```
var world[8]= [0,1,0,0,0,1,1,0]
var beliefs[8]
var i
var hit
var temp
var sum
var count
var red var green var blue
var s0 var s1 var s2 var s3 var s4
var prisoner[3]
var state
var found
var block

callsub stop

onevent button.backward
	callsub stop

onevent button.right
	callsub memorisePrisoner

onevent button.center
	when button.center == 0 do
		call math.fill(beliefs, 1000/8)
		callsub display_beliefs
	end

onevent button.forward
	callsub forward

onevent prox
	if state != STOPPED then
		if state == FORWARD or state == AVOID_FORWARD then
			if (prox.ground.delta[0] > 500 and prox.ground.delta[0] < 800)
or (prox.ground.delta[1] > 500 and prox.ground.delta[1] < 800) then
				hit = 0

			elseif (prox.ground.delta[0] > 200 and prox.ground.delta[0] <
500) or (prox.ground.delta[1] > 200 and prox.ground.delta[1] < 500) then
				hit = 1

			end
```

```
                end

                if state == FORWARD then
                        if (prox.ground.delta[0]<200) then
                                callsub slowDown

                        end
                        #*
                        when (prox.ground.delta[0] < 850 or prox.ground.delta[1] < 850
) do
                                callsub localize
                        end
                        *#

                elseif state == LEFT_ALLIGN then
                        if (prox.ground.delta[0]<200) then
                                while prox.ground.delta[0]<200 do
                                        motor.left.target = -50
                                        motor.right.target = 50
                                end
                                callsub approach

                        end


                elseif state == AVOID_TURN then
                        if (prox.ground.delta[1] < 200) then
                                while prox.ground.delta[1]<200 do
                                        motor.left.target = 50
                                        motor.right.target = -50
                                end
                                callsub avoidForward
                        end

                elseif state == CHECK_BLOCK then
                        if prox.horizontal[2] > 0 then
                                block = 1
                        else
                                block = 0
                        end
                        callsub rescueReturn

                elseif state == RESCUE_RETURN then
```

```
                        if (prox.ground.delta[0]<200) then
                              while prox.ground.delta[0]<200 do
                                    motor.left.target = -50
                                    motor.right.target = 50
                              end
                              callsub rescueFinish
                        end
                  elseif state == APPROACH then
                        if prox.ground.delta[0] > 750 or prox.ground.delta[1] > 750 th
en
                              callsub searchPrisoner
                              callsub recallPrisoner
                              if found == 1 then
                                    callsub rescueTurn
                              else
                                    callsub backoff
                              end
                        end
                  end
             end



onevent timer0
      if state == SLOW_DOWN then
            callsub leftAllign

      elseif state == BACKOFF then
            callsub rightTurn

      elseif state == RIGHT_TURN then
            if prox.horizontal[2] > 0 then
                  callsub avoidTurn
            else
                  callsub forward
            end

            #*
      elseif state == APPROACH then
            callsub searchPrisoner
            callsub recallPrisoner
            if found == 1 then
                  callsub rescueTurn
            else
                  callsub backoff
            end
            *#
      elseif state == AVOID_FORWARD
            then   callsub forward

      elseif state == RESCUE_TURN then
                  callsub checkBlock


      elseif state == RESCUE_FINISH then
            callsub stop
      end

sub localize
      callsub rotate
      callsub sense
      callsub display_beliefs

sub display_beliefs
      call leds.circle( beliefs[0]/LED, beliefs[1]/LED, beliefs[2]/LED, beliefs[3
]/LED, beliefs[4]/LED, beliefs[5]/LED, beliefs[6]/LED, beliefs[7]/LED)

sub rotate
      temp = beliefs[7]
      for i in 7:1 step -1 do beliefs[i] = beliefs[i-1] end
      beliefs[0] = temp

sub sense
      for i in 0:7 do
            if (hit==1 and world[i]==1) or (hit==0 and          world[i]==0) the
n
                  call math.muldiv(beliefs[i], beliefs[i], HIT, 100)
            else
                  call math.muldiv(beliefs[i], beliefs[i], MISS, 100)
            end
      end
      callsub normalize

sub normalize
      sum = 0
      for i in 0:7 do sum += beliefs[i] end
```

```
	for i in 0:7 do call math.muldiv(beliefs[i], beliefs[i],1000,sum) end


sub stop
	state = STOPPED
	timer.period[0] = 0
	timer.period[1] = 0
	call leds.circle(0,0,0,0,0,0,0,0)
	motor.left.target = 0
	motor.right.target = 0
	red = 0
	green = 0
	blue = 0
	hit = 0
	call leds.top(0,32,0)
	prisoner=[0,0,0]
	found = 0


sub memorisePrisoner
	callsub searchPrisoner
	prisoner = [red,green,blue]


sub searchPrisoner
	state = SEARCH

	red = 0
	green = 0
	blue = 0

	motor.left.target = 0
	motor.right.target = 0

	s0 = prox.horizontal[0]
	s1 = prox.horizontal[1]
	s2 = prox.horizontal[2]
	s3 = prox.horizontal[3]
	s4 = prox.horizontal[4]

	# red = WBW
	if s1 > BLACK_TH and s2 < BLACK_TH and s3 > BLACK_TH then
		red = 32
```

```
            green = 0
            blue = 0


        # green = BBW
        elseif s1 < BLACK_TH and s2 < BLACK_TH and s3 > BLACK_TH then
            red = 0
            green = 32
            blue = 0


        # blue = BWB
        elseif s1 < BLACK_TH and s2 > BLACK_TH and s3 < BLACK_TH then
            red = 0
            green = 0
            blue = 32


        # white = WWB
        elseif s1 > BLACK_TH and s2 > BLACK_TH and s3 < BLACK_TH then
            red = 32
            green = 32
            blue = 32
        end

        callsub displayLED

sub displayLED
        call leds.top(red, green, blue)

sub recallPrisoner
        if prisoner[0] == red and prisoner[1] == green and prisoner[2] == blue then
            call leds.circle(10, 11, 12, 13, 14, 15, 16, 17)
            found = 1
            callsub rescueTurn

        end

sub square
        callsub forward

sub forward
        state = FORWARD
        call leds.top(0,0,0)
        motor.left.target = 300
        motor.right.target = 300


sub slowDown
        state = SLOW_DOWN
        motor.left.target = 50
        motor.right.target = 50
        timer.period[0]=3700


sub leftAllign
        state = LEFT_ALLIGN
        motor.left.target = -50
        motor.right.target = 50

sub approach
        state = APPROACH
        motor.left.target = 50
        motor.right.target = 50
        #timer.period[0] = 2000


sub backoff
        state = BACKOFF
        motor.left.target=-50
        motor.right.target=-50
        timer.period[0] = 2000


sub checkBlock
        state = CHECK_BLOCK

sub rescueTurn
        state = RESCUE_TURN
        motor.left.target = 50
        motor.right.target = -50
        timer.period[0]= 3500

sub rescueReturn
        state = RESCUE_RETURN
        motor.left.target = -50
        motor.right.target = 50
```

```
sub rescueFinish
	state = RESCUE_FINISH
	if block == 1 then
		motor.left.target = -220
		motor.right.target = -300
		timer.period[0] = 5000
	elseif block == 0 then
		motor.left.target = -300
		motor.right.target = -300
		timer.period[0] = 3000
	end

sub rightTurn
	state = RIGHT_TURN
	call leds.top(0,0,0)
	motor.left.target = 50
	motor.right.target = -50
	timer.period[0]= 3500

sub avoidTurn
	state = AVOID_TURN
	motor.left.target = 50
	motor.right.target = -50

sub avoidForward
	state = AVOID_FORWARD
	motor.left.target = 300
	motor.right.target = 300
	timer.period[0] = 2000

sub stopRotate
```