# Installation

Download your preferred version from the Itch.io download page or use GitHub for older releases.

## Windows

Two versions of GB Studio are available for Windows. The *Squirrel Installer* version just requires you to unzip, double click and then wait a few seconds while the application installs to your C:\ drive. Once installed a shortcut will be added to your desktop automatically and the application will start. The application will be installed to `%LocalAppData%\gb_studio`, if you need to install to a different location use the *Manual* version.

The *Manual* version is a zip containing the application files, you can unzip this to any location. Once unzipped double click `gb-studio.exe` to start.

## macOS

For macOS unzip the downloaded file and move `GB Studio.app` to your *Applications* folder. Double click to start.

If you're having trouble building or running your game you may also need to install Apple's Command Line Tools by opening `Applications/Terminal.app` and entering the following command.

```
xcode-select --install
```

## Ubuntu / Debian-based Linux

For Debian-based Linux distros, download the .deb version and run the following commands (Tested on Ubuntu 18.10)

```
> sudo apt-get update
> sudo apt-get install build-essential
> sudo dpkg -i gb-studio_1.0.0_amd64.deb
> gb-studio
```

If you have issues with graphical glitches appearing on Ubuntu try running GB Studio using the following command.

```
> gb-studio --disable-gpu
```

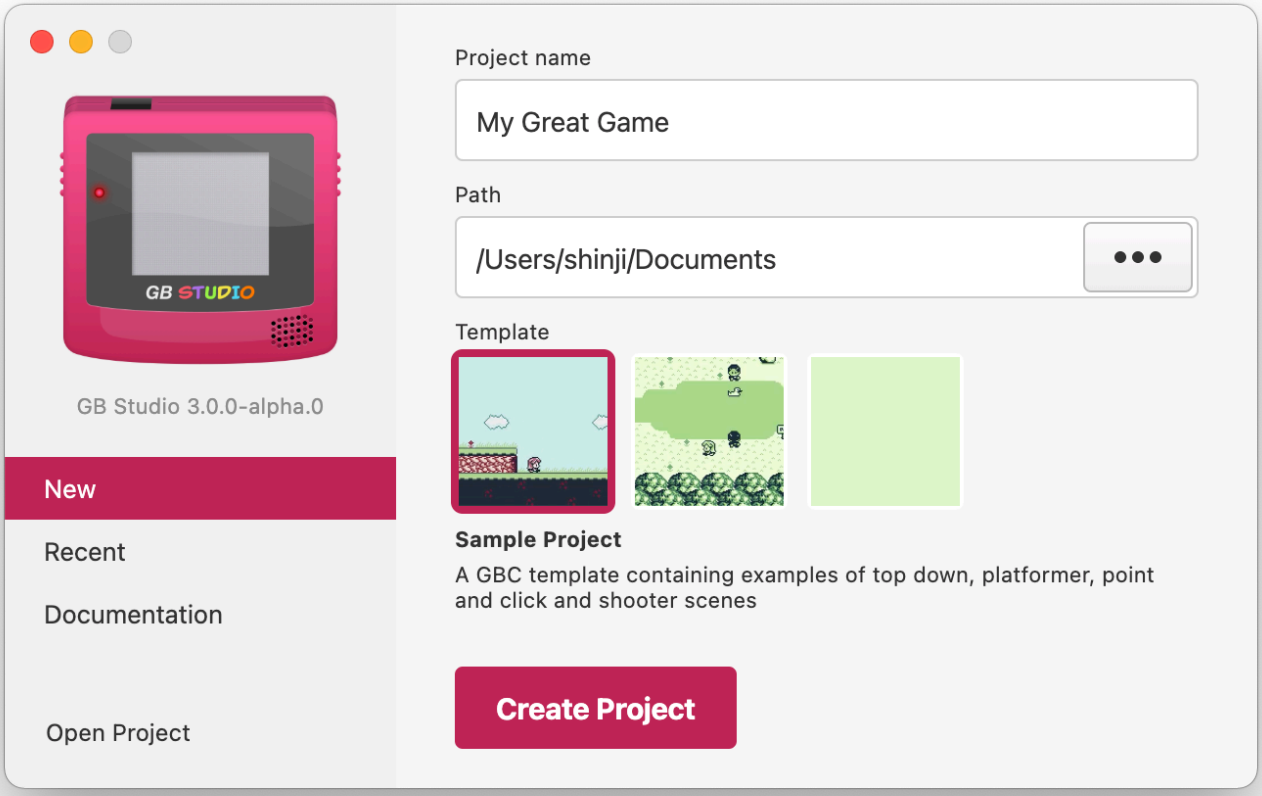
## Fedora / RPM-based Linux

For RPM-based Linux distros, download the .rpm version and run the following commands (Tested on Fedora 29)

```
> sudo yum install libXScrnSaver make lsb
> sudo rpm --ignoreos -i gb-studio-1.0.0.x86_64.rpm
> gb-studio
```

# Getting Started

When you first open GB Studio you will see the *New Project* window.

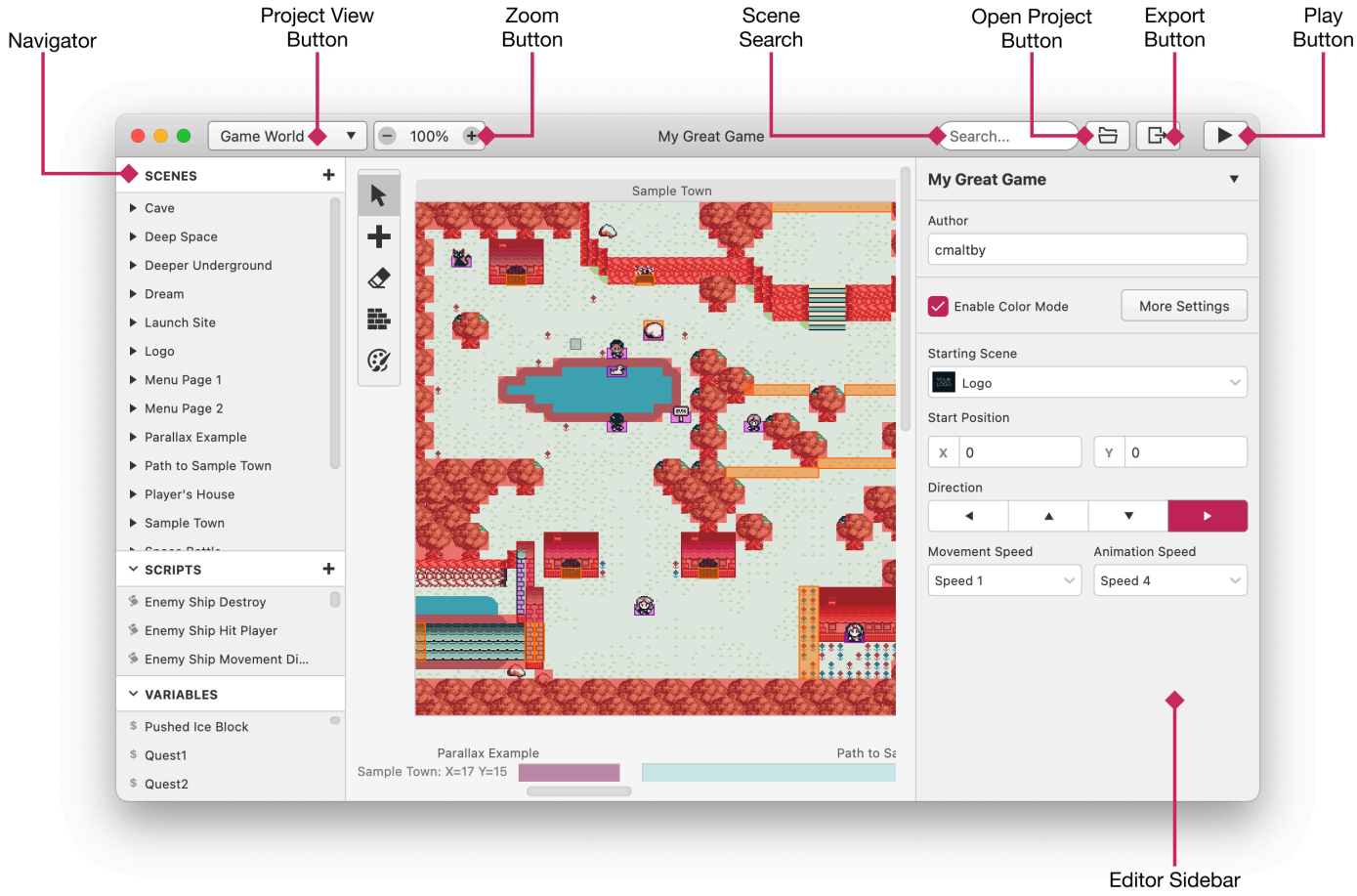



> **TIP**
>
> It's recommended to start by using the Sample Project template as it contains examples of lots of the functionality that GB Studio provides.

If you have an existing project you can open it from here by clicking *Open* and navigating to the `.gbsproj` file.

## New Project

Give your project a name to get started (don't worry, you can change this later) and choose a project template. If you're new to GB Studio then I would recommend using the *Sample Project* template which contains a few example scenes and scripts already set up so you can get a small idea of what's possible. Click *Create Project* and you'll be taken to the *Project Editor*.




As soon as you see this screen you can click the *Play button* in the top right which will build and run the project. After playing the sample project you can try clicking around the editor to see how the project is set up. Select one of the people or signposts and edit the their dialogue using the sidebar on the right then try running the project again, you've just made your very own version of the game! Don't worry if you break anything, you can always make a new project with the sample template later.

# Keyboard Shortcuts

## Play Window

When playing your game inside GB Studio use the following keyboard controls:

**Up** - `Up Arrow` / `W`
**Down** - `Down Arrow` / `S`
**Left** - `Left Arrow` / `A`
**Right** - `Right Arrow` / `D`
**A** - `Alt` / `Z` / `J`
**B** - `Ctrl` / `K` / `X`
**Start** - `Enter`
**Select** - `Shift`

These controls can be modified at any time by going to the *Settings View* under the *Controls* section.

You can also control the *Play Window* using a supported gamepad. If your web browser has gamepad support you can also use it when running a web build.

## Navigating The Menus

Much of the functionality of GB Studio is accessible through the menu bar and many of the menu items contain keyboard shortcut labels. Try clicking around on the menus to discover all of the shortcuts but the following are a few you should find useful:

**Save Project** - `Ctrl/Cmd` + `S`
**Open Project** - `Ctrl/Cmd` + `O`
**Switch View Mode** - `Ctrl/Cmd` + `1–8`
**Run Game** - `Ctrl/Cmd` + `B`
**Export ROM file** - `Ctrl/Cmd` + `Shift` + `B`

## Game World

While editing the game world you can use the following keys to quickly manipulate your scenes.

**Select Mode** - `V`
**Add Actor** - `A`
**Add Trigger** - `T`
**Add Scene** - `S`
**Eraser Mode** - `E`
**Collisions Mode** - `C`
**Set Player Start Position** - `P` (while hovering over desired location)
**Pan View** - Hold `Space` (while clicking and dragging on *Game World*)

## Drawing Mode

Drawing mode is automatically enabled in the *Collision tool* and the *Colorize tool*.

**Draw** - Click on scene
**Draw line from last point** - Click to set first point, hold `Shift`, click to set next point
**Lock brush to axis** - Hold `Shift` + Hold Click
**8px Brush** - `8`
**16px Brush** - `9`
**Fill** - `0`
**Hide Triggers/Actors** - `-`

## Collision Types

These are only available when using the *Collision tool*.

Each tile can hold a maximum of 1 ladder and 3 collision sides. Ladders will not replace existing collision when placed on top of other colliders.

**Select multiple collision types** - `Shift` + Click
**Erase collision tile** - Click on a collision tile
**Solid** - `1`
**Collision Top** - `2`
**Collision Bottom** - `3`
**Collision Left** - `4`
**Collision Right** - `5`
**Ladder (Platformer only)** - `6`

## Colorize Palettes

These are only available when using the *Colorize tool*.

**Change Brush Palette** - `1-6`
**Change Palettes** - Hold click on existing palette

## Music Editor

These are only available when using the *Music Editor*.

**Save Song** - `Ctrl/Cmd` + `S`
**Play/Pause** - `Space`
**Play from position** - `Alt/Option` + `Space`
**Switch to Tracker/Piano Roll** - `` ` ``

### Tracker

**Navigate grid** - `Arrow Keys`
**Next Column** - `Tab`
**Previous Column** - `Shift` + `Tab`

**Cell Selection** - `Shift` + `Arrow Keys` *or* Click to set first cell, hold `Shift`, click to set next cell
**Select Column** - `Ctrl/Cmd` + `A`
**Select Pattern** - `Ctrl/Cmd` + `A`, `Ctrl/Cmd` + `A`
**Copy Selection** - `Ctrl/Cmd` + `C`
**Paste Selection** - `Ctrl/Cmd` + `V`
**Delete Selection** - `Backspace/Delete`

**Transpose Selection (small step up)** - `Ctrl` + `=` *or* `Ctrl` + `Scroll wheel`
**Transpose Selection (small step down)** - `Ctrl` + `-`
**Transpose Selection (big step up)** - `Ctrl` + `Shift` + `=` *or* `Ctrl` + `Shift` + `Q`
**Transpose Selection (big step down)** - `Ctrl` + `Shift` + `-` *or* `Ctrl` + `Shift` + `A`

## Piano roll

**Change Instrument** - `1-9`
**Quick Select** - `Shift` + drag and drop mouse
**Duplicate Selection** - `Alt/Option` (while moving selected notes)

**Select Channel** - `Ctrl/Cmd` + `A`
**Copy Selection** - `Ctrl/Cmd` + `C`
**Paste Selection** - `Ctrl/Cmd` + `V`
**Paste and Replace Channel** - `Ctrl/Cmd` + `Shift` + `V`
**Delete Selection** - `Backspace/Delete`

# Saving and Loading

## Saving

To save your project select `File > Save` from the menu or press Ctrl/Cmd + S. If you try to close a project with unsaved changes GB Studio will warn you giving you a chance to save your project first. On macOS any unsaved changes in your project will be represented by a dot in the window close button.

## Loading

To load your project again, either use the *Open* button on the *New Project* window or select `File > Open` from the menu and navigate to your project's folder then select the `.gbsproj` file.

You can also return to the *Recent_Projects* window by selecting `File > Switch Project` from the menu.

## Version Control

The project folder layout and `.gbsproj` file is designed to work well with version control systems such as Git with each change by the application taking place on a new line in the data file allowing history to be tracked easily. If you want to use version control on your project you can just create the repository at the project root folder.

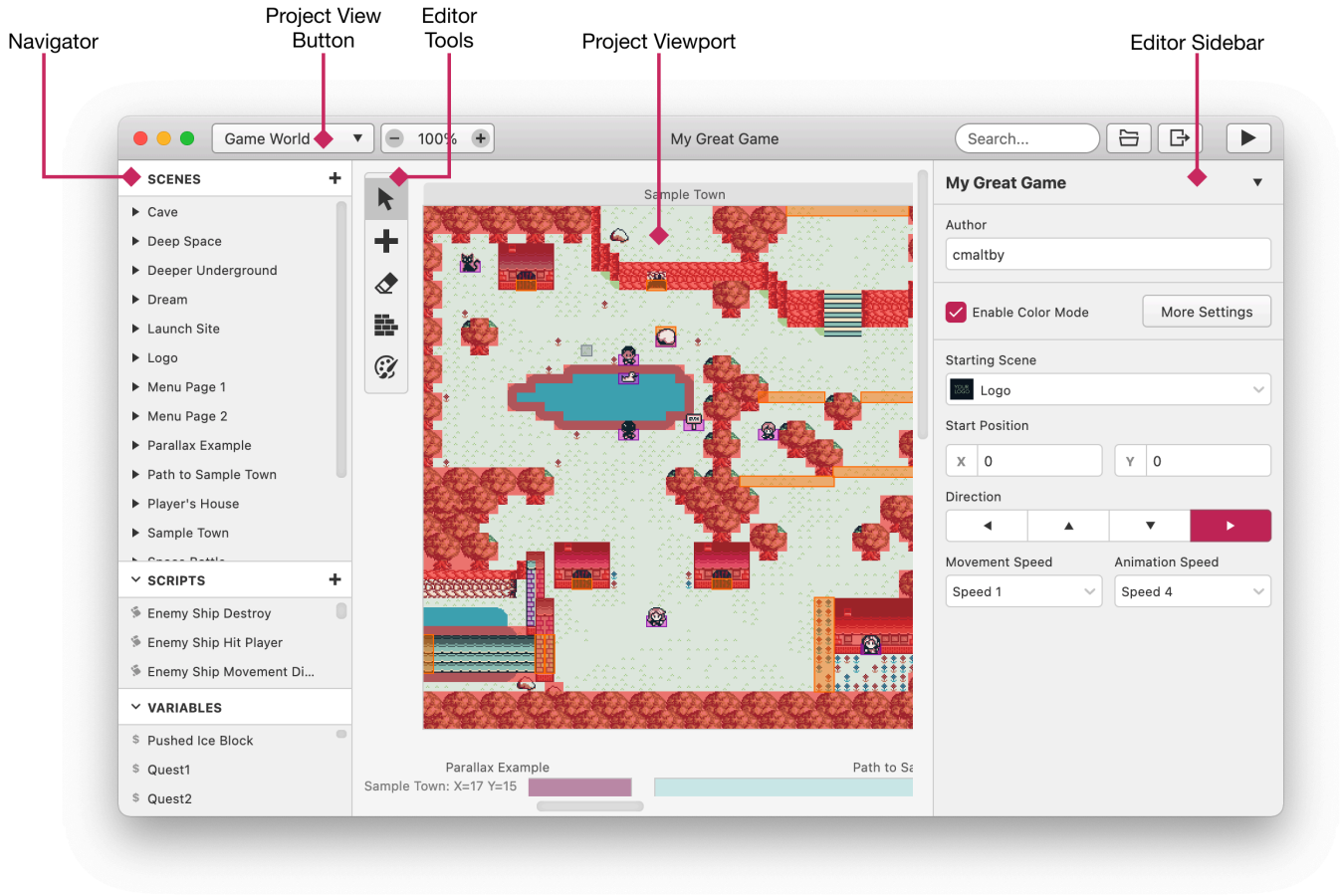
It's recommended to ignore the `build` folder from your repository using a `.gitignore` file or similar.

## Backups

Each time you save your project the previous version is saved to your project folder with the extension `.gbsproj.bak`. If you ever wish to roll back to the previous version in your project you can rename this file to have the extension `.gbsproj` and open this file instead.

# Project Editor

The default view for the *Project Editor* as shown below is the *Game World*. This is where you can create your game by combining scenes, adding actors and triggers then building scripting events to add interactions.




Use the *Editor Tools* to switch between Select, Add, Erase, Collision, and Color Drawing modes.

By default, your project's properties are shown in the *Editor Sidebar* on the right. Here you can set the project name and choose the starting scene. This project view is also where initial values for the Player actor are set. See the page on The Player for more information on the Player.

To look at project properties again from the *Editor Sidebar*, click on any empty space between scenes.

## Editor Tools

**Select tool:** Clicking any scenes, actors, or triggers will update the *Editor Sidebar* to show the properties and scripts for the item you selected. You can switch back to the Project's properties by clicking outside of a scene.

**Add tool:** You are given the choice of adding a new Actor, Trigger or Scene. After clicking any of the 3 options, your mouse

cursor will be loaded with a new item. You can place the new item by clicking inside the Project Editor, and cancel the action by pressing Escape or selecting another tool from *Editor Tools*.

**Erase tool:** All collisions, actors, and triggers will be removed when clicked. Scenes are not affected by *Erase mode*. To delete a scene, use *Select mode* and click the scene's background. In the *Editor Sidebar* click the down arrow at the top to reveal the "Delete Scene" button. All erase actions can be undone by pressing Control Z.

**Collision tool:** Allows you to add collisions to any type of scene using GB Studio's *Drawing mode*.

**Colorize tool:** Allows each tile to be given a different palette to use in place of GB Studio's default palette. The *Colorize tool* also uses GB Studio's *Drawing mode*. The palettes used here are determined in the *Palette* tab in the *Project Editor*.

See the documentation on Keyboard Shortcuts for editor tool shortcuts.

## Project Views

Using the *Project View Button* you can switch between different views of your project and its assets.

**Game World:** Create your game by combining scenes, actors and triggers.

**Sprites:** Edit your sprites and create animations.

**Backgrounds:** Preview your background assets.

**Music:** Preview and edit (hUGEDriver only) your music files.

**Palettes:** Edit your palettes for GBC games.

**Dialogue Review:** Preview and edit all the text in your game.

**Build and Run:** View logs of progress while building your game.

**Settings:** Change your project's settings such as default sprites, color palettes and keyboard controls.

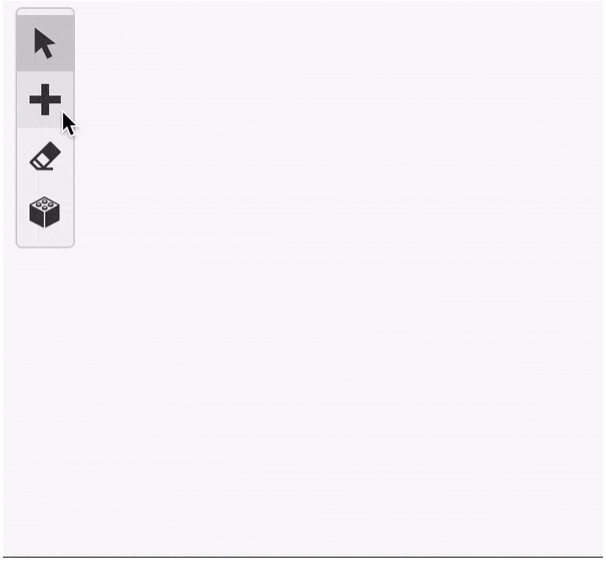
See the documentation on Assets for more information on how to add new assets.

# Scenes

A scene is a single screen of your game, it can contain multiple actors and triggers. A game is typically made-up of many scenes connected together with triggers using the *Change Scene* event.

## Adding a Scene

Click the **+** *button* in the *Editor Tools* and select *Scene* from the menu. Click on any empty space in the *Project Viewport* to place the new scene.



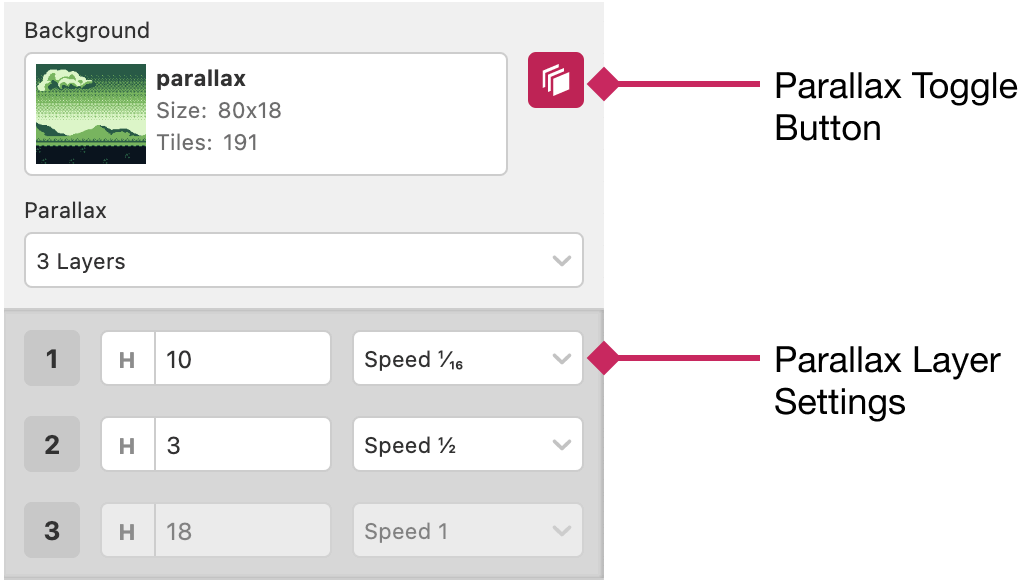
You can use the *Editor Sidebar* to give your scene a name and a background from your project's assets. See the documentation for Backgrounds for more information on adding background images.

## Scene Properties

- **Name** - Names your scene. Useful for locating your scene with the search bar.
- **Type** - Lets you choose from the list of game modes such as *Top Down 2D* or *Platformer*.
- **Background** - Lets you choose a background from the `assets/backgrounds` folder.
- **Background Palettes (Color Mode Only)** - The eight palettes that will be used when colorizing the scene.
- **Sprite Palettes (Color Mode Only)** - The eight palettes that will be used for sprites in your scene.
- **Player Sprite Sheet** - Used to set a custom player sprite for this scene. By default the scene will use the default player sprite for the selected scene *type*.

## Parallax Mode

Clicking the *Parallax Toggle Button* to the right of the *Background Selector* allows you to turn on parallax mode for the scene. When parallax mode is enabled you can split the background into up to three slices which can be modified to scroll at different speeds as the camera moves in game.




## Scripting

Scenes can contain an *On Init* script that will be called as soon as the scene is loaded in game. You can use this to do things like playing music as the scene loads, configuring events to happen on button presses, initialise actors based on the values of variables, and much more.

You can also define scripts to call when the player collides with *Actors* that have a *Collision Group* set by clicking the *On Hit* tab and choose a collision group.

To start building a script, select a scene, click the script type you want to edit and click the *Add Event button* in the *Editor Sidebar* to open the event menu. Select an event to add it to the script.

For more information see the documentation for Scripting.

## Adding Collision to a Scene

Select the *Collision Tool* from the *Editor Tools*. There are 6 collision types that can be added.

- **Solid** Stops colliding actors from entering the tile on any side.
- **Top/Bottom/Left/Right** Stops colliding actors from entering the tile from that specific side. This is useful for one-way collision and semi-solid platforms.
- **Ladder (Platformer only)** Allows moving up and down in *Platformer* scenes.

Each tile can hold a maximum of 1 ladder and 3 collision sides. Adding 4 collision sides will replace the sides with a single solid block. Ladders will not replace existing collision when placed on top of another collision.

## Colorizing a Scene

Select the *Colorizer Tool* from the *Editor Tools*. There are 8 palettes types that can be added to a scene with Color Mode enabled. Palettes can be adjusted in Settings. Note that the 8th palette in a scene will also be used for *Dialogue Windows* and menus.

The palettes used in the *Colorizer Tool* can be swapped out for existing palettes (such as the UI palette) by long-clicking on a palette.

For more information about the drawing mode used for the *Colorize Tool* and the *Collision Tool*, see Keyboard Shortcuts.

# Scene Limits

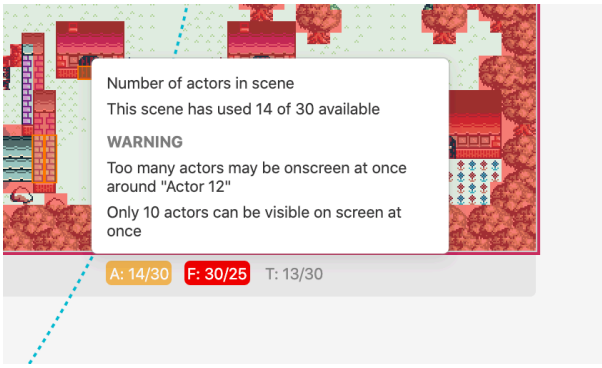
There are several limits that GB Studio has put in place to keep game performance consistent, and to minimize visual issues.

Each scene can have a maxmimum of 20 actors, and 30 triggers, and between 96 and 64 sprite tiles depending on the complexity of the background used. You can check this information by selecting a scene and looking for the gray bar under your scene that reads: `A: 0/20 S: 0/96 T: 0/30`. The letters on this bar represent the following:

- `A:` represents the number of actors that the scene is using.
- `S:` represents the number of unique sprite tiles that each actor is using in their sprite sheet.
- `T:` represents the number of triggers that the scene is using.

## Actor Limits

Each scene can have a maximum of 20 actors. Ideally, there should never be more than 10 actors within a 20 x 18 tile boundary, equivalent to `160px x 144px`. Clustering more than 10 actors together in a scene will cause some actors to become invisible in-game. GB Studio will warn you if it thinks this will be the case for a scene:




You can address this message by moving or deleting actors so no more than 10 will be seen in a 20 x 18 tile boundary. You can use the Eraser Tool to delete actors. Actors will still become invisible if more than 10 actors move into the screenspace after the scene starts.

## Sprite Tile Limits

The exact number of sprite tiles available in a scene depends on the amount of tiles used in the selected background image as some memory is shared between sprite and background tiles. If the selected background uses less than 128 unique tiles, you can use 96 sprite tiles, each background tile above 128 takes away from sprite tiles available until a minimum of 64 tiles are available.

## Trigger Limits

Each scene can have a maximum of 30 triggers. You can use the Eraser Tool to delete triggers.

# The Player

## Start Position

The player starting position is indicated in the game world view by the  icon.

Clicking in the background between scenes switches the sidebar back to the Project Editor where you'll have options to set the player starting scene, position, and direction.

You can also change the player start position by dragging the  icon and can even drag between scenes.

## Default Sprite Sheet

Each scene type (*Top Down 2D*, *Platformer* etc.) can have a different default player sprite sheet that will be used in any scenes of this type unless you override this for the specific scene.

You can edit the default player sprite sheets for each scene type from the Settings View.

## Scripting

Most actor script events can also be applied to the player. In addition you can use *Set Player Sprite Sheet* event to change the graphics used for the player character mid-game. Changing the sprite sheet will only affect the current scene unless you choose to *Replace Default For Scene Type* which will causes any other scenes of the same type to also use this player sprite (unless and override was provided).

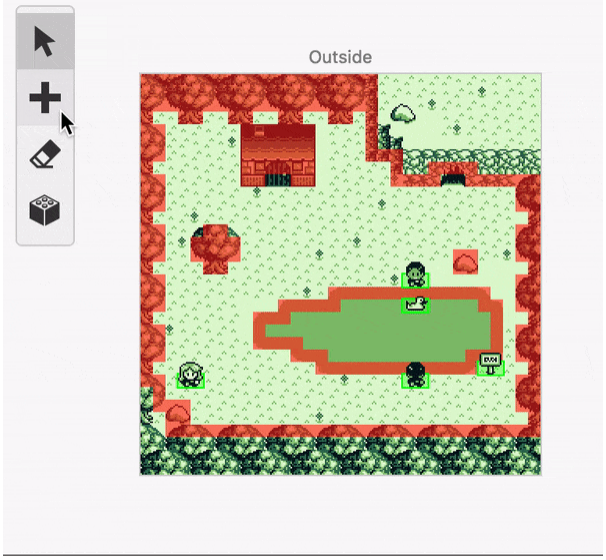
When switching between scenes the player will always become visible at the scene start location regardless of previous visibility options. if you want the player to be hidden on a scene e.g when showing a title screen or cutscene add a *Hide Actor* event to the scene's *On Init* script.

# Actors

Actors are the characters and objects in your scene that you can interact with.

## Adding an Actor

To add an actor to a scene click the **+** *button* in the *Editor Tools* and select *Actor* from the menu (alternatively press the **A** key), then click on the scene and position where you wish to place the actor.




## Actor Properties

- **Name** - Names your actor. Giving your actors a *name* helps organize them in your project. An actor's name will be visible in any drop-down menu that asks you to pick an actor, such as the *Actor: Hide* event.
- **Position** - Sets the X and Y position where the actor will be positioned in a scene. You can also change this by dragging the actor around the the *Game World*.
- **Pin to Screen** - Using the *Pin Button* next to the actor position you can choose to pin the actor to the screen which cause it to not move as the game screen scrolls.
- **Sprite Sheet** - Choose which sprite graphics should be used to display the actor.
- **Movement Speed** - Choose how fast the actor should move when scripting events are used.
- **Animation Speed** - Choose how fast the actor animations should play.
- **Collision Group** - Choose if scripts should play automatically when colliding with this actor.

### Pin to Screen

When an actor is pinned, the actor will not move without a script, and does not create collisions with other actors in your scene.

Enabling this property will temporarily change your scene to be blacked-out, with a `160px x 144px` boundary in the top-left corner showing part of your original scene. Use your mouse to drag the actor to where you want it to be pinned to the screen.

Select a different actor, the scene, or the project to return the blacked-out view of your scene to normal.

### Collision Groups

Actors can be given a collision group in the *Editor Sidebar*. When enabled, the option to run scripts based on collisions will appear in the *Editor Sidebar*. To learn more about On Hit scripts, see the documentation for Scripting.

## Scripting

Actors can contain multiple scripts that will be called a different points in your game.

- **On Interact:** This is called if the player stands in front of this actor and presses the *Interact* button.
- **On Hit:** (only if collision group is set) This is called when this actor collides with either the player or a projectile with a specified collision group
- **On Init:** Called as soon as the scene is loaded in game.
- **On Update:** Repeatedly called while the actor is on screen, and once the script finishes it will repeat. You can use this to create movement scripts

To start building a script, select an actor, click the script type you want to edit and click the *Add Event button* in the *Editor Sidebar* to open the event menu. Select an event to add it to the script.

For more information see the documentation for Scripting.

## Limits

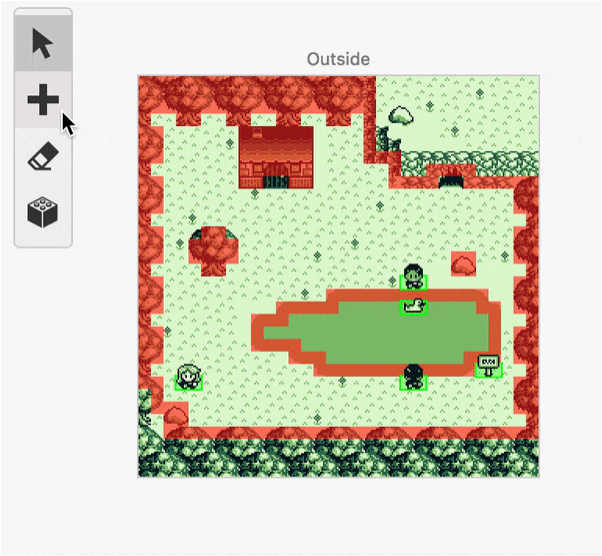
There are limits to how actors and their sprites can be used in GB Studio. These limits are to make sure your game appears as intended, as well as to keep your actor logic running smoothly. The exact limits depend on the complexity of the background image used in your scene, see Scenes for more information.

# Triggers

Triggers are areas in a scene that, when the player walks over them, will cause a script to play. They are useful for creating doorways between scenes and to start cutscenes when the player moves to a specific position.

## Adding a Trigger

To add a trigger to a scene click the **+** *button* button in the *Editor Tools* and select *Trigger* from the menu (alternatively press the **T** key), then click and drag across the scene where you wish to place the trigger setting the desired width and height.




The *Editor Sidebar* will switch to show the trigger settings where you can give the trigger a name for easier navigation later, reposition and scale the trigger and create the script that will play when the player walks on the trigger.

## Scripting

When the trigger is selected click the *Add Event button* in the *Editor Sidebar* to open the event menu and start building the script. For more information see the documentation for Scripting.

# Assets

When your project was created an `assets` folder was also made within the project containing a number of subfolders for each asset type in your game.

GBStudio doesn't currently contain any ability to edit the graphics in your game directly, you instead can use your favourite existing applications and save files into these folders where they will instantly appear ready to use in your project. If you edit a sprite or background PNG file and save using an external image editor the change will be seen in your *Project Window* as soon as you switch back to it.

While you can create graphics in any application that can output PNG files it is recommended to use Aseprite or Photoshop to create your sprites and UI elements then to use Tiled Map Editor to create your backgrounds. Each image asset type has a different set of requirements detailed over the new few sections of this documentation. You can select the default application to open when clicking asset edit buttons in the *GB Studio Preferences* window.

For music you can either create UGE files or MOD files depending on the *Music Driver* you choose in the *Settings View* (hUGEDriver is recommended).

- UGE files can be edited directly within the GB Studio Music Editor.
- MOD files can be edited with OpenMPT or MilkyTracker.

## Community Assets

If you're looking for a collection of free assets, ready to be used in GB Studio there is a community run repository on Github available at GB Studio Community Assets.

# Sprites

Sprites are the graphics used by playable or interactive characters in your scenes. Add sprites to your game by including `.png` files in your project's `assets/sprites` folder.

Because there are limits to how many sprites tiles can be loaded into a single scene, be sure to check your the frame limits across your scenes when adding new sprites. See Scene Limits for more information.

## Simple Sprites

A simple sprite has one or more `16px` x `16px` frames laid out horizontally in an image file. A sprite with a single frame will be `16px` x `16px` while a sprite with three frames will be `48px` x `16px`.

### Static sprites

For sprites that only need a single frame (e.g. static items such as signposts) create your `.png` as a `16px` x `16px` image containing just the one frame required.

### Animated sprites

If you want to have sprites that play short animations, you can make a `.png` with between 2 frames at `32px` x `16px` and 25 frames at `400px` x `16px`. Using these sprites on an actor will let you select which frame you want to display by default, on top of playing the full animation at a specified speed.

### Actor

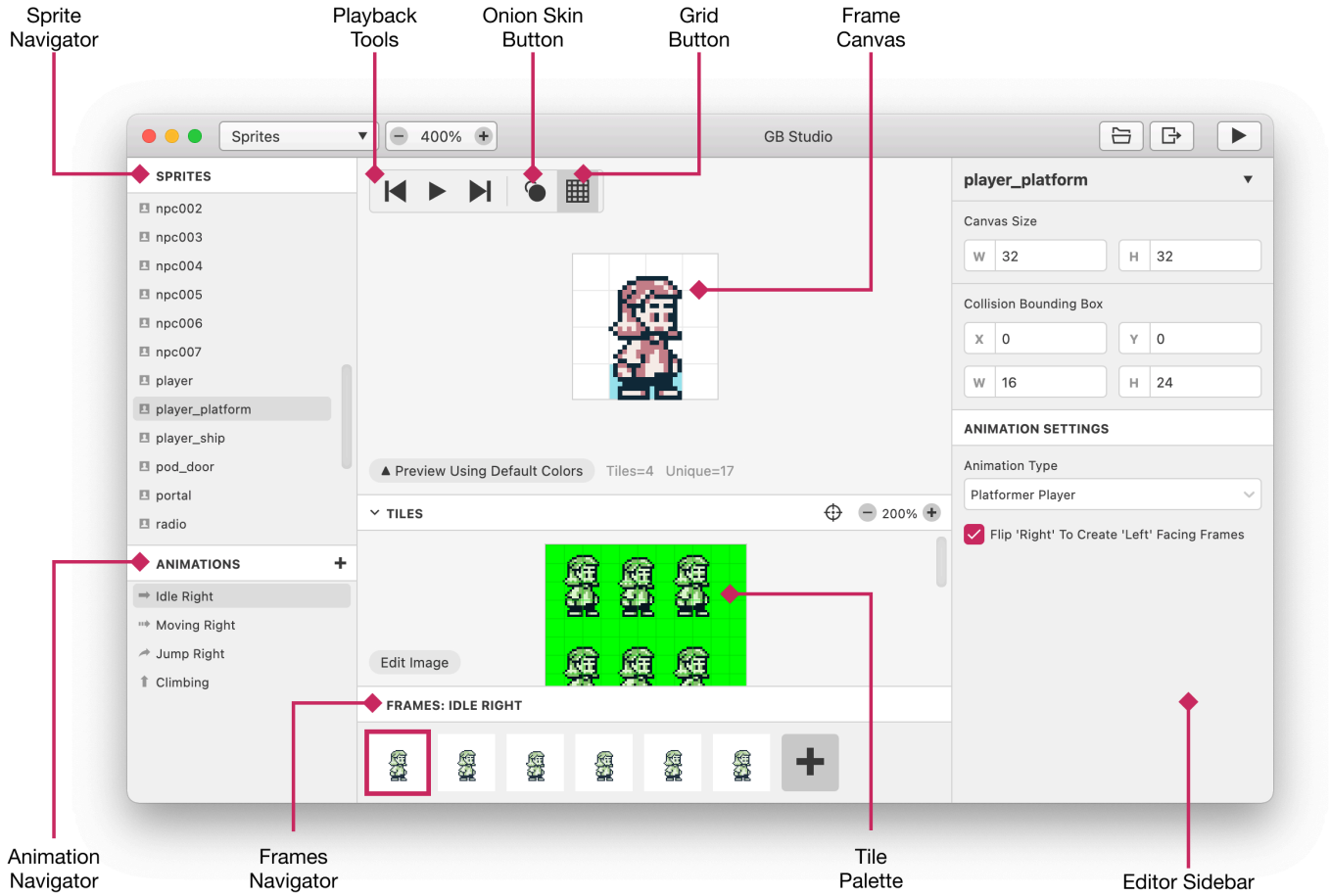
To make a static sprite that changes based on the actor's direction, create a `48px` x `16px` `.png` containing the three frames: One forward facing, one upwards facing and one right facing. The left facing sprite is automatically generated by flipping the right facing frame horizontally.

### Animated Actor

To make sprites that have animated movement, or that can be used as a player character, create a `96px` x `16px` `.png` containing six frames: Two forward facing, two upwards facing and two right facing.

## Sprite Editor

When you want to progress to making more complex sprites you can use the *Sprite Editor* by clicking the *Project View Button* and selecting *Sprites*.




## Composition of a Sprite

A sprite consists of:

- Multiple *Animation States*, by default only a single animation state is created for a sprite, you can make a new one by clicking the `+` button in the *Animation Navigator*.
    - Each animation state consists of multiple animation frames, viewable in the *Frames Navigator*, click the `+` button to create a new frame, and click a frame to view it in the *Frame Canvas* for editing.
- A *Tile Palette*, this is the `.png` file from your assets folder. Click into the tiles palette to select a tile, you can then draw it by clicking into the *Frame Canvas*.
- A *Canvas Size* this is the width and height of your *Frame Canvas*, set this from the *Editor Sidebar* to the size you want your sprite to be.
- A *Collision Bounding Box* this is the width, height and position of an invisible box used for collision detection within the game engine, set this to fit as closely as possible around the collidable area of your sprite.

## Animation Settings

In the *Editor Sidebar* you can choose from a list of sprite types, setting this will determine the number of animations available for your sprite and what names they have in the *Animation Navigator*. For example while you can use any sprite type for a *Platformer*

scene player, it's recommended to set the type to be *Platformer Player* as this will allow you to configure the `Jump` and `Climbing` animations.

Some sprite types also allow you to "Flip 'Right' to Create 'Left' Facing Frames", this lets you create both the left and right sprite animations from a single animation that gets flipped automatically saving you from creating these animations manually.

### Animation States

Using the `+` button in the *Animation Navigator* you can create new *Animation States*. These let you create custom animations that can be triggered from scripts.

Once you've created a new *Animation State* you can name it by typing in the *State Name* input in the *Editor Sidebar*, or by selecting an existing sprite name.

The list of sprite names is global for your project and it's recommended to keep the number of unique names low. Each one you add increases the amount of memory required for all sprites in your game. For example, rather than having two unique sprites with states `Explode` and `Squash`, consider making a single state used by both called `Destroy`.

To switch which animation state an actor should use in your game, you can use a `Set Actor Animation State` event. This allows you to choose an actor and which animation state you should switch to. Make sure that the spritesheet you're using has animations defined for the state you've chosen in the event!

### Frame Canvas

Once you have selected an *Animation* and *Frame* to edit you can use the *Tile Palette* and *Frame Canvas* to create an animation frame.

Start by clicking on the tile you wish to use in the *Tile Palette*.

- You can select multiple tiles by clicking and dragging in the *Tile Palette*
- By default the *Tile Palette* snaps to an `8px` grid, this is to increase the chance of tile reuse as each unique tile you use in your sprite takes away from limits when used in scenes. If you know what you're doing and want to disable this grid you can turn on `Precision Mode` by clicking the button in the top right of the *Tile Palette* or by holding `Alt` while making your selection.

Once you have a tile selection click into the *Frame Canvas* to draw the tiles into your frame. You can then move tiles around in the *Frame Canvas* by dragging them and if you have tile selection you can move frames to the front or back and flip them horizontally or vertically by using the *Editor Sidebar*.

### Onion Skin

Using the *Onion Skin* button you can toggle the *Onion Skin* feature. This displays a semi-transparent version of the previous frame behind the *Frame Canvas* allowing you to more easily make decisions about the flow of the animation.

### Deleting Tiles and Frames

To delete a tile or frame, select it in either the *Frame Canvas* or *Frames Navigator* and press `Backspace` on your keyboard. Alternatively to can make your selection then click the dropdown button arrow in the top right of the *Editor Sidebar* to access a menu where deleting is available.

## Image Requirements

Sprite `.png`s must only contain the following four colors:

#071821

#86c06c

#e0f8cf

#65ff00

Download the GB Studio Palette Swatches for:

Adobe Photoshop

Aseprite

The color `#65ff00` is used to represent a transparent background in game and will be invisible in-game and in the *World Editor*.

Colors that are not one of the above hex codes will be matched to the nearest color. Unlike backgrounds, the color `#306850` can not be used in sprites.

# Backgrounds

Each of your scenes requires a background image that defines how that scene should look. You can add backgrounds to your game by including PNG files in your project's `assets/backgrounds` folder.

## Color Requirements

Background PNGs must only contain the following four colors:

#071821

#306850

#86c06c

#e0f8cf

Download the GB Studio Palette Swatches for:

Adobe Photoshop

Aseprite

Colors that are not one of the above hex codes will be matched to the nearest color. Unlike sprites, the color `#65ff00` can not be used in backgrounds.

## Size Requirements

- Backgrounds are divided into `8px` x `8px` tilesets so the total image size must be a multiple of `8px` in both width and height.
- A background has a minimum size of `160px` x `144px` (the GB screen size)
- Both the width and height of a background must be less than or equal to `2040px`.
- The width of the image multiplied by the height must be less than or equal to `1,048,320`. For example an image with the width `2016px` will have a max height of `520px` (because `2016 * 520 = 1048320`)

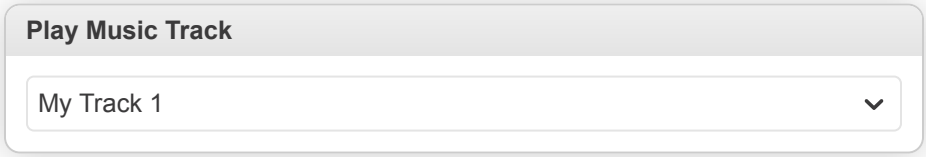
## Tile Requirements

In most scene types a background image can contain no more than **192** unique `8px` x `8px` tiles at once due to memory limits. This means that even using the smallest background size possible you must repeat about half of your tiles. Where possible repeat tiles between images as they will be grouped together saving on total game size. It is recommended to use a tile map editor such as Tiled to ensure your backgrounds conform to the pixel grid.

The exception to this is scenes with their *Scene Type* set as *Logo*, these scenes can use a `160px` x `144px` sized image with no limits on unique tiles but note that in *Logo* scenes you are unable to use *Actors* or display a *Player*.

# Music

Music can be played in your game using the Play Music Track event in your *Actor*, *Trigger*, or *Scene* scripts.




You can add music to your game by including `.uge` or `.mod` files in your project's `assets/music` folder.

A project can only support one type of music files, this can be configured on the Settings View by selecting either MOD or UGE as the Music Format.

`.uge` files can be created and edited with the *Music Editor*.

See the Music Editor documentation for more information.

`.mod` files are created and edited using an external Tracker software. You can select the default application to open when clicking asset edit buttons in the *GB Studio Preferences* window.

See the MOD files documentation for more information.

# Music Editor

If you have your *Music Format* in the *Settings View* set to `UGE (hUGEDriver)` (the default in GB Studio 3 and above) you can add music to your game by including `.uge` files in your project's `assets/music` folder.

Those files can be edited using the *Music Editor* by clicking the *Project View Button* and selecting *Music*. The editor also allows to create new songs by pressing the `+` button on top of the Song list.

> `.uge` files can also be edited using **hUGETracker**

## Getting Started

The *Music Editor* is divided in three parts:

- **Navigator**: Contains the list of songs and instruments for the selected song
- **Song Composer**: The music editor itself. Has two views: Piano Roll and Tracker. The first icon in the toolbar allows to change views.
- **Editor Sidebar**: Allows to edit the song title, artist name and tempo and also shows the instrument or effect editor when selected.

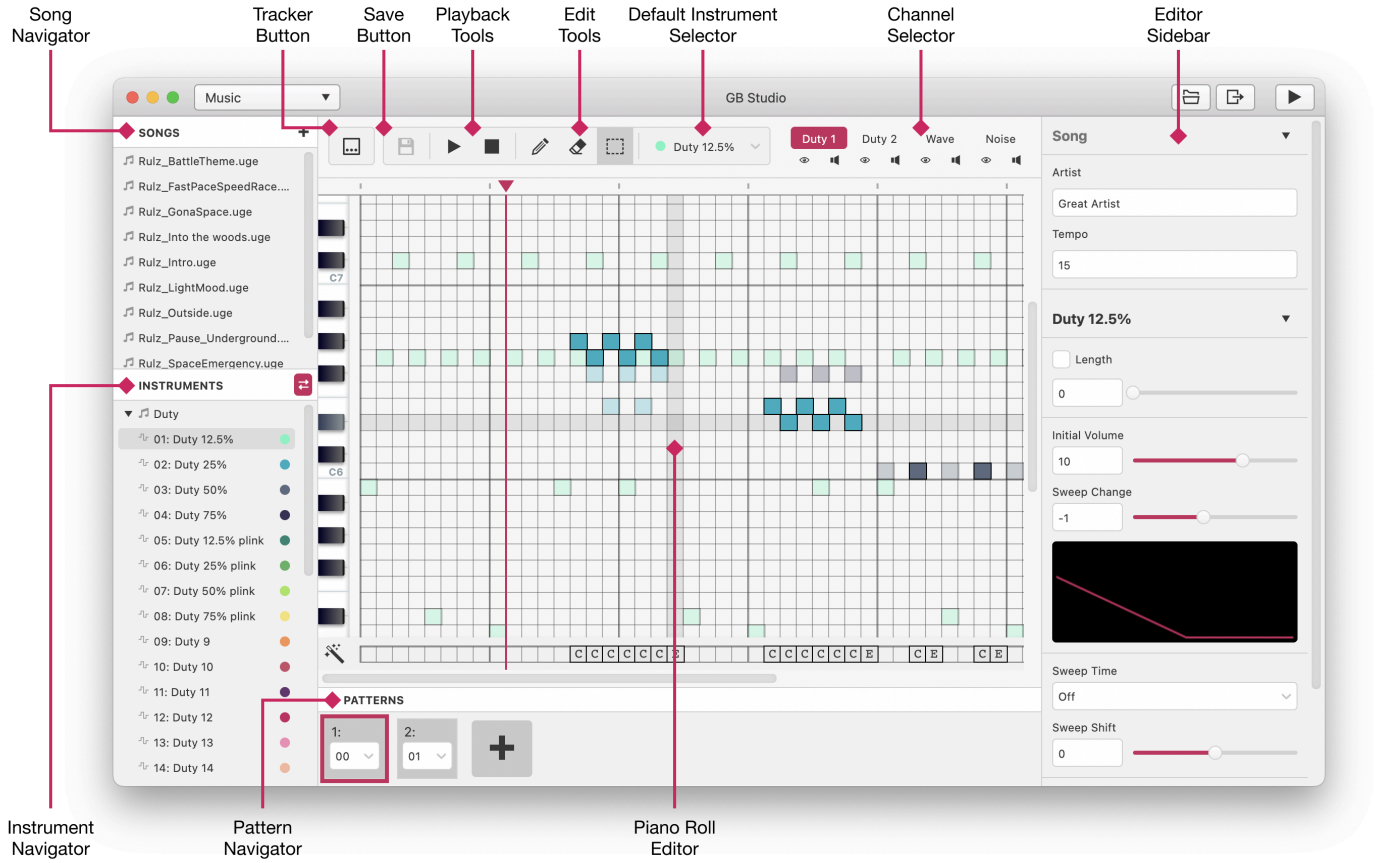
### Structure of a Song

A song consists of:

- Four *Channels*: Duty 1, Duty 2, Wave and Noise.
  - Each channel is better suited for a different type of sound (for example: the Noise channel is usually fit for drum rythyms).
  - Each channel has its own set of 15 *Instruments*. Duty 1 and Duty 2 share the same set of instruments.
- Multiple *Patterns*, a unique group of notes in each of the four channels.
  - Each pattern contains a sequence of up to 64 notes per channel, and each note is formed by a pitch ranging from `C-3` to `B-8`, an instrument and an effect.
  - Patterns can be repeated or arranged to form the full song using the *Pattern Editor*.
- A *Tempo*, how many ticks (64 per second) have to elapse before a row is complete. The greater the number of ticks, the slower the song is.

## Piano Roll

In Piano Roll mode you use the mouse to add notes to the pattern. It reads like a music sheet, the time is represented in the horizontal axis (columns) while the note pitch is represented on the vertical axis (rows).




You can only add notes to one channel at a time, selectable on the top right toolbar. The channels can be muted with the speaker icon for each channel. The channels that aren't selected can be previewed by clicking the eye icon.

### Using the Piano Roll

To input a note, select the pen tool in the toolbar and click on a cell. The note will use the selected instrument in the toolbar.

To remove a note, select the eraser tool in the toolbar and click on an existing note. You can also right click on an existing note to remove it.

To select a note, select the selector tool or press `Shift`. Once selected a group of notes and drag and drop them anywhere else in the grid.

The effect bar, at the bottom of the piano grid, allows to add an effect to a given note using the effect editor in the right pange.

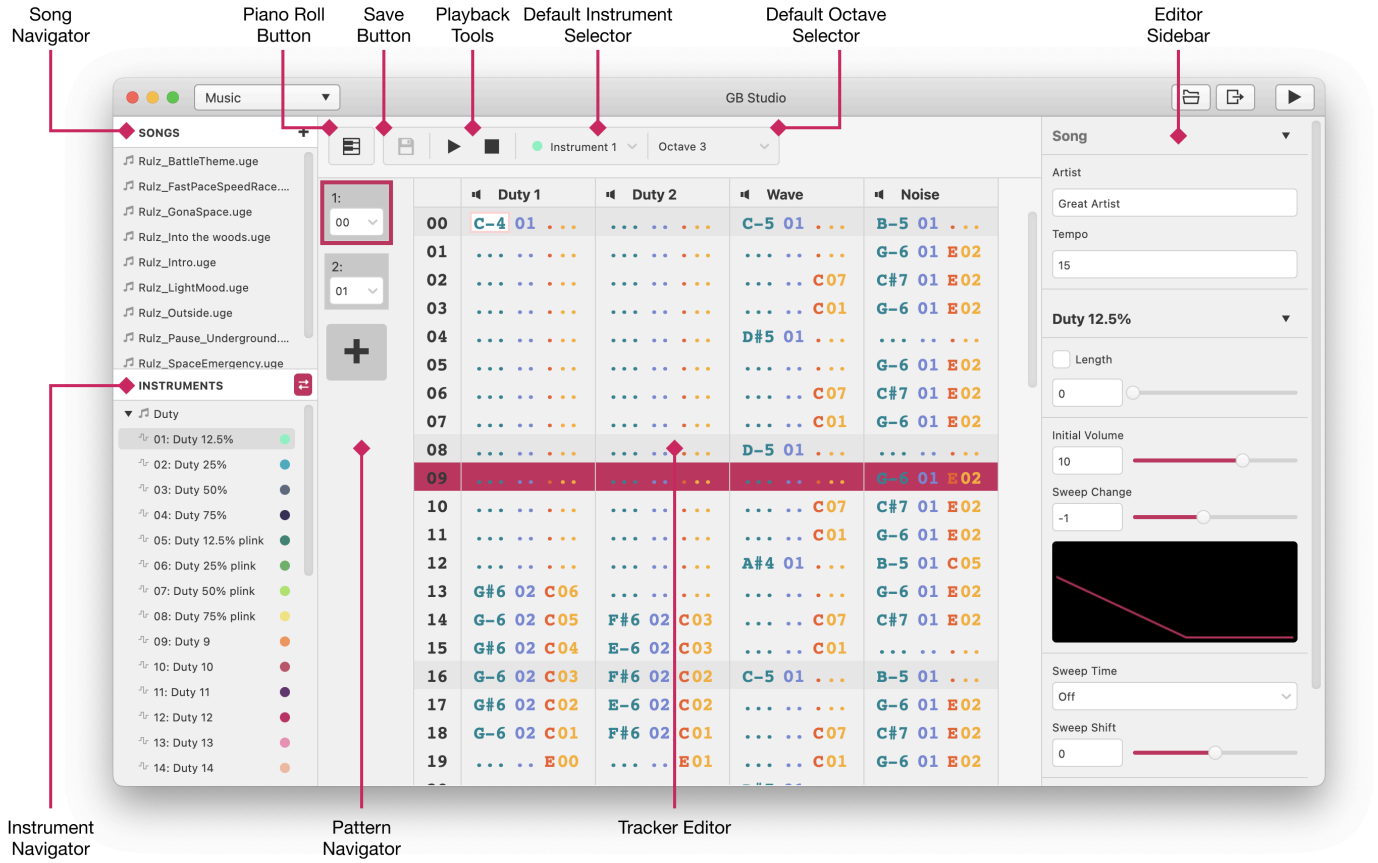
The song can be previewed at any time by pressing the play button.

To set the playback starting position, click the area above the piano roll, where the playback head is shown.

The song can be saved by pressing the save button or Ctrl/Cmd + S.

## Tracker

In Tracker mode you use the keyboard to add notes to the pattern. The song advances from top to bottom, with each row representing a position of the song.




There's one column for each channel, and each column is divided in 3 fields: Pitch (or Note), Instrument and Effect.

```
C-5 01 240
--- -- ---
 |   |  |
 |   |  +------ Effect column (Volume changes, arpeggios, panning, etc.)
 |   +--------- Instrument
 +------------- Note and octave (A C note in the 5th octave. The dash can be a #, which
signifies a sharp note e.g. C#, D#)
```

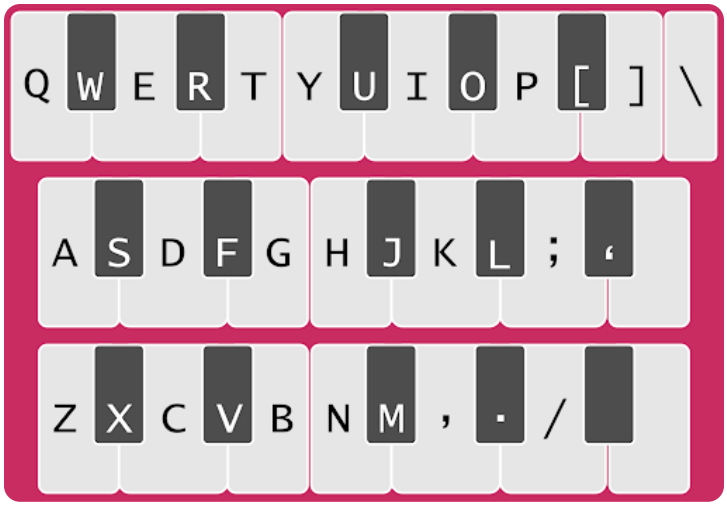
Rows can be empty, or can be partially filled (with just an effect, for example).

## Using the Tracker

The song grid can be navigated with the cursor keys.

There's two keyboard layouts to input the values in the note column. The layout can be selected in the *GB Studio Preferences* window.

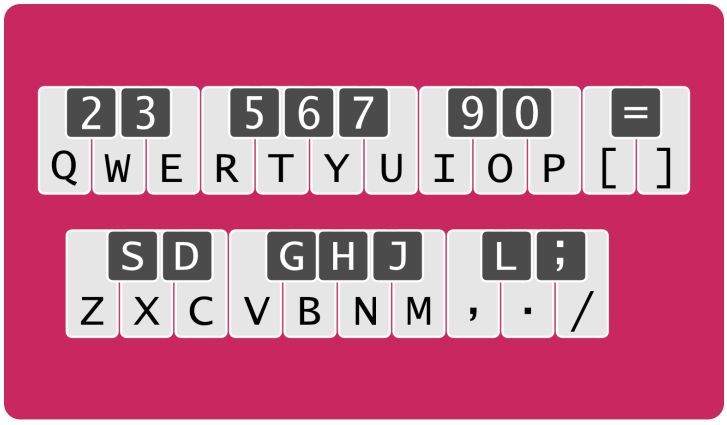
***Linear layout***




*This is the layout used by trackers like OpenMPT and hUGETracker.*

Each keyboard row (or "line") represents one octave on a piano. Keys from `Q` to `/` are used to input the values, starting with `C` in the base octave (3 by default).

***Piano layout***




*This is the layout used by trackers like MilkyTracker or FastTracker2.*

The keyboard is split in two of groups of two rows of keys. On each group the top keys represent the black keys of a piano, andthe bottom keys the white ones. Keys from `2` to `/` are used to input the values, starting with `C` in the base octave + 1 (4 by default).

The base octave can be selected in the toolbar.

The numeric keys are used to input the value in the instrument column. A default instrument can be selected in the toolbar and be used automatically when adding a new note.

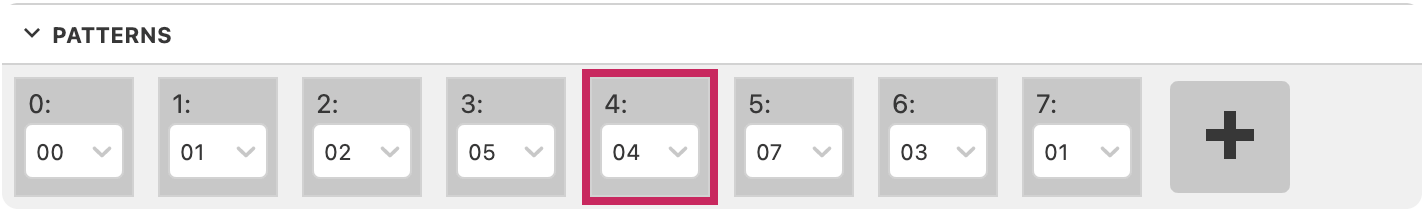
The numeric keys, and keys `A` through `F` are used to input values in the effect column.

The song can be previewed at any time by pressing the play button.

To set the playback starting position, click the row number on the left side of the tracker grid.

The song can be saved by pressing the save button or Ctrl/Cmd + S.

## Pattern Navigator




The dropdown menu on each cell allows you to select one of the existing patterns or assign an empty one to the current position.

The plus button allows you to add a new pattern to the song.

**Note:** any unused pattern will be removed from the song on save.

## Instruments

> Borrowing from the descriptions on the hUGETracker manual

Selecting an instrument in the left sidebar will open the instrument editor on the right sidebar.

Changes to the instrument can be previewed at any time by pressing the "Test Instrument (C5)" button, which will play the C5 note for a few seconds with the selected instrument.

Other than the instrument name, each instrument has its own set of fields that can be edited.

### Duty Instruments

**Length:** When enabled, the note will be cut off immediately at a specific length. If not enabled the note will play until a new note starts.

**Initial Volume**: Sets the starting volume for the envelope. If there's no sweep change set this will be the volume for the note.

**Sweep Change:** Defines how steep the volume change will be. The higher or lower the value, the quicker the note will fade in or out.

**Sweep Time:** Selects the "sweep time" for the note to take. The greater the value, the slower the sweep.

**Sweep Shift:** Selects the direction and magnitude of sweep for the note to take per "tick" as specified by Sweep Time. Positive values portamentos the note up, negative values portamentos it down.

**Duty:** Selects the timbre of note to play. Each one sounds different, and they are useful when you don't want both of the duty channels to clash with one another.

### Wave Instruments

**Length:** When enabled, the note will be cut off immediately at a specific length. If not enabled the note will play until a new note starts.

**Volume**: Specifies at what volume a wave instrument shall play unless overridden by a volume effect command. There are only 3 possible values here, as the wave channel's volume interface is more limited than the other channels.

**Waveform**: Selects which waveform should play as part of this instrument. The selected waveform can be edited by drawing on the waveform preview.

### Noise Instruments

**Length:** When enabled, the note will be cut off immediately at a specific length. If not enabled the note will play until a new note starts.

**Initial Volume**: Sets the starting volume for the envelope. If there's no sweep change set this will be the volume for the note.

**Sweep Change:** Defines how steep the volume change will be. The higher or lower the value, the quicker the note will fade in or out.

**7-bit counter:** When checked, the instrument will sound more like a musical tone rather than noise.

**Noise Macro:** Like an arpeggio effect, set up to 8 pitch changes +-32 from the noise frequency, advancing every frame. Freat for kick drums or fast sweeping noise. Must be shorter than your current song tempo.

## Effects

> Borrowing from the descriptions on the hUGETracker manual

| Effect | Name | Description |
|---|---|---|
| 0xy | Arpeggio | On every tick, switch between the playing note, note + `x`, and note + `y`, where `x` and `y` are values in semitones. Can be used to create "chords" or a strum effect. |
| 1xx | Portamento Up | Slide the pitch up by `xx` units every tick. |
| 2xx | Portamento Down | Slide the pitch down by `xx` units every tick. |
| 3xx | Tone Portamento | Slide the pitch towards the specified note value by `xx` units every tick. Stops when it reaches the specified note value.<br>**This effect cannot be used in a cell with an instrument value.** |
| 4xy | Vibrato | Rapidly switch between the specified note value and note + y, at the rate of `x`, where `y` is a value in units. This is similar to arpeggio, except you can control the frequency, and the amount is specified in units rather than semitones. |
| 5xx | Set Master Volume | Sets the master volume control of the Gameboy for the left and right speakers. Use the effect editor to create one of these effects. Note that a volume of zero is not completely silent. |
| 6xx | Call Routine | Call a user-defined routine. Routines can be created by using the Set Music Routine event. |

| Effect | Name | Description |
|---|---|---|
| 7xx | Note Delay | Wait `xx` ticks before playing the note in this cell. |
| 8xx | Set Panning | Sets which channels play on which speakers. Use the effect editor to create one of these effects. Can also be used as a mute for a channel by setting it to output on neither left nor right. |
| 9xx | Set Duty Cycle | Select duty cycle for either the Duty 1 or Duty 2 channels. If this effect appears on the Noise or Wave channels, it will affect the Duty 2 channel. Valid values for `xx` are 00, 40, 80, C0. Under the hood, the `xx` value is loaded directly into Duty 1 or Duty 2's length register, so you could theoretically achieve other effects than just duty cycle changing. |
| Axy | Volume Slide | Slide the note's volume up by `x` units, and then down by `y` units.<br>This effect actually retriggers the note on each tick, which might not be noticeable for instruments without length/envelope, but could potentially sound bad if those are present.<br>Recommended to use either instrument envelopes, or the `C` command instead if you can.<br>**This effect does not work in the same cell as a note/instrument!** |
| Bxx | Position Jump | Jump to the start of pattern `xx`. If `xx` is `00` jump to the next pattern. |
| Cev | Set Volume | Set the envelope `e` and volume `v` of the channel. Must be accompanied by a note and instrument to work (except on the Wave channel).<br>Valid volumes range from 00-0F (00,04,08,0F for Wave channel).<br>Valid envelopes for `Cev` 00-F0, 0 use instrument, 8 no fade, 1-7 fade quieter, 9-F fade louder, smaller values fade faster. |
| Dxx | Pattern Break | Jump to the next pattern early, and start on row `xx`. |
| Exx | Note Cut | Cut the note short after `xx` ticks. |
| Fxx | Set Speed | Set the number of ticks per row to `xx`. Can be used in an alternating fashion to create a swing beat. |

## Keyboard Shortcuts

See Keyboard Shortcuts > Music Editor

# MOD Music

If you have your *Music Format* in the *Settings View* set to MOD (GBT Player) (the default in GB Studio 2 and below) you will need to provide music as `.mod` files.

## Requirements

Add music to your game by including .mod files in your project's `assets/music` folder. GBT Player is a driver that takes .mod files and converts them to instructions for the Gameboy. GBT Player interprets .mod files differently than the Amiga computers that the .mod format was originally designed for, so every .mod file that GBT Player reads should be composed/arranged to be used with GBT Player.

As an alternative to composing, there is a way to import .midi files to OpenMPT for playback in GBT Player. More information can be found under Frequently Asked Questions. You can also browse the GB Studio Community Assets to find free, GBT-compatible music under the MIT licence.

To compose GBT-compatible .mod files, you can use software such as **OpenMPT** (for Windows or Linux using Wine), **MilkyTracker** (for Windows, Mac and Linux), **ProTracker**, and **BassoonTracker** (browser-based) to name a few. Any software that loads and exports .mod files can write files that are compatible with GBT Player.

## Resources

It is recomended you read through your tracker's documentation to learn about your tracker:

- OpenMPT's Documentation
- MilkyTracker's Documentation
- BassoonTracker's Documentation

Lastly, the GB Studio Discord also has a dedicated #music-help channel and a #tutorials channel in case you get stuck.

## Getting Started

1. Create a blank GB Studio project, find the file `assets/music/template.mod` and open it with your tracker of choice.
    - **You must edit this file to get an accurate representation of the instruments you can use.**
    - MilkyTracker users should save this file as an `.XM` file. Saving a .mod file in MilkyTracker will corrupt it. Export your song as a .mod file every time you want to test your song in-game.
2. Use the instrument list shown later in this document to pick the sounds you want. Changing the samples in your tracker will not affect how they sound in-game.

When done, add your .mod files to the `assets/music` folder of your project. **Test your song in-game often to keep track of any audible in-game differences.**

Here is a quick rundown of how a tracker works:

```
C-5 01 v64 ...
--- -- --- ---
```

This is what comprises of a channel's row. Rows can be empty, or can only be partially filled (with just an effect, for example). There's 4 of those columns in total.

Any part in this documentation where you see data that starts with `ModPlug Tracker MOD`, you can copy that entire block into OpenMPT as-is. Any data copied from OpenMPT looks like that when you paste it into any text application.

## GBT Player's Channel Limitations

.MOD files need to use 4 channels. Loading a copy of template.mod before composing will ensure this is set-up correctly.

| Channel # | Sound type | Note Range[1] | Instruments | Effects |
|---|---|---|---|---|
| Channel 1 & 2 | Pulse | C3 to B8 | 1-4 | 0, C, E8, EC, B, D, F |
| Channel 3 | Waveform | C3 to B8 | 8-15 | 0, C, E8, EC [2] |
| Channel 4 | Noise | Only C5 | 16-31 | C, E8, EC, B, D, F |

*[1] This range is for One-Indexed Trackers (C1 is the lowest-possible note). This is comparable to OpenMPT in default settings. Trackers that are Zero-Indexed by default (C0 is the lowest-possible note) should interpret these Note Ranges a full octave down. This is comparable to MilkyTracker in default settings.*

Using default settings on OpenMPT and MilkyTracker, C3 to B8 in OpenMPT sounds the same as C2 to B7 in MilkyTracker.

*[2] Effects B, D, and F can be also used on Channel 3 if the same row isn't being used to set a note/instrument.*

## Volume Limitations

Currently, volume can only be adjusted by using the `Cxx` effect for each channel.

The Gameboy has 16 unique volume settings for Channels 1, 2 and 4. GBT Player will floor (round-down) the values in a `Cxx` volume effect to multiples of 4.

### Unique Volume Settings for Channels 1, 2 and 4:

`00, 04, 08, 0C, 10, 14, 18, 1C, 20, 24, 28, 2C, 30, 34, 38, 3C`

Any number that's not a multiple of 4 will be rounded-down to one of the above numbers.

**Example:** Entering `C01`, `C02` and `C03` will sound the same as entering `C00`.

**Example:** Entering `C40` will sound the same as entering `C3C`.

Channel 3 is the exception to this with only 4 unique volume settings.

### Unique Volume Settings for Channel 3:

`00, 10, 20, 40`

GBT Player will round `Cxx` effects on Channel 3 to the nearest number listed above.

**Example:** Entering `C30` will round the volume up to `C40`.

## Volume Persistence

In most trackers, if a note is played without a volume command, the note's volume is reset to the maximum. When a .mod file is converted by GBT Player, notes without a volume effect will play at the same volume as the previous `Cxx` effect that the channel read. For example, take this scenario:

```
ModPlug Tracker MOD
|C-502...C40|
|...........|
|...........|
|...........|
|........C..|
|...........|
|E-502......|
```

In the tracker, the E-5 note will resume at full volume after the C00 effect.

In-game, you will not hear the E-5 note. This is because the C00 persists until another `Cxx` effect is set. To make the tracker playback sound identical to the in-game playback, the following must be done:

```
ModPlug Tracker MOD
|C-502...C40|
|...........|
|...........|
|...........|
|........C..|
|...........|
|E-502...C40|
```

Additionally, Channel 3 requires that the instrument and note is set during a volume change for the volume change to have any effect. (Except for `C00`.) For example:

```
ModPlug Tracker MOD
|C-511...C40|
|...........|
|...........|
|...........|
|........C20|
|...........|
|G-511...C40|
```

You will not hear any volume change from the C20 in-game. Add a note and instrument on `C20` to register the volume change.

```
ModPlug Tracker MOD
|C-511...C40|
```

## Instruments

*All numbers listed here are in base-10 unless otherwise noted.*

The pulse channels 1 and 2 have four instrument options:

1. 25% pulse
2. 50% pulse (square wave)
3. 75% pulse (inverted 25% pulse)
4. 12.5% pulse

Instruments 5 through 7 are intentionally left blank.

Channel 3, the wave channel, has 8 instrument options:

8. Buzzy (Source code calls this "random :P")
9. Ringy (useful for SFX)
10. (A) Sync Saw
11. (B) Ring Saw
12. (C) Octave Pulse + Triangle
13. (D) Sawtooth
14. (E) Square
15. (F) Sine

As of GB Studio 1.2.1, GBT Player uses 16 instruments to access pre-determined noise settings - instruments 16 to 32.

Instruments 16 to 23 use Periodic (looped) Noise at various pitches, while instruments 24 to 32 use Pseudorandom noise at various pitches.

The nicknames and descriptions next to these instruments are not official for GBT Player, they are intended to help identify these noise presets at a glance.

Periodic Noise:

16. (10hx) "stutter" - A square plus a pulse at random pulse widths
17. (11hx) "rumble" - The same waveform but faster
18. (12hx) "engine" - The same waveform but even faster
19. (13hx) "low tone" - Sounds like D5
20. (14hx) "undertone" - Sounds like E5 + 50cents
21. (15hx) "middletone" - Sounds like B5 + 50cents
22. (16hx) "overtone" - Sounds like D6 + 50cents
23. (17hx) "high tone" - Sounds like D7

Pseudorandom Noise:

24. (18hx) "earthquake" - A square with a thin pulse at random pulse widths

25. (19hx) "spaceship" - The same as 24 but faster
26. (1Ahx) "ocean" - etc.
27. (1Bhx) "scratch" - etc.
28. (1Chx) "glitch" - A fairly clean white-noise sample, unrelated to other instruments
29. (1Dhx) "volcano" - A pulse with rapidly changing pulse width
30. (1Ehx) "scream" - The same as 29 but faster
31. (1Fhx) "static" - etc.

As of GB Studio 1.2.1 there are no GBT Player-readable instruments beyond 31. (1Fhx)

## Effects

There are two types of effects: Note-effects and Command-effects.

The only restrictions on effects is the Command-effects with Channel 3. It can use them when it's not trying to play a note/set the instrument on the same row.

**Note-effects** (uses bit 3) - All channels can use these effects freely

| Effect | Name | Notes on effect usage |
|---|---|---|
| **0xy** | Arpeggio | Rapidly cycles between 3 notes. `x` and `y` both represent the # of semitones above the note the arpeggio effect is attached to. |
| **Cxx** | Volume | Sets the volume to `xx`. See **Volume Limitations** for more info. |
| **E8x** | Pan | Sets the panning to `x`. `0–3` = Left, `4–B` = Centre, `C–F` = Right. |
| **ECx** | Note cut | Cuts the note after `x` frames. Must be below the `Fxx` speed for the cut to be heard. `EC0` will reset the duty cycle instead of cutting the note. |

**Command-effects** (uses bit 4) - Channel 3 can use these effects if it's not trying to play a note/instrument on the same row.

| Effect | Name | Notes on effect usage |
|---|---|---|
| **Bxx** | Jump | Jump to a specific position in the song, `xx`. |
| **Dxx** | Pattern break | Jumps to the next pattern early, where `xx` is the row it should jump to in the next pattern. Using this on the last pattern will break the song by reading garbage data beyond the song. |

| Effect | Name | Notes on effect usage |
|---|---|---|
| **Fxx** | Set speed | Sets the song speed to `xx`. Valid values are `01` to `1F`. The value represents how many frames should the song wait before moving on to another row. Setting BPM speed has no effect upon conversion. |

For Channel 3 only, the instrument data is too large to allow the 4th bit of a Command effect to occur while it's trying to play a note/set the instrument. Command-effects will ignore new notes on Channel 3 to compensate.

## Speed Table

| Fxx Value (in tracker) | BPM (in tracker) | BPM (in game) |
|---|---|---|
| **F01**[1] | 750 BPM | 900 BPM |
| **F02**[1] | 375 BPM | 450 BPM |
| **F03**[1] | 250 BPM | 300 BPM |
| **F04**[1] | 187.5 BPM | 225 BPM |
| F05 | 150 BPM | 150 BPM |
| F06 | 125 BPM | 128.57 BPM |
| F07 | 107.14 BPM | 112.50 BPM |
| F08 | 93.75 BPM | 100 BPM |
| F09 | 83.33 BPM | 90 BPM |
| F0A | 75 BPM | 81.82 BPM |

This is not a full table, it's just the top few speeds. It's here to highlight some of the speed discrepancies, albeit small to not be very noticeable, with the exception of the values marked with 1.

You might notice that the value of the F effect, when converted to decimal, is just the speed divisor. For instance, F03 divides the BPM by 3 (`750 / 3 = 250`, or `900 / 3 = 300`).

Because of how GB Studio is set up, a 60hz F05 effect, which would result in 180 BPM in-game, is impossible here.

*While not in GB Studio, GBT has a flag called `-speed` that will handle BPM differently, which would require F96 effects for every speed, as it won't handle any internal conversions to get the speed closer. This is the reason why F01 to F04 require F96 in both modes, there's no equivalent for it in tracker speed.*

**1. Values marked with 1 require an additional F96 effect for the song to sound closer in speed when converted, or setting the song BPM to 150.** This F96 effect can be removed once you're done with your song, there won't be any difference as GBT ignores this -- It's only here to set the BPM to something closer to the in-game version.

# Tricks and Tips

## 1. High Speed

By using F01 to F04, you can achieve much higher granularity when it comes to changing volumes and creating sounds of sorts. This means that with a high enough speed, you can create more varied bodies for sounds, with sort-of envelopes, or elaborate effects (like 1 channel echos, which I'll cover here in a moment).

This trick means you're going from drums that sound flimsy and primitive to something more impressive.

Here's an example of a Snare Drum, at speed F02, that might sound good for you.

```
ModPlug Tracker MOD
|C-526...C40
|C-527...C28
|........C20
|........C18
|........C10
|........C08
|........C04
|........C..
(this is on the noise channel)
```

If this is longer than what you need, simply crop it starting from the bottom.

You can also use this for tones and stuff, like short staccato notes or flutes that fade in.

**If you do this, keep in mind the GB Sound hardware has an annoying bug that resets the phase of each waveform on a volume set, meaning you can get scratchy noise in a few emulators and also the real GB.**

## 2. One channel echoes

This works on most speeds. This is useful for when you need a melody on top of some sort of echoing ostinato, or phrase, or whatever.

To illustrate it, I'm going to try illustrating it like this, instead of a tracker layout:

```
A _ B _ C _ E _ G _ E _ C _ B _
Without 1ch Echo

    +-----+ +-----+ +-----+
A _ B a C b E c G e E g C e B c
+-----+ +-----+ +-----+ +-----+
```

Notice how each lowercase letter takes the form of it's 3 step behind louder cousin? That's how the trick works. By having shorter notes that, on each step, has another quieter note that's way behind, you get a cool echoing effect.

I can't explain it very well via text, so I recommend you check out this video by **explod2a03** covering how this trick works with a better example and actual audio: https://www.youtube.com/watch?v=6GI9gngTn_Y

The best way to do this in a tracker is to use a channel you're not using temporarily, copy your note sequence to it, delay it by 3 (or however many you need) rows, then right clicking on the selection and clicking "Amplify...", and setting the amplitude to something lower than 50%.

After that, you should have both channels "alternate". Select the entirety of the channel with the echoes (from top to bottom), go to the channel you want to merge the echoes with, right click, go to "Paste special", then click "Mix paste" (This should have a shortcut, might want to learn it as it can be fairly useful).

### 3. Quick volume envelopes

Are you in a hurry? No problem, this simple trick will create linear envelopes:

1. Select two volume / C values of two separate notes (within the same channel), and everything in between
2. Right click and hover over "Interpolate"
3. Click on "Effect column"
4. You're done!

You might wonder how's it going to sound in-game; well, it'll sound as close as possible. The volumes it can't play it'll just clamp it to the nearest ones it can play.

## Frequently Asked Questions

**Q: Can I use mp3/wav files?**

A: No, but you can use .midi files. If you're looking for an easy way to add music to your game, you can ask the #collaborations channel of the GB Studio Discord or browse the GB Studio Community Assets.

This has limited success, and there are easier options to get music in your game, such as the

**Q: How do I convert a .midi file to .mod?**

A: OpenMPT can open MIDI files and save the result to .mod Some resources on how to do this include a video tutorial as well as Kazy's write-up article pinned in the #music-help section of the GB Studio discord.

**Q: Can I use this .mod file I found online?**

A: It won't sound as intended, but it can be made to sound good-enough with some adjustments. Any `===` notes need to be replaced with the `EC1` effect. All instrument restrictions should apply, and no melodic instruments should be using Channel 4. You may also need to transpose the notes of a channel to account for differently-tuned samples, which you can learn more about in your tracker's documentation.

**Q: How do I stop a note from playing?**

A: `EC1` will mute a channel's note, `C00` will mute the channel until it recieves another `Cxx` effect.

**Q: What do I do if my song sounds completely giltched-out?**

A: It's probably corrupted. It can likely be saved by using OpenMPT and saving it as a different filetype. If you're using **MilkyTracker**, don't press "Save" on a .mod file, always work in a .xm file instead.

**Q: Why is my song speed is faster in-game than it is in the tracker?**

A: If you're using an `Fxx` effect with a value lower than `F05`, add `F96` to the first row of your song. This will not impact your in-game playback speed.

**Q: Can I play back voice clips/sound effects?**

A: Not on GBT Player. Pokemon Yellow's method is unique, and LSDj does not leave much processing power for games to be played while it's running.

**Q: Can I use a different tool to write my music?**

A: If the tool can natively export to .mod, try it!

**Q: Why is my song playing glitched sounds when it tries to loop?**

A: `D00` is a problematic effect, try using `Bxx` instead. If you're already using `Bxx`, make sure the `xx` number does not go above the number of pattern-slots in your song. A song's first pattern is always in slot 00.

**Q: Why do some notes in OpenMPT appear red and sound higher/lower than they're supposed to sound?**

A: Go over to the "General" tab that's under the New File, Open and Save buttons. Click the big button next to the "Name" field that says "MOD (ProTracker), 4 channels". Once there, disable both **ProTracker 1/2 Mode (MOD)** and **Amiga Frequency Limits.** This is a thing because the format here is meant to be used with the Amiga line of computers (that's where it was made), which has frequency limits.

**Q: Why does my song start out with garbage noise?**

A: If your song doesn't start using the first two channels, add a note to their first row with a `C00` effect on each.

**Q: Can I play sound effects?**

A: Yes, with limitations. View the next page of the documentation for more information. Playing sound effects will not interrupt the song being played by GBT Player.

## Tips

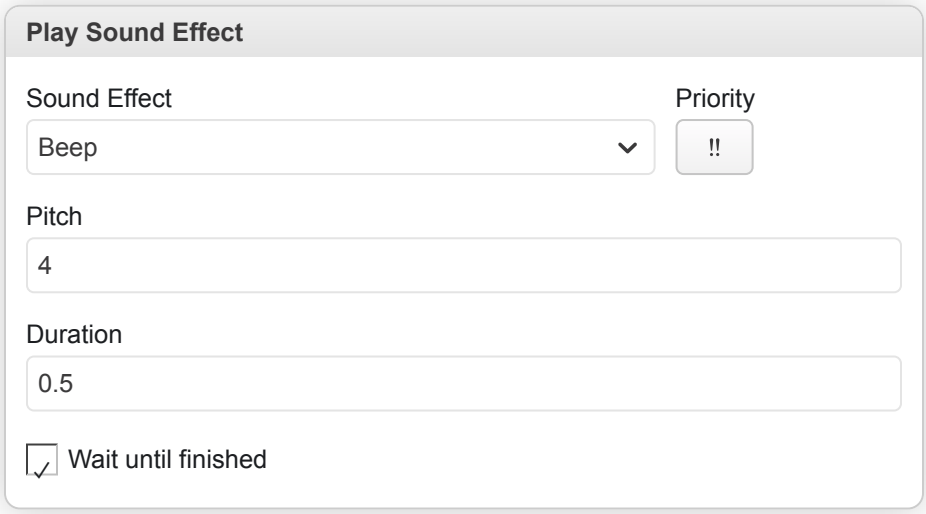
- **Make sure you save frequently and also back-up your files.** This is important in anything that you do and it's worth mentioning here.
- **If you're stuck, please ask for help in the Discord server, in `#music-help`.** There's usually a few handful of people who are willing to help out at most times.
- **Frequently try out your music in your game.** Things don't sound 1:1, and the built in preview just plays the .mod file rather than building the music and previewing that.
- **Keep it simple!** Don't jump into this, trying to emulate what several artists have done with LSDj or whatever other tools, you'll just get stuck.
- **Don't be afraid of failure.** I get this is kind of an unfitting tip, but it's important. Your first song won't be good, and that's okay. You'll fail, sure, but you'll also gain knowledge on what you might've done wrong, or how you want to go on about with your next endeavor.
- **OpenMPT has a manual to help you get started.** Here's a link, give it a read if you're stuck (or just ask for help)

- **Give the GBT Player documentation a read.**

# Sound Effects

Sound effects can be added to your game using the Play Sound Effect event in your *Actor*, *Trigger* or *Scene* scripts.

You can choose from playing a beep with a given pitch, a tone with a given frequency or cymbal crash.




You can also place the following file types into `assets/sounds` which then become available:

- **.wav** WAV audio file, preferably very short in length (3.64 seconds is about the limit!) and 8-bit mono (though GB Studio will attempt to convert files not in this format).
- **.vgm** VGM audio file (Game Boy format only).
- **.sav** FX HAMMER sound effects.

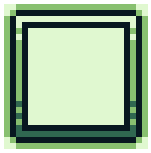
# UI Elements

Your project contains a number of files in `assets/ui` with fixed file names that define parts of your game's user interface. Editing these files allows you to change the default font, set the window frame and modify the selection cursor.

If you remove any of the files in the ui folder they will be replaced with the default assets the next time you build your game allowing you to revert any unwanted changes.

## frame.png

The game engine uses 9-slice scaling of this image to create the frame around text boxes. Editing this image will allow you to change the frame design or set it to a solid color.



## cursor.png

This image is used as a selection cursor when showing multiple choice options in your game.

## Requirements

UI PNGs must only contain the following four colors:

- #071821
- #306850
- #86c06c
- #e0f8cf

Download the GB Studio Palette Swatches for:

Adobe Photoshop

Aseprite

## Fonts

Fonts are stored in `assets/fonts`, see Settings for more information.

## Emotes

Emotes are stored in `assets/emotes` and must be defined as `16px x 16px` sized `.png` files following the same color

requirements used for creating spritesheets. You can display an emote by using the `Show Emote Bubble` event in a script.

## Avatars

Avatars are stored in `assets/avatars` and must be defined as `16px x 16px` sized `.png` files following the same color requirements used for creating spritesheets. You can display an avatar in a `Display Text` event by clicking `Add Avatar` within the event.

# Scripting Events

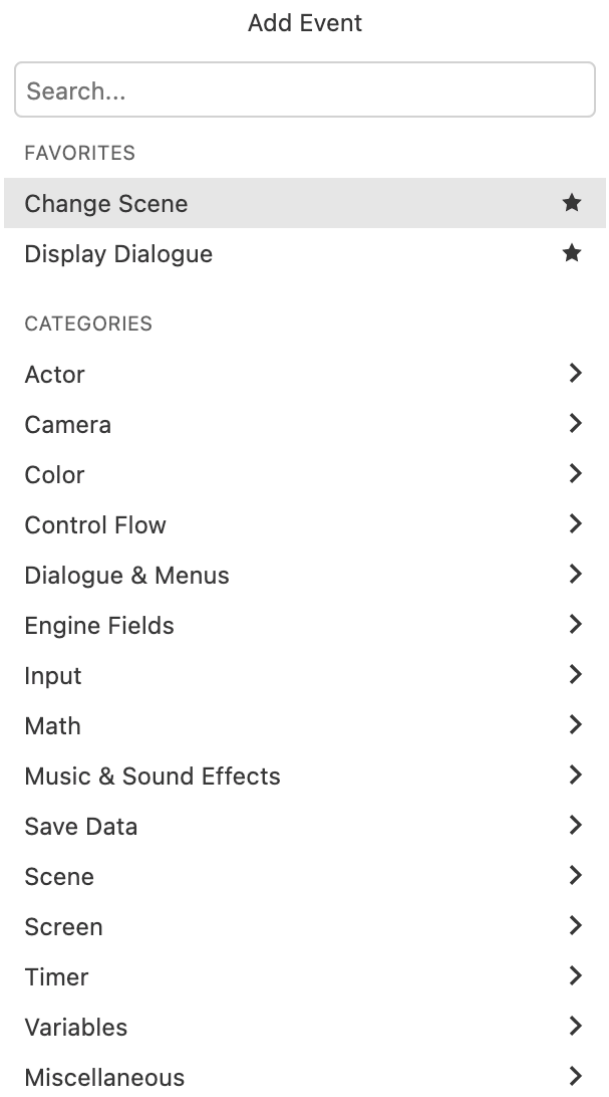
Scripting events allow you to control parts of your game based on interactions from the player. They can be used to connect scenes together, change variables, give dialogue to characters, and more.

Scripts can be added to scenes, actors, or triggers. Selecting one of these objects will update the *World Editor* to show the script of the selected object in the *Editor Sidebar*.

To start building a script, select an object and click the *Add Event button* in the *Editor Sidebar* to open the event menu. Select an event to add it to the script. The topmost event is the first event to be run for that script.

## Adding Events

After clicking the *Add Event* button, a menu will appear to choose the event to add. If you start typing you can filter this list or you can click through the menu find what you're looking for. Click an event or press the *Enter* key to add the highlighted event to your script.



## Favourite Events

You can choose a number of events to be favourites, causing them to appear at the top of the *Add Event Menu*. To favourite an event, hover over the menu item and click the *Star* button or press the `Tab` key.

## Copy and Paste Events

To copy an event, click the down arrow next to an event. All scripts have this same down arrow for copying/pasting. Clicking the down arrow on another event allows you to paste the clipboard event either before or after the selected one. You also have the option to paste the values from the first event into the second.

As a shortcut for pasting, you can press the *Alt* key to turn all *Add Event* buttons into *Paste Event* buttons.

## Types of Scripts

There are multiple script tabs to choose from the *Editor Sidebar* depending on which object you have currently selected.

### Scene Scripts

These scripts can be accessed in the *Editor Sidebar* by selecting a scene in your project.

- **On Init:** This script will run once at the beginning of the Scene. The Scene On Init script is always run after the On Init script for Actors in the Scene.
- **On Player Hit:** This script runs when the player is hit by an actor belonging to a collision group.

### Actor Scripts

These scripts can be accessed in the *Editor Sidebar* by clicking an Actor in your project.

- **On Init:** This script will run once at the beginning of the Scene. Actors in a Scene will always run their On Init script before their Scene's On Init script.

- **On Interact:** Standing the Player next to an Actor and pressing the **A** button will cause the Player to "interact" with the Actor. Interacting with an Actor will begin this script. In Shoot 'Em Up scenes, interacting can additionally be done by colliding with the Actor.

  This script is often used for dialogue, using the "Text: Display Dialogue" event.

  Enabling a collision group for an actor will convert this script to *On Hit: Player*, which looks for Player collision rather than Player interaction. This behaviour is identical to *On Interact* in Shoot 'Em Up scenes.

- **On Hit:** This script runs when the Actor is hit by another Actor or Projectile belonging to a collision group.

- **On Update:** This script is run once every frame, and can only be added to non-player Actors.

### Trigger Scripts

These scripts can be accessed in the *Editor Sidebar* by clicking a Trigger in your project.

- **On Enter:** This script runs when the player collides with the trigger.

- **On Leave:** This script runs when a player that was previously colliding leaves the trigger.

# Event Glossary

## Actor

Activate Actor

## Camera

Camera Lock To Player

## Color

If Color Mode Is Available

## Control Flow

Call Script

## Dialogue & Menus

Display Dialogue

## Engine Fields

Engine Field Update

## Input

Attach Script To Button

## Math

Evaluate Math Expression

## Miscellaneous

Comment

## Music & Sound Effects

Play Music Track

## Save Data

Game Data Load

## Scene

Change Scene

## Screen

Fade Screen In

## Timer

Attach Timer Script

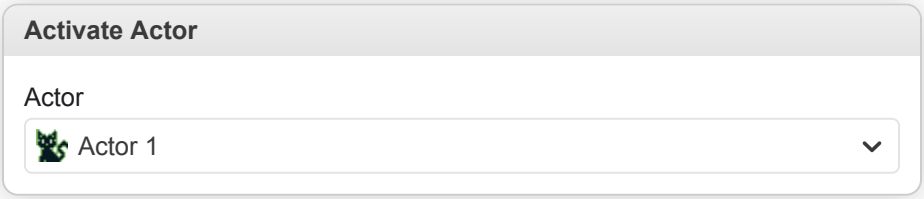
## Variables

Evaluate Math Expression

# Actor

## Activate Actor

Activate an actor, causing it to become visible (if not also hidden) and for its OnUpdate script to start.




- **Actor**: The actor you want to activate.

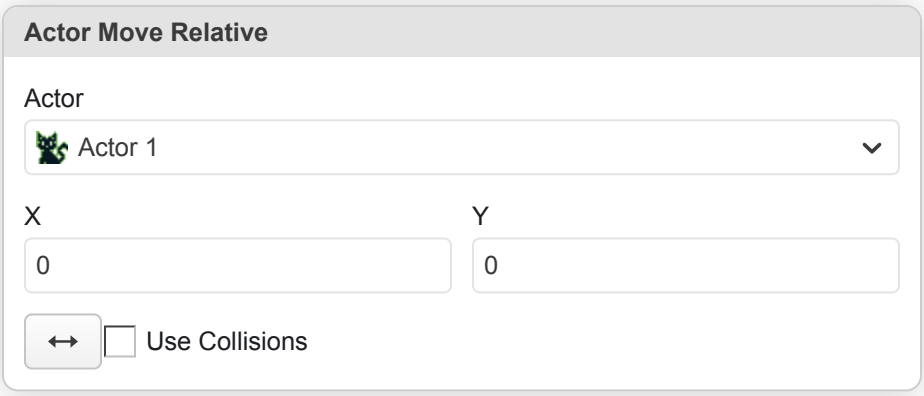
## Actor Move Cancel

Cancel any currently running "Actor Move" events affecting this actor. Causes the actor to stop in its current location.


- **Actor**: The actor you want to cancel movement for.

## Actor Move Relative

Move an actor relative to its current position.

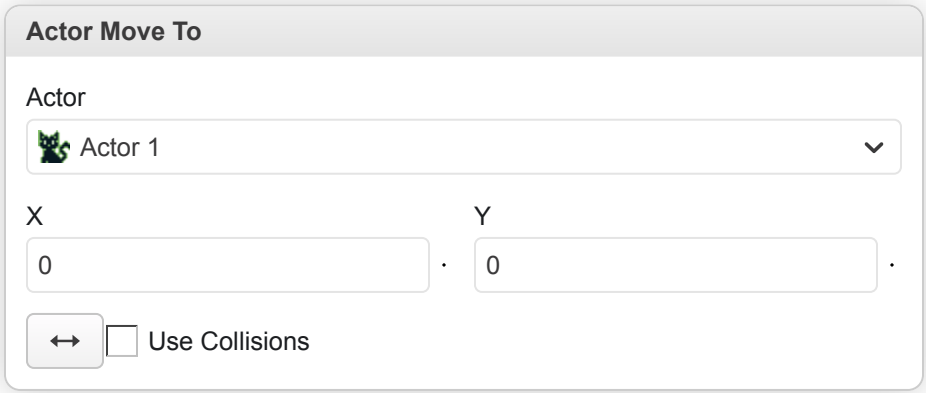



- **Actor**: The actor you want to move.
- **X**: The horizontal offset relative to the current position.
- **Y**: The vertical offset relative to the current position.
- **Movement Type**: Choose if should move in horizontal/vertical axis first or if it should move diagonally to destination.

- **Use Collisions**: Set if collisions with both scene and actors should be taken into account while moving.

## Actor Move To

Move an actor to a new position.




- **Actor**: The actor you want to move.
- **X**: The horizontal position.
- **Y**: The vertical position.
- **Movement Type**: Choose if should move in horizontal/vertical axis first or if it should move diagonally to destination.
- **Use Collisions**: Set if collisions with both scene and actors should be taken into account while moving.

## Deactivate Actor

Deactivate an actor, causing it to act as if it had gone offscreen. It will become invisible and its OnUpdate script will be stopped.


- **Actor**: The actor you want to deactivate.

## Hide Actor

Hide an actor, causing it to become invisible. Its OnUpdate script will continue to run while hidden.


- **Actor**: The actor you want to hide.

## Hide All Sprites

Disable rendering of sprite layer causing all sprites to become hidden until sprite rendering is reenabled.

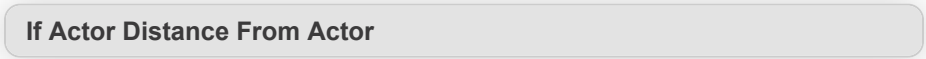

## If Actor At Position

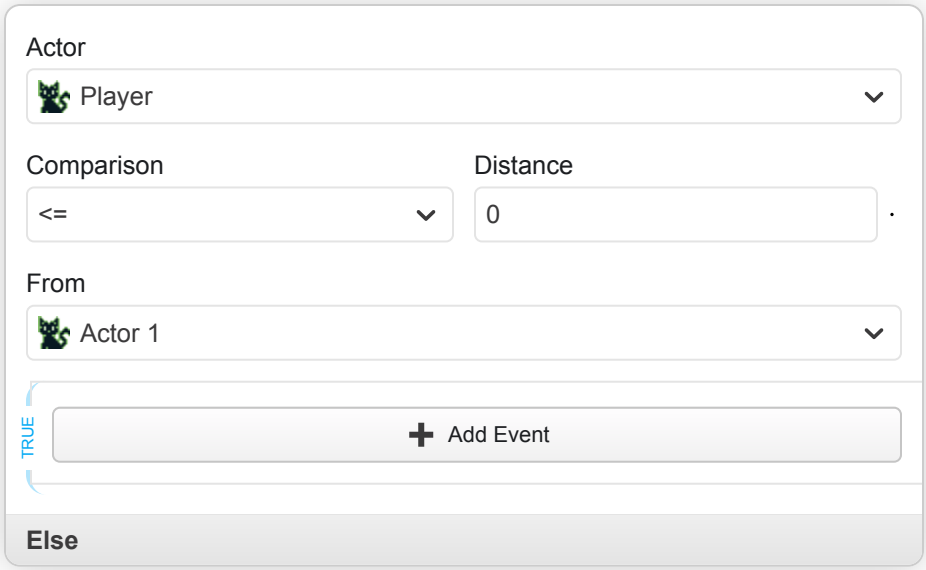
Conditionally run part of the script if an actor is at a specified position.


- **Actor**: The actor you want to check.
- **X**: The horizontal position.
- **Y**: The vertical position.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Actor Distance From Actor

Conditionally run part of the script if an actor is within a certain distance of another actor.

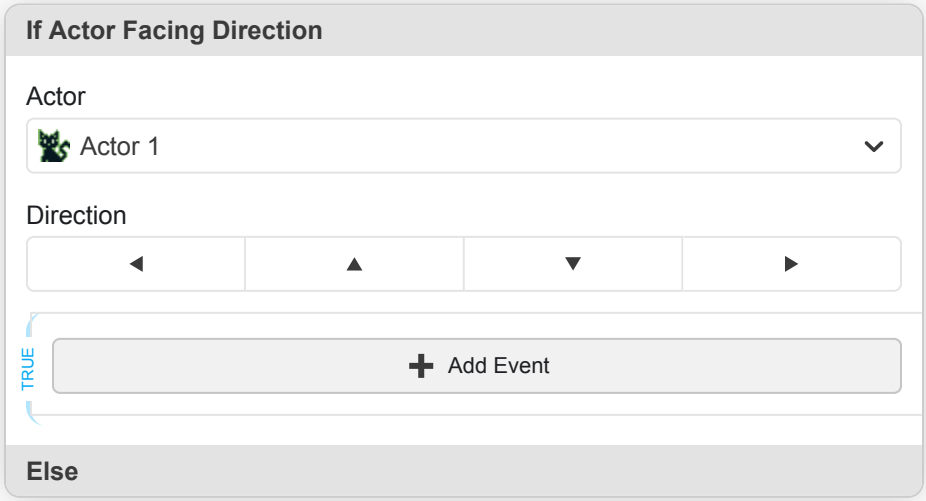





- **Actor**: The actor you want to check.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **Distance**: The distance value.
- **From**: The actor to compare distance with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Actor Facing Direction

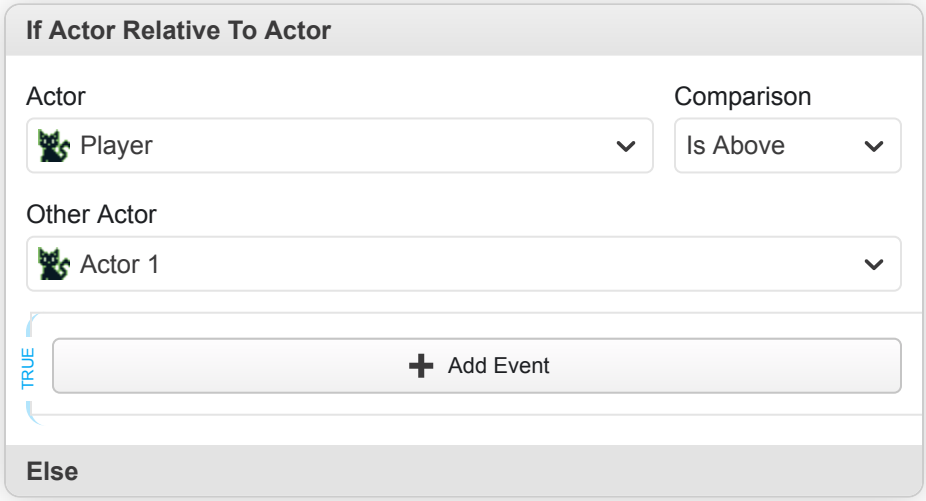
Conditionally run part of the script if an actor is facing in a specified direction.




- **Actor**: The actor you want to check.
- **Direction**: The actor direction.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Actor Relative To Actor

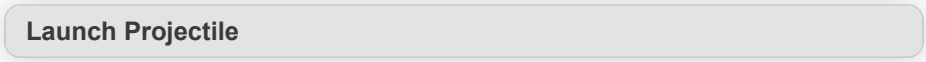
Conditionally run part of the script based on the position of one actor relative to another.

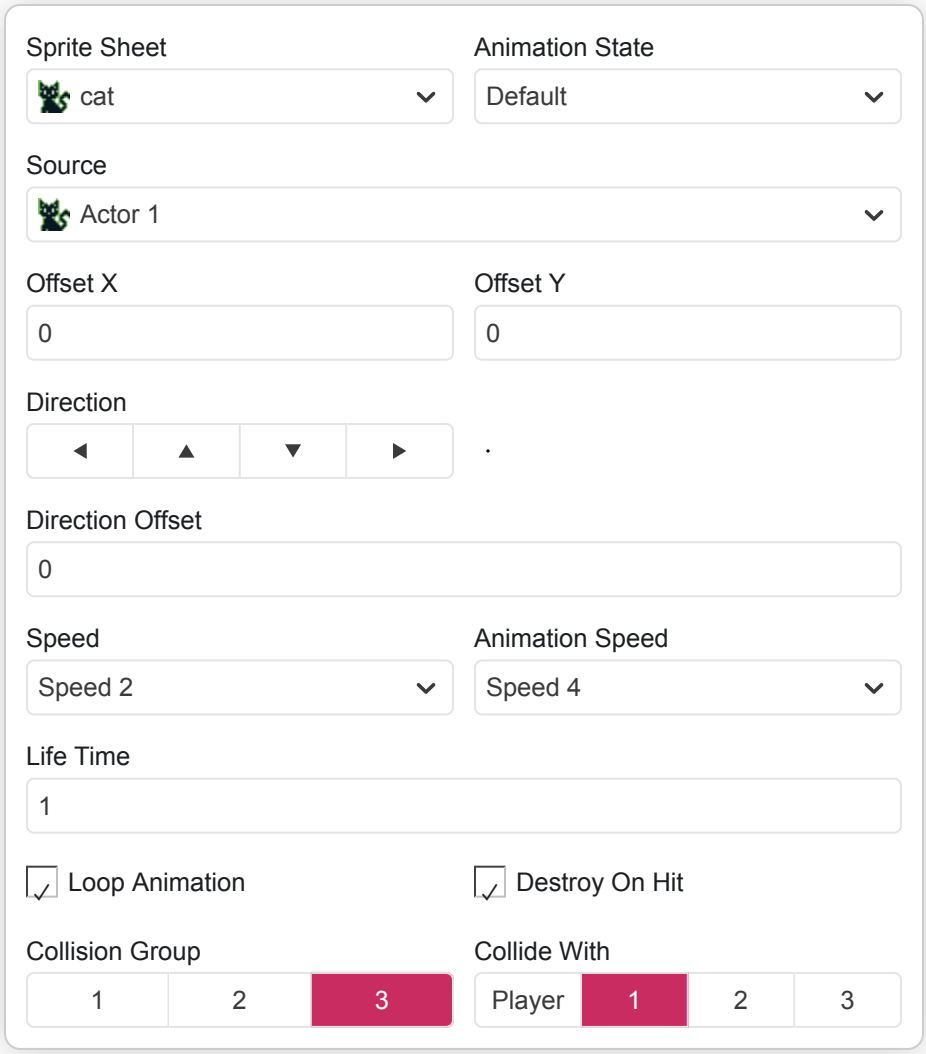



- **Actor**: The actor you want to check.
- **Comparison**: The relative position comparison to use e.g. 'Is Above' or 'Is Below'.
- **Other Actor**: The actor to compare position with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## Launch Projectile

Launch a projectile from an actor in a specified direction. When a project collides with other actors it will trigger their OnHit scripts.



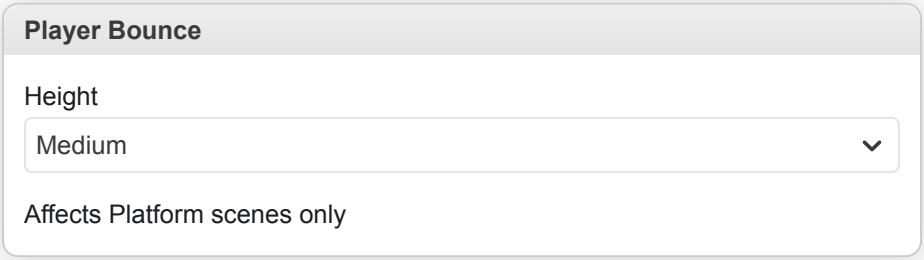



- **Sprite Sheet**: The sprite to use for rendering the projectile.
- **Animation State**: The sprite animation state to use.
- **Source**: The actor to launch the projectile from.
- **Offset X**: The horizontal offset from the source actors position to start launching the projectile.
- **Offset Y**: The vertical offset from the source actors position to start launching the projectile.
- **Direction**: The direction to launch the projectile. Can either be a fixed direction or based on an actor's current direction.
- **Angle**: The angle to launch the projectile.
- **Direction Offset**: The distance the projectile should move from launch position in its launch direction before becoming visible.
- **Speed**: The movement speed.
- **Animation Speed**: The animation speed.
- **Life Time**: The amount of time in seconds that the projectile will live for.
- **Loop Animation**: Set if animation should loop.
- **Destroy On Hit**: Set if the projectile should be destroyed after its first collision.
- **Collision Group**: The collision group that should be used when registering collisions with actors.

- **Collide With**: The groups of actors that will be checked for collisions. e.g. If it should pass through any actors but the player set this field to just 'Player'.

## Player Bounce

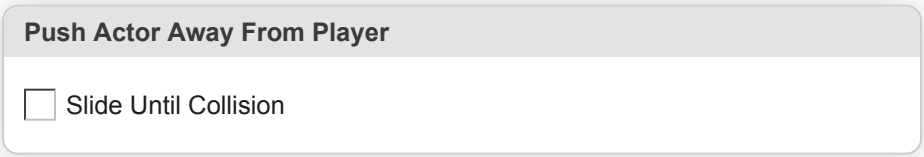
In platform scenes causes the player to bounce upwards by setting the player's velocity Y value.




- **Height**: How high the player should bounce.

## Push Actor Away From Player

Causes the specified actor to be moved in the direction that the player is currently facing. Useful for creating block puzzles.




- **Slide Until Collision**: Set to make the actor continue to move until a collision with another actor or the scene occurs.

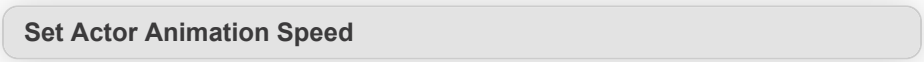
## Set Actor Animation Frame

Set an actor's animation to a specified frame value.


- **Actor**: The actor you want to update.
- **Animation Frame**: The animation frame value.

## Set Actor Animation Speed

Set the animation speed of an actor to a new value.




- **Actor**: The actor you want to update.
- **Animation Speed**: The animation speed.

## Set Actor Animation State

Change the sprite animation state for a specified actor.


- **Actor**: The actor you want to update.
- **Animation State**: The sprite animation state to use.

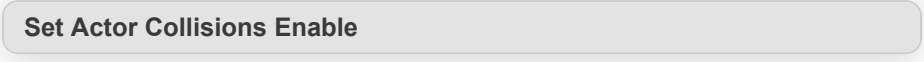
## Set Actor Collisions Disable

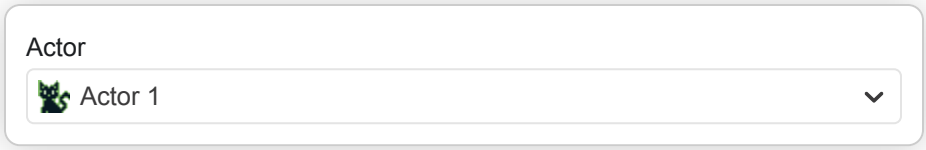
Disable all collision checks for an actor allowing the player and all other actor's to pass through it while moving.


- **Actor**: The actor you want to update.

## Set Actor Collisions Enable

Re-enable collisions for an actor causing it to become solid again if collisions had previously been disabled.

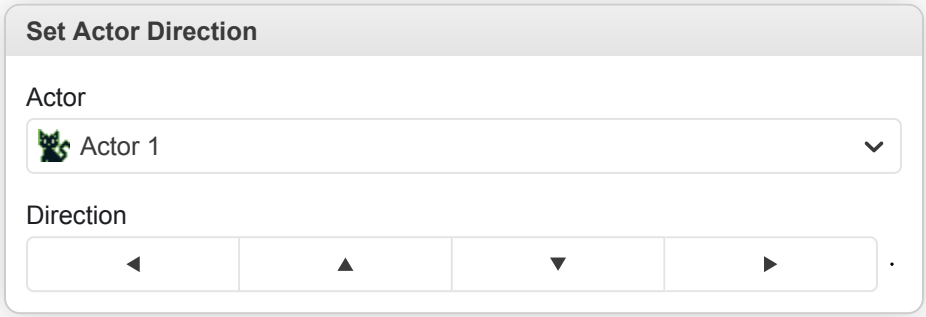





- **Actor**: The actor you want to update.

## Set Actor Direction

Change the direction that an actor is currently facing.




- **Actor**: The actor you want to update.
- **Direction**: The actor direction.

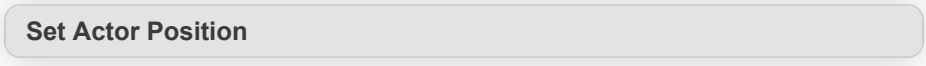
## Set Actor Movement Speed

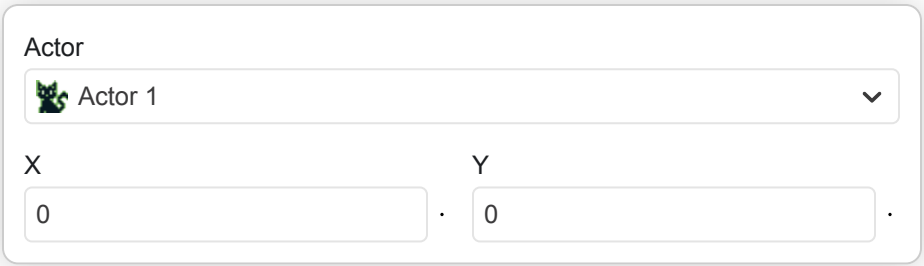
Set the movement speed of an actor to a new value.


- **Actor**: The actor you want to update.
- **Speed**: The movement speed.

## Set Actor Position

Set the position of an actor, causing it to instantly move to the new location.






- **Actor**: The actor you want to update.
- **X**: The horizontal position.
- **Y**: The vertical position.

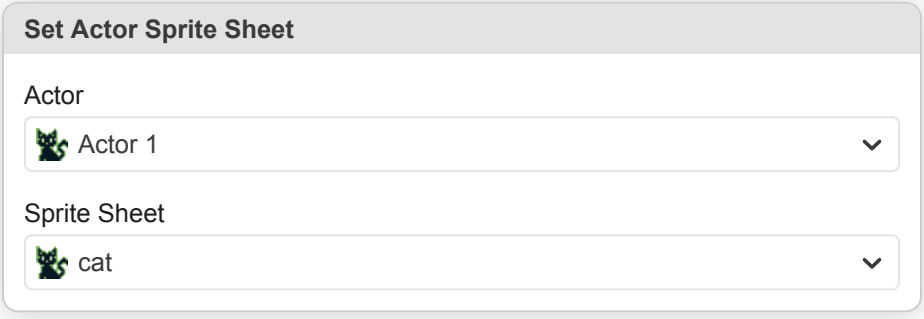
## Set Actor Relative Position

Set the position of an actor relative to it's previous position, causing it to instantly move to the new location.


- **Actor**: The actor you want to update.
- **X**: The horizontal offset relative to the current position.
- **Y**: The vertical offset relative to the current position.

## Set Actor Sprite Sheet

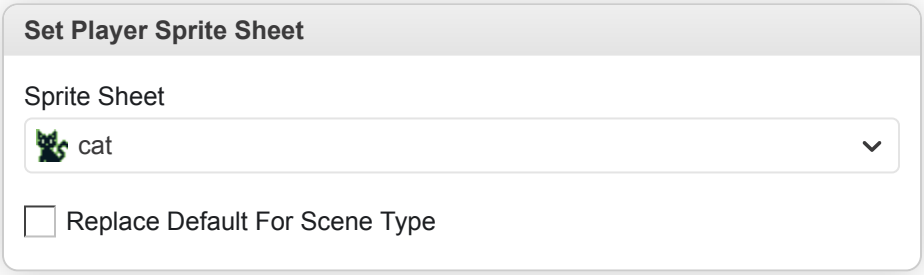
Set the sprite that should be used to render an actor.




- **Actor**: The actor you want to update.
- **Sprite Sheet**: The sprite to use for rendering the actor.

## Set Player Sprite Sheet

Set the sprite that should be used to render the player.




- **Sprite Sheet**: The sprite to use for rendering the player.
- **Replace Default For Scene Type**: Causes this sprite to override the default for all scenes of the current type. i.e. If you are currently in a platformer scene, all other platformer scenes using the default sprite will now load using this replacement sprite automatically instead.

## Show Actor

Unhide a previously hidden actor.

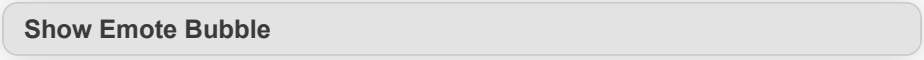

- **Actor**: The actor you want to show.

## Show All Sprites

Re-enable rendering of the sprite layer if previously disabled.


## Show Emote Bubble

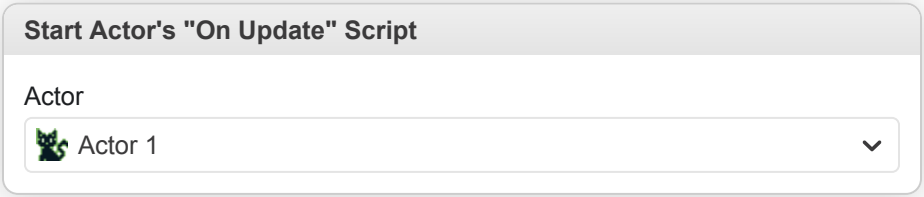
Show an emote image above a specified actor. The image will be positioned centrally above the actor's collision bounding box.




- **Actor**: The actor to display an emote image above.
- **Emote**: The emote image to display.

## Start Actor's "On Update" Script

Start an actors OnUpdate script if it is not currently running. If the actor is currently offscreen its script may become deactivated causing the script to stop running again, to prevent this set the 'Keep Running While Offscreen' setting for the actor's OnUpdate script.




- **Actor**: The actor you want to update.

## Stop Actor's "On Update" Script

Stop an actors OnUpdate script if it was currently running.


- **Actor**: The actor you want to update.

## Store Actor Direction In Variable

Store the current direction of an actor within a variable.

- **Actor**: The actor you want to check.
- **Variable**: The variable to use for the direction.

## Store Actor Position In Variables

Store the current position of an actor within two variables, one to store the horizontal position and another to store the vertical position.


- **Actor**: The actor you want to check.
- **X**: The variable to use for the horizontal position.
- **Y**: The variable to use for the vertical position.

# Camera

## Camera Lock To Player

Move the camera back to centering on the player, locking into position when the player moves. Optionally allows locking to follow player in only horizontal or vertical axis.


- **Speed**: The movement speed, use 'Instant' to immediately move to the new location.
- **Lock Axis**: Set if either horizontal axis, vertical axis or both should be locked.

## Camera Move To

Move the camera to a new position.


- **X**: The horizontal position.
- **Y**: The vertical position.
- **Speed**: The movement speed, use 'Instant' to immediately move to the new location.

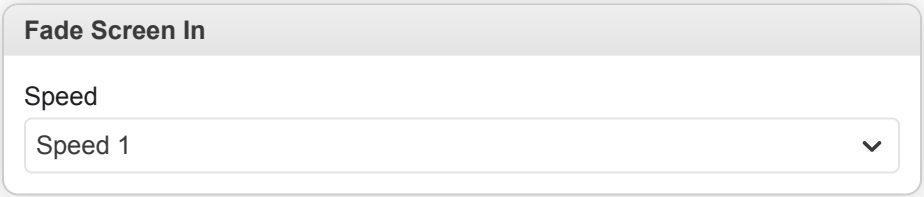
## Camera Shake

Shake the camera for a period of time.


- **Duration**: The length of time to shake camera for in seconds or frames.
- **Movement Type**: Choose if camera should shake only in horizontal or vertical axis or if should shake in both directions.

## Fade Screen In

Fade the scene from a blank screen.




- **Speed**: The speed of the fade animation.

## Fade Screen Out

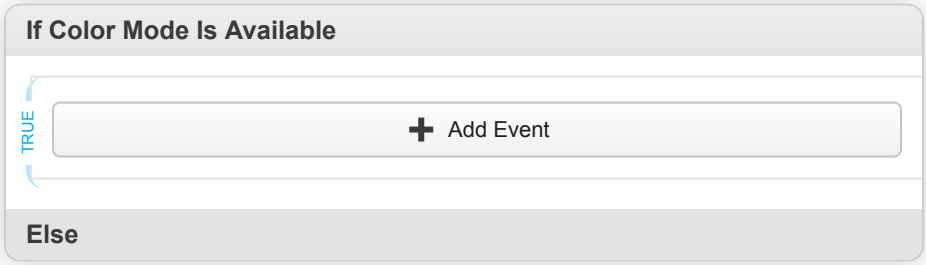
Fade the scene to a blank screen.


- **Speed**: The speed of the fade animation.

# Color

## If Color Mode Is Available

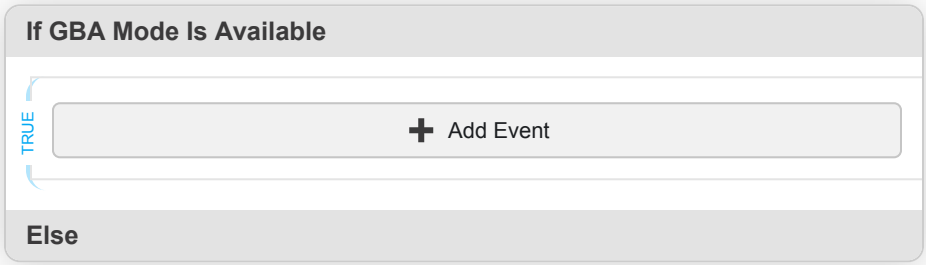
Conditionally run part of the script if the game is being played on a device or emulator that supports color games.




- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If GBA Mode Is Available

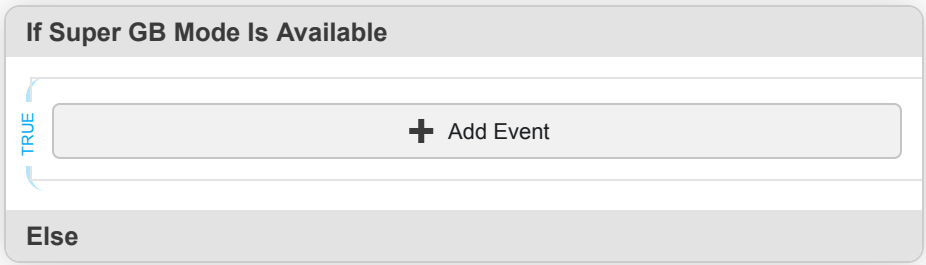
Conditionally run part of the script if the game is being played on a device or emulator that supports GBA games.




- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Super GB Mode Is Available

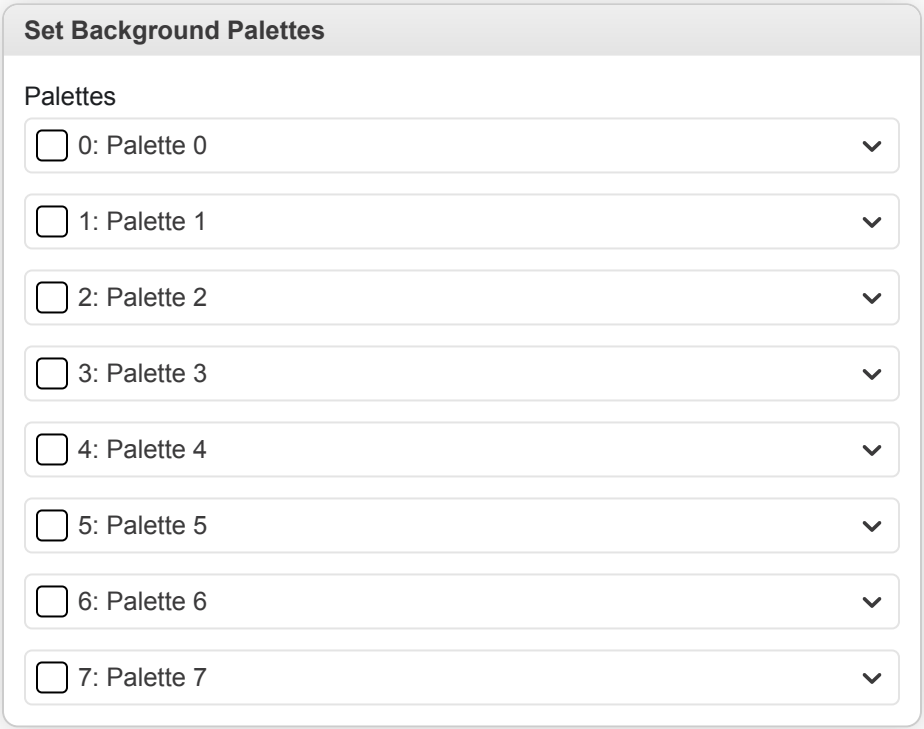
Conditionally run part of the script if the game is being played on a device or emulator that supports Super GB games.




- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## Set Background Palettes

Replace some or all of the current scene's background palettes.




- **Palettes**: The new palettes to use.

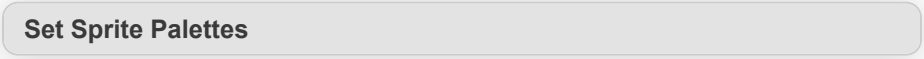
## Set Emote Palette

Replace the palette used for emotes (sprite palette #8).


- **Palette**: The new palette to use.

## Set Sprite Palettes

Replace some or all of the current scene's sprite palettes.




- **Palettes**: The new palettes to use.

## Set UI Palette

Replace the palette used for the UI (background palette #8).


- **Palette**: The new palette to use.

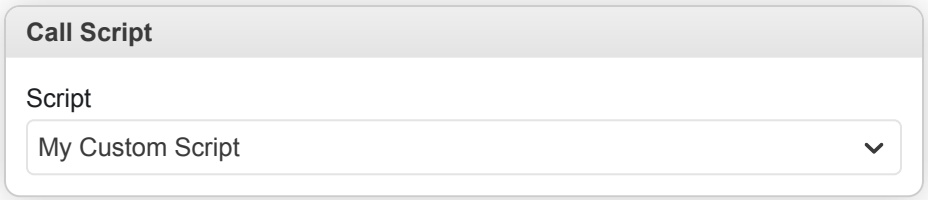
# Control Flow

## Call Script

Call one of your custom scripts. Once you have chosen a script you will be able to hook up any parameters required.

**References**

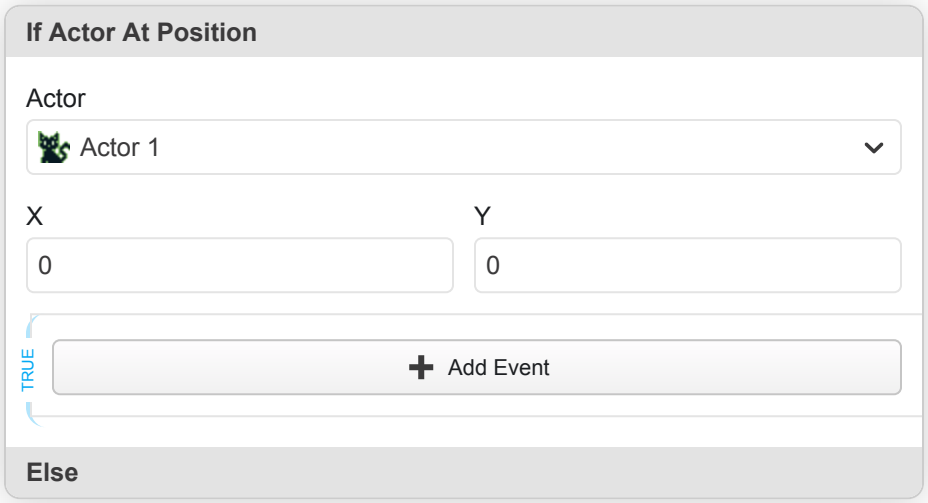
/docs/scripting/custom-scripts




- **Script**: The script to run.

## If Actor At Position

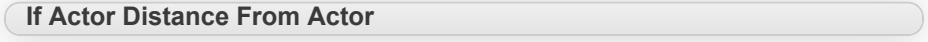
Conditionally run part of the script if an actor is at a specified position.

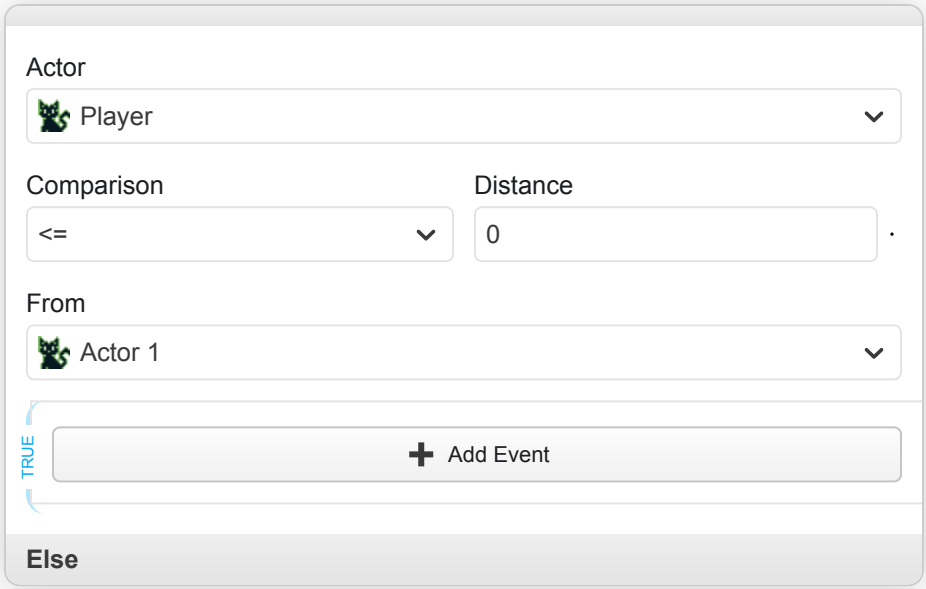



- **Actor**: The actor you want to check.
- **X**: The horizontal position.
- **Y**: The vertical position.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Actor Distance From Actor

Conditionally run part of the script if an actor is within a certain distance of another actor.

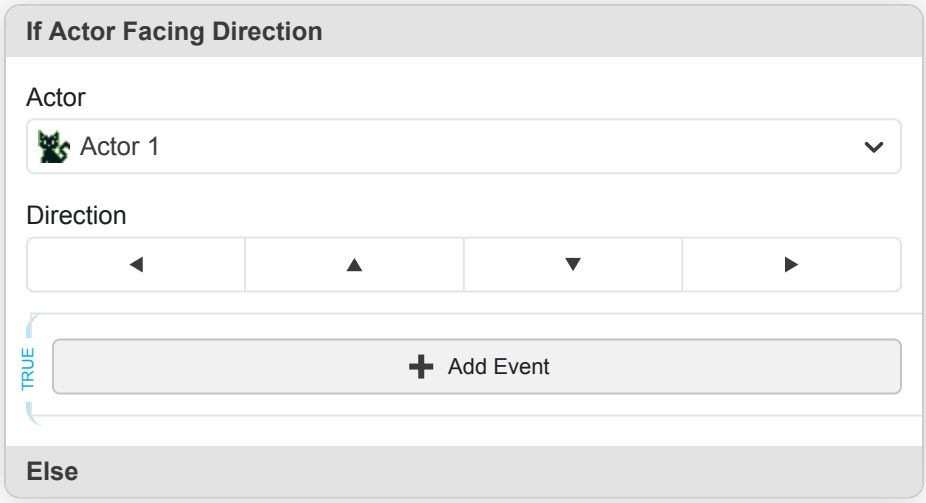





- **Actor**: The actor you want to check.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **Distance**: The distance value.
- **From**: The actor to compare distance with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Actor Facing Direction

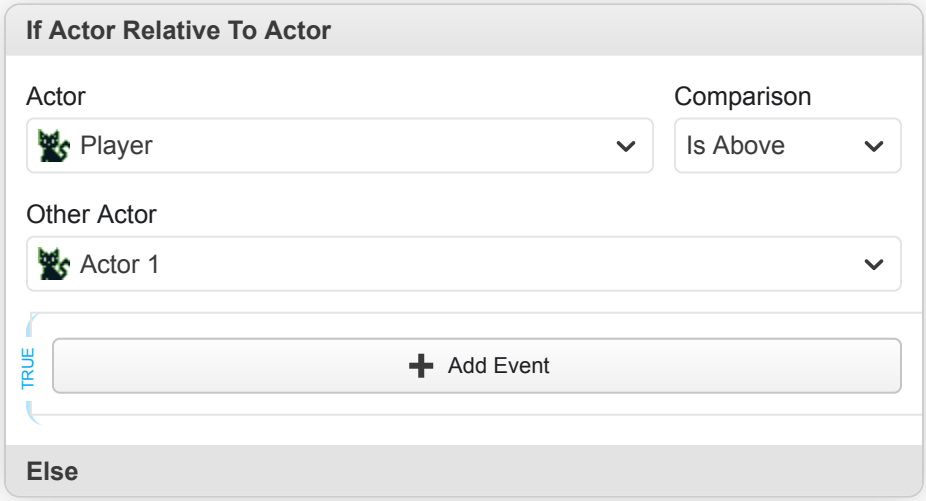
Conditionally run part of the script if an actor is facing in a specified direction.




- **Actor**: The actor you want to check.
- **Direction**: The actor direction.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Actor Relative To Actor

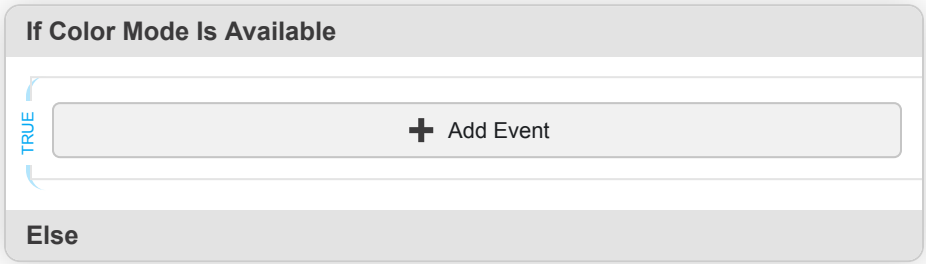
Conditionally run part of the script based on the position of one actor relative to another.




- **Actor**: The actor you want to check.
- **Comparison**: The relative position comparison to use e.g. 'Is Above' or 'Is Below'.
- **Other Actor**: The actor to compare position with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Color Mode Is Available

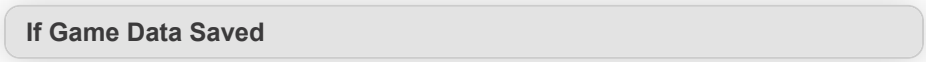
Conditionally run part of the script if the game is being played on a device or emulator that supports color games.

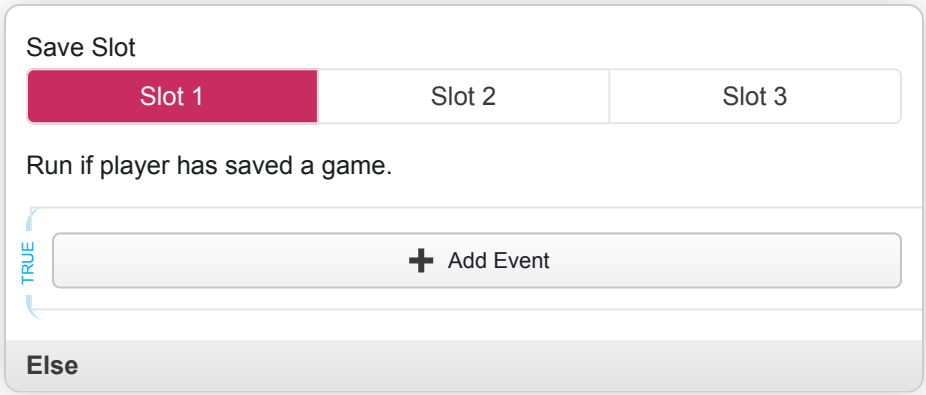



- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Game Data Saved

Conditionally run part of the script if save data is present within the specified save slot.

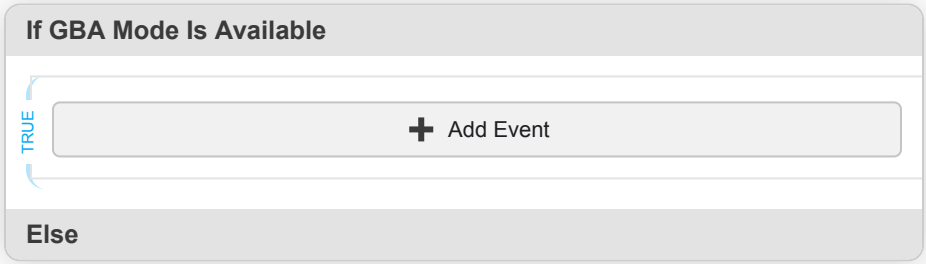





- **Save Slot**: The save slot to use.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If GBA Mode Is Available

Conditionally run part of the script if the game is being played on a device or emulator that supports GBA games.



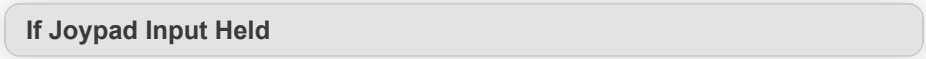

- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

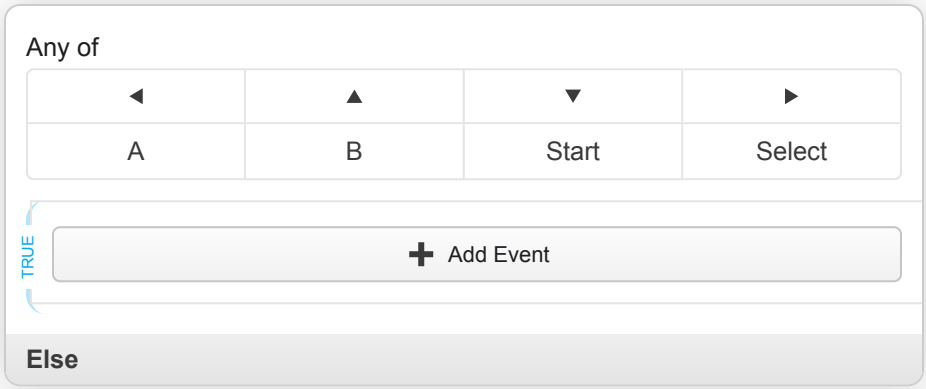
## If Joypad Input Held

Conditionally run part of the script if the specified joypad button is currently pressed. Will not wait for user input and will only execute once, if you wish to run a script every time a button is pressed use Attach Script To Button instead.

**References**

/docs/scripting/script-glossary/input#attach-script-to-button





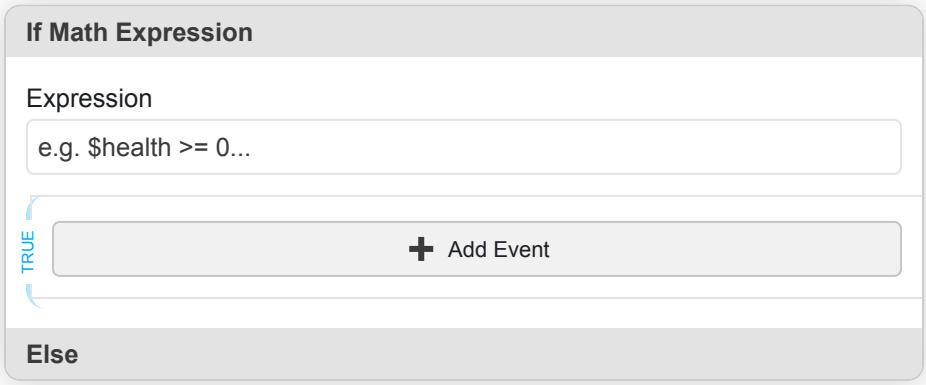

- **Any of**: The input buttons to check.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Math Expression

Conditionally execute part of the script if the specified math expression evaluates to true.

**References**

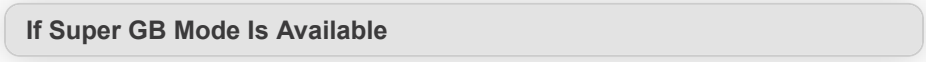
/docs/scripting/math-expressions

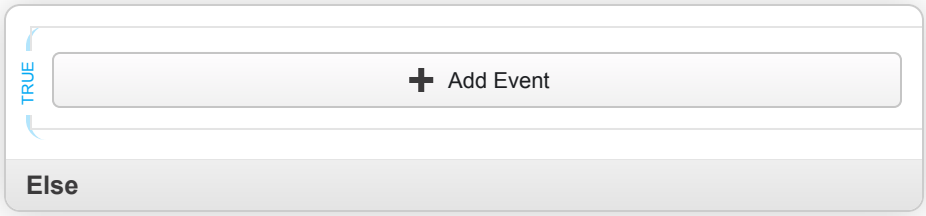



- **Expression**: The expression to evaluate.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Super GB Mode Is Available

Conditionally run part of the script if the game is being played on a device or emulator that supports Super GB games.

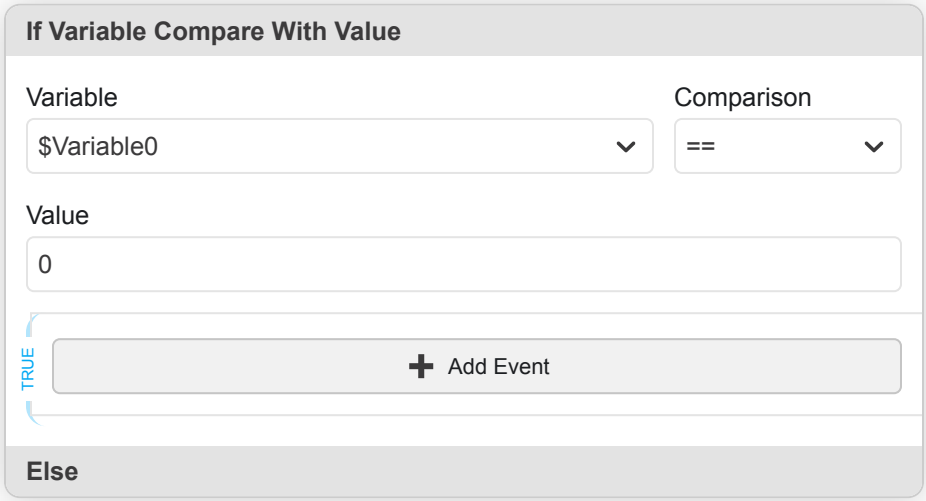





- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Variable Compare With Value

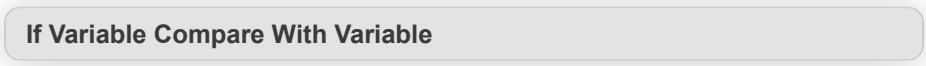
Conditionally run part of the script based on the value of a variable compared with a value.

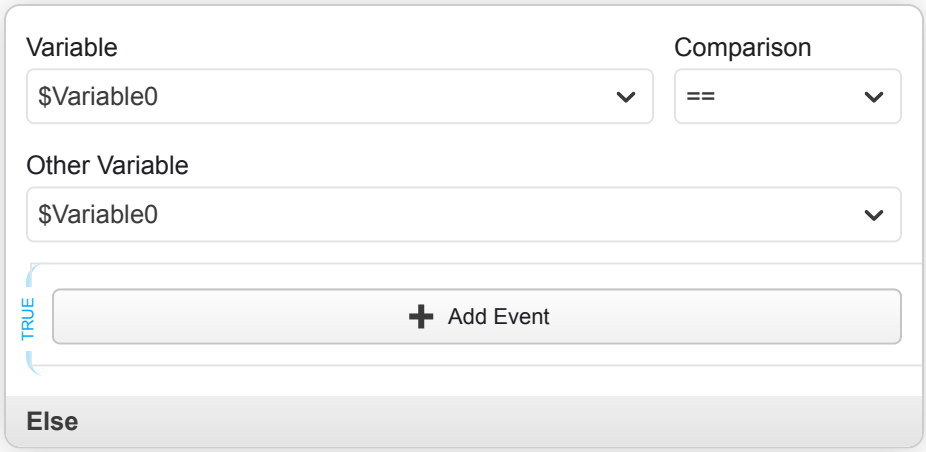



- **Variable**: The variable to use.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **Value**: The value to compare with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Variable Compare With Variable

Conditionally run part of the script based on the value of a variable compared with another variable.

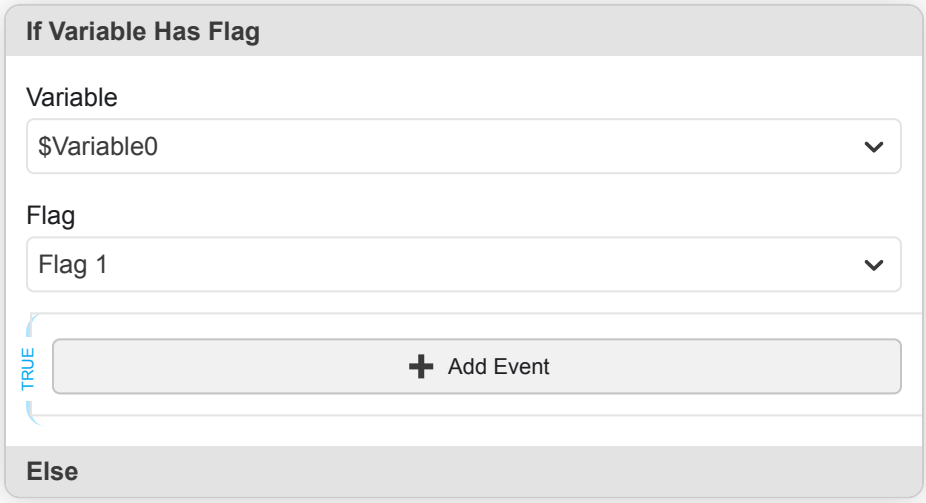





- **Variable**: The variable to use.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **Other Variable**: The variable to compare with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Variable Has Flag

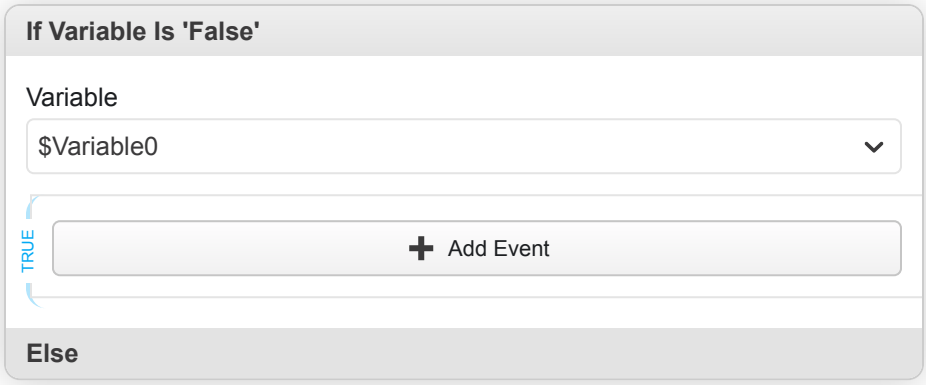
Conditionally run part of the script if the specified variable has the chosen flag set as true.




- **Variable**: The variable to use.
- **Flag**: The flag to check.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Variable Is 'False'

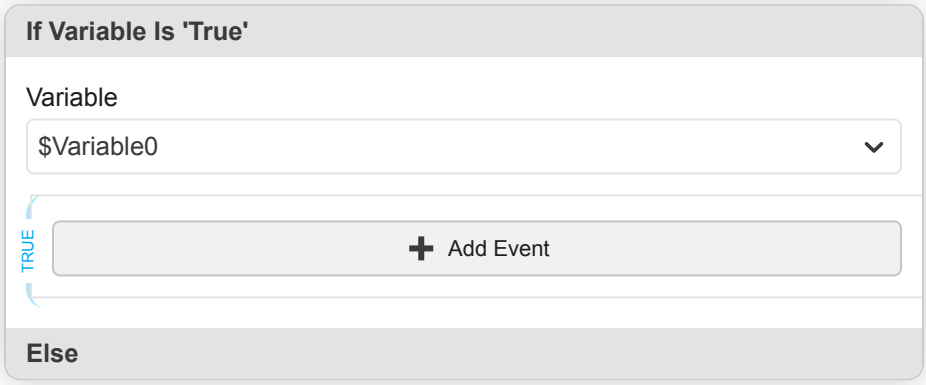
Conditionally run part of the script if the specified variable is set to false.




- **Variable**: The variable to use.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Variable Is 'True'

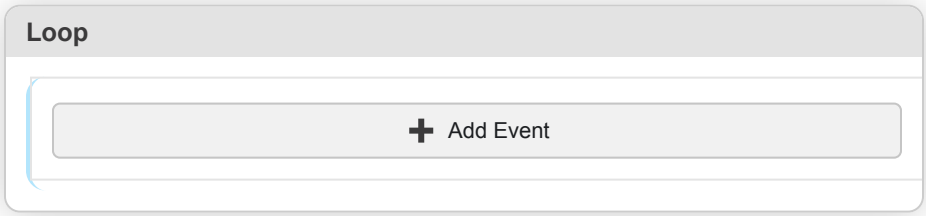
Conditionally run part of the script if the specified variable is set to true.




- **Variable**: The variable to use.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

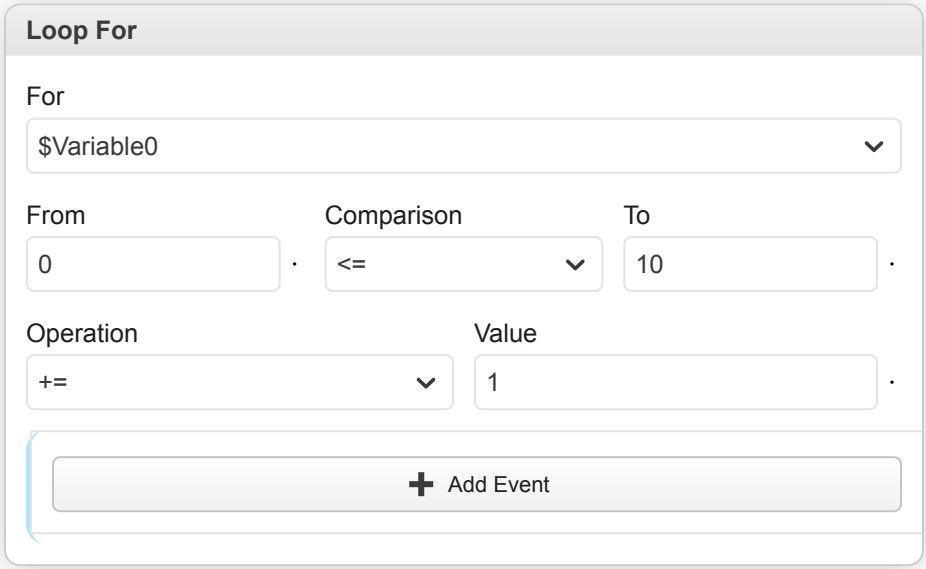
## Loop

Run part of the script in a loop forever. Remember to break out of the loop otherwise the player will become stuck at this point. You can use a Stop Script or Change Scene event to stop the loop.



## Loop For

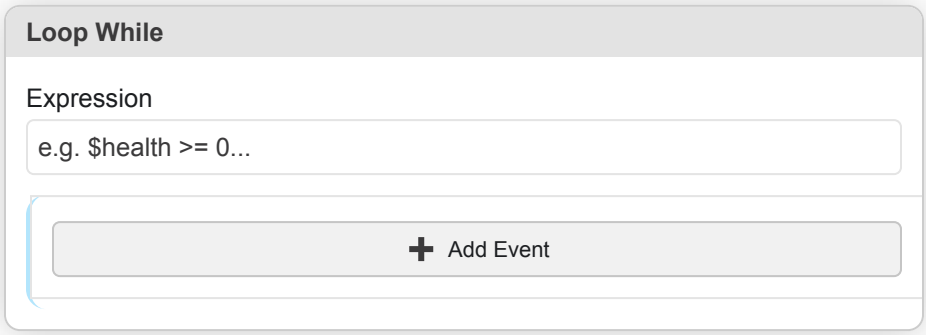
Run part of the script in a loop while a counter variable is within a specified range. On each loop the counter variable is modified before the next check.




- **For**: The variable to use.
- **From**: The initial value of the counter variable.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **To**: The end value of the counter variable.
- **Operation**: The operation to use for combining a value with the counter variable after each loop.
- **Value**: The value to combine with the counter variable after each loop.

## Loop While

Run part of the script in a loop while an expression is true.




- **Expression**: The expression to evaluate.

## Stop Script

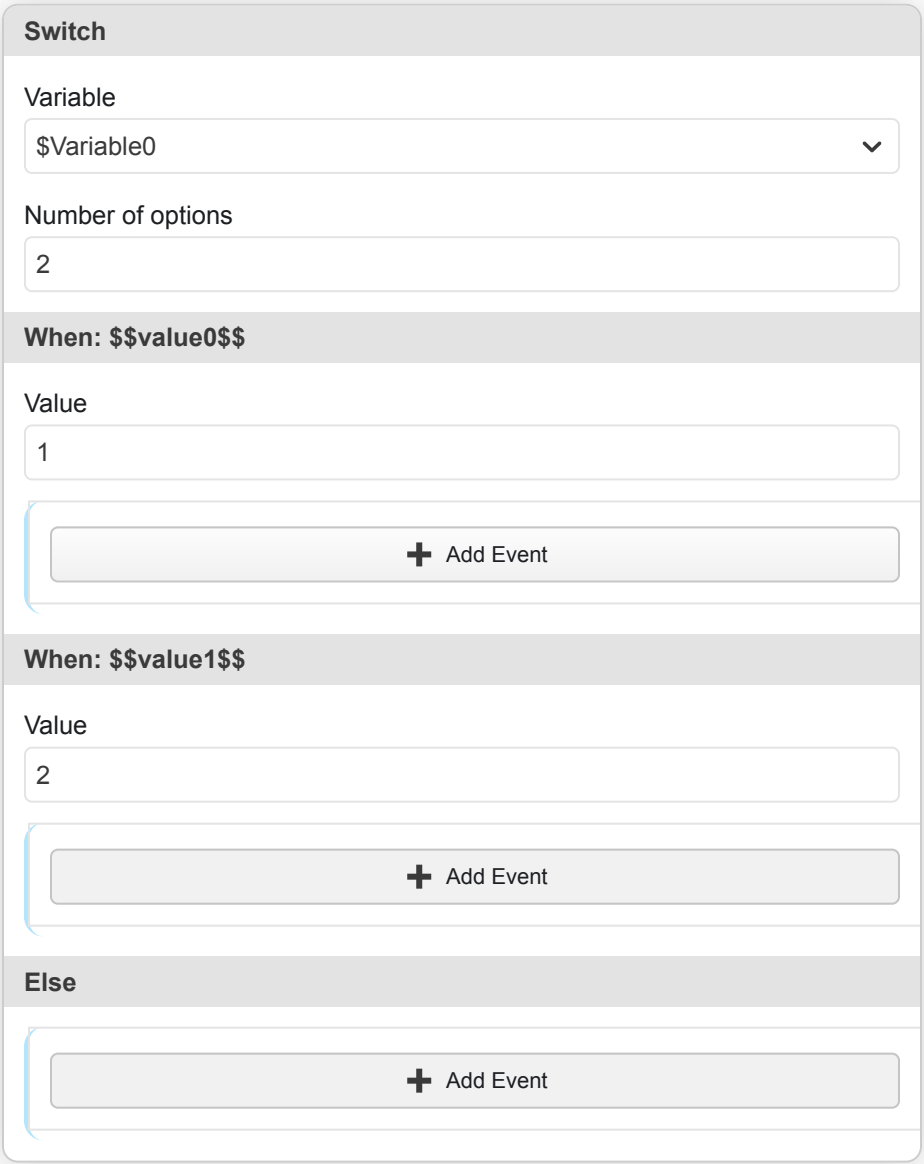
Stops the current script from running.

**Stop Script**

Stops current script from running.

## Switch

Conditionally run from multiple options depending on the value of the specified variable. First choose how many options you want to compare the variable against, then set the values to compare and what scripts to execute when the value is matched.

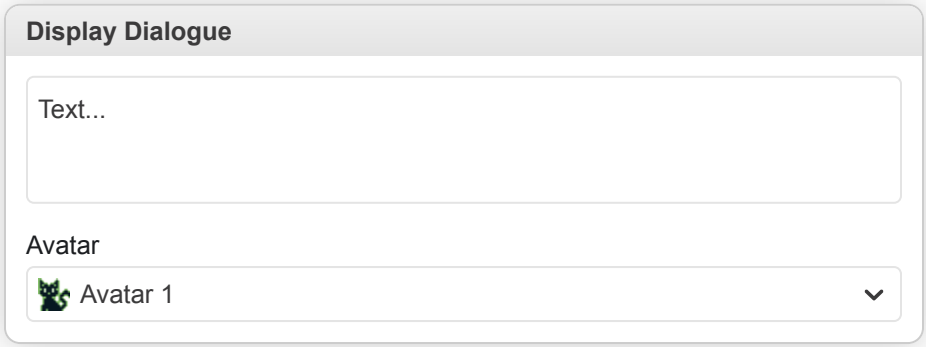



- **Variable**: The variable to use.
- **Number of options**: The number of options required.
- **Value**: The value to compare the variable with for running this branch of the script.

# Dialogue & Menus

## Display Dialogue

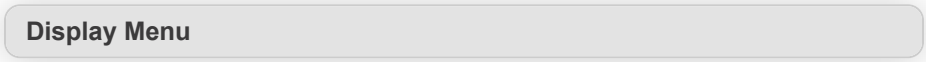
Show a dialogue box at the bottom of the game screen. When text is shown the dialogue box will slide up from the bottom of the screen and will slide down after it has been shown.

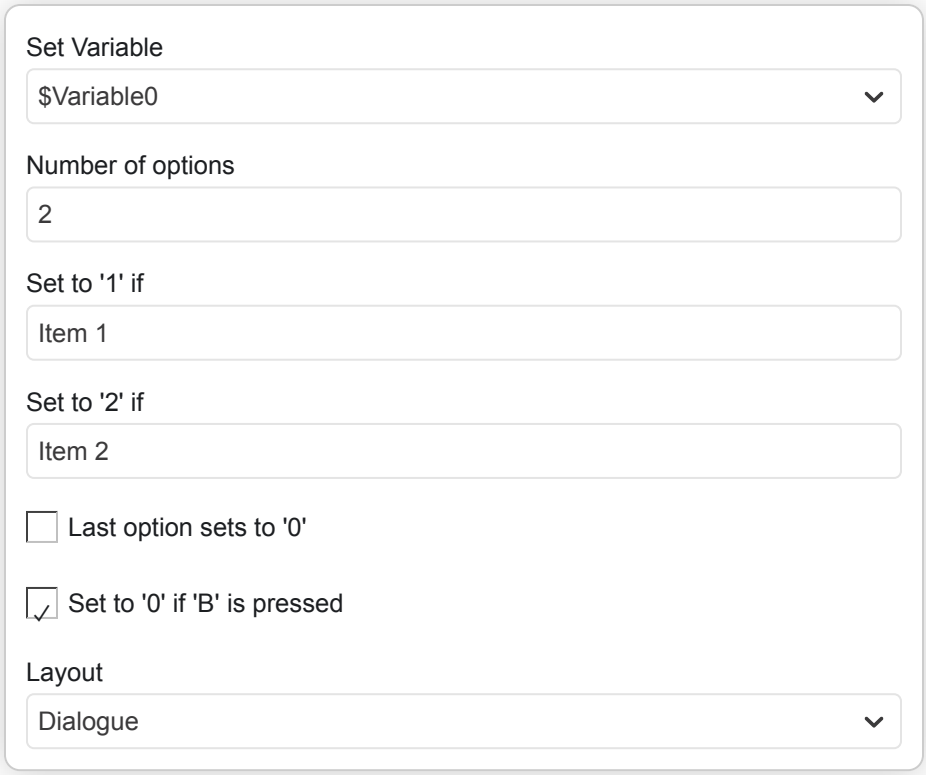



- **Avatar**: The avatar image to optionally display on the left hand side of the dialogue box.

## Display Menu

Display a menu of multiple options and set the specified variable to the value of the chosen option. Multiple layouts are provided, 'Menu' displays as a single column on the right hand side of the game screen and 'Dialogue' displays a full width dialogue box with two columns. You can optionally set the 'B' button to close the menu setting the variable to '0' and can also make the last menu item return '0' when selected.






- **Set Variable**: The variable to use.
- **Number of options**: The number of options required.
- **Set to '1' if**: The menu item text label which when selected will set variable to '1'.
- **Set to '2' if**: The menu item text label which when selected will set variable to '2'.
- **Last option sets to '0'**: Set if last menu item should cause variable to become '0' when selected.
- **Set to '0' if 'B' is pressed**: Set if pressing 'B' should cause menu to close and variable to become '0'.
- **Layout**: Set the layout style of the menu.

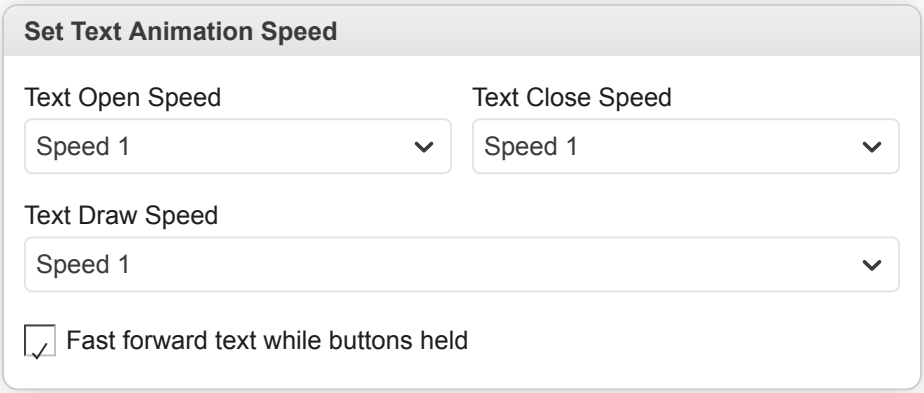
## Display Multiple Choice

Present two options to player allowing them to make a choice, will set the specified variable to true if the first option is chosen and to false if the second option is chosen.

- **Set Variable**: The variable to use.
- **Set to 'True' if**: The menu item text label which when selected will set variable to 'true'.
- **Set to 'False' if**: The menu item text label which when selected will set variable to 'false'.

## Set Text Animation Speed

Set the speed that dialogue boxes appear and disappear and how fast text appears within the box.




- **Text Open Speed**: The speed that the text and menu dialogue boxes scroll on to the screen.
- **Text Close Speed**: The speed that the text and menu dialogue boxes scroll off of the screen.
- **Text Draw Speed**: The speed that characters are drawn into the dialogue boxes.
- **Fast forward text while buttons held**: Allow skipping through animation of text if joypad buttons are pressed.

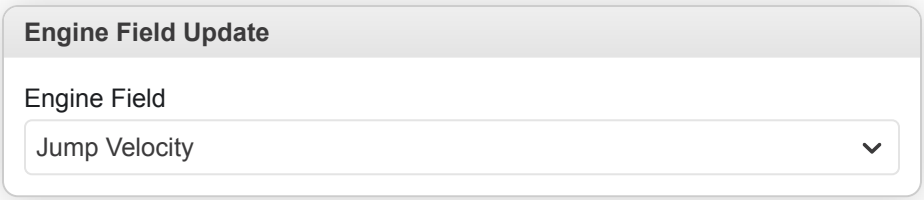
# Engine Fields

## Engine Field Update

Change the value of an Engine Field.

**References**

/docs/settings/#engine-settings




- **Engine Field**: The engine field to update.

## Store Engine Field In Variable

Store the value of an Engine Field in a variable.

**References**

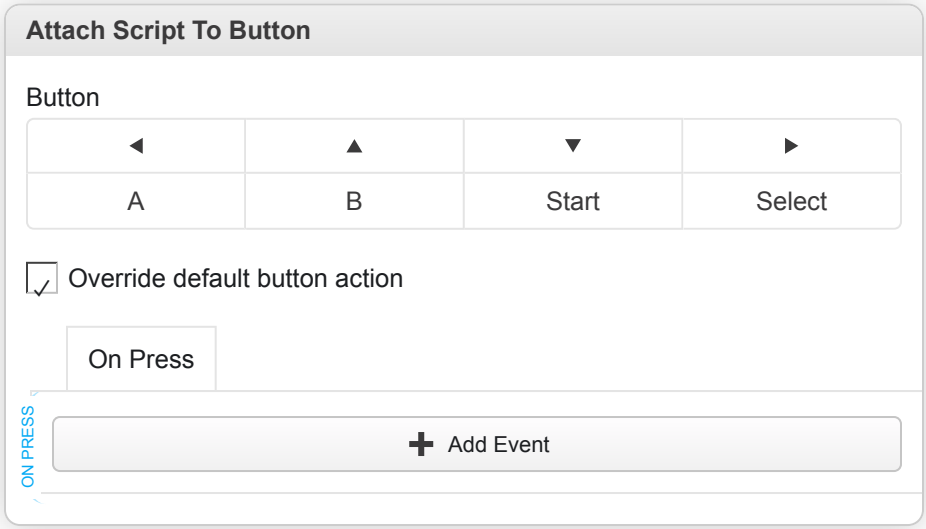
/docs/settings/#engine-settings


- **Engine Field**: The engine field to read the value of.

# Input

## Attach Script To Button

Run the specified script any time a joypad button is pressed.



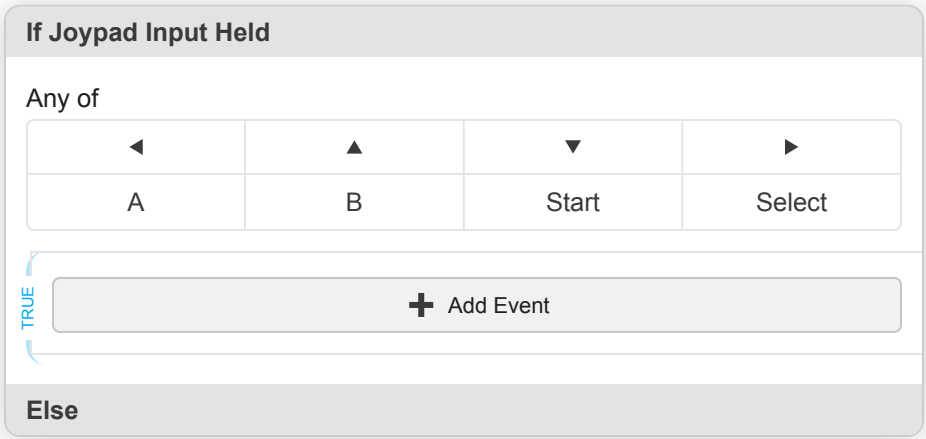

- **Button**: The joypad button to check.
- **Override default button action**: Set if the script should replace the default game action for the specified button.
- **On Press**: The script to run when the button is pressed.

## If Joypad Input Held

Conditionally run part of the script if the specified joypad button is currently pressed. Will not wait for user input and will only execute once, if you wish to run a script every time a button is pressed use Attach Script To Button instead.

**References**

/docs/scripting/script-glossary/input#attach-script-to-button

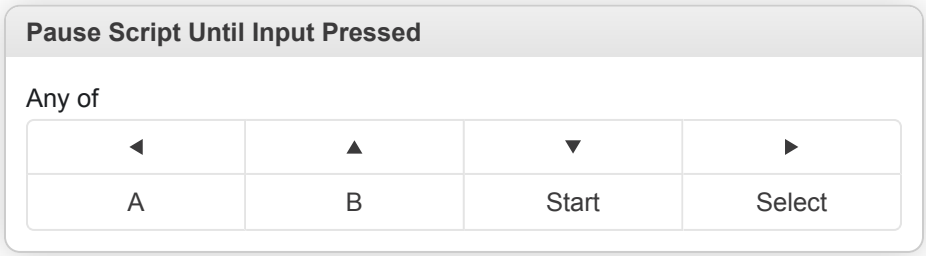



- **Any of**: The input buttons to check.

- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## Pause Script Until Input Pressed

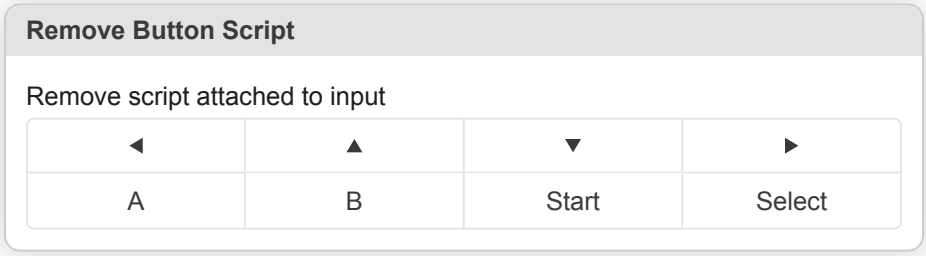
Pauses the script until one of the specified joypad buttons are pressed.




- **Any of**: The input buttons to check.

## Remove Button Script

Remove an attached script from a joypad button restoring the default functionality of the button.




- **Remove script attached to input**: The joypad button to remove the attached script from.

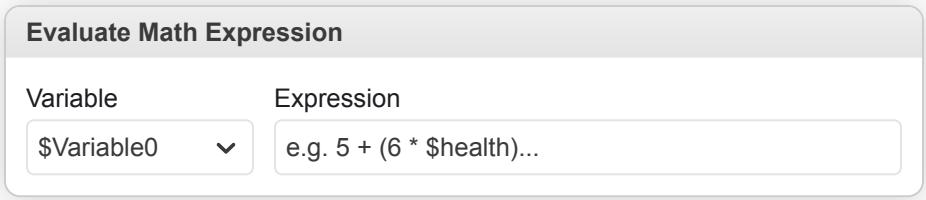
# Math

## Evaluate Math Expression

Set a variable to the result of evaluating a math expression.

**References**

/docs/scripting/math-expressions



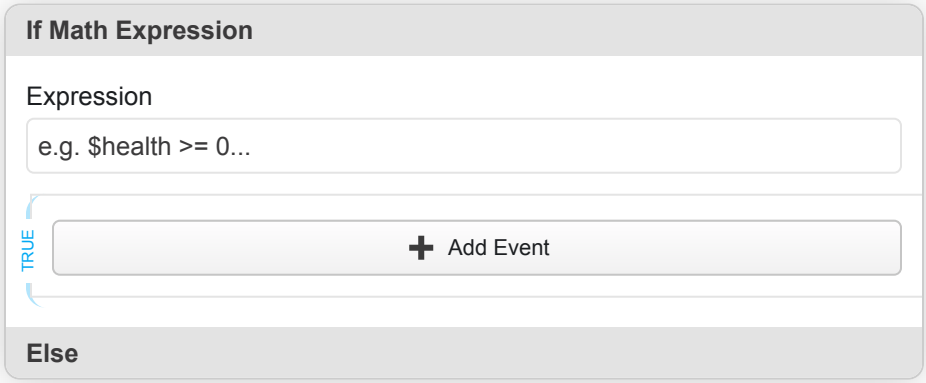

- **Variable**: The variable to use.
- **Expression**: The expression to evaluate.

## If Math Expression

Conditionally execute part of the script if the specified math expression evaluates to true.

**References**

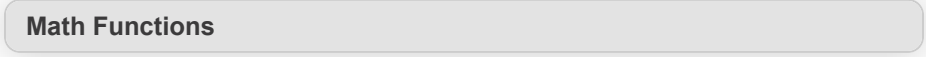
/docs/scripting/math-expressions




- **Expression**: The expression to evaluate.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## Math Functions

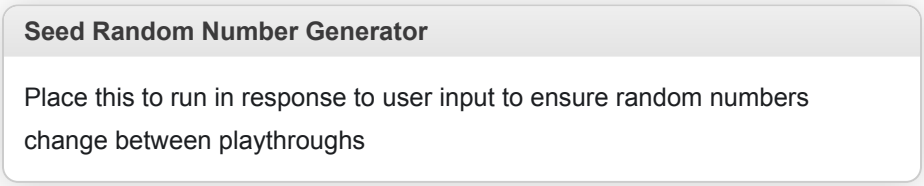
Allows you to perform various maths functions on a variable to add/subtract/multiply/divide/modulus a value/variable/random number.




- **Variable**: The variable to use.
- **Operation**: The operation to use for modifying the variable value.
- **Value**: The value to combine with the variable using the selected operation.

## Seed Random Number Generator

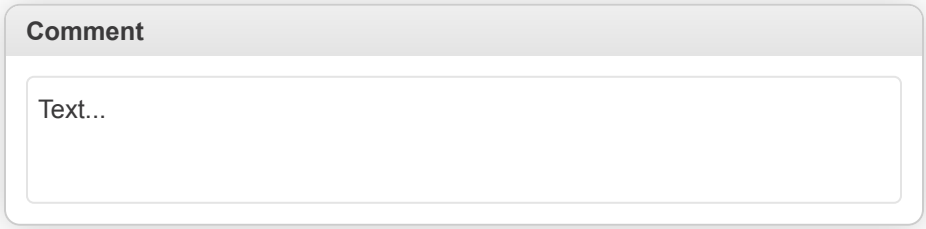
Place this to run in response to user input to ensure random numbers change between playthroughs.



# Miscellaneous

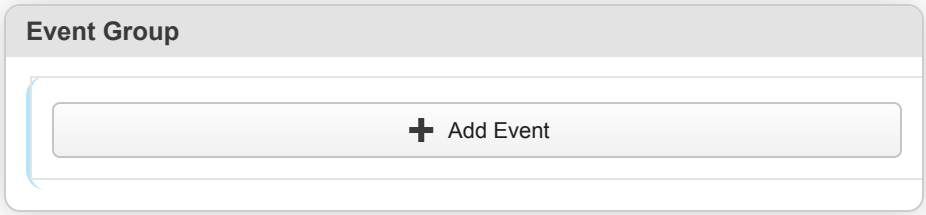
## Comment

Allows you to leave notes within your scripts. Provides no functionality in-game. The text you type automatically gets set in the event title, so you can collapse the comment and still read its content.




## Event Group

Alows you to group together parts of your script for organizational purposes.



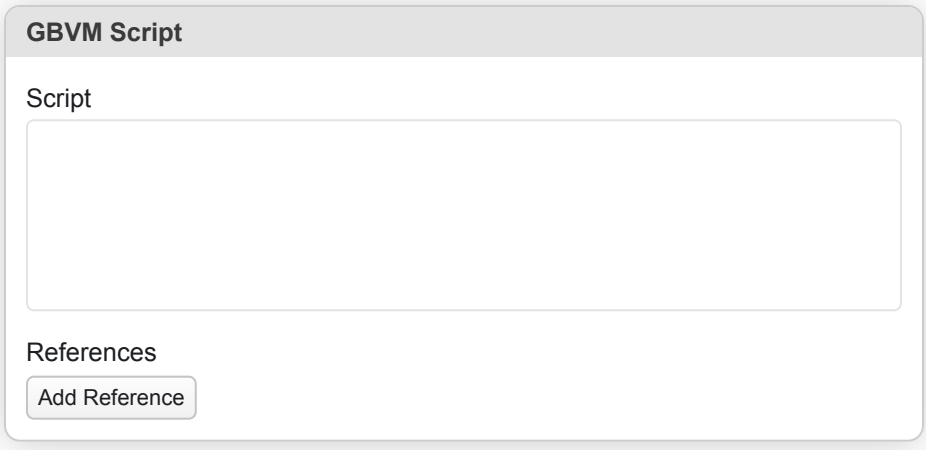

## GBVM Script

Run a GBVM script.

**References**

/docs/scripting/gbvm/

/docs/scripting/gbvm/gbvm-operations




- **Script**: A valid GBVM Script to execute.
- **References**: A list of the assets and entities used in your GBVM script. Use this to let GB Studio know that a file is needed

by your script, preventing it from being excluded in the final build.

## Link: Close

**Link: Close**

Close the current link session.

## Link: Host

**Link: Host**

Host a link session.

## Link: Join

**Link: Join**

Join a link session.

## Link: Transfer

**Link: Transfer**

Send Variable

$Variable0

Receive Variable

$Variable0

Packet Size

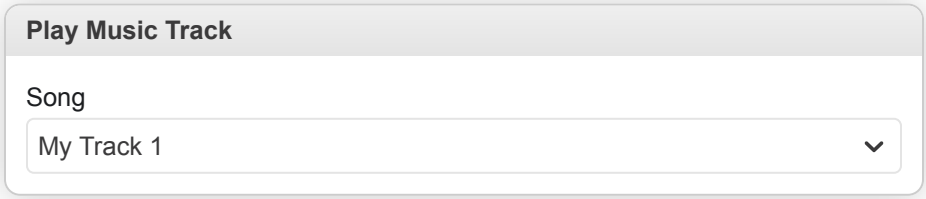
1

- **Send Variable**
- **Receive Variable**
- **Packet Size**

# Music & Sound Effects

## Play Music Track

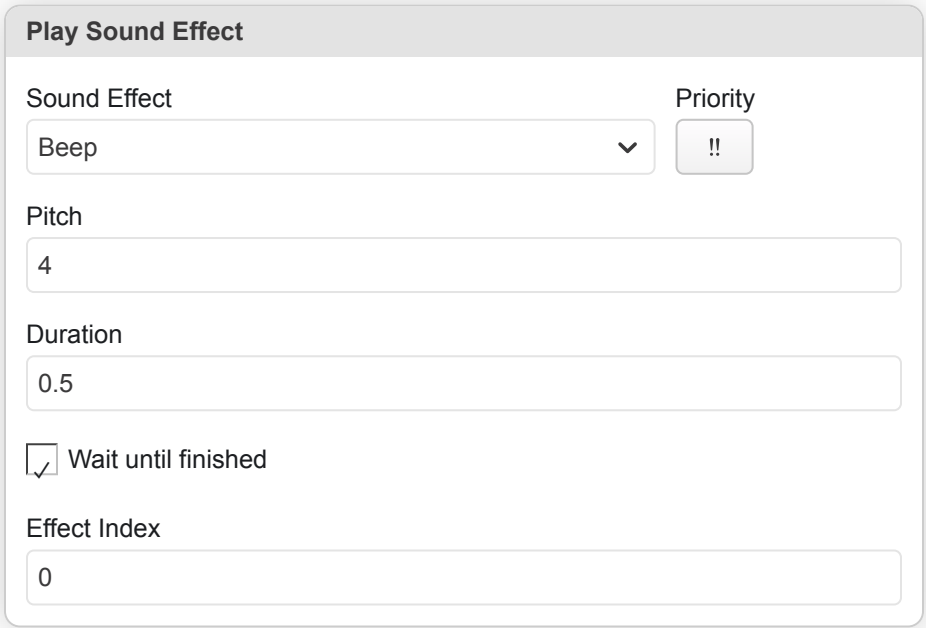
Plays a music file. If you play a new song while another song is playing, the old song will stop automatically.




- **Song**: The song to play.

## Play Sound Effect

Play a sound effect, choose from playing a .WAV, .VGM, or .SAV (fxhammer) file from `/assets/sounds` or a preset sound effect.




- **Sound Effect**: The sound effect to play. Can choose from files within `/assets/sounds` or from preset sounds like `Beep`, `Pitch` and `Tone`.
- **Priority**: The priority of the effect, high, medium or low. If two sound effects are playing at the same time then higher priority sound effects will take precedence.
- **Pitch**: The pitch of the sound effect (Beep effect only).
- **Frequency in hz**: The frequency of the sound effect in hz (Tone effect only).
- **Duration**: The length of time to play the sound effect.
- **Wait until finished**: Set if script should pause until sound effect has finished playing.

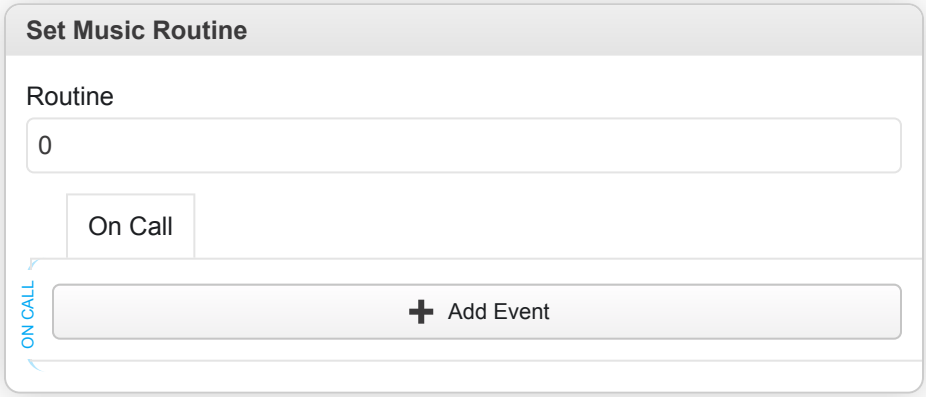
- **Effect Index**: The effect number to play (for fxhammer only).

## Set Music Routine

Attach a script to one of the four music routines that can be triggered from a .uge file. In the music editor you are able to use the call routine effect in your songs to trigger these scripts in time to music.

**References**

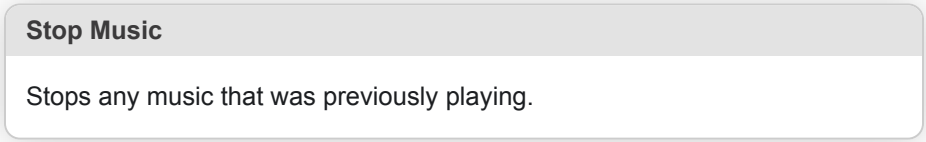
/docs/assets/music/music-huge#effects




- **Routine**: The music routine, either 0, 1, 2 or 3.
- **On Call**: The script to run when the routine is called.

## Stop Music

Stops any currently playing music.



# Save Data

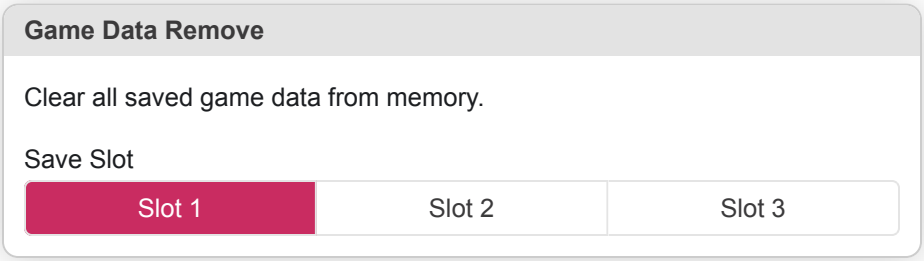
## Game Data Load

Load the saved game data from the selected slot.


- **Save Slot**: The save slot to use.

## Game Data Remove

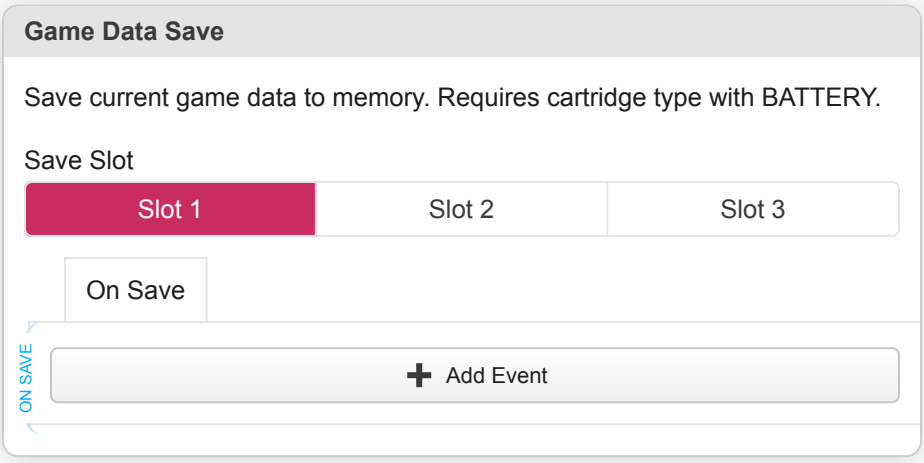
Remove any previously saved game data in the selected slot.




- **Save Slot**: The save slot to use.

## Game Data Save

Save the current game data into the selected slot.

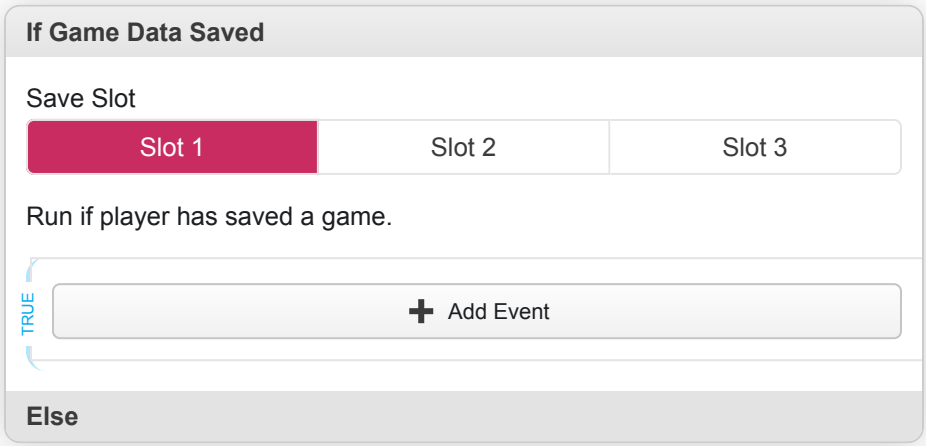



- **Save Slot**: The save slot to use.

- **On Save**: A script to run after the save is completed. This won't be run on game load so you can use it show a 'Save Was Successful' message.

## If Game Data Saved

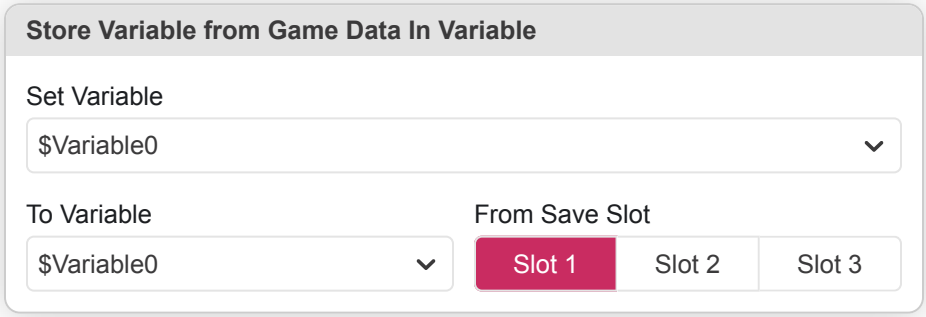
Conditionally run part of the script if save data is present within the specified save slot.




- **Save Slot**: The save slot to use.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## Store Variable from Game Data In Variable

Read a variable's value from a specified save slot and store it in a variable.

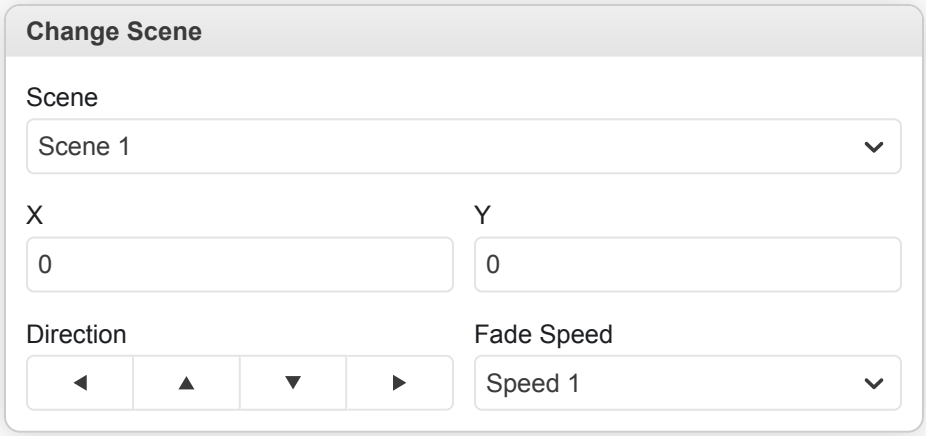



- **Set Variable**: The variable to update.
- **To Variable**: The variable to read the value of.
- **From Save Slot**: The save slot to use.

# Scene

## Change Scene

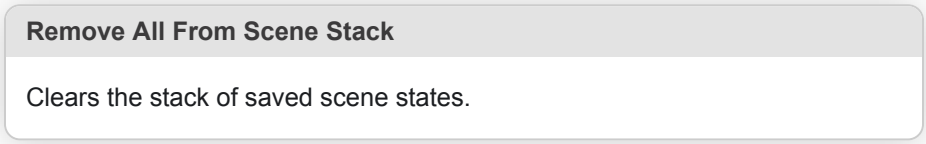
Transition to a new scene with player at a specified position and direction. A connection line will be drawn between the source of the event and the destination scene with an icon appearing at the destination position. It's possible to drag this icon around and between scenes to modify the event.




- **Scene**: The scene to transition to.
- **X**: The initial player horizontal position in the new scene.
- **Y**: The initial player vertical position in the new scene.
- **Direction**: The initial player direction.
- **Fade Speed**: The speed of the fade animation.

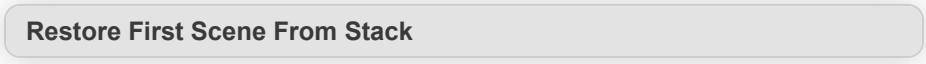
## Remove All From Scene Stack

Remove all scenes from the scene stack without leaving the current scene.




## Restore First Scene From Stack

Transition to the very first scene stored on the stack, for instance if you had multiple levels of menu scenes you could use this to imediately return to the game scene. This event will cause the scene stack to become empty.



Pop all scene state from stack.

Fade Speed

Speed 1

- **Fade Speed**: The speed of the fade animation.

## Restore Previous Scene From Stack

Transition to the last stored scene from the scene stack using the specified fade speed. The previous scene will then be removed from the stack so the next time this event is used it will transition to the scene before that.

**Restore Previous Scene From Stack**

Pop the top scene state from stack.

Fade Speed

Speed 1

- **Fade Speed**: The speed of the fade animation.

## Store Current Scene On Stack

Store the current scene and player state on to the scene stack, this allows you to return to this exact location later using the Scene Restore events. A common use of this event would be to include in a script just before a Change Scene event to open a menu scene, in the menu scene you could wait for the player to press a close button and then use the Restore Previous From Stack event to return to where the player opened the menu.

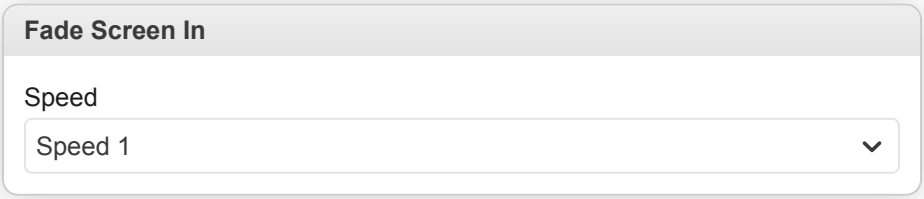
**Store Current Scene On Stack**

Push scene state to stack.

# Screen

## Fade Screen In

Fade the scene from a blank screen.




- **Speed**: The speed of the fade animation.

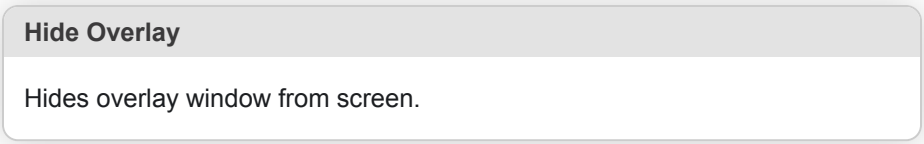
## Fade Screen Out

Fade the scene to a blank screen.


- **Speed**: The speed of the fade animation.

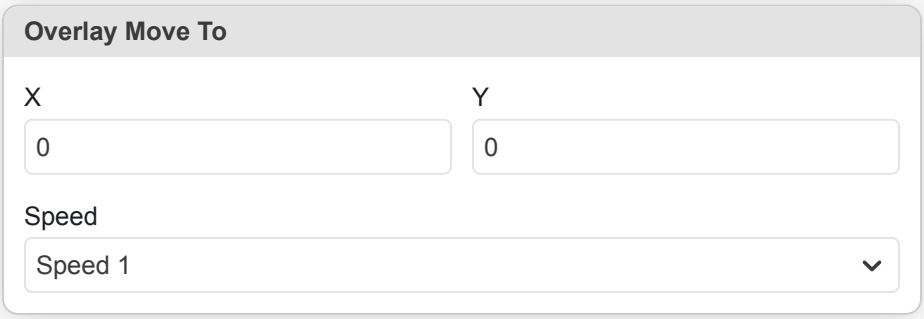
## Hide Overlay

Hides the screen overlay.




## Overlay Move To

Moves the overlay to a new position on the screen.



- **X**: The horizontal position.
- **Y**: The vertical position.
- **Speed**: The movement speed.

## Show Overlay

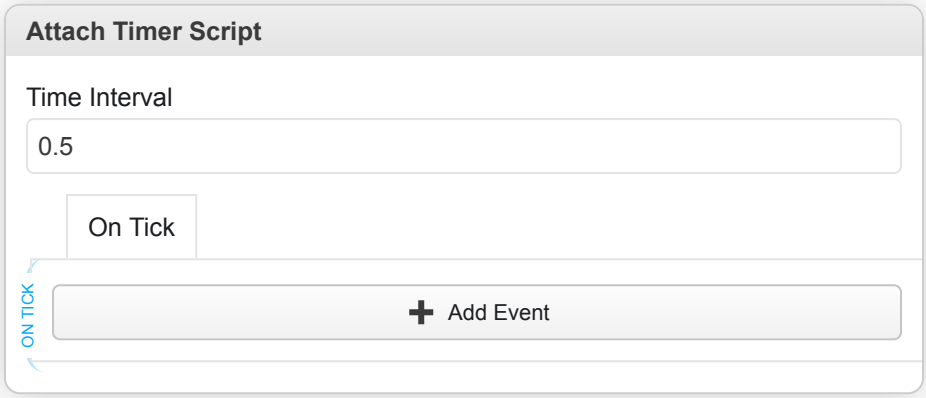
Show either a black or white window over the top of the current game screen. Can be used to obscure and then reveal parts of the scene background for example on the sample project logo screen.


- **Fill Color**: The color to fill the overlay with, either black or white.
- **X**: The horizontal position.
- **Y**: The vertical position.

# Timer

## Attach Timer Script

Run the specified script repeatedly after a time interval. The script will keep running in the background until a Remove Timer Script event is called or the scene is changed using a Change Scene event.



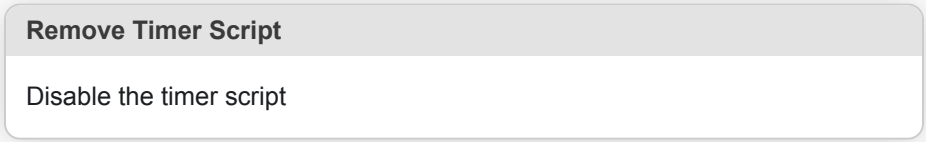

- **Time Interval**: The length of time to wait before running the script each time.
- **On Tick**: The script to run when the timer is triggered.

## Idle

Pause the script for a single frame.


## Remove Timer Script

Remove the timer script so it will no longer be called.




## Restart Timer

Reset the countdown timer back to zero. The script will call again after the time specified originally.

# Wait

Pause the script for a period of time.


- **Duration**: The length of time to pause the script for in seconds or frames.

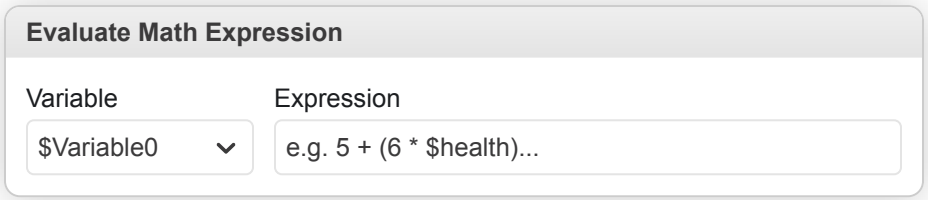
# Variables

## Evaluate Math Expression

Set a variable to the result of evaluating a math expression.

**References**

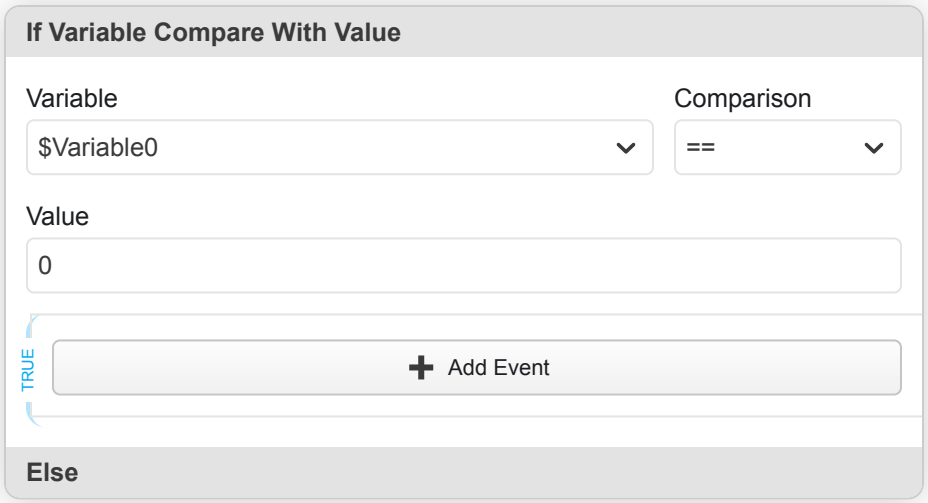
/docs/scripting/math-expressions




- **Variable**: The variable to use.
- **Expression**: The expression to evaluate.

## If Variable Compare With Value

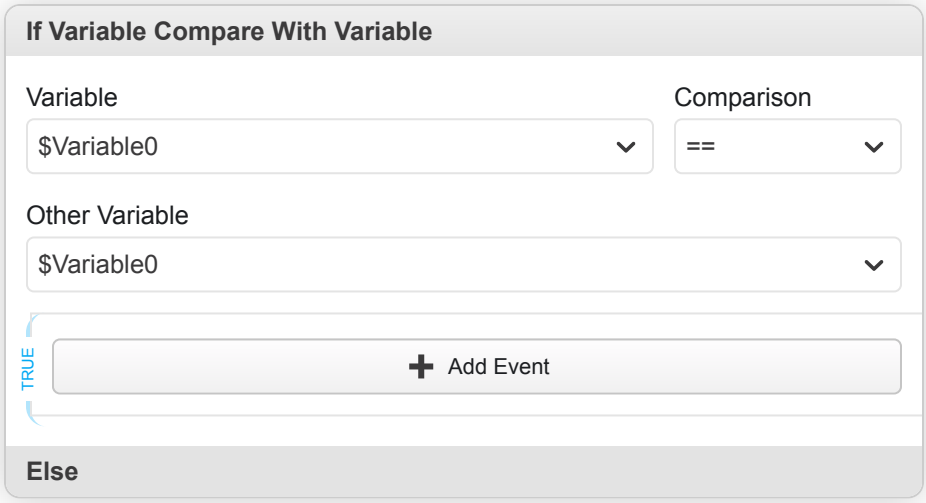
Conditionally run part of the script based on the value of a variable compared with a value.




- **Variable**: The variable to use.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **Value**: The value to compare with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## If Variable Compare With Variable

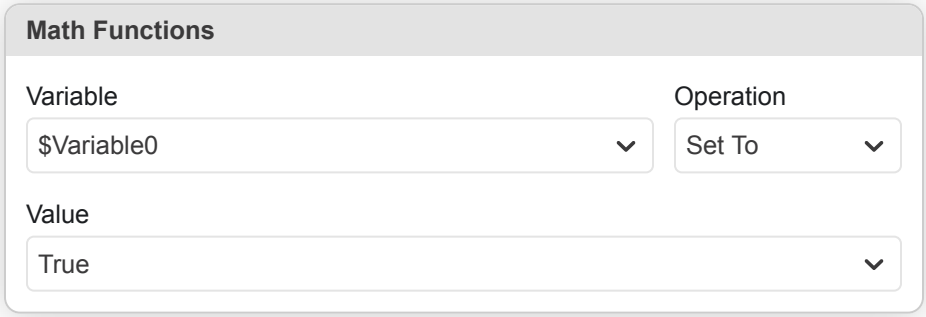
Conditionally run part of the script based on the value of a variable compared with another variable.




- **Variable**: The variable to use.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **Other Variable**: The variable to compare with.
- **True**: The script to run if the condition is true.
- **False**: The script to run if the condition is false.

## Math Functions

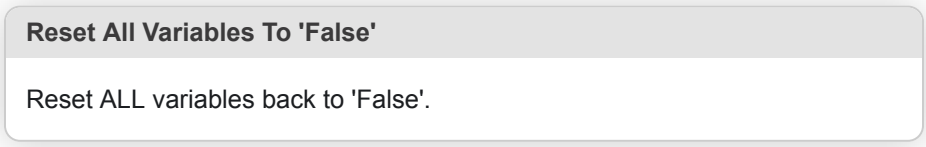
Allows you to perform various maths functions on a variable to add/subtract/multiply/divide/modulus a value/variable/random number.




- **Variable**: The variable to use.
- **Operation**: The operation to use for modifying the variable value.
- **Value**: The value to combine with the variable using the selected operation.

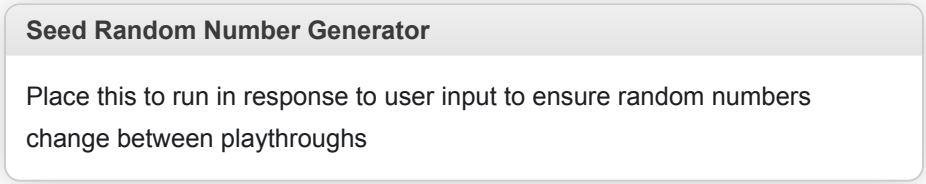
## Reset All Variables To 'False'

Reset all variables used by your project back to false.



## Seed Random Number Generator

Place this to run in response to user input to ensure random numbers change between playthroughs.




## Store Actor Position In Variables

Store the current position of an actor within two variables, one to store the horizontal position and another to store the vertical position.


- **Actor**: The actor you want to check.
- **X**: The variable to use for the horizontal position.
- **Y**: The variable to use for the vertical position.

## Store Engine Field In Variable

Store the value of an Engine Field in a variable.

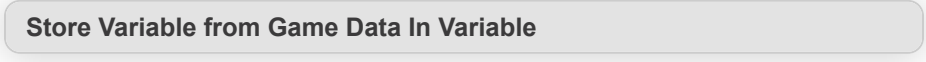
**References**

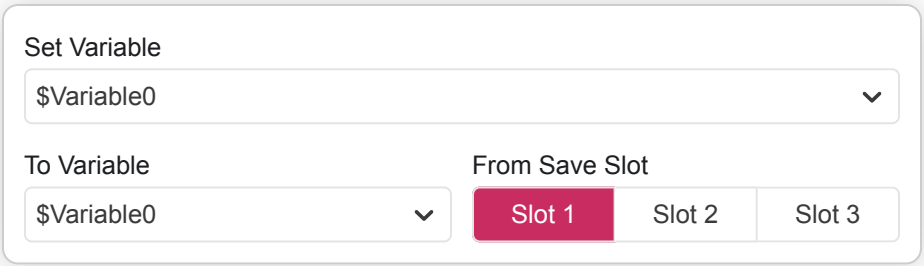
/docs/settings/#engine-settings


- **Engine Field**: The engine field to read the value of.

## Store Variable from Game Data In Variable

Read a variable's value from a specified save slot and store it in a variable.

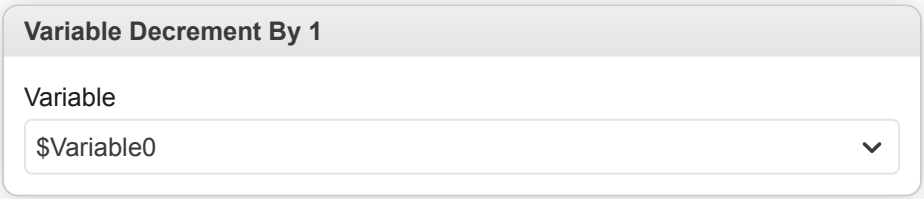





- **Set Variable**: The variable to update.
- **To Variable**: The variable to read the value of.
- **From Save Slot**: The save slot to use.

## Variable Decrement By 1

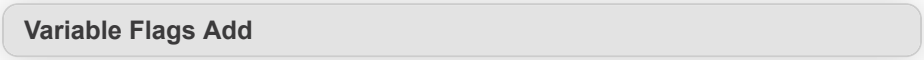
Decrease the value of the specified variable by one.

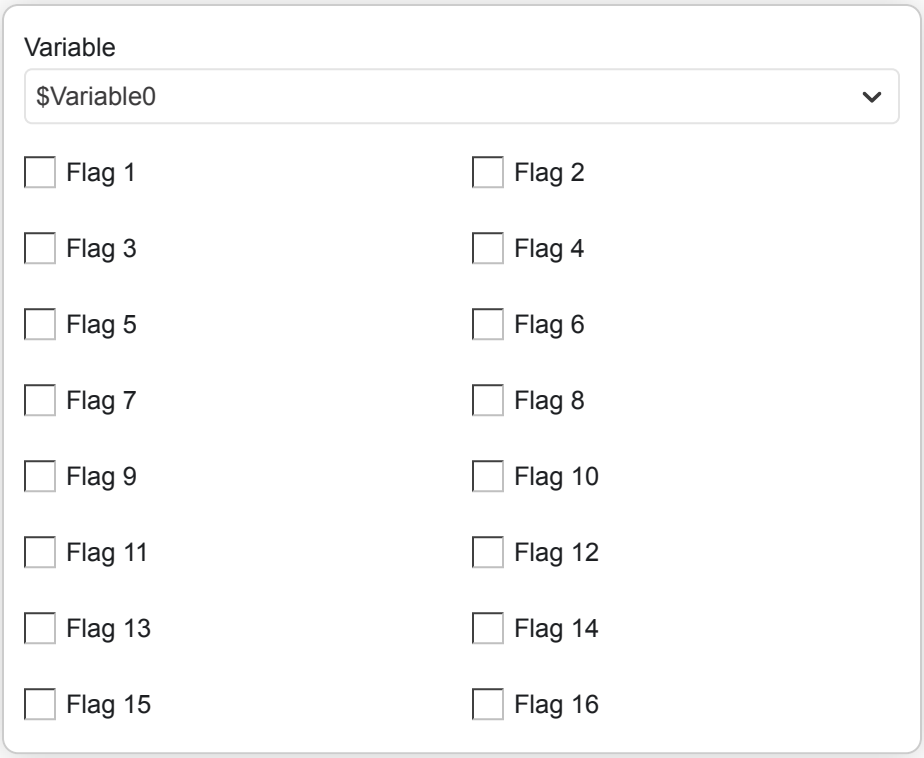



- **Variable**: The variable to use.

## Variable Flags Add

Set selected flags to true on a variable. All unselected flags will keep their previous value.



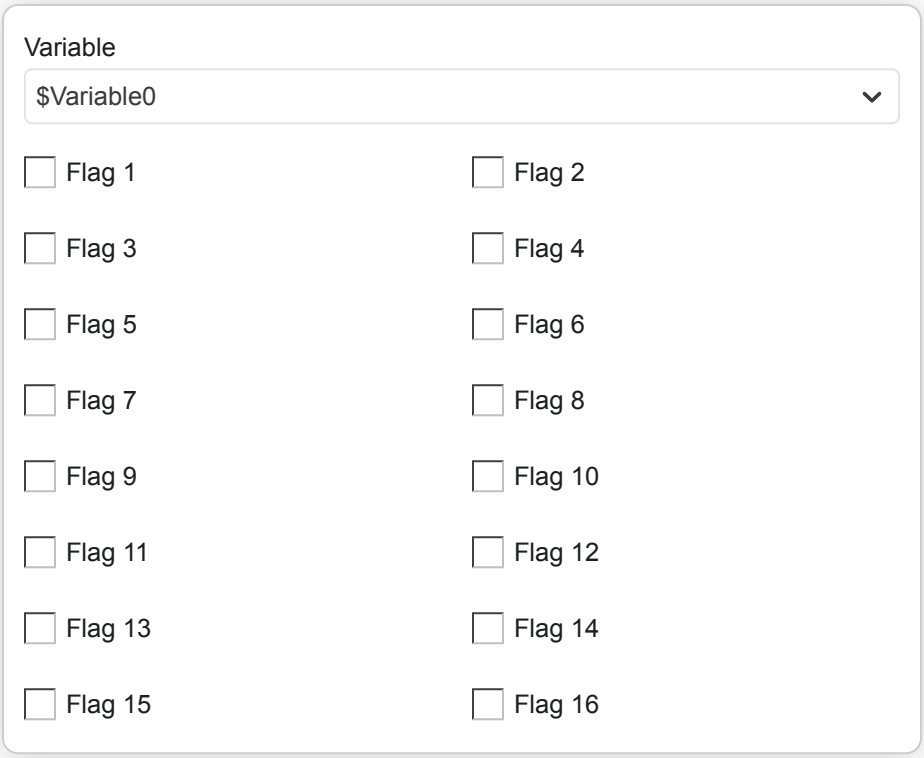



- **Variable**: The variable to use.
- **Flag 1**: Set flag 1 to true.
- **Flag 2**: Set flag 2 to true.
- **Flag 3**: Set flag 3 to true.

## Variable Flags Clear

Set selected flags to false on a variable. All unselected flags will keep their previous value.

**Variable Flags Clear**




- **Variable**: The variable to use.
- **Flag 1**: Set flag 1 to false.
- **Flag 2**: Set flag 2 to false.
- **Flag 3**: Set flag 3 to false.

## Variable Flags Set

Set the value of a variable by enabling individual bits of the 16-bit number. Allows 16 true/false values to be stored within a single variable. Setting the flags will replace the previous value of the variable.

**Variable Flags Set**

Variable

$Variable0

- [ ] Flag 1
- [ ] Flag 2
- [ ] Flag 3
- [ ] Flag 4
- [ ] Flag 5
- [ ] Flag 6
- [ ] Flag 7
- [ ] Flag 8
- [ ] Flag 9
- [ ] Flag 10
- [ ] Flag 11
- [ ] Flag 12
- [ ] Flag 13
- [ ] Flag 14
- [ ] Flag 15
- [ ] Flag 16

- **Variable**: The variable to use.
- **Flag 1**: Set flag 1 to true.
- **Flag 2**: Set flag 2 to true.
- **Flag 3**: Set flag 3 to true.

## Variable Increment By 1

Increase the value of the specified variable by one.

**Variable Increment By 1**

Variable

$Variable0

- **Variable**: The variable to use.

## Variable Set To 'False'

Set the value of the specified variable to false.

**Variable Set To 'False'**

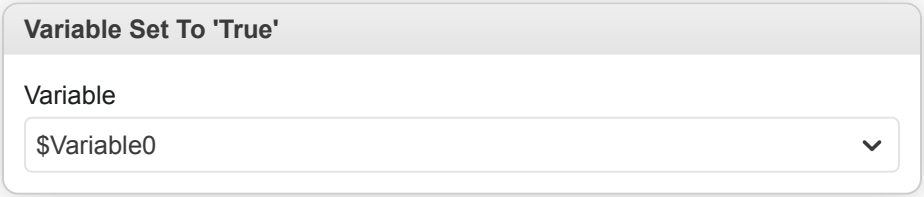
Variable

$Variable0

- **Variable**: The variable to use.

## Variable Set To 'True'

Set the value of the specified variable to true.




- **Variable**: The variable to use.

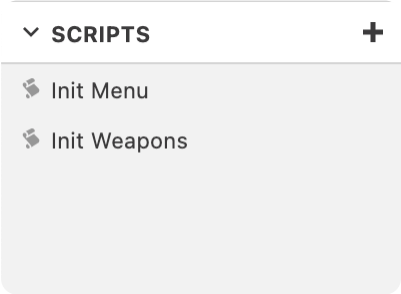
## Variable Set To Value

Set the specified variable to a defined value.


- **Variable**: The variable to use.
- **Value**: The value to set the selected variable to.

# Custom Scripts

*Custom Scripts* allow you to create reusable procedures in your game that can be called from any of your scripts.




Your *Custom Scripts* will be listed in the *Scripts* section of the *Navigator* while on the *Game World View*.

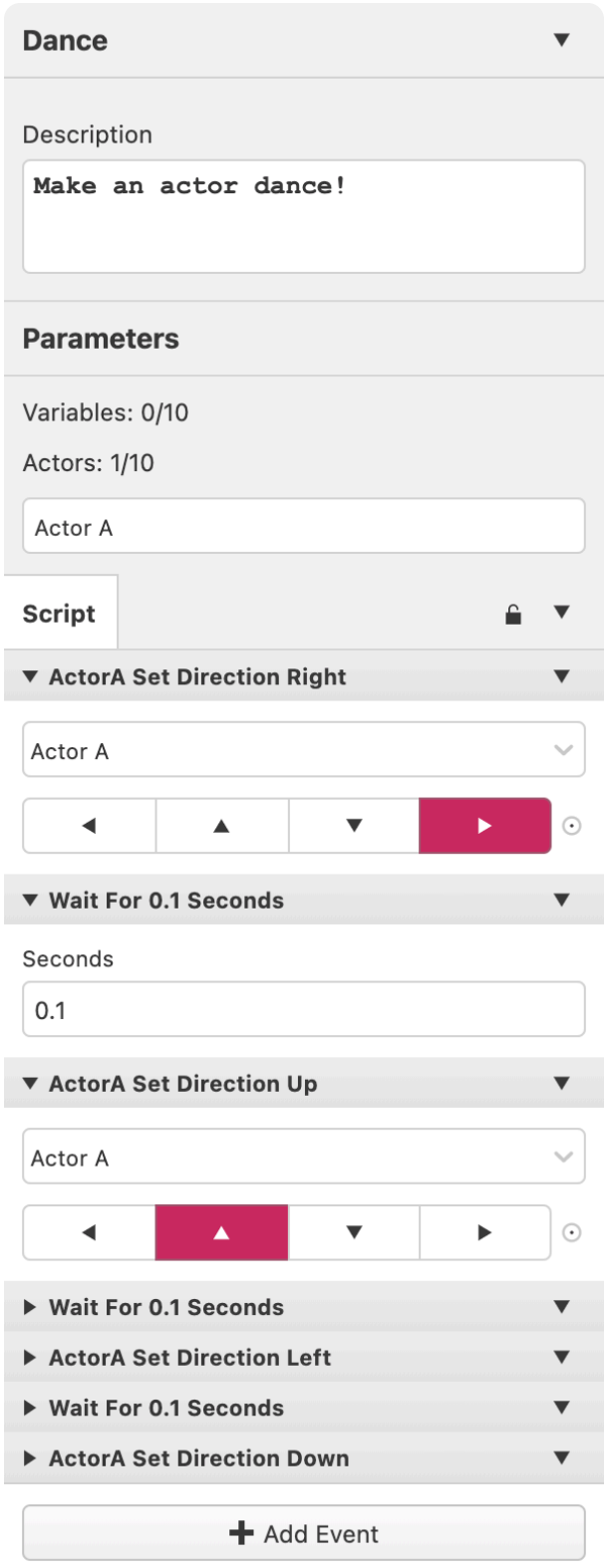
Click the `+` button to create a new *Custom Script* or select one to edit from the list.

Once you've given your *Custom Script* a name you can start building a script in the same way you would for *Actors*, *Triggers* and *Scenes*.

## Parameters

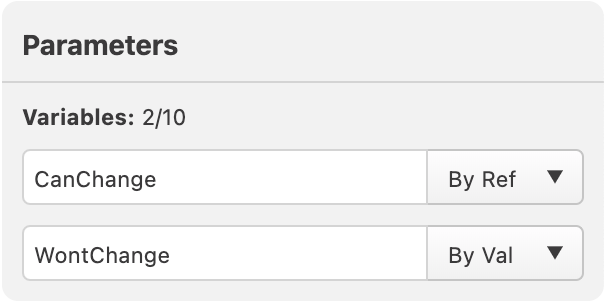
Whenever you add an event that reads a *Variable* it will get added to the list of input parameters for the *Custom Script*, where you are able to give that input a memorable name. Events that affect *Actors* will, by default, apply to the player but if you use the actor selector you will be able to set the event to read the *Actor* value from an input parameter also.

For example the following custom script makes `Actor A` rotate in a circle.




## Passing by Reference or Value

When using variables in a custom script you have the choice to pass by reference (*By Ref*) or to pass by value (*By Val*) by clicking the drop down button next to each variable in the parameters list.

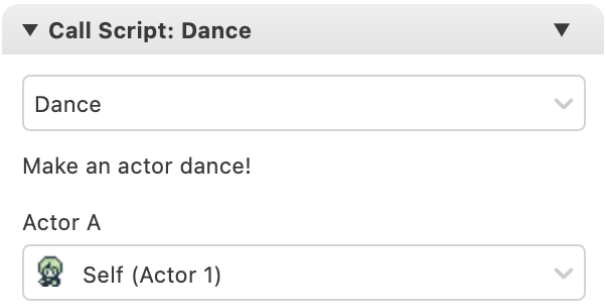



- **Pass By Reference** Allows the custom script to modify the value of a variable parameter. Any changes to the variable's value from inside the script will also update the variable's value outside of the script. Use this if you want the custom script to be able to change the value of a variable that was passed in.

- **Pass By Value** Copies the current value of the variable at call time so that any changes to the variable parameter from within the script will not affect the variable that was passed in. Use this if you want the custom script to **NOT** be able to change the variable that was passed in.

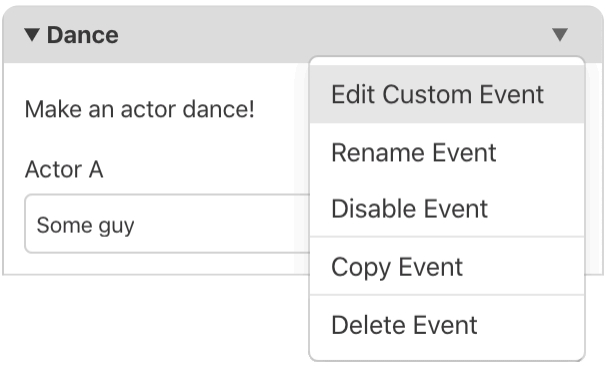
## Calling a Custom Script

Once you have created a *Custom Script* you can call it from any other script by adding a *Call Script* event which will appear as follows.



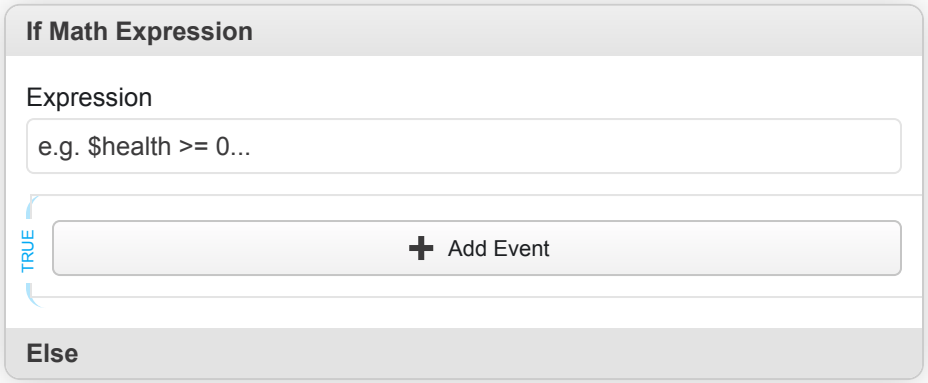

You first must choose the script which you wish to call, if that script has any variable or actor parameters you can then choose which inputs to use.

If you ever want to edit the *Custom Script* you can return to it using the list on the *Navigator* or by selecting *Edit Custom Script* from the event dropdown menu.



# Math Expressions

The Evaluate Math Expression and If Math Expression events allow mathematical expressions to be used for performing calculations.




Expressions allow you to use many mathematical operations such as:

- `+` add
- `-` subtract
- `*` multiply
- `/` divide
- `==` Equal to
- `!=` Not equal to
- `>=` Greater than or equal to

You are also able to use the following functions

- `min(a, b)` return the minimum of two values a and b
- `max(a, b)` return the maximum of two values a and b
- `abs(a)` return the absolute value of a

You can use variables in expressions by typing `$` and searching for the variable's name.

# GBVM

GB Studio's game engine runs on a stack based virtual machine called GBVM game engine. You can access the virtual machine directly by using a GBVM Script event in you game.

**GBVM Script**

Script

```
VM_PUSH_CONST           0       ; Y coord
VM_PUSH_CONST           0       ; X coord
VM_PUSH_CONST           128
VM_LOAD_TILESET         .ARG0, ___bank_bg_cave, _bg_cave
```

References

Add Reference

## Learning GBVM

If you want to know more about GBVM and how to use it then check out GB Studio Central's getting started guide, Learning GBVM.

# GBVM Operations

## Core

### VM_STOP

```
VM_STOP
```

Stops execution of context

### VM_PUSH_CONST

```
VM_PUSH_CONST VAL
```

Pushes immediate value to the top of the VM stack

- **VAL**: immediate value to be pushed

### VM_POP

```
VM_POP N
```

Removes the top values from the VM stack

- **N**: number of values to be removed from the stack

### VM_CALL

```
VM_CALL ADDR
```

Call script by near address

- **ADDR**: address of the script subroutine

### VM_RET

```
VM_RET
```

Returns from the near call

### VM_RET_N

```
VM_RET_N N
```

Returns from the near call and clear N arguments on stack

- **N**: number of arguments to be removed from the stack

### VM_GET_FAR

```
VM_GET_FAR IDX, SIZE, BANK, ADDR
```

Get byte or word by the far pointer into variable

- **IDX**: Target variable
- **SIZE**: Size of the ojject to be acquired:
  `.GET_BYTE` - get 8-bit value
  `.GET_WORD` - get 16-bit value
- **BANK**: Bank number of the object
- **ADDR**: Address of the object

### VM_LOOP

```
VM_LOOP IDX, LABEL, N
```

Loops while variable is not zero, by the near address

- **IDX**: Loop counter variable
- **LABEL**: Jump label for the next iteration
- **N**: amount of values to be removed from stack on exit

### .CASE

```
.CASE VAL, LABEL
```

### VM_SWITCH

```
VM_SWITCH IDX, SIZE, N
```

Compares variable with a set of values, and if equal jump to the specified label.
values for testing may be defined with the .CASE macro, where VAL parameter is a value for testing and LABEL is a jump label

- **IDX**: variable for compare
- **SIZE**: amount of entries for test.
- **N**: amount of values to de cleaned from stack on exit

### VM_JUMP

```
VM_JUMP LABEL
```

Jumps to near address

- **ARG0**: jump label

## VM_CALL_FAR

```
VM_CALL_FAR BANK, ADDR
```

Call far routine (inter-bank call)

- **BANK**: Bank number of the routine
- **ADDR**: Address of the routine

## VM_RET_FAR

```
VM_RET_FAR
```

Return from the far call

## VM_RET_FAR_N

```
VM_RET_FAR_N N
```

Return from the far call and remove N arguments from stack

- **N**: Number of arguments to be removed from stack

## VM_INVOKE

```
VM_INVOKE BANK, ADDR, N, PARAMS
```

Invokes C function until it returns true.

- **BANK**: Bank number of the function
- **ADDR**: Address of the function, currently 2 functions are implemented:
  `_wait_frames` - wait for N vblank intervals
  `_camera_shake` - shake camera N times
- **N**: Number of arguments to be removed from stack on return
- **PARAMS**: Points the first parameter to be passed into the C function

## VM_BEGINTHREAD

```
VM_BEGINTHREAD BANK, THREADPROC, HTHREAD, NARGS
```

Spawns a thread in a separate context

- **BANK**: Bank number of a thread function
- **THREADPROC**: Address of a thread function
- **HTHREAD**: Variable that receives the thread handle

- **NARGS**: Amount of values from the stack to be copied into the stack of the new context

## VM_IF

```
VM_IF CONDITION, IDXA, IDXB, LABEL, N
```

Compares two variables using for condition.

- **CONDITION**: Condition for test:
  `.EQ` - variables are equal
  `.LT` - A is lower than B
  `.LTE` - A is lower or equal than B
  `.GT` - A is greater than B
  `.GTE` - A is greater or equal than B
  `.NE` - A is not equal to B
- **IDXA**: A variable
- **IDXB**: B variable
- **LABEL**: Jump label when result is TRUE
- **N**: Number of values to be removed from stack if the result is true

## VM_PUSH_VALUE_IND

```
VM_PUSH_VALUE_IND IDX
```

Pushes a value on VM stack or a global indirectly from an index in the variable

- **IDX**: variable that contains the index of the variable to be pushed on stack

## VM_PUSH_VALUE

```
VM_PUSH_VALUE IDX
```

Pushes a value of the variable onto stack

- **IDX**: variable to be pushed

## VM_RESERVE

```
VM_RESERVE N
```

Reserves or disposes amount of values on stack

- **N**: positive value - amount of variables to be reserved on stack, negative value - amount of variables to be popped from stack

## VM_SET

```
VM_SET IDXA, IDXB
```

Assigns variable B to variable A

- **IDXA**: Variable A
- **IDXB**: Variable B

## VM_SET_CONST

```
VM_SET_CONST IDX, VAL
```

Assigns immediate value to the variable

- **IDX**: Target variable
- **VAL**: Source immediate value

## VM_RPN

```
VM_RPN
```

Reverse Polish Notation (RPN) calculator, returns result(s) on the VM stack

## .R_INT8

```
.R_INT8 ARG0
```

## .R_INT16

```
.R_INT16 ARG0
```

## .R_REF

```
.R_REF ARG0
```

## .R_REF_IND

```
.R_REF_IND ARG0
```

## .R_REF_SET

```
.R_REF_SET ARG0
```

### .R_OPERATOR

```
.R_OPERATOR ARG0
```

### .R_STOP

```
.R_STOP
```

### VM_JOIN

```
VM_JOIN IDX
```

Joins a thread

- **IDX**: Thread handle for joining

### VM_TERMINATE

```
VM_TERMINATE IDX
```

Kills a thread

- **IDX**: Thread handle for killing

### VM_IDLE

```
VM_IDLE
```

Signals thread runner, that context is in a waitable state

### VM_GET_TLOCAL

```
VM_GET_TLOCAL IDXA, IDXB
```

Gets thread local variable.

- **IDXA**: Target variable
- **IDXB**: Thread local variable index

### VM_IF_CONST

```
VM_IF_CONST CONDITION, IDXA, B, LABEL, N
```

Compares a variable to an immediate value

- **CONDITION**: Condition for test:

`.EQ` - variables are equal
`.LT` - A is lower than B
`.LTE` - A is lower or equal than B
`.GT` - A is greater than B
`.GTE` - A is greater or equal than B
`.NE` - A is not equal to B

- **IDXA**: A variable
- **B**: immediate value to be compared with
- **LABEL**: Jump label when result is TRUE
- **N**: Number of values to be removed from stack if the result is true

## VM_GET_UINT8

```
VM_GET_UINT8 IDXA, ADDR
```

Gets unsigned int8 from WRAM

- **IDXA**: Target variable
- **ADDR**: Address of the unsigned 8-bit value in WRAM

## VM_GET_INT8

```
VM_GET_INT8 IDXA, ADDR
```

Gets signed int8 from WRAM

- **IDXA**: Target variable
- **ADDR**: Address of the signed 8-bit value in WRAM

## VM_GET_INT16

```
VM_GET_INT16 IDXA, ADDR
```

Gets signed int16 from WRAM

- **IDXA**: Target variable
- **ADDR**: Address of the signed 16-bit value in WRAM

## VM_SET_UINT8

```
VM_SET_UINT8 ADDR, IDXA
```

Sets unsigned int8 in WRAM from variable

- **ADDR**: Address of the unsigned 8-bit value in WRAM
- **IDXA**: Source variable

## VM_SET_INT8

```
VM_SET_INT8 ADDR, IDXA
```

Sets signed int8 in WRAM from variable

- **ADDR**: Address of the signed 8-bit value in WRAM
- **IDXA**: Source variable

## VM_SET_INT16

```
VM_SET_INT16 ADDR, IDXA
```

Sets signed int16 in WRAM from variable

- **ADDR**: Address of the signed 16-bit value in WRAM
- **IDXA**: Source variable

## VM_SET_CONST_INT8

```
VM_SET_CONST_INT8 ADDR, V
```

Sets signed int8 in WRAM to the immediate value

- **ADDR**: Address of the signed 8-bit value in WRAM
- **V**: Immediate value

## VM_SET_CONST_UINT8

```
VM_SET_CONST_UINT8 ADDR, V
```

Sets unsigned int8 in WRAM to the immediate value

- **ADDR**: Address of the unsigned 8-bit value in WRAM
- **V**: Immediate value

## VM_SET_CONST_INT16

```
VM_SET_CONST_INT16 ADDR, V
```

Sets signed int16 in WRAM to the immediate value

- **ADDR**: Address of the signed 16-bit value in WRAM
- **V**: Immediate value

### VM_INIT_RNG

```
VM_INIT_RNG IDX
```

Initializes RNG seed with the value from the variable

- **IDX**: Seed value

### VM_RANDOMIZE

```
VM_RANDOMIZE
```

Initializes RNG seed

### VM_RAND

```
VM_RAND IDX, MIN, LIMIT
```

Returns random value between MIN and MIN + LIMIT

- **IDX**: Target variable
- **MIN**: Minimal random value
- **LIMIT**: range of the random values

### VM_LOCK

```
VM_LOCK
```

Disable switching of VM threads

### VM_UNLOCK

```
VM_UNLOCK
```

Enable switching of VM threads

### VM_RAISE

```
VM_RAISE CODE, SIZE
```

Raises an exception

- **CODE**: Exception code:
  `EXCEPTION_RESET` - Resets the device.
  `EXCEPTION_CHANGE_SCENE` - Changes to a new scene.
  `EXCEPTION_SAVE` - Saves the state of the game.

`EXCEPTION_LOAD` - Loads the saved state of the game.

- **SIZE**: Length of the parameters to be passed into the exception handler

## VM_SET_INDIRECT

```
VM_SET_INDIRECT IDXA, IDXB
```

Assigns variable that is addressed indirectly to the other variable

- **IDXA**: Variable that contains the index of the target variable
- **IDXB**: Source variable that contains the value to be assigned

## VM_GET_INDIRECT

```
VM_GET_INDIRECT IDXA, IDXB
```

Assigns a variable to the value of variable that is addressed indirectly

- **IDXA**: Target variable
- **IDXB**: Variable that contains the index of the source variable

## VM_TEST_TERMINATE

```
VM_TEST_TERMINATE FLAGS
```

Terminates unit-testing immediately

- **FLAGS**: terminate flags:
  `.TEST_WAIT_VBL` wait for VBlank before terminating

## VM_POLL_LOADED

```
VM_POLL_LOADED IDX
```

Checks that VM state was restored and reset the restore flag

- **IDX**: Target result variable

## VM_PUSH_REFERENCE

```
VM_PUSH_REFERENCE IDX
```

Translates IDX into absolute index and pushes result to VM stack

- **IDX**: index of the variable

### VM_CALL_NATIVE

```
VM_CALL_NATIVE BANK, PTR
```

Calls native code by the far pointer

- **BANK**: Bank number of the native routine
- **PTR**: Address of the native routine

### VM_MEMSET

```
VM_MEMSET DEST, VALUE, COUNT
```

Clear VM memory

- **DEST**: First variable to be cleared
- **VALUE**: Variable containing the value to be filled with
- **COUNT**: Number of variables to be filled

### VM_MEMCPY

```
VM_MEMCPY DEST, SOUR, COUNT
```

copy VM memory

- **DEST**: First destination variable
- **SOUR**: First source variable
- **COUNT**: Number of variables to be copied

## Actor

### VM_ACTOR_MOVE_TO

```
VM_ACTOR_MOVE_TO IDX
```

Move actor to the new position

- **IDX**: points to the beginning of the pseudo-structure that contains these members:
  `ID` - Actor number
  `X` - New X-coordinate of the actor
  `Y` - New Y-coordinate of the actor
  `ATTR` - Bit flags:
  `.ACTOR_ATTR_H_FIRST` - move horizontal first
  `.ACTOR_ATTR_CHECK_COLL` - respect collisions
  `.ACTOR_ATTR_DIAGONAL` - allow diagonal movement

## VM_ACTOR_MOVE_CANCEL

```
VM_ACTOR_MOVE_CANCEL ACTOR
```

Cancel movement of actor

- **ACTOR**: Variable that contains the actor number

## VM_ACTOR_ACTIVATE

```
VM_ACTOR_ACTIVATE ACTOR
```

Activate the actor

- **ACTOR**: Variable that contains the actor number

## VM_ACTOR_SET_DIR

```
VM_ACTOR_SET_DIR ACTOR, DIR
```

Set direction of the actor

- **ACTOR**: Variable that contains the actor number
- **DIR**: one of these directions:
  `.DIR_DOWN` - actor faces down
  `.DIR_RIGHT` - actor faces right
  `.DIR_UP` - actor faces up
  `.DIR_LEFT` - actor faces left

## VM_ACTOR_DEACTIVATE

```
VM_ACTOR_DEACTIVATE ACTOR
```

Deactivate the actor

- **ACTOR**: Variable that contains the actor number

## VM_ACTOR_SET_ANIM

```
VM_ACTOR_SET_ANIM ACTOR, ANIM
```

Set actor animation

- **ACTOR**: Variable that contains the actor number
- **ANIM**: Animation number

## VM_ACTOR_SET_POS

```
VM_ACTOR_SET_POS IDX
```

Set new actor position

- **IDX**: points to the beginning of the pseudo-structure that contains these members:
  `ID` - Actor number
  `X` - New X-coordinate of the actor
  `Y` - New Y-coordinate of the actor

## VM_ACTOR_EMOTE

```
VM_ACTOR_EMOTE ACTOR, AVATAR_BANK, AVATAR
```

Set actor emotion

- **ACTOR**: Variable that contains the actor number
- **AVATAR_BANK**: Bank of the avatar image
- **AVATAR**: Address of the avatar image

## VM_ACTOR_SET_BOUNDS

```
VM_ACTOR_SET_BOUNDS ACTOR, LEFT, RIGHT, TOP, BOTTOM
```

Set actor bounding box

- **ACTOR**: Variable that contains the actor number
- **LEFT**: Left boundary of the bounding box
- **RIGHT**: Right boundary of the bounding box
- **TOP**: Top boundary of the bounding box
- **BOTTOM**: Bottom boundary of the bounding box

## VM_ACTOR_SET_SPRITESHEET

```
VM_ACTOR_SET_SPRITESHEET ACTOR, SHEET_BANK, SHEET
```

Set actor spritesheet

- **ACTOR**: Variable that contains the actor number
- **SHEET_BANK**: Bank of the sprite sheet
- **SHEET**: Address of the sprite sheet

## VM_ACTOR_SET_SPRITESHEET_BY_REF

```
VM_ACTOR_SET_SPRITESHEET_BY_REF ACTOR, FAR_PTR
```

Set actor spritesheet using far the pointer in variables

- **ACTOR**: Variable that contains the actor number
- **FAR_PTR**: points to the pseudo-struct that contains the address of the sprite sheet:
  `BANK` - Bank of the sprite sheet
  `DATA` - Address of the sprite sheet

## VM_ACTOR_REPLACE_TILE

```
VM_ACTOR_REPLACE_TILE ACTOR, TARGET_TILE, TILEDATA_BANK, TILEDATA, START, LEN
```

Replace tile in the actor spritesheet

- **ACTOR**: Variable that contains the actor number
- **TARGET_TILE**: Tile number for replacement
- **TILEDATA_BANK**: Bank of the tile data
- **TILEDATA**: Address of the tile data
- **START**: Start tile in the tile data array
- **LEN**: Amount of tiles for replacing

## VM_ACTOR_GET_POS

```
VM_ACTOR_GET_POS IDX
```

Get actor position

- **IDX**: points to the beginning of the pseudo-structure that contains these members:
  `ID` - Actor number
  `X` - X-coordinate of the actor
  `Y` - Y-coordinate of the actor

## VM_ACTOR_GET_DIR

```
VM_ACTOR_GET_DIR IDX, DEST
```

Get direction of the actor

- **IDX**: Variable that contains the actor number
- **DEST**: Target variable that receive the actor direction

## VM_ACTOR_GET_ANGLE

```
VM_ACTOR_GET_ANGLE IDX, DEST
```

Get actor angle

- **IDX**: Variable that contains the actor number
- **DEST**: Target variable that receive the actor angle

## VM_ACTOR_SET_ANIM_TICK

```
VM_ACTOR_SET_ANIM_TICK ACTOR, TICK
```

Set actor animation tick

- **ACTOR**: Variable that contains the actor number
- **TICK**: Animation tick

## VM_ACTOR_SET_MOVE_SPEED

```
VM_ACTOR_SET_MOVE_SPEED ACTOR, SPEED
```

Set actor move speed

- **ACTOR**: Variable that contains the actor number
- **SPEED**: Actor move speed

## VM_ACTOR_SET_FLAGS

```
VM_ACTOR_SET_FLAGS ACTOR, FLAGS, MASK
```

Set actor flags

- **ACTOR**: Variable that contains the actor number
- **FLAGS**: bit values to be set or cleared:
  `.ACTOR_FLAG_PINNED` - pin/unpin the actor
  `.ACTOR_FLAG_HIDDEN` - hide/show actor
  `.ACTOR_FLAG_ANIM_NOLOOP` - disable animation loop
  `.ACTOR_FLAG_COLLISION` - disable/enable collision
  `.ACTOR_FLAG_PERSISTENT` - set persistent actor flag
- **MASK**: bit mask of values to be set or cleared

## VM_ACTOR_SET_HIDDEN

```
VM_ACTOR_SET_HIDDEN ACTOR, HIDDEN
```

Hide/show actor

- **ACTOR**: Variable that contains the actor number
- **HIDDEN**: `.ACTOR_VISIBLE` shows actor, `.ACTOR_HIDDEN` hides the actor

## VM_ACTOR_SET_COLL_ENABLED

```
VM_ACTOR_SET_COLL_ENABLED ACTOR, ENABLED
```

Enable/disable actor collisions

- **ACTOR**: Variable that contains the actor number
- **ENABLED**: `.ACTOR_COLLISION_DISABLED` disables actor collision, `.ACTOR_COLLISION_ENABLED` enables actor collision

## VM_ACTOR_TERMINATE_UPDATE

```
VM_ACTOR_TERMINATE_UPDATE ACTOR
```

Terminates the actor update script

- **ACTOR**: Variable that contains the actor number

## VM_ACTOR_SET_ANIM_FRAME

```
VM_ACTOR_SET_ANIM_FRAME ACTOR
```

Set animation frame for the actor

- **ACTOR**: pseudo-struct that contains these members:
  `ID` - Actor number
  `FRAME` - Animation frame

## VM_ACTOR_GET_ANIM_FRAME

```
VM_ACTOR_GET_ANIM_FRAME ACTOR
```

Get animation frame of the actor

- **ACTOR**: pseudo-struct that contains these members:
  `ID` - Actor number
  `FRAME` - Animation frame

## VM_ACTOR_SET_ANIM_SET

```
VM_ACTOR_SET_ANIM_SET ACTOR, OFFSET
```

Set animation frame for the actor

- **ACTOR**: Variable that contains the actor number
- **OFFSET**: Animation set number

## Camera

### VM_CAMERA_MOVE_TO

```
VM_CAMERA_MOVE_TO IDX, SPEED, AFTER_LOCK
```

Moves the camera to the new position

- **IDX**: Start of the pseudo-structure which contains the new camera position:
  `X` - X-coordinate of the camera position
  `Y` - Y-coordinate of the camera position
- **SPEED**: Speed of the camera movement
- **AFTER_LOCK**: Lock status of the camera after the movement
  `.CAMERA_LOCK` - lock camera by X and Y
  `.CAMERA_LOCK_X` - lock camera by X
  `.CAMERA_LOCK_Y` - lock camera by Y
  `.CAMERA_UNLOCK` - unlock camera

### VM_CAMERA_SET_POS

```
VM_CAMERA_SET_POS IDX
```

Sets the camera position

- **IDX**: Start of the pseudo-structure which contains the new camera position:
  `X` - X-coordinate of the camera position
  `Y` - Y-coordinate of the camera position

## Color

### .DMG_PAL

```
.DMG_PAL COL1, COL2, COL3, COL4
```

### .CGB_PAL

```
.CGB_PAL R1,G1,B1 R2,G2,B2 R3,G3,B3 R4,G4,B4
```

### VM_LOAD_PALETTE

```
VM_LOAD_PALETTE MASK, OPTIONS
```

## Game Boy

### VM_LOAD_TILES

```
VM_LOAD_TILES ID, LEN, BANK, ADDR
```

### VM_LOAD_TILESET

```
VM_LOAD_TILESET IDX, BANK, BKG
```

Loads a new tileset into the background VRAM tiles starting at a given tile id (`IDX`).

### VM_SET_SPRITE_VISIBLE

```
VM_SET_SPRITE_VISIBLE MODE
```

### VM_SHOW_SPRITES

```
VM_SHOW_SPRITES
```

### VM_HIDE_SPRITES

```
VM_HIDE_SPRITES
```

### VM_INPUT_WAIT

```
VM_INPUT_WAIT MASK
```

### VM_INPUT_ATTACH

```
VM_INPUT_ATTACH MASK, SLOT
```

### VM_INPUT_GET

```
VM_INPUT_GET IDX, JOYID
```

### VM_CONTEXT_PREPARE

```
VM_CONTEXT_PREPARE SLOT, BANK, ADDR
```

### VM_OVERLAY_SET_MAP

```
VM_OVERLAY_SET_MAP IDX, X_IDX, Y_IDX, BANK, BKG
```

### VM_GET_TILE_XY

```
VM_GET_TILE_XY TILE_IDX, X_IDX, Y_IDX
```

### VM_REPLACE_TILE

```
VM_REPLACE_TILE TARGET_TILE_IDX, TILEDATA_BANK, TILEDATA, START_IDX, LEN
```

### VM_POLL

```
VM_POLL IDX_EVENT, IDX_VALUE, MASK
```

### VM_SET_SPRITE_MODE

```
VM_SET_SPRITE_MODE MODE
```

### VM_REPLACE_TILE_XY

```
VM_REPLACE_TILE_XY X, Y, TILEDATA_BANK, TILEDATA, START_IDX
```

### VM_INPUT_DETACH

```
VM_INPUT_DETACH MASK
```

## GB Printer

### VM_PRINTER_DETECT

```
VM_PRINTER_DETECT ERROR, DELAY
```

Detect printer

- **ERROR**: Target variable that receives the result of detection
- **DELAY**: Detection timeout

### VM_PRINT_OVERLAY

```
VM_PRINT_OVERLAY ERROR, START, HEIGHT, MARGIN
```

Print up to HEIGHT rows of the overlay window (must be multiple of 2)

- **ERROR**: Target variable that receives the result of printing
- **START**: Start line of the overlay window
- **HEIGHT**: Amount of lines to print
- **MARGIN**: Lines to feed after the printing is finished

# Load and Save

## .SAVE_SLOT

```
.SAVE_SLOT SLOT
```

## VM_SAVE_PEEK

```
VM_SAVE_PEEK RES, DEST, SOUR, COUNT, SLOT
```

Reads variables from the save slot

- **RES**: Result of the operation
- **DEST**: First destination variable to be read into
- **SOUR**: First source variable in the save slot
- **COUNT**: Number of variables to be read
- **SLOT**: Save slot number

## VM_SAVE_CLEAR

```
VM_SAVE_CLEAR SLOT
```

Erases data in save slot

- **SLOT**: Slot number

# Math

## VM_SIN_SCALE

```
VM_SIN_SCALE IDX, IDX_ANGLE, SCALE
```

## VM_COS_SCALE

```
VM_COS_SCALE IDX, IDX_ANGLE, SCALE
```

## Music and Sound

### VM_MUSIC_PLAY

```
VM_MUSIC_PLAY TRACK_BANK, TRACK, LOOP
```

Starts playing of music track

- **BANK**: Bank number of the track
- **ADDR**: Address of the track
- **LOOP**: If the track will loop on end (`.MUSIC_LOOP`) or not (`.MUSIC_NO_LOOP`)

### VM_MUSIC_STOP

```
VM_MUSIC_STOP
```

Stops playing of music track

### VM_MUSIC_MUTE

```
VM_MUSIC_MUTE MASK
```

Mutes/unmutes mysic channels.

- **MASK**: Mute Mask. The 4 lower bits represent the 4 audio channels.

| `MASK` | Channel 1 | Channel 2 | Channel 3 | Channel 4 |
|---|---|---|---|---|
| `0b0000` | Muted | Muted | Muted | Muted |
| `0b0001` | Muted | Muted | Muted | Not Muted |
| `0b0010` | Muted | Muted | Not Muted | Muted |
| `0b0011` | Muted | Muted | Not Muted | Not Muted |
| `0b0100` | Muted | Not Muted | Muted | Muted |
| `0b0101` | Muted | Not Muted | Muted | Not Muted |
| `0b0110` | Muted | Not Muted | Not Muted | Muted |

| `MASK` | Channel 1 | Channel 2 | Channel 3 | Channel 4 |
|---|---|---|---|---|
| `0b0111` | Muted | Not Muted | Not Muted | Not Muted |
| `0b1000` | Not Muted | Muted | Muted | Muted |
| `0b1001` | Not Muted | Muted | Muted | Not Muted |
| `0b1010` | Not Muted | Muted | Not Muted | Muted |
| `0b1011` | Not Muted | Muted | Not Muted | Not Muted |
| `0b1100` | Not Muted | Not Muted | Muted | Muted |
| `0b1101` | Not Muted | Not Muted | Muted | Not Muted |
| `0b1110` | Not Muted | Not Muted | Not Muted | Muted |
| `0b1111` | Not Muted | Not Muted | Not Muted | Not Muted |

## VM_SOUND_MASTERVOL

```
VM_SOUND_MASTERVOL VOL
```

Sets master volume

- **VOL**: The volume value

## VM_MUSIC_ROUTINE

```
VM_MUSIC_ROUTINE ROUTINE, BANK, ADDR
```

Attach script to music event

- **ROUTINE**: The routine Id. An integer between 0 and 3.
- **BANK**: Bank number of the routine
- **ADDR**: Address of the routine

## VM_SFX_PLAY

```
VM_SFX_PLAY BANK, ADDR, MASK, PRIO
```

Play a sound effect asset

- **BANK**: Bank number of the effect
- **ADDR**: Address of the effect
- **MASK**: Mute mask of the effect
- **PRIO**: Priority of the sound effect. Effects with higher priority will cancel the ones with less priority:
  `.SFX_PRIORITY_MINIMAL` - Minmium priority for playback
  `.SFX_PRIORITY_NORMAL` - Normal priority for playback0
  `.SFX_PRIORITY_HIGH` - High priority for playback

### VM_MUSIC_SETPOS

```
VM_MUSIC_SETPOS PATTERN, ROW
```

Sets playback position for the current song.

- **PATTERN**: - The pattern to set the song position to
- **ROW**: - The row to set the song position to

## Projectiles

### VM_PROJECTILE_LAUNCH

```
VM_PROJECTILE_LAUNCH TYPE, IDX
```

### VM_PROJECTILE_LOAD_TYPE

```
VM_PROJECTILE_LOAD_TYPE TYPE, BANK, ADDR
```

## RTC

### VM_RTC_LATCH

```
VM_RTC_LATCH
```

Latch RTC value for access

### VM_RTC_GET

```
VM_RTC_GET IDX, WHAT
```

Read RTC value

- **IDX**: Target variable
- **WHAT**: RTC value to be read

`.RTC_SECONDS` - Seconds
`.RTC_MINUTES` - Minutes
`.RTC_HOURS` - Hours
`.RTC_DAYS` - Days

### VM_RTC_SET

```
VM_RTC_SET IDX, WHAT
```

Write RTC value

- **IDX**: Source variable
- **WHAT**: RTC value to be written
  `.RTC_SECONDS` - Seconds
  `.RTC_MINUTES` - Minutes
  `.RTC_HOURS` - Hours
  `.RTC_DAYS` - Days

### VM_RTC_START

```
VM_RTC_START START
```

Start or stop RTC

- **START**: Start or stop flag
  `.RTC_STOP` - stop RTC
  `.RTC_START` - start RTC

## Rumble

### VM_RUMBLE

```
VM_RUMBLE ENABLE
```

Enables or disables rumble on a cart that has that function

- **ENABLE**: 1 - enable or 0 - disable

## Scenes

### VM_SCENE_PUSH

```
VM_SCENE_PUSH
```

Pushes the current scene to the scene stack.

### VM_SCENE_POP

```
VM_SCENE_POP
```

Removes the last scene from the scene stack an loads it.

### VM_SCENE_POP_ALL

```
VM_SCENE_POP_ALL
```

Removes all scenes from the scene stack and loads the first one.

### VM_SCENE_STACK_RESET

```
VM_SCENE_STACK_RESET
```

Removes all the scenes from the scene stack.

## Screen Fade

### VM_FADE

```
VM_FADE FLAGS
```

### VM_FADE_IN

```
VM_FADE_IN IS_MODAL
```

### VM_FADE_OUT

```
VM_FADE_OUT IS_MODAL
```

## SGB

### VM_SGB_TRANSFER

```
VM_SGB_TRANSFER
```

Transfers SGB packet(s). Data of variable length is placed after this instruction, for example:

```
VM_SGB_TRANSFER
.db ((0x04 &lt;&lt; 3) | 1), 1, 0x07, ((0x01 &lt;&lt; 4) | (0x02 &lt;&lt; 2) | 0x03), 5,5,
10,10,  0, 0, 0, 0, 0, 0, 0, 0 ; ATTR_BLK packet
```

SGB packet size is a multiple of 16 bytes and encoded in the packet itself.

# SIO

## VM_SIO_SET_MODE

```
VM_SIO_SET_MODE MODE
```

## VM_SIO_EXCHANGE

```
VM_SIO_EXCHANGE SOUR, DEST, SIZE
```

# Text Sound

## VM_SET_TEXT_SOUND

```
VM_SET_TEXT_SOUND BANK, ADDR, MASK
```

Set the sound effect for the text output

- **BANK**: Bank number of the effect
- **ADDR**: Address of the effect
- **MASK**: Mute mask of the effect

# Timer

## VM_TIMER_PREPARE

```
VM_TIMER_PREPARE TIMER, BANK, ADDR
```

Load script into timer context

## VM_TIMER_SET

```
VM_TIMER_SET TIMER, INTERVAL
```

Start a timer calling once every `INTERVAL` * 16 frames

## VM_TIMER_STOP

```
VM_TIMER_STOP TIMER
```

Stop a timer

### VM_TIMER_RESET

```
VM_TIMER_RESET TIMER
```

Reset a timers countdown to 0

## UI

### VM_LOAD_TEXT

```
VM_LOAD_TEXT NARGS
```

Loads a text in memory

- **NARGS**: Amount of arguments that are passed before the null-terminated string

The text string is defined using the `.asciz` command:

```
VM_LOAD_TEXT   0
.asciz "text to render"
```

### Displaying variables:

The following format specifiers allow to render variables as part of the text:

- `%d` Render a variable value
- `%Dn` Render a variable value with `n` length
- `%c` Render a character based on the variable value
  The variables need to be defined before the `.asciz` call using `.dw` followed by a list of `N` variables in the order they'll be rendered.

```
VM_LOAD_TEXT   3
.dw VAR_0, VAR_1, VAR_1
.asciz "Var 0 is %d, Var 1 is %d, Var 2 is %d"
```

### Escape Sequences:

The text string can contain escape sequence that modify the behavior or apparence of the text.

- `\001\x` Sets the text speed for the next characters in the current text. `x` is a value between `1` and `8` that represents the number of frames between the render of a character using `2^(x-2)`.
- `\002\x` Sets the text font
- `\003\x\y` Sets the position for the next character
- `\004\x\y` Sets the position for the next character relative to the last character
- `\005\` TBD
- `\006\mask` Wait for input to continue to the next character.

- `\007\n` Inverts the colors of the following characters.
- `\n` Next line
- `\r` Scroll text one line up

## VM_DISPLAY_TEXT_EX

```
VM_DISPLAY_TEXT_EX OPTIONS, START_TILE
```

Renders the text in the defined layer (overlay, by default)

- **OPTIONS**: Text rendering options:
  `.DISPLAY_DEFAULT` - default behavior
  `.DISPLAY_PRESERVE_POS` - preserve text position
- **START_TILE**: Tile number within the text rendering area to be rendered from; use .TEXT_TILE_CONTINUE to proceed from the current position

## VM_DISPLAY_TEXT

```
VM_DISPLAY_TEXT
```

Renders the text in the defined layer (obsolete)

## VM_SWITCH_TEXT_LAYER

```
VM_SWITCH_TEXT_LAYER LAYER
```

Changes the `LAYER` where the text will be rendered.

- **LAYER**:
  `.TEXT_LAYER_BKG` - Render text in the background layer
  `.TEXT_LAYER_WIN` - Render text in the overlay layer

## VM_OVERLAY_SETPOS

```
VM_OVERLAY_SETPOS X, Y
```

Set position of the overlay window in tiles

- **X**: X-coordinate of the overlay window in tiles
- **Y**: Y-coordinate of the overlay window in tiles

## VM_OVERLAY_HIDE

```
VM_OVERLAY_HIDE
```

Hide the overlay window

## VM_OVERLAY_WAIT

```
VM_OVERLAY_WAIT IS_MODAL, WAIT_FLAGS
```

Wait for the UI operation(s) completion

- **IS_MODAL**: indicates whether the operation is modal: .UI_MODAL, or not: .UI_NONMODAL
- **WAIT_FLAGS**: bit field, set of events to be waited for:
  `.UI_WAIT_NONE` - No wait
  `.UI_WAIT_WINDOW` - Wait until the window moved to its final position
  `.UI_WAIT_TEXT` - Wait until all the text finished rendering
  `.UI_WAIT_BTN_A` - Wait until "A" is pressed
  `.UI_WAIT_BTN_B` - Wait until "B" is pressed
  `.UI_WAIT_BTN_ANY` - Wait until any button is pressed

## VM_OVERLAY_MOVE_TO

```
VM_OVERLAY_MOVE_TO X, Y, SPEED
```

Animated move of the overlay window to the new position

- **X**: X-coordinate of the new position
- **Y**: Y-coordinate of the new position
- **SPEED**: speed of the movement:
  `.OVERLAY_IN_SPEED` - default speed for appearing of the overlay
  `.OVERLAY_OUT_SPEED` - default speed for disappearing of the overlay
  `.OVERLAY_SPEED_INSTANT` - instant movement

## VM_OVERLAY_SHOW

```
VM_OVERLAY_SHOW X, Y, COLOR, OPTIONS
```

Show the overlay window

- **X**: X-coordinate of the new position
- **Y**: Y-coordinate of the new position
- **COLOR**: initial color of the overlay window:
  `.UI_COLOR_BLACK` - black overlay window
  `.UI_COLOR_WHITE` - white overlay window
- **OPTIONS**: display options:
  `.UI_DRAW_FRAME` - draw overlay frame
  `.UI_AUTO_SCROLL` - set automatic text scroll area; text will be scrolled up when printing more lines than the overlay height.

## VM_OVERLAY_CLEAR

```
VM_OVERLAY_CLEAR X, Y, W, H, COLOR, OPTIONS
```

Clear the rectangle area of the overlay window

- **X**: X-coordinate in tiles of the upper left corner
- **Y**: Y-coordinate in tiles of the upper left corner
- **W**: Width in tiles of the rectangle area
- **H**: Height in tiles of the rectangle area
- **COLOR**: initial color of the overlay window:
  `.UI_COLOR_BLACK` - black overlay window
  `.UI_COLOR_WHITE` - white overlay window
- **OPTIONS**: display options:
  `.UI_DRAW_FRAME` - draw overlay frame
  `.UI_AUTO_SCROLL` - set automatic text scroll area; text will be scrolled up when printing more lines than the overlay height.

## .MENUITEM

```
.MENUITEM X, Y, iL, iR, iU, iD
```

## VM_CHOICE

```
VM_CHOICE IDX, OPTIONS, COUNT
```

Execute menu

- **IDX**: Variable that receive the result of the menu execution
- **OPTIONS**: bit field, set of the possible menu options:
  `.UI_MENU_STANDARD` - default menu behavior
  `.UI_MENU_LAST_0` - last item return result of 0
  `.UI_MENU_CANCEL_B` - B button cancels the menu execution
  `.UI_MENU_SET_START` - if set IDX may contain the initial item index
- **COUNT**: number of menu items

instruction must be followed by the COUNT of .MENUITEM definitions:
.MENUITEM X, Y, iL, iR, iU, iD
where:
`X` - X-coordinate of the cursor pointer in tiles
`Y` - Y-coordinate of the cursor pointer in tiles
`iL` - menu item number where the cursor must move when you press LEFT
`iR` - menu item number where the cursor must move when you press RIGHT
`iU` - menu item number where the cursor must move when you press UP
`iD` - menu item number where the cursor must move when you press DOWN

## VM_SET_FONT

```
VM_SET_FONT FONT_INDEX
```

Sets active font

- **FONT_INDEX**: the index of the font to be activated

## VM_SET_PRINT_DIR

```
VM_SET_PRINT_DIR DIRECTION
```

Sets print direction

- **DIRECTION**: direction of the text rendering:
  `.UI_PRINT_LEFTTORIGHT` - text is rendered from left to right (left justify)
  `.UI_PRINT_RIGHTTOLEFT` - text is rendered from right to left (right justify)

## VM_OVERLAY_SET_SUBMAP_EX

```
VM_OVERLAY_SET_SUBMAP_EX PARAMS_IDX
```

Copies rectange area of the background map onto the overlay window

- **PARAMS_IDX**: points to the beginning of the pseudo-structure that contains these members:
  `x` - X-coordinate within the overlay window in tiles
  `y` - Y-coordinate tithin the overlay window in tiles
  `w` - Width of the copied area in tiles
  `h` - Height of the copied area in tiles
  `scene_x` - X-Coordinate within the background map in tiles
  `scene_y` - Y-Coordinate within the background map in tiles

## VM_OVERLAY_SCROLL

```
VM_OVERLAY_SCROLL X, Y, W, H, COLOR
```

Scrolls the rectangle area

- **X**: X-coordinate of the upper left corner in tiles
- **Y**: Y-coordinate of the upper left corner in tiles
- **W**: Width of the area in tiles
- **H**: Height of the area in tiles
- **COLOR**: Color of the empty row of tiles that appear at the bottom of the scroll area

## VM_OVERLAY_SET_SCROLL

```
VM_OVERLAY_SET_SCROLL X, Y, W, H, COLOR
```

Defines the scroll area for the overlay. When the text overflows that area it'll scroll up by 1 row

- **X**: X-coordinate of the upper left corner in tiles
- **Y**: Y-coordinate of the upper left corner in tiles
- **W**: Width of the area in tiles
- **H**: Height of the area in tiles
- **COLOR**: Color of the empty row of tiles that appear at the bottom of the scroll area

## VM_OVERLAY_SET_SUBMAP

```
VM_OVERLAY_SET_SUBMAP X, Y, W, H, SX, SY
```

Copies a rectange area of tiles from the scene background

- **X**: X-coordinate within the overlay window of the upper left corner in tiles
- **Y**: Y-coordinate within the overlay window of the upper left corner in tiles
- **W**: Width of the area in tiles
- **H**: Height of the area in tiles
- **SX**: X-coordinate within the level background map
- **SY**: Y-coordinate within the level background map

# Building Your Game

## Play

Clicking the *Play button* in the top right of the *Project Editor* window will start a build of your game and once complete will open a new window where you can play your game. See Keyboard Shortcuts for details on how to play your game in the *Play Window*.

## Build Terminal

Clicking the *Project View Button* and selecting *Build & Run* will take you to the *Build Terminal* where you can see a log of the project build. You also get to this screen by clicking the *Play* button while a build is taking place. This screen will show you if there's any errors in your build to help you correct them.

## Build as ROM

Clicking the *Export button* and selecting *Export ROM* will build your game and create a ROM file in your project's build folder as `$PROJECT_ROOT/build/rom/game.gb`. You can play this ROM file in any compatible emulator such as OpenEMU or KiGB.

## Build and deploy for Web

Clicking the *Export button* and selecting *Export Web* will build your game and create a HTML5 web build in the folder `$PROJECT_ROOT/build/web`. You can upload this folder to any web server and navigate to the `index.html` file to play your game in a web browser. If you use a mobile or tablet web browser the game will also include touch controls.

If you zip the `build/web` folder you can upload it to Itch.io as a HTML game. In this case the recommended viewport size to use is `480px` x `432px`.

## Build for Pocket

Clicking the *Export button* and selecting *Export Pocket* will build your game as a `.pocket` file for use on Analogue Pocket devices.

To play your `.pocket` game:

- Create a folder at the root of a MicroSD card called `GB Studio`.
- Copy the `.pocket` file into the `GB Studio` folder
- Insert the MicroSD card into your Pocket device.
- From the Pocket menu choose `Tools` / `GB Studio` / `Play Creations` and select your file from the list.

## Troubleshooting

On macOS if you're having trouble building or running your game you may also need to install Apple's Command Line Tools by opening `Applications/Terminal.app` and entering the following command.

```
xcode-select --install
```

On Windows you may need to whitelist the application in your Anti Virus software to perform a build.

# Extending GB Studio

## Engine Eject

Engine Eject copies the GBVM game engine that GB Studio uses into a folder in your project, named assets/engine. You can edit these source files to your liking using an IDE to hav…

## Plugins

Plugins are a way to extend GB Studio and share reusable assets, create custom scripting events and even build engine modifications.

# Engine Eject

Engine Eject copies the GBVM game engine that GB Studio uses into a folder in your project, named `assets/engine`. You can edit these source files to your liking using an IDE to have more control over how your GB Studio game is built. This feature is only recomended for developers familiar with GBDK.

To use Engine Eject, click on *Game* at the top of the GB Studio window and navigate to the *Advanced* tab to show the *Engine Eject* button.

After clicking *Eject* your project will gain a new folder named `/engine` with the subfolders `/include` and `/src`.

## Reverting Files

To revert any GBDK file back to its GB Studio default, delete it from the `assets/engine` folder. Deleting the whole `assets/engine` folder ensures that all GBDK code reverts back to the GB Studio defaults. You can also do this by pressing *Engine Eject* again, which will overwrite your `assets/engine` folder with the GB Studio defaults.

## Compile Errors

If you have bugged or incompatible files in the `/engine` folder, GB Studio will not be able to build your game. Error messages can be found in the *Build & Run* window.

The error message will often explain which files have problems and point you to the line number where the problem was found, for example this error is showing that line 77 of `src/core/actor.c` is using a variable that has not yet been defined:

```
Compiling: src/core/actor.c
src/core/actor.c:77: error 20: Undefined identifier 'emote_offsets'
src/core/actor.c:77: error 22: Array or pointer required for '[]' operation
src/core/actor.c:77: error 47: indirections to different types assignment
```

These errors will not be caused by missing files. GB Studio refers to its default engine in place of any missing `assets/engine` files. Fixing or removing the files that caused the error will allow your game to build and run again.

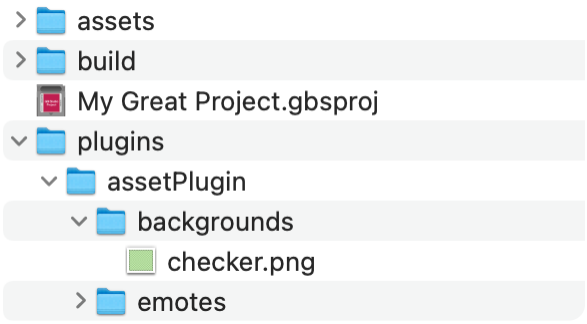
# Plugins

Plugins are a way to extend GB Studio and share reusable assets, create custom scripting events and even build engine modifications.

## Installing Plugins

To use plugins you must first create a `plugins` folder within your project in the same folder as your `.gbsproj` file. You can then place any plugins you have within this folder.

The structure should look something like this:




You may need to close and reopen your project after adding plugins for the changes to appear.

## Asset Plugins

The simplest kind of plugin you can make for GB Studio is an asset plugin, these allow you to share sprites, backgrounds, fonts, sounds, anything that is normally placed in the `assets` folder.

To create an asset plugin first create a new folder within your `plugins` folder with the name you want to give your plugin (Above we used `assetPlugin` as the name). Within that folder you can create any of the normal project `asset` folders (such as `backgrounds`) and place files within it. These assets will appear as normal in your project but are now easier to package up and share between projects or with others.

Download Example Asset Plugin

## Script Event Plugins

These plugins allow you to create new script events that will appear anywhere you use Scripting Events.

To create a script event plugin first create a new folder within your `plugins` folder with the name you want to give your plugin (Such as `myPlugin`). Within that folder create an `events` folder, and within that you can place the Javascript definition of your events. See the GB Studio source for examples of how these files should be structured and how they generate GBVM output. Note your event plugin Javascript filename MUST begin with `event` e.g. `eventMyFirstEvent.js`.

Download Example Script Event Plugin

## Engine Plugins

An engine plugin allows similar functionality to ejecting your engine but allows just changing single files or you can use it to add completely new files to the engine.

Engine plugins contain an `engine` folder which follows the same structure as an ejected game engine. Below you can download an example plugin that adds a new game engine function that causes the screen to flash (only when Color mode is disabled) and also includes a script event plugin to allow calling the new function.

Download Example Script Event Plugin

# Migration Guide

GB Studio 3.0 introduces a number of changes to previous versions in an effort to improve and future proof the game engine and project format. While we try our best to automate as much of the migration as possible there are a few instances where it was not possible to do that this time and you may need to make some changes to your project if you wish to migrate from previous GB Studio versions.

## Actors

- Actors will now always animate while stationary (allowing idle animations), which may cause issues when you want to step through animations manually (like checkboxes in the GB Studio 2.0 sample game menu scenes), if you wish to control an animation manually as before, set the Actor's animation speed to "None". You should also consider using the new Sprite Editor and Animation States as you can accomplish similar goals with a lot more flexibility.

- If you have many actors in a scene that use `Actor Set Sprite Sheet` events you may find your sprite tile counter has become too high. This is because in GB Studio 3.0 we made a different tradeoff in how to handle this situation, previously all scripted sprite sheets needed to be loaded into memory as the scene initialised limiting how many unique sprites could be used in a single scene, instead we now reserve memory for every actor that uses scripted sprite sheets but you can apply as many sprite sheets as you want to a single actor. The recommended solution is to replace switching sprite sheets with using Animation States instead. For an example of a scene affected by this compare the scene "Space Battle" from the GBC Sample Project in GB Studio 2 with the version in GB Studio 3, where ship explosion animations are now part of the enemy sprite animations rather than a separate sprite sheet.

- If you are migrating from GB Studio 2 you may notice the per scene actor limits is now reduced to 20 actors per scene, this may increase in future releases. Depending on how you were using actors you may be able to use larger sprites to achieve the same effect.

## Sprites

- The default player sprite is now set per scene type (*Top Down 2D*, *Platformer*, etc), so there is no need to switch to a different player sprite manually anymore in the scene init script, unless you wish to do so conditionally. When migrating a project using multiple scene types you will need to set the default player sprite for each scene type from the Settings View.

- Collision bounding boxes can now be configured per sprite. Previously all actors had a collision box of `16px` x `16px` and the player had a collision box of `8px` x `16px`. When migrating your project we set the spritesheet you set as the player default to use a `8px` x `16px` collision box to maintain compatibility with previous versions but if you ever changed the player sprite though scripts you may also need to set the collision boxes on these sprites manually using the Sprite Editor.

- Platformer player sprites now have a custom jump and climb animation which you will need to configure. To use these go into the Sprite Editor, select your platform player sprite, and in the right hand sidebar set the animation type to "Platform Player" which adds a few more animations you can define for the sprite, see Animation Settings for more information.

## Scenes

- Ladder tiles now snap the player sprite to the center of the tile while climbing. If you are using ladders in your game, make sure to test them as you may need to reposition the collision tiles to match the new alignment.

## Save / Load

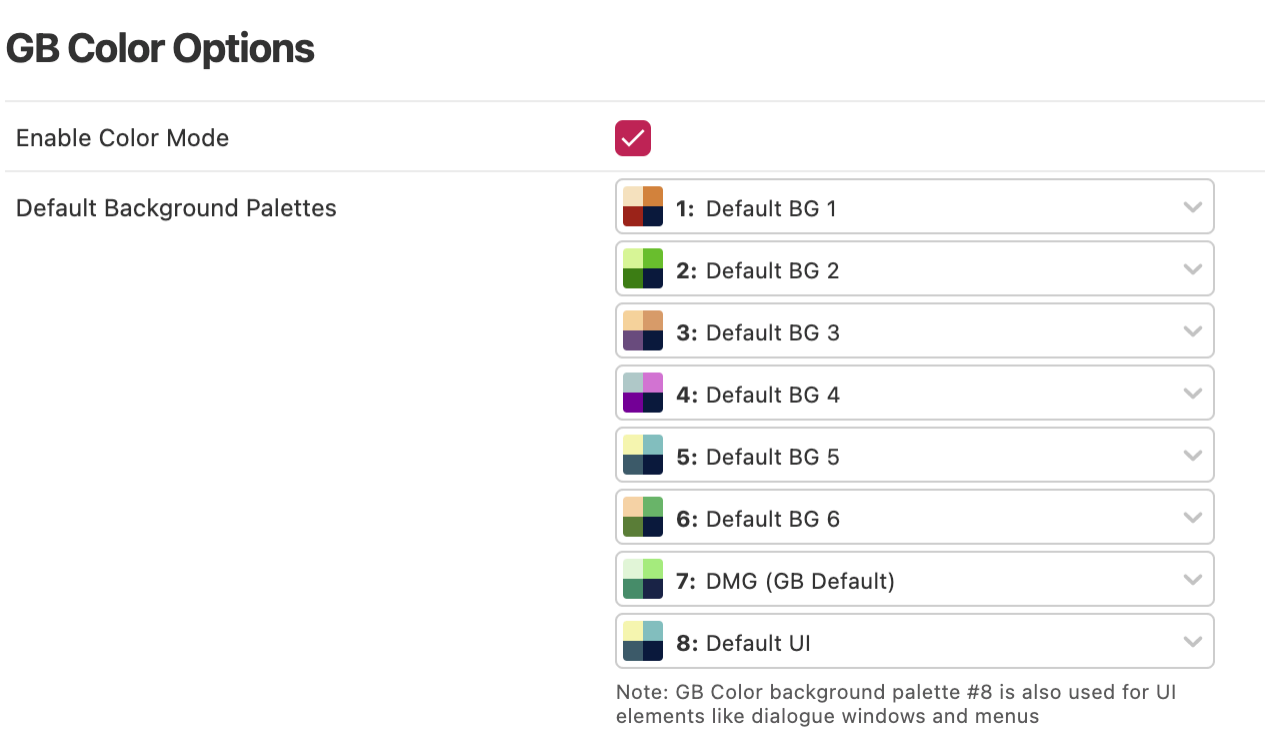
- When loading a save game, the game engine now continues any scripts that were previously running. This means that if you included a message such as “It is now safe to turn off your system.” immediately after the save it will also be shown when loading that game. The save data event now use an *On Save* callback, this will only be called when you save, and not when you load the game back. If you were previously displaying a message after saving you will likely need to move it into the *On Save* script. See the save points in the latest example projects for how to implement this.

# Settings

Clicking the *Project View Button* and selecting *Settings* will take you to a list of your project's settings.

## GB Color Options

GB Studio has support for GB Color when your game is run on compatible hardware or emulators. Click the `Enable Color Mode` checkbox to enable.

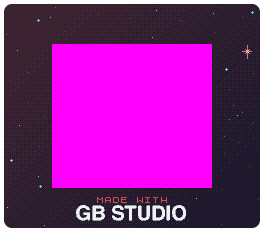



Once color mode is enabled you can select up to 8 Default Background Palettes and 8 Default Sprite Palettes, these are the palettes that every new scene in your game will use unless you specifically override them. See Colorizing a Scene for how to use background palettes.

## Super GB Options

To enable support for Super GB click the `Enable Super GB Mode` checkbox.

## Super GB Options

| | |
|---|---|
| Enable Super GB Mode | ☑ |
| Default Palette | Top Down Ship |
| Border Image<br>Update by editing /assets/sgb/border.png |  |

This mode will allow you to set a custom `256px×224px` border image and color palette to use when your game is run on compatible hardware or emulators.

The first time you build your game after enabling this mode a default border image will be copied to your project in `assets/sgb/border.png`, edit this image to replace the border with your own.

## Default Player Sprites

Each *Scene Type* can have a different player sprite sheet, use this section to replace the default spritesheet that will be used for each type. You can override the sprite sheet used for individual scenes by editing the scene's properties or by using scripts, see The Player.

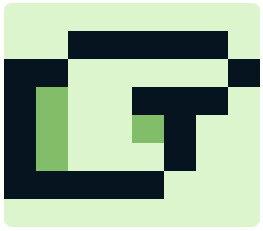
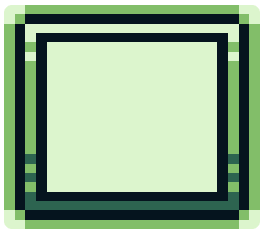
## Default Player Sprites

| | |
|---|---|
| Top Down 2D | player |
| Platformer | player_platform |
| Adventure | player |
| Shoot Em' Up | player_ship |
| Point and Click | cursor |
| Logo | player |

## UI Elements & Fonts

Use this section to view the frame image used for dialogue windows in your game, the cursor image used in menus and to select the default font for your project. Clicking the cursor or frame image will open them in your selected image editor, alternatively you can find the files in your project's `assets/ui` folder.

## UI Elements

| | |
|---|---|
| Default Font | T GBS Variable Width |
| Cursor Image |  |
| Frame Image |  |

Fonts can be found in `assets/fonts` and consist of a `.png` image and `.json` definition file.

You can create variable width fonts (with characters less than 8px wide) by filling the right edge of your font's characters with magenta `#ff00ff` like the example below.

By default fonts use an ASCII mapping with character code 32 (Space) mapping to the top left character in your font. You can provide a custom mapping by editing your font's `.json` file as follows.

```
{
  "name": "Japanese Font",
  "mapping": {
    "ヲ": 166,
    "ァ": 167,
    "ィ": 168
  }
}
```

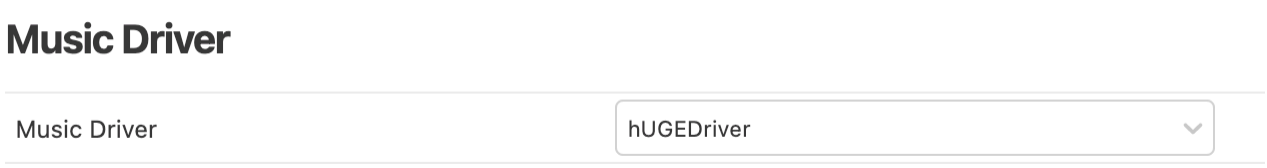
In this example using a `ヲ` character will now display character 166 from your `.png`.

**Please note** that as the first 32 ASCII characters are not included in your image you need to account for this in your mapping, for example if you wanted to map the character `?` to the second tile in your `.png` you would set the mapping to be `"?": 33`

## Music Format

The music format chooses which music engine to use in your game, this in turn determines the format of the music files supported in your project.

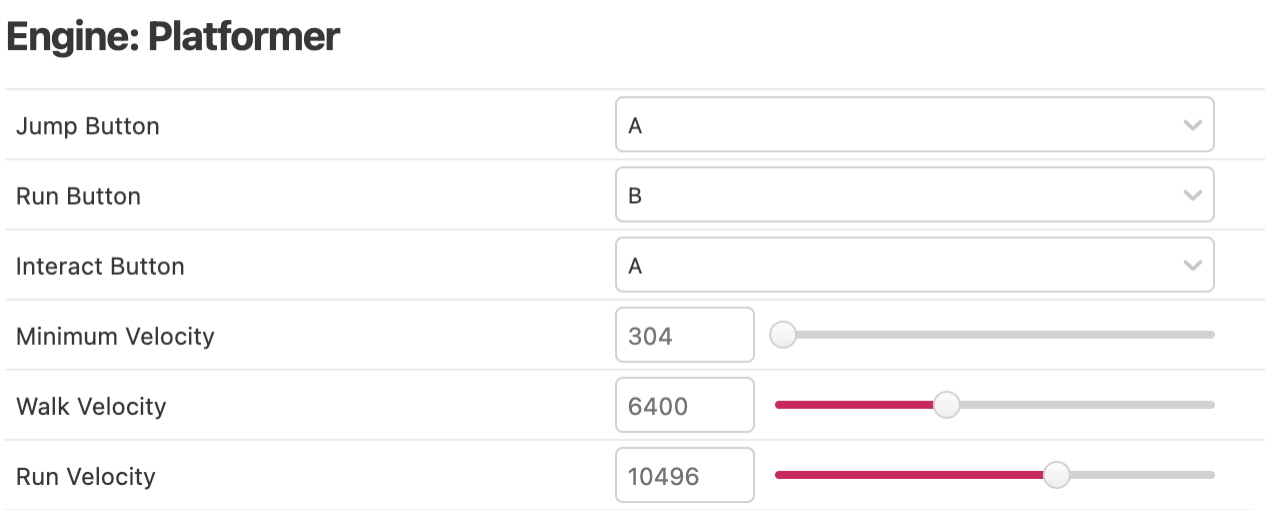
The recommended setting is `UGE (hUGEDriver)` (`.uge` files), as this enables you to use the inbuilt music editor, though if you have created a project in GB Studio 2.0 or below you will need to keep this setting as `MOD (GBT Player)` (`.mod` files) to maintain compatibility with your existing music files.




See Music for more information.

## Engine Settings

The GB Studio game engine has a number of custom settings split by scene type that you can change to adjust the feel of your game, for example to reduce the gravity in *Platformer* scenes or to make *Top Down 2D* scenes use a 16px grid.




To reset to the original values you can use the *Restore Default* button.

## Controls

The *Controls* section allows you to override the default controls used when playing your game from a web build and the *Play Window*.

To edit the controls for a button click on the input box and while the input is highlighted type the key you wish to assign. To remove all the assigned keys click the input and then press the *Backspace* key on your keyboard.

## Controls

| | |
|---|---|
| Up | ArrowUp, w |
| Down | ArrowDown, s |
| Left | ArrowLeft, a |
| Right | ArrowRight, d |
| A | Alt, z, j |
| B | Control, k, x |
| Start | Enter |
| Select | Shift |

Restore Default

To reset to the original controls you can use the *Restore Default* button.

## Cartridge Type

The *Cartridge Type* section allows you to choose which Memory Bank Controller you want to use and if you want to enable Batteryless Saving for compatible Flash Carts.

If you don't know what these settings mean it's best to keep this as the default of MBC5, with Batteryless disabled which you can do by using the *Restore Default* button.

## Custom HTML Header

You can use the *Custom HTML Header* section to add content to the HTML `<head>` when generating a web build of your game. You can use this to add any custom CSS or Javascript you want to the web build HTML page.