ModUtil: Mods für Repaints vorbereiten

Table Of Contents

1 Einleitung

Die Repaint Funktionen des ModUtil Scripts ermöglichen beliebig viele Repaints, ohne die Mesh- und Group-Dateien zu ändern, wodurch diese beim Repaint auch nicht mehr mitgeliefert werden müssen. Damit das funktioniert, sind aber gewisse Änderungen an den Meshes, Groups und der mod.lua nötig, die hier erklärt werden sollen.
[info=info]Wird ein Objekt so auf Repaints vorbereitet, gilt das auch als implizite Erlaubnis für Repaints, solange diese ebenfalls über die Repaint Funktion realisiert werden, auf das original Objekt (welches dann benötigt wird) verlinken und der Autor des original Objekts erwähnt wird.
Diese Erlaubnis kann aber natürlich auch durch eine eigenen Lizenz erweitert, eingeschränkt oder ersetzt werden.[/info]

2 mod.lua

Hier muss zunächst das ModUtil Script eingebunden werden (sofern es nicht wegen der Einstellungen schon eingebunden ist):

```
local modUtil = require "merk_modutil_1"
```

Danach wird eine Liste mit allen Dateien benötigt, die durch Repaints verändert werden können, die Dateinamen müssen dabei ausgehen von "res/models/mesh" bzw. "res/models/group" angegeben werden, also so, wie sie auch in der .mdl stehen würden (Groß-/Kleinschreibung beachten!):
Code

1. local files = {
2. "vehicle/waggon/4yg/AB4yg_body_lod0.msh",
3. "vehicle/waggon/4yg/4yg_axle_lod0.msh",
4. "vehicle/waggon/4yg/4yg_bogie_lod0.msh",
5. "vehicle/waggon/4yg_bogie_lod0.grp",
6. }

Diese Liste muss leider noch per Hand erstellt werden, aber vielleicht findet jemand Zeit ein kleines Tool zu erstellen, dass eine Mod nach .msh und .grp Dateien durchsucht und sie in so einer Liste ausgibt.

Die letzte zwingende Änderung betrifft die "runFn", dort werden dem Script alle nötigen Informationen mitgeteilt. Im Normalfall sollte dafür die Funktion "initialize" genutzt werden:

```
modUtil.initialize("mod_id", settings_def, files, nil, nil, {devMode = true})
```

Der erste Parameter ist dabei eine eindeutige ID für die Mod, z.B. der Name des Mod Ordners, der zweite

die Mod Einstellungen (falls vorhanden, damit entfällt dann die Zeile `modUtil.userSettings.create("mod_id", settings_def)`. Wenn es keine Einstellungen gibt, einfach `nil` nutzen) und der dritte die Variable, in der die Liste der Dateien gespeichert wurde (man könnte die Liste auch direkt in den Funktionsaufruf schrieben, aber so ist es meiner Meinung nach übersichtlicher).
Die letzten drei Parameter sollten nur während der Entwicklung des Modells genutzt werden, damit Änderungen in den Mesh oder Group Dateien in die "repaintFiles.lua" übertragen werden (bzw. diese überhaupt erst erstellt wird). Beim Veröffentlichen sollte dort also nur noch `modUtil.initialize("mod_id", settings_def, files)` stehen.

Die vollständige mod.lua sollte also diese Struktur haben:

Code: mod.lua

```
local modUtil = require "merk_modutil_1"

-- optional: Einstellungen
local settings_def = {
option1 = {
type = "boolean",
name = _("Option1"),
},
}
-- Einstellungen Ende
local files = {
"vehicle/waggon/4yg/AB4yg_body_lod0.msh",
"vehicle/waggon/4yg/4yg_axle_lod0.msh",
"vehicle/waggon/4yg/4yg_bogie_lod0.msh",
"vehicle/waggon/4yg_bogie_lod0.grp",
}
function data()
return {
info = {
-- ...
},
settings = settings_def, -- nur wenn Einstellungen genutzt werden
runFn = function(settings)
modUtil.initialize("mod_id", settings_def, files, nil, nil, {devMode = true})
-- bzw bei Veröffentlichung: modUtil.initialize("mod_id", settings_def, files)
end
}
end
```

Display More

3 .msh und .grp Dateien

Alle Meshes und Gruppen, die durch Repaints verändert werden können, müssen ebenfalls etwas angepasst werden. Dazu gibt es mehrere Möglichkeiten.

3.1 Blender

Die neueste Version des Blender Addons kann die entsprechenden Dateien schon angepasst exportieren. Dazu muss lediglich beim Export die Option "Repaintable" aktiviert und die Repaint ID aus "modUtil.initialize" angegeben werden:

3.2 Manuell

Die aufwändigste Methode, aber auch nicht besonders kompliziert. Jede .msh und .grp datei sollte vorher diese Struktur haben:
Code

```
1. function data()
2. return {...}
3. end
```

Diese wird dann einfach wie folgt geändert:

Code

```
1. local result = {...}
2.
3. if merk_vehicleUtil and merk_vehicleUtil[1] then
4. merk_vehicleUtil[1].getMeshData("mod_id", "vehicle/waggon/4yg/4yg_axle_lod0.msh", result)
5. end
6.
7. function data()
8. return result
9. end
```

Display More

Der Teil `{...}` bleibt dabei unverändert, angepasst werden muss nur Zeile 4. Dort ist "mod_id" wie immer die eindeutige ID der Mod. Danach kommt der Pfad zur Datei die gerade bearbeitet wird (so wie er in der .mdl stehen würde) und bei Gruppen muss `getMeshData` durch `getGroupData` ersetzt werden.

3.3 changeFiles

Ab Version 1.1 des ModUtil Scripts gibt es eine Funktion `modUtil.vehicles.changeFiles(files, "mod_id")`, die alle angegebenen Dateien automatisch anpasst. "files" ist dabei wieder die gleiche Dateiliste wie bei "modUtil.initialize" und "mod_id" ist ebenfalls die eindeutige ID der Mod. Die Funktion kann dann einfach direkt vor `function data()` eingefügt und nachdem die mod.lua im Spiel einmal ausgeführt wurde (einfach auf "Freies Spiel" oder "Spiel laden" klicken) wieder gelöscht werden. Sie ist zwar so konzipiert, dass die Dateien auch bei mehreren aufrufen nur einmal geändert werden, es ist aber trotzdem sinnvoll sie nicht unnötig oft aufzurufen, da das Auslesen der Dateien auch etwas Zeit benötigt.

4 Beispiele

Diese Mods nutzen die Repaint Funktionalität und können daher als Referenz dienen:
https://www.transportfever.net…chsige-DB-Umbauwagen-TPF/
https://www.transportfever.net…ex.php/Entry/3247-DB-v60/