Luxor 2

Documentation for Luxor 2

State File Modification

State files control certain functions of UI screens in games using the Luxor 2 engine. They have an extension of `sm` and can be found in `data/state`.

For Luxor 2 (not HD), you may want to use a pre-fixed plaintext version of the state folder that should work with all versions HERE.

General Ideas

Here's some basic ideas you could do with state files:

- Simple screen additions, such as a credits screen without the need for replacing something else for it, such as the Enable Cheats dialog
- "Are you sure you wish to quit?"
- Being able to assign **MORE** Map IDs in the Achievements screen, which **might** give way for mods with 26+ maps that properly function in Survival/Challenge/Gauntlet modes

> **Remark:** I noticed that it seems like Luxor AR HD (or Luxor 1 HD?) used to be a 2-in-one pack of sorts, you can check `achievements.sm` and notice how there's "Luxor Completion" stuff commented out and the map counts starting from **26**, this is obvious when playing a map in Survival/Practice mode.
> **Remark 2:** It might be already possible to have 26+ maps in the Survival/Challenge/Gauntlet maps, due to the fact that these levels start from the index of 26.

- Splitting achievements screen into two (achievements and personal stats), but I dunno why would you do that.
- Set music playlists for each screen (without the need for hacky workarounds, such as playing an eiSound in the "Danger" track and setting "Normal" track volume to 0)

But before you get too excited, know that you can't do these:

- An actual cutscene system (damn, I wish)
- Conditional statements
- A fix for 15+ stages (see: DRTPIAB's Epic Wins/Fails counter attempt, ask anyone lol)
- Implement things from other Luxor 2-based engine games (LXE, NCQ, MM, etc)

- Per-level tracks

> **Remark:** It seems like Luxor Evolved-specific features are in Luxor Amun Rising HD, and possibly other HD remasters. While looking at it's memory allocation in HxD, I noticed Luxor Evolved specific things like `uiSpectrumFrame` and `Song` are in the game. Sadly, it is not possible to assign level tracks this way. It just seems like those were just sitting in the code, and the devs only worked on those when Evolved was next to be developed.
> It only makes sense considering Luxor HD remasters and Evolved were released during 2012, and the other HD releases never needed such functionality.

Basic State File Modification

Creating Simple Screens

Let's say we'd like to create a simple confirmation screen for when the player wants to exit. We don't need to worry much about it - the UI system takes care of our UI elements, and the state system takes care of the "programming".

First we need to look at `vars_pc.gvf` (Don't mind the iOS and x360 files, unless you're over your head and want to create a Luxor AR HD mod for Android devices, then yes, go ahead).

```
 7  global TOPLEVEL_UI = ~data/scripts/interface/toplevel.ui
 8  global TRANSITION_UI = ~data/scripts/interface/transition.ui
 9  global GAME_UI = ~data/scripts/interface/game.ui
10  global DIALOG_OPTIONS_UI = ~data/scripts/interface/dialog_options.ui
11  global DIALOG_MODE_SELECT_UI = ~data/scripts/interface/dialog_mode_select.
```

As you can see, the `vars` pretty much define UI files for state machine files. We want to define a new UI here, as shown:

```
 7  global TOPLEVEL_UI = ~data/scripts/interface/toplevel.ui
 8  global TRANSITION_UI = ~data/scripts/interface/transition.ui
 9  global GAME_UI = ~data/scripts/interface/game.ui
+ 10  global DIALOG_CONFIRM_EXIT = ~data/scripts/interface/dialog_confirm_exit.ui
11  global DIALOG_OPTIONS_UI = ~data/scripts/interface/dialog_options.ui
```

We're going to the ui files now. I'm not going to talk much about it, but we're mostly interested in the `Command` property.

```
  ..  uiButton Button_ActuallyQuit
  ..  {
  ..    Position = [ 108.000000, 390.000000, 0.000000 ]
  ..    AnchorHorz = CENTER
+ ..    Command = "QuitForReal"
  ..
  ..    uiTextWidget Text
  ..    {
  ..      Text = T( "Yes" )
  ..    }
  ..
  ..    uiSprite Icon
  ..    {
  ..      Style = "Icon"
  ..      Position = [ 48.000000, 48.000000, 0.000000 ]
  ..      Color = [ 1.000000, 1.000000, 1.000000, 0.908038 ]
  ..      AnchorHorz = CENTER
  ..      AnchorVert = CENTER
  ..      Sprite = ~data/scripts/interface/styles/dialog/button_ipad/sprites/icons/quit_game.png
  ..      BlendMode = ADD
  ..    }
  ..  }
```

The command name does not matter as long as it's readable and it's the exact same name you'll be using in the state file.

The state machine file we're interested in is `mainmenu.sm`.

```
37  Command "Profiles"     StateMachine::csPushState "Profile_Manage"
38  Command "Play"         StateMachine::csPushState "ModeSelect"
39  Command "Options"      StateMachine::csPushState "Options"
40  Command "HighScores"   StateMachine::setState "HighScores"
41  Command "Instructions" StateMachine::setState "Instructions"
42  Command "Achievements" StateMachine::setState "Achievements"
43  Command "MoreGames"    enClientLocal::invoke_3rdPartyMoreGamesURL
44  Command "QuitMenu"     StateMachine::setState "MainMenuQuit"
```

The string after the keyword "Command" is what triggers the function/callback/whatever in the right. We want to change line 44 to something like:

```
  37   Command "Profiles"     StateMachine::csPushState "Profile_Manage"
  38   Command "Play"         StateMachine::csPushState "ModeSelect"
  39   Command "Options"      StateMachine::csPushState "Options"
  40   Command "HighScores"   StateMachine::setState "HighScores"
  41   Command "Instructions" StateMachine::setState "Instructions"
  42   Command "Achievements" StateMachine::setState "Achievements"
  43   Command "MoreGames"    enClientLocal::invoke_3rdPartyMoreGamesURL
! 44   Command "QuitMenu"     StateMachine::csPushState "ConfirmExit"
```

We want to "push" the state and not "set" it, because our exit confirmation dialog is a pop-up, not full-screen. All we need to do now is to create the new state!

```
+ ..   StateDialog ConfirmExit : BaseState
+ ..   {
+ ..     Dialog = DIALOG_CONFIRM_EXIT
+ ..
+ ..     Command "Cancel"         StateMachine::popState
+ ..     Command "QuitForReal"    StateMachine::setState "MainMenuQuit"
+ ..   }
```

You might be able to understand quickly how this works now. The "Cancel" command just "pops" the state and goes back to the previous state (which is the main menu). The "QuitForReal" command sets the state to "MainMenuQuit", which is already defined, which well... exits the game!

Think of the "Command" property as an alias to a function, with the state file providing the "link" to the function.

Assigning Music to Screens

Let's say we want to change the adventure Stage Map to play something else instead of the same old boring main menu theme. First we need to modify `data\game\music.gvf`.

We just need to add a new objEffectMap, like so:

```
1  objEffectMap Menu
2  {
3    Effect = ~data/scripts/effects/sound/music/menu.ofx
4  }
```

```
  5
+ 6  objEffectMap Map
+ 7  {
+ 8    Effect = ~data/scripts/effects/sound/music/map.ofx
+ 9  }
 10
 11 objEffectMap Level
 12 {
 13   Effect = ~data/scripts/effects/sound/music/level.ofx
 14 }
```

The OFX file handles the actual music track, we just need to create one, like so:

```
  1 objEffect MapMusic
  2 {
  3   Loop = true
  4   ObjectOffset = "Normal"
  5
  6   eiSound Song
  7   {
! 8     File = ~data/music/sparkleunleashed-map.ogg
  9     HandleGroup = "Music"
 10   }
 11 }
```

Then we edit `data\state\common\gcladventure.sm` like so:

```
 31 StateDialog StageMap_Adventure : BaseState
 32 {
 33   Dialog = DIALOG_STAGEMAP_ADVENTURE_LUXOR
 34   TrTransInDone = gameClientLocal_Luxor::trigger_advanceLevelCb
 35
 36   Init   = gameClientLocal_Luxor::init_stageMapAdventureCb
+37    Init   = enClientLocal::setMusicPlaylist "Map"
 38
 39   Command "Cancel"   gameClientLocal_Luxor::command_quitGame "StageMap_QuitGame"
 40   Command "Start"    gameClientLocal_Luxor::command_startGameCb
 41   Command "SignOut"  StateMachine::setState "TransToMainMenu"
 42 }
```

And now when you enter the stage map (**not** the Challenge of Horus stage map), the new map music track should play. It should also stop playing when you return to the Main Menu.

Documentation, So Far

Common Properties

Init

```
Init = namespace::Function
```

A function is called on state initialization.

Term

```
Term = namespace::Function
```

A function is called on state termination.

Functions

StateMachine

csPushState

```
StateMachine::csPushState "StateLabel"
```

Pushes a state into the current stack. Used for modal dialogs (such as confirmations).

csPopState

```
StateMachine::csPopState
```

Pops the current state pushed by `csPushState` and returns to the previous state.

csPopPushState

```
StateMachine::csPopPushState "StateLabel"
```

Pops the current state, then pushes a state into the current stack.

setState

```
StateMachine::setState "StateLabel"
```

Sets the state. Used for screens, like the High Scores screen or the Achievements screen.

enClientLocal

setMusicPlaylist

```
enClientLocal::setMusicPlaylist "Label"
```

Sets the currently playing music to the objEffectMap defined in `music.gvf`.

stopMusic

```
enClientLocal::stopMusic
```

Stops any playing music.

gameClientLocal_Luxor

command_gamePauseCb

```
gameClientLocal_Luxor::command_newGameOrContinueGameCb "Pause" | "Menu"
```

- `"Pause"` will pause the game with the "Paused: Press Spacebar to Continue" dialog.
- `"Menu"` will pause the game and open the pause menu that allows you to quit/exit/change options.

command_newGameOrContinueGameCb

> ⚠ **Luxor Evolved-specific.**

```
gameClientLocal_Luxor::command_newGameOrContinueGameCb "DifficultySelect"
```

Checks if there's a currently running game. Else, show the difficulty selector.

State Types

StateDialog

Represents a dialog UI.

```
StateDialog Label : BaseState
{
  ...
}
```

Properties

Dialog

```
Dialog = NAME_OF_DIALOG
```

`NAME_OF_DIALOG` must be defined in the target vars file (usually `vars_pc.gvf`).

DialogTransIn / Out

```
DialogTransIn  = true
DialogTransOut = false
```

Both properties are boolean. Tells the game if the dialog should transition in or out.

EffectTransIn / Out

```
EffectTransIn  = "LabelOfObjEffectMap"
EffectTransOut = "AnotherObjEffectMap"
```

Both properties take a string. Uses these effects as transin/out effects.

Command

```
Command "Label"  namespace::Function
```

A function is called when the command is invoked. **Must be defined in the ui file.**

StateServer_Luxor

Used in `gsv_*.sm` files. I don't recommend you poking around `gsv_*.sm` and `gcl_*.sm` files, since they pretty much handle the game itself.

```
StateServer_Luxor Label : Server_Common
{
  ...
}
```

Properties

MusicState

> ⚠ **Luxor Evolved-specific.**

```
MusicState = "NameOfMusicState"
```

Sets the music state of the level.

UI File Modification

UI files control the layout of a specific element in the game - including level backgrounds.

Objects

Every object should have it's type and an optional label, with 3 spaces as indentation:

```
uiFrame theLabel
{
   uiSprite theLabel
   {

   }
}
```

uiFrame

A uiFrame is a container, usually used for dialog boxes. uiFrames can be adjusted with a uiFlowLayout.

uiScrollFrame

A uiScrollFrame is just like a uiFrame. However, overflow content is clipped and scrollbars appear.

uiContainer

A uiContainer is usually used for adding props, or as an anchor for OFX files.

uiInputFrame

A uiInputFrame is a container usually used for buttons or controls to change it's contents or icon depending on the active input device, such as a mouse or a controller.

uiDialog

A uiDialog is the root of either a dialog box, or a dialog screen.

uiSpectrumFrame

> ⚠ **Luxor Evolved-specific.**

A uiSpectrumFrame is a container for uiSpectrumChannels. However, it is not a direct replacement for a uiFrame. A uiSpectrumFrame still needs to be wrapped in a uiFrame.

uiFlowLayout

A uiFlowLayout controls the layout of a uiFrame. This can be used to create grids, for example.

objEffectMap

- `Effect` `<Path>` The OFX file to use.

An objEffectMap applies an OFX file to an object. It takes the object label as the event.

Example:

```
objEffectMap Idle
{
  Effect = ~data/maps/town/idle.ofx
}
```

Valid labels are:

- `TransIn`
- `TransOut`
- `Idle`

- `ShowMap`
- `ShowMap_Bonus`
- `HideMap`
- `HideMap_Bonus`
- `ContinueIn`
- `NewLife`
- `Announce_PU_<powerup>` - LXE specific, `<powerup>` must be a powerup ID
- `Milestone_Combo_<n>` - LXE specific, `<n>` must be of: `6`, `9`, `12`

uiBackground

A uiBackground defines the background of an object. It can be a fixed size or a 9-slice background.

uiTextWidget

A uiTextWidget inserts text into a container.

uiButton

A uiButton represents a button contained by it's elements.

uiProgressBar

A uiProgressBar represents a progress bar, usually used in the level HUD to indicate a rough amount of spheres left to destroy, or in the splash screen to indicate loading progress.

uiSprite

A uiSprite adds a sprite to a container.

Common Properties

GVF Types

Simple Types

- Number: A literal number. The game engine can usually take either a float or an int.
- Int64: Ending the number with an `i64` means it's an int and is usually used for score values.
- String: A literal string encased in double quotation marks (`"`).

Localized String

`T( "string" )`

Often used in UI elements.

Tuple

`[ 1, 2, 3<,4> ]`

So far only used for numbers. Commonly used in UI and OFX.

General GVF Syntax

GVF files are (presumedly) "Game Variable Files" that also take on different extensions depending on the usage:

- `.gvf` - Game variables
- `.ui` - UI elements
- `.uis` - UI snippets
- `.ofx` - Object effects
- `.sm` - State Machine files

Objects

```
objectName
{


}
```

Includes

```
# path/to/whatever/you/need.txt
```

It doesn't *have* to end with txt.

Variables/Properties

```
varname = value
```

To assign it globally, add `global` before the variable name. This is only done in `vars.gvf`:

```
global varname = value
```