EEP Lua Automatische Zugsteuerung

Rudy Boer, Frank Buchholz, April 2022
Version 2.0.1

Inhalt:

Zweck

Diese Lua-Software erzeugt automatischen Zugverkehr auf jeder EEP-Anlage, ohne dass Sie selbst viel Lua-Code schreiben müssen. Um eine Anlage zu steuern, müssen lediglich einige Daten über Züge und Weichen, die für eine bestimmte Strecke geschaltet werden müssen, in eine Reihe von Lua-Tabellen und Variablen eingegeben werden.

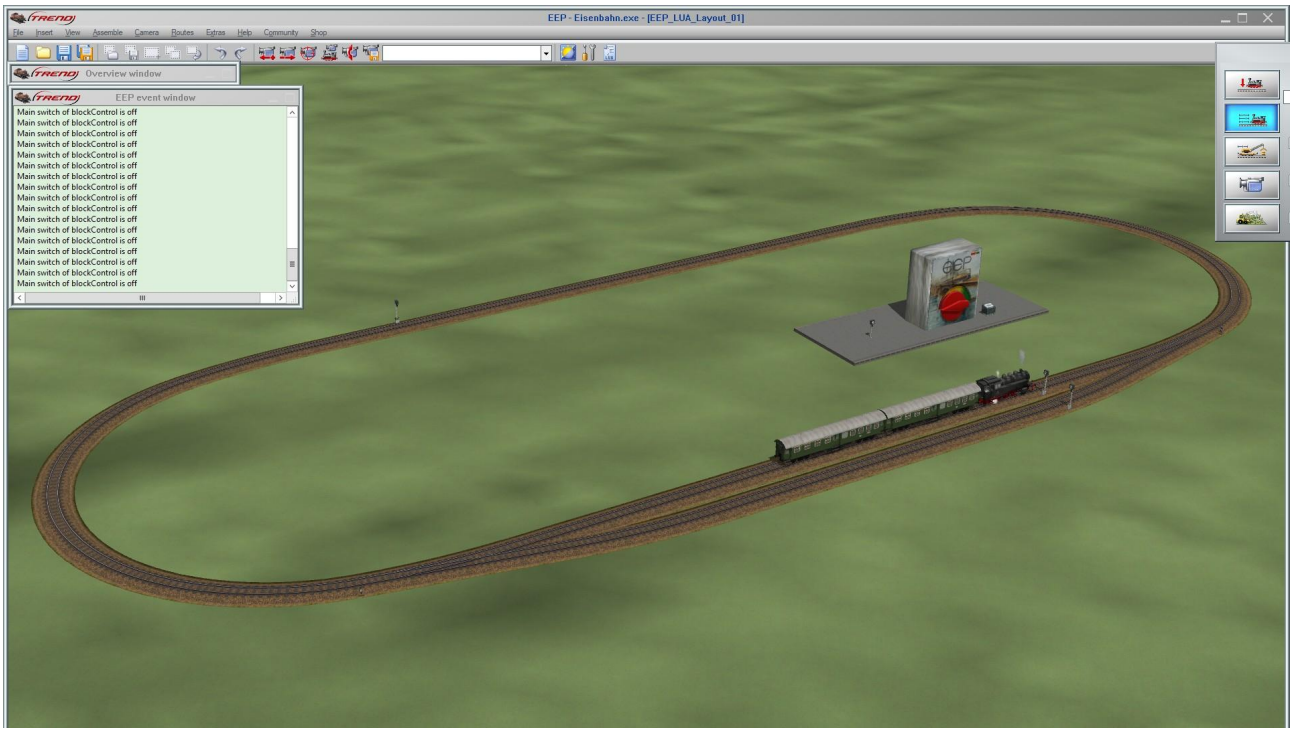
Inhalt der Zip-Datei

Benutzerhandbuch
Erhältlich in Englisch und Deutsch.

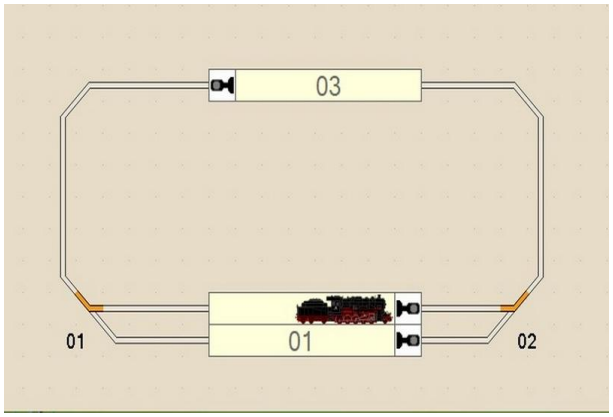
Die Datei `blockControl.lua`
Diese Datei muss im Ordner \LUA Ihrer EEP-Programminstallation abgelegt werden.

Demo-Dateien `EEP_Lua_Layout_01 ... _05`.
Die Demo-Layouts werden in diesem Handbuch als Beispiele für die Konfiguration der Lua-Datentabellen beschrieben, die ein Layout definieren.

Demo EEP_Lua_Layout_01



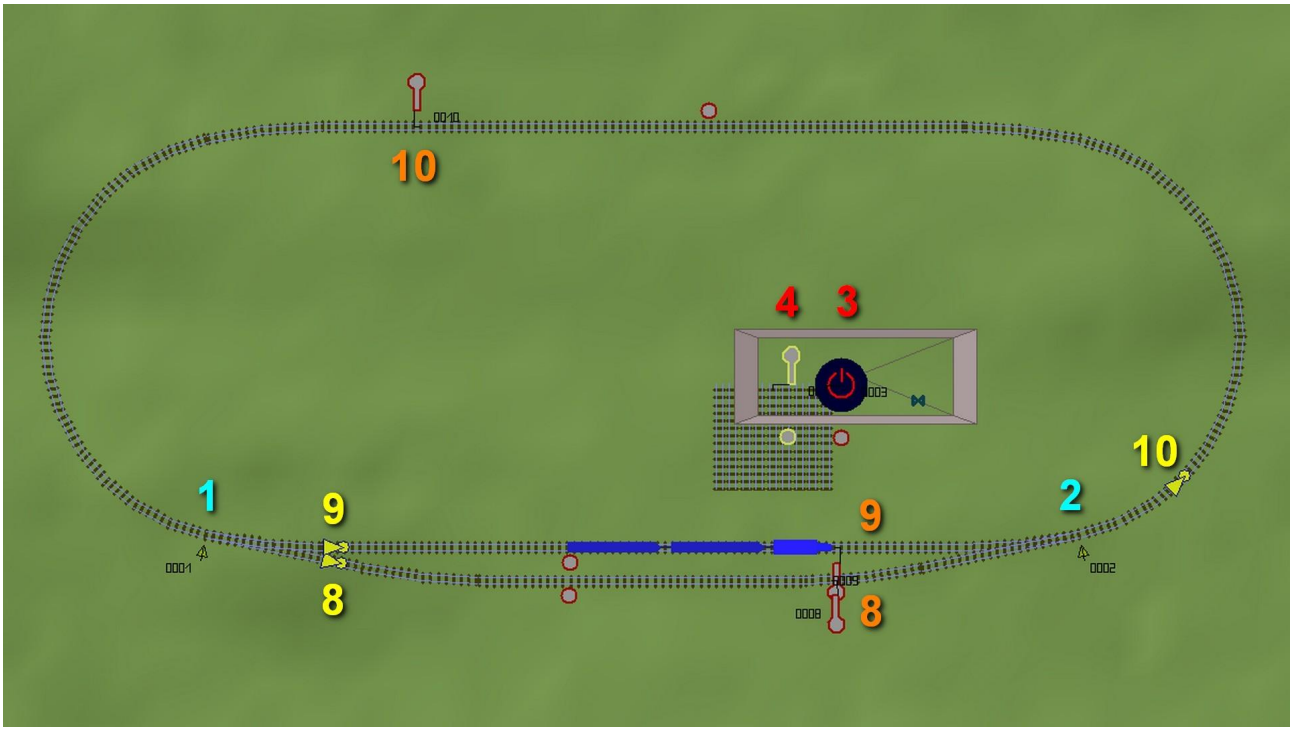
Schreiben wir die Lua-Konfiguration für diese einfache Anlage und lassen wir den Zug vollautomatisch fahren, mit Wartezeiten am Bahnhof.



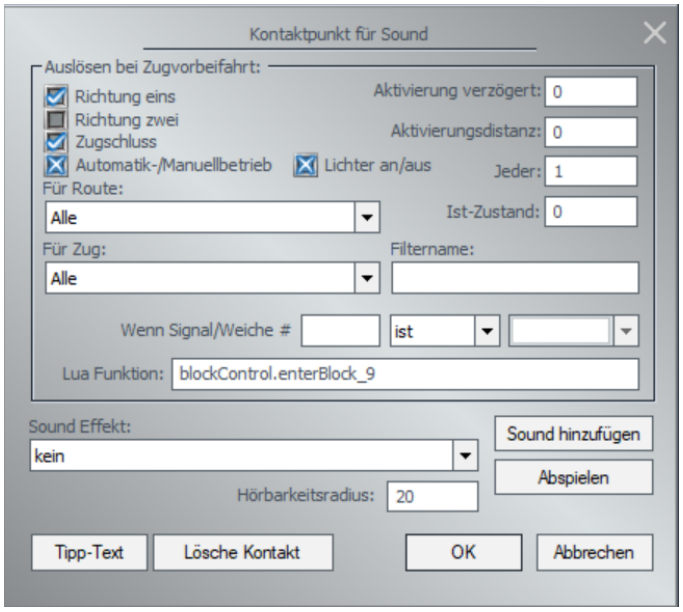
Beim automatischen Verkehr fahren die Züge von Block zu Block. Die erste Entscheidung, die wir treffen müssen, ist, welche Blöcke wir definieren wollen. Hier gibt es nur eine Regel: **Weichen können nicht Teil eines Blocks sein, sie sind Teil der Strecken zwischen den Blöcken.**

Für dieses Layout bietet es sich an, drei Blöcke wie hier gezeigt zu verwenden.

Um einen Block in EEP zu definieren, wird ein Signal an der Stelle platziert, an der der Zug anhalten soll.



Das obige Bild zeigt die Weichennummern (cyan), die Blocksignalnummern (orange) und die Nummern der Ein- und Ausschalter (rot) für diese Anlage. Wenn Sie eine ähnliche Anlage erstellen, können Ihre Nummern abweichen, da sie von EEP automatisch generiert werden und wir sie nicht auswählen können.



Neben dem Signal, das am Ende eines Blocks platziert wird, muss in jedem Block an dessen Eingang ein Gleiskontakt platziert werden. Beachten Sie, dass sich zwischen dem Kontakt und dem Signal keine Weichen befinden dürfen.

Diese Blockeingangskontakte sind normalerweise vom Typ "Sound".[1] Es muss ein Verweis auf eine Lua-Funktion angegeben werden, z.B.: `blockControl.enterBlock_9`.[2] Die Nummer muss die des Blocksignals sein. Dieser Kontakt teilt Lua nun mit, wenn ein Zug im Block angekommen ist.

[1] Jede Art von Kontakt funktioniert gut. Der wichtigste Teil ist die Eingabe der Lua-Funktion, die dem Modul mitteilt, dass Züge in einen Block einfahren.

[2] Denken Sie daran, dass Sie in einer bestimmten Reihenfolge arbeiten müssen:

1. Platzieren Sie Blocksignale und optional die Kontakte (aber ohne Verweis auf eine Lua-Funktion in diesen Kontakten - das Feld muss leer sein)
2. Passen Sie den Lua-Code an, um alle Blocksignale in einigen der Tabellen aufzulisten
3. Starten Sie das Layout im 3D-Modus
4. Platzieren Sie die Kontakte, falls noch nicht geschehen
5. Tragen Sie den Verweis auf die Lua-Funktion für die Blocksignale in die Kontakte ein

Diese Einschränkungen können durch die Verwendung des Lua-Moduls "BetterContacts" umgangen werden.

In den meisten Fällen ist "Ende des Zuges" die sichere Wahl, da die vorherigen Blöcke und Weichen freigegeben werden, sobald der letzte Wagen den Kontakt passiert hat. Wenn es keinen Gegenverkehr über die Weichen hinter dem Zug gibt, kann dieses Häkchen weggelassen werden. In diesem Fall werden die vorherigen Blöcke und Weichen freigegeben, sobald die Spitze des Zuges den Kontakt passiert hat, was zu einer etwas schnelleren Zugfolge führen kann.

Der Konfigurationscode für `EEP_Lua_layout_01` lautet:

```
main_switch = 3

local trains = {
  { name = "Steam", signal = 4, allowed = {[8]=15,[9]=1,[10]=1,}},
}

local routes = {
  { 8,10, turn={2,1} },
  { 9,10, turn={2,2} },
  {10, 8, turn={1,1} },
  {10, 9, turn={1,2} },
}
```

Das ist alles, was nötig ist, um dieses Layout zu beschreiben. Wenn Sie das Fenster "Lua-Skript-Editor" öffnen, oder wenn Sie die Lua-Datei in einem Editor wie Notepad++ öffnen, werden Sie feststellen, dass der Code eigentlich länger ist, aber es ist nur dieser obere Teil, der das Layout definiert. Der untere Teil des Lua-Codes muss nur bearbeitet werden, wenn die Zustände der verwendeten Signale unterschiedlich sind (siehe Kapitel “Signalzustände”) oder wenn Sie einige der verfügbaren Optionen ändern möchten (siehe Kapitel “Optionen”).

`main_switch = 3`
Hier wird die ID-Nummer des Signals angegeben, das als Hauptschalter (ein/aus) für das blockControl-Modul fungiert.

```
local trains = {
  { name = "Steam", signal = 4, allowed = {[8]=15,[9]=1,[10]=1,}},
}
```

`name = "Steam"`
Damit die automatische Zugerkennung funktioniert, muss der hier verwendete Zugname (mit oder ohne führendes "#") mit dem Namen identisch sein, der im Popup-Fenster eingegeben wurde, als der Zug auf das Gleis gestellt wurde. Siehe Kapitel “Wie man Züge auf dem Gleis platziert und Lua initialisiert”.

`signal = 4`
Jeder Zug kann sein eigenes individuelles Start-/Haltesignal besitzen. Diese ID-Nummer wird hier angegeben[3].

`allowed = {[8]=1,[9]=15,[10]=1,}`
Die Untertabelle `allowed` gibt an, welche Blöcke (identifiziert durch ihre Signalnummer) dieser Zug anfahren darf und ob es eine Haltezeit gibt:

- Wenn ein Block nicht erwähnt wird[4], bedeutet dies, dass dieser Zug niemals dorthin fahren wird. Wenn Sie zum Beispiel den Zug in Block 8 nicht zulassen wollen, sollte es heißen: `allowed = {[9]=15,[10]=1,}`

[3] Sie können dasselbe Signal für mehrere Züge verwenden, die Sie gemeinsam starten oder anhalten wollen.
[4] …oder den Wert `nil` oder 0 erhält

- Eine Blockangabe wie `[8]=1` bedeutet, dass dieser Zug den Block 8 benutzt und nur dann anhält, wenn das Signal rot bleibt, weil Blöcke oder Weichen vor ihm noch nicht frei sind.
- Eine Blockangabe wie `[9]=15` bedeutet, dass dieser Zug den Block 9 benutzt und mindestens 15 Sekunden lang in diesem Block verbleibt. Dies beinhaltet die Fahrzeit vom Sensor zum Signal, die von der Geschwindigkeit des Zuges, der Länge des Blocks und der Position des Vorsignals abhängt. Um genaue Haltezeiten zu erhalten, sollten Sie diese Fahrzeit messen.
- Wenn Sie die erlaubte Untertabelle weglassen, kann dieser Zug jeden Block anfahren und hat keine Haltezeiten.

```
local routes = {
  { 8,10, turn={2,1} },
  { 9,10, turn={2,2} },
  {10, 8, turn={1,1} },
  {10, 9, turn={1,2} },
}
```

Die Streckentabelle sagt Lua, wie die Weichen beim Fahren von einem Block zum nächsten[5] geschaltet werden müssen[6]. In diesem Beispiel gibt es bei allen Strecken genau eine Weiche zwischen den Blöcken. Es könnte auch keine sein[7], oder es könnten mehrere Weichen zwischen den Blöcken liegen:

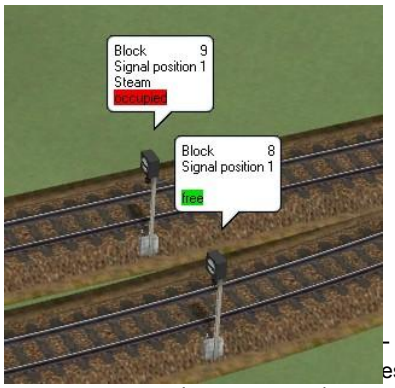
- Wenn zwischen den Blöcken 13 und 14 keine Weiche liegt, würde die Streckenbeschreibung wie folgt aussehen: `{13,14, turn={} },`
- Wenn die Blöcke 23 und 24 drei Weichen mit den Nummern 14, 5 und 6 dazwischen haben, würde die Streckenbeschreibung wie folgt aussehen:
  `{23,24, turn={14,1, 5,2, 6,1} },`

Sie finden ein Video zu Layout 01 hier:
https://www.youtube.com/watch?v=6X1fmBAHgpY&ab_channel=Rudysmodelrailway

Signalzustände

Leider haben in EEP nicht alle Signaltypen den gleichen Status. Bei einigen Signalen gilt Halt / 'rot' = 1, bei anderen Halt / 'rot' = 2. Wir müssen Lua mitteilen, ob die Zustände für Rot und Grün der Signale, die auf dem Layout verwendet werden, 1 oder 2 sind.

ACHTUNG: Alle auf der Anlage verwendeten Blocksignale müssen die gleichen Zustände haben. Das Gleiche gilt für alle Zugsignale.



Signalzustände werden in den Popups, die beim ersten Start eines Layouts erscheinen, als "Signalposition #" angezeigt. Sie können hier leicht ausgelesen werden und der Lua-Code kann bei Bedarf entsprechend geändert werden. Dies geschieht in den folgenden Codezeilen[8]:

```
MAINON      = 1, -- ON    state of main switch
MAINOFF     = 2, -- OFF   state of main switch
BLKSIGRED   = 1, -- RED   state of block signals
```

[5] es Nach-Blocks ist wichtig, nicht aber die genaue Position. Sie können Routen auch auf diese Weise definieren: `{ 23, turn={14,1 5,2, 6,1,}, 24 }`

[6] Der genaue Wert der Position einer Weiche spielt keine Rolle, wenn der Zug die Weiche aufschneidet, es ist jedoch in jedem Fall sinnvoll, korrekte Werte anzugeben.

[7] Wenn es keine Weiche gibt, können Sie für den Parameter `turn` eine leere Liste angeben oder diesen Parameter weglassen.

[8] Sie brauchen keine Signalzustände zu definieren, wenn die Vorgabe wie beschrieben mit den Standardwerten übereinstimmt.

```
BLKSIGGRN   = 2, -- GREEN state of block signals
TRAINSIGRED = 1, -- RED   state of train signals
TRAINSIGGRN = 2, -- GREEN state of train signals
```

Um diese Popups ein- oder auszuschalten, schalten Sie den Hauptschalter zweimal schnell mit Umschalt + Linksklick um.

Wie man Züge auf dem Gleis platziert und Lua initialisiert

Wenn der Lua-Code zum ersten Mal ausgeführt wird oder wenn Sie im Fenster "Lua-Skript-Editor" auf "Skript neu laden" klicken, startet Lua im *Suchmodus für Züge*. Es findet automatisch jeden Zug, der durch ein rotes Signal angehalten wird.

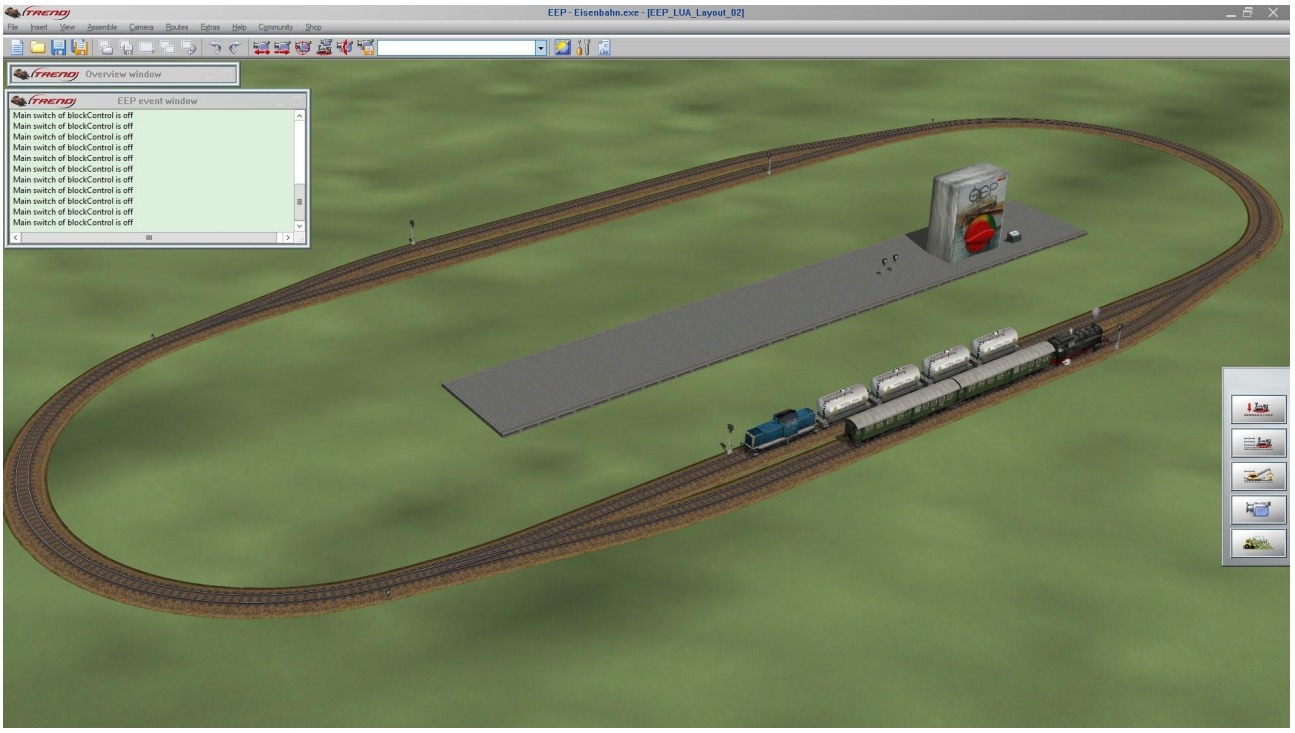
Wenn sich noch keine Züge auf der Anlage befinden, müssen wir mindestens einen platzieren. Aktivieren Sie den 3D-Bearbeitungsmodus im Fenster "Steuerungsdialog" und platzieren Sie einen Zug vor dem Vorsignal eines Blocks, auf dem Sie diesen Zug fahren lassen. In dem sich öffnenden Popup-Fenster geben Sie den Namen des Zuges ein, der genau mit dem Namen übereinstimmen muss, den er in der Lua-Konfiguration erhalten hat oder erhalten wird.

Wechseln Sie nun im Fenster "Steuerungsdialog" in den Fahrbetrieb. Wählen Sie den Zug mit einem linken Mausklick aus und geben Sie manuell eine Zuggeschwindigkeit ein. Der Zug fährt nun bis zum Blocksignal und hält dort an. Lua hat den Zug nun erkannt, er meldet dies im EEP-Ereignisfenster.

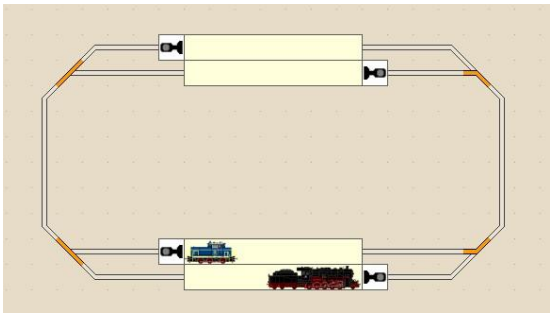
Wenn alle gewünschten Züge platziert und erkannt worden sind, kann die Hauptsignal eingeschaltet werden, gefolgt von den einzelnen Zugsignalen. Ab hier steuert Lua die Züge automatisch.

Falls Sie später Züge hinzufügen oder entfernen möchten, öffnen Sie das Fenster "Lua-Skript-Editor" und klicken Sie auf "Skript neu laden". Lua befindet sich nun wieder im *Suchmodus für Züge*.

Demo EEP_Lua_Layout_02

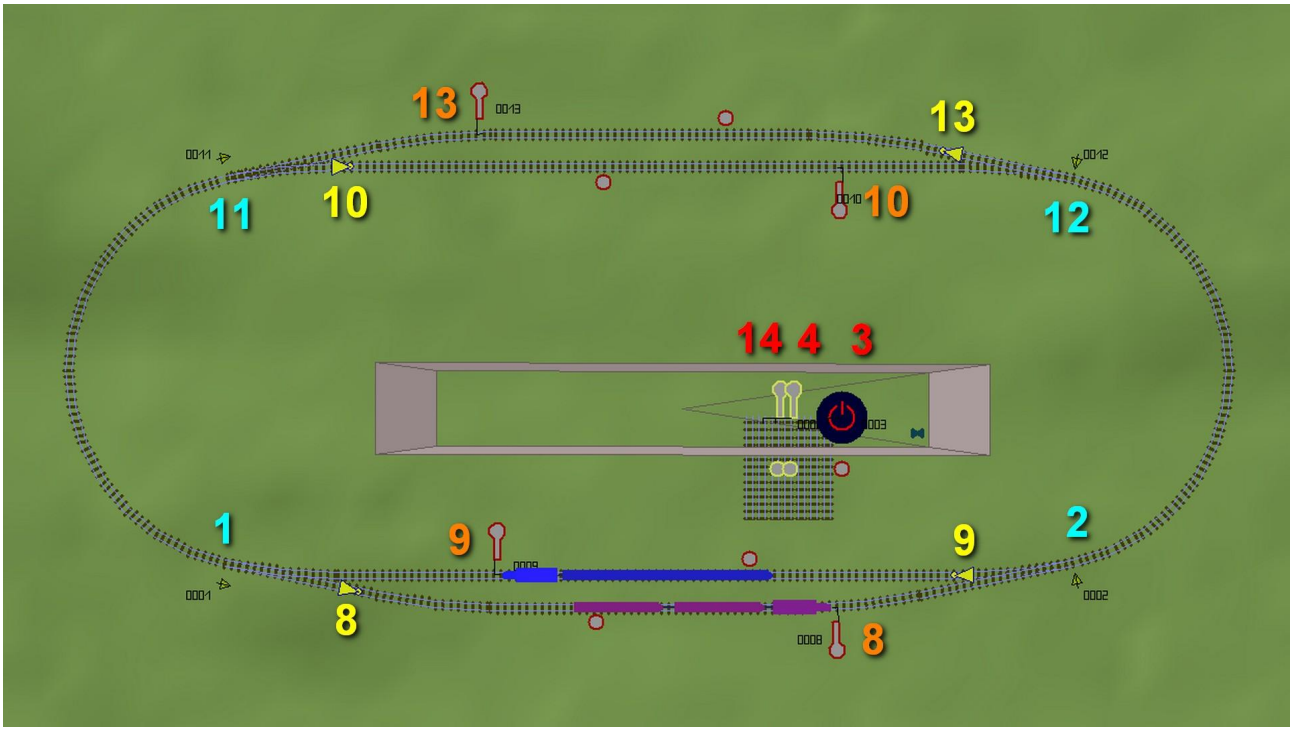


Fügen wir einen Block und einen zweiten Zug hinzu und haben wir Verkehr in zwei Richtungen.



Für dieses Layout definieren wir 4 Blöcke, 2 am Bahnhof Nord, 2 am Bahnhof Süd. Auch wenn die Züge in entgegengesetzte Richtungen fahren, ist der Verkehr in allen Blöcken immer noch in eine Richtung. Im nächsten Demo-Layout 03 werden wir sehen, wie man Blöcke mit Verkehr in zwei Richtungen erstellen kann.

Die 2D-Übersicht unten zeigt die Weichen (cyan), die Blocksignale (orange) und die Ein/Aus-Schaltsignale (rot). Da wir nun Gegenverkehr haben, müssen wir darauf achten, dass die Blockeinfahrsensoren so eingestellt sind, dass sie bei "Ende des Zuges" auslösen, da sonst die Weiche zu früh freigegeben werden könnte und der Gegenzug bereits losfährt, während das Ende des Zuges, der in den Bahnhof einfährt, noch auf der Weiche steht.



Der Konfigurationscode für `EEP_Lua_layout_02` lautet:

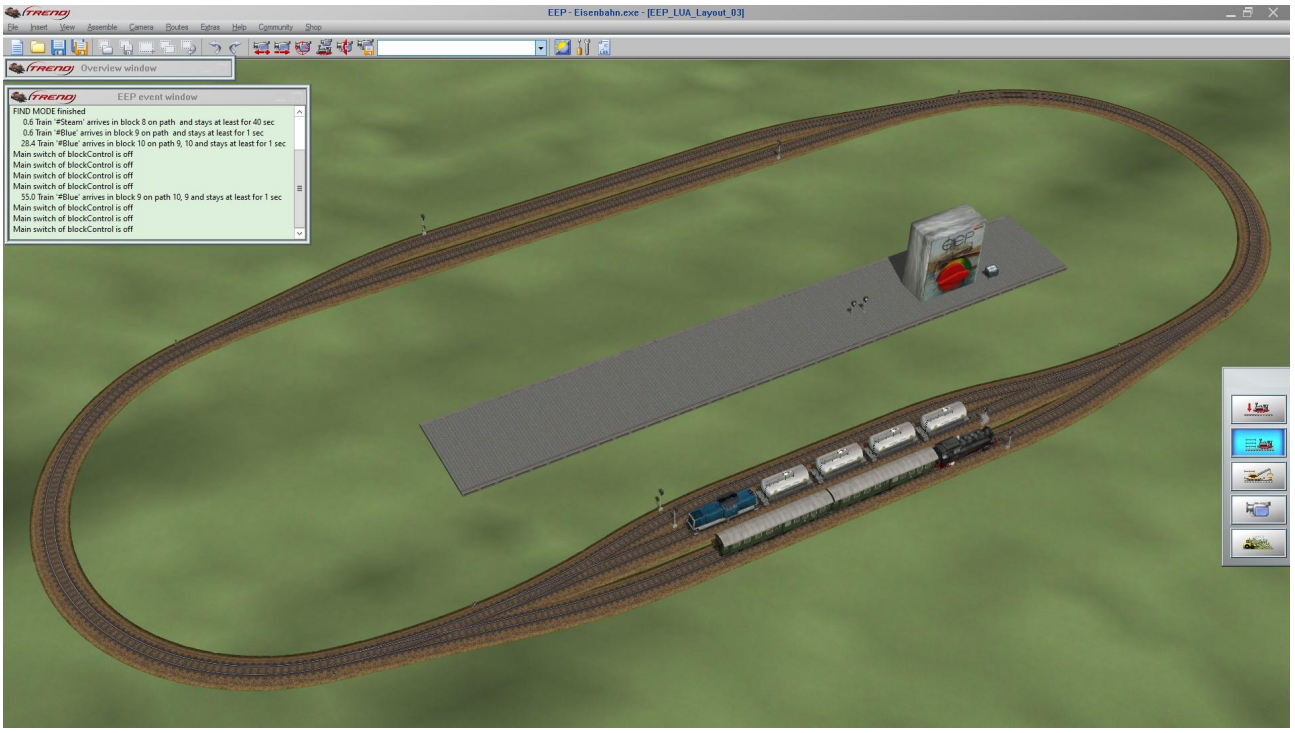
```
local main_signal = 3

local counterclockwise = {[8]=15, [13]=1, } -- allowed blocks for CCW trains
local clockwise        = {[9]=20, [10]=1, } -- allowed blocks for CW  trains

local trains = {
  { name = "Steam", signal = 14, allowed = counterclockwise },
  { name = "Blue",  signal =  4, allowed = clockwise        },
}

local routes = {
  {  8, 13, turn={  2,1, 12,1 }}, -- from block A, to block B, turnouts
  {  9, 10, turn={  1,2, 11,2 }},
  { 10,  9, turn={ 12,2,  2,2 }},
  { 13,  8, turn={ 11,1,  1,1 }},
}
```

In diesem Beispiel werden die zulässigen Blöcke in Gruppen in separaten Tabellen angegeben. Eine Tabelle gibt die zulässigen Blöcke und die Wartezeiten für Züge "gegen den Uhrzeigersinn" an. Die andere Tabelle beschreibt die Züge im Uhrzeigersinn. Dies sind nur Namen, Sie können jeden beliebigen Namen verwenden, wie "Fracht" oder "ICE", solange dieselben Namen in der Tabelle `trains` verwendet werden.

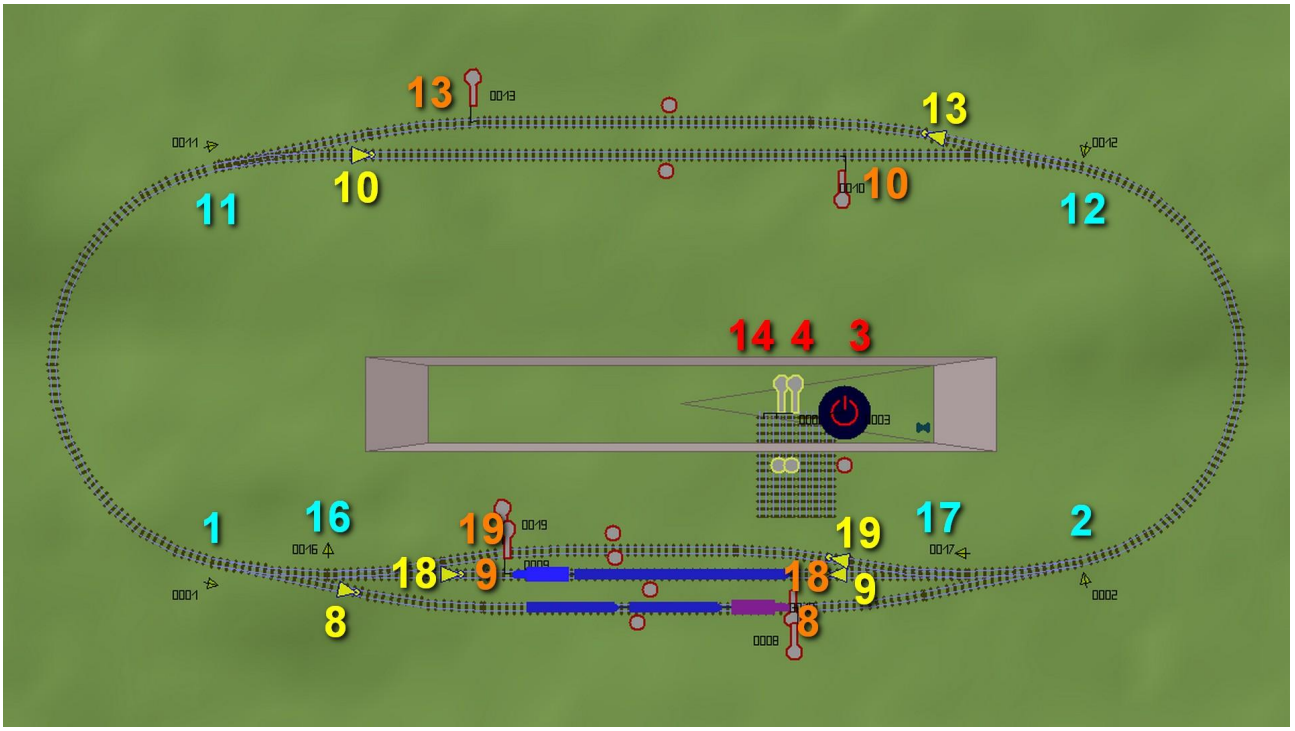
Auf dieser Anlage mit nur zwei Zügen mag der Vorteil der Angabe von erlaubten Blöcken auf diese Weise gering erscheinen. Auf einer größeren Anlage mit mehreren Zügen, die auf denselben Blöcken erlaubt sind, und mehreren anderen Zügen, die auf anderen Blöcken erlaubt sind, ist diese Gruppierung der erlaubten Blöcke jedoch sehr vorteilhaft, da sie die endlose Wiederholung identischer `allowed` Untertabellen für jeden Zug verhindert.

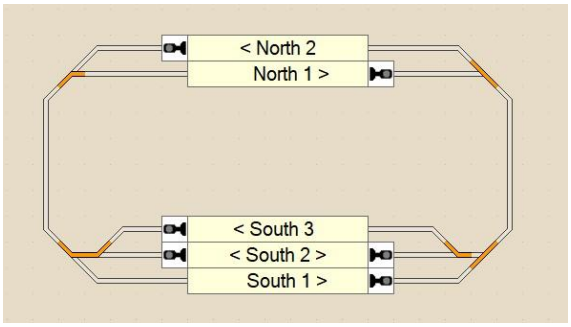
Demo EEP_Lua_Layout_03

Wir haben jetzt Gegenverkehr in einem Block. Der Bahnhof Süd wird um ein mittleres Gleis erweitert, auf dem die Züge sowohl in Richtung Osten als auch in Richtung Westen fahren können sollen.



Das bedeutet, dass wir ein Signal an beiden Enden dieses Gleises platzieren müssen, sie haben die Nummern 9 und 18 in der Abbildung unten. Wir brauchen auch einen Blockeingangssensor an den gegenüberliegenden Seiten. Dadurch wird dieser Block effektiv zu einem Paar überlappender Zwei-Wege-Blöcke.





Sobald ein Zug einen dieser Zwei-Wege-Blöcke reserviert, muss Lua auch den anderen Block reservieren. Das Gleiche gilt, wenn der Zug an seinem Zielblock ankommt und der Abfahrtsblock freigegeben werden kann, dann muss auch der zugehörige Zwei-Wege-Block freigegeben werden.

Wir müssen also Lua über diese Zwei-Wege-Blöcke informieren. Dies geschieht über die Tabelle `two_way_blocks`.

Der Konfigurationscode für `EEP_Lua_layout_03` lautet:

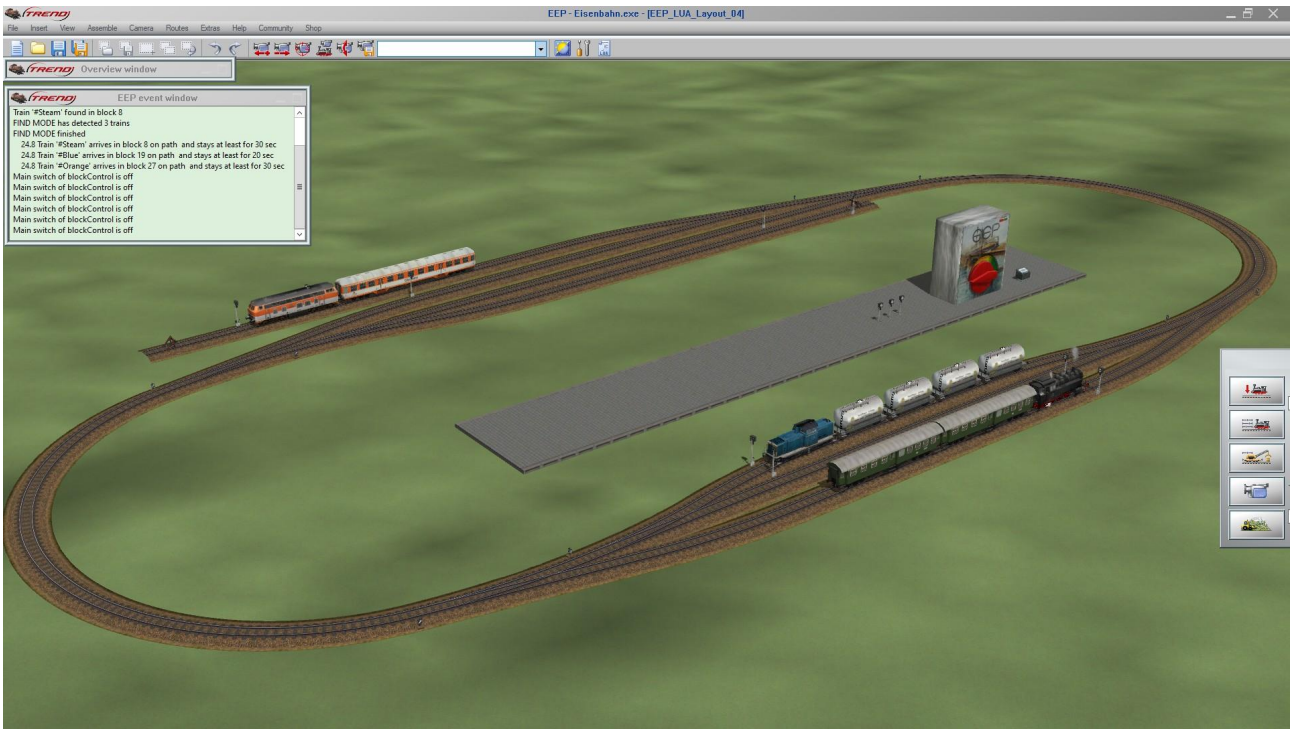
```
local main_signal = 3

local counterclockwise = { -- allowed blocks for CCW trains
  [ 8] = 40, -- station South track 1 -> East
  [18] =  1, -- station South track 2 -> East
  [13] =  1, -- station North track 2 -> West
}

local clockwise = { -- allowed blocks for CW trains
  [ 9] =  1, -- station South track 2 -> West
  [19] = 20, -- station South track 3 -> West
  [10] =  1, -- station North track 1 -> East
}

local trains = {
  { name = "Steam", signal = 14, allowed = counterclockwise },
  { name = "Blue",  signal =  4, allowed = clockwise },
}

local two_way_blocks = { {9, 18} }
```

```
local routes = {
  {  8, 13, turn={  2,1, 12,1 }}, -- from block A, to block B, turnouts
  { 18, 13, turn={  2,2, 12,1, 17,1 }},
  {  9, 10, turn={ 16,1,  1,2, 11,2 }},
  { 19, 10, turn={ 16,2,  1,2, 11,2 }},
  { 10,  9, turn={ 12,2,  2,2, 17,1 }},
  { 10, 19, turn={ 12,2,  2,2, 17,2 }},
  { 13,  8, turn={ 11,1,  1,1 }},
  { 13, 18, turn={ 11,1,  1,2, 16,1 }},
}
```

Die Tabelle `two_way_blocks` wird nur benötigt, wenn es mindestens ein Paar von Zwei-Wege-Blöcken zu deklarieren gibt. Wenn es keine gibt, kann die Tabelle einfach weggelassen werden. Die Zwei-Wege-Blöcke werden immer paarweise deklariert. Wenn es mehrere Zwei-Wege-Blöcke gibt, könnte die Tabelle wie folgt aussehen:
`local two_way_blocks = { {82, 81}, {32, 33}, {74, 38}, }`

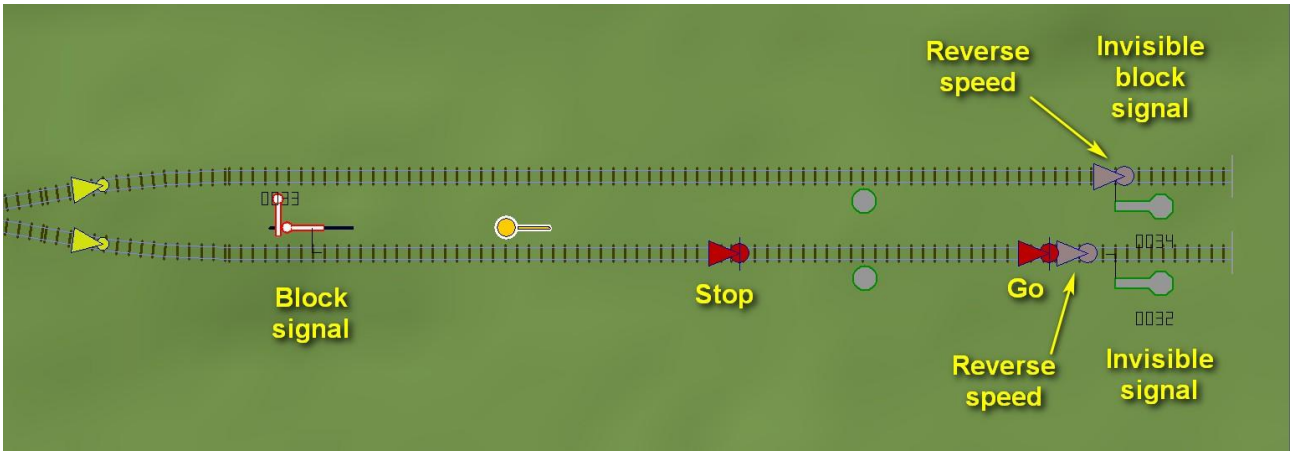
Was sich ebenfalls geändert hat, ist die Art und Weise, wie die zulässigen Blöcke aufgelistet werden. Durch die Verwendung mehrerer Zeilen, wie hier gezeigt, ist es möglich, bei größeren Layouts Kommentare hinzuzufügen, um zu beschreiben, welcher Block welcher ist.

Demo EEP_Lua_Layout_04



Für das Lua-Steuerungssystem funktioniert ein Sackgassenblock ähnlich wie ein normaler Block. Und obwohl der Verkehr auf einem Sackgassengleis in beide Richtungen verläuft, handelt es sich immer noch um einen einzelnen Block mit nur einem Blocksignal.

Das folgende Bild zeigt zwei Möglichkeiten, einen Sackgassenblock in EEP zu bauen, die beide gut mit dem Lua-Kontrollsystem für automatischen Verkehr funktionieren.

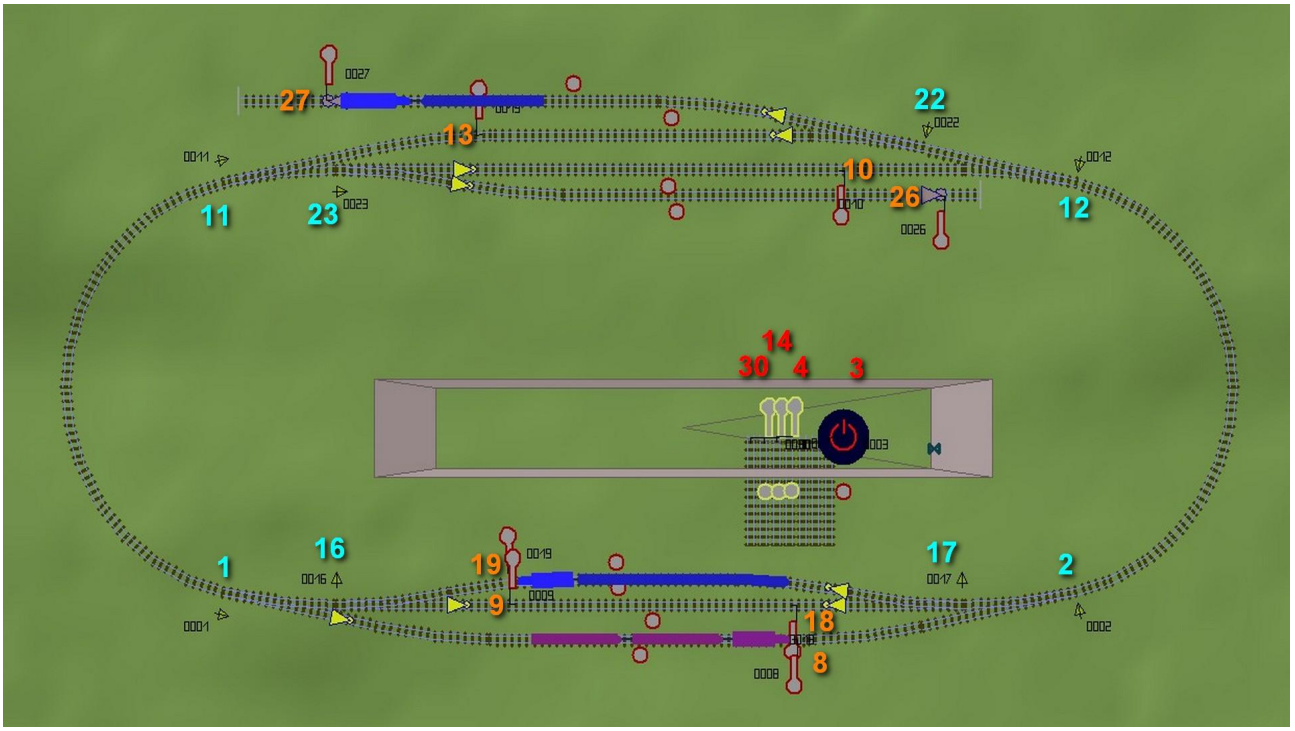


Der oberste Block hat sein (unsichtbares) Blocksignal am Ende. Der Zug wird dort anhalten. Wenn Lua ihn wieder fahren lässt, fährt er auf die Sackgasse zu, aber direkt hinter dem Signal befindet sich ein "Fahrzeug"-Sensor, der seine Geschwindigkeit umkehrt. Es kann ein wenig Feinabstimmung erfordern, um diesen Sensor so nah wie möglich am Signal zu platzieren, so dass die Geschwindigkeit des Zuges immer noch so niedrig ist, dass die Geschwindigkeitsumkehr fast nicht auffällt. Der Vorteil dieser Methode ist, dass sie schnell zu bauen ist. Der Nachteil ist, dass es an diesem Block kein sichtbares Signal gibt, das anzeigt, dass der Zug den Block verlassen darf, obwohl es dafür eine Abhilfe gibt: Platzieren Sie ein sichtbares Signal am Blockausgang und verbinden Sie in den Eigenschaften des Lua-Blocks das Signal mit diesem sichtbaren Signal. Es schaltet nun auf grün, wenn das (unsichtbare) Blocksignal von Lua auf grün geschaltet wird, und ebenso schaltet es wieder auf rot.

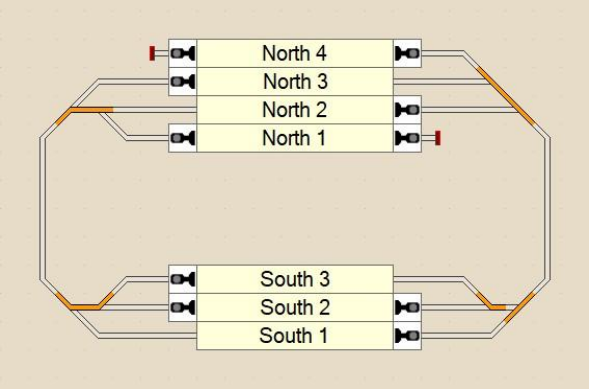
Der untere Block hat ein sichtbares Blocksignal, das direkt von Lua gesteuert wird. Bei dieser Sackgassen-Methode wird der Zug durch ein unsichtbares Signal am Gleisende gestoppt, das nur dazu da ist, den Zug rückwärts fahren zu lassen, das blockControl-Modul kennt dieses Signal nicht. Dieses Signal wird über einen "Signal"-Sensor, der vor dem Vorsignal platziert werden muss, auf "Rot"/"Halt" gesetzt. Kurz bevor der Zug zum Stillstand kommt, wird das Signal durch einen zweiten "Signal"-Sensor auf "grün"/"los" gesetzt. Der Zug wird nun über einen "Fahrzeug"-Sensor in der Nähe des Signals, das mit sehr geringer Geschwindigkeit angefahren werden soll, reversiert. Die genaue Platzierung dieser Sensoren erfordert möglicherweise eine gewisse Feinabstimmung, damit die Geschwindigkeitsumkehr gut aussieht. Der Zug fährt nun zurück zum Blocksignal, wo er anhält.

Sie können wählen, welche Methode Sie bevorzugen, beide funktionieren gut mit der automatischen Zugsteuerung von Lua.

Das folgende Bild zeigt die 2D-Ansicht von `EEP_Lua_Layout_04`.



Der Konfigurationscode für EEP_Lua_layout_04 lautet:




```lua
local main_signal = 3

local block_signals = {
   8, -- South 1 -> East
  18, -- South 2 -> East two way
   9, -- South 2 -> West two way
  19, -- South 3 -> West
  26, -- North 1 dead end
  10, -- North 2 -> East
  13, -- North 3 -> West
  27, -- North 4 dead end
}

local everywhere = {
  [ 8] = 30, -- numbers > 1 are stop times
  [18] =  1,
  [ 9] =  1,
  [19] = 30,
  [26] = 30,
  [10] = 20,
  [13] = 20,
  [27] = 30,
}

local trains = {
  { name = "Orange", signal =  4, allowed = everywhere },
  { name = "Steam",  signal = 30, allowed = everywhere },
  { name = "Blue",   signal = 14, allowed = everywhere },
}

local two_way_blocks = { {18, 9} }
```

```
local routes = {
  {  8, 13, turn={  2,1, 12,1, 22,2 }},
  {  8, 27, turn={  2,1, 12,1, 22,1 }},
  { 18, 13, turn={ 17,1,  2,2, 12,1, 22,2 }},
  { 18, 27, turn={ 17,1,  2,2, 12,1, 22,1 }},
  {  9, 26, turn={ 16,1,  1,2, 11,2, 23,2 }},
  {  9, 10, turn={ 16,1,  1,2, 11,2, 23,1 }},
  { 19, 26, turn={ 16,2,  1,2, 11,2, 23,2 }},
  { 19, 10, turn={ 16,2,  1,2, 11,2, 23,1 }},
  { 26,  8, turn={ 23,2, 11,2,  1,1 }},
  { 26, 18, turn={ 23,2, 11,2,  1,2, 16,1 }},
  { 10,  9, turn={ 12,2,  2,2, 17,1 }},
  { 10, 19, turn={ 12,2,  2,2, 17,2 }},
  { 13,  8, turn={ 11,1,  1,1 }},
  { 13, 18, turn={ 11,1,  1,2, 16,1 }},
  { 27,  9, turn={ 22,1, 12,1,  2,2, 17,1 }},
  { 27, 19, turn={ 22,1, 12,1,  2,2, 17,2 }},
}
```

Da sich die Sackgassenblöcke im blockControl-Modul ähnlich wie normale Blöcke verhalten, muss im Konfigurationscode nichts anders gemacht werden, damit die Sackgassenblöcke funktionieren.

Neu ist hier die Einführung der Tabelle `block_signals`. Diese Tabelle listet einfach alle Blocksignale auf, in keiner bestimmten Reihenfolge. Die Tabelle ist nicht zwingend erforderlich, wenn sie weggelassen wird, funktioniert der Lua-Code einwandfrei. Es kann zwei Gründe geben, die Tabelle einzufügen:

- Eine Liste aller verfügbaren Blöcke, vielleicht mit einigen Kommentaren zu den einzelnen Blöcken, kann dabei helfen, Fehler beim Ausfüllen der Tabellen mit den erlaubten Blöcken oder der Streckentabelle zu vermeiden.
- Der Lua-Code hat einige Konsistenzprüfungen eingebaut, von denen eine prüft, ob alle aufgelisteten Blöcke in der Routentabelle mindestens einmal als Abfahrtsort und auch mindestens einmal als Zielort verwendet werden, oder ob versehentlich ein nicht vorhandener Block in einer Route verwendet wird. Wenn diese Prüfung fehlschlägt, wird eine Warnung ausgegeben. Siehe das Kapitel über Konsistenzprüfungen für weitere Einzelheiten.

In dem obigen Beispiel verwenden alle drei Züge alle Blöcke. Angenommen, wir wollen etwas anderes, wie:

- Der "Dampf"-Zug fährt auf den Blöcken 8 und 13 gegen den Uhrzeigersinn, mit einem Halt bei 8.
- Der "Orange" Zug fährt im Uhrzeigersinn auf den Blöcken 10 und 9 oder 19. Nur bei 19 gibt es einen Halt, bei 9 verhält er sich wie ein Intercity und fährt durch den Bahnhof.
- Der "blaue" Güterzug fährt zwischen den Sackgassen 26 und 27 hin und her und fährt immer über die Zwei-Wege-Blöcke 9 und 18.

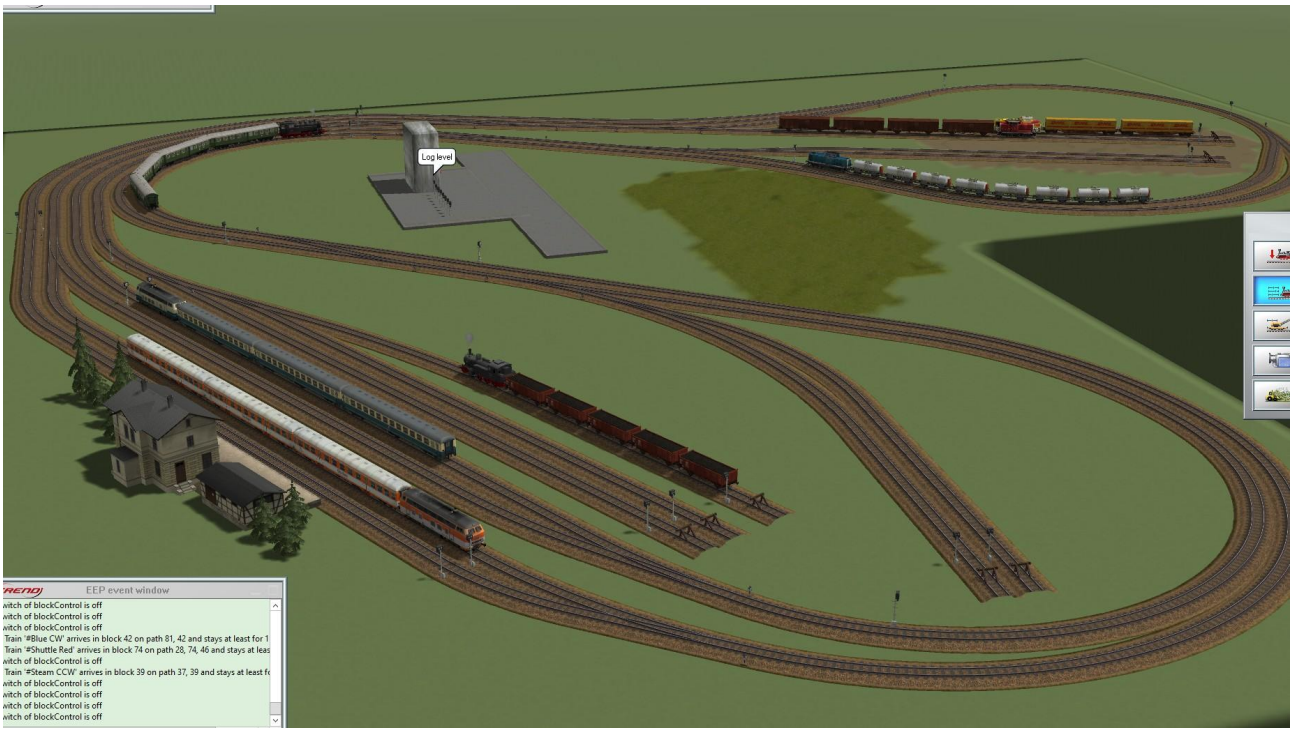
Damit dies funktioniert, müssen Sie lediglich sicherstellen, dass die aktuelle Position und Richtung der Züge mit den zukünftigen Einstellungen übereinstimmt – Zug „Orange“ muss im Uhrzeigersinn fahren, Zug „Steam“ muss gegen den Uhrzeigersinn fahren, Zug “Blue” muss sich auf erlaubten Blöcken befinden – und dann die zulässigen Blöcke und die Zugtabellen ändern:

```
local CCW   = { [ 8]=30, [13]= 1, }          -- counterclockwise
local CW    = { [ 9]= 1, [10]= 1, [19]=30, } -- clockwise
local cargo = { [ 9]= 1, [18]= 1, [26]=30, [27]=30, }

local trains = {
```

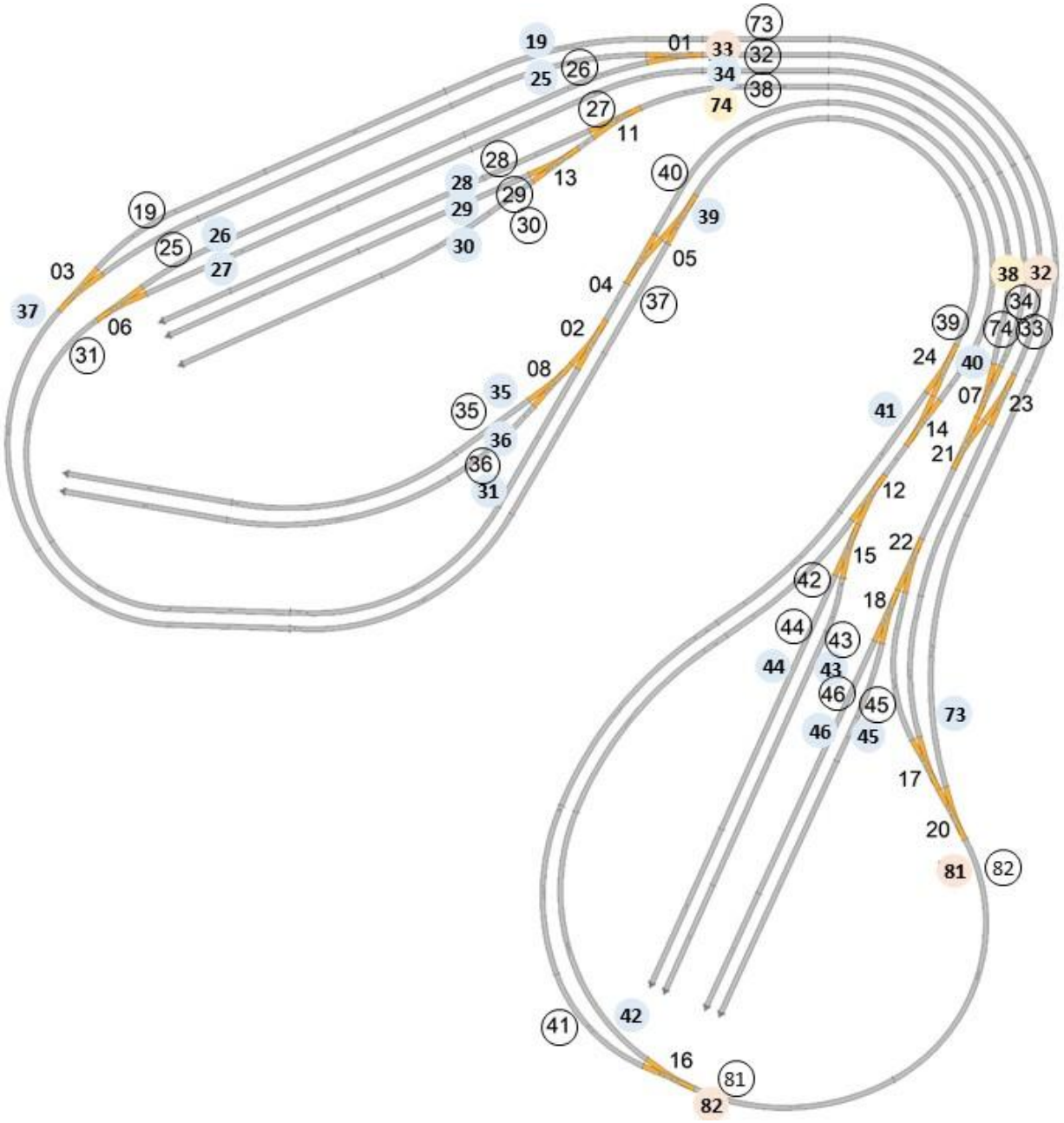
```
  { name = "Orange", signal =  4, allowed = CW },
  { name = "Steam",  signal = 30, allowed = CCW },
  { name = "Blue",   signal = 14, allowed = cargo },
}
```

Demo EEP_Lua_Layout_05

Es ist Zeit für etwas Anspruchsvolleres ... eine Anlage mit 27 Blöcken, auf denen 7 Züge vollautomatisch fahren, Lua-gesteuert.



Für den Entwurf dieser Anlage wurde das Modellbahn-Editorprogramm SCARM verwendet. Die untenstehende SCARM-Zeichnung zeigt die Blocksignalnummern (weiße Kreise) und deren Gleiskontakte (farbige Kreise) sowie die Weichennummern, wie sie von EEP generiert wurden.



Der Konfigurationscode für `EEP_Lua_layout_05` lautet:

```
local main_signal = 80

local block_signals  = { -- This table is optional, just for info
  19, 25, 26, 27,             -- Station North
  28, 29, 30,                 -- Cargo Station North
  35, 36,                     -- Cargo Station West
  43, 44,                     -- Cargo Station South-West
  45, 46,                     -- Cargo Station South-East
  37, 39, 41, 82, 73, 32, 38, -- Connecting tracks CCW
  33, 34, 74, 81, 42, 40, 31, -- Connecting tracks CW
}

local CCW = { -- allowed blocks and wait times CCW trains
  [19] = 45, [25] = 45,
  [37] = 1,
```

```
  [39] = 1,
  [41] = 1,
  [82] = 1,
  [73] = 1, [32] = 1,
}

local CW = { -- allowed blocks and wait times CW trains
  [26] = 40, [27] = 30,
  [33] = 1,  [34] =  1,
  [81] = 1,
  [42] = 1,
  [40] = 1,
  [31] = 1,
}

local shuttles = { -- allowed blocks and wait times cargo shuttles
  [28] = 28, [29] = 28, [30] = 28,
  [74] = 1,  [38] = 1,
  [45] = 28, [46] = 28,
  [32] = 1,
  [25] = 1,
  [37] = 1,
  [39] = 1,
  [43] = 28, [44] = 28,
  [40] = 1,
  [35] = 28, [36] = 28,
  [31] = 1,
  [26] = 1, [27] = 1,
  [33] = 1, [34] = 1,
}

local trains = {
  { name="Steam CCW",       signal =  9, allowed = CCW },
  { name="Orange CCW",      signal = 72, allowed = CCW },
  { name="Blue CW",         signal = 77, allowed = CW  },
  { name="Cream CW",        signal = 78, allowed = CW  },
  { name="Shuttle Red",     signal = 79, allowed = shuttles },
  { name="Shuttle Yellow", signal = 92, allowed = shuttles },
  { name="Shuttle Steam",  signal = 93, allowed = shuttles },
}

local two_way_blocks = { {82, 81}, {32, 33}, {74, 38}, }

local f = 1 -- turnout position "fahrt / main"
local a = 2 -- turnout position "abzweig / branch"
local routes = {
  { 19, 37, turn={  3,f }},
  { 25, 37, turn={  3,a }},
  { 26, 33, turn={  1,a }},
  { 27, 34, turn={}},
  { 28, 74, turn={ 11,f }},
  { 29, 74, turn={ 13,f, 11,a }},
  { 30, 74, turn={ 13,a, 11,a }},
  { 31, 26, turn={  6,f }},
  { 31, 27, turn={  6,a }},
  { 32, 25, turn={  1,f }},
  { 33, 45, turn={ 23,a, 21,a, 22,f, 18,a }},
  { 33, 46, turn={ 23,a, 21,a, 22,f, 18,f }},
  { 33, 81, turn={ 23,f, 17,a, 20,f }},
```

```
  { 34, 45, turn={  7,f, 21,f, 22,f, 18,a }},
  { 34, 46, turn={  7,f, 21,f, 22,f, 18,f }},
  { 34, 81, turn={  7,f, 21,f, 22,a, 17,f, 20,f }},
  { 35, 39, turn={  8,a,  2,a,  4,a,  5,f }},
  { 36, 39, turn={  8,f,  2,a,  4,a,  5,f }},
  { 37, 39, turn={  5,a }},
  { 38, 28, turn={ 11,f }},
  { 38, 29, turn={ 11,a, 13,a }},
  { 38, 30, turn={ 11,a, 13,f }},
  { 39, 41, turn={ 24,a }},
  { 39, 44, turn={ 24,f, 14,a, 12,a, 15,f }},
  { 39, 43, turn={ 24,f, 14,a, 12,a, 15,a }},
  { 40, 31, turn={  4,f,  2,f }},
  { 40, 35, turn={  4,f,  2,a,  8,a }},
  { 40, 36, turn={  4,f,  2,a,  8,f }},
  { 41, 82, turn={ 16,f }},
  { 42, 40, turn={ 12,f, 14,f }},
  { 43, 40, turn={ 15,a, 12,a, 14,f }},
  { 44, 40, turn={ 15,f, 12,a, 14,f }},
  { 45, 32, turn={ 18,a, 22,f, 21,a, 23,a }},
  { 45, 38, turn={ 18,a, 22,f, 21,f,  7,a }},
  { 46, 32, turn={ 18,f, 22,f, 21,a, 23,a }},
  { 46, 38, turn={ 18,f, 22,f, 21,f,  7,a }},
  { 73, 19, turn={}},
  { 74, 45, turn={  7,a, 21,f, 22,f, 18,a }},
  { 74, 46, turn={  7,a, 21,f, 22,f, 18,f }},
  { 74, 81, turn={  7,a, 21,f, 22,a, 17,f, 20,f }},
  { 81, 42, turn={ 16,a }},
  { 82, 32, turn={ 20,f, 17,a, 23,f }},
  { 82, 73, turn={ 20,a }},
}
```

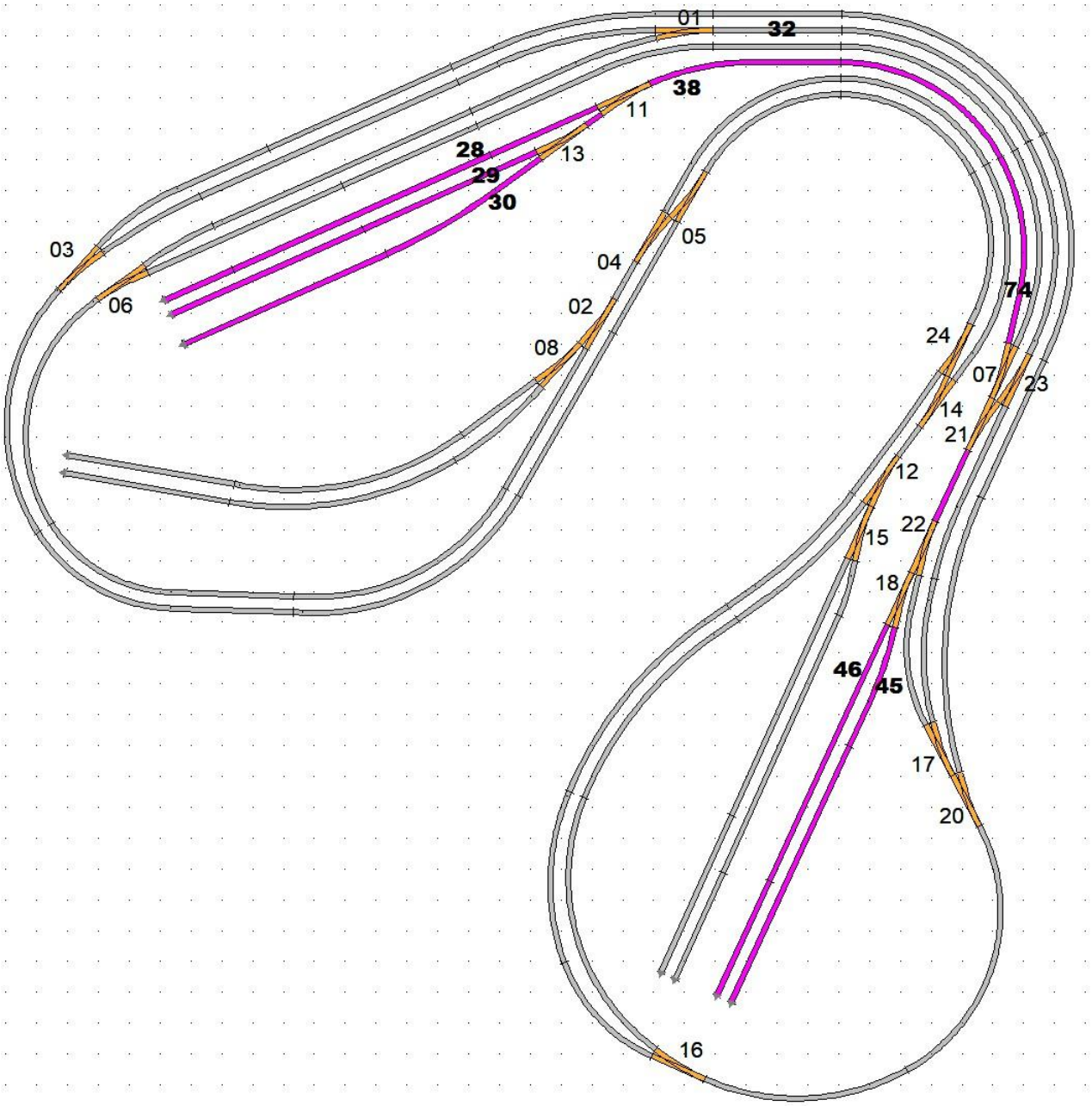
Das einzige neue Prinzip, das hier eingeführt wurde, ist die Verwendung der Variablen `f=1` und `a=2`. Sie machen die Zustände "Fahrt/main" und "Abzweig/branch" der Weichen besser lesbar.

Ja, es ist etwas mehr Arbeit, ein solches Layout zu definieren, aber es ist ziemlich einfach. Die Herausforderung besteht darin, konzentriert zu bleiben und keinen einzigen Fehler in einer der Tabellen zu machen. Es gibt etwas Hilfe ... der Lua-Code hat mehrere Konsistenzprüfungen eingebaut, die Warnungen ausgeben, wenn nicht alle Blöcke in Strecken verwendet werden oder wenn versehentlich eine Nummer verwendet wird, die keine existierende Signal- oder Weichennummer ist. Siehe das Kapitel über Konsistenzprüfungen für weitere Informationen.

Wie man Stillstände vermeidet

Bei manchen Anlagen kann es vorkommen, dass Züge in entgegengesetzte Richtungen fahren und keiner von ihnen ein freies Gleis findet, auf das er fahren kann. Die Züge kommen zum Stillstand, ein sogenannter "Deadlock" ist eingetreten.

Das Bild unten zeigt einen Abschnitt von `layout_05` in Pink, wo es zu einer Blockade kommen kann, wenn wir 3 Pendelzüge auf diesem Abschnitt haben und wir den Zügen nicht erlauben würden, über Block 32 zu "entkommen". Angenommen, die beiden Blöcke 45 und 46 sind besetzt und der dritte Zug, der sich auf einem der Blöcke 28, 29, 30 befindet, beginnt, zum Block 74 zu fahren. Sobald er in Block 74 angekommen ist, kann keiner der 3 Züge mehr eine neue Route finden, da es keine freien Blöcke mehr gibt ... wir haben einen Stillstand.



Wie lassen sich Stillstände vermeiden?

- Die naheliegende Lösung wäre, eine Anlage zu entwerfen, bei der es gar nicht erst zu Blockierungen kommen kann. Im `Layout_05` ist dies leicht möglich, indem die Züge auch in den Block 32 fahren können und so einen der Blöcke 45 oder 46 freigeben.
- Eine andere Lösung könnte darin bestehen, 38 und 74 in Lua nicht als Blöcke zu definieren, so dass diese Gleise im Wesentlichen Teil der Weichen sind. Lua prüft, ob ein benachbarter Block frei ist. Wenn 74 kein Block mehr ist, prüft Lua 45 und 46. Wenn beide besetzt sind, wird ein Zug an 28,29,30 nicht losfahren, erst ein Zug aus 45,46 wird in die andere Richtung fahren. Um trotzdem sichtbare Zugsignale zu haben, können weitere Signale verwendet werden, die über Gleiskontakte gesteuert werden, jedoch nicht unter Kontrolle des blockControl-Moduls stehen.
- Die dritte Lösung besteht darin, Lua über den möglichen Stillstandspfad zu informieren und weiter als einen Block vorauszuschauen. Lua sollte nicht zulassen, dass ein Zug auf 28, 29 oder 30 losfährt, wenn es keinen freien Block in 45 oder 46 gibt. Wir weisen Lua an, dies über die Tabelle `anti_deadlock_paths` zu tun.

```
local anti_deadlock_paths = {
  { {28,29,30}, 74, {46,45} },
}
```

Der Pfad in der anderen Richtung, `{ {46,45}, 38, {28,29,30} },` kann ebenfalls angegeben werden, ist aber in diesem Fall nicht erforderlich, da für die 3 Züge 3 Blöcke zur Verfügung stehen und eine Blockade in dieser Richtung einfach nicht auftreten kann.

Wenn Sie einen sehr langen Zwischenweg haben, den Sie in mehrere Blöcke aufteilen möchten, ist das völlig in Ordnung, der Anti-Stopp-Pfad könnte dann z.B. so aussehen:
`{ {28,29,30}, 74, 75, 76, {46,45} },`

Wie man Kollisionen an Kreuzungen verhindert

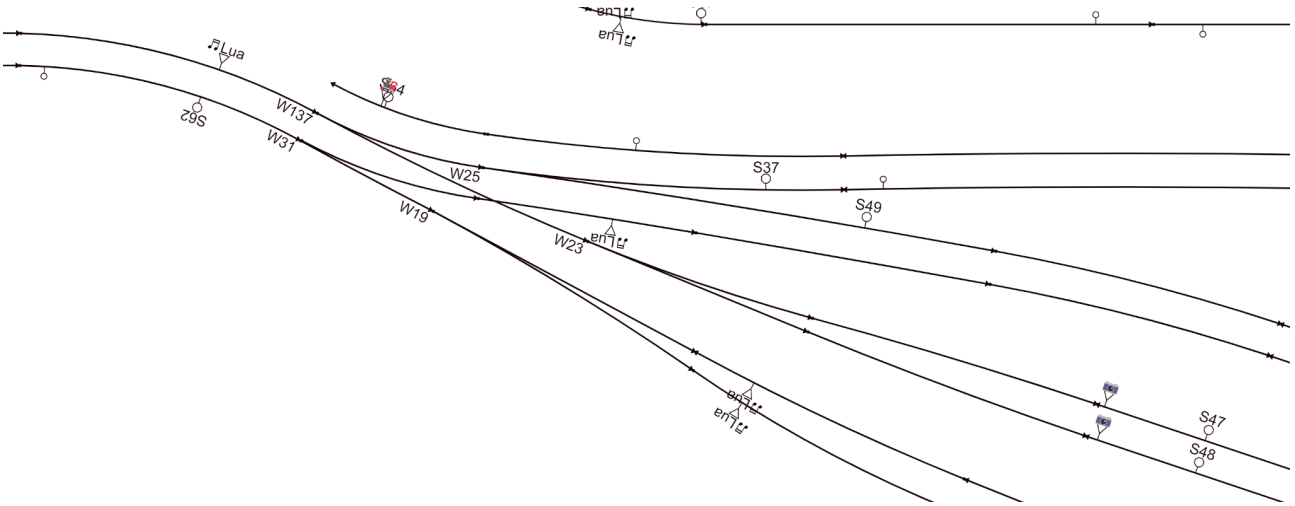
Da Kreuzungen keine Weichen sind, werden sie nicht in der Streckentabelle deklariert, d.h. sie werden nicht reserviert, wenn Lua die Strecke auswählt, wie es bei Blöcken und Weichen der Fall ist. Dies hat zur Folge, dass ohne Vorsichtsmaßnahmen zwei Züge gleichzeitig auf einer Kreuzung fahren können. Das macht keine Probleme, der automatische Verkehr funktioniert weiterhin, es sieht nur nicht gut aus.

Kollisionen auf einer Kreuzung können auf zwei Arten verhindert werden:

1. Suchen Sie die beiden Blöcke auf den Gleisen unmittelbar hinter der Kreuzung und deklarieren Sie sie als `two_way_blocks` (siehe Kapitel Demo EEP_Lua_Layout_03 wie man Zwei-Wege-Blöcke deklariert). Das Ergebnis ist, dass, sobald einer dieser Blöcke für eine Strecke reserviert ist, auch der andere Block reserviert wird. Kein Zug fährt dorthin, bis er freigegeben wird, wodurch die Kreuzung frei von anderem Verkehr bleibt.
2. Wenn die Strecke B, die die Kreuzung benutzt, mindestens eine Weiche enthält, dann kann diese Weiche in die Anmeldung der Strecke A aufgenommen werden. Es spielt keine Rolle, ob Sie die Weiche auf 1 oder 2 stellen, der einzige Grund für die Aufnahme ist, dass Sie sicherstellen wollen, dass diese Weiche reserviert wird, sobald die Strecke A aktiviert wird. Umgekehrt nehmen Sie eine Weiche der Strecke A in die Anmeldung der Strecke B auf. Das Ergebnis ist, dass kein Zug die Strecke B anfahren kann, solange die Strecke A eine Weiche reserviert hat, die zur Strecke B gehört, die Kreuzung ist für den Zug auf der Strecke A sicher.

Hier ist ein Beispiel für die Lösung des Kreuzungsproblems unter Verwendung der zweiten Option. Schauen wir uns zunächst das Eindringen von Zügen auf einer Kreuzung an:

Hier ist der entsprechende Teil des Gleisplans::



Der Weg vom Block 62 auf der linken Seite über die Weiche 31 zum Block 77 (nicht sichtbar weiter rechts) und die Wege auf der rechten Seite von den Blöcken 47 und 48 über die Weichen 23 und 137 zum Block 50 (nicht sichtbar weiter links) haben beide die gleiche Kreuzung.

Diese Kreuzung ist nicht Teil eines Blocks - sie liegt zwischen Blöcken - und dies ermöglicht eine einfache Lösung:

1. Erzeugen Sie die Routen

2. Erweitern Sie die Strecken manuell, indem Sie eine Weiche von der anderen Strecke hinzufügen, um diese zusätzliche Weiche zu sperren. Die Position für diese Weiche spielt keine Rolle, daher können Sie den Dummy-Wert 0 verwenden.

Dies führt zu folgenden geänderten Routen:

```
-- The generated routes are extended to avoid crashes at the crossing.
-- The extended routes include a turnout from the other route.
-- The turnout setting does not matter, therefore use dummy value 0.

local routes = {
  ...
--{ 47, 50, turn={ 23,2, 137,2, }},
  { 47, 50, turn={ 23,2, 137,2, 31,0, }}, -- Crossing
--{ 48, 50, turn={ 23,1, 137,2, }},
  { 48, 50, turn={ 23,1, 137,2, 31,0, }}, -- Crossing
  ...
--{ 62, 77, turn={ 31,1, }},
  { 62, 77, turn={ 31,1, 23,0, }}, -- Crossing
  ...
}
```

Optionen

Unten sehen Sie den Code, der auf den Konfigurationsteil folgt. Dieser Teil des Codes ist für jedes Layout gleich und kann oft unberührt bleiben, es sei denn, es werden Signale im Layout verwendet, die unterschiedliche Rot/Grün-Zustände haben, oder wenn Sie Optionen ändern möchten.

```
clearlog()
```
Dies löscht den Inhalt des EEP-Ereignisfensters nach dem Start und nach einem Neuladen des Lua-Codes. Sie können diese Zeile löschen oder auskommentieren, wenn Sie das Ereignisfenster nicht löschen wollen.

```
local blockControl = require("blockControl")
```
Damit wird die Datei `blockControl.lua` geladen, die den eigentlichen Code für die Erzeugung des automatischen Zugverkehrs darstellt. Sie wurde mit der Download-Zip-Datei geliefert und muss im Ordner `\LUA` der EEP-Installation abgelegt werden.

```
blockControl.init({
  trains       = trains,
  blockSignals = block_signals,
  twoWayBlocks = two_way_blocks,
  routes       = routes,
  paths        = anti_deadlock_paths,
  MAINSW       = main_signal,

  MAINON       = 1, -- ON    state of main switch
  MAINOFF      = 2, -- OFF   state of main switch
  BLKSIGRED    = 1, -- RED   state of block signals
  BLKSIGGRN    = 2, -- GREEN state of block signals
  TRAINSIGRED  = 1, -- RED   state of train signals
  TRAINSIGGRN  = 2, -- GREEN state of train signals
```

```
})
```

Die linke Spalte enthält die Parameter, wie sie im blockControl-Code verwendet werden; sie dürfen nicht verändert werden. Die rechte Spalte sind die Variablen, wie sie in Ihrem Layout-Konfigurationscode verwendet werden, Sie können beliebige Namen für sie verwenden.

Entsprechend dem Kapitel über Signalzustände können Sie sich über den Zustand der Signale informieren, die Sie auf Ihrer Anlage verwenden. Sie können die Signalzustände hier bei Bedarf ändern.

Darüber hinaus können Sie einige Laufzeitparameter jederzeit setzen (vor oder nach dem `init`-Aufruf oder zu bestimmten Zeiten innerhalb von `EEPMain`):

```
-- Optional: set these parameters to you personal liking
blockControl.set({
  logLevel        = 1,     -- 0:off, 1:normal, 2:more, 3:extreme
  showTippText    = true,  -- Show signal popups: true / false
  start           = false, -- Activate the main signal: true / false
  startAllTrains  = true,  -- Activate all train signals: true / false
})
```

Diese selbsterklärenden Parameter können Sie nach Ihren persönlichen Vorlieben einstellen. Der Parameter `logLevel` kann auch direkt im `init`-Aufruf angegeben werden.

```
-- Optional: Activate a control desk for the EEP layout
-- This only works as of EEP 16.1 patch 1
if EEPActivateCtrlDesk then
  local ok = EEPActivateCtrlDesk("Block control")
  if ok then print("Show control desk 'Block control'") end
end
```

Ab der EEP Version 16.1 Patch 1 ist `EEPActivateCtrlDesk` eine eingebaute Funktion. Alle 5 Demo-Layouts verfügen über ein Control Panel, über das der Zugverkehr gesteuert und überwacht werden kann. Dieser Code öffnet das Bedienpult standardmäßig. Wenn Sie das nicht wollen, löschen oder kommentieren Sie diesen Code einfach. Das Control Panel kann auch per Shift-Klick auf die kleine Konsole neben dem Hauptschalter geöffnet werden.

```
-- Optional: Use a counter signal to set the log level
local counterSignal = 47
EEPRegisterSignal( counterSignal )
_ENV["EEPOnSignal_"..counterSignal] = function(pos)
  local logLevel = pos - 1
  if logLevel > 3 then
    logLevel = 0
    blockControl.set({ logLevel = logLevel })
    EEPSetSignal( counterSignal, logLevel + 1 )
  else
    blockControl.set({ logLevel = logLevel })
  end
  print("Log level set to ", logLevel)
end
```

Im Layout kann ein Zählersignal platziert werden, über das der Protokollierungslevel manuell geändert werden kann, ohne das Lua-Editor-Fenster zu öffnen. Ein Beispiel wird auf `EEP_Lua_Layout_05` gezeigt.

```
function EEPMain()
  blockControl.run()
  return 1
```

```
end
```

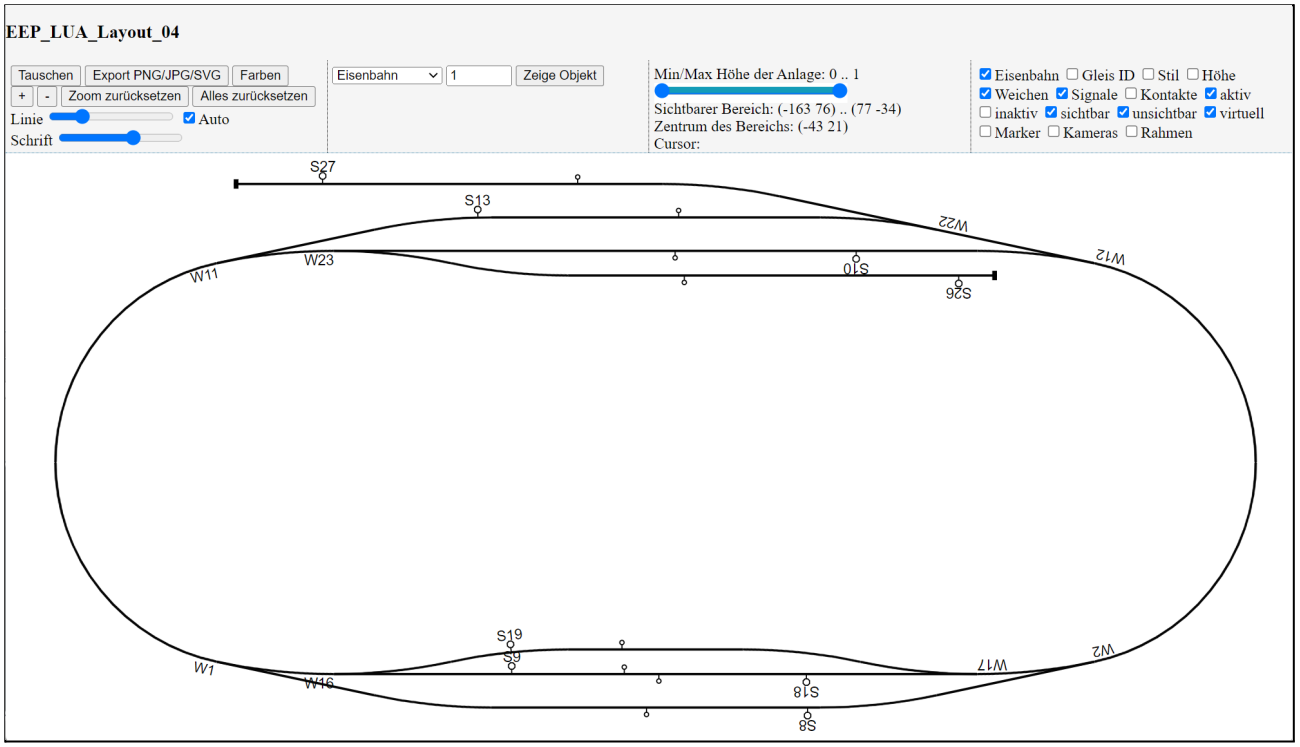
Die EEP-Hauptschleife. Die einzige Codezeile innerhalb der Hauptschleife, die nach EEP-Konvention fünfmal pro Sekunde abläuft, dient der Ausführung des blockControl-Codes.

Werkzeuge zur Erzeugung der Lua-Konfigurationstabellen

Die Tabelle `routes` im Lua-Konfigurationsabschnitt beschreibt die verfügbaren Strecken von einem Block zum nächsten, indem sie die Zustände "Haupt" / "Abzweig" der Weichen zwischen den beiden Blöcken definiert. Diese Tabelle kann von Hand erstellt werden, aber das erfordert große Aufmerksamkeit ... ein kleiner Fehler bei einem Weichenstatus kann dazu führen, dass ein Zug zu einem unerwarteten Block fährt, was zu unberechenbarem automatischen Verkehr führt.

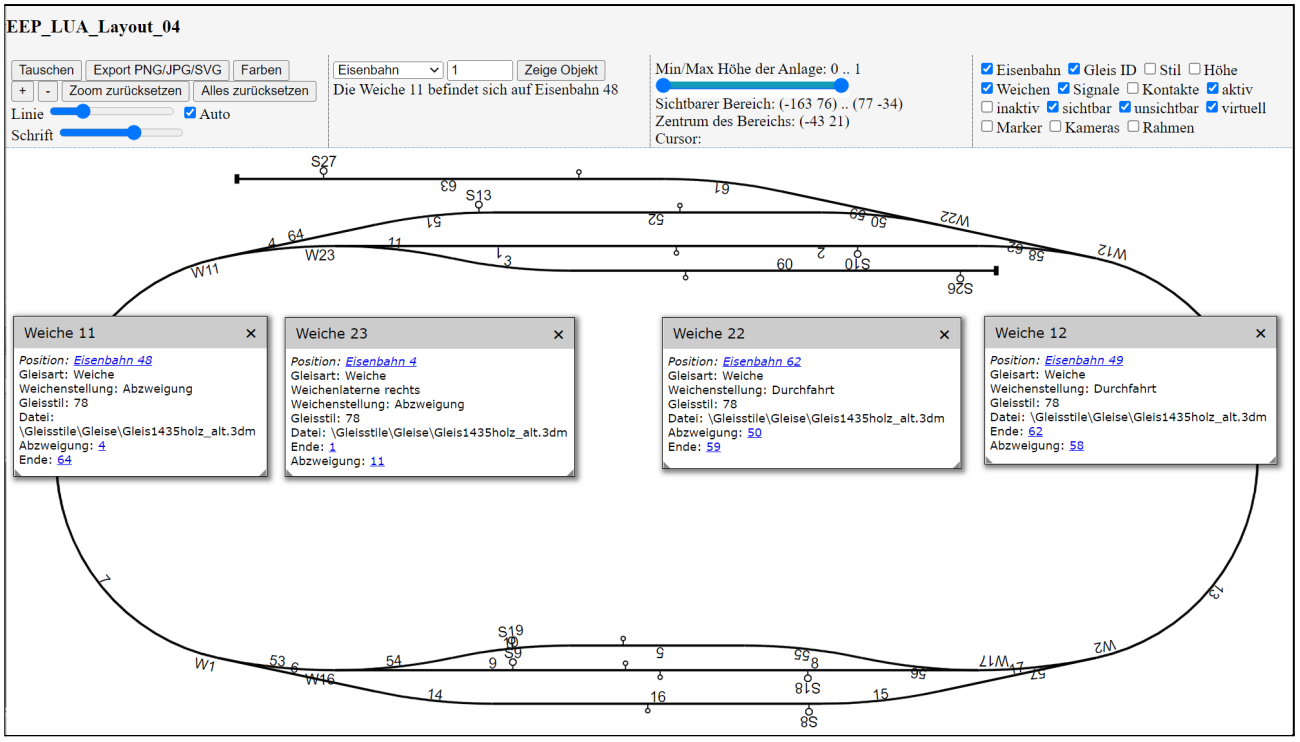
Zwei Hilfsmittel helfen hier weiter.

Als ersten Schritt können Sie die **Gleisplan-Program**[9] nutzen. Es zeigt den EEP-Gleisplan in Ihrem Browser-Fenster, einschließlich der Signale und Weichennummern, in einem leicht lesbaren Format.

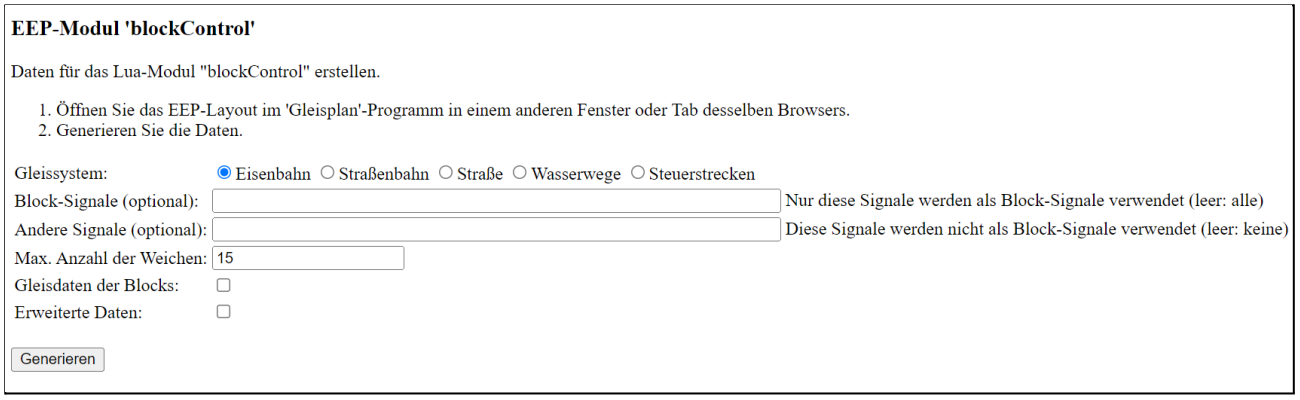


Bei EEP können die Zustände "Haupt" und "Abzweig" einer Weiche manchmal verwirrend sein. Was auf den ersten Blick wie eine "Hauptstrecke" aussieht, könnte in EEP als "Abzweig" bezeichnet werden. Was helfen kann, ist, zunächst alle Weichen auf der EEP-Anlage so zu schalten, dass der Zug in den Zustand fährt, der Ihrer Meinung nach als 'Haupt' bezeichnet werden sollte, und dann die Datei im **Gleisplan-Program**. Wenn Sie mit der linken Maustaste auf eine Weiche klicken, öffnet sich ein Info-Popup, das unter anderem den Zustand der Weiche anzeigt. Umschalttaste + Linksklick öffnet mehrere dieser Popup-Fenster.

[9] https://frankbuchholz.github.io/EEP_convert_anl3_file/EEP_Gleisplan.html



Ein zweites Werkzeug geht einen großen Schritt weiter: Es kann einen Vorschlag für alle Lua-Konfigurationstabellen mit Ausnahme der Pfade-Tabelle erstellen. Öffnen Sie zunächst den Gleisplan in der **Gleisplan-Programm**. Öffnen Sie nun das **blockControl-Programm**[10] in einer anderen Registerkarte desselben Browsers und klicken Sie auf die Schaltfläche "Generieren":



Einige Layouts enthalten zusätzliche Signale, die nicht durch das Modul gesteuert werden sollen. In einem solchen Fall können Sie entweder alle Blocksignalnummern eingeben, die gesteuert werden sollen, oder Sie können die Blocksignalnummern, die nicht gesteuert werden sollen, in das entsprechende Feld eingeben. Zwischen den Nummern können Sie ein beliebiges Trennzeichen verwenden.

Die Option zur Festlegung der maximalen Anzahl von Weichen pro Strecke ist nicht sehr wichtig. Sie verhindert hauptsächlich Endlosschleifen bei unvollständigen Layouts.

[10] https://frankbuchholz.github.io/EEP_convert_anl3_file/EEP_blockControl.html

Sie können die Option "Erweiterte Daten" aktivieren, die dem Ergebnis Informationen über Signalpositionen (sofern dem Programm bekannt) und besuchte Gleise bzw. Fahrwege hinzufügt.

Hier ist ein typisches Ergebnis:

EEP-Modul 'blockControl'

Daten für das Lua-Modul "blockControl" erstellen.

1. Öffnen Sie das EEP-Layout im 'Gleisplan'-Programm in einem anderen Fenster oder Tab desselben Browsers.
2. Generieren Sie die Daten.

Gleissystem: (•) Eisenbahn ( ) Straßenbahn ( ) Straße ( ) Wasserwege ( ) Steuerstrecken
Block-Signale (optional): Nur diese
Andere Signale (optional): Diese Sig
Max. Anzahl der Weichen: 15
Gleisdaten der Blocks: ☐
Erweiterte Daten: ☐

Generieren

```
-- EEP Datei 'EEP_LUA_Layout_04'
-- Lua-Programm für Modul 'blockControl.lua' für Gleisart 'Eisenbahn'

local main_signal = 3

local block_signals = { 8, 9, 10, 13, 18, 19, 26, 27, }

local two_way_blocks = { { 9, 18 }, }

-- Erlaubte Blöcke mit Wartezeit
local all = { [8]=1, [9]=1, [10]=1, [13]=1, [18]=1, [19]=1, [26]=1, [27]=1, }

local trains = {
 { name="#Blue",   signal=0, allowed=all },
 { name="#Orange", signal=0, allowed=all },
 { name="#Steam",  signal=0, allowed=all },
}

local routes = {
  { 8, 13, turn={ 2,1, 12,1, 22,2, }},
  { 8, 27, turn={ 2,1, 12,1, 22,1, }},
  { 9, 10, turn={ 16,1, 1,2, 11,2, 23,1, }},
  { 9, 26, turn={ 16,1, 1,2, 11,2, 23,2, }},
  { 10, 9, turn={ 12,2, 2,2, 17,1, }},
  { 10, 19, turn={ 12,2, 2,2, 17,2, }},
  { 13, 8, turn={ 11,1, 1,1, }},
  { 13, 18, turn={ 11,1, 1,2, 16,1, }},
  { 18, 13, turn={ 17,1, 2,2, 12,1, 22,2, }},
  { 18, 27, turn={ 17,1, 2,2, 12,1, 22,1, }},
  { 19, 10, turn={ 16,2, 1,2, 11,2, 23,1, }},
  { 19, 26, turn={ 16,2, 1,2, 11,2, 23,2, }},
  { 26, 8, turn={ 23,2, 11,2, 1,1, }},
  { 26, 18, turn={ 23,2, 11,2, 1,2, 16,1, }},
  { 27, 9, turn={ 22,1, 12,1, 2,2, 17,1, }},
  { 27, 19, turn={ 22,1, 12,1, 2,2, 17,2, }},
}
```

Sie können Signal- und Weichennummern anklicken, um die Ansicht im Gleisplan-Programm auf das ausgewählte Objekt zu positionieren.

Nun können Sie das Ergebnis per Copy & Paste in Ihre Lua-Datei einfügen und nach Ihren Wünschen anpassen.

Die Variable `start_signal` zeigt die Nummer des Hauptsignals ("Switch_standing", "Switch_lying", oder "StartSwitch_usertex") mit der kleinsten Nummer wenn solch ein Signal auf der Anlage existiert. Ansonsten wird die Nummer 0 gezeigt und sie können diesen Werte selber anpassen.

Die Tabellen `block_signals` und `two_way_blocks` enthalten die komplette Liste der Blöcke.

```
local block_signals = { 8,  9,  10,  13,  18,  19,  26,  27, }

local two_way_blocks = { { 9, 18 }, }
```

Sie erhalten dann eine oder mehrere Tabellen mit einer vollständigen Liste von Blöcken mit der Standardwartezeit 1 für jede EEP-Route, die einigen Zügen zugewiesen ist. Wenn keine EEP-Routen verwendet werden, erhalten Sie eine Tabelle mit dem Namen "all". Sie können diese Vorschläge verwenden, um die Wartezeit anzupassen und Blöcke zu entfernen, die für bestimmte Züge nicht zugelassen werden sollen.

```
-- Allowed blocks with wait time
local all = { [8]=1, [9]=1, [10]=1, [13]=1, [18]=1, [19]=1, [26]=1,
[27]=1, }
```

Die Tabelle `trains` zeigt alle Züge auf der Anlage und in Depots mit zugewiesenen `allowed` Tabellen, aber ohne eine bestimmte Zugsignalnummer.

```
local trains = {
 { name="#Blue",   signal=0, allowed=all },
 { name="#Orange", signal=0, allowed=all },
 { name="#Steam",  signal=0, allowed=all },
}
```

Schließlich erhalten Sie die Tabellen `routes` mit korrekten Weichenstellungen. Manchmal werden Sie von den Ergebnissen überrascht sein, vor allem wenn es mehrere Strecken zwischen Blöcken gibt. Sie können solche überflüssigen Strecken entweder entfernen (oder kommentieren) oder weitere Blöcke in den Plan einfügen. Andererseits könnten Sie einige Strecken übersehen, insbesondere wenn Sie mit Zwei-Wege-Blöcken arbeiten. Beachten Sie, dass benachbarte Gleise zwischen Weichen entweder zu keinem Block (kein Signal), zu einem Einbahnblock (ein Signal) oder zu einem Zweibahnblock (zwei Signale für entgegengesetzte Richtungen) gehören.

```
local routes = {
  { 8, 13, turn={ 2,1, 12,1, 22,2, }},
  { 8, 27, turn={ 2,1, 12,1, 22,1, }},
  { 9, 10, turn={ 16,1, 1,2, 11,2, 23,1, }},
  { 9, 26, turn={ 16,1, 1,2, 11,2, 23,2, }},
  { 10, 9, turn={ 12,2, 2,2, 17,1, }},
  { 10, 19, turn={ 12,2, 2,2, 17,2, }},
  { 13, 8, turn={ 11,1, 1,1, }},
  { 13, 18, turn={ 11,1, 1,2, 16,1, }},
  { 18, 13, turn={ 17,1, 2,2, 12,1, 22,2, }},
  { 18, 27, turn={ 17,1, 2,2, 12,1, 22,1, }},
```

```
  { 19, 10, turn={ 16,2, 1,2, 11,2, 23,1, }},
  { 19, 26, turn={ 16,2, 1,2, 11,2, 23,2, }},
  { 26, 8, turn={ 23,2, 11,2, 1,1, }},
  { 26, 18, turn={ 23,2, 11,2, 1,2, 16,1, }},
  { 27, 9, turn={ 22,1, 12,1, 2,2, 17,1, }},
  { 27, 19, turn={ 22,1, 12,1, 2,2, 17,2, }},
}
```

Beschränkungen:

- Sie erhalten keine Tabelle `paths`, um mögliche Stillstände zu lösen. Wenn Sie auf Stillstände stoßen, müssen Sie Ihre eigene Lösung entwickeln, indem Sie die oben genannten Tipps verwenden.
- Doppelkreuzungsweichen werden nicht unterstützt.
- Müll rein - Müll raus: Wenn das Layout nicht über geeignete Blocksignale verfügt, können Sie keine vernünftigen Ergebnisse erwarten.

Schließlich können Sie das **Inventar-Programm**[11] nutzen, um die Einstellungen der Kontakte einschließlich der Lua-Funktion in den Kontakten zu überprüfen:

Vielleicht möchten Sie einige Spalten ausblenden, um die Ansicht zu optimieren, oder Sie möchten filtern, indem Sie `[nonempty]` (einschließlich der Klammern) in das Filterfeld für die Spalte "Lua-Funktion" eingeben:

Kontakte: 10 Spalten verstecken ▼ ?

Filter ...						
Kontakttypcode	**Kontakttyp**	**Gleis**	**Auslösung**	**Zugposition**	**Lua-Funktion**	**Tipp-Text**
5	Fahrzeug -0 km/h max	Eisenbahn 60	in Gleisrichtung	Spitze		Block 26 reverse direction
5	Fahrzeug -0 km/h max	Eisenbahn 63	in Gleisrichtung	Spitze		Block 27 reverse direction
256	Sound undefined	Eisenbahn 1	gegen Gleisrichtung	Schluss	blockControl.enterBlock_10	Block 10
256	Sound undefined	Eisenbahn 3	in Gleisrichtung	Schluss	blockControl.enterBlock_26	Block 26
256	Sound undefined	Eisenbahn 6	in Gleisrichtung	Schluss	blockControl.enterBlock_8	Block 8
256	Sound undefined	Eisenbahn 8	gegen Gleisrichtung	Schluss	blockControl.enterBlock_9	Block 9
256	Sound undefined	Eisenbahn 9	in Gleisrichtung	Schluss	blockControl.enterBlock_18	Block 18
256	Sound undefined	Eisenbahn 52	in Gleisrichtung	Schluss	blockControl.enterBlock_13	Block 13
256	Sound undefined	Eisenbahn 55	in Gleisrichtung	Schluss	blockControl.enterBlock_19	Block 19
256	Sound undefined	Eisenbahn 61	in Gleisrichtung	Schluss	blockControl.enterBlock_27	Block 27

Anhalten der Fahrt und Speichern des aktuellen Zustands

Vor dem Verlassen von EEP ist es sinnvoll, den Hauptschalter auszuschalten und zu warten, bis alle Züge an einem Signal zum Stillstand gekommen sind. Dank des "FIND-MODE" kann jedoch nach dem Laden einer Layout-Datei oder nach dem Nachladen des Lua-Codes das Layout jederzeit gespeichert werden, in der nächsten Sitzung wird Lua die Züge wieder finden. Es ist nicht nötig, das Layout über das Menü zu speichern, der aktuelle Zustand wird automatisch gespeichert, wenn EEP verlassen wird.

[11] https://frankbuchholz.github.io/EEP_convert_anl3_file/EEP_Inventar.html

Wie man das Modul "BetterContacts" zur Vereinfachung der Konfiguration verwendet

Der EEP-Editor für Kontakte hat die Einschränkung, dass Sie nur forhandene Lua-Funktionen eingeben können. Daher müssen Sie in einer bestimmten Reihenfolge arbeiten:

1. Platzieren Sie Blocksignale und optional die Kontakte (aber ohne Verweis auf eine Lua-Funktion in diesen Kontakten - das Feld muss leer sein)
2. Anpassung des Lua-Codes, um alle Blocksignale in einigen der Tabellen aufzulisten
3. Führen Sie das Layout im 3D-Modus aus
4. Platzieren Sie die Kontakte, falls noch nicht geschehen
5. Geben Sie den Verweis auf die Lua-Funktion für die Blocksignale in die Kontakte ein

Jedes Mal, wenn Sie weitere Blocksignale hinzufügen, müssen Sie diese Reihenfolge für diese neuen Blöcke erneut einhalten.

Sie können diese Einschränkung umgehen, indem Sie das Modul "BetterContacts" von Benny verwenden, das Sie hier erhalten können: https://emaps-eep.de/lua/bettercontacts

Legen Sie das Modul in den LUA-Ordner und bereiten Sie Ihr Layout für die Verwendung der beiden Module "BetterContacts" und "blockControl" vor, indem Sie die folgenden Zeilen am Anfang des Lua-Codes hinzufügen (in diesem Fall ist es notwendig, eine globale Variable `blockControl` zu verwenden, daher deklarieren Sie sie nicht als lokal)[12]:

```
require("BetterContacts_BH2")
blockControl = require("blockControl")
```

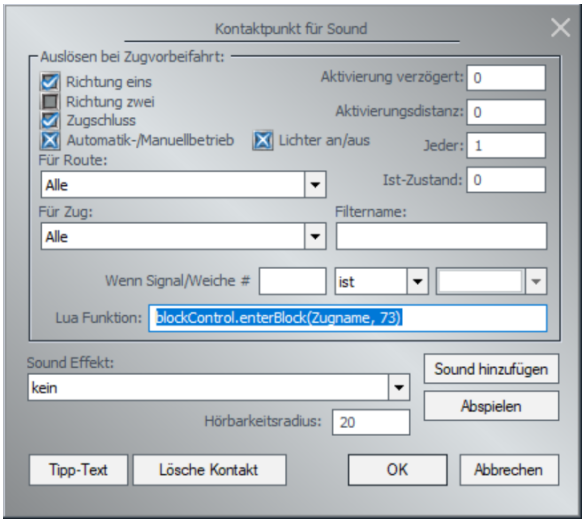
Führen Sie das Layout einmal aus, um Lua über diese Module zu informieren.

Jetzt können Sie Signale und entsprechende Kontakte einschließlich der Lua-Funktionsaufrufe ohne besondere Reihenfolge platzieren. Verwenden Sie die parametrisierte Form der Lua-Funktion (ersetzen Sie N durch die Nummer des Blocksignals)[13]:

```
blockControl.enterBlock(Zugname, N)
```

Hier ist ein Beispiel:

[12] Zusätzlich können Sie die Option `BetterContacts = true` im `init`-Aufruf des Moduls blockControl verwenden, um die Erzeugung globaler Funktionen für dieses Modul zu verhindern.
[13] Das Modul "BetterContacts" bietet eine zusätzliche Möglichkeit, den Variablennamen `Zugname` zu ändern, wie in der Dokumentation des Moduls beschrieben.



Überblick, wie Lua automatischen Zugverkehr erzeugt

Wenn ein Zug in einen Block einfährt, überfährt er den Einfahrsensor, der über die Funktion `enterBlock`, die Sie in das Lua-Feld der Kontakte auf dem EEP-Layout eingetragen haben, die folgende Abfolge von Ereignissen auslöst.

Die Funktion speichert lediglich das Ereignis. Der nächste Aufruf der Funktion `blockControl.run` in `EEPMain` führt alle folgenden Schritte aus:

1. Verringern der verbleibenden Wartezeit für alle Züge innerhalb von Blöcken.

2. Prüfen, ob ein Zug in einen Block eingefahren ist. Wenn dies der Fall ist:
    a. Wird der Block für den Zug registriert
    b. Wird die Wartezeit für den Zug innerhalb dieses Blocks festgelegt
    c. Wird der vorherigen Block freigegeben und das Signal des vorherigen Blocks wird auf "rot" gesetzt.
    d. Wird der entsprechenden Zwei-Wege-Block freigegeben
    e. Werden die Weichen auf der Stecke hinter dem Zug freigegeben
    f. Der Zug fährt weiter wenn das Ende der aktuellen Stecke noch nicht erreicht ist, andernfalls wird ein Steckenantrag gestellt.

3. Wenn der Hauptschalter eingeschaltet ist: Erstellen einer Liste mit allen möglichen Stecken für alle Züge
    a. deren Zugsignal (wenn vorhanden) "grün" ist und
    b. die einen Streckenantrag haben und
    c. deren Wartezeit abgelaufen ist.

4. Um diese Liste zu erstellen, werden ausgehend vom aktuellen Block Stecken gesucht, für die
    a. alle Durchgangsblöcke (falls vorhanden) und der Zielblock für den Zug erlaubt und frei sind und
    b. für die alle Weichen auf der Stecke frei sind.

5. Nun wird nach dem Zufallsprinzip eine einzelne Strecke aus dieser Liste ausgewählt und der entsprechenden Zug wird auf dieser Strecke fahren gelassen:

a. Alle Blöcke und Weichen auf der Strecke werden gesperrt.
b. Alle Blocksignale (außer dem letzten) auf der Strecke werden auf "grün" geschaltet.
c. Alle Weichen des Weges werde entsprechend den definierten Routen geschaltet.

Tabellenparameter zur Definition des Layouts, die vom Benutzer konfiguriert werden

`trains = {}`
`trains[1] = {name="Passenger", signal=14, allowed={...}}`
`trains[2] = {name="Cargo", signal=15, allowed={...}}`

- **key**. Sie können jeden beliebigen Schlüssel verwenden, um Einträge in der Tabelle zu identifizieren (hier: 1 und 2).
- ***`name="Passenger"`***. Die Zugnamen (mit oder ohne führendes "#") müssen mit den in der EEP angezeigten Zugnamen übereinstimmen.
- ***`signal=14`***. (optional) Die Nummer des Signals, das als Ein/Aus-Schalter für diesen Zug verwendet wird.
- ***`allowed={...}`***. (optional) Die Blocknummern und Wartezeiten, in denen dieser Zug fahren darf (Einzelheiten siehe unten).

Immer wenn Sie eine Layout-Datei öffnen oder den Lua-Code neu laden, weiß das Programm nicht, wo sich die Züge befinden und startet den *Suchmodus für Züge*. Während dieses Modus werden die Züge aus dieser Tabelle identifiziert sowie jeder andere Zug, der in einen Block einfährt oder dort bleibt.

`allowed = {}`
`allowed[1] = 1`
`allowed[5] = 23`

Diese Tabelle gibt an, ob ein Zug in einem Block erlaubt ist und ob er eine Haltezeit hat:
- nicht definiert oder 0: dieser Zug ist in diesem Block nicht erlaubt
- 1: dieser Zug ist in diesem Block erlaubt, keine Haltezeit
- >1: Dieser Zug ist in diesem Block zugelassen und hat eine Haltezeit. Die Zeit, die der Zug benötigt, um vom Blockeingangssensor zum Blocksignal zu fahren, muss mit eingerechnet werden. Beispiel: Wenn die gemessene Fahrzeit vom Sensor zum Signal dieses Zuges in diesem Block 15 Sekunden beträgt und eine Haltezeit von 8 Sekunden gewünscht wird, ist die einzugebende Zahl 15+8=23.

Im obigen Beispiel:
- Der Zug ist in Block 1 zugelassen. Es ist keine Wartezeit angegeben.
- Der Zug ist in Block 5 zugelassen. Wenn die Fahrzeit vom Sensor zum Signal etwa 15 Sekunden beträgt, hat er eine Haltezeit von etwa 8 Sekunden vor dem Blocksignal (unter der Annahme, dass die Fahrzeit vom Sensor zum Signal in beiden Blöcken 15 Sekunden beträgt).

`blockSignals = {14,18,16,…}`

(optional)
Jeder Block hat ein Signal, an dem die Züge anhalten, wenn es rot ist. Das Signal wird von Lua gesteuert. Es wird auf grün geschaltet, wenn der Zug eine neue Strecke zu einem benachbarten Block beginnen darf. Es wird wieder auf rot geschaltet, wenn der Zug im nächsten Block ankommt.

Wenn ein Block in zwei Richtungen verwendet wird, wird er als zwei Blöcke in entgegengesetzten Richtungen behandelt, jeder mit seinem eigenen Blocksignal. Dies wird durch den Parameter `twoWayBlock` angegeben.

WICHTIG: In EEP haben nicht alle Signale den gleichen Status für "rot" und "grün". Bei einigen Signalen bedeutet 1 "geschlossen", 2 "offen", bei anderen ist es umgekehrt. Verwenden Sie daher nur Blocksignale, die für das gesamte Layout den gleichen "Rot"/"Grün"-Zustand haben. Die korrekten Zustände der verwendeten Blocksignale werden mit Variablen definiert, siehe das Kapitel 'Variablen ...' weiter unten.

twoWayBlocks = { {4, 3} }

In diesem Beispiel sind die Blöcke 3 und 4 "Zwillinge" von Zwei-Wege-Blöcken. Wenn ein Zug Block 3 für seine neue Strecke reserviert, liest Lua in Tabelle `two_way_blocks`, dass er auch Block 4 reservieren muss. Umgekehrt, wenn ein Zug Block 4 reserviert, liest Lua in Parameter `twoWayBlocks`, dass er auch Block 3 reservieren muss.

routes = {}

```
route[1]={2,3, turn={ }}
route[2]={3,8, turn={ 7,1,  12,2 } -- 1:main, 2:branch, 3:alternate branch
```

Es werden Strecken von einem Block zum nächsten angegeben. Die Tabelle enthält den Abfahrtsblock, den Zielblock und eventuell eine oder mehrere Weichen, die geschaltet werden müssen. Im obigen Beispiel:

- für die Strecke 1 ist 2 der Abfahrtsblock, 3 der Zielblock, und es sind keine Weichen zu schalten.
- für die Strecke 2 ist 3 der Abfahrtsblock, 8 ist der Zielblock, die Weiche 7 muss auf "Haupt" und die Weiche 12 auf "Abzweig" geschaltet werden.

Manchmal können solche einfachen Strecken zu einer Sperrung führen. Ein einfaches, aber typisches Beispiel wären 2 parallele Gleise in einem Bahnhof, 3 parallele Gleise im anderen Bahnhof und ein eingleisiger Weg dazwischen.

```
===T1==1===\                /===5==T3===
===T2==2====W1=====3=====W2===4=======
                            \===6=======
```

Sie können auf dieser kleinen Anlage problemlos zwei Züge fahren lassen, aber wenn Sie einen dritten Zug hinzufügen, können Sie in Schwierigkeiten geraten, wenn beide Gleise des kleinen Bahnhofs von wartenden Zügen belegt sind und der dritte Zug eine Strecke findet, die vom größeren Bahnhof ausgeht: Er würde in das dazwischen liegende eingleisige Gleis einfahren und dann kann kein Zug mehr fahren.

Um diese Art von Problem zu lösen, können Sie die kritischen Strecken in bestimmten Pfaden abdecken:

paths = {}

```
paths[1]= { {4,5,6}, 3, {1,2} }
```

Sie definieren Pfade, die von bestimmten Blöcken ausgehen, über einen oder mehrere Blöcke führen und in bestimmten Blöcken enden. Sie schließen parallele Blöcke in Klammern ein.

Das obige Beispiel zeigt eine typische Situation mit 2 parallelen Gleisen in einem Bahnhof, 3 parallelen Gleisen im anderen Bahnhof und einem eingleisigen Gleis dazwischen.

Sie können Pfade auch in erweiterter Form definieren. Typische Situationen wie oben erfordern jedoch mehrere primitive Pfade. Für das obige Beispiel würde dies zu dieser erweiterten Definition führen:

```
paths = { {4,3,1}, {4,3,2}, {5,3,1}, {5,3,2}, {6,3,1} {6,3,2}, }
```

Vom Benutzer konfigurierte Parameter

`MAINSW = 27`
Die Adresse des Signals, das als Hauptschalter verwendet wird. Wenn der Hauptschalter eingeschaltet ist, können die Züge fahren. Darüber hinaus kann jeder Zug über einen individuellen Ein-/Ausschalter verfügen. Wenn ein Zugschalter oder der Hauptschalter ausgeschaltet wird, fahren die Züge zunächst auf ihrer aktuellen Strecke weiter, bis sie ihren Zielblock erreichen, wo sie durch ein rotes Signal angehalten werden. Von dort aus fahren sie erst wieder auf eine neue Strecke, wenn sie wieder eingeschaltet werden.

`MAINON = 1, MAINOFF = 2, or vice versa`
Die Werte können entweder 1 oder 2 sein, je nachdem, welche Art von Signal als Hauptschalter verwendet wird. Der Benutzer muss dies im EEP überprüfen.

`BLKSIGRED = 1, BLKSIGGRN = 2, or vice versa`
Der im EEP verwendete Wert für ein rotes und grünes Blocksignal, der je nach Art der verwendeten Blocksignale 1 oder 2 sein kann. Der Benutzer muss dies in EEP überprüfen. Achten Sie darauf, nur Blocksignale zu verwenden, die im gesamten Layout denselben Status haben.

`TRAINSIGRED = 1, TRAINSIGGRN = 2, or vice versa`
Der in EEP verwendete Wert für ein rotes und grünes Zugsignal, der je nach Art der verwendeten Zugsignale 1 oder 2 sein kann. Der Benutzer muss dies in EEP überprüfen. Achten Sie darauf, dass Sie nur Zugsignale verwenden, die auf der gesamten Anlage den gleichen Status haben.

Fehlersuche

Allgemeine Tipps

- Im *Suchmodus für Züge* sollten alle Blocksignale "rot" zeigen und alle Züge sollten an diesen Signalen halten. Während des Suchmodus sehen Sie die Signalposition auch im Tipptext. Verwenden Sie dies, um zu überprüfen, ob dieser Wert mit dem Parameter `BLKSIGRED` übereinstimmt.

- Parameter `blockSignals`
  Verwenden Sie diesen optionalen Parameter zu Dokumentationszwecken und um zusätzliche Konsistenzprüfungen auszulösen:
  - Sind alle in den Parametern `allowed`, twoWayBlocks, `routes` und `paths` verwendeten Blöcke vorhanden?

    - Haben alle Blöcke in `blockSignals` einen Start- und Endblock in `routes`?

- Option `showTippText`
  Sie können die Tipptexte bei Signalen aktivieren/deaktivieren, indem Sie den Hauptschalter zweimal umlegen oder über die Funktion `set` einstellen.

- Option *`logLevel`*
  Sie können diese Option über die Funktion init einstellen oder festlegen, dass weniger oder mehr Ereignisse im EEP-Protokoll angezeigt werden:
    - 0: kein Protokoll
    - 1: normal
    - 2: vollständig
    - 3: extrem

Wie man den Status von Signalen und Weichen anzeigt

Mit dem folgenden Lua-Programm kann man den Status von Signalen und Weichen in den Tipp-Texten anzeigen:

```
local function showStatus()
  for i = 1, 1000 do
    -- Show signal status
    local pos = EEPGetSignal( i )
    if pos > 0 then
      local trainName = EEPGetSignalTrainName( i,  1)
      EEPChangeInfoSignal( i,
        string.format("<c>Signal %d = %d<br>%s", i, pos, trainName) )
      EEPShowInfoSignal( i, true )
    end

    -- Show turnout status
    local pos = EEPGetSwitch( i )
    if pos > 0 then
      EEPChangeInfoSwitch( i, string.format("Switch %d = %d", i, pos) )
      EEPShowInfoSwitch( i, true )
    end
  end
end

function EEPMain()
  showStatus()
  return 1
end
```

Technische Details zum Modul

Das Blocksteuerungsprogramm ist als Modul implementiert, das eine eigene Schnittstelle hat und keine globalen Namensräume belegt (mit Ausnahme von generierten Lua-Funktionen, die Sie in Gleiskontakte auf dem Layout eingeben). Das Modul verwendet keine Daten-Slots oder Tag-Texte.

Das Modul enthält folgende Teile:

- Initialisierung (einmalig)
  - Konsistenzprüfung
  - Benutzerdefinierte Daten in interne Tabellen kopieren
- Laufzeitparameter setzen (jederzeit)
- Suchmodus für Züge (einmalig)
- Betriebsmodus (mit Aufruf in EEPMain)

Initialisierung (einmalig)

Die Initialisierung des Moduls ist in der Funktion `init` implementiert. Diese Funktion führt einige Konsistenzprüfungen an den Benutzerdaten durch, kopiert die Benutzerdaten in die internen Tabellen und aktiviert den Suchmodus.

Konsistenzprüfungen

Es ist optional, Daten für den Parameter `blockSignals` anzugeben (eine ungeordnete Liste von Blocksignalnummern, die beschreiben, welche Signale unter der Kontrolle des Moduls stehen). Wenn Sie diesen Parameter angeben, erhalten Sie zusätzliche Konsistenzprüfungen als Teil des `init`-Aufrufs:

- Sind alle in den Parametern `allowed`, `twoWayBlocks`, `routes` und `paths` verwendeten Blöcke vorhanden?
- Haben alle Blöcke in `blockSignals` einen Start- und Endblock in `routes`?

Benutzerdefinierte Parameter in interne Tabellen kopieren

Die Benutzerdaten werden nicht in ihrer ursprünglichen Form verwendet, sondern transformiert und in interne Tabellen kopiert. Diese zusätzliche Transformation entkoppelt das Format der Benutzereingabe von der internen Darstellung und bereitet Performance-Optimierungen vor. Beispiel: ein komplexer Pfad wie

```
{ {1,2}, 3, {4,5} }
```

in Parameter `paths`, in einer kompakte Schreibweise, die dennoch für Benutzer leicht verständlich ist, wird in einen Satz von vier einfachen Pfaden umgewandelt, wie z.B.

```
{ {1,3 4}, {1,3,5}, {2,3,4}, {2,3,5} }
```

die einen optimierten Zugriff für das Programm ermöglichen.

Laufzeitparameter setzen (jederzeit)

`logLevel`: Das Modul verwendet zwei Hilfsfunktionen, um Meldungen im EEP-Protokollfenster anzuzeigen. Die Funktion `printLog` prüft die aktuelle Protokollstufe, die mit dem Parameter `logLevel` definiert ist, und ruft die ursprüngliche Druckfunktion auf, die alle anderen Parameter an diese Funktion übergibt. Die Funktion `check` benutzt nicht den Parameter `logLevel`, sondern verwendet einen ähnlichen Ansatz, um eine Bedingung zu prüfen und die Meldung anzuzeigen, wenn die Bedingung nicht erfüllt ist. Dies ist ähnlich wie bei der Lua-Funktion `assert`, jedoch ohne den Programmfluss zu unterbrechen.

showTippText: Die Funktion `showSignalStatus` wird in jedem Zyklus der Funktion `EEPMain` aufgerufen. Der boolesche Wert des Parameters `showTippText` wird direkt zum Aktivieren/Deaktivieren der Tipp-Texte verwendet.

start: Sie können diesen Parameter verwenden, um das Hauptsignal zu starten (`true`) oder zu stoppen (`false`) (anstatt `EEPSetSignal` selbst aufzurufen). Im *Suchmodus für Züge* wird der Start verzögert, bis alle Züge identifiziert und einem Block zugewiesen sind. Daher können Sie diese Option auch während des Suchmodus setzen, ohne das Programm zu stören.

startTrains: Verwenden Sie diesen Parameter, um alle Zugsignale zu aktivieren oder zu deaktivieren (anstatt für alle Zugsignale `EEPSetSignal` selbst aufzurufen).

Suchmodus für Züge (einmalig)

Der Zweck des *Suchmodus für Züge* ist es, alle auf der Anlage fahrenden Züge zu finden und die Blöcke zu identifizieren, in denen sich diese Züge befinden. Durch diesen Modus werden keine Datenslots benötigt, um den internen Status von EEP zwischen dem Schließen und dem erneuten Öffnen einer Anlage zu verfolgen. Sie können die Anlage jederzeit speichern und laden.

Der *Suchmodus für Züge* setzt zunächst das Hauptsignal und alle Blocksignale auf "rot". Die Züge befinden sich entweder bereits in der Reichweite eines Blocksignals oder fahren auf den nächsten Block zu. In jedem Fall werden diese Züge erkannt und halten an dem Blocksignal.

Die Hinweistexte auf den Signalen sind aktiviert und zeigen den Status der Blöcke an. (Zusätzlich sehen Sie auch die Signalposition, um zu überprüfen, ob alle Signale "rot" zeigen).

Um den Suchmodus zu beenden, haben Sie zwei Möglichkeiten:

a) manuell: Überprüfen Sie einfach die Anlage, ob alle Züge an einem Blocksignal zum Stehen gekommen sind. Aktivieren Sie dann das Hauptsignal.
b) automatisch: Wenn Sie eine vollständige Tabelle Züge mit allen Zügen bereitstellen, können Sie den automatischen Start über den Laufzeitparameter `start = true` anfordern, den Sie mit der Funktion `set` setzen können. Sobald der *Suchmodus für Züge* alle Züge identifiziert hat, startet er den Betriebsmodu.

Betriebsmodus (mit Aufruf über EEPMain)

Die Sensoren (Kontakte) informieren das Modul über Züge, die in einen Block einfahren, indem sie pro Block eine generierte Funktion aufrufen.

Die Funktion `run` ist das Herzstück des Moduls. Die Funktion

- identifiziert Züge, die Blöcke verlassen haben, und gibt diese Blöcke frei (optional, über zusätzliche Kontakte),
- identifiziert Züge, die in Blöcke eingefahren sind, und initialisiert die Wartezeit für diese Züge,
- dekrementiert die Wartezeit für alle Züge,
- sucht nach verfügbaren Strecken, sobald die Wartezeit abgelaufen ist,
- wählt eine der verfügbaren Strecken nach dem Zufallsprinzip aus,
- gibt Blöcke und Weichen hinter den Zügen frei,
- sperrt Blöcke und Weichen auf den Strecke der Züge,
- und setzt schließlich die Blocksignale, um die Züge fahren zu lassen.

Intern vom Lua-Modul verwendete Daten

Tabelle `TrainTab`

Zusätzlich zu den Komponenten `name`, `signal` und `allowed` sehen wir folgende Komponenten in dieser Tabelle:

`block`
Aktueller Block, in dem sich der Zug befindet (oder `nil`)

`path`
Aktuelle restliche Strecke, auf der der Zug fährt (oder `nil`). Wenn ein Zug eine neue Strecke erhält, wird sie in diese Komponente kopiert. Immer wenn ein Zug einen nächsten Block auf seiner aktuellen Strecke erreicht, wird dieser Block aus der Strecke entfernt.

Tabelle `pathTab`

Diese Tabelle enthält die erweiterten Pfade, die für das Routing zur Verfügung stehen.

Tabelle `routeTab`

Diese Tabelle enthält die Weicheneinstellungen für die Strecke zwischen zwei benachbarten Blöcken.

Tabelle `BlockTab`

Zusätzlich zu den Komponenten `signal` und `twoWayBlock` sehen wir folgende Komponenten in dieser Tabelle:

`reserved`
Diese Komponente enthält `nil`, wenn ein Block frei ist. Wenn ein Block reserviert ist, enthält sie den Zug, der ihn reserviert. Diese Blöcke sind für neue Strecken nicht verfügbar. Wenn ein Zug eine Strecke reserviert, reserviert er nicht nur die Durchgangsblöcke (falls vorhanden) und den Zielblock, sondern auch die Weichen zwischen den Blöcken, um zu verhindern, dass andere Züge sie benutzen. Wenn ein Zug am nächsten Block seiner Strecke ankommt, werden der Block und die Weichen hinter dem Zug, die für diese Strecke reserviert waren, freigegeben.

`occupied / occupiedOld`
Diese Komponenten werden verwendet, um Ereignisse für ankommende Züge in Blöcken aufzufangen.

`request`
Wenn ein Zug bereit ist, eine neue Strecke aufzunehmen, wird er hier registriert.

`stoptimer`
Die Haltezeiten pro Zug und Block sind in der Tabelle der zulässigen Zeiten angegeben. Bei Ankunft wird `stoptimer` mit 5x der definierten Haltezeit initialisiert (weil Lua 5x pro Sekunde zykliert) und die Haltezeiten werden bei jedem Zyklus dekrementiert. Wenn `stoptimer` den Wert

0 erreicht, ist die Haltezeit für diesen Zug in diesem Block abgelaufen und der Zug wird für eine neue Strecke verfügbar.

`availablePath`
Diese Tabelle wird in jedem Lua-Zyklus geleert und dann neu aufgebaut. Sie enthält die verfügbaren Strecken für alle Züge, die neue Strecken angefordert haben. Lua wählt zufällig eine Strecke aus der Liste aus.
Dieser Zug ist der einzige, der in diesem Lua-Zyklus eine neue Strecke einrichten darf. Lua reserviert nun die Durchgangsblöcke (falls vorhanden) und den Zielblock sowie die Weichen dieser Strecke. Dies könnte bedeuten, dass andere Strecken, die in diesem Zyklus verfügbar waren, nun nicht mehr verfügbar sind. Deshalb wird die Tabelle in jedem Lua-Zyklus geleert und neu berechnet.

– Ende –