OGT Level Manager

v1.0 Reference Docs

by J. A. Whye
OutlawGameTools.com

Contents

From OutlawGameTools.com

OGT Level Manager (OGTLM) is a code library that makes quick work of creating a level selection screen, complete with locking and unlocking support, number of stars per level, and even scoring for each level.

The OGTLM code library consists of three files:

`ogt_lmdata.lua` -- The specific variables for a given project.

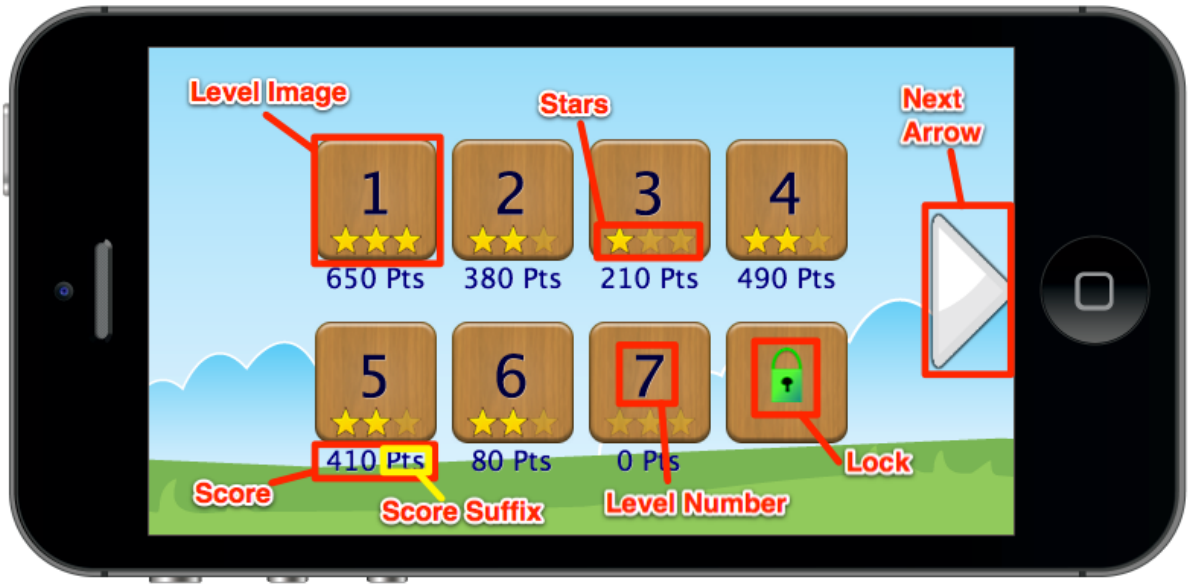
`ogt_levelmanager.lua` -- The utility functions for the library.

`GGData.lua` -- A 3rd-party library that quickly loads and saves data.

Those three files will go in your project, but the only one you'll need to edit is ogt_lmdata.lua -- that holds all the variables that are tweaked to make OGTLM sit up and beg. You'll also end up calling some of the functions inside the ogt_levelmanager.lua file.

Reference Intro

In the following reference docs I'll refer to different parts of the Level Manager. Here's a picture showing most of those parts.



Following are descriptions of the variables that make up the "tweakable parts" of OGT Level Manager.

Level Names - How OGTLM finds the scene in your game when the user chooses a level.

Asset Filenames - Setting up the names of the images OGTLM will use to create the level selector.

Level Numbers - Discover how to tweak the numbers shown in each level image.

Showing Score - You can display the high score for each level on the level select screen if you like.

Image Offsets - There are default positions for the stars, lock, and paging arrows but you can change all those to suit your game.

Grid Settings - Each page of levels is just a big grid. Change the rows and columns and even the base position of the grid.

Misc Variables - The last of the user-configurable variables in OGT Level Manager.

Utility Functions - Functions available to help make your use of OGTLM faster and easier.

Level Names

(This is a topic that can get complicated, but I'll try to keep it easy to understand...)

If your game has 30 levels, should your project contain 30 different Storyboard scene files, one for each level, or should it have 1 scene file that is basically called 30 times, each time pulling the data specific for that level?

In most cases the latter is the best way to do it because it means you'll have fewer duplicated chunks of code. However, OGT Level Manager will handle both cases.

If you have a single storyboard scene file, enter that scene name into the k.playScene variable. If you have multiple scene files, set this to nil.

```
k.playScene = "play" -- name of SB scene or nil to go to sequential scenes
```

If you have a series of scene files they are usually named the same with an increasing number at the end, such as stage1, stage2, etc. If that's how you have things set up, enter the base scene name (without any number) in the k.sequentialScene variable.

```
k.sequentialScene = "level" -- will turn into level1, level2, etc.
```

And if you're doing something wild and have a different name for each scene level, you're still covered. Just set the previous two variables to nil and turn the k.sceneNames into a table holding the names of your scene files in the order they should be played.

```
k.sceneNames = {"scene01", "playme", "world3"}
```

Otherwise, just set that variable to nil.

```
k.sceneNames = nil
```

Asset Filenames

Most developers put their audio and art assets in folders to keep things organized. There are two variables available to set the prefix for audio files and image files. The folder name can be anything, but it must end with a slash.

```
k.imagePrefix = "images/"

k.audioPrefix = "audio/"
```

If a folder isn't being used, set the variable to nil, like this:

```
k.imagePrefix = nil
```

There are also a series of variables for the different images that may be used in the level selector. Only one (k.levelSquareImage) is required, if the others aren't needed, set their value to nil.

In many cases the background of the level select screen will be set inside the actual choose level storyboard scene. But you can specify a background image to be used here if you like.

```
k.backgroundImage = "background.png"
```

The level select image (the graphic that shows behind the level number and stars, etc.) can either be a simple image (typical) or a group made up of multiple images. If you want to use a single image for the level select image then just enter that image name in k.levelSquareImage as seen below.

But in some cases a simple image isn't what you need. In which case you can enter a function name in k.customLevel that will return an image or display group. Writing that function is up to you, just make sure it returns an image or group and OGT Level Manager will handle the rest. The current level number is passed in to that function so you can use it if you need to.

```
k.customLevel = nil

k.levelSquareImage = "level-wood.png" -- required unless k.customLevel is used
```

Note: If you're using k.customLevel you'll set that to the correct function name in

the calling file, not in ogt_lmdate.lua itself. So it's left blank and then set such as: lm.customLevel = myFunctionName

The next two variables contain the image you want for the lock and the stars. If you don't use locked levels or the whole "Get more stars!" thing, just set one or both to nil and OGT Level Manager will ignore them.

```
k.lockImage = "lock.png"

k.starImage = "staryellow.png"
```

If you have enough levels that you need to page from one set to another, here's where you specify the arrow graphics you want to use. Set them to nil if you only have a single screen of levels and they won't be shown.

```
k.prevImage = "prev-white.png"

k.nextImage = "next-white.png"
```

There are three sounds that can be triggered in OGT Level Manager. When the player chooses a level or clicks the next or previous arrows. Set the audio file names here. Next and Previous will usually use the same sound, but you can make them each different if you like.

```
k.selectSoundFile = "levelselect.wav"

k.nextPageSoundFile = "changepage.wav"

k.prevPageSoundFile = "changepage.wav"
```

Grid Settings

The foundation of OGT Level Manager is a grid of images that allows the user to choose a level. Here are the variables that allow you to change the look and position of that grid.

You can have as few as 1 level or as many as... Well, I don't know what the upper limit is, but it's higher than you'll want to go simply from a user-experience standpoint. Set `k.totalLevels` to the total number of levels. Based on the column and row settings the pages of levels will be created automatically.

```
k.totalLevels = 30
```

How many columns of levels do you want on a single screen? Set `k.numCols` to that value. Set `k.numRows` to the number of rows you want. Remember that if `k.totalLevels` is high enough paging will be activated and you'll need to leave room on the right and left sides of the screen for your next and previous arrow buttons. (Or, if you're just going to rely on swiping to page the screen, you don't need to leave extra room.)

```
k.numCols = 4

k.numRows = 2
```

To add some extra spacing between each element of the grid, set these variables. Since the score is shown below the level image you must set `k.rowSpace` to make room for those if `k.showScore` is set to true.

```
k.colSpace = 10

k.rowSpace = 10
```

The grid of level elements is set to display centered on the screen, both horizontally and vertically. You can use the following variable to change the position of the entire grid.

```
k.gridOffsetX = 0

k.gridOffsetY = 0
```

Level Numbers

Most of the time you'll be showing the level number for each level indicator. These variables allow you to tweak the number quite a bit.

If you don't want to show the level number (typically because you're using a `k.customLevel` function), set `k.showLevelNum` to false and then you can just ignore the rest of the variables in this section.

```
k.showLevelNum = true
```

If you want to use `k.defaultFont` as the font to show the level number, set the following to nil, else put in the name of the font to use. (It's still up to you to make sure that font is available to your project/device.)

```
k.levelNumFontName = nil
```

The next three variable set the size and color of the level number and allow you to use normal or an embossed font.

```
k.levelNumFontSize = 38

k.levelNumFontColor = {0, 0, 0.2}

k.levelNumEmbossedFont = true
```

The level number is designed to show in the center of the level image, but if you're using stars at the bottom of the image maybe you want the level number itself to be higher. The variable shown below allow you to move the level number up, down, left, and right

```
k.levelNumTextXOffset = 0 -- negative nums move left, positive right

k.levelNumTextYOffset = 0 -- negative nums move up, positive down
```

The default for level numbers is to start with 1 and continue sequentially until it reaches `k.totalLevels`. But you may have a game that doesn't use level

numbers, in which case you can specify the text to use instead of numbers.

In this example the level numbers will be replaced with the abbreviations for the Periodic Table of Elements.

```
    k.tileNums =
{"H","He","Li","Be","B","C","N","O","F","Ne"}
```

If you're using this method you'll want to make sure you've provided a name for each level in your game.
If you're just using numbers, set this variable to nil.

Finally, by default the level numbers are not shown when a level is locked. If you'd rather they show all the time (the lock image displays in front of the level number) set the following variable to true.

```
    k.showLevelNumTextWhenLocked = false
```

Showing Scores

While most games don't show a score on the level selector screen, I have one that does, so I decided to include that capability into OGT Level Manager. If you don't need the score to be shown on the level select screen, set `k.showScore` to false. You can still use the scoring capabilities of OGTLM to save and retrieve scores for each level even if they're not displayed on the level selection screen.

```
k.showScore = false
```

The score will be shown with the `k.defaultFont` but you can override that for the score using the `k.scoreFontName` variable. Make sure the font you choose is available on your targeted devices.

```
k.scoreFontName = nil -- optional, use instead of default font for score
```

Use the following variables to set the score font size, color, and to determine whether to use an embossed font or not.

```
k.scoreFontSize = 18

k.scoreFontColor = {1, 1, 1}

k.scoreEmbossedFont = true
```

The default location for the score is positioned directly under the level image. Changing the following variables will reposition the score based on that initial location. Negative numbers move the score left or up, positive numbers move it right or down.

```
k.scoreTextXOffset = 0

k.scoreTextYOffset = 0
```

If you want to display "300 Points" or "$250 Earned" you can do that by setting up a prefix and/or suffix for the score. Be sure and include a space if needed since the prefix, score, and suffix are concatenated directly together.

```
k.scorePrefix = nil
k.scoreSuffix = nil
```

In the second example I used above, this is how the two variables would be set

```
k.scorePrefix = "$"
k.scoreSuffix = " Earned"
```

Notice I didn't use a space after the dollar sign because we want it butted against the score, but I did use a space in front of the suffix.

If you don't want the score to show on a locked level (that's usually the case because the score will be 0 on a level that hasn't been played yet), use the following variable.

```
k.showScoreWhenLocked = true
```

Image Offsets

If you want a different number for maximum stars, you can set that here. Just remember you have to make sure there's room for all of them.

```
k.maxStars = 3
```

Stars are defaulted to display in a horizontal row on the bottom edge of the level image. You can tweak the horizontal and vertical offset here.

```
k.starOffsetX = 0

k.starOffsetY = 0
```

Note that the offset is for the stars as a group, not for an individual star. If you want to move the left star up and the right star down, for example, you can do that with the following variables.

```
k.singleStarOffsetX = {0, 0, 0}

k.singleStarOffsetY = {0, 0, 0}
```

If you want the first star to be shifted up 5 pixels, the table for `k.singleStarOffsetY` will look like this: `{-5, 0, 0}` Of course, if you have `k.maxStars` set to 5, you'll need to change those singleStarOffset tables to 5 elements instead of the default 3.

The arrows on the side of the screen can be tweaked as well using these variables.

```
k.nextOffsetX = 0

k.nextOffsetY = 0

k.prevOffsetX = 0

k.prevOffsetY = 0
```

The default position of the lock is centered on the level image. You may want it higher or lower, depending on other settings, so change the position here.

```
k.lockOffsetX = 0
```

```
k.lockOffsetY = -5
```

Misc Variables

You can specify fonts to be used for displaying the level numbers and score or you can set one or both as nil and OGT Level Manager will use whatever you've set in the `k.fontName` variable. Make sure whatever font you choose is available on your target device.

```
k.fontName = "Helvetica" -- required
```

In many games you want the player to progress from Level 1 until the end, with later levels locked until they have passed the previous ones. You have to have at least the first level unlocked, and you can have as many as you want by setting the `k.numUnlocked` variable.

If you don't want any levels to be locked, make sure you set the value to the same value as in the `k.totalLevels` variable.

```
k.numUnlocked = 1 -- how many of the first levels are unlocked (min 1)
```

When the player returns to the choose level screen do you want it to always appear on the first page of levels? Or would you rather it start from the page where they last selected a level?

```
k.rememberPage = true -- if true, shows page from last selected level
```

If you want a different number for maximum stars, you can set that here. Just remember you have to make sure there's room for all of them.

```
k.maxStars = 3
```

By default the code shows a dim star as a kind of placeholder for a star that wasn't earned, so if the user gets two stars on a level two normal stars will be shown and a third very dim star. If you don't want the missing stars to be shown at all, set the following variable to false.

```
k.showMissingStar = true
```

When the player chooses a level you can select the Storyboard (or Composer) effect and timing to go from the choose level scene to the play scene.

```
k.sboardEffect = "crossFade"

k.sboardTime = 700
```

You can set the amount of time it takes for the paging to happen. Set `k.slideTime` lower to make sliding to next or previous pages happen faster. The time is milliseconds, so a value of 500 in `k.slideTime` equals a half second.

```
k.slideTime = 300 -- milliseconds (lower number = faster slide)
```

You can allow the user to swipe the level chooser left or right to page through the levels in addition to (or instead of) tapping the arrows. Turn on or off the swipe with the first variable below, and tweak the distance a user's finger has to swipe to activate it with the second.

```
k.swipe = true -- allow swiping left/right to page
k.minSwipeDistance = 20 -- number of pixels required to trigger paging
```

There may be times when you need to do some "cleanup" before leaving the choose level screen. For example, in one of my games I have sprites flying around and I want to disable them before leaving that screen. You can set a function to be called before leaving:

```
k.beforeLeaving = nil
```

While that lives inside the ogt_lmdata.lua file you'll set it inside your "choose level.lua" file (or whatever you've named it). Call it like this:
`lm.beforeLeaving = myFunctionName`

Utility Functions

Tweaking the variables at the top of the `ogt_lmdata.lua` file is half of using the library. The other half is taking advantage of the utility functions available for you in the `ogt_levelmanager.lua` file.

In the following examples I will assume you've done this at the top of your code:

```
local lm = require("ogt_levelmanager")
```

Then when I show how a function is called I'll do it like this:

```
lm.foo()
```

Also, in most functions where you can pass in a level number that parameter is optional. If you don't pass it in, the function will use the currently set level. Just remember that so I don't have to repeat it a dozen times. ;) Basically, if I say that a parameter is optional, that's because it will automatically default to whatever is currently being used.

Turn Audio On/Off

Example: `lm.updateAudio(true|false)`

Pass true or false in to enable or disable the sound effects that are available when the player clicks the level select image or the next and previous buttons (if sound effects have been set).

Save a Score

Example: `lm.updateScore(score, lvlNum)`

Pass in the score and the level number (optional) to save that score.

Load a Score

Example: `local myScore = lm.getLevelScore(lvlNum)`

Retrieve the saved score for a specific level (lvlNum is optional).

Save Username

Example: `lm.updateUserName(uname, dataFile)`

Save the specified username (required) to the chosen dataFile (optional).

Load Username

Example: `local currUser = lm.getUserName(dataFile)`

Retrieve the username for the specified data file (optional).

Save Number of Stars

Example: `lm.updateStars(numStars, lvlNum)`

Save the number of stars (required) earned on the chosen level (optional).

Load Number of Stars

Example: `local starsEarned = lm.getLevelStars(lvlNum)`

Retrieve the number of stars earned on the specified level (optional).

Lock/Unlock a Level

Example: `lm.updateLock(true|false, lvlNum)`

Pass in true or false to answer the question, "Is level lvlNum locked?" So true means it's locked, while passing in false means it's unlocked.

While lvlNum is optional, it's a little different than most -- if you leave that blank it will not unlock or lock the current level, but the *next* level.

Unlock the Next Level

Example: `lm.unlockNextLevel()`

This is just a shortcut function that you'd use in many cases instead of the previous one. When a user is playing and you want the next level unlocked, use this function and don't worry about passing true, false, or anything else.

Get Lock Status

Example: `local levelIsLocked = lm.getLockStatus(lvlNum)`

Pass in a level number and it will return a boolean value telling you whether the specified level is locked (true) or not (false).

While the level number parameter is technically optional, this is one function where you'll probably almost always specify the level.

Is There Another Level

Example: `local canKeepPlaying = lm.anotherLevel(lvlNum)`

Check to see whether there's another level after the one specified (optional). Returns true if there's another level or false if the level number passed in (or the current level) is the last one.

Go To Next Level

Example: `lm.loadNextLevel(lvlNum)`

This will launch the level *after* the level number (optional) that's passed in (the next level).

This function ONLY works on sequential or named levels. It does not work when you use the same storyboard scene for all levels (k.playScene = "foo").

Reset All Levels

Example: `lm.resetLevels(dataFile)`

Use this to reset the specified data file (optional) -- all scores to 0, all stars earned to 0, and lock all levels except for the number specified in k.numUnlocked. Does

not change the username in the data file.