main.lua - Scripting / Modding

Inhaltsverzeichnis

- 1 Allgemeines
- 2 Die main.lua
  - 2.1 getConfig und setConfig
  - 2.2 Scripting mit „addModifer“

Mein dank geht an Basil Weber (Urban Games) und Gwinda, die mir die nötigen Erklärungen zu den neuen Möglichkeiten und die Erlaubnis zur Veröffentlichung der Informationen gegeben haben.
Urban Games wird demnächst auch noch weitere Informationen Rund um das Thema Modding veröffentlichen.

Dieses Guide enthält nur Informationen rund um die main.lua.
Für Informationen zu der info.lua und dem neuen Modding System empfehle ich den folgenden Lexikon Eintrag: Making mods work with the in-game mod manager.

1 Allgemeines

Die main.lua befindet sich zusammen mit der info.lua und strings.lua im Hauptverzeichnis einer Modifikation oder DLCs. Über die main.lua können Script Eingriffe in das Spiel gemacht werden, ohne das elementare Dateien dafür geändert oder überschrieben werden müssen. Dadurch werden die Modding Möglichkeiten sehr dynamisch und auch kompatibler zu einander.

2 Die main.lua

In diesem Abschnitt werde ich nun auf die verschiedenen Möglichkeiten mit der main.lua eingehen.

[hl=2]game.config.[x][/hl]

Mit der Methode `game.config.[x]` kann man auf diverse LUA-Confog Dateien zugreifen, erweitern und nutzen. Diese liegen im Verzeichnis `Train Fever\res\config`. Um ein Verständis und die Möglichkeiten mit der Methode zu verdeutlichen werde ich genauer auf `game.config.costs` und `game.config.cargotypes` eingehen.

[hl=3]game.config.costs [Parameter editieren][/hl]

In meiner Sandbox Mod benutze ich die Methode game.config.costs um die kosten für Terrain-Bearbeitung, Gleis-, Straßen-, Tunnel- und Brückenbau auf Null zu setzen.

Dafür habe ich die Methode `game.config.costs` auf `costs` verkürzt um eine bessere Übersicht zu behalten. Mittels `costs` kann ich ich nun auf die Parameter von `game.config.costs` zugreifen. Die Parameter finden sich in der Datei `Train Fever\res\config\costs.lua` wieder. Also setzen wir in der `main.lua` nun die Parameter auf Null.

Der Quellcode sieht nun wie folgt aus:

Code

```
local costs = game.config.costs
costs.terrainRaise
costs.terrainLower
costs.railroadTrack
costs.railroadHighSpeedTrack = 0
costs.railroadCatenary
costs.railroadSwitch
costs.railroadBridge
costs.railroadBridgeVol
costs.railroadTunnel
costs.railroadTunnelLen
costs.roadBusLane
costs.roadTramLane = 0
```

Alles anzeigen

So könnt ihr also sehr einfach bestimmte `game.config` Parameter direkt setzen, ohne das ihr die `costs.lua` direkt editieren müsst.

[hl=3]game.config.cargotypes [Parameter hinzufügen][/hl]

Mit der Methode `game.config.cargotypes` können wir auch direkt neue CargoTypen hinzufügen ohne das wir die `cargotypes.lua` bearbeiten müssen. Dadurch werden neue CargoTypen dynamisch zum Spiel hinzugefügt und es sollte möglich sein, mehrere CargoMods gleichzeitig zu nutzen (nicht getestet, rein theoretische Annahme meinerseits).

Damit wir nun neue CargoTypes hinzufügen können müssen wir als erstes ein Script am Anfang der `main.lua` einbinden.
Das Script `tableutil` laden wir mit dem Command `require`.
Unsere `main.lua` sieht also bisher wie folgt aus:

Code

```
require "tableutil"
```

Für die Methode `game.config.cargotypes` erstelle ich wieder ein Variable `cargoType` als Verkürzung.

Code

```
require "tableutil"
local cargoType = game.config.cargotypes
```

Um nun einen neuen CargoTyp hinzu zufügen benötigen wir das Command `table.insert([param1], [param2])`. Diesen befüllen wir nun mit dem nötigen Inhalt.
An der Stelle von `[param1]` fügen wir die gewählte `game.config` aus, in unseren Fall also die Variable `cargoType`.
An der Stelle von `[param2]` fügen wir als nächstes den Array mit Informationen für den neuen CargoTyp ein. Dieser Array sieht wie folgt aus:

Code

```
{ id = "[id]",   name = _("[id]"  ) }
```

An der Stelle von ID geben wir nun eine Eindeutige ID für die spätere Identifizierung und Verwendung ein, in meinen Beispiel also „PIG_IRON“ und anschließend noch einen geeigneten Namen „Roheisen“. Das Array fügen wir nun an der Stelle von [param2] ein und sollten nun folgende Zeile vorfinden:

Code

```
table.insert(cargoType, { id = "PIG_IRON",   name = _("Roheisen"  ) })
```

Unsere `main.lua` sollte nun wie folgt aussehen:

Code

```
require "tableutil"

local cargoType = game.config.cargotypes
table.insert(cargoType, { id = "PIG_IRON", name = _("Roheisen" ) })
```

Natürlich können wir mit dem `table.insert()` Command so beliebig weitere CargoTypen definieren.
Wie die CargoTypes dann im Spiel genutzt werden können (Lieferketten etc.) muss dementsprechend dann in den .mdl Dateien festgelegt werden, der jeweiligen CargoTypen.

2.1 getConfig und setConfig

Mit Hilfe von getConfig und setConfig können wir beispielsweise eine neue Konfiguration für die Auswahl von Landschaften und Umgebungen definieren. Dies ermöglicht uns widerum eine dynamische Auswahl beim Spielstart, ohne das wir Dateien verändern müssen oder Mods aktivieren/ deaktivieren müssen für unser Spiel.
Um zu verstehen wie die nachfolgenden Parameter funktionieren und eine vollständige Liste der möglichen Parameter zu erhalten, lohnt sich ein Blick in die Train Fever\res\config base_config.lua.

Im folgenden Beispiel werden wir eine neue Auswahl für Landschaft und Umgebung erstellen.

Als ersten benötigen wir wieder das Script `tableutil` und binden dies mit `require` ein.
Anschließend kopieren uns mit dem Befehl `table.copy(getConfig("terrain", "/default"))` die Konfiguration der Standard Landschaft (Europa) und speichern diese in der Variable `newTer`.

Code

```
require "tableutil"
local newTer = table.copy(getConfig("terrain", "/default"))
```

Im nächsten Schritt legen wir nun einen Namen für unsere Landschaft fest. Dieser Name erscheint später in der Auswahlliste der Option Landschaft.

Code

```
newTer.name = _("main.lua - Guide")
```

In den nächsten Schritte legen wir nun die Texturen im Texture-Array für forestFloor (1) über die Parameter fest.

Code

```
newTer.shader.textures[1].detailTex.fileName = "terrain/mod_name/grass_detail.tga"
newTer.shader.textures[1].detailNrmlTex.fileName = "terrain/mod_name/grass_detail_nrml.tga"
newTer.shader.textures[1].colorTex.fileName = "terrain/mod_name/grass_colors.tga"
```

Anschließend setzen wir die Konfiguration mit `setConfig([option], [mod_name], [arg])` aktiv.

Code

```
setConfig("terrain", "mod_name", newTer)
```

Unsere `main.lua` sieht nun wie folgt aus:

Code

```
require "tableutil"
local newTer = table.copy(getConfig("terrain", "/default"))

newTer.name = _("main.lua - Guide")
newTer.shader.textures[1].detailTex.fileName = "terrain/mod_name/grass_detail.tga"
newTer.shader.textures[1].detailNrmlTex.fileName = "terrain/mod_name/grass_detail_nrml.tga"
newTer.shader.textures[1].colorTex.fileName = "terrain/mod_name/grass_colors.tga"

setConfig("terrain", "mod_name", newTer)
```

Alles anzeigen

Wenn wir unsere Mod komplett fertig haben, und diese im Ingame Modmenu aktiv setzen, können wir nun auch bei Landschaft unsere Mod als Landschaft auswählen.

[Blockierte Grafik: http://img4host.net/upload/0707481854fa9f32b9d3a.png]

2.2 Scripting mit „addModifer“

Mit der Funktion `addModifier` können sämtliche Inhalte bei Spielerstellung dynamisch modifiziert werden, ohne das jede Datei dafür geändert werden muss. Bei der timeless und costmod war es früher notwendig, jede einzelne Datei mit einem Variable und einer Scripteinbindung auszustatten. Bei neu hinzugefügten Mods musste also jedesmal auch die cost/timeless mod erneut installiert werden oder per Hand die entsprechende Datei der Mod geändert werden. Dies ist dank addModifier nicht mehr nötig und ich erkläre nun wie dies in der Sandbox Mod funktioniert.

Wer mehr über die Möglichkeiten und weiteren Funktionen erfahren möchte, sollte sich das Script `mod.lua` in `Train Fever\res\scripts` genauer anschauen.

Zu aller erst fügen wir eine Funktion `metadataHandler` hinzu, die die übergebene Datei auf vordefinierte Werte überprüft und dementsprechend für diesen Spielstand ändert.

Code

```
local function metadataHandler(fileName, data)
    if data.metadata.cost and data.metadata.cost.price the

  end

end
```

Mittels Data können wir auf die übergebene Datei (fileName) zugreifen und auf die dortigen Parameter. Welche Parameter genau vorliegen kann man einfach Überprüfen in dem eine .mdl eines Modells öffnet uns sich dort den Quelltext näher betrachtet.
Im Grunde erstellt man nun für jeden möglichen Fall eine Überprüfung und setzt dann den Parameter auf den gewünschten Wert. Die gesetzten Parameter werden dann per `return Data` zurückgegeben und später dem `addModifier` übergeben.

Für eine bessere Strukturierung und Übersicht empfiehlt es sich mehrere Funktionen anzulegen, da die Strukturen nicht gleich sind in jeder .mdl Datei.

https://dev.transportfever.net/lexicon/entry/85-main-lua-scripting-modding/

Zum Schluss fügt man noch den entsprechenden `addModifier` hinzu und verknüpf diesen mit der erstellten Funktion.

```
addModifier( "loadModel", metadataHandler )
```

Folgende `addModifier` sind möglich:

- loadModel
- loadBridge
- loadBuilding
- loadBuildingMaterial
- loadMultipleUnit
- loadRailroadCrossing
- loadSoundSet
- loadStreet
- loadTrainStation
- loadTunnel

Wer sich noch näher mit der Vorgehensweise beschäftigen möchte, sollte sich den timeless mod, cost mod oder meine sandbox mod genauer anschauen.

Ich hoffe mit diesem kleinen Guide konnte ich euch mehr Licht ins Dunkeln bringen, gerne könnt ihr auch Fragen oder Anregungen, Kritik posten.
Sollten neue Informationen bekannt/veröffentlicht werden, werde ich diese auch gerne hier hinzufügen und erläutern.

Mit freundlichen Gruß
Revyn112

https://dev.transportfever.net/lexicon/entry/85-main-lua-scripting-modding/