

# EEP Lua Automatischer Zugsteuerung

Rudy Boer, Frank Buchholz, August 2022  
Version 2.4.1

## Inhalt

Dies sind anklickbare Links.

Zweck	3
Installation	3
Schritte zur Erstellung eines automatischen Zugverkehrs in EEP	5
1 Entwerfen einer (Modell)-Eisenbahnanlage	6
2 Verlegen der Gleise in EEP	7
3 Blöcke für den beabsichtigten Zugverkehr erstellen	7
4 Züge im 3D-Modus platzieren	9
5 Layout speichern	10
6 Layout im EEP Layout Tool Öffnen	10
7 Code mit dem Lua ATC Code Generator generieren	11
8 Code in den Lua-Skript-Editor einfügen	13
9 'enterBlock' Funktion in die Kontakte eintragen	14
10 Tabellen der Zulässigen Blöcke erstellen	14
11 Zugtabelle Bearbeiten	15
12 Zugsuche-Modus	15
13 Bereit, einige Züge zu fahren!	16
Wie man mit dem Fahr-Modus aufhört und den aktuellen Zustand speichert	16
Züge später hinzufügen oder entfernen	16
Zugumkehr in Sackgassen	17
Zugumkehr bei nicht-Sackgassen	18
Vermeiden von Deadlocks	18
Kollisionen an Kreuzungen vermeiden	20
Wie man Züge früher in den nächsten Block einfahren lässt	22

Lua-Funktion zum Verlassen eines Blocks	22
Lua-Funktion zur Freigabe einer Weiche	22
Erleichterung der Definition der zulässigen Blöcke	23
Eine Inline-Tabelle innerhalb der Zugtabelle	23
Verweise auf Tabellen außerhalb der Zugtabelle	23
Verweise auf mehrere Tabellen außerhalb der Zugtabelle	24
Kombinieren Sie eine Inline-Tabelle und einen Verweis auf eine externe Tabelle	24
Kombinieren Sie Verweise auf externe Tabellen und fügen Sie Blöcke hinzu	24
Kombinieren Sie Verweise auf externe Tabellen, aber entfernen Sie Blöcke	24
Zufällige Wartezeiten	25
Optionen	25
Löschen des Ereignisfensters	25
Signalzustände ändern	26
Sprache wählen	26
Parameter ein-/ausschalten	26
Gleisbildstellpult standardmäßig Aktivieren	26
Zählersignal zum Einstellen der Protokollierungsstufe verwenden	27
Demo-Layouts	28
EEP_LUA_ATC_Demo_1	28
EEP_LUA_ATC_Demo_2	31
EEP_LUA_ATC_Demo_3	33
EEP_LUA_ATC_Demo_4	35
EEP_LUA_ATC_Model_Railway_Layout_1	36
EEP_LUA_ATC_Model_Railway_Layout_2	38
EEP_LUA_ATC_Peace_River	39
EEP_LUA_ATC_Double_Slip_Turnouts	40
EEP_LUA_ATC_Swyncombe	43
EEP_LUA_ATC_Depots	45
Wie man das Modul "BetterContacts" zur Vereinfachung der Konfiguration verwendet	51
Überblick, wie Lua automatischen Zugverkehr erzeugt	52
Fehlersuche	52
Allgemeine Tipps	52
Typische Probleme beim Hinzufügen von blockControl zu bestehenden Layouts	53
Problem "Weiche zwischen Vorsignal und Hauptsignal"	53
Problem "Zwei Signale auf demselben Gleis"	54
Problem "Sackgasse ohne Blocksignal"	54
Potenzielles Problem "Zu viele Zwei-Wege-Blöcke"	54
So zeigen Sie den Status von Signalen und Weichen an	54

## Zweck

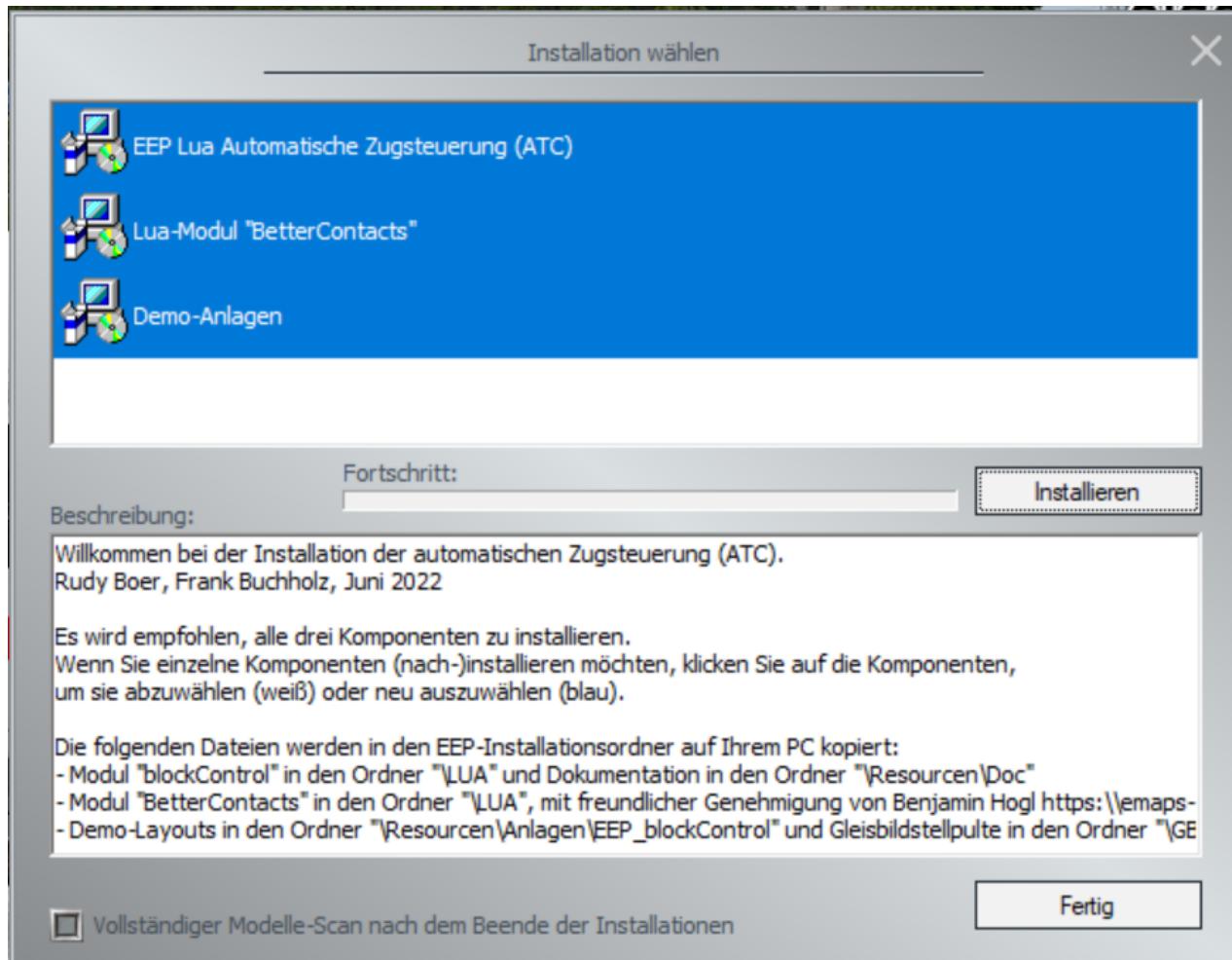
Dieses Lua-Modul automatisiert den Zugverkehr auf jeder EEP-Anlage, ohne dass man selbst Lua-Code schreiben muss. Alles, was der Benutzer tun muss, ist in Lua-Tabellen zu beschreiben, welche Züge in welchen Blöcken erlaubt sind. In diesen Tabellen können auch Haltezeiten, z.B. an Bahnhöfen, pro Zug(gruppe) und pro Block angegeben werden.

Das Lua-Modul wird die Züge von Block zu Block fahren. Ein Block ist ein Streckenabschnitt, der mit einem Kontakt beginnt, der eine Lua-Funktion auslöst, wenn ein Zug in den Block eingefahren ist, und der mit einem Signal endet, das vom Lua-Modul gesteuert wird, um einen Zug anzuhalten oder freizugeben.

Die Weichen zwischen den Blöcken werden vom Lua-Modul geschaltet. Welche Weichen geschaltet werden sollen, wird in der Streckentabelle festgelegt. Diese Tabelle kann vom Benutzer manuell erstellt werden, sie kann aber auch mit dem "Lua ATC Code Generator" automatisch generiert werden.

## Installation

Laden sie die Datei EEP\_blockControl.zip entweder von [GitHub](#) oder aus der EEP Forum Firebase [https://www.eepforum.de.firebaseio/file/2297-eep-lua-automatischer-zugsteuerung/](https://www.eepforum.de/firebase/file/2297-eep-lua-automatischer-zugsteuerung/). Installieren Sie diese zip-Datei über die Schaltfläche "Modelle installieren" auf der EEP-Startseite.



Dadurch wird das EEP-Installationsprogramm geöffnet. Es wird empfohlen, alle drei Komponenten zu installieren, die standardmäßig ausgewählt sind. Wenn Sie einzelne Komponenten installieren möchten, klicken Sie auf die Komponenten, um die Auswahl aufzuheben (weiß) oder sie erneut auszuwählen (blau).

Die folgenden Dateien werden in den EEP-Installationsordner auf Ihrem PC kopiert:

- Modul **blockControl** zum Ordner \LUA  
Dokumentation zum Ordner \Resourcen\Doc
- Modul **BetterContacts** zum Ordner \LUA  
Mit freundlicher Genehmigung von [Benjamin Hogl](#)
- **Demo-Layouts** in den Ordner \Resourcen\Anlagen\EEP\_blockControl  
**Gleisbildstellpulte** zum Ordner \GBS

Es sind keine Modelle enthalten, daher ist es nicht notwendig, neue Modelle nachträglich zu scannen.

Eine weitere Option für die Installation ist das Entpacken der Archivdatei und das manuelle Verschieben der Dateien:

- LUA Ordner:
  - Verschieben Sie die Datei blockControl.lua in den Ordner \LUA Ihrer EEP-Installation. Damit ist die Installation eigentlich abgeschlossen.
  - Die Dateien blockControl\_template\_ENG.lua bzw. blockControl\_template\_GER.lua können als Ausgangspunkt für die Entwicklung eigener Lua-ATC-Skripte verwendet werden. Speichern Sie sie an einem beliebigen Ort zur späteren Verwendung.
  - Verschieben Sie die Datei BetterContacts\_BH2.lua in den Ordner \LUA Ihrer EEP-Installation. Dieses Modul von [Benjamin Hogl](#) kann verwendet werden, um die Definition der Lua-Zeile in Kontakten zu vereinfachen.
- EEP\_blockControl Ordner:
  - Dieser Ordner enthält mehrere Demo-Layouts mit voll funktionsfähigen Beispielen für ATC. Speichern Sie diese an beliebiger Stelle.
- GBS Ordner:
  - Verschieben Sie die Dateien im Ordner GBS in den Ordner \GBS Ihrer EEP-Installation. Dies ist nur erforderlich, wenn Sie die GBS der Demos anpassen möchten.
- Benutzerhandbücher. Speichern Sie diese, wo immer Sie wollen. Der Standardspeicherort für EEP-Dokumente ist \Resourcen\Doc in Ihrer EEP-Installation.
  - English
  - Deutsch

Der Einfachheit halber sollten Sie diese Links zu Ihren Browser-Favoriten hinzufügen. Dies sind die Werkzeuge, die zur automatischen Generierung des Lua-Codes verwendet werden, der die Routen in einem EEP-Layout festlegt:

- [EEP Layout Tool](#)
- [Lua ATC Code Generator](#)

Vielleicht gefällt Ihnen auch dieses Tool, das den kompletten Bestand eines EEP-Layouts anzeigt:

- [EEP Inventar Tool](#)

## Schritte zur Erstellung eines automatischen Zugverkehrs in EEP

1. Entwerfen Sie eine (Modell)-Eisenbahnanlage und planen Sie den Zugverkehr, d. h. entscheiden Sie, wo die verschiedenen Züge fahren (dürfen) und wo sie halten (können).<sup>1</sup>
2. Verlegen Sie die Gleise in EEP.
3. Die Züge werden von Block zu Block fahren. Erstellen Sie die Blöcke, die den vorgesehenen Zugverkehr ermöglichen, indem Sie ein Signal am Ende und einen Kontakt am Anfang jedes Blocks platzieren.
4. Gehen Sie in den 3D-Modus und platzieren Sie die gewünschten Züge. Geben Sie ihnen einen Namen und die gewünschte Geschwindigkeit. Fahren Sie sie bis zu einem roten Signal, wo sie anhalten werden.
5. Speichern Sie das EEP-Layout.
6. Öffnen Sie das [EEP Layout Tool](#) in einer Browser-Registerkarte und laden Sie das Layout.
7. Öffnen Sie den [Lua ATC Code Generator](#) in einer anderen Browser-Registerkarte und klicken Sie auf die Schaltfläche "Generieren".
8. Fügen Sie im 3D-Modus den Lua-Code in den Skript-Editor ein und klicken Sie auf "Skript neu laden", um den Code auszuführen. Dadurch werden die Lua-Funktionen erstellt, die die Kontakte auslösen werden.
9. Geben Sie in jeden Kontakt die Lua-Funktion: `blockControl.enterBlock_#` ein, wobei # die Nummer des Blocksignals ist.<sup>2</sup>
10. Erstellen Sie die Tabellen, die die zulässigen Blöcke pro Zug oder Zuggruppen definieren.
11. Bearbeiten Sie die Zugtabelle so, dass jeder Zug eine Tabelle mit erlaubten Blöcken (`allowed=...`) besitzt, sowie optional ein Ein/Aus-Zugsignal (`signal=#`) und nur für EEP-Versionen 14.1 oder niedriger, einen Slot (`slot=#`) zum Speichern von Daten hat.
12. Speichern Sie das EEP-Layout. Gehen Sie in den 3D-Modus, öffnen Sie den Skript-Editor und klicken Sie auf "Skript neu laden", um den Code auszuführen. Lua wird im Modus "Zugsuche" gestartet.
13. Wenn alle Züge gefunden sind, schalten Sie den Hauptschalter sowie einen oder mehrere der einzelnen Zugschalter ein und ... staunen Sie und haben Sie Spaß!

Jeder Schritt wird in den folgenden Kapiteln ausführlich behandelt.

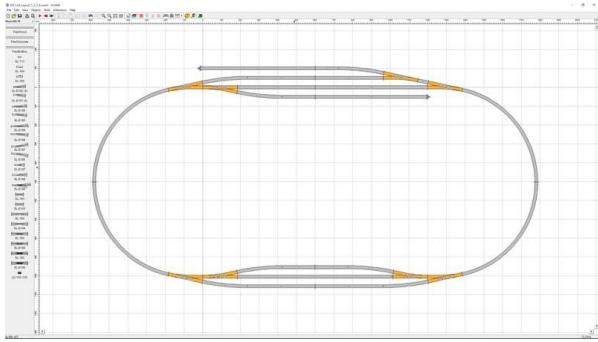
Folgen Sie diesem Link zum [YouTube-Video 9](#) für eine Einführung in Version 2.3.

---

<sup>1</sup> Entscheiden Sie, ob Sie das Modul [betterContacts](#) von Benjamin Hogl nutzen wollen

<sup>2</sup> Wenn Sie [betterContacts](#) verwenden, dann verwenden Sie eine andere Lua-Funktion `blockControl.enterBlock(Zugname, ##)` die Sie bereits in Schritt 3 zu den Kontakten hinzufügen können.

# 1 Entwerfen einer (Modell)-Eisenbahnanlage

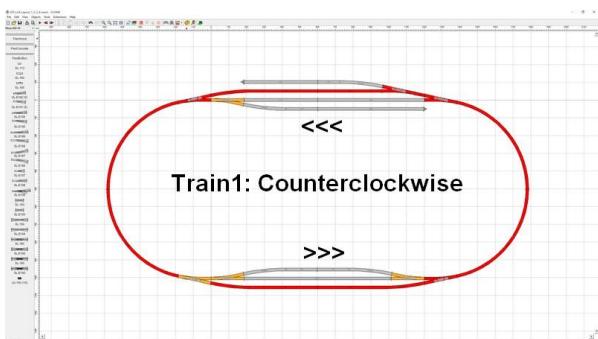


Bei der Gestaltung einer (Modell)-Eisenbahnanlage kann man sie im Kopf entwerfen, auf Papier skizzieren oder ein Modellbahnpogramm verwenden.

Diese Zeichnung einer kleinen Modellbahnanlage wurde erstellt mit [SCARM](#).

Es ist geplant, mit 3 Zügen auf dieser Anlage zu fahren.

Eine einfache Wahl bei der Gestaltung des Verkehrsflusses könnte darin bestehen, einfach jeden Zug überall fahren zu lassen. Das wird aber wahrscheinlich nicht sehr realistisch aussehen. In der Realität fahren die Züge je nach Land auf den rechten oder den linken Gleisen von Parallelstrecken. Außerdem wollen wir nicht, dass Personenzüge in einem Industriegebiet landen. Wenn man alle Züge überall fahren lässt, kann dies zu einem Stillstand führen, wenn entgegenkommende Züge, die beide keine andere Möglichkeit haben, als den Block zu nutzen, den der jeweilige andere Zug besetzt. Für weitere Informationen siehe Kapitel [Vermeidung von Deadlocks](#).

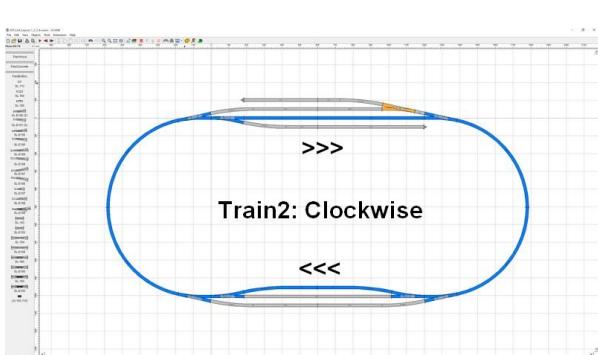


Eine sorgfältige Verkehrsplanung hilft, die Züge realistischer und ohne Probleme fahren zu lassen. Schauen wir uns einmal an, wie 3 Züge auf dieser Anlage schön herumfahren können.

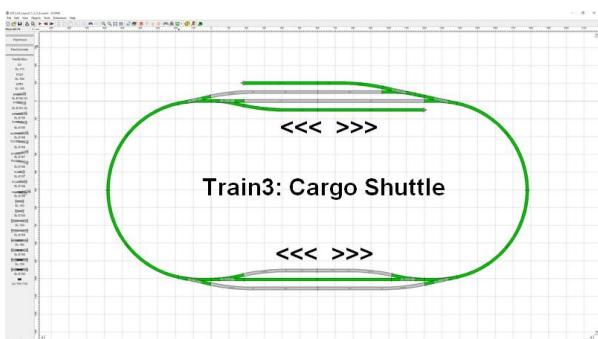
Zug 1 ist ein Personenzug, der gegen den Uhrzeigersinn fährt, mit einem planmäßigen Halt von, sagen wir, 30 Sekunden im Süden, und keinem planmäßigen Halt im Norden. Das heißt nicht, dass er dort nie hält, außer er muss warten, wenn auf der eingleisigen Kurve Verkehr herrscht.

Zug 2 ist ebenfalls ein Personenzug und fährt im Uhrzeigersinn. Er hat ebenfalls einen planmäßigen Halt von, sagen wir, 30 Sekunden im Süden, und keinen Halt im Norden.

Für beide Personenzüge gilt an den Bahnhöfen Rechtsverkehr.

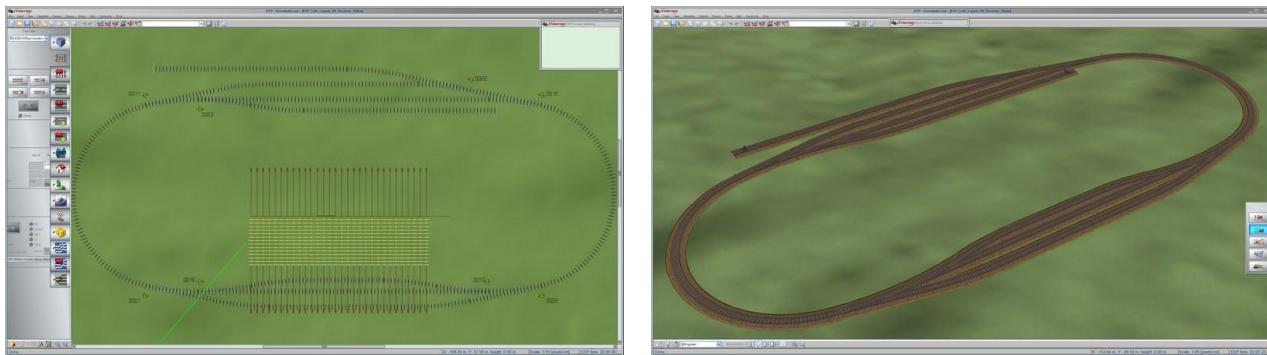


Zug 3 ist ein Güterzug, der zwischen den beiden Sackgassen im Norden pendelt, wo er planmäßige Halte von etwa 30 Sekunden hat. Im Süden benutzt er das mittlere Gleis, ohne planmäßigen Halt. Dieses mittlere Gleis wird in beide Richtungen befahren.



## 2 Verlegen der Gleise in EEP

Die EEP-Datei für dieses Layout ist EEP\_LUA\_ATC\_Demo\_4, die mit der Installation geliefert wurde.



Natürlich können Sie die Gleise für diese Anlage auch selbst verlegen. Wenn Sie eine (Modell)-Eisenbahnanlage mit großer Präzision nachbauen möchten, dann zeigt [dieses Video](#), wie ein Bild als Vorlage in der EEP 2D / 3D Ansicht verwendet werden kann, um die Gleise zu verlegen.

Die Gleise in der Mitte, die nur in der 2D-Ansicht zu sehen sind, sind eine Reihe von unsichtbaren Gleisen, die für die Platzierung von Signalen für unsere Hauptschalter und die Ein- und Ausschalter der einzelnen Züge vorgesehen sind.

## 3 Blöcke für den beabsichtigten Zugverkehr erstellen

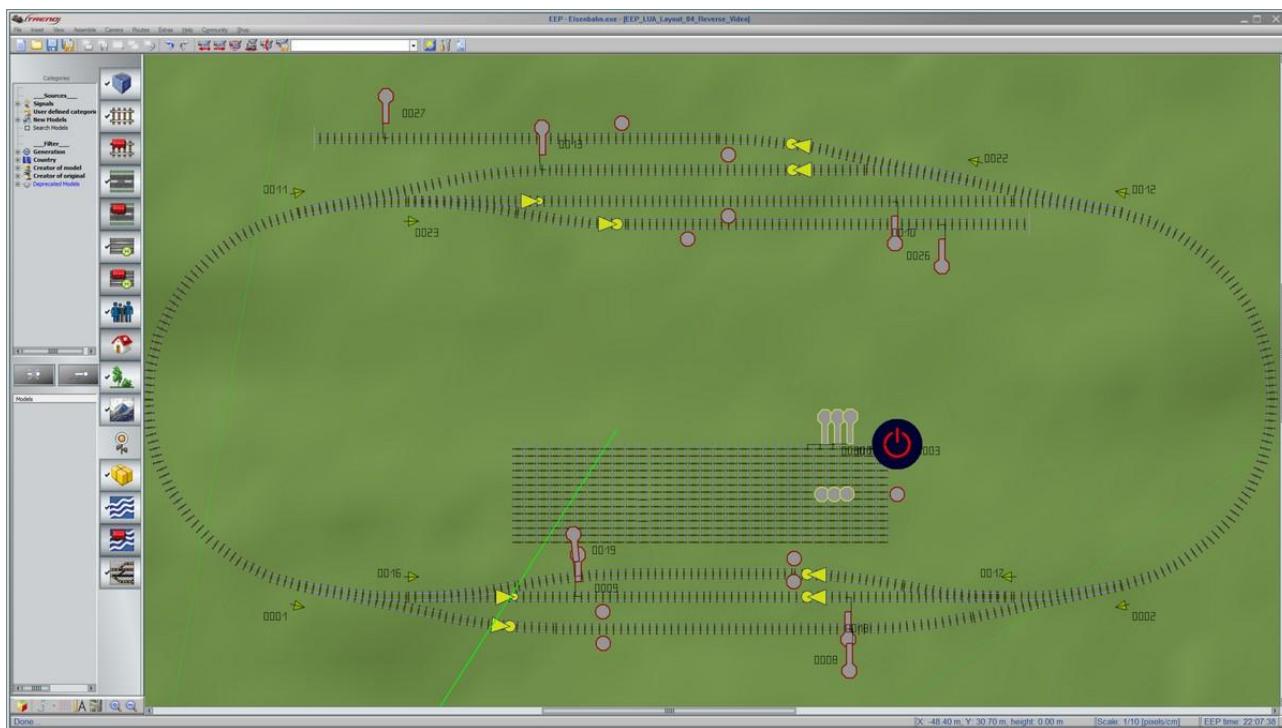
Das Lua-Modul der automatischen Zugsteuerung (blockControl.lua) schickt die Züge von Block zu Block. Ein Block ist ein Streckenabschnitt mit einem Kontakt am Anfang und einem Signal am Ende. Die Blockeingangskontakte können von jedem Typ sein. In den Demos werden 'Sound'-kontakte verwendet, die mit ihrer gelben Farbe in der 2D Ansicht gut sichtbar sind. Die Blocksignale werden von dem Modul gesteuert. Die Züge werden freigegeben, wenn ihre Haltezeit abgelaufen ist und wenn der nächste Block und die Weichen auf dem Weg dorthin nicht von einem anderen Zug belegt sind.

Wie Sie die Anlage in Blöcke unterteilen, bleibt Ihnen überlassen. Es gibt nur eine wichtige **REGEL: Innerhalb eines Blocks darf sich nie eine Weiche befinden.** Weichen sind nur auf den Strecken zwischen den Blöcken erlaubt. Das Modul schaltet die Weichen in die Zustände, die benötigt werden, um den nächsten Block zu erreichen.

Das Ziel der Aufteilung in Blöcke ist es, den Zuverkehr zu ermöglichen, den wir in Schritt 1 entworfen haben. Die Signale sind die einzigen Orte, an denen die Züge halten werden. Ein sinnvoller Ort für Signale sind daher Bahnhöfe und das Ende von Sackgassen. Auf langen Streckenabschnitten können Blöcke in Reihe geschaltet werden, damit die Züge - wie in der Realität - reibungsloser hintereinander fahren können.

Auf unserer Demo-Anlage legen wir auf jedem Gleis im Bahnhof Nord sowie in -Süd einen Block an. Das mittlere Gleis in Süd ist ein Zweirichtungsgleis und erhält daher ein Signal und einen Kontakt an beiden Enden, um zwei Blöcke zu bilden, einen für jede Richtung.

Sie werden sich fragen, warum man die Kurven nicht ebenfalls zu einem Block macht? Nun ... für den Verkehrsfluss ist das nicht notwendig, aber wenn man ein paar Signale mehr haben möchte, als Augenschmaus, können beide Kurven in Zweiwege-Blöcke verwandelt werden.

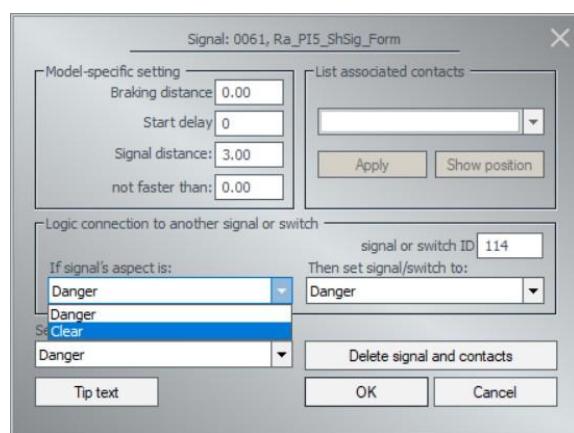


Wir platzieren auch einen Hauptschalter auf einem der unsichtbaren Gleise. Ein Signal vom Typ *Signale > Sonstige > Anlagen Startsignal* wird vom Lua ATC Code Generator automatisch erkannt.

Jeder Zug kann auch einen eigenen Ein-/Aus-Schalter erhalten. Hierfür können Signale beliebigen Typs verwendet werden. Züge ohne ein solches Signal werden immer fahren und können nicht abgeschaltet werden.

Leider können in EEP verschiedene Signaltypen unterschiedliche numerische Werte für ihre Ein- und Ausschaltzustände haben. Das Modul braucht Konsistenz, deshalb gibt es diese **REGEL:** **verwende nur Signaltypen, die einheitliche Ein- und Ausschaltzustände haben.** Dies gilt sowohl für die Block- als auch für die Zugsignale, auch wenn ihre Zustände unterschiedlich sein können, da

beide separat angegeben werden können. Lesen Sie im Kapitel [Signalzustände ändern](#) nach, wie Sie dem Modul die Zustände der verwendeten Signale mitteilen können, falls erforderlich (nur erforderlich, wenn nicht standardmäßig)<sup>3</sup>.



Beachten Sie, dass die Sackgassen keinen Zugkontakt für die Geschwindigkeitsumkehr erfordern. Der Zug wird an dem (unsichtbaren) Signal anhalten. Lua weiß, dass es sich um eine Sackgasse handelt und kehrt die Geschwindigkeit des Zuges für uns um. Mehr dazu im Kapitel [Zugumkehr an Sackgassenblöcken](#).

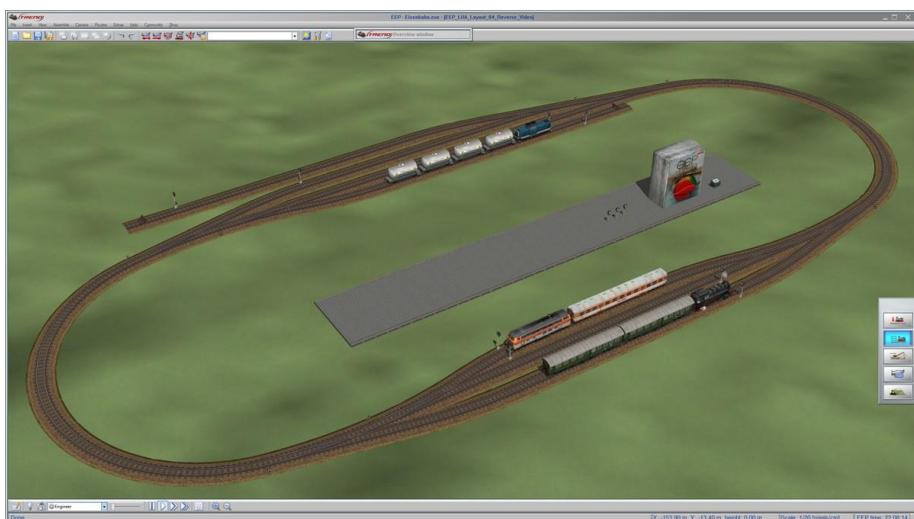
<sup>3</sup> Es ist möglich, andere sichtbare Signale mit unsichtbaren Signalen zu verbinden, die für ATC verwendet werden. Solche verbundenen Signale können andere Signalmuster haben.

Aus Gründen der Realitätsnähe können Sie an der Stelle, an der der Zug die Sackgasse verlässt, ein sichtbares Signal aufstellen. Ein solches Signal muss mit dem Blocksignal verbunden werden, das von Lua gesteuert wird, und zwar, wie im Bild gezeigt, von Halt zu Halt und Fahrt zu Fahrt.

Dieses Signal dient nur als "Augenweide", es spielt in der Lua-ATC keine Rolle und muss sogar aus dem Lua-ATC-Code-Generator ausgeschlossen werden, sonst geht alles schief. In Schritt 7, wenn wir den Lua-Code generieren, werden wir sehen, wie wir diese Signale ausschließen können.

Züge können auch an Blöcken, die keine Sackgassen sind, umgedreht werden. Dies ermöglicht mehr Variation im Zugverkehr. Siehe Kapitel [Zugumkehr an nicht Sackgassenblöcken](#).

## 4 Züge im 3D-Modus platzieren



In Schritt 1, dem Layout Design, haben wir entschieden, wie viele und welche Art von Zügen wir auf dieser Anlage fahren wollen. Dies ist ein guter Zeitpunkt, um EEP in den 3D-Modus zu versetzen und die Züge zu platzieren.

Im Pop-up-Fenster, das nach der Platzierung erscheint, geben Sie jedem Zug einen Namen,

unter dem er später leicht wiedererkannt werden kann. Dieser Name wird in der Lua-Konfiguration verwendet.

Es können auch Wagons hinzugefügt werden. Ihre Namen sind irrelevant.

Stellen Sie sicher, dass vor einem neu platzierten Zug ein freier Block steht. Geben Sie dem Zug eine Geschwindigkeit und lassen Sie ihn bis zum roten Signal des nächsten Blocks fahren, wobei Sie darauf achten, dass Sie vorher die Weichen so umstellen, dass der Zug in einem Block landet, in dem er nach dem Verkehrsplan erlaubt ist.

Das Bild oben zeigt das Ergebnis für unsere Demo-Anlage. Beachten Sie, dass der blaue Zug in einer Sackgasse steht. Das ist im Allgemeinen keine gute Idee, denn beim ersten Start weiß Lua noch nicht, in welche Richtung es den Zug umkehren muss, und er könnte in den Puffer laufen. Am besten platziert man alle Züge in Blöcken, die keine Sackgassen sind.



Es empfiehlt sich, die vorderen und hinteren Kupplungen der Züge zu deaktivieren, um zu vermeiden, dass Züge zusammenkoppeln, wenn sie versehentlich aufeinander treffen.

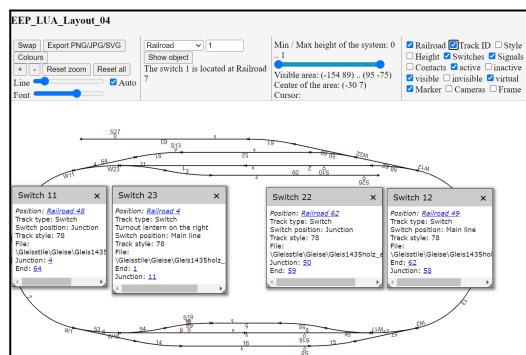
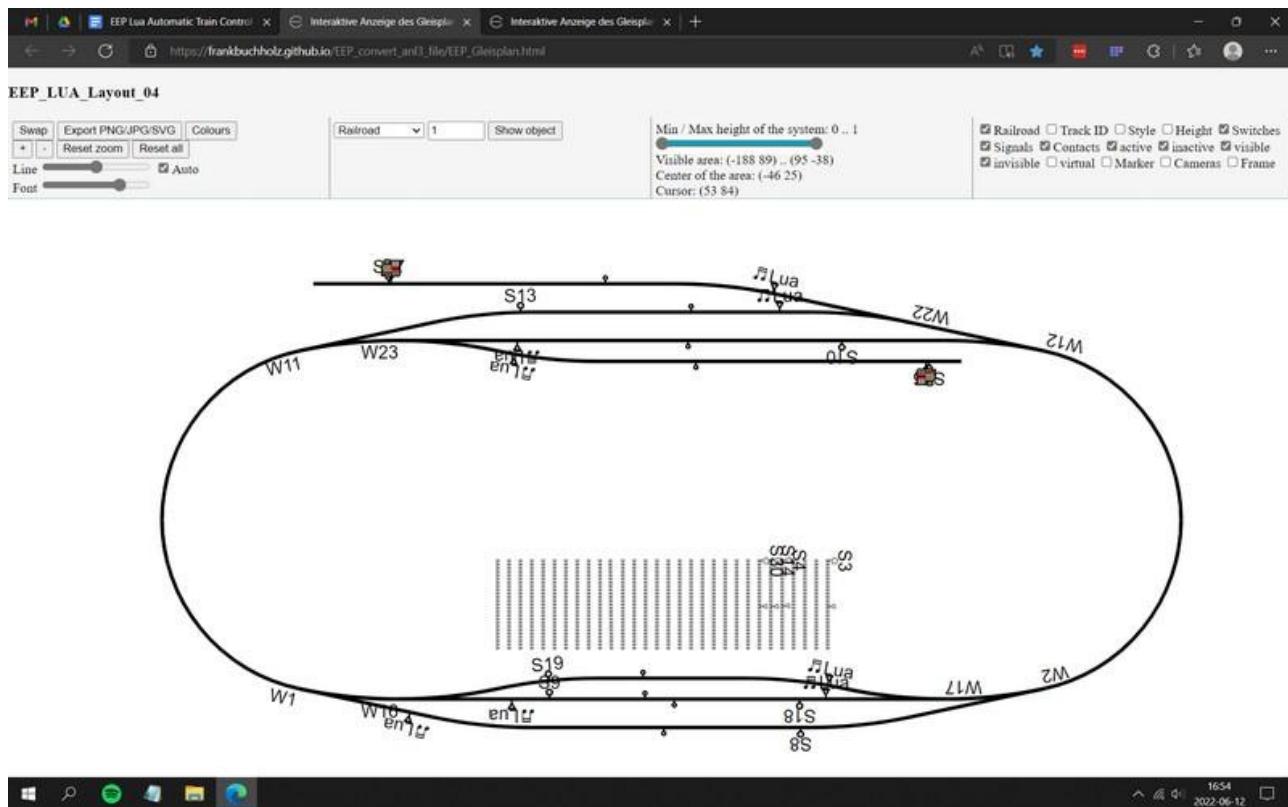
## 5 Layout speichern

Im nächsten Schritt benötigen wir die EEP .anl3-Datei. Wenn alle Signale, Kontakte und Züge platziert sind und alle Züge einen Namen erhalten haben und auf ein rotes Signal gefahren sind, ist es an der Zeit, die Anlage zu speichern. Dies kann über das Menü *Datei > speichern unter* erfolgen.

## 6 Layout im EEP Layout Tool Öffnen

Öffnen Sie das [EEP Layout Tool](#) in einer Registerkarte des Webbrowsers. Die Anzeigesprache kann über die Schaltflächen oben rechts geändert werden.

Laden Sie die EEP-Layoutdatei über die Schaltfläche "Datei auswählen". Der Bildschirm sollte in etwa wie folgt aussehen. Die Sichtbarkeit von Elementen kann über die Häkchen oben rechts geändert werden. Farben, Liniendicke und Textgröße können über die Schaltflächen und Schieberegler oben links geändert werden.



Sie können nun prüfen, ob die Signale, Weichen und Kontakte ihrem Plan entsprechen.

Wenn Sie die Streckentabelle selbst erstellen und nicht den Auto-Generator (nächstes Kapitel) verwenden möchten, beachten Sie, dass bei EEP die Zustände "Haupt" und "Abzweig" einer Weiche manchmal verwirrend sein können ... was sichtbar wie "Haupt" aussieht, kann die Bezeichnung "Abzweig" haben.

Was helfen kann, ist, alle Weichen auf der EEP-Anlage so zu schalten, dass der Zug auf die Stelle fährt, die Ihrer Meinung nach als "main" bezeichnet werden sollte, und dann die Datei im [EEP Layout Tool](#). Wenn Sie mit der linken Maustaste auf eine Weiche klicken, öffnet sich ein Info-Popup, das den Zustand der Weiche und einige andere Informationen anzeigt. Umschalttaste + Linksklick öffnet mehrere dieser Popup-Fenster.

## 7 Code mit dem Lua ATC Code Generator generieren

Öffnen Sie den [Lua ATC Code Generator](#) in einem anderen Browser-Tab. Die zuvor ausgewählte Sprache wird auch hier verwendet.

Drücken Sie die Schaltfläche "Generieren". Das Tool generiert nun den Lua ATC-Code für dieses Layout:

```
-- EEP File 'EEP_LUA_Layout_04'
-- Lua program for module 'blockControl.lua' for track system 'Railroad'
-- For every block entry contact enter the following into field 'Lua function' (where ## is the
-- number of the block signal): blockControl.enterBlock_##
-- Allowed blocks with wait time
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, speed=44 },
  { name="#Orange", signal=0, allowed=all, speed=67 },
  { name="#Steam",  signal=0, allowed=all, speed=62 },
}

local main_signal = 3

local block_signals = { 8, 9, 10, 13, 18, 19, 26, 27, }
-- 8 signals use BLKSIGRED = 1, BLKSIGGRN = 2

local two_way_blocks = { { 9, 18 }, }

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, }, reverse=true },
  { 26, 18, turn={ 23,2, 11,2, 1,2, 16,1, }, reverse=true },
  { 27, 9, turn={ 22,1, 12,1, 2,2, 17,1, }, reverse=true },
  { 27, 19, turn={ 22,1, 12,1, 2,2, 17,2, }, reverse=true },
}

-- @@@@@@@@@@@@@@@@ Remaining part of main script in EEP
-- @@@@@@@@@@@@@@@@

clearlog()

local blockControl = require("blockControl") -- Load the module

blockControl.init({
  logLevel      = 1,                      -- Initialize the module
  extreme       = true,                   -- (Optional) Log level 0 (default): off, 1: normal, 2: full, 3:
})
```

```

trains      = trains,          -- (Optional) Unknown trains get detected automatically,
however, such trains do not have a train signal and can go everywhere.

blockSignals = block_signals, -- Block signals
twoWayBlocks = two_way_blocks, -- Two way twin blocks (array or set of related blocks)
routes      = routes,         -- Routes via turnouts from one block to the next block
paths       = anti_deadlock_paths, -- Critical paths on which trains have to go to avoid
lockdown situations

MAINSW      = main_signal,    -- ID of the main switch (optional)

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
})

--[[ Optional: Set one or more runtime parameters at any time
blockControl.set({
  logLevel      = 1,          -- (Optional) Log level 0 (default): off, 1: normal, 2: full, 3:
extreme
  showTippText  = true,        -- (Optional) Show tipp texts true / false (Later you can toggle
the visibility of the tipp texts using the main switch.)
  start         = false,       -- (Optional) Activate / deactivate main signal. Useful to start
automatic block control after finding all known train.
  startAllTrains = true,      -- (Optional) Activate / deactivate all train signals
})
--]]

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

```

**HINWEIS:** Wenn Sie zusätzliche Signale aus optischen Gründen platziert haben, zum Beispiel an Sackgassenblöcken, müssen diese von der Lua-Codegenerierung ausgeschlossen werden. Wenn sie nicht ausgeschlossen werden, werden falsche Blöcke und Strecken erstellt, und die Züge werden nicht richtig fahren. Signale können ausgeschlossen werden, indem man sie in das entsprechende Feld auf der Seite des Lua-Code-Generators einträgt.

#### Einschränkungen:

- Im Moment kann das Modul keine alternativen Routen für das gleiche Paar von Start- und Endblöcken verarbeiten. Das würde alle Weichen aller alternativen Routen sperren. Daher wird nur eine der vorgegebenen Routen verwendet, und zwar eine derjenigen die die geringste Anzahl von Weichen enthält.
- Sie erhalten keine generierten Pfade, um mögliche Stillstände aufzulösen. Wenn Sie auf Stillstände stoßen, müssen Sie Ihre eigene Lösung entwickeln, indem Sie die oben gezeigten Tipps verwenden.
- Doppelkreuzungsweichen, die aus 4 einzelnen Weichen bestehen, können problemlos verwendet werden, da das Modul einfach diese 4 Weichen sieht. Vergessen Sie nicht, die Kreuzung zu sichern, indem Sie eine Weiche vom anderen Teil der Kreuzung zu den (generierten) Routen hinzufügen (wenn der Abstand der Gleise ausreichend ist, können Sie dies für die geraden Teile der Kreuzung weglassen).
- Gleisobjekte mit Doppelkreuzungsweichen haben nur 1 Weiche mit 4 Stellungen. Das Generierungsprogramm kennt solche speziellen Weichen noch nicht. Daher müssen Sie die benötigten Routen manuell erstellen, was auch gut funktioniert.
- Wendestrecken für Sackgassen erhalten Sie automatisch, Wendestrecken für Zweirichtungsblöcke müssen Sie jedoch selbst definieren.

- Müll rein - Müll raus: Wenn die Anlage nicht über geeignete Blocksignale verfügt, können Sie keine vernünftigen Ergebnisse erwarten.

Folgen Sie diesem Link zum [YouTube-Video 6 zum Lua ATC Code Generator](#)

Zur Information können Sie das [Inventar programm](#)<sup>4</sup> verwenden, 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.

Contacts: 10							Hide columns ▼	?	X
Type of contact	Type of contact	Track	Trigger	Train position	Lua function	Tip's text			
5	Vehicle -0 km/h max	<a href="#">Railroad 60</a>	in the direction of the track	Front		Block 26 reverse direction			
5	Vehicle -0 km/h max	<a href="#">Railroad 63</a>	in the direction of the track	Front		Block 27 reverse direction			
256	Sound undefined	<a href="#">Railroad 1</a>	against track direction	End	blockControl.enterBlock_10	Block 10			
256	Sound undefined	<a href="#">Railroad 3</a>	in the direction of the track	End	blockControl.enterBlock_26	Block 26			
256	Sound undefined	<a href="#">Railroad 6</a>	in the direction of the track	End	blockControl.enterBlock_8	Block 8			
256	Sound undefined	<a href="#">Railroad 8</a>	against track direction	End	blockControl.enterBlock_9	Block 9			
256	Sound undefined	<a href="#">Railroad 9</a>	in the direction of the track	End	blockControl.enterBlock_18	Block 18			
256	Sound undefined	<a href="#">Railroad 52</a>	in the direction of the track	End	blockControl.enterBlock_13	Block 13			
256	Sound undefined	<a href="#">Railroad 55</a>	in the direction of the track	End	blockControl.enterBlock_19	Block 19			
256	Sound undefined	<a href="#">Railroad 61</a>	in the direction of the track	End	blockControl.enterBlock_27	Block 27			

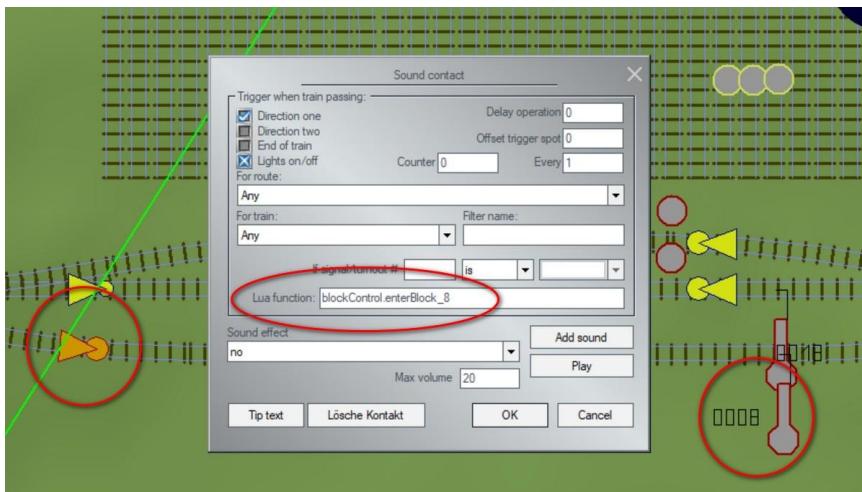
## 8 Code in den Lua-Skript-Editor einfügen

Der vom Tool generierte Code wird automatisch in die Zwischenablage kopiert. Kehren Sie zur EEP 3D Ansicht zurück, öffnen Sie den Lua Script Editor und fügen Sie den Code dort ein.

Drücken Sie nun auf "Skript neu laden", um den Code auszuführen. Dadurch werden die notwendigen Funktionen für die Blockeintragskontakte erstellt.

<sup>4</sup> [https://frankbuchholz.github.io/EEP\\_convert\\_anl3\\_file/EEP\\_Inventar.html](https://frankbuchholz.github.io/EEP_convert_anl3_file/EEP_Inventar.html)

## 9 'enterBlock' Funktion in die Kontakte eintragen



Wir müssen einen Lua-Funktionsaufruf zu jedem Block kontakt hinzufügen.

Gehen Sie in den 2D-Modus und fügen Sie ein

`blockControl.enterBlock_#`

in das Lua-Funktionsfeld ein<sup>5</sup>, wobei # die Nummer des Blocksignals ist. Wiederholen sie dies für jeden Blockeingangskontakt.

## 10 Tabellen der Zulässigen Blöcke erstellen

Der Codegenerator kennt unsere Absichten mit dem Verkehrsplan nicht. Er hat eine Tabelle namens "all" erstellt, die alle Blöcke enthält, und in der Tabelle "trains" lässt er jeden Zug in "all"-Blöcken zu:

```
-- Allowed blocks with wait time
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, speed=44 },
    { name="#Orange", signal=0, allowed=all, speed=67 },
    { name="#Steam",  signal=0, allowed=all, speed=62 },
}
```

In dem Verkehrsplan, den wir in Schritt 1 entworfen haben, verkehrt der Zug '#Steam' gegen den Uhrzeigersinn, '#Orange' verkehrt im Uhrzeigersinn und '#Blue' pendelt zwischen den Sackgassen. Wir müssen die Tabelle 'all' durch drei Tabellen ersetzen, die wir selbst festlegen:

```
-- Allowed blocks with wait time
local CW      = { [10]= 1, [19]=30, }
local CCW     = { [ 8]=30, [13]= 1, }
local Shuttle = { [ 9]= 1, [18]= 1, [26]=30, [27]=30, }
```

Die Zahl hinter dem = steht für eine Wartezeit. Zum Beispiel bedeutet `[19]=30`, dass der Zug mindestens 30 Sekunden in diesem Block verbleiben wird. Dies schließt die Fahrzeit vom Blockeinfahrtskontakt bis zum Signal ein. Ein Wert von 1 Sekunde bedeutet, dass der Zug durchfährt, sofern er nicht durch anderen Verkehr zum Anhalten gezwungen wird. Weitere Informationen zu Wartezeiten und deren Zufallsgenerierung finden Sie im Kapitel [Zufällige Wartezeiten](#).

<sup>5</sup> Wenn Sie [BetterContacts](#) verwenden, dann benutzen Sie stattdessen diese Lua-Funktion:  
`blockControl.enterBlock(Zugname, #)`

Im Kapitel [Erleichterung der Definition der zulässigen Blöcke](#) finden Sie weitere Informationen zur Definition der Tabellen mit den zulässigen Blöcken und zur Vermeidung von Wiederholungen. Wenn sich die zulässigen Blöcke für verschiedene Züge nur um ein paar Blöcke unterscheiden, gibt es elegante Möglichkeiten, dies effizient festzulegen.

**HINWEIS:** Der EEP-Lua-Skript-Editor ist nicht der schönste Editor zum Arbeiten. Die Bearbeitung des Codes ist mit einem externen Editor wie [Notepad++](#) viel einfacher.

## 11 Zugtabelle Bearbeiten

Wir ordnen nun jedem Zug in der Zugtabelle eine Tabelle der zulässigen Blöcke zu:

```
local trains = {
    { name="#Blue",   signal= 4, allowed=Shuttle, speed=40, },
    { name="#Orange", signal=14, allowed=CW,        speed=70, },
    { name="#Steam",  signal=30, allowed=CCW,       speed=60, },
}
```

In diesem Beispiel hat jeder Zug seine eigene Tabelle mit zulässigen Blöcken. Es ist jedoch völlig in Ordnung, mehreren Zügen dieselbe Tabelle zuzuweisen, wie es bei der Standardtabelle "all" der Fall war.

Es ist nicht zwingend erforderlich, auf eine Tabelle der erlaubten Blöcke zu verweisen. Die erlaubten Blöcke können auch in einer Tabelle innerhalb der Zugtabelle aufgelistet werden, etwa so:

```
{ name="#Blue", signal=1, speed=44, allowed={ [9]=1, [19]=1, [26]=30, [27]=30, }, },
```

Für kleine Anlagen kann dies gut funktionieren. Sobald jedoch mehr als ein Zug auf denselben Blöcken erlaubt ist, sind die benannten Tabellen praktischer. Sie vermeiden übermäßige Wiederholungen ähnlicher Daten, Kommentare lassen sich leichter einfügen, und Änderungen sind einfacher durchzuführen.

Die Signalnummer für jeden Zug ist diejenige, die als individueller Ein- und Ausschalter verwendet wird.

### HINWEIS für Benutzer der EEP-Versionen 11 - 14.1:

Benutzer von EEP v14.1 und niedriger müssen die Angabe slot=# zu jedem Zug hinzufügen (zumindest für alle Züge, deren Richtung sich ändern kann), wobei # eine eindeutige Nummer für jeden Zug ist. Beispiel:

```
{ name="#Blue", signal=1, allowed=Shuttle, speed=44, slot=1, },
```

Diese Slots sind Speicherplätze, an denen Lua Daten zwischen den Sitzungen speichert. In den Version EEP v14.2 und höher werden diese Slot-Nummern nicht benötigt, da in diesen Versionen die Daten auf eine andere Art gespeichert werden.

## 12 Zugsuche-Modus

Wir sind bereit für eine Probefahrt.

- Wenn das EEP-Ereignisfenster noch nicht angezeigt wird, aktivieren Sie es über das Häkchen in den EEP-Einstellungen.

- Wenn ein externer Editor verwendet wurde, ist dies der Moment, Ihren Code in den Lua-Skript-Editor zu kopieren und einzufügen.
- Klicken Sie auf "Skript neu laden" und behalten Sie das EEP-Ereignisfenster im Auge. Es sollte 'Find Mode is Active' anzeigen.
- Das Lua-Modul startet im Zugsuche-Modus. Es wird automatisch versuchen, alle Züge auf der Anlage zu finden, was möglich ist, wenn die Züge durch ein Blocksignal "gefangen" sind.
- Wenn es nicht gelingt, alle Züge zu finden, kann es den Suchmodus nicht verlassen. Dann ist ein manuelles Eingreifen erforderlich. Stellen Sie im Ereignisfenster fest, welche(r) Zug(e) nicht erkannt wurde(n). Stellen Sie die entsprechenden Weichen so ein, dass der Zug bis zum nächsten Block fahren kann, und schalten Sie das Signal auf Grün. Der Zug wird nun zum nächsten Block fahren und sobald er von einem Signal "erwischt" wird, findet Lua ihn.
- Wiederholen Sie diesen Vorgang, bis alle Züge gefunden sind.

## 13 Bereit, einige Züge zu fahren!

Schalten Sie den Hauptschalter ein und, falls noch nicht geschehen, schalten Sie die einzelnen Zugschalter ein.

**Staunen Sie und haben Sie Spaß!**

## Wie man mit dem Fahr-Modus aufhört und den aktuellen Zustand speichert

Vor dem Verlassen von EEP ist es sinnvoll, den Hauptschalter auszuschalten und zu warten, bis alle Züge an einem roten Signal zum Stillstand gekommen sind.

Dank des 'Zugsuche-Modus', der beim Laden einer Layout-Datei oder nach 'Reload script' gestartet wird, kann das Layout jedoch jederzeit gespeichert werden, auch während die Züge fahren. In der nächsten Sitzung wird Lua die Züge dann wieder finden. Wenn Sie keine Änderungen am Layout vorgenommen haben und keine Lua-Bearbeitung durchgeführt haben, brauchen Sie das Layout nicht über das Menü zu speichern, der aktuelle Zustand wird automatisch gespeichert, wenn EEP beendet wird.

## Züge später hinzufügen oder entfernen

Wenn Sie später Züge hinzufügen oder entfernen möchten, dann:

- Schalten Sie den Hauptschalter aus und lassen Sie alle Züge zum Stillstand kommen.
- Entfernen Sie im 3D-Modus einen Zug, oder legen Sie einen neuen Zug an und geben Sie ihm einen Namen
- Geben Sie ihm eine Geschwindigkeit
- Lassen Sie ihn zu einem Block fahren, in dem er an einem roten Signal anhalten darf.
- Öffnen Sie den Lua-Skript-Editor
- Falls nötig, füge eine neue Tabelle 'erlaubt' hinzu

- Bearbeiten Sie die Zugtabelle, um den Zug, sein Ein-/Aus-Signal und die erlaubten Blöcke hinzuzufügen.
- Klicken Sie auf "Skript neu laden"
- Lua befindet sich nun wieder im "Zugsuchmodus". Lesen Sie in Kapitel [12 Zugsuchmodus](#) nach, um alle Züge zu finden. Starten Sie dann die Anlage neu.

## Zugumkehr in Sackgassen

Um Züge umzukehren, würde die "klassische" Methode darin bestehen, einen Zugkontakt mit Geschwindigkeitsumkehr als Aktion zu verwenden, und zwar in dem Moment, in dem der Zug bereits eine sehr niedrige Geschwindigkeit hat oder gerade erst zu beschleunigen beginnt, so dass die Umkehr fast unmerklich und ohne Ruck erfolgt.

Mit Lua ATC können diese Kontakte weggelassen werden. Wir weisen Lua mit dem Token `reverse=true` an, die Richtung des Zuges an dem Blocksignal umzukehren. Dieses Token wird für Sackgassen automatisch vom Lua ATC Code Generator erzeugt.

Vorteile:

- Es werden weniger Zugkontakte benötigt.
- Es ist keine genauere Positionierung der Kontakte erforderlich, damit die Umkehrung ruckelfrei aussieht.
- Es gibt keine sichtbare Geschwindigkeitsumkehr mehr. Die Züge beginnen zu fahren und beschleunigen sanft in die richtige Richtung.
- Zusätzlich zum Reversieren an Sackgassenblöcken können Züge nun auch an Nicht-Sackgassenblöcken reversiert werden. Ein Block kann sowohl Vorwärts- als auch Rückwärtsrouten haben, mit gleichmäßig verteilter Wahrscheinlichkeit über alle möglichen Routen.

Damit Lua die gewünschte (umgekehrte) Zuggeschwindigkeit kennt, fügen wir in der Zugtabelle die Konfigurationseinstellung "Geschwindigkeit" hinzu. Lua verwendet diese Einstellung, wenn es die Zugrichtung umkehrt<sup>6</sup>:

```
local trains = {
  { name = "#Orange", signal = 4, allowed = everywhere, speed = 80, slot=1, },7
  { name = "#Steam", signal = 30, allowed = everywhere, speed = 45, slot=2, },
  { name = "#Blue", signal = 14, allowed = everywhere, speed = 50, slot=3, },
}
```

Diese Geschwindigkeiten werden automatisch in der Zugtabelle aufgeführt, wenn der Lua ATC Code Generator mit bereits auf der Anlage platzierten Zügen verwendet wird. Diese Angabe wird nur für Züge verwendet, deren Richtung an einem Blocksignal umgekehrt wird. Sie hat keine Funktion für Züge, die nie die Richtung wechseln. Die Geschwindigkeiten, die für diese Züge angegeben sind, dienen nur zur Information.

Die Geschwindigkeit der Züge kann während der Fahrt immer noch manuell über die EEP-Steuerung oder über Zugkontakte geändert werden, aber wenn ein Zug die Richtung umkehrt, erhält er die Geschwindigkeit aus der Zugtabelle.

---

<sup>6</sup> Verwenden Sie immer positive Geschwindigkeitswerte.

<sup>7</sup> Bei EEP-Versionen ab 14.2 können die Slot-Nummern weggelassen werden.

## Zugumkehr bei nicht-Sackgassen

Die Umkehrung von Zügen über Lua-Code statt über Kontakte ermöglicht die Umkehrung von Zügen in einem normalen Block oder in einem Block mit Gegenverkehr. Dies geschieht durch manuelles Hinzufügen einer Strecke zur Streckentabelle, vom aktuellen Block zu einem vorhergehenden Block, und Hinzufügen von `reverse = true`. Beispiel:

```
{ 8, 26, turn={ 23,2, 11,2, 1,1, }, reverse=true },
```

Bei Gegenverkehrsblocks ist zu beachten, dass bei der Umkehrung eines Zuges die Stelle, die vorher das Ende des Zuges war, nun zur Spitze des Zuges wird. Wenn der Zug in die entgegengesetzte Richtung abfährt, passiert diese neue Spitze des Zuges die Kontakte und das Vorsignal und löst die Aktion aus.

Bei Zweirichtungsblöcken, in denen eine Zugumkehr stattfinden soll, kann dies das Vorsignal des Zweirichtungsblocks selbst oder das Vorsignal des nächsten Blocks sein, je nach Platzierung des Vorsignals und der Länge des Zuges.

Es sind nun zwei Situationen möglich:

1. Das Vorsignal des zweiseitigen Zwillingsblocks ist das erste, das ankommt. In diesem Fall muss der Weg vom aktuellen Block zu seinem zweiseitigen Zwillingsblock in der Streckentabelle definiert sein.
2. Das Vorsignal des nächsten Blocks ist das erste. Die Route vom aktuellen Block zu diesem nächsten Block muss in der Routentabelle definiert werden.

**REGEL:** Es ist wichtig, dass diese Situation für ALLE Züge gleich ist. Würden wir diese Situationen vermischen, könnten die Züge am roten Signal des Zwillingsblocks stecken bleiben. Bitte platzieren Sie das Vorsignal im Doppelblock sorgfältig so, dass es sich entweder hinter oder vor dem Ende ALLER Züge befindet.

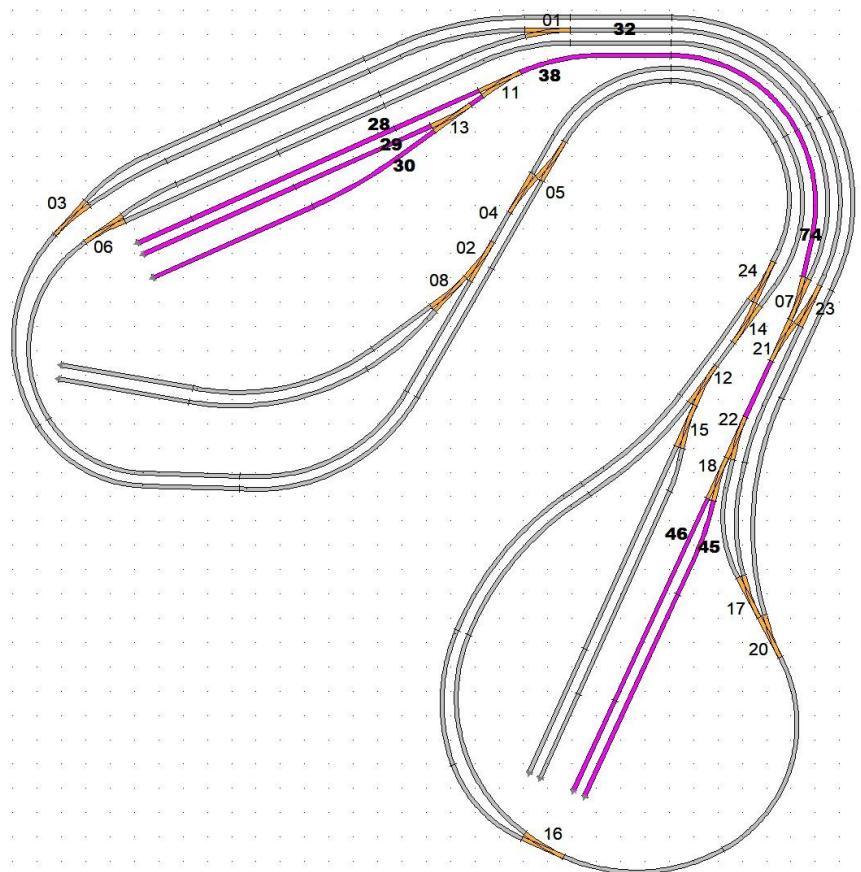
[Dieses YouTube-Video 8](#) zeigt, wie man Züge in einem beliebigen Block umkehren kann, nicht unbedingt in einer Sackgasse.

## Vermeiden von Deadlocks

Es kann zu einer Situation kommen, in der Züge in entgegengesetzten Richtungen aufeinander zufahren und keiner von ihnen einen freien Block findet, in den er einfahren kann. Beide Züge kommen dann zum Stillstand und ein sogenannter "Deadlock" ist entstanden.

Das Bild unten zeigt einen Abschnitt der Demo-Anlage

"EEP\_LUA\_ATC\_Model\_Railway\_Layout\_1" in Pink, an der ein Deadlock auftreten 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 ankommt, kann keiner der 3 Züge mehr eine neue Strecke finden, da es keine freien Blöcke mehr gibt ... wir haben eine Blockade.



Wie kann man Deadlocks vermeiden?

- Die naheliegende Lösung wäre, eine Anlage zu entwerfen, bei der es gar nicht erst zu Blockierungen kommen kann. In der Demo-Anlage ist dies leicht möglich, indem man den Zügen erlaubt, auch den Block 32 anzufahren und so einen der Blöcke 45 oder 46 freizugeben.
- Eine andere Lösung könnte darin bestehen, die Blöcke 38 und 74 in Lua nicht als Blöcke zu definieren und diese Gleise im Wesentlichen zu einem Teil der Weichen zu machen. 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 auf 28,29,30 nicht losfahren, erst ein Zug aus 45,46 wird in die andere Richtung fahren. Um trotzdem sichtbare Zugsignale zu haben, können andere Signale verwendet werden, die über Gleiskontakte gesteuert werden, die nicht unter der Kontrolle des Lua-Moduls stehen.
- Die dritte Lösung besteht darin, das Lua-Modul über den möglichen Stillstand zu informieren und weiter als einen Block vorauszuschauen. Das Lua-Modul soll nicht zulassen, dass ein Zug auf 28, 29 oder 30 losfährt, wenn kein Block in 45 oder 46 frei ist. Wir weisen das Lua-Modul 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 3 Blöcke für die 3 Züge zur Verfügung stehen und eine Blockade in dieser Richtung einfach nicht auftreten kann.

Falls Sie ein sehr langes Zwischengleis haben, das Sie in mehrere Blöcke aufteilen möchten, ist das völlig in Ordnung, der Anti-Stillstands-Pfad könnte dann z.B. so aussehen:

```
{ {28,29,30}, 74, 75, 76, {46,45} },
```

## Kollisionen an Kreuzungen vermeiden

Kreuzungen werden nicht in der Streckentabelle angegeben. Das bedeutet, dass sie nicht reserviert werden, 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 eine Kreuzung fahren können. Das führt nicht zu Problemen, die Züge fahren weiter, es gibt keine Entgleisung ... es sieht nur nicht realistisch aus.

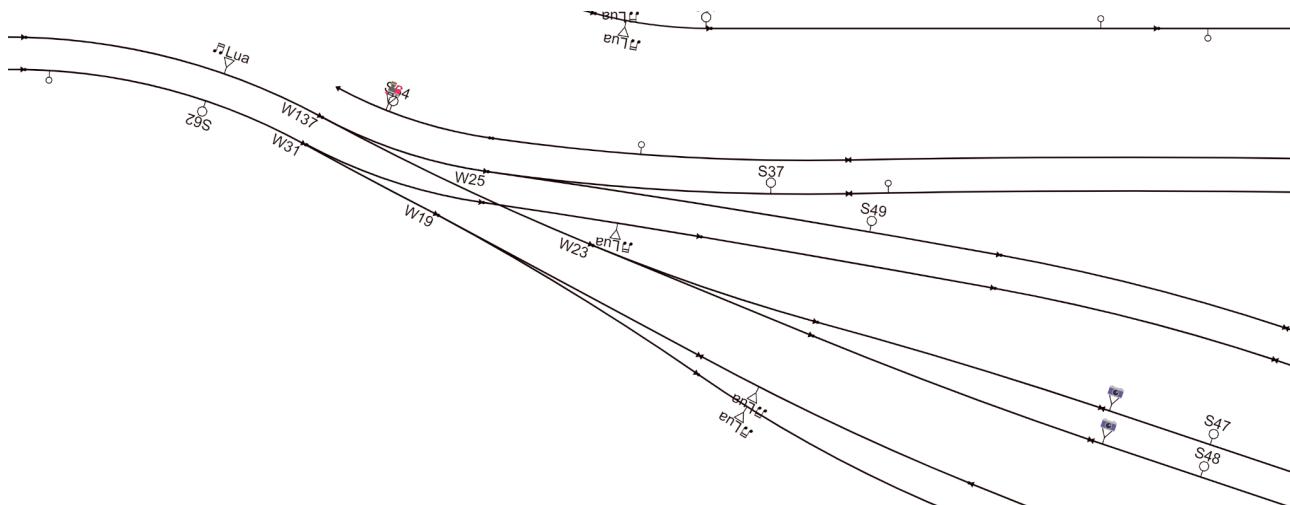
Ohne Vorsichtsmaßnahmen kann genau das passieren ... zwei Züge durchdringen einander:



Kollisionen (bzw. Durchdringungen) auf einer Kreuzung können auf mehrere Arten verhindert werden:

1. Finde die beiden Blöcke auf den Gleisen direkt hinter der Kreuzung und deklariere sie als `two_way_blocks`. 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 A, die über die Kreuzung führt, mindestens eine Weiche enthält, dann nehmen Sie diese Weiche in die Strecke B auf, die ebenfalls über die Kreuzung führt. Dabei spielt es keine Rolle, ob Sie die Weiche auf 1 oder 2 stellen (oder den Dummy-Wert 0 verwenden). Der einzige Grund für die Aufnahme ist, dass Sie sicherstellen wollen, dass diese Weiche reserviert wird, sobald Strecke A aktiviert wird. (Sie können das Gleiche umgekehrt tun, indem Sie eine Weiche der Strecke B in die Strecke A aufnehmen. Dies ist jedoch nicht erforderlich, da die Kreuzung bereits geschützt ist.) Das Ergebnis ist, dass kein Zug eine Fahrt über die Kreuzung beginnen kann, solange entweder Strecke A oder Strecke B diese Weiche reserviert hat.
3. Für Doppelkreuzungsweichen, die aus einzelnen Weichen aufgebaut sind, kann eine spezielle Angabe in der Lua-Konfiguration verwendet werden. Dies wird im Kapitel [EEP LUA ATC Double Slip Turnouts](#) beschrieben.

Hier ist ein Beispiel für eine Kreuzung zwischen den Weichen 137 und 23 in der Anlage "Peace River":



Diese Kreuzung ist nicht Teil eines Blocks, sie liegt zwischen den Blöcken. Dies ermöglicht eine einfache Lösung entsprechend der Option 2: Fügen Sie für die Strecke, die über die Weiche 31 führt, manuell eine Weiche von der anderen Strecke hinzu. Diese zusätzliche Weiche wird nun ebenfalls reserviert und verhindert, dass Züge in die Kreuzung einfahren.

Dies führt zu den folgenden geänderten Strecken für das Peace River Layout:

```
-- 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, } },
  --{ 48, 50, turn={ 23,1, 137,2, } },
```

```

{ 48, 50, turn={ 23,1, 137,2, }},  

...  

--{ 62, 77, turn={ 31,1, }},  

{ 62, 77, turn={ 31,1, 23,1, }}, -- Protect the crossing  

...
}

```

## Wie man Züge früher in den nächsten Block einfahren lässt

Wenn ein Zug einen Block verlässt und über eine Reihe von Weichen - vielleicht sogar mit längeren Gleisabschnitten dazwischen - fahren muss, dann dauert es eine Weile, bis er den nächsten Block erreicht und dort schließlich vom Blockeingangskontakt erkannt wird, vor allem, wenn der Kontakt auf "Zugende" eingestellt ist. Andere Züge, die die Weichen benutzen wollen, die der erste Zug bereits verlassen hat, müssen daher (viel) länger als nötig warten.

Um die anderen Züge früher starten zu lassen, gibt es zwei Möglichkeiten.

### **Lua-Funktion zum Verlassen eines Blocks**

Ein zusätzlicher Kontakt kann dicht hinter dem Blocksignal platziert werden. Die Lua-Funktion, die in diesem Kontakt platziert werden muss, lautet<sup>8</sup>:

`blockControl.leaveBlock_#`

wobei # die Nummer des Blocksignals ist.

Dadurch wird der Block frei, so dass ein anderer Zug bereits von hinten in den Block einfahren kann. Die vorausliegenden Weichen bleiben reserviert, bis der erste Zug sein Ziel erreicht hat.

### **Lua-Funktion zur Freigabe einer Weiche**

Wenn es sicher ist, eine Weiche freizugeben, bevor der erste Zug sein Ziel erreicht hat, kann dies durch das Setzen eines zusätzlichen Kontakts mit der Lua-Funktion erreicht werden:

`blockControl.releaseTurnout(Zugname, #)`

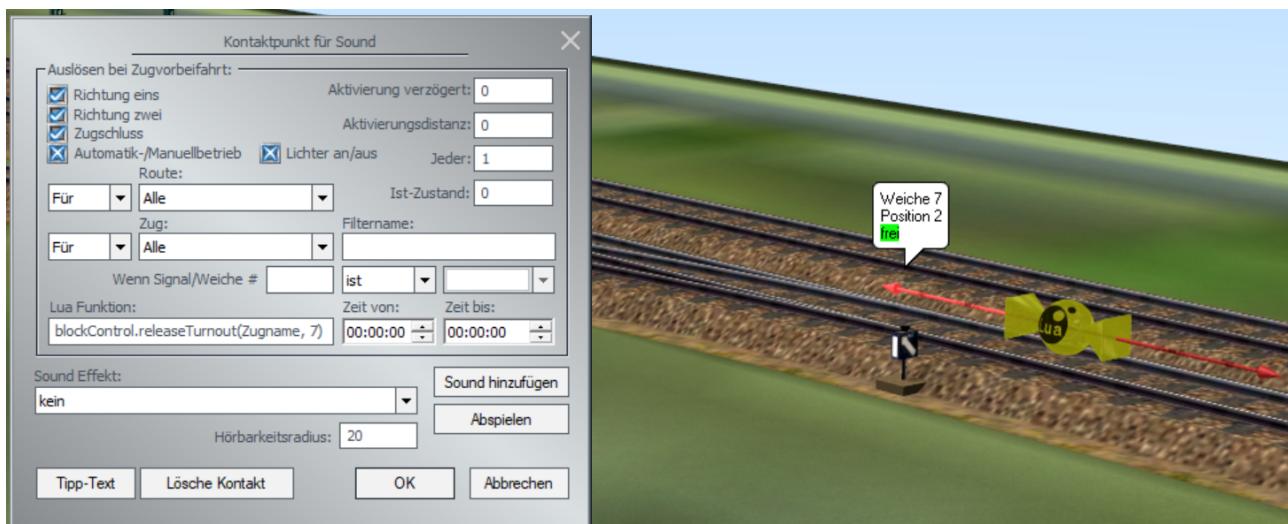
wobei # die Nummer der Weiche ist.

Diese Funktion erfordert die Verwendung von BetterContacts, das in der Installations-ZIP-Datei enthalten ist und automatisch installiert wird.

---

<sup>8</sup> Wenn Sie BetterContacts verwenden, dann benutzen Sie stattdessen diese Lua-Funktion:

`blockControl.leaveBlock(Zugname, #)`



## Erleichterung der Definition der zulässigen Blöcke

In Schritt [10 Erstellen der Tabellen für zulässige Blöcke](#) haben wir gesehen, wie wir festlegen können, welche Züge wir in welchen Blöcken zulassen wollen und wie lang die Wartezeit in den Blöcken sein wird. Höchstwahrscheinlich möchten wir davon abweichen, dass alle Züge in allen Blöcken zugelassen werden und keine Wartezeit haben, was der Lua-Codegenerator für uns erstellt.

Es gibt mehrere Möglichkeiten, die erlaubten Blöcke in der Zugtabelle festzulegen. Werfen wir einen Blick auf die verschiedenen Methoden, die verwendet werden können.

### Eine Inline-Tabelle innerhalb der Zugtabelle

```
local trains = {
    { name="#Steam", signal=1, speed=44, allowed={ [9]=1, [19]=1, [26]=30, [27]=30, }, },
}
```

In diesem Beispiel ist der Zug #Steam in den Blöcken 9, 19, 26, 27 erlaubt. In den Blöcken 26 und 27 hat er eine Wartezeit. Die angegebenen 30 Sekunden sind nicht die Netto-Haltezeit, sondern beinhalten die Fahrzeit vom Blockeingangssensor bis zum Signal. Ein Wert von 1 Sekunde bedeutet, dass der Zug tatsächlich durchfährt, es sei denn, er muss wegen anderen Verkehrs anhalten.

Blöcke, die nicht in der Tabelle aufgeführt sind (oder den Wert 0 haben), sind Blöcke, in die der Zug NICHT hineinfahren darf.

### Verweise auf Tabellen außerhalb der Zugtabelle

```
local CW = { [10]= 1, [19]=30, }
local CCW = { [ 8]=30, [13]= 1, }

local trains = {
    { name="#Blue",   signal=4, allowed=CW,   speed=40, slot=1, },
    { name="#Orange", signal=5, allowed=CW,   speed=70, slot=2, },
    { name="#Steam",  signal=6, allowed=CCW,  speed=50, slot=3, },
}
```

Dies ist eine Arbeitsweise, die die Zugtabelle übersichtlicher macht und außerdem Wiederholungen vermeidet, wenn mehrere Züge auf denselben Blöcken erlaubt sind, wie die beiden CW-Züge (im Uhrzeigersinn).

## Verweise auf mehrere Tabellen außerhalb der Zugtabelle

```
local CW = { [10]= 1, [19]=30, }
local CCW = { [8]=30, [13]= 1, }
local Depot = { [34]=30, [35]=30, }

local trains = {
  { name="#Blue", signal=4, allowed= { CW, Depot, }, speed=40, },
  { name="#Orange", signal=5, allowed= { CW, }, speed=70, },
  { name="#Steam", signal=6, allowed= { CCW, Depot, }, speed=60, },
}
```

In diesem Beispiel ist der Zug #Blau auf den in der Tabelle CW genannten Blöcken sowie auf den Blöcken in der Tabelle Depot erlaubt. #Orange ist im Depot nicht erlaubt. #Steam ist auf den CCW-Blöcken und dem Depot erlaubt.

## Kombinieren Sie eine Inline-Tabelle und einen Verweis auf eine externe Tabelle

```
local CW = { [10]= 1, [19]=30, }

local trains = {
  { name="#Blue", signal=4, allowed= { CW, }, speed=40, },
  { name="#Orange", signal=5, allowed= { CW, [34]=30, }, speed=70, },
  { name="#Steam", signal=6, allowed= { CW, [34]=30, [35]=30 }, speed=60, },
}
```

Der Zug #Blau ist in den in der Tabelle CW genannten Blöcken erlaubt. Der Zug #Orange darf zusätzlich auch in Block 34 fahren. Der Zug #Dampf ist in CW, 34 und 35 erlaubt.

## Kombinieren Sie Verweise auf externe Tabellen und fügen Sie Blöcke hinzu

```
local CW = { [10]= 1, [19]=30, }
local Depot = { [34]=30, [35]=30, }

local trains = {
  { name="#Blue", signal=4, allowed= { CW, Depot, [12]=20 }, speed=40, },
}
```

Der Zug #Blue ist auf den CW- und Depotblöcken sowie auf Block 12 erlaubt.

## Kombinieren Sie Verweise auf externe Tabellen, aber entfernen Sie Blöcke

```
local CW = { [10]= 1, [19]=30, }
local depot = { [34]=30, [35]=30, [36]=30, [37]=30, }

local trains = {
  { name="#Blue", signal=4, allowed= { CW, Depot, [37]=0 }, speed=40, },
}
```

Zug #Blue ist sowohl auf den CW- als auch auf den Depotblöcken erlaubt, jedoch ... NICHT auf Block 37. Blöcke, die den Wert 0 haben, werden ausgeschlossen.

## Zufällige Wartezeiten

Zügen kann ein planmäßiger Halt in einem Block zugewiesen werden. Dies geschieht durch Angabe einer Wartezeit in Sekunden:

```
local CCW = { [12] = 40, [13] = 1, }
```

Im Block 12 wird der Zug eine Wartezeit von 40 Sekunden haben. Dies beinhaltet die Fahrzeit vom Blockeingangskontakt bis zum Signal.

Ein Wert von 1 Sekunde bedeutet, dass der Zug durchfährt, es sei denn, ein anderer Verkehr zwingt ihn zum Anhalten.

Ein Wert von 0 bedeutet, dass der Zug nicht in diesen Block einfahren darf.

Um Vorhersehbarkeit zu vermeiden, kann den Zügen eine zufällige Wartezeit zugewiesen werden, indem ein Mindest- und ein Höchstwert angegeben werden:

```
local CCW = { [12]={20,40}, [13]=1, }
```

In Block 12 wird der Zug eine zufällige Wartezeit zwischen 20 und 40 Sekunden haben.

Wenn wir mehreren Zügen und/oder Blöcken dieselbe Zufallszeit von 20 bis 40 zuweisen möchten, kann die folgende Konstruktion verwendet werden, um eine endlose Wiederholung ähnlicher Werte in Tabellen zu vermeiden:

```
local T1 = { 20,80 } -- Random Wait Time 1
local T2 = { 15,30 } -- Random Wait Time 2

local Depot = { [34]=T1, [35]=T1, [36]=T1, [37]=T2, }
local CCW   = { [12]=T2, [13]=1, }

local trains = {
  { name="#Blue", signal=4, allowed= { CW, depot, [37]=0 }, speed=40, },
}
```

## Optionen

Der Code, der auf den Konfigurationsteil folgt, ist für jedes Layout identisch und kann oft unangetastet bleiben, es sei denn, auf dem Layout werden Signale verwendet, die unterschiedliche Rot-/Grün-Zustände haben, oder wenn Sie die verfügbaren Optionen ändern möchten.

## Löschen des Ereignisfensters

```
clearlog()
```

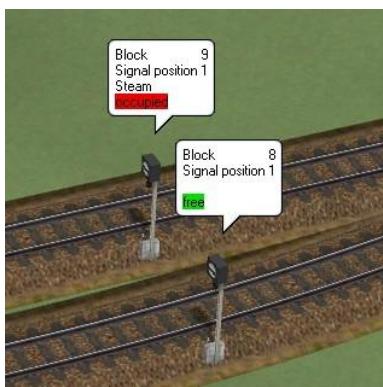
Dadurch wird der EEP-Ereignisfenster nach dem Start und nach einem "Skript neu laden" gelöscht. Wenn Sie den Bildschirm nicht löschen wollen, können Sie diese Zeile löschen oder in einen Kommentar umwandeln: `-- clearlog()`.

## Signalzustände ändern

Leider haben im EEP nicht alle Signaltypen die gleichen Zustände. Für einige Signale ist Halt / 'rot' = 1, während für andere Halt / 'rot' = 2 ist. Wir müssen Lua die Zustände für Rot und Grün der Signale mitteilen, die auf der Anlage verwendet werden: 1 oder 2.

**HINWEIS:** Alle auf der Anlage verwendeten Blocksignale müssen identische Zustände haben. Das Gleiche gilt für alle Zugsignale. Ihre Zustände müssen identisch sein, auch wenn sie sich von den Blocksignalen unterscheiden können.

Wenn Sie andere Signale mit einem anderen Signalmuster anzeigen möchten, können Sie diese mit unsichtbaren Blocksignalen verbinden. Achten Sie darauf, solche Signale vom Generierungsprozess auszuschließen.



Die Signalzustände werden in den Popups, die beim ersten Start eines Layouts erscheinen, als "Signalposition #" angezeigt. Sie lassen sich hier leicht ablesen und der Lua-Code kann bei Bedarf entsprechend geändert werden. Dies geschieht in den Codezeilen, die lauten<sup>9</sup>:

```
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
```

Um die Pop-ups ein- oder auszuschalten, schalten Sie den Hauptschalter zweimal schnell mit Umschalt + Linksklick um.

## Sprache wählen

Die Statusanzeigen im Ereignisfenster werden entsprechend der Sprache, mit der EEP installiert ist, in Deutsch, Englisch oder Französisch angezeigt. Die gewünschte Sprache kann auch mit einem Parameter über die Funktionen `init` oder `set` festgelegt werden:

```
language      = "GER",      -- GER: Deutsch, ENG: Englisch, FRA: Französisch
```

## Parameter ein-/ausschalten

Die folgenden Parameter können ein- oder ausgeschaltet werden:

```
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
```

## Gleisbildstellpult standardmäßig Aktivieren

```
-- Optional: Activate a control desk for the EEP layout
```

<sup>9</sup> Sie brauchen keine Signalzustände zu definieren, wenn die Vorgabe wie beschrieben mit den Standardwerten übereinstimmt.

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

## Zählersignal zum Einstellen der Protokollierungsstufe verwenden

Ein Zählersignal kann auf dem Layout platziert werden, über das der Protokollierungslevel manuell geändert werden kann, ohne das Lua-Editor-Fenster zu öffnen. Ein Beispiel wird in der Demo `EEP LUA ATC Model Railway Layout_1` gezeigt.

Der dazugehörige Lua-Code für dieses Signal 47 lautet wie folgt:

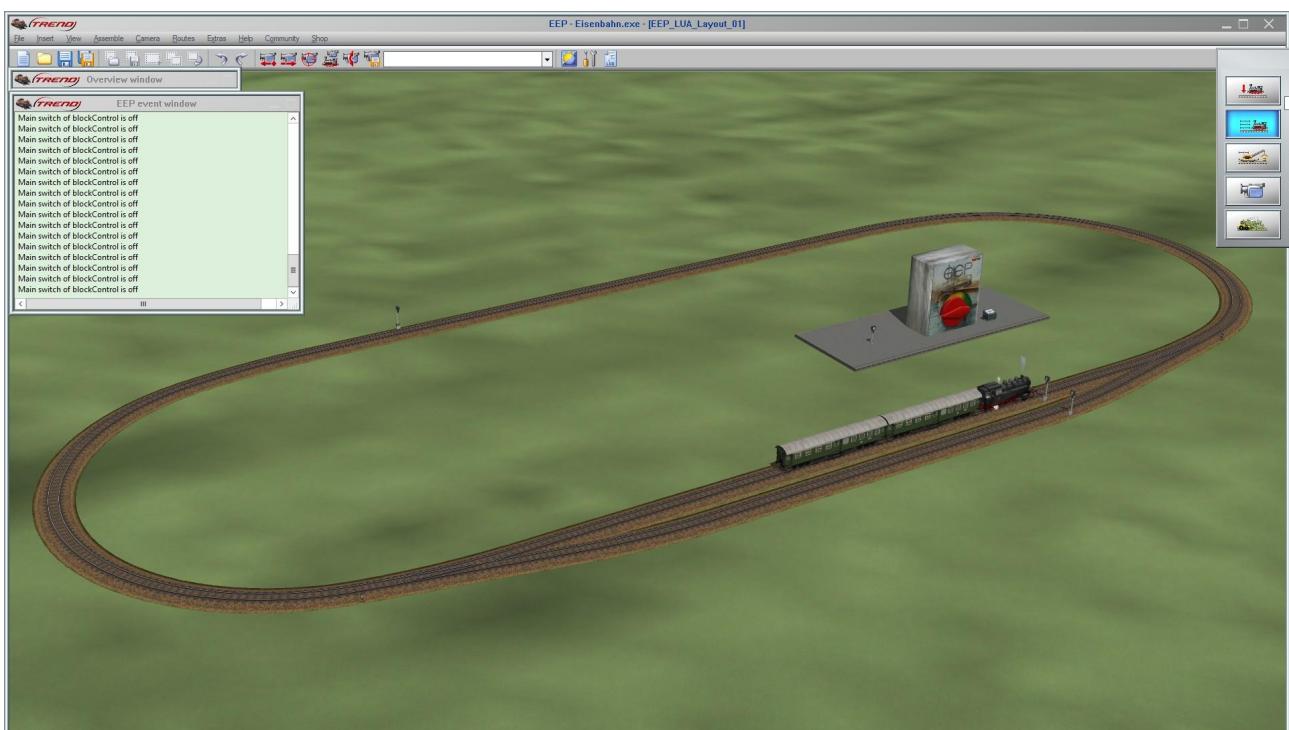
```
-- Optional: Use a counter signal to set the log level
local counterSignal = 47
blockControl.set({ logLevel = EEPGetSignal( counterSignal ) - 1 })
EEPRegisterSignal( counterSignal )
_ENV["EEPOnSignal_"..counterSignal] = function(pos)
  local logLevel = pos - 1
  if logLevel > 3 then
    logLevel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

## Demo-Layouts

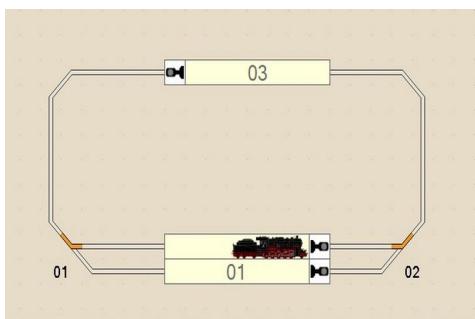
Wenn Sie die Demos installiert haben, die mit dem Paket geliefert wurden, sollten Sie diesen Ordner in Ihrer EEP-Installation haben \Resourcen\Anlagen\EEP\_blockControl, der eine Sammlung von Demo-Layouts enthält.

Die Demo-Layouts werden in den folgenden Kapiteln ausführlich beschrieben.

### EEP\_LUA\_ATC\_Demo\_1

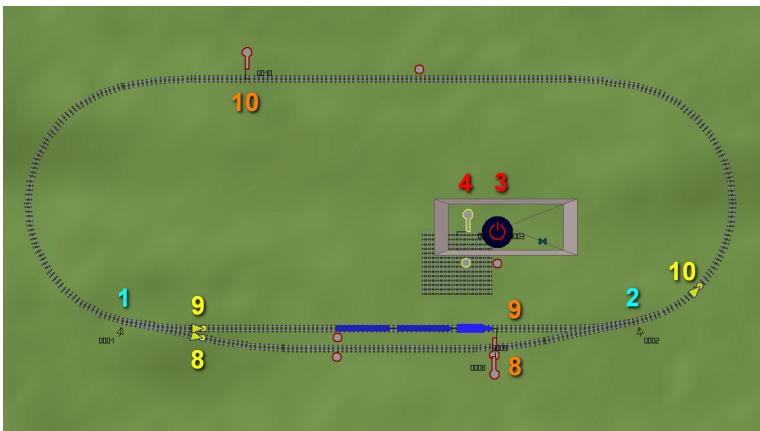


Schreiben wir die Lua-Konfiguration für diese einfache Anlage und lassen wir den Zug vollautomatisch fahren, mit einer Wartezeit am Bahnhof.



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.**

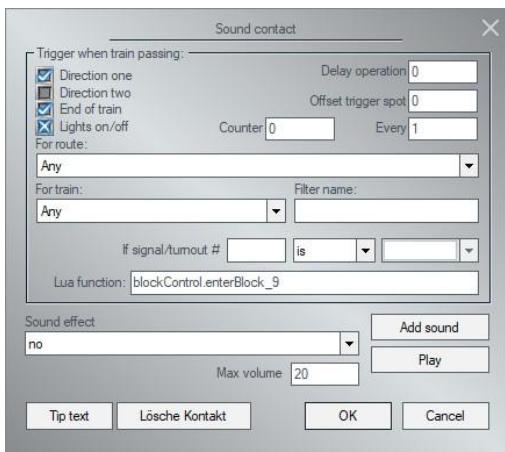
Für diese Anlage erscheint es logisch, die drei Blöcke wie hier gezeigt zu erstellen.



automatisch generiert werden.

Um einen Block in EEP zu definieren, wird ein Signal an der Stelle platziert, an der der Zug anhalten soll. Ein Kontakt wird am Anfang des Blocks platziert.

Das Bild zeigt die Weichen (cyan), die Blocksignale (orange), die Blockeingangskontakte (gelb) und die Ein- und Ausschalter (rot) für diese Anlage. Wenn Sie eine ähnliche Anlage erstellen, können die Nummern abweichen, da sie von EEP



Die Kontakte für den Blockeintrag können von beliebiger Art sein. Gelbe 'Sound'-kontakte sind gut sichtbar und einfach zu verwenden. Im Lua-Funktionsfeld muss diese Funktion eingegeben werden:

`blockControl.enterBlock_#10`.

wobei # die Nummer des Blocksignals ist. Dieser Kontakt lässt Lua wissen, wann ein Zug im Block angekommen ist.

In den meisten Fällen ist "Ende des Zuges" die sichere Wahl. In diesem Fall werden die vorherigen Blöcke und Weichen freigegeben, sobald der letzte Wagen den Kontakt passiert hat. Wenn es keinen Gegenverkehr gibt,

der die Weichen benutzen muss, gibt die Auslösung an der Spitze des Zuges die Weichen und den vorherigen Block eine Zuglänge früher frei, was den Verkehr beschleunigen kann.

Der Konfigurationscode für diese Anlage 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} },
}
```

<sup>10</sup> Denken Sie daran, dass Sie in einer bestimmten Reihenfolge arbeiten müssen:

1. Passen Sie den Lua-Code an, um alle Blocksignale aufzulisten
2. Starten Sie das Layout im 3D-Modus
3. Platzieren Sie die Kontakte, falls noch nicht geschehen
4. Geben 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.

Dies ist alles, um das Layout zu beschreiben. Wenn Sie das Fenster "Lua-Skript-Editor" oder die Lua-Datei in einem Editor wie Notepad++ öffnen, werden Sie feststellen, dass der Code eigentlich länger ist, aber nur dieser obere Teil definiert das Layout. Der untere Teil des Lua-Codes muss nur geändert werden, wenn sich die Zustände der verwendeten Signale unterscheiden (siehe Kapitel [Signalzustände ändern](#)) oder wenn Sie einige der verfügbaren Optionen ändern möchten (siehe Kapitel [Parameter ein-/ausschalten](#)).

Erläuterung des Konfigurationscodes:

```
main_switch = 3
```

Die ID-Nummer des Signals, das als Hauptschalter fungiert.

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

Die Bestandteile haben folgende Bedeutung:

```
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 [4 Züge im 3D Modus platzieren](#).

```
signal = 4
```

Jeder Zug könnte sein eigenes individuelles Start-/Stoppsignal haben. Diese ID-Nummer wird hier angegeben<sup>11</sup>.

```
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, bedeutet dies, dass dieser Zug niemals dorthin fahren wird. Wenn Sie zum Beispiel den Zug in Block 8 nicht zulassen, sollte es heißen: `allowed = {[9]=15,[10]=1,}`
- 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 im Block bleibt. Dies beinhaltet die Fahrzeit vom Kontakt zum Signal, die von der Geschwindigkeit des Zuges, der Länge des Blocks und der Position des Vorsignals abhängt. Um genaue Haltezeiten zu erhalten, lassen Sie die Anlage laufen und schauen Sie sich das Ereignisfenster an. Die Fahrzeit und die Haltezeit werden gemessen und dort angezeigt. Die Zeiten können nun in der Tabelle geändert werden, um eine spezifische Stoppzeit zu erstellen.
- Wenn die erlaubte Untertabelle weggelassen wird, 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} },
}
```

---

<sup>11</sup> Sie können dasselbe Signal für mehrere Züge verwenden, die Sie gemeinsam starten oder anhalten möchten.

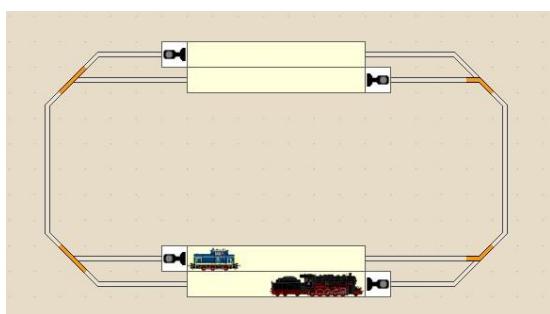
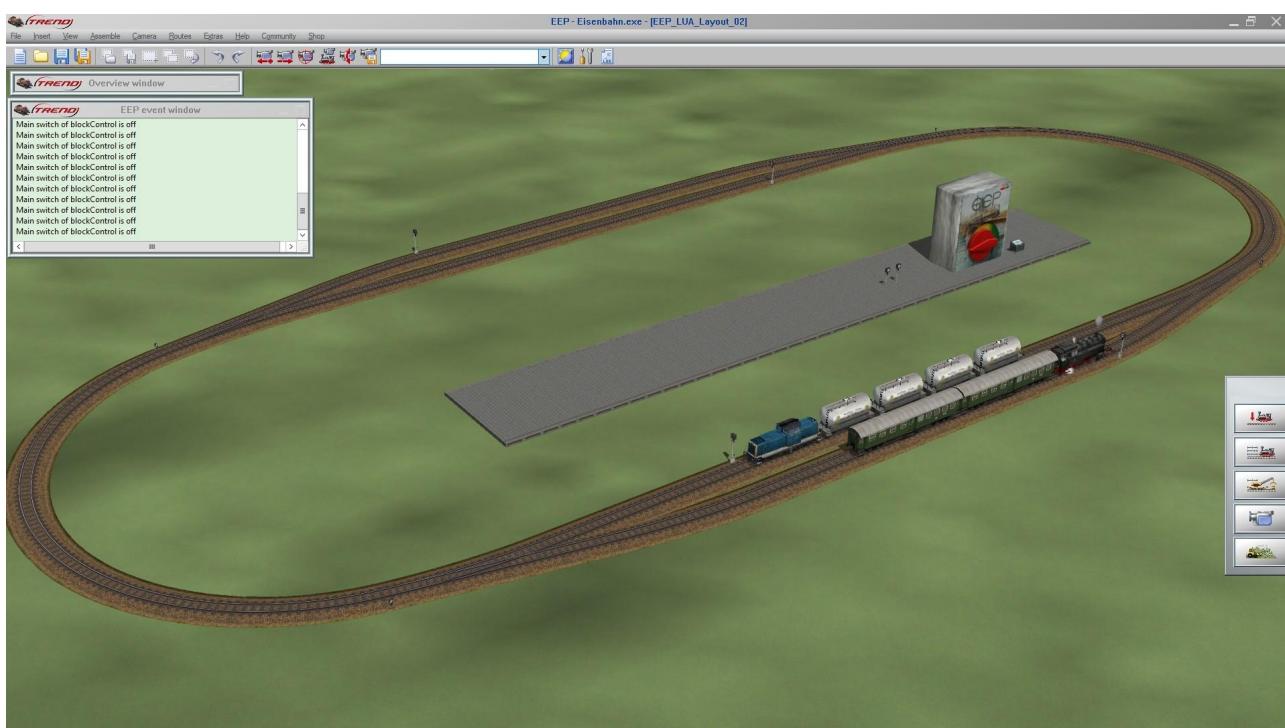
Die Streckentabelle teilt Lua mit, wie die Weichen bei der Fahrt von einem Block zum nächsten geschaltet werden müssen. In diesem Beispiel gibt es genau eine Weiche zwischen den Blöcken aller Routen. Es könnte aber auch keine oder mehrere Weichen zwischen den Blöcken liegen. Beispiele:

- `{13, 14, turn={ } }`, Die Blöcke 13 und 14 haben keine Weiche zwischen ihnen.
- `{23, 24, turn={14, 1, 5, 2, 6, 1} }`, In den Blöcken 23 und 24 befinden sich drei Weichen.

Folgen Sie diesem Link für eine [YouTube-Video zu Demo 1](#).

## EEP LUA ATC Demo 2

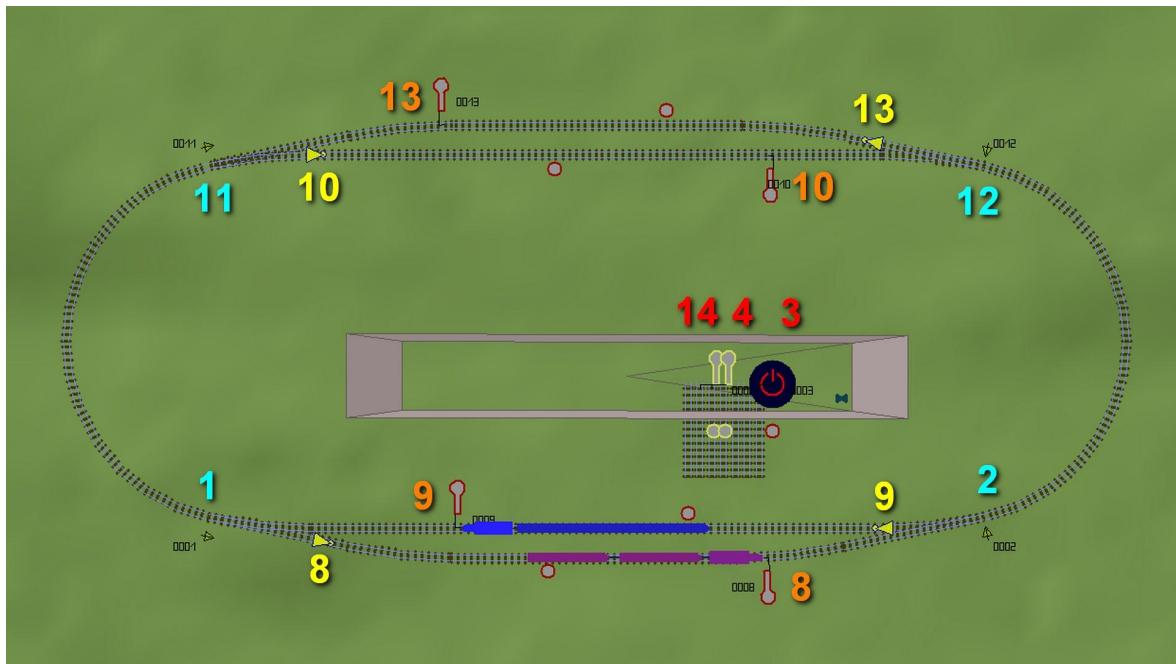
Fügen wir einen Block hinzu, um den "Bahnhof Nord" zu erstellen, einen zweiten Zug zu platzieren und Verkehr in zwei Richtungen zu haben.



Für dieses Layout legen wir 4 Blöcke fest, 2 am Bahnhof Nord und 2 am Bahnhof Süd. Wir fahren rechtsherum. Obwohl die Züge in entgegengesetzte Richtungen fahren, erfolgt der Verkehr in den Blöcken in eine Richtung. In Demo 3 werden wir sehen, wie man Blöcke mit Verkehr in zwei Richtungen erstellt.

Die 2D-Übersicht unten zeigt die Weichen (cyan), die Blocksignale (orange), die Kontakte (gelb) und die Ein- und Ausschaltsignale (rot). Da wir nun Gegenverkehr

haben, müssen wir darauf achten, dass die Blockeingangssensoren 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.



Die Konfiguration für dieses Layout lautet:

```

local main_signal = 3

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

local trains = {
  { name = "Steam", signal = 14, allowed = CCW },
  { name = "Blue",   signal =  4, allowed = CW  },
}

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 }},
}
  
```

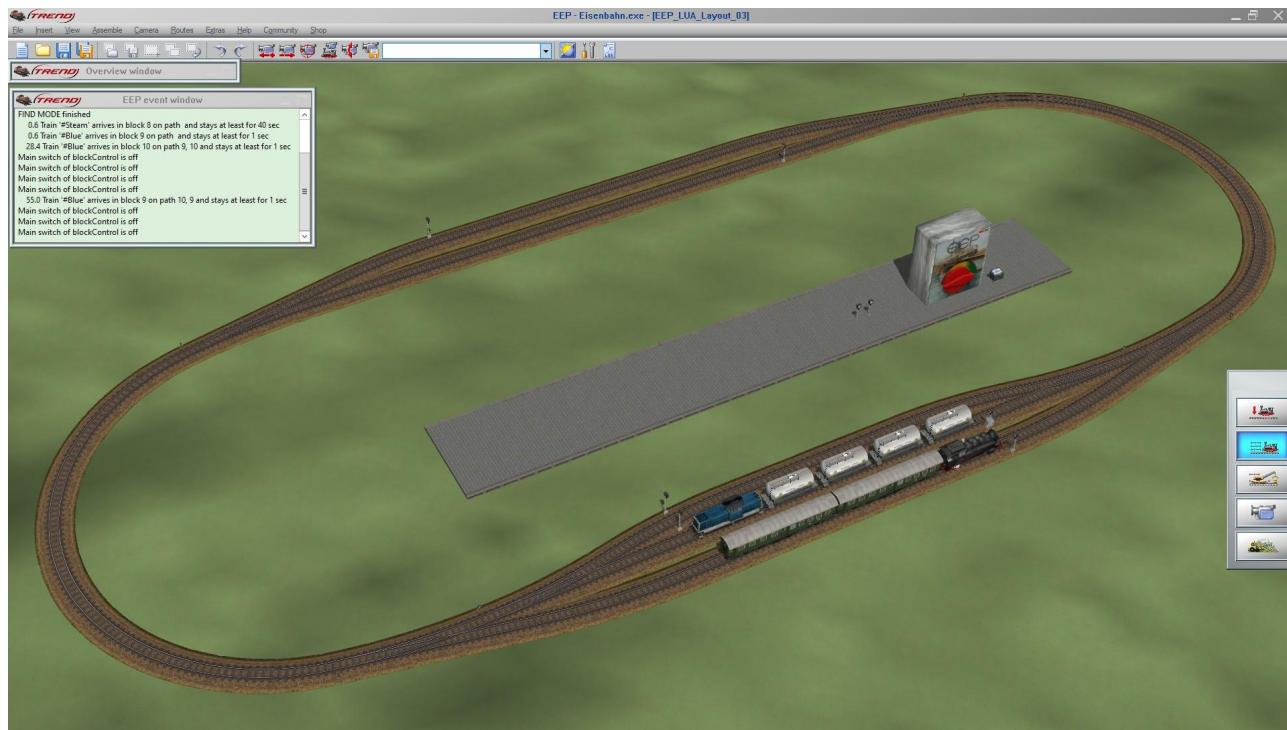
In diesem Beispiel werden die zulässigen Blöcke und die Wartezeiten in separaten Tabellen angegeben, eine für Züge "gegen den Uhrzeigersinn", eine für Züge "im Uhrzeigersinn". Dies sind nur Namen, Sie können jeden beliebigen Namen verwenden, "cargo\_trains", "ICE", solange dieselben Namen in der Tabelle "trains" verwendet werden.

Bei dieser Anlage mit nur zwei Zügen ist der Vorteil der Angabe von erlaubten Blöcken gering, aber bei einer größeren Anlage, bei der möglicherweise mehrere Züge auf denselben Blöcken erlaubt sind, vermeiden diese "erlaubten Blöcke"-Tabellen eine endlose Wiederholung identischer Tabellen für jeden Zug.

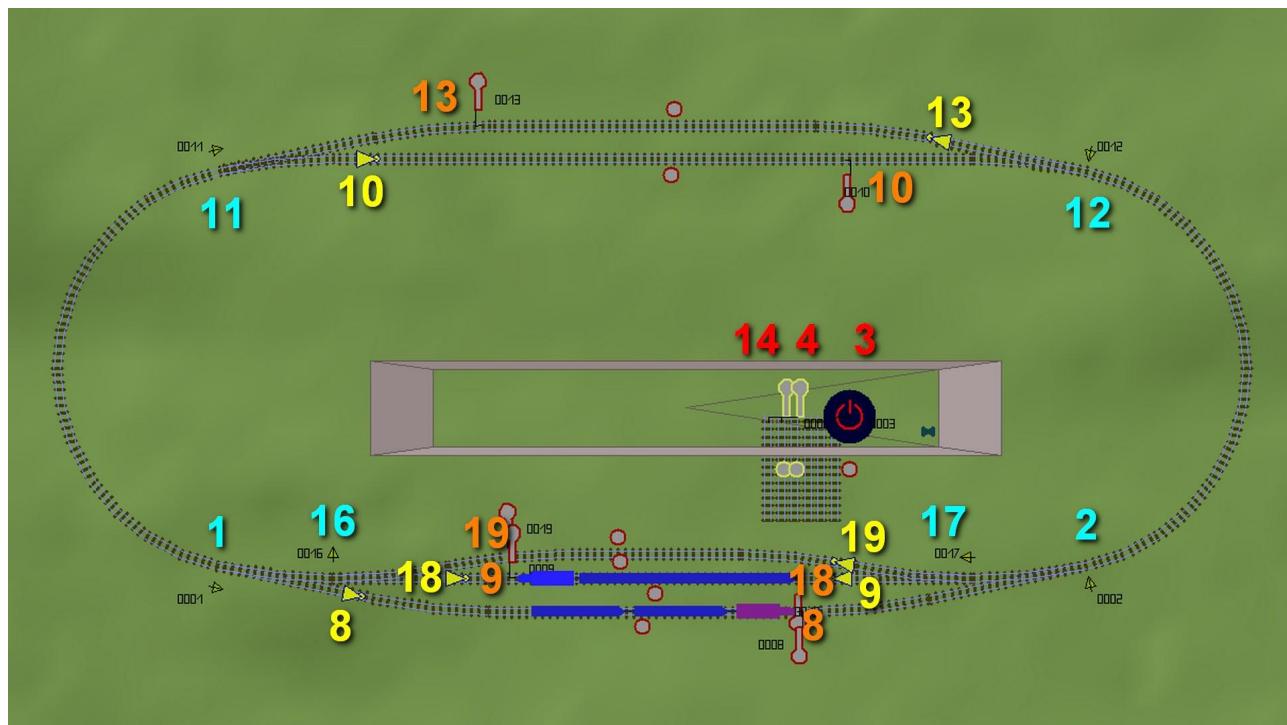
Folgen Sie diesem Link zum [YouTube-Video zu Demo 2](#).

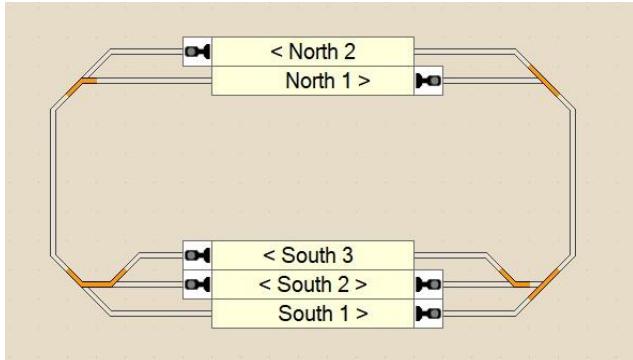
EEP\_LUA\_ATC\_Demo\_3

Der Bahnhof Süd wird um ein Mittelgleis erweitert, auf dem die Züge in beide Richtungen fahren sollen. Schauen wir uns an, wie man die Lua-Tabellen konfiguriert, um dies zu ermöglichen.



Zuerst müssen wir ein Signal an beiden Enden dieses Gleises platzieren, sie haben die Nummern 9 und 18 in der Abbildung unten. Wir brauchen auch einen Blockeingangskontakt an beiden Seiten. Dadurch wird dieser Block effektiv zu einem Paar überlappender "Zwillingsblöcke".





Sobald ein Zug einen dieser Zwillingsblöcke reserviert, muss Lua auch den anderen Block reservieren. Dasselbe gilt, wenn der Zug an seinem Zielblock ankommt und der Abfahrtsblock freigegeben werden kann, dann muss auch der Zwillingsblock freigegeben werden. Wir müssen also Lua über diese Zweiweg-Zwillingsblöcke informieren. Dies geschieht über die Tabelle `two_way_blocks`.

Der Konfigurationscode für dieses Layout lautet:

```

local main_signal = 3

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

local CW= {   -- allowed blocks for clockwise 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 = CCW, },
  { name = "Blue",   signal = 4, allowed = CW,  },
}

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 Zweiwegblöcken zu deklarieren gibt. Wenn es keine gibt, kann die Tabelle einfach weggelassen werden. Die Zwillingsblöcke werden immer paarweise deklariert. Wenn es mehrere Zweiwegblöcke gibt, könnte die Tabelle wie folgt aussehen<sup>12</sup>:

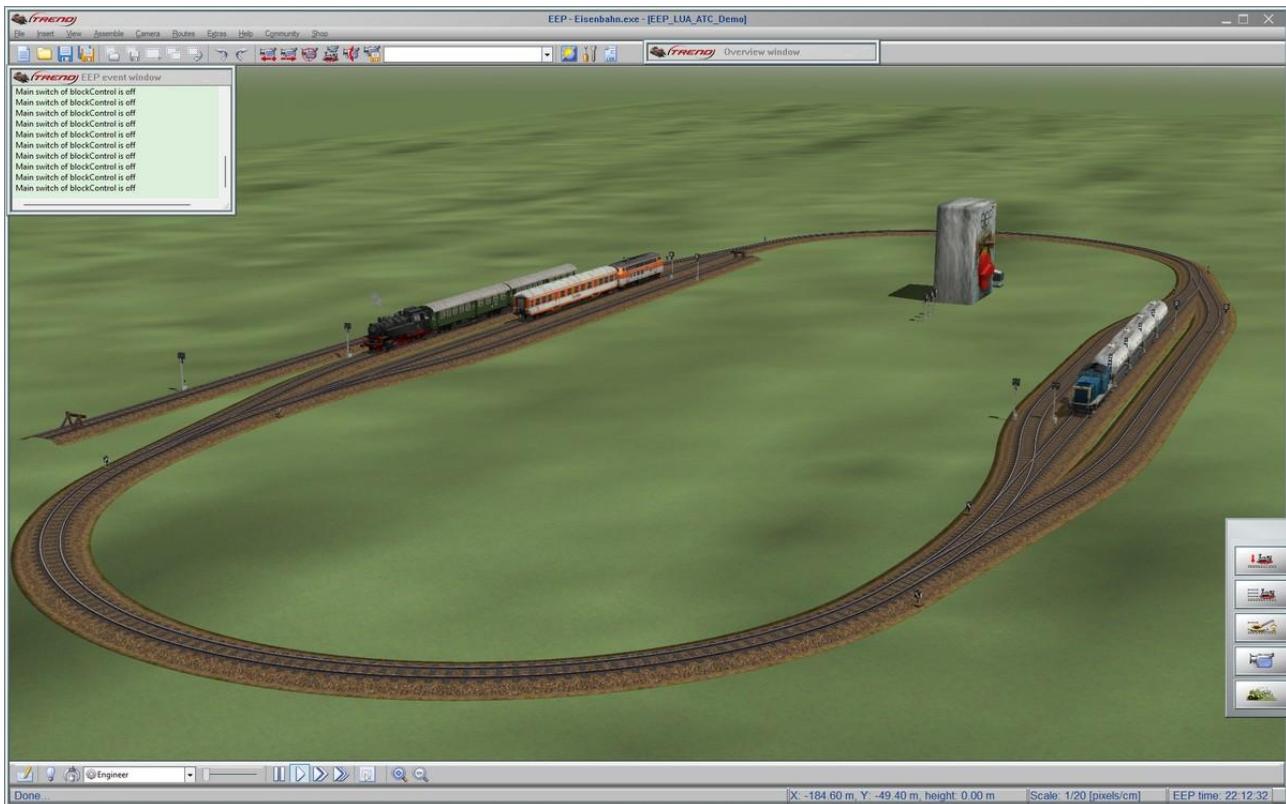
```
local two_way_blocks = { {82, 81}, {32, 33}, {74, 38}, }
```

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.

Folgen Sie diesem Link für eine [YouTube-Video zu Demo 3](#).

<sup>12</sup> Die Reihenfolge der Paare sowie die Reihenfolge der Blöcke innerhalb der Paare spielt keine Rolle.

## EEP\_LUA\_ATC\_Demo\_4



Dies ist das Layout, das in diesem Benutzerhandbuch in den Kapiteln 1 - 13 verwendet wurde, um zu demonstrieren, wie der Lua-Code automatisch generiert wird und wie man Tabellen mit zulässigen Blöcken erstellt.

Die Anlage enthält zwei Sackgassen und ein Gegengleis, um zu demonstrieren, wie diese in denLua-Tabellen, die die Anlage beschreiben, konfiguriert werden.

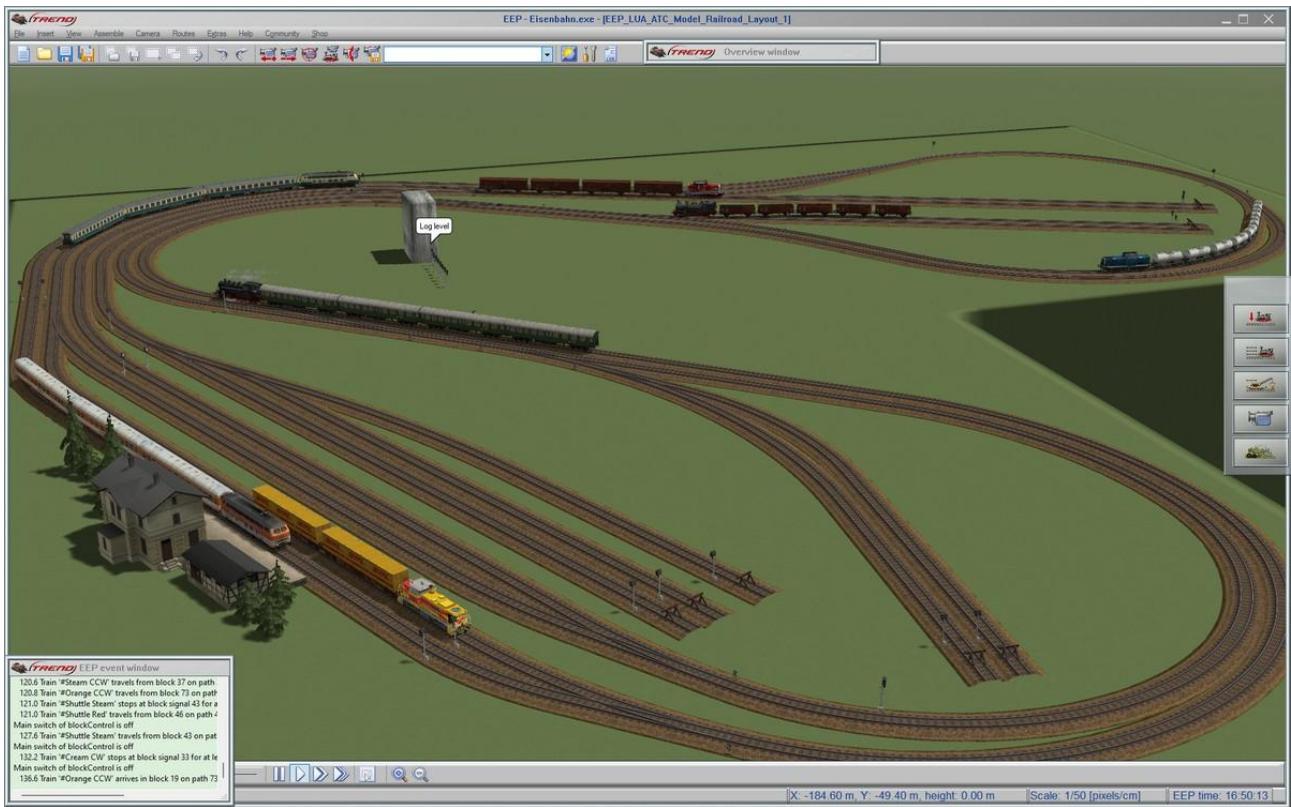
Drei Züge fahren auf dieser Anlage.

Folgen Sie diesen Links zu [YouTube-Video 4A zu Demo 4](#) und [YouTube-Video 4B zu Demo 4](#).

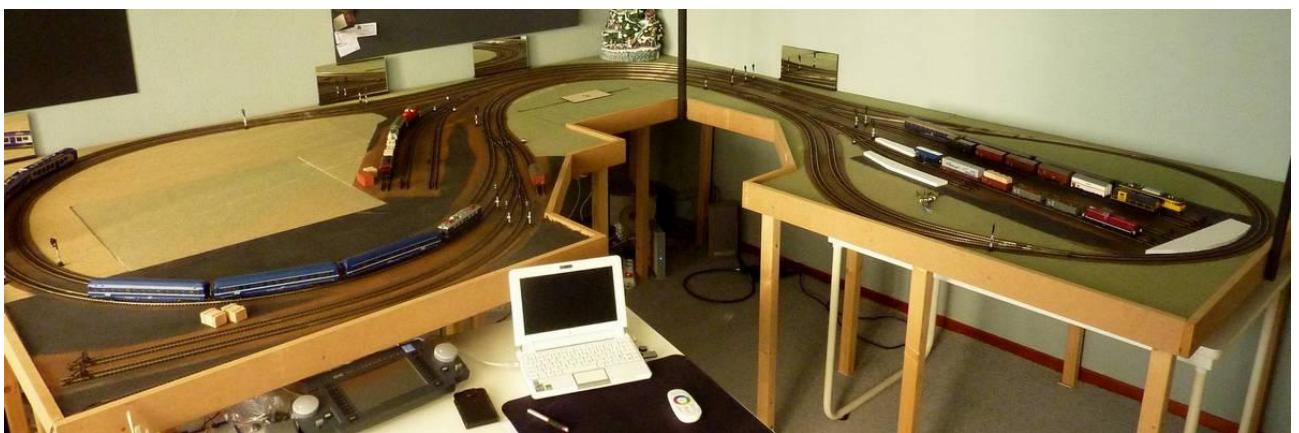
[Dieses YouTube-Video 7](#) demonstriert alle Schritte, die in diesem Benutzerhandbuch beschrieben sind, einschließlich der Zugumkehr ohne Verwendung von Kontakten für die Umkehrung.

[Dieses YouTube-Video 8](#) zeigt, wie man Züge in einem beliebigen Block umkehren kann, nicht unbedingt in einer Sackgasse.

## EEP\_LUA\_ATC\_Model\_Railway\_Layout\_1

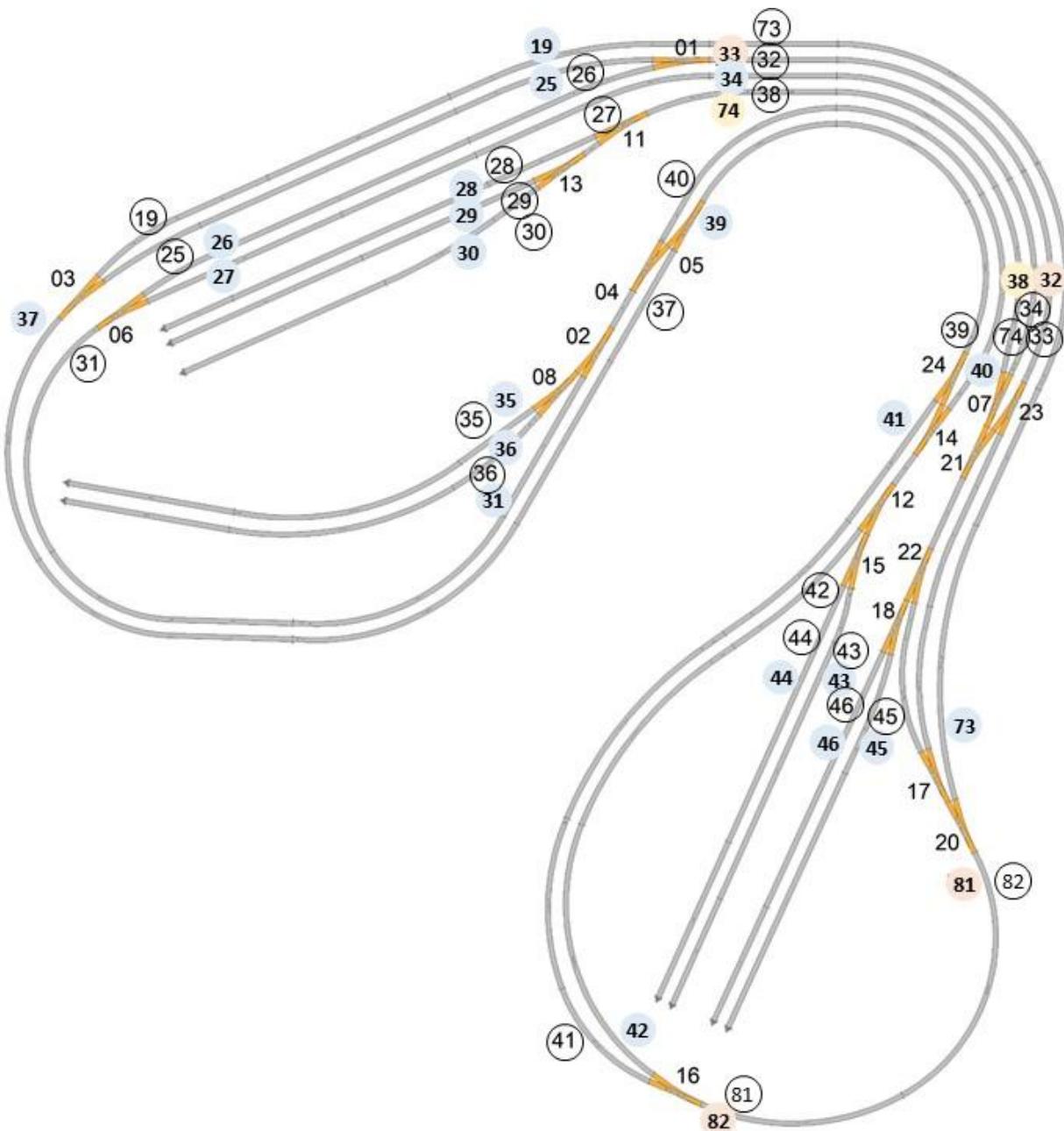


Es handelt sich um eine EEP-Simulation einer bestehenden Modelleisenbahnanlage, auf der sieben Züge verkehren.

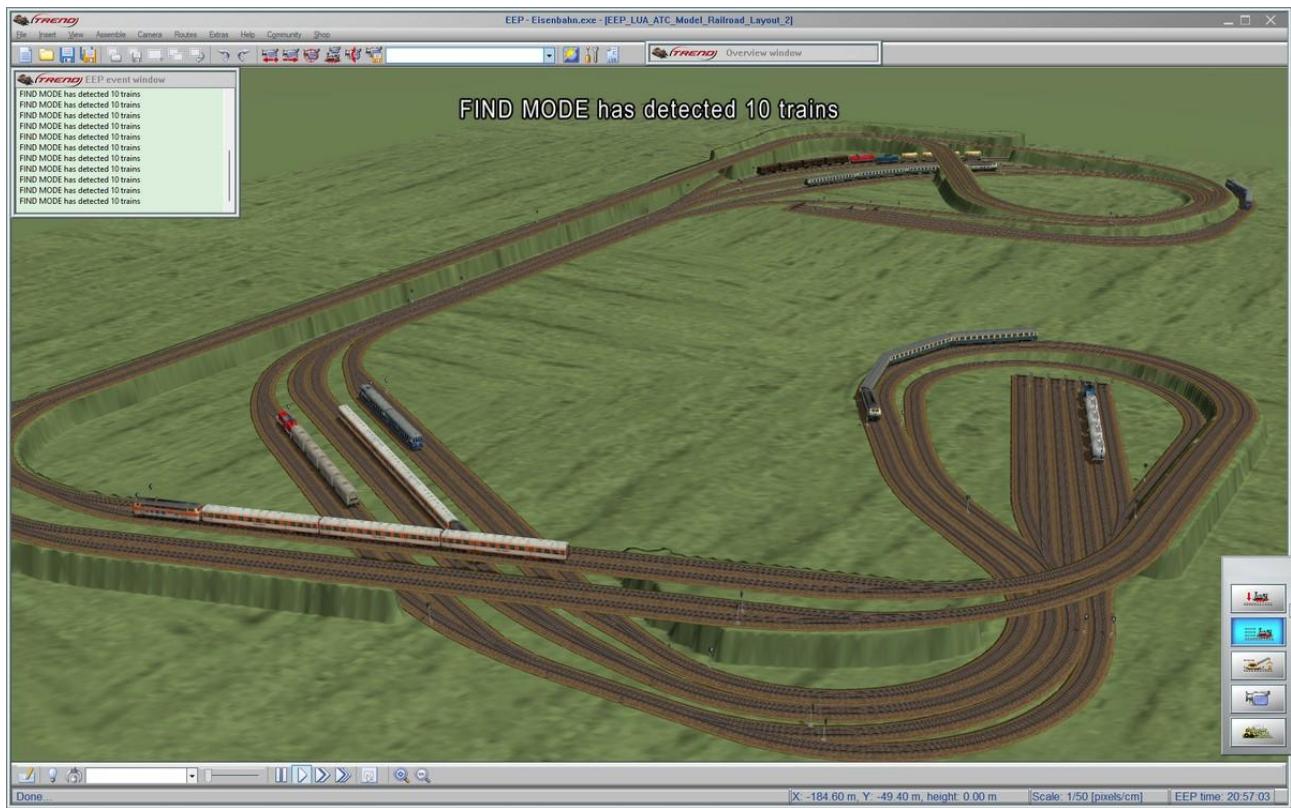


Folgen Sie diesem Link zum [YouTube-Video 5 zu Model Railway Layout 1](#).

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.



## EEP LUA ATC Model Railway Layout 2

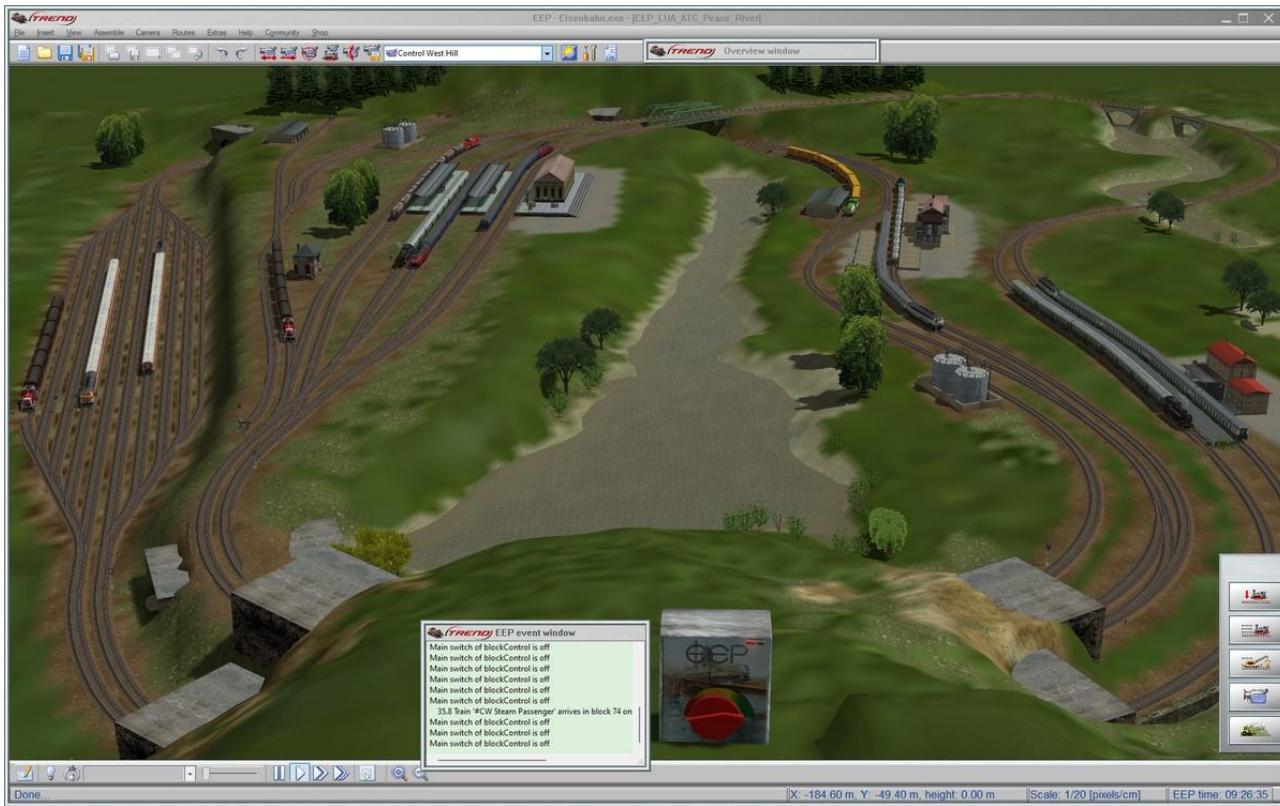


Auch dies ist eine Simulation einer bestehenden Modelleisenbahnanlage, auf der zehn Züge fahren.



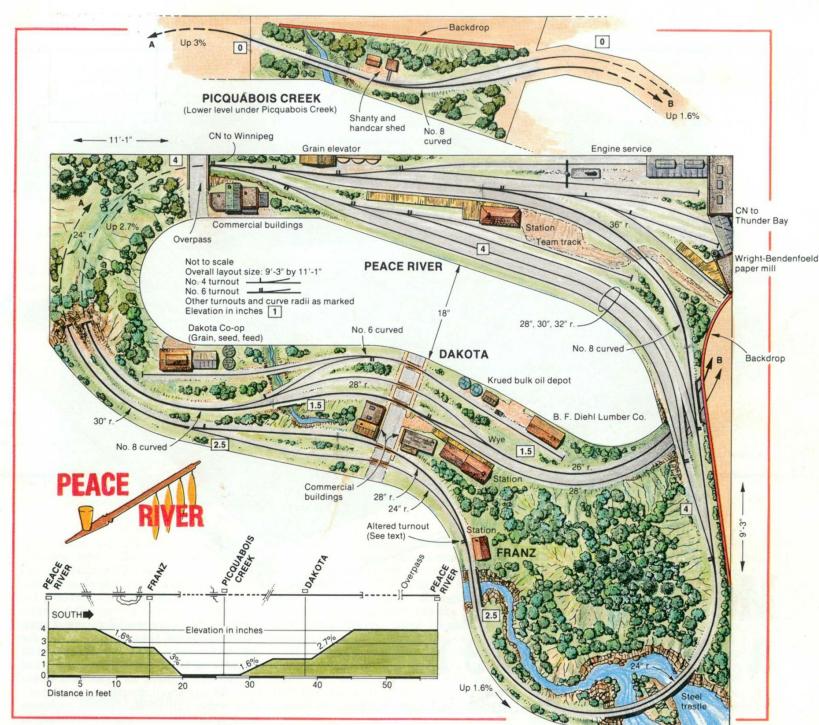
Folgen Sie diesem Link für eine [YouTube-Video 6 zu Model Railway Layout 2](#).

## EEP LUA\_ATC\_Peace\_River



Diese EEP-Anlage basiert grob auf einem im Internet gefundenen Modellbahn Entwurf. Dreizehn Züge fahren auf dieser Anlage herum.

Folgen Sie diesem Link zum [YouTube-Video 10 zu Model Railway Peace River](#).



Sie finden auch eine erweiterte Demo, die zeigt, wie Sie eine Anlage anpassen und erweitern können:

Das Original-Depot verwendet 2\*5 Gleise, um Züge in verschiedene Richtungen zu steuern. Warum nicht alle 10 Gleise für die Verwaltung von Zügen in beide Richtungen verwenden?

Für die erweiterte Demo fügten wir 4 Weichen hinzu, um eine Kreuzung zu schaffen, und 5 Blocksignale und Kontakte, um Zweiwege-Blöcke auf beiden Seiten des Depots zu bauen, und dann generierten wir den Lua-Code erneut. Dann haben wir die "allowed"-Tabellen für Züge erweitert, um die neuen Blöcke zu benutzen und zusätzliche Weichen zu den Strecken hinzugefügt, um beide Kreuzungen zu schützen. Funktioniert prima!

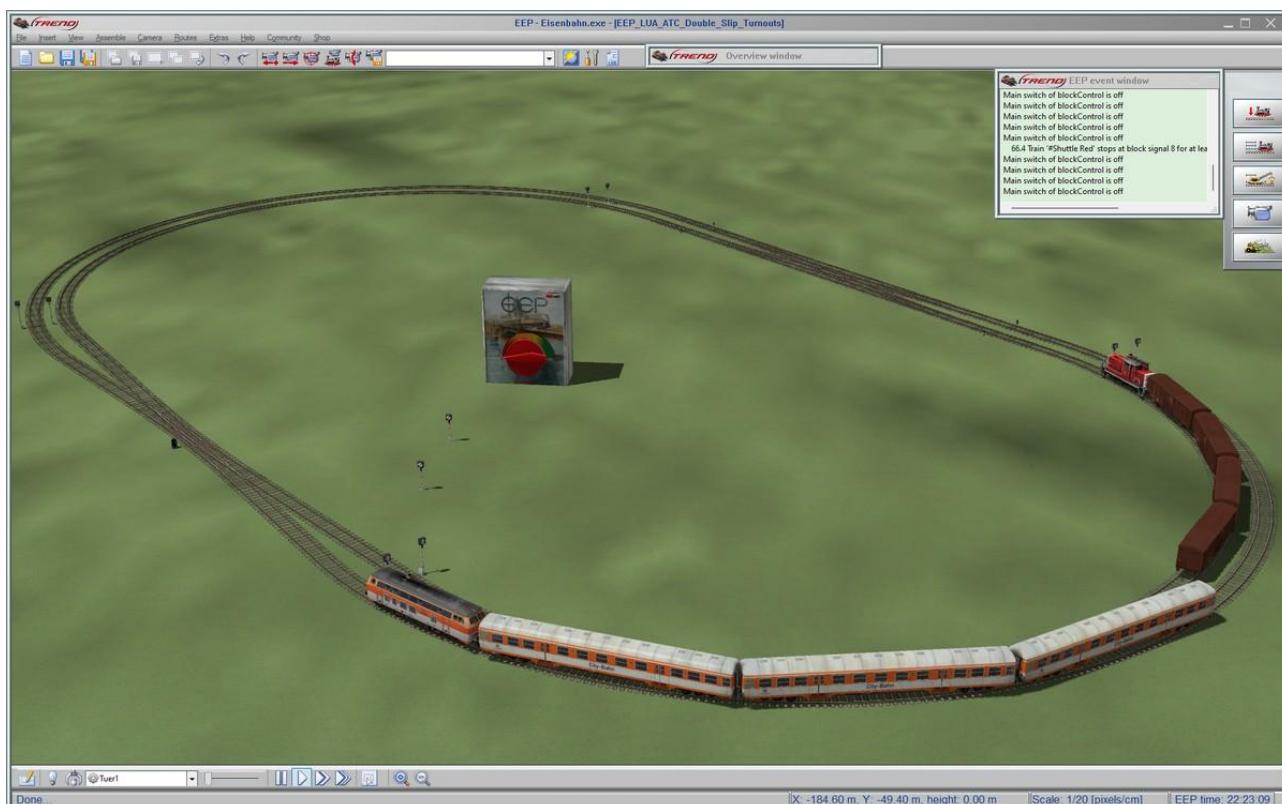
Zusätzlich haben wir dem Zug "#North Red Cargo" erlaubt, den Pendelbahnhof zu verlassen und eine komplette Fahrt gegen den Uhrzeigersinn zu machen. Aus diesem Grund haben wir einen ähnlichen Zug "#North Blue Cargo" hinzugefügt, um mehr Verkehr auf dem Pendelbahnhof zu haben.

## **EEP\_LUA\_ATC\_Double\_Slip\_Turnouts**

Doppelkreuzungsweichen (DKW) gibt es in zwei Varianten, die eine besondere Behandlung erfordern:

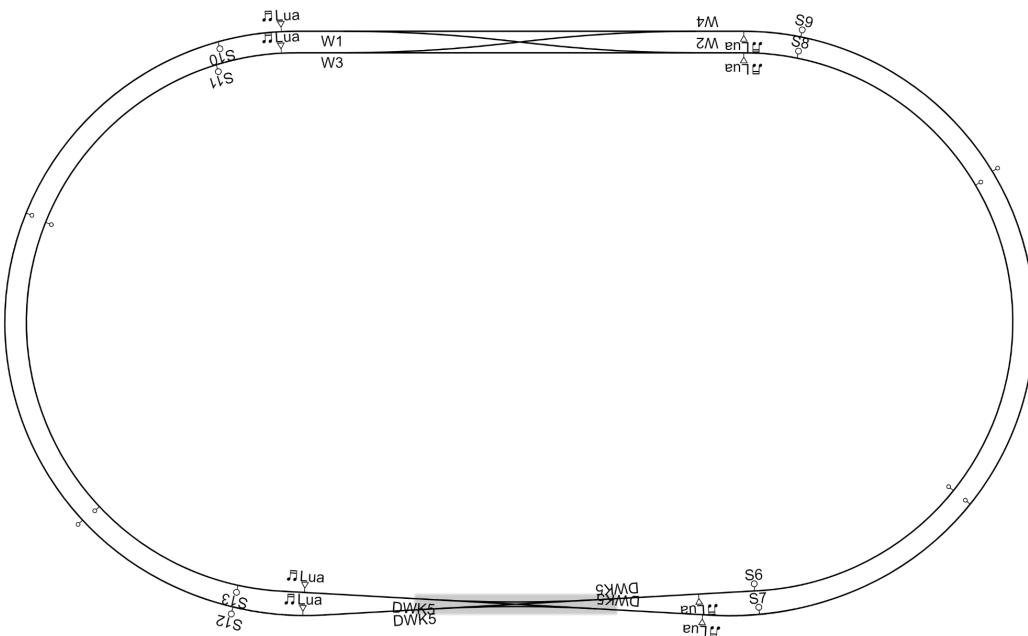
- a) Sie können eine DKW, die aus 4 Weichen besteht, manuell oder mit Hilfe der Vorlagen in EEP erstellen. Sie sollten die Kreuzung dieser DKW sichern, indem Sie die direkten Routen mit einer Weiche aus dem anderen Teil der DKW erweitern.
- b) Sie können eine Gleisobjekt-DKW verwenden, das aus einer Weiche mit 4 Positionen besteht. Sie können die Routen wie gewohnt definieren.

Diese Demo-Anlage zeigt beide Varianten, oben eine 4-Weichen-DKW und unten eine Gleisobjekt-DKW:



Der Lua-Code dieser Anlage demonstriert die verschiedenen Möglichkeiten zum Bau von Doppelkreuzungsweichen und zeigt wie die Kreuzungen zur Vermeidung von Kollisionen geschützt werden können. Dies ist eine Erweiterung zu den Optionen die in Kapitel [Kollisionen an Kreuzungen vermeiden](#) beschrieben sind.

Hier ist das entsprechende Bild aus dem [Gleisplan-Programm](#):



Für die aus 4 Weichen aufgebaute Doppelkreuzungsweiche wird eine spezielle Konfiguration über die Tabelle [crossings\\_protection](#) verwendet. Dadurch werden die erforderlichen Erweiterungen der Stecken automatisch erstellt.

Für die Gleisobjekt-DKW ist das leider nicht möglich. Hierfür müssen die Stecken manuell erstellt werden.

Der Konfigurationscode lautet:

```
-- Allowed blocks with wait time
local passenger = { [6]=1, [7]=1, [8]=1, [9]=1, [10]=1, [11]=1, [12]=1, [13]=1, }
local cargo      = { [6]=20, [7]=30, [8]=20, [9]=20, [10]=20, [11]=30, [12]=30, [13]=30, }

local trains = {
{ name="#Orange",      signal=14, allowed=passenger },
{ name="#Shuttle Red", signal=15, allowed=cargo     },
}

local main_signal = 16

local block_signals = { 6, 7, 8, 9, 10, 11, 12, 13, }

local two_way_blocks = { { 6, 8 }, { 7, 9 }, { 10, 12 }, { 11, 13 }, }

-- Crossings protection: Pairs or triples of coupled turnouts
-- Automatically add the last turnout to the routes which contain the preceding turnouts
local crossings_protection = {
{ 1, 2, 3 },
}
```

```

}

local routes = {
-- CCW via DST using 4 turnouts
{ 8, 12, turn={{ 2,2, 1,2, },}, -- crossing
{ 8, 13, turn={{ 2,1, 3,1, },}, -- straight
{ 9, 12, turn={{ 4,1, 1,1, },}, -- straight
{ 9, 13, turn={{ 4,2, 3,2, },}, -- crossing

-- CW via DST using 4 turnouts
{ 10, 6, turn={{ 1,2, 2,2, },}, -- crossing
{ 10, 7, turn={{ 1,1, 4,1, },}, -- straight
{ 11, 6, turn={{ 3,1, 2,1, },}, -- straight
{ 11, 7, turn={{ 3,2, 4,2, },}, -- crossing

-- CCW via track object DST (manually created)
{ 13, 8, turn={{ 5,1 }}, -- left/left
{ 13, 9, turn={{ 5,2 }}, -- left/right
{ 12, 9, turn={{ 5,3 }}, -- right/right
{ 12, 8, turn={{ 5,4 }}, -- right/left
}
[...]
blockControl.init({
    logLevel      = 1,
    trains        = trains,
    blockSignals   = block_signals,
    twoWayBlocks   = two_way_blocks,
    routes         = routes,
    paths          = anti_deadlock_paths,
    crossings      = crossings_protection, -- Coupled turnouts to protect crossings
[...]

```

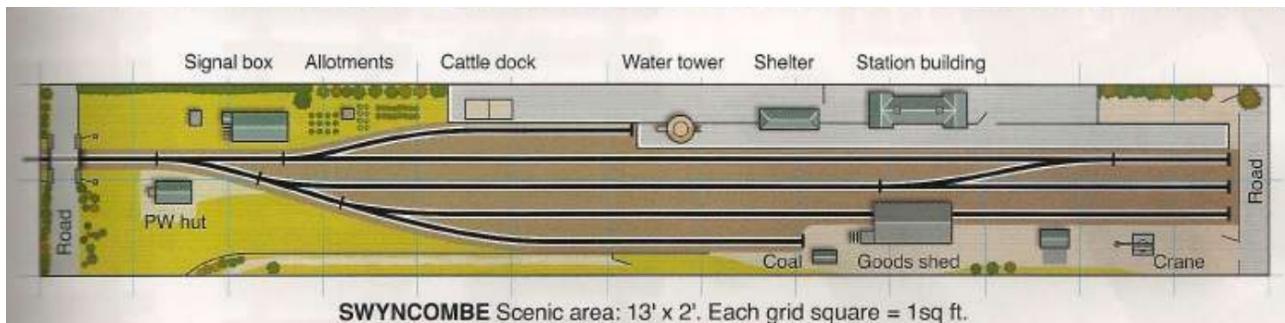
Alle Züge haben unterschiedliche Geschwindigkeiten und können mit unterschiedlichen Wartezeiten überall hinfahren. Auf diese Weise entstehen unterschiedliche Begegnungen, die Überhol- und Wartesituationen zeigen.

Die direkten Routen der 4-Weichen-DKW werden zur Sicherung der Kreuzung um eine Weiche aus dem anderen Teil der Kreuzung erweitert. Der konstante Abstand zwischen den parallelen Gleisen dieser DWK erlaubt es, dies für die geraden Abschnitte wegzulassen.

Die Routen der Gleisobjekt-DKW zeigen die 4 verschiedenen Positionen.

Tipp: Aktivieren Sie im Lua-Skript-Editor die Checkbox zu Weichenereignissen, um die Positionsnummern dieser Weiche in Abhängigkeit von den Positionen anzuzeigen. Auf diese Weise ist es recht einfach, die Routen für diese DKW zu definieren.

## **EEP\_LUA\_ATC\_Swyncombe**



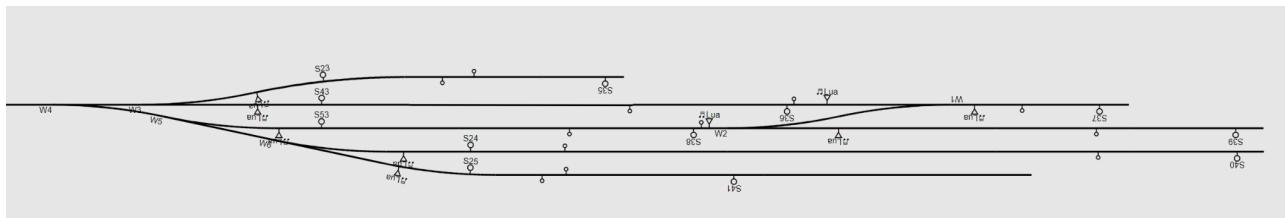
Swyncombe ist eine Modellbahnanlage im Bücherregal, die in der Zeitschrift British Railway Modelling, Juni 2013, beschrieben wurde. Es ist ein Sackgassenbahnhof mit nur einem Gleis für die Ein- und Ausfahrt<sup>13</sup>.

Wir haben ein 7-gleisiges Depot hinzugefügt, in dem die Züge rückwärts fahren können, aber sie können auch weiterfahren und eine Schleife zurück zum Bahnhof fahren.

Auf diese Weise fährt jeder der 12 Züge nach dem Zufallsprinzip entweder vorwärts oder rückwärts in den Bahnhof ein, was für maximale Abwechslung sorgt.

Folgen Sie diesem Link zum [YouTube Video 11 on Model Railway Swyncombe](#).

Hier ist das entsprechende Bild aus dem Gleisplan-Programm:



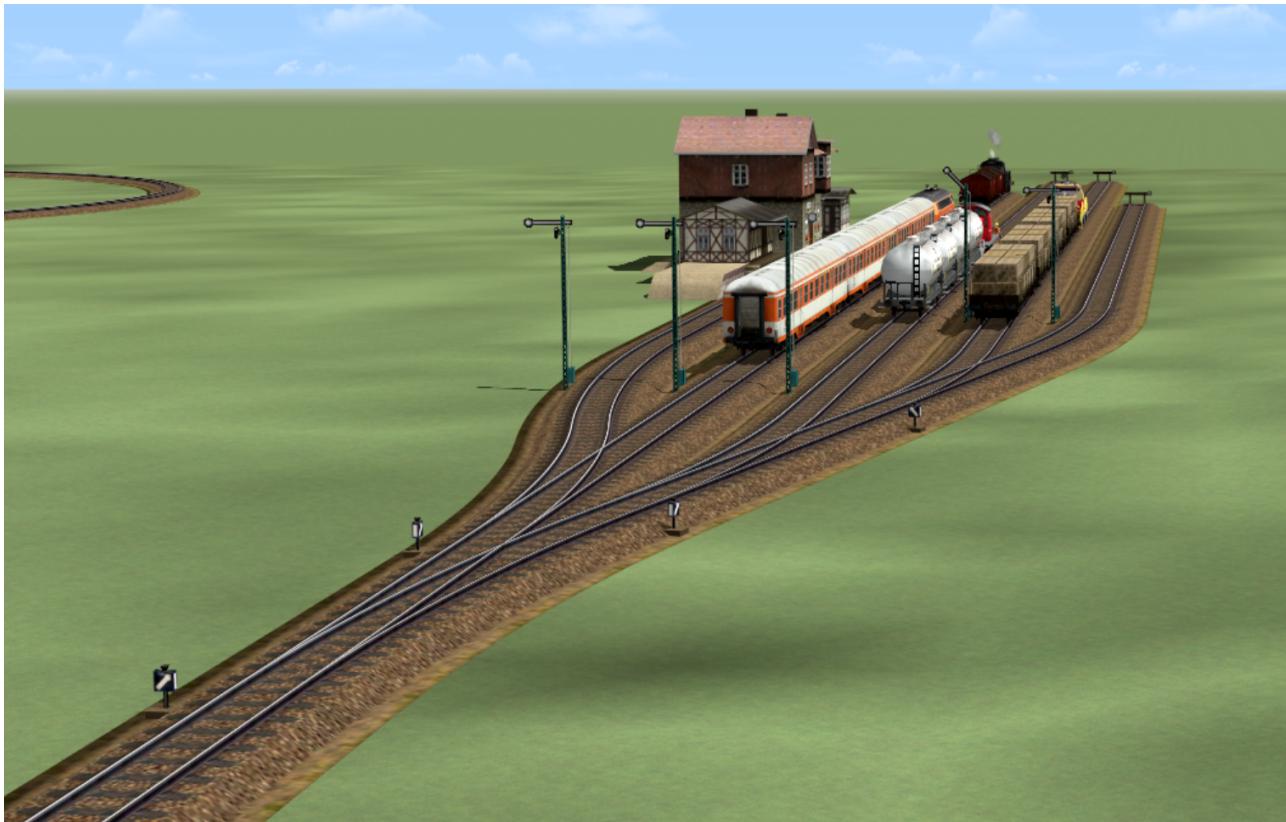
Kurze Züge können überall im Bahnhof fahren, Züge mittlerer Länge und lange Züge haben weniger Möglichkeiten. Alle Züge haben vollen Zugang zum Betriebshof. Dazu definieren wir 4 Tabellen mit erlaubten Blöcken: kurz, mittel, lang, Depot.

Alle Depotgleise haben eine Route zurück zum Bahnhof in beide Richtungen, eine mit `reverse=true`, eine mit Vorwärtsfahrt. Zufällige Wartezeiten werden durch die Definition von `rt={10, 40}` verwendet, die wir nun in den Tabellen der erlaubten Blöcke verwenden können.

<sup>13</sup> Quelle: Phil Giesebe, Designing Small Shelf Layouts for Operations  
[http://thoroughbredlimited2015.yolasite.com/resources/Clinic\\_Presentations/Giesebe%20Designing%20small%20shelf%20layouts%20for%20operating%20fun%202015-2.pdf](http://thoroughbredlimited2015.yolasite.com/resources/Clinic_Presentations/Giesebe%20Designing%20small%20shelf%20layouts%20for%20operating%20fun%202015-2.pdf)

## Bilder der schönen Landschaft

<http://www.gwr.org.uk/layoutsswvnccombe1.html>



Der interessante Teil des Konfigurationscodes lautet:

```

local rt = { 10, 40 }          -- Random (minimum) waiting time for the depot
                               -- between 10 and 40 seconds
local depot = {
  [44]=rt, [45]=rt, [46]=rt, [47]=rt, [48]=rt, [49]=rt, [50]=rt, -- depot
  [42]=1,  [51]=20, [52]=20,           -- access to depot
}

local short = {
  [35]=20, [36]=1, [37]=120, [38]=20, [39]=120, [41]=20, [43]=1, [53]=1, }
local mid   = {
  [35]=20, [36]=20,                 [38]=20,                   [41]=20, }
local long  = {
  [36]=20,                  [38]=20,                   [40]=20, }

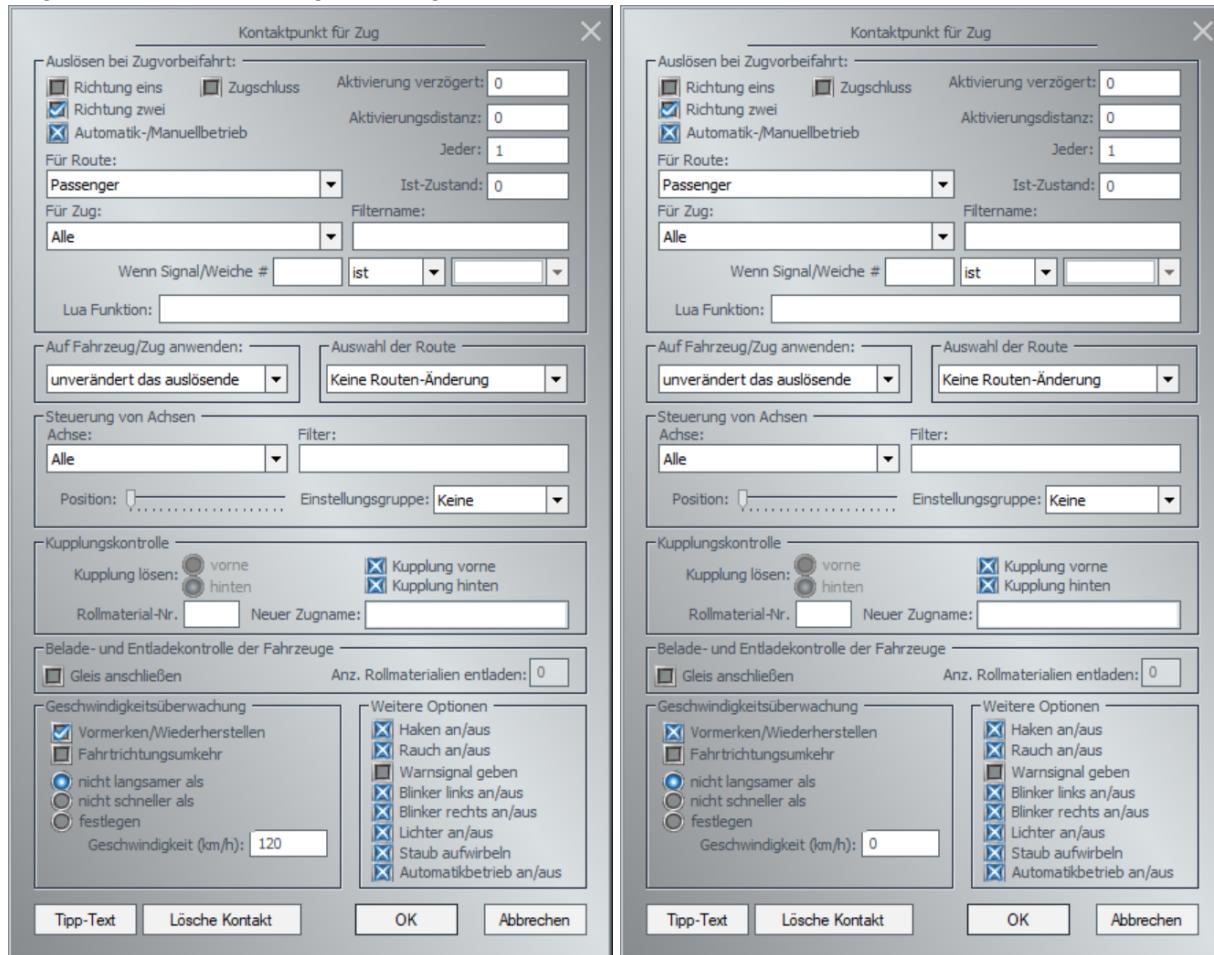
local trains = {
  { name = "#Long1",   speed = 30, allowed = { long,   depot, } },
  { name = "#Long2",   speed = 32, allowed = { long,   depot, } },
  { name = "#Mid1",    speed = 30, allowed = { mid,    depot, } },
  { name = "#Mid2",    speed = 32, allowed = { mid,    depot, } },
  { name = "#Short1",  speed = 30, allowed = { short,  depot, } },
  { name = "#Short2",  speed = 32, allowed = { short,  depot, } },
}

```

Die Demo zeigt auch eine Beispiellösung für Geschwindigkeitsänderungen. Auf der einen Seite haben Züge ihre spezifische Reisegeschwindigkeit - langsam für Güterzüge, schnell für Personenzüge - und auf der anderen Seite gibt es Beschränkungen in Bahnhöfen und auf Weichenabschnitten in Bezug auf eine maximale Geschwindigkeit. In einer EEP-Anlage definiert man solche Geschwindigkeitsänderungen mit Hilfe von Zugkontakten, und das funktioniert gut zusammen mit der Definition einer Geschwindigkeit in der Zugtabelle, die immer dann verwendet wird, wenn ein Zug seine Richtung wechselt.

In der Demo sind die Geschwindigkeiten in der Zugtabelle eher klein, um den Einschränkungen für Bahnhöfe zu entsprechen. Darüber hinaus erhöht ein Zugkontakt auf der langen Strecke vom Depot zum Bahnhof die Geschwindigkeit von Personenzügen bzw. stellt einen weiteren Zugkontakt die ursprüngliche Geschwindigkeit wieder her, indem die entsprechenden Kontrollkästchen aktiviert werden:

*Tipp-Text: "Mit dieser Funktion zeichnen Sie die augenblickliche Geschwindigkeit eines Fahrzeugs auf, um sie bei Bedarf wiederherzustellen. Die Aufzeichnung der Geschwindigkeit erfolgt, sobald das Kästchen mit einem Haken (✓) versehen ist. Ist das Kästchen mit einem X markiert, erfolgt die Wiederherstellung der vorgemerken Geschwindigkeit. Ein graues Feld bedeutet 'keine Aktion'."*



## EEP\_LUA\_ATC\_Depots

In den bisherigen Demos haben wir klassische Depots mit echten Gleisen verwendet.

Sie können aber auch ein oder mehrere EEP-Depots einbinden. Solche EEP-Depots werden nicht durch das Modul gesteuert. Sie können sie selbst verwalten, entweder über die Zeit, über Signale oder mit Lua.

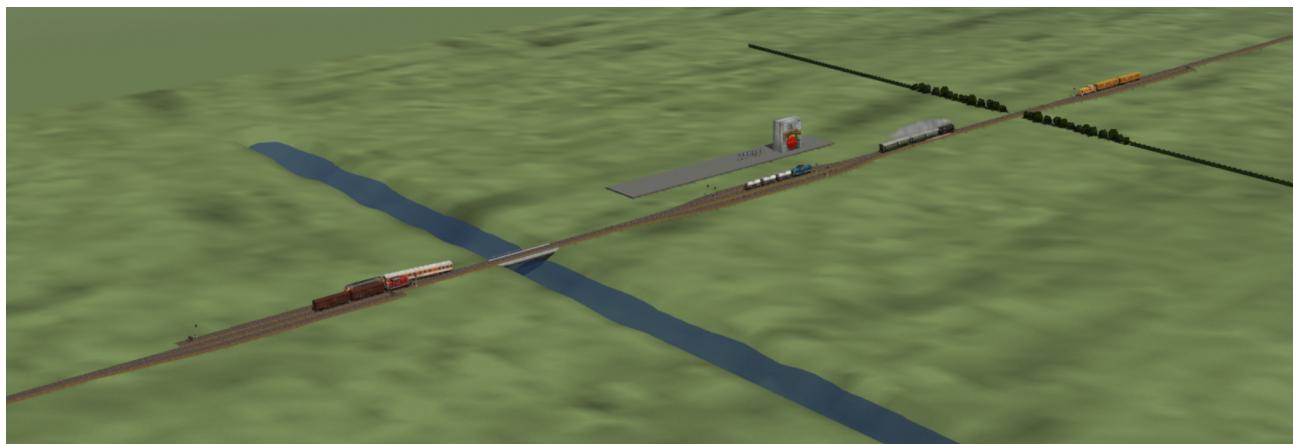
Dieses Demo-Layout demonstriert die Einbindung solcher EEP-Depots und erklärt die Schnittstelle zwischen Depotverwaltung und automatischer Blocksteuerung.

Die Grundidee für die Integration ist die folgende:

1. Züge, die auf ein Blocksignal zufahren, werden von einem Depot abgefangen, bevor sie das Signal erreichen können. Ein Kontakt mit einem Lua-Aufruf teilt dem Modul mit, dass der Zug diesen Block verlässt.
2. Züge, die von einem Depot freigegeben werden, fahren einfach auf die übliche Weise in einen Block ein ....
3. ... es sei denn, der Block ist noch von einem anderen Zug belegt. In diesem Fall wird der Zug stattdessen in ein Depot geschickt. Dies geschieht über Weichen und normale Kontakte, die diese Weichen steuern.

Die Demo-Anlage enthält einen kleinen Zwei-Wege-Bahnhof. Daher verwenden wir auch zwei Depots, um Züge in beide Richtungen ein- und ausfahren zu lassen. Die Züge können in beide Depots ein- und ausfahren, um einen zufälligen Verkehr zu erzeugen.

Die Demo zeigt ein Depot im Westen, einen Bahnhof in der Mitte und ein ähnliches Depot im Osten. Der Fluss und das Gebüsch trennen die Teile.



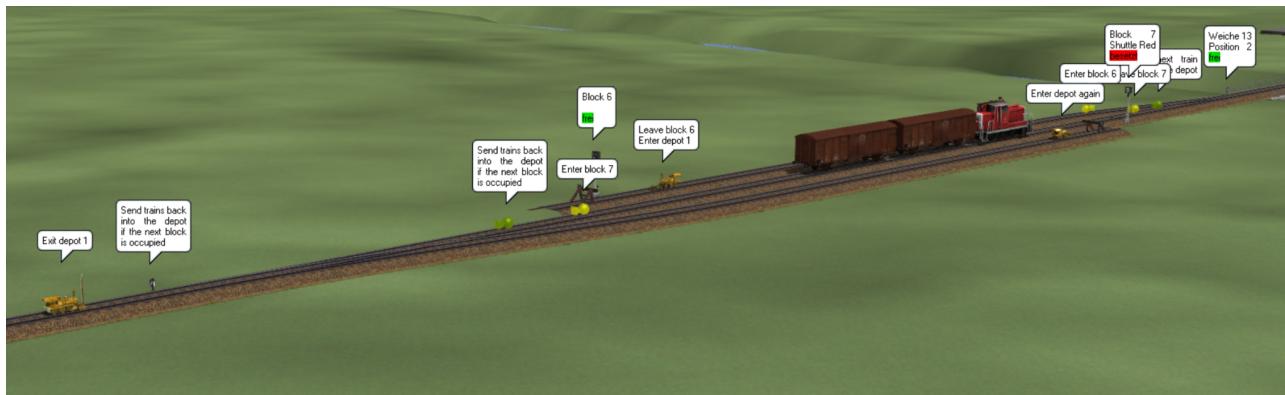
### Depot West

Die Züge kommen von rechts aus dem Bahnhof und benutzen das obere Gleis der Weiche 13, um in den Block 6 einzufahren. Sie erreichen das Blocksignal jedoch nicht, da sich davor der Einfahrtkontakt Depot befindet. Dieser Kontakt enthält die Lua-Funktion `leaveBlock`, die dem Modul mitteilt, dass der Zug den Block verlassen hat.

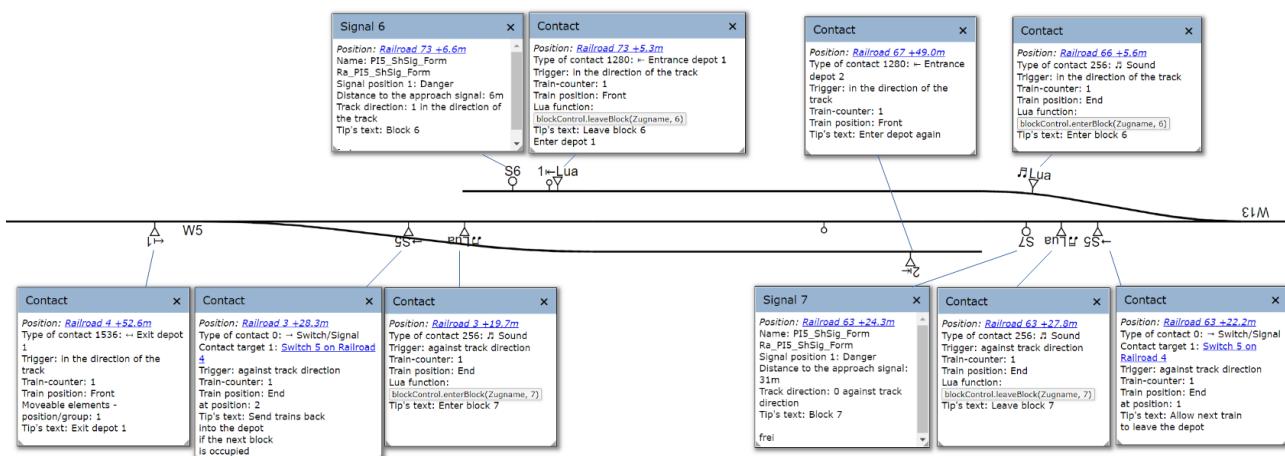
Für das Depot sehen wir folgende Elemente auf dem mittleren Gleis:

- Ein Depotausfahrtkontakt, der Züge in zufälligen Zeitabständen freigibt
- Die Weiche 5 zum Schutz des nächsten Blocks
- Ein Weichenkontakt, um die Weiche 5 auf "Wenden" zu stellen. Weitere Züge fahren dann über das untere Gleis und umgehen den Block.
- Ein Lua-Kontakt mit der Funktion `enterBlock` für Block 7
- Das Blocksignal 7, an dem die Züge anhalten können, wenn die vorausliegenden Blöcke nicht frei sind.
- Ein Weichenkontakt, um die Weiche 5 auf "gerade" zu stellen. Der nächste Zug fährt dann wieder in den Block ein.
- Ein Lua-Kontakt mit der Funktion `leaveBlock` für Block 7, um diesen Block so schnell wie möglich freizugeben.

Das untere Gleis enthält nur einen weiteren Depoteinfahrtkontakt, um den Zug zurück in ein Depot zu schicken.



### Anzeige im Gleisplan-Programm:



### Depot Ost

Das Depot Ost funktioniert im Grunde genauso wie das Depot West mit einer kleinen Änderung zur Optimierung der Ansicht.

Die Züge fahren vom Bahnhof aus von links ein und benutzen das untere Gleis hinter Weiche 15, um in Block 11 einzufahren. Sie erreichen das Blocksignal jedoch nicht, da sich davor der Kontakt "Einfahrt Depot" befindet. Dieser Kontakt enthält die Lua-Funktion `leaveBlock`, die dem Modul mitteilt, dass der Zug den Block verlassen hat.

Für das Depot sehen wir folgende Elemente auf dem geraden Gleis:

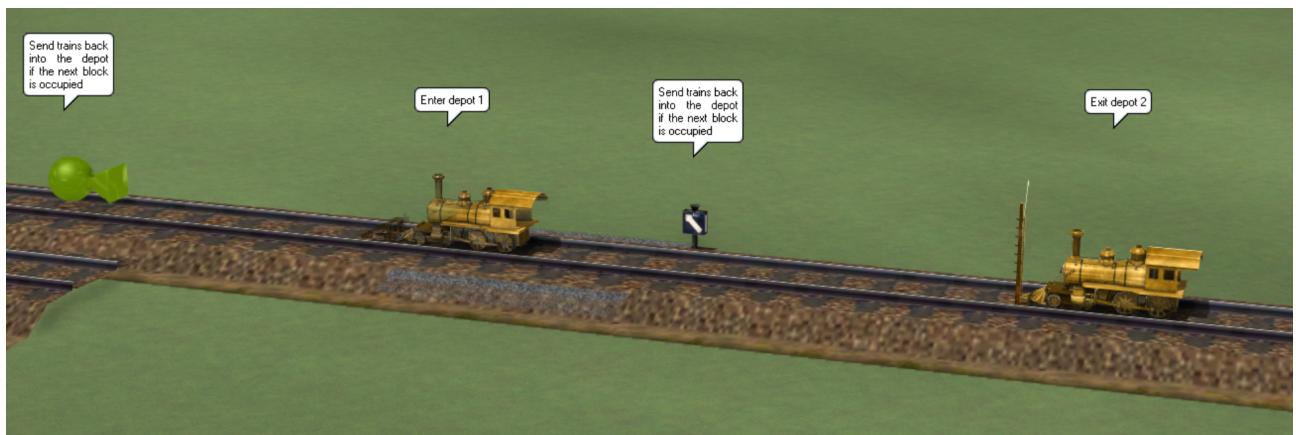
- Ein Depotausfahrtkontakt, der Züge in zufälligen Zeitabständen freigibt
- Die Weiche 10 zum Schutz des nächsten Blocks
- Ein Weichenkontakt, um die Weiche 10 auf "WendenAbzweig" zu stellen. Weitere Züge nehmen dann das andere Gleis, um den Block zu umgehen.
- Ein Lua-Kontakt mit der Funktion `enterBlock` für Block 12
- Das Blocksignal 12, an dem Züge anhalten können, wenn die vorausliegenden Blöcke nicht frei sind.
- Ein Weichenkontakt, um die Weiche 10 auf "gerade" zu stellen. Der nächste Zug fährt dann wieder in den Block ein.

- Ein Lua-Kontakt mit der Funktion leaveBlock für Block 12, um diesen Block so schnell wie möglich freizugeben.

Das zusätzliche Gleis, das die Züge zurück ins Depot schickt, ist grau, sehr kurz, gerade und liegt über dem anderen Gleis, was es fast unsichtbar macht.

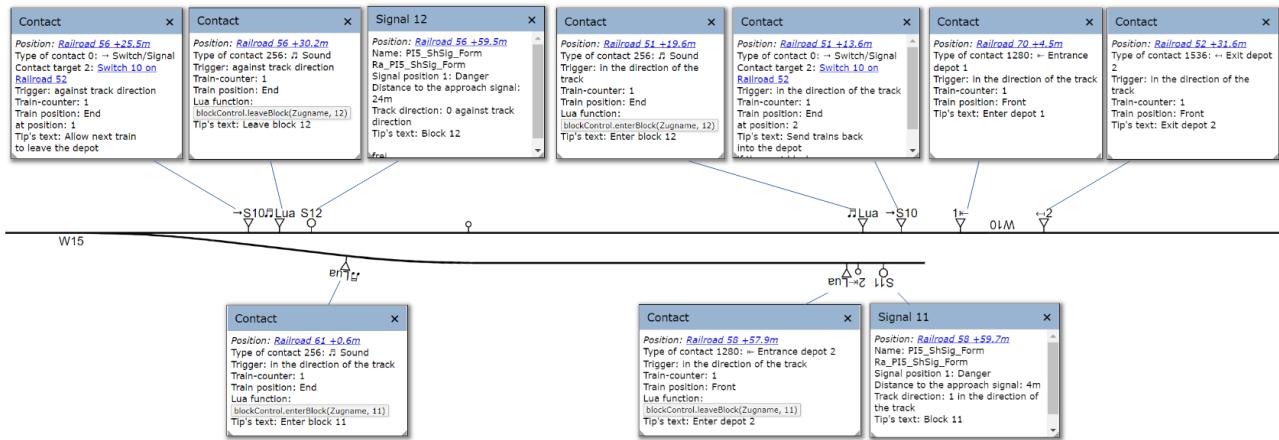


Hier sehen Sie das zusätzliche Gleis mit dem Depotkontakt auf dem regulären Gleis:



Tipp: Legen Sie den Einfahrtskontakt des Depots an eine andere Stelle, stellen Sie die Weiche ein und bewegen Sie den Kontakt auf das kurze Gleis, bis er das Ende erreicht.

Anzeige im [Gleisplan-Programm](#):



Schauen wir uns nun den entsprechenden Lua-Code an, der zunächst vom [Generierungsprogramm erzeugt](#) und dann nach Bedarf angepasst und [erweitert](#) wurde, um Fehlermeldungen im Ereignisfenster zu vermeiden:

```
-- Erlaubte Blöcke mit Wartezeit
local rt = { 40, 60 }
-- Der zusätzliche Eintrag mit dem Dummy-Block 0 verhindert einige Fehlermeldungen
-- hat aber keinen Einfluss auf den Verkehr auf diesem Layout.
local all = {
    [0] = 1,      -- (zusätzlich) Die Züge dürfen den kontrollierten Teil verlassen

    [8] = rt,     -- Station CCW
    [18] = rt,    -- Station CCW
    [9] = rt,     -- Station CW
    [19] = rt,    -- Station CW

    [6] = 99,     -- Entry Depot 1 (sicherstellen, dass Züge bis zum Depot fahren)
    [7] = 1,       -- Exit Depot 1
    [11] = 99,    -- Entry Depot 2 (sicherstellen, dass Züge bis zum Depot fahren)
    [12] = 1,     -- Exit Depot 2
}

local trains = {
    { name="#DB_216 beige", signal=22, slot=4, speed=70, allowed=all },
    { name="#Shuttle Red", signal=24, slot=6, speed=50, allowed=all },
    { name="#Steam", signal= 4, slot=3, speed=50, allowed=all },
    { name="#Blue", signal=30, slot=1, speed=70, allowed=all },
    { name="#Orange", signal=14, slot=2, speed=70, allowed=all },
    { name="#Shuttle Yellow", signal=23, slot=5, speed=50, allowed=all },
}
}

local main_signal = 3

local block_signals = { 6, 7, 8, 9, 11, 12, 18, 19, }
-- Blocks without any contact calling the corresponding enterBlock function: 8,
-- 8 signals use BLKSIGRED = 1, BLKSIGGRN = 2

local two_way_blocks = { { 9, 18 }, }

-- Die zusätzlichen Einträge mit dem Dummy-Block 0 verhindern einige Fehlermeldungen
-- haben aber keinen Einfluss auf den Verkehr auf dieser Anlage.
-- Einige Einträge sind nutzlos, weil Züge nie das Signal hinter einem
-- Depoteingangskontakt erreichen
local routes = {
    { 0, 7, }, -- (zusätzlich) Züge fahren in den kontrollierten Teil der Anlage ein
    { 0, 12, }, -- (zusätzlich) Züge fahren in den kontrollierten Teil der Anlage ein
    { 6, 8, turn={ 13,1, 1,2, }, reverse=true }, -- (nutzlos)
    { 6, 18, turn={ 13,1, 1,1, 2,1, }, reverse=true }, -- (nutzlos)
}
```

```
{ 7, 8, turn={ 13,2, 1,2, } },
{ 7, 18, turn={ 13,2, 1,1, 2,1, } },
{ 8, 11, turn={ 16,1, 15,2, } },
{ 9, 6, turn={ 2,1, 1,1, 13,1, } },
{ 11, 9, turn={ 15,2, 16,2, 17,1, }, reverse=true }, -- (nutzlos)
{ 11, 19, turn={ 15,2, 16,2, 17,2, }, reverse=true }, -- (nutzlos)
{ 12, 9, turn={ 15,1, 16,2, 17,1, } },
{ 12, 19, turn={ 15,1, 16,2, 17,2, } },
{ 18, 11, turn={ 17,1, 16,2, 15,2, } },
{ 19, 6, turn={ 2,2, 1,1, 13,1, } },
}
```

#### Änderungen am generierten Lua-Code:

- Generierte Routen, die von den Blocksignalen hinter den Eingangskontakten der Depots ausgehen, werden nie verwendet und sind daher nutzlos. Sie können solche Routen auskommentieren.
- Wenn Züge die Depots verlassen, tauchen sie plötzlich im kontrollierten Teil der Anlage auf. Das Modul erzeugt Log-Meldungen über solche unerwarteten Ereignisse. Sie können diese Meldungen unterdrücken, indem Sie zusätzliche Fahrstraßen hinzufügen, die mit dem Dummy-Block 0 beginnen bzw. dort enden.
- Wenn Sie solche Fahrstraßen für den Dummy-Block 0 hinzufügen, können Sie auch die allowed-Tabellen für die Züge für diesen Dummy-Block 0 erweitern. Andernfalls würden Sie einige Protokollmeldungen zu diesem Dummy-Block 0 erhalten.

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

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

1. Platzieren von Blocksignalen und optional der 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

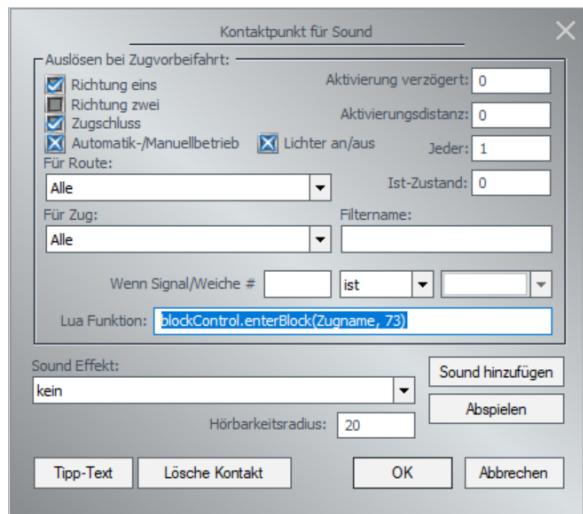
Jedes Mal, wenn Sie weitere Blocksignale hinzufügen, müssen Sie diese Reihenfolge für diese neuen Blöcke erneut einhalten.

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 einfügen (in diesem Fall ist es notwendig, eine globale Variable blockControl zu verwenden, daher deklarieren Sie sie nicht als lokal)<sup>14</sup>:

```
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)<sup>15</sup>:

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

<sup>14</sup> In addition you can use the option BetterContacts = true, in the init call of module blockControl to prevent generating global functions for this module.

<sup>15</sup> The module "BetterContacts" offers an additional option to change the variable name Zugname as described in the documentation of the module.

## Überblick, wie Lua automatischen Zugverkehr erzeugt

Wenn ein Zug in einen Block einfährt, überfährt er den Einfahrtsensor, 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 einfach 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.<sup>16</sup>
  - 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 werden entsprechend den definierten Routen geschaltet.
  - d. Die Geschwindigkeit des Zuges wird auf die umgekehrte Richtung gesetzt wenn der erste Teil des Pfades eine richtungsumkehrende Route ist.

## Fehlersuche

### Allgemeine Tipps

---

<sup>16</sup> Es ist möglich, den vorherigen Block freizugeben, sobald der Zug diesen Block verlässt. Um das Programm über dieses Ereignis zu informieren, müssen Sie einen zusätzlichen Kontakt hinter dem Blocksignal platzieren. Verwenden Sie diesen Kontakt, um die Funktion `blockControl.leaveBlock_nn` aufzurufen.

- Während des "Zugsuche-Modus" sollten alle Blocksignale "rot" anzeigen und alle Züge sollten an diesen Signalen halten. Im Modus "Zugsuche" sehen Sie die Signalstellung auch im TippText. Verwenden Sie dies, um zu überprüfen, ob dieser Wert mit dem Parameter **BLKSIGRED** übereinstimmt.
- Tabelle `blockSignals`  
Verwenden Sie diese optionale Tabelle zu Dokumentationszwecken und um zusätzliche Konsistenzprüfungen auszulösen:
  - Existieren alle in den Parametern `allowed`, `twoWayBlocks`, `routes` und `paths` verwendeten Blöcke?
  - Haben alle Blöcke in `blockSignals` einen Start- und Endblock in `routes`?
- Option `showTippText`  
Sie können die Tipp-Texte 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: keine Meldungen
  - 1: normale Meldungen
  - 2: umfassende Meldungen
  - 3: extreme Meldungen (nur in Englisch)

## Typische Probleme beim Hinzufügen von `blockControl` zu bestehenden Layouts

Nehmen wir an, Sie arbeiten an nicht-trivialen Layouts und konvertieren das vorgegebene Steuerungssystem eines bestehenden Layouts in automatisierte `blockControl`. Das funktioniert im Allgemeinen gut, erfordert aber einige Zeit für die Konfiguration. Sie müssen die Blocksignale sorgfältig auswählen und definieren. Es ist möglich, zusätzliche Signale (die von `blockControl` ignoriert werden) zu haben, z.B. für Anzeigezwecke oder zur Unterstützung einer Straßenkreuzung, aber das erfordert Vorsicht: die automatisierte `blockControl` geht davon aus, dass sie alle Blocksignale und zugehörigen Weichen vollständig steuert. Kein anderer Prozess oder keine Benutzerinteraktion darf sie berühren.

Werfen wir einen Blick auf einige Spezialfälle:

### Problem "Weiche zwischen Vorsignal und Hauptsignal"

Es gibt Signale mit Weichen, die zwischen dem Vorsignal und dem Hauptsignal liegen. Abhängig von der Richtung der Züge, die über diese Weiche fahren, könnte dies möglich und sinnvoll sein, und es könnte möglich sein, dass Sie die erforderliche Fahrwegdefinition manuell festlegen können. Allerdings kann das Generierungsprogramm einen solch komplizierten Fall nicht behandeln. Erinnern Sie sich an die goldene Regel vom Anfang dieses Dokuments: "*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.*"

## Problem "Zwei Signale auf demselben Gleis"

Es kann sein, dass Sie zwei Signale auf demselben Gleis finden. Das Generierungsprogramm befasst sich nicht mit den genauen Positionen der Signale auf den Gleisen, sondern verwendet nur die Verbindung zwischen den Gleisen, um Routen zwischen den Signalen zu finden. Das vereinfacht die Berechnung, führt aber zu Problemen, wenn sich zwei Signale auf demselben Gleis befinden. Höchstwahrscheinlich dienen beide Signale unterschiedlichen Zwecken und nur eines von ihnen ist ein echtes Blocksignal, das den automatischen Verkehr steuert. Sie können das andere Signal bei der Erstellung des Lua-Codes auf die "Ignorierliste" setzen.

## Problem "Sackgasse ohne Blocksignal"

Möglicherweise finden Sie Sackgassen hinter Weichen, die noch keine Blocksignale haben. Das Generierungsprogramm blockControl verlangt an jeder Sackgasse ein Blocksignal. Der Grund dafür ist, dass das Programm den Gleisen folgt und die Suchrichtung an Sackgassen widerspiegelt. Ohne Blocksignal wird dieselbe Weiche bei der Suche nach einem Weg in und aus einer Sackgasse zweimal angefahren. Dies wird als ein verbotener Zyklus interpretiert.

## Potenzielles Problem "Zu viele Zwei-Wege-Blöcke"

Sie haben viele Zwei-Wege-Blöcke definiert. Das ist möglich und funktioniert von vornherein gut. So können die Züge überall und in jede Richtung fahren. Allerdings erhöht sich dadurch das Risiko von Sperrungen, vor allem wenn viele Züge unterwegs sind und Sie gezwungen sind, eine lange Liste von Anti-Sperrstrecken manuell zu definieren. Wir schlagen vor, mit nur den erforderlichen Zwei-Wege-Blöcken zu beginnen und sie später vorsichtig zu erweitern.

## So zeigen Sie den Status von Signalen und Weichen an

Verwenden Sie den folgenden Lua-Code, um den aktuellen Status aller Signale und Weichen in den Tipp-Texten anzuzeigen:

```
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
```

**... ENDE ...**