

# PyBahn

## Eisenbahnsimulation mit Python

Martin Reiche, Version 1 vom 24.2.2026

### Inhaltsverzeichnis

1 Einführung.....	2
2 Persönlicher Hintergrund.....	3
3 Ziel.....	3
4 Konzepte.....	4
4.1 Nutzungsphasen.....	4
4.2 Gleis.....	4
4.3 Gerade.....	5
4.4 Kurve.....	5
4.5 Weiche.....	5
4.6 Zug.....	5
4.7 Animation.....	5
5 Einschränkungen.....	6
6 Schwachstellen.....	6
7 Implementierung.....	6
8 Erweiterungen.....	7
8.1.1 Initiale Fahrtrichtung.....	7
8.1.2 Kreuzweichen.....	7
8.1.3 Tunnel.....	7
8.1.4 Wurmlöcher.....	7
8.1.5 Prellbock.....	8

# 1 Einführung

Die Simulation von Eisenbahnen bzw. U-, S-, oder Straßenbahnen im Computer kann sehr einfach oder sehr aufwändig ausfallen: Programme wie z.B. [Zusi](#) oder [3D-Modellbahn](#) bieten einen beeindruckenden Umfang an Funktionalität inklusive realistischer Grafik. PyBahn hingegen fokussiert auf das Elementare:

**Gleise und Weichen fügen sich zu Gleisanlagen, auf denen Züge fahren und dabei nicht auf die eine oder andere Weise kollidieren.**

Nicht mehr und nicht weniger. Dabei sollte die Grafik sich auf diese minimalen wesentlichen Features beschränken. Das wäre doch vielleicht ein Projekt für meine Python AG, oder?

Zwar konnte ich bei der Entwicklung schon nach wenigen Stunden die elementaren Strukturen des Gleisaufbaus in funktionierenden Code übersetzen, doch die grafische Darstellung, besonders was die Animation angeht, stellte sich als recht herausfordernd dar. Zu viel und viel zu komplex für eine Python AG!

Trotzdem wollte ich nicht ablassen und setzte mir ein zweites Ziel: Einen Baukasten zu schaffen, mit dem es möglich ist, für größere Anlagen die aus der Realität bekannten Konzepte der Fahrpläne, [Fahrstraßen](#), Signale und Sicherungssysteme selbst zu programmieren<sup>1</sup> – oder aber einfach nur eine Modellbahn zu simulieren, bei der man händisch die Weichen stellt und die Züge steuert.

Und so funktioniert es:

Zunächst konfiguriert man seinen Gleisaufbau, indem man nacheinander die Konstruktoren der beteiligten Klassen aufruft:

```
gleisreferenz = <Gleistyp> ( Vorgänger-Gleis, Anzahl gleichartiger Nachfolger )
```

Schleifen schließt man explizit:

```
gleis.nachgelagert bzw. gleis.vorgelagert = gleisreferenz
```

Nun fügt man den Verzweigungen Weichen hinzu, indem man den daran beteiligten Gleisen Rollen zuschreibt:

```
Weiche ( Eingang, Gabel, linker Abzweig, rechter Abzweig )
```

Abschließend definiert man seine Züge:

```
Zug ( Farbe, Länge, Startgleis )
```

Jetzt kann man die Züge mit dem beiliegenden Hauptprogramm fahren lassen. Sie vermeiden Kollisionen, indem sichergestellt wird, dass nicht zwei Züge gleichzeitig ein Gleis besetzen.

<sup>1</sup>Um das Verständnis des existierenden Codes zu befördern, habe ich grundsätzlich deutsche Wörter verwendet – was mir nicht immer leicht fiel...

Nachrangige Züge warten bzw. kehren um. Alle Weichen werden in zufälligen Zeitabständen umgeschaltet.

Auf dieser Ebene kann der Programmierer nun seine eigene Zugsteuerung implementieren. Jeder Zug „weiß“, auf welchen Gleisen er unterwegs ist und in welche Richtung. Jedes Gleis weiß, ob und welcher Zug gerade auf ihm fährt und benachbarte Gleise verweisen aufeinander. Alle Gleise, Züge und Weichen sind über globale Verzeichnisse auffindbar.

## 2 Persönlicher Hintergrund

In meinen Kindheitstagen gehörte eine elektrische Modelleisenbahn der Marke Märklin zu den wenigen Spielzeugen. Ich lernte an ihr, wie man ein Gleisanlage nach eigener Idee aus wenigen Sorten von Teilen zusammenbaut, also aus Geraden, Kurven und Weichen. So konnte zum Beispiel mein Zug im Kreis fahren oder auf dem Abstellgleis einzelne Wagen parken.

Später im Leben entstand immer wieder die Idee, solch einen Baukasten einmal auch in Form von Computerprogrammen zu entwickeln, mit dessen Hilfe man seine virtuelle Modelleisenbahn erstellen kann. Doch dies erschien mir immer als viel zu komplex und vor allem zeitaufwändig, sodass ich nie dazu kam, solch ein Projekt in Angriff zu nehmen.

Auf der beständigen Suche nach Programmieraufgaben für meine Python-AG kam diese Idee nun wieder einmal ans Tageslicht: Angesichts der hohen Produktivität mit Python und dem kompakten Umfang der resultierenden Programme sollte es vielleicht möglich sein, elementare Funktionen von Eisenbahnen zu simulieren.

## 3 Ziel

Ziel des Projektes PyBahn war die Erstellung eines Baukastens für virtuelle Modellbahnen. Dieser Baukastens soll jeden Interessierten (und der Programmierung in Python Befähigten) in die Lage versetzen,

- eine eigene beliebig große virtuelle Gleisanlage zu definieren. Dazu kann er Gleise, also Geraden, Kurven und Weichen in beliebiger Art zusammenstecken. Die Geometrie, also die exakte Ausrichtung der Gleise zueinander, wird programmatisch gesichert. Der Benutzer muss sich in der Regel nicht darum kümmern.
- eine eigene Zug- und Weichensteuerung zu implementieren. Sie kann händisch erfolgen, also mit Knöpfen auf der Benutzeroberfläche<sup>2</sup>, oder beliebig automatisiert. Auch lassen sich lang andauernde Versuche veranstalten, bei denen die Einhaltung von Fahrplänen bei hohem Verkehrsaufkommen protokolliert wird.

Natürlich kann das Programm auch einfach als Beispiel dienen, will man Simulationskonzepte

<sup>2</sup> Der benutzte tkinter Canvas bietet viele Möglichkeiten, zusätzliche Informationen darzustellen, wie z.B. Signale oder Bahnhöfe.

studieren oder eigene Simulationen bauen.

## 4 Konzepte

### 4.1 Nutzungsphasen

PyBahn operiert in zwei Stufen, die typischerweise aufeinander folgen:

1. Gleisaufbau: Zunächst wird der Gleisplan und die Züge erstellt. Zu diesem Zweck ruft man die Konstruktoren der betroffenen Klassen. Globale Listen für Gleise, Weichen und Züge dienen der nächsten Stufe:
2. Bei der Simulation fahren Züge auf den Gleisen nach den Aufrufen des Hauptprogramms: Züge schreiten voran oder kehren um, Weichen werden gestellt.

### 4.2 Gleis

Die Bedeutung von *Gleis* in PyBahn entspricht der bei einer Modellbahn wie z.B. von Märklin: Hier wie dort ist ein Gleis ein Element zur Erstellung der Gleisanlage. Dazu wird es mit anderen Gleisen an seinen Enden verbunden. Was in der Modellbahn die Steckverbindung leistet, erledigen in PyBahn Referenzen auf benachbarte Gleise.

In PyBahn ist *Gleis* eine abstrakte Basisklasse, von der es zwei Unterklassen gibt: Gerade und Kurve. (Von letzterer gibt es, genau wie in der Realität, keine rechte und linke. Dazu wird sie erst in der Konfiguration.)

Gleise spielen zweierlei Rollen, in denen es ankommt auf:

1. ihren Ort und ihre Orientierung im Raum bzw. ihre Vorgänger und Nachfolger im Prozess des Gleisaufbaus.
2. ihre Vorgänger und Nachfolger im Gleisplan, ihre Beteiligung an Weichen, sowie die Belegung mit Zügen während der Simulation.

Bei der Programmierung müssen beide Aspekte der Gleise berücksichtigt werden. Um hier möglichen Verwirrungen zu entgehen, was die jeweilige Rolle angeht, werden alle Funktionen, die sich mit der Geometrie bzw. der grafischen Darstellung beschäftigen, in den Unterklassen, also *Gerade* und *Kurve* implementiert. Die Basisklasse *Gleis* implementiert die Funktionen, die bei der Simulation anfallen.

Alle Gleise sind in dem globalen Verzeichnis (Python dictionary) *gleise* hinterlegt, wobei die fortlaufende Nummer aus ihrer Erzeugung als Schlüssel dient. Gleichzeitig enthält jedes Gleisobjekt eine Member-Variable namens *nummer*. Dies erleichtert u.a. das Debugging. Die Gleisnummern dienen u.a. auch den Zügen zur Verwaltung der von ihnen belegten Gleise.

### 4.3 Gerade

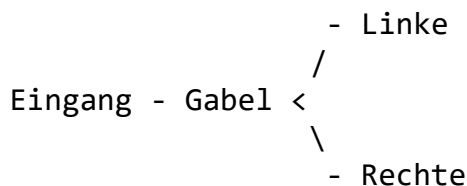
Bei der Konstruktion einer Gleisanlage kann es – zum Beispiel bei Diagonalen – vorkommen, dass nicht genau ein Vielfaches der Einheitslänge (= Länge eines Kurvengleises) verwendet werden kann. Um diesem Problem auszuweichen, wählt man einen Gleisplan, bei dem ungefähr ein Vielfaches der Einheitslänge benötigt wird. Dann werden die betroffenen Gleise mit einem Längenfaktor erzeugt, der nahe bei Eins liegt. Dann fällt die resultierende Verlängerung oder Verkürzung des Zuges nicht auf.

### 4.4 Kurve

Der Segment-Winkel einer Kurve errechnet sich aus der (konfigurierbaren) Anzahl Kurven-Gleise pro Vollkreis. Es gibt nur Kurven mit dem Standard-Radius. Die Länge eines Kurven-Gleises bestimmt die Länge aller Geraden.

### 4.5 Weiche

Beim Gleisaufbau erstellt man zunächst Verzweigungen, indem man an ein vorhandenes Gleis (Gerade oder Kurve) direkt zwei andere Gleise anhängt d.h. eine Gerade und eine Kurve oder eine Linkskurve und eine Rechtskurve. In einem zweiten Schritt dann definiert man Weichen. Dazu verbindet die separate Klasse *Weiche* vier Gleise zu einem Verbund mit den Rollen *Eingang*, *Gabel*, *Linke* und *Rechte*, entsprechend folgender Skizze:



Daher können Weichen nicht räumlich direkt nacheinander konstruiert werden. Zumindest können sich die Gabel-Gleise nicht überlappen. Weitere Einschränkungen müsste man erst erforschen.

### 4.6 Zug

Ein Zug besteht aus einem Kopf und folgenden Elementen. Der Kopf bestimmt über das nächst befahrene Gleis und vermeidet Kollisionen. Alle befahrenen Gleise werden mit ihrer Nummer in einer Liste geführt, deren Anfang immer das nächst befahrene Gleis hinzugefügt wird und an deren Ende nacheinander die verlassenen Gleise abgetrennt werden. Beim Richtungswechsel des Zuges wird diese Liste umgedreht (*reverse*) und die Rollen entsprechend angepasst.

### 4.7 Animation

Der als flüssig wahrgenommene Ablauf der Simulation kommt dadurch zustande, dass zwischen den Sprüngen der Züge von einem Gleis zum nächsten die betroffenen Gleisabschnitte allmählich mit

der Zugfarbe gefüllt (vorne) bzw. von ihr geleert und durch die Gleisdarstellung ersetzt werden. Sowohl die Logik als auch die Anbindung an die Arc-Funktion von tkinter's Canvas (create\_arc) erfordert recht komplexe Programmierung (die mich eine Menge Zeit gekostet hat).

Bei PyBahn lässt sich die flüssige Animation über die Variable *ANIMATION* zentral ein- und ausschalten.

## 5 Einschränkungen

Die folgenden Einschränkungen sind Design-bedingt, können also nicht so einfach überwunden werden.

1. Züge bewegen sich Gleis für Gleis schrittweise voran.
2. Alle Züge fahren gleich schnell oder stehen bewegungslos. Beschleunigungs- oder Verzögerungsphasen existieren nicht.
3. Alle Gleise sind gleich lang. Nur das gewährt eine konstante sichtbare Länge der Züge.
4. Züge können nur vielfache von Gleisen belegen.
5. Die flüssige Animation versteckt die schrittweise Bewegung der Züge. Darum können Züge beim Übergang nicht auf „halb-befahrenen“ Gleisen stehenbleiben.

## 6 Schwachstellen

1. Begegnen sich Züge auf einer Weiche, wartet der später kommende. Dabei belegt er einen der Schenkel-Gleise (Linke oder Rechte). Das sieht dann fälschlicherweise so aus, als würden sie gleichzeitig dasselbe Gleis befahren.
2. Wenn Züge direkt hintereinander fahren, kommt es an der Nahtstelle zu Fehlern in der animierten Darstellung.
3. Manchmal erscheinen die Enden der Züge abgerundet. Dieser Fehler liegt aber in der tkinter-Implementierung.

## 7 Implementierung

PyGame wurde unter Windows 11 mit der Python Version 3.10.11 programmiert. Die [Thonny](#) Entwicklungsumgebung Version 4.1.7 erwies sich dafür als ausreichend leistungsfähig. Mir fehlte lediglich die „zurück“-Navigation im Editor. Thonny läuft stabil und zeigte nur einen einzigen Hänger beim Umbenennen einer Datei und keinerlei andere Fehlfunktionen.

Der PyBahn-Code ist unterteilt in zwei Dateien:

1. PyBahn.py enthält die Hilfsklasse Vector, die Modellklassen und ein paar Hilfsfunktionen.

2. main.py enthält die Gleisdefinition sowie den Code zur schrittweisen und animierten Simulation.

Startet man die Simulation mit dem Knopf Start/Stop, so fahren 3 Züge auf einem Gleisbild mit Schleifen und Weichen, sodass sie permanent in Konflikt geraten. Dies ist aus Testzwecken so gewollt.

Die grafische Skalierung, also Größe der Darstellung wird durch wenige Konstanten definiert:

- RADIUS – der Krümmungsradius der Kurven. Er bestimmt die Einheitslänge aller Gleise und somit auch die Größe des Gleisplans im Fenster.
- ZUGBREITE – Die Dicke der Linie, welche die Züge darstellt.
- GLEISBREITE - Die Dicke der Linie, welche die unbefahrenen Gleise darstellt.
- WEICHENINDIKATOR – Die Größe des Indikators für die Weichenstellung.

## 8 Erweiterungen

### 8.1.1 Initiale Fahrtrichtung

Aktuell richtet sich die initiale Fahrtrichtung nach der Richtung der Gleise beim Aufbau. Man könnte diese aber auch nach Himmelsrichtungen festlegen, bei der die x-Achse von West nach Ost und die y-Achse von Nord nach Süd verläuft. Weil die Gleise die Koordinaten ihrer Nachbarn kennen, können sie das gewünschte nächste Gleis auf der Fahrt bestimmen.

### 8.1.2 Kreuzweichen

Es ließen sich Weichen mit 4 Zugängen integrieren, ohne am bestehenden Gefüge der Klassen Veränderungen vornehmen zu müssen.

### 8.1.3 Tunnel

Bei einer Modelleisenbahn sieht man Züge im Tunnel nicht. Bei PyBahn ließe sich den Gleisen ein Sichtbarkeitsattribut hinzufügen. Sind sie während der Simulation unsichtbar, funktionieren sie doch wie erwartet.

### 8.1.4 Wurmlöcher

Bei diesen aus der Science Fiction bekannten Phänomenen verschwinden Körper an einem Ort und tauchen plötzlich woanders wieder auf. Dies kann bei PyBahn so realisiert werden, dass zwei entfernte Gleise als Nachbarn ausgegeben werden. Für eine Simulation ohne Animation ist dafür keine Erweiterung des Programms notwendig. Mit flüssiger Animation werden sich aber Schwierigkeiten ergeben...

### **8.1.5 Prellbock**

Dieser ließe sich implementieren vielleicht als dauerbelegtes Gleis ohne Ausdehnung, das keine nahgelagerten Gleise enthält. (Mit der aktuellen Zugsteuerung würden Züge vor ihm anhalten und schließlich umkehren.)