SCRIPT DE BASE POUR SIGNAUX LUMINEUX FRANÇAIS

```
-- Fréquence de clignotement 0.5 Sec
LIGHT_FLASH_OFF_SECS          = 0.5
LIGHT_FLASH_ON_SECS           = 0.5

gTimeSinceLastFlash           = 0
gLightFlashOn                 = false
gFirstFlashLight              = true

gFlashingLightActive          = false
gFlashingLight                = {}
--------------------------------------------------------------------------------
-- INITIALISE

function Initialise ( )

        DefaultInitialise()

        gInitialised = false

        gSignalState = SIGNAL_VOIELIBRE

        Call( "BeginUpdate" )
end
--------------------------------------------------------------------------------
-- DEFAULT INITIALISE

function DefaultInitialise()

        gLinkCount = Call( "GetLinkCount" )

        gOccupationTable = {}
        for i=0, gLinkCount - 1 do
                gOccupationTable[i] = 0
        end

        gLinkState = {}
        for link = 0, gLinkCount - 1 do
                gLinkState[link] = SIGNAL_VOIELIBRE
        end
end

--------------------------------------------------------------------------------
-- RESET SIGNAL STATE

function ResetSignalState ( )

        Initialise()

end
--------------------------------------------------------------------------------
```

Inutile dans le cas des panneaux sans lumière clignotante

Inutile dans le cas des panneaux F

Inutile dans le cas des panneaux F

-- UPDATE

```
function Update (time)
        if gFlashingLightActive then
          if gFirstLightFlash then
            gTimeSinceLastFlash = 0
            gFirstLightFlash = false
            gLightFlashOn = false
      else
        gTimeSinceLastFlash = gTimeSinceLastFlash + time

        if (not gLightFlashOn) and gTimeSinceLastFlash >= LIGHT_FLASH_OFF_SECS then
          for i, v in ipairs(gFlashingLight) do
            Call ("ActivateNode", v, 1)
          end
          gLightFlashOn = true
          gTimeSinceLastFlash = 0
        end

        if gLightFlashOn and gTimeSinceLastFlash >= LIGHT_FLASH_ON_SECS then
          for i, v in ipairs(gFlashingLight) do
            Call ("ActivateNode", v, 0)
          end
          gLightFlashOn = false
          gTimeSinceLastFlash = 0
        end

      end
    end

    DefaultUpdate()
end
```

Inutile dans le cas des panneaux sans lumière clignotante

-- DEFAULT UPDATE

```
function DefaultUpdate( time )

        gInitialised = true

        if gLinkCount > 1 then
                OnJunctionStateChange( 0, "", 1, 0 )
        end

        if gSignalState == SIGNAL_VOIELIBRE then
                SetState(SIGNAL_VOIELIBRE)
        end
end
```

Inutile dans le cas des panneaux F mais impératif pour les signaux protégeant une jonction (sinon pas d'initialisation correcte)

-- GLOBALS

```
CLEAR          = 0
WARNING        = 1
BLOCKED        = 2

RESET_SIGNAL_STATE                  = 0
INITIALISE_SIGNAL_TO_BLOCKED        = 1
JUNCTION_STATE_CHANGE               = 2
INITIALISE_TO_PREPARED              = 3

OCCUPATION_INCREMENT                = 10
OCCUPATION_DECREMENT                = 11

SIGNAL_CARRE                        = 12
SIGNAL_VOIELIBRE                    = 13
SIGNAL_AVERTISSEMENT                = 14
SIGNAL_JAUNEcli                     = 15
SIGNAL_ROUGEcli                     = 16
SIGNAL_SEMAPHORE                    = 17
SIGNAL_VERTcli                      = 18
SIGNAL_R30                          = 19
SIGNAL_R60                          = 20
SIGNAL_R60JAUNEcli                  = 21
SIGNAL_RR30                         = 22
SIGNAL_RR60                         = 23
SIGNAL_RR30AVERTISSEMENT            = 24
SIGNAL_RR30JAUNEcli                 = 25
SIGNAL_RR60AVERTISSEMENT            = 26
SIGNAL_RR60JAUNEcli                 = 27
SIGNAL_AVERTISSEMENTBJ              = 28
SIGNAL_ARR30BJ                      = 29
SIGNAL_DISQUE                       = 30
SIGNAL_MANOEUVRE                    = 31
SIGNAL_BLANCcli                     = 32
SIGNAL_ETEINT                       = 33
SIGNAL_ID0                          = 34
SIGNAL_ID1                          = 35
SIGNAL_ID2                          = 36
SIGNAL_ID3                          = 37
SIGNAL_ID4                          = 38
SIGNAL_GASAM                        = 39

PASS_OFFSET                         = 40

PASS_RESET_SIGNAL_STATE             = PASS_OFFSET + RESET_SIGNAL_STATE  -- Never used!
PASS_JUNCTION_STATE_CHANGE          = PASS_OFFSET + JUNCTION_STATE_CHANGE
PASS_INITIALISE_SIGNAL_TO_BLOCKED   = PASS_OFFSET + INITIALISE_SIGNAL_TO_BLOCKED
PASS_INITIALISE_TO_PREPARED         = PASS_OFFSET + INITIALISE_TO_PREPARED
```

```
PASS_OCCUPATION_INCREMENT          = PASS_OFFSET + OCCUPATION_INCREMENT
PASS_OCCUPATION_DECREMENT          = PASS_OFFSET + OCCUPATION_DECREMENT
PASS_SIGNAL_CARRE                  = PASS_OFFSET + SIGNAL_CARRE
PASS_SIGNAL_VOIELIBRE              = PASS_OFFSET + SIGNAL_VOIELIBRE
PASS_SIGNAL_AVERTISSEMENT          = PASS_OFFSET + SIGNAL_AVERTISSEMENT
PASS_SIGNAL_JAUNEcli               = PASS_OFFSET + SIGNAL_JAUNEcli
PASS_SIGNAL_ROUGECLI               = PASS_OFFSET + SIGNAL_ROUGECLI
PASS_SIGNAL_SEMAPHORE              = PASS_OFFSET + SIGNAL_SEMAPHORE
PASS_SIGNAL_VERTcli                = PASS_OFFSET + SIGNAL_VERTcli
PASS_SIGNAL_R30                    = PASS_OFFSET + SIGNAL_R30
PASS_SIGNAL_R60                    = PASS_OFFSET + SIGNAL_R60
PASS_SIGNAL_R60JAUNEcli            = PASS_OFFSET + SIGNAL_R60JAUNEcli
PASS_SIGNAL_RR30                   = PASS_OFFSET + SIGNAL_RR30
PASS_SIGNAL_RR60                   = PASS_OFFSET + SIGNAL_RR60
PASS_SIGNAL_RR30AVERTISSEMENT      = PASS_OFFSET + SIGNAL_RR30AVERTISSEMENT
PASS_SIGNAL_RR30JAUNEcli           = PASS_OFFSET + SIGNAL_RR30JAUNEcli
PASS_SIGNAL_RR60AVERTISSEMENT      = PASS_OFFSET + SIGNAL_RR60AVERTISSEMENT
PASS_SIGNAL_RR60JAUNEcli           = PASS_OFFSET + SIGNAL_RR60JAUNEcli
PASS_SIGNAL_AVERTISSEMENTBJ        = PASS_OFFSET + SIGNAL_AVERTISSEMENTBJ
PASS_SIGNAL_ARR30BJ                = PASS_OFFSET + SIGNAL_ARR30BJ
PASS_SIGNAL_DISQUE                 = PASS_OFFSET + SIGNAL_DISQUE
PASS_SIGNAL_MANOEUVRE              = PASS_OFFSET + SIGNAL_MANOEUVRE
PASS_SIGNAL_BLANCcli               = PASS_OFFSET + SIGNAL_BLANCcli
PASS_SIGNAL_ETEINT                 = PASS_OFFSET + SIGNAL_ETEINT
PASS_SIGNAL_ID0                    = PASS_OFFSET + SIGNAL_ID0
PASS_SIGNAL_ID1                    = PASS_OFFSET + SIGNAL_ID1
PASS_SIGNAL_ID2                    = PASS_OFFSET + SIGNAL_ID2
PASS_SIGNAL_ID3                    = PASS_OFFSET + SIGNAL_ID3
PASS_SIGNAL_ID4                    = PASS_OFFSET + SIGNAL_ID4
PASS_SIGNAL_GASAM                  = PASS_OFFSET + SIGNAL_GASAM

AWS_MESSAGE                        = 11
TPWS_MESSAGE                       = 12
SPAD_MESSAGE                       = 13
CROCO_MESSAGE                      = 14

gConnectedLink = 0
gUpdating = 0


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

-- JUNCTION STATE CHANGE

```
function OnJunctionStateChange( junction_state, parameter, direction, linkIndex )

        if junction_state == 0 then
                if linkIndex == 0 then
                        if gLinkCount == 1 then

                        else

                                linkCountAsString = "" .. (5 * (gLinkCount + 1))

                                local newConnectedLink = Call( "GetConnectedLink", linkCountAsString,
                                 1, 0 )

                                if newConnectedLink == gConnectedLink then

                                else

                                        gConnectedLink = newConnectedLink

                                        if gConnectedLink > 0 then

                                                if (gOccupationTable[gConnectedLink] == 0) and
                                                (gOccupationTable[0] == 0) then

                                                        CheckSignalState( )

                                                else

                                                        Occupe( gConnectedLink )

                                                end

                                        elseif gConnectedLink == -1 then

                                                Carre( 0 )

                                        end
                                end
                        end
                end
        end
end
```

Inutile dans le cas des panneaux F

-- ON CONSIST PASS

```
function OnConsistPass ( prevFrontDist, prevBackDist, frontDist, backDist, linkIndex )

	local crossingStart = 0
	local crossingEnd = 0

	if ( frontDist > 0 and backDist < 0 ) or ( frontDist < 0 and backDist > 0 ) then

		if ( prevFrontDist < 0 and prevBackDist < 0 ) or ( prevFrontDist > 0 and prevBackDist > 0
		 ) then

			crossingStart = 1
		end

	else

		if ( prevFrontDist < 0 and prevBackDist > 0 ) or ( prevFrontDist > 0 and prevBackDist < 0
		 ) then

			crossingEnd = 1
		end
	end
```

```
if (crossingStart == 1) then
-------------------------------------------------------------------------------
        --Le train est en train de franchir un lien vers l'avant
        if (prevFrontDist > 0 and prevBackDist > 0) then

                if (linkIndex == 0) then

                        if (gSignalState == SIGNAL_BLOCKED) then

                                Call( "SendConsistMessage", SPAD_MESSAGE, "" )

                        end

                        Occupe( 0 )

                        gOccupationTable[0] = gOccupationTable[0] + 1

                elseif (linkIndex > 0) then

                        gOccupationTable[linkIndex] = gOccupationTable[linkIndex] + 1

                end

-------------------------------------------------------------------------------
        --Le train est en train de franchir un lien vers l'arrière
        elseif (prevFrontDist < 0 and prevBackDist < 0) then


                if (linkIndex == 0) then

                        if gOccupationTable[0] == 1 and gConnectedLink ~= -1 then

                                local signalStateMessage = gLinkState[gConnectedLink]

                                Call( "SendSignalMessage", signalStateMessage, "", -1,1, 0 )
                        end

                        Call( "SendSignalMessage", OCCUPATION_INCREMENT, "", -1, 1, 0 )

                elseif (linkIndex > 0) then

                        if (gConnectedLink == linkIndex) then

                                gOccupationTable[0] = gOccupationTable[0] + 1

                        else

                        end
                end
        end
```

Voir page 20 pour le détail

```
	elseif (crossingEnd == 1) then
	--------------------------------------------------------------------------------
		--Le train a fini de franchir un lien vers l'arrière
		if (frontDist > 0 and backDist > 0) then

			if (linkIndex == 0) then
				if gOccupationTable[0] > 0 then
					gOccupationTable[0] = gOccupationTable[0] - 1

				else

				end

				if (gOccupationTable[0] == 0 and gConnectedLink ~= -1 and
				 gOccupationTable[gConnectedLink] == 0) then
					CheckSignalState( )
				end

			elseif (linkIndex > 0) then
				if gOccupationTable[linkIndex] > 0 then

					gOccupationTable[linkIndex] = gOccupationTable[linkIndex] - 1

				else
				end
			end
	--------------------------------------------------------------------------------
		--Le train a fini de franchir un lien vers l'avant
		elseif (frontDist < 0 and backDist < 0) then

			if (linkIndex == 0) then

				Call( "SendSignalMessage", OCCUPATION_DECREMENT, "", -1, 1, 0 )

			elseif (linkIndex > 0) then

				if (gConnectedLink == linkIndex) then

					if gOccupationTable[0] > 0 then

						gOccupationTable[0] = gOccupationTable[0] - 1

					else

					end
				else
				end
			end
		end
	end
end
--------------------------------------------------------------------------------
```

Pour les panneaux G et H, voir la fonction adaptée page suivante

```
-- NONOCCUPE

function NonOccupe( linkIndex )

	if (linkIndex == 0 and gConnectedLink > 0) then
		linkIndex = gConnectedLink
	end

	if (gConnectedLink == linkIndex) then

		if (gOccupationTable[linkIndex] ~= nil) and
			(gOccupationTable[linkIndex] == 0) and
			(gOccupationTable[0] == 0) then

			if gSignalState ~= SIGNAL_VOIELIBRE then

				SetState( SIGNAL_VOIELIBRE )
				Call( "SendSignalMessage", SIGNAL_VOIELIBRE, "", -1, 1, 0 )
			end
		end
	end

	gLinkState[linkIndex] = SIGNAL_VOIELIBRE
end

-------------------------------------------------------------------------------
-- OCCUPE

function Occupe( linkIndex )

	if gSignalState ~= SIGNAL_SEMAPHORE then

		SetState( SIGNAL_SEMAPHORE )
		Call( "SendSignalMessage", SIGNAL_SEMAPHORE, "", -1, 1, 0 )
	end

end

-------------------------------------------------------------------------------
-- CARRE

function Carre( linkIndex )

	if gSignalState ~= SIGNAL_CARRE then

		SetState( SIGNAL_CARRE )
		Call( "SendSignalMessage", SIGNAL_CARRE, "", -1, 1, 0 )
	end

end
-------------------------------------------------------------------------------
```

Inutile dans le cas des panneaux F

-- NONOCCUPE

Pour les panneaux G et H uniquement

```
function NonOccupe( linkIndex )

        if (linkIndex == 0 and gConnectedLink > 0) then

                linkIndex = gConnectedLink
        end

        if (gConnectedLink == linkIndex) then

                if (gOccupationTable[linkIndex] ~= nil) and
                        (gOccupationTable[linkIndex] == 0) and
                        (gOccupationTable[0] == 0) then

                                if (linkIndex > 1 ) then

                                SetState( SIGNAL_RR30 )
                                Call( "SendSignalMessage", SIGNAL_RR30, "", -1, 1, 0 )
                                gLinkState[linkIndex] = SIGNAL_RR30

                                else

                                SetState( SIGNAL_VOIELIBRE )
                                Call( "SendSignalMessage", SIGNAL_VOIELIBRE, "", -1, 1, 0 )
                                gLinkState[linkIndex] = SIGNAL_VOIELIBRE

                                end

                end
        end
end
```

Mettre le nom de l'état adéquat

Etat vers la(les) voie(s) déviée(s)

Etat vers la voie directe

Cette fonction sera remplacée, dans le cas des panneaux G et H, par la fonction Annonce()

```
-- AVERTISSEMENT
function Avertissement( linkIndex )

	if (linkIndex == 0 and gConnectedLink > 0) then

		linkIndex = gConnectedLink
	end

	if (gConnectedLink == linkIndex) then

		if (gOccupationTable[linkIndex] ~= nil) and
			(gOccupationTable[linkIndex] == 0) and
			(gOccupationTable[0] == 0) then

			if gSignalState ~= SIGNAL_AVERTISSEMENT then

				SetState( SIGNAL_AVERTISSEMENT )
				Call( "SendSignalMessage", SIGNAL_AVERTISSEMENT, "", -1, 1, 0 )
			end
		end
	end

	gLinkState[linkIndex] = SIGNAL_AVERTISSEMENT
end
```

Cette fonction sera remplacée, dans le cas des panneaux G et H, par la fonction Preannonce()

```
---------------------------------------------------------------------------
-- JAUNEcli
function Jaunecli( linkIndex )

	if (linkIndex == 0 and gConnectedLink > 0) then

		linkIndex = gConnectedLink
	end

	if (gConnectedLink == linkIndex) then

		if (gOccupationTable[linkIndex] ~= nil) and
			(gOccupationTable[linkIndex] == 0) and
			(gOccupationTable[0] == 0) then

			if gSignalState ~= SIGNAL_JAUNEcli then

				SetState( SIGNAL_JAUNEcli )
				Call( "SendSignalMessage", SIGNAL_JAUNEcli, "", -1, 1, 0 )
			end
		end
	end

	gLinkState[linkIndex] = SIGNAL_JAUNEcli

end
```

Remplace la fonction Avertissement() pour les panneaux G et H uniquement

```
-----------------------------------------------------------------------
-- ANNONCE

function Annonce( linkIndex )

        if (linkIndex == 0 and gConnectedLink > 0) then

                linkIndex = gConnectedLink

        end

        if (gConnectedLink == linkIndex) then

                if (gOccupationTable[linkIndex] ~= nil) and
                        (gOccupationTable[linkIndex] == 0) and
                        (gOccupationTable[0] == 0) then

                        if (linkIndex > 1 ) then

                        SetState( SIGNAL_RR30A )
                        Call( "SendSignalMessage", SIGNAL_RR30A, "", -1, 1, 0 )
                        gLinkState[linkIndex] = SIGNAL_RR30A

                        else

                        SetState( SIGNAL_AVERTISSEMENT )
                        Call( "SendSignalMessage", SIGNAL_AVERTISSEMENT, "", -1, 1, 0 )
                        gLinkState[linkIndex] = SIGNAL_AVERTISSEMENT

                        end

                end
        end
end

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

Mettre le nom de l'état adéquat

Etat vers la(les) voie(s) déviée(s)

Etat vers la voie directe

Remplace la fonction Jaunecli() pour les panneaux G et H uniquement

```
----------------------------------------------------------------------------
-- PREANNONCE

function Preannonce( linkIndex )

        if (linkIndex == 0 and gConnectedLink > 0) then

                linkIndex = gConnectedLink
        end

        if (gConnectedLink == linkIndex) then

                if (gOccupationTable[linkIndex] ~= nil) and
                        (gOccupationTable[linkIndex] == 0) and
                        (gOccupationTable[0] == 0) then

                                if (linkIndex > 1 ) then

                                SetState( SIGNAL_RR30AA )
                                Call( "SendSignalMessage", SIGNAL_RR30AA, "", -1, 1, 0 )
                                gLinkState[linkIndex] = SIGNAL_RR30AA

                                else

                                SetState( SIGNAL_JAUNEcli )
                                Call( "SendSignalMessage", SIGNAL_JAUNEcli, "", -1, 1, 0 )
                                gLinkState[linkIndex] = SIGNAL_JAUNEcli

                                end
                end
        end
end
----------------------------------------------------------------------------
```

Mettre le nom de l'état adéquat

Etat vers la(les) voie(s) déviée(s)

Etat vers la voie directe

Rajouter ensuite, si nécessaire, toutes les fonctions d'états comme R30(), R60(), RR30(), RR60()……sur le modèle de celui de l'Avertissement()

```
--------------------------------------------------------------------------------
-- ON SIGNAL MESSAGE

function OnSignalMessage( message, parameter, direction, linkIndex )

	if (linkIndex > 0) then

		if message > PASS_OFFSET then
			Call( "SendSignalMessage", message, parameter, -direction, 1, linkIndex )

		elseif message ~= RESET_SIGNAL_STATE and message ~= JUNCTION_STATE_CHANGE
		 and parameter ~= "DoNotForward" then

			Call( "SendSignalMessage", message + PASS_OFFSET, parameter, -direction, 1,
			linkIndex )
		end
	end

	if (linkIndex >= 0) then

		if message >= PASS_INITIALISE_SIGNAL_TO_BLOCKED then

			if linkIndex > 0 then

				ReactToSignalMessage( message - PASS_OFFSET, parameter, direction,
				 linkIndex )
			end

		else
			ReactToSignalMessage( message, parameter, direction, linkIndex )
		end
	end
end

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


```
-- SET STATE

function SetState( newState )

        if (newState == SIGNAL_VOIELIBRE) then

                Call ("Set2DMapSignalState", CLEAR)
                SetLights(SIGNAL_VOIELIBRE)

         elseif (newState == SIGNAL_SEMAPHORE) then

                Call ("Set2DMapSignalState", BLOCKED)
                SetLights(SIGNAL_SEMAPHORE)

        elseif (newState == SIGNAL_CARRE) then

                Call ("Set2DMapSignalState", BLOCKED)
                SetLights(SIGNAL_CARRE)

         elseif (newState == SIGNAL_AVERTISSEMENT) then

                Call ("Set2DMapSignalState", WARNING)
                 SetLights(SIGNAL_AVERTISSEMENT)

        elseif (newState == SIGNAL_JAUNEcli) then

                Call ("Set2DMapSignalState", WARNING)
                SetLights(SIGNAL_JAUNEcli)

        end

gSignalState = newState

end

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


```
-- SET LIGHTS

function SetLights ( newState )

        if (newState == SIGNAL_VOIELIBRE) then
                Call ( "ActivateNode", "mod_5feux_voie_libre", 1 )
                Call ( "ActivateNode", "mod_5feux_avertissement", 0 )
                Call ( "ActivateNode", "mod_5feux_carre", 0 )
                Call ( "ActivateNode", "mod_5feux_oeilleton", 1 )
                gFlashingLightActive = false

        elseif (newState == SIGNAL_CARRE) then
                Call ( "ActivateNode", "mod_5feux_voie_libre", 1 )
                Call ( "ActivateNode", "mod_5feux_avertissement", 0 )
                Call ( "ActivateNode", "mod_5feux_carre", 0 )
                Call ( "ActivateNode", "mod_5feux_oeilleton", 0 )
                gFlashingLightActive = true
                gFlashingLight = { "mod_5feux_voie_libre" }
                gFirstLightFlash = true

        elseif (newState == SIGNAL_SEMAPHORE) then
                 Call ( "ActivateNode", "mod_5feux_voie_libre", 0 )
                Call ( "ActivateNode", "mod_5feux_avertissement", 0 )
                Call ( "ActivateNode", "mod_5feux_carre", 1 )
                Call ( "ActivateNode", "mod_5feux_oeilleton", 1 )
                 gFlashingLightActive = false

        elseif (newState == SIGNAL_AVERTISSEMENT) then
                 Call ( "ActivateNode", "mod_5feux_voie_libre", 0 )
                Call ( "ActivateNode", "mod_5feux_avertissement", 1 )
                Call ( "ActivateNode", "mod_5feux_carre", 0 )
                Call ( "ActivateNode", "mod_5feux_oeilleton", 1 )
                gFlashingLightActive = false

        elseif (newState == SIGNAL_JAUNEcli) then
                Call ( "ActivateNode", "mod_5feux_voie_libre", 0 )
                Call ( "ActivateNode", "mod_5feux_avertissement", 1 )
                Call ( "ActivateNode", "mod_5feux_carre", 0 )
                Call ( "ActivateNode", "mod_5feux_oeilleton", 1 )
                gFlashingLightActive = true
                gFlashingLight = { "mod_5feux_avertissement" }
                gFirstLightFlash = true
        else
                Print( ("ERROR: SetLights trying to switch to invalid state " .. newState ) )
        end
end
--------------------------------------------------------------------------------
```

Mettre les états du signal en conformité avec ceux pris en compte par le crocodile

```
-- GET SIGNAL STATE

function GetSignalState( )

        local signalState = -1

        if (gSignalState == SIGNAL_VOIELIBRE) then

                signalState = CLEAR

        elseif (gSignalState == SIGNAL_AVERTISSEMENT) or
        (gSignalState == SIGNAL_JAUNEcli) then

                signalState = WARNING

        elseif (gSignalState == SIGNAL_SEMAPHORE) then

                signalState = BLOCKED

        else
                Print( "ERROR: GetSignalState: failed - invalid signal state" )
        end

        return signalState

end
```

Inutile car le crocodile ne prend en compte que les aspects « ouverts » ou « fermés »

Mettre tous les états du signal possibles autres que les feux rouges

```
-- CHECK SIGNAL STATE

function CheckSignalState( )

        if gConnectedLink < 0 then

                Print("ERROR: CheckSignalState() received negative gConnectedLink - scripts should
                 check for this before calling the function!")

        elseif gLinkState[gConnectedLink] == SIGNAL_VOIELIBRE then

                NonOccupe( gConnectedLink )

        elseif gLinkState[gConnectedLink] == SIGNAL_AVERTISSEMENT then

                Avertissement( gConnectedLink )

        else

        end
end
```

```
-- REACT TO SIGNAL MESSAGE

function ReactToSignalMessage( message, parameter, direction, linkIndex )

    if ( message == SIGNAL_VOIELIBRE or message == SIGNAL_JAUNEcli ) then

        NonOccupe( linkIndex )

    elseif ( message == SIGNAL_AVERTISSEMENT ) then

        Jaunecli( linkIndex )

    elseif ( message == SIGNAL_SEMAPHORE ) then

        Avertissement( linkIndex )

    elseif (message == OCCUPATION_DECREMENT) then

        if gOccupationTable[linkIndex] > 0 then

            gOccupationTable[linkIndex] = gOccupationTable[linkIndex] - 1

        else
            Print( ("ERROR: OnSignalMessage: OCCUPATION_DECREMENT received...
            gOccupationTable[" .. linkIndex .. "] was already 0!") )
        end

        Avertissement( linkIndex )

    elseif (message == OCCUPATION_INCREMENT) then

        gOccupationTable[linkIndex] = gOccupationTable[linkIndex] + 1

        if (gConnectedLink == linkIndex) then

            Occupe( linkIndex )
        end

    elseif (message == INITIALISE_SIGNAL_TO_BLOCKED) then

        gOccupationTable[linkIndex] = gOccupationTable[linkIndex] + 1

        if (gLinkCount == 1) then
            Occupe( 0 )
        end

    elseif (message == JUNCTION_STATE_CHANGE) then

        if gInitialised and linkIndex == 0 and parameter == "0" and gLinkCount > 1 then

            OnJunctionStateChange( 0, "", 1, 0 )
```

Attention ! les message == xxx ne doivent apparaître qu'une seule fois dans la fonction

Inutile dans le cas des signaux lumineux français sauf pour les heurtoirs

```
                Call( "SendSignalMessage", message, parameter, -direction, 1, linkIndex )
            end

        elseif (message == RESET_SIGNAL_STATE) then

            ResetSignalState()
        end
end
```

Complément pour la fonction OnConsistPass

En cas de franchissement d'un carré fermé, on envoie un SPAD_MESSAGE qui déclenche le freinage d'urgence.

En cas de franchissement d'un sémaphore ou d'un feu rouge clignotant à une vitesse supérieure à 15 km/h, on envoie également un SPAD_MESSAGE. Les vitesses sont exprimées en mètres par seconde (m/s).

Pour les rappels 30 et 60, il y a aussi un SPAD si ces bifurcations sont prises à une vitesse supérieure.

```
if (crossingStart == 1)  then
        ----------------------------------------------------------------------------
                --Le train est en train de franchir un lien vers l'avant
                if (prevFrontDist > 0 and prevBackDist > 0) then

                        if (linkIndex == 0) then

                                if (gSignalState == SIGNAL_CARRE) then

                                        Call( "SendConsistMessage", SPAD_MESSAGE, "" )

                                elseif (gSignalState == SIGNAL_SEMAPHORE) or (gSignalState ==
                                SIGNAL_ROUGEcli) then

                                        local consistSpeed = Call( "GetConsistSpeed" )
                                        local speedLimit = 4.17

                                        if ( consistSpeed > speedLimit ) then

                                                Call( "SendConsistMessage", SPAD_MESSAGE, "" )
                                        end

                                elseif (gSignalState == SIGNAL_RR30) or (gSignalState ==
                                SIGNAL_RR30AVERTISSEMENT) or (gSignalState ==
                                SIGNAL_RR30JAUNEcli) then

                                        local consistSpeed = Call( "GetConsistSpeed" )
                                        local speedLimit = 8.33

                                        if ( consistSpeed > speedLimit ) then

                                                Call( "SendConsistMessage", SPAD_MESSAGE, "" )
                                        end

                                end

                                Occupe( 0 )

                                ……..
```

Partie à inclure pour un signal présentant le carré

Partie à inclure pour un signal présentant le sémaphore

Partie à inclure pour un signal présentant le RR30. Pour le RR60, mettre RR60 à la place de RR30

Pour le RR60, mettre 16.67 à la place de 8.33