Game Script

Inhaltsverzeichnis

- 1 Einleitung
- 2 Speicherort
- 3 Inhalt / Grundgerüst
- 4 Funktionen
- 5 Script- und GUI-Thread
- 6 GameScript State Handling (Eigenarten)
- 7 Interface und Gui Funktionen

1 Einleitung

Mit einem *Game Script* können Mods im laufenden Spiel regelmäßig Funktionen ausführen und auf verschiedene Ingame Events reagieren.

2 Speicherort

Man erstellt dazu im Mod-Ordner einen Ordner `res/config/game_script`. Dort legt man dann eine Datei `modname.lua` an.

Auch im Vanilla-Ordner gibt es einige GameScript-Dateien, die zB das Guidesystem, die ToolTips oder die Anzeige für die transportierten Personen/Güter (unten in der Leiste) beeinhalten. Diese kann man ebenfalls als Beispiel nehmen und sich daran orientieren.

3 Inhalt / Grundgerüst

Im Folgenden ein Beispiel für ein GameScript mit allen grundlegenden Funktionen. Dieses könnt ihr als Grundlage für eine Modentwicklung oder zum weiteren Testen verwenden.

Zunächst werden die 8 (so viele habe ich zumindest gefunden) GameScript-Funktionen definiert und unten in der `function data()` zusammengefasst zurückgegeben. Es empfiehlt sich dabei, beim return in der Tabelle immer nur die Funktionen nicht auskommentiert zu lassen, die gerade wirklich benötigt werden.

In `game.res` sind alle GameScript-Funktionen (Vanilla und andere Mods) des aktiven Spiels enthalten.

Code

```
local                          function                          getGameTimeStr()
                                                  local           gametime           =
                                                                       local              da
                                                                local              month
                                                                   local              year
                                                                         local
          return   string.format("%.2d.%.2d.%d\t(%s)",   day,   month,   year,   ttime   )
end

local                          function                          getRealTimeStr()
  return os.date("%Y-%m-%d %H:%M:%S") .."\t".. string.format("(%s)", os.clock() )
end


local   function   init()     --   called   once   at   startup   (only   new   game!)
    print("------------INIT------------")
                           --commonapi.dmp(game.res)             --      all      gameScri
end

local   function   update()      --   called   every   0.2   seconds   (Game   Time)
                                                                                         pr
end

local       function       handleEvent(src,       id,       name,       param)
                                         if         src=="guidesystem.lua"          then
                               print("handleEvent:",        src,        id,        name,
    --commonapi.dmp(param)
end

local     function     save()          --     called     as     often     as     update
    print("save")

               return    state       --    this    return    is    the    parameter    in    lo
end

local                         function                          load(loadedState)
    print("load")
    --commonapi.dmp(loadedState)
end

local     function     guiInit()          --     called     once     at     startup

end

local     function     guiUpdate()          --     called     every     GUI     frame
                                                                                   print("g
end

local          function          guiHandleEvent(id,          name,          param)
                                            --if         name=="visibilityChange"        then
                                                print("guiHandleEvent:",          id,          name,
end


function                                                                             data()









    }
end
```

https://www.transportfever.net/lexicon/entry/271-game-script/

Alles anzeigen

4 Funktionen

Die `init` Funktion wird einmal während des Ladens aufgerufen, aber nur bei einem neuen Spiel.

`guiInit` wird einmalig nach dem Laden eines Spiels/Spielstands aufgerufen.

Die `update` Funktion wird alle 0.2 Sekunden aufgerufen. Das hängt zwar mit der internen Spielzeit zusammen (beim Spiel-Geschwindigkeit erhöhen wird sie öfter aufgerufen), aber auch wenn das Spiel pausiert ist, wird sie regelmäßig aufgerufen.

`guiUpdate` wird mit jedem Frame aufgerufen.

Es gibt verschiedene Script Events im Spiel, auf die mit `handleEvent` reagiert werden kann. Wenn ihr die entsprechende Zeile oben auskommentiert, werdet ihr merken, dass das Log direkt von Events des G*uidesystems* geflutet wird, komischerweise auch wenn dieses gar nicht aktiviert ist.

Auch relativ viele Events gibt es bei `guiHandleEvent`. Fast jeder Klick oder jede Bewegung in der UI wird hier registriert.

Zu `save` und `load` siehe unten.

5 Script- und GUI-Thread

Wie ihr vielleicht schon wisst, gibt es seit Transport Fever 2 in LUA verschiedene Threads, die voneinander getrennt sind und unabhängig laufen. Im **Script-Thread (Game Thread)** werden die grundlegenden Berechnungen und Script-Funktionen ausgeführt, im **Gui-Thread** alle Benutzerinteraktionen und grafischen Funktionen.

init, **update**, **handleEvent** und **save** laufen im **Script Thread**.

guiInit, **guiUpdate**, **guiHandleEvent** und **load** laufen im **Gui Thread**.

Man kann Daten von der Gui zum Script schicken mit game.interface.sendScriptEvent(id, name, param)

Dabei dran denken, dass dieses Event alle anderen aktiven GameScripte auch erhalten können.

6 GameScript State Handling (Eigenarten)

save() und load() werden während eines Fames mehrmals (bei mir 3 mal) abwechselnd aufgerufen.

Wenn ein Spieler sein Spiel speichert, wird zusätzlich save() mehrmals aufgerufen.

Und wenn der Spieler ein Spiel startet, wird load() mehrmals ausgelöst, aber nur für die Game-Threads.

Anzumerken ist auch, das beim laden eines Spielstandes die load() und save() ebenfalls mehrmals ausgeführt werden. Außerdem überprüft TPF2 dabei, ob save() in jedem Fall den gleichen Zustand zurück liefert aus Game-Thread und GUI-Thread. Das ist natürlich meistens der Fall, wenn aber z.B. Zeitstempel hier mit gespeichert werden, werden diese sich in beiden Threads unterscheiden und TPF2 wird mit einer kryptischen Fehlermeldung abbrechen. Empfehlen kann ich dafür nur, Zeitstempel erst im guiUpdate() Funktion zu erstellen und per sendScriptEvent() an den Game-Thread zu schicken.

Der Zustand innerhalb eines GameScripts kann nur durch die Wechselaufrufe von save() und load() vom Game-Thread zum GUI-Thread gelangen.

Anders rum muss game.interface.sendScriptEvent() im GUI-Thread aufgerufen werden und dieser in handleEvent() dann im Game-Thread verarbeitet.

7 Interface und Gui Funktionen

In den jeweiligen GameScript-Funktionen können dann zum Beispiel `game.interface` oder `game.gui` Funktionen aufgerufen werden.

Für Transport Fever (1) gibt es eine Dokumentation dieser Funktionen, die einem vielleicht weiterhelfen kann. Für TPF2 gibt es mittlerweile wesentlich mehr Funktionen, die man nun verwenden kann, bei der Anwendung muss man manchmal etwas raten oder ausprobieren, was nicht selten zum Crash führt.

`game.gui` ist nur im GUI Thread vorhanden und enthält 43 Funktionen:

Spoiler anzeigen

`game.interface` ist in beiden Threads verfügbar, allerdings sind im **Gui Thread** davon nur 42 Funktionen und im **Script Thread** 60 enthalten.

Einen Sonderfall stellt dabei, wie schon beschrieben, **sendScriptEvent** dar.

Spoiler anzeigen

Mehr Infos auch hier: Data structures returned by game.interface