Class Diagram

The following page shows us the class diagram of our application. Note that this class diagram is not final, that is, some form of refactoring would take place throughout the development process.

In actual fact, over the last 6 weeks, this is actually a revised version of the class diagram. The initial class diagram (first class diagram) is also included, to show the progress and the refactoring that was done to make the first revision of the class diagram.

Old diagram here

new class diagram to go here

Class Descriptions

The following are the class descriptions, as per the class diagram that has been prepared. Note that this is not the final list, hence there may be some changes as we progress with the development of the application.

AbstractClassState

This is an abstract class, used to define classes that would have some form of state.

- Variables:
  - valueOnOff (bool) : Stores the current state of the item
- Functions:
  - setValue() : sets a value into the variable
  - getValue () : returns the value of the variable in the class

AbstractClassDirection

This is an abstract class, used to define classes that would involve direction (like wheels).

- Variables:
  - valueDirection (int) : Stores the current value of the direction, like speed
- Functions:
  - setValue(int) : Saves the integer value passed in into the valueDirection variable
  - getValue() : Returns the current stored value in valueDirection

Lights:AbstractClassState

A parent class that is defined based on the AbstractClassState. Acts a parent class for all light related classes.

RearFogLights

Child of the Lights class. Represents the rear fog light in a car.

- Functions:
  - Turn_On_Lights() : Sets the valueOnOff to true, indicating it is on.
  - Turn_Off_Lights(): Sets the valueOnOff to false, indicating it is off.

Headlights

Child of the Lights class. Represents the headlights light in a car.

- Functions:

- Turn_On_Lights() : Sets the valueOnOff to true, indicating it is on.
- Turn_Off_Lights(): Sets the valueOnOff to false, indicating it is off.

RearFogLights

Child of the Lights class. Represents the rear fog light in a car.

- Functions:
  - Turn_On_Lights() : Sets the valueOnOff to true, indicating it is on.
  - Turn_Off_Lights(): Sets the valueOnOff to false, indicating it is off.

Highbeams

Child of the Lights class. Represents high beam light in a car.

- Functions:
  - Turn_On_Lights() : Sets the valueOnOff to true, indicating it is on.
  - Turn_Off_Lights(): Sets the valueOnOff to false, indicating it is off.

FrontFogLights

Child of the Lights class. Represents the front fog light in a car.

- Functions:
  - Turn_On_Lights() : Sets the valueOnOff to true, indicating it is on.
  - Turn_Off_Lights(): Sets the valueOnOff to false, indicating it is off.

FrontIndicatorLights

Child of the Lights class. Represents the front indicator in a car.

- Functions:
  - Turn_Hazard_On() : Sets the valueOnOff to true, indicating it is on.
  - Turn_Hazard_Off(): Sets the valueOnOff to false, indicating it is off.

RearIndicatorLights

Child of the Lights class. Represents the rear indicator in a car.

- Functions:
  - Turn_Hazard_On() : Sets the valueOnOff to true, indicating it is on.
  - Turn_Hazard_Off(): Sets the valueOnOff to false, indicating it is off.

Wipers:AbstractClassState

This represents the wipers in a car. Inherits from AbstractClassState.

- Functions
  - Sends_Power(): Sends power to the wipers, sets valueOnOff to true
  - Stop_Power() : Turns off wipers, sets valueOnOff to false

CruiseControl:AbstractClassState

This represents the cruise control system in a car. Inherits from AbstractClassState.

- Functions
  - Sends_Power(): Sends power to the cruise control system, sets valueOnOff to true
  - Stop_Power() : Turns off cruise control system, sets valueOnOff to false

AudioSystem:AbstractClassState

This represents the audio system in a car. Inherits from AbstractClassState.

- Functions
  - Sends_Power(): Sends power to the audio system, sets valueOnOff to true
  - Stop_Power() : Turns off audio system, sets valueOnOff to false

RearDefroster:AbstractClassState

This represents the rear defroster in a car. Inherits from AbstractClassState.

- Functions
  - Sends_Power(): Sends power to the rear defroster, sets valueOnOff to true
  - Stop_Power() : Turns off rear defroster, sets valueOnOff to false

CarAttachment:AbstractClassState

This represents an attachment that can be added to a car. Inherits from AbstractClassState.

- Functions
  - Check_Mounted(): Checks to see if there is an attachment mounted to the car.
  - Attach_To_Car(): Adds an attachment to the car
  - Unmount_Attachment(): Removes an attachment from the car

Trailer

Represents a trailer that can be added to the car as an attachment

- Functions
  - Add_Trailer(): Function to add the trailer and set it up

Towbox

Represents a towbox that can be added to the car as an attachment

- Functions
  - Add_Towbox(): Function to add the tow box and set it up

Program

This class represents the core for the simulation program.

- Functions
  - Engine_High_Temp() : Handles a situation where engine high temperature occurs
  - Engine_Shut_Down(): Shuts down engine
  - Validate_Headlight_Check() : Check to see if headlights have been left on

Battery:AbstractClassState

Represents the car battery in a car

- Functions
  - Electrical_Audio_Func(): Starts the audio system
  - Off_Audio_Func() : Disables the audio system
  - Cruise_Control_Func(): Starts the cruise control
  - Off_Cruise_func(): Disable the cruise control
  - Rear_Defrost_Func(): Starts the rear defrost
  - Stop_Rear_Defrost_Func(): Disables rear defrost
  - Light_Rear_Foglights_func(): Enables rear foglight
  - Off_Rear_Foglights(): Disable rear foglights
  - Lights_foglight_func(): Enable front foglight
  - Off_foglights() : disable front foglights
  - Hazard_Lights_Func(): Enable hazard lights
  - Off_Hazard_Lights_Func(): Disable hazard lights
  - Lights_Headlights_Func() : Enable headlights
  - Off_Headlights_Func(): Disable headlights
  - Light_Highbeams_Func(): Enable highbeams
  - Off_Highbeams(): Disable highbeams
  - On_Wiper_Func(): Enable wipers
  - Off_Wiper_Func(): Disable wipers
  - Get_Power(): Provide power to electrical item
  - Request_Stop_Power() : Terminate power supply

MessageCenter

Acts as a message center for the simulation.

- Functions
  - Display_Message(): Display messages that are pending
  - Remove_Message() : Clear off messages that have been displayed

FuelTank

Represents the fuel tank of a car.

- Variables
  - int rateCombust : Stores the current combustion rate
  - int fuelLevel : Stores the current fuel level in the tank
- Functions
  - InclinedCombFunc(): Adapts for providing fuel in an inclined road
  - setFuelLevel() : Stores new fuel level into fuelLevel variable
  - getFuelLevel() : Returns current fuel level of tank
  - Low_Fuel_Func() : Invokes low fuel warning
  - Fuel_Empty() : Invokes empty tank warning
  - Wrong_Fuel_Func() : Invokes incorrect fuel in tank warning
  - Check_Fuel() : Returns type of fuel in tank
  - RequestFuel() : Supplies fuel out of tank
  - Hybrid_Fuel_Func() : For use in a hybrid car
  - Higher_Altitude_Comb() : Adapts for providing fuel in a high altitude situation

Dashboard

Represents the dashboard in the car.

- Variables:
  - int fuelLevel: Stores the fuel level returned from fuel tank
  - int currentSpeed: Stores the current speed of the car
  - int engineTemp : Stores the current engine temperature returned

- Variables
  - setFuelLevel(): Sets the current fuel level from tank into the fuelLevel variable in the dashboard
  - setCurrentSpeed(): Set the current speed obtained into the currentSpeed variable
  - setEngineTemp(): Sets the engine temperature obtained into the engineTemp variable
  - getFuelLevel(): Query the fuel level from the fuel tank
  - getCurrentSpeed(): Query the current speed
  - getEngineTemp(): Query the current engine temperature
  - Light_Up_Dash(): Lights up the dash when headlights are turned on
  - Display_Wrong_Fuel(): Handles the necessary responses when wrong fuel is filled into the car
  - Show_Overheat(): Handles the necessary responses when the car overheats
  - Display_Check_High_Temp(): Displays the check engine and high temperature warning
  - Display_Defrost_Rear(): Enable the rear defrost symbol on the dashboard
  - Remove_Defrost_Rear(): Disable the rear defrost symbol on the dashboard
  - Display_Rear_Foglights(): Enable the rear fog light symbol on the dashboard
  - Remove_Rear_Foglights(): Disable the rear fog light symbol on the dashboard

- Display_Foglights(): Enable the front fog light symbol on the dashboard
- Remove_Foglights(): Disable the front fog light symbol on the dashboard
- Display_Hazard(): Enable the hazard light symbol on the dashboard
- Remove_Hazard(): Disable the hazard light symbol on the dashboard
- Display_Headlights(): Enable the headlights symbol on the dashboard
- Remove_Headlights(): Disable the headlight symbol on the dashboard
- Display_Highbeams(): Enable the highbeam symbol on the dashoard
- Remove_Highbeams(): Disable the highbeam symbol on the dashboard
- Display_Handbrake(): Enable the handbrake on symbol on the dashboard
- Remove_Handbrake() : Disable the handbrake on symbol on the dashboard

FuelPump: AbstractClassState

Represents the fuel pump in a car. Inherits from the AbstractClassState.

- Variables
  - pressure: Sets the pressure in which the fuel pump gets fuel from the tank
- Functions
  - Send_Fuel(): Gets the fuel from the tank and sends it to the engine

Engine:AbstractClassState

Represents the engine in a car. Inherits from the AbstractClassState.

- Variables
  - int temperature: Stores the engine temperature

- Functions:
  - Pumps_Fuel(): Engages the fuel pump to get fuel
  - Combust_Fuel(): Burns fuel to produce power
  - fwd_func(): Send power to front wheels
  - rwd_func(): Send power to rear wheels
  - 4wd_func(): Send power to all wheels
  - Normal_Temp_func(): Prepares for simulation of a normal operating temperature
  - Set_Initial_Temp(): Sets the value of the temperature variable
  - Needs_Coolant(): Checks to see if the engine needs coolant
  - Request_Coolant(): Request coolant from the coolant system
  - Send_Coolant_Back(): Pushes coolant out of engine back into the cooling system
  - High_Temp_Func(): Prepares for simulation of an overheat
  - Sends_Signal(): Sends out warning to dashboard
  - Stop_Signal(): Disables warnings that have been sent out

Inclinement

Class to handle driving on an incline.

- Variables:

- int inclinementValue: stores the incline of the road

- Functions:
  - setInclinement(): Set the inclinementValue variable
  - getInclinement(): Return the value in the inclinementValue variable Increase_Inclinement(): Increase the value in the inclinementValue variable
  - Decrease_Inclinement(): Decrese the value in the inclinementValue variable

HybridBattery

Class to represent the battery part of the hybrid system in a hybrid car.

- Functions:
  - Send_Power(): Sends out power from the battery to the motor

ElectricBattery

Class to represent the battery as part of the electric drive system in an electric car

- Functions:
  - Electric_Fuel_Funct(): Prepare for electric car simulation
  - Send__Electricity(): Sends electricity to motor

HydrogenTank

Class to represent the hydrogen tank in a hydrogen car

- Function:
  - Send_Fuel(): Send fuel to the fuel stack

FuelStack

Class to represent the fuel stack in a hydrogen car

- Function:
  - Send_Signal_To_Tank(): Sends a signal to the hydrogen tank to get hydrogen out
  - Produces_Electricity(): Produces electricity and sends to the motor

FrontMotor

Represents the motor for the front wheels in an electric/hybrid/hydrogen car

- Function
  - Combine_Power(): Combines power it receives (in the form of electricity) to produce the propulsion to drive the wheels

Radiator

Represents the radiator in the car

- Function:
    - Coolant_Flow(): Handles the flow of coolant into the radiator and out of the radiator.

TemperatureGauge

Represents the temperature gauge that measures the engine temperature

- Functions:
    - Check_Temp_Periodically(): Periodically check the engine temperature
    - Send_Temperature(): Returns the temperature the gauge obtains

CoolantTank: AbstractClassState

Represents the coolant tank in the cooling system of a car.

- Functions:
    - Send_Coolant(): Sends coolant out to the engine

FrontDriveShaft

Represents the front drive shaft in a car.

- Variables
    - bool poweredByEngine : True if the engine is powering this driveshaft. False if it is not.
- Functions:
    - Send_Power(): Sends power to the wheels
    - getPoweredByEngine(): returns the value of the poweredByEngine variable
    - setPoweredByEngine(): sets the value of the poweredByEngine variable

RearDriveShaft

Represents the rear drive shaft in a car.

- Variables
    - bool poweredByEngine : True if the engine is powering this driveshaft. False if it is not.
- Functions:
    - Send_Power(): Sends power to the wheels
    - getPoweredByEngine(): returns the value of the poweredByEngine variable
    - setPoweredByEngine(): sets the value of the poweredByEngine variable

FrontWheels:AbstractClassDirection

Represents the front wheels in a car. Inherits from AbstractClassDirection

- Functions:
  - Rotate_Wheels(): Rotates the wheels
  - Turn_Wheels_Left(): Angles the wheel to the left
  - Turn_Wheels_Right(): Angles the wheels to the right
  - Stop_Wheels(): Reduce the speed of the wheels to 0
  - Move_Wheels(): Speed up the movement of the wheels
  - Slow_Down(): Reduce the speed of the wheels

RearWheels

Represents the rear wheels in a car

- Functions:
  - Rotate_Wheels(): Rotates the wheels
  - Stop_Wheels(): Reduce the speed of the wheels to 0
  - Move_Back_Wheels(): Speed up the movement of the wheels
  - Slow_Down(): Reduce the speed of the wheels
  - Lock_Wheels(): Lock the rear wheels when parking brake is engaged
  - Unlock_Wheels(): Unlock the rear wheels when the parking brake is disengaged

SteeringWheel:AbstractClassDirection

Represents the steering wheel in the car. Inherits from the AbstractClassDirection

- Functions:
  - Left_Steer_Func(): Turns the steering wheel to the left. Will turn the steering rod to the left as well.
  - Right_Steer_Function(): Turns the steering wheel to the right. Will turn the steering rod to the right.

SteeringRod

Represents the steering rod in the car.

- Functions:
  - Turn_Rod_Left(): Turn the steering rod to the left. Will cause the steering rack to move to the right
  - Turn_Rod_Right(): Turns the steering rod to the right. Will cause the steering rack to move to the left

SteeringRack

Represents the steering rack in the car.

- Functions:
    - Turn_Rack_Left(): Turn the steering rack to the left, allowing the car to turn to the right
    - Turn_Rack_Right (): Turns the steering rack to the right, allowing the car to turn to the left

BrakePump:AbstractClassState

Represents the brake pump, as part of the braking system in a car. Inherits from the AbstractClassState.

- Functions:
    - Send_Brake_Fluid(): Pumps brake fluid to the brake system

Brakes

Represents the brakes in the car

- Functions:
    - Send_Brake_Signal(): Sends out signal that brakes are being engaged
    - Hybrid_Brakes_Func(): Run the regenerative version of the braking system for hybrid cars
    - Electric_Brakes_Func(): Run the regenerative version of the braking system for electric cars (higher regenerative rate)

Handbrakes:AbstractClassState

Represents the handbrake inside the car interior. Inherits from AbstractClassState.

- Functions:
    - Engage_Handbrake(): Engages the parking brake in the car
    - Disengage_ Handbrake(): Disengage the parking brake in the car

ParkingBrakes

Represents the parking brakes located in the braking system of the car

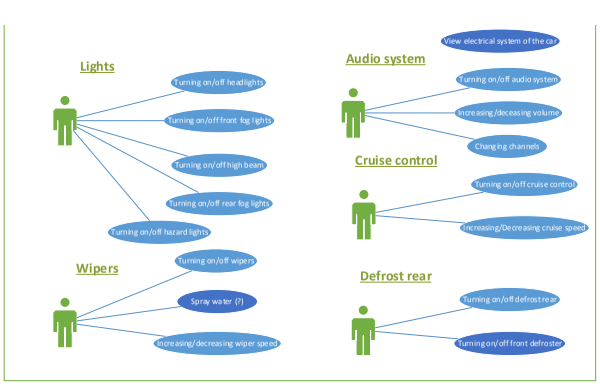
- Functions
    - SendSignal(): Sends out a signal that parking brakes are being engaged
    - GivePower(): Sends out power to the brakes in the rear wheels to lock them
    - StopPower(): Stops sending power to the brakes in the rear wheels hence unlocking them

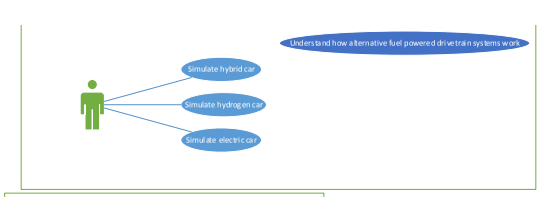
Use Cases

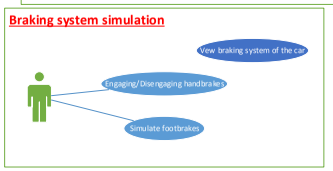
Over the last few weeks of discussion and brainstorming the functionalities of our application, we have come up with a number of use cases that should be present in the application we are developing.

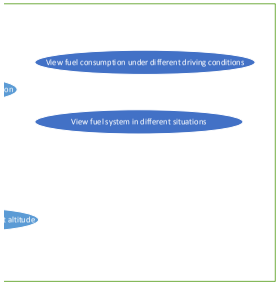
Use case diagrams help as a quick checklist to ensure all functionalities have been implemented. In addition, the use case descriptions also assist in the development process as they act as a step-by-step guide on how each use case should run. This would help in the development process as it acts like pathway that a user needs to take to do something, hence it becomes like pathway on how a function should work.

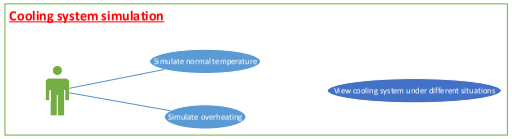
The following pages show the use case diagram for our application, in addition to the use case descriptions. As this is a draft manual, there are some changes that can be expected to this section, as the development of the application is under way.

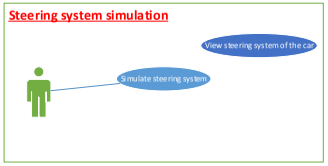


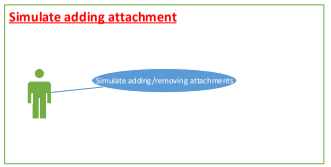




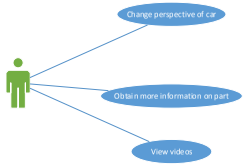








Infographics



**All actors represent USERS.

Use Case Descriptions

Infographics of car

Name: Change perspective of car	**ID:** 1
Stakeholders and Goals: User – to view different parts of the car	
Description: A user wants to view different parts of the car	
Actors: User	
Trigger: User runs the application and wishes to view different parts of the car.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Infographics" button. Upon clicking on Infographics, the user will be brought to the infographics page where the user will be able to view the picture from the corner perspective (the default view). 2. To change to a different perspective (i.e to see a different part of a car), the user will then click on an arrow button which will be located above the description box, either to the left or to the right. If the user wishes to zoom in a perspective the up and down buttons can be used. The user will be able to view different components from different perspectives of the car. Different components will surround the picture of the different perspective of the car (View user manual for an example of interface). 3. Step 2 is repeated if the user wishes to see another perspective of the car. 4. End	
Sub-Flows: None	
Alternative/Exceptional Flows: None	

Name: Obtain more information on part	**ID:** 2
Stakeholders and Goals: User – to get more information on a component	
Description: A user wants to know more about a specific component of the car	
Actors: User	
Trigger: User runs the application and wants to know more about a specific part of the car	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Infographics" button. Upon clicking on Infographics, the user will be brought to the infographics page where the user will be able to view the corner perspective of the car (the default view). 2. The user will choose a perspective of the car he/she wishes to view. 3. Upon selecting a perspective that users want (except the corner perspective), to obtain more information on a particular component, the user clicks on the name of the component. The user can also click on the part on the picture. 4. The system will then display the description of the component on the description box. 5. Step 2 to 4 are repeated if the user wishes to see another component of the car. 6. Upon completion of this use case (Use Case ID: 3), the next use case **can/will** be triggered (**The reason behind "can/will" is due to the ability of the user to choose if he/she wants to run the use case or not).	

Sub-Flows: None
Alternative/Exceptional Flows: None

Name: View videos	**ID:** 3
Stakeholders and Goals: User – to view videos on a component	
Description: A user wants to gain more information on a component by viewing a video prepared by the developers.	
Actors: User	
Trigger: User runs the application and wishes to view a video on a specific component of the car	
Normal Flow: 1. This use case is continued from the previous use case. Under the description box, the user will be able to click on a video link at the end. 2. The system will then pop up an overlay with the video but will not start playing the video 3. The user clicks "Play". 4. The system starts playing the video. 5. Upon finishing the video or in the middle of watching the video, the user will be able to click the outer surroundings of the video overlay to go back to the Infographics screen. 6. If the user wishes to view a video for a different component, the user will have to run the previous use case again (Use Case ID: 2). 7. End	
Sub-Flows: None	
Alternative/Exceptional Flows: (IF overlay cannot be done on the software itself) 2. The system will then prompt the user if he/she wishes to launch his/her default media player. 3. The user enters his/her input 4. The video will be played in an external player and upon completion the media player will be closed and the user will be brought back to the Infographics page of the system.	

Simulation of car

1. **Persistent dashboard**

Name: Start/Stop engine	**ID:** 4
Stakeholders and Goals: User – to start/stop the engine	
Description: A user wants to start or stop an engine	
Actors: User	
Trigger: User runs the application and wishes to start or stop the engine	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 3. The system will show that the engine has started, by showing different components being lit up on the dashboard. 4. End	
Sub-Flows: None	
Alternative/Exceptional Flows: (The user wishes to turn off the engine instead) 2. On the persistent dashboard the user will be able to see the "Stop Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to stop the engine. 3. The system will show that the engine has stopped, by showing different components being dim down on the dashboard. (The user wishes to turn off the engine while the car is still moving) 2. On the persistent dashboard the user will be able to see the "Stop Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to stop the engine. 3. The system will provide a warning to the user that the car is still moving and the engine cannot be stopped until the car has been turned off	

Name: Increase/Decrease incline	**ID:** 5
Stakeholders and Goals: User – to increase/decrease the incline	
Description: A user wants to increase or decrease the degree of incline of the car.	
Actors: User	
Trigger: User runs the application and wishes to increase/decrease the incline	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 3. The system will show that the engine has started, by showing different	

components being lit up on the dashboard.
4. To increase the degree of incline, the user simply clicks on the "+" symbol under the "Incline" section.
5. The system will then display an increase in the degree of incline.
6. End

Sub-Flows: None

Alternative/Exceptional Flows:
(The user is trying to decrease the degree of incline)
4. To decrease the degree of incline, the user simply clicks on the "-" symbol under the "Incline" section.
5. The system will then display a decrease in the degree of incline.

Name: Increase/Decrease speed	**ID:** 6

Stakeholders and Goals: User – to increase/decrease the speed

Description: A user wants to increase or decrease the speed of the car.

Actors: User

Trigger: User runs the application and wishes to increase/decrease the speed of the car.

Normal Flow:
1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
2. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
3. The system will show that the engine has started, by showing different components being lit up on the dashboard.
4. To increase the speed of the car, the user simply clicks the "+" symbol under the "Speed" section.
5. The system will then display an increase in speed of the car on the dashboard.
6. End

Sub-Flows: None

Alternative/Exceptional Flows:
(The user is trying to decrease the degree of incline)
4. To decrease the speed of the car, the user simply clicks on the "-" symbol under the "Speed" section.
5. The system will then display a decrease in the speed of the car on the dashboard.

Name: Increase/Decrease altitude	**ID:** 7

Stakeholders and Goals: User – to increase/decrease the altitude

Description: A user wants to increase or decrease the altitude that the car is currently at.

Actors: User

Trigger: User runs the application and wishes to increase/decrease the altitude that the car is currently at.

Normal Flow:

1. User reaches the main page of the system and clicks on the “Simulation” button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
2. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
3. The system will show that the engine has started, by showing different components being lit up on the dashboard.
4. To increase the altitude that the car is currently at, the user simply clicks the “+” symbol under the “Altitude” section.
5. The system will then display an increase in altitude that the car is currently at.
6. End

Sub-Flows: None

Alternative/Exceptional Flows:
(The user is trying to decrease the degree of incline)

4. To decrease the altitude that the car is currently at, the user simply clicks on the “-” symbol under the “Altitude” section.
5. The system will then display a decrease in altitude that the car is currently at.

Name: View car’s temperature, fuel and speed	**ID:** 8
Stakeholders and Goals: User – to view the car temperature, fuel and speed	
Description: A user wants to view the temperature, fuel level and speed of the car.	
Actors: User	
Trigger: User runs the application and wishes to view the current car temperature, fuel level and speed.	

Normal Flow:

1. User reaches the main page of the system and clicks on the “Simulation” button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
2. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
3. The system will show that the engine has started, by showing different components being lit up on the dashboard.
4. The user will also be able to see on the car dashboard, from the left and going in order, the first values represent the fuel level of the car, the second value represent the speed of the car and the final values represent the temperature of the car.
5. End

Sub-Flows: None

Alternative/Exceptional Flows: None

2. Fuel system simulation

Name: Simulate normal consumption	**ID:** 9
Stakeholders and Goals: User – to simulate normal consumption of fuel of the car	
Description: A user wants to view the consumption of fuel of the car under normal circumstances.	
Actors: User	
Trigger: User runs the application and wishes to view the normal consumption of fuel of the car.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Fuel system". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the fuel tank, fuel pump and the engine. There will also be a few buttons available for the user which includes "Simulate wrong fuel", "Simulate filling up fuel" and "Simulate low fuel". 3. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 4. The system will show that the engine has started, by showing different components being lit up on the dashboard. 5. The user will then be able to see the fuel flowing from the fuel tank to the fuel pump and to the engine. 6. End	
Sub-Flows: None	
Alternative/Exceptional Flows: None	

Name: Simulate low fuel	**ID:** 10
Stakeholders and Goals: User – to simulate low fuel	
Description: A user wants to view how the car responds when the fuel level is low.	
Actors: User	
Trigger: User runs the application and wishes to view the response of the car when the level of fuel is low.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Fuel system". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the fuel tank, fuel pump and the engine. There will also be a few buttons available for the user which includes "Simulate wrong fuel", "Simulate filling up fuel"	

and “Simulate low fuel”.
3. The user will then click on “Simulate low level” button in the simulation window.
4. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
5. The system will show that the engine has started, by showing different components being lit up on the dashboard.
6. The user will then be able to see the fuel flowing from the fuel tank to the fuel pump and to the engine.
7. The system will then light up the fuel symbol on the persistent dashboard. The fuel symbol lighting up on the dashboard represents the fuel level is low.
8. End

Sub-Flows: None

Alternative/Exceptional Flows: None

Name: Simulate wrong fuel	**ID:** 11

Stakeholders and Goals: User – to simulate wrong fuel

Description: A user wants to view the effects of adding the wrong fuel into the car.

Actors: User

Trigger: User runs the application and wishes to simulate wrong fuel being added to the car.

Normal Flow:

1. User reaches the main page of the system and clicks on the “Simulation” button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
2. On the list of simulations available to the user, the user will then click on “Fuel system”. The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the fuel tank, fuel pump and the engine. There will also be a few buttons available for the user which includes “Simulate wrong fuel”, “Simulate filling up fuel” and “Simulate low fuel”.
3. The user then clicks on “Simulate wrong fuel” on the simulation window.
4. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
5. The system will show that the engine has started, by showing different components being lit up on the dashboard.
6. The user will then be able to see the fuel flowing from the fuel tank to the fuel pump and to the engine.
7. After 3 seconds, the system will then shut off the engine, by turning off the lights on the persistent dashboard. This is to show that the engine has broken down from adding the wrong fuel into the car.
8. End

Sub-Flows: None

Alternative/Exceptional Flows: None

Name: Simulate fuel consumption in different altitude	**ID:** 12
Stakeholders and Goals: User – to simulate fuel consumption in different altitude	
Description: A user wants to view the difference in the rate of fuel consumption in different altitudes.	
Actors: User	
Trigger: User runs the application and wishes to simulate fuel consumption in different altitude.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Fuel system". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the fuel tank, fuel pump and the engine. There will also be a few buttons available for the user which includes "Simulate wrong fuel", "Simulate filling up fuel" and "Simulate low fuel". 3. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 4. The system will show that the engine has started, by showing different components being lit up on the dashboard. 5. The user will then be able to see the fuel flowing from the fuel tank to the fuel pump and to the engine. 6. The user will then increase the current altitude by clicking on the "+" symbol under the "Altitude" section (This is closely related to the increase/decrease altitude use case, Use case ID: 7). 7. The system will then show the user the increased rate of consumption of fuel by showing more arrows flowing from the fuel tank, to the fuel pump and to the engine. 8. End	
Sub-Flows: None	
Alternative/Exceptional Flows: (The user decides to decrease the current altitude instead) 6. The user will then decrease the current altitude by clicking on the "-" symbol under the "Altitude" section (This is closely related to the increase/decrease altitude use case, Use case ID: 7). 7. The system will then show the user the decreased rate of consumption of fuel by showing less arrows flowing from the fuel tank, to the fuel pump and to the engine.	

Name: Simulate fuel consumption in different incline	**ID:** 13
Stakeholders and Goals: User – to simulate fuel consumption in different degree of incline	

Description: A user wants to view the difference in the rate of fuel consumption in different degrees of incline.
Actors: User
Trigger: User runs the application and wishes to simulate fuel consumption in different degrees of incline
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Fuel system". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the fuel tank, fuel pump and the engine. There will also be a few buttons available for the user which includes "Simulate wrong fuel", "Simulate filling up fuel" and "Simulate low fuel". 3. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 4. The system will show that the engine has started, by showing different components being lit up on the dashboard. 5. The user will then be able to see the fuel flowing from the fuel tank to the fuel pump and to the engine. 6. The user will then increase the degree of incline by clicking on the "+" symbol under the "Incline" section (This is closely related to the increase/decrease incline use case, Use case ID: 5). 7. The system will then show the user the increased rate of consumption of fuel by showing more arrows flowing from the fuel tank, to the fuel pump and to the engine. 8. End
Sub-Flows: None
Alternative/Exceptional Flows: (The user decides to decrease the current degree of incline instead) 6. The user will then decrease the degree of incline by clicking on the "-" symbol under the "Incline" section (This is closely related to the increase/decrease altitude use case, Use case ID: 5). 7. The system will then show the user the decreased rate of consumption of fuel by showing less arrows flowing from the fuel tank, to the fuel pump and to the engine.

Name: Simulate filling up fuel	**ID:** 14
Stakeholders and Goals: User – to simulate filling up fuel	
Description: A user wants to view the process of filling up fuel for the car	
Actors: User	
Trigger: User runs the application and wishes to simulate filling up fuel for the car.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation"	

button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
2. On the list of simulations available to the user, the user will then click on "Fuel system". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the fuel tank, fuel pump and the engine. There will also be a few buttons available for the user which includes "Simulate wrong fuel", "Simulate filling up fuel" and "Simulate low fuel".
3. The user then clicks on "Simulate filling up fuel" on the simulation window.
4. The system will then show arrows pointing upwards on the fuel tank to simulate the addition of fuel into the car (i.e the level of petrol in the car is increasing).
5. End

Sub-Flows: None

Alternative/Exceptional Flows:
(User tries to fill up fuel when the car is NOT turned off, car is turned off when the engine is not started)

4. The system will then display to the user that the car should first be turned off before filling up fuel for the car.
5. On the persistent dashboard the user will be able to see the "Stop Engine" button. The user will then click on it to stop the engine.
6. The user then clicks on "Simulate filling up fuel" on the simulation window.
7. The system will then show arrows pointing upwards on the fuel tank to simulate the addition of fuel into the car (i.e the level of petrol in the car is increasing).

3. Wheel drive system simulation

Name: View difference among different wheel drive system	**ID:** 15
Stakeholders and Goals: User – to view the difference among different wheel drive system	
Description: A user wants to understand the difference between four wheel drive, rear wheel drive and front wheel drive	
Actors: User	
Trigger: User runs the application and wishes to view difference among different wheel drive system	

Normal Flow:

6. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
7. The user clicks on "Wheel drive system". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then able to see different components on the screen which consists of the Engine, Axle and Wheels. There will also be 3 buttons available for the

user to click which is “4 wheel drive”, “Front wheel drive (2 wheel drive)” and “Rear wheel drive (2 wheel drive). 8. The user clicks on the “4 wheel drive” button. 9. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 10. The system will show that the engine has started, by showing different components being lit up on the dashboard. 11. The user will then be able to see the power of the engine flowing towards the front and back axle of the car and towards the wheels. 12. End
Sub-Flows: None
Alternative/Exceptional Flows: (The user wishes to view a front wheel drive car instead) 8. The user clicks on “Front wheel drive (2 wheel drive)” button. 9. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 10. The system will show that the engine has started, by showing different components being lit up on the dashboard. 11. The user will then be able to see the power of the engine flowing only flowing towards the front axle of the car and towards the wheels. (The user wishes to view a rear wheel drive car instead) 8. The user clicks on the “Rear wheel drive (2 wheel drive)” button. 9. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 10. The system will show that the engine has started, by showing different components being lit up on the dashboard. 11. The user will then be able to see the power of the engine flowing only flowing towards the back axle of the car and towards the wheels.

4. Cooling system simulation

Name: Simulate normal temperature	**ID:** 16
Stakeholders and Goals: User – to simulate cooling system under normal temperature	
Description: A user wants to simulate the cooling system of the car under normal temperature	
Actors: User	
Trigger: User runs the application and wishes to view the cooling system under normal temperature.	
Normal Flow: 1. User reaches the main page of the system and clicks on the “Simulation” button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on “Cooling system”. The window (i.e the simulation window) below the	

persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the radiator, coolant tank and the engine. There will also be a few buttons available for the user which includes “Simulate normal temperature” and “Simulate high temperature”.
3. The user will then click on “Simulate normal temperature” button in the simulation window.
4. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
5. The system will show that the engine has started, by showing different components being lit up on the dashboard.
6. The system will then show the user the flow of the coolant from the coolant tank to the engine and to the radiator in a continuous loop.
7. The system will also be updating the temperature of the car dynamically.
8. End

Sub-Flows: None

Alternative/Exceptional Flows: None

Name: Simulate overheating (high temperature)	**ID:** 17

Stakeholders and Goals: User – to simulate cooling system overheating or under high temperature

Description: A user wants to simulate the cooling system of the car under high temperature and overheating

Actors: User

Trigger: User runs the application and wishes to view the cooling system under high temperature.

Normal Flow:

1. User reaches the main page of the system and clicks on the “Simulation” button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
2. On the list of simulations available to the user, the user will then click on “Cooling system”. The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view different components on the screen, which includes the radiator, coolant tank and the engine. There will also be a few buttons available for the user which includes “Simulate normal temperature” and “Simulate high temperature”.
3. The user will then click on “Simulate high temperature” button in the simulation window.
4. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
5. The system will show that the engine has started, by showing different components being lit up on the dashboard.
6. The system will then show the user the flow of the coolant from the coolant tank to the engine and to the radiator in a continuous loop (The rate that the coolant is flowing though the different components would be

much slower than the rate of flow of coolant of the previous use case, "Simulate normal temperature", use case ID: 16). 7. The system will also be updating the temperature of the car dynamically until it reaches 140 degrees Celsius. 8. Within the next 5 seconds, the system will then display to the user that the car is overheating and the engine will shut down. 9. After 5 seconds, the system will then shut off the engine, by turning off the lights on the persistent dashboard. The only lights on the dashboard that will remain on is the **check engine light** and the **high temperature light**. This is to show that the engine has overheated. 10. End
Sub-Flows: None
Alternative/Exceptional Flows: None

5. Electronic system simulation

i. Lights

Name: Turning on/off headlights	**ID:** 18
Stakeholders and Goals: User – to simulate turning on/off the headlights	
Description: A user wants to simulate turning on/off the headlights	
Actors: User	
Trigger: User runs the application and wishes to turn on/off the headlights.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Electrical System". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the electrical components of the car. There will also be a few buttons available for the user which includes "Lights", "Audio", "Wipers", "Cruise control" and "Defrost rear". 3. The user will then click on "Lights" button in the simulation window. 4. The system will then display to the user the different kinds of light available by showing more buttons to the user, which includes "Head lights", "High beam", "Front fog light", "Rear fog light" and "Hazard light". 5. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 6. The system will show that the engine has started, by showing different components being lit up on the dashboard. 7. The user will then click on the "Headlights on" button. 8. The system will then show the flow of power from the battery to the headlights. 9. The system will also update the persistent dashboard to light up the **headlights** symbol. 10. **IF** the user wishes to turn on the other lights (i.e high beam, front fog light or the rear fog lights), it will continue from this point of this use case.	

Sub-Flows: None

Alternative/Exceptional Flows:

(User wishes to turn off the headlights instead, we are assuming the headlights are already on)

7. The user will then click on the "Headlights off" button.
8. The system will then stop the flow of power from the battery to the headlights.
9. The system will also update the persistent dashboard to turn off the headlights symbol.
10. At the same time, if there are any other lights on (e.g high beam, front fog light and rear fog light), the system will also turn them off.

Name: Turning on/off high beam	**ID:** 19

Stakeholders and Goals: User – to simulate turning on/off high beam lights

Description: A user wants to simulate turning on/off the high beam lights

Actors: User

Trigger: User runs the application and wishes to turn on/off the high beam lights

Normal Flow:

1. This use case is continued from a use case detailed before, "Turning on/off headlights", use case ID: 18. We continue from point number 9 of the normal flow of the use case. At this point, the headlights are already ON. The reason why this use case is continued from before is because the user must turn on the headlights first before being able to turn on the other lights (i.e high beam, front fog light or the rear fog lights).
2. The user will then click on the "High beam on" button.
3. The system will then show the flow of power from the battery to the high beam lights
4. The system will also update the persistent dashboard to light up the **high beam light** symbol.
5. End

Sub-Flows: None

Alternative/Exceptional Flows:

(User wishes to turn off the high beam lights instead, we are assuming the high beam lights are already on)

2. The user will then click on the "High beam off" button.
3. The system will then stop the flow of power from the battery to the high beam light.
4. The system will also update the persistent dashboard to turn off the high beam light symbol.

(User HOLDS on high beam lights while the headlights are off. In a car, you usually use the high beam at times to provide a signal to another driver (e.g to move away, or allowing the driver to make a turn)

2. The user will HOLD on the "High beam on" button.
3. The system will then show the flow of power from the battery to the high beam lights.
4. The system will also update the persistent dashboard to light up the high beam light symbol
5. Once the user stops holding the "High beam on" button, the system will

then stop the flow of power from the battery to the high beam lights.
6. The system will also update the persistent dashboard to turn off the high beam light symbol.

(User attempts to turn on the high beam lights without having the headlights turned on)

1. The user attempts to click on the "High beam on" button.
2. The system displays to the user that the headlights should be turned on first before the high beam light can be turned on.
3. The process of turning on the high beam lights are detailed in use case "Turning on/off headlights (Use case ID: 18)" and "Turning on/off high beam (Use case ID: 19)".

Name: Turning on/off front fog lights	**ID:** 20
Stakeholders and Goals: User – to simulate turning on/off front fog lights	
Description: A user wants to simulate turning on or off the front fog lights of the car	
Actors: User	
Trigger: User runs the application and wishes to turn on/off the front fog light.	

Normal Flow:

1. This use case is continued from a use case detailed before, "Turning on/off headlights", use case ID: 18. We continue from point number 9 of the normal flow of the use case. At this point, the headlights are already ON. The reason why this use case is continued from before is because the user must turn on the headlights first before being able to turn on the other lights (i.e high beam, front fog light or the rear fog lights).
2. The user will then click on the "Front fog light on" button.
3. The system will then show the flow of power from the battery to the front fog light
4. The system will also update the persistent dashboard to light up the **front fog light** symbol.
5. End

Sub-Flows: None

Alternative/Exceptional Flows:

(User wishes to turn off the front fog light instead, we are assuming that the front fog lights are already turned on)

1. The user will then click on the "Front fog light off" button.
2. The system will then stop the flow of power from the battery to the front fog light.
3. The system will also update the persistent dashboard to turn off the front fog light symbol.

(User attempts to turn on the front fog light without having the headlights turned on)

1. The user attempts to click on the "Front fog light on" button.
2. The system displays to the user that the headlights should be turned on first before the front fog light can be turned on.
3. The process of turning on the front fog lights are detailed in use case "Turning on/off headlights (Use case ID: 18)" and "Turning on/off front fog lights (Use case ID: 20)".

Name: Turning on/off rear fog lights	**ID:** 21
Stakeholders and Goals: User – to simulate turning on/off rear fog lights	
Description: A user wants to simulate turning on or off the rear fog lights of the car	
Actors: User	
Trigger: User runs the application and wishes to turn on/off the rear fog light.	
Normal Flow: 6. This use case is continued from a use case detailed before, "Turning on/off headlights", use case ID: 18. We continue from point number 9 of the normal flow of the use case. At this point, the headlights are already ON. The reason why this use case is continued from before is because the user must turn on the headlights first before being able to turn on the other lights (i.e high beam, front fog light or the rear fog lights). 7. The user will then click on the "Rear fog light on" button. 8. The system will then show the flow of power from the battery to the rear fog light 9. The system will also update the persistent dashboard to light up the **rear fog light** symbol. 10. End	
Sub-Flows: None	
Alternative/Exceptional Flows: (User wishes to turn off the rear fog light instead, we are assuming that the rear fog lights are already turned on) 4. The user will then click on the "Rear fog light off" button. 5. The system will then stop the flow of power from the battery to the rear fog light. 6. The system will also update the persistent dashboard to turn off the rear fog light symbol. (User attempts to turn on the rear fog light without having the headlights turned on) 4. The user attempts to click on the "Rear fog light on" button. 5. The system displays to the user that the headlights should be turned on first before the rear fog light can be turned on. 6. The process of turning on the rear fog lights are detailed in use case "Turning on/off headlights (Use case ID: 18)" and "Turning on/off rear fog lights (Use case ID: 21)".	

Name: Turning on/off hazard lights	**ID:** 22
Stakeholders and Goals: User – to simulate turning on/off hazard lights	
Description: A user wants to simulate turning on or off the hazard lights of the car	
Actors: User	
Trigger: User runs the application and wishes to turn on/off the hazard light.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on	

"Electrical System". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the electrical components of the car. There will also be a few buttons available for the user which includes "Lights", "Audio", "Wipers", "Cruise control" and "Defrost rear". 3. The user will then click on "Lights" button in the simulation window. 4. The system will then display to the user the different kinds of light available by showing more buttons to the user, which includes "Head lights", "High beam", "Front fog light", "Rear fog light" and "Hazard light". 5. The user will then click on the "Hazard light on" button. 6. The system will then show the flow of power from the battery to the hazard lights 7. The system will also update the persistent dashboard to light up the hazard light symbol. 8. End
Sub-Flows: None
Alternative/Exceptional Flows: (User wishes to turn off the hazard lights instead, we are assuming that the hazard lights are already turned on) 7. The user will then click on the "Hazard light off" button. 8. The system will then stop the flow of power from the battery to the hazard lights. 9. The system will also update the persistent dashboard to turn off the hazard light symbol.

ii. Wipers

Name: Turning on/off wipers	**ID:** 23
Stakeholders and Goals: User – to simulate turning on/off wipers	
Description: A user wants to simulate turning on or off the wipers of the car	
Actors: User	
Trigger: User runs the application and wishes to turn on/off the wipers	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Electrical System". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the electrical components of the car. There will also be a few buttons available for the user which includes "Lights", "Audio", "Wipers", "Cruise control" and "Defrost rear". 3. The user will then click on "Wiper" button in the simulation window. 4. The system will then display to the user the "Turn wiper on" button (what user see may differ depending on the status of the wiper) and the speed bar (for changing the speed of the wiper, similar to the bar we use to increase/decrease altitude/speed/incline). 5. The user will then click on the "Turn wiper on" button.	

6. The system will then show the flow of power from the battery to the wipers to show that the wipers are on
7. The system will also tell the user that the wiper has been turned on by displaying "Wiper is now on!" (How? In "functionality pathway" document, mentioned something called message centre).
8. **IF** the user wishes to increase/decrease the speed of the wipers, it is continued in the next use case "Increasing/Decreasing wiper speed", Use case ID: 24.

Sub-Flows: None

Alternative/Exceptional Flows:

(User wishes to turn off the wipers instead, we are assuming that the wipers are already turned on)

5. The user will then click on the "Wipers off" button.
6. The system will then stop the flow of power from the battery to the wipers.
7. The system will also tell the user that the wiper has been turned off by displaying "Wiper is now off!"

Name: Increasing/decreasing wiper speed	**ID:** 24

Stakeholders and Goals: User – to simulate increase/decrease of wiper speed

Description: A user wants to simulate increasing or decreasing the speed of the wiper

Actors: User

Trigger: User runs the application and wishes to increase/decrease the wiper speed

Normal Flow:

1. This use case is continued from a use case detailed before, "Turning on/off wipers", use case ID: 23. We continue from point number 7 of the normal flow of the use case. At this point, the wipers are already ON. The reason why this use case is continued from before is because the user must turn on the wipers first before being able to increase/decrease the speed of the wipers
2. The user will then increase the speed of the wiper by clicking on the "+" symbol on the speed bar.
3. The system will then show the increase of flow of power from the battery to the wiper by increasing the number of arrows to the flow.
4. The system will also tell the user that the speed of the wiper has been successfully changed via the message centre.
5. End

Sub-Flows: None

Alternative/Exceptional Flows:

(User wishes to reduce the speed of the wipers)

1. The user will reduce the speed of the wiper by clicking on the "-" symbol on the speed bar.
2. The system will then show the decrease of flow of power from the battery to the wiper by reducing the number of arrows to the flow.
3. The system will also tell the user that the speed of the wiper has been successfully changed via the message centre.

(User attempts to increase/decrease the speed of the wipers without having the

wipers on)

4. The user attempts to click on the "+"/"-" on the speed bar.
5. The system displays to the user that the wipers should be turned on first before being able to increase/decrease the speed of the wipers.
6. The process of increasing/decreasing the speed of the wipers are detailed in use case "Turning on/off wipers (Use case ID: 23)" and "Increasing/decreasing wiper speed (Use case ID: 24)".

iii. Audio system

Name: Turning on/off audio system	**ID:** 25
Stakeholders and Goals: User – to simulate turning on/off the audio system	
Description: A user wants to simulate turning on/off the audio system	
Actors: User	
Trigger: User runs the application and wishes to turn on/off the audio system.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Electrical System". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the electrical components of the car. There will also be a few buttons available for the user which includes "Lights", "Audio", "Wipers", "Cruise control" and "Defrost rear". 3. The user will then click on "Audio" button in the simulation window. 4. The system will then display to the user the "Radio on" button, the lists of channel (list of channel is just a list of numbers representing different channels, 6 in total) and volume bar (the volume bar is similar to the bar we use to increase/decrease incline/speed/altitude). 5. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 6. The system will show that the engine has started, by showing different components being lit up on the dashboard. 7. The user will then click on the "Radio on" button. 8. The system will then show the flow of power from the battery to the radio and also to the speakers around the car (4 in total, 1 on each door). 9. The system will also update the user the current channel that it is currently playing and followed by the current volume in the **message centre**. 10. **IF** the user wishes to increase/decrease the volume of the radio or change the channels, it is listed in the next 2 use cases, use case "Increasing/decreasing volume (Use case ID: 26)" and use case "Changing channels (Use Case ID: 27)".	
Sub-Flows: None	
Alternative/Exceptional Flows: (User wishes to turn off the radio instead, we are assuming the radio is already	

on)

7. The user will then click on the “Radio off” button.
8. The system will then stop the flow of power from the battery to the radio and also to the speakers around the car.

Name: Increasing/decreasing volume	**ID:** 26
Stakeholders and Goals: User – to increase/decrease the volume	
Description: A user wants to simulate increasing or decreasing the volume of the radio.	
Actors: User	
Trigger: User turns on the radio and wishes to increase/decrease the radio volume.	

Normal Flow:

1. This use case is continued from a use case detailed before, “Turning on/off audio system”, use case ID: 25. We continue from point number 9 of the normal flow of the use case. At this point, the radio is already ON. The reason why this use case is continued from before is because the user must turn on the radio first before being able to increase/decrease the volume of the radio.
2. The user will then increase the volume of the radio by clicking on the “+” symbol on the volume bar.
3. The system will then represent the increase in volume by increasing the number of arrows/flows from the battery to the speakers around the car.
4. The system will once again update the user the current channel that it is currently playing and followed by the increased volume level in the message centre.
5. End

Sub-Flows: None

Alternative/Exceptional Flows:

(User wishes to reduce the volume)

2. The user will then decrease the volume of the radio by clicking on the “-” symbol on the volume bar.
3. The system will then represent the decrease in volume by decreasing the number of arrows/flows from the battery to the speakers around the car.
4. The system will once again update the user the current channel that it is currently playing and followed by the decreased volume level in the message centre.

(User attempts to increase/decrease the volume of the radio without having the radio on)

1. The user attempts to click on the “+”/“-” on the volume bar.
2. The system displays to the user that the radio should be turned on first before being able to increase/decrease the volume of the radio.
3. The process of increasing/decreasing the volume of the radio is detailed in use case “Turning on/off radio (Use case ID: 25)” and “Increasing/decreasing wiper speed (Use case ID: 26)”.

Name: Changing channels	**ID:** 27
Stakeholders and Goals: User – to change channels	
Description: A user wants to simulate changing the channels on the radio	

Actors: User
Trigger: User turns on the radio and wishes to change the channel
Normal Flow: 1. This use case is continued from a use case detailed before, "Turning on/off audio system", use case ID: 25. We continue from point number 9 of the normal flow of the use case. At this point, the radio is already ON. The reason why this use case is continued from before is because the user must turn on the radio first before being able to change the channels on the radio. 2. The user will then change the channel by clicking on a different number among the lists of channel available for the user. 3. The system will once again update the user the current channel (should be updated to the one user has recently changed) that it is currently playing and followed by the current volume in the message centre. 4. End
Sub-Flows: None
Alternative/Exceptional Flows: (User attempts to change channels on the radio without having the radio on) 4. The user attempts to click on one of the numbers on the list of channels. 5. The system displays to the user that the radio should be turned on first before being able to change the channels on the radio. 6. The process of changing channels on the radio is detailed in use case "Turning on/off radio (Use case ID: 25)" and "Changing channels (Use case ID: 27)".

iv. Cruise control

Name: Turning on/off cruise control	**ID:** 28
Stakeholders and Goals: User – to simulate turning on/off cruise control	
Description: A user wants to simulate turning on/off cruise control in the car.	
Actors: User	
Trigger: User runs the application and wishes to turn on/off cruise control.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Electrical System". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the electrical components of the car. There will also be a few buttons available for the user which includes "Lights", "Audio", "Wipers", "Cruise control" and "Defrost rear". 3. The user will then click on "Cruise control" button in the simulation window. 4. The system will then display to the user the "Cruise on" button and cruise speed bar (the cruise speed bar is similar to the bar we use to increase/decrease incline/speed/altitude). 5. In addition to the electrical components of the car (only for cruise	

control), the user will also be able to see additional components which includes the engine, the fuel tank and also the fuel pump.

6. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
7. The system will show that the engine has started, by showing different components being lit up on the dashboard.
8. The user will also be able to see fuel being pumped from the fuel tank to the engine by the fuel pump.
9. To be able to use turn on the cruise control the user needs to increase the speed of the car to **at least 45km/h**. To do so the user simply clicks the "+" symbol under the "Speed" section (This is on the persistent dashboard) until he/she has reached 45km/h.
10. The system will then display an increase in speed of the car on the dashboard.
11. The user will then click on the "Cruise on" button.
12. The system will then update the persistent dashboard to display the cruise control symbol.
13. The system will also show flow of power/arrows from the battery to the cruise control module.
14. **IF** the user wishes to increase/decrease the cruise speed, it will be detailed in the next use case "Increase/Decrease cruise speed (Use case ID: 29)".

Sub-Flows: None

Alternative/Exceptional Flows:

(User wishes to turn off the cruise control instead, we are assuming the cruise control is already on)

11. The user will then click on the "Cruise control off" button.
12. The system will then update the persistent dashboard to turn off the cruise control symbol.
13. The system will then stop the flow of power from the battery to the cruise control module.

(User increases the degree of incline while the cruise control is on)

14. The user will increase the incline by clicking on the "+" symbol on the incline section (This is located below the persistent dashboard).
15. The system will attempt to maintain the speed of the car (The speed where the user turned on the cruise control) by pumping more fuel into the engine.
16. The system will then display an increase in flow of fuel from the fuel tank, to the fuel pump and to the engine.

(User decreases the degree of incline while the cruise control is on)

14. The user will decrease the incline by clicking on the "-" symbol on the incline section (This is located below the persistent dashboard).
15. The system will attempt to maintain the speed of the car (The speed where the user turned on the cruise control) by pumping less fuel into the engine.
16. The system will then display a decrease in flow of fuel from the fuel tank, to the fuel pump and to the engine.

(User increases the altitude while the cruise control is on)

14. The user will increase the altitude by clicking on the "+" symbol on the altitude section (This is located below the persistent dashboard).
15. The system will attempt to maintain the speed of the car (The speed where the user turned on the cruise control) by pumping more fuel into the engine. More fuel is also required to be burnt at higher altitudes.
16. The system will then display an increase in flow of fuel from the fuel tank, to the fuel pump and to the engine.

(User decreases the altitude while the cruise control is off)

14. The user will decrease the altitude by clicking on the "-" symbol on the altitude section (This is located below the persistent dashboard).
15. The system will attempt to maintain the speed of the car (The speed where the user turned on the cruise control) by pumping less fuel into the engine. Less fuel is also required to be burnt at lower altitudes.
16. The system will then display a decrease in flow of fuel from the fuel tank, to the fuel pump and to the engine.

(User attempts to turn on cruise control when the speed of the car is not at 45 km/h)

1. The user will attempt to click on "Cruise control on".
2. The system will then display to the user that the speed of the car must be at least 45 km/h before being able to turn on cruise control.
3. The process to turn on the cruise control is defined in this use case.

(User attempts to turn on cruise control without starting the engine for the car)

1. The user will attempt to click on "Cruise control on".
2. The system will then display to the user that the engine of the car must be on and the speed of the car must be at least 45 km/h before being able to turn on cruise control.
3. The process to turn on the cruise control is defined in this use case.

Name: Increasing/decreasing cruise speed	**ID:** 29
Stakeholders and Goals: User – to increase/decrease the cruise speed	
Description: A user wants to simulate increasing or decreasing the current speed of the cruise.	
Actors: User	
Trigger: User turns on the cruise control and wishes to increase/decrease the current speed of the cruise.	

Normal Flow:

1. This use case is continued from a use case detailed before, "Turning on/off cruise control", use case ID: 28. We continue from point number 13 of the normal flow of the use case. At this point, the cruise control is already ON. The reason why this use case is continued from before is because the user must turn on the cruise control first before being able to increase/decrease the current cruising speed.
2. The user will then increase the cruising speed by clicking on the "+" symbol on the cruise speed bar.
3. The system will then represent the increase in cruising speed by pumping more fuel from the fuel tank to the fuel pump and to the engine. More fuel is required for the engine to achieve the increased in speed.
4. The system will also update the persistent dashboard with the new speed.
5. End

Sub-Flows: None
Alternative/Exceptional Flows: (User wishes to decrease the cruising speed) 5. The user will then decrease the cruising speed by clicking on the "-" symbol on the cruise speed bar. 6. The system will then represent the decrease in cruising speed by pumping less fuel from the fuel tank to the fuel pump and to the engine. Less fuel is required when the speed is reduced. 7. The system will also update the persistent dashboard with the new speed. (User attempts to increase/decrease the cruising sped without having the cruise control on) 7. The user attempts to click on the "+"/"-" on the cruise speed bar. 8. The system displays to the user that the cruise control should be turned on first before being able to increase/decrease the cruising speed. 9. The process of increasing/decreasing the cruising speed is detailed in use case "Turning on/off cruise control (Use case ID: 28)" and "Increasing/decreasing cruising speed (Use case ID: 29)".

v. Defrost rear

Name: Turning on/off rear defroster	**ID:** 30
Stakeholders and Goals: User – to simulate turning on/off the rear defroster	
Description: A user wants to simulate turning on/off the rear defroster	
Actors: User	
Trigger: User runs the application and wishes to turn on/off the rear defroster.	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Electrical System". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the electrical components of the car. There will also be a few buttons available for the user which includes "Lights", "Audio", "Wipers", "Cruise control" and "Defrost rear". 3. The user will then click on "Defrost rear" button in the simulation window. 4. The system will then display to the user the "Rear defroster on" button. 5. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 6. The system will show that the engine has started, by showing different components being lit up on the dashboard. 7. The user will then click on the "Rear defroster on" button. 8. The system will then show the flow of power from the battery to the rear defroster. 9. The system will then update the persistent dashboard to light up the rear defroster symbol.	

10. End.
Sub-Flows: None
Alternative/Exceptional Flows: (User wishes to turn off the rear defroster instead, we are assuming the rear defroster is already on) 7. The user will then click on the “Rear defroster off” button. 8. The system will then stop the flow of power from the battery to the rear defroster.

6. Steering system simulation

Name: Simulate the steering system	**ID:** 31
Stakeholders and Goals: User – to simulate the steering system	
Description: A user wants to simulate turning the steering wheel in different directions, either left or right	
Actors: User	
Trigger: User runs the application and wishes to simulate the steering system.	
Normal Flow: 1. User reaches the main page of the system and clicks on the “Simulation” button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on “Steering System”. The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to see different components of the steering system of the car, which includes the steering wheel, the steering pinion, the steering rack and the wheels. The user will also be able to see two buttons, “< (To turn left)” and “> “To turn right” under the “Steer angle” section. 3. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 4. The system will show that the engine has started, by showing different components being lit up on the dashboard. 5. The user then clicks on the “<” button. 6. The system then updates the “Steer angle” column (It will constantly be updated by -15 degrees for every click, the negative symbol representing that it is going left). 7. The system will also update the different components, if the user is turning left, the steering wheel is turned to the left, the steering pinion will rotate towards the left, the steering rack will rotate in the opposite direction of the steering pinion, which is right, and the tire would then turn to the left by 15 degrees. 8. The image that is placed above the “Steer angle” section will also be replaced with another image that shows that the user is steering left. 9. End	
Sub-Flows: None	
Alternative/Exceptional Flows: (User wishes to turn to the right instead)	

5. The user will then click on the “>” button.
6. The system then updates the current steer angle column (It will update by 15 degrees for every click, positive numbers mean that it is going right).
7. The system will also update the different components, if the user is turning right, the steering wheel is turned to the right, the steering pinion will rotate towards the right, the steering rack will rotate in the opposite direction of the steering pinion, which is left, and the tire would then turn to the right by 15 degrees.
8. The image that is placed above the “Steer angle” section will also be replaced with another image that shows that the user is steering right.

7. Simulate braking system

Name: Engaging/Disengaging hand brakes	**ID:** 32
Stakeholders and Goals: User – to simulate engaging/disengaging the hand brakes	
Description: A user wants to simulate engaging or disengaging the hand brakes.	
Actors: User	
Trigger: User runs the application and wishes to engage/disengage the hand brakes.	
Normal Flow: 1. User reaches the main page of the system and clicks on the “Simulation” button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on “Braking System”. The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the components for the braking system, which includes the battery, the brake pump, the brake fluid, the hand brake and the brake pads. There will also be a few buttons available for the user which includes “Handbrake” and the “Foot brake”. 3. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 4. The system will show that the engine has started, by showing different components being lit up on the dashboard. 5. The user will then click on the “Handbrake on” button (Upon starting the car the handbrakes remain disengaged). 6. The system will then show A flow of power from the battery to the parking brakes (current only goes once to the parking brakes). 7. The system will then show that the parking brakes has been engaged by updating the persistent dashboard to light up the handbrake symbol. 8. End.	
Sub-Flows: None	
Alternative/Exceptional Flows: (User wishes to disengage the handbrake) 9. The user will then click on the “Handbrake off” button. 10. The system will then show A flow of power from the battery to the	

parking brakes (current only goes once to the parking brakes).

11. The system will then show that the parking brakes has been disengaged by updating the persistent dashboard to turn off the handbrake symbol.

**probably need to take into consideration the current speed of the car. The user should be able to engage the handbrake even when the car is moving. Emergency braking.

Name: Simulating foot brakes	**ID:** 33
Stakeholders and Goals: User – to simulate the foot brakes	
Description: A user wants to simulate the foot brakes	
Actors: User	
Trigger: User runs the application and wishes to simulate the foot brakes	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Braking System". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to view the components for the braking system, which includes the battery, the brake pump, the brake fluid, the hand brake and the brake pads. There will also be a few buttons available for the user which includes "Handbrake" and the "Foot brake". 3. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 4. The system will show that the engine has started, by showing different components being lit up on the dashboard. 5. The user increase the speed of the car by clicking on the "+" symbol under the "Speed" section (This is on the persistent dashboard) until he/she has reached 60km/h. 6. The system will then display an increase in speed of the car on the dashboard. 7. The user will then CLICK AND HOLD on the "Foot brake" button. 8. The system will then send a signal from the foot brake to the brake pump. 9. The system will also show that the brake pump will constantly pump brake fluid to the 4 wheels of the car. 10. The system will then constantly update the speed of the car on the persistent dashboard at a reducing rate of 15km/h. 11. End.	
Sub-Flows: None	
Alternative/Exceptional Flows: None	

8. Simulate adding attachment

Name: Simulate adding/removing attachments	**ID:** 35
Stakeholders and Goals: User – to simulate adding attachments	
Description: A user wants to simulate adding attachments to a car	

Actors: User
Trigger: User runs the application and wishes to simulate adding attachments
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Adding attachment". The window (i.e the simulation window) below the persistent dashboard will then be updated. The user will then be able to see a picture which represents the back view of the car. There will also be a few buttons available for the user which includes "Add tow box", "Add trailer" and the "Remove attachment" button (In "functionality pathway", the word used was "unmount", it doesn't look like the word exists, so phrase "Remove attachment" used instead"). 3. The user will then click on the "Add tow box" button. 4. The system will then update the image (Image that shows the back of the car) to another image which shows the rear of the car with a tow box attached to it. 5. End.
Sub-Flows: None
Alternative/Exceptional Flows: (The user wishes to add a trailer instead, we are assuming that the user has not added any attachments to the car) 4. The user will then click on the "Add trailer" button 5. The system will then update the image (Image that shows the back of the car) to another image which shows the rear of the car with a trailer attached to it. (The user wishes to remove any attachments that are on the car) 4. The user then click on the "Remove attachment" button. 5. The system will then update the image (Image that shows the back of the car WITH an attachment, could be either the tow box or the trailer) to the original image which only shows the rear of the car. (The user wishes to add an attachment when there is already an attachment on the car) 4. The user attempts to click on "Add tow box"/"Add trailer" button (We are assuming that either the tow box or the trailer has already been attached to the back of the car) 5. The system then displays to the user that there is already an existing attachment on the car and should remove it first before being able to add another attachment to the car. 6. The process to removing an attachment to the car is explained in this use case (The section right above of this, "The user wishes to remove any attachments that are on the car").

9. Simulate alternative fuel powered drivetrains

Name: Simulate a hybrid car	**ID:** 36
Stakeholders and Goals: User – to simulate a hybrid car	

Description: A user wants to simulate a hybrid car instead of a normal petrol car.

Actors: User

Trigger: User runs the application and wishes to simulate a hybrid car

Normal Flow:

1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations.
2. On the list of simulations available to the user, the user will then click on "Alternative fuel powered drivetrains". The window (i.e the simulation window) below the persistent dashboard will then be updated. There will be a few buttons available for the user which includes "Hybrid", "Electric" and the "Hydrogen" button.
3. The user will then click on the "Hybrid" button to simulate a hybrid car.
4. The system will show different components on the screen that will represent a hybrid car, which includes the battery, the engine, fuel tank, fuel pump, battery powered motor, and also the wheels
5. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
6. The system will show that the engine has started, by showing different components being lit up on the dashboard.
7. The system will then show fuel flowing from the fuel tank through the fuel pump and to the engine. At the same time, power from the battery (which is placed at the back of the car) will also flow to the motor which is placed at the front part of the car. Both the engine and the motor will power the car and rotate the front wheels (This happens as the car is moving at an INCREASING speed).
8. End.

Sub-Flows: None

Alternative/Exceptional Flows:

Different things happen in a hybrid car in different situations compared to a petrol car.

(When the car is at constant speed)

7. The system will then show the rotation of the back wheels sending power to the battery (which is placed at the back of the car). The rotation of the back wheels is recharging the battery.

(When the car remains static)

8. The system will then stop the flow of fuel from the fuel tank to the fuel pump and to the engine. The power from the battery will remain flowing to the motor. The engine has been shut down to save power.

(When the user slows down the car or applies the brake)

6. The system will then show the rotation of the back wheels sending power to the battery. The kinetic energy from the car which is slowing down is converted to heat energy and is recharging the battery of the car (Regenerative braking).

**This use case needs work.

Name: Simulate a hydrogen car	**ID:** 37
Stakeholders and Goals: User – to simulate a hydrogen car	
Description: A user wants to simulate a hydrogen car instead of a normal petrol car.	
Actors: User	
Trigger: User runs the application and wishes to simulate a hydrogen car	
Normal Flow: 9. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 10. On the list of simulations available to the user, the user will then click on "Alternative fuel powered drivetrains". The window (i.e the simulation window) below the persistent dashboard will then be updated. There will be a few buttons available for the user which includes "Hybrid", "Electric" and the "Hydrogen" button. 11. The user will then click on the "Hydrogen" button to simulate a hydrogen car. 12. The system will show different components on the screen that will represent a hydrogen car, which includes hydrogen tank, fuel cell stacks, motor and the wheels. 13. On the persistent dashboard the user will be able to see the "Start Engine" button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine. 14. The system will show that the engine has started, by showing different components being lit up on the dashboard. 15. The system will then show hydrogen flowing from the hydrogen tank to the fuel stack cells which then produces power and is sent to the front motors to move the front wheels. 16. End.	
Sub-Flows: None	
Alternative/Exceptional Flows: None	

Name: Simulate an electric car	**ID:** 38
Stakeholders and Goals: User – to simulate an electric car	
Description: A user wants to simulate an electric car instead of a normal petrol car.	
Actors: User	
Trigger: User runs the application and wishes to simulate an electric car	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Simulation" button. The user will be brought to the simulation page where the user will be able to view the persistent dashboard and also a list of simulations. 2. On the list of simulations available to the user, the user will then click on "Alternative fuel powered drivetrains". The window (i.e the simulation window) below the persistent dashboard will then be updated. There will be a few buttons available for the user which includes "Hybrid", "Electric" and the "Hydrogen" button.	

3. The user will then click on the “Electric” button to simulate an electric car.
4. The system will show different components on the screen that will represent an electric car, which includes the battery, motor and the wheels.
5. On the persistent dashboard the user will be able to see the “Start Engine” button (What the user may see may differ depending if the engine of the car is on/off). The user will then click on it to start the engine.
6. The system will show that the engine has started, by showing different components being lit up on the dashboard.
7. The system will then show power flowing from the battery towards the motor which is located at the front of the car and then rotating the front wheels. .
8. End.

Sub-Flows: None

Alternative/Exceptional Flows:

Different things happen in an electric car in different situations compared to a petrol car.

(When the car is at constant speed)

7. The system will then show the rotation of the back wheels sending power to the battery (which is placed at the back of the car). The rotation of the back wheels is recharging the battery.

(When the user slows down the car or applies the brake)

7. The system will then show the rotation of the back wheels sending power to the battery. The kinetic energy from the car which is slowing down is converted to heat energy and is recharging the battery of the car (Regenerative braking).

**This use case needs work.

Quiz

Name: Take a quiz	**ID:** 39
Stakeholders and Goals: User – to take a quiz	
Description: A user wants to take a quiz to test his/her understanding on car processes and components	
Actors: User	
Trigger: User runs the application and wishes to take a quiz	
Normal Flow: 1. User reaches the main page of the system and clicks on the "Quiz" button. Upon clicking the quiz button, the user will be brought into the quiz page. 2. The user will be prompted to choose the difficulty of the quiz he wishes to work on (At the moment there will be no difficulty, stretch goal). 3. Upon selecting the difficulty, a progress bar will appear on the screen to show that the system is preparing the questions to be asked for the quiz. The quiz will then start after the loading is complete. A timer will start. 4. The user will then answer the questions by clicking on the answer the user thinks that it is correct. The user will be able to use the keyboard as a form of input for the answers, numbers will represent each answer of the question (i.e 1 for A, 2 for B). 5. The user then clicks the next button 6. The system will show the next question. 7. The user will be able to have quick access to different questions by clicking on the quick access bar which will be placed above the question. 8. Steps 4 to 7 are repeated until the users have answered all the questions 9. Upon completion of all the questions available the user will be able to click on "Complete Quiz" button which will appear on the top right of the question box. 10. The user will then be brought to the "Review quiz" page, which will be the next use case.	
Sub-Flows: None	
Alternative/Exceptional Flows: **(After the quiz has been started from step 4 onwards)** 7. If the user wishes to quit halfway during the quiz, the user will be able to click on a "X" button which will be present on the top right of the screen. No statistics will be saved. 8. The systems prompts for reconfirmation to quit the quiz. 9. The user enters confirmation. **10.** The system brings the user back to the main menu.	

Name: Review quiz	**ID:** 40
Stakeholders and Goals: User – to review the quiz he/she has completed	
Description: A user wants to gain feedback (correct and wrongly answered questions) on the quiz that he/she has completed.	
Actors: User	
Trigger: User finishes the quiz and wishes to review his/her progress.	
Normal Flow: 1. Continuing from the previous use case, the user will now be on the "Review page" and user will be able to see his/her final score for the quiz. 2. The interface for reviewing the questions will be similar to the interface	

of the quiz. The user will then be able to click on the quick access bar to access different previously answered questions. Different colour on the quick access bar will represent if the user has answered the questions correctly or wrongly (Green for correctly answered questions, red for wrongly answered questions). 3. Upon entering a wrongly answered question, the answer that the user has chosen (which is the wrong one, if it is not clear enough) will be highlighted in red and the correct answer will be highlighted in green. If the user enters a question that he/she has already answered correctly, the user will see the correct answer (which is also the answer that he/she has chosen) in green. 4. The user will then be able to click the "X" button which will be on the top right corner to quit the quiz review and go back to the main menu. 5. End
Sub-Flows: None
Alternative/Exceptional Flows: None

Change log of Use Cases

Date	Changes made
16/5/2015	- Use case 1, updated to zoom in, use the up and down - Use case 2 updated - Use case 3 update to be linked from previous use case. Since video links can only been seen under the description of the video. Updated alternative - Use case 4 added - Use case 5 updated, updated from number 4 to 5, updated on what the user needs to do in order to view different wheel drive system - Use case “Take a quiz”, updated how one will take a quiz - Use case “Review quiz”, added today - Split use case into 3 different part and added change log

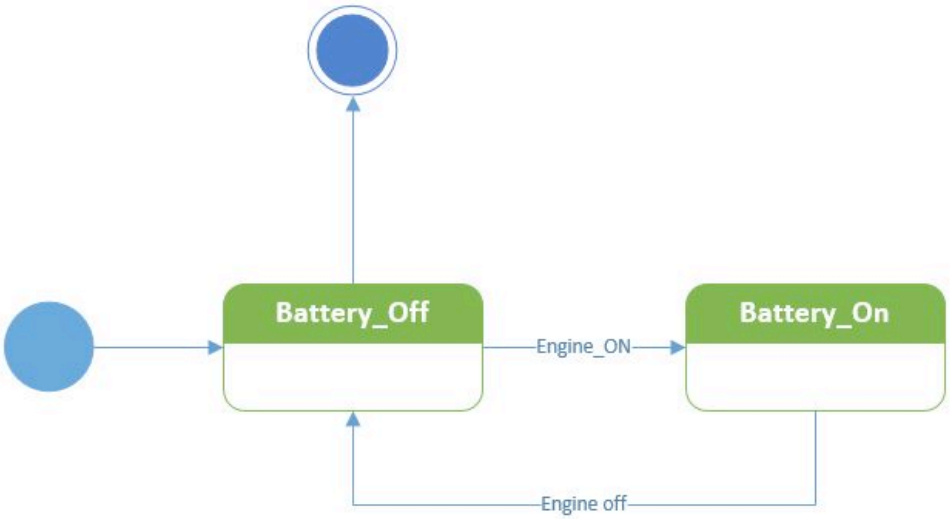
State Diagrams

State diagrams also act as a good source of information in the development process, as it assists the user to identify the various states a certain object can be in. This would assist the developer as they would be able to best choose the type of variables used and the variables that are needed in order to realise the various states that have been defined in the state diagrams.

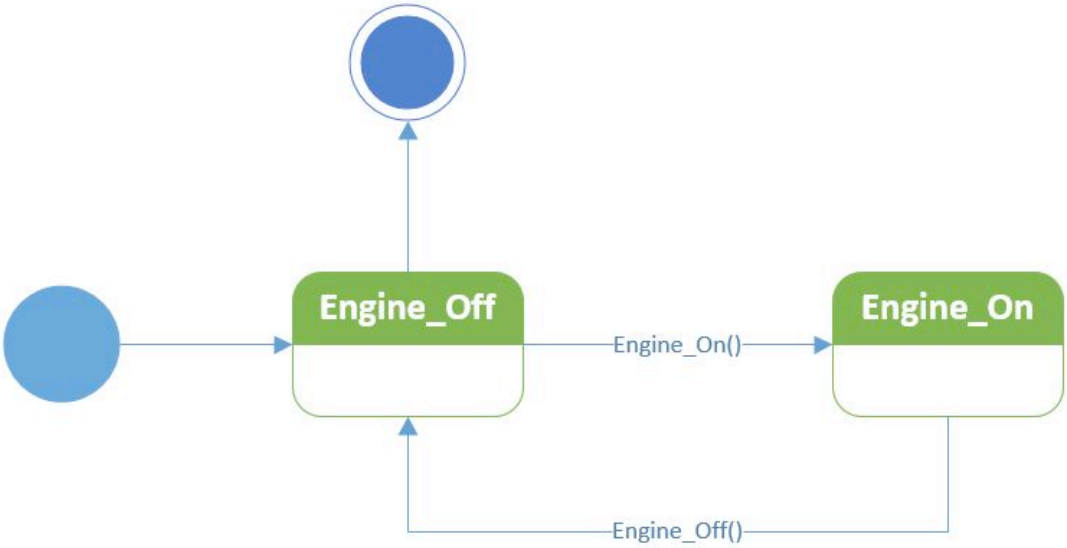
The following pages show the various state diagrams that are part of our application. There may be some changes in the state diagrams seen here, as we may have to add/remove/modify some states when we proceed in the development process.

State diagrams

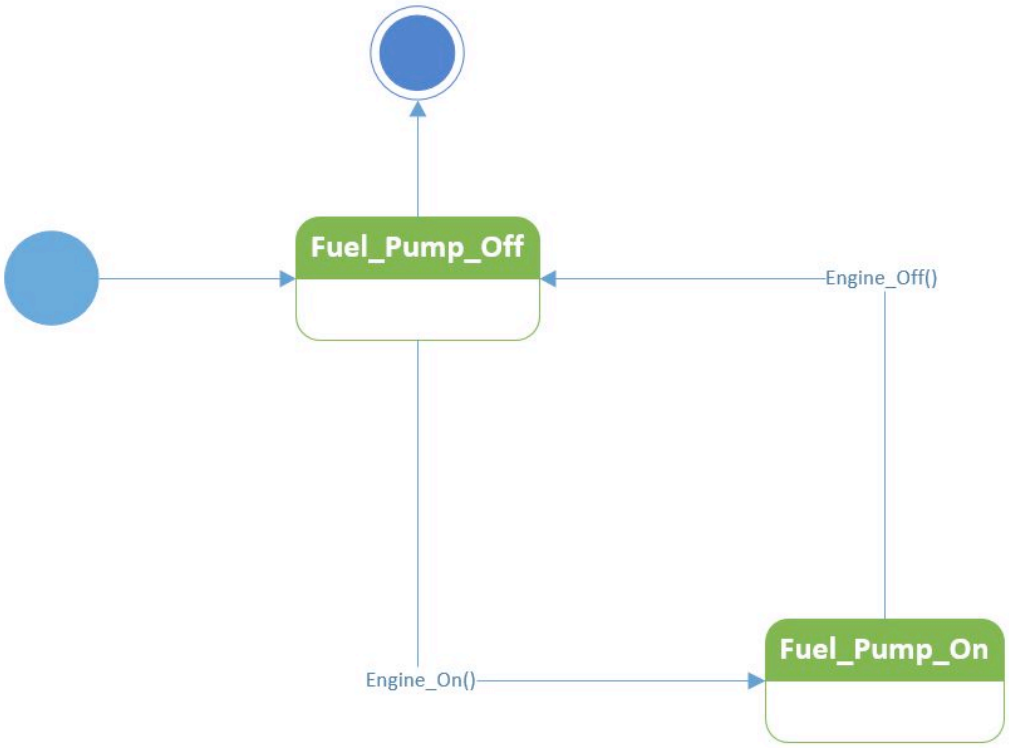
1. Battery



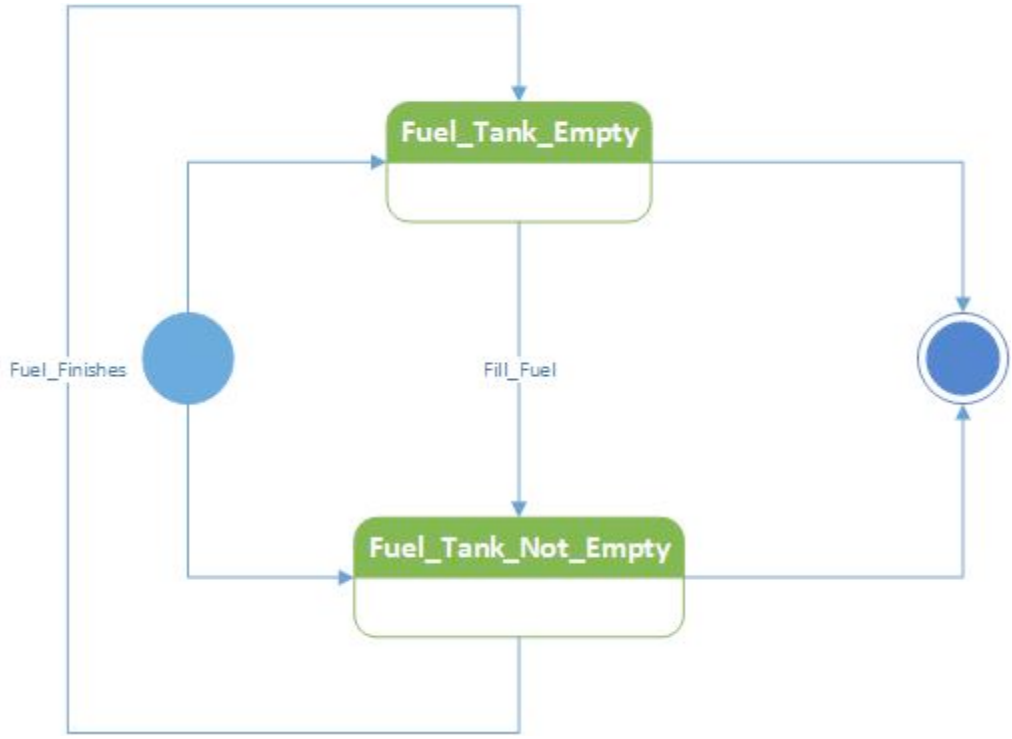
2. Engine



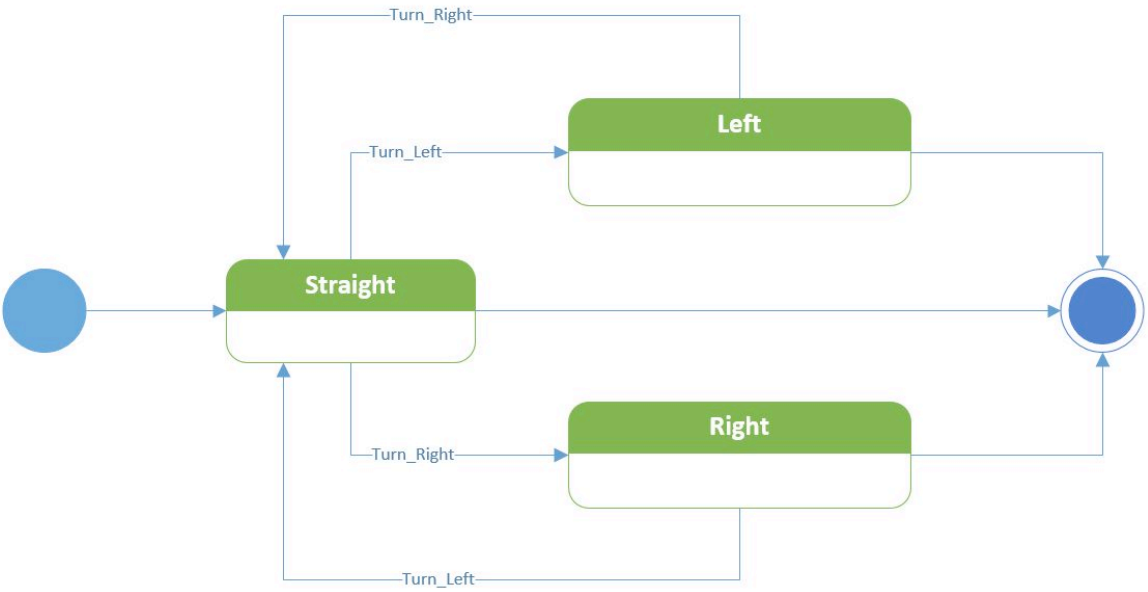
3. Fuel pump



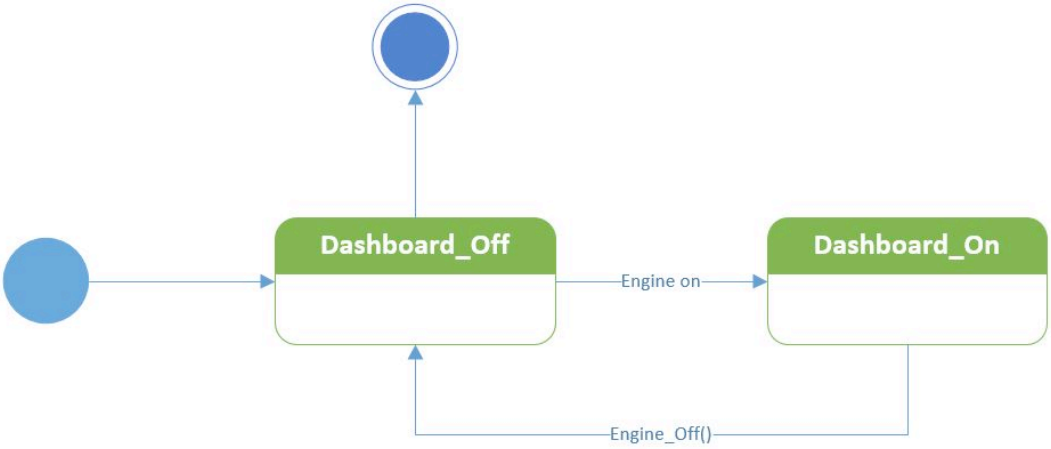
4. Fuel Tank



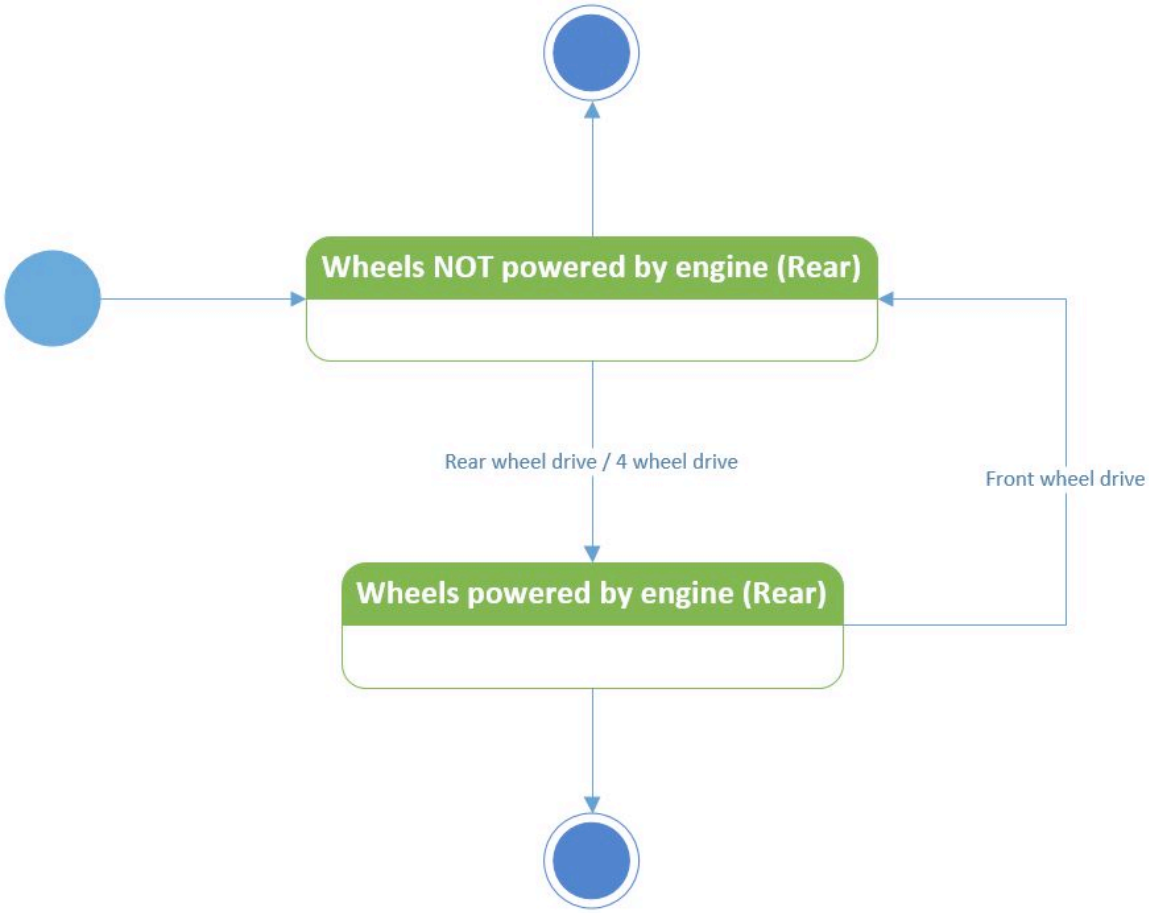
5. Front Wheels



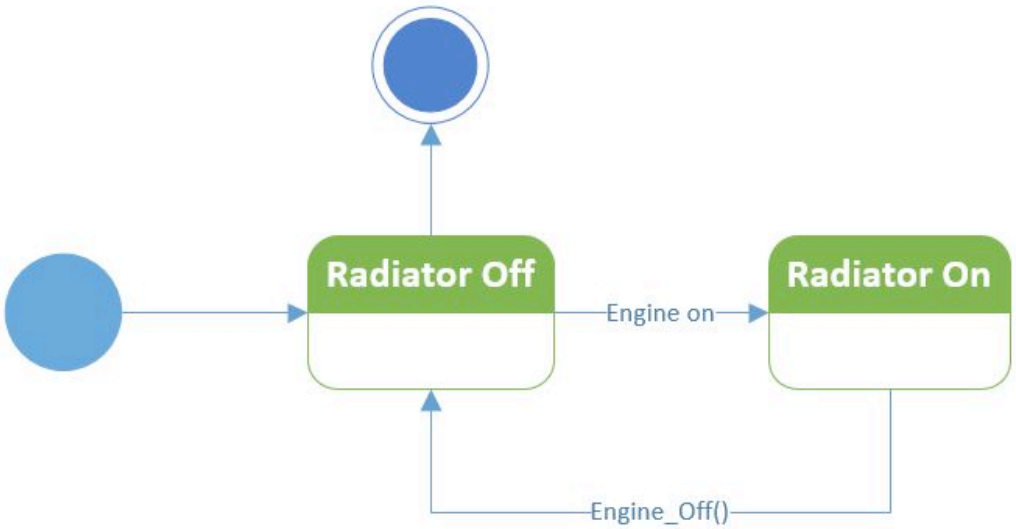
6. Dashboard



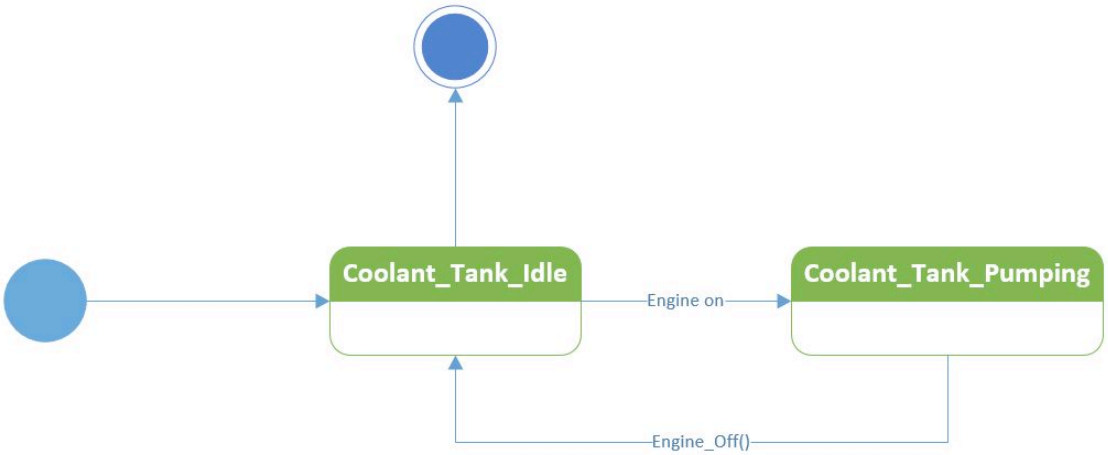
7. Rear wheel drivetrain



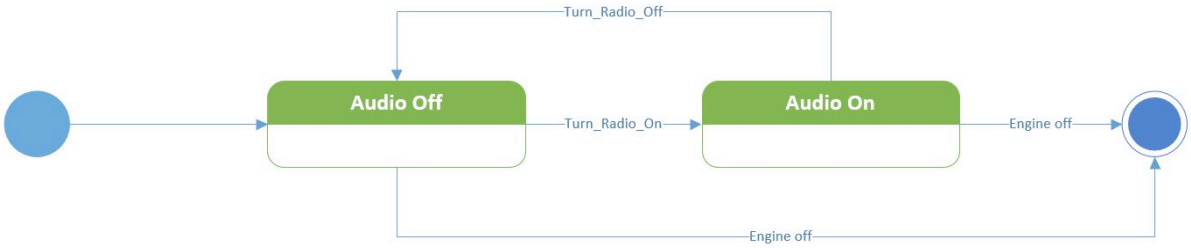
8. Radiator



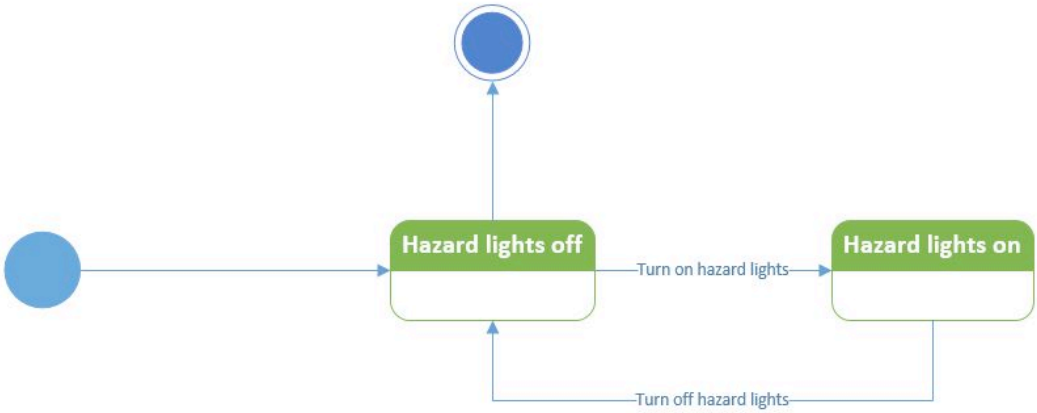
9. Coolant tank



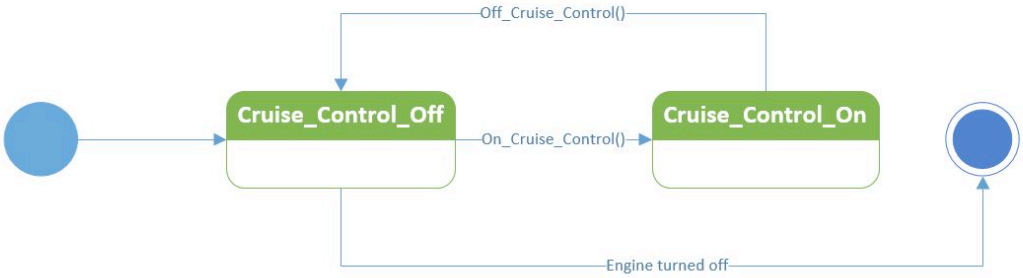
10. Audio system



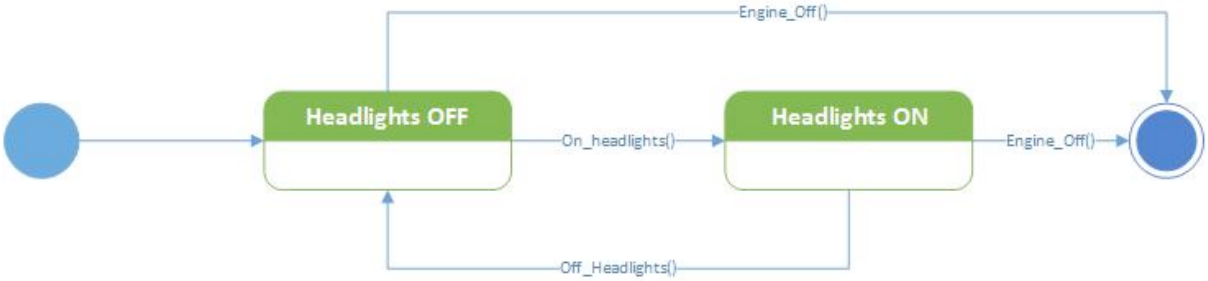
11. Hazard lights



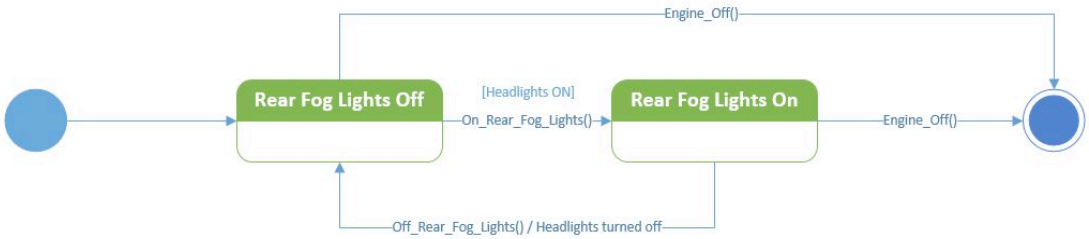
12. Cruise control



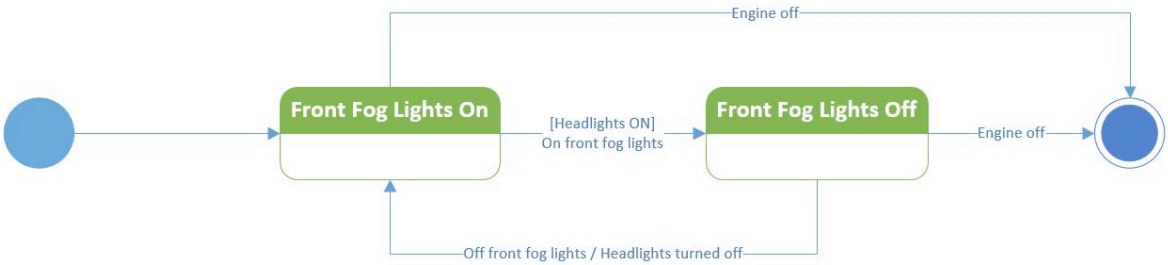
13. Headlights



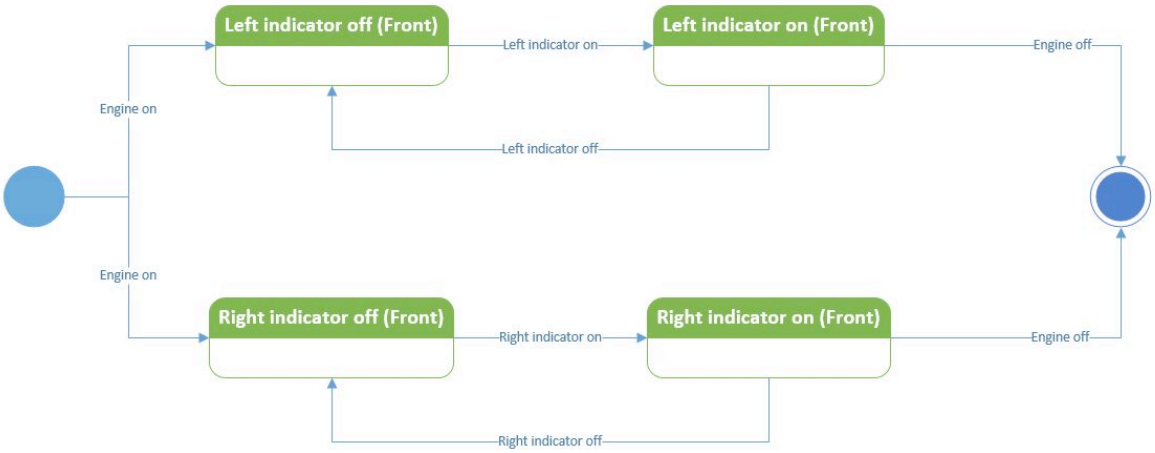
14. Rear fog lights



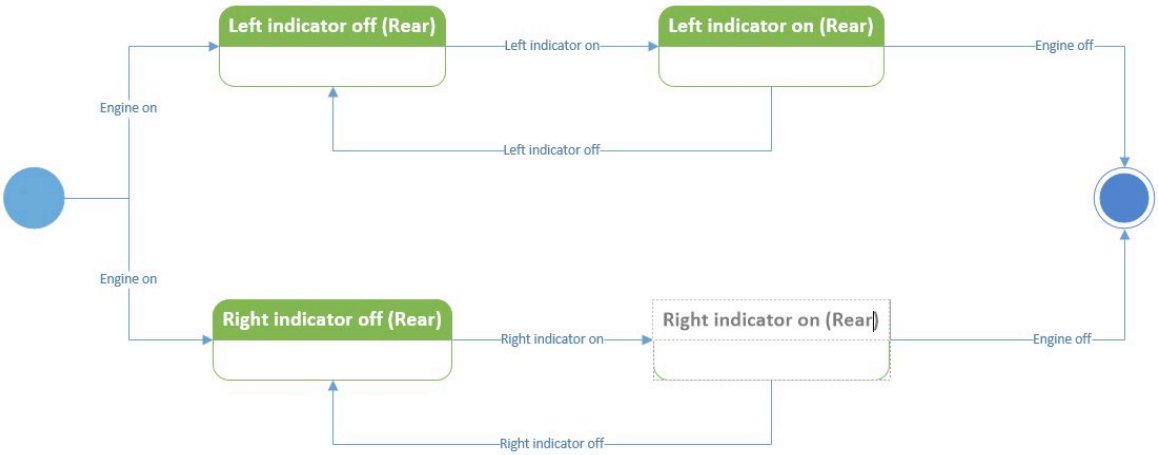
15. Front fog lights



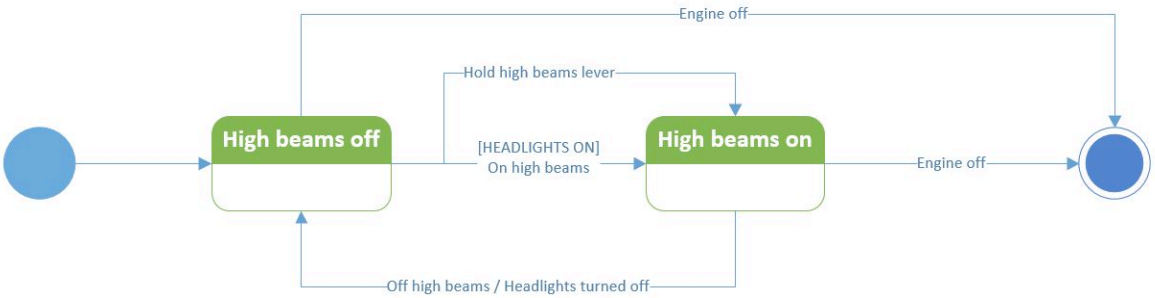
16. Front indicator lights



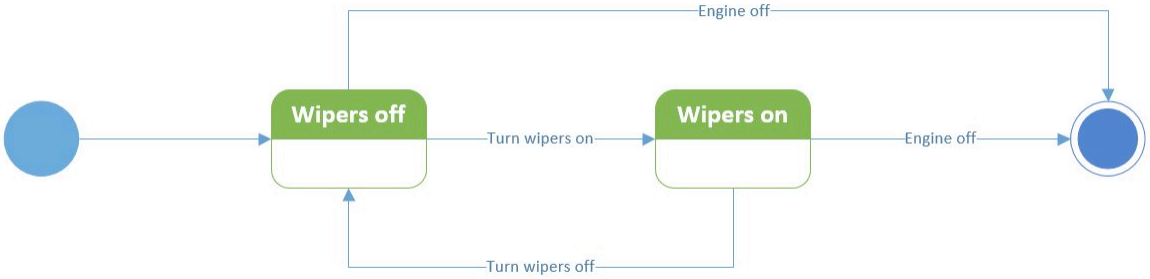
17. Rear indicator lights



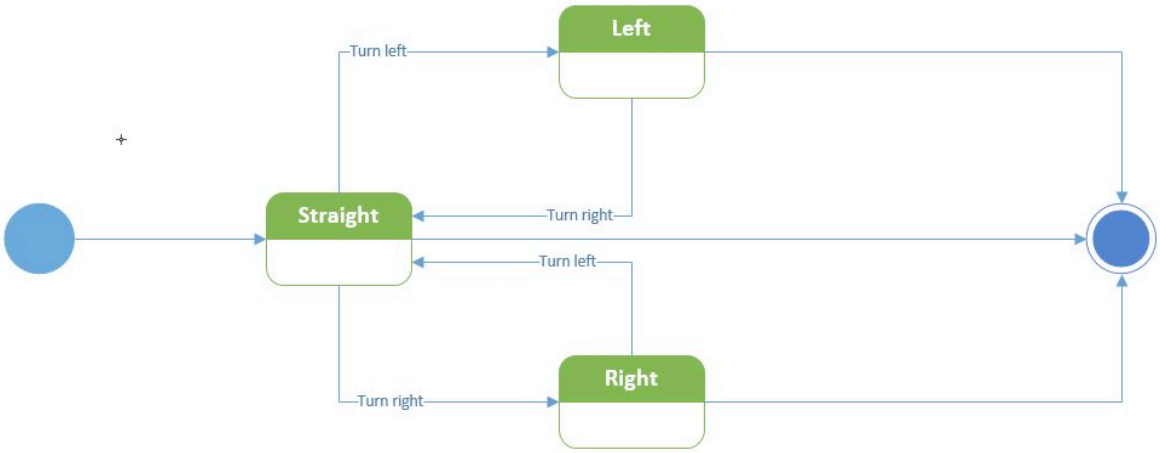
18. High beams



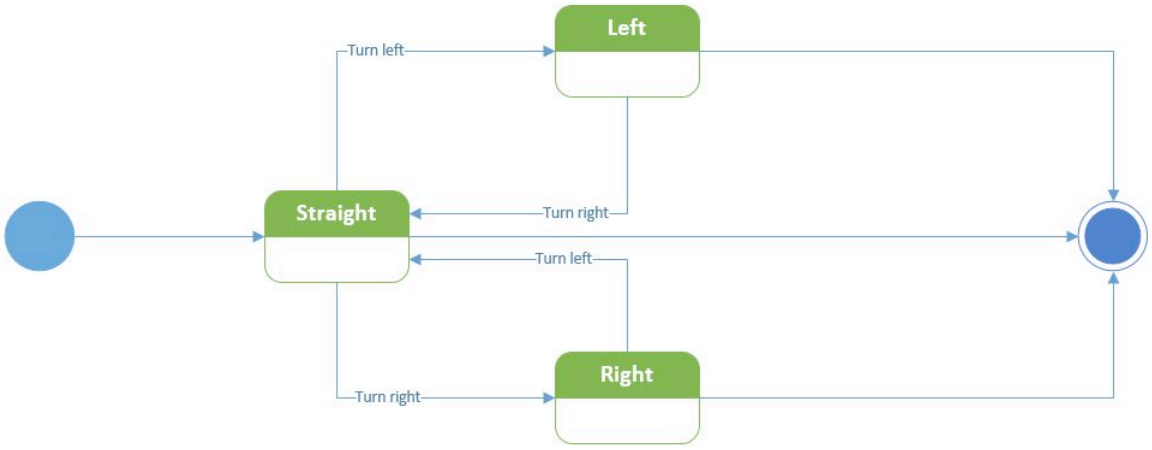
19. Wipers



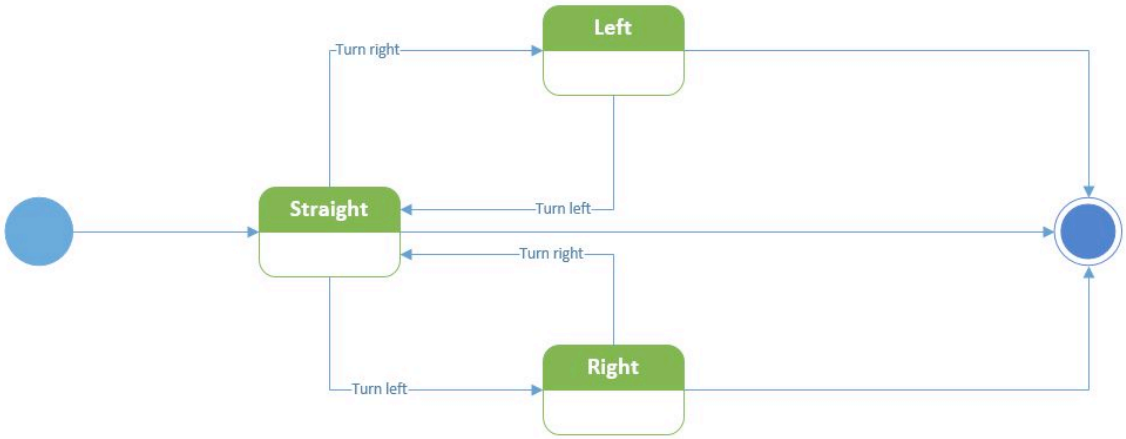
20. Steering wheels



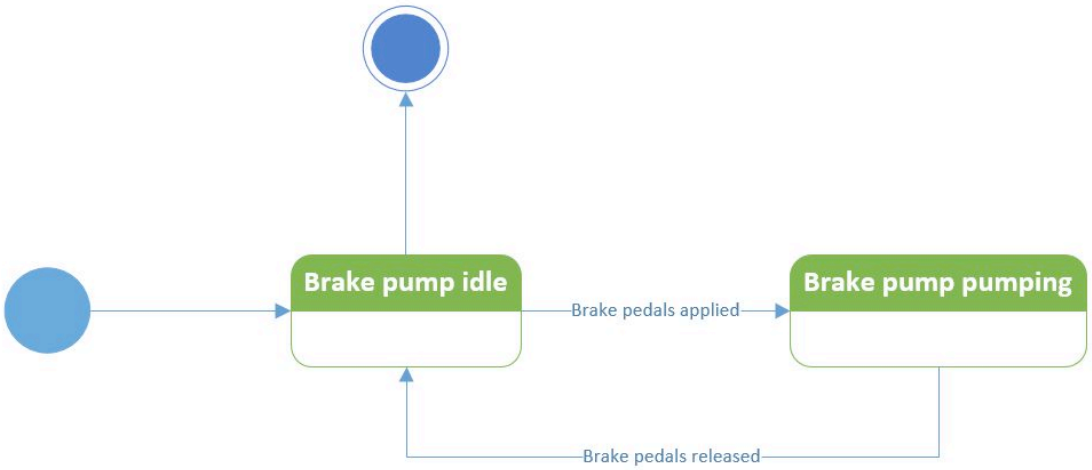
21. Steering rod



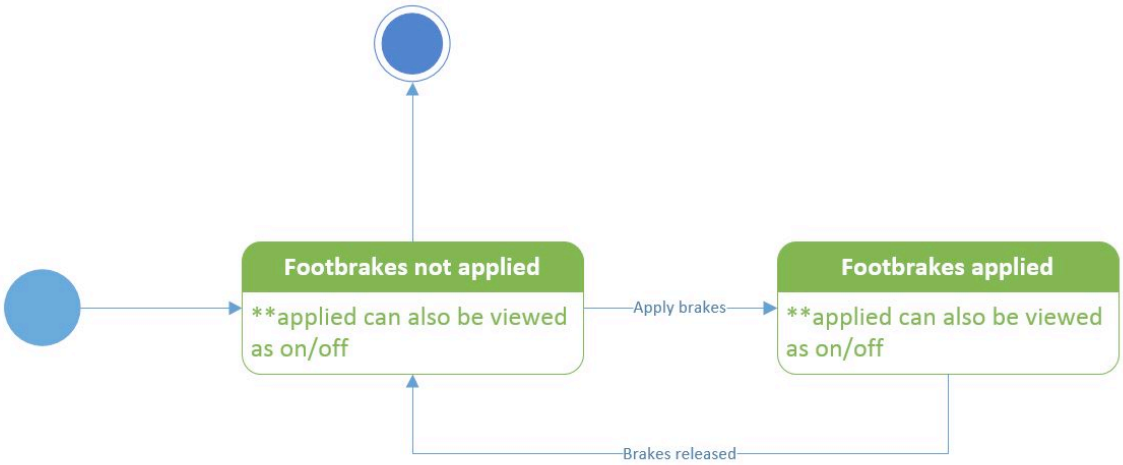
22. Steering rack



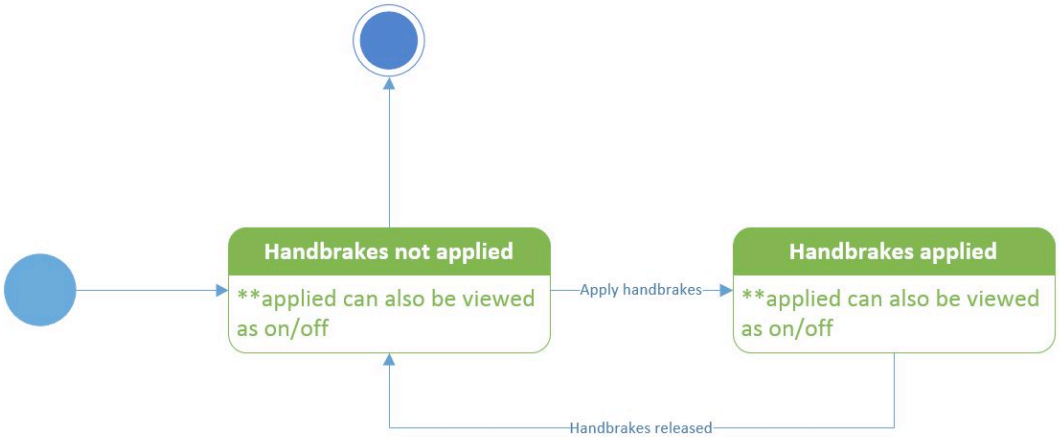
23. Brake pumps



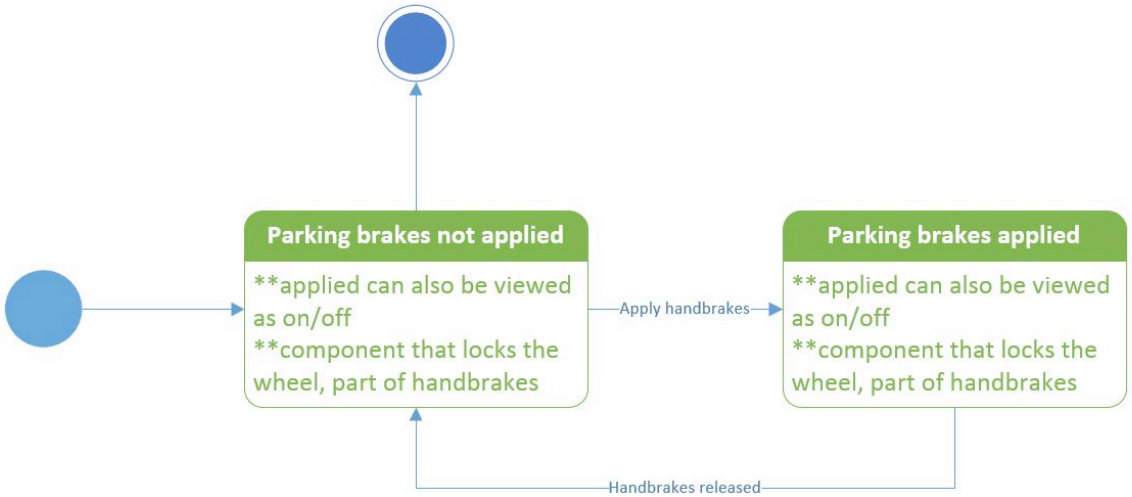
24. Foot brakes



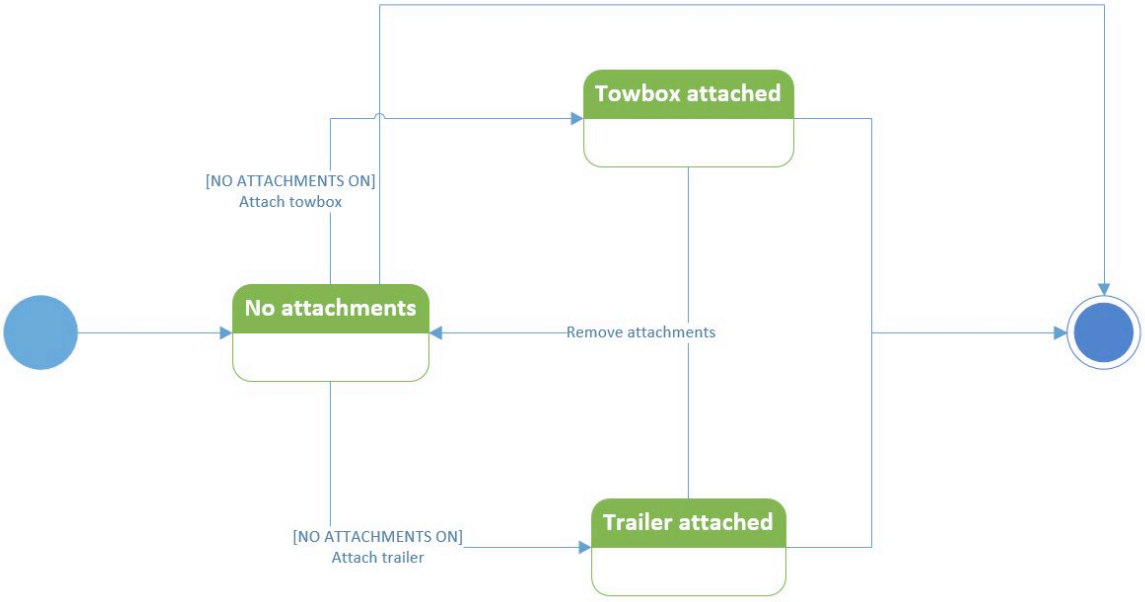
25. Handbrakes



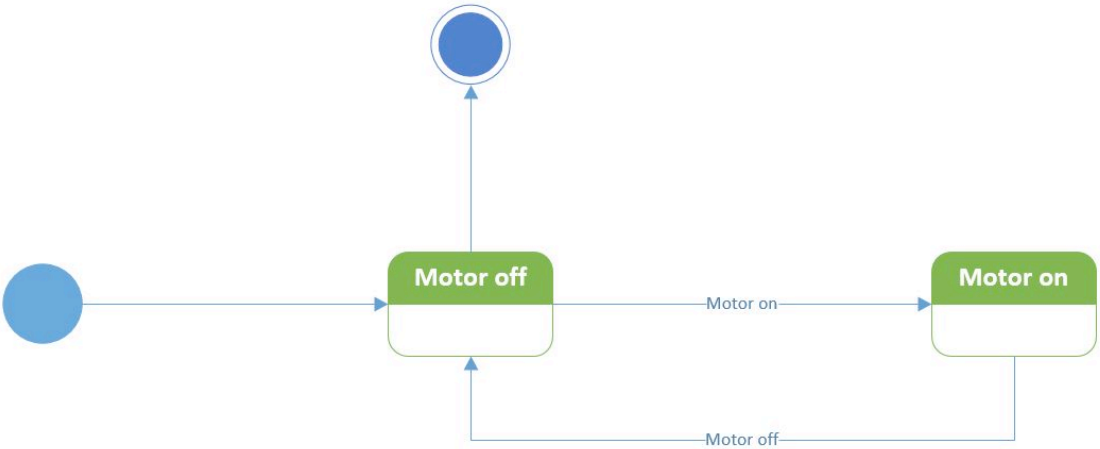
26. Parking brakes



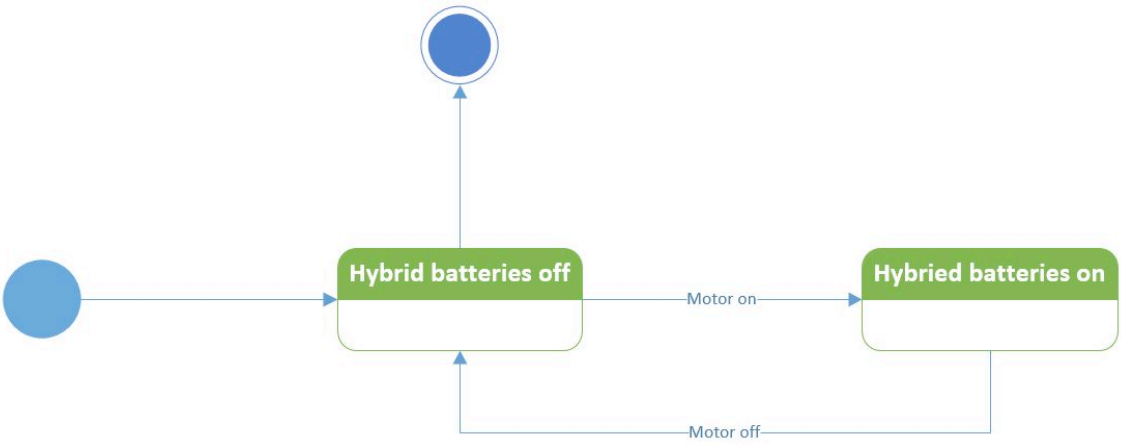
27. Attachments



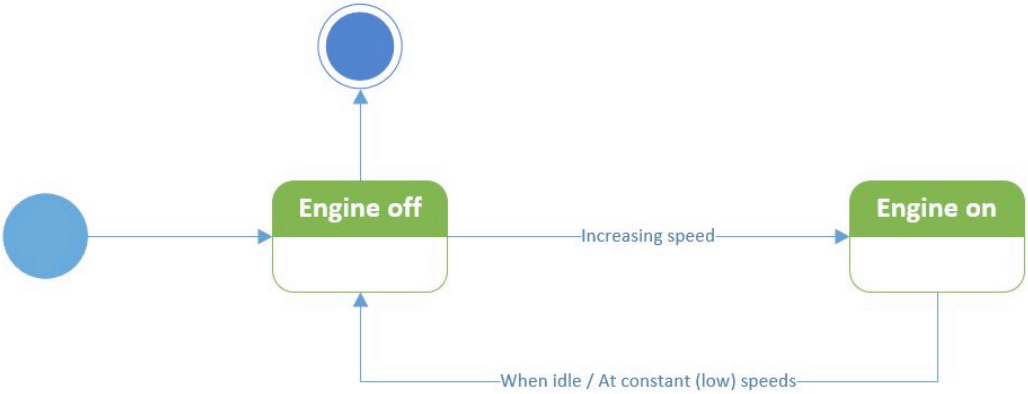
28. Motors



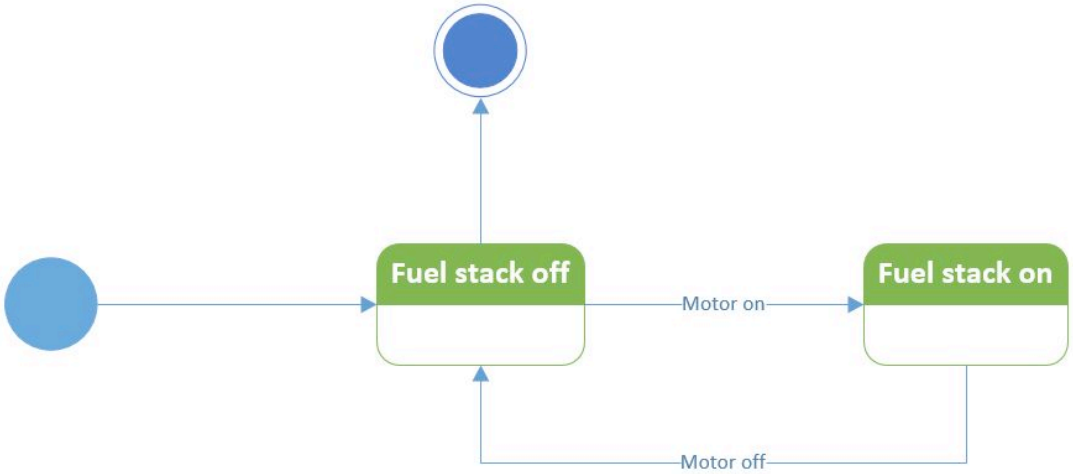
29. Hybrid battery



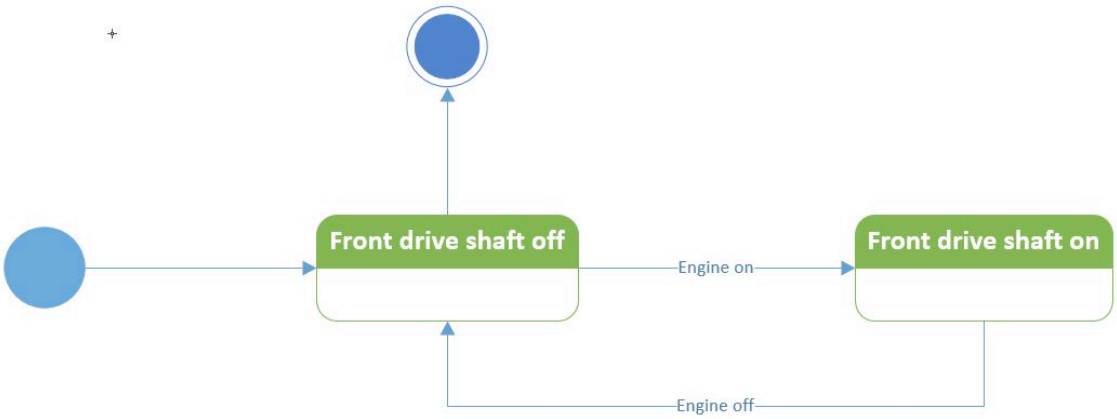
30. Hybrid engine



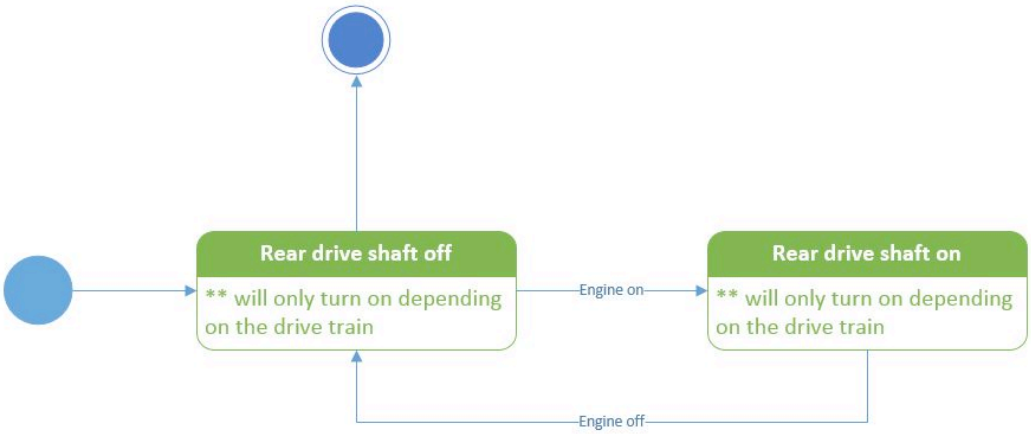
31. Fuel stack



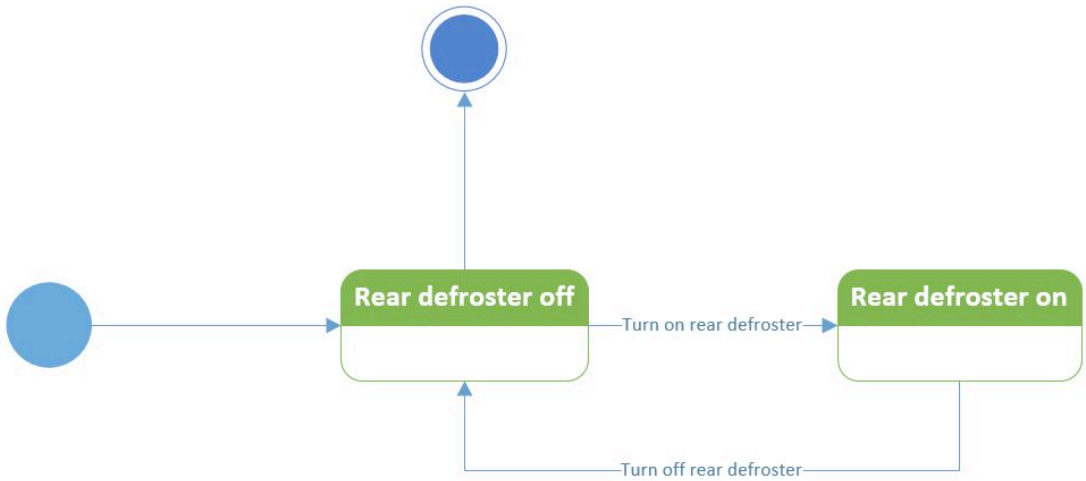
32. Front drive shaft



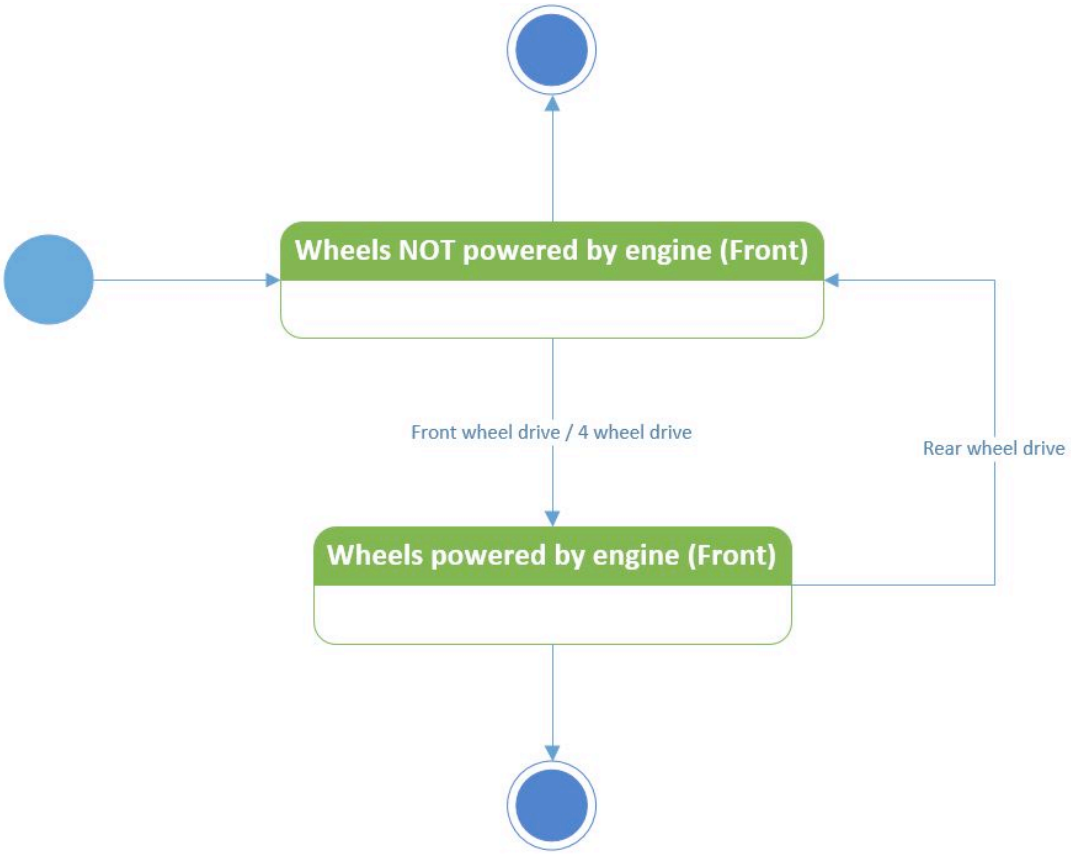
33. Rear drive shaft



34. Rear defroster



35. Front wheel drive train

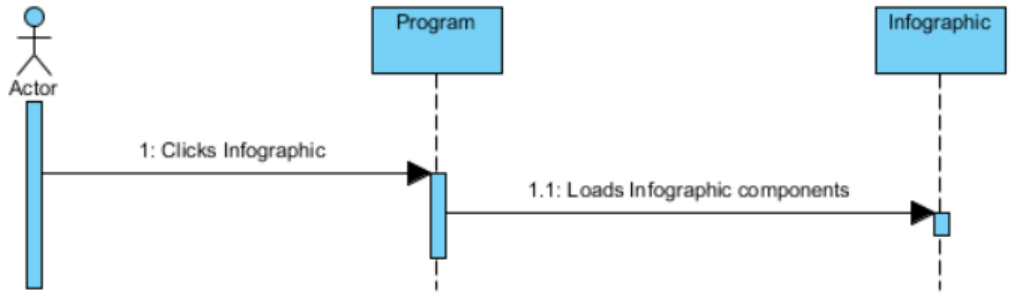


Sequence Diagrams

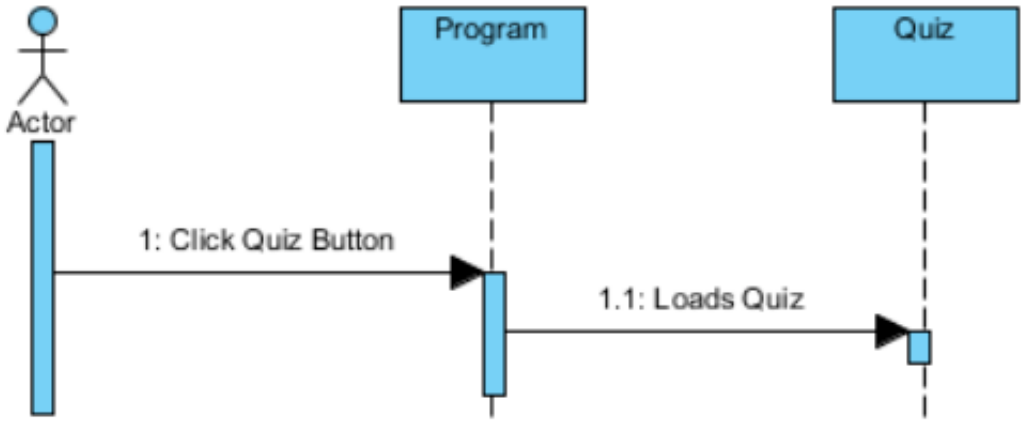
The following pages summaries the sequence diagrams of the main functionalities in our application. As we have not fully developed the application, we cannot come up with final sequence diagrams yet.

At this point, these sequence diagrams would act as a workflow on how a function should run in the application. We may have to change them as development proceeds as further research on how some of the functions of a car take place is still underway.

Infographic

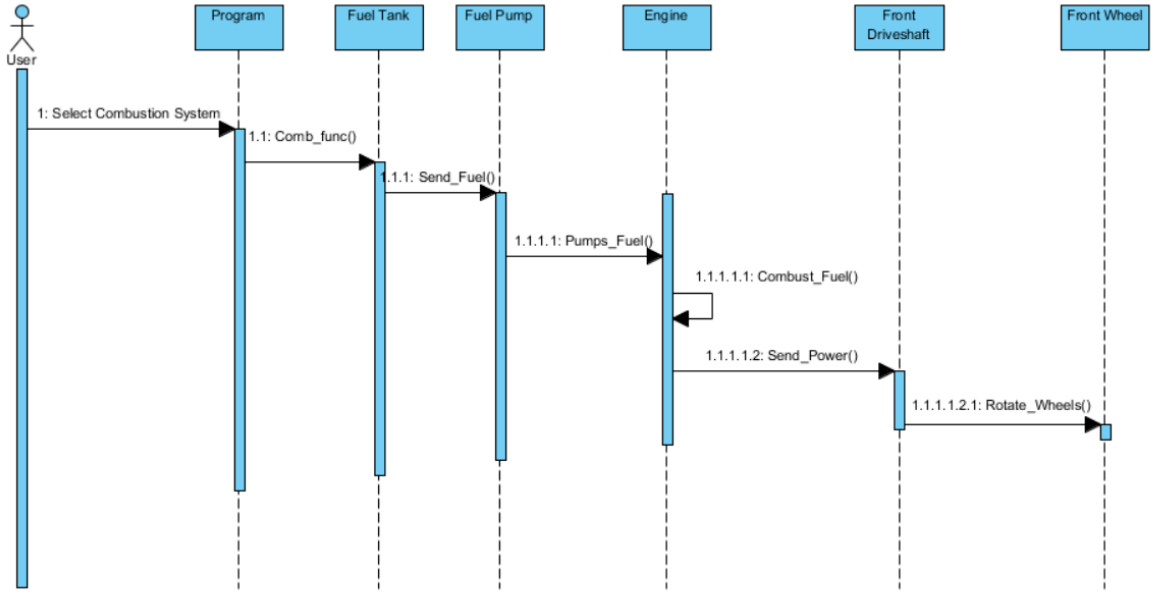


Quiz

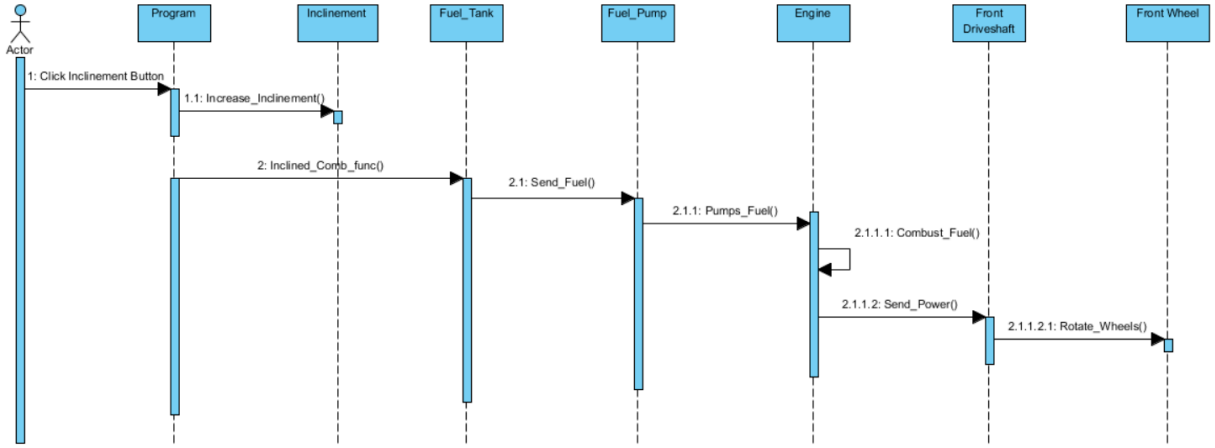


Combustion System

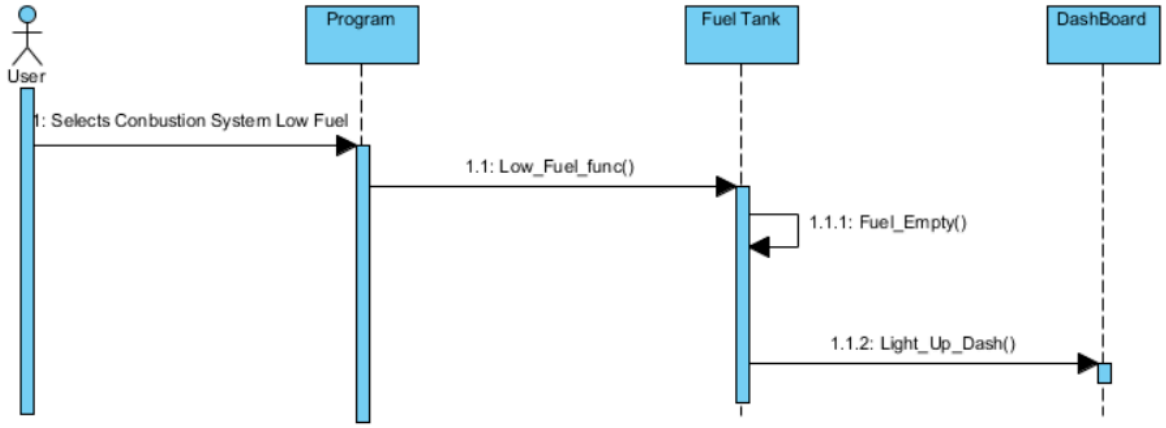
Combustion System (Normal)



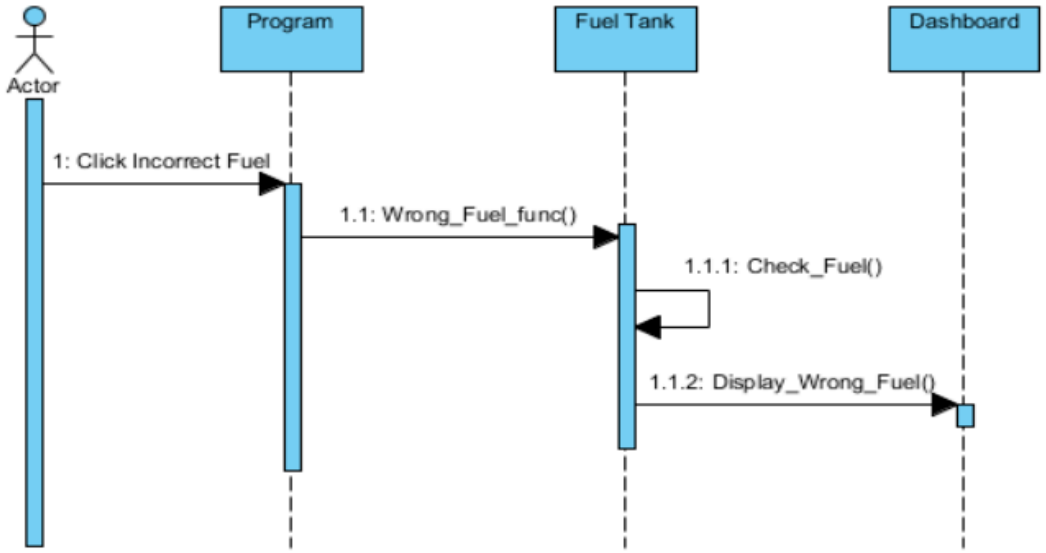
Combustion System (Inclined)



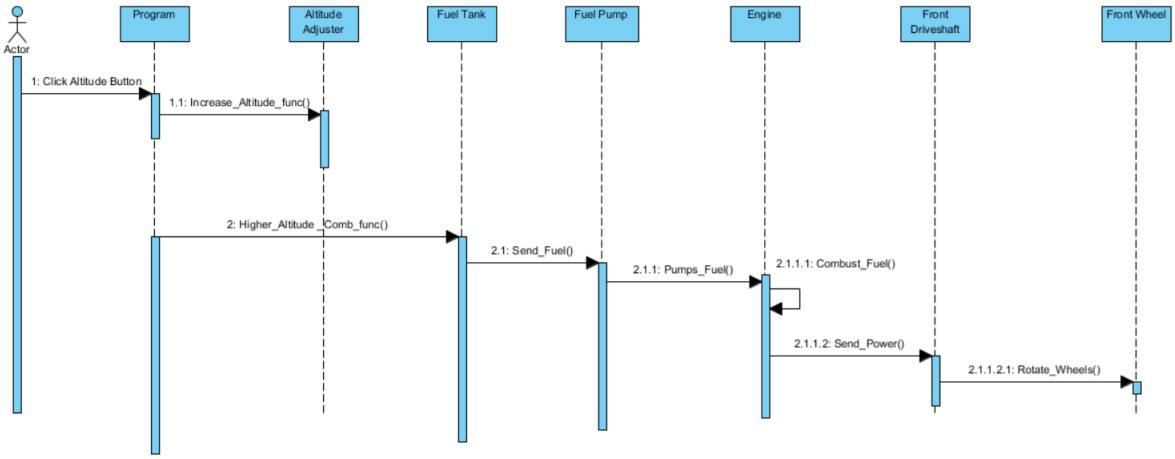
Combustion System (Low Fuel)



Combustion System (Incorrect Fuel)

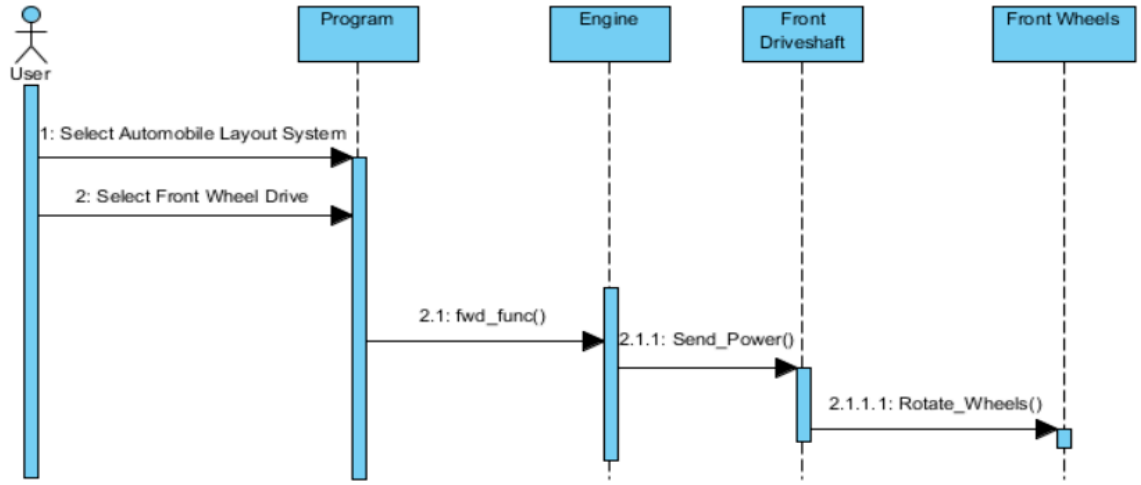


High Altitude Driving System

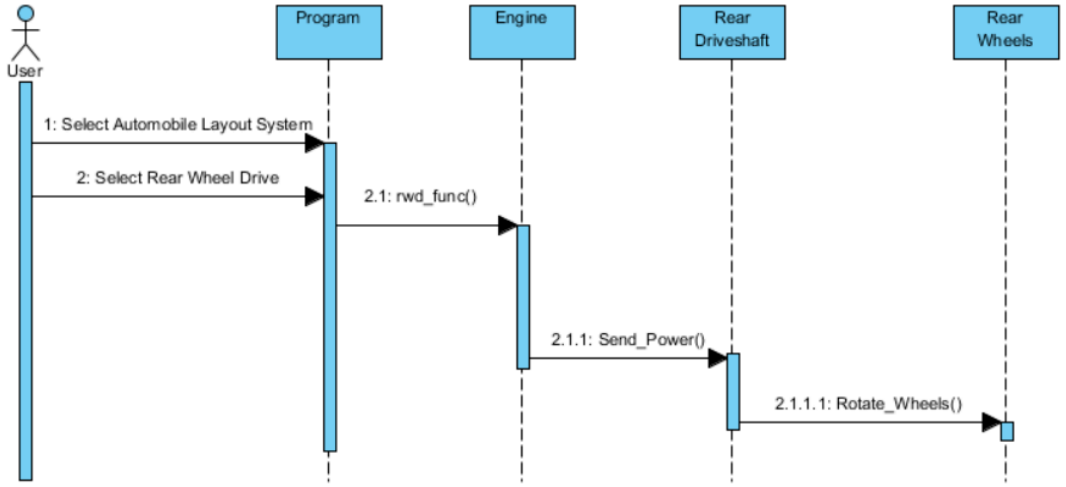


Drive Wheel System

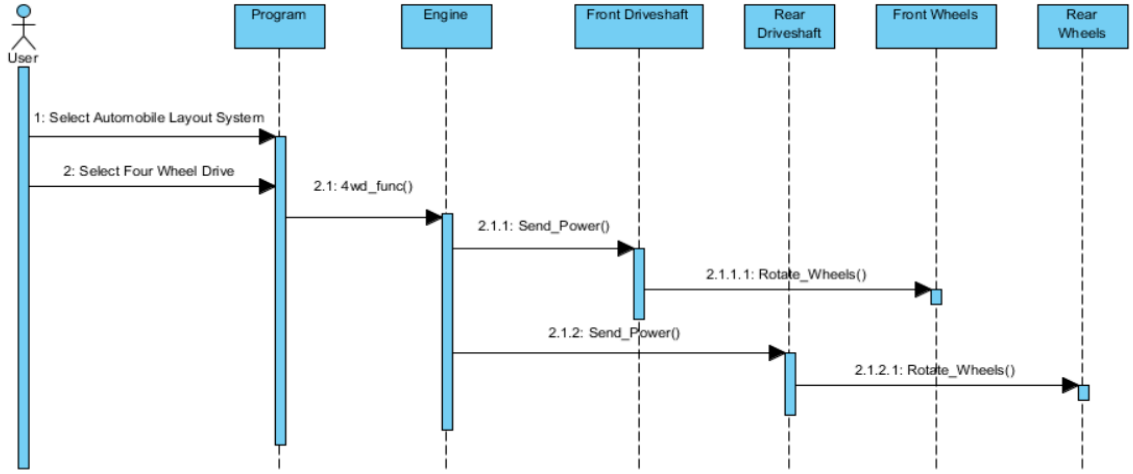
Front Wheel Drive



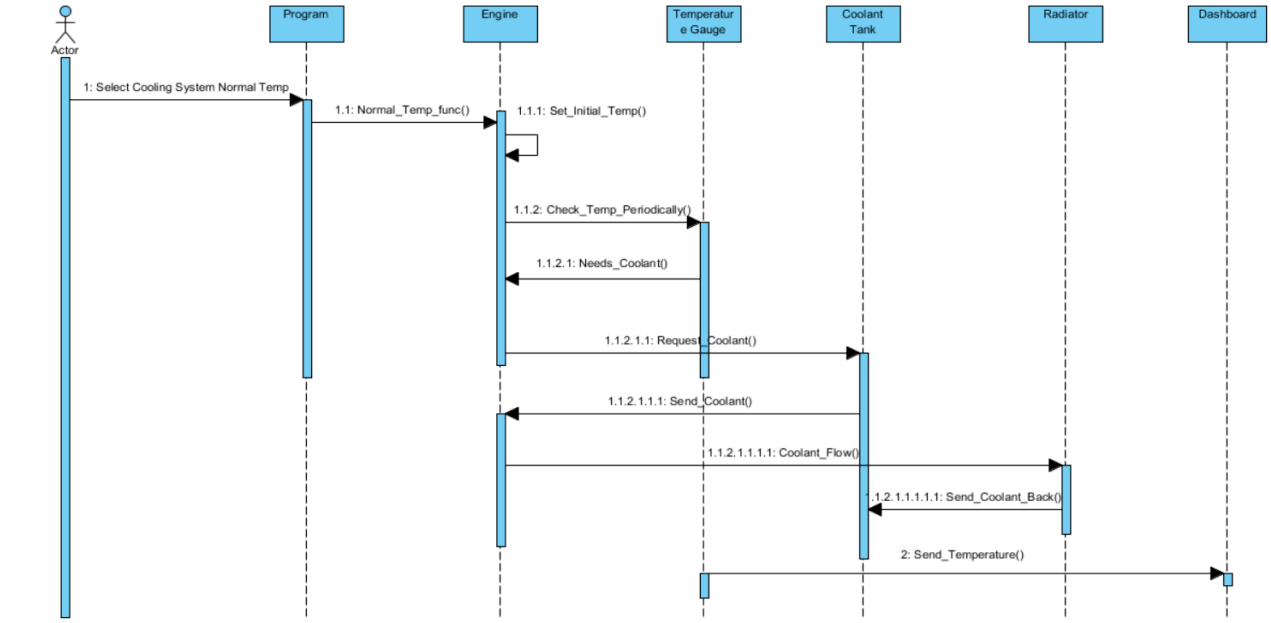
Rear Wheel Drive



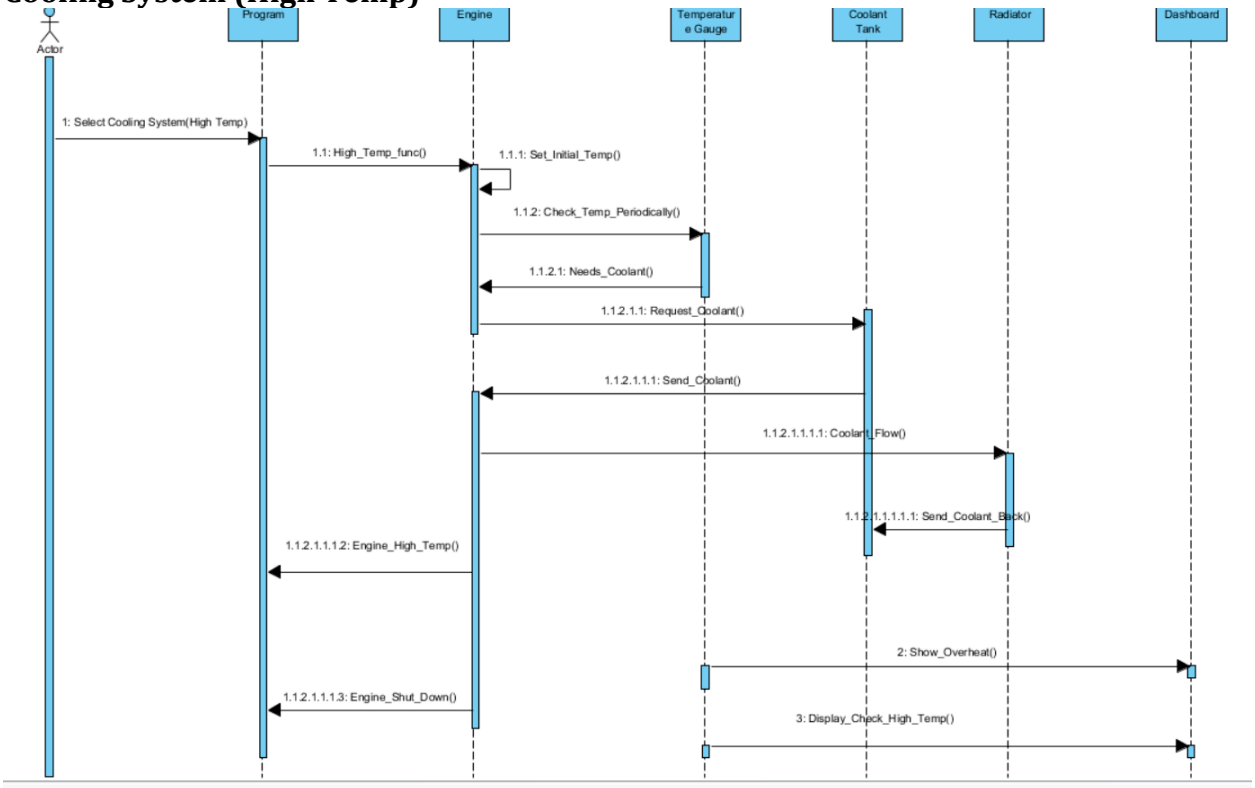
Four Wheel Drive



Cooling System
Cooling System (Normal Temp)

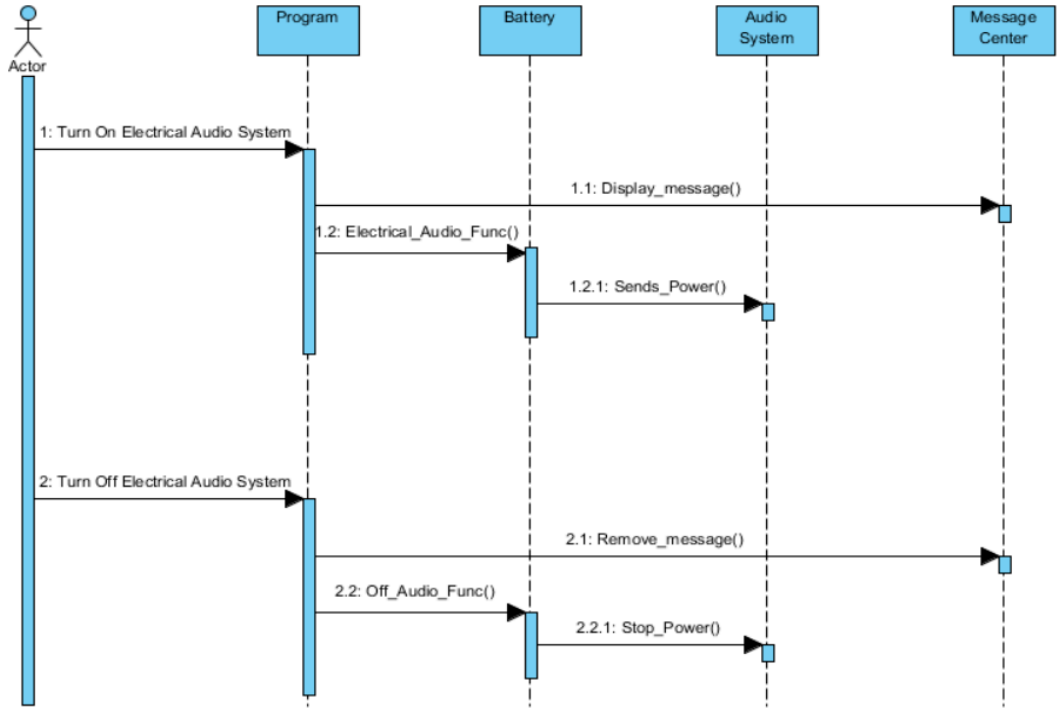


Cooling System (High Temp)

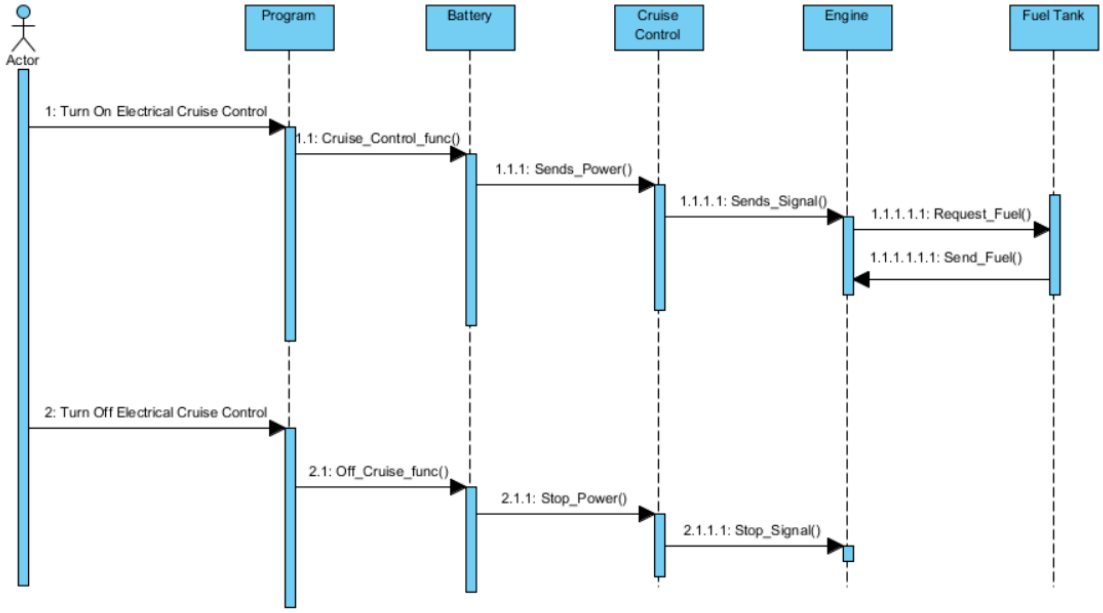


Electrical System

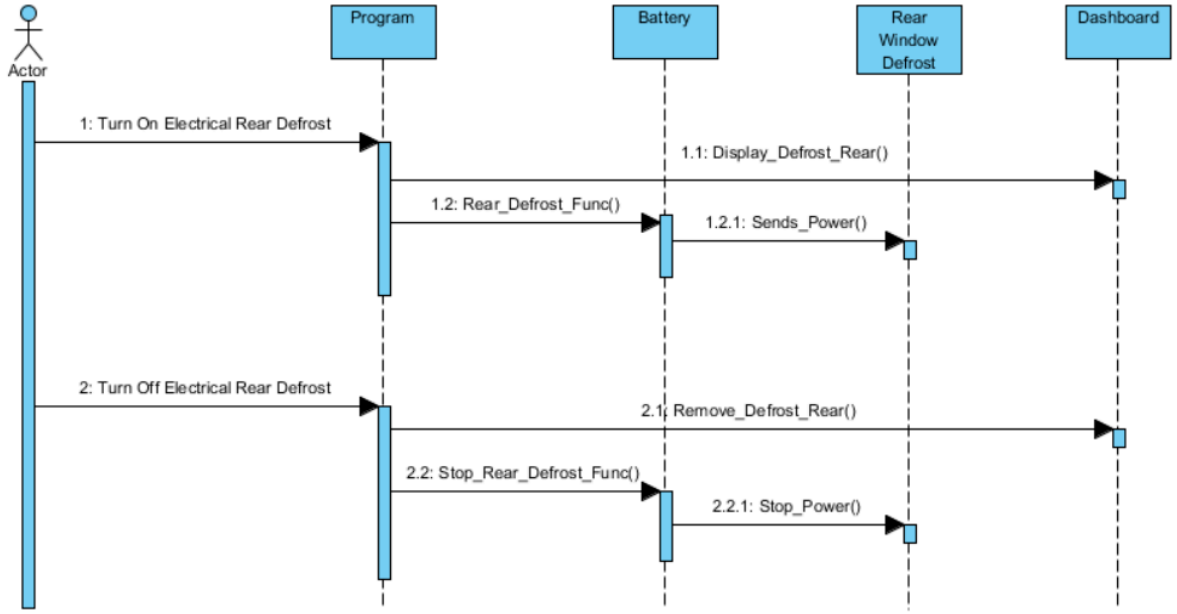
Electrical System (Audio System)



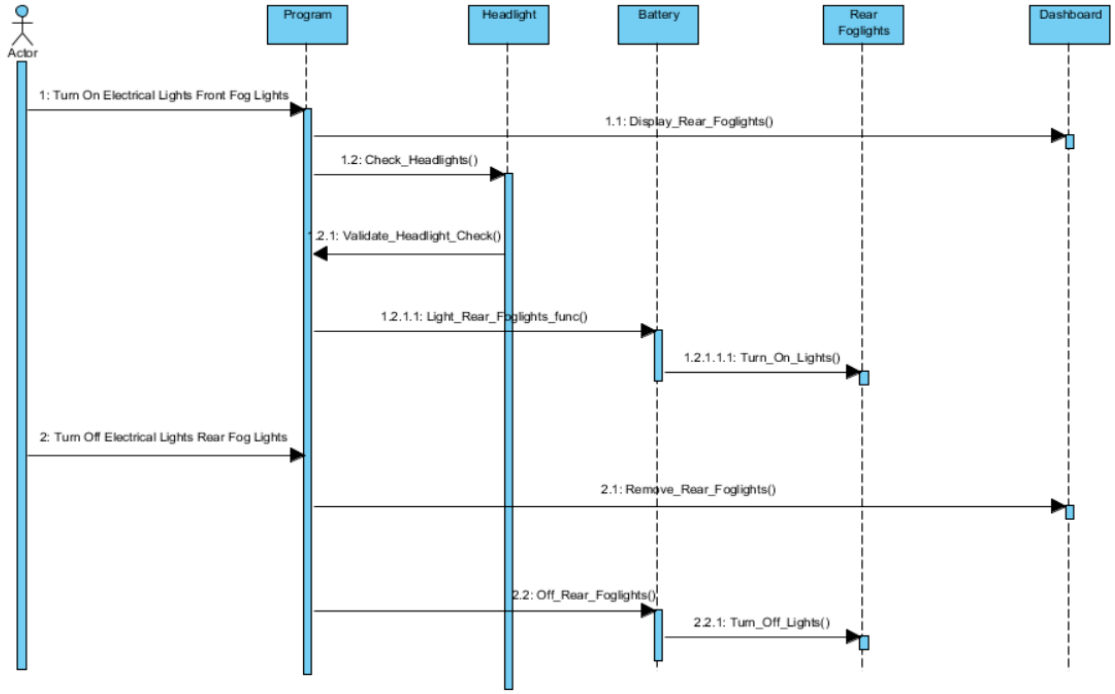
Electrical System (Cruise Control)



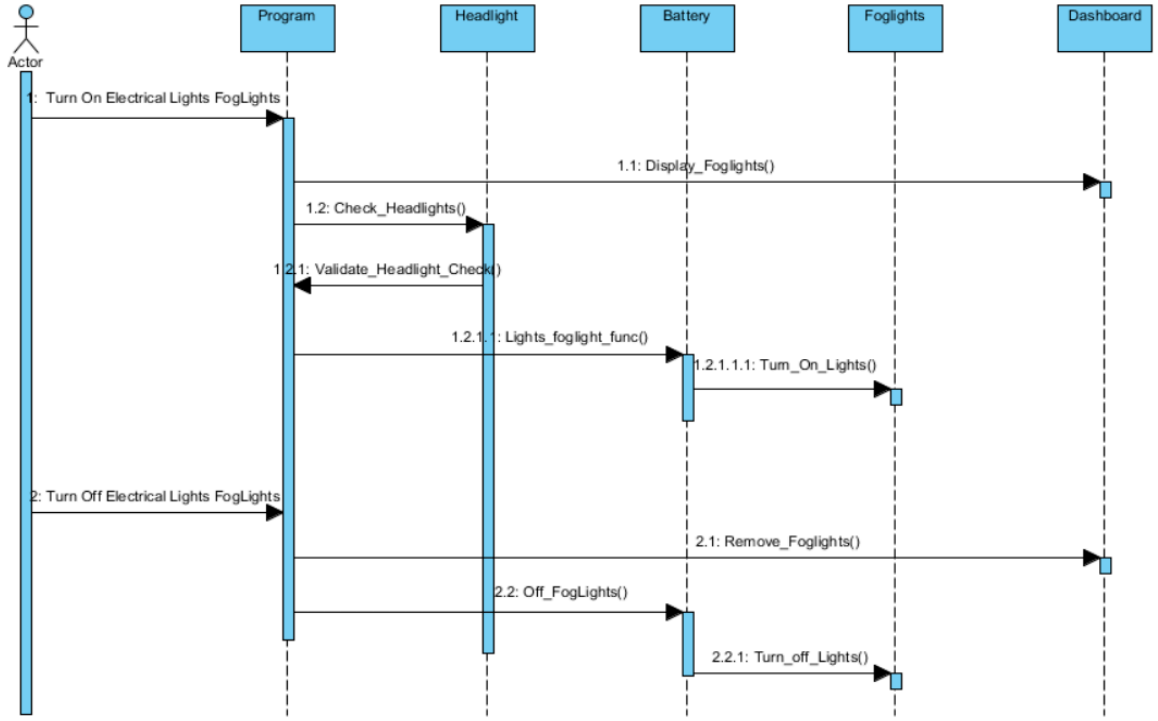
Electrical System (Defrost Rear)



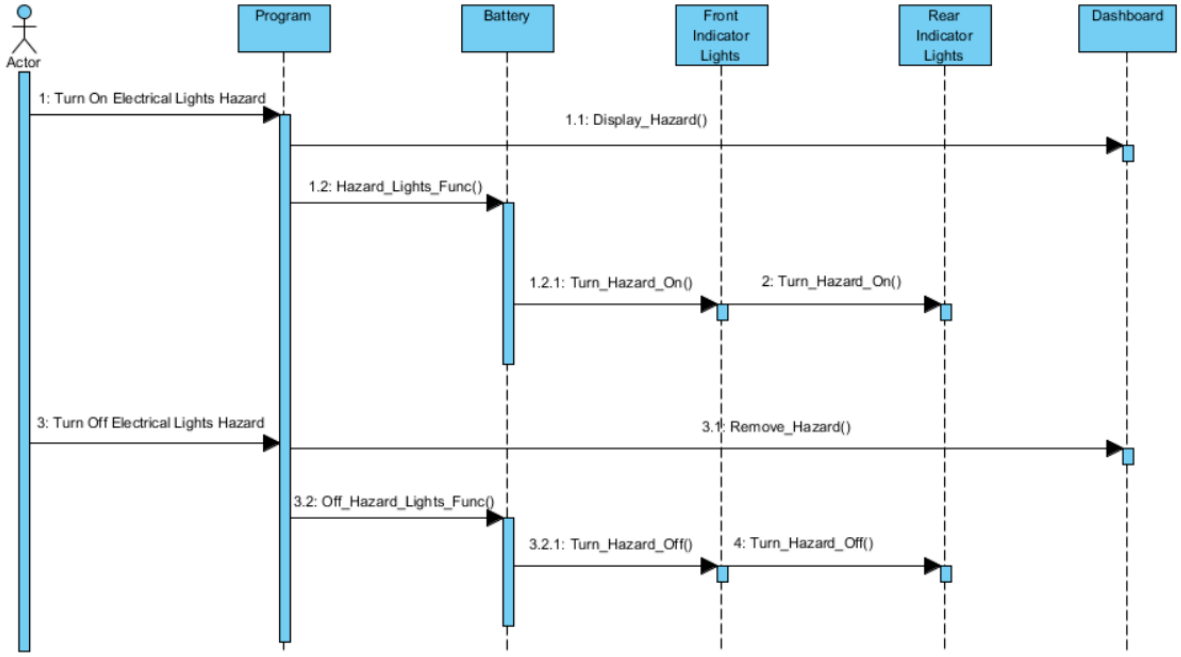
Electrical System (Light Rear Fog Lights)



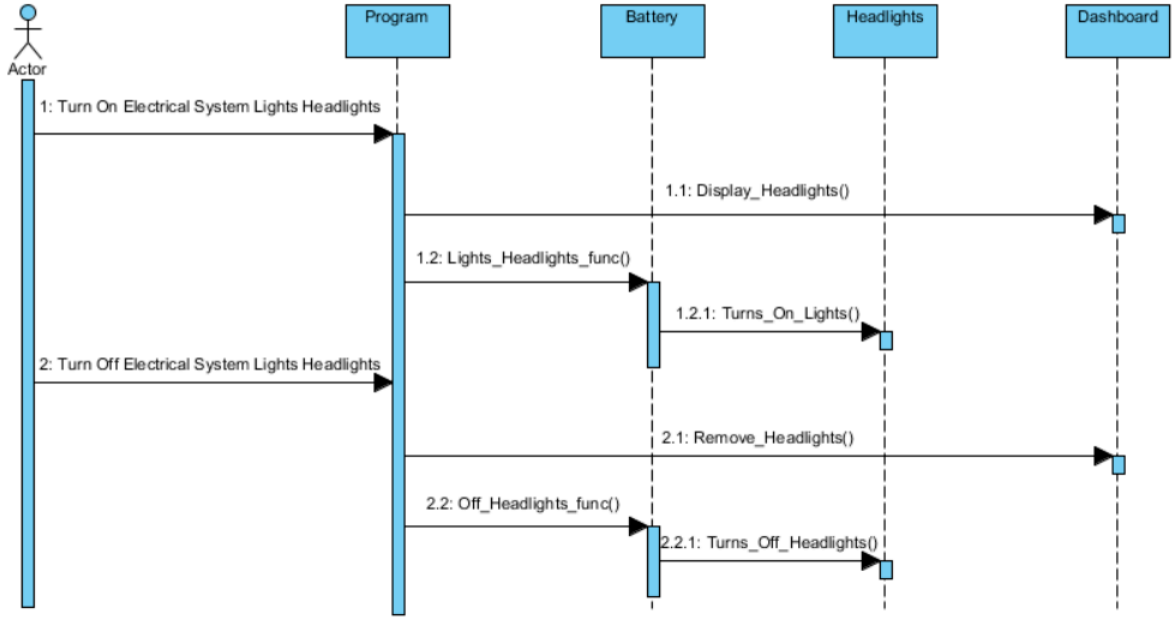
Electrical System (Light Front Fog Lights)



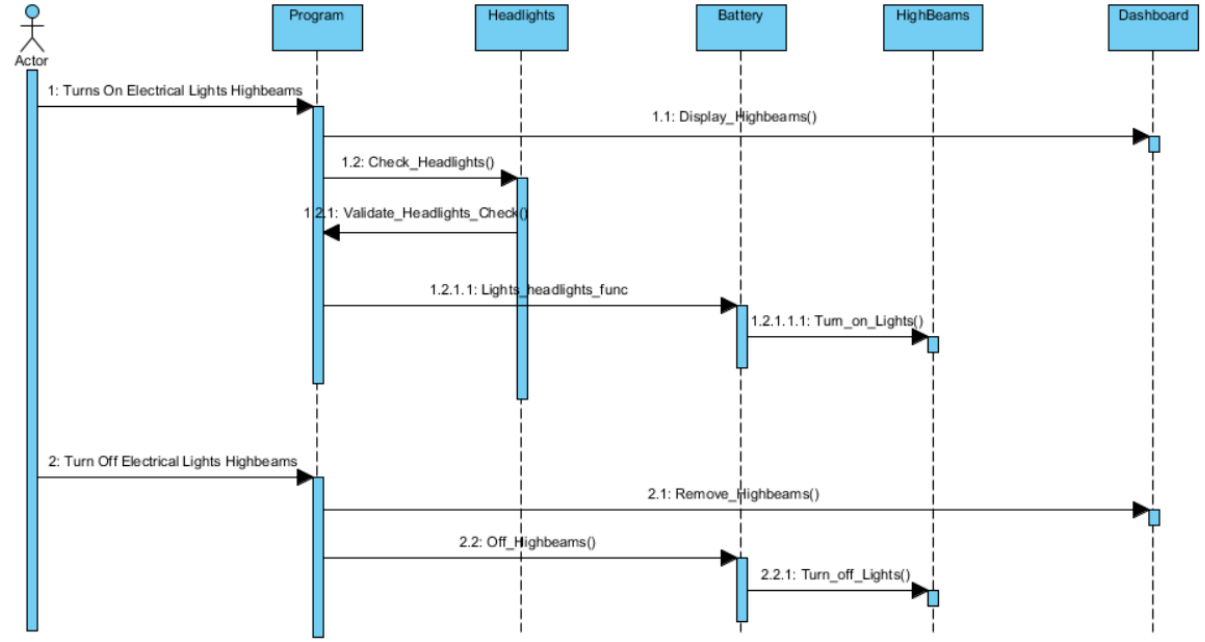
Electrical System (Light Hazard Lights)



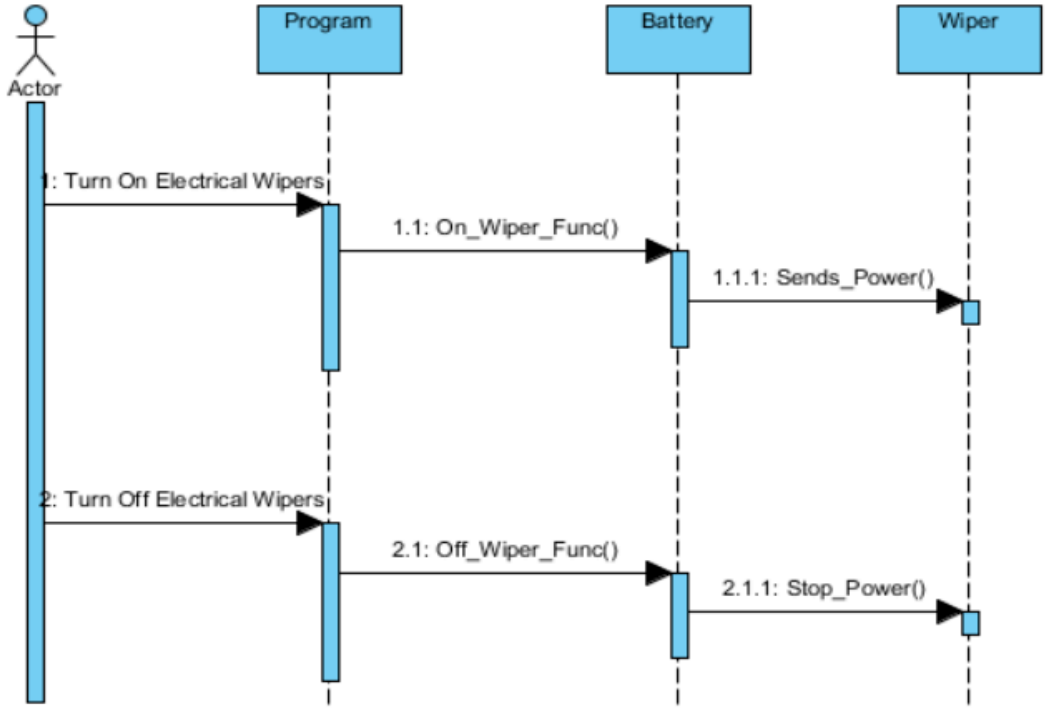
Electrical System (Light Headlights)



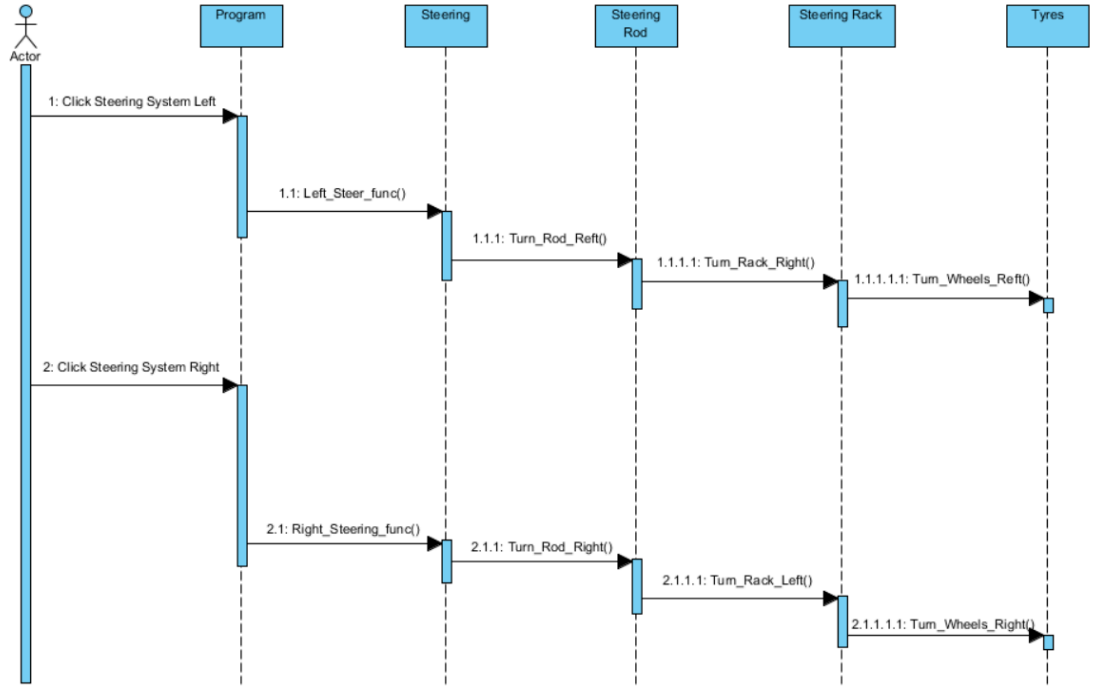
Electrical System (Light High beams)



Electrical System (Wiper)

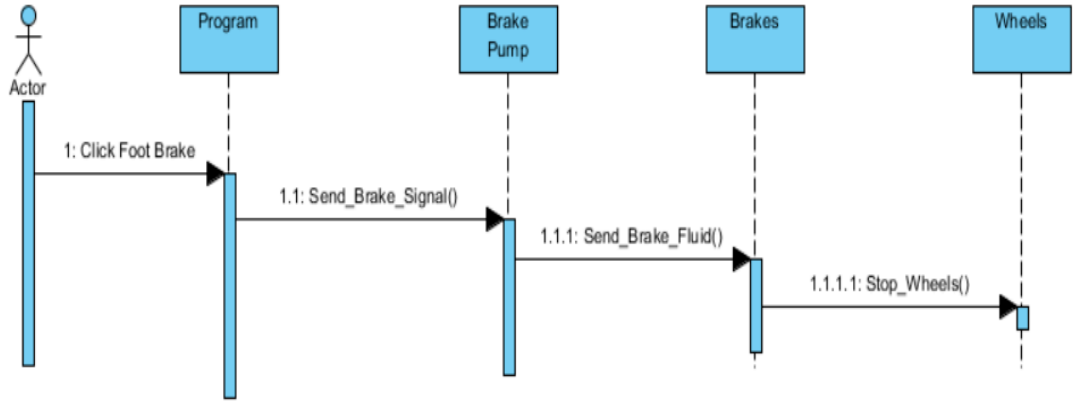


Steering System

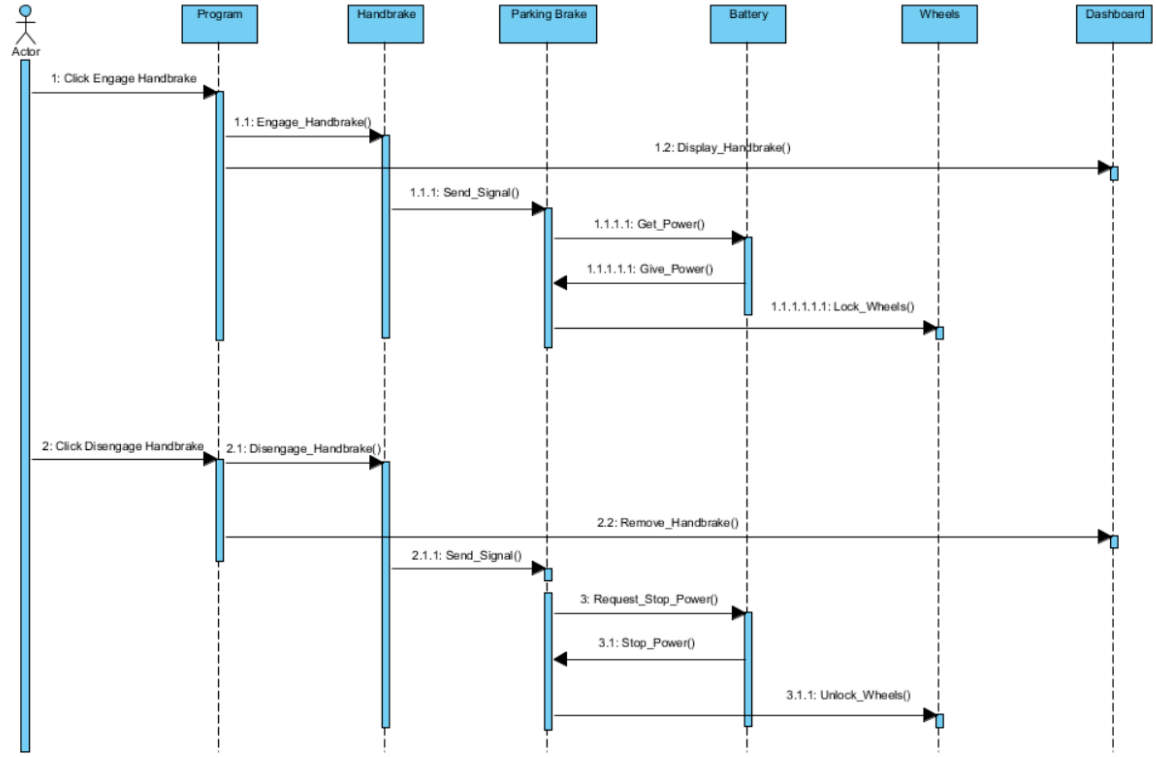


Braking System

Braking System (Foot Brake)

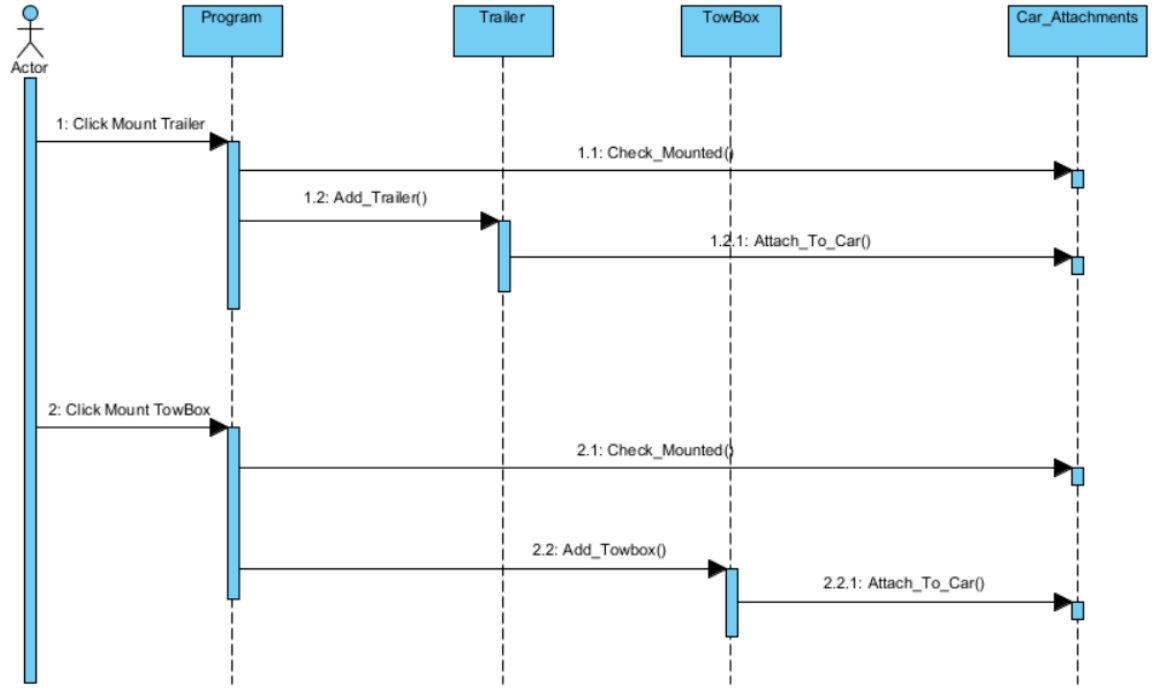


Braking System (Hand Brake)

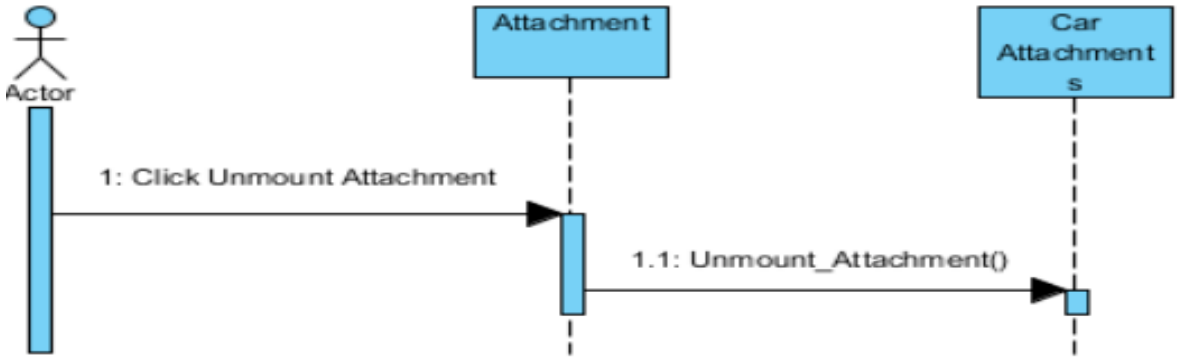


Attachment System

Adding Attachment

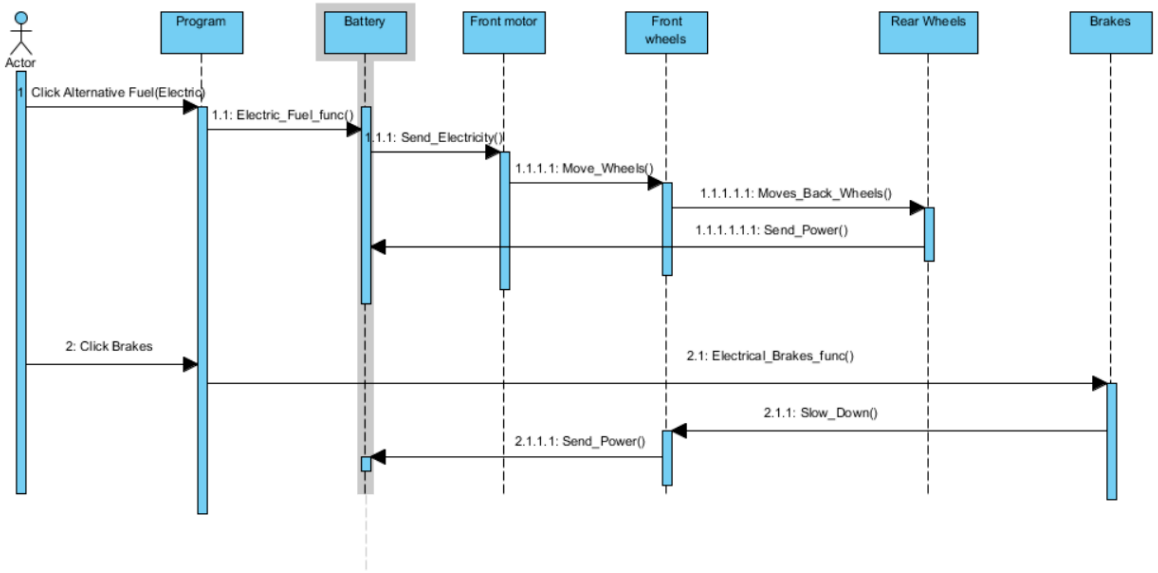


Remove Attachment

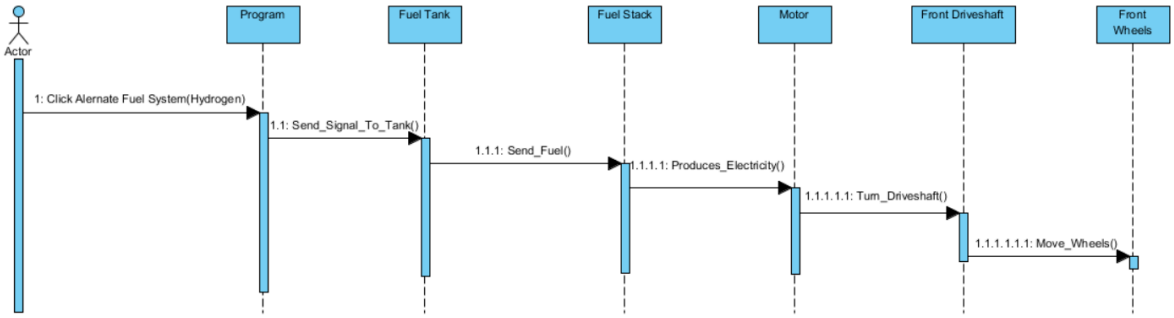


Alternative Fuel System

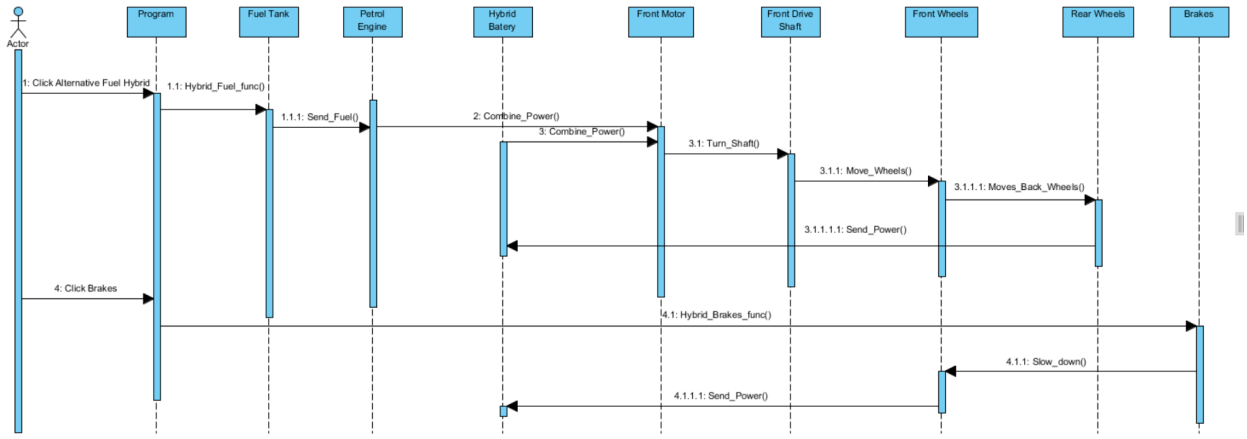
Alternative Fuel System (Electric)



Alternative Fuel System (Hydrogen)



Alternative Fuel System (Hybrid)



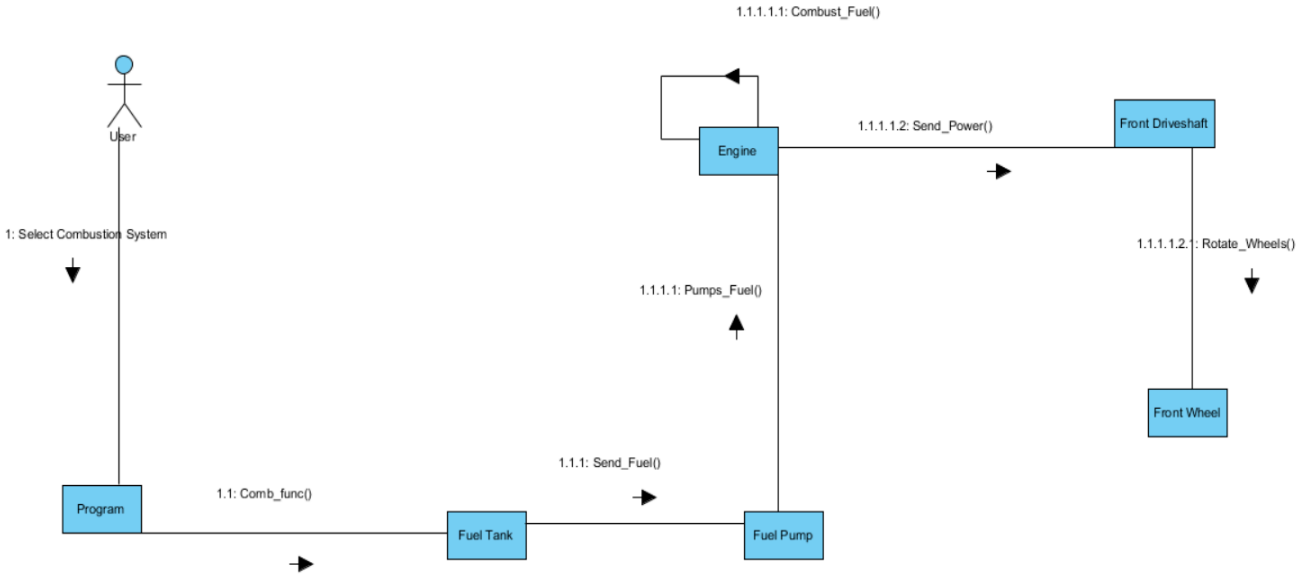
Communication Diagrams

Communication diagrams are in essence, another way for us to view a sequence diagram. It is a complement to the sequence diagrams as the removal of the timeline allows us to see the interactions between the classes in a certain activity, further assisting in the development process.

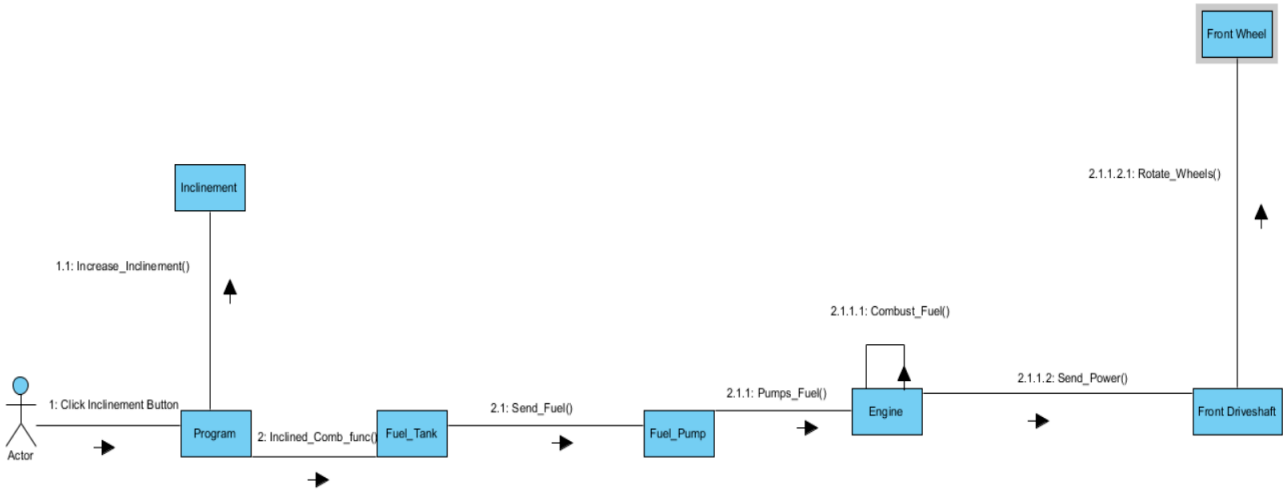
The following pages are the communication diagrams that have been prepared based on the sequence diagrams that have been generated in the previous section.

Combustion System

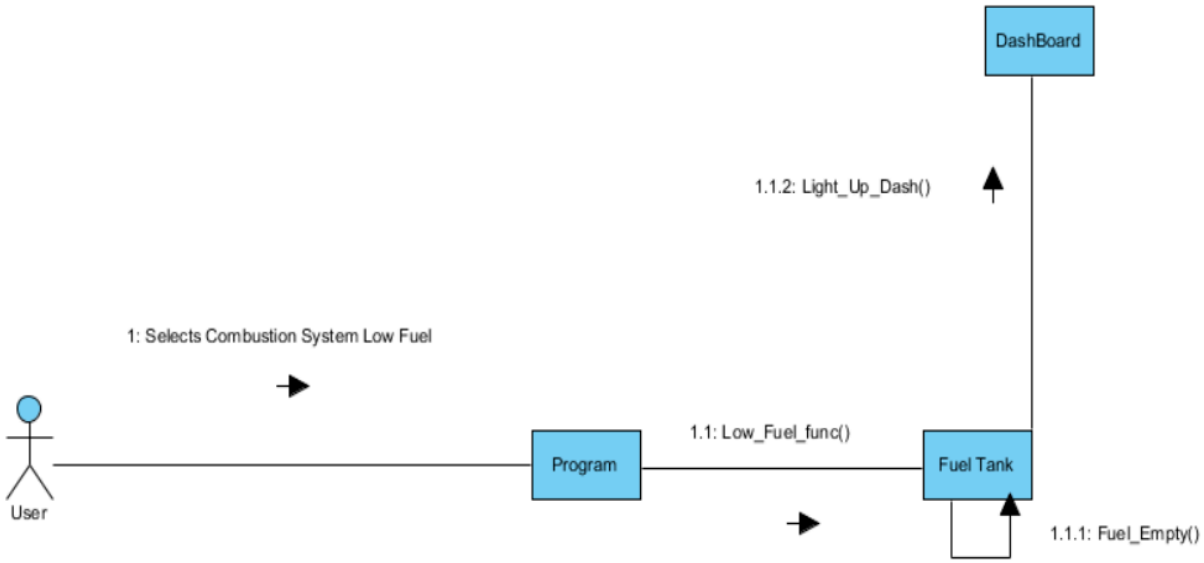
Combustion System (Normal)



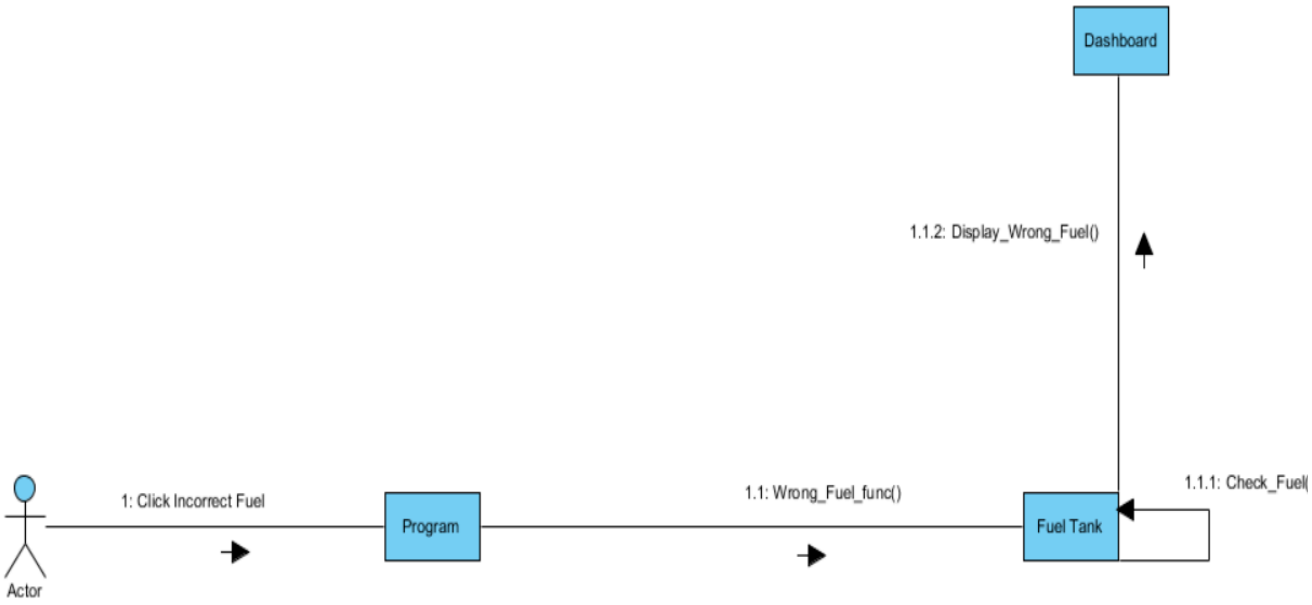
Combustion System (Inclined)



Combustion System (Low Fuel)



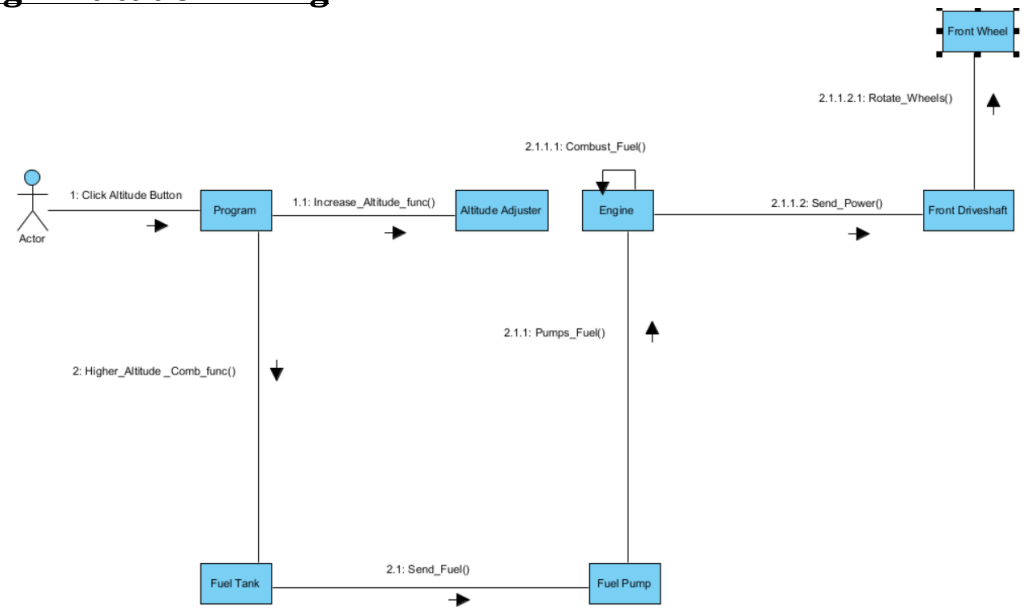
Combustion System (Incorrect Fuel)



High Altitude Driving System

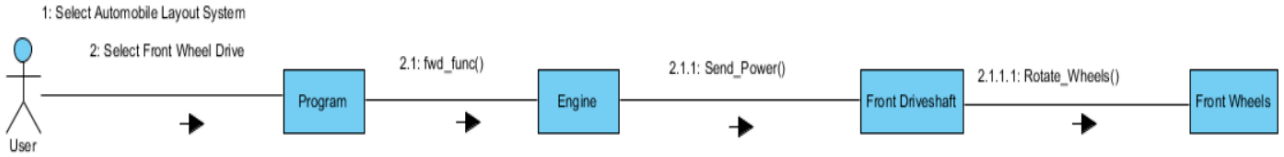
High Altitude Driving System

High Altitude Driving

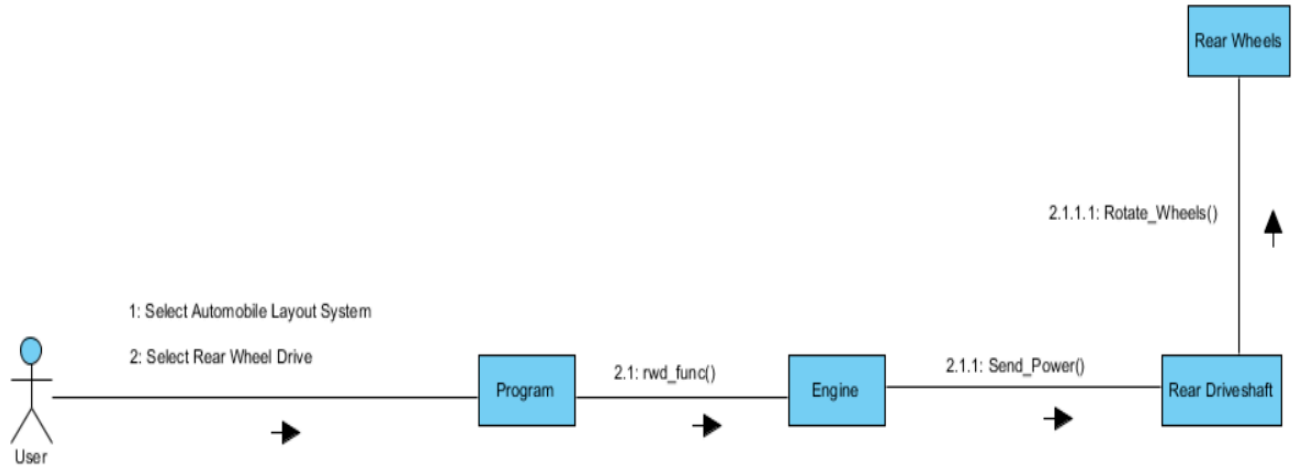


Drive Wheel System

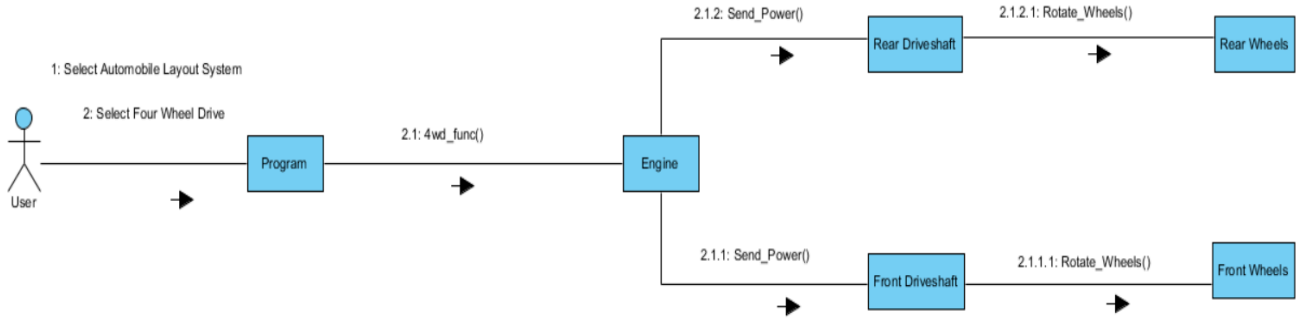
Front Wheel Drive System



Rear Wheel Drive System

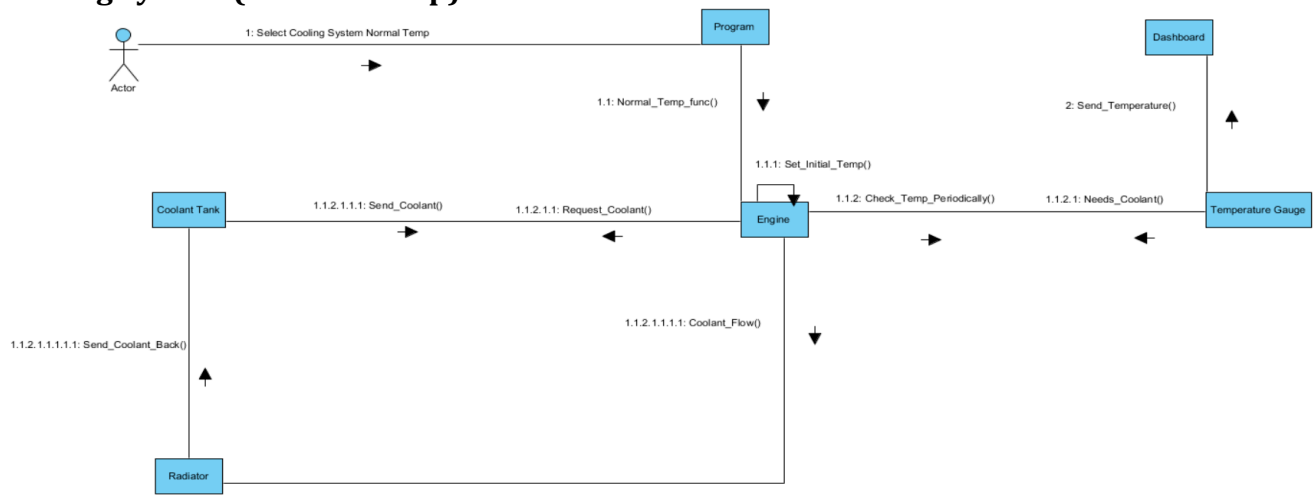


Four Wheel Drive

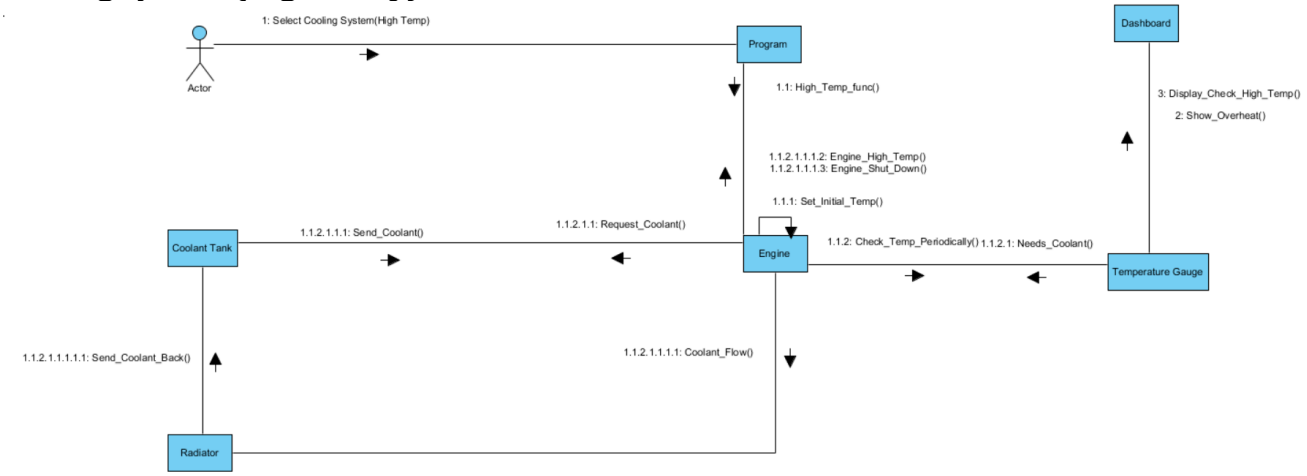


Cooling System

Cooling System (Normal Temp)

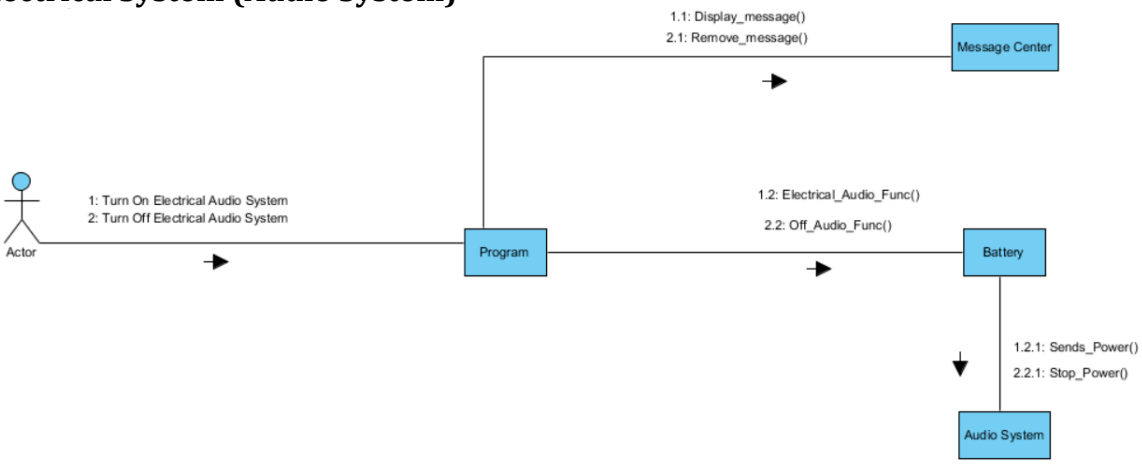


Cooling System (High Temp)

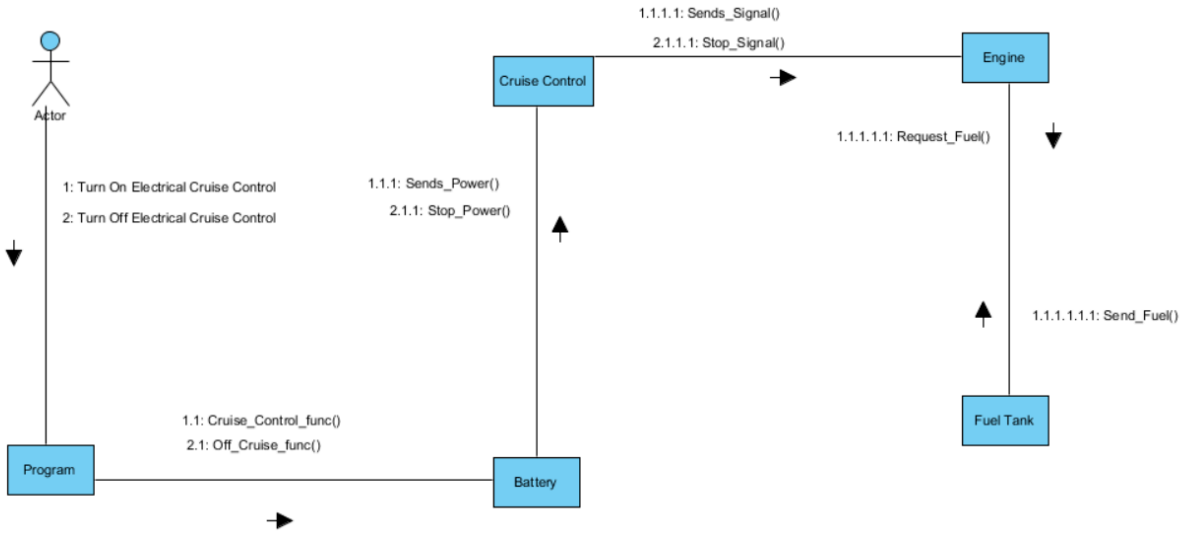


Electrical System

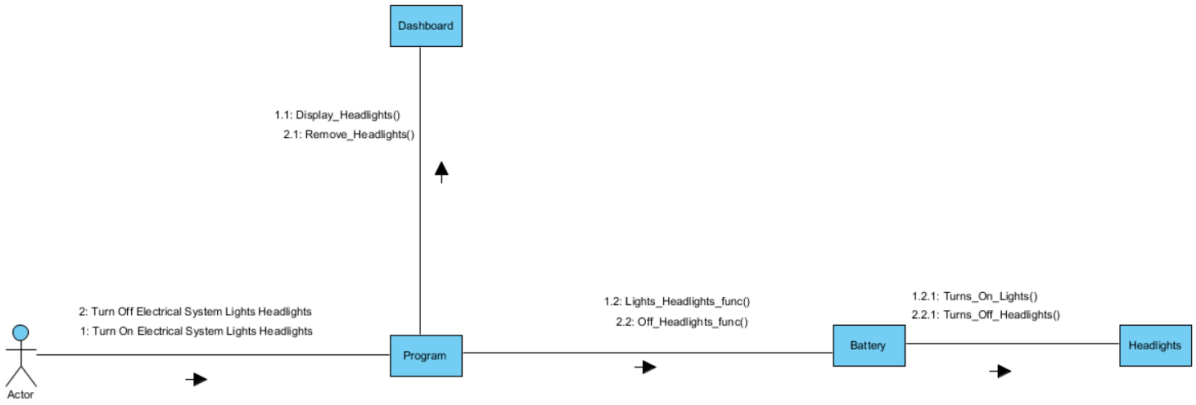
Electrical System (Audio System)



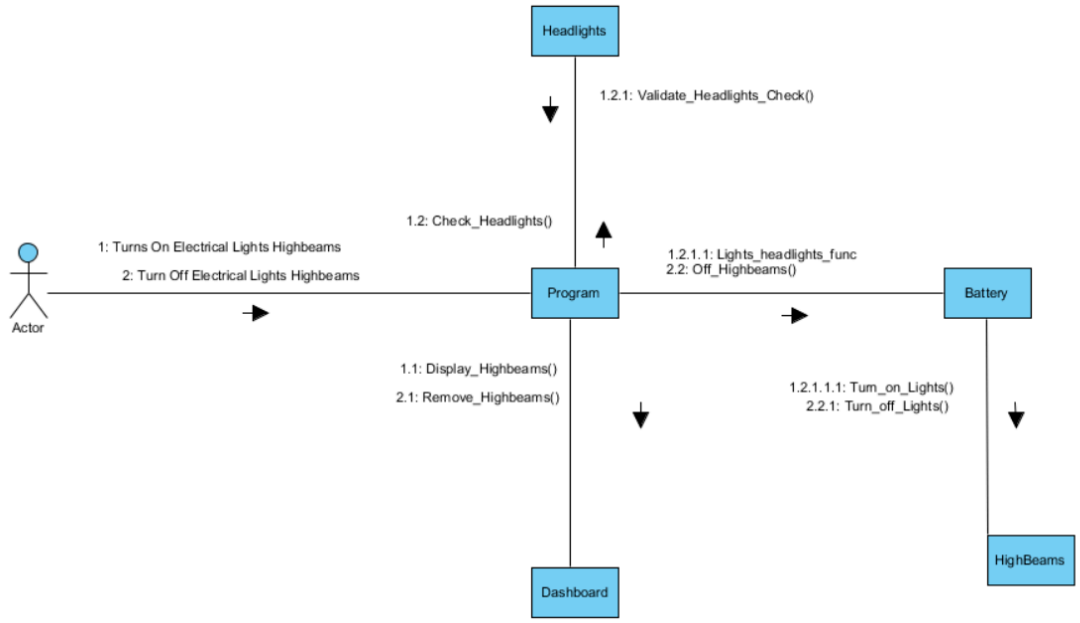
Electrical System (Cruise Control)



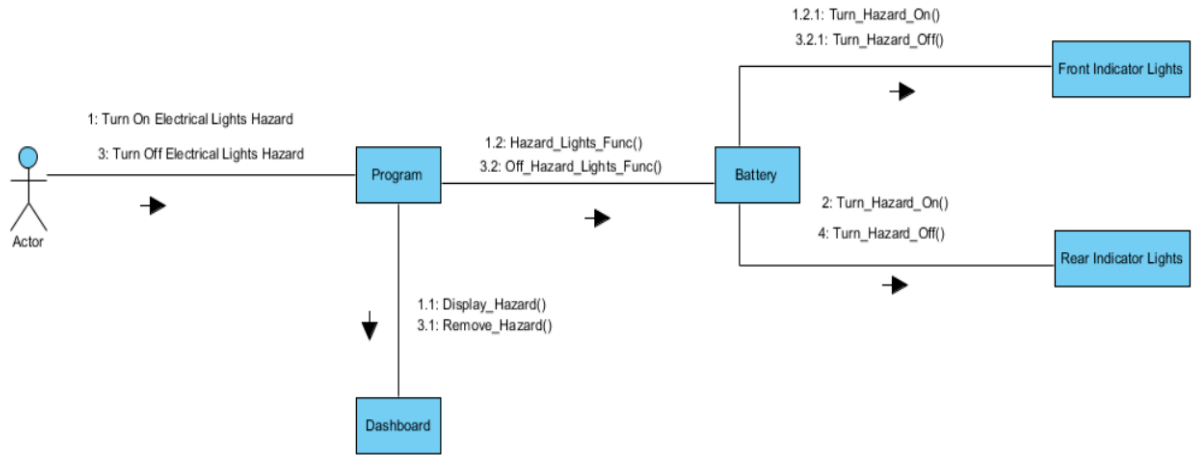
Electrical System (Headlights)



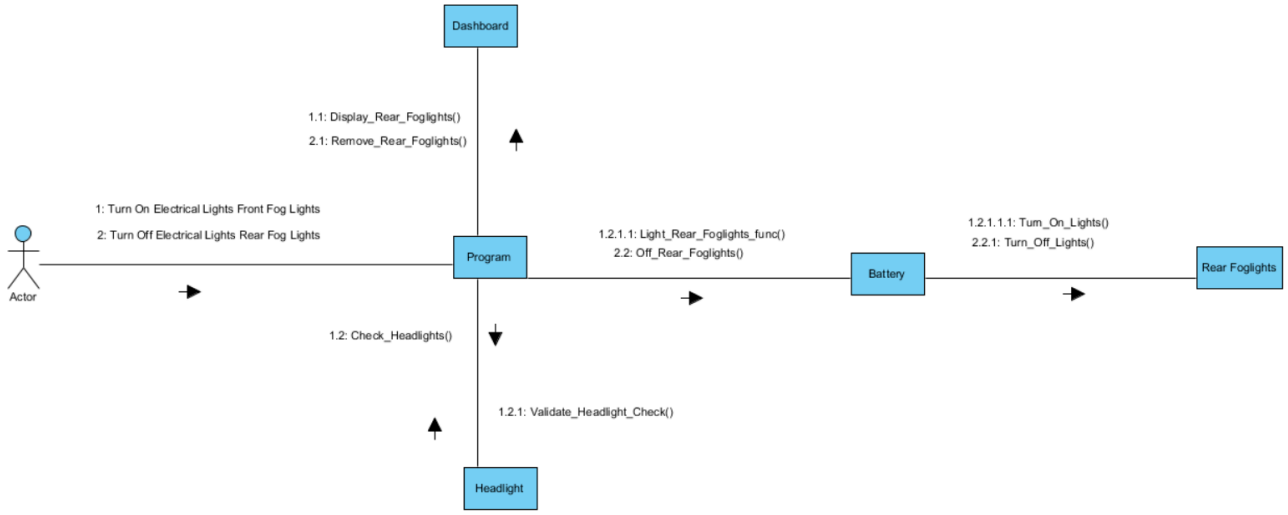
Electrical System (High beams)



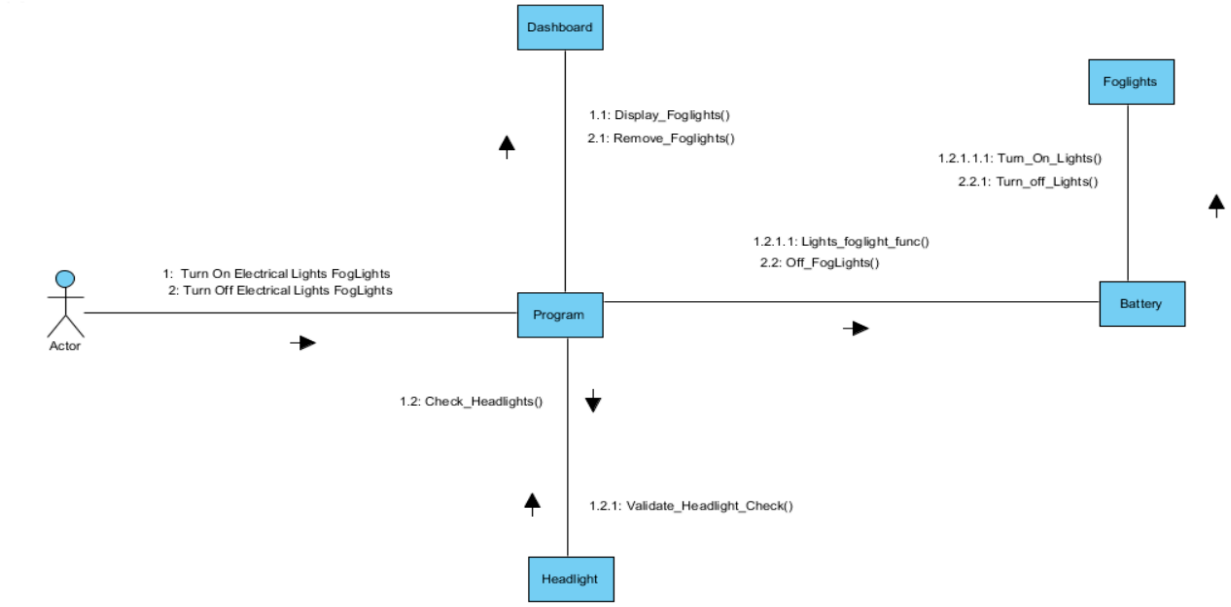
Electrical System (Lights Hazard)



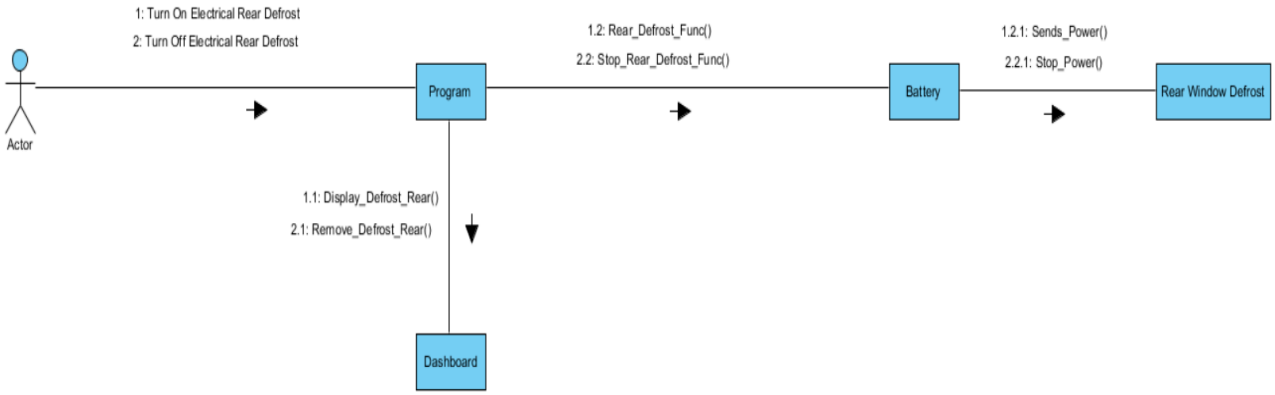
Electrical System (Rear Fog Lights)



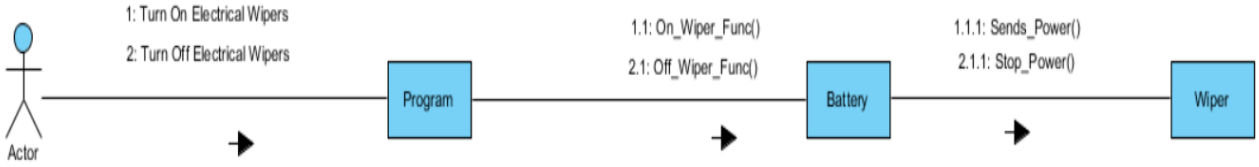
Electrical System (Turn on Front Fog)



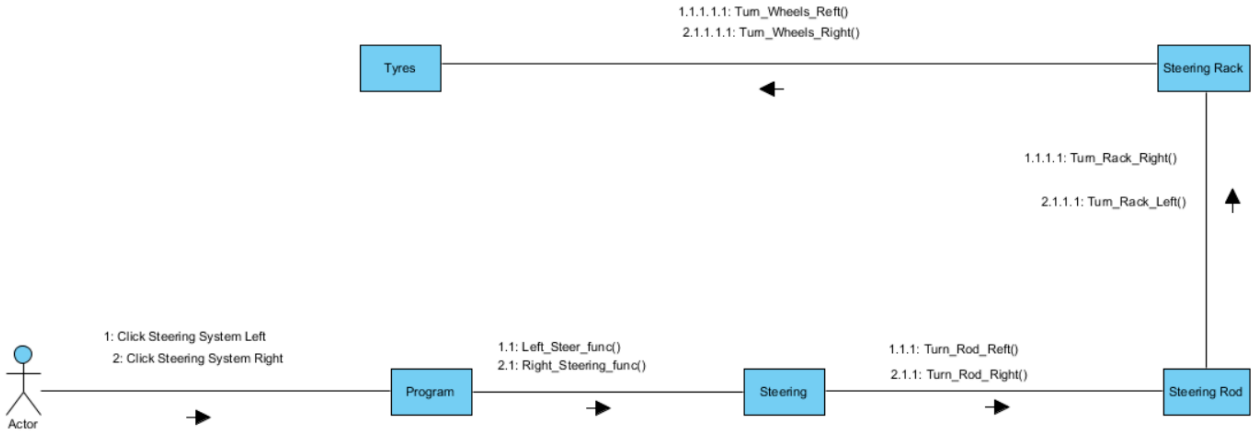
Electrical System (Window Defrost)



Electrical System (Wiper)

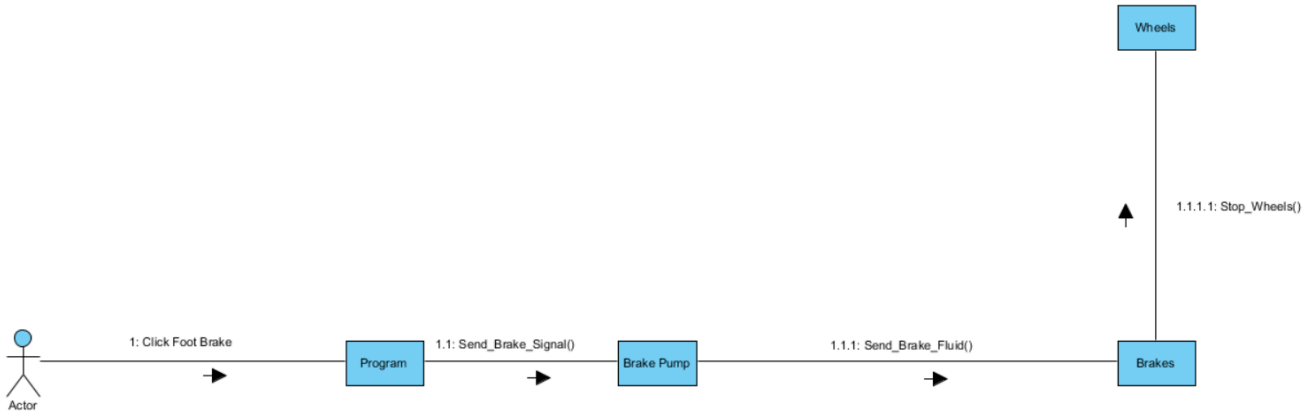


Steering System

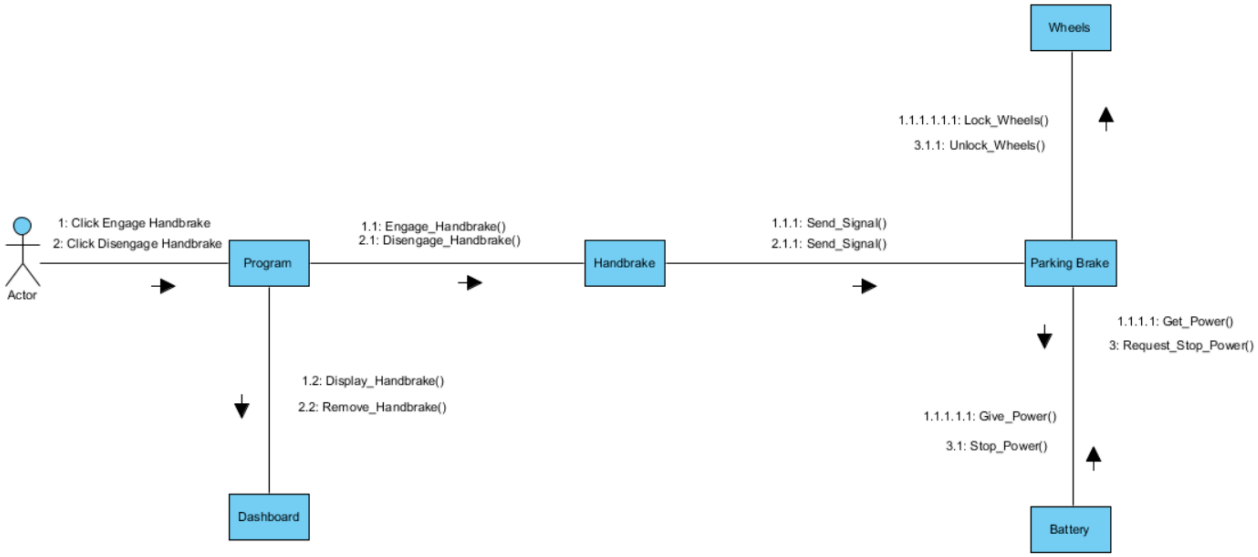


Braking System

Braking System (Foot Brake)

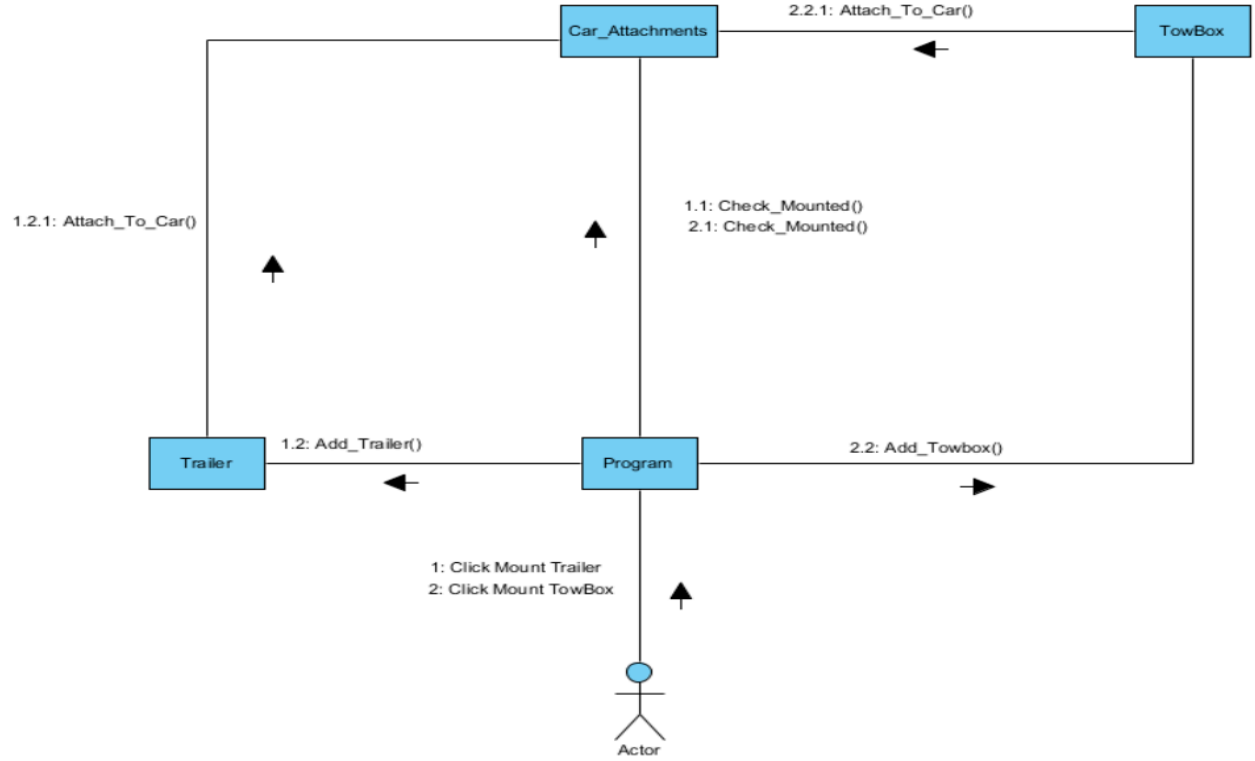


Braking System (Hand Brake)

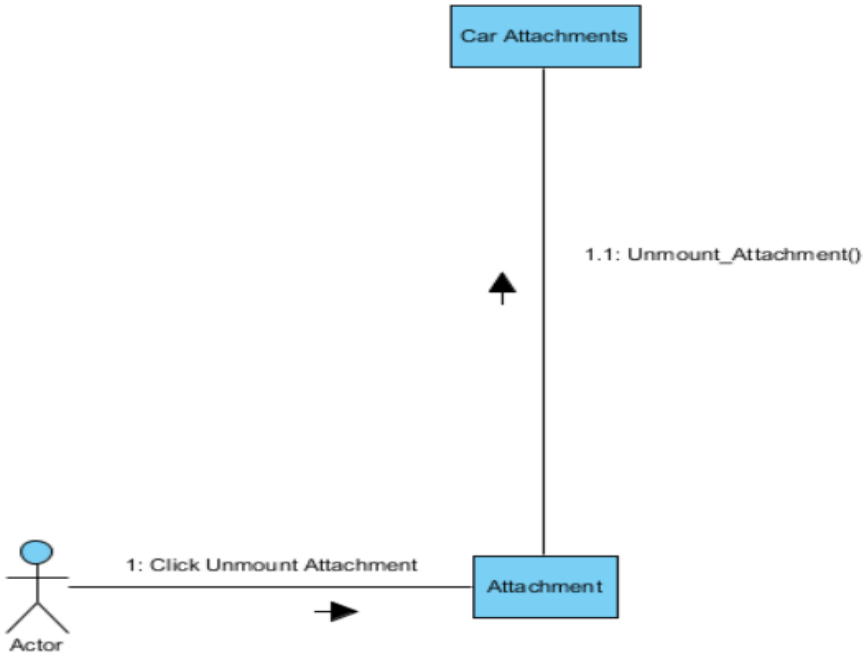


Attachment System

Attachment System (Add Attachment)

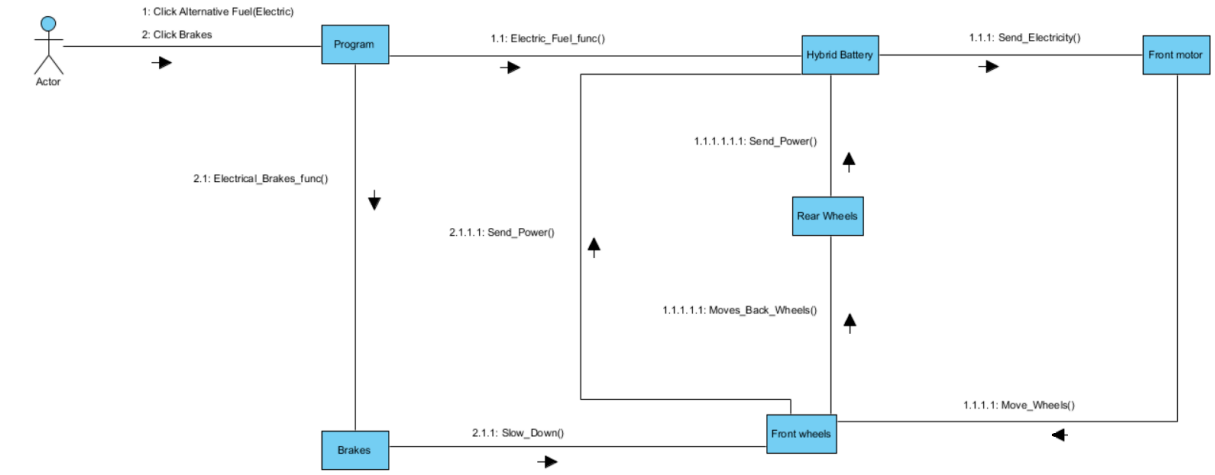


Attachment System (Remove Attachment)

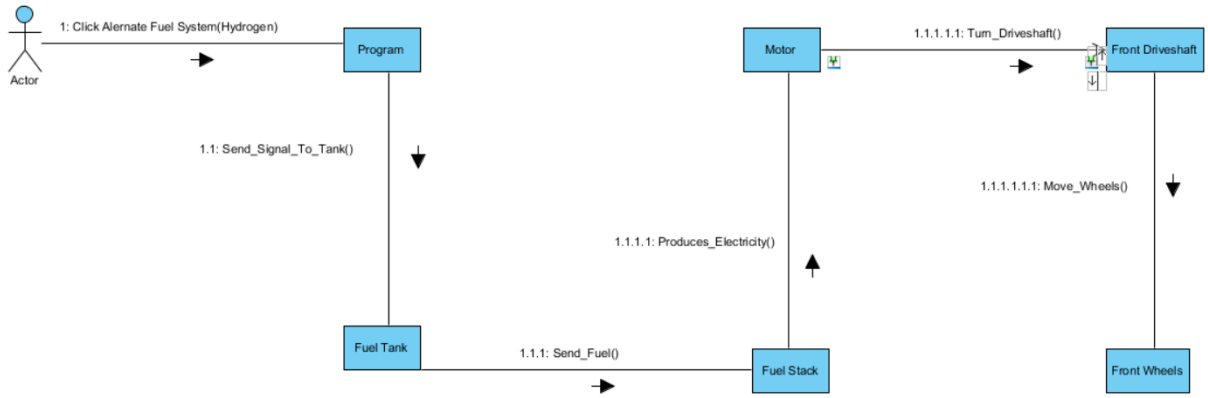


Alternative Fuel System

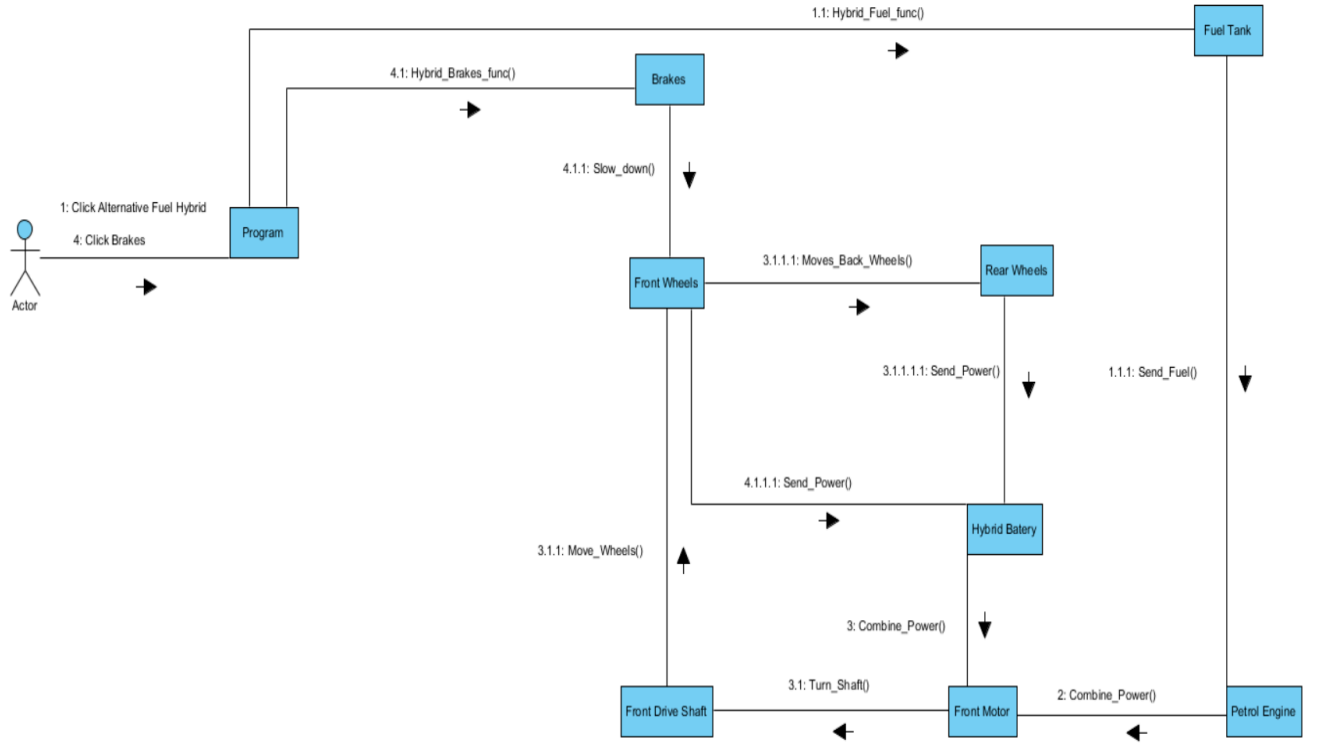
Alternative Fuel System (Electric)



Alternative Fuel System (Hydrogen)



Alternative Fuel System (Hybrid)



Implementing thread management

The core of technology of our application is the use of threads, which allows the simulation to be similar to how a car works, in the sense that the parts of a car are not one large piece, it is made up of individual components that work together, passing instructions or information or even substances to each other for the components to function and realise the feature of the car. Also from the software development standpoint, it allows us to be able to take advantage of the multithreaded nature of modern CPUs in order to improve performance, and to a certain extent, also isolate each subsystem from the other subsystems.

In our application, threads will be used in the following ways:

- Each main subsystem of the application (simulation, infographic, and quiz) should run in their own threads, independent of the main menu screen that allows us to choose which subsystem we would like to go to
- In the simulation subsystem, each component of the car being simulated should be in their own threads, with inter-thread communication allowing us to be able to handle information sharing among the threads.

For the first use of threads, that is the implementation of each subsystem in their own threads, it is not too difficult as all we would need would be to create the thread and call the function that acts as the "starting point" for the subsystem to begin the functionality of that subsystem. As each subsystem is independent of each other, we won't really need to implement any form of information sharing between the various subsystems in the application.

However in the case of having each component of the car in the simulator to be in their own threads, it is not so easy, as we would need to implement some form of information sharing between the threads. For example the fuel tank needs to send a message to the instrument cluster when the level of fuel is low, and the fuel tank will need to supply fuel to the engine for it to be able to run.

Threads make it a bit easier to implement the sharing of information in the sense that threads are able to access all the variables that have been created in the parent function that creates and starts the thread. Because of this, we would be able to create a set of variables that can be accessed by the threaded functions, and the threaded functions can then grab the information stored in those variables to vary their execution. To prevent race conditions, we could implement some mutual execution locks in order to ensure at any one time, only one function accesses the variable. This would be necessary mainly if there is more than 1 function being able to access the variable at the same time, which in the case of our application would be highly possible. For example, if we have a function running in that calculates the fuel consumption of a car by getting the fuel level stored in a variable in the parent function that created that thread, and at the same time we have fuel tank that regularly updates the value of the fuel level in the same parent function, we would most definitely need a mutex lock as

this would prevent instances where the fuel tank updates the value in the variable the same time the fuel consumption function gets the value stored in the variable.

Implementing the mutexes are not that hard however, as all we would need is to use the functions that have been predefined in the pthread library which will allow us to easily setup the mutexes in our application. However, care must be taken during the development process to ensure that once done, the mutex is always unlocked to prevent issues from deadlocks, especially in the cases when an error occurs and if the execution of a block of code between a mutex lock and unlock is aborted before the mutex is unlocked (or the function terminates through a return instruction encased between a mutex lock and a mutex unlock).

Testing methodologies

In any application that we develop, it is essential for us to be able to test it to ensure we are able to ensure it meets the qualities that we have set to achieve with our application. These qualities act as goals, and are a good measure to see up to what degree we have been able to develop the application.
There are a number of qualities we would have to test on for our application, and each of these qualities have different ways to assess them. Among the qualities that we would like to achieve would be:

- Correctness

Correctness is defined as the conformance of the application to the original specifications that have been set out in the application. We can only say a software is correct if it satisfies the functional requirements set out in the software requirement specifications. The process of verification is to ascertain if the requirements are met.

There are a number of ways for us to work towards correctness, one of it being the use of proven methodologies and processes, and even the use of standard algorithms and libraries in our application development process. However this doesn't allow us to be able to verify the correctness of the application.
For us to verify, we would need to take the following steps:

- Formal methods
  - We would try to prove the logic of the code mathematically to ensure the logic of the code is correct and is consisted to the logic set out in our software requirement specification
- Inspection
  - Static inspection of the code can be done to see that the code makes sense. It can help to a certain extent to ensure the code is doing the right thing, like calling the right functions or using the correct data types
- Testing with dynamic tools like CREST
  - Because it is difficult for us to manually generate various test cases, it would be possible to use some dynamic testing tools to test the code. It can help to provide better code coverage, with automated testing capabilities
- Blackbox testing
  - We can use blackbox testing in order to ensure the output we obtain is consistent to the output we expect to get out of the system. This could be a good starting point for us to begin our correctness testing.
- Whitebox testing
  - Whitebox testing on the other hand will allow us to be able to actually inspect the inner workings of the application. It would be best for all the code that we have written to undergo whitebox

testing as it allows us to ensure that the program is truly giving out the correct values. It would be the second step in testing after blackbox testing. While the usage of dynamic testing tools and code inspection may take some time, as we have not really done it before, we could use whitebox testing in code blocks, which are of appropriate length for us to test by this method.

- Reliability

Reliability is a relative concept, that is, it is the measure of dependability of our software. There is no fixed threshold that defines unreliable software; hence we would need to depend on creating one ourselves.
For our application, reliability testing can be done after we are sure the program is correct. The best way to do reliability testing would be to measure the number of times the application crashes over a certain number of attempts to run the application. For our application, it would be reasonable for us to set a benchmark of 2 failures per 10 attempts to run the application.
As it is possible for reliable software to be incorrect, it would be necessary for us to ensure correctness is achieved first. In most, if not all cases, if we achieve correctness, we indirectly achieve reliability as well.
We could also potentially introduce a beta release of our software to be used by a select number of people in order to simulate random user usage. As different users will be running the applications through different steps (like some users may start the simulation before starting the infographics section, and vice versa), it gives better, more varied input to ensure our program is truly reliable.

- Robustness

Robustness is the ability for our application to be able to cope in unspecified situations, which is the application is able to handle even unexpected input.
The best way to test this is to supply our application with input that is close to the correct bounds of the function, to see if we are able to trigger the error detection functions and to see if the program is able to handle the incorrect input. For example in the case of vehicle speed, we would try to enter in a negative value or an extremely large value, beyond the range of speed values that are accepted by our system to see if it is able to handle this incorrect values properly.

- Performance

Performance is mainly the efficient use of resources. As every computer has a finite set of resources, we would need to ensure that performance of the application is well optimized. Especially in our case where we are making use of threads, we would need to ensure that threads are well managed to ensure we don't have any runaway threads wasting the computer's resources.
We should set a benchmark for the performance of our application, so that we can compare our application's performance. For now, it would be reasonable for us to set that our application should not take up more than 20% of CPU usage on a computer, with a maximum usage of 512 MB of RAM at any one time.

- Usability

Usability would probably be one of the most important qualities we would need to achieve in order to consider our application to be a success. Usability is essentially the level of user friendliness of our application. The best way to test this is by running surveys on a select group of people to use a beta version of our user interface to collect feedback. We could also test to see how long would it take for a user to learn how to use our application and be able to use it fully. In our application, we could do this test by getting a group of users who have sufficient general computer knowledge to use our system after reading the user manual 2 times. This limit can be altered later on once we are closer to the completion of the first beta of the application, as we would need a beta release to conduct this form of usability testing.

- Portability

Another relatively important quality we have to consider is also the portability quality. One of the main things with Windows machines is that it varies in terms of hard ware from machine to machine, mainly because Windows is able to work on a diverse range of computers. In addition, as we are supporting Windows 8, Windows 8.1 and newer versions of Windows, we would have to ensure we run our application on all these versions of Windows on the same hardware first. Once we have succeeded running it on multiple versions of Windows, we could then look at running the software on various hardware. At least, we should ensure that the software runs without any issues on the lab computers made available to us.

- Effectiveness

Testing effectiveness would be important in our application as our application is designed to improve the user's knowledge on cars. In the event our application shows poor effectiveness in improving the user's knowledge on cars, we would have to seriously reconsider the way we are presenting information to the user (i.e. the way the user is taught about the various parts and functions of the car). One possible way for us to test effectiveness is by conducting some surveys on the users who have tried out our application to gauge feedback on their experience with the application, in order to see if they found the application effective in improving their knowledge on cars. We may need to ensure that we get a varied set of respondents for the survey as having only a certain group of users (like predominantly people who are well versed in cars) may skew the feedback we get. In addition, monitoring the quiz results of the users who tried out our application would also be a great way to see if the application was effective in improving the users knowledge, as the quiz is essentially a tool to test the user's understanding on the parts and functions of the car. We would have to get a pre-usage quiz result and a post-usage of application quiz result from every user that is part of our study, so that we can see the difference the application made in the understanding of the user, as an increase in understanding would indicate a positive improvement in the quiz results.

By achieving the abovementioned qualities, confirmed via the suggested testing methods as above, we would be able to guarantee that the product produced at the end would be of high standards and would be highly marketable.

Code conventions

Code conventions are essential especially in developing software with multiple developers working on it at the same time. One of the reasons for this is because each person has a different style of coding, just like a language, where different people have different writing styles.

Having this form of consistency does bring some advantages to the table:

- Any programmer will be able to get into the code that is being developed and would have a clear understanding on what is going on in the code
- Less mistakes can be made because we have consistency
- Should the project be developed further in the future, by ensuring code consistency, one can get a good grasp on the code easily.

The following conventions have been defined for this project:

- Header files
    - Avoid the use of inline functions as they are quite vague at times
    - The order of parameters in the function should be input parameters, followed by output parameters
    - Avoid the inclusion of namespaces in the header files
    - Header file format should be .h
    - Include the C and C++ libraries first before including user defined header files
- Code files (CPP files)
    - Avoid the use of static and global variables as much as possible
    - The use of classes is encouraged over structures. Data variables should be made private.
    - Avoid operator overloading
    - Functions used should be short and not be overly complicated
    - Avoid the use of exceptions
    - When doing type casting, use static_cast
    - When doing sizeof, run it on the variable itself, not on it's type
    - Be careful when using a C++11 language extension, make sure the compiler will be able to compile the application
    - When coding, it is good to compile and test as every function is developed
    - Avoid the use of unstructured programming
- Naming variables
    - Avoid the use of abbreviations, unless it is a common one like max for maximum and min for minimum (for example)
    - No underscores in variable names instead separate words with the use of capitalisation of the first character of each word (like myVariableName).
- Commenting
    - It is essential to write the comment that describes what the file is at the top of the file
    - For each function in the header file, comment to tell what the function's purpose is
    - When declaring classes, comment on the purpose of the class

    - Comments are generally not necessary for variable names as they are quite self explanatory, however if necessary then it is OK to have some comments to describe the purpose of a variable
    - In the implementation, have comments that tell what the implementation code is trying to do in logical positions. It is not necessary to have comments on every line
    - Try not to fit long comments on a single line. Instead split them up into multiple lines
    - Comments should have clear punctuation and grammar and should be meaningful
- Formatting
    - Code should be clearly indented
    - Use tabs instead of spaces for indentation
    - Else keyword should be on a new line
    - Do not have spaces around a period or arrow for pointers
    - When writing blocks that make use of the curly bracket({}), ensure the open and closing brackets are on their own line, for example:

```
int main ()
{
        cout << endl << “Hello” << endl;
        return 0;
}
```

By maintaining to these conventions, we can make the development process a smoother experience.

UI Design Process

When we set out to design Drvr, we wanted to make sure it was a fun and interactive application. Our app had to be not only simple to use, but it should also appeal to a large part of our target market, and the target user should be properly engaged when using our software. One of the reasons for this is that we truly believe that we would be able to make a change in the way people learn about cars, and people will genuinely become well versed about their cars with the use our application.

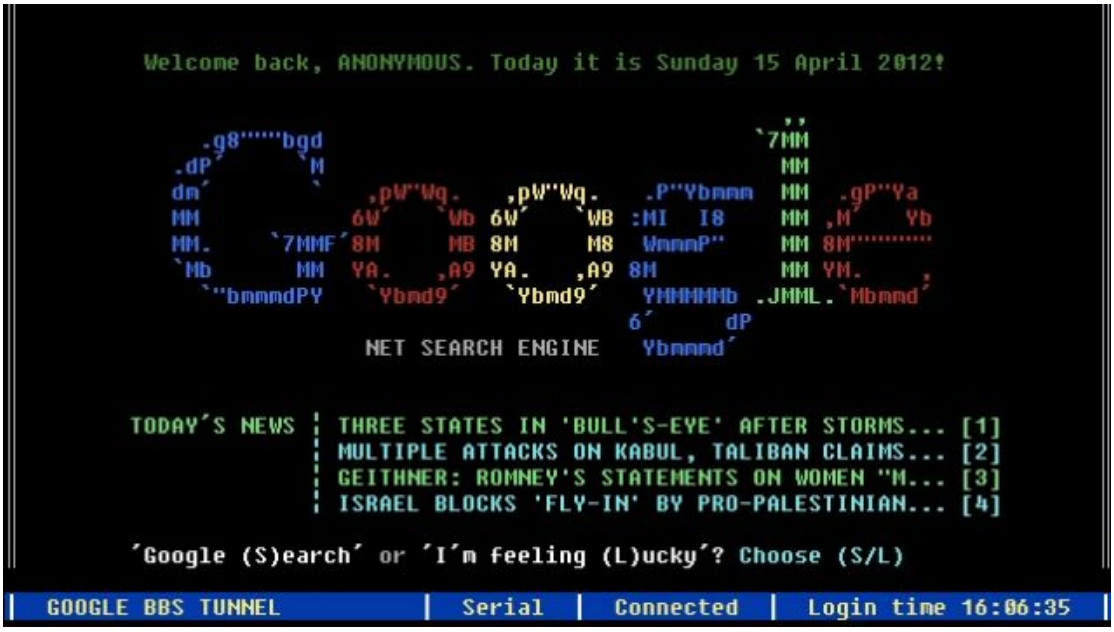
A line that sums up our idea is none other than the design philosophy of a car manufacturer, Volvo. Volvo has a very unique design philosophy, which is "The development of all Volvo products and services must spring from the needs of the people and end with their satisfaction". In fact, their latest campaign sums up the importance of people centric design, expressed by the following transcript:

"We love technology, only if it makes life less complicated for people. Some say cars are all about going fast. Some say cars are all about looking cool. They aren´t wrong. But our main passion is to help our drivers. To help them take responsibility for the world around them and to help them live life less complicated. Cars are driven by people. And that´s why we design them around you."

From the statement above, it was clear that we had to ensure the level of usability has to be very high. The thing about design is that if it isn't done right from the start, once we were to go ahead with the development process it would be very difficult to modify it, as the number of changes that would have to be made would be immense. Another issue is that every time you modify the interface, it is highly likely that you would run into a series of troubles, and this is a fact as per what we have been taught in CSCI318 Software Engineering Practices and Principles, that is currently running in the university.

Core design: Graphical user interface

It was given from the start that we would have to have our application to ship with a graphical user interface. It is known that professional computer users prefer not to use a GUI due to the command line interface being faster to operate and can run even on low powered machines. However faster is not always better, especially in this situation.



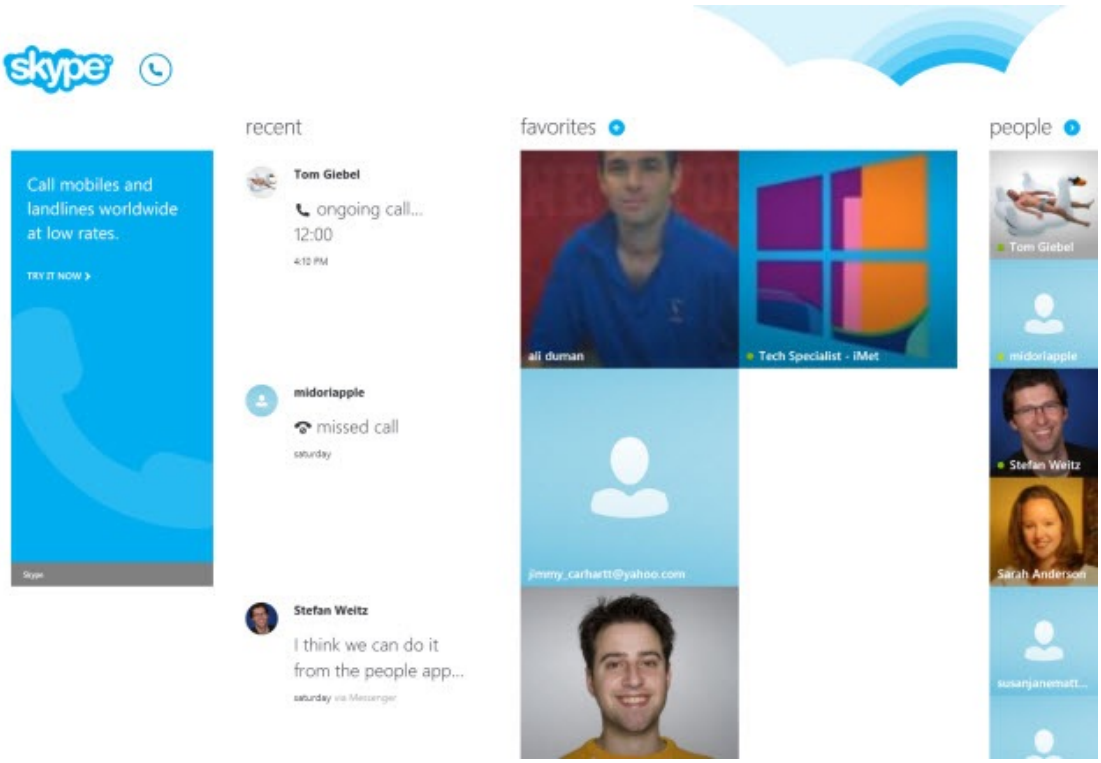
(Image credit: io9, Gawker Media. Sample command line interface with the use of colours and ASCII symbols to make up images)

While it would be somewhat interesting to see how a car would look like in a command line interface, the reality is that a large proportion of users dare not touch the command line tools available on their computers. Also from a usability standpoint, for novice users, it would be better for them to operate our software using a mouse as opposed to a keyboard, as a mouse is easier to handle and the user would not need to remember the instructions he/she needs to type into the system to invoke a function in our program. Having said that, in some cases like setting a new speed of the car, or using the on screen arrow buttons, it would be great to give users the flexibility to use the number buttons on the keyboard or use the arrow buttons on the keyboard for the same functionality, so we would attempt to integrate some control that can also be done by the keyboard.

A web-based interface was something we did consider, however in order to realise the multithreaded nature of the simulation part of the system, it would be very difficult. In addition to that, web based interfaces generally are not as responsive as desktop GUIs, due to the lack of many web browsers being able to fully take advantage of the processing power of a computer. While having a web-

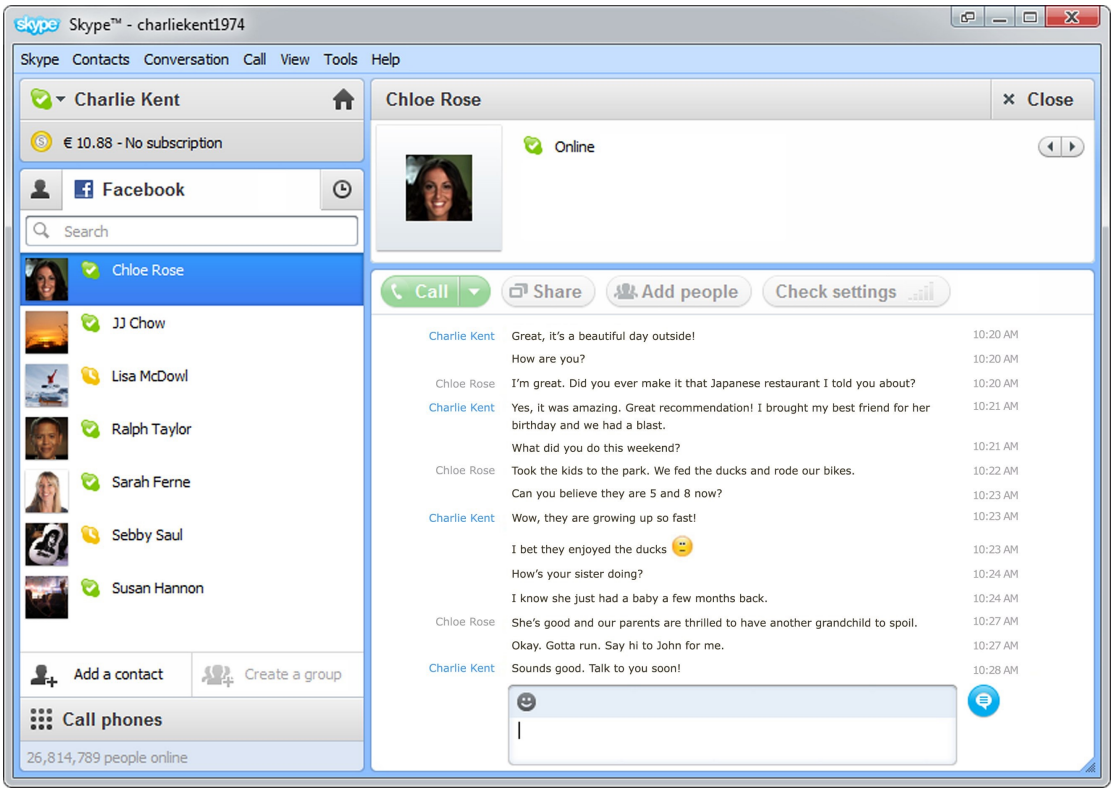
based interface would increase the appeal, it would potentially reduce the user satisfaction due to poor performance.

As a result of this, we decided that it would be best to work with delivering the app with a GUI. However it isn't that easy. There are two kinds of GUI systems available for our target OS, Windows 8, namely Modern UI and the Classic UI. The Modern UI is the new interpretation of the GUI on Windows 8 and above machines. It takes advantage of large fonts, crisp and slim text, in addition to an image centric approach. The image below shows Skype, a Modern UI application running on a Windows 8 machine:



(Image credit: PCMag.com)

However one of the main limitations we noted when we tried out some of the Modern UI based apps was how it wasn't very interactive to use especially when there isn't a touch screen built into the computer. Because the design depends on gestures for it to work best, it felt very clunky when used with a mouse. Since not all computers with Windows 8 shipped with a touchscreen, in fact many didn't, it meant we had to look at the Classic UI instead to ensure the experience would not be compromised.



(Skype on the Classic UI for Windows 7. Image Credit: PC Advisor)

One of the great things about the Classic UI was how well it works with a mouse. Also, the added familiarity would mean that users who generally know how to operate a Windows computer with a mouse would be able to operate the application. We noticed how many users would rather use the Classic UI as opposed to the Modern UI, as noted from what we have seen when walking around computer labs and in libraries. Another advantage of building the application with the Classic UI was the vast amount of resources available for us to use, after all this was a UI that has been used publically as of 1995, making it 20 years old.

This Classic UI supports both input from a mouse and a keyboard, but in our case, we would be having our application to be primarily operated by a mouse only, mainly for simplicity, with keyboard support in some situations. We find that it is much easier for a user to operate a program with one kind of input device, as opposed to using multiple input devices at the same time. In addition, it is actually easier from a development point of view, as we are able to control the types of input we are accepting from a user. For example, in the case of adjusting the speed of the car, it would be better for us to have a plus and minus button as opposed to bringing up a dialog box that prompts the user to key in a new vehicle speed. By avoiding the user to have to key in a new value, we can prevent accidental input of an invalid value (like a negative value – which will invoke a warning message that would require the user to check the value entered) or an incorrect value (like keying in 100 as opposed to 10). However, as we understand that some users would prefer to have keyboard input available, we would still try to implement some keyboard support in.

An interesting question we did get during our planning presentation in regards to the UI being a Classic UI but setting the minimum requirements to be Windows 8 as opposed to Windows 7 (which doesn't have the Modern UI), we decided to use the Classic UI while maintaining a more modern OS mainly because in general we find that Windows 8 is more polished and it is actively being supported by Microsoft, unlike Windows 7 which has ended it's mainstream support as of January 13, 2015 (Microsoft Support Life Cycle 2015). This means that Windows 7 is only going to be receiving security patches, and no more performance improvements, unlike Windows 8, which will still receive general improvements as part of the mainstream support up to January 9, 2018 (Microsoft Support Life Cycle 2015). In regards to this, it would make sense to work on a platform that has some form of active support, as opposed to a platform that is being deprecated soon.

Constantine and Lockwood's book on user interface design also gave us some guidelines in the UI process:

1. The structure principle

Common things should be grouped together. In our application, we would be grouping the features into 3 distinct parts, namely the infographic section, the simulator and the quiz section.

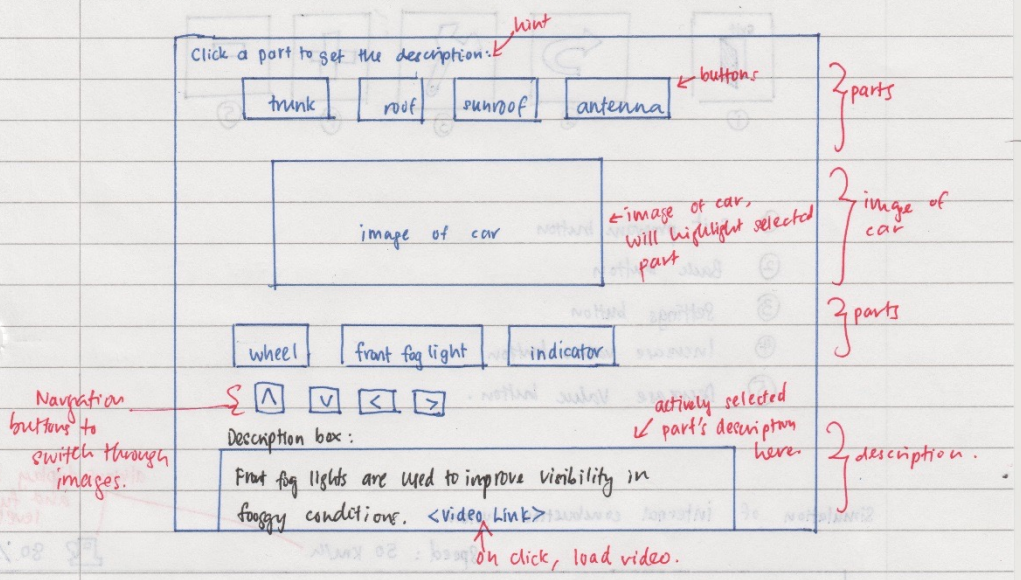
In the case of the infographic section, there would be a clear distinction between where the names of the parts of the car are listed and the description of the parts of the car.

As for the simulator, common items like the car status would be kept in a specific section of the screen, while the actual simulation itself would be kept in a different part of the screen. The same applies for the quiz section where the question has to be separated clearly from the choice of answers.

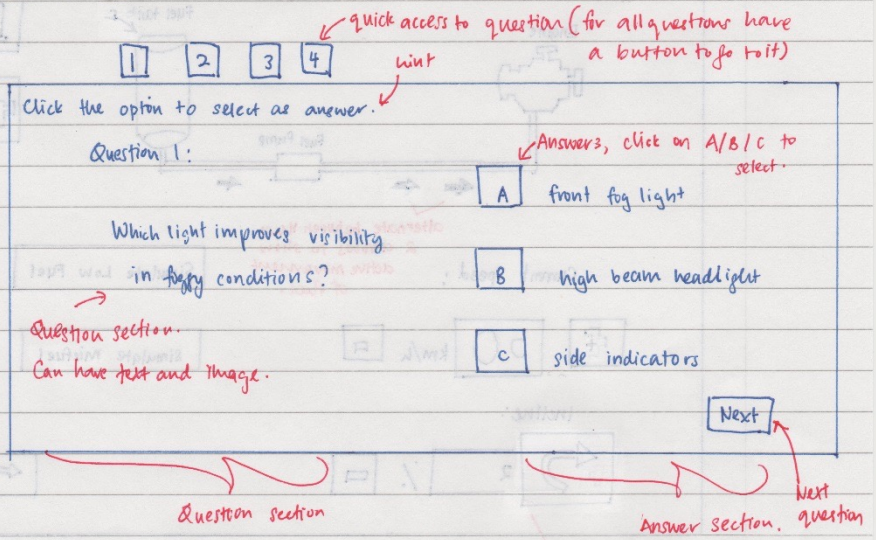
The following paper prototype shows how we would be able to achieve this segmentation and separation.

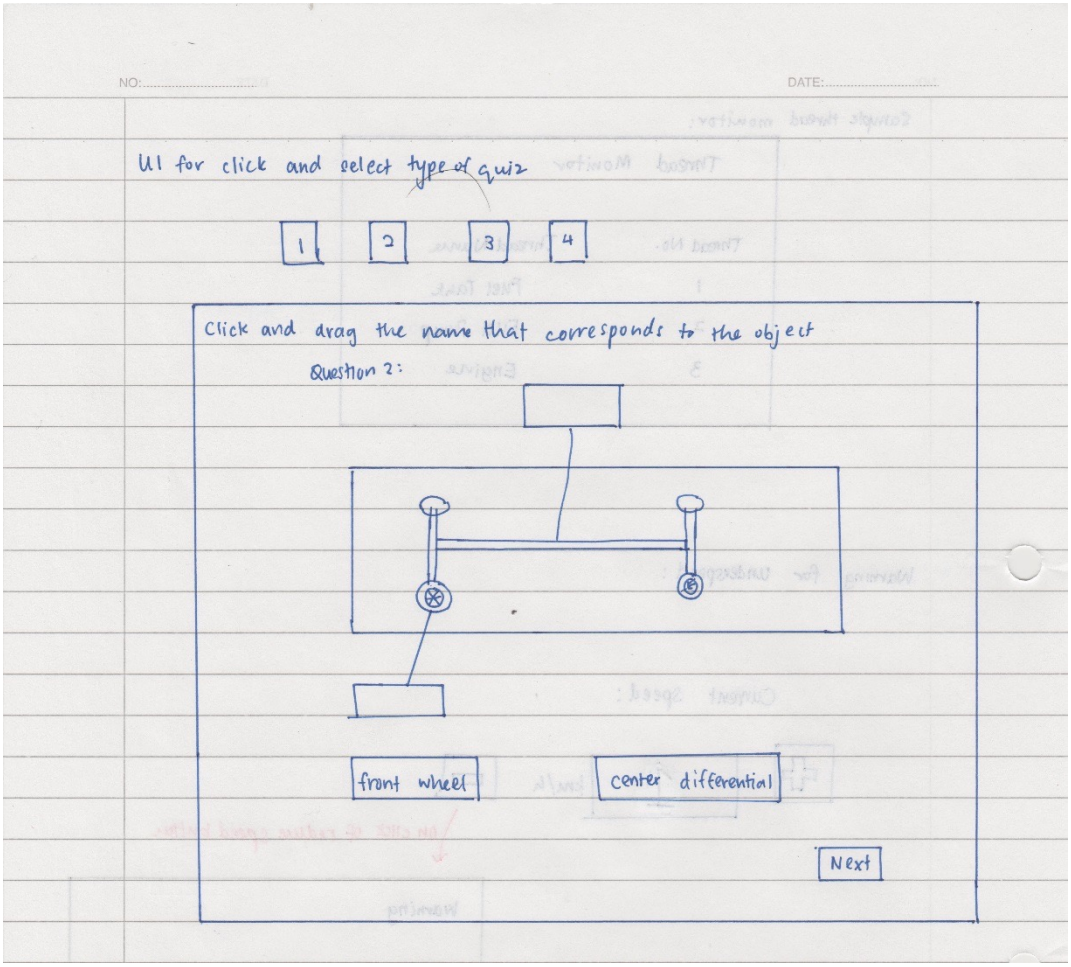
NO: DATE:

Ui for infographic section.



Ui for Quiz section





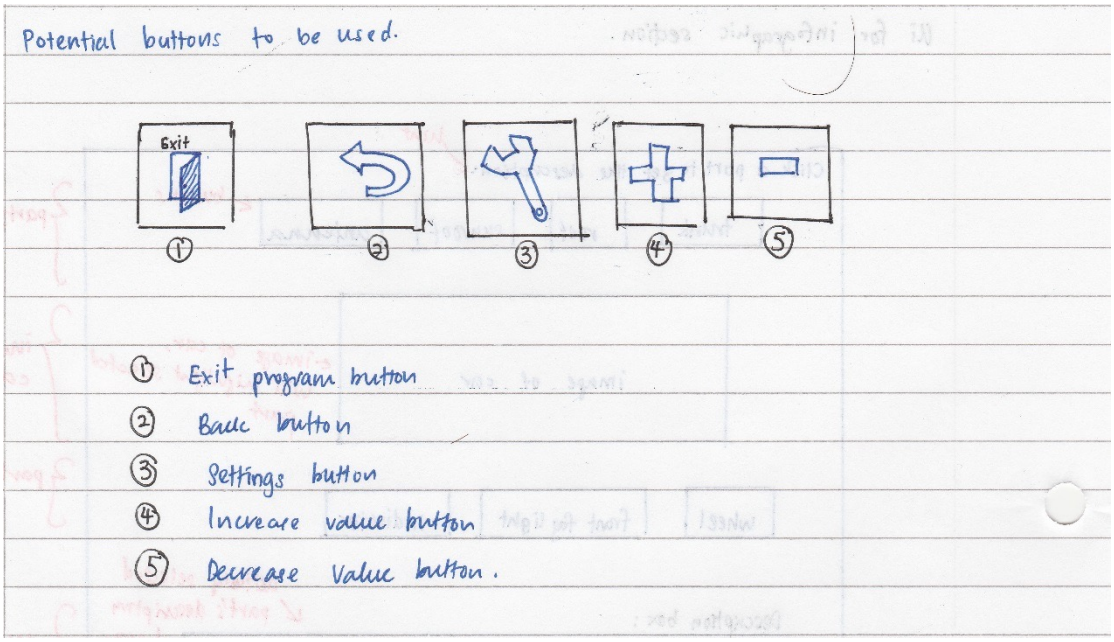
2. Simplicity

The design approach should ensure that simplicity is kept as the main point of our user interface. For a start, buttons should be clearly labelled, and should have meaningful icons (if icons are to be used), for example the exit button should have an icon of an exit door.

There should be adequate guides and hints throughout the application that guides the users to be able to easily navigate the application, in the unlikely event should they get confused. Hints should be clear and be in natural English.

In order to ensure we are on the right track, it would be best if we are able to get some feedback from the target market, like some form of acceptance testing in order to make the system easy to use.

The following shows some prototype buttons that have been designed:

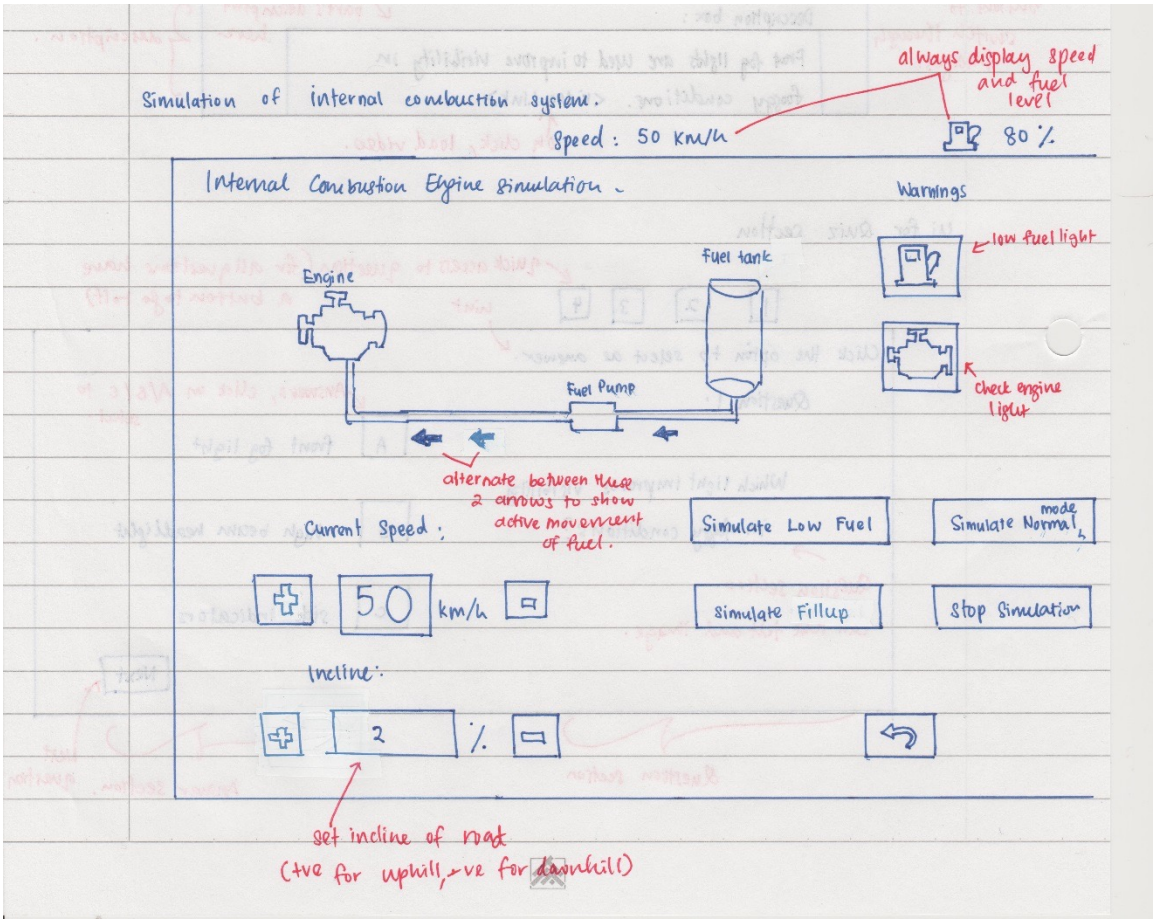


3. Visibility

While a function is being executed, it should be that only buttons and other visual elements that are necessary are displayed. All other visual elements that are not required should be hidden away as this could potentially confuse the user with redundant information.

For example in the case of the simulation of an internal combustion system, it would be redundant for us to have a button that turns on the lights in the car being displayed in the window. There is no relation of the headlights to the internal combustion system. The presence of additional unnecessary visual elements like unused buttons only add to the clutter in the program, affecting the user's visibility of the actual options they can click on while running a certain function.

The following shows how we only have relevant buttons in the internal combustion engine simulation:



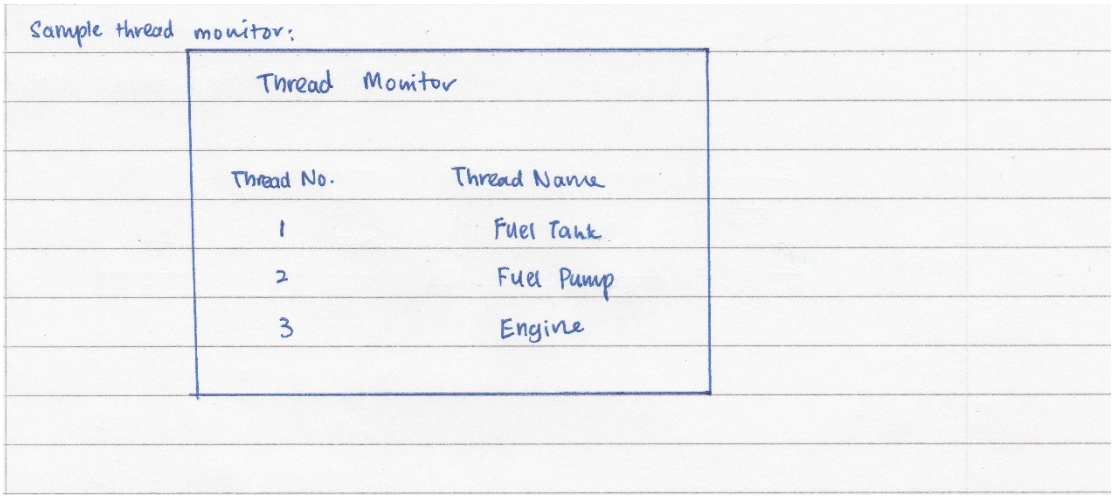
4. Feedback

Throughout the program, there should be some form of feedback, be it visual, acoustic or textual, to keep the user informed of the current operation of the application. Feedback would be important, as it is essential for the user to be able to know what is going on in the system, like in the simulation.

This should be achieved with clear and concise textual feedback, or in the case of some features being turned on; their icons should light up (like in the simulation of the electrical system in the car, should the hazard lights be turned on, the indicators should light up). Or in the case of the simulation of the flow of fuel or the braking power, arrows should be actively displayed showing the direction of flow of fuel or brake fluid in the car, such that the user knows at any one time how are the fluids moving about in this subsystem.

In addition, this principle should also be achieved using a thread monitor that is available throughout the runtime of the program, such that the users can track at any one time what threads are currently being run by the program.

The following shows the potential thread monitor we would have. Refer to the prototype image of the internal combustion engine simulator to see how the movement of fuel can be shown using the arrows.

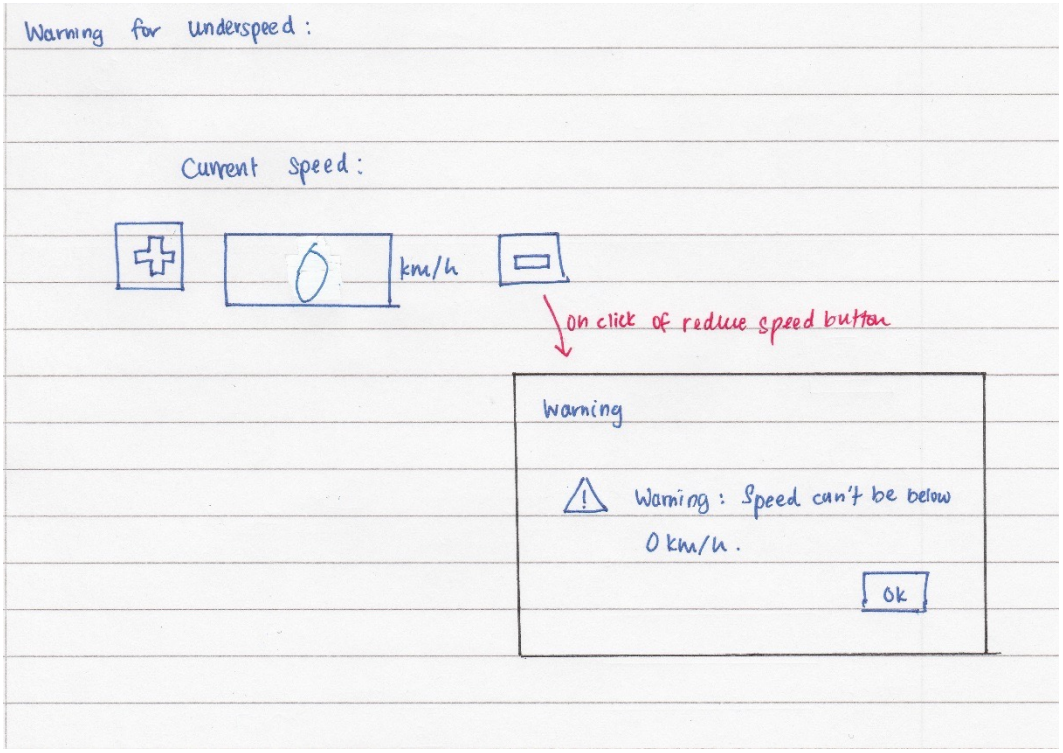


5. Tolerance

Error prevention would be the best approach in our system in order to achieve the tolerance principle. One of the issues for this is that by having a primarily mouse only interface, we actually have a limited set of options the user could possibly take in any situation.

However there has to be some error checking that should be done. For example in the case of simulating the fuel consumption of a car, it should be such that the speed of the vehicle cannot be reduced or increased after a certain speed threshold to prevent invalid or unfeasible values being used in the system. It would be illogical for a car to have a negative speed or a speed of over 200 km/h.

The following shows an underspeed warning, that is caused when attempting to reduce the speed to below 0 km/h.



6. Reuse

As much as possible, reusing design would be able to increase the usability in the system. This is to ensure the user is able to use their familiarity with a section of the to relate to the other parts of the system. However in our system, it would be only best for us to reuse the design of buttons in the system (i.e. ensure consistency in the buttons used, like a button with a picture of an exit door should only act as the exit button throughout all the sections of the program), but not the structure of the UI in each of the sections, as each of the sections require the user to respond differently. For example in the case of the quiz section, it would be messy to have the potential answers to be placed both above and below the question, unlike the infographic section where it makes more sense to have the names of the car parts to be both above and below the image of the car as that helps in differentiating the parts of the car from the top half of the car from the parts from the bottom half of the car, to a certain extent.

Assumptions

When developing our software, we had to make a number of assumptions in order for us to be able to feasibly develop the software. Assumptions are an inherent part of software development, as throughout the entire development process, be it developing an algorithm or a user interface, we have to have some assumptions in place like the environment it will run in, or the expected user capability. The difference between an assumption from a specification is that assumptions aren't explicitly provided to the developer, hence the developer has to make these assumptions on his own (Lewis et. Al 2004).

For our application, we too have made a number of assumptions during the design process of the application. These assumptions are in relation to the user's skills.

In terms of user skills, we would expect that in general the user should be able to install software using a wizard. It would be expected that the user is able to read instructions in English, and follow the instructions in the installation wizard to setup the software on his/her machine.

In addition, to use the software, we are under the assumption that the user is at least able to control a mouse properly and knows how to point, click and move the mouse around, and if possible knows how to operate a keyboard. The knowledge on working a keyboard isn't a must however, as our application can still be fully control with just a mouse. The user is also assumed to have reasonably good vision and is able to read and understand English at least, as we don't have any plans at the moment to setup any form of accessibility assistance like black and white display or a magnified interface.

Resources

It is technically unfeasible to make an application to be compatible with all kinds of computers, as there is such a large variety of computers available out there, from Windows PCs to Linux workstations and Macintosh computers, in addition to various form factor like desktops, embedded systems, laptops and tablets. Many of them come with various specifications too, some being more powerful than the others.

However, to ensure wide appeal and availability of our software, we are assuming that the customer would be using a Windows machine, as Windows computers are one of the most widely used computers in the market. We are specifically going to be building our application to be fully compatible with Windows 8 and above on the basis that it is currently the oldest version of Windows with full support from Microsoft. Windows 7 and prior is no longer getting mainstream support, which means it is about to be phased out in a few years. As we would like to ensure our application is future proof and is able to take advantage of the new technologies released in Windows 8 and on, hence we chose to have Windows 8 as the minimum OS required for our application.

In addition, we are assuming that the user has access to a computer running an x86/x64 architecture, like a mainstream Intel or AMD chip. The main reason we aren't supporting ARM based Windows 8 devices like the Surface RT is due to the lack of the number of people using such devices. It would be expected that the computer being used should have at least a single core 1GHz processer, with at least 1GB of RAM and at least 2GB of free hard disk space (however it is likely the final amount of hard disk space used would be less). We don't require the computer to have a dedicated graphics card in our application, as we won't be tapping into the power of a graphic card.

Essentially our system requirements are pretty similar to the minimum requirements for Windows 8, and in general any modern computer able to run Windows 8 will be able to run our application.

As part of our stretch goals, if we were able to implement support for VR and interaction using a digital steering wheel, we would assume that the user has a digital steering wheel connected over USB and an Oculus Rift VR headset.