

TRAFFIC MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

***Sanyog
Dani***

In partial satisfaction of the requirements for the degree of

**BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING**

with specialization in IT

SCHOOL OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

MAY 2023

TABLE OF CONTENTS

S.No	CONTENTS	PAGE NO
1.	Problem Statement	4
2.	Modules of Project	5
3.	Diagrams	6
	3.1 Use case Diagram	7
	3.2 Class Diagram	8
	3.3 Sequence Diagram	9
	3.4 Collaboration Diagram	10
	3.5 State Chart Diagram	11
	3.6 Activity Diagram	12
	3.7 Package Diagram	13
	3.8 Component Diagram	14
	3.9 Deployment Diagram	15

4.	Code Screenshots	17
5.	Output Screenshots	18
6.	Conclusion and Results	19
7.	References	20

1. PROBLEM STATEMENT

Develop a series of systems model for traffic passing through a 4-way intersection, controlled by traffic light. We will assume that arrangement of traffic lights and road lanes is fixed and that the lights switch from red to green to amber in a regular repetitive pattern. Moreover, we assume that driver behaviour is constrained by the road rules and the desire to avoid vehicle collisions.

CONTENTS



PROBLEM
STATEMENT



MODULES
INVOLVED



UML DIAGRAM:
ACTIVITY & STATE
CHART DIAGRAM



PROCEDURE



CODE/OUTPUT
SCREENSHOTS



CONCLUSION/
RESULT

2. MODULES OF PROJECT

- **iostream:** This module provides basic input and output services for C++ programs.
- **queue:** This module provides a container class to hold a queue of elements.
- **unistd.h:** This module provides access to the POSIX operating system API, including the `sleep()` function used in the program.
- **LightColor enum:** This enumeration defines three values to represent the different colors of a traffic light: **GREEN**, **YELLOW**, and **RED**.
- **TrafficLight class:** This class defines a traffic light object that has a color member variable of type **LightColor**. It has a default constructor that initializes the color to **GREEN**, a `changeColor()` method that changes the color of the traffic light based on its current color (**GREEN** to **YELLOW**, **YELLOW** to **RED**, and **RED** to **GREEN**), and a `getColor()` method that returns the current color of the traffic light.
- **Vehicle class:** This class defines a vehicle object that has an id member variable of type **int**. It has a constructor that takes an id argument and sets the id member variable, and a `getId()` method that returns the id of the vehicle.
- **TrafficManagementSystem class:** This class defines a traffic management system object that has a `vehicleQueue` member variable of type `queue<Vehicle>` to hold the vehicles waiting to cross the intersection, and a `trafficLight` member variable of type **TrafficLight** to manage the traffic light. It has an `addVehicle()` method to add a vehicle to the queue, and a `run()` method that simulates the operation of the traffic management system. The `run()` method uses a while loop that executes for a fixed number of iterations (in this case, 10) or a specific time has passed. It checks the current color of the traffic light and dequeues a vehicle from the queue if the light is green. It then changes the color of the traffic light and sleeps for 5 seconds before starting the next iteration .

3. UML DIAGRAM

Unified Modelling Language (UML) diagrams are visual representations used for modelling object-oriented software systems. UML diagrams provide a standardized way to visualize and communicate software designs, making them easier to understand and implement.

There are two main types of UML diagrams: behavioral and structural. Structural UML diagrams describe the static structure of a system, including its components, classes, interfaces, and relationships between them. Behavioral UML diagrams describe the dynamic behavior of a system, including the interactions between objects and the changes in their states.

Structural UML Diagrams include:

1. Class Diagrams
2. Object Diagrams
3. Component Diagrams
4. Deployment Diagrams
5. Package Diagrams





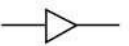

Behavioral UML Diagrams include:

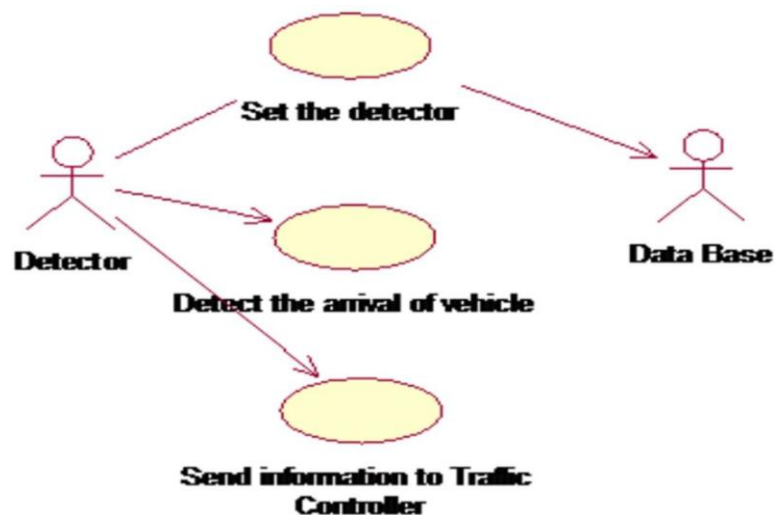
1. Use Case Diagrams
2. Sequence Diagrams
3. Collaboration Diagrams
4. State Diagrams
5. Activity Diagrams

3.1 USE CASE DIGRAM

A class diagram is a UML diagram that depicts the structure of a system by showing the classes, objects, interfaces, and their relationships. It provides a static view of the system and is used to visualize the objects and their interactions within the system. Classes are represented as boxes with their attributes and methods, and relationships between classes are shown using lines with various symbols.

Notations are :-

1.  - Actor symbol
2.  - Use Case symbol
3.  - Include relationship
4.  - Extend relationship
5.  - Generalization relationship
6.  - Association relationship







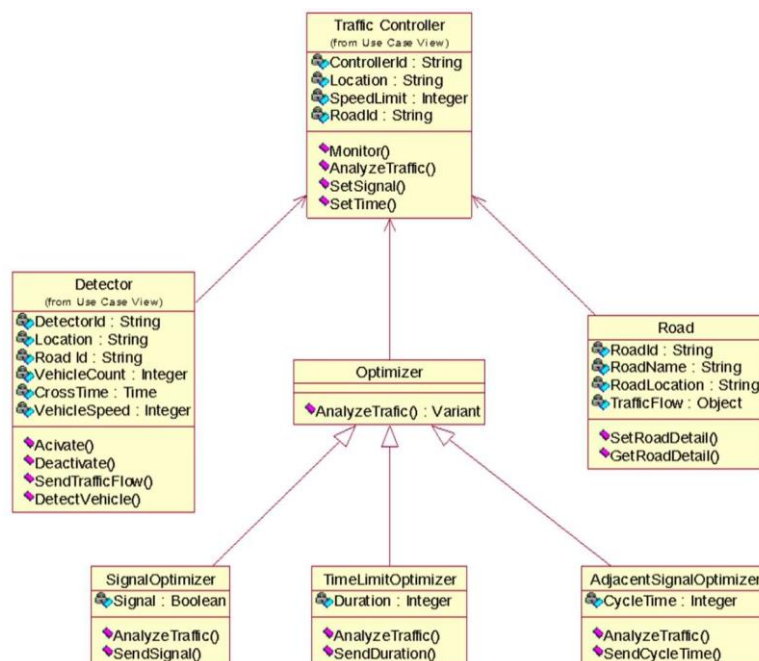
A use case diagram of an traffic management system illustrates the different types of actors (such as vehicle light colour , timer)and the various actions they can perform within the system.

3.2 CLASS DIAGRAM

A class diagram is a UML diagram that depicts the structure of a system by showing the classes, objects, interfaces, and their relationships. It provides a static view of the system and is used to visualize the objects and their interactions within the system. Classes are represented as boxes with their attributes and methods, and relationships between classes are shown using lines with various symbols.

The notations used are:

1.  - Class symbol
2.  - Association line
3.  - Composition line
4.  - Aggregation line






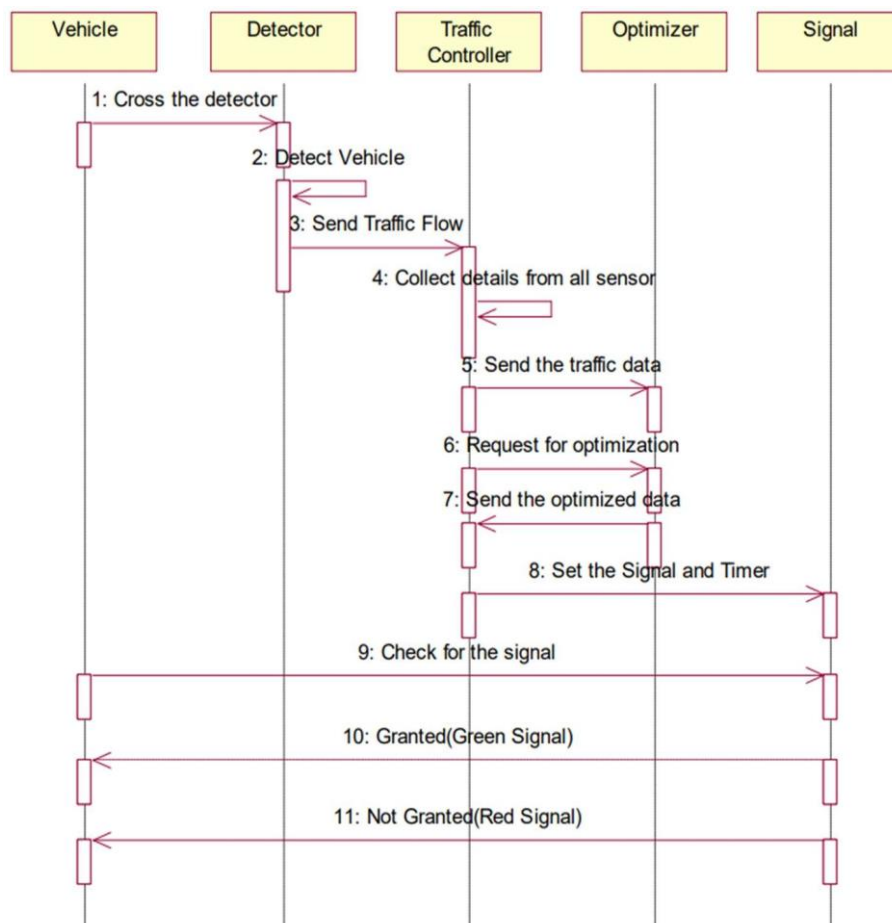
A class diagram of an traffic management system shows the different classes, their attributes, and the relationships between them, such as vehicle , detector and traffic light classes

3.3 SEQUENCE DIAGRAM

A sequence diagram is a type of UML diagram that illustrates the interactions between objects or components in a system over time. It depicts the order in which objects interact with each other to achieve a specific task or goal. It is a dynamic view of the system that shows the flow of messages exchanged between objects and the timing of their interactions.

The notations used in are:

1.  - Actor symbol
2.  - Package symbol
3.  - Lifeline symbol






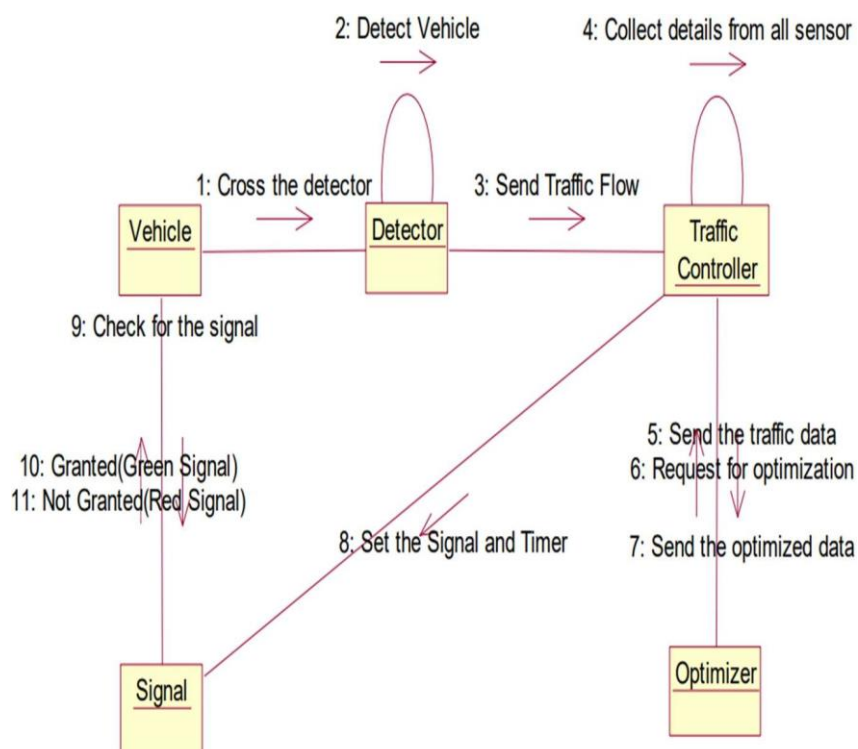
A sequence diagram of an traffic management system illustrates the interactions between different objects and actors involved such as detector , vehicles list and traffic light colour

3.4 COLLABORATION DIAGRAM

A collaboration diagram is a type of UML diagram that shows the interactions between objects or components in a system to achieve a specific task or goal. It is a visual representation of the collaboration and communication among the objects in a system.

The notations used in are:

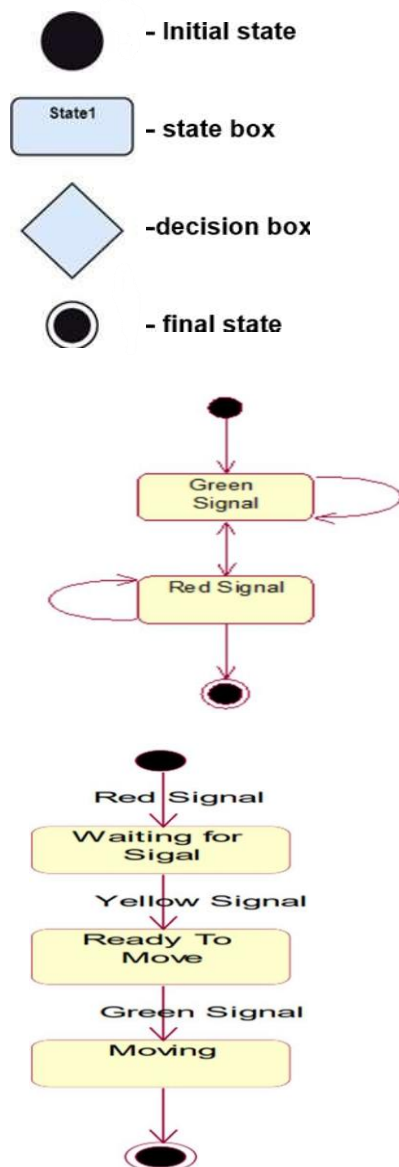
1.  - Object
2.  - Relation/Association
3.  - Messages



A collaboration diagram of an traffic management system depicts the objects and their associations, messages exchanged between them, and the sequence of events that occur during the process.

3.5 STATE CHART DIAGRAM

A state chart diagram is a type of UML diagram that shows the different states and transitions of an object or system over time. It is a visual representation of the behavior of an object or system, and how it responds to external stimuli or events.



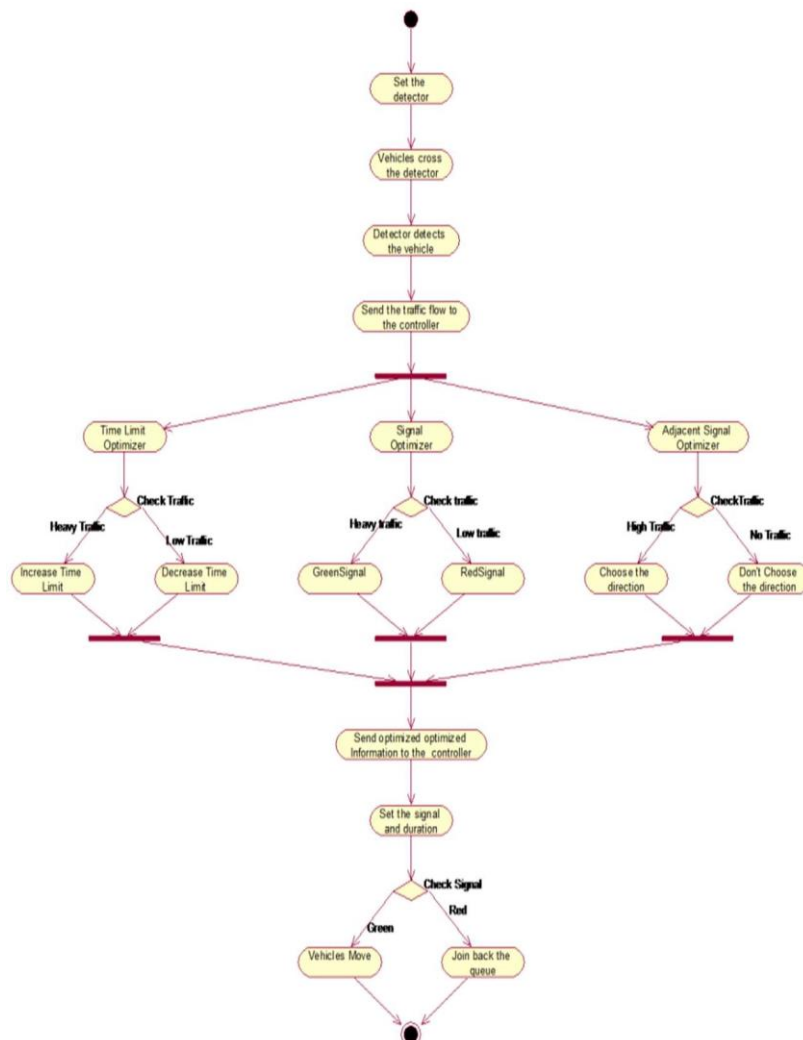
The state chart diagram illustrates the various states that an object or entity can be in during the process, such as number of vehicles , traffic light and the events or triggersthat cause the object to transition between these states.

3.6 ACTIVITY DIAGRAM

An activity diagram is a type of UML diagram that shows the workflow or activities of a system or process. It depicts the sequence of activities, decision points, and the flow of control or data between them.

The notations used are:

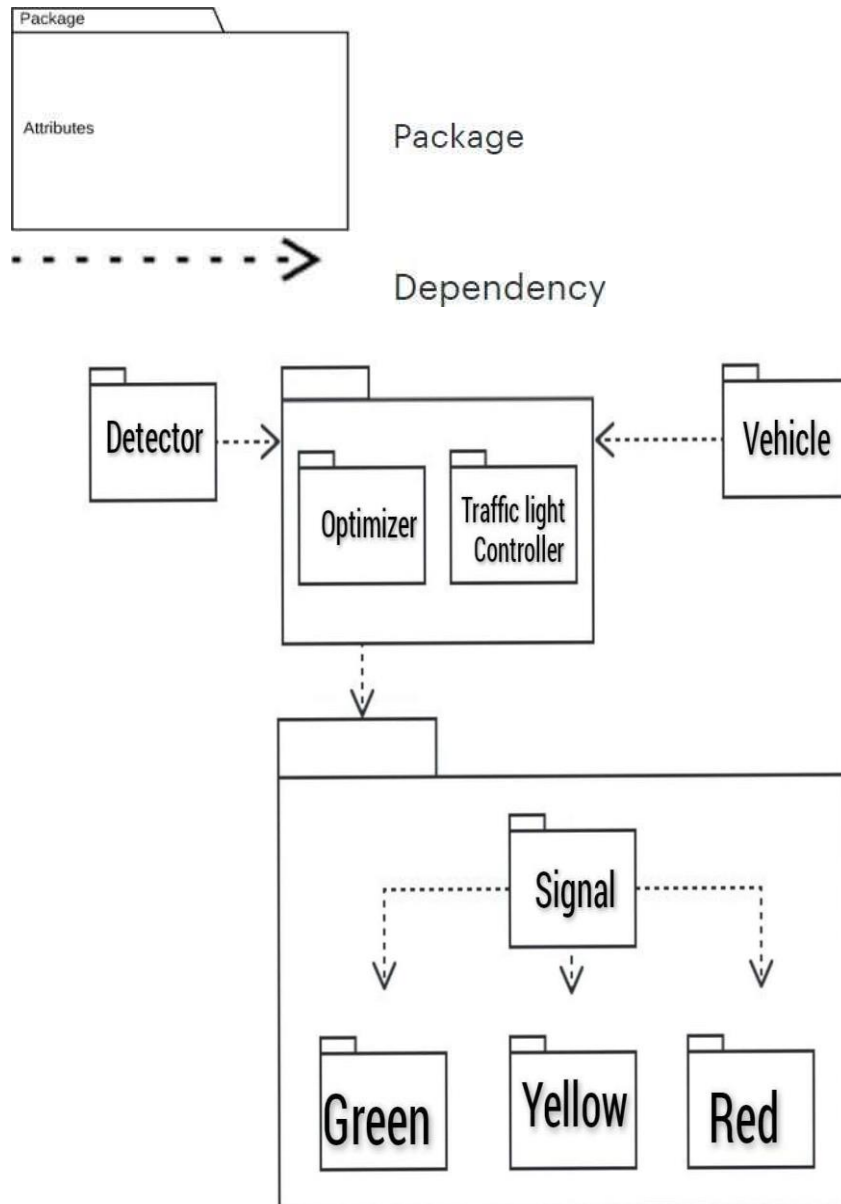
1. ● - Start symbol
2. ● - End symbol
3. □ - Activity symbol
4. ◇ - Decision symbol
5. → - Object flow arrow



An activity diagram of an traffic management system illustrates the flow of activities involved in the process , and the decision points or conditions that determine the path taken through the process

3.7 PACKAGE DIAGRAM

A package diagram is a type of UML diagram that shows the dependencies and organization of the packages or modules in a system. It depicts the relationships between packages and the elements they contain, such as classes, interfaces, and other packages.

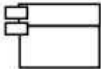




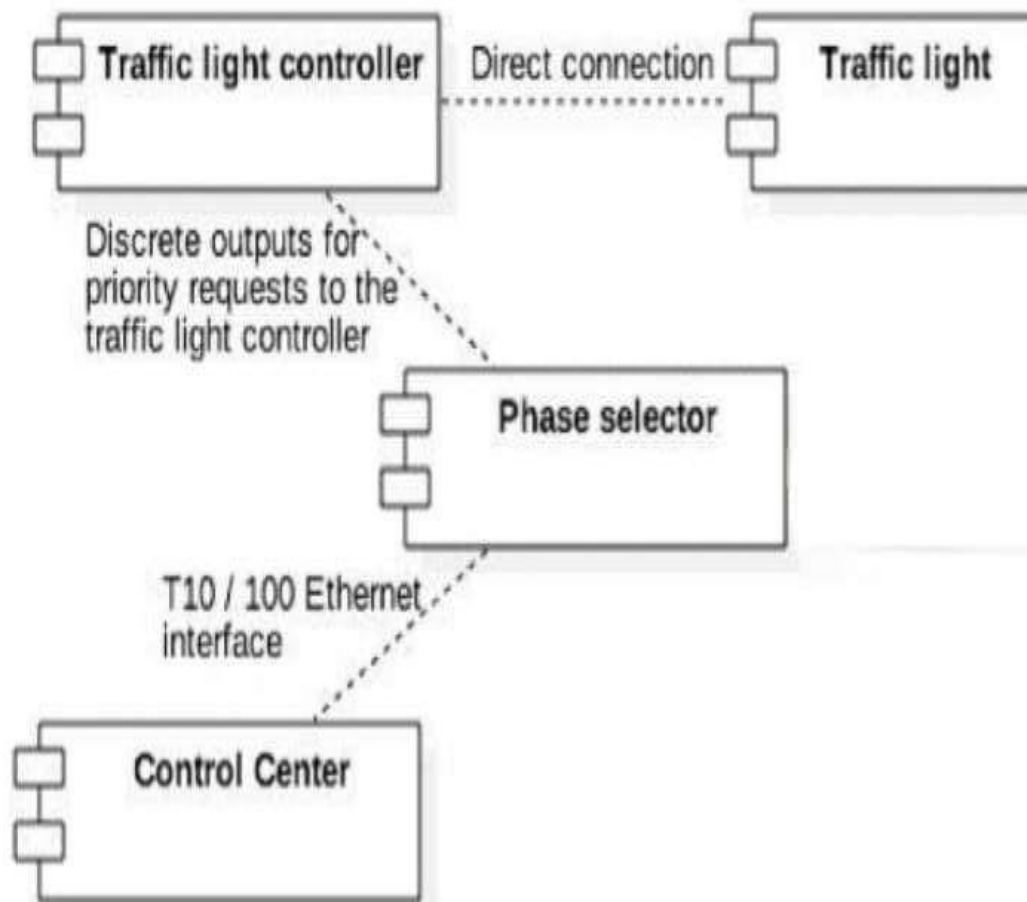
A package diagram of an traffic management system organizes the various components and modules of the system into logical groups or packages, showing the dependencies and relationships between them.

3.8 COMPONENT DIAGRAM

A component diagram is a type of UML diagram that shows the components, interfaces, and dependencies of a system or software application. It depicts the physical or logical components and their relationships, as well as the interfaces between them.

The notations used are:

1.  - Component symbol
2.  - Dependency arrow
3.  - Port symbol






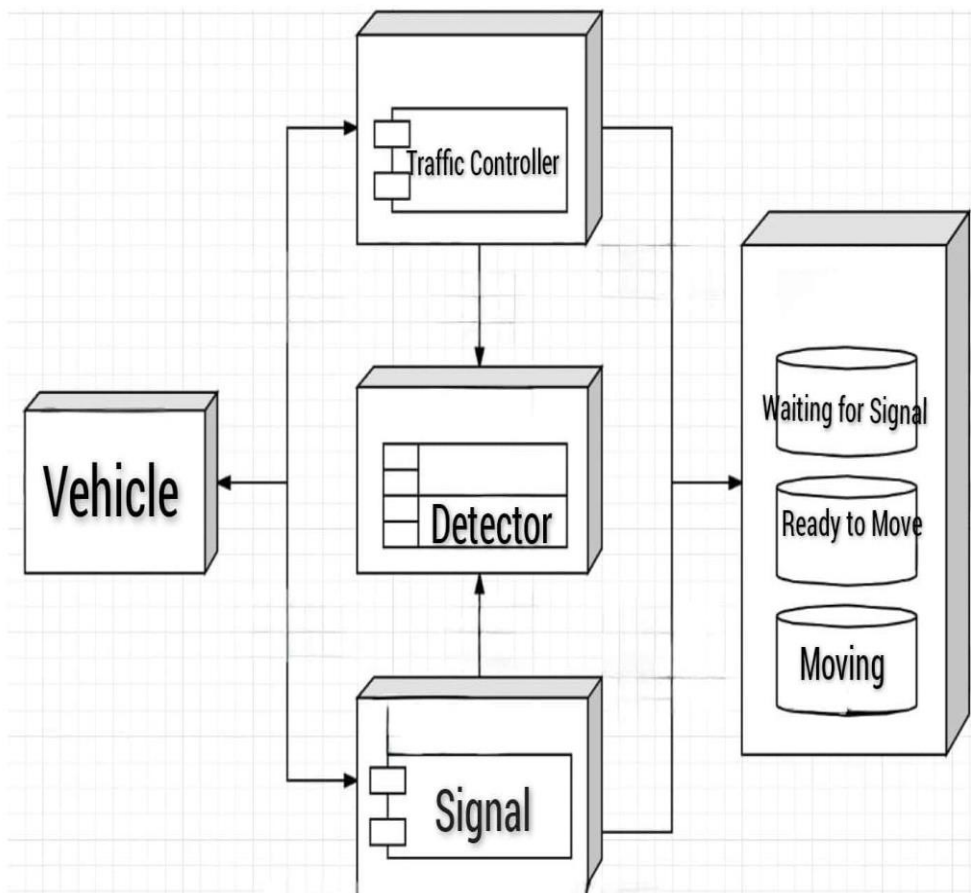
A component diagram of an traffic management system illustrates the physical components and modules of the system, such as databases, servers, APIs, and interfaces, and the dependencies and relationships between them, showing how they work together to provide the system's functionality.

3.9 DEPLOYMENT DIAGRAM

A deployment diagram is a type of UML diagram that shows the physical or logical deployment of a system or software application. It depicts the hardware or software nodes, the artifacts or components deployed on them, and the communication and dependencies between them.

The notations used are:

1.  - Artifact symbol
2.  - Association relationship
3.  - Dependency relationship



A deployment diagram of an traffic management system illustrates the physical deployment of the system's components and modules onto hardware or software nodes, such as servers, databases, and devices, and the connections and communication paths between them, showing how the system is deployed in a real-world environment.

4. CODING

```
#include <iostream>
#include <queue>
#include <unistd.h> // for sleep() function

using namespace std;

// Define the traffic light colors
enum LightColor { GREEN, YELLOW, RED };

// Define the traffic light class
class TrafficLight {
private:
    LightColor color;
public:
    TrafficLight() {
        color = GREEN;
    }
    void changeColor() {
        switch (color) {
            case GREEN:
                color = YELLOW;
                break;
            case YELLOW:
                color = RED;
                break;
            case RED:
                color = GREEN;
                break;
        }
    }
    LightColor getColor() const {
        return color;
    }
};

// Define the vehicle class
class Vehicle {
private:
    int id;
public:
    Vehicle(int id) {
```



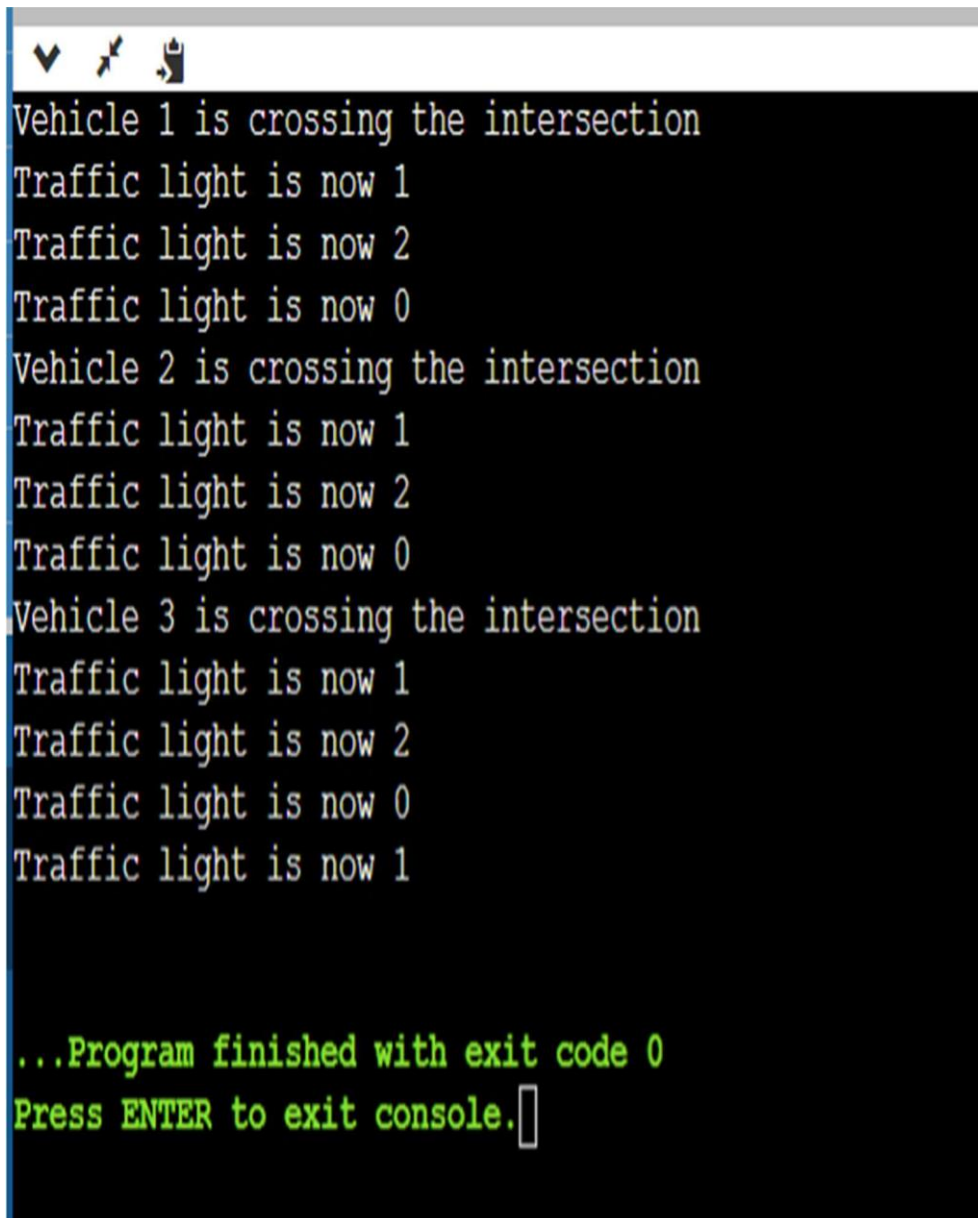
```

        this->id = id;
    }
    int getId() const {
        return id;
    }
};
// Define the traffic management system class
class TrafficManagementSystem {
private:
    queue<Vehicle> vehicleQueue;
    TrafficLight trafficLight;
public:
    void addVehicle(Vehicle vehicle) {
        vehicleQueue.push(vehicle);
    }
    void run() {
        int iterations = 0; // add termination condition
        while (iterations < 10) { // or a specific time has passed
            // Check the traffic light color and dequeue a vehicle if the light
            // is green
            if (trafficLight.getColor() == GREEN && !vehicleQueue.empty()) {
                Vehicle vehicle = vehicleQueue.front();
                vehicleQueue.pop();
                cout << "Vehicle " << vehicle.getId() << " is crossing the
intersection" << endl;
            }
            // Change the traffic light color every 5 seconds
            trafficLight.changeColor();
            cout << "Traffic light is now " << trafficLight.getColor() << endl;
            sleep(5);
            iterations++; // increment iteration counter
        }
    }
};

int main() {
    TrafficManagementSystem tms;
    tms.addVehicle(Vehicle(1));
    tms.addVehicle(Vehicle(2));
    tms.addVehicle(Vehicle(3));
    tms.run();
    return 0;
}

```

5. OUTPUT

A terminal window with a dark background and light green text. The window has a title bar with three icons: a downward arrow, a magnifying glass, and a document. The output text is as follows:

```
Vehicle 1 is crossing the intersection
Traffic light is now 1
Traffic light is now 2
Traffic light is now 0
Vehicle 2 is crossing the intersection
Traffic light is now 1
Traffic light is now 2
Traffic light is now 0
Vehicle 3 is crossing the intersection
Traffic light is now 1
Traffic light is now 2
Traffic light is now 0
Traffic light is now 1

...Program finished with exit code 0
Press ENTER to exit console.
```

6. CONCLUSION AND RESULTS

In conclusion, an traffic management system in C++ can be a powerful tool for managing traffic on busy intersection effectively and smartly. When properly designed and implemented, such a system can provide an efficient and reliable way to solve traffic problems and jams.

One of the key advantages of using C++ for an traffic management system is its ability to handle large volumes of vehicals, making it suitable for use in high-traffic environments. Additionally, C++ offers strong memory management capabilities, which can help ensure the system's stability and performance over time.

However, the success of an traffic management system in C++ also depends on factors such as its design, implementation, and maintenance. It should also be scalable and adaptable to changing technologies.

Overall, an traffic management system in C++ can be an asset for the people by managing the traffic more effectively. By leveraging the strengths of C++ we can create a powerful and reliable system that helps avoid traffic jams and inconvenience,

In conclusion, the result of an traffic management system in C++ can be highly effective if implemented correctly and maintained properly. With the right design, implementation, and maintenance, such a system can provide efficient and reliable management of traffic and thus solving many problems including time management

