

Transport Network Provider - Lanes und ihre Bedeutungen

Inhaltsverzeichnis

- [1 Einführung](#)
- [2 Kleine Vorbereitungen](#)
- [3 Edges und Nodes - die "Ecken" und "Knoten"](#)
- [4 Eine Lane erstellen](#)
- [5 Terminals](#)
- [6 Ausrichtung der Passagiere](#)

Zu aller erst: Dieser Eintrag entsteht ein Stück weit in der learning-by-doing Methode und kann durchaus unvollständig sein ;)) Ich werde hier mein Wissen über die Thematik so gut es geht versuchen zu vermitteln. Desweiteren sei gesagt, dass ICH >>mein<< Wissen hierbei hauptsächlich durch [@Bandion](#) erlangt habe. Das bedeutet nicht, dass er der einzige ist, aber er war gerade redseelig, als ich zuhörte. Ich möchte ihn daher hier keinesfalls unerwähnt lassen. Mittlerweile darf ich auch noch [@Tom](#) für seine Erklärungen hinsichtlich der Terminals und Zählweise der Edges und Nodes danken! :thumbsup:

Weiterhin werde ich mich im Zuge von @Bandions aktuellem Projektes [\[MOD\] Busbahnhof/Tramstationen/Frachtstationen](#) mit seinem Busbahnhof auseinandersetzen und auf Grundlage dieses Modells hier die einzelnen Schritte erklären. Somit ein dickes Danke an Bandion für sein Einverständnis!

So soll es beginnen...

1 Einführung

Öffnet man die .mdl einer Station, so befindet sich dort der Eintrag **transportNetworkProvider**. Dieser befindet sich im Block **metadata** und enthält die Sub-Blöcke (oder -Listen) **laneLists** und **terminals**.

laneLists enthält für jede "Transportart" einen Eintrag. Das bedeutet, dass die Fußgänger beispielsweise einen Eintrag bekommen, sowie bei einem Busbahnhof ebenfalls die Busse. Bei einer Tram-Station würden zu den Bussen und Passagieren noch die Trams kommen. Bei LKW-Stationen hingegen würde man

einerseits für die LKW's Lanes anlegen sowie auch Fracht-Lanes anstelle der für Fußgänger. Jeder **laneLists**-Eintrag (Block) enthält somit die folgenden 2 Parameter: **transportModes** und **nodes**. Die Werte für **transportModes** sind eben jene, für welche die im Eintrag/Block angelegten Nodes gültig sind -> Für Busse oder LKW's? Passagiere oder Cargo? Die Werte die man hierfür angeben kann sind diese hier:

- "BUS"
- "TRAM"
- "ELECTRIC_TRAM"
- "TRUCK"
- "PERSON"
- "CARGO"

Man kann auch mehrere dieser Werte gleichzeitig angeben. Sollen die **nodes** dieses Eintrags für Busse UND Trams gelten, so braucht man sich die Arbeit nicht doppelt machen, sondern gibt als **transportModes** einfach beides an (bzw alle drei - mit den Electrics...). Die **nodes** sind dann die eigentliche Wissenschaft hierbei. Darauf werde ich gleich noch einmal detailliert eingehen. Hier aber erst noch ein Codeausschnitt, wie das Ganze in etwa aussieht:

Code: myStation.mdl

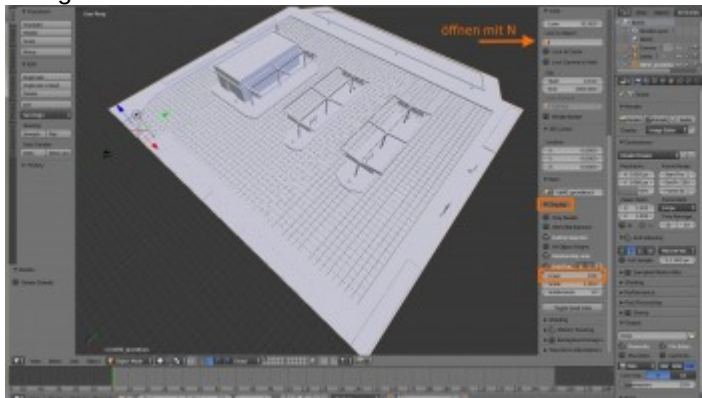
```
...
metadata                                     =                                     {
...
  transportNetworkProvider                   =                                     {
    laneLists                               =                                     {
      {
        transportModes                       =                                     { "BUS" },
        nodes                               = laneutil.makeLanes({
          {2.0,-1.00.0},{0.01.00.0},{0.01.00.0},{1.00.00.0}, 3.0},
          {0.01.00.0},{2.0,-1.00.0},{1.00.00.0},{0.0,-1.00.0}, 3.0},
        })),
      {
        transportModes                       =                                     { "PERSON" },
        nodes                               = laneutil.makeLanes({
          {6.5,-1.00.3 },{0.05.50.3},{0.01.00.0},{1.00.00.0}, 1.5},
          {0.05.50.3 },{6.5,-1.00.3},{1.00.00.0},{0.0,-1.00.0}, 1.5},
        })),
      },
    },
    terminals                               =                                     {
      { vehicleNode = 1, personNodes = { 1 }, personEdges = { 1 } },
    },
    ...
  },
  ...
}
```

Alles anzeigen

2 Kleine Vorbereitungen

Bei euch in Steam gibt es für TrainFever eine **settings.lua**. Bei mir liegt diese in **...\\Steam\\userdata\\195337234\\304730\\local** wobei die kursive Zahl meines Wissens nach bei jedem eine andere ist. Wahrscheinlich ist das die Steam-ID von uns oder sowas und die Zahl danach ist die Steam-ID für TF. Irgendwie so. Is ja auch pups. Wo die Datei bei GoG-Nutzern liegt kann ich leider nicht sagen, aber ich denke eine Suche über den Explorer sollte hier Abhilfe schaffen. Öffnet diese nun mit einem Texteditor (ich empfehle Notepad++) und ihr findet folgende Zeile dort: `28 debugMode = false`, . macht aus dem **false** ein **true** und speichert das Ganze (oder schaut euch noch ein bisschen um ^^). Nun könnt ihr Ingame mit der Tastenkombination AltGr+L die Lanes anzeigen lassen (und mit AltGr+G das UI/HuD ausblenden lassen für schönere Screenshots ^^).

Desweiteren nutze ich Blender und hier hab ich mir ein leeres Projekt erstellt und @Bandions Busbahnhof importiert. Ihr werdet sicher beim Lesen etwas eigenes haben. Ich fand das Grid eigentlich recht hilfreich beim Positionieren und Abschätzen bisher. Allerdings war das leider etwas klein. Das lässt sich aber ändern! Falls ihr im 3D-View noch nicht das rechte Panel geöffnet habt, so drück einmal **N** (mit dem Maus-Zeiger in der 3D-View) und sucht nun in diesem Panel die Option **> Display**. Mit einem Klick aufs **>** öffnet sich der Eintrag und ihr könnt den Wert der **Lines** erhöhen. Der sollte default auf 16 stehen, ich habe ihn mir hier auf 100 gezwirbelt.



Das Grid liegt nun schön über dem ganzen Modell - leider verschwindet es bei der Top-View !map not found or type unknown. Aber es ist besser wie nichts. Im Bild zu sehen ist eine weitere Technik, die ich mir fürs Nachbauen von Bandion abgeschaut habe: Ich habe mir einen Hilfs-Cube erstellt und ihn in den Modell-Origin gelegt. Hier erfolgt der Straßenanschluss und dahingehend habe ich den Würfel ausgerichtet. Die obere und untere Fläche sowie die "nach draussen" habe ich gelöscht. Jede Kante dieser so entstanden Wand soll später eine Node darstellen. Die einzelnen Wand-Flächen entsprächen somit den Edges. Eigentlich reicht eine Linie, aber die Wand sieht man besser wie ich finde. Weitere "Nodes" füge ich einfach mit Subdivides ein und fertig. Am Ende baue ich mir daraus meine gewünschte Linienführung vor und kopiere die Koordinaten einfach ab - laut Plan.

Nun aber erstmal weiter im Text.

3 Edges und Nodes - die "Ecken" und "Knoten"

Ich habe erstmal frech alles von Bandion schon gebaute gelöscht um einen eigenen Einstieg zu finden. Meinen Code könnt ihr oben bewundern - das ist genau dieser "minimale Code" den ich gebaut habe. Wir haben unseren Network-Provider mit den Lanes und den Terminals und bei den Lanes gibt es welche für Busse und für Sims. Und die Nodes sind eine sehr kryptisch aussehende Menge von geschwungenen Klammern und Zahlen:

```
{ { 2.0, -1.0, 0.0 }, { 0.0, 1.0, 0.0 }, { 0.0, 1.0, 0.0 }, { -1.0, 0.0, 0.0 }, 3.0 },
```

Also mich hat das erstmal bissl erschlagen. Magnetfrühdurchbruch Aber ich erkannte zumindest schonmal eine gewisse Struktur:

```
{ Vektor, Vektor, Vektor, Vektor, Zahl },
```

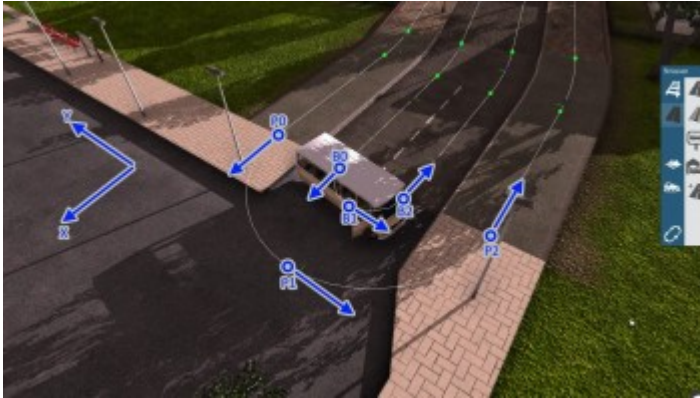
Also 4 Vektoren und eine Zahl. Bandion hatte es dann freundlicherweise erklärt:

```
{ Start-Punkt, Ziel-Punkt, Start-Ausrichtung, Ziel-Ausrichtung, Spurbreite },
```

Was bedeutet das nun konkret? Zwei **Nodes** erstellen zwischen sich einen Pfad - eine sogenannte **Edge**. Genau diese zwei Punkte/Nodes sind die ersten beiden Vektoren. Die Ausrichtungsvektoren sind etwas spezieller. Es wird nicht zwingend eine gerade Linie zwischen den Punkten erstellt sondern kann durchaus auch in einer Kurve enden. Und diese Kurve bzw Gerade, je nachdem, kann man beeinflussen. Stellt euch vor, ihr steht auf der Straße (Vektor Nummer 1) und schaut in Fahrtrichtung. Genau diese Angabe "in Fahrtrichtung" wäre schonmal der dritte Vektor in der Liste. Nun wollt ihr nach links abbiegen. Ihr gebt also einen Ziel-Punkt an (der zweite Vektor) und an diesem Zielpunkt angekommen, wollt ihr dann nach links schauen. Also gebt ihr dieses "nach links" als vierten Vektor an. Wenn ihr nun loslauft schaut ihr erst nach vorne und lauft dann in einem Bogen nach links. Ihr endet nun nach links schauend im Zielpunkt. Ihr hättet auch den selben Ausrichtungsvektor für Punkt2 wählen können. Dann wäht ihr ein S gelaufen. Also erst (eine engere) Kurve nach links, dann wieder nach rechts zum Zielpunkt. Ihr seid wieder im Zielpunkt, diesmal aber schaut ihr wieder "in Fahrtrichtung".

Wie das in genauen Werten aussieht, erkläre ich gleich.

Schauen wir uns einmal an, wie das im Spiel aussieht. Ich habe mal ein wenig auf dem Bild rumgeschmiert, damit sich besser erläutern lässt. Im Screenshot sieht man auch schön die aktivierte Lane-Ansicht (AltGr+L).



Ich habe hier 6 Punkte markiert: P0, P1 und P2 als Person-Nodes sowie B0, B1 und B2 als Bus-Lanes. Zudem zeigen die Pfeile unsere Richtungen an. Wo steckt das also nun in unserem Code von oben? Ziehen wir uns noch einmal die Bus-Lanes heraus:

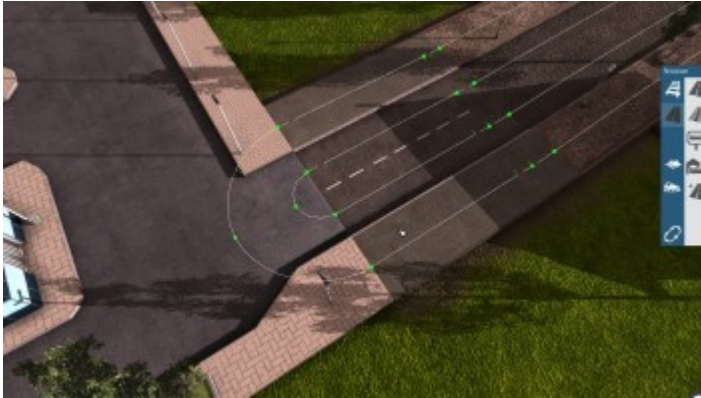
```
{ { 2.0, -1.0, 0.0 }, { 0.0, 1.0, 0.0 }, { 0.0, 1.0, 0.0 }, { -1.0, 0.0, 0.0 },
3.0 },
{ { 0.0, 1.0, 0.0 }, { -2.0, -1.0, 0.0 }, { -1.0, 0.0, 0.0 }, { 0.0, -1.0, 0.0 },
3.0 },
```

Um erst einmal die Punkte wieder zu finden, schreibe ich mal anstelle der Vektoren einfach meine Punkt-Namen hier rein:

```
{ B0, B1, { 0.0, 1.0, 0.0 }, { -1.0, 0.0, 0.0 }, 3.0 },
{ B1, B2, { -1.0, 0.0, 0.0 }, { 0.0, -1.0, 0.0 }, 3.0 },
```

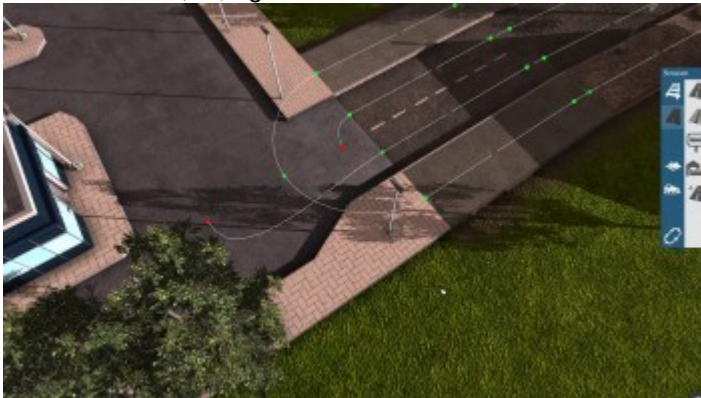
Wie wir sehen gibt die erste Zeile die Verbindung zwischen den Punkten B0 und B1 an und die zweite dann die zwischen B1 und B2. Nun kurz zu den Achsen: Wie im ersten Bild ersichtlich wird, ist die X-Achse von Bild 2 her gesehen vom Bus aus nach hinten weg ausgerichtet. Also da wo der Bus steht zu seinem Heck hinaus - das wäre die Richtung der X-Achse. Die Y-Achse führt hinein ins Depot - also wie die Pfeile an B0 und P0. Mit diesem Wissen können wir dann auch unsere Richtungsvektoren bestimmen. Der blaue Pfeil von B0 zeigt genau in Y-Richtung, also sind die Werte für X und Z = 0 und für Y = 1. Bedeutet nichts anderes wie "Von B0 schaue ich 1 nach Y und kein Ding nach links/rechts bzw oben/unten". Bei B1 ist meine Blickrichtung nun aber um 90° gedreht. ich schaue nicht mehr in Y-Richtung, dies wird also 0, Z bleibt unverändert bei 0 und X wird zur minus 1. Ich schaue ja quasi in Fahrtrichtung des Busses - was genau entgegengesetzt der Richtung der X-Achse ist, also -1. Somit haben wir unsere Richtungsvektoren. Die letzte Zahl für die Spurbreite ist standardmäßig für die Strasse bei 3 und für den Gehweg wohl 1.5. An Haltebuchten kann man die PERSON-Lane sicher auch breiter gestalten (dann erhöht man die Kapazität), Fahrzeug-Spuren sollte man aber besser bei 3m belassen.

So, nun will ich mal ein wenig experimentieren. Wir haben nun sehr "gerade" Vektoren hier gehabt und immer 1 oder -1 angegeben. Sieht also nach einem Normalenvektor aus (Länge 1). MUSS das ein Normalenvektor sein, oder kann man auch einfach mal ne 2 da hinschreiben? Probieren wirs aus!



Das war mit 0,2,0 für den Richtungsvektor von B0 -> wie wir sehen, es ändert sich nichts. Wir müssen also NUR eine Richtung angeben und das genügt. Größere Zahlen verformen die Kurven nicht (Gewichtung oder sowas) und normiert muss auch nichts sein. Habe es auch mit 0.1 getestet, auch wenns Ergebnis eigentlich klar war - wieder keine Änderung.

Damit kann ich aber Prima zur nächsten Geschichte überleiten: Wir müssen immer dafür Sorge tragen, dass die Lanes einen geschlossenen Weg von der Ein- zur Ausfahrt bilden. Ich habe einfach mal aus B1 in der zweiten Zeile 0,11,0 gemacht. Das lässt sich zwar bauen und sieht so aus:



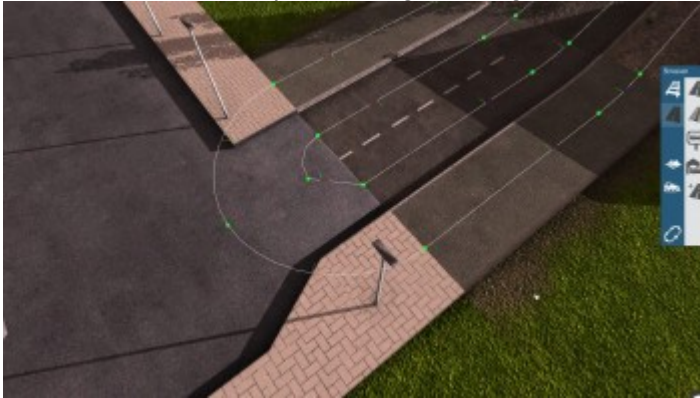
...allerdings zeigen die roten Punkte schon deutlich, dass hier was fehlt. Beim Versuch diese Station einer Linie zuzuweisen schmiert TF dann sang- und klanglos ab. Bei den PERSON- und Cargo-Lanes stellen "freie Enden" allerdings kein Problem dar. Diese wandern ja nur rein auf ihre Warteposition, aber nicht wieder heraus.

Worauf ich nun aber hinaus wollte: Die Punkte sollten immer zusammenhängend sein - dies gilt aber nicht für die Richtungsvektoren! Wir könnten bspw von B1 einfach mal direkt auf B2 zeigen und nicht entgegen der X-Achse. Bei diesem Konstrukt ist der Vektor noch ziemlich simpel zu errechnen - es geht in 45° entgegen X und entgegen Y, wir erhalten also den Vektor -1,-1,0 für B1 in der zweiten Zeile (kleine Anmerkung: Hätten wir normieren müssen, wäre der Vektor in etwa 0.7, 0.7, 0 gewesen). Da wir ja aber auch "gerade" an B2 ankommen wollen, müssen wir die Richtung hier ebenfalls so ausrichten, sonst haben wir am Ende zu B2 hin doch eine kleine Kurve. Bauen wir in der zweiten Bus-Lane Zeile also so etwas hier:

{ { 0.0, 1.0, 0.0 }, { -2.0, -1.0, 0.0 }, { -1.0, -1.0, 0.0 }, { -1.0, -1.0, 0.0 }

} , 3.0 } ,

Dann erhalten wir im Spiel das folgende Ergebnis:



Die Richtungsvektoren können also ruhig unterschiedlich sein. Man sollte allerdings bedenken, dass man sowas lieber nur bei PERSON- oder CARGO-Lanes macht, da die Fahrzeuge sonst sicher ziemlich ruckig fahren werden, was sicher unschön aussieht und die Qualität eurer Konstruktion nur ruiniert. Aber prinzipiell geht es. Ich hab es gerade nochmal getestet. Der Bus fährt problemlos durch. Da B1 bei mir aber der Haltepunkt für ihn ist, fährt er normal in einer Kurve ran und schnappt dann herum in Richtung des 45° Vektors von uns. Wenn er weiter fährt, schnappt er aber dann rauswärts nicht sprunghaft um die Ecke, sondern fährt sie als Kurve. Wenns also nicht gerade ein Haltepunkt ist, sollte das selbst für Fahrzeuge einigermaßen ordentlich aussehen.

Habt ihr nun aber Vektoren, die sich nicht so schön herleiten lassen, dann lässt sich die Richtung auch sehr simpel errechnen. Sagen wir, wir wollen von 1,2,0 zu 5,8,0. Wie muss hier jetzt die direkte Richtung aussehen? Wir ziehen von unserem Ziel-Vektor einfach den Start-Vektor ab: $(5,8,0) - (1,2,0) = (5-1,8-2,0-0) = (4,6,0)$. Da wir nicht normieren müssen, langt diese Form der Angabe schon völlig.

4 Eine Lane erstellen

Wie schon Eingangs angedeutet, habe ich mich hier des Tricks von [@Bandion](#) angenommen mit diesem Hilfskubus. Den habe ich mir zurecht geformt, so dass mir die Linienführung erstmal passend erschien. Das Ergebnis sieht nun erst einmal so aus (dank eines Fehlers und dem damit verbundenen Verschwinden aller Bilder, hier nun also ein nachgestelltes Bild mit "fertiger" Bus-Lane):



Ich habe den Würfel wie gesagt so ausgerichtet, dass sein Ursprung dem der Station entspricht. Denkt auch dran die Skalierung anzupassen! Drückt sicherheitshalber lieber mal bei ausgewähltem Kubus/Linien-Mesh **Strg+A** und wählt die option **Rotation & Scale**. Auch habe ich hier glatt mal die ersten Vertices ausgewählt (hätte auch einer gelangt, geht ja nicht um die Z-Koord hierbei). Wer genau hinschaut, erkennt die Nachahmung unseres Punktes B0. Nun können wir im rechten Panel der 3D-View oben auch die Koordinaten ablesen: 2, -1, 0. Und was haben wir als ersten Node eingetragen? $\{ 2.0, -1.0, 0.0 \}$ - passt also wie die Faust aufs Auge!

Für das Beispiel habe ich extra zwei Linien gebaut. Als erstes möchte die Lane aussen herum bauen, also die große Runde. Wie man das macht sollte eigentlich völlig egal sein. Wenn wir hier von B0 ausgehend die anderen "chronologisch" benennen wollen, so sollte es keinen Unterschied machen, ob wir B0->B1, B1->B2, B2->B3 usw usf angeben oder das auch kreuz und quer machen. Also B3->B4, B0->B1, B2->B3 usw usf. TF legt hier noch unbekannte Punkte neu an und schon vorhandene werden einfach mitgenutzt. Man sollte jedoch auf die Richtung bei Fahrzeug-Lanes Acht geben! Ich denke mal, TF wird wieder abschmieren, wenn da plötzlich eine Edge "rückwärts" kommt, da er so die Linie nicht vervollständigen/berechnen kann.

Ich empfehle dennoch dringend, das Ganze "chronologisch" aufzuziehen. Ihr werdet für jedes "System" in eurem Nodes-Gewusel dankbar sein! Im Beispiel habe ich hier 15 Punkte (an der Haltestelle auf der Geraden ist noch jeweils die Node des Haltepunkts) - für die große Runde würden auch 13 reichen, aber wir benötigen später noch die beiden Verbindungspunkte für die "kleine Runde". Diese Verbindungen wären demnach die Punkte B3 und B9, wenn ich mich nicht verzählt habe. Wir haben schlussendlich also 14 Einträge von B0 bis hindurch zu B14. 15 Punkte, 14 Verbindungen bzw Edges. Wollen wir nun die kleine Runde eingliedern, müssen wir nicht wieder bei B0 beginnen und bis zu B14 alles erneut durchexerzieren, es reicht, wenn wir B3->B15 (der Punkt nach B14 eben ~~Image not found or type unknown~~ wir haben ja hier einen neuen bisher nicht angegebenen) und am Ende B17->B9 angeben (B17 weil wieder ein Haltepunkt dazwischen liegt). Wie er B3 erreicht und wie er von B9 zu B14 kommt, dass haben wir ja schon angegeben und diese Angaben kann er nun wieder nutzen.

Nachdem ich nun alle Koordinaten-Werte abgepinselt habe und auch die Richtungsvektoren angegeben hab, hab ich auch noch ein paar gescheite personNodes hinzugefügt. Zudem gibt es auch eine "invertierte" Runde, so dass die Busse quasi hintenrum fahren und die anderen Plattformen noch nutzen können. Im Spiel sieht das nun wie folgt aus (das Bild von oben mit Person-Lanes):



Der Code dazu ist viel Schreiberei gewesen, aber dank einfachem Ablesen jetzt auch nicht sehr viel mehr. Sieht dann so aus (ergibt wohl wie gesagt nicht genau das Bild von oben, da ein exaktes Nachstellen der Bilder nicht mehr möglich war ^{in der 3D-Modellierung}).

Code: myStation.mdl

[illegible]

Alles anzeigen

Soweit sogut also. Die Wege für alle sind verlegt, aber wie bekommen wir jetzt die Leute von ihren **Person-Lanes** in den Bus auf den **Vehicle-Lanes**? Wer sich das Code-Stück bis zum Schluss angeschaut hat, wird schon den Punkt der **terminals** gesehen haben. Hiermit wird diese Verknüpfung hergestellt.

5 Terminals

Der **terminals**-Eintrag vereint nun schlussendlich unsere Lanes, so dass der Abtransport funktioniert. Ist hier gar nichts angegeben, dann stürzt TF beim Bau ab. Mindestens ein Eintrag muss also vorhanden sein. Und der Prototyp davon sieht so aus:

```
{ vehicleNode = NodeID, personNodes = { NodeID-List }, personEdges = { EdgeID-List } },
```

Hier wieder ersteinmal eine kleine Begriffsklärung: **NodeID** und **EdgeID**... wie ist das zu verstehen? Wenn wir im Code eine Edge angeben, also eine Zeile mit unseren 4 Vektoren und der Spurbreite, dann bekommt diese Zeile selbst eine ID. Die erste Edge/Zeile bekommt die ID 0, die nächste die 1, dann die 2 und so weiter. Diese Form des Zählens reicht auch ÜBER Lanes hinaus! Endet unsere BUS-Lane also beispielsweise bei EdgeID 20, so hat die erste Edge der nächsten Lane (PERSON meinetwegen) NICHT die ID 0 sondern zählt von der 20 aus weiter - also die ID 21.

Und wie wir nun wissen, stehen in jeder Zeile unsere zwei Punkte. Jede Edge besteht aus 2 Nodes. Auch diese werden von 0 beginnend über Lanes hinweg durchnummeriert. Der erste Punkt der ersten Edge hat demnach wieder die ID 0. Die zweite Node der ersten Edge bekommt die ID 1.

Nun zu einem kleinen Punkt, der mich sehr hartnäckig verwirrt hatte. Oben habe ich euch ja gezeigt, wo wir unsere Punkte aus dem Spiel im Code wieder finden. Da hatte ich folgendes geschrieben gehabt um es zu verdeutlichen:

```
{ B0, B1, { 0.0, 1.0, 0.0 }, { -1.0, 0.0, 0.0 }, 3.0 },  
{ B1, B2, { -1.0, 0.0, 0.0 }, { 0.0, -1.0, 0.0 }, 3.0 },
```

Hier habe ich mir die tatsächlichen Punkte notiert. Edge0 geht also vom tatsächlichen Punkt B0 zum tatsächlichen Punkt B1 und Edge1 nun vom tatsächlichen Punkt B1 zum tatsächlichen Punkt B2. Dies entspricht aber NICHT der NodeID Zählweise! Folgende Auffassung der Zahlenvergabe ist also falsch:

```
EdgeID 0 { NodeID 0, NodeID 1, { 0.0, 1.0, 0.0 }, { -1.0, 0.0, 0.0 }, 3.0 },  
EdgeID 0 { NodeID 1, NodeID 2, { -1.0, 0.0, 0.0 }, { 0.0, -1.0, 0.0 }, 3.0 },
```

In Zeile eins für Edge0 stimmt es noch, Edge1 aber zählt die ID's ihrer Nodes konsequent weiter, egal ob der tatsächliche Punkt, den die Node beschreibt, der selbe ist. Während also B1 in Edge0 die NodeID 1 hat, so hat der selbe Punkt B1 in Edge1 jedoch die ID 2! Richtig wäre also dies hier:

```
EdgeID 0 { NodeID 0, NodeID 1, { 0.0, 1.0, 0.0 }, { -1.0, 0.0, 0.0 }, 3.0 },  
EdgeID 0 { NodeID 2, NodeID 3, { -1.0, 0.0, 0.0 }, { 0.0, -1.0, 0.0 }, 3.0 },
```

Hiermit nochmal ein dickes Dankeschön an @Tom, der mir hier die Erleuchtung brachte. Um die NodeID's zu erhalten, braucht ihr also einfach nur die EdgeID verdoppeln für die erste Node bzw die verdoppelte EdgeID um 1 erweitern für die zweite Node. Gestärkt mit diesem Wissen können wir uns nun also die Werte anschauen.

Mit **vehicleNode** gebt ihr die Stelle an, an der euer Fahrzeug halten soll. Dieser Punkt wird auch *nur* bei Straßenstationen benötigt. Bei Bahnhöfen wird das über die Gleise an den Plattformen geregelt. Was ihr hier angebt ist die ID einer Node. Die **personNodes** bestimmen die Zugänge zu euren Wartebereichen. Also über welche Node können die Fußgänger eure Warte-Zone(n) erreichen. Wie hier schon angedeutet, können das auch mehrere Warte-Zonen mit mehreren Anschlüssen sein. Daher erfolgt diese Angabe in Listen-Form (-> {}). Eintragen tut ihr hier (per Komma getrennt) alle benötigten NodeID's eurer Zugänge. Die **personEdges** schlussendlich sind eben jene Wartebereiche. Auch hierbei handelt es sich um eine Angabe in Listen-Form, also per Komma getrennte Werte. Diesmal jedoch gebt ihr eine ganze Edge per ID an.

So, nun habe ich zur Veranschaulichung mal dieses Bild gebastelt:



Dröflzig Millionen Zahlen xD Das schlimme daran ist: Ohne dieses Gewusel fällt das Verstehen ungleich schwerer. Ich versuch es nun also anhand dieser Grafik Schritt für Schritt zu erklären.

(Anmerkung: Habe das Bild im Zuge der Neuerstellung etwas vereinfacht dargestellt und mich auf das für die Erklärungen Nötige beschränkt *faul* !saint. hoffe ihr vergebt mir)

Die **vehicleNode** lässt sich durch abzählen herausfinden. Jede unserer **Edges** hat ja ihre eigene ID. Will ich nun bei unserer ersten großen Runde den Haltepunkt, so ist dies die 6. Node auf dem Bild, zusätzlich Blau eingekreist - also EdgeID 6 (die kleine grüne Zahl neben dem Pfeil). Wenn man die Plattformen von links nach rechts durchnummeriert, dann ist das ist der Bus an Plattform 4 szs. Ich habe hier auch die einzelnen Edges eingetragen, so wie ich sie im Code angegeben hatte. Edge0 zwischen den Punkten Blau1 und Blau2 ist also unsere allererste Zeile im Code (also, die erste angegebene Edge). Danach biege ich nach rechts ab, dann gehts zur Verbindungsstelle/Abzweigung zur kleinen Runde und weiter bis ans "Ende". Hier biege ich schlussendlich wieder nach oben ab und fahre zu unserem Haltepunkt. Ich hoffe, ihr konntet folgen, ich bin jetzt beim eingekreisten Punkt 6. Wir können nun also den tatsächlichen Punkt Blau6, an dem wir halten wollen, in unsere **vehicleNode** umwandeln, indem wir die Edge mit unserem Punkt ermitteln (was wir oben taten -> EdgeID 6) und deren ID verdoppeln (um die NodeID des ersten Punktes der Edge zu bekommen). Unsere **vehicleNode** ist also die 12! Wir hätten wohl auch den zweiten Punkt der 6 Edge nehmen können. Also wäre das dann wie folgt: Die 6. Edge hat die ID 5 und hierbei brauchen wir die zweite Node welche $\text{EdgeID} * 2 + 1$ als ID hätte -> $5 * 2 + 1 = 10 + 1 = 11$. Der Eintrag **vehicleNode = 11** dürfte also eigentlich auch funktionieren. Weiterhin gilt zu bedenken: NUR Edges von "Vehicle-Lanes" (also BUS oder TRAM oder so)

können hier angegeben werden. Also angeben könnt ihr auch "vehicleNode = Käse" - aber ihr wisst, was ich meine :) Auf einer PERSON-Lane wird kein Bus fahren um dort halten zu können

Die **personEdges** definieren nun wie gesagt unsere Wartezonen. Bei mir wäre das nur eine und keine ganze Liste - und zwar die von Punkt Rot19 zu Rot20. Können wir auf dem Bild ja sehr schön ablesen, das ist die Edge neben unserem Haltepunkt auf Plattform4. Nochmal zur Erinnerung: Auch wenn wir Bus-Lanes und Person-Lanes getrennt angeben (bei mir durch blaue und rote Punkte dargestellt), so werden die Edges Und Nodes in einem Rutsch durchgezählt. In meinem Bild habt ihr vllt die letzte Edge gesehen, ich hab's im Code auch als Kommentar drunter geschrieben: 40 ist die letzte Edge bei den Bus-Lanes. Die erste Person-Lane Edge ist nun also die 41! Da ich das im Bild schon komplett durchnummeriert hab, brauchen wir also nur noch die Nummer abgreifen - und die ist 60. Was tragen wir also nun als Wert ein? Nein, nicht 120 - denn wir wollen ja die ID der Edge und keine NodeID! Da muss jetzt tatsächlich die 60 rein. Hier können nun wiederum nur Nodes von PERSON- oder CARGO-Lanes angegeben werden.

Bleiben nur noch die **personNodes** übrig. Diese werden nun wieder wie die **vehicleNodes** gezählt - also EdgeID ermitteln und verdoppelt und ggf um 1 erweitern. Dies sind die Nodes, über die unsere lieben TF-Bürger ihre Wartebereiche (die **personEdges**) erreichen. Nehmt ihr hier einen Falschen, dann verschwindet euer Männchen an diesem Punkt und wird ab da schon zur Bushaltestelle gezählt. Der scheint dann intern bissl zu joggen, jedenfalls dauerts ne Weile bis er irgendwann doch mal im Wartebereich wieder auftaucht und sich nen Platz zum hinstellen sucht. Diese Node sollte also sinnigerweise der direkte Ankerpunkt der definierten Gehwege mit den Wartebereichen sein. Wenn wir uns das Bild nun genau betrachten, so ist das die erste Node der Edge 60. Also brauchen wir den Wert 120. Und warum genau das nicht unbedingt die beste Idee war, erzähle ich euch später. Angegeben werden können auch hier nur Edges von PERSON- oder CARGO-Lanes.

Unser fertiges Terminal sieht also so aus:

```
{ vehicleNode = 12, personNodes = { 120 }, personEdges = { 60 } },
```

Diese Prozedur können wir also nun für jeden Haltepunkt wiederholen. Die erste Node aus Edge 16 wäre also der zweite Haltepunkt an Plattform 2. Dazu gehört Edge 54 als Wartebereich und ebenso die erste Node von Edge 54 als Zugang. Resultiert also in 32,108,54.

6 Ausrichtung der Passagiere

Bei den folgenden drei Plattformen kommen wir noch zu einer kleinen Besonderheit, auf die geachtet werden sollte. Bei den ersten beiden hat das zufällig gepasst. Nun aber kommt unser Bus aus der anderen Richtung. Und es gilt bei den Wartebereichen zu beachten: Die Wartebereichs-Edge-Richtung sollte der Einfahrts-Edge-Richtung des Fahrzeuges entsprechen. Wenn ihr im Spiel mal die Lanes mit AltGr+L anschaut, dann wandern da so kleine Koordinaten-Achsen eure Edges entlang. Das tun die in genau einer Richtung: Vom Start-Vektor zum Ziel-Vektor. Bei unseren ersten beiden Haltepunkten kommen wir von unten und fahren nach oben - so bewegen sich auch die kleinen Achsen. Die Fußgänger kommen aus genau der gleichen Richtung. Beide Achsen-Symbole bewegen sich in der selben Richtung entlang ihrer Edges - die wartenden

Passagiere schauen zum Bus hin. Oder man kann auch etwas allgemeiner sagen: Die Passagiere warten in Laufrichtung links. Ja, das so zu formulieren trifft es glaube ich deutlich besser. Bei unseren drei letzten Haltestellen kommen wir ja aber von oben und fahren nach unten. Die Passagiere müssten hier also nach rechts schauen, und nicht nach links. Zum Glück sind links und rechts rein relative Angaben. Anstatt unsere **personEdges** also von unten nach oben zu definieren (am Bsp der Plattform1 also von Punkt Rot10 nach Rot11), definieren wir sie also einfach andersherum (sprich, von Rot11 zu Rot10). Wer mein Bild genau anschaut, der wird genau diesen umgekehrten Pfeil auch wieder erkennen.

Und an diesem Punkt möchte ich euch nun erklären, warum diese personNode-Angabe, wie wir sie oben vornahmen, nicht unbedingt die beste Idee war. Dadurch, dass wir nun die Edge-Definition umgedreht haben, ist auch unsere angegebene NodeID "gewandert". jetzt beschreibt diese nämlich nicht mehr den unteren Punkt, sondern den oberen. Die allgemeine Beschreibung war ja "zuerst definierte Node der Edge". Und da wir die Edge nun von oben nach unten statt von unten nach oben definiert haben, ist nun die zuerst definierte Node eben die obere und nicht mehr die untere. Wir sollten jetzt also diese Angabe ändern, indem wir die zweite Node der Edge definieren, welche die NodeID 121 hätte. Um dieses hin und her gänzlich zu vermeiden, könnt ihr auch folgenden Tipp beachten: Nutzt nicht eine Node der personEdge, sondern die der Edge davor. Die "Zubringer"-Edge hätte in unserem Beispielfalle die ID59 gehabt und wurde meinerwegen "normal" von unten nach oben definiert. Dann bräuchten wir also die zweite Node der 60. Edge mit der ID 59, also NodeID 119. Dann kann man die eigentliche Edge drehen wie man will und muss nichts ändern.

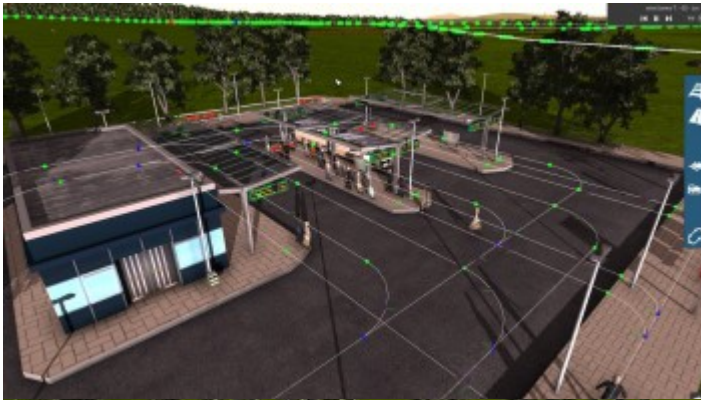
Weiterhin hat @laemi noch herausgefunden gehabt, dass die Passagiere die Haltestellen nicht wechseln. Es war ihnen nicht möglich von einer personEdge zur anderen zu gelangen um auf eine andere Bus-Linie umzusteigen. Dies lag daran, dass ich die Person-Lanes so angelegt hatte, dass man unten von der "Haupt-Lane" her in einer Kurve zu den Plattform-Lanes lief und diese Kurve die Sims in eine Richtung zwangen - nämlich gen Ausgang. Folgende Lösungen des Dilemmas gab es also: Entweder eine zweite Kurve in die gegenrichtung implementieren, so dass die Sims auch dorthin wandern können. Oder einfach ganz auf eine Kurve verzichten und die Lanes im rechten Winkel abzweigen lassen. Oder aber bei den Plattformen nochmal separate Verbindungs-Lanes ziehen. @laemi hatte hierbei zwei Möglichkeiten ausprobiert: Oberhalb der Plattformen und unterhalb davon. Wir hatten zwischenzeitlich noch eine sechste Haltestelle ganz rechts am Rand eingebaut und mit der unteren Verbindungs-Lane liefen die Sims nun plötzlich nicht mehr unten den Gehweg (was ich eben als "Haupt-Lane" bezeichnete) entlang, sondern zweigtem am Gehweg hinten hin zu Plattform 5 ab und von dort aus über die Straße zu Plattform 6. Mit den Verbindungen oberhalb der Plattformen liefen sie wieder wie gewünscht. Bedenkt sowas also bei eurer "Streckenführung". In diesem Falle müsste man dann im Übrigen auch Gebrauch von der "Listenfunktion" für die **personNodes** machen. Man muss einerseits die Zugangs-Node unterhalb der Plattformen angeben sowie die oberhalb.



Aktion. Funktioniert einwandfrei, die Kurvenradien b. Aber schaut selbst einmal:

Ein Test mit voller Linienbelegung.

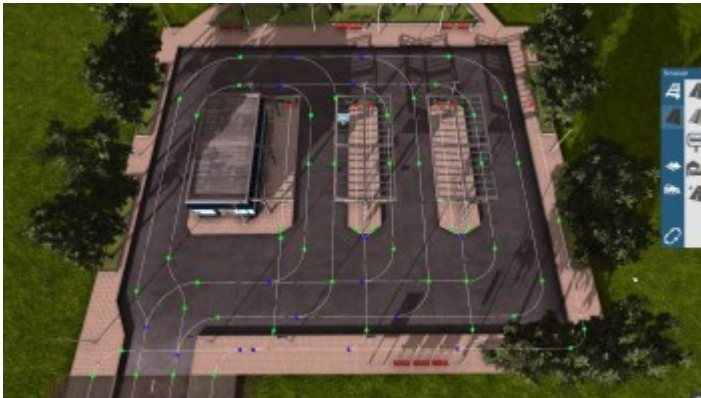
<https://www.transportfever.net/lexicon/entry/105-transport-network-provider-lanes-und-ihre-bedeutungen/>



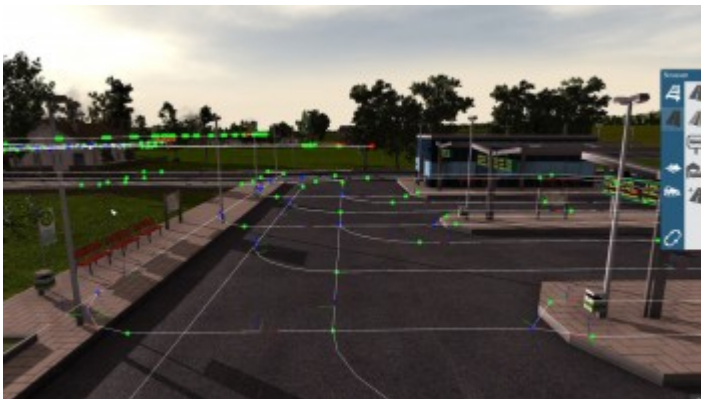
und die Leute stehen noch falsch herum.



Jetzt stehen auch die Leute richtig rum !D

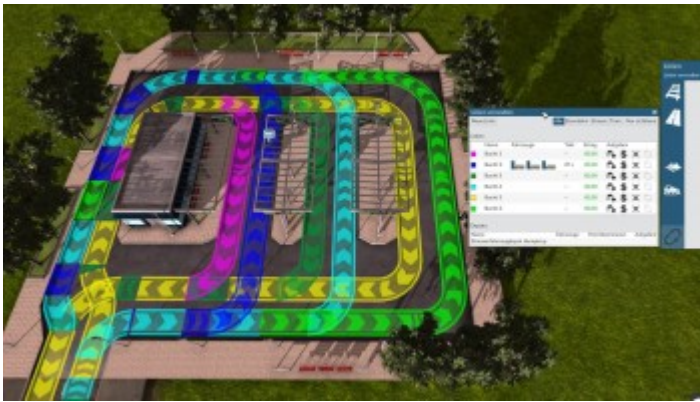


Eine 6. Linie wurde hinzugefügt, Bandion durfte



Die Leute müssen nicht mehr in luftigen Höhen

schweben und wurden auf Bodenniveau abgesenkt.



Alles funktioniert ! thumbs up

Was bleibt zu sagen? Folgende Probleme treten bei dieser Aufstellung auf:

- Die Fußgänger schweben beim Überqueren der Fahrbahn etwas in der Luft, da ich hier keine Extraknoten eingefügt hatte, die sie auf Bodenniveau setzen. Fällt aber eigentlich kaum auf. Weiterhin fahren die Busse durcheinander hindurch. Beispielsweise kurz vorm Ausgang bei Edge32 zwischen den Punkten Blau31 und Blau18. Hier kreuzt die Edge19 und die Busse fahren schlicht durcheinander hindurch. Was allerdings auffiel, gleich nebeneinander beim Zusammentreffen der Edges 32 und 12 warten sie aufeinander. Ich würde also schlussfolgern, dass sie bei gemeinsam genutzten Punkten durchaus aufeinander warten. Wäre interessant, ob das bei Fußgängern auch funktioniert. Ich empfehle also, solche "berührungslosen" Kreuzungen zu vermeiden. Versucht am besten das Linienlayout so zu gestalten, dass bei sich kreuzenden Edges diese aufgetrennt werden und sich in einem gemeinsamen Punkt treffen. Es kann allerdings auch passieren, dass dann die Linienführung durcheinander kommt *grübel* Müsste man probieren - falls ihr da beim selber tüfteln was heraus bekommt, schreibt einfach in die Kommentare unten !)

Weiterhin empfehle ich dringsten euren Code ausführlich zu kommentieren! Einen Kommentar beginnt ihr mit -- und alles dahinter bis zum Zeilenende gilt als Kommentar. Nun denn, dann hoffe ich, dass ihr hiermit etwas anfangen könnt. Ich fand es sehr spannend, oft auch kurios - aber es ist vollbracht. Nun freue ich mich über zahlreiche neue Stationen in der Webdisk !

MfG, euer Mo...