

電車玩具シミュレータ兼コントローラ

Toy Train Lua

操作説明書

RapidNack.com

2017/07/16



このドキュメントは、電車玩具シミュレータ兼コントローラ Toy Train Lua を使って電車玩具の走行を自動制御する手順を説明します。

目次

はじめに	4
【ステップ1】シミュレータ上にレールレイアウトを作成.....	5
レールレイアウトを削除.....	7
レールを追加したい側の四角をクリック	7
レールを追加	7
ポイントの作成.....	8
駅を追加.....	10
センサー.....	10
3灯式信号機	10
【ステップ2】スクリプト言語 Lua で制御スクリプトを記述.....	12
Rail クラスのプロパティ	15
Rail クラスのメソッド.....	16
Train クラスのプロパティ	17
【ステップ3】制御スクリプトでシミュレータ上の電車の走行を制御.....	18
自動列車停止機能.....	18
【ステップ4】実物の電車玩具を制御するためのハードウェアを用意.....	19
コマンド文字列、問い合わせ文字列.....	19
*S(スキャン問い合わせ).....	20
PM (pinMode)	20
DW (digitalWrite).....	21
DR(digitalRead) — 実装省略可.....	21
AW (analogWrite) — 実装省略可	21
AR(analogRead) — 実装省略可.....	21
SA (Servo.attach).....	21
SW (Servo.write)	22
SR (Servo.read) — 実装省略可.....	22
SD (Servo.detach).....	22
BW(byteWrite).....	22
IO サーバーの Arduino スケッチサンプル.....	23
【ステップ5】シミュレータと同じ制御スクリプトで実物の電車玩具の走行を制御.....	33
制御開始手順	33
制御終了手順	34
付録 A 制御スクリプトのイベントハンドラ	35
function connectClicked()	36

function disconnectClicked()	36
function startTrainClicked()	36
function stopTrainClicked()	36
function startScanClicked()	36
function stopScanClicked()	36
function reset()	36
function detected(rail, isYellow, train)	36
付録 B 制御スクリプトの組み込みメソッド	38
connect(ipOrHost, port)	38
disconnect()	38
startTrain()	38
stopAllTrains()	38
startScan()	38
stopScan()	38
command(command)	38
query(query)	38
付録 C 制御スクリプトのグローバル変数	39
rails	39
付録 D Rail クラスのハードウェア専用プロパティ、メソッド	40
Rail クラスのハードウェア専用プロパティ (IO モジュールのピンアサイン等)	40
Rail クラスのハードウェア専用メソッド	41

《免責事項》

このソフトウェアを使用したことによって生じたすべての障害・損害・不具合等に関しては、私と私の関係者および私の所属するいかなる団体・組織とも、一切の責任を負いません。各自の責任においてご使用ください。

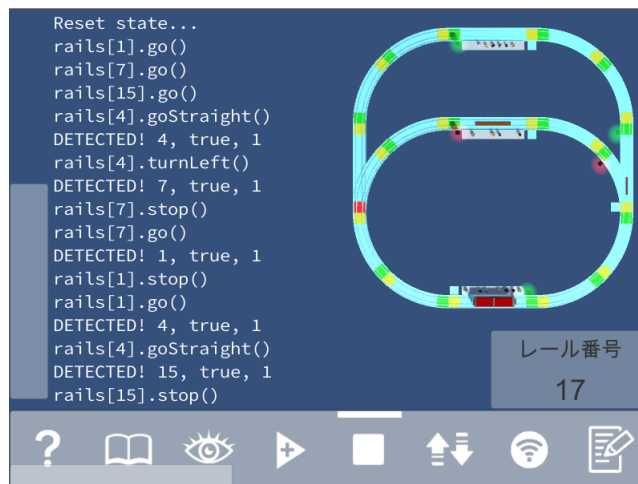
はじめに

このドキュメントは、電車玩具シミュレータ兼コントローラ **Toy Train Lua** の使用方法を説明します。

Toy Train Lua は電車玩具の走行を自動制御することを目的としています。そのために必要な作業を次の5つのステップに分けて説明します。

- 【ステップ1】シミュレータ上にレールレイアウトを作成
- 【ステップ2】スクリプト言語 **Lua** で制御スクリプトを記述
- 【ステップ3】制御スクリプトでシミュレータ上の電車の走行を制御
- 【ステップ4】実物の電車玩具を制御するためのハードウェアを用意
- 【ステップ5】シミュレータと同じ制御スクリプトで実物の電車玩具の走行を制御

ステップ1～3は **Toy Train Lua** の編集画面で実施できます。



Toy Train Lua の編集画面

シミュレータ上の電車の走行は **Toy Train Lua** の車窓画面、俯瞰画面でも確認できます。



Toy Train Lua の車窓画面、俯瞰画面

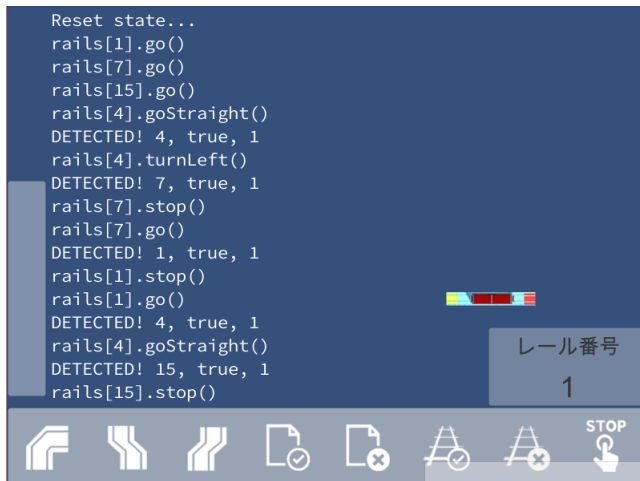
【ステップ1】シミュレータ上にレールレイアウトを作成



をクリックして編集画面に切り替えます。



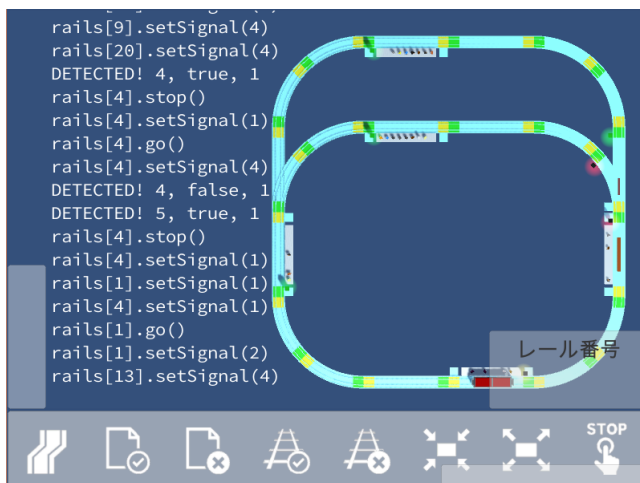
をクリックすると直線レール1本だけのレールレイアウトになります。



削除時のレールレイアウト



をクリックするとインストール時のレールレイアウトに戻ります。



インストール時のレールレイアウト

レールレイアウトの編集方法を説明するため、削除した状態から下記のレールレイアウトを手作業で作成します。

The screenshot displays a rail layout editor interface. On the left, a dark blue panel contains a command log with the following text:

```
Reset state...
rails[1].go()
rails[7].go()
rails[15].go()
rails[4].goStraight()
DETECTED! 4, true, 1
rails[4].turnLeft()
DETECTED! 7, true, 1
rails[7].stop()
rails[7].go()
DETECTED! 1, true, 1
rails[1].stop()
rails[1].go()
DETECTED! 4, true, 1
rails[4].goStraight()
DETECTED! 15, true, 1
rails[15].stop()
```

To the right of the log is a 3D-style rendering of a track layout on a dark blue background. The tracks are light blue with yellow and green segments. A red train is visible on the right side of the layout. Below the track rendering is a grey panel with the text "レール番号" (Rail Number) and the number "17". At the bottom of the interface is a toolbar with several icons: a track junction, a track curve, a track straight, a track with a checkmark, a track with an 'x', a track with a checkmark, a track with an 'x', and a "STOP" button with a hand icon.

レールレイアウトを削除

直線レール一本だけになります。



レールを追加したい側の四角をクリック

レールの端の四角をクリックすると赤色に変わります。この赤い四角の方向にレールを繋いでいきます。



右端の四角が赤色になった状態にします。



レールを追加



をクリックして赤い四角側に左曲線レールを繋ぎます。



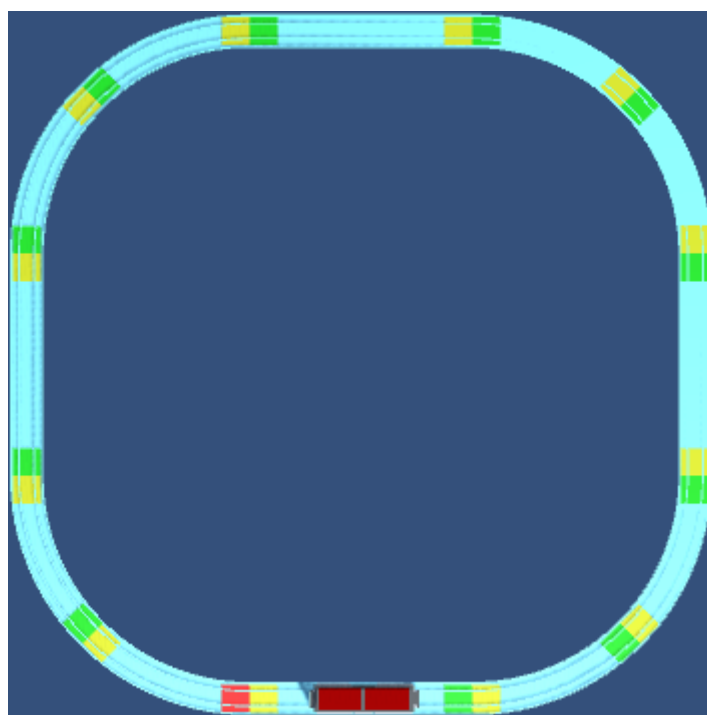
をクリックすると赤い四角のあるレールを削除できます。

レールを繋ぐと赤い四角が繋いだレールに移動するので、そのまま続けてレールを追加します。



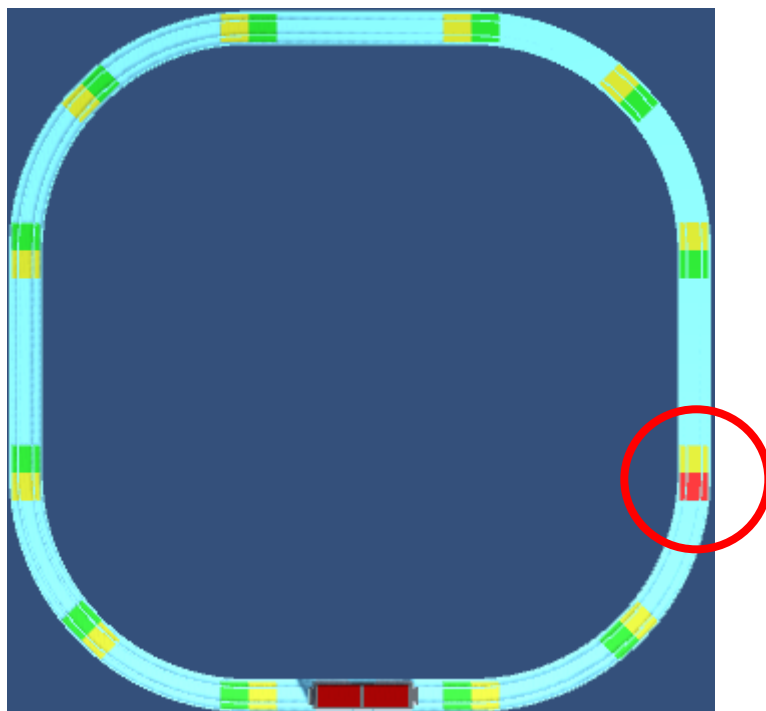
をクリックしてレールレイアウトを

次の状態にします。

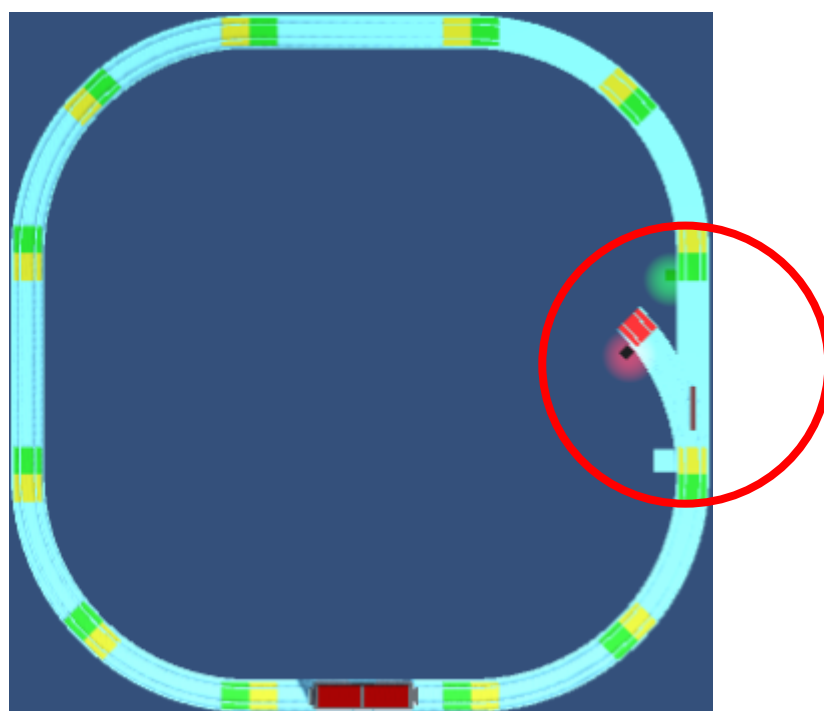


ポイントの作成

1つの四角に2本以上のレールを繋ぐとポイントになります。
既にレールが繋がっている四角をクリックして赤色にします。



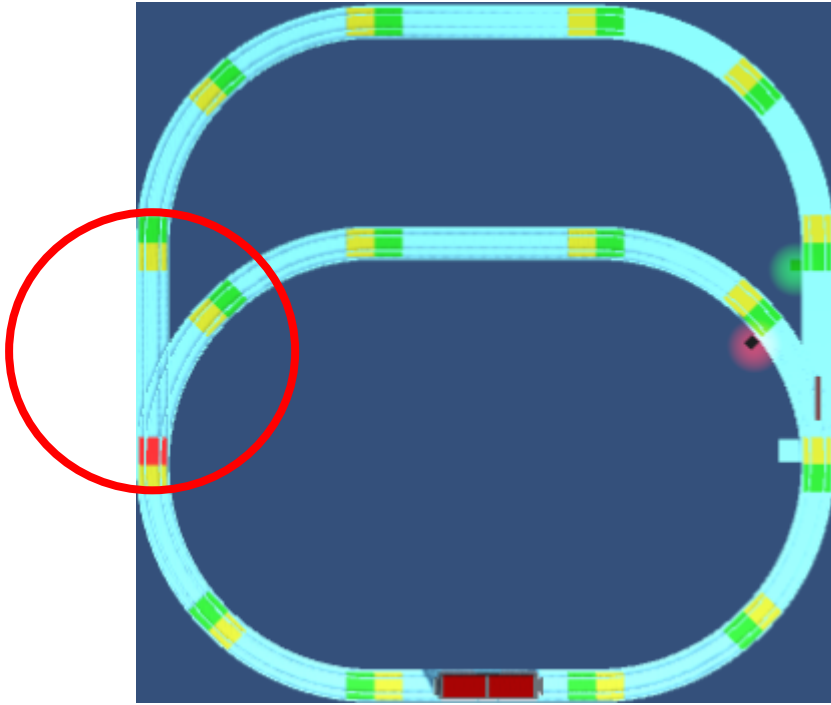
ををクリックするとポイントになります。電車はレールの黄色の端から緑色の端の方向に進むのでここは分岐ポイントになります。分岐ポイントには、現在の進行方向を示す茶色いバー、緑赤2灯式の信号機、四角いセンサーが表示されます。





をクリックしてレールを繋ぎます。

合流ポイントにはバー、信号機、センサーは表示されません。

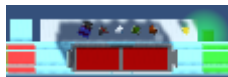


駅を追加

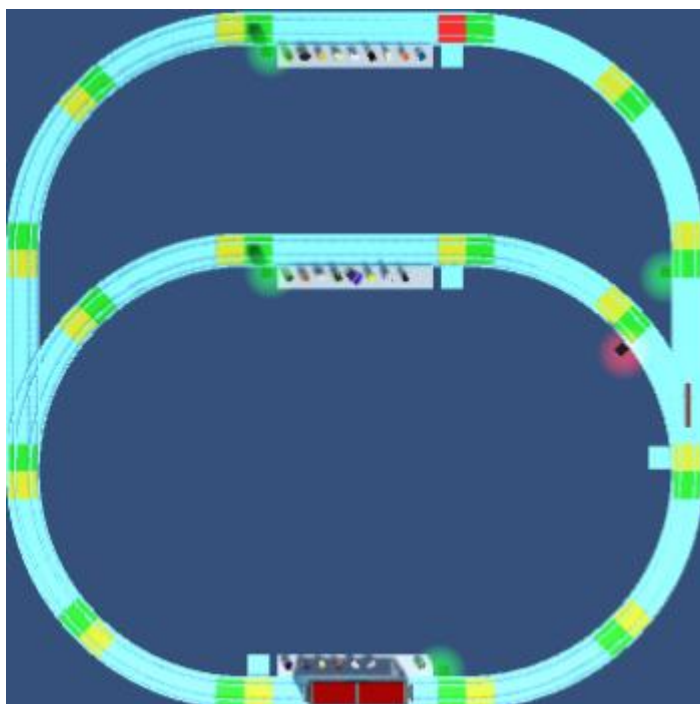
駅は直線レールにのみ設置できます。直線レールのどちらかの四角をクリックして赤色にします。



をクリックすると駅が設置されます。停止・走行を示す緑赤2灯式の信号機と四角いセンサーも表示されます。



同様にほかの直線レールにも駅を設置します。



以上の手順で、手作業で目的のレールレイアウトを作成できました。



をクリックしてレールレイアウトを保存します。次回アプリケーションを起動したとき復元されます。

センサー

ポイント、駅を設置すると黄色の四角の横にセンサーが自動的に設置されます。

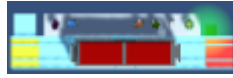


をクリックして、選択されている四角の横にセンサーを設置・撤去できます。黄色、緑色両方に設置することもできます。



3灯式信号機

ポイント、駅に表示される2灯式信号機は、ポイント、駅の状態に合わせて自動的に点灯します。



制御スクリプトから点灯を制御したいときは、をクリックして3灯式信号機を設置します。



2灯式信号機と3灯式信号機

【ステップ2】 スクリプト言語 Lua で制御スクリプトを記述



をクリックして編集画面に切り替えます。



をクリックすると制御のためのコードを追加していない最小限の内容になります。



をクリックするとテキストを編集できるアプリケーションが起動され制御スクリプトの内容が表示されます。

```
function connectClicked()
    connect('192.168.4.1', 2001)
end

function reset()
    print('Reset state...!')
end

function detected(rail, isYellow, train)
    print(string.format('DETECTED! %d, %s, %d', rail.id, tostring(isYellow), train.id))
end

print('Script started...!')
```

- **function connectClicked()**

ハードウェアと接続するとき実行されます。

【ステップ5】で説明します。

- **function reset()**



をクリックして1台目の電車が走行する直前に実行されます。

グローバル変数のテーブル `rails` 経由で、すべてのレールに対して後述のプロパティ、メソッドを使うことができます。

`rails[<レール番号>]`

例) `rails[4].turnLeft()`

レール番号4のレールのポイントを左方向に切り替えます。

- **function detected(rail, isYellow, train)**

駅、ポイントに設置したセンサーが電車を検出したとき実行されます。

rail

Rail クラスのインスタンスが渡されます。後述のプロパティ、メソッドを使うことができます。

isYellow

黄色のセンサーのときは `true`、緑色のセンサーのときは `false` が渡されます。

train

Train クラスのインスタンスが渡されます。後述のプロパティを使うことができます。



をクリックしてシミュレータ上の電車を走行させます。停止するときは

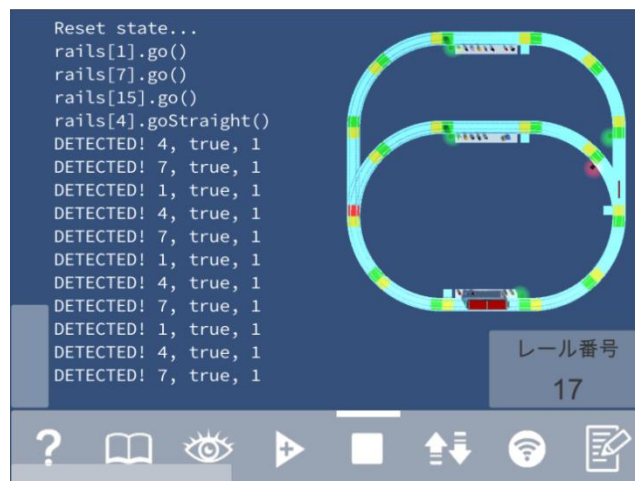


をクリックします。

まず `function reset()` に記述された "Reset state..." が画面に表示されます。

次にレールの状態が初期化されます。 `function reset()` にレールの状態を初期化するコードを記述するとそちらが優先されます。

センサーの横を電車が通過するとセンサーは黄色に変わり、 `function detected()` に記述された "DETECTED! ..." が画面に表示されます。



`function detected()` に駅、ポイントを制御するコードが記述されていないため、電車は外周を単調に走行します。



をクリックするとインストール時の制御スクリプトの状態に戻ります。

【ステップ3】以降はこの制御スクリプトを使います。

`function detected(rail, isYellow, train)` に、駅、ポイントを制御するコードが追加されています。

【注意】 ソフトウェアのバージョンによって、閉塞区間用のコードが追加されている場合があります。

```


function detected(rail, isYellow, train)
    print(string.format('DETECTED! %d, %s, %d', rail.id, tostring(isYellow), train.id))

    if isYellow and rail.isStation then
        rail.stop()
        coroutine.yield(0.5 / train.speedFactor)
        rail.addPassengers()
        coroutine.yield(1)
        rail.removePassengers()
        coroutine.yield(1)
        rail.go()
    end

    if isYellow and rail.isPoint then
        coroutine.yield(1.5 / train.speedFactor)
        rail.nextDirection()
    end
end
end

```



をクリックしてシミュレータ上の電車を走行させます。停止するときは  をクリックします。駅のセンサーの横を電車が通過すると、`function detected()` に記述されたコードによって次のように動作します。

停止

0.5 秒待機（電車の速度により調整）

ホームに新しい乗客を追加

1 秒待機

ホームから古い乗客を移動

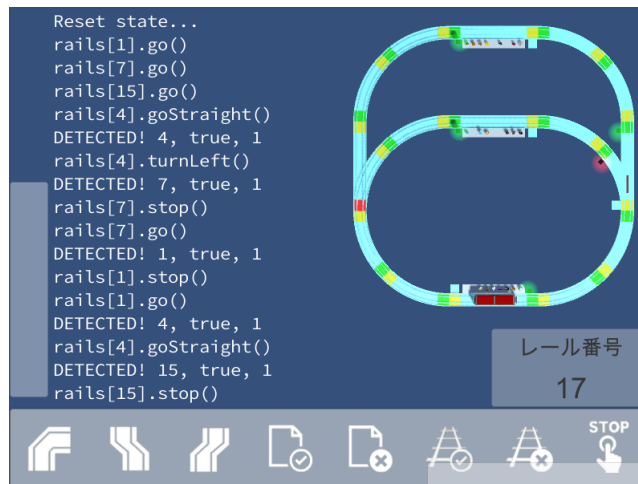
1 秒待機

走行

ポイントのセンサーの横を電車が通過すると、次のように動作します。

1.5 秒待機（電車の速度により調整）

ポイント切り替え



function detected(rail, isYellow, train)で渡された Rail クラスのインスタンス、Train クラスのインスタンスに対して下記のプロパティ、メソッドを使えます。

Rail クラスのプロパティ

- **id**

レール番号が返ります。

- **isStation**

レールに駅が設置されているときは true、設置されていないときは false が返ります。

- **isSignal**

レールに3灯式信号機が設置されているときは true、設置されていないときは false が返ります。

- **isPoint**

レールにポイントが設置されているときは true、設置されていないときは false が返ります。

- **isBlocking**

閉塞区間の状態を設定・取得します。侵入禁止のときは true、侵入許可のときは false が返ります。

- **station**

駅の状態を取得します。停止のときは”stop”、走行のときは”go”が返ります。

- **point**

ポイントの状態を取得します。進行方向が左のときは”left”、直進のときは”straight”、右のときは”right”が返ります。

- **signal**

ポイントを設置していないレールの3灯式信号機の状態を設定・取得します。ポイントを設置したレールは leftSignal、straightSignal、rightSignal を使います。

ビット2が緑、ビット1が黄色、ビット0が赤の状態を示します。緑だけを点灯しているときは4、黄

だけが点灯しているときは2、赤だけが点灯しているときは1、すべて点灯しているときは7、すべて消灯しているときは0が返ります。

- **leftSignal**

ポイントの左方向の3灯式信号機の状態を設定・取得します。値の意味は `signal` と同じです。

- **straightSignal**

ポイントの直進方向の3灯式信号機の状態を設定・取得します。値の意味は `signal` と同じです。

- **rightSignal**

ポイントの右方向の3灯式信号機の状態を設定・取得します。値の意味は `signal` と同じです。

- **forward**

進む方向にある駅、ポイントを設置しているレールを取得します。

駅のレールに対して使用すると `nil` または進行方向のレールが返ります。

ポイントのレールに対して使用すると `nil` または現在の進行方向のレールが返ります。

- **forwards**

進む方向にある駅、ポイントを設置しているレールのテーブルを取得します。

駅のレールに対して使用すると0個または1個のレールが返ります。

ポイントのレールに対して使用すると0個~3個のレールが返ります。

- **behinds**

戻る方向にある駅、ポイントを設置しているレールのテーブルを取得します。

- **leftForward**

ポイントの左方向にある駅、ポイントを設置しているレールのテーブルを取得します。

- **straightforward**

ポイントの直進方向にある駅、ポイントを設置しているレールのテーブルを取得します。

- **rightForward**

ポイントの右方向にある駅、ポイントを設置しているレールのテーブルを取得します。

Rail クラスのメソッド

- **stop()**

駅を停止状態にします。

- **go()**

駅を走行状態にします。

- **addPassengers()**

駅に新しい乗客を追加します。

- **removePassengers()**

駅から古い乗客を除きます。

- **turnLeft()**

ポイントを左方向に切り替えます。

- **goStraight()**

ポイントを直進方向に切り替えます。

- **turnRight()**

ポイントを右方向に切り替えます。

- **randomDirection()**

ポイントをランダムに切り替えます。

- **nextDirection()**

ポイントを直進→左→右の順番で切り替えます。

- **defaultDirection()**

ポイントに直進方向があれば直進方向に、なければ左方向に切り替えます。

Train クラスのプロパティ

- **id**

電車番号が返ります。

- **speedFactor**

0.35～1.0 の数値が返ります。シミュレータ上の電車のうち 2 台目以降の電車は 1 台目に比べて遅くしています。また自動列車停止機能が有効なときは駅を設置したレール上で電車の速度が遅くなります。制御スクリプトで待ち時間でタイミングを合わせるとき、電車によって適切な時間が異なります。制御スクリプトの待ち時間を **speedFactor** で割ることによって電車によるばらつきを吸収できます。

【ステップ3】 制御スクリプトでシミュレータ上の電車の走行を制御

【車窓画面】【俯瞰画面】【編集画面】 のどれでもシミュレータ上の電車の走行を制御できます。



をクリックして電車を走行させます。クリックするたびにレール番号1のレールから電車が追加されます。レール番号1のレールとその次のレールに電車が在るときは追加できません。



をクリックしてすべての電車を停止します。

自動列車停止機能



をクリックして、自動列車停止機能の有効・無効を切り替えることができます。

複数の電車が走行しているとき、自動列車停止機能は電車を強制的に一時停止させ衝突を回避します。

電車の衝突を回避させる制御スクリプトを作成するときは、自動列車停止機能を無効にします。

【ステップ4】実物の電車玩具を制御するためのハードウェアを用意

ハードウェアとして、Toy Train Lua から Wifi 経由で送られるコマンド文字列、問い合わせ文字列を処理する一台の IO サーバーと、IO サーバーに I2C で接続された複数の IO モジュールを想定しています。

● IO サーバーの例

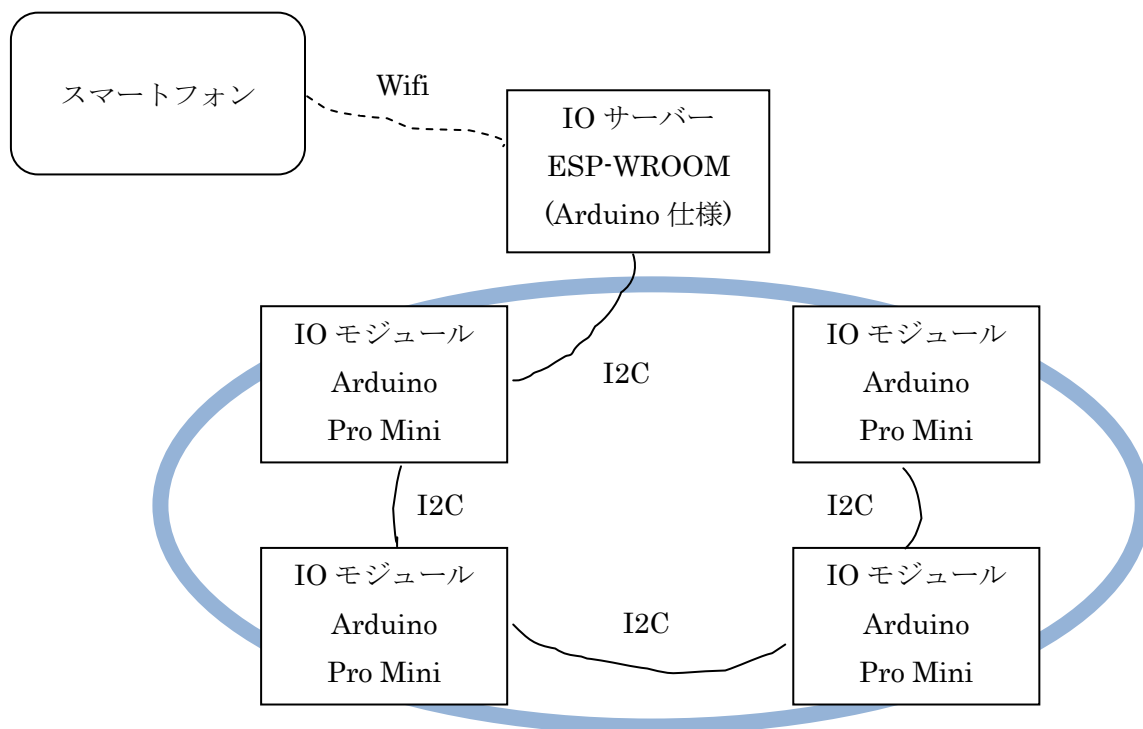
ESP-WROOM (Arduino 仕様)

Toy Train Lua から Wifi 経由でコマンド文字列、問い合わせ文字列を受け取り、I2C バス経由で複数の IO モジュールにコマンドを配信します。



● IO モジュールの例

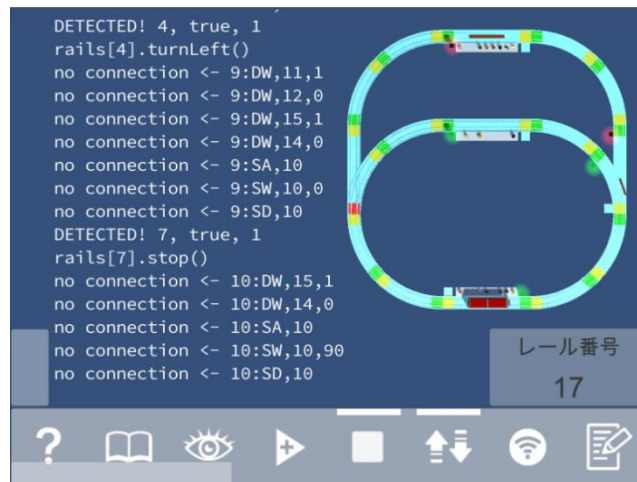
Arduino Pro Mini

I2C バス経由で IO サーバーからコマンドを受け取り、サーボモーター、LED、センサーを制御します。



コマンド文字列、問い合わせ文字列

ハードウェアを用意する前でも、 をクリックして通信接続待ち状態にしてから  をクリックしてシミュレータ上の電車をセンサーに検出させると、IO サーバーに送られるコマンド文字列、問い合わせ文字列が画面に表示されます。



Toy Train Lua から IO サーバーに、次のフォーマットのコマンド文字列、問い合わせ文字列が送られてきます。

- 7:*S,<I2C アドレス文字列>
- <I2C アドレス>:<コマンド>,<ピン>
- <I2C アドレス>:<コマンド>,<ピン>,<値>
- <I2C アドレス>:<コマンド>,<ピン 1>,<ピン 2>,<値>

*S(スキャン問い合わせ)

Arduino スケッチに直接相当するものではありません。電車を検出する機能を新たに実装する必要があります。IO モジュールに配送されるコマンドではないことを示すため<I2C アドレス>に使用されない 7 を使います。

7:*S,<I2C アドレス文字列>

I2C アドレス 8、9、10、11 のレールに黄色のセンサーが設置されたときの<I2C アドレス文字列>は”89AB”になります。IO サーバーは各 IO モジュールにセンサーの状態を問い合わせ、電車を検出していないときは 0、検出したときは電車番号を返します。I2C アドレス 9 のセンサーが電車番号 1 の電車を検出していたときのコマンドの戻り値は文字列”0100”になります。

I2C アドレス 8、10、11 のレールに黄色、緑色両方のセンサーが設置されたときの<I2C アドレス文字列>は”889AABB”になります。I2C アドレス 10 の黄色センサーが電車番号 1 の電車を検出していたときのコマンドの戻り値は文字列”0001000”になります。

ハードウェアの都合で電車番号を識別できない場合、電車番号 1 を返します。

PM (pinMode)

Arduino スケッチの `pinMode()` に相当します。

<アドレス>:PM,<ピン>,{I|O|P} → `pinMode(<ピン>,{INPUT|OUTPUT|INPUT_PULLUP})`

例) 8:PM,14,0 → I2C アドレス 8 の IO モジュールで pinMode(14, OUTPUT) を実行

DW (digitalWrite)

Arduino スケッチの digitalWrite() に相当します。

<アドレス>:DW,<ピン>,{0|1} → digitalWrite(<ピン>,{0|1})

例) 8:DW,14,1 → I2C アドレス 8 の IO モジュールで digitalWrite(14, 1) を実行

DR(digitalRead) — 実装省略可

Arduino スケッチの digitalRead() に相当します。

<アドレス>:DR,<ピン> → digitalRead(<ピン>)

例) 8:DR,14 → I2C アドレス 8 の IO モジュールで digitalRead(14) を実行
コマンドの戻り値は 0 か 1 になります。

AW (analogWrite) — 実装省略可

Arduino スケッチの analogWrite() に相当します。

<アドレス>:AW,<ピン>,<値> → analogWrite(<ピン>,<値>)

例) 8:AW,6,127 → I2C アドレス 8 の IO モジュールで analogWrite(6, 127) を実行

AR(analogRead) — 実装省略可

Arduino スケッチの analogRead() に相当します。

<アドレス>:AR,<ピン> → analogRead(<ピン>)

例) 8:AR,6 → I2C アドレス 8 の IO モジュールで analogRead(6) を実行
コマンドの戻り値は数値になります。

SA (Servo.attach)

Arduino スケッチの Servo.attach() に相当します。

<アドレス>:SA,<ピン> → Servo.attach(<ピン>)

例) 8:SA,10 → I2C アドレス 8 の IO モジュールで Servo.attach(10) を実行

SW (Servo.write)

Arduino スケッチの `Servo.write()` に相当します。

<アドレス>:SW,<ピン>,<角度> → `Servo.write(<ピン>, <角度>)`

例) 8:SW,10,90 → I2C アドレス 8 の IO モジュールで `Servo.write(10, 90)` を実行

SR (Servo.read) — 実装省略可

Arduino スケッチの `Servo.read()` に相当します。

<アドレス>:SR,<ピン> → `Servo.read(<ピン>)`

例) 8:SR,10 → I2C アドレス 8 の IO モジュールで `Servo.read(10)` を実行
コマンドの戻り値は角度になります。

SD (Servo.detach)

Arduino スケッチの `Servo.detach()` に相当します。

<アドレス>:SD,<ピン> → `Servo.detach(<ピン>)`

例) 8:SD,10 → I2C アドレス 8 の IO モジュールで `Servo.detach(10)` を実行

BW(byteWrite)

Arduino スケッチに相当するコマンドはありません。2本のピンを使って IO モジュールからシリアルで 8ビットを出力します。3灯式以上の信号機の制御に使います。

<アドレス>:BW,<ピン 1>,<ピン 2>,<値>

例) 8:BW,14,15,4

制御スクリプトに `rail.signal = 4` を記述するとそのレールに設置されたシミュレータ上の3灯式信号機の緑が点灯しますが、IO サーバー経由で IO モジュールには BW,14,15,4 が送られ、ピン 14、15 を使って 00000100 がハードウェアの3灯式信号機に送られます。

4灯式以上の信号機の場合は、`rail.setSignal(sim, hw)`を使います。シミュレータ上の3灯式信号機とハードウェアの4灯式以上の信号機に別々の値を設定できます。

IO サーバーの Arduino スケッチサンプル

ESP-WROOM-02 (Arduino 仕様) で実装した簡易的な IO サーバーのスケッチの例を示します。

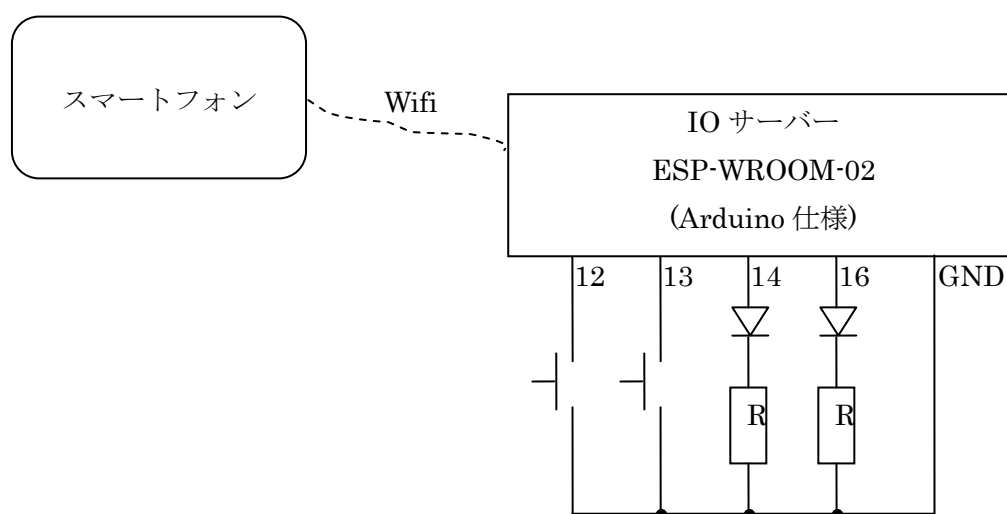
IO モジュールを別途用意せずに、センサーの代わりに押釦スイッチを2つ、信号機の代わりに LED を2つ、直接 ESP-WROOM-02 に接続します。

押釦スイッチ1 : レール番号4 (I2C アドレス 9)、ポイントの黄色センサー

押釦スイッチ2 : レール番号7 (I2C アドレス 10)、駅の黄色センサー

LED 1 : レール番号4 (I2C アドレス 9)、ポイントの直進方向の2灯式信号機の赤ランプ

LED 2 : レール番号7 (I2C アドレス 10)、駅の2灯式信号機の赤ランプ



最初のスケッチは ESP-WROOM-02 を Wifi アクセスポイントにする例です。

2 番目のスケッチは ESP-WROOM-02 を Wifi ステーションにする例です。

```

#include <ESP8266WiFi.h>
#include <WiFiClient.h>

/* Set these to your desired credentials. */
const char *ssid = "ESPap";
const char *password = "thereisnospoon";

WiFiServer server(2001);
WiFiClient client;

volatile int INT12 = 0;
volatile int INT13 = 0;
int wait12 = 0; // for debouncing
int wait13 = 0; // for debouncing

void setup() {
  Serial.begin(57600);

  Serial.println();
  Serial.print("Configuring access point...");
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);

  server.begin();
  Serial.println("Server started");

  pinMode(12, INPUT_PULLUP);
  pinMode(13, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(12), ISR12, FALLING);
  attachInterrupt(digitalPinToInterrupt(13), ISR13, FALLING);
}

void loop() {
  if (!client.connected()) {
    // try to connect to a new client
    client = server.available();
  } else {
    // read data from the connected client
    if (client.available() > 0) {
      String inputString = client.readStringUntil('\n');
      inputString.trim();
      String outputString = process(inputString);
      if (outputString != "") {
        client.println(outputString);
      }
    }
  }
}

```



```

String process(String str) {
  Serial.println(str);
  if (str.startsWith("7:*S,") { // scan query
    int int12 = 0;
    if (wait12 > 0) wait12--;
    if (wait12 == 0 && INT12) {
      int12 = INT12;
      wait12 = 10;
    }
    INT12 = 0;
    int int13 = 0;
    if (wait13 > 0) wait13--;
    if (wait13 == 0 && INT13) {
      int13 = INT13;
      wait13 = 10;
    }
    INT13 = 0;
    return "0" + String(int12) + String(int13) + "0";
  } else {
    if (str == "9:PM,15,0") {
      pinMode(14, OUTPUT);
    } else if (str == "9:DW,15,0") {
      digitalWrite(14, 0);
    } else if (str == "9:DW,15,1") {
      digitalWrite(14, 1);
    } else if (str == "10:PM,15,0") {
      pinMode(16, OUTPUT);
    } else if (str == "10:DW,15,0") {
      digitalWrite(16, 0);
    } else if (str == "10:DW,15,1") {
      digitalWrite(16, 1);
    }
    return "";
  }
}

void ISR120 {
  INT12 = 1;
}

void ISR130 {
  INT13 = 1;
}

```

```

#include <ESP8266WiFi.h>
#include <WiFiClient.h>

const char* ssid    = "your-ssid";
const char* password = "your-password";

WiFiServer server(2001);
WiFiClient client;

volatile int INT12 = 0;
volatile int INT13 = 0;
int wait12 = 0; // for debouncing
int wait13 = 0; // for debouncing

void setup() {
  Serial.begin(57600);

  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  server.begin();
  Serial.println("Server started");

  pinMode(12, INPUT_PULLUP);
  pinMode(13, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(12), ISR12, FALLING);
  attachInterrupt(digitalPinToInterrupt(13), ISR13, FALLING);
}

void loop() {
  if (!client.connected()) {
    // try to connect to a new client
    client = server.available();
  } else {
    // read data from the connected client
    if (client.available() > 0) {
      String inputString = client.readStringUntil('\n');
      inputString.trim();
      String outputString = process(inputString);
      if (outputString != "") {
        client.println(outputString);
      }
    }
  }
}
}

```

```

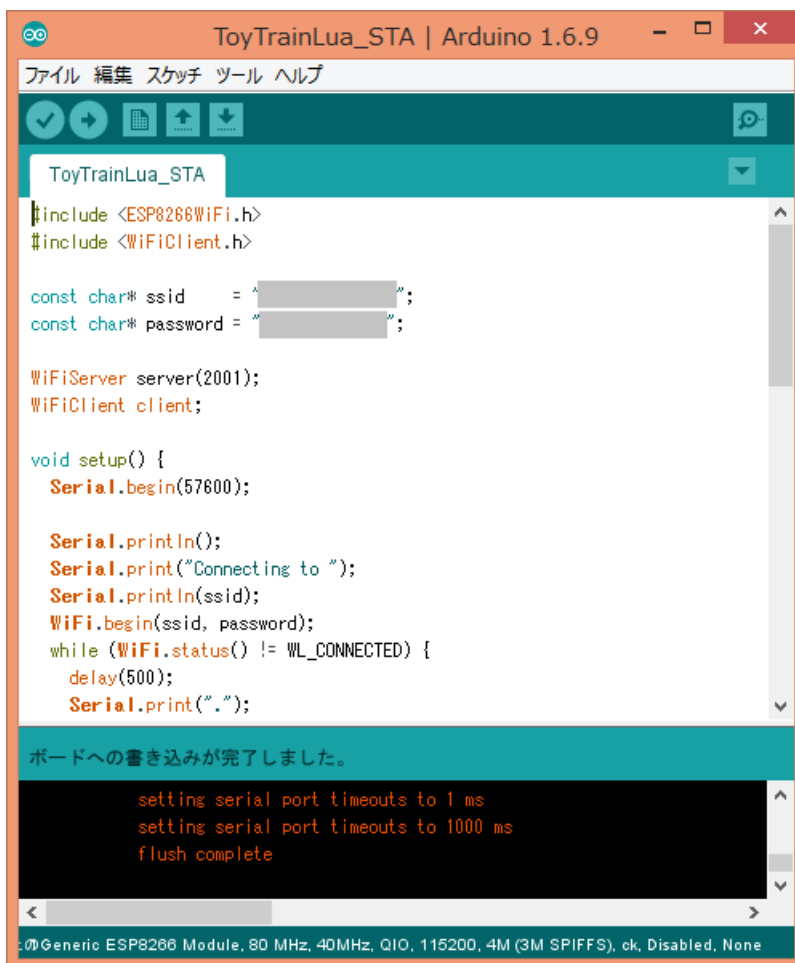
String process(String str) {
  Serial.println(str);
  if (str.startsWith("7:*S,") { // scan query
    int int12 = 0;
    if (wait12 > 0) wait12--;
    if (wait12 == 0 && INT12) {
      int12 = INT12;
      wait12 = 10;
    }
    INT12 = 0;
    int int13 = 0;
    if (wait13 > 0) wait13--;
    if (wait13 == 0 && INT13) {
      int13 = INT13;
      wait13 = 10;
    }
    INT13 = 0;
    return "0" + String(int12) + String(int13) + "0";
  } else {
    if (str == "9:PM,15,O") {
      pinMode(14, OUTPUT);
    } else if (str == "9:DW,15,0") {
      digitalWrite(14, 0);
    } else if (str == "9:DW,15,1") {
      digitalWrite(14, 1);
    } else if (str == "10:PM,15,O") {
      pinMode(16, OUTPUT);
    } else if (str == "10:DW,15,0") {
      digitalWrite(16, 0);
    } else if (str == "10:DW,15,1") {
      digitalWrite(16, 1);
    }
    return "";
  }
}


void ISR12() {
  INT12 = 1;
}

void ISR13() {
  INT13 = 1;
}

```

Arduino IDE を使って ESP-WROOM に Wifi ステーション用のスケッチを書き込みます。




Toy Train Lua の  をクリックするとテキストを編集できるアプリケーションが起動します。IO サーバーに合わせて IP アドレス、ポート番号を書き換えます。以下の例では次の内容を想定しています。

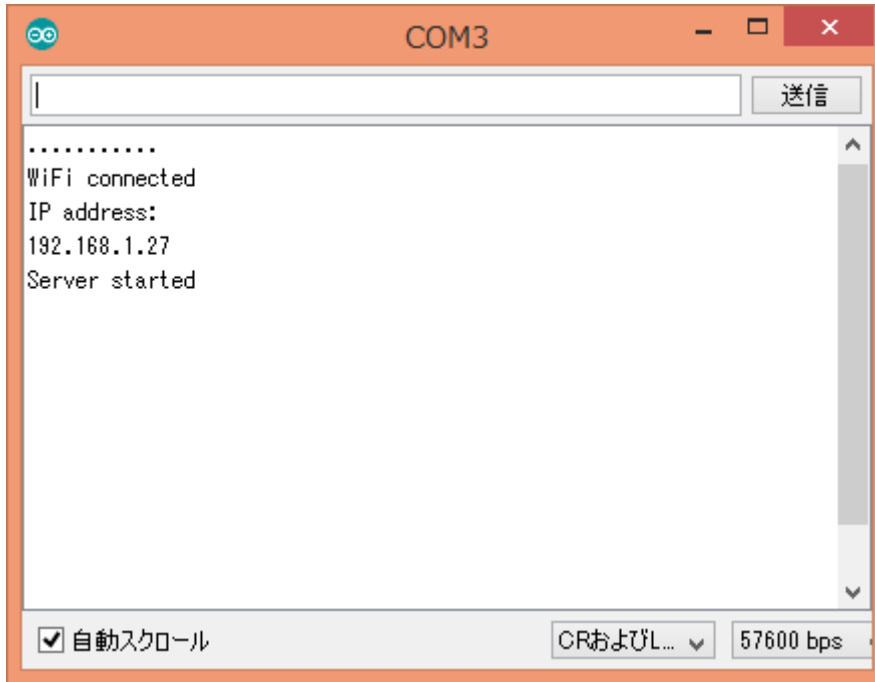
IP アドレス : 192.168.1.27




ポート番号 : 2001

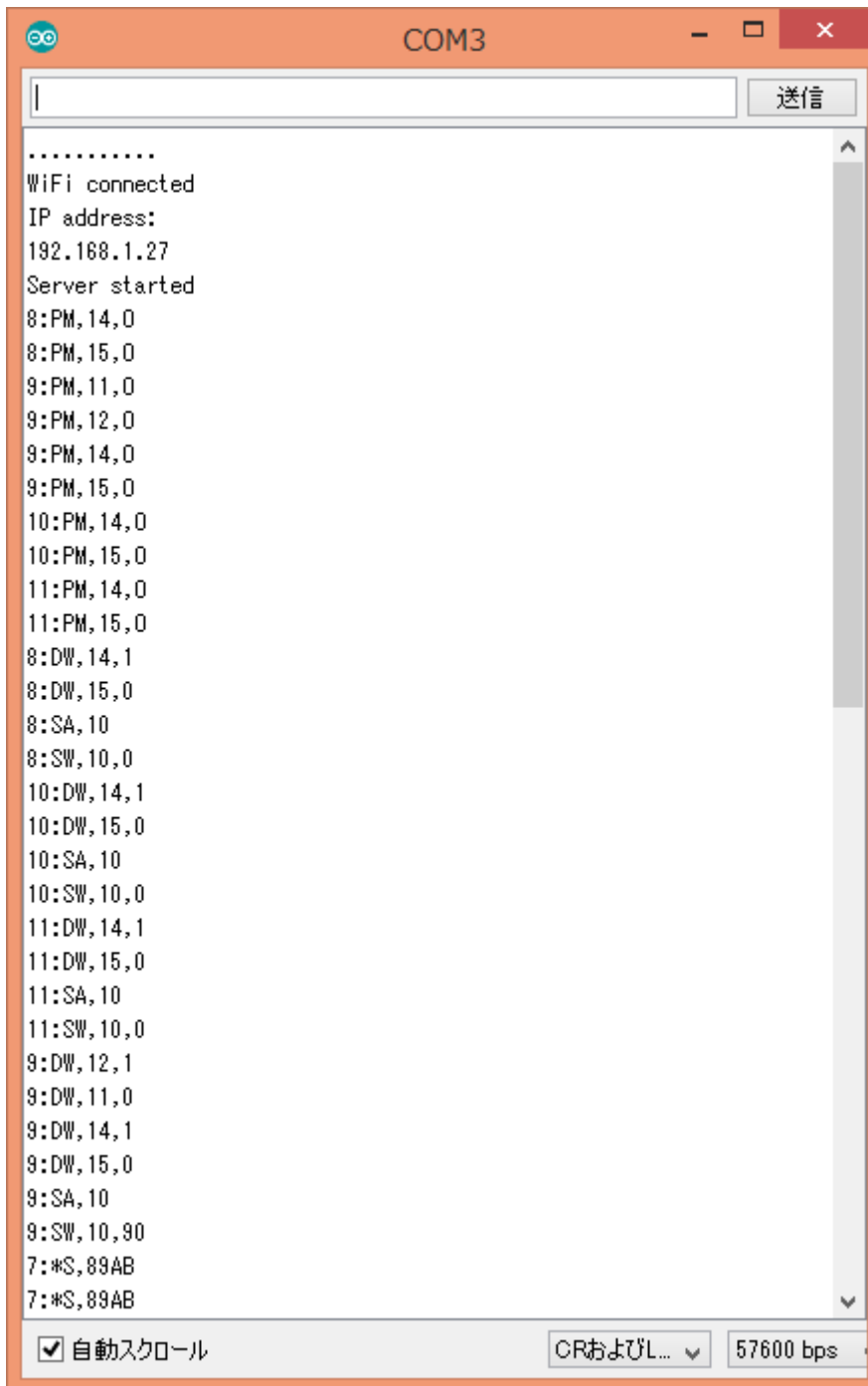
```
function connectClicked()
  connect('192.168.1.27', 2001)
end
```

編集したテキストを上書き保存すると Toy Train Lua が読み込みます。

Arduino IDE のシリアルモニタを表示してから、Toy Train Lua の  をクリックして接続します。操作のタイミングによってシリアルモニタに接続メッセージが表示されない場合もあります。



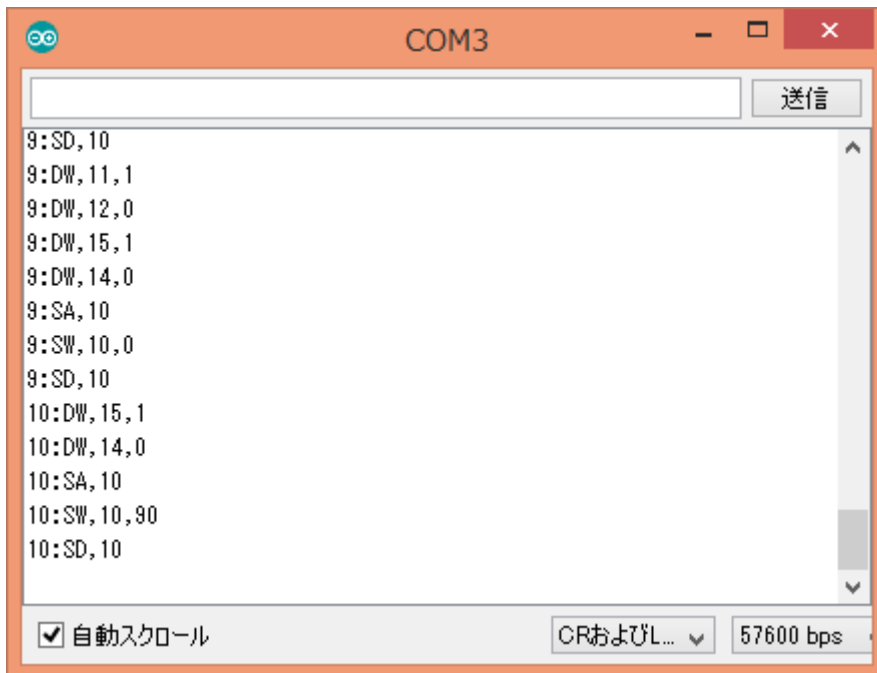
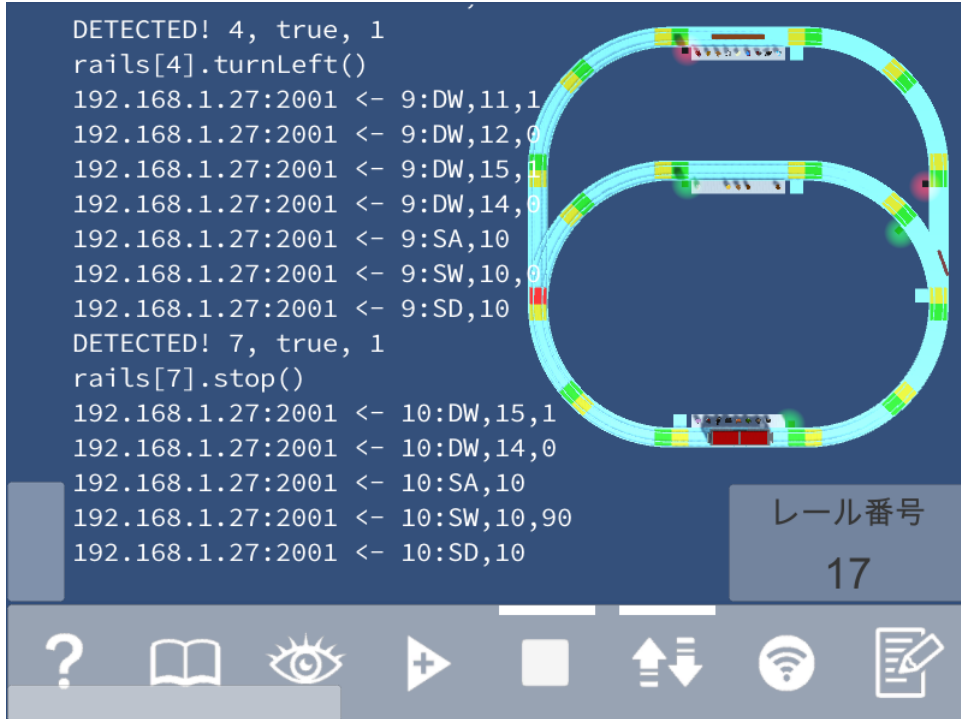
接続後  か  をクリックすると IO モジュールを初期化するコマンド文字列が Toy Train Lua から IO サーバーに送られます。ここでは  をクリックしてみます。




さらに、シミュレータ上の電車が駅、ポイントを通過するとき、サーボモーターと信号機の LED を制御するコマンド文字列が Toy Train Lua から IO サーバーに送られます。

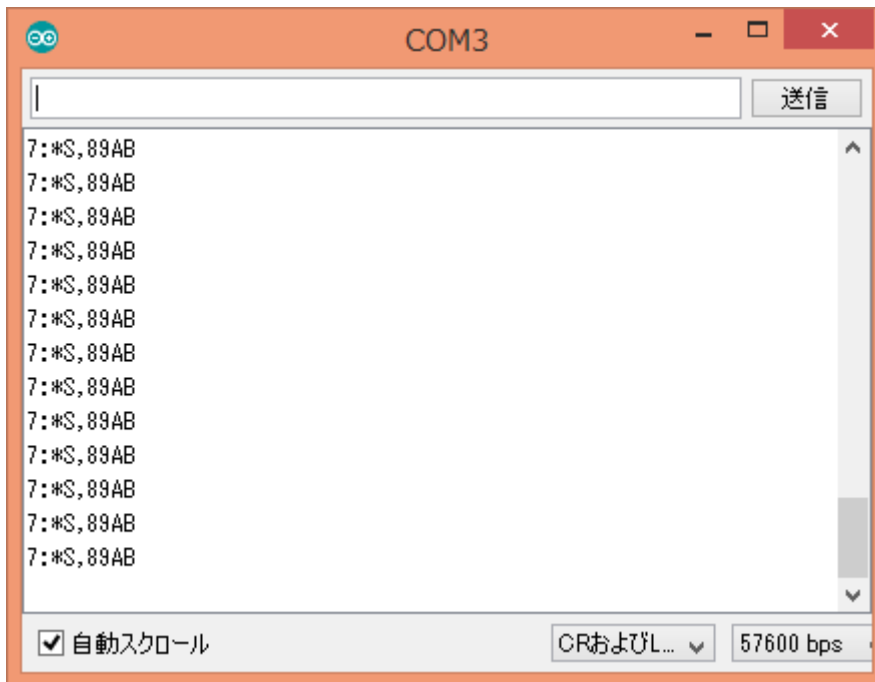
ESP-WROOM-02 のピン 14 に接続された LED がシミュレータ上のポイントの直進方向の信号機の赤ランプに合わせて点滅します。

ESP-WROOM-02 のピン 16 に接続された LED がシミュレータ上の一番上の駅の信号機の赤ランプに合わせて点滅します。





をクリックしてシミュレータ上の電車を停止し、代わりに  をクリックしてセンサースキャンを開始するとスキャン問い合わせ文字列が連続して Toy Train Lua から IO サーバーに送られます。



ESP-WROOM-02 のピン 12,13 に接続したプッシュスイッチを押すと、シミュレータ上のポイント、駅のセンサーが一瞬黄色に光り、制御スクリプトに書かれた動作を実行します。同時に Toy Train Lua から IO サーバーにサーボモーターと LED を制御するコマンド文字列が送られてきます。

ESP-WROOM-02 のピン 14 に接続された LED が、シミュレータ上のポイントの直進方向の信号機の赤ランプに合わせて点滅します。

ESP-WROOM-02 のピン 16 に接続された LED が、シミュレータ上の一番上の駅の信号機の赤ランプに合わせて点滅します。

【ステップ5】 シミュレータと同じ制御スクリプトで実物の電車玩具の走行を制御



をクリックして編集画面に切り替えます。



シミュレータ上の電車が走行している場合は、をクリックしてすべての電車を停止します。



をクリックするとテキストを編集できるアプリケーションが起動します。IOサーバーに合わせてIPアドレス、ポート番号を書き換えます。以下の例では次の内容を想定しています。

IPアドレス : 192.168.1.27

ポート番号 : 2001

```
function connectClicked()
    connect('192.168.1.27', 2001)
end
```

制御開始手順

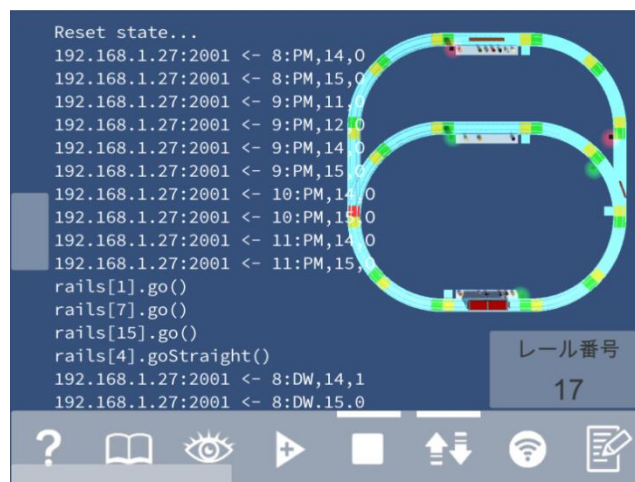


をクリックしてIOサーバーと通信を接続します。接続に成功すると画面に下記のように表示されます。

```
Connecting to 192.168.1.27:2001...
Connected to 192.168.1.27:2001.
```

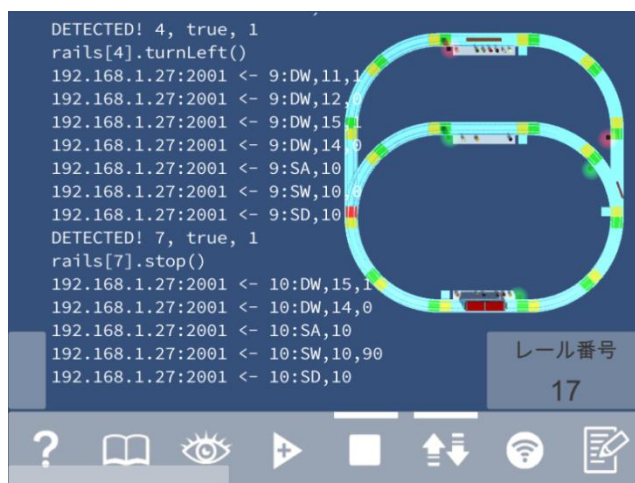



をクリックしてハードウェアのセンサー・スキャンを開始します。駅、ポイント、信号機の状態を初期化するためのコマンド文字列がIOサーバーに送られます。



電車玩具を走行させます。

ハードウェアのセンサーが電車玩具を検出すると制御スクリプトの `function detected(rail, isYellow, train)` が実行されます。



ハードウェアを用意できない場合、 をクリックしてシミュレータ上の電車を検出させることで、ハードウェアのサーボモーター、LED の制御を確認することができます。

制御終了手順

電車玩具を停止します。



をクリックしてハードウェアのセンサー・スキャンを終了します。



をクリックして IO サーバーとの通信を切断します。

付録 A 制御スクリプトのイベントハンドラ

制御スクリプトの内容をすべて削除した場合、下記のイベントハンドラが記述されているものとして動作します。同名のメソッドを制御スクリプトに記述することで差替えることができます。

```
function connectClicked()
    connect('192.168.4.1', 2001)
end

function disconnectClicked()
    disconnect()
end

function startTrainClicked()
    startTrain()
end

function stopTrainClicked()
    stopAllTrains()
end


function startScanClicked()
    startScan()
end

function stopScanClicked()
    stopScan()
end


function reset()
    print('Reset state...')
end

function detected(rail, isYellow, train)
    print(string.format('DETECTED! %d, %s, %d', rail.id, tostring(isYellow),
train.id))
end
```

function connectClicked()

通信切断状態で  をクリックしたとき実行されます。


function disconnectClicked()

通信接続状態で  をクリックしたとき実行されます。

function startTrainClicked()

 をクリックしたとき実行されます。

function stopTrainClicked()

 をクリックしたとき実行されます。

function startScanClicked()


ハードウェアのセンサー・スキャン停止状態で  をクリックしたとき実行されます。


function stopScanClicked()


ハードウェアのセンサー・スキャン実行状態で  をクリックしたとき実行されます。

function reset()

下記の3つのタイミングで実行されます。

 をクリックしてシミュレータ上に1台目の電車が走行する直前

 をクリックしてハードウェアのセンサー・スキャンが開始する直前

電車が走行またはセンサー・スキャンが開始している状態で  をクリックしてハードウェアとの通信接続が完了した直後

function detected(rail, isYellow, train)

シミュレータ上の駅、ポイントに設置したセンサー、またはハードウェアのセンサーが電車を検出したとき実行されます。

- **rail**

クラス **Rail** のインスタンスが渡されます。

- **isYellow**

黄色のセンサーのときは **true**、緑色のセンサーのときは **false** が渡されます。

- **train**

クラス **Train** のインスタンスが渡されます。

付録 B 制御スクリプトの組み込みメソッド

connect(ipOrHost, port)

IO サーバーと通信を接続します。

disconnect()

IO サーバーとの通信を切断します。

startTrain()

シミュレータ上を走行する電車を追加します。

stopAllTrains()

シミュレータ上のすべての電車を停止します。

startScan()

IO サーバーのセンサー・スキャンを開始します。

stopScan()

IO サーバーのセンサー・スキャンを停止します。

command(command)

コマンド文字列を IO サーバーに送ります。使用できるコマンド文字列は【ステップ 4】を参照してください。

例) `command('8:DW,5,1')`

I2C アドレス 8 の IO モジュールのデジタル出力ピン 5 の値を 1 にします。

query(query)

問い合わせ文字列を IO サーバーに送ります。使用できる問い合わせ文字列は【ステップ 4】を参照してください。

例) `query('8:DR,4')`

I2C アドレス 8 の IO モジュールのデジタル入力ピン 4 の値が返ります。

付録 C 制御スクリプトのグローバル変数

rails

レール番号をインデックスとした、Rail クラスのインスタンスのテーブルです。

rails[<レール番号>]

例) rails[4].turnLeft()

レール番号 4 のレールのポイントを左方向に切り替えます。使用できるプロパティ、メソッドは【ステップ 2】の Rail クラスのプロパティ、Rail クラスのメソッドを参照してください。

付録 D Rail クラスのハードウェア専用プロパティ、メソッド

【ステップ 2】の Rail クラスのプロパティ、Rail クラスのメソッドに記述されているプロパティ、メソッドのほかに、ハードウェア専用のプロパティ、メソッドが用意されています。

Rail クラスのハードウェア専用プロパティ (IO モジュールのピンアサイン等)

IO モジュールのピンアサイン等を変更するには、function `reset()` で Rail クラスのプロパティを設定します。

例) `Rail.stopServoPin = 5`

- **leftGreenPin**

ポイントの左方向の 2 灯式信号機の緑色 LED 用のピン番号を取得・設定します。初期値は 11 です。

- **leftRedPin**

ポイントの左方向の 2 灯式信号機の赤色 LED 用のピン番号を取得・設定します。初期値は 12 です。

- **straightGreenPin**

ポイントの直進方向の 2 灯式信号機の緑色 LED 用のピン番号を取得・設定します。初期値は 14 です。

- **straightRedPin**

ポイントの直進方向の 2 灯式信号機の赤色 LED 用のピン番号を取得・設定します。初期値は 15 です。

- **rightGreenPin**

ポイントの右方向の 2 灯式信号機の緑色 LED 用のピン番号を取得・設定します。初期値は 16 です。

- **rightRedPin**

ポイントの右方向の 2 灯式信号機の赤色 LED 用のピン番号を取得・設定します。初期値は 17 です。

- **stopServoPin**

駅のサーボモーター用のピン番号を取得・設定します。初期値は 10 です。

- **goAngle**

駅のサーボモーターの走行時の角度を取得・設定します。初期値は 0 です。

- **stopAngle**

駅のサーボモーターの停止時の角度を取得・設定します。初期値は 90 です。

- **stopServoDulation**

駅のサーボモーターの `attach` 時間[秒]を取得・設定します。初期値は 0.5 です。

- **pointServoPin**

ポイントのサーボモーター用のピン番号を取得・設定します。初期値は 10 です。

- **leftAngle**

ポイントのサーボモーターの左方向時の角度を取得・設定します。初期値は 0 です。

- **straightAngle**

ポイントのサーボモーターの直進方向時の角度を取得・設定します。初期値は 90 です。

- **rightAngle**

ポイントのサーボモーターの右方向時の角度を取得・設定します。初期値は 180 です。

- **pointServoDulation**

ポイントのサーボモーターの attach 時間[秒]を取得・設定します。初期値は 0.5 です。

Rail クラスのハードウェア専用メソッド

- **command(command)**

レールにアサインされた IO モジュールにコマンド文字列を IO サーバーに送ります。

例) rail.command('DW,5,1')

- **dump()**

レールにアサインされた IO モジュール間の繋がりを表示します。

例) Rail.dump()

7,15 >> 1 >> 4 (1 番レールの後方に 7 番、15 番レール、前方に 4 番レール)

1 >> 4 >> 15,7

4 >> 7 >> 1

4 >> 15 >> 1

- **query(query)**

レールにアサインされた IO モジュールに問い合わせ文字列を IO サーバーに送ります。

例) print(rail.query('DR,5'))

- **setSignal(sim, hw)**

ポイント以外のレールに設置した 3 灯式以上の信号機を制御します。

シミュレータ上の 3 灯式とハードウェアの 3 灯式以上の信号機に別々のデータを設定できます。

2 灯式信号機用の straightGreenPin、straightRedPin を流用して 8 ビットのデータを出力します。

- **setLeftSignal(sim, hw)**

ポイントの左方向に設置した 3 灯式以上の信号機を制御します。

シミュレータ上の 3 灯式とハードウェアの 3 灯式以上の信号機に別々のデータを設定できます。

2 灯式信号機用の leftGreenPin、leftRedPin を流用して 8 ビットのデータを出力します。

- **setStraightSignal(sim, hw)**

ポイントの直進方向に設置した 3 灯式以上の信号機を制御します。

シミュレータ上の 3 灯式とハードウェアの 3 灯式以上の信号機に別々のデータを設定できます。

2灯式信号機用の `straightGreenPin`、`straightRedPin` を流用して8ビットのデータを出力します。

- **`setRightSignal(sim, hw)`**

ポイントの右方向に設置した3灯式以上の信号機を制御します。

シミュレータ上の3灯式とハードウェアの3灯式以上の信号機に別々のデータを設定できます。

2灯式信号機用の `rightGreenPin`、`rightRedPin` を流用して8ビットのデータを出力します。