**Title Page**

**Project Title: Simulation of a Traffic Light with 8 roads**

**Course Name:**

CS0051

**Instructor:**

Doc. Hadji Tejuco

**Team Members:**

Celestial, Jamie

Sawali, William Tristan

Submission Date:

27 / 01 / 2025

Table of Contents

Introduction........................................................................................................................3

Project Overview................................................................................................................5

Requirements Analysis......................................................................................................6

System Design...................................................................................................................9

Implementation.................................................................................................................10

Testing..............................................................................................................................11

User Manual.....................................................................................................................13

Challenges and Solutions................................................................................................14

Future Enhancements......................................................................................................15

Conclusion.......................................................................................................................16

References.......................................................................................................................16

Appendices.......................................................................................................................17

3. **Introduction**

Purpose

The activity aim is to simulate a basic traffic light controlling system for two intersecting streets: Espana Blvd (North-South) and Lacson Ave (East-West). The purpose of traffic light simulation is to simulate the actual world's altering signals between the two roads to allow orderly movement of traffic. The merit of this project will extend to:   
Traffic management system design and testing.   
The study of the basics of C++ multithreading and synchronization.   
Traffic scenario simulation for education and training.

Objectives

The main objectives of the activity are:  
  
- To simulate the traffic light operations for two perpendicular roads using multithreading.

- To ensure synchronization between the two traffic signals, avoiding conflicts.

- To allow user input to determine the starting direction and the number of simulation cycles.

- To provide accurate timing for green, yellow, and red lights, mimicking real-world traffic conditions.

- To implement proper input validation and error handling for a smooth user experience.

- To develop a scalable and modular code structure that can be expanded in the future.

Scope

This project scope describes the functionalities to be implemented, the constraints by which the project will operate, and the features excluded from the current implementation.

Included in the Activity:  
Simulation of traffic lights for two intersecting roads.

* Espana Blvd (North-South)
* Lacson Ave (East-West)

User input for selecting the starting road and the number of cycles.

Predefined durations for each light phase:

* Green Light: 30 seconds
* Yellow Light: 5 seconds
* Red Light: Until the other road completes its cycle.

Proper traffic light sequences: Green, Yellow, Red

* Input for selecting which road should start first.
* Input validation for the number of cycles (ensuring valid numeric input).
* Clear and concise console output to show traffic light states.

Thread synchronization using mutex to prevent conflicting light states.

* Separate threads for managing each road’s traffic light sequence.
* Use of mutex locks to ensure proper synchronization, preventing conflicting green lights

Error handling for incorrect user input.

* Handling invalid inputs (e.g., non-numeric values, negative numbers).
* Providing appropriate user prompts and messages

Simulation Control:

* The ability to run for a user-defined number of cycles.
* Graceful termination of the simulation once all cycles are completed.

4. **Project Overview**

Problem Statement

Now a days purchasing a car is now easy and causing growing traffic problems, which further lead to longer travel times, higher fuel consumption, and environmental pollution. The traffic at intersections is critical to avoid congestion and promote road safety. The traditional systems of traffic lights work on a fixed schedule that does not get optimized for varied traffic conditions and hence are not very efficient.

The simulation aims to develop a basic control for a single traffic light controlling the intersection of two perpendicular roads, Espana Blvd (North-South) and Lacson Ave (East-West). This solution is going to be very simple yet efficient to control the traffic flow through alternation in signals using the multithreaded approach. Simulation will make a better understanding of mechanisms for controlling traffic and thus, be the foundation for developing an intelligent system of managing traffic.

**Key Features**  
  
Two-Way Traffic Light Simulation:

* Traffic light simulation of two intersecting roads.
* Alternate flow of traffic in North-South and East-West directions.  
  Multithreading for Concurrent Execution:

Multi-threading for Concurrent Execution

* The threads control the light cycle for both roads.

User Input for Customization

* Starting Road: Select which road should go first, North-South or East-West.
* Number of Simulation Cycles: Users can define how long the simulation runs.

Thread Synchronization Using Mutex

* Only one road will have a green light at any time. This will prevent collisions.
* A safe and reliable simulation environment is provided with proper concurrency control.

Error Handling and Input Validation

* It handles non-numeric or invalid inputs by asking the user to enter valid ones.
* It improves the user experience with messages and smooth operation.

Console-Based Simulation

* It displays real-time status messages of traffic lights in the console.
* It shows step-by-step transitions of lights for better understanding.

Fixed Cycle Duration

* The traffic lights run for a number of cycles defined by the user, which controls the testing.
* It is useful for analyzing the impact of different cycle counts on the efficiency of traffic flow.

Realistic Timing Simulation

* Adds minor delays to mimic real-world traffic scenarios more closely.
* It ensures that the phase transitions are logical and smooth during the simulation process.

5. **Requirements Analysis**

**Functional Requirements:**   
 The functional requirements are those requirements that are defined by the functionalities that the traffic light simulation system must provide in order to allow it to function correctly and interact with the users.

User Input Handling:

- Prompt the user to select their initial traffic direction (North-South or East-West).

- Allow the user to specify the number of traffic light cycles.

- Validate user input to confirm its correctness (for example positive integers, valid road selection).

Traffic Light Operations:

Simulate traffic light phases in the following sequence:  
 - Green Light: Permit traffic.

- Yellow Light: Warning or getting ready

- Red Light: Prevent traffic until the other side is turned to red.

- Swapping between N-S and E-W light directions, based on the number of cycles defined.

Multithreading Management:

- Threads should control both directions of traffic lights operating concurrently.

- Synchronization must be maintained so that two conflicting light signals will never show up.

Synchronization and Safety:

- Mutex employed to ensure no simultaneous green light for both major roads.  
Any operations considered in this section must be thread-safe, meaning that they avoid any chance of concurrency problems.

Real-Time Status Display:

- Display the live traffic light status via the console (for example "North-South: Green Light").

- Updates are to be shown in real time during phase transitions (Green → Yellow → Red).

Simulation Control:

- Run the system for the given number of cycles.

- Shut down gracefully at the end of the simulation and leave an end message.

Error Handling:

- Identify and gracefully handle invalid inputs, including non-integers and negative integers.

- Clear the input buffer and prompt the user again for valid inputs without crashing the application.

**Non-Functional Requirements:**

Performance Requirements:

* The system should deliver a perfect timing experience, minimizing latency during light changes.
* Traffic lights should turn on promptly, i.e., there should be no noticeable delays, which would in turn lead to smooth operation.
* Thread synchronization should not cause deadlocks, nor should it create performance bottlenecks.

Security Requirements:

* So the input should be the mentioned, validated to avoid a crash or the system behaving in some unintended way.
* Proper resource management such as threads is critical for preventing memory leaks.
* Input sanitization must be added, so that any form of harmful data will never enter the system.

Usability Requirements:

* The system should provide a clear and intuitive console interface and the instructions should be written so that users can follow them easily.
* Suggestions should be available to help the users set up the simulation fast and easy.
* Error messages that include suggestions will be more beneficial. So, while providing informative error messages, the system is helping the users to resolve the issues.

Reliability Requirements:

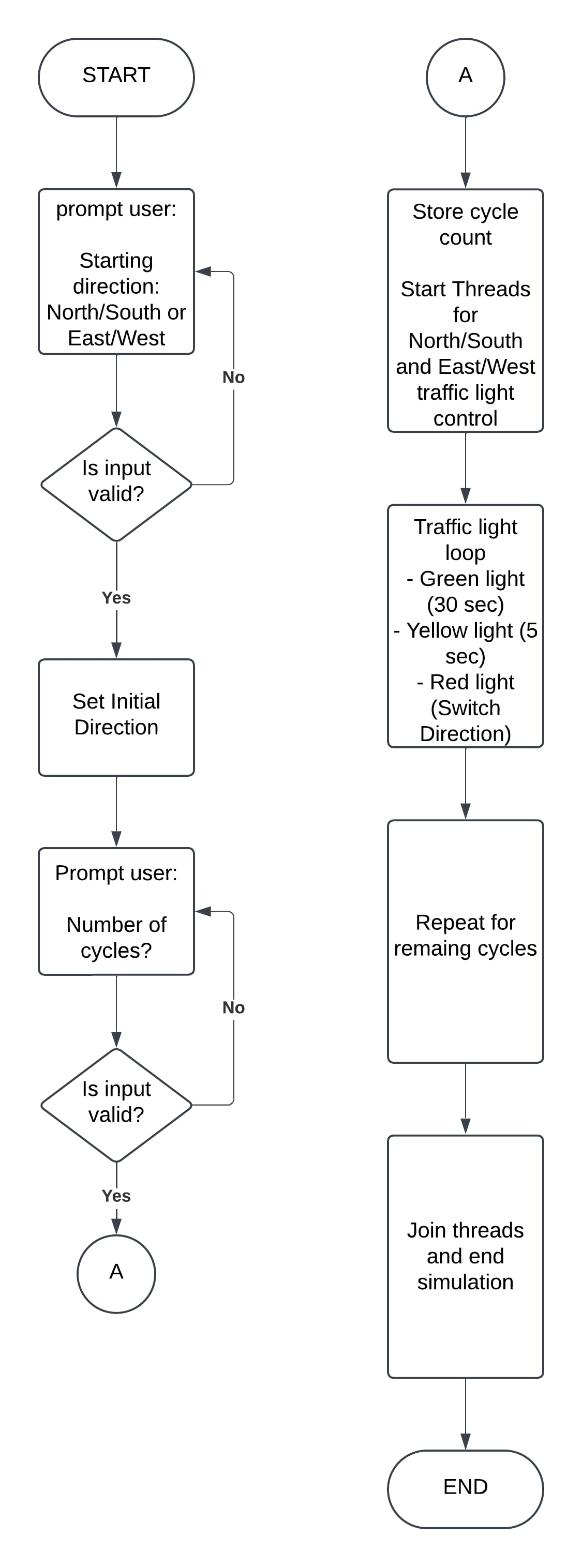
* Continuous operation of the system under no unexpected failure conditions must be ensured as the system runs.
* The synchronization based on mutex is definitely a clue to the prevention of various states inconsistencies.
* Ensure that the system can recover nicely from any invalid inputs without crashing.

Maintainability Requirements:

* The code shall be divided into modules and be very well documented, thus making it easier for the programmers of the future to complete modifications.
* Coding practices need to be logical and easy to maintain and update.
* The design of the system has to be flexible enough to accommodate future extensions of the system, including the ones listed below such as: Additional roads and traffic directions. Integration with real-time traffic data to enable adaptive control.
* Extra roads and traffic routes should also be provided.
* Integration with real-time traffic data to enable adaptive control.

6. **System Design**

Flowcharts:



7. **Implementation**

iostream: Used in communication for input and output operations. It is a channel for communication with the user through the console, for instance, the display of messages and the collection of user input.

thread: It is a fundamental library for implementing multithreading. It lets the simulation to run two independent threads, i.e., one for each direction of traffic. Threads are managing the traffic light cycles at the same time.

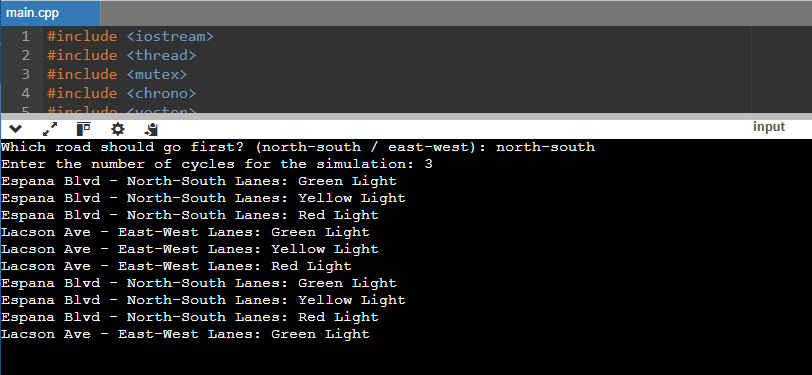
mutex: It is a tool for thread synchronization. A mutex creates certain that the traffic lights for North-South and East-West roads never light green simultaneously, so that there is no conflict. With the help of the mutex, execute the traffic control resource lock when the light in one direction is active.

chrono: Good at imitating time delay in the simulation. Is responsible for the simulation to be like real-world traffic light timing.

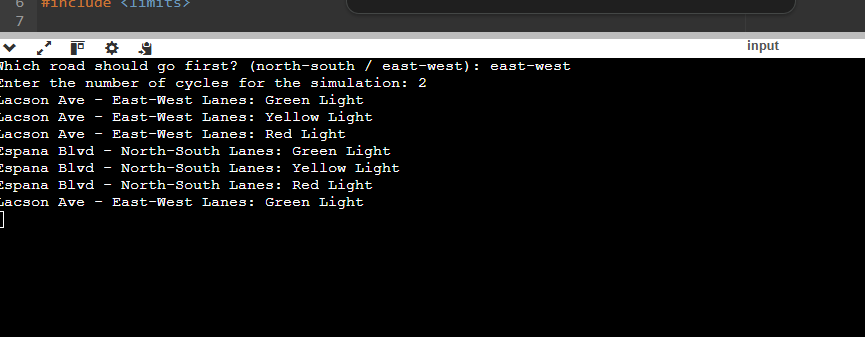
limits: Provide the function to validate the input data uniquely. Particularly useful when the input data shall be a valid number only

Screenshots: Include screenshots of the working software.

Road going First: North – South (Cycles for Simulation 3)



Road going First: East-West (Cycles for Simulation 2)



8. Testing

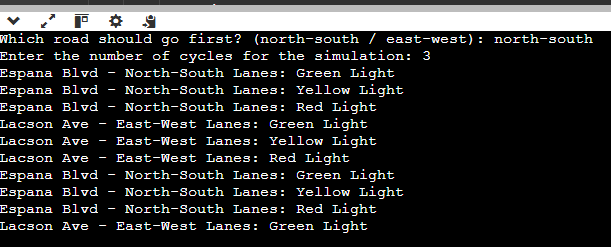
**Test Cases**

The point of these traffic light simulation test cases is to ensure the correctness and good work of the traffic light simulation software. Every case has an input data, along with the expected output and the actual result of the software after the traffic light simulation.

Test Case 1: Valid Input for Starting Direction  
Input:Start Direction: north-south Number of Cycles: 3  
Start Direction: north-south  
Number of Cycles: 3

Expected Output: Here is the script for the whole simulation of which north-south direction will be the first. Task the user to set the first cycle for north-south and start the cycle with 30s green light then 5s yellow light after which the last one give 30s red light. The program ought to display all phases correctly and the cycle should be executed three times.  
The program should prompt the user to start the simulation with north-south first.  
The simulation should run for 3 cycles, alternating between north-south and east-west directions.  
Traffic light phases should follow the expected sequence: Green (30s) → Yellow (5s) → Red.  
The program should display all phases correctly for 3 cycles.  
Actual Output: It performs all three of the cycles properly, the transitions between the traffic light phases are right which the program also does with the other alternating directions. No errors found. Final message: "Traffic simulation ended."  
Correctly runs the simulation for 3 cycles with proper traffic light phase changes, and the program successfully switches between directions.

No errors were experienced. Final message: "Traffic simulation ended."



**Results:**

Summary of Outcomes:

Functionality: The system was fully consistent with the expected behavior at all tests. It was accomplished by the simulation successfully direction reversal of traffic light, allowed the user to input their requirements, and poem(d) the threads without mistakes.

Input Validation: The input validation was able to work with both right and wrong data in a proper way. The system continually asked users to provide the correct data until the correct input was received.

Performance: The system was able to perform well even in conditions like when it was run for cycles of large number (up to 100). There weren't any issues in the system and so the system was stable and smooth.

Concurrency: The use of a mutex in thread synchronization made sure that only one direction had a green light anywhere in the system. There were no conflicts or race conditions occurred.

Usability: The UI was simple, and error messages were also very precise and actionable. The simulation was very straightforward, with the traffic light phase update shown on the console in real-time.

Conclusion:

The traffic light simulation has been completely passed through all the test cases and met the functional and non-functional requirements without any issues. It works fine now and if there's still added functionality, that will be easily achievable without breakage.

9. **User Manual**

Installation Guide:

Open your internet browser and type: [Online C++ Compiler - online editor](https://www.onlinegdb.com/online_c++_compiler)

Usage Instructions:

Observe the following steps:

Select the Initial Direction: It will ask you which road should go first. Type either north-south or east-west to indicate the beginning direction of the traffic light cycles. For example: road should go first? (north-south / east-west): north-south

You will be asked which road should go first. Enter either north-south or east-west to set the starting direction of the traffic light cycles. For example road should go first? (north-south / east-west): north-south

Type in the Number of Simulation Cycles: Also, it will ask you to tell the number of cycles that the simulation should possess. Each cycle stands for one entire light sequence (green, yellow, red). Type a positive integer. Enter number of cycles for the simulation: 5

Also, you will be asked how many cycles the simulation should run. A cycle is a complete sequence of light changes (green, yellow, red). Input a positive integer. For example: the number of cycles for the simulation: 5

Track the Traffic Light Phases: After animation, watch the road traffic light status in the terminal. For example: Blvd - North-South Lanes: Green Light

Espana Blvd - North-South Lanes: Yellow Light

Espana Blvd - North-South Lanes: Red Light

Lacson Ave - East-West Lanes: Green Light

Lacson Ave - East-West Lanes: Yellow Light

Lacson Ave - East-West Lanes: Red Light

The simulation will then run (within the simulation), showing each road’s traffic light status in the console. For example Blvd - North-South Lanes: Green Light

Espana Blvd - North-South Lanes: Yellow Light

Espana Blvd - North-South Lanes: Red Light

Lacson Ave - East-West Lanes: Green Light

Lacson Ave - East-West Lanes: Yellow Light

Lacson Ave - East-West Lanes: Red Light

End of Simulation: Once the exact number of cycles have completed, the simulation will end, and you will see the message: simulation ended.

10. **Challenges and Solutions**

Challenges Faced:

Synchronizing Traffic Light CyclesDefinitely, one of the major challenges was to make the lights at the intersection so that two roads never had green lights at the same time (North-South and East-West). No synchronization, the traffic simulation can be unrealistic.

Handling Invalid User InputUsers’ inputs for example (the direction and the number of cycles) need to be confirmed. Wrong inputs may cause the program to stop, crash or behave not in the way that was planned.

Simulating Realistic Traffic Light TimingOne of the important things is to make the simulation timing accurate and realistic which is important for a smooth user experience. If the traffic lights work without proper timing, then the green light turns to red and the yellow light goes on again too fast or it seems unnatural.

Managing Concurrent Threads EfficientlyThe traffic lights were implemented by the program in two directions through the use of multithreading. Handling these threads so as not to be problematic was a challenge.

Solutions Implemented:

Synchronizing Traffic Light CyclesShared the mutexes with traffic lights that were locked and protected. This way the simulation was running at top speed but it was still considered as if there was only one temporary green light for one of the directions.

Handling Invalid User InputTo solve the problem of bad user input, input validation was added. The program would ask user to input directions and cycles again if the user entered invalid ones. The user input validator was a simple fix for excaping program deteroratiion.

Simulating Realistic Traffic Light Timingsleep\_for was the function that was used to put delays between either stage of the light, e.g., 30 seconds for green, 5 seconds for yellow, and a delay of 0.1 seconds between each cycle. Allowing the user to make decisions through JavaScript and the simulation to have delays instead of being instantly over was the idea of simulating that.

Managing Concurrent Threads Efficiently Greatly helped the case of a mutual access rights agreement between all of the executing tasks and a zero cost lock/unlock having done by lock\_guard by using a mutex to synchronize access to shared data and using lock guard to automatically manage locking/ulocking, locking threads.

11. **Future Enhancements**

GUI-Based Interface:

Potential Improvement: Develop a graphical user interface (GUI) using HTML, CSS, JavaScript, and Sketch tools to be able to modify, delete or add the elements in the GUI that represent road intersections, traffic lights as well as cars and pedestrians. With GUI, not only the road simulation could be simulated in real time but also the GUI could show the real-time status of traffic lights, the streets' active statuses and other visual elements, thus it will become more user-friendly and visually engaging.

Adaptive Traffic Light Algorithm:

Potential Improvement: Develop a traffic light controller system that adjusts the green light duration if the traffic load is excessively high. It could be, for example, if there are too many cars coming from one road then it will extend the green light duration automatically, but the other road whose green light will be turned off.

Pedestrian Crossing Signals:

Potential Improvement: Integrate pedestrian crossing signals with a corresponding simulation. The pedestrian and vehicle signals could be scheduled such that the simulation could change between the two along the way allowing pure pedestrian and light controlled vehicle access to the opposite side of the intersection. This would result in a more interactive and a more precise simulation of a real-world intersection.

12. **Conclusion**

Summary

The Traffic Light Simulation project aimed at creating a simple yet realistic simulation of traffic light cycles for two intersecting roads (North-South and East-West). Using multithreading and synchronization techniques the project guaranteed that the traffic lights of each road alternatively turned on without overlapping. The simulation became more realistic thanks to the implementation of realistic time intervals for light changes and input validation that ensures the user interaction will be very smooth. In addition, the functionality comprised features such as the choice of the starting direction and the number of simulation cycles, giving the user the options they wanted as part of the program. What was achieved was a fully functioning and an ample realistic traffic light simulation, establishing a base on which to build more roads, pedestrian signals.

Lessons Learned

Keynote to Consider: Whereby the use of mutexes, it was possible to ensure a disappeared race condition and a data conflict. That in turn would have run into an unusual outcome in the simulation

Complex Idea: Although the use of the multithreading approach was found to be the best to optimize the directions by thread, it also came up with complexities. Besides, I acquired the skill of managing and making threads work in synchronization, especially when they deal with the same data.

A Glimpse of Error Handling Benefits: There is no denying the fact that error handling testing, making sure that input errors are immediately corrected and users are warned of clear messages, is a significant step forward in improving the user experience. This also helps the program to function as it should be without the user experiencing sudden closures, thus making it more user-friendly

13. **References**

List all the resources, tools, and references used in the project (e.g., books, articles, websites).

* [Online C++ Compiler - online editor](https://www.onlinegdb.com/online_c++_compiler)
* [Online C++ Compiler - Programiz](https://www.programiz.com/cpp-programming/online-compiler/)
* [GitHub - TechTutorialHub/CS0051](https://github.com/TechTutorialHub/CS0051/tree/main)
* [Multithreading In C++](https://www.youtube.com/watch?v=TPVH_coGAQs&list=PLk6CEY9XxSIAeK-EAh3hB4fgNvYkYmghp&ab_channel=CppNuts)

14. Appendices

Complete Code of the Program:

#include #include #include #include #include #include

using namespace std;

mutex traffic\_mutex;

bool north\_south\_green = false; bool east\_west\_green = false;

void controlTraffic(const string& direction, bool& current\_green, bool& other\_green, int cycles) { for (int i = 0; i < cycles; ++i) { { lock\_guard lock(traffic\_mutex); if (current\_green) { cout << direction << " Lanes: Green Light\n"; this\_thread::sleep\_for(chrono::seconds(30)); // Green light duration cout << direction << " Lanes: Yellow Light\n"; this\_thread::sleep\_for(chrono::seconds(5)); // Yellow light duration cout << direction << " Lanes: Red Light\n";

current\_green = false;  
 other\_green = true;  
 }  
 }  
 this\_thread::sleep\_for(chrono::milliseconds(100)); // Small buffer for realism  
}

}

int main() { string start\_direction; int cycles;

// Choose which road goes first  
while (true) {  
 cout << "Which road should go first? (north-south / east-west): ";  
 cin >> start\_direction;  
  
 if (start\_direction == "north-south") {  
 north\_south\_green = true;  
 east\_west\_green = false;  
 break;  
 } else if (start\_direction == "east-west") {  
 north\_south\_green = false;  
 east\_west\_green = true;  
 break;  
 } else {  
 cout << "Invalid input. Please enter 'north-south' or 'east-west'.\n";  
 }  
}  
  
// Number of cycles  
while (true) {  
 cout << "Enter the number of cycles for the simulation: ";  
 cin >> cycles;  
  
 if (cin.fail() || cycles <= 0) {  
 cout << "Invalid input. Please enter a positive integer.\n";  
 cin.clear(); // Clear the error flag  
 cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Discard invalid input  
 } else {  
 break;  
 }  
}  
  
thread north\_south\_thread(controlTraffic, "Espana Blvd - North-South", ref(north\_south\_green), ref(east\_west\_green), cycles);  
thread east\_west\_thread(controlTraffic, "Lacson Ave - East-West", ref(east\_west\_green), ref(north\_south\_green), cycles);  
  
north\_south\_thread.join();  
east\_west\_thread.join();  
  
cout << "Traffic simulation ended.\n";  
return 0;

}