Assets: Aufbau der Dateien

Inhaltsverzeichnis

- 1 Hinweis zu Kategorien
- 2 Ordnerstruktur
- 3 Aufbau der *.lua
  - 3.1 type
  - 3.2 buildMode
  - 3.3 category
  - 3.4 availability
  - 3.5 description
  - 3.6 models
  - 3.7 order
  - 3.8 skipCollision
- 4 Beispiel

Assets sind mit Build 7554 (Changelog) hinzugekommen. Diese besitzen eine eigene Dateistruktur, die eingehalten werden muss. Diese wird im Folgenden erklärt.

[box]

1 Hinweis zu Kategorien

Assets können in selbst erstellte Kategorien eingeteilt werden. Die Community einigte sich auf eine Liste von zu verwendenden Kategorien. Diese ist im Eintrag Assets: Kategorien einsehbar. Auch können von dort die benötigten Icons für die Kategorien geladen werden.
Sollte euer Asset in keine dieser Kategorien passen, könnt ihr euch im Forum Asset-Kategorien Hilfe holen. Dort könnt ihr auch, falls ihr einen Mod mit sehr vielen Assets erstellt, das "Einverständnis" für eine neue Kategorie einholen.
Ich werde alles Nötige für eine neue Kategorie hier, der Vollständigkeit halber, mit ausführen.
[/box]

2 Ordnerstruktur

Assets befinden sich bei Train Fever im Verzeichnis `res/assets`. Darin können eigene Unterordner erstellt werden. Für die Benutzeroberfläche werden Icons benötigt. Diese befinden sich im Ordner `res/textures/ui/assets`. Die Icons sind ggf. im selber Unterordner unterzubringen und gleich zu benennen. Die Grafiken müssen im *.tga-Format ohne Kompression vorliegen.
Für die Kategorisierung der Assets wird/werden zusätzlich Icon für die jeweilige Kategorie(n) benötigt [Siehe Hinweis zu Kategorien]. Diese sind in einer besonderen Form zu benennen. Außerdem müssen diese Icons - sofern es nicht die TF-eignenen Kategorien sind - stets mitgeliefert werden!
Code

```
res
                                                                ?
                                                                                ?
                                                            ?



                    ?? assetBeispiel.tga
```

Alles anzeigen

Die Assets dürfen beliebig benannt werden, müssen aber auf `.lua` enden. Vermieden werden sollten allerdings Sonderzeichen wie äöüß. Diese führen immer wieder zu Problemen. Zudem ist auf GrOß- und kLeiN-Schreibung zu achten! Zudem sollten Assets keinen Namen der Form `asset_category_*.lua` erhalten. Diese Namen beißen sich mit den UI-Icons für die Kategorien und werden zu erheblichen Problemen führen!

3 Aufbau der *.lua

Nun zum Wichtigsten, dem Aufbau der Asset-Dateien. Es handelt sich dabei um eine Scriptdatei, die die Eigenschaften des Assets beinhaltet. Der Quellcode zeigt die Anordung der einzelnen Eigenschaften in der Datei. Die Eigenschaften an sich sind anschließend aufgeführt.
Code

```
function                                                                    data()
return                                                                          {
 type                        =                          ...                     ,
 buildMode                       =                        ...                   ,
 category                       =                         ...                   ,
 availability                      =                        ...                  ,
                     description                          =                   ...
 models                         =                        ...                     ,
 order                         =                         ...                     ,
 skipCollision                       =                       ...                 ,
}
end
```

Alles anzeigen

3.1 type

Der Typ bestimmt die Oberkategorie, in die der Asset eingeordnet wird. Gültig sind dabei zwei Werte:

- `type = "DEFAULT"` - Standardobjekt, welches frei platziert werden kann. Es snappt dabei an kein Objekt in der Nähe an.
- `type = "TRACK"` - Objekt, welches an Gleisen einsnappt wie Wegpunkte und Signale. Können außerdem frei platziert werden (wie `DEFAULT`).

3.2 buildMode

Dieser Parameter bestimmt, wie mehrere Objekte gebaut werden. Gültig sind die Werte:

- `buildMode = "SINGLE"` - Platziert das Objekt 1x. Das ganze Menü wird beim Platzieren geschlossen.
- `buildMode = "MULTI"`- Platziert das Objekt 1x beim Klicken der linken Maustaste.
- `buildMode = "BRUSH"` - Platziert das Objekt so lange wie die linke Maustaste gedrückt wird und die Maus eine Strecke (in der Spielwelt, nicht auf dem Mauspad) von ca. 5-10m zurücklegt.

3.3 category

Zunächst nochmal der Verweis auf: Hinweis zu Kategorien.
Diese Eigenschaft ordnet das Asset der Kategorie zu, die hier übergeben wird. Gültig sind dabei alle Zeichenketten bestehend aus `a-zA-Z0-9_-`. Es muss, sofern nicht die Standardkategorien `misc`, `tree` und `rock` von TF genutzt werden, eine entsprechend benanntes Kategorie-Icon unter `res/textures/ui/assets` vorhanden sein. Dieses muss folgenden Namen tragen: `asset_category_<category>.tga`. Beispiel: `category = "Debug"` -> Iconname muss lauten `asset_category_Debug.tga`
Die Community-Standard-Kategorien sind im Eintrag Assets: Kategorien zu finden.

3.4 availability

Die Verfügbarkeit wird hier, wie in jedem Modell, angegeben. Der Eintrag besteht aus einer Tabelle mit Startjahr (`yearFrom`) und Endjahr (`yearTo`) der Verfügbarkeit. Einträge mit `0` (Null) können dabei weggelassen werden. Es werden nur die hier angegebenen Verfügbarkeiten berücksichtigt. Verfügbarkeiten aus den Modellen werden ignoriert.
Code

```
availability = {
 yearFrom = 1800,
 yearTo = 1802,
},
```

3.5 description

Die Beschreibung besteht ebenso aus einer Tabelle. `name` ist dabei der Anzeigename des Assets. `description` liefert die Beschreibung für das Asset. `_(...)` dient dabei der Übersetzung der Assets. Namen und Beschreibungen aus den Modellen werden ignoriert.
Code

```
description = {
 name = _("Assetname"),
 description = _("Eine ausführliche Beschreibung."),
},
```

3.6 models

Hier wird das zu verwendene Modell angegeben sowie seine Position. Es können dabei mehrere Modelle angegeben werden. Es wird dann beim Platzieren eines davon zufällig ausgewählt. `id` gibt die *.mdl an, in der das Modell an sich definiert ist (also LODs). Die `id` ist relativ zum Ordner `res/models/model`. `transf` ist die normale Transformationsmatrix.
Code

```
=           models
             {
              id = "asset/ampel.mdl",
              transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },
             },
             {
              id = "example.mdl",
              transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },
             },
            },
```

3.7 order

Die Eigenschaft `order` bestimmt die globale Anordnung der Assets in der Kategorienliste. Siehe Assets: Vorgaben zur Sortierung zur Bildung der `order`.

3.8 skipCollision

Hierüber wird bestimmt, ob das Asset bei der Kollisionsprüfung berücksichtigt werden soll. Gültig sind dabei:

- `skipCollision = true` - deaktiviert die Kollisionsprüfung für das Asset. Es können Objekte "durch" das Objekt gebaut werden und umgekehrt. (Beispielanwendung: Schilder, Bahnhofsdekoration)
- `skipCollision = false` - aktiviert die Kollisionsprüfung für das Asset. Es können keine Objekte dort platziert werden, wo das Objekt ist. (Beispielanwendung: Gebäude wie Fernsehturm, Windkraftanlagen)

4 Beispiel

Im Folgenden ein Beispiel für ein Asset:
Code: example.lua

```
function data()
return {
 type = "DEFAULT",
 buildMode = "MULTI",
 category = "misc",
 availability = {
  yearFrom = 2015,
  yearTo = 2016
 },
  description =
  name = _("Beispiel"),
  description = _("Dies ist ein Beispiel-Asset.")
 },
 models = {
  {
   id = "asset/ampel.mdl",
   transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },
  },
  {
   id = "einBeispiel.mdl",
   transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },
  },
 },
 order = 183130103,
 skipCollision = false,
}
end
```

Alles anzeigen