loadIndicators und LoD's

Inhaltsverzeichnis

- 1 Einleitung
- 2 Beispiele
  - 2.1 Stückgut
  - 2.2 Schüttgut
- 3 Was es im Zusammenhang mit LoD's zu beachten gilt
  - 3.1 Abgrenzung der loadIndicators zu anderen Elementen der .mdl in Bezug auf LoD's
  - 3.2 Entwicklung des eigentlichen Problems anhand eines Beispiels
  - 3.3 Die Lösung
- 4 Ergänzung: Berechnungen zum Bestimmen der Indicator-Übergänge

Zuerst ein paar allgemeine Worte, die mir im Original-Eintrag fehlen.

1 Einleitung

LoadIndicators werden genutzt um Fracht visuell darzustellen. Ein leerer Eisenerz-Waggon sieht anders aus wie ein halb oder ganz gefüllter bspw. Die Anzahl der Baumstämme auf einem Rungenwagen ist je nach Füllstand mal größer und mal kleiner. Oder auch bei neuen Ideen mit Güterwaggongs und Containern oder Kisten als Ladung, Autos oder oder oder. Die Anwendung ist natürlich vielfältig. Bei der Aufzählung kristallisieren sich auch schon zwei verschiedene Typen von loadIndicator's heraus:

- einmal "Schüttgut" und
- desweiteren "Stückgut"

2 Beispiele

Im Spiel (bzw in der mdl) werden diese beiden mit dem type-Parameter bestimmt. Möchte man "Schüttgut", so muss der type="LEVEL" sein, bei Stückgut type="DISCRETE". Hier einmal zwei Beispiele aus dem Original - der Rungenwagen von 1850 und der für Erze aus dem selben Zeitraum.

2.1 Stückgut

Code

```
capacities                                      =                                       {
                {

                        {

                                        {

                                                6,

                                        },
                                },
                        },

                        },
                },

                },
        },
```

Alles anzeigen

Dieser Waggon kann eine einzige Sorte Waren transportieren und nutzt den DISCRETE-type. Da es um Holz geht, macht das Sinn, denn Holzstämme sind ja einzelne Stücke. Wie man sieht, sind die loadIndicator's Teil des capacity-Blocks der mdl und enthalten jeweils den type sowie den parameters-Block. Je nach type ist der parameters-Block anders aufgebaut. Hier bei Stückgut gibt man quasi jedes Stück (eigentlich das Mesh - welches auch 3 oder 4 Stück... darstellen kann) händisch an. Beim Modellieren legt ihr also ein Mesh an, welches meinetwegen zwei Baumstämme darstellt und platziert das ordentlich auf dem Waggon. Ihr exportiert am Ende alles und schaut nach, welche ID nun euer Baumstämme-Mesh hat. Dies geht sehr elegant mit **anton95**'s TF ID Counter - einfach die gewünschte mdl damit laden und ihr seht in Listenform alle Meshes und Gruppen mit ihren ID's. Gehen wir jetzt davon aus, dass euer Baumstämme-Mesh die ID10 hat, dann müsst ihr eben die 10 dort eintragen. "Dort"?

Der Aufbau sieht hierbei wie folgt aus: Ihr könnt beliebig viele Zeilen angeben und TF "verteilt" sie gleichmäßig ingame. Habt ihr bspw 4 Zeilen drin, dann wird sich alle 25% Ladevolumen etwas ändern, bei 3 Zeilen alle 33% und bei 5 Zeilen eben alle 20% usw usf (detaillierte Berechnungen sind am Ende des Beitrages zu finden). Die erste Zeile ist der "Leer-Zustand", die letzte der Volle. In eine Zeile wird per Kommata getrennt angegeben, was **versteckt** werden soll! Wie gesagt, was ihr hier angebt, wäre die Mesh-ID. In unserem kleinen Baumstämme-Beispiel mit der ID 10 müsste man also eine 10 eintragen und in die 2. Zeile einmal garnix. Um ein etwas komplexeres Beispiel aufzuzeigen, möchte ich meinen Bus AGG300 hernehmen. Hier ein Auszug aus der mdl mit den Passagier-Meshes und ihren ID's:

Code

```
1:                              id       =       "vehicle/bus/AGG300_Lod_0_bogie_passangers_ba
2:                                    id    =    "vehicle/bus/AGG300/AGG300_Lod_0_passengers_
3:                                    id    =    "vehicle/bus/AGG300/AGG300_Lod_0_passengers_
4:                                    id    =    "vehicle/bus/AGG300/AGG300_Lod_0_passengers_
5:                           id      =      "vehicle/bus/AGG300_Lod_0_bogie_passangers_center
6:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_center_
7:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_center_
8:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_center_
9:                             id      =      "vehicle/bus/AGG300_Lod_0_bogie_passangers_fron
10:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_front_
11:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_front_
12:        id = "vehicle/bus/AGG300/AGG300_Lod_0_passengers_front_l3.msh"
```

Alles anzeigen

Der Bus besteht aus drei Teilen und hat somit für jeden Teil drei "Levels" der Befüllung. Konkret sieht das bei mir so aus, dass ...l1 2-3 Personen sind die irgendwo sitzen, ...l2 sind 2 weitere, die woanders sitzen usw usf. Mein loadIndicator sieht dabei nun folgendermaßen aus:

Code

```
loadIndicators                                  =                                   {
                        {

                                                                                  {  4,  8,  12,  3


                                    },

                                },


                                },
                        },
```

Alles anzeigen

Die ID's 4, 8 und 12 sind jeweils die ...l3's für jeden Teil, ID's 3, 7 und 11 die ...l2's und 2, 6 und 10 für die ...l1's. In der ersten Zeile ist **alles** angegeben, es wird also **nichts** angezeigt (bzw alles versteckt). In Zeile zwei fehlen die ID's für die ...l1's - diese werden also hier dann angezeigt, der Rest ist immernoch versteckt. So geht es eigentlich immer weiter, bis am Ende die Leere Klammer anzeigt, das nichts mehr versteckt wird.

2.2 Schüttgut

Kommen wir nun einmal zu dem Schüttgutwaggon:
Code

```
capacities = {
                {
                                }
                            }
                },
                {
                                }
                            }
                },
            },
```

Alles anzeigen

Einerseits sehen wir hier, dass man den loadIndicator für **jede** Ware seperat angeben muss, andererseits auch, dass mit LEVEL alles anders aussieht. Man gibt nur eine ID an und eine Höhenangabe, das wars. Die ID sollte nun klar sein - das ist bei einem solchen Waggon unsere Mesh-ID vom Erz- oder Kohlehaufen. Die Maximale Höhe gibt nun an, wie hoch dieser sich nun maximal füllt. Das bedeuted konkret folgendes: Ihr habt ein Mesh für den Kohleberg meinetwegen gebaut und das so platziert, dass es den "0-Zustand" symbolisiert. Dieser muss nicht zwangsläufig so liegen, dass er unsichtbar ist, TF scheint den Leerzustand nicht anzuzeigen. Nun ist euer Waggonboden bei bspw z=1.2 (also 1,2m hoch) und dort in etwa platziert ihr euer Mesh. Die Waggonwand endet nun bei (um die hier angegebenen 1.4 nutzen zu können) einer Höhe von z=2.7 - hat also eine Höhe von 1,5m. Der hier angegebene maxHeight-Wert gibt nun also an, wieviel nach oben das Mesh maximal verschoben werden darf - wo also euer "Vollzustand" sein soll. Wir wollen ihn bei einer Höhe von 2.6m haben (also knapp unterm Rand) und geben nun den Unterschied - nicht die Absolute Höhe an! Da unser Mesh bei einer Höhe von 1.2m liegt und auf maximal 2.6 steigen soll, geben wir also nicht die 2.6 an, sondern 1.4 (Meshposition 1.2 + Höhenverschiebung 1.4 = gewünschte Füllhöhenposition 2.6). Je nachdem, wieviel ein Waggon fassen kann, wird TF auch hier wieder die Schritte anpassen. Bei einem Fassungsvermögen von 4, wird er bei einer Maximalhöhenverschiebung von 1.4 also das Mesh je geladener Einheit um 0.35 (1.4/4) in die Höhe hieven.

Kleine Randnotiz: Falls man nicht nur eine simple nach oben geschobene Fläche möchte, kann man natürlich auch hier DISCRETE verwenden. So könnte man bspw verschiedene "Haufen" darstellen - die halt immer Mehr werden. Auch habe ich schon bei Waggongs mit schrägen Seitenwänden von Problemen gelesen, weil dann das Ladungs-Mesh, was immer gleichbreit ist, durch die Seiten tritt. Auch hier könnte man also verschiedene Einzel-Meshes für jede Höhe bauen - mehr Aufwand, aber ein akurateres Ergebnis.

3 Was es im Zusammenhang mit LoD's zu beachten gilt

So, damit sollten die Grundlagen geklärt sein. Nun zum eigentlich Anliegen für dieses Eintrag: loadIndicator in Verbindung mit LoD's!

3.1 Abgrenzung der loadIndicators zu anderen Elementen der .mdl in Bezug auf LoD's

Was gibt es generell bei LoD's alles zu beachten? Gruppen kann man für jedes LoD einzeln neu einstellen (Thema fakeBogies), aus welchen Meshes jedes LoD besteht natürlich auch. Was allerdings **nicht** "pro LoD" angeben wird sind ... unsere loadIndicator's. Hier gibt es nur eine Angabe für alle LoD's in den MetaDaten. Wieso führt das zu Problemen? Dazu ein kleiner Auszug aus der Entwicklung meines Busses AGG300.

3.2 Entwicklung des eigentlichen Problems anhand eines Beispiels

Der Aufbau des Busses gestaltete sich grob in 5 Bereiche: Front-, Center- und Back-Gruppe sowie die Faltenbälge vorn und hinten. Und das Prinzip von LoD's ist ja Detailverringerung. Ich habe also beim Haupt-LoD viele Einzel-Meshes in den Gruppen (Türen, Rampen, Blinker... alles was sich so animiert bewegt muss ja als seperates Mesh vorhanden sein) welche im Distanz-LoD schlicht rausfliegen. Um es einmal an der Back-Gruppe zu verdeutlichen, hier die jeweiligen Auszüge für LoD0 und LoD1 (Je LoD werden die ID's von 1 an beginnend gezählt. Endet LoD0 also bei ID 20, so fängt LoD1 wieder bei 1 an zu zählen - das wird uns später retten).

LoD0-Back:

Code

```
13:                                          id          =          "vehicle/bus/AGG300_Lod_
14:                                                                 id        =        "vehicl
15:                                            id     =     "vehicle/bus/AGG300/AGG300_Lod_0_
16:                                         id     =     "vehicle/bus/AGG300/AGG300_Lod_0_blin
17:                                            id     =     "vehicle/bus/AGG300/AGG300_Lod_0_
18:                                         id     =     "vehicle/bus/AGG300/AGG300_Lod_0_blin
19:                                                        id       =       "vehicle/bus/AGG3
20:                                         id     =     "vehicle/bus/AGG300/AGG300_Lod_0_body
21:                                      id     =     "vehicle/bus/AGG300/AGG300_Lod_0_body_ba
22:                                                                 id        =        "vehicl
23:                                                  id      =      "vehicle/bus/AGG300/AGG300_
24:                                                                  id        =        "vehicl
25:                                                  id      =      "vehicle/bus/AGG300/AGG300_
26:                                                                  id        =        "vehicl
27:                                                       id       =       "vehicle/bus/AGG300/
28:                                                       id       =       "vehicle/bus/AGG300/
29:                                                       id       =       "vehicle/bus/AGG300/
30:         id = "vehicle/bus/AGG300/AGG300_Lod_0_ventilator.msh"
```

Alles anzeigen

Wie ihr seht - jede Menge einzelne Meshes. Bei Lod1 sieht das nun anders aus:

Code

```
13:                                         id          =          "vehicle/bus/AGG300_Lod_
14:                                                                id        =        "vehic
15:        id = "vehicle/bus/AGG300/AGG300_Lod_1_body_back.msh"
```

Wo ist da nun unser Problem? Nun, ich hatte zuerst meine Passanger-Meshes ans Ende der children-List in der .grp gesetzt, weil ich zu faul war, die Event-ID's für die Animationen wieder alle zu ändern. Füge ich ganz oben was ein, ändern sich die ID's darunter natürlich und ich muss bei Animations-Events die ID's anpassen. Da ich das schon zur genüge getan hatte (Faltenbälge und Änderungen am Mesh sind ein Freudengarant xD), hab ich die nun Also einfach ans Ende gebastelt und gut. Zu der Zeit hatte ich noch garkein LoD1 - alles kein Problem. Dann kam aber eben LoD1 dazu und es fing an. TF schmiert einfach ab! Keine Fehlermeldung, nichts. Hat mich einige meiner eh schon seltenen Haare gekostet, aber ich hatte zum Glück eine gute Intuition. Was war nun also das Problem? Gehen wir davon aus, ich hab die Passenger-Meshes ans Ende geknallt gehabt, dann hätten sie in LoD0 die ID's 31, 32 und 33 gehabt. Diese hätte ich dann auch bei den loadIndicator's angegeben gehabt. Bei Lod1 hab ich darauf schlicht verzichtet, weil es ja eh nicht zu sehen ist - Fehler 1. TF versucht nun (da es eben keine LoD-basierte Angabe gibt) auch bei LoD1 die ID's 31-33 zu verstecken. Problem: Soviele Meshes hab ich bei LoD1 garnicht. Gut, Also erster Lösungsversuch: An die Spitze damit! Man muss dazu sagen, vor der endgültigen Problemlösung hatte die Back-Group ganz oben gestanden, also bei 1 angefangen. Dies berücksichtigend, hätten meine Passenger-Meshes nun also die ID's 2-4 (1 ist die Group). Im LoD1 (auch hier ursprünglich ganz oben) sieht es genauso aus - perfekt!

Naja, fast. Das ganze passt jetzt zwar bei der ersten Gruppe, bei Gruppe 2 fängts aber schon wieder an. In LoD0 hat meine erste Gruppe 17 Meshes in sich vereint, endet also bei 18 und Gruppe Center beginnt bei 19. Die dort "ganz oben" platzierten Passenger-Meshes hätten nun also die ID's 20-22. Bei LoD1 haben wir ja aber schon gesehn, dass dort bei weitem nicht soviel in den Gruppen drin ist. Gruppe Back enthält gerade einmal 2 Meshes - Gruppe Center beginnt hier also schon bei ID 3 - wenn wir unsere 3 Passenger-Meshes (die wir trotz unsichtbarkeit doch erstellen müssen - ich hab einfach ein einziges Dreieck dafür genommen) mit einbeziehen, welche ja dann auch in der ersten Gruppe sind, wären wir also bei ID 6. Die Passenger-Meshes für die Center Gruppe hätten nun also die ID's 8-10. Im loadIndicator haben wir ja nun aber - von LoD0 ausgehend - die ID's 20-22 eingetragen -> Fehler 2.

3.3 Die Lösung

Wir haben nun also schonmal festgestellt, dass die "load-Mesches" am besten ganz oben platziert werden. Aufgrund unterschiedlicher Gruppen-Stärken, müssen wir diese aber aus den Gruppen herauslösen! Ich musste Nun natürlich auch die Gruppen nachbauen, habe also für die Passagier-Meshes extra Gruppen angelegt, welche ich genauso wie die bisherigen Gruppen mit fakeBogies in die Spur zwang. Somit eierten meine Passagiere in Kurven nicht in der Luft herum, sondern sind dort, wo sie hingehören. Das Prozedere wiederholte ich auch für LoD1 (generell gesagt: für jedes LoD wird das gebraucht) und ich platzierte diese Passenger-Gruppen nun jeweils ganz an die Spitze der cildren-Listen. Sieht konkret dann bei mir so aus:

Code

```
children                                        =                                       {
                    {
                                                    id   =   "vehicle/bus/AGG300_Lod_0_bogie_passa

.0.0.0.0.1.0.0.0.0.0.0.0.1.0.0.0,-8.944.0.0.0.1.0,
                                },


                                        id  =  "vehicle/bus/AGG300_Lod_0_bogie_passangers_center.grp",

.0.0.1.0.0.0.0.0.0.0.1.0.0.0,-1.33646.0.0.0.1.0,
                                },


                                                id  =  "vehicle/bus/AGG300_Lod_0_bogie_passangers_front

0.0.0.0.0.1.0.0.0.0.0.0.0.1.0.0.0,7.605.0.0.0.1.0,
                                },

                    },

                    {


.0.0.0.0.1.0.0.0.0.0.0.0.1.0.0.0,-8.944.0.0.0.1.0,
                                },

{
```

Alles anzeigen

Bei LoD1 genau das selbe. Dann wie gesagt noch die fakeBogies kopieren und mit den neuen ID's ausstatten, damit alle Gruppen gleichmäßig bewegt werden und ein abschließender Blick in anton95's ID Tool bestätigt unsere bemühungen.

LoD0:

Code

```
1:                                id       =       "vehicle/bus/AGG300_Lod_0_bogie_passangers_ba
2:                                      id    =    "vehicle/bus/AGG300/AGG300_Lod_0_passengers_
3:                                      id    =    "vehicle/bus/AGG300/AGG300_Lod_0_passengers_
4:                                      id    =    "vehicle/bus/AGG300/AGG300_Lod_0_passengers_
5:                              id      =      "vehicle/bus/AGG300_Lod_0_bogie_passangers_center
6:                                 id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_center_
7:                                 id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_center_
8:                                 id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_center_
9:                               id      =      "vehicle/bus/AGG300_Lod_0_bogie_passangers_fron
10:                                 id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_front_
11:                                 id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_front_
12:                                 id    =   "vehicle/bus/AGG300/AGG300_Lod_0_passengers_front_
13:                                           id          =          "vehicle/bus/AGG300_Lod_
...
```

Alles anzeigen

LoD1:

Code

```
1:                              id       =       "vehicle/bus/AGG300_Lod_1_bogie_passangers_ba
2:                                    id    =    "vehicle/bus/AGG300/AGG300_Lod_1_passengers_
3:                                    id    =    "vehicle/bus/AGG300/AGG300_Lod_1_passengers_
4:                                    id    =    "vehicle/bus/AGG300/AGG300_Lod_1_passengers_
5:                           id      =      "vehicle/bus/AGG300_Lod_1_bogie_passangers_center
6:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_1_passengers_center_
7:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_1_passengers_center_
8:                               id    =   "vehicle/bus/AGG300/AGG300_Lod_1_passengers_center_
9:                            id       =      "vehicle/bus/AGG300_Lod_1_bogie_passangers_fron
10:                              id    =   "vehicle/bus/AGG300/AGG300_Lod_1_passengers_front_
11:                              id    =   "vehicle/bus/AGG300/AGG300_Lod_1_passengers_front_
12:                              id    =   "vehicle/bus/AGG300/AGG300_Lod_1_passengers_front_
13:                                      id          =          "vehicle/bus/AGG300_Lod_
...
```

Alles anzeigen

Voila! Unsere ID's stimmen in beiden LoDs überein - die Abstürze sind Geschichte !D Image not found or type unknown

Ich hoffe, der Beitrag ist verständlich und hilft vorallem anderen, diese Probleme zu lösen.

4 Ergänzung: Berechnungen zum Bestimmen der Indicator-Übergänge

Ich möchte noch eine Ergänzung anfügen. Mich beschäftigte gerade die Frage, wie TF die Verteilung löst. Also, wenn ich 4 Zustände beim loadIndicator einstelle, wann wird der Übergang vollzogen? Bei meinen AG(G) Bussen habe ich 4 DISCRETE leves angegeben. Also 3 eigene und der {} default. Jetzt konnte ich Folgendes beobachten:
Beim Einsteigen wurden meine Passenger-Meshes wie folgt angezeigt:

5/30 -> **keine Anzeige** wechselt zu **l1 Anzeige**
15/30 -> **l1 Anzeige** wechselt zu **l2 Anzeige**
25/30 -> **l2 Anzeige** wechselt zu **l3 Anzeige**

Meine Erwartung wäre zuerst in 25% Schritten gewesen, da wir ja 4 Zustände angegeben haben, scheinbar zählt aber der {} default nicht mit. Ergo haben wir 3 Schritt, was bei 30 Passagieren bei jeweils 10 Eingestiegenen Einen Anzeige-Wechsel hervorrufen sollte. Also 10, 20 und 30. Scheinbar wird das aber noch etwas "mittig verschoben". Sprich, jeweils um einen halben Abstand nach vorn verschoben. Die Übergänge befinden sich also wohl bei

Code

```
Kapazität
level                                *                                  ------------
            (levels - 1)       (levels - 1) * 2
```

Hätten wir für meine Beobachtung von eben glaube ich eine Bestätigung. Für l1 würden wir also einen Schwellenwert von 1 * 30/3 - 30/6 = 1*10-5 = 5 erhalten - stimmt. l2 wäre dann 2 * 30/3 - 30/6 = 20-5 = 15... passt auch soweit.

Gut, will ich also mal ein Level rausnehmen, so dass wir nur noch 3 haben. Laut Rechnung müssten die Grenzen nun also bei *rechne* 1 * 30/2 - 30/4 = 1*15-7.5 = 7.5 und 2 * 30/2 - 30/4 = 30-7.5 = 22.5 liegen. Joa, fix mdl umtüdeln und Spiel laden und schauen *mom*...

Jawoll! Ist tatsächlich nachvollziehbar. Reaktionszeit bedingt (Mesh verschwindet und der alte Mann drückt Pause ^^) hab ich beim Entladen (20/30, also nur eine Schwell) bei 5 gestoppt und beim Beladen auf 30/30 bei 9 und 24. (5+9) / 2 = 7 - könnte man also von einem Abrunden ausgehen mit 2 Passagieren Reaktionszeit. Übertragen auf Schwelle 2 bei 22.5 (bzw abgerundet 22) wäre das also die 24 - mensch, das passt ja xD Ehrlich, das war jetzt nicht gefaked lol - bin selber gerade überrascht.

Jetzt mal eine Grenzwert-Betrachtung: Was passiert bei nur einer Stufe. Also ich mein default und eine eigene. Rein rechnerisch müsste es dann bei 1 * 30/1 - 30/2 = 30-15 = 15 schnappen. Mal schauen... Oha, 12 und 18 Passagiere beim Pausieren gemessen - ihr seht, ich altere im Minutentakt xD nun schon 3 Sekunden Reaktionszeit, aber es passt. Wenn man nur den default {} angibt, müsste es laut dieser Formel - sofern nicht abgefangen - aber dann zu einem Fehler kommen (Division durch 0). Mal schauen :P Image not found or type unknown

Oha, Wird scheinbar korrekt behandelt und der loadIndicator-Eintrag missachtet. Bus fährt wunderbar, nur dass immer alles angezeigt wird. Nun gut, ich fand es sehr interessant, ich hoffe, ein paar andere auch :) Image not found or type u