Level Selector Menu Tutorial
For Game Maker Studio

Written by: Noel Dundas (AKA Noele)

Contents

Disclaimer and Copyright Information

Level Selector Menu Overview

It is sometimes desirable to allow the player to select a level to play rather than force them to go back and play the game from the first level each time. The Level Selector Menu allows a number of worlds (these can be lands, locations, buildings etc in your game) each of which contains levels the player can complete.

It can also be an advantage to lock certain levels to prevent them being played before the player has accomplished tasks or successfully completed other levels. The Level Selector Menu also includes a level locking mechanism which you can customize for your game. Note that this is an advanced feature – See the section on Level Locking for details.

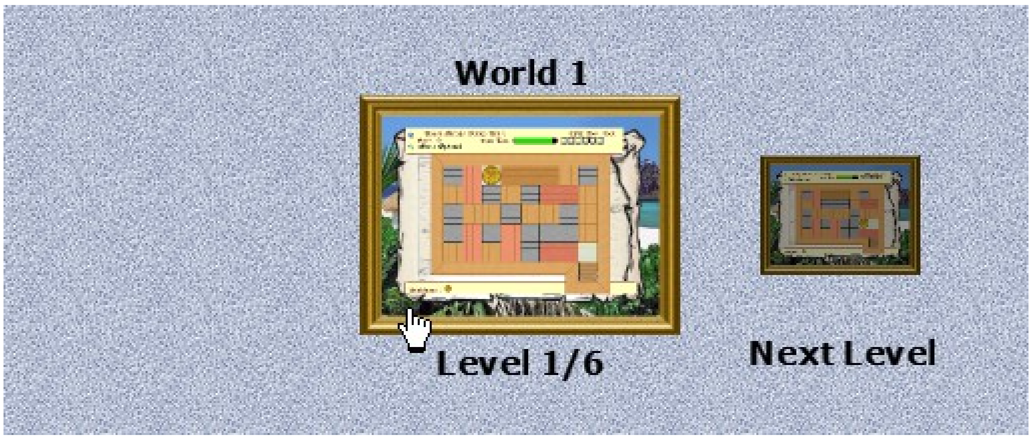
The Level Selector allows an (almost) unlimited number of Worlds, each of which can contain an (almost) unlimited number of levels (See Array Usage for limitations). To keep the control aspect simple and to cater for touch-screen devices only the left mouse button is used.

Although the Level Selector is written extensively in GML and uses arrays, only minimal knowledge of GML is required to use it and this step by step tutorial is here to assist you in customizing it to fit your game.

Using the Level Selector Menu

The easiest way to see how it operates is to import it into GM Studio (Select File > Import Project and navigate to LevelSelect.gmz) and run it from the GM Interface.

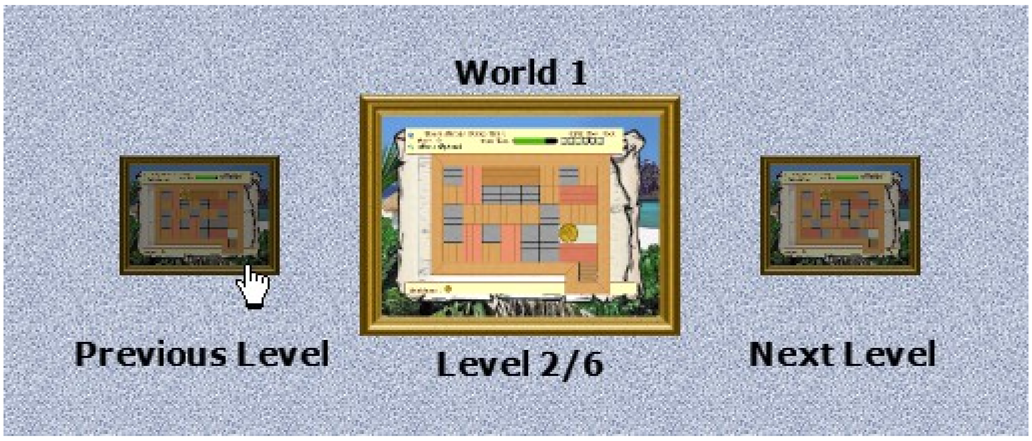
The Level Selector Menu displays images of the Previous Level (if there was one), the Current Level and the Next Level (if there is one).



The **Current Level** is the full sized image in the center. It can be selected using the Left Mouse Button.

The **Next Level** can be selected using the mouse and Left Mouse Button. This will cause the image to scroll to the left to replace the current level, pushing the current one to the left to replace or become the previous level.

The **Previous Level** can be selected using the mouse and Left Mouse Button to move it to the current selection and revert the Current Level to become the Next Level.



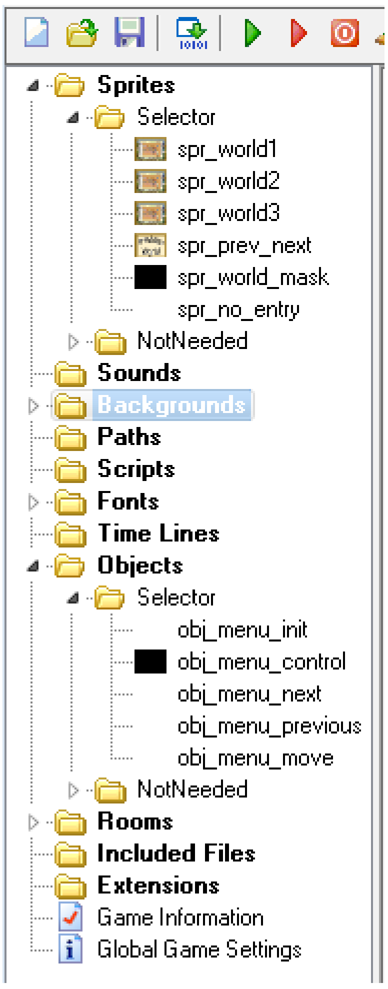
Note: The Next and Previous levels are only shown if there is a next or previous Level or World to go to. The demo as supplied only contains one world (so there is no Next or Previous Worlds) however this tutorial details how you can create additional levels and add more worlds.

Before Getting Started

This Level Selector Menu is supplied in its basic form together with several files and resources, some of which you may want whilst others are optional. The basic starter has just one World containing six levels. We use the term "**World**" to refer to something containing a collection of "**levels**". A "**world**" could be a land in your game, a beginners set of levels, a building or a location in your game.

The number of "levels" each "**World"** contains can be easily adjusted. If you only need a single world containing your levels you can skip the later section on Creating New Worlds. You will however still want to change the default level images to fit your own game and probably the number of levels it contains.

Some resources included with this example exist solely to demonstrate how it works and do not need to be replicated in your game.



```
Sprites
    Selector
        spr_world1
        spr_world2
        spr_world3
        spr_prev_next
        spr_world_mask
        spr_no_entry
    NotNeeded
Sounds
Backgrounds
Paths
Scripts
Fonts
Time Lines
Objects
    Selector
        obj_menu_init
        obj_menu_control
        obj_menu_next
        obj_menu_previous
        obj_menu_move
    NotNeeded
Rooms
Included Files
Extensions
Game Information
Global Game Settings
```

The illustration on the left shows the resources required. Resources in Not Needed groups are only used for this demo and not required for your game(s).

Before getting involved in examining these resource contents, and if you have not done so already, run it to see what it does.

The basic Level Selector Menu as supplied displays three thumbnail images representing the Next, Current and Previous levels. The Next and Previous images are scaled to be half the size of the original and darkened slightly. The current selection in the middle is full scale with no darkening mask.

Selecting the Next moves both it and the Current Level left, scaling proportionately. Selecting the Previous one moves it and the Current level back, also scaling proportionately.

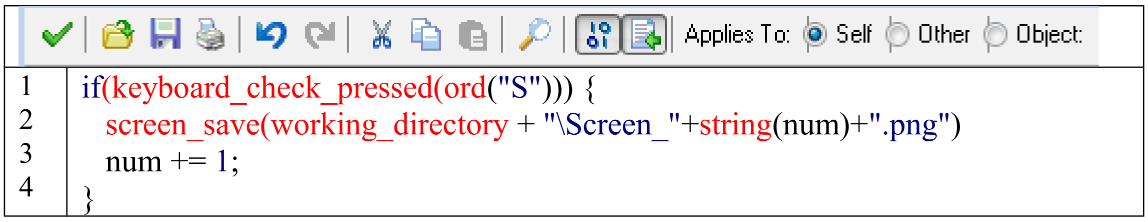
The overall effect gives the illusion all three images are being rotated to enlarge and bring the new one to the front, pushing the current one to the back. This allows you to have any number of selectable levels.

The level selection menu as supplied has just one world containing six levels. It contains a thumbnail image for each level, which are probably not going to be suitable for your game and you will want to change.

Although the Level Selector Menu is designed to allow a player to select a level to play, you are not confined to using actual screenshots of your game as level images; you can use the selection menu as a graphical menu selector to offer options like Play, Load, Help, and Quit etc to create any kind of menu offering any number of options.

Creating Screen Shots

Previous versions of Game Maker allowed you to create screen shots with ease however due to platform limitations this functionality is limited in Studio. The following code snippet will work for most platforms although not HTML5 (browsers control and limit the local file storage)



Applies To: [✓ Self] Other Object:

```
if(keyboard_check_pressed(ord("S"))) {
   screen_save(working_directory + "\Screen_"+string(num)+".png")
   num += 1;
}
```

Alternatively you can use a screen capture application such as MWSnap or use the Windows sipping tool to capture and save screen shots.

Once you have the screenshots you want, re-load the Level Selector project and save it as a name of your choice. This allows you to keep a copy of the original – e.g. for another game.

Creating Level Sprites

The screenshots will invariably be much too large to use so we need to scale them down to create smaller thumbnail images. You can use your favorite art or graphics program to scale down images or you can use GM's built in image editor.

Thumbnail image sizes of around 160 pixels wide are usually adequate but you must be able to fit three images side by side in your game window. The Level Selector uses the sprite dimensions to proportionately position the previous and next level images either side of the current level image so ideally your images should be uniform in size or it may look a bit weird.

Using GM's Image Editor

You can use the sprite image editor to rescale your images. Create a new sprite and Load the first screenshot. Select Edit, Transform, Stretch (actually shrink) and ensure Keep Aspect Ratio is selected to avoid distortions. Setting the width in pixels to the desired width (e.g. 160) will also set the height proportionately.

Save each rescaled image, selecting Yes if prompted to over-write as we will not be keeping the original sizes. Do this for each screenshot so they are all the same size.

Changing the Sprite

1. Select **spr_world1** from the resources, select **Load Sprite** and browse to locate your first (rescaled) screenshot (e.g. Screen1.png).
2. Select **Edit Sprite** and **File** > **Add From File** and add your next screenshot. Repeat until all screenshots (levels) have been added in the order you want each level to appear.
3. Before saving the sprite select **Center** to center the x, y origins. This is important to ensure they scale properly.

Run the "game" to test it and ensure the results are as expected. The number of Levels should adjust to the number of sub-images you added.

Congratulations! You have successfully changed the levels the menu offers. You can now delete all the screenshots from the directory you saved them to ready for your next World.

If you only need a single world for your game you can jump ahead to the section on Selecting a Level.

Creating New Worlds

As previously mentioned a **world** is simply the term used here to describe a container for the levels within it. You could, for example use Beginner, Advanced and Expert as you Worlds or you could use locations in your game – Fairy Land, Pixie Land, Leprechaun Land etc.

Each **world** you create can have a different number of levels. World1 could contain 5 levels and World2 6 levels but each world must have at least two levels. A World where the player could only select a single level would be rather pointless anyway!

Creating a new world is simply a matter of creating a new sprite containing its level images (See Creating Level Sprites) and adding them to the global.land_sprite array. You can name the new world sprite anything you like but it is recommended you give them a meaningful name like spr_world2, spr_world3 or maybe spr_land1, spr_land2 etc…

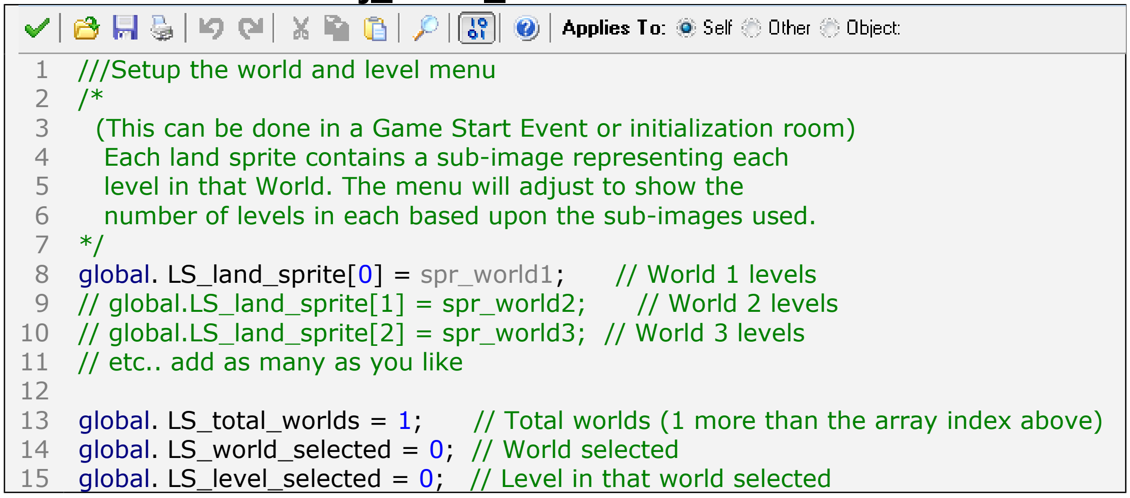
The Level Selector will automatically set the number of levels a world contains to the number of sub-images its sprite contains. When adding a new world to the Level Selector Menu you have to tell it that world exists.

This is not hard, even if you know little about GML coding.

Adding a new world to the Level Selector Menu is simply a matter of editing / adding 2 lines of code in the Create Event of obj_menu_init. After creating the new world sprite containing its level images (spr_world2) we must add it to the code.

The following shows the changes made to add the new world:

Game Start Event obj_menu_init


Applies To: [✓ Self] Other Object

```
1   ///Setup the world and level menu
2   /*
3     (This can be done in a Game Start Event or initialization room)
4      Each land sprite contains a sub-image representing each
5      level in that World. The menu will adjust to show the
6      number of levels in each based upon the sub-images used.
7   */
8   global. LS_land_sprite[0] = spr_world1;      // World 1 levels
9   // global.LS_land_sprite[1] = spr_world2;      // World 2 levels
10  // global.LS_land_sprite[2] = spr_world3;  // World 3 levels
11  // etc.. add as many as you like
12
13  global. LS_total_worlds = 1;      // Total worlds (1 more than the array index above)
14  global. LS_world_selected = 0;  // World selected
15  global. LS_level_selected = 0;  // Level in that world selected
```

Uncomment line 9 and change spr_world2 to the name of your sprite. Edit line 13 and change the = 1 to = 2 (the number of worlds) and save the code.

If there is a Next or Previous world the Level Selector Menu will automatically display Previous World and Next World sprites. These can be found in the sprite resource folder; feel free to edit these or create your own replacements.

You can add more worlds as required however the system puts a limit of 32,000 on each array index and 1 million indexes globally for all arrays. Clearly you would not require anywhere near this number of room levels however consideration to other array usage should be made. The arrays used by the Level Selector Menu are global so although they only exist once and their contents minimal they will exist until the game ends.

Global Variable Use

A **local variable** is one which is local to the instance which creates it. There can be 10 instances of the same object each with a variable called **fred** yet each can hold a value specific to that instance.

The value of a **global variable** can be shared by any instance. Once defined global variables exist until the end of the game and can be accessed by any instance anywhere in the game. An instance changing the value of a global variable changes it for all instances that access it.

The Level Selector menu uses several global variables. To help ensure these do not conflict with any global variables already in your game and help you to identify them, these are all prefixed with **global.LS_** (LS standing for Level Selector).

global.LS_paused ………………..….	Game paused (true/false)
global.LS_land_sprite[Image] ……….	1D Array containing selection sprites
global.LS_total_worlds ……………...	Total Worlds available
global.LS_world_selected …………...	Current world selected
global.LS_level_selected ………..…..	Current level selected
global.LS_land_access[World,Level] .	2D array for access (locking)
global.LS_location[World,Level] …...	2D array containing room location

You may find it useful to use some of these in your game, particularly for level / world locking or unlocking but this will be covered in a later chapter.

Note that the global.LS_paused variable is simply set to true or false; it does not physically pause the game. It is used to prevent some instances executing code in their events when it is set to true. As it is a global variable you can use it in any objects in your game you need to pause.

E.G.

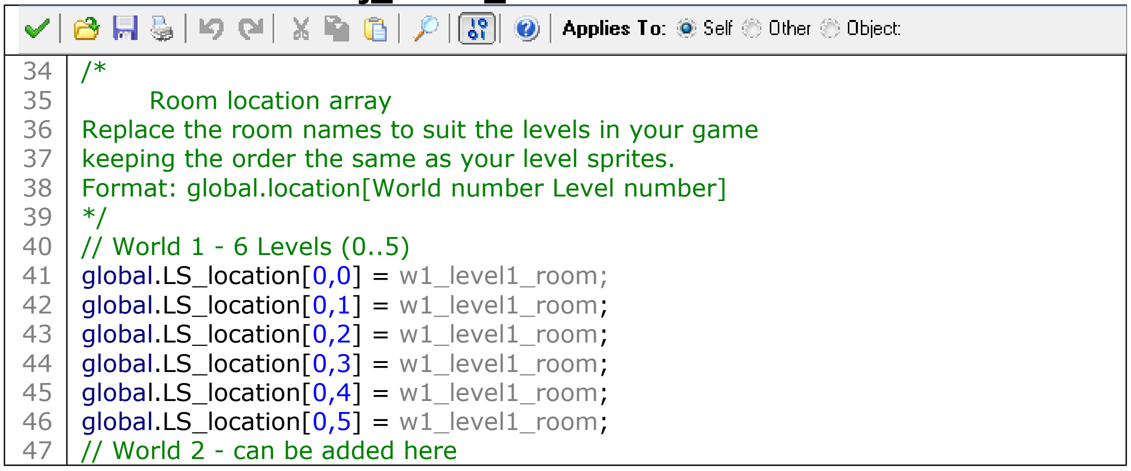
```
if(global.LS_paused == true) {exit;}
// Event code to execute when not paused goes here
```

Selecting a Level

The object of allowing a player to select a level is usually to take them to that level. The Selector Menu contains a system to allow you to specify which rooms to go to by adding the room locations you want each level to go to. These must be defined

The object obj_menu_init also defines these locations.

Game Start Event obj_menu_init



```
/*
      Room location array
Replace the room names to suit the levels in your game
keeping the order the same as your level sprites.
Format: global.location[World number Level number]
*/
// World 1 - 6 Levels (0..5)
global.LS_location[0,0] = w1_level1_room;
global.LS_location[0,1] = w1_level1_room;
global.LS_location[0,2] = w1_level1_room;
global.LS_location[0,3] = w1_level1_room;
global.LS_location[0,4] = w1_level1_room;
global.LS_location[0,5] = w1_level1_room;
// World 2 - can be added here
```

As supplied the Level Selector Menu contains 3 worlds but only the first world containing six levels is enabled. This tutorial is not a game thus only contains a single level room called w1_level1_room, which is used as a demo to simulate a level. You would of course replace these with your own game rooms.

These must be listed in the same order as your level sprites.

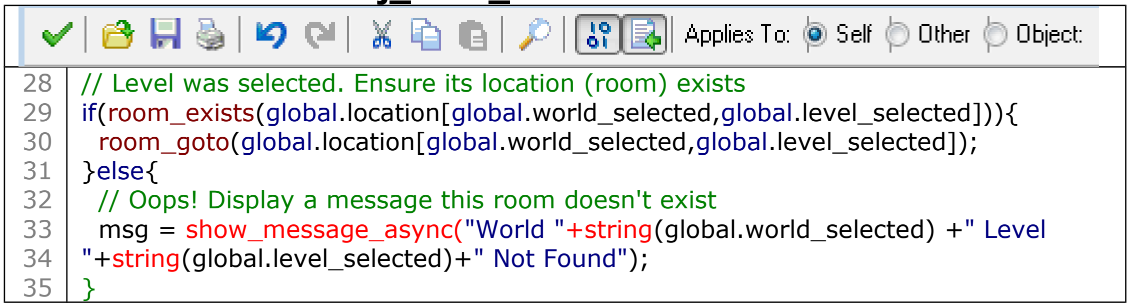
global.LS_location[0,0] refers to world 0 Level 0 – the first level in the first world. It contains the room you would like the player to go to when this room is selected from the level selector menu.
global.LS_location[0,1] would contain the room location for the next level in world one.
global.LS_location[1,0] would contain the room location for the first level in world two.

[*Arrays start at zero so an index of 0 refers to the fist one.*]

It might seem a little tedious if you have many rooms to add however it only has to be set up once and makes selecting the levels throughout the rest of the game a much simpler affair.

The code to take the player to the selected level should not need to be altered. It can be found in the Left Pressed Event of obj_menu_control however all you need do is populate the array in obj_menu_init with the names of your rooms for it to work.

Left Pressed Event obj_level_control


Applies To: [✓ Self] Other Object:

```
// Level was selected. Ensure its location (room) exists
if(room_exists(global.location[global.world_selected,global.level_selected])){
  room_goto(global.location[global.world_selected,global.level_selected]);
}else{
  // Oops! Display a message this room doesn't exist
  msg = show_message_async("World "+string(global.world_selected) +" Level
"+string(global.level_selected)+" Not Found");
}
```

Notice how we check the room exists before going to that room. GM will issue an error if you assign an array with an unknown variable but doesn't mind what that variable value is. Attempting to go to a non-existent room can cause all kinds of problems thus we ensure it exists and issue a warning message if it doesn't.

Level Locking

In some situations you may want to prevent certain levels from being selected until the player has completed a task or level.

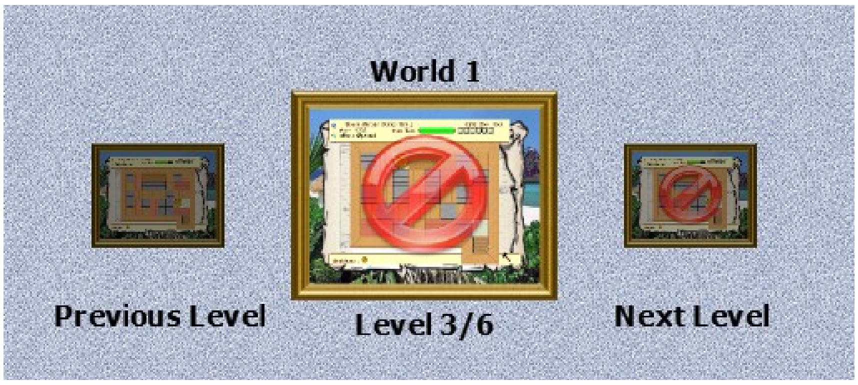
Level locking can seem quite complex at first, particularly if one is still learning the GML programming language and little or no experience of other programming languages or arrays. The basic concept is actually quite simple but often the mere mention of 2 dimensional arrays has the effect of putting novices off so a section is included to explain arrays in a simple and easy way.

Arrays are nothing to shy away from but if you are not conversant with array usage or unsure how to implement them see Arrays towards the end of this tutorial for a basic introduction.

The Level Locking Mechanism

The Level Selector Menu superimposes a **no entry image** over any level which is locked. You can of course change this image but it must contain two sub-images. Two alternative images to the no entry one (a Tick and a Padlock) are supplied with the download.)

The no entry sprite has its x, y origins centered with the first image being completely transparent – you will understand why momentarily.

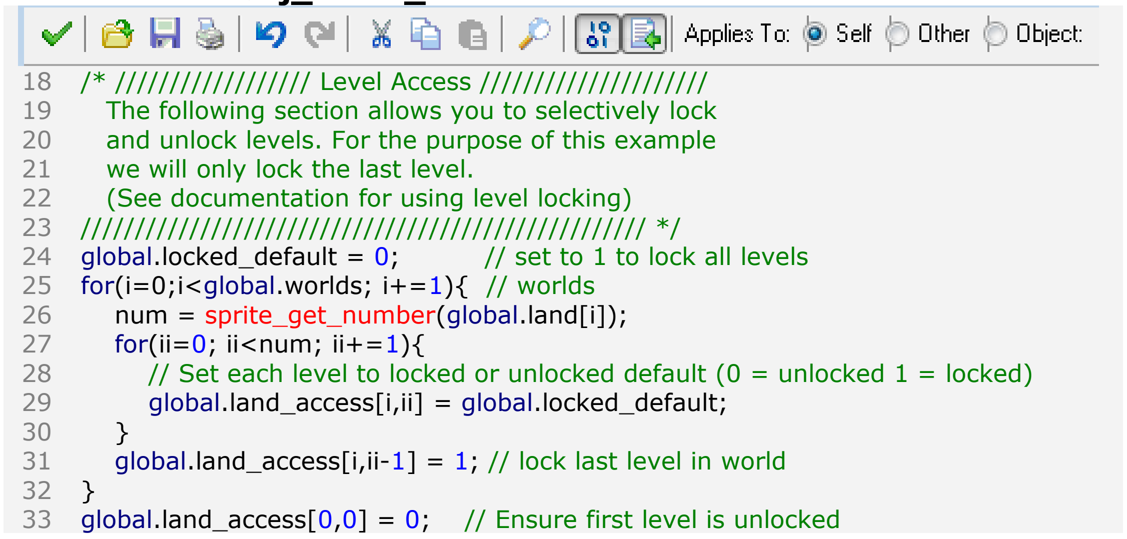


To lock or unlock a level we are storing either zero (unlocked) or one (locked) against that level. These values correspond to the no entry sub-images, superimposing either image zero or image one - transparent or the no entry symbol. Thus a locked level will show the no entry whilst an unlocked level just the level image without the need for further checks.

The Next and Previous locked levels will scroll to the middle as normal but a locked level cannot be selected from the menu. This does not prevent the actual level (room) locations from being entered but rather maintains a list of locations which you can use to grant or disallow access.

Setting up the initial level locking system needs only to be done once after the game is loaded so once again we use obj_menu_init for this.

Create Event obj_menu_init


Applies To: [✓ Self] Other Object:

```
/* ///////////////// Level Access ///////////////////
   The following section allows you to selectively lock
   and unlock levels. For the purpose of this example
   we will only lock the last level.
   (See documentation for using level locking)
////////////////////////////////////////////////// */
global.locked_default = 0;          // set to 1 to lock all levels
for(i=0;i<global.worlds; i+=1){  // worlds
    num = sprite_get_number(global.land[i]);
    for(ii=0; ii<num; ii+=1){
        // Set each level to locked or unlocked default (0 = unlocked 1 = locked)
        global.land_access[i,ii] = global.locked_default;
    }
    global.land_access[i,ii-1] = 1; // lock last level in world
}
global.land_access[0,0] = 0;    // Ensure first level is unlocked
```

This may look complex at first glance so let's examine what it does by breaking it down into its component parts. If you are conversant with using "for loops" in the above code you can skip to the Unlocking and Locking Levels section.

The routine consists of two **FOR loops**. As the name suggests the section of code between a set of curly brackets is executed in a loop whilst a particular condition is true. A **for loop** takes 3 arguments.

- First it sets a variable to use as a counter to a starting value.
- Next it tests the condition to be met.
- Finally it defines what happens at the end of each pass.

An open curly bracket after the loop definition signifies the start of the code to be executed and its matching closing curly bracket the end of the loop code. Note that the condition specified is tested **before** the start of each loop thus if the condition is never met the code inside the loop will not be executed.

The code above uses two loops, one inside the other. This is termed "nested loops" and requires a little thought, particularly when it comes to placement. Avoid putting them anywhere which gets executed on each execution step (like Step or Draw events) or performance can suffer.

In our case this loop structure is in a Create Event and does not have much to do.

The first loop sets "i" as its counter and executes the code it contains while this counter is less than the number of worlds. By convention "i" (standing for iteration, which is what all loops do) is used but you can use any variable you like as a counter.

The first part of the code in its loop initializes another for loop. So as not to mess up the first loops counter variable, this second loop uses ii as its loop counter. It loops through its code while the ii counter is less than the number of **levels** in that world, setting our default level_access each time.

When the number of **levels** in each world is reached the inner loop terminates allowing the outer one (the first one) to increment the **world counter** and start the inner loop once more.

Once the number of worlds is reached both loops terminate.

Since locked levels cannot be selected the last line, which is outside both loops, ensures the first level of the first world is always unlocked.

Locking the last level in a world prevents the next world (if there is one) from being selected. If you don't want to use any level locking you can remove or comment out the following line:

Game Start Event obj_menu_init

```
global.land_access[i,ii-1] = 1; // lock last level in world
```

Equally if you wanted all levels in your game to be locked initially you would simply set global.locked_default = 1.

In effect these two loops translate to do the following:

```
global.land_access[0,0] = 0;
global.land_access[0,1] = 0;
global.land_access[0,2] = 0;
global.land_access[0,3] = 0;
global.land_access[0,4] = 0;
global.land_access[0,5] = 1;
```

It may seem simpler to use the above instead of nested loops until you consider adding more worlds. For just two worlds, each with just 6 levels this would become:

```
global.land_access[0,0] = 0;
global.land_access[0,1] = 0;
global.land_access[0,2] = 0;
global.land_access[0,3] = 0;
global.land_access[0,4] = 0;
global.land_access[0,5] = 1;
global.land_access[1,0] = 0;
global.land_access[1,1] = 0;
global.land_access[1,2] = 0;
global.land_access[1,3] = 0;
global.land_access[1,4] = 0;
global.land_access[1,5] = 1;
```

Addling more worlds and allowing for each world to have a different number of levels would become difficult to follow at best and become a nightmare to debug if things go wrong or you make changes later.

Using a couple of loops to set things up for us simplifies things. As these are only executed at the start of the game there is no danger of causing any frame rate drop (or game lag as it is sometimes known as.)

The two loops allow for worlds having a differing number of levels and setting all levels initially to a default state ensures the array is properly defined. If you miss out an array element which the game tries to access you are likely to get a variable or array index out of bounds error.

Unlocking and Locking Levels

It is likely you will want to unlock (or even lock) levels when the player completes a task or level. Although the mechanics for the level locking is in place, using it becomes game-specific.

- You might want the next level to become unlocked only when the current one is completed.
- You might want the player to collect a number of items before the next level becomes available.
- You might want allow any level in the current world to be selected and visited but lock the next world's levels until all these levels are completed.

Clearly there is no "one size fits all" solution but since our global.level_access array is a global variable, once defined it can be accessed from any object or event from anywhere in your game.

The biggest mistake you can make is to attempt to access an array element (index) which does not exist.

For example if a player has completed a level in the first world we couldn't simply unlock the next level by increasing the level number. The Level Selector Menu allows you to have a different number of levels in each world therefore we are likely to get an "Index out of bounds" error if we attempted to access an array element which does not exist.

The Level Selector contains some useful global variables to help you.

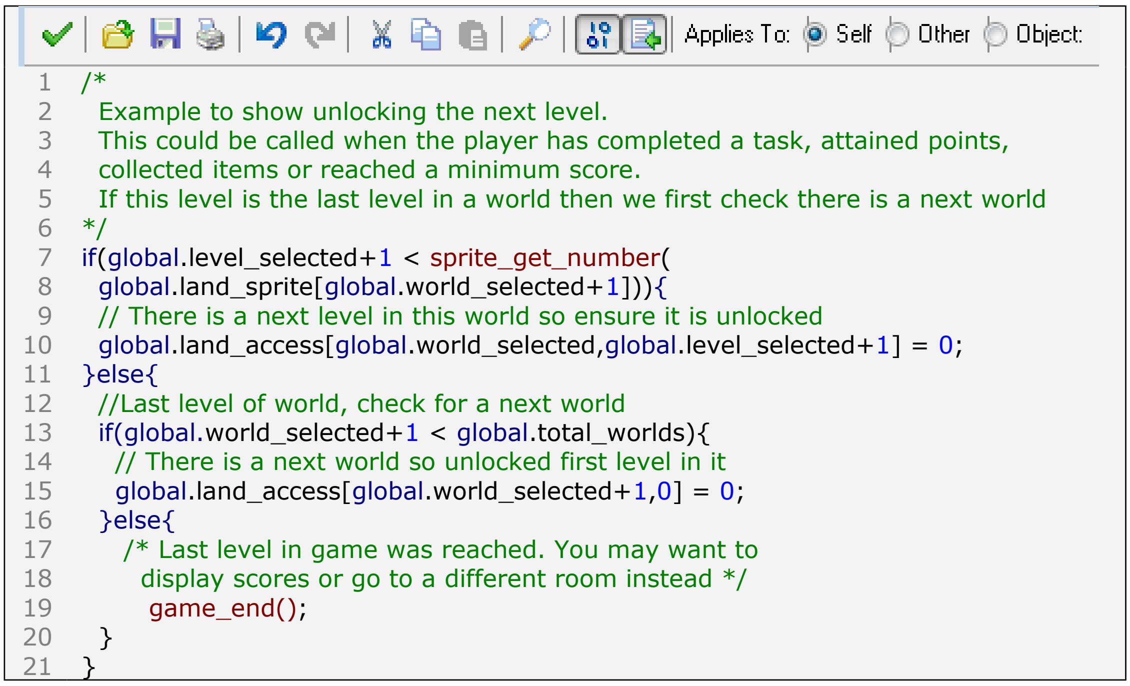
global.world_selected	This contains the currently selected world the payer is currently on – zero being the first one.
global.level_selected	This contains the current level for the world the player is on.
global.total_worlds	This contains the total number of worlds in the game.

Using these variables allows you to create a level locking system which is as simple or as complex as you care to design.

You can, for example lock the current level by way of a penalty, forcing the player to play the previous level again in order to unlock it once more. You can also have bonus levels which remain locked until the player completes specific tasks.

In most cases it is likely you will only want to unlock levels either when a previous level has been completed, specific tasks done or sufficient points or score attained.

Unlocking the Next Level Example



```
/*
  Example to show unlocking the next level.
  This could be called when the player has completed a task, attained points,
  collected items or reached a minimum score.
  If this level is the last level in a world then we first check there is a next world
*/
if(global.level_selected+1 < sprite_get_number(
  global.land_sprite[global.world_selected+1])){
  // There is a next level in this world so ensure it is unlocked
  global.land_access[global.world_selected,global.level_selected+1] = 0;
}else{
  //Last level of world, check for a next world
  if(global.world_selected+1 < global.total_worlds){
    // There is a next world so unlocked first level in it
    global.land_access[global.world_selected+1,0] = 0;
  }else{
     /* Last level in game was reached. You may want to
       display scores or go to a different room instead */
        game_end();
  }
}
```

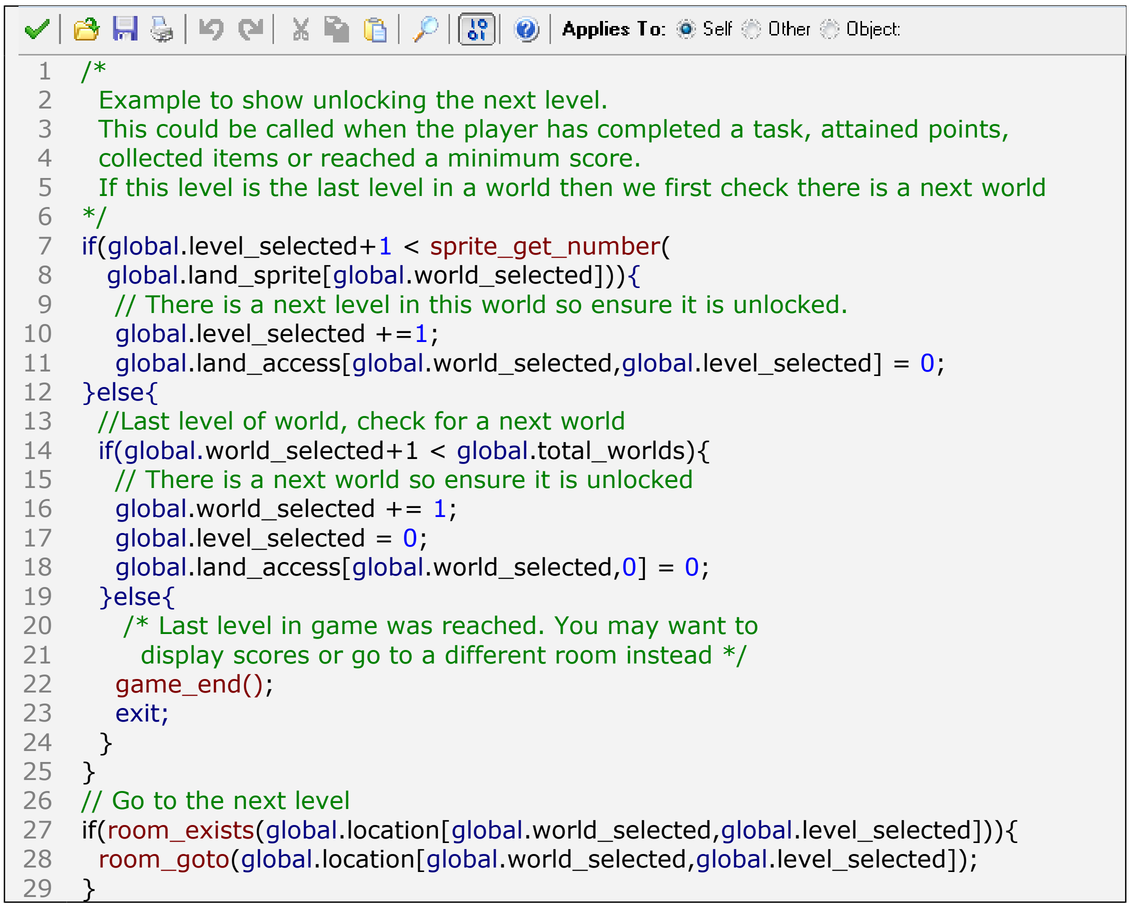
Locking and unlocking levels simply means they cannot be selected from the menu. If you want to actually prevent access the onus is on you to first check a specific level is locked or unlocked before the actual move.

The above example simply unlocks the next level (or first level in the next World). It does not move to the new level or increase the world or level counters. It could be called when a player has accomplished something to make an exit appear which, until it appeared, would prevent the player from visiting it anyway.

Quitting back to the Level Selector Menu will display the currently selected world and level using the global.world_selected and global.level_selected variables. It does not however preserve the current game. If you need this functionality you must create your own game saving routines.

In many cases you might want to unlock a level prior to moving to it.

Unlocking and Going to the Next Level Example


Applies To: [✓ Self] Other Object:

```
/*
  Example to show unlocking the next level.
  This could be called when the player has completed a task, attained points,
  collected items or reached a minimum score.
  If this level is the last level in a world then we first check there is a next world
*/
if(global.level_selected+1 < sprite_get_number(
   global.land_sprite[global.world_selected])){
    // There is a next level in this world so ensure it is unlocked.
    global.level_selected +=1;
    global.land_access[global.world_selected,global.level_selected] = 0;
}else{
  //Last level of world, check for a next world
  if(global.world_selected+1 < global.total_worlds){
    // There is a next world so ensure it is unlocked
    global.world_selected += 1;
    global.level_selected = 0;
    global.land_access[global.world_selected,0] = 0;
  }else{
     /* Last level in game was reached. You may want to
       display scores or go to a different room instead */
    game_end();
    exit;
  }
}
// Go to the next level
if(room_exists(global.location[global.world_selected,global.level_selected])){
  room_goto(global.location[global.world_selected,global.level_selected]);
}
```

The code above is similar to that used in the Step Event of obj_room_goto. It could be called anywhere in your game when you want to the player to go to the level.

Locking Completed Levels

You might want to only allow the player access to a single level at a time and also prevent them from selecting easier levels and thus increasing their scores. You could lock all levels by default (the Level Selector will unlock the first one for you) and as each level is completed it can be locked and the next one unlocked.

We can lock the current level using the following line.

```
global.land_access[global.world_selected,global.level_selected] = 1; // lock this level
```

Remember that locking a level does not in itself prevent access to it. Thus a player will not get kicked out of a room if we lock it. Locking alone simply prevents that level from being selected from the menu.

Adding the Level Selector to Your Game

Once you have finished your customization and tested it all works as expected you can add it to your game.

Ensure you save your work before continuing.
Game Maker Studio can be set to automatically keep backups however these should not be relied upon. A backup is created each time you compile the game to test it so all backups can very soon become those of a broken game.

A more robust (and professional) approach is to create your own backups at milestones by exporting the project to a GMZ export file, giving it a meaningful name. Backuo01, Backup02, Backup03 etc is less than helpful.

You can export as many "backups" as you need, either overwriting previous ones or creating new ones, but this is best performed AFTER testing new additions.

E.G
Blitz_Boss_added.gmz
Blitz_19Levels.gmz
Blitz_hiscores_added.gmz

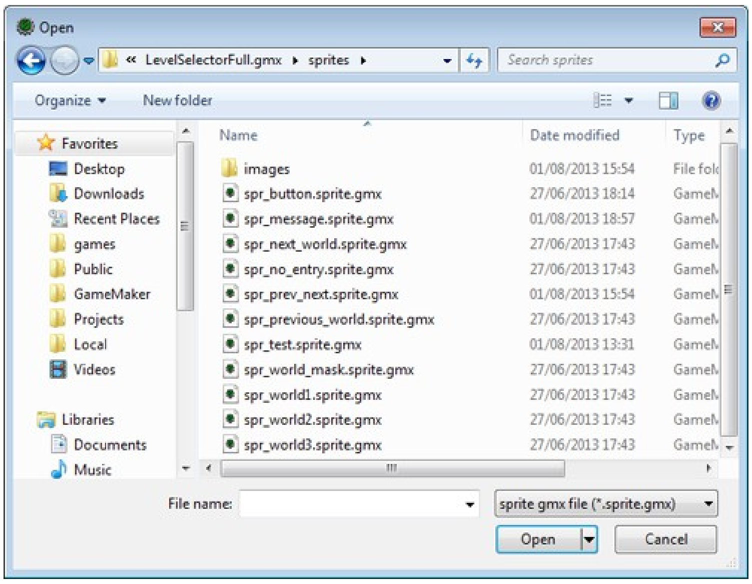
If all goes terribly wrong you can go back to a known working backup at a specific stage of development by importing the relevant exported file to create a new project from.

Going back to older "versions" will lose any later additions however it can be easier to work with a working game than to debug a broken one.

Adding Resources

In previous versions of *Game Maker* only registered versions could import resources. Even so, it was a complicated process which could easily lead to a corrupt game or loss of data. *Game Maker Studio* makes sharing resources between projects a much simpler and painless procedure.

To add resources from another project to the current one select the resource type you want to add from the resource tree on the left (EG Sprites) Right Click and select **Add Existing Sprite**. A file selector window will open displaying the folders containing your projects. Navigate to the project you want to import resources from. If importing sprites select the Sprites folder and select the sprite resources you want to import.



Name	Date modified	Type
images	01/08/2013 15:54	File fol
spr_button.sprite.gmx	27/06/2013 18:14	GameM
spr_message.sprite.gmx	01/08/2013 18:57	GameM
spr_next_world.sprite.gmx	27/06/2013 17:43	GameM
spr_no_entry.sprite.gmx	27/06/2013 17:43	GameM
spr_prev_next.sprite.gmx	01/08/2013 15:54	GameM
spr_previous_world.sprite.gmx	27/06/2013 17:43	GameM
spr_test.sprite.gmx	01/08/2013 13:31	GameM
spr_world_mask.sprite.gmx	27/06/2013 17:43	GameM
spr_world1.sprite.gmx	27/06/2013 17:43	GameM
spr_world2.sprite.gmx	27/06/2013 17:43	GameM
spr_world3.sprite.gmx	27/06/2013 17:43	GameM

You can select multiple files in this way too and they will all be imported into *Game Maker Studio*, but **they must all have the same sub-extension** (ie: you can load ten *.object.gmx at once, but not a *.sound.gmx and a couple of *.sprite.gmx).

Note: If you add an object resource into your game which has a sprite associated with it *this sprite will not be imported*.

You probably won't want all resources from the demo in your game. Appendix B contains a reference list of all resources to help you identify those you need.

Troubleshooting Guide

Most problems are as result of simple mistakes or omissions which are easily corrected. This section helps you to quickly locate the possible problem.

Level Images don't scale properly
Ensure the sprite origins for the level images are centered.

A message World x1 Level x2 appears
This indicates the array contains a location which is not a room. The x1 reported is the world number and x2 the level number in that world. You can look these up in the Create Event of obj_menu_init. For example World 1 Level 1 would correspond to global.location[1,0].

Levels always appear locked even when they can be selected.
Check the no_entry sprite contains two images and the first is fully transparent.

Array Index is out of bounds after adding new worlds.
Check the global.location array contains the correct locations for each of your levels and there are no gaps in the index numbers. To assist you pressing F12 in the menu will list the locked and unlocked levels for that world and also the room locations each go to.

New World does not appear.
Ensure you have edited obj_menu_init to include the world sprite and altered global.total_worlds to include the newly created world.

Appendix A - Arrays

An array is like any variable in as much it is a container for a value which you give a name to. Like all variables you create arrays can be local to an instance (meaning each instance will have a copy with its own values) or global so all objects using them share the same values.

The key difference is that an array variable can contain several values. GM handles arrays slightly differently from some other programming languages but the basic structure is much the same.

Imagine the array as a column of numbered boxes:

Variable Name: Fruit:

(Index)	(Value)
0	Apple
1	Pear
2	Banana
3	Grape
4	Orange

Each box number (index or element as it is known) can contain a different value which we can access using its index number.

Note that an array index always starts from zero, thus the first box (or index number) is zero not one.

In the above example we can access each box by using its index number enclosed in square brackets.
Fruit[0] contains Apple
Fruit[1] contains Pear
Fruit[2] contains Banana
Etc…

All alarm events are local arrays. When dragging an alarm action, selecting alarm number 0 and setting the steps value to 30 you are actually setting a value stored in an array. In GML this translates to:
alarm[0]=30;

Arrays are also useful to store data which we can look up without having to use multiple lines of code or if…else statements. If a game had 4 different items to collect we would not need to check each one to find out what item 4 was they collected; a single line could do it for us.
collected = Fruit[item]; // collected = Orange

Unlike data structures there is no function to return the size of an array so it is important to record how many "things" are contained or, to be more precise the last index element in every array your create. If you attempt to access data from an array for an index which does not exist you will get an undefined variable or array index out of bounds error.

Two Dimensional Arrays

These are similar to single arrays but have two indexes rather than just one. To create our level locking system we need to store a value against each level in our game. Our game will contain several levels in each world and may contain several worlds so we need to use both the world number and the level number to lookup our locked data.

Variable: level_access[world,level]

(world)	(level)	(Value)
0	0	0
0	1	0
0	2	0
0	3	0
0	4	0
0	5	1
1	0	0
1	1	0
1	2	0
1	3	0
1	4	0
1	5	1

Two dimensional arrays take two indexes. In this case the first index is our world number and the second one our level number. The value contained in each element can be anything we like but we will use a zero to represent that level is unlocked and one if it is locked.

To see if any level is locked we just need to use the world and level number and test the value.

Similarly we can lock or unlock any level.

The table above shows the last level of each world is locked and all other levels unlocked. You access a 2D array in GM much the same way as a single array (using indexes enclosed in square brackets) but supplying two indexes separated with a comma.

```
if(level_access[world,level] == 0) {
   // This level is unlocked do something here
}
```

Locking a level is simply a matter of setting the value for a specific level.

```
level_access[world,level] = 1;
```

In reality you would probably not want to check (or lock) just the current level and so would need to ensure the level you want to access existed prior to checking. Even with these addition checks the coding would become much simpler than individually checking each level.

By default all variables you create are local to the instance which creates them, thus each instance will have a copy of the variable unless you specify them as **global variables**. Arrays are no different and since we want to be able to access our arrays from any instance in any part of the game we make them global by prefixing them with the keyword global. (global dot).

Array Usage

Whilst 2D arrays seem the easiest to use they can consume more memory than single arrays, particularly when storing large amounts of data like strings.

Appendix B – Object Use

The Level Selector Menu contains several resources which are only for this example and you won't need in your game. Font Arial12 (used by obj_menu_control) can also be omitted if not displaying world and level numbers.

Most of the customization is done in obj_menu_init and by changing / adding your own sprites. You will probably want to pretty up the menu display to fit your game. You can use your own objects for this or alter obj_menu_control.

obj_menu_init:
This initialization object contains just one event. Its sole job is to initialize the global variables the level selection menu will use so should be called just once at the start of the game and not returned to. **Most of the customization is performed in this object.**

obj_menu_ctontrol:
This object is the selection menu controller. It displays the center image and creates the previous and next menu objects. Selecting this object takes the player to that level or world. Locked levels are superimposed with a no entry symbol and cannot be selected.

obj_menu_next and obj_menu_previous:
These display the next and previous menu items respectively and need not be altered. Selecting (using the mouse or pressing the Left / Right arrow keys) has the effect of rotating one or the other to become the center image. A no entry symbol is superimposed onto the image if either is locked but they can still be scrolled.

obj_menu_move:
Displays the moving images giving the impression the selected images are rotated. The images are scaled proportionate to the direction they move.

The following objects are not required:

obj_room_goto:
Simulates completing a level and unlocking the next or failing and returning to the selection menu.

Obj_complete and obj_failed:
Displays the Level Complete and Level Failed buttons used by obj_room_goto.

obj_instructions:
Displays the instructions for using the menu.

Additionally the sprites, backgrounds should be replaced to suit your game.

www.software4me.org/contact