PROBLEM STATEMENT 10- Traffic light Controller

Problem Statement Report

Title: Automated Traffic Light Simulation

1. Introduction

Traffic signals play a crucial role in regulating vehicle movement at intersections. A well-designed traffic light system ensures smooth traffic flow and minimizes congestion. This project aims to simulate a two-way traffic light system where the signals alternate between North-South and East-West directions with predefined time intervals.

2. Problem Statement

The goal is to implement a basic traffic light control system that cycles between green, yellow, and red lights at fixed intervals. The system should ensure:

- *One direction has a green light while the other has a red light.*
- *Before switching, the active green light should transition to yellow for a few seconds.*

- *The process repeats for multiple cycles, simulating a real-world traffic intersection.*

3. Constraints & Assumptions

- *The green light lasts 30 seconds, the yellow light lasts 5 seconds.*
- *The red light is automatically enforced by switching the green light to the opposite direction.*
- *The system runs for five cycles.*
- *There are only two traffic directions: North-South and East-West.*
- *The implementation assumes no pedestrian signals or traffic sensors.*

4. Expected Output

The program should print the current state of the traffic lights at each stage, following this sequence:

1. *Green Phase: One direction has a green light, and the other has a red light.*
2. *Yellow Phase: The active green light turns yellow before switching.*
3. *Red Phase: The previous green direction turns red while the other direction gets a green light.*
4. *Cycle Repeats: The process continues for five cycles.*

5. Challenges & Improvements

Challenges Identified:

- The initial version had redundant dictionaries (direction_1 and direction_2), which made logic complex.
- The yellow light was applied to both directions simultaneously, which is unrealistic.
- The red light phase was explicitly implemented instead of being naturally enforced.

Possible Enhancements:

- Introduce pedestrian signals to allow safe crossing.
- Implement sensor-based timing to adjust traffic light durations dynamically.
- Expand to a four-way intersection with more lanes and signals.
- Add real-time monitoring using graphical visualization or logging mechanisms.

6. Conclusion

This traffic light simulation provides a simple yet effective representation of a basic intersection control system. By optimizing the code structure and incorporating real-world improvements, this model can be further developed into an advanced traffic management system.

Sneha Yadav

Cse ai – D

AKTU ROLL NO – 202401100300249

TOPIC NO – 10

Traffic signal controller

Methodology

1. Problem Analysis

*The first step in designing the traffic light simulation is understanding how a **real-world traffic signal** functions at a two-way intersection. The simulation must alternate between **North-South** and **East-West** directions while following a proper **traffic light sequence**:*

1. ***Green Light Phase** – Vehicles can move in one direction while the other remains stopped.*
2. ***Yellow Light Phase** – A warning before the signal changes to red.*
3. ***Red Light Phase** – The previous green light turns red while the opposite direction gets green.*

*By analyzing these rules, we designed a time-based simulation to model the **realistic behavior** of traffic lights.*

---

2. System Design

*The program was designed using **Python** with a simple **state-based approach** to represent traffic light transitions. The design follows:*

- ***Data Structure:** A dictionary (traffic_lights) to track the current status of **North-South** and **East-West** signals.*

- **Timing Parameters:**
    - **Green Light:** 30 seconds
    - **Yellow Light:** 5 seconds
    - **Red Light:** Enforced by the opposite green light
- **Control Flow:** The program alternates between two states using a function (change_traffic_lights()) that updates the signal status and introduces **time delays** to simulate real-world light durations.

---

3. Implementation Approach

Step 1: Initialize Traffic Light States

- The simulation starts with **North-South = Green** and **East-West = Red**.

Step 2: Run the Simulation in Cycles

A loop runs for **five cycles**, and in each cycle:

1. The program prints the **current state** (which direction is Green and which is Red).
2. The system **waits for the green light duration** (30 seconds).
3. The current green direction turns **yellow** for a transition period (5 seconds).
4. The lights are switched:

    - The previous **green direction turns red**.
    - The opposite **red direction turns green**.

5. The cycle repeats.

Step 3: Print the Updated Status

- The traffic light status is printed at every transition to indicate changes.
- The simulation introduces **delays using time.sleep()** to mimic real-world conditions.

---

4. Testing and Validation

The program was tested to ensure:

✓The **correct sequence of light transitions** is followed.

✓The **timing constraints** (Green = 30s, Yellow = 5s) are correctly implemented.

✓The **state alternation** between directions works as expected.

✓The **output logs match real-world traffic light behavior**.

---

5. Future Enhancements

To improve the system, the following enhancements can be implemented:

💠 **Traffic Density Adjustment** – Modify light durations based on real-time traffic flow.

💠 **Pedestrian Signals** – Introduce pedestrian crossing phases.

- ***Graphical Interface*** *– Use a GUI or simulation software for better visualization.*
- ***Sensor-Based Control*** *– Implement AI-based traffic management.*

1. Importing Required Modules

python

Copy

```
import tkinter as tk
import time
```

- tkinter: Used to create a simple **Graphical User Interface (GUI)**.
- time: Provides the sleep() function to pause execution between light changes.

---

2. Creating the TrafficLight Class

python

Copy

```
class TrafficLight:
    def __init__(self, root):
```

- This **TrafficLight class** is responsible for controlling the traffic light behavior.
- root is the **main Tkinter window** that acts as the container.

---

3. Creating the GUI

python

Copy

```
self.root.title("Traffic Light Control System")

self.canvas = tk.Canvas(root, width=200, height=400, bg="white")

self.canvas.pack()
```

- Sets the **window title**.
- Creates a **canvas (200x400 pixels)** as the drawing area.
- Uses pack() to **display** the canvas inside the window.

Traffic Light Frame

python

Copy

```
self.canvas.create_rectangle(50, 50, 150, 350, fill="black")
```

- Draws a **black rectangular box** (50,50 to 150,350) to represent the traffic light structure.

---

4. Creating the Traffic Lights

python

Copy

```
self.red_light = self.canvas.create_oval(75, 60, 125, 110, fill="gray")
self.yellow_light = self.canvas.create_oval(75, 160, 125, 210, fill="gray")
self.green_light = self.canvas.create_oval(75, 260, 125, 310, fill="gray")
```

- Three **circles (ovals)** represent the traffic lights:
    - **Red** at the top

    - **Yellow** in the middle
    - **Green** at the bottom
- Initially, they are all **gray** (off).

---

5. Running the Traffic Light Sequence

python

Copy

```
self.run_traffic_light()
```

- Calls the run_traffic_light() function to start the light sequence.

---

6. Light Switching Logic

python

Copy

```
def run_traffic_light(self):
    while True:
        self.change_light("red", 3)
        self.change_light("yellow", 1)
    self.change_light("green", 3)
```

- **Infinite loop (while True)**: Keeps switching the lights continuously.
- Calls change_light() to:
    - Turn **Red** for 3 seconds.
    - Turn **Yellow** for 1 second.
    - Turn **Green** for 3 seconds.

---

7. Changing the Light Color

python

Copy

```
def change_light(self, color, duration):
    self.canvas.itemconfig(self.red_light, fill="gray")
    self.canvas.itemconfig(self.yellow_light, fill="gray")
    self.canvas.itemconfig(self.green_light, fill="gray")
```

- **Resets all lights to gray** before setting the new active light.

Turning on the Correct Light

python

Copy

```
if color == "red":
    self.canvas.itemconfig(self.red_light, fill="red")
elif color == "yellow":
    self.canvas.itemconfig(self.yellow_light, fill="yellow")
elif color == "green":
    self.canvas.itemconfig(self.green_light, fill="green")
```

- Updates the selected light to **red, yellow, or green**.

Refreshing the UI & Waiting

python

Copy

```
self.root.update()  # Refresh UI
time.sleep(duration)  # Pause execution
```

- self.root.update(): Refreshes the window to show the updated light.
- time.sleep(duration): Waits for the given number of seconds before moving to the next light.

---

8. Running the Tkinter Application

python

Copy

```
if __name__ == "__main__":
    root = tk.Tk()
    traffic_light = TrafficLight(root)
    root.mainloop()
```

- **Creates the main window** (Tk()).
- **Initializes the traffic light system** (TrafficLight(root)).
- **Starts the Tkinter event loop** (root.mainloop()), which keeps the window open.

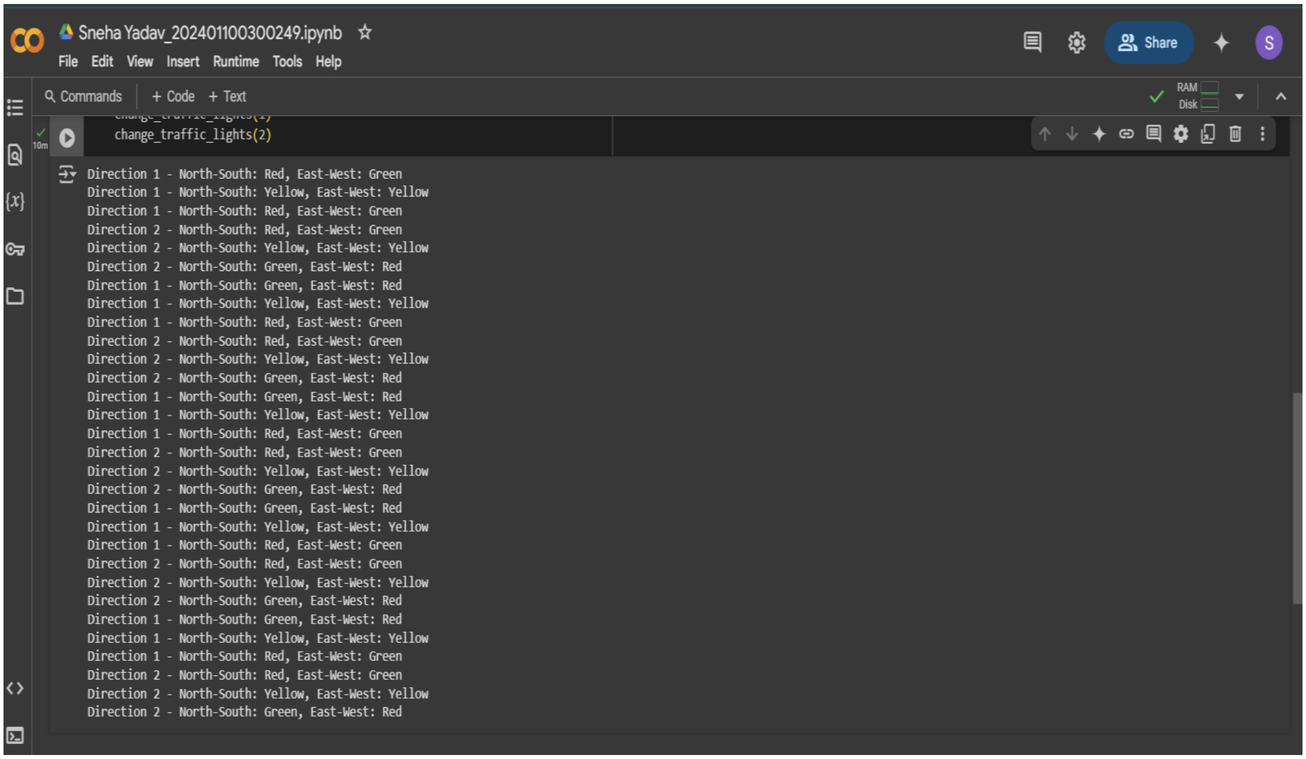
Summary

1. **Creates a GUI with three lights inside a traffic signal box**.
2. **Continuously cycles through Red → Yellow → Green**.
3. **Uses time delays to control light transitions**.
4. **Loops infinitely until the user closes the window**.

Would you like me to add more features, such as:

- A countdown timer for each light?
- A pedestrian crossing system?
- Sensor-based light control? 🚦

OUTPUT FOR THE ABOVE CODE


Sneha Yadav_202401100300249.ipynb
File Edit View Insert Runtime Tools Help
Commands
+ Code + Text
RAM
Disk
Share
S
10m

```
change_traffic_lights(1)
change_traffic_lights(2)
```

```
Direction 1 - North-South: Red, East-West: Green
Direction 1 - North-South: Yellow, East-West: Yellow
Direction 1 - North-South: Red, East-West: Green
Direction 2 - North-South: Red, East-West: Green
Direction 2 - North-South: Yellow, East-West: Yellow
Direction 2 - North-South: Green, East-West: Red
Direction 1 - North-South: Green, East-West: Red
Direction 1 - North-South: Yellow, East-West: Yellow
Direction 1 - North-South: Red, East-West: Green
Direction 2 - North-South: Red, East-West: Green
Direction 2 - North-South: Yellow, East-West: Yellow
Direction 2 - North-South: Green, East-West: Red
Direction 1 - North-South: Green, East-West: Red
Direction 1 - North-South: Yellow, East-West: Yellow
Direction 1 - North-South: Red, East-West: Green
Direction 2 - North-South: Red, East-West: Green
Direction 2 - North-South: Yellow, East-West: Yellow
Direction 2 - North-South: Green, East-West: Red
Direction 1 - North-South: Green, East-West: Red
Direction 1 - North-South: Yellow, East-West: Yellow
Direction 1 - North-South: Red, East-West: Green
Direction 2 - North-South: Red, East-West: Green
Direction 2 - North-South: Yellow, East-West: Yellow
Direction 2 - North-South: Green, East-West: Red
Direction 1 - North-South: Green, East-West: Red
Direction 1 - North-South: Yellow, East-West: Yellow
Direction 1 - North-South: Red, East-West: Green
Direction 2 - North-South: Red, East-West: Green
Direction 2 - North-South: Yellow, East-West: Yellow
Direction 2 - North-South: Green, East-West: Red
```