@startuml

hide empty description

[\*] --> Idle : count = 0;\nintersectionFlag = free;

Idle --> Idle : detectCarInput() == true / count++;

Idle --> RippleOut : count > 0 / \nif not in queue\n addToQueue()

RippleOut --> RippleIn

RippleIn --> Idle : intersectionFlag== busy /

RippleIn --> TurnOnGreen : intersectionFlag== free /\nremoveFromQueue();\intersectionFlag= busy;

RippleIn --> InvalidMessages

TurnOnGreen : turnOn(greenLight);\nstartTimer();

TurnOnGreen --> TurnOffGreen : timerExpired() == true /

TurnOnGreen --> TurnOnGreen : detectCarInput() == true /\n count++;

TurnOnGreen --> TurnOnGreen : detectCarOutput() == true /\n count--;

TurnOnGreen --> TurnOffGreen : count == 0 /

TurnOffGreen : turnOff(greenLight);\nintersectionFlag = free;

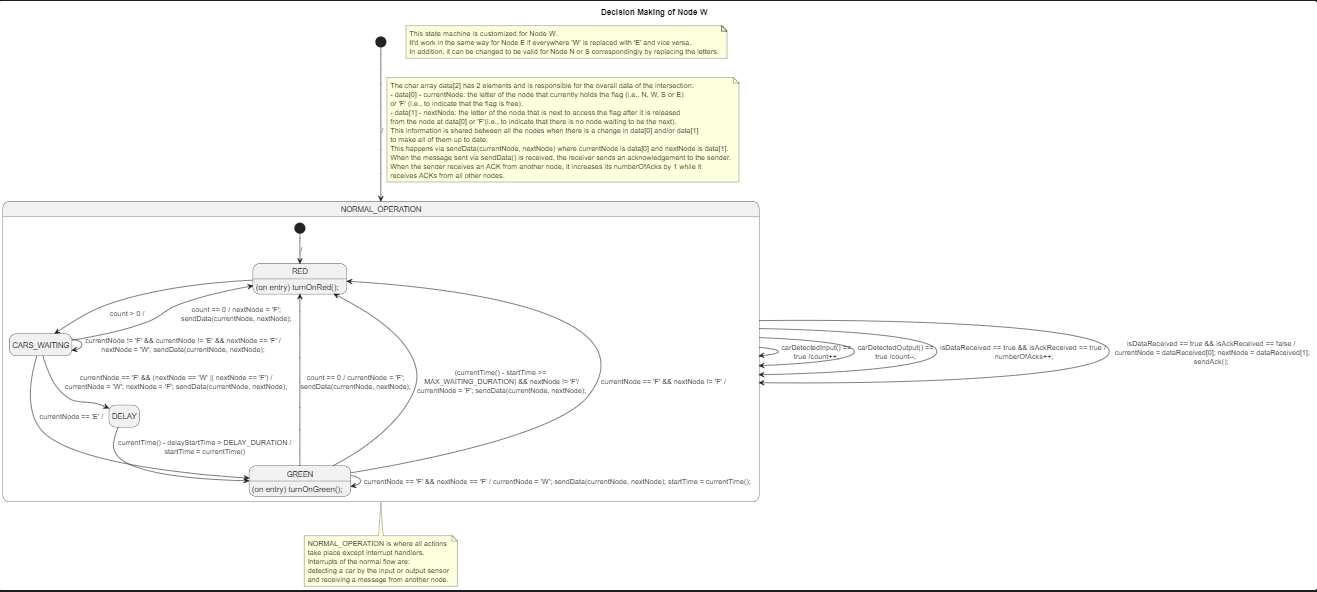
TurnOffGreen --> RippleOutFinished : sendMsg(intersectionFlag);

RippleOutFinished --> RippleInFinished

RippleInFinished --> Idle

RippleInFinished --> InvalidMessages

@enduml



NODE APPROVED BY SUZANA WITH ACKS (BUT NOT ACK STATES

@startuml Decision Making

title Decision Making of Node W

hide empty description

[\*] --> NORMAL\_OPERATION : /

note right on link

    The char array data[2] has 2 elements and is responsible for the overall data of the intersection:

    - data[0] - currentNode: the letter of the node that currently holds the flag (i.e., N, W, S or E)

    or 'F' (i.e., to indicate that the flag is free).

    - data[1] - nextNode: the letter of the node that is next to access the flag after it is released

    from the node at data[0] or 'F'(i.e., to indicate that there is no node waiting to be the next).

    This information is shared between all the nodes when there is a change in data[0] and/or data[1]

    to make all of them up to date.

    This happens via sendData(currentNode, nextNode) where currentNode is data[0] and nextNode is data[1].

    When the message sent via sendData() is received, the receiver sends an acknowledgement to the sender.

    When the sender receives an ACK from another node, it increases its numberOfAcks by 1 while it

    receives ACKs from all other nodes.

end note

note as N1

    This state machine is customized for Node W.

    It'd work in the same way for Node E if everywhere 'W' is replaced with 'E' and vice versa.

    In addition, it can be changed to be valid for Node N or S correspondingly by replacing the letters.

end note

note bottom of NORMAL\_OPERATION

    NORMAL\_OPERATION is where all actions

    take place except interrupt handlers.

    Interrupts of the normal flow are:

    detecting a car by the input or output sensor

    and receiving a message from another node.

end note

' state BLINK : blinkYellow();

state NORMAL\_OPERATION {

    state RED : (on entry) turnOnRed();

    state GREEN : (on entry) turnOnGreen();

    [\*] --> RED : /

    RED --> CARS\_WAITING : count > 0 /

    CARS\_WAITING --> RED : count == 0 / nextNode = 'F';\nsendData(currentNode, nextNode);

    ' CARS\_WAITING --> WAIT\_ACKS\_AFTER\_CARS\_WAITING : count == 0 / nextNode = 'F';\nsendData(currentNode, nextNode);

    CARS\_WAITING --> GREEN : currentNode == 'E' /

    CARS\_WAITING --> DELAY : currentNode == 'F' && (nextNode == 'W' || nextNode == 'F') /\ncurrentNode = 'W'; nextNode = 'F'; sendData(currentNode, nextNode);

    ' CARS\_WAITING --> WAIT\_ACKS\_BEFORE\_DELAY : currentNode == 'F' && (nextNode == 'W' || nextNode == 'F') /\ncurrentNode = 'W'; nextNode = 'F'; sendData(currentNode, nextNode);\n delayStartTime = currentTime();

    ' WAIT\_ACKS\_BEFORE\_DELAY --> DELAY : numberOfAcks == NUMBER\_OF\_OTHER\_NODES /\nnumberOfAcks = 0;

state DELAY : delayStartTime = currentTime();

    DELAY --> GREEN : currentTime() - delayStartTime > DELAY\_DURATION /\nstartTime = currentTime()

    CARS\_WAITING --> CARS\_WAITING : currentNode != 'F' && currentNode != 'E' && nextNode == 'F' /\nnextNode = 'W'; sendData(currentNode, nextNode);

    GREEN --> RED : count == 0 / currentNode = 'F';\nsendData(currentNode, nextNode);

    GREEN --> RED : (currentTime() - startTime >= \nMAX\_WAITING\_DURATION) && nextNode != 'F'/\ncurrentNode = 'F'; sendData(currentNode, nextNode);

    GREEN --> RED : currentNode == 'F' && nextNode != 'F' /

    GREEN --> GREEN : currentNode == 'F' && nextNode == 'F' / currentNode = 'W'; sendData(currentNode, nextNode); startTime = currentTime();

}

NORMAL\_OPERATION --> NORMAL\_OPERATION : carDetectedInput() ==\ntrue /count++;

NORMAL\_OPERATION --> NORMAL\_OPERATION : carDetectedOutput() ==\ntrue /count--;

NORMAL\_OPERATION --> NORMAL\_OPERATION : isDataReceived == true && isAckReceived == true /\n numberOfAcks++;

NORMAL\_OPERATION --> NORMAL\_OPERATION : isDataReceived == true && isAckReceived == false /\n currentNode = dataReceived[0]; nextNode = dataReceived[1];\nsendAck();

' NORMAL\_OPERATION --> BLINK : dataReceived() == \n"ER" /

@enduml

STATE MACHINE ACKS STATES INCLUDED

@startuml Decision Making

title Decision Making of Node W

hide empty description

[\*] --> NORMAL\_OPERATION : /

note right on link

The char array data[2] has 2 elements and is responsible for the overall data of the intersection:

- data[0] - currentNode: the letter of the node that currently holds the flag (i.e., N, W, S or E)

or 'F' (i.e., to indicate that the flag is free).

- data[1] - nextNode: the letter of the node that is next to access the flag after it is released

from the node at data[0] or 'F'(i.e., to indicate that there is no node waiting to be the next).

This information is shared between all the nodes when there is a change in data[0] and/or data[1]

to make all of them up to date.

This happens via sendData(currentNode, nextNode) where currentNode is data[0] and nextNode is data[1].

When the message sent via sendData() is received, the receiver sends an acknowledgement to the sender.

When the sender receives an ACK from another node, it increases its numberOfAcks by 1 while it

receives ACKs from all other nodes.

end note

note as N1

This state machine is customized for Node W.

It'd work in the same way for Node E if everywhere 'W' is replaced with 'E' and vice versa.

In addition, it can be changed to be valid for Node N or S correspondingly by replacing the letters.

end note

note bottom of NORMAL\_OPERATION

NORMAL\_OPERATION is where all actions

take place except interrupt handlers.

Interrupts of the normal flow are:

detecting a car by the input or output sensor

and receiving a message from another node.

end note

' state BLINK : blinkYellow();

state NORMAL\_OPERATION {

state RED #red: (on entry) turnOnRed();

state GREEN #green: (on entry) turnOnGreen();

state WAIT\_ACKS\_CARS\_WAITING\_RED #Gray

state WAIT\_ACKS\_CARS\_WAITING\_DELAY #Gray

state WAIT\_ACKS\_CARS\_WAITING\_TO\_ITSELF #Gray

state WAIT\_ACKS\_GREEN\_RED\_1 #Gray

state WAIT\_ACKS\_GREEN\_RED\_2 #Gray

state WAIT\_ACKS\_GREEN\_TO\_ITSELF #Gray

[\*] --> RED : /

CARS\_WAITING --> WAIT\_ACKS\_CARS\_WAITING\_RED : count == 0 / nextNode = 'F';\nsendData(currentNode, nextNode);

WAIT\_ACKS\_CARS\_WAITING\_RED --> RED : numberOfAcks == NUMBER\_OF\_OTHER\_NODES /\nnumberOfAcks = 0;

RED --> CARS\_WAITING : count > 0 /

CARS\_WAITING --> GREEN : currentNode == 'E' /

state DELAY : delayStartTime = currentTime();

CARS\_WAITING --> WAIT\_ACKS\_CARS\_WAITING\_DELAY : currentNode == 'F' && (nextNode == 'W' || nextNode == 'F') /\ncurrentNode = 'W'; nextNode = 'F'; sendData(currentNode, nextNode);\n delayStartTime = currentTime();

WAIT\_ACKS\_CARS\_WAITING\_DELAY --> DELAY : numberOfAcks == NUMBER\_OF\_OTHER\_NODES /\nnumberOfAcks = 0;

DELAY --> GREEN : currentTime() - delayStartTime > DELAY\_DURATION /\nstartTime = currentTime()

CARS\_WAITING --> WAIT\_ACKS\_CARS\_WAITING\_TO\_ITSELF : currentNode != 'F' && currentNode != 'E' && nextNode == 'F' /\nnextNode = 'W'; sendData(currentNode, nextNode);

WAIT\_ACKS\_CARS\_WAITING\_TO\_ITSELF --> CARS\_WAITING : numberOfAcks == NUMBER\_OF\_OTHER\_NODES /\nnumberOfAcks = 0;

GREEN --> WAIT\_ACKS\_GREEN\_RED\_1 : count == 0 / currentNode = 'F';\nsendData(currentNode, nextNode);

WAIT\_ACKS\_GREEN\_RED\_1 --> RED : numberOfAcks == NUMBER\_OF\_OTHER\_NODES /\nnumberOfAcks = 0;

GREEN --> WAIT\_ACKS\_GREEN\_RED\_2 : (currentTime() - startTime >= \nMAX\_WAITING\_DURATION) && nextNode != 'F'/\ncurrentNode = 'F'; sendData(currentNode, nextNode);

WAIT\_ACKS\_GREEN\_RED\_2 --> RED : numberOfAcks == NUMBER\_OF\_OTHER\_NODES /\nnumberOfAcks = 0;

GREEN --> RED : currentNode == 'F' && nextNode != 'F' /

GREEN --> WAIT\_ACKS\_GREEN\_TO\_ITSELF : currentNode == 'F' && nextNode == 'F' / currentNode = 'W'; sendData(currentNode, nextNode); startTime = currentTime();

WAIT\_ACKS\_GREEN\_TO\_ITSELF --> GREEN : numberOfAcks == NUMBER\_OF\_OTHER\_NODES /\nnumberOfAcks = 0;

}

NORMAL\_OPERATION --> NORMAL\_OPERATION : carDetectedInput() ==\ntrue /count++;

NORMAL\_OPERATION --> NORMAL\_OPERATION : carDetectedOutput() ==\ntrue /count--;

NORMAL\_OPERATION --> NORMAL\_OPERATION : isDataReceived == true && isAckReceived == true /\n numberOfAcks++;

NORMAL\_OPERATION --> NORMAL\_OPERATION : isDataReceived == true && isAckReceived == false /\n currentNode = dataReceived[0]; nextNode = dataReceived[1];\nsendAck();

' NORMAL\_OPERATION --> BLINK : dataReceived() == \n"ER" /

@enduml

A screenshot of a computer

Description automatically generated

