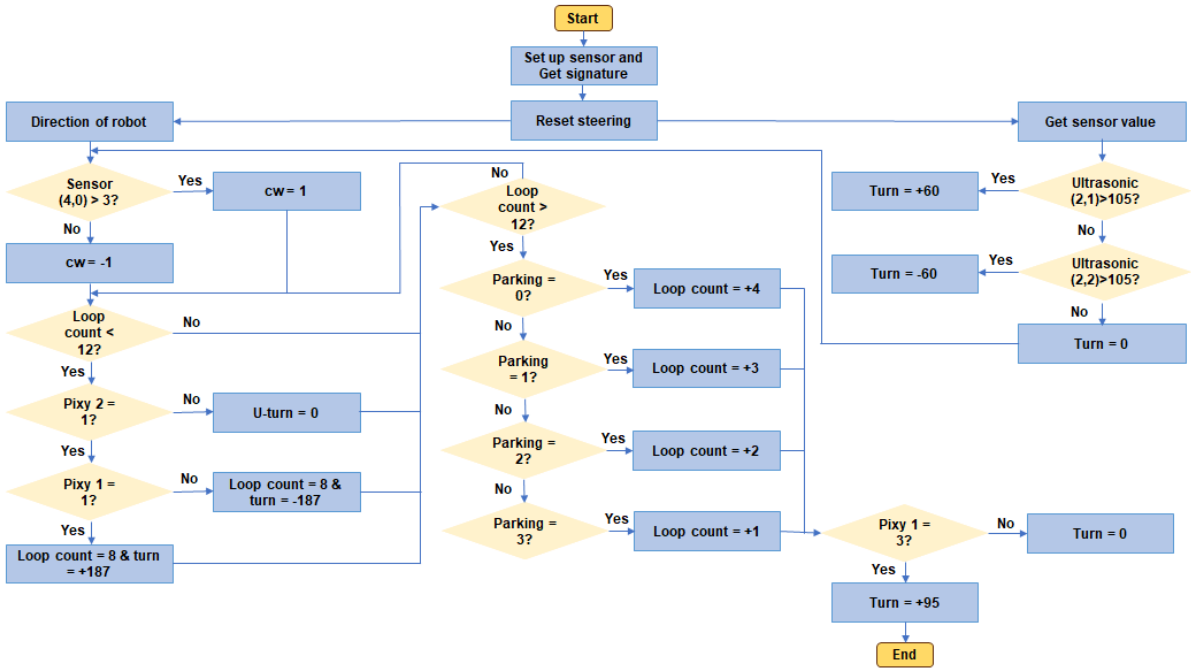
Obstacle challenge

The upper Pixy2 camera detects traffic signs. If it detects a red traffic sign, the self-driving car's steering will turn right; if it detects a green traffic sign, the self-driving car will respond similarly with the specified action for green. The bottom color sensor helps the self-driving car follow the colored lines on the map. If it detects an orange line first, the self-driving car will turn clockwise; if it detects a blue line first, it will turn counterclockwise.

The rear Pixy2 camera assists with U-turn decisions. When it detects a red signal, the self-driving car performs a U-turn; if it detects green, the self-driving car continues completing three normal laps. The gyro sensor maintains its primary role of ensuring the self-driving car travels in a straight line and executes precise turns.

The diagram below shows the flowchart for the Obstacle Challenge:



Below shows the explanation of the programming for the obstacle challenge.

```
folder "prjs""Clever" 'Download create a folder in ev3
Speaker.Tone(100,400,50)
Sensor.SetMode(4,2) 'set sensor to color mode
```

Folder creation, the code is creating or using a folder named "Clever" in the EV3's projects area. The color sensor connected to Port 4 is being set to detect colors.

```
'Display Function
Function Display (in number value1,in number value2)
  LCD.Clear()
  LCD.Text(1,50,0,2,value1)
  LCD.Text(1,50,20,2,value2)
EndFunction
```

“Function Display (in number value1, in number value2)” is a function called Display that takes two input numbers, value1 and value2. “LCD. Clear ()” is used to clear the EV3's LCD screen to make sure there is no previous content displayed. The brackets LCD. Text (1,50,0,2, value1) represents the colors, x, y, font and text respectively.

```
'Set Multiplexer Sensor Mode
Function setSensorMode(in number port, in number chanel, in number mode)
  address = 80 + 1 * (chanel - 1)
  Sensor.WriteI2CRegister(port,address,82,mode)
EndFunction
```

The setSensorMode function configures a multiplexer sensor by specifying the communication port, channel number, and desired mode. It calculates the I2C address based on the channel and sends a command to the sensor to set the specified mode using the Sensor.WriteI2CRegister function. This allows the robot to adapt the sensor's functionality according to the requirements of the task.

```
'Get Multiplexer Sensor Value
Function getValue(in number port, in number chanel, out number values)
  address = 80 + 1 * (chanel - 1)
  readData = Sensor.ReadI2CRegisters(port,address,84,2)
  values = readData[1] * 256 + readData[0]
EndFunction
```

The getValue function reads the value from a multiplexer sensor connected to a specified port and channel on the EV3. It calculates the I2C address based on the channel, retrieves two bytes of data from the sensor, and combines these bytes to create a single sensor value, which is stored in the output variable values.

```
'-----Pixy Function-------'
Function getSignature(in number port, in number sig, out number x, out number y)
  address = 80 + sig
  values = Sensor.ReadI2CRegisters(port, 1, address, 5)
  x = values[1]
  y = values[2]
EndFunction
```

The getSignature function retrieves the x and y coordinates of an object detected by the Pixy camera on a specified port using a given signature. It calculates the I2C address with 80 + sig, reads 5 bytes of data from the Pixy camera, and assigns the x and y coordinates from the data to the x and y output variables.

```
'-----Sensor Function-----'
@relativeHeading=0 'Define Relative Heading (Gyro Error)

setSensorMode(2,1,0)
setSensorMode(2,2,0)
setSensorMode(2,3,0)
```

The sensor function initializes the relative heading to zero, indicating no gyro error. It then sets the modes for three channels of the multiplexer sensor on port 2 to mode 0 using the setSensorMode function. This prepares the sensors for operation by configuring them to read data correctly for the robot's tasks.

```
Sub ReadSensor
  While "True"
    getValue(2,3,RawGyro)
    If RawGyro >= 32768 Then
      Gyro = RawGyro - 65536
    Else
      Gyro = RawGyro
    EndIf
    @relativeHeading = Gyro * -1 + target
    getValue(2,1,leftwall)
    getValue(2,2,rightwall)
    getSignature(1,1,greenx,greeny)
    getSignature(1,2,redx,redy)
    getSignature(1,3,pinkx,pinky)
    getSignature(3,3,backpinkx,backpinky)
  EndWhile
EndSub

Thread.Run=ReadSensor 'Multitask Read Sensor
```

The code defines a sensor function that continuously reads values from a gyro and wall sensors on the EV3. It sets the sensor modes for multiple channels and enters a loop in the ReadSensor subroutine. Inside this loop, it retrieves the raw gyro value and

adjusts it to account for gyro errors, calculating the relative heading. It also reads values for the left and right walls and retrieves color signature values for different colors. The ReadSensor subroutine runs as a separate thread, allowing multitasking while reading sensor data.

```
Sub ResetSteering
  Motor.StartPower("A",50)
  Program.Delay(500)
  While Motor.GetSpeed("A")<>0
    'wait
  EndWhile
  Motor.Move("A",-50,105,"True")
  Motor.ResetCount("A")
EndSub
```

The ResetSteering subroutine controls motor A to reset its position. It starts the motor at 50% power for 500 milliseconds, then waits until the motor stops. Afterwards, it moves the motor backwards at -50% speed for 105 degrees and finally resets the motor's position count. The Motor.Move function specifies the motor port, speed, rotation degrees, and brake mode.

```
'Define Variables
target = 0 'Direction robot heading
lastPillar=0 'Save the last pillar robot sees
followRed=200 'Distance to follow pillars
followGreen=220
avoidRed=followRed-10 'Distance to avoid pillars
avoidGreen=followGreen-5
```

The code defines variables for the self-driving car’s navigation: target sets the self-driving car’s heading direction to 0, lastPillar stores the last detected pillar, followRed and followGreen determine the distances to follow red and green pillars (200 and 220 units, respectively), and avoidRed and avoidGreen set the distances to avoid red and green pillars (10 and 5 units less than their following distances). These variables guide the car's movement in relation to the pillars it encounters.

```
'----- Loop Functions------'
speed = 70
loopCount = 0
round = 12
cw = 0
uturn = 0
uturnnow = 0
wallCheck = 1
parking = 0
correction = 1
ParkingLogic = 1

'--------Check U-Turn-----'
getSignature(3,1,lastGreenX,lastGreenY)
getSignature(3,2,lastRedX,lastRedY)
getSignature(1,1,greenx,greeny)
getSignature(1,2,redx,redy)

If lastRedY>lastGreenY Then
  uturn=1
  If redy>greeny And redx>50 And redx<200 Then
    uTurnPiller = 1 'red
  Else
    uTurnPiller = -1 'green
  EndIf
EndIf
'Display(lastRedY,lastGreenY)

While loopCount<round
  If loopCount = 0 Then 'When robot is doing 1st section
    ResetSteering()
    waitDegrees(80)

    While Sensor.ReadRawValue(4,0) = 6 'Loop until color sensor see orange or blue
      Steering()
    EndWhile
    waitDegrees(10)


    If Sensor.ReadRawValue(4,0) > 3 Then 'orange
      cw = 1
      Speaker.Tone(100,50,50)
    Else
      cw = -1
      Speaker.Tone(100,1200,50)
    EndIf
    ParkingLogic=0
    target = target - 89 * cw

  Else

    waitDegrees(500)
    ParkingLogic=1

    If cw=lastPillar Then
      waitDegrees(700)
      Speaker.Tone(100,220,50)
    Else
      waitDegrees(1200)
      Speaker.Tone(100,660,50)
    EndIf
```

```
    If cw = 1 Then
      While Sensor.ReadRawValue(4,0) <> 4 And Sensor.ReadRawValue(4,0) <> 5 'WaitOrange
        Steering()
      EndWhile
    Else
      While Sensor.ReadRawValue(4,0) <> 2 'Wait Blue
        Steering()
      EndWhile
    EndIf
    ParkingLogic=0

    Speaker.Tone(100,220,150)

    If loopCount = 8 And uturn = 1 Then 'Time to U-Turn
      Speaker.Tone(100,880,50)
      followRed = 5
      followGreen = 5

      cw = cw * -1
      If uTurnPiller = -1 Then
        target = target - 187
      Else
        target = target + 187
      EndIf
```

```
      uturnnow = 1
      waitDegrees(850)
      uturnnow=0
      followRed = 220
      followGreen = 220

    ElseIf loopCount < 8 Then
      target = target - 89 * cw
    Else
      target = target - 89 * cw
    EndIf
  EndIf

  If uturn=0  Then
    round = 12 + Math.Remainder(parking,4)
  Else
    round = 13 + (4 - Math.Remainder(parking,4))
  EndIf
  'Display(parking,0)
  loopCount = loopCount + 1
EndWhile
```

```
If cw = 1 And lastPillar=-1 Then
  parkingDegrees=1000
ElseIf cw = 1 And lastPillar = 1 Then
  parkingDegrees=850
ElseIf cw = -1 And lastPillar = -1 Then
  parkingDegrees=1200
ElseIf cw = -1 And lastPillar = 1 Then
  parkingDegrees=1900
EndIf

dDegrees = Motor.GetCount("D")
degrees=parkingDegrees
While Math.Abs(Motor.GetCount("D") - dDegrees) < degrees
  Steering()
  If cw=-1 And greenx<>0 Or cw=1 And redx<>0 Then
    degrees=parkingDegrees+300
  EndIf
EndWhile

uturnnow=1
speed=50
target=target + 95 * cw
waitDegrees(1300)
Speaker.Tone(100,220,200)
speed=0
```

In the Loop Functions, several variables are initialized to manage the car's operation, including speed, loop counts, and wall checks. The car assesses colors using the getSignature function to determine its last detected colors. If the last red object's y-coordinate is greater than that of the last green object, it prepares for a U-turn based on the detected colors. The main loop runs until loopCount reaches the predefined round. During the first section, it resets steering, waits for specific degrees, and reacts to color sensor readings, adjusting its direction accordingly. If a U-turn is needed after the eighth iteration, it modifies its target angle and waits before continuing. Finally, parking degrees are calculated based on the last detected pillar's color, and the car manoeuvres its parking position while adjusting for color detection.