

<label>North (↓):</label>

<input

type="text"

value={inputValues.north}

onChange={(e) => handleInputChange('north', e.target.value)}

placeholder="c,b,t,a"

disabled={isSimulating}

/>

</div>

<div className="direction-input south">

<label>South (↑):</label>

<input

type="text"

value={inputValues.south}

onChange={(e) => handleInputChange('south', e.target.value)}

placeholder="c,b,t,a"

disabled={isSimulating}

/>

</div>

<div className="direction-input east">

<label>East (←):</label>

<input

type="text"

value={inputValues.east}

onChange={(e) => handleInputChange('east', e.target.value)}

placeholder="c,b,t,a"

disabled={isSimulating}

/>

</div>

<div className="direction-input west">

<label>West (→):</label>

<input

type="text"

value={inputValues.west}

onChange={(e) => handleInputChange('west', e.target.value)}

placeholder="c,b,t,a"

disabled={isSimulating}

/>

</div>

<button

className="start-button"

onClick={startSimulation}

disabled={isSimulating}

>

{isSimulating ? 'Simulating...' : 'Start Simulation'}

</button>

{/\* Vehicles \*/}

{vehicles.map((vehicle) => {

const move = shouldMove(vehicle);

const stopPosition = getStopPosition(vehicle.direction);

const targetPosition = getTargetPosition(vehicle.direction);

return (

<motion.div

key={vehicle.id}

className={`vehicle ${vehicle.type} ${!move ? 'stopped' : ''}`}

initial={getInitialPosition(vehicle.direction)}

animate={move ? targetPosition : stopPosition}

transition={{

duration: 5,

ease: 'linear',

delay: vehicle.delay

}}

onAnimationStart={() => {

vehiclePositions.current[vehicle.id] = {

...vehiclePositions.current[vehicle.id],

hasStarted: true

};

}}

style={{

position: 'absolute',

left: 0,

top: 0,

zIndex: vehicle.type === 'ambulance' ? 20 : 15

}}

>

{getEmoji(vehicle.type)}

{vehicle.type === 'ambulance' && <span className="siren">🚨</span>}

</motion.div>

);

})}

</div>

</div>

);

}

export default App;

App.css

.app {

height: 100vh;

width: 100vw;

background-color: #2e8b57;

overflow: hidden;

position: relative;

}

.intersection {

position: absolute;

height: 100%;

width: 100%;

}

/\* Roads - made wider to show more of the approach \*/

.road {

position: absolute;

background-color: #333;

z-index: 1;

}

.road.horizontal {

top: 50%;

left: -20%;

width: 140%;

height: 140px;

transform: translateY(-50%);

}

.road.vertical {

left: 50%;

top: -20%;

height: 140%;

width: 140px;

transform: translateX(-50%);

}

/\* Lane markings \*/

.lane-markings {

position: absolute;

background-repeat: repeat;

background-size: 20px 20px;

z-index: 2;

}

.lane-markings.horizontal {

top: 50%;

left: -20%;

width: 140%;

height: 4px;

background-image: repeating-linear-gradient(to right, white 0, white 10px, transparent 10px, transparent 20px);

transform: translateY(-2px);

}

.lane-markings.vertical {

left: 50%;

top: -20%;

width: 4px;

height: 140%;

background-image: repeating-linear-gradient(to bottom, white 0, white 10px, transparent 10px, transparent 20px);

transform: translateX(-2px);

}

/\* Zebra crossings \*/

.zebra {

position: absolute;

background: white;

z-index: 3;

}

.zebra-top {

top: calc(50% - 70px);

left: calc(50% - 60px);

width: 120px;

height: 10px;

}

.zebra-bottom {

top: calc(50% + 60px);

left: calc(50% - 60px);

width: 120px;

height: 10px;

}

.zebra-left {

left: calc(50% - 70px);

top: calc(50% - 60px);

width: 10px;

height: 120px;

}

.zebra-right {

left: calc(50% + 60px);

top: calc(50% - 60px);

width: 10px;

height: 120px;

}

/\* Traffic lights - positioned at each approach \*/

.traffic-light {

position: absolute;

width: 30px;

height: 90px;

background-color: #111;

display: flex;

flex-direction: column;

justify-content: space-between;

align-items: center;

padding: 8px 0;

border-radius: 8px;

z-index: 4;

box-shadow: 0 0 10px rgba(0,0,0,0.5);

}

/\* Position each traffic light at the approach \*/

.traffic-light.north {

top: calc(30% - 45px);

left: 50%;

transform: translateX(-50%);

}

.traffic-light.south {

bottom: calc(30% - 45px);

left: 50%;

transform: translateX(-50%);

}

.traffic-light.east {

top: 50%;

right: calc(30% - 15px);

transform: translateY(-50%);

}

.traffic-light.west {

top: 50%;

left: calc(30% - 15px);

transform: translateY(-50%);

}

.traffic-light .light {

width: 20px;

height: 20px;

border-radius: 50%;

background-color: #444;

}

.traffic-light .red.active {

background-color: red;

box-shadow: 0 0 15px rgba(255,0,0,0.7);

}

.traffic-light .yellow.active {

background-color: yellow;

box-shadow: 0 0 15px rgba(255,255,0,0.7);

}

.traffic-light .green.active {

background-color: #00ff00;

box-shadow: 0 0 15px rgba(0,255,0,0.7);

}

.traffic-light .off {

opacity: 0.2;

}

/\* Direction inputs \*/

.direction-input {

position: absolute;

background: rgba(255, 255, 255, 0.9);

padding: 10px;

border-radius: 6px;

z-index: 10;

display: flex;

flex-direction: column;

min-width: 140px;

box-shadow: 0 2px 8px rgba(0,0,0,0.2);

}

.direction-input label {

font-size: 14px;

margin-bottom: 6px;

color: #222;

font-weight: bold;

}

.direction-input input {

padding: 8px;

border: 1px solid #ddd;

border-radius: 4px;

font-size: 14px;

}

.direction-input.north {

top: 10px;

left: 50%;

transform: translateX(-50%);

}

.direction-input.south {

bottom: 10px;

left: 50%;

transform: translateX(-50%);

}

.direction-input.east {

top: 50%;

right: 10px;

transform: translateY(-50%);

}

.direction-input.west {

top: 50%;

left: 10px;

transform: translateY(-50%);

}

.start-button {

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

padding: 12px 24px;

background: #4CAF50;

color: white;

border: none;

border-radius: 6px;

cursor: pointer;

z-index: 20;

font-weight: bold;

font-size: 16px;

box-shadow: 0 4px 8px rgba(0,0,0,0.2);

transition: all 0.3s;

}

.start-button:hover:not(:disabled) {

background: #45a049;

transform: translate(-50%, -50%) scale(1.05);

}

.start-button:disabled {

background: #aaa;

cursor: not-allowed;

}

.vehicle {

position: absolute;

font-size: 28px;

transform: translate(-50%, -50%);

will-change: transform;

}

.vehicle.stopped {

opacity: 0.7;

filter: brightness(0.8);

}

.siren {

position: absolute;

font-size: 14px;

top: -15px;

left: 50%;

transform: translateX(-50%);

animation: blink 0.5s infinite alternate;

}

@keyframes blink {

from { opacity: 0.3; }

to { opacity: 1; }

}

.direction-input input:disabled {

background: #eee;

cursor: not-allowed;

}

And phyton

from flask import Flask, jsonify, request

from flask\_cors import CORS

from collections import defaultdict, deque

import time

import threading

app = Flask(\_\_name\_\_)

CORS(app)

# Traffic simulation state

simulation\_state = {

'lights': {

'north': 'red',

'south': 'red',

'east': 'green',

'west': 'green'

},

'vehicles': {

'queued': defaultdict(deque),

'active': {}

},

'next\_id': 1

}

def traffic\_light\_controller():

while True:

# North-South green, East-West red

simulation\_state['lights'] = {

'north': 'green',

'south': 'green',

'east': 'red',

'west': 'red'

}

time.sleep(8)

# North-South yellow

simulation\_state['lights'] = {

'north': 'yellow',

'south': 'yellow',

'east': 'red',

'west': 'red'

}

time.sleep(2)

# East-West green, North-South red

simulation\_state['lights'] = {

'north': 'red',

'south': 'red',

'east': 'green',

'west': 'green'

}

time.sleep(8)

# East-West yellow

simulation\_state['lights'] = {

'north': 'red',

'south': 'red',

'east': 'yellow',

'west': 'yellow'

}

time.sleep(2)

def vehicle\_controller():

while True:

# Move vehicles from queues to active based on traffic lights

for direction in ['north', 'south', 'east', 'west']:

light\_state = simulation\_state['lights'][direction]

queue = simulation\_state['vehicles']['queued'][direction]

if queue and (light\_state == 'green' or

(light\_state == 'yellow' and queue[0]['type'] == 'ambulance')):

vehicle = queue.popleft()

vehicle['start\_time'] = time.time()

vehicle['progress'] = 0

simulation\_state['vehicles']['active'][vehicle['id']] = vehicle

# Update active vehicles

current\_time = time.time()

for vid, vehicle in list(simulation\_state['vehicles']['active'].items()):

elapsed = current\_time - vehicle['start\_time']

progress = min(elapsed / 5, 1) # 5 seconds to cross intersection

if progress >= 1:

del simulation\_state['vehicles']['active'][vid]

else:

vehicle['progress'] = progress

time.sleep(0.1)

@app.route('/add\_vehicle', methods=['POST'])

def add\_vehicle():

data = request.json

vehicle = {

'id': simulation\_state['next\_id'],

'type': data['type'],

'direction': data['direction'],

'origin': data['direction']

}

simulation\_state['next\_id'] += 1

simulation\_state['vehicles']['queued'][data['direction']].append(vehicle)

return jsonify({'success': True})

@app.route('/get\_state', methods=['GET'])

def get\_state():

return jsonify({

'lights': simulation\_state['lights'],

'vehicles': {

'queued': {k: list(v) for k, v in simulation\_state['vehicles']['queued'].items()},

'active': simulation\_state['vehicles']['active']

}

})

if \_\_name\_\_ == '\_\_main\_\_':

threading.Thread(target=traffic\_light\_controller, daemon=True).start()

threading.Thread(target=vehicle\_controller, daemon=True).start()

app.run(port=5000, debug=True) App.js import React, { useState, useEffect, useRef } from 'react';

Now I want some things to be done

1.i want the movement of vechile in phyton

2.i want to vechile react traffic light and stop wait or go according to it and even after spwaning if the light become vechile stop

3 its better to have graph or some coordinate to stop or move vechle one point to another and to stop at traffic light at certain porint

4.i want vechile to go from one direction to all three direction not just up down left right

5. I want proper arrangement of traffic light at the point of intersection and all lane marking visble and proper road and improved little design by before things intact