CSE 564: Software Design
Spring 2023

CYBER PHYSICAL TRAFFIC CONTROL SYSTEM

Team 5		
Kanav Sharma	Jacob Jose	Sai Varun Vaka
7920-972	1667-713	9368-107
MS in Software Engineering	MS in Software Engineering	MS in Computer Science
School of Computing and Augmented Intelligence		
Arizona State University, Tempe, AZ, USA, 85281		

Table of Contents

		Kanav Sharma *	**Jacob Jose**	**Sai Varun Vaka**		
Parts	**Part**	**% Effort**	**% Effort**	**% Effort**	**Points ****	**Earned Points**
Problem description	1	35%	40%	25%	5	
SRC design specifications with description	1	33%	33%	34%	25	
UML design specifications with description	1	33.3%	33.3%	33.3%	25	
Implementation	2	35%	30%	35%	10	
Experiments & results	1	30%	40%	30%	10	
Conclusions***	1	33%	33%	34%	5	
Demonstration	2	33.3%	33.3%	33.3%	10	
Presentation	2	33.3%	33.3%	33.3%	10	
Code quality	2	40%	20%	40%	5	
Report quality	1	30%	20%	50%	5	
Total					110	

	Team member contributions			
	What went well?	What could have been done better as part of this project?	What were the learnings?	How was working with this team?
Kanav Sharma (Team Leader)	The team was quick with adapting the concepts of Synchronous Reactive Components. Assignments given during the course were of great help to solidify the concepts learnt during the class. Adopting a "design first and code later" approach shifted our mindset and enabled us to identify potential software loopholes early in the project lifecycle.	While I found Webster to be functioning as intended and providing adequate green time, I believe that there may be more efficient algorithms available that could improve performance. Additionally, I believe that we could have been more proactive in seeking feedback from other teams working on the same project topic. While we received feedback from our professor, exchanging ideas and suggestions with other teams could have been helpful in identifying potential improvements and best practices.	Through my experience, I realized that effective communication with the team is crucial, especially when dealing with task dependencies. I also learned the importance of utilizing team meetings to their full potential, by posing questions that prompt the entire team to come to a consensus on a solution before proceeding. This approach helped avoid potential inconsistencies in our assumptions.	The team was very collaborative, responsive and owned their task well. Working with this team was a positive experience overall. As the team lead, I focused on fostering a collaborative and communicative environment, which helped ensure that everyone was on the same page and that we could work effectively together. I found that each team member brought unique strengths and perspectives to the project, which made the overall outcome stronger.
Jacob Jose	Even though we were required to give the presentation first, we coordinated well to finish the project on time and deliver it.	We should have used third party websites to make the scenario's explanation better with graphical representation.	Generating test case scenarios to understand what went wrong or what is right in the component can help to design the application in a correct manner.	The project involved multiple team members with different skills and expertise, but we understood each other's perspective. Effective collaboration and teamwork can help to ensure that everyone's contributions are valued, and that the project is completed on time.

Sai Varun Vaka	Discussion for designing the SRC components like how variables can be utilized and how feedback loop can be implemented for cyber physical systems.	Multi-threading could have been implemented for concurrent execution of getting the vehicle data and calculation of the green time.	Using synchronous reactive components can help to simplify the software design and make it more efficient. This approach can help to reduce the likelihood of bugs and errors in the code and can make it easier to maintain and update the software over time.	Working with the team on this project was a great experience overall. One of the positive experiences was the collaborative spirit of the team. We were all committed to the success of the project, and everyone was willing to lend their expertise and support to each other. For example, when one of us got stuck with designing a light signal controller component, we resolved it by discussing among ourselves by verifying the rounds of the component.

1. PROBLEM DESCRIPTION

Traffic congestion is a major problem faced by urban areas around the world, leading to increased travel time, fuel consumption, and greenhouse gas emissions. The traditional traffic control systems based on fixed timings and simple traffic models are not sufficient to handle the dynamic traffic flow in modern cities. To address this issue, a traffic signal control cyber physical system is proposed, which integrates real-time traffic data from multiple sources, such as sensors, cameras, and GPS devices, to dynamically adjust the traffic signal timings and optimize traffic flow. The objective of this system is to reduce traffic congestion, travel time, fuel consumption, and greenhouse gas emissions, while improving overall traffic safety. However, the development of such a system poses several challenges, such as data collection and processing, algorithm design, system implementation, and evaluation. This project aims to design, implement, and evaluate a traffic signal control cyber physical system, which can effectively optimize traffic flow and improve the overall transportation system in urban areas. Most of today's road traffic control systems are currently based on pre-defined time intervals and do not consider real-time changes in traffic or emergency evacuations. While this approach may be simple and cost-effective, it inevitably leads to excess congestion due to its inability to adapt to changing traffic patterns.

PROPOSED SOLUTION

For this project, we propose a cyber physical system (CPS) that utilizes a centralized traffic controller to make real-time traffic-light-change decisions based on data received by implemented road sensors. CPS can incorporate various sensors, cameras, and GPS devices to monitor traffic conditions in real-time. This data can be used to adjust traffic signal timings, optimize traffic flow, and detect traffic incidents, such as accidents or road closures. CPS can use advanced algorithms to dynamically adjust the timing of traffic signals based on real-time traffic data. By optimizing the flow of vehicles through intersections, CPS can reduce congestion and improve travel time. For this proposal, we will be focusing on road sensors to register/count the number of cars at an intersection at a given time, and a traffic light controller that will decide the light change based on the number of cars, velocity of cars and direction of its movement. Additionally, our cyber physical system will include sensors for receiving communication requests from emergency vehicles to immediately alter the current traffic signal.

- Type of Sensors: Inductive loop detector.
- Data Collection and processing
- Webster's Algorithm

2. DESIGN

Assumptions:

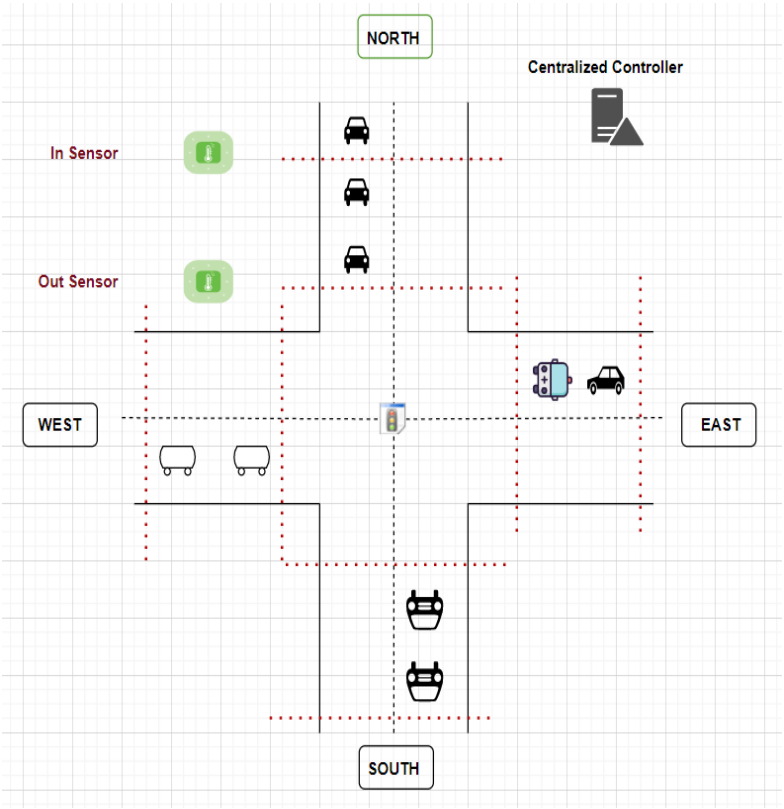
- All Vehicles will move in a straight road. No Vehicle will take a left or right turn at the intersection.
- Vehicles do not break down while moving and will not take a U turn before the intersection.
- Emergency Vehicles have a special radio frequency transmitter for identification. Radio frequency sensors will identify emergency vehicles through the transmitter.
- Each lane has an individual sensor and summation of cars calculated by each lane will be considered as input.
- Only one emergency vehicle can be at the intersection at a time.
- If there is an emergency vehicle at any intersection, our component allocates 20 seconds to it pass, irrespective of green time allocation done by the webster module.
- The North-South bound traffic signal is Green at the initial state & East West bound traffic signal is Red at the initial state.

Algorithm for traffic-responsive adaptive signal control:

- Collect real-time traffic data from sensors at each approach road.
- Calculate the volume and direction of vehicles approaching each intersection.
- Calculate the optimal green time for each road based on the current traffic conditions. This can be done using Webster's formula, which takes into account the volume, speed, and delay of vehicles at each approach.
- Adjust the signal timings based on the calculated green time for each approach road.
- Monitor the traffic flow and adjust the signal timings as necessary based on the real-time traffic data. For example, if the traffic conditions change and congestion increases, the green time for the congested approach will be extended to reduce delays.

Glimpse of intersection and sensors:

At intersections, there are corridors running in the north-south and east-west directions. To determine the net traffic waiting for the traffic lights to turn green, two sensors (shown as red dotted marks) are placed: one near the intersection to count the vehicles moving out, and another 200 meters away to count the vehicles moving in. The traffic controllers use the data from both sensors to make their calculations.



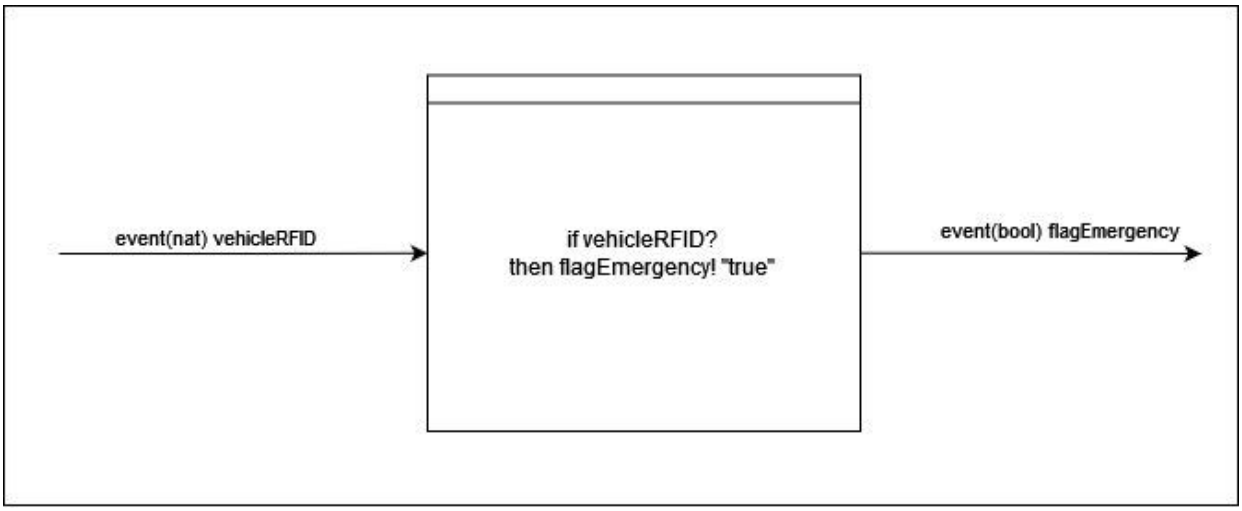
2.1. SRC design specifications with descriptions

Emergency Vehicle RFID Receiver:

The SRC is intended to function as an emergency vehicle sensor and is not a software component.
RFID transmitters are installed in emergency vehicles, which transmit a unique RFID signal. The SRC detects the RFID signal and generates an event value (bool) as output.

Below is the {I,O,S,Init,React} for Emergency Vehicle RFID Receiver

- Input I: {vehicleRFID }
  vehicleRFID is of type event(nat).
- OutputO:{flagEmergency}
  flagEmergency is a type of event(bool).
- State variables S: NA
- Init: NA. As there are no state variables, initialization is not required.
- React: Refer SRC components below.

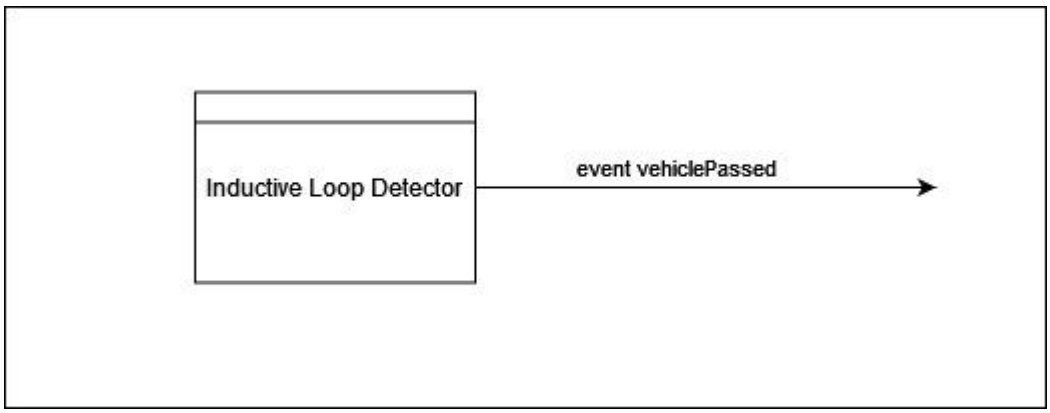


Inductive Loop Detector

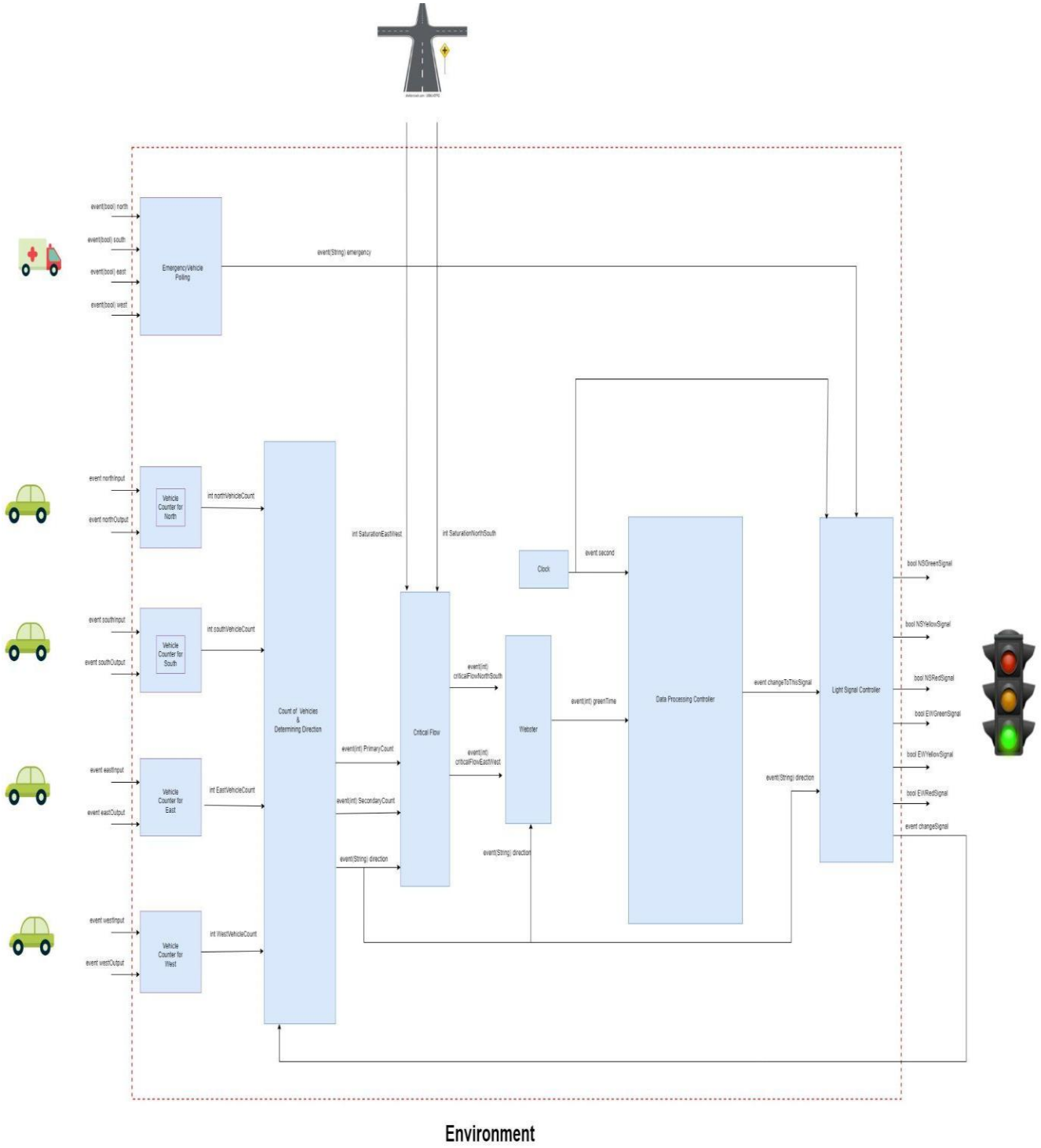
The SRC is intended to function as an vehicle sensor and is not a software component. An inductive loop detector is a wire that is installed on the road to detect vehicles. When vehicles pass over these sensors, it gives an output of type 'event'.

Below is the {I,O,S,Init,React} for Inductive Loop Detector

- Input I: NA
- OutputO:{vehiclePassed}
  vehiclePassed is a type of event.
- State variables S: NA
- Init= NA. As there are no state variables, initialization is not required.
- React = Refer SRC components below.



Composite Block diagram



Composite SRC diagram

Below figure represents the composite SRC diagram for the Cyber Physical Traffic Control System. This SRC consists of the 13 tasks in the task graph.

Task A1:

Read Set: {northInputVehicle, northOutgoingVehicle}
Write Set: {northCounter}

This initial task takes the input from the sensor present at the north road. It stores the current count of vehicles present at the road in the state variable. Updates the current number of vehicles present by checking incoming vehicles and outgoing vehicles. It updates the current number of vehicles present at North to Task A5.

Task A2:

Read Set: {southInputVehicle, southOutgoingVehicle}
Write Set: {southCounter}

This initial task takes the input from the sensor present at the south road. It stores the current count of vehicles present at the road in the state variable. Updates the current number of vehicles present by checking incoming vehicles and outgoing vehicles. It updates the current number of vehicles present at South to Task A5.

Task A3:

Read Set: {eastInputVehicle, eastOutgoingVehicle}
Write Set: {eastCounter}

This initial task takes the input from the sensor present at the east road. It stores the current count of vehicles present at the road in the state variable. Updates the current number of vehicles present by checking incoming vehicles and outgoing vehicles. It updates the current number of vehicles present at East to Task A5.

Task A4:

Read Set: {westInputVehicle, westOutgoingVehicle}
Write Set: {westCounter}

This initial task takes the input from the sensor present at the west road. It stores the current count of vehicles present at the road in the state variable. Updates the current number of vehicles present by checking incoming vehicles and outgoing vehicles. It updates the current number of vehicles present at West to Task A5.

Task A5:

Read Set: {changeSignal, LastExecution, northCounter, southCounter, eastCounter, westCounter}
Write Set: {LastExecution, countPrimaryRoad, direction, countSecondaryRoad}

This task keeps the control of the application by deciding whether to send the output or not. This task tries to execute only when the output of task A12 sends the event to this task. If no event input is present it doesn't execute. Note that other tasks A1, A2, A3, A4 still execute by keeping the count of the vehicles. Task determines the direction to be changed to green by considering changeSignal input and by checking the number of vehicles present at the intersection in all the directions. It updates the number of vehicles at primary road, secondary road and determined direction.

Task A6:

Read Set: {countPrimaryRoad, countSecondaryRoad, direction, saturationNS, saturationEW}
Write Set: {criticalFlowNS, criticalFlowEW}

This task accumulates the values required for the calculation of the green time. It consists of a subpart of the webster algorithm where the values like road dimensions, number of vehicles traveling from each direction per hour. It determines the values for the critical flow in both directions to understand the ratio of the vehicles present with respect to the vehicles traveling per hour.

Task A7:

Read Set: {criticalFlowNS, direction, criticalFlowEW, countPhase, LostTime, AllRed, L, ya, OCL}
Write Set: {ya, L, OCL, greenTime}

This task calculates the greenTime needed for a particular direction with respect to the vehicles in all the directions. This takes the consideration of a few variables like criticalFlow of both the directions, LostTime, AllRed, countPhase etc. and uses Webster's formula to get the greenTime for the direction which is going to change.

Task A8:

Read Set: {greenTime }
Write Set: {greenTimecounter}

This task checks for the event and maintains the value in the state variable greenTimecounter.

Task A9:

Read Set: {second, greenTimecounter }
Write Set: {greenTimecounter}

This task checks for the event second which is a timer for second. It also has a condition for greenTimecounter being greater than 0, if it is greater than 0 then the greenTimecounter will be decremented.

Task A10:

Read Set: {greenTimecounter }
Write Set: {changeSignalWithoutEmergency}

This task triggers the event changeSignalWithoutEmergency when the greenTimecounter equals to 0.

Task A11:

Read Set: {emergency}
Write Set: {emergency, counter, dir, isEmergency}

This task checks for the presence of any emergency vehicle being detected. If it is detected it updates the values needed to trigger an emergency.

Task A12:

Read Set: {isEmergency, second, counter, changeSignalWithoutEmergency, direction}
Write Set: {counter, dir, NSgsLatch, EWrsLatch, EWgsLatch, NSrsLatch, isEmergency, NSysLatch, EWysLatch, changeSignal}

This task checks if there is an emergency and changes the values for the traffic lights according to the emergency direction received. If there is no emergency vehicle, it changes the green light for the direction which it received.

Task A13:

Read Set: {northEmergency, southEmergency, eastEmergency, westEmergency }
Write Set: {emergency}

This task checks for any emergency vehicle being present at the intersection, if there is an emergency vehicle event string emergency is triggered.

This task graph follows all required rules:

- The precedence relation between them is Acyclic.

$A1 \prec A2 \prec A3 \prec A4 \prec A13 \prec A5 \prec A6 \prec A7 \prec A8 \prec A9 \prec A10 \prec A11 \prec A12$

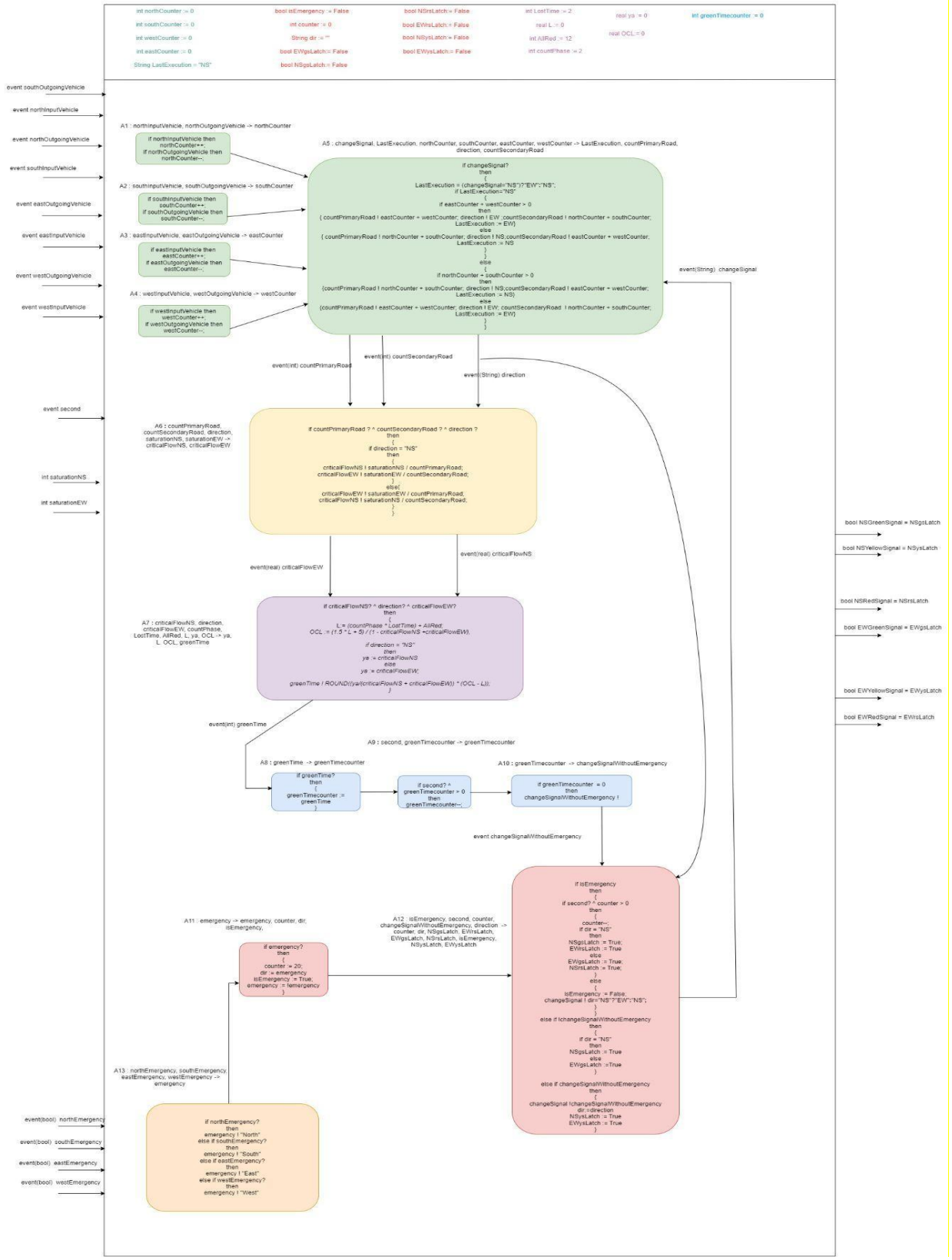
- Output variables {NSgsLatch, EWrsLatch, EWgsLatch, NSrsLatch, NSysLatch, EWysLatch } belong to write set of only one task A12.

- Compatibility in variable names: There is no name conflict in state variable names of all the atomic components.

- Disjoint Output Sets: All the output variables of components are disjoint to each other's output variable set.

- Acyclicity of await dependencies in input-output variables:

    - northCounter $\succ$ northInputVehicle, northOutgoingVehicle,
    - southCounter $\succ$ southInputVehicle, southOutgoingVehicle,
    - eastCounter $\succ$ eastInputVehicle, eastOutgoingVehicle ,
    - westCounter $\succ$ westInputVehicle, westOutgoingVehicle,
    - countPrimaryRoad, direction, countSecondaryRoad $\succ$ changeSignal, northCounter, southCounter, eastCounter, westCounter
    - criticalFlowNS, criticalFlowEW $\succ$ countPrimaryRoad, countSecondaryRoad, direction, saturationNS, saturationEW
    - greenTime $\succ$ criticalFlowNS, criticalFlowEW
    - hangeSignalWithoutEmergency $\succ$ greenTime, second,
    - NSgsLatch, EWrsLatch, EWgsLatch, NSrsLatch, isEmergency, NSysLatch, EWysLatch, changeSignal $\succ$ second, emergency, changeSignalWithoutEmergency


int northCounter := 0
int southCounter := 0
int westCounter := 0
int eastCounter := 0
String LastExecution = "NS"
bool isEmergency := False
int counter := 0
String dir := ""
bool EWgsLatch:= False
bool NSgsLatch:= False
bool NSrsLatch:= False
bool EWysLatch:= False
bool NSysLatch:= False
bool EWysLatch:= False
int LostTime := 2
real L := 0
int AllRed := 12
int countPhase := 2
real ya := 0
real OCL := 0
int greenTimecounter := 0
event southOutgoingVehicle
event northInputVehicle
event northOutgoingVehicle
event southInputVehicle
event eastOutgoingVehicle
event eastInputVehicle
event westOutgoingVehicle
event westInputVehicle
A1 : northInputVehicle, northOutgoingVehicle -> northCounter
if northInputVehicle then northCounter++;
if northOutgoingVehicle then northCounter--;
A2 : southInputVehicle, southOutgoingVehicle -> southCounter
if southInputVehicle then southCounter++;
if southOutgoingVehicle then southCounter--;
A3 : eastInputVehicle, eastOutgoingVehicle -> eastCounter
if eastInputVehicle then eastCounter++;
if eastOutgoingVehicle then eastCounter--;
A4 : westInputVehicle, westOutgoingVehicle -> westCounter
if westInputVehicle then westCounter++;
if westOutgoingVehicle then westCounter--;
A5 : changeSignal, LastExecution, northCounter, southCounter, eastCounter, westCounter -> LastExecution, countPrimaryRoad, direction, countSecondaryRoad
if changeSignal?
then
{
LastExecution = (changeSignal="NS")?"EW":"NS";
if LastExecution=="NS"
{
if eastCounter + westCounter > 0
then
{ countPrimaryRoad ! eastCounter + westCounter; direction ! EW ;countSecondaryRoad ! northCounter + southCounter; LastExecution := EW}
else
{ countPrimaryRoad ! northCounter + southCounter; direction ! NS;countSecondaryRoad ! eastCounter + westCounter; LastExecution := NS
}
}
else
{
if northCounter + southCounter > 0
then
{countPrimaryRoad ! northCounter + southCounter; direction ! NS;countSecondaryRoad ! eastCounter + westCounter; LastExecution := NS}
else
{countPrimaryRoad ! eastCounter + westCounter; direction ! EW; countSecondaryRoad ! northCounter + southCounter; LastExecution := EW}
}
}
event(String) changeSignal
event(int) countPrimaryRoad
event(int) countSecondaryRoad
event(String) direction
event second
A6 : countPrimaryRoad, countSecondaryRoad, direction, saturationNS, saturationEW -> criticalFlowNS, criticalFlowEW
if countPrimaryRoad ? ^ countSecondaryRoad ? ^ direction ?
then
{
if direction = "NS"
then
{
criticalFlowNS ! saturationNS / countPrimaryRoad;
criticalFlowEW ! saturationEW / countSecondaryRoad;
}
else{
criticalFlowEW ! saturationEW / countPrimaryRoad,
criticalFlowNS ! saturationNS / countSecondaryRoad;
}
}
int saturationNS
int saturationEW
event(real) criticalFlowEW
event(real) criticalFlowNS
A7 : criticalFlowNS, direction, criticalFlowEW, countPhase, LostTime, AllRed, L, ya, OCL -> ya, L, OCL, greenTime
if criticalFlowNS? ^ direction? ^ criticalFlowEW?
then
{
L = (countPhase * LostTime) + AllRed;
OCL := (1.5 * L + 5) / (1 - criticalFlowNS +criticalFlowEW);
if direction = "NS"
then
ya := criticalFlowNS
else
ya := criticalFlowEW;
greenTime ! ROUND((ya/(criticalFlowNS + criticalFlowEW)) * (OCL - L));
}
event(int) greenTime
A9 : second, greenTimecounter -> greenTimecounter
A8 : greenTime -> greenTimecounter
if greenTime?
then
{
greenTimecounter := greenTime
}
if second? ^ greenTimecounter > 0
then
greenTimecounter--;
A10 : greenTimecounter -> changeSignalWithoutEmergency
if greenTimecounter = 0
then
changeSignalWithoutEmergency !
event changeSignalWithoutEmergency
if isEmergency
then
{
if second? ^ counter > 0
then
{
counter--;
if dir = "NS"
then
NSgsLatch := True;
EWrsLatch := True
else
EWgsLatch := True;
NSrsLatch := True;
}
else
{
isEmergency := False;
changeSignal ! dir="NS"?"EW":"NS";
}
}
else if !changeSignalWithoutEmergency
then
{
if dir = "NS"
then
NSgsLatch := True
else
EWgsLatch :=True
}
else if changeSignalWithoutEmergency
then
{
changeSignal !changeSignalWithoutEmergency
dir :=direction
NSysLatch := True
EWysLatch := True
}
A11 : emergency -> emergency, counter, dir, isEmergency,
if emergency?
then
{
counter := 20;
dir := emergency
isEmergency := True;
emergency := !emergency
}
A12 : isEmergency, second, counter, changeSignalWithoutEmergency, direction -> counter, dir, NSgsLatch, EWrsLatch, EWgsLatch, NSrsLatch, isEmergency, NSysLatch, EWysLatch
A13 : northEmergency, southEmergency, eastEmergency, westEmergency -> emergency
if northEmergency?
then
emergency ! "North"
else if southEmergency?
then
emergency ! "South"
else if eastEmergency?
then
emergency ! "East"
else if westEmergency?
then
emergency ! "West"
event(bool) northEmergency
event(bool) southEmergency
event(bool) eastEmergency
event(bool) westEmergency
bool NSGreenSignal = NSgsLatch
bool NSYellowSignal = NSysLatch
bool NSRedSignal = NSrsLatch
bool EWGreenSignal = EWgsLatch
bool EWYellowSignal = EWysLatch
bool EWRedSignal = EWrsLatch

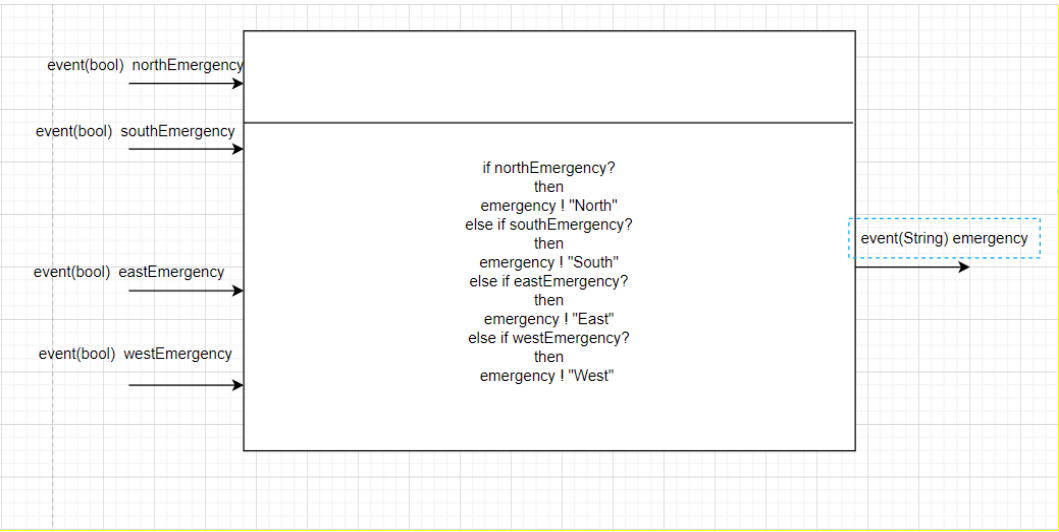
Emergency Vehicle Polling:

Emergency Vehicle Polling component takes the input values from Emergency Vehicle RFID Receiver to detect presence of emergency vehicles. Each input of a component is referred to by the direction of the sensor from which the component is getting information. All the inputs are of Boolean event type. If any of the input events occurs, the component will send the output as String with the value of the direction of the emergency vehicle.

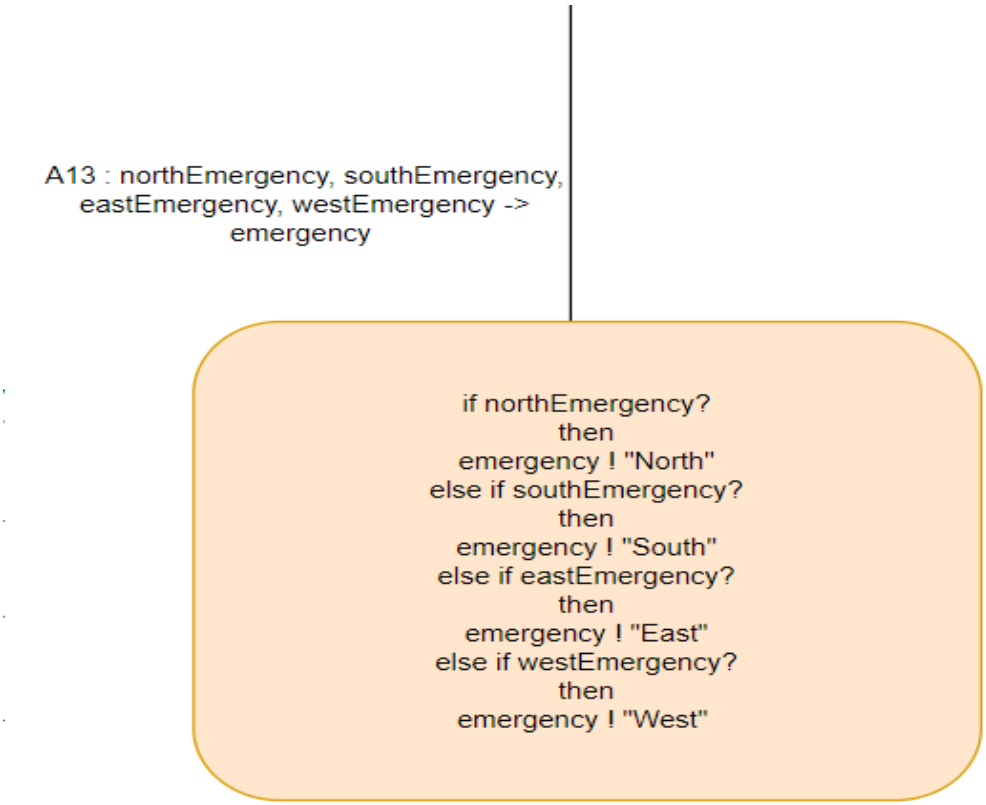
Below is the {I,*O,S,Init,React}* for Emergency Vehicle Polling:

- Input I: {northEmergency, southEmergency, eastEmergency, westEmergency}
  northEmergency, southEmergency, eastEmergency, westEmergency are of event type boolean.
- Output O: {emergency}, emergency is of event type String.
- State variables S: {}
- Init= {}. As there are no state variables, no initialization is required.
- React: Refer SRC components below.

SRC Component Diagram:



SRC Task Graph Diagram:



Vehicle Counter:

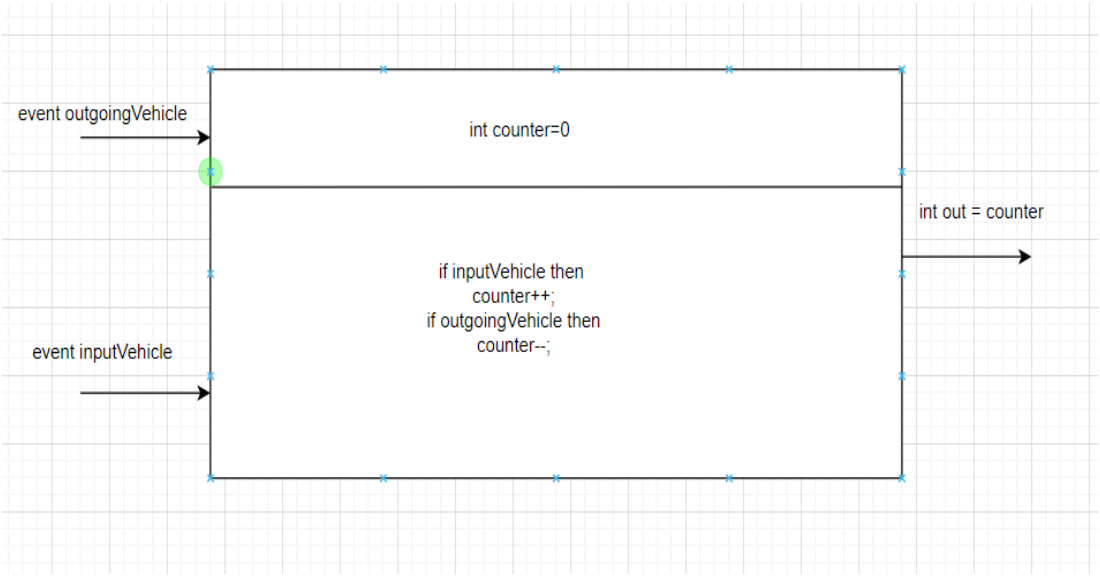
The Vehicle Counter component takes the input values from Inductive Loop Detectors that detect which determine the incoming and outgoing vehicles. Two sensors are placed on the road. One sensor is placed at the junction starting point of each road and another at a 200m distance on each road. The component takes the event from Inductive Loop Detectors and maintains the count of the vehicles present in the distance frame. This is done through maintaining the state variable. It returns the latched output out with the count of vehicles. There is a Vehicle Counter for each approaching road.

Below is the {I,O,S,Init,React} for Vehicle Counter Component:

- Input I: {inputVehicle, outgoingVehicle} inputVehicle, outgoingVehicle are of event type.
- Output O: {out}, out is of type int.

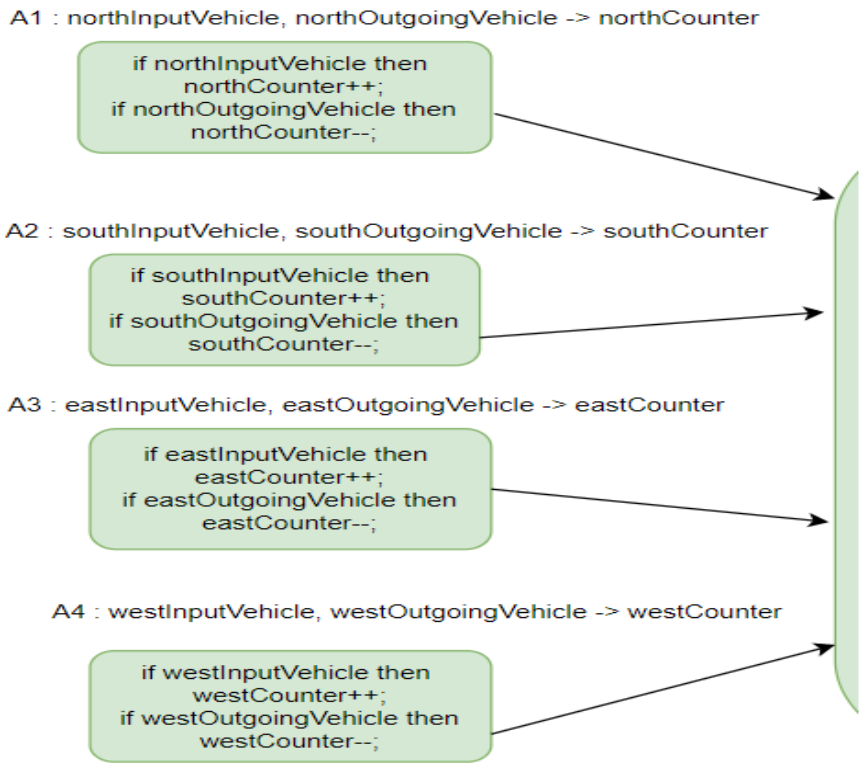
- State variables S: {counter}, counter is of type int.
- Init= {counter=0}
- React: Refer SRC components below.

SRC Component Diagram:



Since there are four sensors installed, one for each direction, we have utilized four vehicle counter components. These components determine the net vehicle count at the intersection and then forward that information to the direction SRC for further processing.

SRC Task Graph Diagram:



Determine Direction:

The Determine Direction component serves two main purposes. It helps in toggling the signal from North-South approach road to East-West Approach Road and vice versa in every round. It is also responsible for determining the traffic or number of vehicles in each approach road which is used to calculate or determine the approach road to be prioritized for green signal.

In addition, this component accounts for the changeSignal event, which serves as a barrier to subsequent rounds until the previous green signal phase is completed. The feedback functionality of a CPS is demonstrated by this component.

Below is the {I,O,S,Init,React} for Determine Direction:

- Input I: {north, south, east, west, changeSignal}
  north, south, east, west is of integer type. This input contains the number of vehicles in each direction's approach road. changeSignal is a type of event.

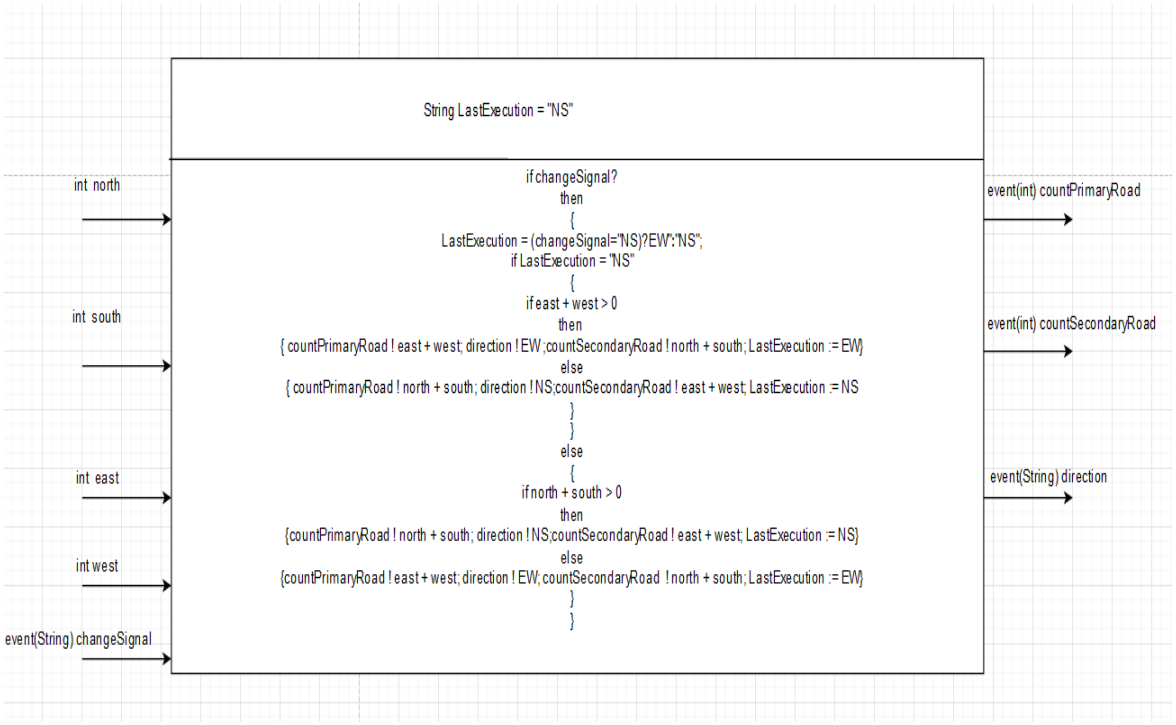
- Output O: {countPrimaryRoad, countSecondaryRoad, direction}. countPrimaryRoad and countSecondaryRoad are of type event(int) and direction is of type event (String).

- State variables S: {LastExecution}. LastExecution is of String type.

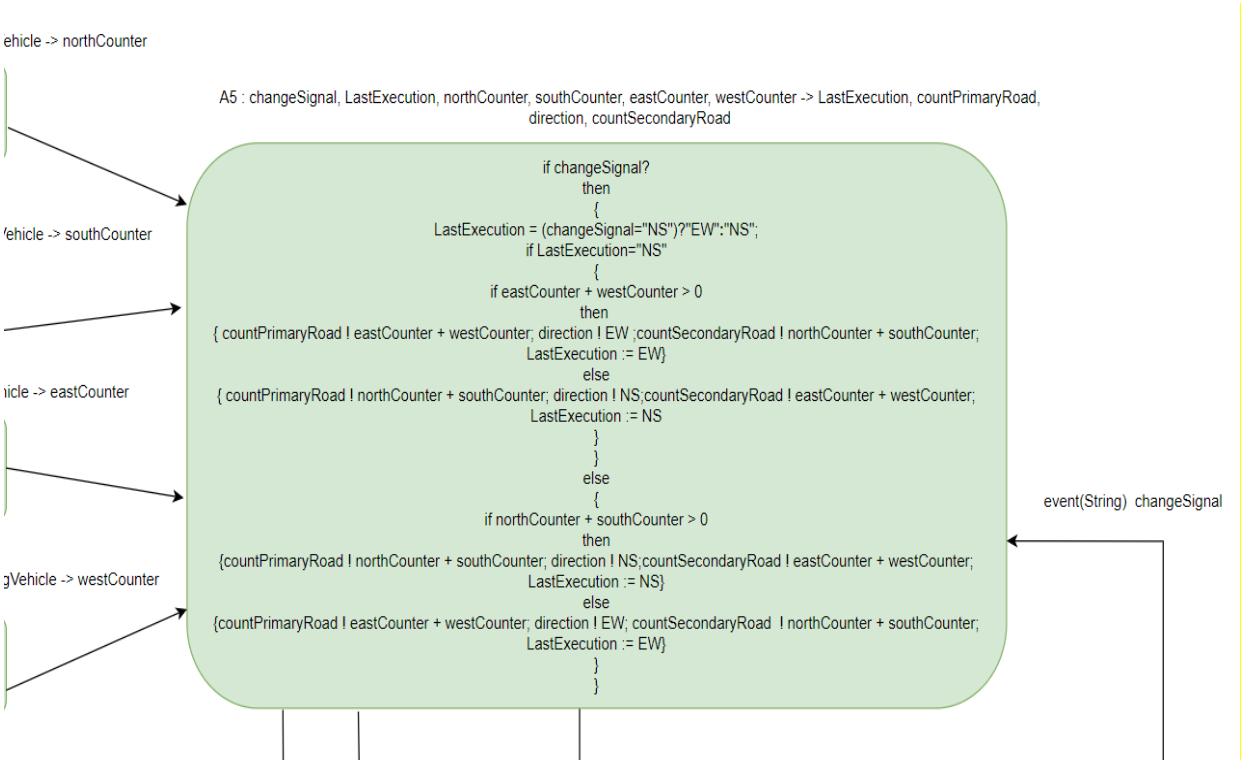
- Init= {LastExecution := “NS”}. We assume that component states initialize with a North-South approach road having a green signal.

- React = Refer SRC components below.

SRC Component Diagram:



SRC Task Graph Diagram:



Critical Flow:

As part of the Webster algorithm process, this component plays a crucial role by providing essential input required for calculations. The Critical Flow calculation, which is necessary for determining the optimal green time for an approach road, is performed using the following formula:

Critical flow ratio at ith phase = observed volume / saturation flow

Below is the {I,O,S,Init,React} for Critical Flow:

- Input I: {countPrimaryRoad, countSecondaryRoad, direction, saturationNS, saturationEW} saturationNS & saturationEW are constants provided externally to the components and are of type integer. This is unique for each intersection. The directions are of event (String) & countPrimaryRoad and countSecondaryRoad are of type event(int).

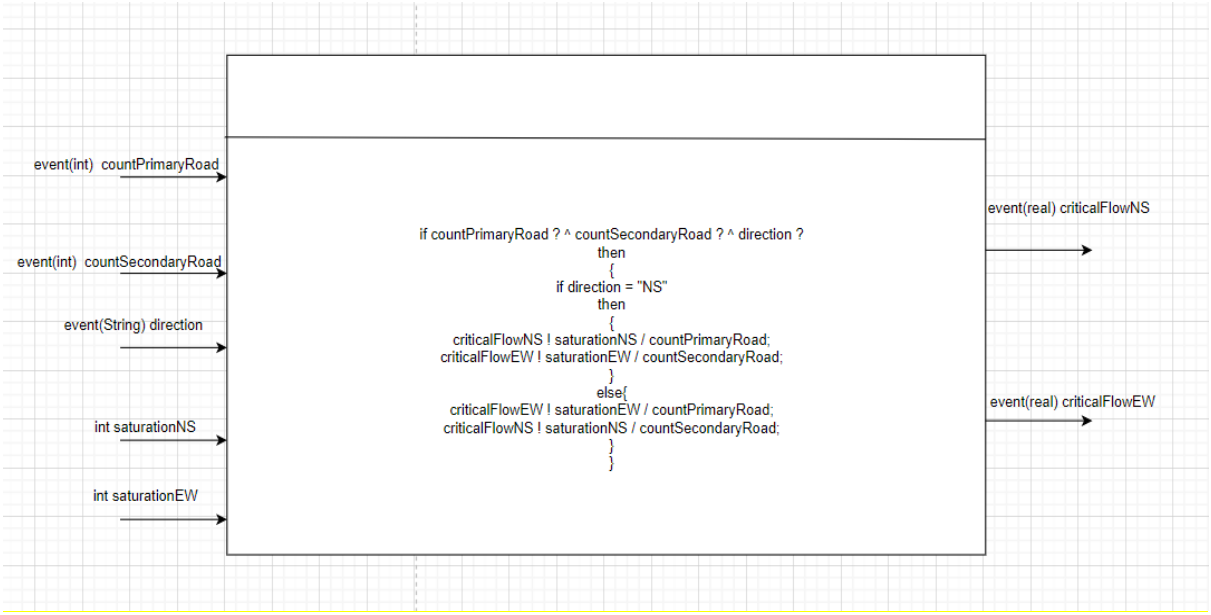
- Output O: {criticalFlowNS, criticalFlowEW}. criticalFlowNS and criticalFlowEW are of type event(real). As the calculation includes division, it's best to have a real data type for accurate calculations.

- State variables S: {}

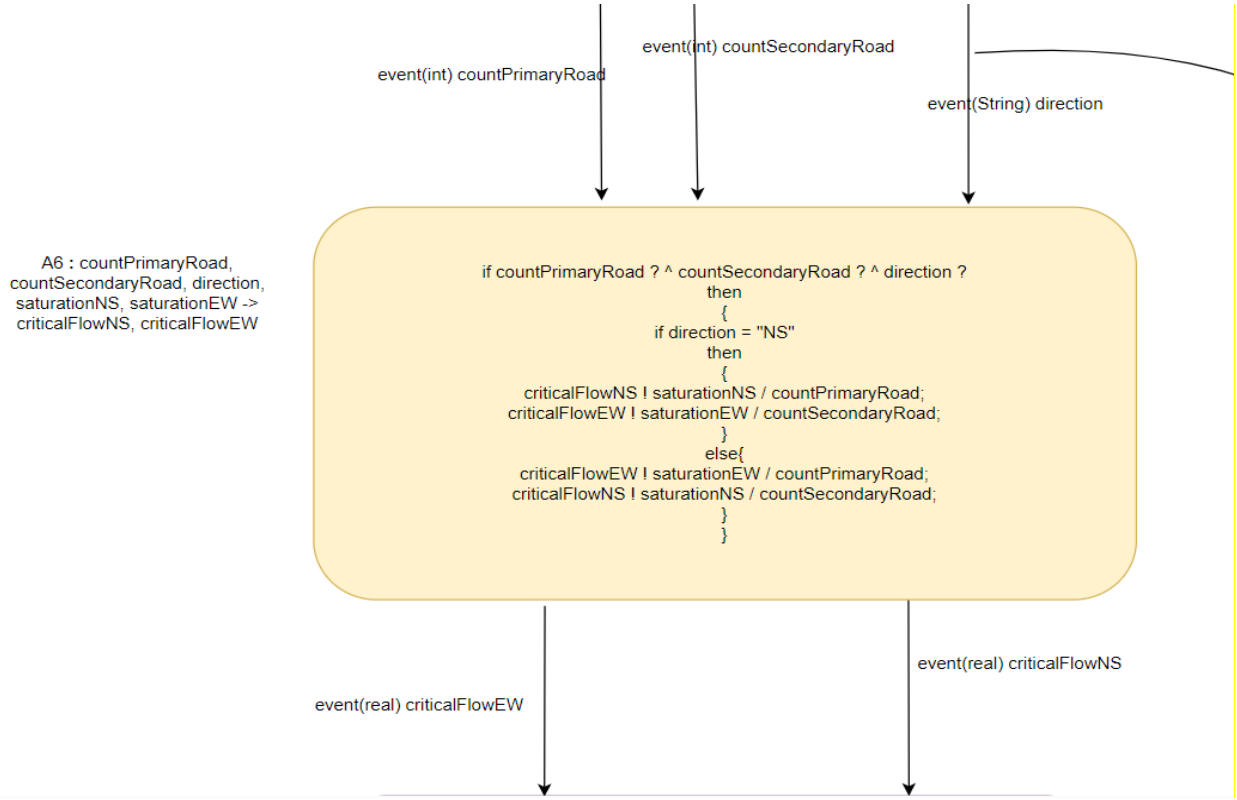
- Init= {}. As there are no state variables, initialization is not required.

- React = Refer SRC components below.

SRC Component Diagram:



SRC Task Graph Diagram:



Webster's Algorithm:

The Webster's Algorithm component implements webster to calculate rational approach for designing traffic signals. It is simple and is based on the formulae given by Webster. It is a calculation needed to determine the optimum green time for an approach road and it's given by below formula:

*Ga = (ya/y) * (Co - L)*

- ya - critical flow ratio for road 'a'
- y - summation of all critical flow ratio
- Co - Optimum cycle length
- L - lost time including all red time

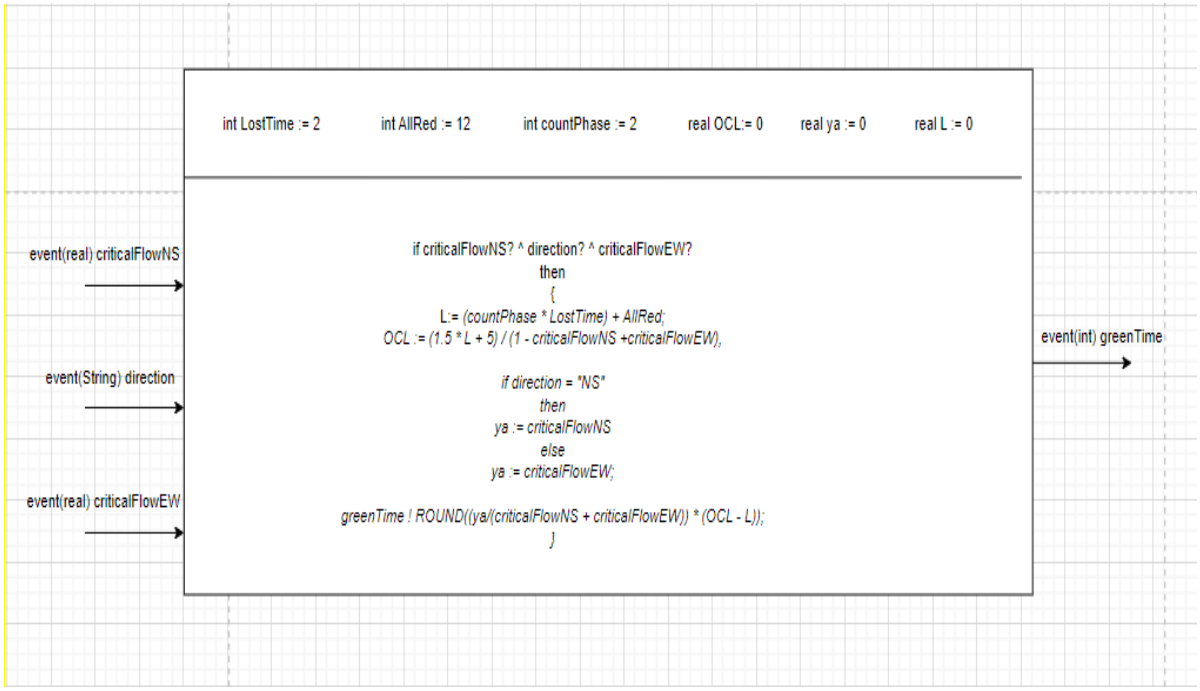
Assumption:

- LostTime at a phase is assumed to be 2 seconds.
- All Red time is assumed to be 12 seconds.
- Phase Count is assumed to be 2.

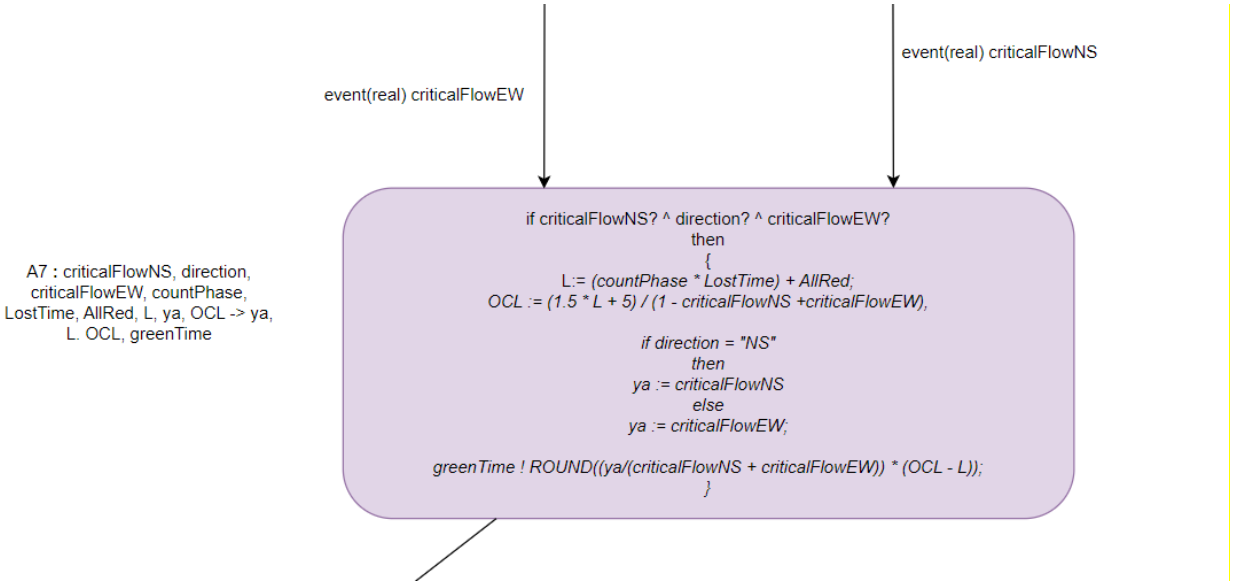
Below is the *I,O,S,Init,React}* for Webster Component:

- Input I: {criticalFlowNS, direction, criticalFlowEW}
  direction is of event type String, criticalFlowNS, criticalFlowEW are of event type real.
- Output O: {greenTime}, greenTime is of event type int.
- State variables S: {LostTime, AllRed, countPhase, OCL, ya}, LostTime, AllRed, countPhase is of type int, OCL, ya is of type real.
- Init= {LostTime=2, AllRed=12, countPhase=2, OCL=0, ya=0}
- React = Refer SRC components below.

SRC Component Diagram:



SRC Task Graph Diagram:



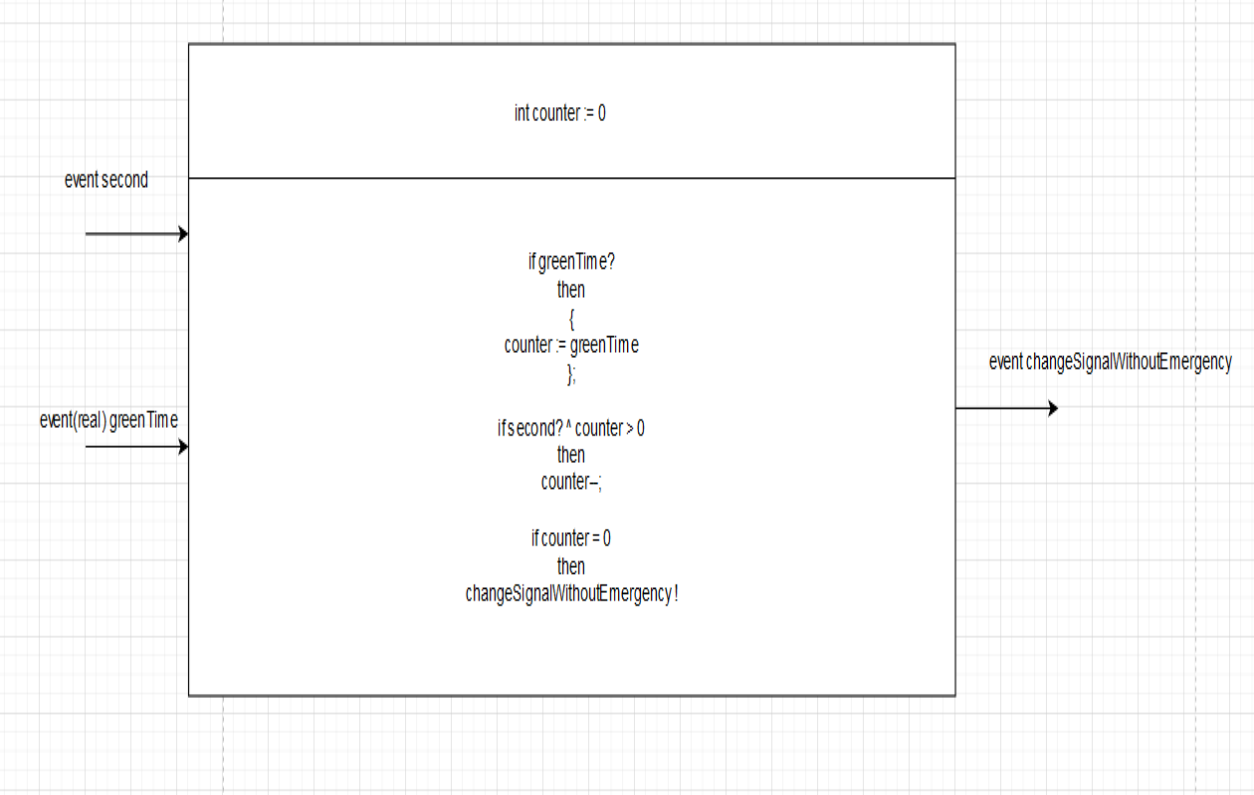
Data Processing Controller:

The Data Processing Controller component takes the input value greenTime which indicates the number of seconds for the signal to be kept green, and 'second' input which acts as the timer for seconds. When a greenTime event occurs, the value is assigned to the state variable. The state variable counter is decreased each time when the event second occurs acting as a timer. It sends the output event changeSignal whenever the counter becomes 0 by indicating that the timer has been completed for the lane.

Below is the {*I,O,S,Init,React}* for Data Processing Component:

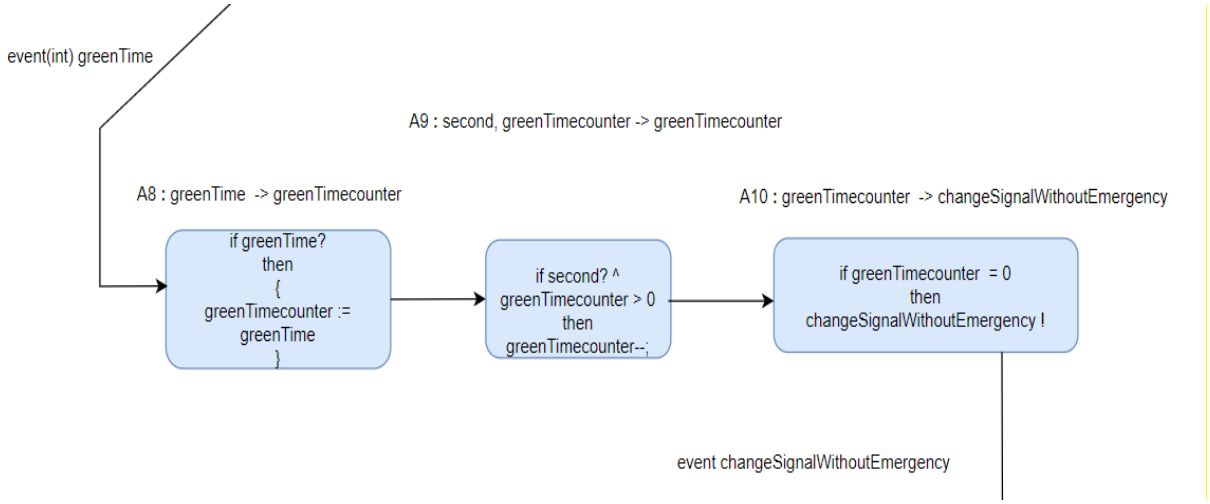
- Input I: {second, greenTime}
  second is of type event, greenTime is of event type real.
- Output O: {changeSignal}, changeSignal is of event type.
- State variables S: {counter}, counter is of type int.
- Init= {counter=0}
- React: Refer SRC components below.

SRC Component Diagram:



SRC Task Graph Diagram:

Several modifications were made to integrate this component into the composite SRC. Since the "counter" state variable was already assigned as a keyword in another task, we had to rename it to "greenTimeCounter." This adjustment preserves the cohesion necessary in the composite task graph and ensures that the state variables remain in separate sets (disjoint set property), as required.



Light Signal Controller:

The Light Signal Controller component is responsible for toggling traffic light signals based on calculation of optimum green time and emergency vehicle arrival event. If there is an emergency vehicle on any of the approach roads, the green signal for that approach will be on for 20 seconds. At the same time a red signal will be on for another approach road. We have allocated *20 seconds* to pass an emergency vehicle through an intersection.

If there is no emergency vehicle, it will check the normal traffic toggle based on optimum green time calculations from the Data Processing Controller component above.

Below is the {*I,O,S,Init,React}* for Light Signal Controller:

- Input I: {second, changeSignal, direction, emergency}
  Second, changeSignal, direction, emergency is of event type.

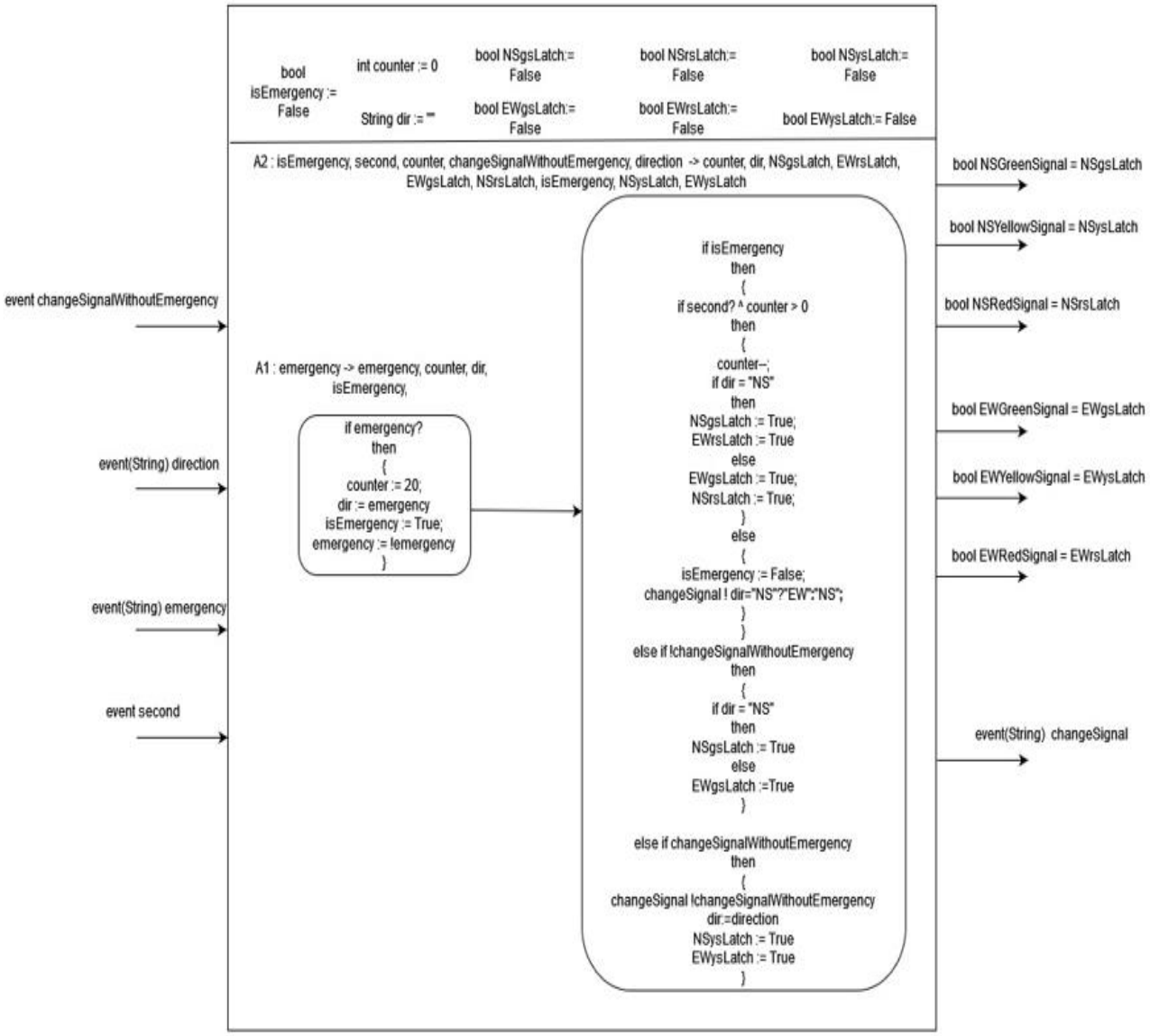
- Output O: {NSGreenSignal, NSYellowSignal, NSRedSignal, EWGreenSignal, EWYellowSignal, EWRedSignal}. All the outputs are boolean type and are latched to state variables.

- State variables S: {isEmergency, counter, dir, NSgsLatch, NSysLatch, NSrsLatch, EWgsLatch, EWrsLatch, EWysLatch}. NSgsLatch, NSysLatch, NSrsLatch, EWgsLatch, EWrsLatch, EWysLatch & isEmergency variable are of type boolean. counter is of type integer and dir shows the direction of the flow and is String in type.

- Init= {isEmergency: = 0, int counter:= 0, bool NSgsLatch:= False, bool NSrsLatch:= False, bool NSysLatch:= False, String dir := "", bool EWgsLatch:= False, bool EWrsLatch:= False, bool EWysLatch:= False}

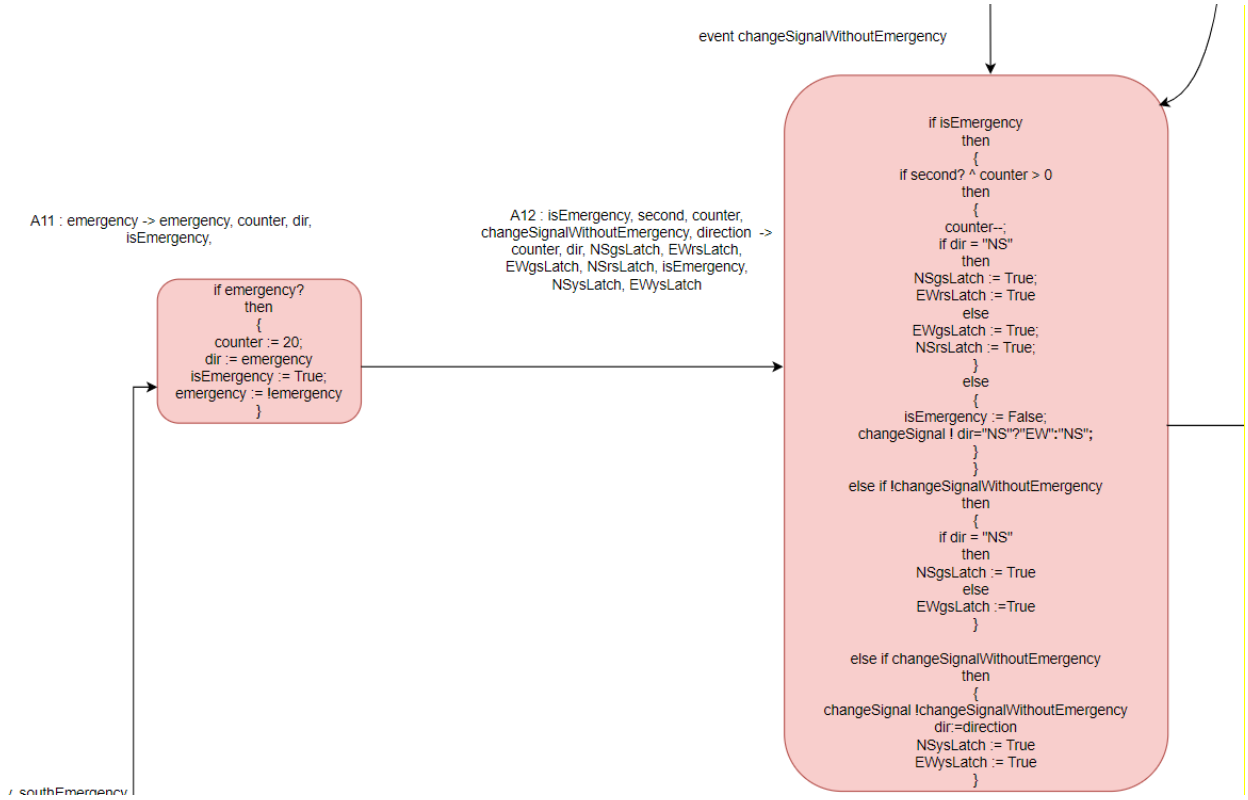
- React = Refer SRC components below.

SRC Component Diagram:



SRC Task Graph Diagram:

Components are divided into two task graphs with one handling the emergency and toggling the emergency flag and initiating the counter. Another task A12 is responsible for toggling the light signals for duration of time decided by previous webster algorithm or 20 seconds if its an emergency vehicle.



2.2. UML design specifications with descriptions

It provides UML design specifications including class diagrams with descriptions describing design choices and tradeoffs.

The UML diagram includes several classes that are integral to the traffic control system.

- The Application class serves as the main class for the system, while the Constants class provides a central location for storing important system constants.
- The CriticalFlow class is responsible for calculating the critical flow for each approach road, which is necessary for determining the optimum green time for each signal phase.
- The DataProcessingController class acts as a mediator between the different system components and handles data processing tasks. The DetermineDirection class determines the direction of each vehicle, while the EmergencyVehiclePolling class polls for signals from emergency vehicles.
- The InputData class handles input data from various sensors and sources, while the LightSignalController class controls the traffic lights at the intersection.
- The Node class represents each intersection node, while the ParseData class parses data from various sources.
- The system also includes classes for vehicle counting, including VehicleCounterEast, VehicleCounterNorth, VehicleCounterSouth, and VehicleCounterWest, which count vehicles for each approach road. Finally, the Webster class implements the Webster algorithm for determining the optimum green time for each signal phase.

Together, these classes form a comprehensive traffic control system capable of handling various traffic scenarios.
Each class is explained in detail further in the report.

Relationships

Most of the classes in our design exhibit an aggregation relationship, whereby they are one or more instances of another class. This type of relationship is characterized as a part-whole relationship, where the part class forms an integral component of the whole class. Although instances of the part class can exist independently of the whole class, their life cycle is closely intertwined with that of the whole class. As a result, the diamond shape on the association line at the whole class end is frequently used to represent this type of relationship in our classes.

Multiplicity

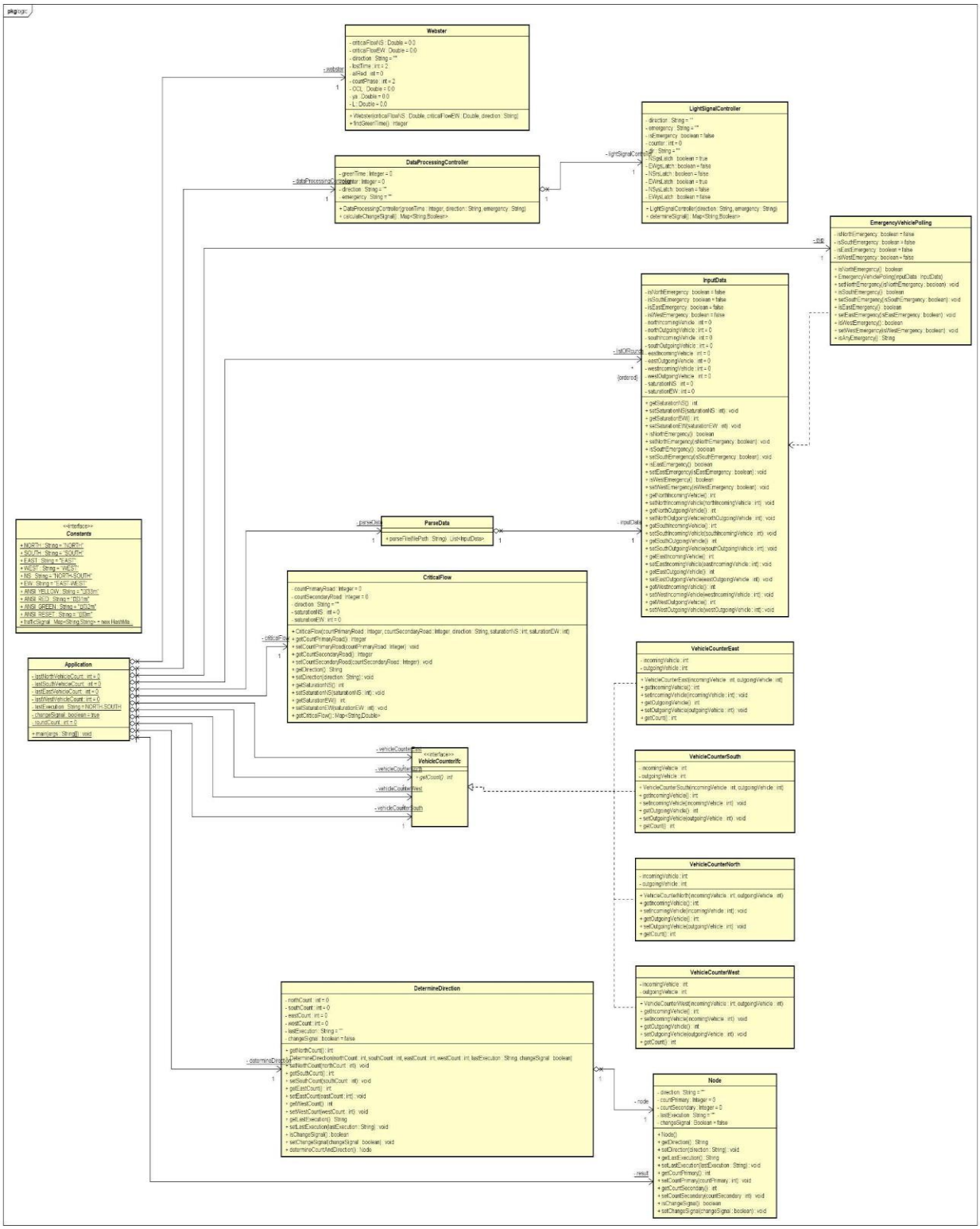
- The Application class has a one-to-many multiplicity relationship with instances of the InputData class, which serves to illustrate the round behavior. Each instance of InputData corresponds to a single round and contains all of the necessary data for

processing, such as the number of vehicles from each direction, emergency vehicle data, and constant values used to calculate green time.
- All other classes are having one to one multiplicity.

Classes & Interfaces

Our UML class diagram consists of 15 classes and 1 interface.

Comprehensive UML:



Reading Vehicle Data from Sensors:

The given class diagram is a subset that focuses on the feature of retrieving vehicle data from sensors. To achieve this, the data is obtained from a text file. The process involves invoking the "read" function from the Application class, retrieving the file from the resource folder, and parsing it into a format that can be easily read by the Application class.

InputData.txt

The following is the input data file that the Application class reads. It contains multiple lines, with each line representing data for a single round. Each line is composed of the following elements:

Sample inputData:

NorthIncomingVehicle	NorthOutgoingVehicle	SouthIncomingVehicle	SouthOutgoingVehicle	EastIncomingVehicle
10	0	10	0	20

EastOutgoingVehicle	WestIncomingVehicle	WestOutgoingVehicle	SaturationNS	SaturationEW
0	20	0	1000	1250

NorthEmergency	SouthEmergency	EastEmergency	WestEmergency
FALSE	FALSE	FALSE	FALSE

The Application class and the ParseData class have an aggregate relationship, where the latter fetches data from the inputData.txt file and creates a list of inputData objects. These objects correspond to the rows in the inputData.txt file.

The Facade Design Pattern was utilized to establish a single point of contact (i.e., the Application) for making method calls to various parts of the program.

Below is the detailed explanation of class, its attributes, and methods:

Class definition

Name	Description
Application	The Application class serves as the entry point for the program and is responsible for capturing the input data, which represents the data for a single round. This data is subsequently passed to multiple instances of other classes for processing, which involve determining the direction and green time for each direction.

ParseData	The class in question is a utility class designed to parse the data retrieved from the inputData.txt file and convert it into a format that can be accepted by the inputData.java entity.
InputData	The InputData class serves as an entity class for the input data. It is composed of fields and variables that correspond to all of the inputs required by the application.

Attribute definitions (Application Class)

Name	Description
lastNorthVehicleCount	The attribute in question is of the Integer data type and is declared as private. It is used for capturing the count of vehicles traveling from the North direction. Upon initialization, the attribute is set to 0 at the beginning.
lastSouthVehicleCount	The attribute in question is of the Integer data type and is declared as private. It is used for capturing the count of vehicles traveling from the South direction. Upon initialization, the attribute is set to 0 at the beginning.
lastEastVehicleCount	The attribute in question is of the Integer data type and is declared as private. It is used for capturing the count of vehicles traveling from the East direction. Upon initialization, the attribute is set to 0 at the beginning.
lastWestVehicleCount	The attribute in question is of the Integer data type and is declared as private. It is used for capturing the count of vehicles traveling from the West direction. Upon initialization, the attribute is set to 0 at the beginning.

lastExecution	The attribute in question is of the String data type and is used to determine the direction that was processed and given green time in the previous execution. This information is utilized to determine the direction to which green time should be allocated in the current round. The attribute is initialized to "NS" since the assumption is made that the North-South direction will have green time allocated initially.
changeSignal	The attribute in question is of the boolean data type and is used to hold on a round until the green time for the previous round is fully executed. It is declared as private and is initialized to true during the initialization time.
roundCount	The variable in question is a utility variable used to hold the count of rounds that have been executed so far. Its purpose within the program is to facilitate printing to the console, which is utilized in experiments and demos. It is initialized to 0 at the start.

Attribute definitions (ParseData Class)

Parse Data does not have any attributes.

Attribute definitions (InputData Class)

isNorthEmergency	The attribute in is of the boolean data type and is used to capture emergency Vehicles from North direction. It is declared as private and is initialized to false.
isSouthEmergency	The attribute in is of the boolean data type and is used to capture emergency Vehicles from South direction. It is declared as private and is initialized to false.

isEastEmergency	The attribute in is of the boolean data type and is used to capture emergency Vehicles from East direction. It is declared as private and is initialized to false.
isWestEmergency	The attribute in is of the boolean data type and is used to capture emergency Vehicles from West direction. It is declared as private and is initialized tofalse.
northIncomingVehicle	The attribute in is of the integer data type and is used to capture incoming vehicles from the north. It is declared as private and is initialized to 0.
northOutgoingVehicle	The attribute in is of the integer data type and is used to capture outgoing vehicles from north. It is declared as private and is initialized to 0.
southIncomingVehicle	The attribute in is of the integer data type and is used to capture incoming vehicles from the south. It is declared as private and is initialized to 0.
southOutgoingVehicle	The attribute in is of the integer data type and is used to capture outgoing vehicles from the south. It is declared as private and is initialized to 0.

eastIncomingVehicle	The attribute in is of the integer data type and is used to capture incoming vehicles from the east. It is declared as private and is initialized to 0.
eastOutgoingVehicle	The attribute in is of the integer data type and is used to capture outgoing vehicles from the east. It is declared as private and is initialized to 0.
westIncomingVehicle	The attribute in is of the integer data type and is used to capture incoming vehicles from west. It is declared as private and is initialized to 0.
westOutgoingVehicle	The attribute in is of the integer data type and is used to capture outgoing vehicles from west. It is declared as private and is initialized to 0.
saturationNS	The attribute in is of the integer data type and is used to capture saturation value of North South Road. This value will be used in Webster calculations.It is declared as private and is initialized to 0.
saturationEW	The attribute in is of the integer data type and is used to capture saturation value of East West Road. This value will be used in Webster calculations.It is declared as private and is initialized to 0.

Method definitions (Application Class)

Name	Description
main	This is a main method of the program which kicks off the program execution.

Method definitions (Parse Data)

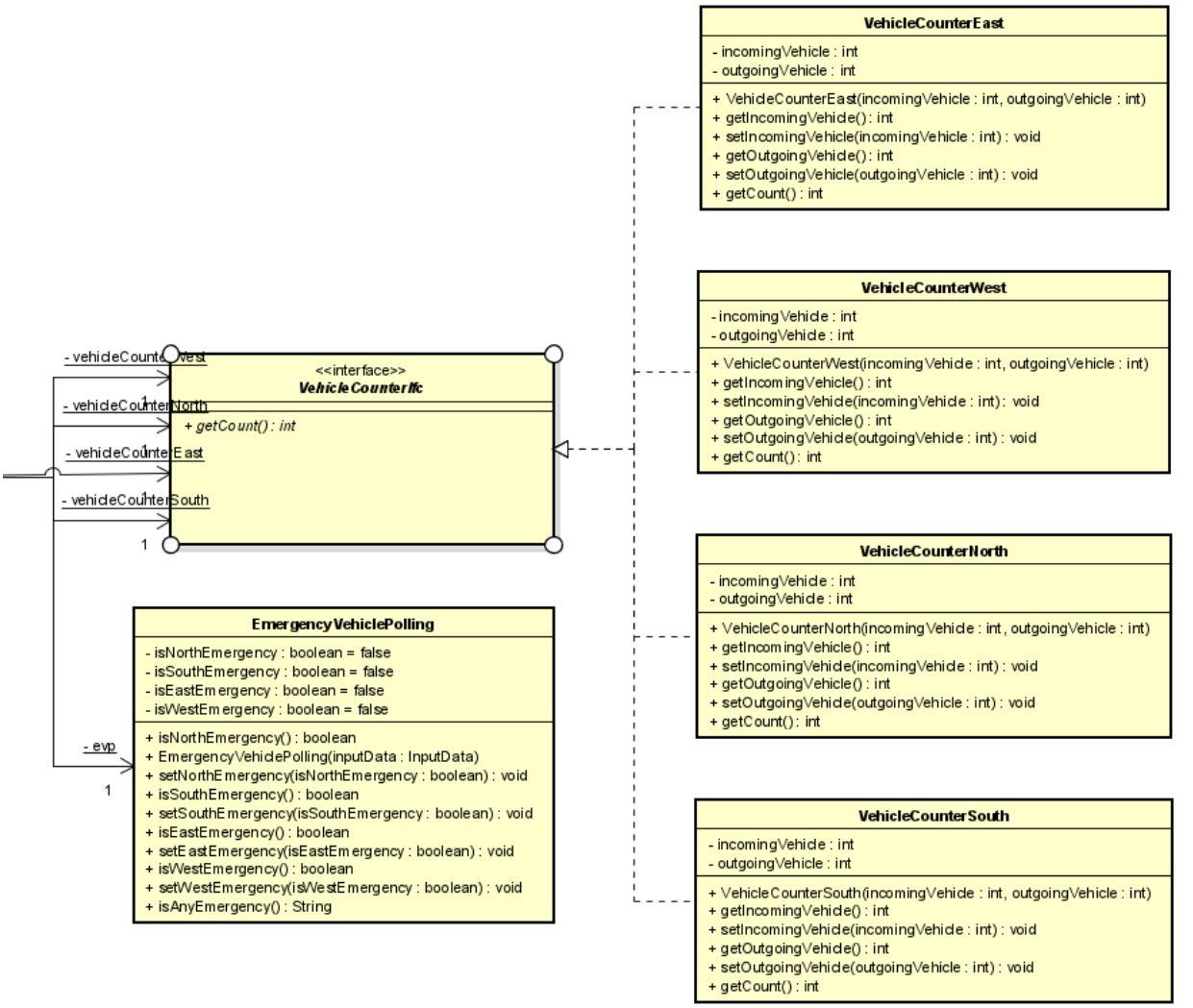
Name	Description
parseFile	This is a private method called Application class. With List<InputData> return type, this allows fetching data from inputData.txt file and preparing inputData objects corresponding to each row.

Method definitions (InputData Class)

This class contains methods in the form of getters and setters for all its attributes. This facilitates the fetching and setting of data into the inputData object.

Count of Vehicles/Emergency Vehicle at intersection:

The following UML diagram depicts the classes that count the number of vehicles arriving and departing at the interaction traffic lights. 'VehicleCounter' of each direction implements the 'VehicleCounterIfc' interface. Boolean input from the inductive loop detectors placed a specific distance from traffic light and on the traffic, light send their input to VehicleCounter. Based on these sensor data, the VehicleCounter for each direction sends the exact number of vehicles present at a round. EmergencyVehiclePolling is the class that detects the presence of emergency vehicles at the intersection. This class has attributes for inputs from sensors in the four directions; getters and setters are used to access these class attributes.



Interface Definition	
Name	**Description**
VehicleCounterIfc	Interface to be implemented by VehicleCounter class.
Method Definition	
getCount	Method to implemented by VehicleCounter classes

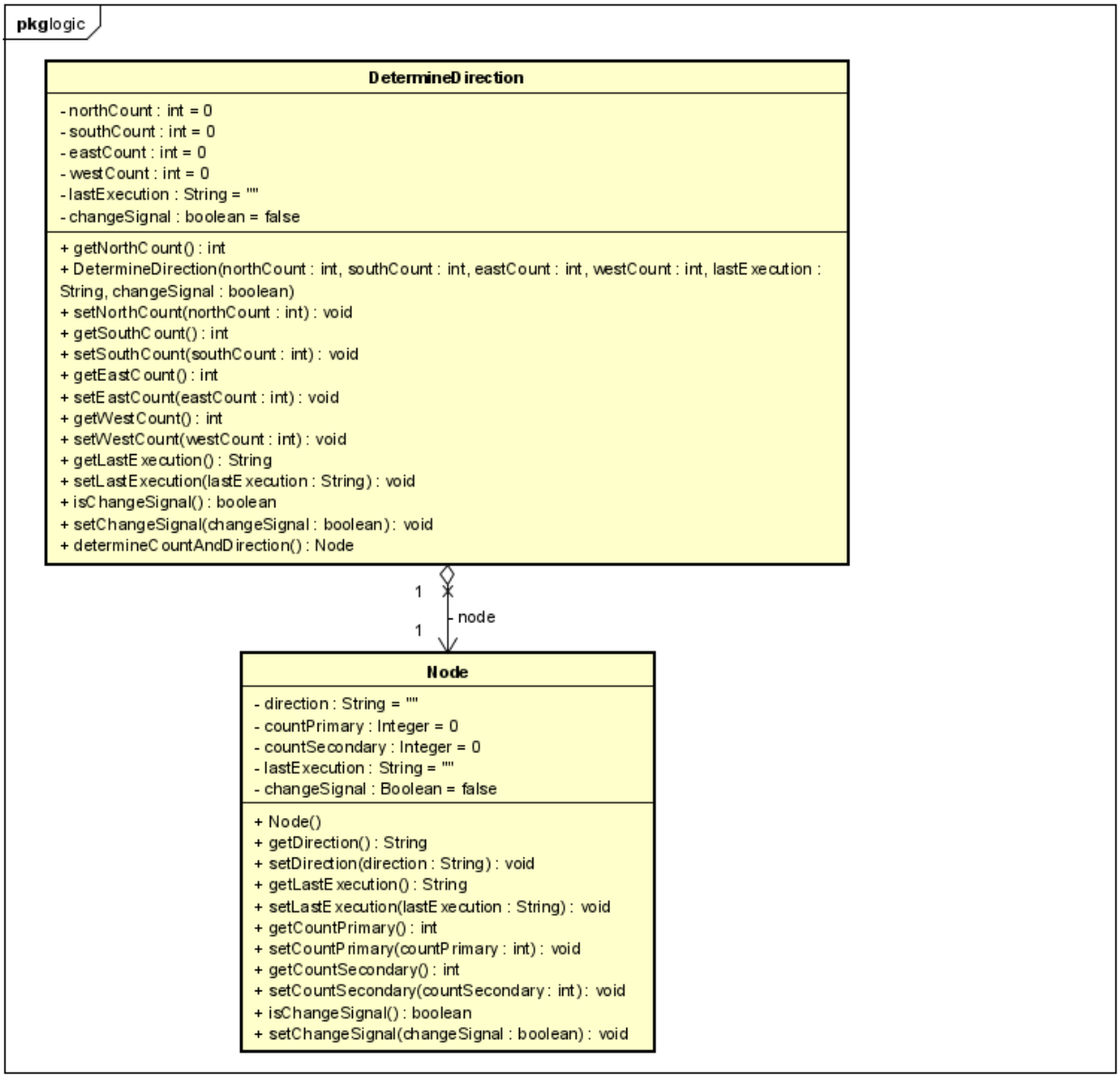
Class Definition	
Name	**Description**
VehicleCounter	The 'VehicleCounter' of each direction implements the ‘VehicleCounterIfc’ interface. This class has attributes for inputs from sensors in the four directions; getters and setters are used to access these class attributes.
Attribute Definition	
incomingVehicle	The attribute is of the integer data type and is used to capture the number of vehicles approaching the traffic light. This is marked as a private variable.
outgoingVehicle	The attribute is of the integer data type and is used to capture the number of vehicles departing the intersection. This is marked as a private variable.
Method Definition	
getCount	The method returns the number of vehicles that are waiting to cross the intersection. It returns an integer value (difference between incoming and outgoing vehicles).
VehicleCounter	Constructor to initialize the class. Takes the integer values number of incoming vehicle and outgoing vehicles are parameters.

Class Definition	
Name	**Description**
EmergencyVehiclePolling	Class that detects the presence of emergency vehicles at the intersection.
Attribute Definition	

isNorthEmergency	The attribute is of the boolean data type and is used to detect presence of emergency vehicles from the North direction. This is marked as a private variable.
isSouthEmergency	The attribute is of the boolean data type and is used to detect presence of emergency vehicles from the South direction. This is marked as a private variable.
isEastEmergency	The attribute is of the boolean data type and is used to detect presence of emergency vehicles from the East direction. This is marked as a private variable.
isWestEmergency	The attribute is of the boolean data type and is used to detect presence of emergency vehicles from the West direction. This is marked as a private variable.
Method Definition	
EmergencyVehiclePolling	Constructor to initialize the class. Takes object of InputData class as argument.
isAnyEmergency	Method checks if any of the boolean attributes are true, if true, it turns a String value of the direction. Return type is String.

Determine Direction:

The provided class diagram is a subset that concentrates on the capability of summing up the vehicles from all directions. Based on the last execution, the count of vehicles, and the changeSignal flag, it determines the next direction that should have a green time. The count of vehicles is the driving force to identify directions needing green time.


pkglogic
DetermineDirection
- northCount : int = 0
- southCount : int = 0
- eastCount : int = 0
- westCount : int = 0
- lastExecution : String = ""
- changeSignal : boolean = false
+ getNorthCount() : int
+ DetermineDirection(northCount : int, southCount : int, eastCount : int, westCount : int, lastExecution : String, changeSignal : boolean)
+ setNorthCount(northCount : int) : void
+ getSouthCount() : int
+ setSouthCount(southCount : int) : void
+ getEastCount() : int
+ setEastCount(eastCount : int) : void
+ getWestCount() : int
+ setWestCount(westCount : int) : void
+ getLastExecution() : String
+ setLastExecution(lastExecution : String) : void
+ isChangeSignal() : boolean
+ setChangeSignal(changeSignal : boolean) : void
+ determineCountAndDirection() : Node
1
1
- node
Node
- direction : String = ""
- countPrimary : Integer = 0
- countSecondary : Integer = 0
- lastExecution : String = ""
- changeSignal : Boolean = false
+ Node()
+ getDirection() : String
+ setDirection(direction : String) : void
+ getLastExecution() : String
+ setLastExecution(lastExecution : String) : void
+ getCountPrimary() : int
+ setCountPrimary(countPrimary : int) : void
+ getCountSecondary() : int
+ setCountSecondary(countSecondary : int) : void
+ isChangeSignal() : boolean
+ setChangeSignal(changeSignal : boolean) : void

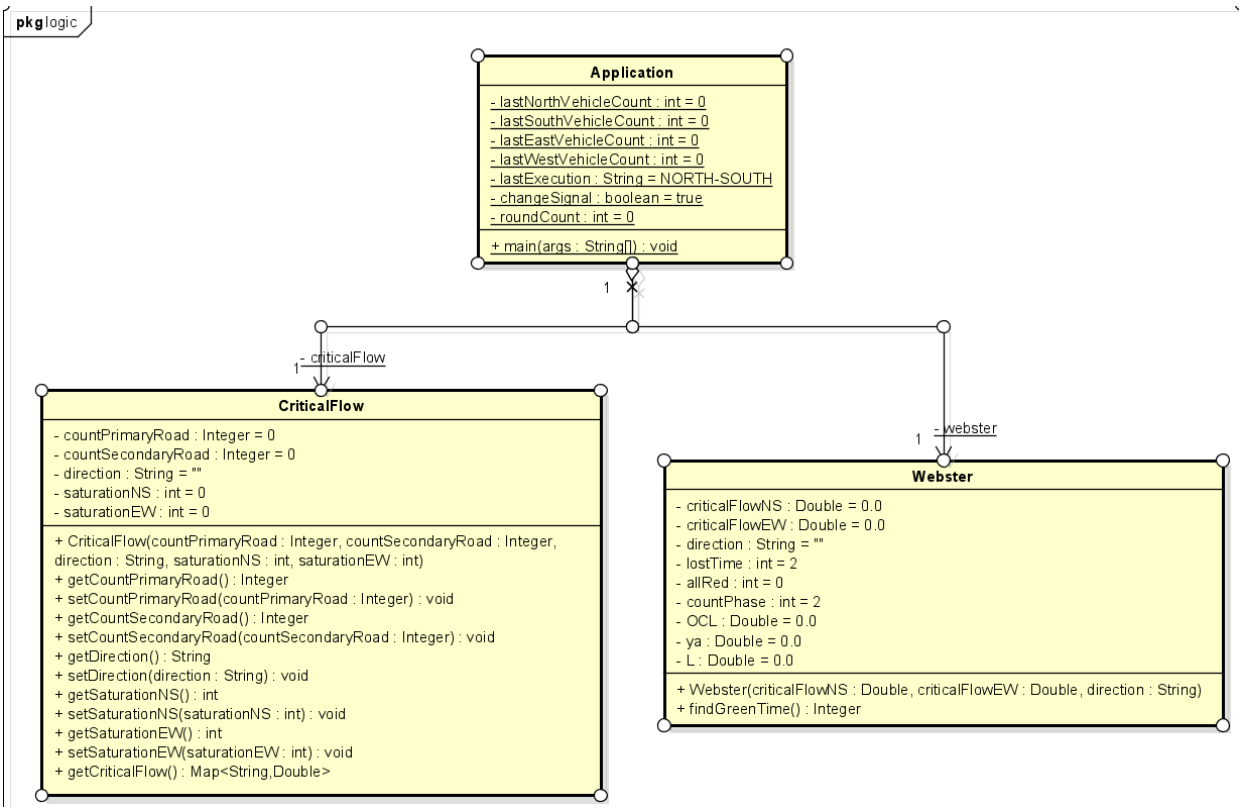
Class Definition	
Name	**Description**
DetermineDirection	The DetermineDirection class is responsible for determining the direction for the next green signal based on the count of vehicles in each direction and the last direction to have a green signal.
Attribute Definition	
northCount	The attribute is of the integer data type and is used to capture count of vehicles from the north. This is marked as a private variable.
southCount	The attribute is of the integer data type and is used to capture counts of vehicles from the south. This is marked as a private variable.
eastCount	The attribute is of the integer data type and is used to capture count of vehicles from the east. This is marked as a private variable.
westCount	The attribute is of the integer data type and is used to capture counts of vehicles from the west. This is marked as a private variable.
lastExecution	The lastExecution flag ensures that the side having lesser traffic too gets smaller green time in round robin fashion. This attribute is of the String data type and marked as private.
changeSignal	The changeSignal flag is a feedback flag which is used to hold on further rounds till the last determined greentime is elapsed.
Method Definition	
DetermineDirection(int northCount, int southCount, int eastCount, int westCount, String lastExecution, boolean changeSignal)	This is a constructor which assigns the value for all the variables when an object of this class is created.
determineCountAndDirection(): Node	This method, determineCountAndDirection(), which uses the instance variables to calculate the primary and secondary counts of vehicles and the direction in which the vehicle should proceed. The method creates a new Node object and sets its attributes based on the calculations.

getters and setters	Getters and Setters methods are utility methods which are used to set and fetch values to/from Node class.

Class Definition	
Name	**Description**
Node	The Node class is used as a container for all the parameters returned by an instance of DetermineDirection. This is considered good practice as it allows all the returned elements to be consolidated in a single instance before being returned.
Attribute Definition	
direction	The attribute is of the String data type and is used to capture the determined direction according to the round robin algorithm. This is marked as a private variable.
countPrimary	The attribute is of the integer data type and is used to capture counts of vehicles in primary roads. Primary road is one which will be going to get green time. This is marked as a private variable.
countSecondary	The attribute is of the integer data type and is used to capture counts of vehicles in secondary roads. Primary road is one which will be going to get green time. This is marked as a private variable.
lastExecution	The attribute is of the String data type and is used to capture the direction which got green time in the last iteration/round. This is marked as a private variable.
changeSignal	The changeSignal flag is a feedback flag which is used to hold on further rounds till the last determined greentime is elapsed.
Method Definition	
getters and setters	Getters and Setters methods are utility methods which are used to set and fetch values to/from Node class.

Webster Algorithm:

The below UML class diagram consists of the classes which calculate the green time required for a particular direction by performing Webster's algorithm for the current intersection. This algorithm uses the values like road dimensions, number of vehicles traveling from one direction per hour, ratio between the opposite direction vehicles etc.



Class Definition	
Name	**Description**
CriticalFlow	This class is used to fetch all the values required for the calculation of the webster's algorithm. It takes the values of the saturationFlow for all the directions and determines the criticalFlow for them. This class consists of the getters and setters which are used for manipulating the data.

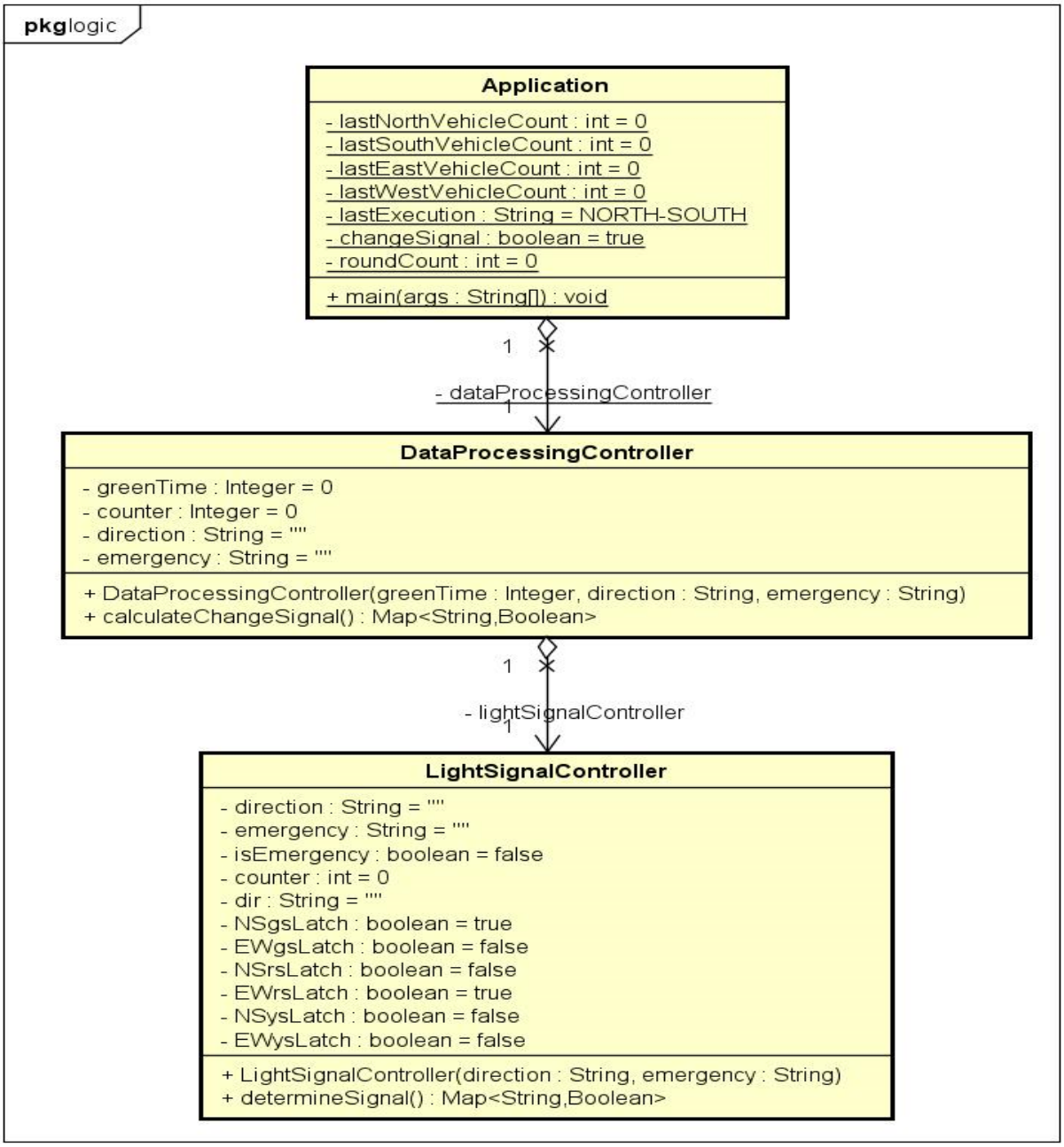
Attribute Definition	
countPrimaryRoad	This attribute has the integer value which represents the number of vehicles present in the direction which is going to be changed to green.
countSecondaryRoad	This attribute has the integer value which represents the number of vehicles present in the direction perpendicular to the direction which is going to be changed to green.
direction	This attribute has the string value which represents the direction which has to be changed. This helps in storage for determining direction.
saturationNS	This attribute has the integer value which represents the number of vehicles flowing in the north-south per hour.
saturationEW	This attribute has the integer value which represents the number of vehicles flowing in the east-west per hour.
Method Definition	
CriticalFlow(countPrimaryRoad : Integer, countSecondaryRoad, direction : String, saturationNS : Integer, saturationEW : Integer)	This is a constructor which assigns the value for all the variables when an object of this class is created.
getCriticalFlow(): Map<String, Double>	This method makes use of the vehicle count in all the directions, takes the flow of vehicles per hour in all the directions and determines the criticalFlow values for both the directions.

Class Definition	
Name	**Description**
Webster	This class is used to perform calculation of the webster's algorithm. This class determines the value of the greenTime by considering all the variables with respect to the intersection and flow of the traffic.
Attribute Definition	
criticalFlowNS	This attribute has the double value which represents the critical flow ratio at a phase is the ratio between the observed volume of flow to the

	saturation flow occurring at all the phases of an intersection at north south.
criticalFlowEW	This attribute has the double value which represents the critical flow ratio at a phase is the ratio between the observed volume of flow to the saturation flow occurring at all the phases of an intersection at east-west.
direction	This attribute has the string value which represents the direction which must be changed. This helps in storage for determining direction.
lostTime	This attribute has the integer value which represents lost time at a phase is usually taken as 2 seconds
allRed	This attribute has the integer value which represents all red time is usually taken as zero
OCL	This attribute is of double value which represents the optimum cycle length is also taken as the total cycle time for a signal system
ya	This attribute is of double value which represents the critical flow ratio for primary roads.
L	This attribute is of double value which represents lost time including all red time.
Method Definition	
Webster (criticalFlowNS : Double, criticalFlowEW : Double, direction : String)	This is a constructor which assigns the value for all the variables when an object of this class is created.
findGreenTime(): Integer	This method makes use of the vehicle count in all the directions, takes the flow of vehicles per hour in all the directions and determines the greenTime for the signal.

Changing Light Signal:

The below UML class diagram consists of the classes which take the greenTime and change the output of the traffic signal depending upon the emergency and direction which needs to be changed to green.



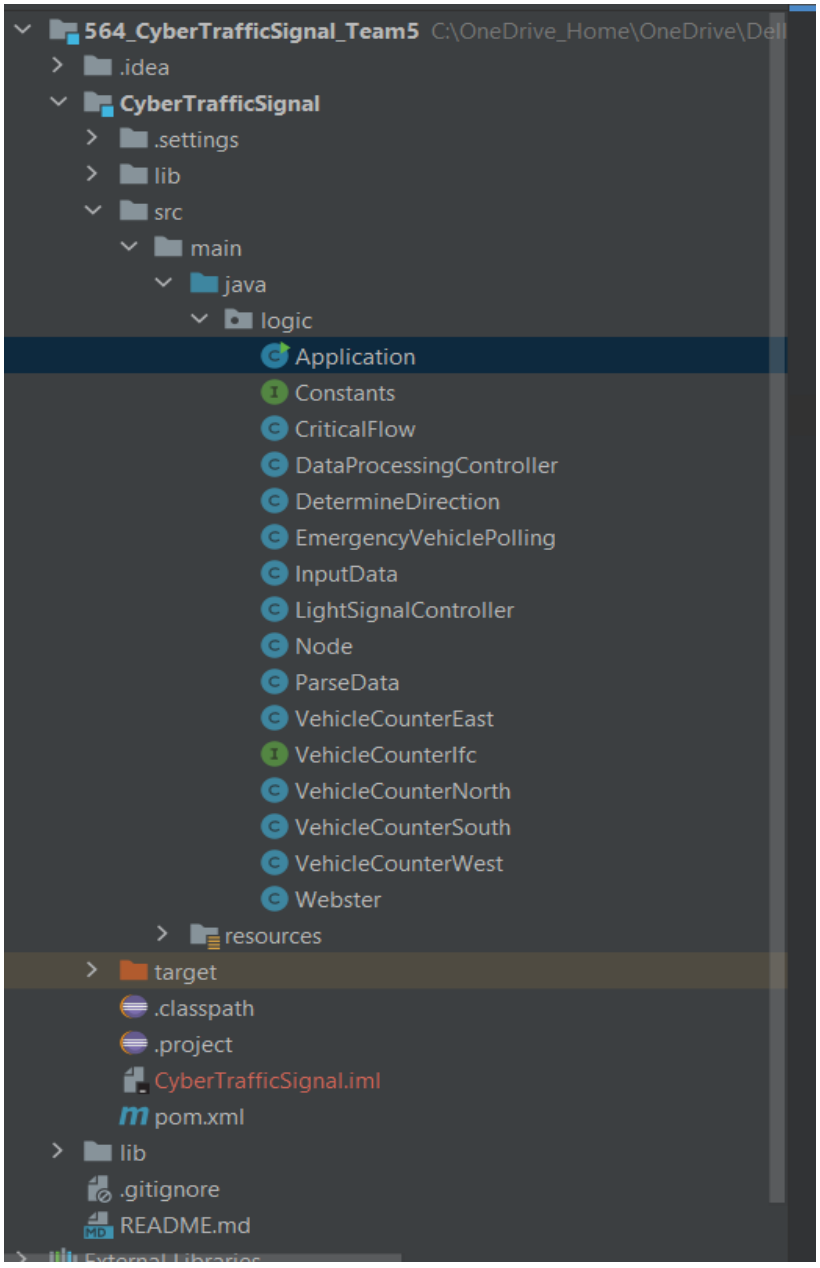
Class Definition	
Name	**Description**
DataProcessingController	This class is used to determine whether there is a need to change the signal or not. This change signal is only sent when the greenTime counter variable is changed to 0.
Attribute Definition	
greenTime	This attribute has the integer value which represents the number of seconds a signal is turned to green.
counter	This attribute has the integer value which represents the number of seconds remaining for the signal to be changed.
direction	This attribute has the string value which represents the direction which has to be changed. This helps in storage for determining direction.
emergency	This attribute has the string value which represents the emergency vehicle's presence in a particular direction.
Method Definition	
DataProcessingController(greenTime : Integer, direction : String, emergency : String)	This is a constructor which assigns the value for all the variables when an object of this class is created.
calculateChangeSignal(): Map<String, Boolean>	This method makes a call to the LightSignalController and tracks the value for the last execution by storing the values in the map.

Class Definition	
Name	**Description**
LightSignalController	This class is used to determine which signal to be turned green. It checks for the current signal, verifies for the presence of any emergency vehicle.
Attribute Definition	

direction	This attribute has the string value which represents the direction which has to be changed. This helps in storage for determining direction.
emergency	This attribute has the string value which represents the emergency vehicle's presence in a particular direction.
isEmergency	This attribute is used to store the value of emergency and maintain this emergency value for further rounds until the time completes.
counter	This attribute has the integer value which represents the number of seconds the signal remains unchanged.
dir	This attribute has the string value which represents the direction which has to be changed. This helps in storage for determining direction each of the rounds has it is dependent on the emergency vehicles and rounds happening.
NSgsLatch	This attribute has the boolean value which represents whether this signal is enabled or not. This is true when the north-south signal is green else false.
EWgsLatch	This attribute has the boolean value which represents whether this signal is enabled or not. This is true when the east-west signal is green else false.
NSrsLatch	This attribute has the boolean value which represents whether this signal is enabled or not. This is true when the north-south signal is red else false.
EWrsLatch	This attribute has the boolean value which represents whether this signal is enabled or not. This is true when the east-west signal is red else false.
NSysLatch	This attribute has the boolean value which represents whether this signal is enabled or not. This is true when the north-south signal is yellow, else false.
EWysLatch	This attribute has the boolean value which represents whether this signal is enabled or not. This is true when the east-west signal is yellow, else false.
Method Definition	
LightSignalController(direction : String, emergency : String)	This is a constructor which assigns the value for all the variables when an object of this class is created.

determineSignal(): Map<String, Boolean>	This method determines which signal to be turned green by using the value for the last execution by storing the values in the map.

3. IMPLEMENTATION



Module	Name	Type	Description	Size

				(LoC)
Main	Application	Class	The software application begins execution from this class.	132
Main	DataProcessingController	Class	Manages the light signal controller. Calls the Webster algorithm to determine maximum green light duration, presence of emergency vehicles.	60
Main	DetermineDirection	Class	Identifies the direction which has the most number of vehicles. Returns the number of vehicles and the direction. (that has the most number of vehicles.)	109
Main	VehicleCounter	Class	One class for each direction(ie, North-bound, South-bound, East-bound, West-bound). Inductive loop sensors give their output to this class.	36
Main	LightSignalController	Class	Object that represents the traffic light controller hardware,data processing controller outputs this class's object.	81
Main	Constants	Interface	Constant attributes specified.	35
Main	VehicleCounterfc	Interface	Interface for vehicle count-get method. This interface is implemented by VehicleCounter.	6
Main	Node	Class	Stores the current state of the system. Current state includes the number of vehicles, direction which has green signal in the last round, current direction having green signal, and, a boolean value indicating if the current round is a signal change.	55
Main	inputData	txt	Text file to simulate inputs from the sensors on the road. Each row represents the number of incoming and outgoing vehicles, and also the presence of emergency vehicles.	11
Input	ParseData	Class	To simulate the sensors, data is stored in a text file, ParseData reads and passes this data to InputData.java.	48

Input	InputData	Class	During each round, this class's object is passed to Application.java. Class has attribute all parameters (i.e. vehicles waiting at each intersection, presence of emergency vehicles, etc.). Application class uses this object for operation.	137
Emergency	EmergencyVehiclePolling	Class	Indicates the presence of emergency vehicles. Returns the direction in which the emergency vehicle is detected.	67
Websters	CriticalFlow	Class	Calculates the green signal time based on Webster's algorithm. Returns the direction to which green light must be turned on.	85
Websters	Webster	Class	Executes the Websters algorithm based on the input parameters, returns the green signal duration.	51

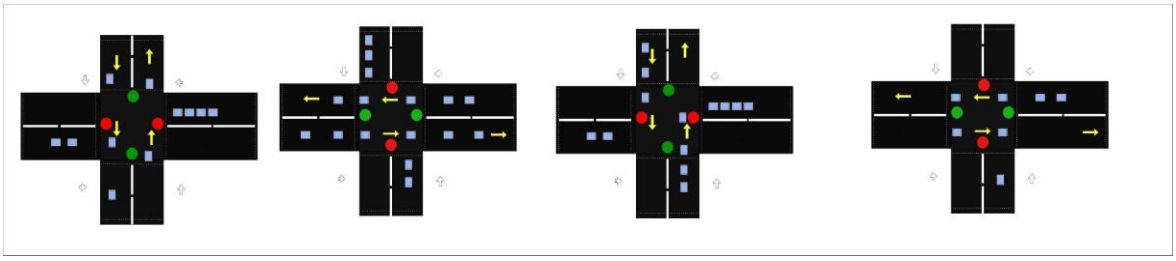
4. EXPERIMENTS AND RESULTS

To conduct our experiments and demos, we treated each row in the "inputData.txt" file as a single round.

During the normal scenario,
It provides outputs for the normal operation, configuration, and other things consistent with the project topic [5].

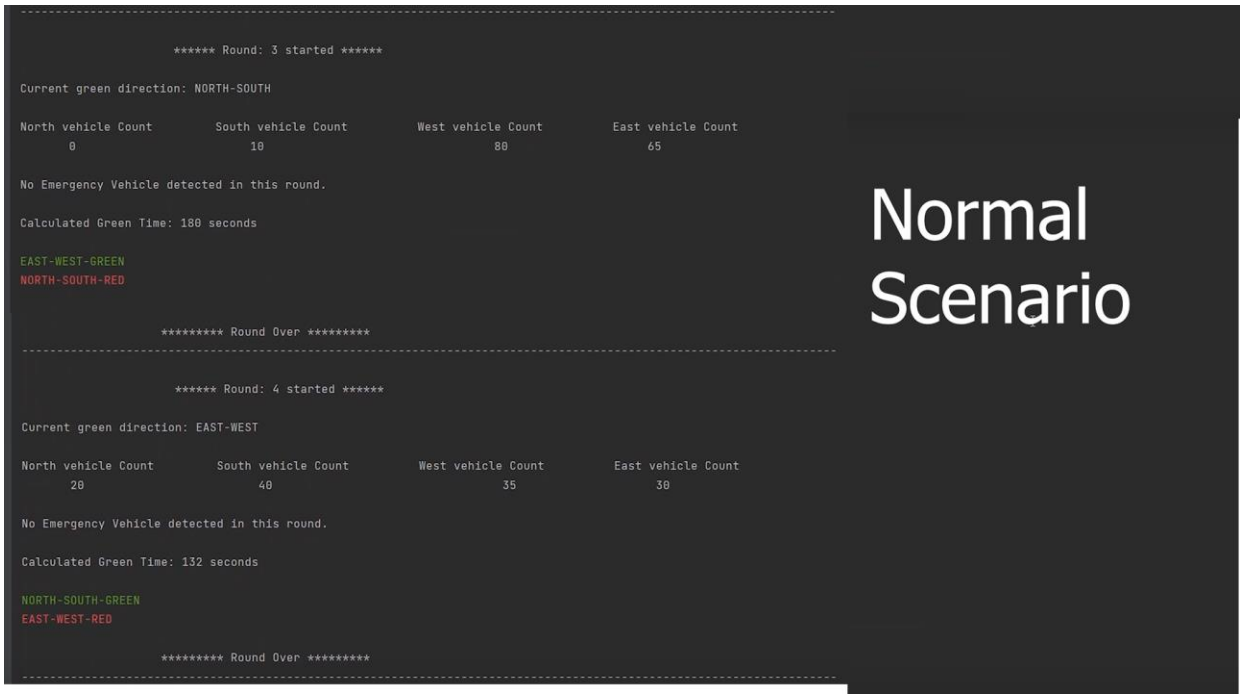
Experiment and Results

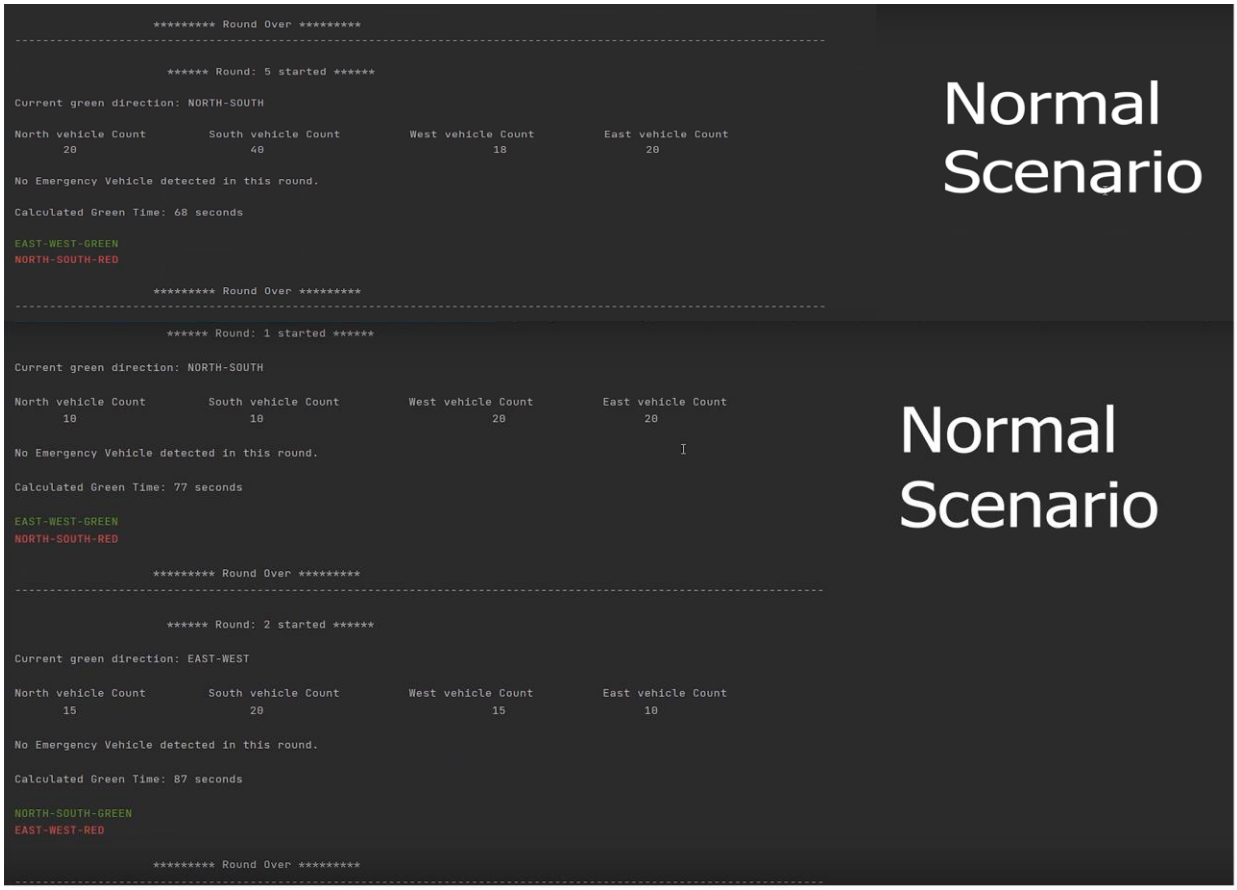
1. Normal Scenario (no Emergency vehicles approaching).



The system is said to be operating in a normal mode when there are no emergency vehicles in the intersection. During this mode, based on the number of vehicles in both directions

(north-south and east-west), the system, according to Webster's algorithm, calculates the optimal green for each direction-pair.
The following screenshots show the system in normal mode with green light alternating between the two directions.




********* Round Over *********

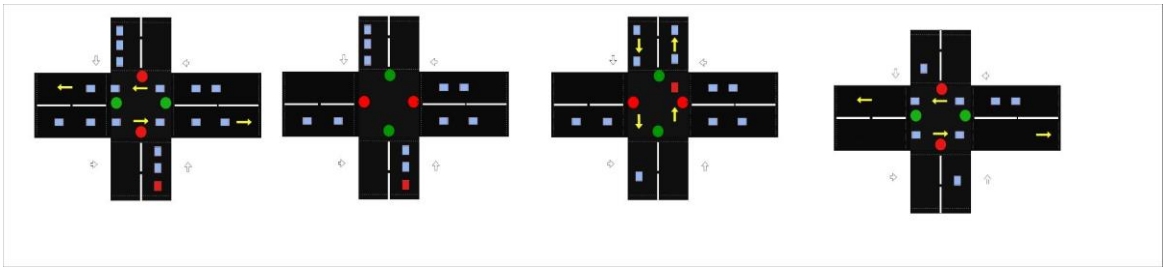
****** Round: 5 started ******
Current green direction: NORTH-SOUTH
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
20 40 18 20
No Emergency Vehicle detected in this round.
Calculated Green Time: 68 seconds
EAST-WEST-GREEN
NORTH-SOUTH-RED
********* Round Over *********

Normal Scenario
****** Round: 1 started ******
Current green direction: NORTH-SOUTH
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
10 10 20 20
No Emergency Vehicle detected in this round.
Calculated Green Time: 77 seconds
EAST-WEST-GREEN
NORTH-SOUTH-RED
********* Round Over *********

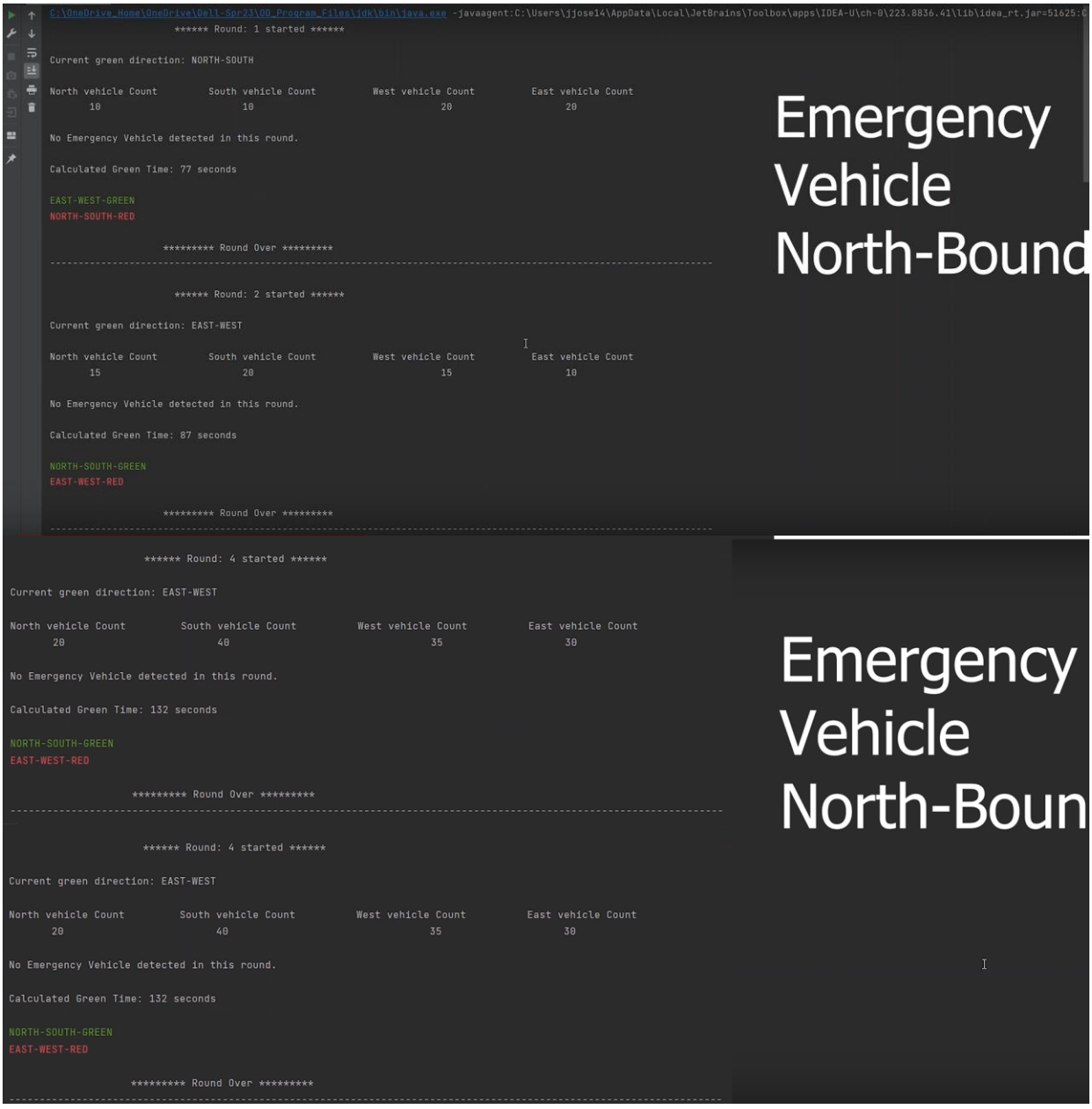
****** Round: 2 started ******
Current green direction: EAST-WEST
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
15 20 15 10
No Emergency Vehicle detected in this round.
Calculated Green Time: 87 seconds
NORTH-SOUTH-GREEN
EAST-WEST-RED
********* Round Over *********

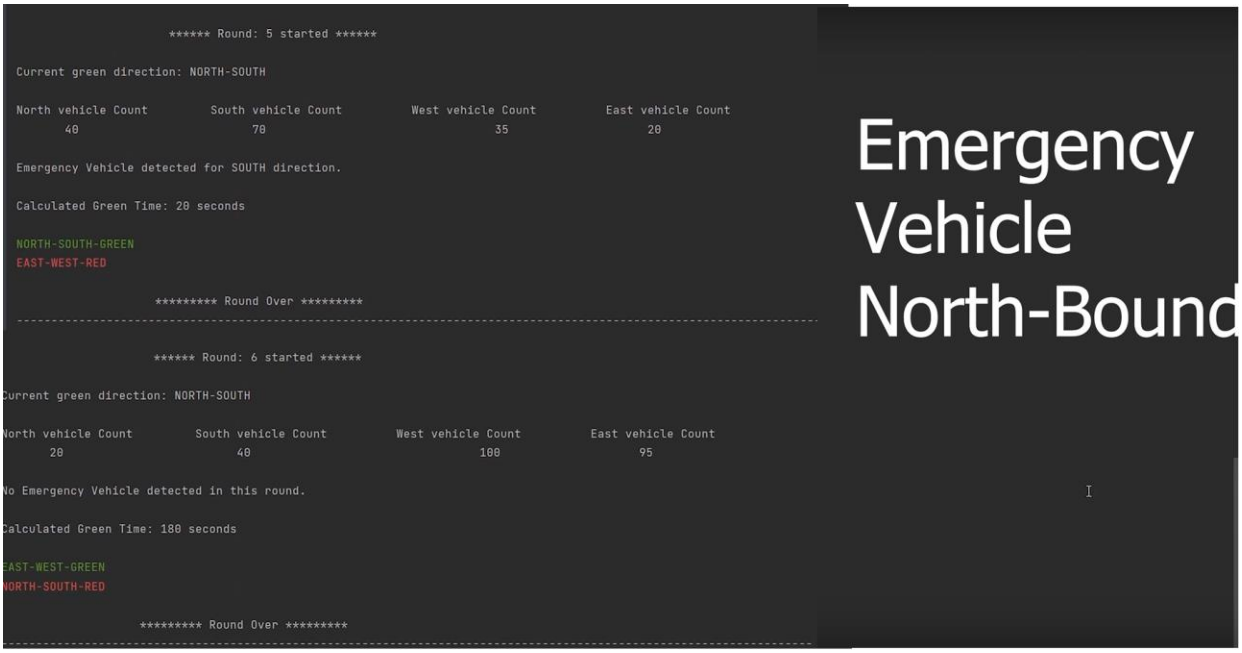
Normal Scenario

2. Emergency vehicle approaching red light.

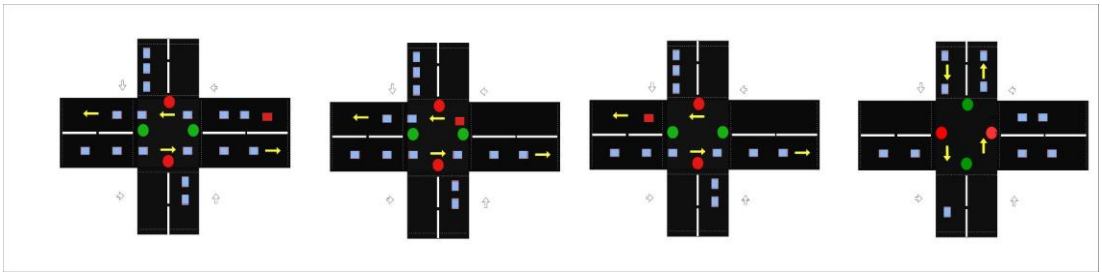


The following screenshots depict the system behavior when an emergency vehicle is approaching the intersection, and when it is facing a red traffic light. In such a condition, the controller switches the current direction to red and then changes the traffic light of the emergency vehicle to green. Here, we are assuming that people on the road will make way for the emergency vehicle to pass the intersection, because of this assumption the system shows green for 20 seconds. After 20 seconds, the system resumes normal operation.


C:\OneDrive_Home\OneDrive\Dell-Spr23\OO_Program_Files\jdk\bin\java.exe -javaagent:C:\Users\jjose14\AppData\Local\JetBrains\Toolbox\apps\IDEA-U\ch-0\223.8836.41\lib\idea_rt.jar=51625:C
****** Round: 1 started ******
Current green direction: NORTH-SOUTH
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
10 10 20 20
No Emergency Vehicle detected in this round.
Calculated Green Time: 77 seconds
EAST-WEST-GREEN
NORTH-SOUTH-RED
********* Round Over *********
****** Round: 2 started ******
Current green direction: EAST-WEST
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
15 20 15 10
No Emergency Vehicle detected in this round.
Calculated Green Time: 87 seconds
NORTH-SOUTH-GREEN
EAST-WEST-RED
********* Round Over *********
Emergency Vehicle North-Bound
****** Round: 4 started ******
Current green direction: EAST-WEST
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
20 40 35 30
No Emergency Vehicle detected in this round.
Calculated Green Time: 132 seconds
NORTH-SOUTH-GREEN
EAST-WEST-RED
********* Round Over *********
****** Round: 4 started ******
Current green direction: EAST-WEST
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
20 40 35 30
No Emergency Vehicle detected in this round.
Calculated Green Time: 132 seconds
NORTH-SOUTH-GREEN
EAST-WEST-RED
********* Round Over *********
Emergency Vehicle North-Boun

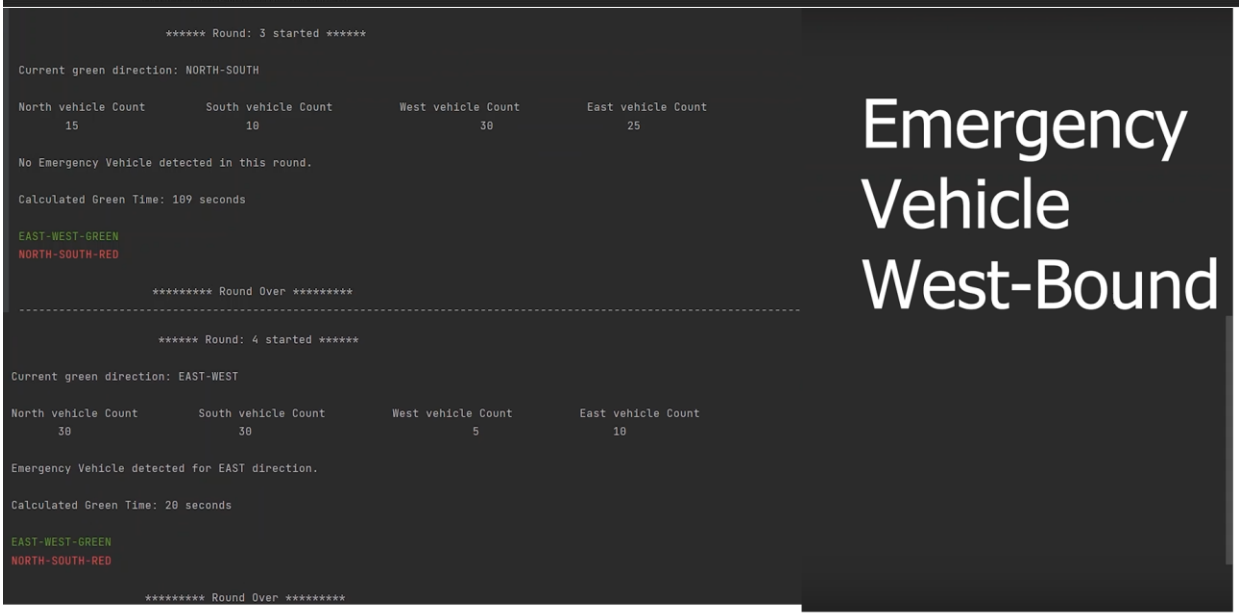


3. Emergency vehicles approaching green light.



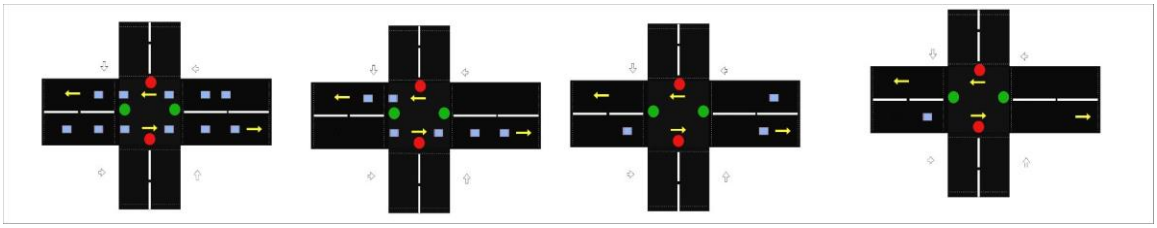
When an emergency vehicle approaches an intersection and when its direction is having a green light, the controller does not change the light for 20 seconds. Here it is also assumed that other civilian vehicles would facilitate the emergency vehicle by giving its right of way.


****** Round: 1 started ******
Current green direction: NORTH-SOUTH
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
10 10 20 20
No Emergency Vehicle detected in this round.
Calculated Green Time: 77 seconds
EAST-WEST-GREEN
NORTH-SOUTH-RED
********* Round Over *********
****** Round: 2 started ******
Current green direction: EAST-WEST
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
15 20 15 10
No Emergency Vehicle detected in this round.
Calculated Green Time: 87 seconds
NORTH-SOUTH-GREEN
EAST-WEST-RED
********* Round Over *********
Emergency Vehicle West-Bound

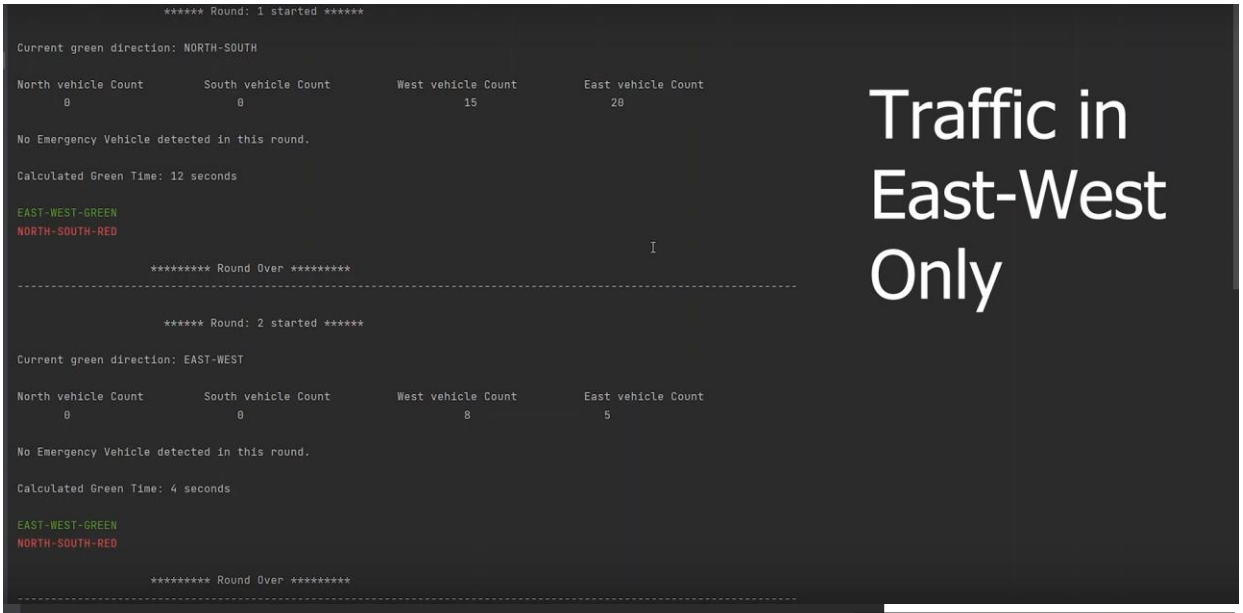

****** Round: 3 started ******
Current green direction: NORTH-SOUTH
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
15 10 30 25
No Emergency Vehicle detected in this round.
Calculated Green Time: 109 seconds
EAST-WEST-GREEN
NORTH-SOUTH-RED
********* Round Over *********
****** Round: 4 started ******
Current green direction: EAST-WEST
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
30 30 5 10
Emergency Vehicle detected for EAST direction.
Calculated Green Time: 20 seconds
EAST-WEST-GREEN
NORTH-SOUTH-RED
********* Round Over *********
Emergency Vehicle West-Bound


****** Round: 5 started ******
Current green direction: EAST-WEST
North vehicle Count South vehicle Count West vehicle Count East vehicle Count
30 30 0 0
No Emergency Vehicle detected in this round.
Calculated Green Time: 25 seconds
NORTH-SOUTH-GREEN
EAST-WEST-RED
********* Round Over *********
Emergency Vehicle West-Bound

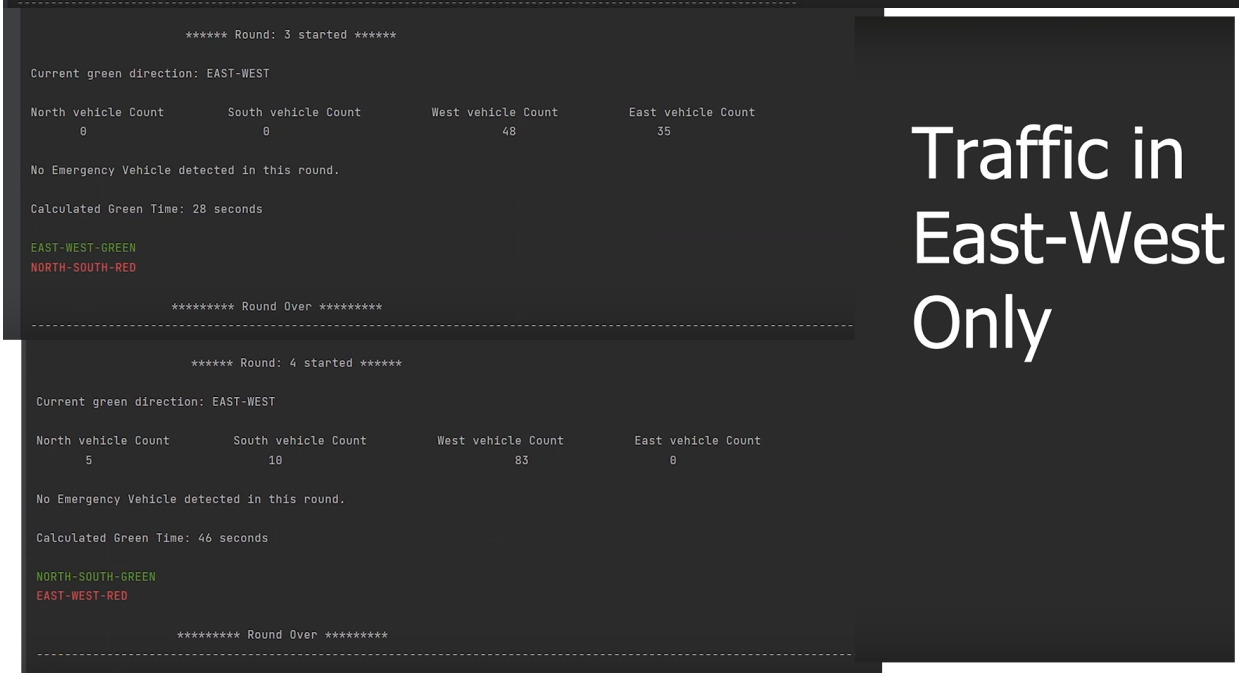
4. Traffic only in one direction



If no vehicles approach an interaction, that direction will not have a green light. This will ensure that the vehicles in other directions are able to move freely.
The following screenshots show a scenario where there are no vehicles approaching in the north or south direction. The controller can be seen to be outputting green light only to east-west bound lights.



Traffic in East-West Only



Traffic in East-West Only

5. FRAMEWORKS AND SOFTWARE TOOLS

Software	Purpose
Astah	For UML Diagrams.
Draw.io	Class diagrams.
Eclipse, IntelliJ	IDEs for Java coding and validation.
Git	Source code version control.
Apache Maven	For source code libraries.

6. CONCLUSIONS

Cyber-Physical Traffic Control System is our first software project that involved a very close relationship with hardware components. By actively taking part in each of the development processes which started with identifying the requirements of the system to deploying the system using software simulation, we were able to have a deep understanding of synchronous reactive components. The behavior of components based on rounds was a new concept for us, however, we were quick to grasp.

In conclusion, the development of a traffic signal control cyber physical system is an important step towards improving traffic efficiency and safety in modern cities. Our system integrates real-time data from multiple sensors and uses advanced algorithms to optimize traffic flow and reduce congestion. Through extensive testing and evaluation, we have demonstrated the effectiveness of our system in improving traffic flow and reducing travel time. We believe that our system has the potential to make a significant impact on the way traffic is managed in urban areas and we are excited to continue exploring new ways to optimize traffic control using cyber physical systems.

As a team, we learned that using SRC for software design, particularly for safety-critical applications, is beneficial. Conducting a dry run of the state machine and transitioning to a semi-formal design approach helps identify potential loopholes in the application, which could otherwise lead to disastrous consequences. Furthermore, this approach provides a useful template or starting point for designing mission-critical applications.

Evaluation - Team

The objective of this project was to design and develop an innovative traffic control system that can adapt to the traffic flow and minimize congestion at intersections. The team was assigned to carry out the project and was evaluated on various aspects, including project management, team collaboration, technical expertise, and overall project success.

Project Management:

The team demonstrated excellent project management skills throughout the project. They followed a well-defined project plan, which included the scope, timeline, milestones, and deliverables. The team regularly updated the project status and communicated effectively with the professors and TA. The team ensured that the project was completed on time even though our team was first to do presentations.

Team Collaboration:

Though located at different parts of the city, the team exhibited outstanding collaboration skills, which played a critical role in the project's success. They worked cohesively and efficiently, and every member of the team contributed to the project's success. The team members were respectful of each other's opinions and worked together to resolve any conflicts.

Technical Expertise:

The team possessed excellent technical expertise, which was evident in the quality of the system they developed. They demonstrated a thorough understanding of synchronous reactive components and applied their knowledge to design and implement an effective adaptive light signal controller. The team was proficient at implementing the concepts of rounds, synchronicity learnt during the course.

Overall Project Success:

The team's hard work, dedication, and technical expertise resulted in a successful project outcome. The adaptive light signal controller that the team developed was effective in reducing traffic congestion and improving traffic flow at intersections.

Evaluation - Self - Jacob Jose

I was able to get a good understanding of how hardware components interact with software. The various assignments in the course helped me attain a very thorough grasp of synchronous-reactive components. I would also like to say that by doing the project towards the end of all assignments greatly helped me implement designs in the correct way. Overall, it was a rewarding experience.

Evaluation - Self - Sai Varun Vaka

I am very proud of my work on this project, where I designed a part of the synchronous reactive component, UML, and implementation for a cyber physical traffic control system. I believe that my work on this project was highly effective, as I was able to design a system that integrated physical and cyber components to optimize traffic flow and reduce congestion in real-time. One of my key strengths in this project was my ability to design the system using UML. I was able to create clear and concise diagrams that accurately represented the system architecture and design. Additionally, my work on the synchronous reactive component was highly effective. By designing a system that could dynamically adjust the timing of traffic signals based on real-time traffic data, I was able to identify feedback loop for the application. I believe that my work on this component was critical to the success of the project, as it was a key part of the overall system design. In terms of areas for improvement, I believe that I could have done more to document my work and communicate my progress with the rest of the team. While I was able to design and implement the system effectively, I could have done more to keep the team updated on my progress and provide clear documentation of my work.

Evaluation - Self - Kanav Sharma

Since the project was assigned after the completion of the first chapter, I had a strong foundation to understand the connection between the concepts of Synchronous Reactive Components (SRC) taught in the class and the development of the project. Initially, I identified that our project was a Cyber-Physical System (CPS) and determined its boundaries with the physical environment. To simplify the project, I divided it into smaller subproblems and identified the appropriate software reactive components. After creating individual SRCs, I utilized the composite SRC concepts that I learned in the course to merge them and give shape to the project.

Conducting a dry run of the state machine was beneficial for me to reinforce my understanding of the concepts learned and to address any issues encountered during the process. I applied the lessons learned in transitioning from formal to informal language to develop UML class diagrams, establish relationships between the classes, and finalize the multiplicity. Additionally, I gained insights into the concept of asynchronicity and learned how to implement threading to achieve a particular goal, but SRC was sufficient to accomplish the same tasks.

My understanding of synchronous reactive components was excellent, and I applied my technical expertise to implement the adaptive light signal controller system successfully. I efficiently identified and resolved technical challenges that arose during the project. As the team leader, I ensured that everyone had a solid understanding of the concepts learned in class and during assignments, and I fostered a collaborative team environment. Managing the project timeline was a key responsibility, and I carefully considered various factors such as assignment due dates, presentation deadlines, and remaining work.

I not only served as a team lead but also dedicated myself as a team member who was receptive to feedback and suggestions from the team.

References

Additional references should be included as needed.

[1] R. Alur, (2015), Principles of Cyber-Physical Systems, MIT Press.
[2] G. Booch, et al., (2007), Object Oriented Analysis and Design (OOAD), 3rd Ed., Addison Wesley.
[3] Java Platform, Standard Edition (Java SE), (2019), https://www.oracle.com/java/technologies/java-se.html.
[4] OMG 2012. "Unified Modeling Language version 2.5.1". https://www.omg.org/spec/UML/2.5.1/.
[5] "Traffic Signal Design | Webster's Formula for Optimum Cycle Length." *Traffic Signal Design | Webster's Formula for Optimum Cycle Length*, 17 July 2022, www.apsed.in/post/traffic-signal-design-webster-s-formula-for-optimum-cycle-length.

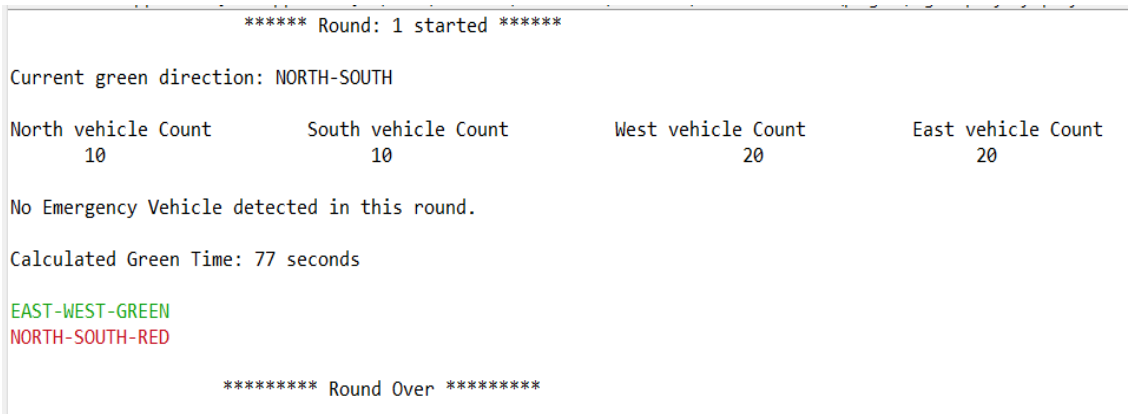
[6] Ren Caigui, "Research and design of signal phase scheme for different traffic flow and traffic volume," *2011 International Conference on Remote Sensing, Environment and Transportation Engineering*, Nanjing, China, 2011, pp. 1473-1475, doi: 10.1109/RSETE.2011.5964562.
[7] Y. Quan, L. Jin-guang, L. Pei-hua, R. Jian and L. Xiao-ming, "Dynamic Optimization Project Study between the Traffic Organization and the Traffic Signal Control of Urban Traffic," *2009 WRI World Congress on Computer Science and Information Engineering*, Los Angeles, CA, USA, 2009, pp. 182-186, doi: 10.1109/CSIE.2009.63.
[8] C. Wei, W. Dianhai, C. Yuguang, Y. Manrong and L. Xuemin, "Research on Signal Control Methods of Traffic Bottlenecks in City Road Network," *2009 WRI Global Congress on Intelligent Systems*, Xiamen, China, 2009, pp. 450-454, doi: 10.1109/GCIS.2009.416.

7. APPENDICES

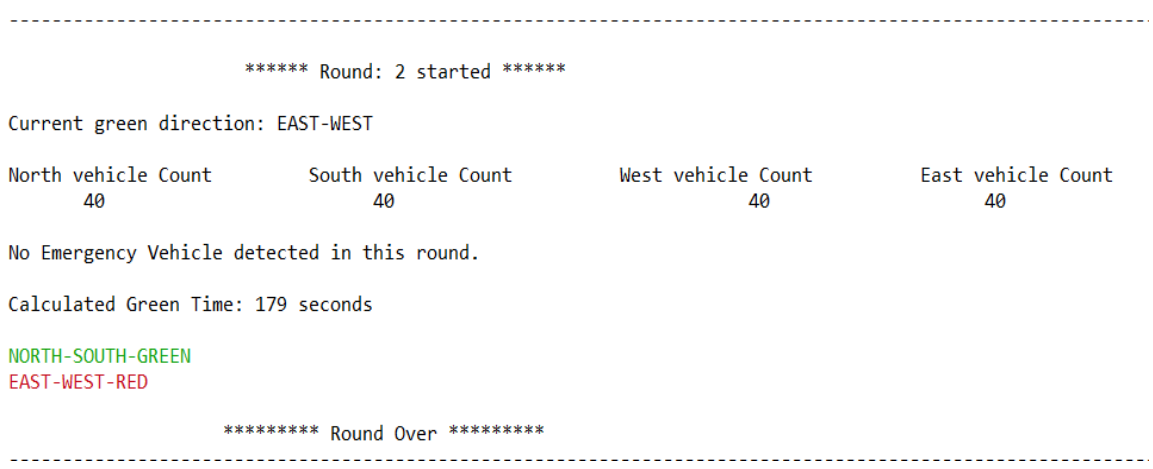
Sample Data:

1. Round 1:

   10,0,10,0,20,0,20,0,1000,1250,false,false,false,false



2. 30,0,30,0,40,20,40,20,1000,1250,false,false,false,false



3. 10,30,20,30,55,0,65,0,1000,1250,false,false,false,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 3 started ******

Current green direction: NORTH-SOUTH

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       20                          30                          105                         95

No Emergency Vehicle detected in this round.

Calculated Green Time: 180 seconds

EAST-WEST-GREEN
NORTH-SOUTH-RED

                  ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```

4. 20,0,30,0,20,55,20,65,1000,1250,true,false,false,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 4 started ******

Current green direction: EAST-WEST

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       40                          60                          60                          60

Emergency Vehicle detected for NORTH direction.

Calculated Green Time: 20 seconds

NORTH-SOUTH-GREEN
EAST-WEST-RED

                  ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```

5. 0,20,0,30,75,0,65,0,1000,1250,false,false,false,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 5 started ******

Current green direction: NORTH-SOUTH

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       20                          30                          125                         135

No Emergency Vehicle detected in this round.

Calculated Green Time: 180 seconds

EAST-WEST-GREEN
NORTH-SOUTH-RED

                  ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```

6. 0,0,0,0,0,50,0,50,1000,1250,false,false,true,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 6 started ******

Current green direction: EAST-WEST

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       20                          30                          75                          85

Emergency Vehicle detected for EAST direction.

Calculated Green Time: 20 seconds

EAST-WEST-GREEN
NORTH-SOUTH-RED

                    ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```

7. 0,0,0,0,0,30,0,20,1000,1250,false,false,false,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 7 started ******

Current green direction: EAST-WEST

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       20                          30                          55                          55

No Emergency Vehicle detected in this round.

Calculated Green Time: 110 seconds

NORTH-SOUTH-GREEN
EAST-WEST-RED

                    ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```

8. 0,20,0,30,0,0,0,0,1000,1250,false,false,false,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 8 started ******

Current green direction: NORTH-SOUTH

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       0                           0                           55                          55

No Emergency Vehicle detected in this round.

Calculated Green Time: 37 seconds

EAST-WEST-GREEN
NORTH-SOUTH-RED

                  ********* Round Over *********
```

9. 0,0,0,0,0,55,0,55,1000,1250,false,false,false,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 9 started ******

Current green direction: EAST-WEST

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       0                           0                           0                           0

No Emergency Vehicle detected in this round.

Calculated Green Time: 30 seconds

EAST-WEST-GREEN
NORTH-SOUTH-RED

                  ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```

10. 0,0,0,0,0,0,0,0,0,1000,1250,false,false,false,false

```
------------------------------------------------------------------------------------------------------------

                    ****** Round: 10 started ******

Current green direction: EAST-WEST

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       0                           0                           0                           0

No Emergency Vehicle detected in this round.

Calculated Green Time: 30 seconds

EAST-WEST-GREEN
NORTH-SOUTH-RED

                  ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```

11. 1,0,0,0,0,0,0,0,0,1000,1250,false,false,false,false

```
------------------------------------------------------------------------------------------------------------

                      ****** Round: 11 started ******

Current green direction: EAST-WEST

North vehicle Count         South vehicle Count          West vehicle Count          East vehicle Count
       1                          0                            0                          0

No Emergency Vehicle detected in this round.

Calculated Green Time: 30 seconds

NORTH-SOUTH-GREEN
EAST-WEST-RED

                     ********* Round Over *********
------------------------------------------------------------------------------------------------------------
```