Handbook

A user’s guide to SMBX2 b4

Introduction

SMBX2 (2.0.0 b4, henceforth referred to as SMBX2 b4) is the latest stable build of SMBX2, as of January 2020. It is a direct update to the last preview build 2.0.0 b4 p2 (a.k.a. SMBX2 PAL), and fixes many of the issues present in that version. **As such, it is stable for use creating episodes as well as levels.**

Please refer to the **Encountering Bugs** segment below for how to proceed if you encounter anything out of the ordinary.

Installation

If you haven't already, start by downloading the program from here.
The installer is directly created from a ZIP archive of our repository and will deploy the project into the directory you specify. The package will be installed to "$TARGETDIR/SMBX2", but you're free to change the name of the folder afterwards.

Add_leveltest_assoc.bat

The script of this name in the data folder can be used in order to enable you to open lvl and lvlx files by double-clicking them, without the need to go through the editor.

Editor

The Editor can sometimes be a tricky UI to navigate. If you know where to look, however, you will be able to find what you need fairly quickly. Below are some pointers to the most important features of the editor.

Important buttons and options

The following options in the editor are among the most commonly used ones. You can learn about the functions of the buttons that are not listed by hovering over them for a second.

The button with this icon opens the Tileset Itembox, the main method of adding objects to a level. Its functionality is further explained in a later segment.

Before you can test your level, you need to place a start point for the first player. Green Mario's start point is optional.

These buttons are used to draw zones of water and quicksand respectively. These zones can be moved and resized after placement, too.

Opens the "Warps and Doors" window, in which warps can be configured. There are various unsupported features in this window, all of which are further detailed in the "Unsupported Features" segment below.

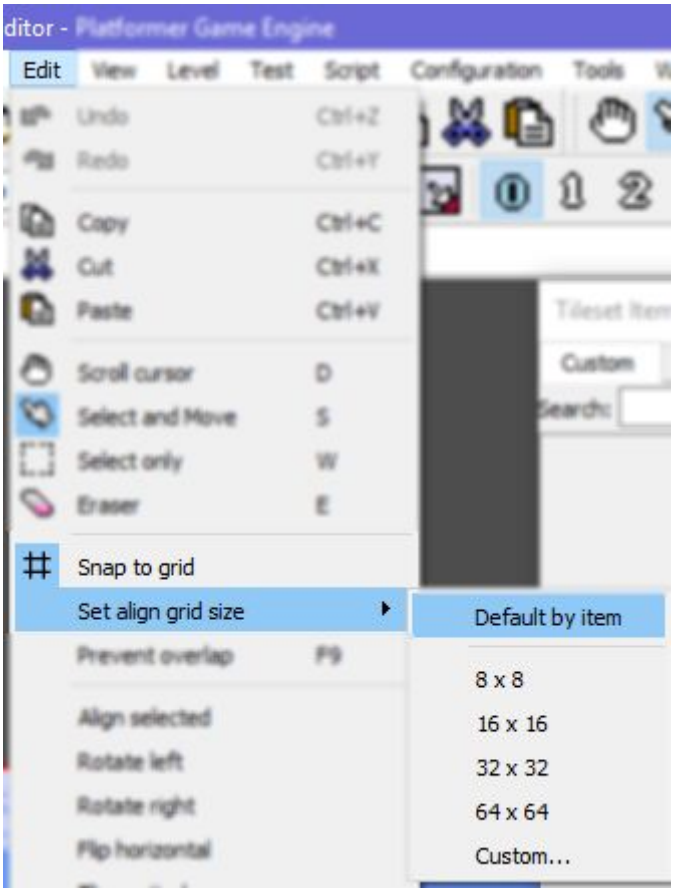
Opens the "Section Settings" window, in which various section-specific properties can be configured. Unsupported features of this window are listed in the "unsupported Features" segment.

Open the "Layers" and "Events" windows respectively. Together, these can be used to create moving layers and toggle layer visibility. Events can further be used to play sound effects, lock player input, manipulate the section and even execute lua code (using LunaLua's onEvent event).

The section buttons transport you to the specified section instantly.

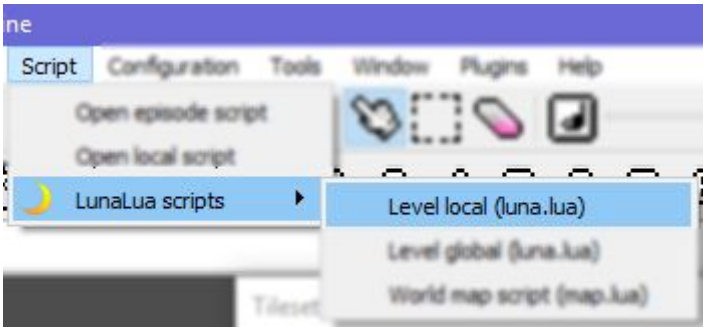
Warning! Although there is a button to the right for adding sections beyond Section 20, such extended sections are not currently supported by SMBX2. Further unsupported editor features can be found in the “Unsupported Features” segment below.

Enabling a lock will prevent any of the tiles of that type to be interacted with. From left to right, the locks are: Blocks, Background Objects, NPCs, Warps, Liquids.

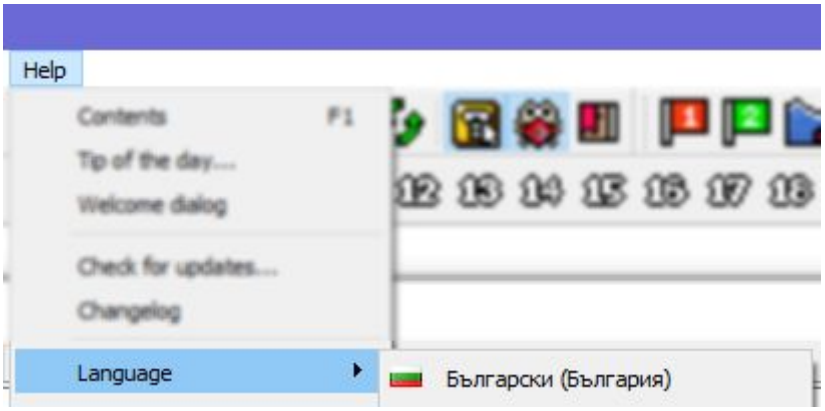


“Snap to grid” and manipulation of grid size can be used for more granular placement of elements such as blocks and background objects. “Default by item” describes the default grid alginment option.

If you are starting to notice that the editor is lagging, disabling the animation with a click of this button should help!



Opens (and, if necessary, creates) a lunalua script in the level folder of the currently active level.



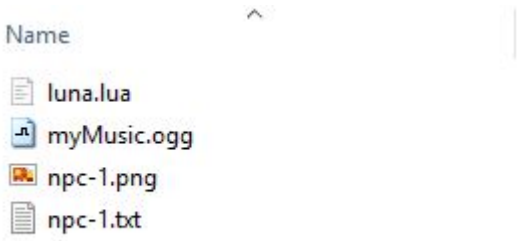
Changes the editor's language. Translations may not always be perfectly accurate, but if English doesn't do the job for you, hopefully you will be able to find your way around more easily with a different language!

Setting Up Your Level Folder

In order to use custom assets in your level, you best adhere to the recommended folder structure. The following figure displays what is often referred to as the "Episode Folder" (since, if this level were part of an episode, it would be the directory in which every level would be located).

Make sure you're using lvlx files rather than the old lvl file format. The old format doesn't support various new features, like per-npc behaviour settings for certain NPCs.

Above the .lvlx file is a folder using the same name as the level file. This folder is what's called a "Level Folder". In it you can place custom assets which will be applied locally to the level:

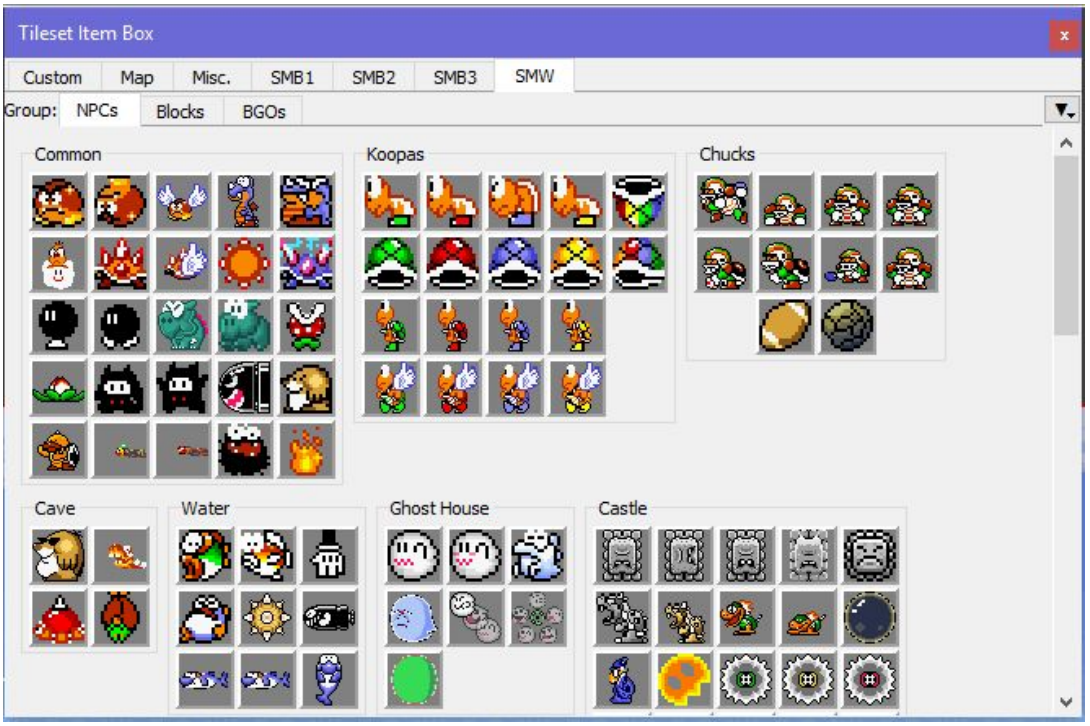


From lua scripts, to custom graphics, to music - any custom asset for the level should be placed in here to make it obvious which level what assets belong to.

If you are already testing your level in the editor and are making changes to the custom graphics in the level folder while editing, you need to press the **F8** key in order to refresh the contents of the level folder, in order to get them to display properly in the editor.

Using the Tileset Itembox

The Tileset Itembox is the main container for all elements accessible in SMBX2.

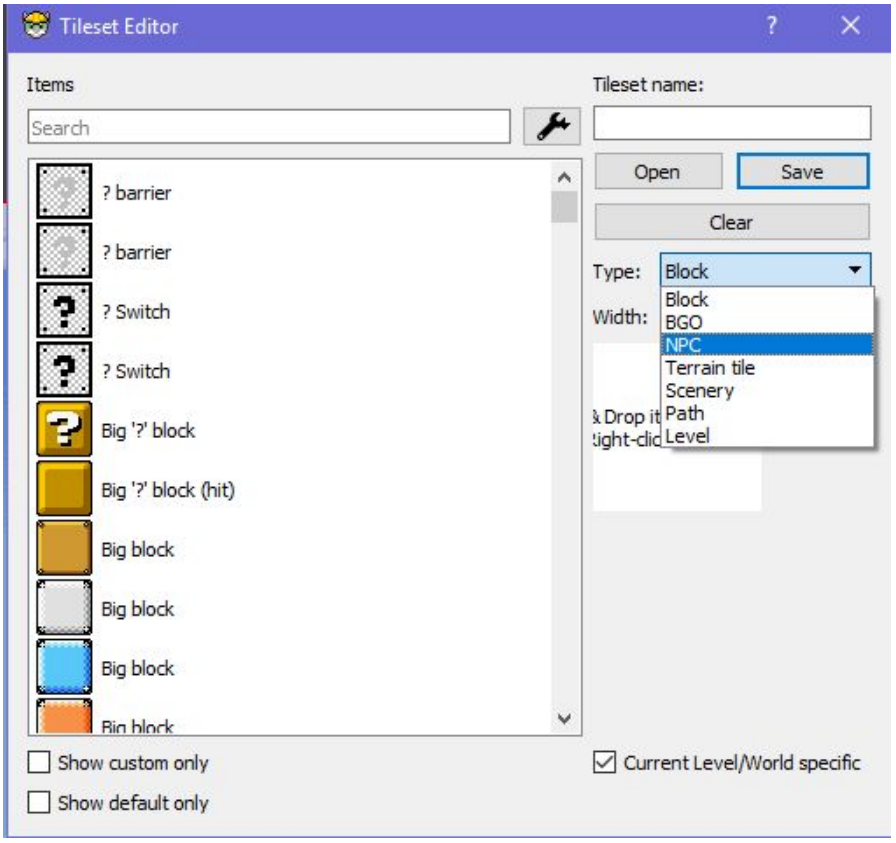


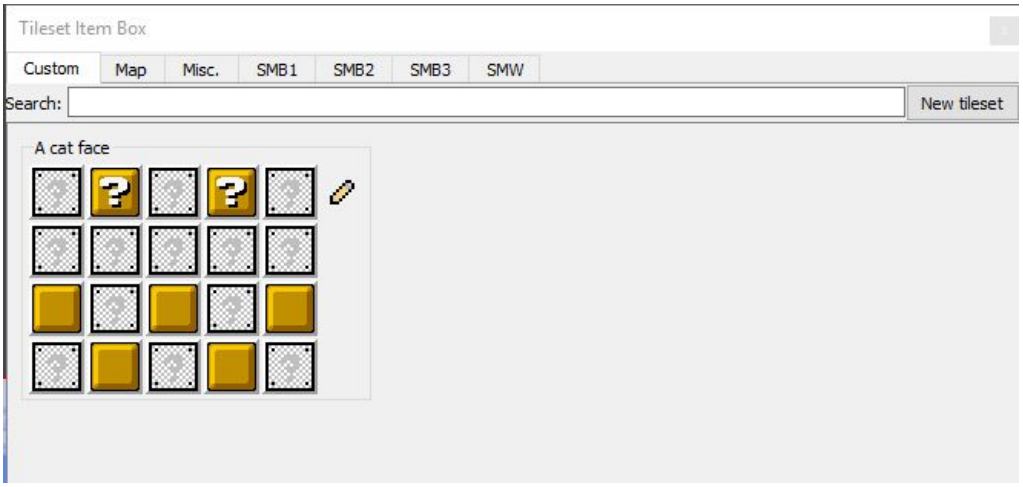
By switching between the different tabs, you are able to navigate to the items you want to place in your level.

Warning! We are still looking into hiding the “Map” tab unless a world map is being worked on. Please ignore that tab.

Even though quick access to all features is nice, there is still a lot of clicking involved in order to get to the items you want to access. This is where the “Custom” tab is helpful.

With a press of this button, you will be brought into a Tileset Editor, in which you can create your own tilesets of any kind.
The editor features a search bar, with which you can search by an item’s name, as well as its ID. You can also filter tiles in order to only show sprites with custom assets, or sprites without. After giving the tileset a name and hitting “Save”, it will be saved into the level folder and the tileset will automatically be displayed as part of the “Custom” tab in the Tileset Itembox.



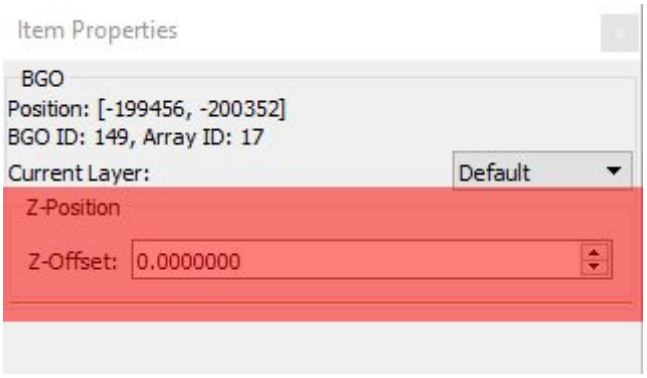


Certain features have been specifically removed from the Tileset Itembox for this build. This is because these NPCs, Blocks, BGOs, etc. are unfinished and likely to behave unexpectedly, or cause crashes or errors. If an NPC you are looking for is not in the Tileset Itembox, please avoid using it in your levels, as it is very likely to behave differently in future releases.

Unsupported Features

Since the Editor is not exclusively used by SMBX2 but also by its own engine (currently in alpha), there are various features present in that engine which aren't implemented in SMBX2. While most of these are hidden, the sections below highlight the remaining unsupported areas, all highlighted in red.

BGO Settings



Sections

Adding sections beyond Section 20 is unsupported.

Testing

With a press of the **F5** key, a SMBX window will open that allows you to test your creation. You can tab out of the Test Window at any time to make changes to your level in the background. Another press of the **F5** key then refreshes the Test Window with these new changes.

The Testing Menu

The Testing Menu has been massively upgraded since its Beta 3 incarnation. Its features include the following:

- Setting the amount of players to test with
- Start position for testing. Available options:
    - Level start point
    - Any checkpoint
    - Any warp exit
- Colorblind filters. Available options:
    - Protanopia
    - Protanomaly
    - Deuteranopia
    - Deuteranomaly
    - Tritanopia
    - Tritanomaly
    - Achromatopsia
    - Achromatomaly
- Selecting the character for each player, including all new characters. You can also:
    - Set their powerup
    - Set their mount

The Console

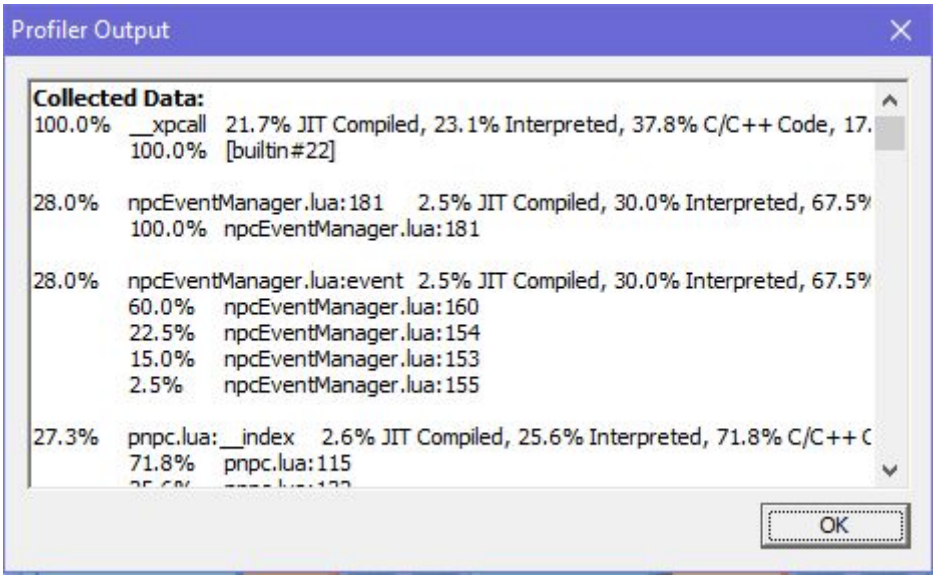
A press of the **TAB** key during gameplay brings up the developer console.

The console can be used to execute lines of lua code. This is particularly useful when working with permanent storage using SaveData. The SaveData library will be further discussed in the LunaLua segment of the handbook.

Performance Debugging

If you are noticing poor performance in your level, you can investigate it by pressing **F3** during gameplay, which brings up the profiler.

Profiling lasts until **F3** is pressed again. The second press opens a dialog window with the results:



Results are ordered by impact on performance. Individual entries detail the line numbers, allowing you to track down the lines of code that contribute to your level running poorly.

Many basegame-related libraries are not easy to understand, and if you run into performance issues with basegame libraries, we recommend talking to us directly on the codehaus discord server.
https://discord.gg/aCZqadJ

Red Warnings

While playing levels that make use of various libraries for SMBX2b3 (2.0.0 Beta 3) in this build, you might encounter briefly visible red text near the top of the screen when starting a level. This message and the corresponding dialog box are only visible in the editor and warn of deprecated libraries that the level's lua code uses, or other non-vital errors. The libraries in question are slated for removal in a future version of SMBX2, one version after a replacement has been provided. By updating early, you can make sure your levels retain compatibility across a leap between versions.
If you want to continue using deprecated libraries for niche reasons, make sure to make a local copy inside your level or episode folder for it, so that the removal from basegame in a future version doesn't affect your work in the long run.

Encountering Bugs

If you encounter unexpected behaviour or blatant crashes, please use the bug-shaped icon on the launcher to submit a bug report.

New Features

The _templates Folder

A lot of features in this section make use of specific files and need those to be formatted in a certain way in order to function. We have included a folder called “_templates”, which includes various dummy files for different purposes. Here is how you can use them:

ach-n.ini

Template for an achievement definition. When put into an episode’s “achievements” folder, this will show up as one of the achievements on the launcher. More info on achievements can be found further down in the document.

Dummy.png

A generic image file referred to in multiple other configuration files in the dummy folder. Only needs to be copied for testing before the image is replaced.

background2-n.ini, background2-n.txt

A template for parallaxing backgrounds, discussed in the section below. Copy it and Dummy.png to your level folder and rename the “n” in the file to the number of the background you want to replace, then make changes to the file until you get the desired result! Your should, however, delete all the fields you aren’t using, since some will override others and give unexpected results!

background-n.ini, background-n.txt

A template for new BGOs in the 751-1000 range detailed below. Take a look in the files after copying them over to your level folder to see different customization options. The “ini” file determines the editor appearance, while the “txt” file has effects on the game while it’s played. Rename the “n” in the files to the ID you wish to occupy and make sure the BGO has an image file associated with it (for example background-751.png).

Block-n.lua, block-n.ini

A template for new Blocks in the 751-1000 range detailed below. Rename the “n” in the file to the number of the Block ID you want to occupy and make sure the Block has a corresponding image file associated with it. The example block-n.lua will simply destroy the block as a check to see if everything works. The “ini” file determines the editor appearance, while the “lua” file has effects on the game while it’s played.

block-n.txt

A template for a Block config file for existing Blocks. Includes defaults for all configurable options. Excess options can be removed.

npc-n.lua, npc-n.ini

A template for new NPCs in the 751-1000 range detailed below. Rename the “n” in the file to the number of the NPC ID you want to occupy and make sure the NPC has a corresponding image file associated with it. The example npc-n.lua will simply jump up and down in place, as a check to see if everything works. The “ini” file determines the editor appearance, while the “lua” file has effects on the game while it’s played.

npc-n.txt

A template for a NPC config file for existing NPCs. Includes defaults for all configurable options. Excess options can be removed.

example.lua

A template for a LunaLua library file. Contains various example functions, but doesn’t do anything by itself. Copy the file over and load it to get started more easily with creating your own libraries.

particles_example.ini, ribbon_example.ini

Configuration files for particle emitters and ribbon trails for the particles library discussed further below in the handbook. Copy the files over and play around with the configuration options to get a feel for how the systems are set up! Don’t forget to load the emitter and draw it, too! More details in the section about particles.

music.ini

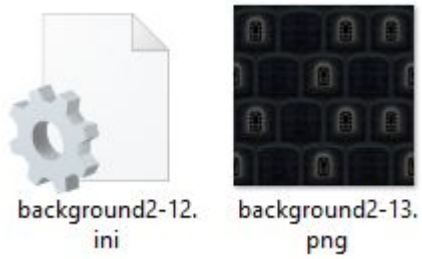
Configuration files for the global music locations. Allows you to customise music on a per-level or per-episode basis, depending on whether the files are placed in the level or episode folder. Simply rewrite the file paths as needed.

standard.vert, standard.frag

Example files for GLSL shaders. Shaders and the details of these files are discussed further in the Shader Programming section of the handbook.

Parallaxing Backgrounds

Creating parallaxing backgrounds in levels has been streamlined to the point where no user-written lua code is necessary for most purposes.
Parallaxing backgrounds are now automatically applied based on the file name of a corresponding .txt (or .ini) file:



Worth noting is that the naming for the txt files follows SMBX's **internal** numbering for backgrounds, which may sometimes differ from the numbering of the .png files. If the numbers are off in certain places, however, they're never off by more than one. The example above replaces the SMW Forest background. The correct numbering is visible in the editor when selecting a section background.

The txt file can pull from other images in the folder and define the features of the background's individual layers, like so:

```
[Backdrop]
name="BG"
depth=450
alignY=TOP
img="prlx_clouds.png"
fitY=true
repeatX=true
speedX=-0.1

[Grid1]
name="Grid1"
depth=150
alignY=BOTTOM
img="prlx_foreground.png"
repeatX=true
repeatY=true
```

Below is a list of parameters for each layer of the background:

```
img           --(file name) REQUIRED: the image to draw in this layer
name          --(string) A name for the layer, used by Background:Get. Defaults to "Layer#", where # is the
layer index

x, y                    --(numbers) layer offset from top left of boundary, defaults to 0,0
depth                   --(number or INFINITE) Depth at which to position the layer (0 = in line with scene,
                          >0 = behind scene, <0 = in front of scene), computed from fit if not supplied
                          Default of depth.INFINITE if fit is disabled
fitX, fitY              --(booleans) Should the layer attempt to fit its parallaxing to the boundaries?
priority                --(number) Render priority, computed from depth if not supplied
opacity                 --(number) How transparent this layer should be, defaults to 1
speedX, speedY          --(numbers) How fast this layer should move of its own volition, defauts to 0,0

parallaxX, parallaxY    --(numbers) Override for parallax scrolling speed (0 = no scrolling,
                          1 = scroll with scene,  >1 = scroll faster than scene)
repeatX, repeatY        --(booleans or numbers) How many repeats of this image should be applied?
                          0 or true = infinite repeats, 1 = no repeats, >1 = n repeats

padX, padY                      --(numbers) Padding to place between repeated images
marginLeft, marginRight         --(numbers) Padding to the side of the layer
marginTop, marginBottom         --(numbers) Padding to the top/bottom of the layer

margin                  --(table of numbers) A table containing all 4 margins, named
hidden                  --(boolean) Should this layer be hidden? Defaults to false
frames                  --(integer) Number of animation frames, defaults to 1
framespeed              --(integer) Frame timer between animation frames, defaults to 8
alignX, alignY          --(LEFT, TOP, RIGHT, BOTTOM, CENTRE) Alignment for the x and y coordinates.
                          Defaults to LEFT/TOP
```

In addition to the per-layer properties, a “fill-color” property can be defined, which can be set to color constants or hexadecimal colours. Further information on the color constants can be found below.

You should define as few of these per-layer properties as you need, since some of them will override the effects of others (such as parallaxX and parallaxY overriding depth).

TXT Files for Blocks

Blocks can now be modified through TXT files, similar to how NPCs have been able to since SMBX 1.3. The following parameters are supported:

```
frames              --number of frames on the block sprite, defaults to 1
framespeed          --Animation speed of the block. Lower=faster, defaults to 8
width, height       --Determines the dimensions of a frame on the spritesheet. Automatically inferred from
                      spritesheet dimensions and frame count by default.
sizable             --If true, the block is a sizeable block.
semisolid           --If true, the block is semisolid.
passthrough         --If true, the block has no collision.
lava                --If true, the block is lava.
floorslope          -- Defines a floor slope collision. The value defines the direction: -1, 1.
ceilingslope        -- Defines a ceiling slope collision. The value defines the direction: -1, 1.
bumpable            --If true, the block is bumpable.
smashable           --Enum for smashability. Values from 0 to 3 are valid.
                      1: Destroyed, but blocks the smashing entity.
                      2: Hit, but blocks the smashing entity.
                      3: Destroyed, but does not block the smashing entity.

Darkness.lua-related:
lightradius                 --Radius of light
lightbrightness             --Brightness of light
lightoffsetx, lightoffsety  --Light offset relative to center of the sprite
lightcolor                  --Color constant or hex color specifying the light's colour
lightflicker                --If true, the light source flickers
```

TXT Files for Background Objects

Background objects can now be modified through TXT files, similar to how NPCs have been able to since SMBX 1.3. The following parameters are supported:

```
frames              --number of frames on the bgo sprite, defaults to 1
framespeed          --Animation speed of the bgo. Lower=faster, defaults to 8
priority            --render priority, defaults to -85
width, height       --Determines the dimensions of a frame on the spritesheet. Automatically inferred from
                      spritesheet dimensions and frame count by default
climbable           --Toggles climbability, defaults to false

Darkness.lua-related:
lightradius                 --Radius of light
lightbrightness             --Brightness of light
lightoffsetx, lightoffsety  --Light offset relative to center of the sprite
lightcolor                  --Color constant or hex color specifying the light's colour
lightflicker                --If true, the light source flickers
```

TXT Files for Effects

The effects system currently used by new effects is exposed to the user, and it allows customisation of effects per effect.txt-file **for IDs above 161.** Effect.txt files are seperated into layers of spawned effects by a single spawner. The section below details all configurable properties of individual effect layers:

```
onInit      --defines the name of an onInit method defined in effectconfig.lua. Executes on spawn of this
layer
onTick      --defines the name of an onTick method defined in effectconfig.lua. Executes every frame of this
              layer
onDeath   --defines the name of an onDeath method defined in effectconfig.lua. Executes when the effect
              layer dies.

import              --Imports an effect template from effectconfig.lua
template          --can be used by a layer to use another layer as a template, reducing duplication.

img                --sprite used by this layer. Defaults based on name of the effect-n.txt file, accepts number
                     (for internal effect image) or path to sprite image
xOffset           --horizontal offset relative to anchor
yOffset           --vertical offset relative to anchor
gravity           --acceleration per frame, defaults to 0
lifetime          --lifetime in frames, defaults to 65
delay             --number of frames of delay before which this layer should be spawned, defaults to 0
frames            --frame count of used image, defaults to 1
framestyle        --framestyle, akin to how NPCs use it, defaults to 0
framespeed        --framespeed, defaults to 8
sound             --a sound to be played when this layer spawns. Can either be a number (for internal sfx) or
                     a filepath to an audio file
priority          --render priority, defaults to BACKGROUND but also accepts FOREGROUND

xAlign, yAlign      --LEFT,TOP,MID,RIGHT,BOTTOM, defaults to MID, determines spawner align relative to xy
                                  coordinate passed to it
spawnBindX, spawnBindY   --defaults to LEFT,TOP, determines spawned effect alignment relative to xy
                                   coordinate of spawner

speedX, speedY         --initial velocity of effect
maxSpeedX, maxSpeedY     --speed cap of this layer. Default: no speed cap (-1)

opacity           --translucency, defaults to opaque (1)
direction          --facing direction, defaults to -1 and only matters in conjunction with framestyle
npcID             --multipurpose field emulating vanilla effect's npcID field, defaults to 0
angle             --initial angle of effect, defaults to 0
rotation          --rotation speed of effect, defaults to 0

variants          --number of variants this effect has. Variants multiply with frame count to result in total
                     number of frames on the spritesheet
variant           --determines which subset of frames (variant) to use. Defaults to 0 (0-indexed)
```

The result may look something like this:

Recreation of the vanilla brick block effect:

```
[first]
onTick=TICK_ARC
gravity=0.6
speedY=-6
speedX=-2
framespeed=3
frames=4
lifetime=500
maxSpeedY=10

[2]
template=first
speedX=2
img=1

[3]
template=first
speedY=-9
img=1

[4]
template=first
speedY=-10
speedX=2
sound=4
img=1
```

Recreation of the baby yoshi effect:

```
[1]
import=AI_BABYYOSHI
```

The AI_BABYYOSHI is defined in effectconfig.lua:

```
cfg.defaults.AI_BABYYOSHI = {
    framespeed = 10,
    onInit = "INIT_BABYYOSHI",
    onDeath = "DEATH_SPAWNNPCID",
    frames=2,
    variants=8
}
```

NPC Codes

There are new additions to NPC codes in SMBX2 Beta 4. While some are globally available, others define specific behaviour for specific NPCs.

All available NPC codes can be viewed in the document below. The buttons at the bottom of the linked document can be used to navigate between different lists of NPC codes:
SMBX2 NPC Codes

The 751-1000 Range

We have added a reserved range of IDs for various objects which are reserved entirely for people to customize on a per-level or per-episode basis. The ranges are as follows:

- Effects 751-1000
- Blocks 751-1000
- BGOs 751-1000
- NPCs 751-1000

Once used, slots in these ranges will behave just like any other item of that type.
For Blocks, BGOs and NPCs it's important to remember that their editor appearance of the object is defined by an ini file. Examples of such ini files can be found in the appropriate subdirectory of "SMBX\data\PGE\configs\SMBX2-Integration\items". You can copy any of these as a template for your own custom object into your level or episode folder in order to get started. Please refrain from editing the example files directly, as doing so will mess up your installation.
After the object is set up, press **F8** in the editor to reload the level. Your new object can now be found through the Tileset Itembox's Tileset Editor.

Unused objects outside of those ranges are subject to be used by us (the developers) in future releases, and should be left as-is for the purposes of keeping your levels/episodes compatible.

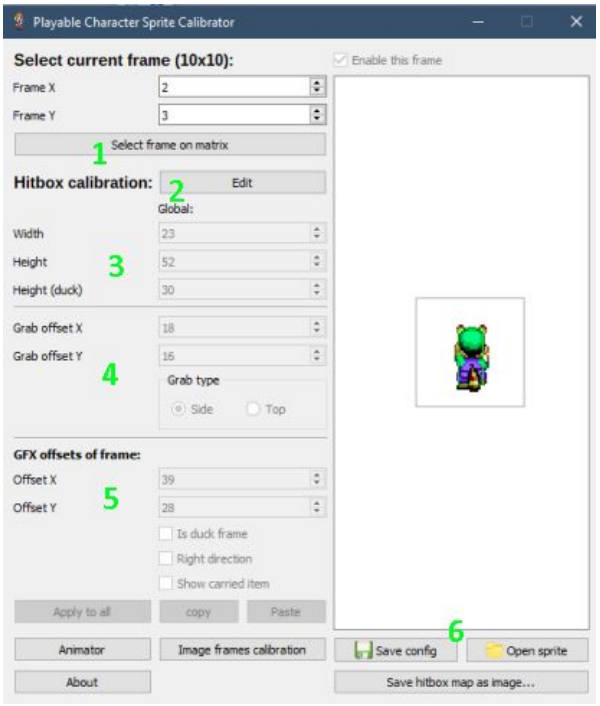
[block/npc]-n.lua Files, Packs and Duplication

When taking a look in the "SMBX/data/scripts/npcs" and "scripts/blocks" folders, you might notice that files are numbered like their respective graphics files are. If you copy one of those files (take the Thwimp, NPC 301, for example) into your level folder and rename it to npc-751.lua, your local NPC 751 will act exactly like a Thwimp would, without any additional lua code! This modular system makes it incredibly easy for people to duplicate basegame NPC and block behaviour, but also makes drag-and-drop NPC and Block plugin packs very easy to create, only requiring .lua, .png and .ini files for each object that's part of the pack! Watch a tutorial on it here! [Part 1] [Part 2]

Player Offsets

Previously, in SMBX 1.3, player sprites were always anchored to the top left of a 100x100 pixel grid cell. This limitation made it difficult to work with character sprites of different dimensions, as player sprites would frequently dip into the ground from the lack of vertical grid space.

SMBX2's editor provides a tool that helps address this. The tool is located in data/pge/pge_calibrator.exe and opens up this GUI:

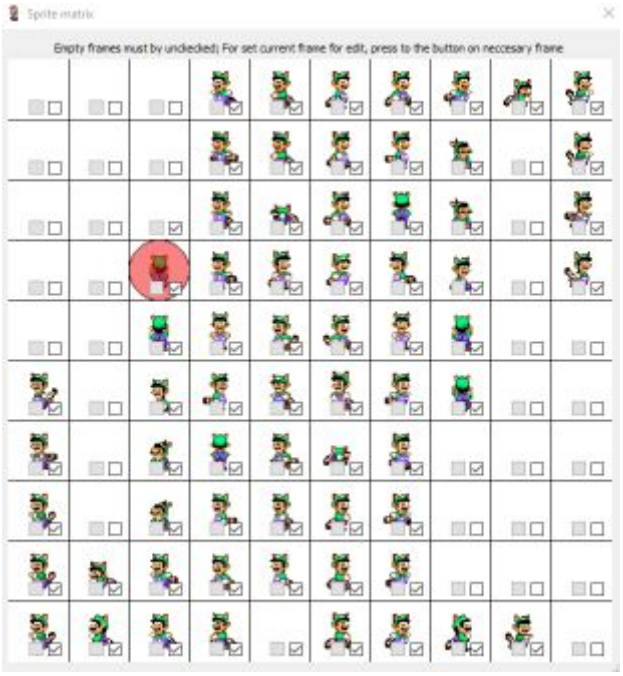


First, load a spritesheet into the calibrator using the "open sprite" button in the lower right. Once a player spritesheet is loaded, a cell from the sheet is displayed on the right. As you can see here, Luigi is already positioned much more centered on the grid than Mario from earlier. The position on the grid is something you can help visualise for yourself using this overlay. Good practice is to make sure that where you want your hitbox to be is in the same position for every cell. It saves a lot of work later on!

The general workflow with this program is as follows:

1. Select a frame on the character matrix. This will open an overlay where you can see all frames on the sheet. Make sure to check the checkboxes next to all sprites you

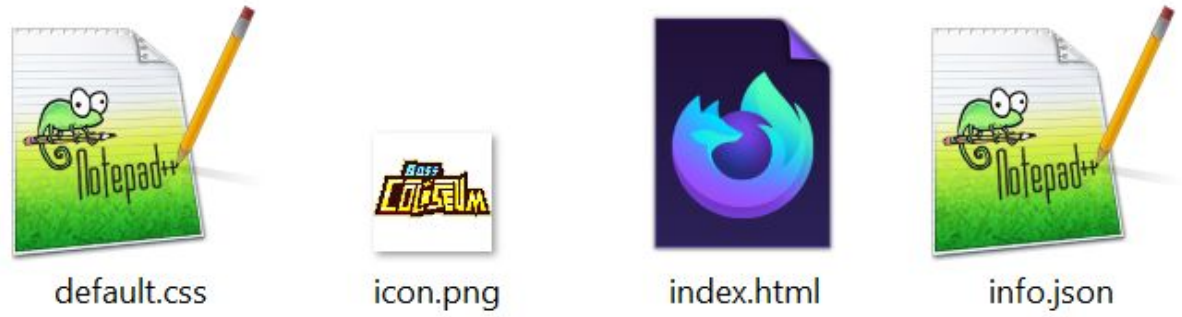
want to use, and select one to begin working



2. Once you have your first sprite selected, hit the “Edit” button to enable hitbox modification for your entire spritesheet.
3. Now it’s time to configure the global defaults for width, height and height while ducking. All units are measured in pixels. Focus for now on the character dimensions. It helps to jump ahead to “GFX Offsets of Frame” to try and roughly align the hitbox with the player, making sure the green rectangle covers the solid collision box you want. Hint: It’s often a good idea to make the hitbox slightly smaller than the sprite, to give players some leeway. Take notice of how Luigi’s arms and head are sticking out of the green rectangle a bit.
   These dimensions are uniform across the entire spritesheet, so setting them up is a one-time manner.
4. Next up are grab offsets. Unfortunately, it isn’t possible to switch from “top” to “side” type at the moment (it does nothing). You can also not change the offset for characters that use “top” offsets natively (Toad, Peach). You can change the offsets for characters based on Mario or Luigi, however. That’s how Wario’s Wario-Land-Style grab offset was made in SMBX2!
5. This lower section includes values unique to each frame. It is where having a uniform offset on the spritesheet really helps out. Because if your sprites are aligned well, you are able to use the same offset values for each frame. “Is duck frame” and “right direction” should also be checked where appropriate. Their names explain their function pretty well. The third checkbox doesn’t affect anything. Repeat this section for every sprite you use on the sheet.
6. Now you can save the spritesheet with the button on the lower right! This will save a .ini file named after the spritesheet you edited. If this file is in your level or episode directoy, SMBX2 will automatically load it for the character and powerup it’s named after!

Launcher Pages

Since SMBX2 b3, custom launcher pages for your episodes have been possible to build. In the new release, however, these have been greatly expanded.

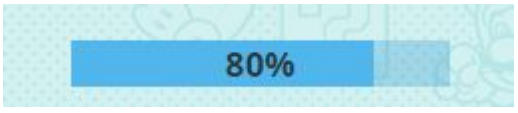


Launcher pages generally consist of a .html file (and optionally a .css file to go with it), an icon, and a file named “info.json”. This documentation will mostly focus on info.json, but will touch on a few other files.

The info.json file is the core of the launcher page. It defines a lot of information about your episode. Here is a simple example of an info.json file:

```
{
  "mainPage": "index.html",
  "Title": "My Super Cool Episode",
  "allowPlayerSelection": false,
  "allowTwoPlayer": false,
  "episodeIcon": "icon.png",
  "progressDisplay": "percent"
}
```

This sets the episode name, determines the .html file to use, disables selecting a character from the launcher (as well as disabling 2 player mode), sets the episode icon, and sets the progress to display as a percentage.



Here is a list of some useful fields for the info.json file:

mainPage	--defines the .html file to use for the launcher page
episodeIcon	--defines the file to use for the icon (should be 64x64 for best results)
title	--defines the name of the episode
allowedCharacters	--a list of character IDs to allow from the launcher (only supports mario, luigi, toad, peach, and link), e.g. [1,2] for only mario and luigi
characterNames	--a list of names to use for the character selection, e.g. [“Demo”, “Iris”, “Kood”, “Raocow”, “Sheath”]
current-version	--a list of version numbers, ranging from major to minor, e.g. [1, 0, 0]
allowTwoPlayer	--set to false to disable character selection for player 2
allowPlayerSelection	--set to false to disable character selection entirely
allowSaveSelection	--set to false to disable the ability to choose a save slot (shouldn’t be commonly used)
starIcon	--set to a 16x16 image file to use it in place of the stars icon on the launcher
collectibles	--set to a string that determines the name of the collectible for this episode
collectible	--an optional string to allow for a singular variant of the name of the episode collectible
maxProgress	--if set to a number, this will be used instead of the total star count for measuring progress. Progress will be measured against Progress.progress set from Lua
progressDisplay	--set to “percent” to display percentage progress for this episode (rather than using x/y format)
customProgress	--if set to true without maxProgress being set, will force the usage of Progress.progress for measurement, displaying the raw value on the launcher
noAchievementBorders	--set to true to disable the borders around achievement icons for this episode

When constructing your launcher .html page, you can create HTML elements with specific classes that will be automatically populated. For example:

```
<div class=”_episodeTitle”></div>
```

This will automatically create an element that displays your episode title. Here is a list of these automatic elements:

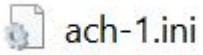
_episodeTitle	--displays the name of the episode
_episodeIcon	--displays the episode icon
_stars	--displays the star icon or name, and the number of stars in the episode (will not display if no stars were found)
_starsIcon	--displays the star icon
_starsCount	--displays the number of stars in the episode (will display 0 if no stars were found)
_starsContainer	--will not display if no stars were found in the episode
_credits	--will display credits from the .wld file (will not display if no credits were found)
_creditsContainer	--will not display if no credits were found in the .wld file

Achievements

Achievements are a new feature that has been added to the game! These one-time collectables will appear in the launcher once you've collected them, and you can design your own achievements for your episodes.

To create an achievement, you must create a folder named "achievements" in your episode folder.

Inside that folder, you can then create "ach-n.ini" files, to define new achievements:



These .ini files are structured as information, followed by lists of conditions. Here is an example .ini file:

```
name="My Super Rad Achievement"
desc="Do the thing!"
condition-1=true
condition-1-desc="Touch fuzzy."
condition-2=true
condition-2-desc="Get dizzy."
```

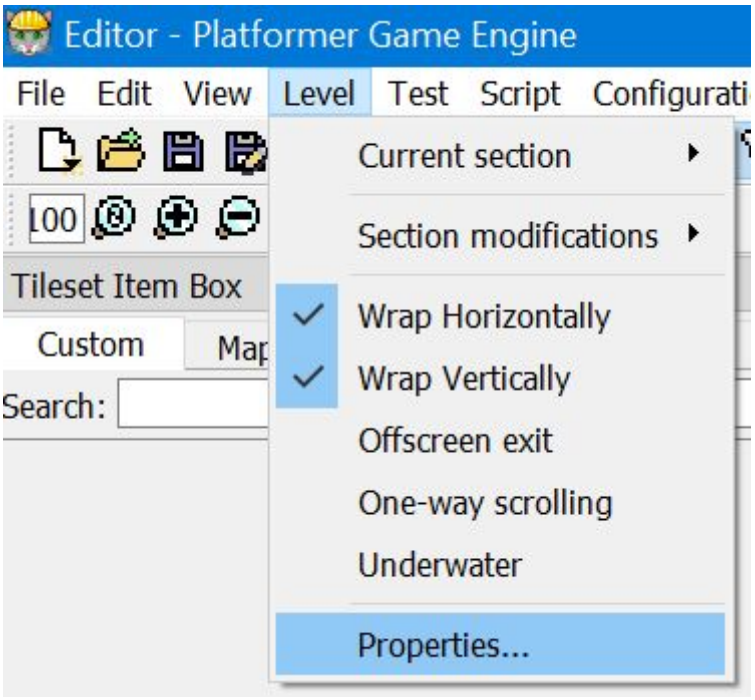
This section shows how to set up their data:

```
name                --defines the name of the achievement
desc                --defines the description of the achievement
collected-desc      --rarely used, but defines a separate description to display when the achievement
                      is unlocked
hidden              --if set to true, the achievement information will not be displayed until it is
                      unlocked

condition-#         --a numbered condition (e.g. condition-1). accepts a range of different values:
                        true          - the condition will be considered cleared when it is
                                        progressed - must be done from Lua
                        "myString"    - the condition will be considered cleared when it is
                                        progressed - progress will occur when the SMBX event with
                                        the given name is triggered
                        #             - any number - the condition will need to be triggered this
                                        number of times before it will be considered cleared - must
                                        be done from Lua

condition-#-desc    --a description of the numbered condition. this must be provided for the
                      condition
                      to appear in the launcher
```

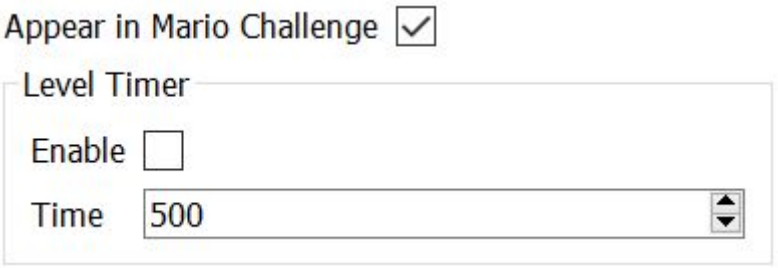
As well as ach-n.ini, you can also provide an ach-n.png. This should be a 64x64 icon for the achievement. You can also opt to provide an ach-nl.png file (e.g. ach-1l.png), which will be used for the icon when the achievement is still locked. Achievements will usually require some Lua to set up, so please see below for more details.

Level Settings

In SMBX2 b4, you'll have access to some new in-editor settings. The first set of these are for the entire level. To find these new settings, select **Level** from the toolbar, and navigate down to **Properties**.



In the window that appears, you will see the following new settings:



Mario Challenge

The first of these new settings is the "Appear in Mario Challenge" setting. Unchecking this box will ensure your level will never appear in the Mario Challenge level roulette. This is useful if your level is a hub, a small bonus level, or might be impossible to complete when thrown in from a Mario Challenge.

Level Timer

The second set of these are the Level Timer settings. This allows you to create a time limit for your levels. Simply check the box, and set how many seconds the timer should last before it runs out (and kills the player!)

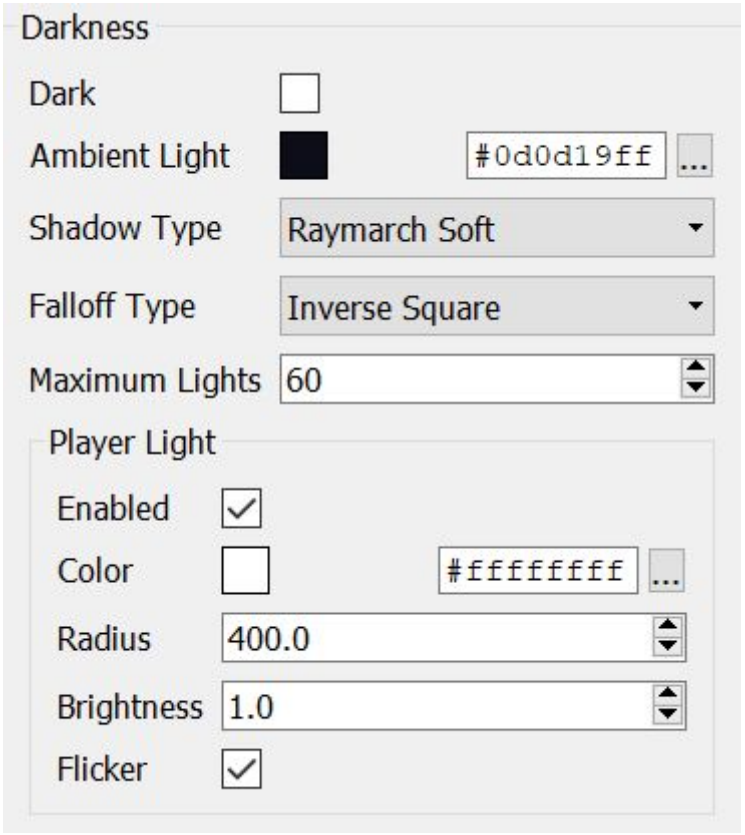
Section Settings

SMBX2 b4 also adds some new functionality to level sections, which allows you to do a lot more without touching Lua. The first new feature available is vertical wrap. Checking this in the section settings will allow players and NPCs to fall off the bottom of the section and reappear at the top (or vice versa).

The rest of the settings require a bit more explanation, and most can be adjusted further with Lua.

Darkness

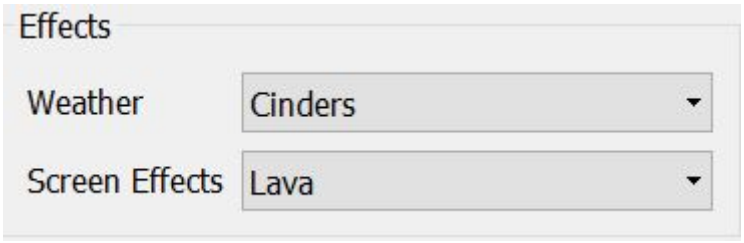
Sections now have settings to allow you to make them dark. Simply checking the “Dark” checkbox in these settings will enable this feature, and relevant blocks, BGOs and NPCs will automatically emit light. However, there are a few extra options you can tweak, as shown here:



Ambient Light	This determines the color of “darkness”. White means the darkness will not be visible at all, while black means nothing will be visible unless lit up.
Shadow Type	You can choose what kind of shadows lights should cast in this section, if any. You can choose between no shadows, soft shadows, or hard shadows.
Falloff Type	This determines how light fades into the dark as it leaves the light source. By default this uses a realistic Inverse Square gradient, but there are a number of other options. Feel free to experiment.
Maximum Lights	This sets the maximum number of light sources that can be visible at any one time. You should try to keep this low if you can.
Player Light > Enabled	This enables a light source around the player.
Player Light > Color	Sets the color of the light source around the player. Black means no light at all.
Player Light > Radius	Sets the radius of the light source, in pixels. A radius of 400 will stretch the light across the entire screen, for example.
Player Light > Brightness	Sets the brightness of the light source. 0 means no light at all, 1 means a normal brightness, and more than 1 means a very bright light.
Player Light > Flicker	If this is checked, the light source around the player will flicker slightly.

Effects

Sections can now have rendering effects applied to them. These come in two flavors: weather and screen effects. These can be combined for interesting combinations.



The options for Weather are:

1. **Rain** - A moderate rainfall effect.

2. **Snow** - A moderate snowfall effect.
3. **Fog** - Fog that covers parts of the stage, obscuring the view.
4. **Sandstorm** - Dust and sand particles blow across the stage from right to left.
5. **Cinders** - Small sparks and flame particles drift up from below.
6. **Wisps** - Etherial lights and will-o-wisps drist in the air.

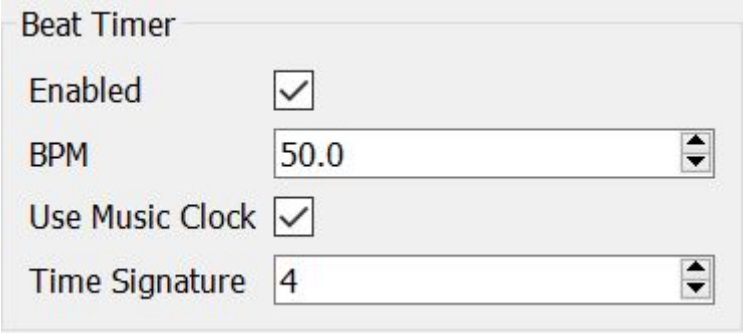
The options for Screen Effects are:

1. **Wavy** - Wobbles the screen slightly, as if in a heat haze or looking through water.
2. **Lava** - Red light, sparks, and smoke drift up from the bottom of the section.
3. **Caustics** - Effects similar to water reflecting light appear on blocks.
4. **Underwater** - A combination of **Wavy** and **Caustics**, helpful for underwater sections.
5. **Mist** - White mist drifts up from the bottom of the section.
6. **Sepia** - Converts the screen to sepia tone.
7. **Grayscale** - Converts the screen to black and white (grayscale).
8. **Inverted** - Inverts the screen (light colors become dark, and vice versa).
9. **Gameboy** - Colors are limited to the gameboy palette and screen resolution reduced.
10. **Dithered Gameboy** - Colors are limited to the gameboy palette and screen resolution reduced. Uses dithering to capture more detail.

These effects can also be accessed through Lua. See Added functionality to basegame classes below for more information.

Beat Timer

SMBX2 b4 adds Blinking Blocks and Timed Spike Blocks. These both use a global timer call the “beat” to synchronize their behaviour. The beat can be adjusted in the section settings, or manually through Lua (See Added functionality to basegame classes below for more information)..



If the beat timer is enabled, then the section settings will be used any time this section is entered, otherwise they will be ignored.

BPM	This determines the speed of the beat timer. This is measured in Beats Per Minute, meaning the higher this number, the faster the beat. A BPM of 60 means one beat every second.
Use Music Clock	Usually, you’ll want to sync the beat to the music. If you are doing that, you should check this box, as it ensures the beat stays in sync even if the game lags.

Time Signature	This determines how many beats are in a bar. The blocks using this timer in basegame all change state after one bar, so increasing this means increasing the time between state changes. Try to match this with the time signature of your music.

LunaLua

Automatically accessible features

These features can be used without the need to load any external libraries. Some are more complicated to use than others, but all are helpful to keep in mind.

Added functionality to basegame classes

Certain basegame namespaces have been augmented with new functionality. Below is a list of all features that have been added.

```
Section.getActiveIndices() --Returns all sections in which players currently are
Section.getWeatherEffect(id) --Gets a particle emitter matching one of the weather effect constants
                                (WEATHER_RAIN, WEATHER_SNOW, WEATHER_FOG,
WEATHER_SANDSTORM,
                                WEATHER_CINDERS, WEATHER_WISPS)
Section:drawScreenEffect(id, camera) --Draws a screen effect using one of the screen effect constants
                                        (SEFFECT_WAVY, SEFFECT_LAVA, SEFFECT_CAUSTICS,
                                        SEFFECT_UNDERWATER, SEFFECT_MIST, SEFFECT_SEPIA,
                                        SEFFECT_GRAYSCALE, SEFFECT_INVERTED, SEFFECT_GAMEBOY,
                                        SEFFECT_DITHERED_GAMEBOY)

Player:render{args} --Renders the player. The function has a lot of customisation options, documented
                        here: https://pastebin.com/dfGz4ZDa
Player.getNearest(x,y) --Returns closest player [[LEVEL ONLY]]
Player:transform(id, shouldSpawnEffect?) --Transforms the player into the character of the ID provided,
with
                                            a smoke cloud effect if desired.
Player.setCostume(id, costumeName, volatile?) --Sets the costume for the character of the ID provided to
                                                the costume with the corresponding name. CostumeName
                                                has to match the folder name of the costume in the
                                                costumes folder. For example, “SMW-Mario”. If volatile is
                                                set to true, the costume state isn’t saved.
Player.getCostume(id) --Returns the name of the costume currently equipped by the character with the
                        given ID

Text.getSize(string text, int fonttype) --Returns width, height of the string specified if rendered with
                                            Text.print with the corresponding font type
```

```
Misc.episodePath() --Gets total path to the episode
Misc.inEditor() --Are we playing in the editor?
Misc.isPaused() --Is the game paused either by vanilla or lua?
Misc.dialog(a) --Shows a dialog box, displaying a. Can accept multiple arguments. Intended for debugging.
Misc.givePoints(index, position, supressSound?) --Provides score or lives relative to the internal score
                                                  indexing, renders the effect at the given position and plays
                                                  a sound if desired
Misc.setBPM(b) --Sets the level's BPM, used for blinking blocks and timed spike blocks
Misc.getBPM() --Gets the level's BPM, used for blinking blocks and timed spike blocks
Misc.setBeatTime(s) --Sets the time between beats, used for blinking blocks and timed spike blocks
                      This also adjusts the level's BPM
Misc.getBeatTime() --Gets the time between beats, used for blinking blocks and timed spike blocks
Misc.setBeatSignature(s) --Sets the number of beats in a bar, used for blinking blocks and timed spike blocks
Misc.getBeatSignature() --Gets the number of beats in a bar, used for blinking blocks and timed spike blocks
Misc.beatUsesMusicClock --If set to true, the beat will be counted to keep sync with the music
Misc.getBeatClock() --Gets the current beat tick counter, used for blinking blocks and timed spike blocks

Level.load(filename, episodename, warpindex) --Loads the specified level [[LEVEL ONLY]]
Level.folderPath() --Returns the path to the current level folder [[LEVEL ONLY]]

Layer.isPaused() --Check if layers are currently paused [[LEVEL ONLY]]

--Transforms an NPC. Centered defaults to true and binds to bottom/center for NPCs with gravity to prevent
clipping. changeSpawn defaults to false, and if changed will change the spawn ID of the NPC to newID
[[LEVEL ONLY]]
NPC:transform(newID, centered, changeSpawn)


Graphics.drawBox{args} --Helper function for glDraw which takes x,y,width,height rather than a vertex table
Graphics.drawScreen{args} --Helper function for glDraw which draws over the entire screen
Graphics.drawLine{args} --Helper function for glDraw which draws a line between x1,y1 and x2,y2
Graphics.drawCircle{args} --Helper function for glDraw which draws a circle with x,y and radius

table.ifindlast(t, value) --Finds last instance of value in t
table.findlast(t, value)  --Works for tables with gaps in their indexing
table.ifind(t, value) --Finds first instance of value in t
table.find(t, value)  --Works for tables with gaps in their indexing
table.ifindall(t, value) --Finds all instances of value in t
table.findall(t, value)  --Works for tables with gaps in their indexing
```

```
table.icontains(t, value) --Returns whether t has value
table.contains(t, value)  --Works for tables with gaps in their indexing
table.iclone(t) --Performs a shallow clone of t
table.clone(t)  --Works for tables with gaps in their indexing
table.ideepclone(t) --Performs a deep clone. Performance-intensive!
table.deepclone(t)  --Works for tables with gaps in their indexing
table.ishuffle(t) --Shuffles a table with consecutive numeric indices
table.map(t) --Returns a lookup map of t
table.unmap(t) --Reserve operation of table.map(t)
table.join(...) --Arguments are joined in reverse order, returns result
table.append(...) --Values of arguments are appended in order, returns result
table.reverse(t) --Reverses a table
table.flatten(t) --Flattens an array of vectors, matrices, colors, or similar contents, turning it to a numerically
                    ordered table without subtables. Useful for uniforms and attributes in shaders, which
                    require flattened arrays as their input.

string.trim(s) --Trims trailing and leading whitespace
string.split(s, pattern, exclude?, plain?)  --Split a string on a pattern into a table of strings
string.compare(left, right) --Compares two strings by lexical ordering and length, returning -1 or 1
depending
                                on whether the left or right string is lexically or literally smaller. Returns 0 when
                                the strings are identical.

math.lerp(a,b,t) --Interpolates from a to b given time value t (between 0 and 1)
math.anglelerp(a,b,t) --Interpolates between 0 and 360, wrapping around the ends
math.invlerp(a,b,v) --Performs an inverse lerp
math.clamp(a,min,max) --Clamps a between the limits
math.sign(a) -- Returns the signum of a
```

Advanced Sprite Drawing

The Sprite namespace makes complicated draw calls easy. While for simple draw calls the Graphics.drawImage family of functions will suffice, you will be unable to handle scaling, rotation and tints without an alternative method.

The Sprite namespace has extended documentation in its source file which can be found in "SMBX\data\scripts\base\Sprite.lua". Its general workflow can be split into three parts though:

1. Creating a Sprite object
2. Manipulating a Sprite object
3. Drawing the Sprite object

Here is an example:

```
--Creation
local image = Sprite{x = 0, y = 0, frames = 10, texture = someImage}

function onDraw()
    image:rotate(1)
    image:draw{frame = 1}
end
```

Collision detection

Using functions of the Colliders namespace, collision detection can be done between Blocks, NPCs, Players, and user-defined collider objects. You can find documentation for the library on the following page, though it is somewhat outdated. These days, Colliders is automatically active, and you do not need to explicitly load the library in order to use it: http://wohlsoft.ru/pgewiki/Colliders.lua

Example:

```
--Creation
local box = Colliders.Box(-200000, -200600, 800, 600)

function onTick()
    --Collision Check
    if not Colliders.collide(player, box) then
        Text.print("The player has left the first screen", 100, 100)
    end
end
```

If you find yourself needing to check collisions between lots of objects, for example, to see if a collider is touching an NPC, you should consider using Colliders.getColliding. This will give you a list of objects that satisfy your conditions. You can also ask for condition pairs (for example, if you need to find which NPCs of a given ID are touching certain blocks of a different ID), which will give you a list of colliding pairs. You can optionally supply a "filter"

function, that will test every object in your provided list. You can use this if you need to fulfil certain conditions such as memory locations.

Example:

```
--Creation
local box = Colliders.Box(-200000, -200600, 800, 600)

function onTick()
    --Collision Check
    local list = Colliders.getColliding{
                                        a = box,
                                        b = NPC.HITTABLE,
                                        btype = Colliders.NPC
                                       }
    for k,v in ipairs(list) do
        Text.print("Touching NPC at "..v.x.."", "..v.y, 100, k*20)
    end
end
```

Colors

This version of SMBX2 introduces a way to define and use colors. Using the Color namespace you can access existing colors, define your own, perform math on these colours and lerp between them. Predefined color constants and their corresponding hexadecimal values can be found below:

```
Color.white          --0xFFFFFFFF
Color.black          --0x000000FF
Color.red            --0xFF0000FF
Color.green          --0x00FF00FF
Color.blue           --0x0000FFFF
Color.alphawhite     --0xFFFFFF00
Color.alphablack     --0x00000000
Color.transparent    --0x00000000
Color.grey           --0x808080FF
Color.gray           --0x808080FF
Color.cyan           --0x00FFFFFF
Color.magenta        --0xFF00FFFF
Color.yellow         --0xFFFF00FF
Color.pink           --0xFF73ABFF
Color.canary         --0xFFF266FF
```

```
Color.purple          --0xAB66ABFF
Color.orange          --0xFF8C54FF
Color.teal            --0x00AB99FF
Color.maroon          --0x730000FF
Color.brown           --0x804D00FF
Color.lightgrey       --0xBFBFBFFF
Color.lightgray       --0xBFBFBFFF
Color.lightblue       --0x33CCFFFF
Color.lightgreen      --0x80CC99FF
Color.lightbrown      --0xBF9966FF
Color.lightred        --0xFF8080FF
Color.darkgrey        --0x404040FF
Color.darkgray        --0x404040FF
Color.darkblue        --0x003373FF
Color.darkgreen       --0x005926FF
Color.darkbrown       --0x4D4040FF
Color.darkred         --0x800000FF
```

Usage examples:

```
--Adjustment of color alpha through concatenation:
local alphared = Color.red .. 0

--Definitions of a custom color
local myColor = Color(1,1,1)
local myColor = Color.fromHex(0x112233FF)
local myColor = Color.fromHexRGB(0x112233)

--While lerp lerps between RGB values
local newColor = math.lerp(Color.red, Color.blue, timer)
--lerpHSV uses the HSV values for its lerp
local newColor = Color.lerpHSV(Color.red, Color.blue, timer)
```

Extended documentation on the Color class can be found in “SMBX\data\scripts\base\engine\color.lua”

Liquid Class

Liquids such as Water and (don’t ask) Quicksand are now accessible from code. While in most situations simply checking for an entity’s “underwater” state can be enough, there are some scenarios where knowing and manipulating the exact bounds of liquid boxes can be useful. The class is equipped with the same functions other base object classes have:

```
Methods:
Liquid.get() -- returns all liquids in the level
Liquid.getIntersecting(x1, y1, x2, y2) -- returns all liquids in area
Liquid.count() -- returns number of liquids in the level

Fields:
myLiquid.idx -- index in Liquid array
myLiquid.isValid -- validity check
myLiquid.layer -- Layer object
myLiquid.layerName -- name of layer object
myLiquid.isHidden -- visibility flag
myLiquid.isQuicksand -- if true, this liquid is quicksand
myLiquid.x
myLiquid.y
myLiquid.width
myLiquid.height
myLiquid.speedX
myLiquid.speedY
```

Save Data

There are two exposed tables for save data storage accessible at all times:

- SaveData - Saves data across sessions
- GameData - Saves data for this session only

By writing to these tables, you can store information that persists past a level reload. This can be used for various effects, like checking whether or not to repeat a cutscene the player has already seen, or saving one-time collectibles. You can simply define new variables to keep track of by storing them as part of these tables:

```
SaveData.hasBeatenLevel = false
GameData.skipCutscene = false
```

SaveData and GameData's tables are **global for an episode**. If you are working on a level that is for a contest or collaboration, and the event in question doesn't specify its own save data handling, please handle your save data as follows in order to avoid any conflicts:

```
--Initialisation of your own save data
SaveData[Level.filename()] = SaveData[Level.filename()] or {}
GameData[Level.filename()] = GameData[Level.filename()] or {}

local saveData = SaveData[Level.filename()]
local gameData = GameData[Level.filename()]
```

You can now use the saveData and gameData variables just like you would use SaveData and GameData, with the added benefit of avoiding any possible conflicts with save data from other levels in the collab or contest.

If you really need to, SaveData and GameData both have special functions you can use for some extra functionality, though usually these won't be necessary:

```
SaveData.flush() --Forceibly write your SaveData to the save file.
                   Normally, this will be automatically done when the
                   Game is saved

SaveData.clear() --Wipes everything from SaveData. Won't be saved until
                   the game is saved (or SaveData.flush() is called).
GameData.clear() --Wipes everything from GameData.
```

Sound and Music

You can use the following function to play music:

```
Audio.MusicChange((number) section, (string) filename or (number) musicID)
```

To stop music entirely, you can set the music to ID 0 - silence.

You can also use the Audio namespace's functions to control music, though using MusicChange is usually simpler:
https:--wohlsoft.ru/pgewiki/LunaLua_global_Sound_and_Music_functions
Sound effects have been streamlined. You can use the versatile SFX.play function to cover your needs for sound effects:

```
--You can use the set overload:
SFX.play((number) internalindex or (string) filename, volume, loops,
delay)
--or named arguments:
SFX.play{args}

--Available arguments:
sound       --REQUIRED: The sound file path
loops       --The number of loops for this sound to play for. 0 to loop forever. Defaults to 1.
volume      --The volume of this audio clip, between 0 and 1. Defaults to 1.
pan         --The left/right panning of this audio clip, between -1 and 1. Defaults to 0.
tags        --List of tags for this sound clip. Allows volume to be adjusted for every sound with a given tag.
tag         --Single tag for this sound clip. Allows volume to be adjusted for every sound with a given tag.
delay       --The number of ticks before the same sound can be played again. Defaults to 4.
```

Examples:

```
SFX.play(4)
SFX.play(“mySound.ogg”, 0.5)
SFX.play(37, 1, 3, 18)
SFX.play{sound=“mySound.ogg”}
```

In addition to SFX.play, you are able to use SFX.create, which creates a physical audio source in the scene and is useful for area-specific sound effects like the rushing of a waterfall.

```
SFX.create{args}

--Available arguments:
--x,y              --REQUIRED: Position of the centre of the audio source.
--falloffRadius    --REQUIRED: Distance the sound travels from the source before it is silent.
--sound            --The sound file path
--falloffType      --The falloff function to use. Supports FALLOFF_NONE, FALLOFF_LINEAR and
                     FALLOFF_SQUARE. Can also use a custom function of the form
                     'falloff(squaredFalloff, squaredDistance)'. Defaults to FALLOFF_SQUARE.
--type             --Shape of the audio source (to emit at max volume). Supports SOURCE_POINT,
                     SOURCE_CIRCLE, SOURCE_BOX and SOURCE_LINE. Defaults to SOURCE_POINT.
--play             --Should the sound play immediately? Defaults to true.
--loops            --The number of loops for this sound to play for. 0 to loop forever. Defaults to 0.
--volume           --The volume of this audio source, between 0 and 1. Defaults to 1.
--parent           --Object to attach the source to. Defaults to nil.
--tags             --List of tags for this sound source. Allows volume to be adjusted for every sound with a
                     given tag.
--tag              --Single tag for this sound source. Allows volume to be adjusted for every sound with a
                     given tag.

----SOURCE_CIRCLE only
--sourceRadius         --REQUIRED: The radius of the audio source.

----SOURCE_BOX only
--sourceWidth/sourceHeight  --REQUIRED: The dimensions of the audio source.

----SOURCE_LINE only
--sourceVector         --REQUIRED: The vector describing the line. Line will span from {x,y} to
                         {x,y}+sourceVector.
```

Vector Math

Using the vector namespace, you are able to use vector mathematics in two, three and four dimensions. Its usage has changed minimally since Beta 3 and the library is now

automatically loaded into the vector namespace. **Note! The namespace “vector” is not capitalised, unlike the rest of the namespaces in this section!**
Vector documentation

```
--To create an n-dimensional vector, you can use the following
constructor. Depending on the number of arguments you provide, you will
get a vector in a different dimension.
local myVector = vector(x,y,z,w)

--Examples
local myVector = vector(12, 3, 4, 8) --4D Vector
local myVector = vector(2, 8, 6) --3D Vector
local myVector = vector(3, 15) --2D Vector
local myVector = vector(5) --Shorthand for vector(5,5)
local myVector = vector() --Shorthand for vector(0,0)

local myVector = vector.zero2 --Shorthand for vector(0,0)
local myVector = vector.right2 --Shorthand for vector(1,0)
local myVector = vector.up2 --Shorthand for vector(0,1)
local myVector = vector.one2 --Shorthand for vector(1,1)


--Some basic operators, fields, and functions exist for vectors:
myVector.length  --Length of the vector
myVector.sqrlength  --Squared length of the vector

local newVector = myNumber * myVector --Scalar multiplication
local newVector = myVector / myNumber --Scalar division
local newVector = myVectorA + myVectorB --Vector addition
local newVector = myVectorA - myVectorB --Vector subtraction
local newVector = myVectorA .. myVectorB --Dot product
local newVector = myVectorA ^ myVectorB --Cross product (3D only)
local newVector = myVectorA * myVectorB --Piecewise multiplication
local newVector = myVectorA / myVectorB --Piecewise division
local newVector = myVectorA % myVectorB --Project A onto B

local newVector = myVector:rotate(degrees) --2D rotation
local newVector = myVector:rotate(degrees, axis) --3D rotation
local newVector = myVector:normalize() --Set vector length to 1
local newVector = myVector:lookat(position) --Look towards vector
local newVector = myVector:planeproject(normal) --3D planar projection

local newVector = myVector:tov2() --Convert to 2D (3D and 4D only)
local newVector = myVector:tov3() --Convert to 3D (2D and 4D only)
local newVector = myVector:tov4() --Convert to 4D (2D and 3D only)
```

```
--You can construct 2x2, 3x3, or 4x4 square matrices, and use them for
transformations by multiplying them with Vectors.
local myMatrix = vector.mat2(m11, m12, m21, m22)

--Examples:
local myMatrix = vector.mat2(1,2,3,4) --Matrix   1  3
                                                 2  4
local myMatrix = vector.mat3(1,2,3,4,5,6,7,8,9) --Matrix   1  4  7
                                                            2  5  8
                                                            3  6  9
local myMatrix = vector.id2 --2D identity matrix (vector.mat2(1,0,0,1))
local myMatrix = vector.empty2 --Empty matrix (vector.mat2(0,0,0,0))
local myMatrix = vector.id3 --3D identity matrix
local myMatrix = vector.empty3 --Empty matrix


--Some basic operators, fields, and functions exist for matrices:
myMatrix.inverse --Inverse matrix (returns nil if matrix has no inverse)
myMatrix.det --Matrix determinant
myMatrix.trace --Matrix trace
myMatrix.transpose --Transposed matrix

local newMatrix = myNumber * myMatrix --Scalar multiplication
local newMatrix = myMatrix / myNumber --Scalar division
local newMatrix = myMatrixA + myMatrixB --Matrix addition
local newMatrix = myMatrixA - myMatrixB --Matrix subtraction
local newMatrix = myMatrixA * myMatrixB --Matrix multiplication
local newVector = myMatrix * myVector --Vector-Matrix multiplication


--You are also able to construct quaternions and use them for rotation
of 3D Vectors
local myQuat = vector.quaternion(axis, degrees)

--Some basic operators, fields, and functions exist for quaternions:
myQuat.inverse --Quaternion inverse
myQuat.norm --Quaternion norm
myQuat.sqrnorm --Quaternion squared norm
myQuat.euler --Vector of euler angles
myQuat.normalised --Normalised quaternion

local newQuat = myNumber * myQuat --Scalar multiplication
local newQuat = myQuat / myNumber --Scalar division
```

```
local newQuat = myQuatA + myQuatB --Quaternion addition
local newQuat = myQuatA - myQuatB --Quaternion subtraction
local newQuat = myQuatA * myQuatB --Quaternion composition
local newVector = myQuatA * myVector --Quaternion application
local newQuat = myQuatA .. myQuatB --Quaternion dot product

local newQuat = myQuat:normalize() --Quaternion normalization
local newQuat = myQuat:lookTo(myVector, (upVector)) --Quaternion look at
local newQuat = myQuatA:rotateTo(myQuatB, (speed)) --Quaternion rotate
local newMatrix = myQuat:tomat() --Convert to 3x3 rotation matrix
local newMatrix = myQuat:tomat4() --Convert to 4x4 rotation matrix


--There are also a few general helper functions:
vector.randomDir2() --Random 2D vector with length 1
vector.randomOnCircle(radius) --Random 2D vector on the edge of a circle
vector.randomInCircle(radius) --Random 2D vector inside a circle

vector.randomDir3() --Random 3D vector with length 1
vector.randomOnSphere(radius) --Random 3D vector on the edge of a sphere
vector.randomInSphere(radius) --Random 3D vector inside a sphere
```

For more information on vectors, matrices, quaternions, and the available functions for them, please refer to the library file in scripts\base\vectr.lua directly.

Achievements

The Achievement namespace allows you to access to adjust achievements. There is not a lot that needs to be done with this other than checking and setting conditions, so this guide will just cover that.

To get a specific achievement, you can use the Achievement constructor:

```
local myAchievement = Achievements(1)
```

From that, you can access the list of conditions, and either check their values, or progress them. For example:

```
Misc.dialog(myAchievement:getCondition(1).value)
myAchievement:progressCondition(1)
Misc.dialog(myAchievement:getCondition(1).value)
```

This code will first display a dialog box showing the current state of the first condition in your achievement. It then progresses that condition, and shows the same dialog box again with the updated condition. Here is the full documentation on these functions:

```
Achievements.get() -- returns all achievements in the episode
Achievement:getCondition(id) -- returns the given condition object (use
                                Condition.value to access the state)
Achievement:progressCondition(id, delay?) -- progresses the condition.
                                             If delay is set to true,
                                             the achievement popup will
                                             not display until the next
                                             loading screen has finished
Achievement:setCondition(id, delay?) -- sets the value of the condition.
                                        If delay is set to true, the
                                        achievement popup will not
                                        display until the next loading
                                        screen has finished
Achievement:resetCondition(id) -- resets the condition so it must be
                                  progressed again
```

Checkpoints

The Checkpoint namespace allows you to create checkpoints from Lua, which are a little more versatile than the editor-based ones. Note that you *must* create any checkpoints outside onStart, because they have to exist right at the start of the level.

```
local myCheckpoint = Checkpoint{section = 0, x = -20000, y = -20000}
```

Checkpoint objects must contains a section, x, and y field, which determine where the player will spawn. Other optional fields are:

```
powerup --Powers up the player to this powerup (e.g. PLAYER_BIG for
          default checkpoint behaviour)
sound --A sound file or ID to play when the checkpoint is collected
        (e.g. 58 for default checkpoint behaviour)
actions --A function that will be run when the player spawns into this
          checkpoint. The function will be run once for each player, and
          will pass the player as an argument.
```

Once you have a checkpoint object, it won't appear in the level, but will exist in Lua and you can use Lua to collect the checkpoint:

```
myCheckpoint:collect(player) --Collects the checkpoint, can optionally
                               specify which player collected it.
myCheckpoint:reset() --Allows the checkpoint to be collected again.
```

Often, it is useful to be able to get the currently active checkpoint. There are a few general functions that can help with things like this:

```
Checkpoint.getActive() --Gets the currently active checkpoint (nil if no
                       checkpoint is active)
Checkpoint.get() --Gets a list of all checkpoints
Checkpoint.get(id) --Gets the checkpoint with a specific ID
Checkpoint.reset() --Resets all checkpoints
```

There is also a global event that exists for checkpoints, which is run whenever a checkpoint is collected, and allows you to run code when the player hits a checkpoint:

```
function onCheckpoint(checkpoint, player)
    --your code here
end
```

Explosion Class

Another new addition to the scripting functionality is the Explosion class. You can use the Lua event onExplosion to detect when an explosion happens, and do your own management of it:

```
function onExplosion(eventobj, explosion, player)
    if eventobj.cancelled then return end
    --your code here
end
```

The eventobj argument allows you to cancel the explosion and prevent it from spawning. The explosion argument is the explosion object itself, and the player argument is the player that caused it.

The explosion object allows you to read and manipulate certain things about the explosion:

```
x        --The x coordinate of the centre of the explosion
y        --The y coordinate of the centre of the explosion
radius   --The radius of the explosion
strong   --If true, will destroy grey brick blocks
friendly --If true, will not harm the player who spawned it
id       --The type of explosion this is
collider --A circle collider object representing the explosion hitbox
```

As well as onExplosion, there are a few basic functions that can help work with explosions:

```
Explosion.spawn(x, y, id, player)   --Spawns an explosion
Explosion.get()   --Gets a list of explosion objects (use onExplosion
                    for reacting to explosions)


Explosion.register(radius, effectID, soundEffect, (optional) strong,
(optional) friendly)
--Registers a new explosion type (and returns its ID). Use this if you
want to use custom explosions in your episode or level. You can then use
this ID in Explosion.spawn. E.g.:

local myExplosion = Explosion.register(64, 13, “bang.ogg”)
Explosion.spawn(-20000, -20000, myExplosion, player)
```

By default, there are 5 different kinds of explosions:

```
ID

 0    --Peach bomb
 1    --Unused (reserved ID)
 2    --SMB2 bomb
 3    --SMB3 bomb
 4    --TNT explosion
 5    --Nitro explosion
```

Using these IDs in Explosion.spawn will change the kind of explosion that is spawned.

Easy-Access Features

Easy-access libraries are fully usable with just a few lines of code.

Autoscrolling

```
local autoscroll = require("autoscroll")
```

Using the autoscroll library, you can create more complicated autoscrolling scenarios than were previously possible in SMBX. While SMBX 1.3 wouldn’t allow for autoscroll past the first section the player spawns in, this library allows you to control when autoscroll starts by yourself:

```
--As soon as the first section is loaded, begin an autoscroll to the
right with a speed of 2.
function onLoadSection0()
    autoscroll.scrollRight(2)
end
```

```
--Further functions such as scrollUp, scrollLeft, scrollDown and
scrollTo are also available.
--ScrollTo's coordinate specifies the bottom left corner of the
destination screen:
function onLoadSection1()
    --Scrolls to the specified X and Y coordinate with a speed of 1.5
    autoscroll.scrollTo(-170000, -185300, 1.5)
end
```

Camera Zones

```
local camlock = require("camlock")
```

Using the camlock library, you can create camera zones. The library is fairly rudimentary in its current stage and doesn't respond flawlessly to overlapping zones, but will allow you to control camera movement more smoothly than you otherwise could with few lines of code:

```
--The following function adds a camera zone with the specified bounds to the
scene. Everything else is handled internally. lerpSpeed determines the speed
at which the camera lerps into the zone once the player is inside of it.
This argument is optional and should be between 0 and 1.
camlock.addZone(x, y, width, height, lerpSpeed)
```

Clear pipes

Clear pipes automatically register a selection of NPCs as candidates for passing through. You can modify these lists as follows:

```
local clearpipe = require("blocks/clearpipe")
--Register Goomba and unregister player fireball
clearpipe.registerNPC(1)
clearpipe.unregisterNPC(13)
```

Lineguides

```
local lineguide = require("lineguide")
```

Lineguide is already used by several NPCs in basegame, but it offers functions to register even more NPCs to its network of rails.

```
--Registers the given IDs to lineguides.
lineguide.registerNpcs(ID or table of IDs)
```

Orbits

```
local orbits = require("orbits")
```

Orbits allow you to create a set of NPCs moving in a circle. The library has detailed documentation on how to use it in its own orbits.lua file.

Switch Colors

```
local switchcolors = require("switchcolors")
```

Switchcolors manages configurations and signals for different colours of switches, including custom ones!

```
--Registers a new switch colour. Returns an activator for it, as well as
its numeric key for further checking.
local activator, myColor = switchcolors.registerColor(string name)

--Calling activator will toggle a switch
activator()

--You can use the switchcolors.onSwitch event to catch switch toggles
function switchcolors.onSwitch(color)
    if color == myColor then
        --do stuff specific to your switch colour
        --here you can use switchcolors.switch to switch blocks of 2
specific IDs
        switchcolors.switch(id1, id2)
    end
end

--The same procedure is also available for palace switches
local activator, myColor = switchcolors.registerPalace(string name)

activator()

function switchcolors.onPalaceSwitch(color)
    if color == myColor then
        --do stuff
    end
end
```

Timer

```
local timer = require("timer")
```

Timer adds a timer to the level. Its addition to a level also causes Green Berries to execute their effect upon being eaten (adding 10 seconds to the timer).

```
--Activates the timer. Arguments default to 500 and false.
timer.activate(time, isFrames?)
```

```
--Sets the length of a second in frames.
timer.setSecondLength(frames)

timer.set(time, isFrames?)  --Sets the value of the timer.
timer.add(time, isFrames?)  --Adds time to the timer.

--Toggles timer active state if no newValue is specified. Else sets it.
timer.toggle(newValue)
```

Advanced Libraries

Advanced libraries provide extra features but often require more complicated setup and a deeper understanding of lunalua.

Actorclass - Easily Controllable Actors for Cutscenes

Actorclass provides full-fledged controllable actors for complex characters. It relies on animatx2.
The actorclass.lua file includes a detailed documentation of the features of individual actors.

Animatx2 - Extended spritesheet animations

Animatx2 simplifies complex animation scripting by taking care of the complicated math, leaving only the creation of animation objects and their states as required things by the user.
The animatx2.lua file includes a detailed documentation of the library's features.

Click - Mouse Input

```
local click = require("click")
```

Click allows you to use a cursor in your level and use clicks and mouse movement for user-defined interactions.

```
--Initialises all cursors. Each cursor definition is a table formatted
as such: {table of images, xOffset, yOffset, framespeed}. You can leave
the argument table empty in order to rely on default configurations.
click.loadCursor{table of cursor definitions}

--Stores current click state. Can be KEYS_PRESSED, KEYS_DOWN,
KEYS_RELEASED or KEYS_UP
click.state

--Stores speed
click.speedX
click.speedY

--Sets which cursor to display
```

```
click.setCursorID(id)
```

For further functionality, refer to the click.lua lua file.

Routine - Coroutines

The Routine namespace serves as a wrapper for coroutines in lunalua. It provides ways to run functions as coroutines. It does not require loading to use. This works much the same way as the older eventu.lua, and most of the behaviours are unchanged. However, there are a few small differences and new things since the PAL version (aside from not needing to load a library):

```
--Routine.run now lets you assign routines directly to variables.
local myRoutine = Routine.run(myFunc, args)

--Routine.wait allows you to provide a number of seconds to wait for.
Routine.wait(seconds, runWhilePaused?)

--Routine.skip allows you to skip to the next frame.
Routine.skip(runWhilePaused?)

--Routine.loop can be used inside routines to run code while waiting.
--This will wait for “frames”, and run myFunc every tick while waiting.
Routine.loop(frames, myFunc, runWhilePaused?)

--Functions like abort can be called directly from the routine object.
myRoutine:abort()

--The pause and resume functions can pause timed waits, and will always
--prevent a routine from continuing, even if the wait condition is met.
myRoutine:pause()
myRoutine:resume()

--A few fields can be used to check the state of the routine.
---isValid: false if the routine has ended
---paused: true if the routine has been paused with myRoutine:pause()
---yielded: true if the routine was yielded with Routine.yield()
---waiting: true if the routine is currently waiting for anything
myRoutine.isValid
myRoutine.paused
myRoutine.yielded
myRoutine.waiting

--Routine.yield gives you manual control over when to resume a routine.
Routine.yield()
```

```
--myRoutine:continue resumes a yielded routine.
myRoutine:continue()
```

Handycam - Advanced Camera Control

```
local handycam = require("handycam")
```

Handycam makes it easy to perform complex operations on SMBX's cameras.

```
--Cameras are accessed through handycam[idx]
handycam[1]

--Available fields:
x,y
xOffset, yOffset
width, height
rotation
zoom
targets    --List of targets the camera is focusing on

--Available methods
--For a detailed explanation of their arguments and function, please
refer to the handycam.lua library file
handycam[idx]:transition    --Immediately performs a transition
handycam[idx]:queue         --Adds a transition to the end of the queue
handycam[idx]:unqueue       --Removes a transition from the queue
handycam[idx]:clearQueue    --Clears the queue
handycam[idx]:release       --Returns camera control to SMBX
handycam[idx]:worldToScreen  --Converts coord from world to camera space
handycam[idx]:screenToWorld  --Vice-versa
Handycam[idx]:reset         --Resets to default behaviour
Handycam[idx]:finish        --Finishes all transitions immediately
Handycam[idx]:abort         --Aborts all transitions immediately
Handycam[idx]:skip          --Finishes active transitions immediately
```

Particles - Particle Effects and Ribbon Trails

Particles allows you to create and attach particle emitters and ribbons to various objects. Its usage hasn't changed much since Beta 3, and the documentation below is still mostly accurate:
wohlsoft.ru/pgewiki/Particles.lua
However, the overload for Emitter:draw has been extended, with all arguments still being optional:

```
Emitter:draw(priority, disableCulling?, renderTarget, sceneCoords?,
color, timeScale, updateWhilePaused?)
```

Darkness - Darkness and Lighting

While this library is accessible directly from the editor, you can also opt to use it yourself in order to create more complex effects. It does not require loading, as it is automatically loaded.

The darkness library allows you to create dark sections. You may have noticed various light-related configuration flags in the sections of this handbook related to BGO and NPC configuration. Once a darkness zone exists in the level, those take effect.
To add a darkness zone, you do the following:

```
local myDarknessField = Darkness.create{args}
--Args allows the following named arguments:
--falloff        - Determines how light intensity in this field should propagate. Defaults to
                   darkness.falloff.DEFAULT. Paths to shaders can also be supplied.
--shadows        - Determines how shadows should be rendered in this field. Defaults to
                   darkness.shadow.DEFAULT. Paths to shaders can also be supplied.
--maxLights      - Maximum number of lights that can be rendered at any one time. Defaults to 60.
--uniforms       - Table of extra uniforms to supply to the shader (can be used to tweak behaviour).
--priorityType   - Determines how lights should be selected if there are too many to render. Defaults to
                   darkness.priority.DISTANCE.
--bounds         - A rect specifying the boundaries of this field. If left nil, will apply to the entire scene.
--boundBlendLength  - Size of fadeout on the boundaries of this field, if they are used. Defaults to 64.
--section        - Which section this field should apply to. -1 means all sections. Defaults to -1.
--sections `     - Takes precedence over "section". Allows multiple sections to be specified in a table.
--ambient        - The ambient light colour. Defaults to 0x0D0D19.
--priority       - The render priority of this field. Defaults to 0.
```

In addition to relying on the lights attached to the player and certain objects by default, you can create your own lights manually:

```
--Creates a light
local myLight = Darkness.light(x,y,radius,brightness,color)

--Once you have a light, you have to activate it
Darkness.addLight(myLight)
Darkness.removeLight(myLight) --Removal also possible

--If you want a light to only affect specific darkness fields, you can
use the following function
myDarknessField:addLight(myLight)
myDarknessField:removeLight(myLight) --Removal also possible
```

For further functions, please refer to the darkness.lua file directly.

Textplus - Advanced Dialogue Framework

```
local textplus = require("textplus")
```

Textplus is a more robust rework of the old textblox library. It's still WIP, and as such doesn't include all of the old library's features by default. Features such as word bubbles have to manually be built around at this stage, but textplus' internal functions provide various features which neither the vanilla text rendering nor textblox are capable of.
The textplus.lua library contains documentation on its functions.

Shader Programming

With this new release we are introducing GLSL shader programming to lunalua. Shaders are programmed using GLSL Version 1.2, with vertex and fragment parts of a shader split between .vert and .frag files. In the data_templates directory you are able to find standard.vert and standard.frag files, which serve as cloneable templates for your own shader files.
You can then use your own shaders in your lua code like so:

```
--Initialize the shader
local myShader = Shader()

--Compile the shader
function onStart()
    --Vert and Frag default to their standard implementations, so if you
use a standard shader, you can leave that argument as “nil”.
    myShader:compileFromFile(vert, frag)
end

--Shaders can be used with the glDraw drawing function.
function onDraw()
    Graphics.glDraw{
        ...
        shader=myShader,
        --texture is parsed as iChannel0,
        uniforms = {
            --Define uniforms that are used by the shader
        },
        attributes = {
            --Define attributes that are used by the shader (a flattened
              array of data with one object per vertex)
        }
    }
end
```

glDraw Documentation
The GLSL Documentation contains example shaders to learn from

Edge Cases and Keeping Compatibility

This build of SMBX2 may be stable, but the engine is being actively developed, so current problems will be addressed for future releases. As such, certain behaviour is subject to change in later versions, and while these are unlikely to cause problems for everyday level design, when delving into complex interactions they may present issues. Below are the most important points that you need to avoid when designing levels.

Interactions Between Certain New NPCs

We’re still missing some new technology to streamline interactions between NPCs and, as such, various interactions between new NPCs and other new NPCs can be buggy or cause other unexpected behaviour. These behaviours are subject to change as soon as we have the new technology required to do so. Please refrain from exploiting them.