MD Engine

Documentation

142 chapters · generated 23 October 2026

Table of Contents

Home 6
- System Requirements 7
- Download & Installation 8
- Frequently Asked Questions 10
- MD Engine vs GB Studio 13
- Open Source Software Licenses 14

Getting Started 21
- UI Overview 23
  - Editor Tools 26
  - Navigator 29
  - Prefabs 32
  - Saving and Loading 38
  - Keyboard Shortcuts 39
  - Minimap 41
- Your First Project 43
- Your First Scene 48
- Your First Actor 50
- Templates & Demos 51
  - Action Platformer 53
  - Array 54
  - Audio 55
  - Auto Scrolling 56
  - Breakout 57
  - Camera Follow 58
  - Collisions 59
  - Co-op Play 60
  - Dialogue Codes 61
  - Dialogue Planes 62
  - Endless Runner 63
  - Game - Inkventure 64
  - Game - Prisonela MD 66
  - Game - Square Brothers 68
  - HBlank - Palette 70
  - HBlank - Window Plane 71
  - Image 72
  - Invaders 73
  - Line Scroll Wave 74
  - MDSDRV 75
  - PAL vs NTSC 76
  - Palette Cycling 77
  - Parallax 78
  - Particles 79
  - Pathfinding 80
  - Point & Click Adventure 81

- Projectiles 82
- Raster Gradient 83
- Save 84
- Shared Sprites 85
- Vertical Shooter 86
- Slopes 87
- Snake 88
- Temporary Triggers 89
- Text 90
- Tile Animation 91
- Top-Down 92
- Transitions 93
- Tweens 94
- Visual Novel 96
- Wall Kick 97

Testing Your First Game 98

Importing Scenes & Projects 100

Asset Types 103

Sprites 104

Backgrounds & Maps 109

- Tilesets 114
- Map Editor 117

Fonts 119

- Font Converter 122

Music 123

Sound Effects 124

Palettes 125

Localization 127

Image Converter 130

Asset Usage Report 132

The Engine 135

Scenes 137

- HBlank & VBlank 146
- Scene Notes 148

Actors 150

Triggers 155

- Temporary Triggers 157

Modules 158

- Arrays 161
- Behaviours 162
- Camera 165
- Collisions & Physics 167
- Dialogue & Text 171
- Localization 173
- Math 174
- Particles 175

- Pathfinding 176
- Projectiles 179
- Threads 180
- Timers 182
- Tweens 184

Engine Fields 186

Save System 188

Debugger 190

Settings 197

Build Process 211

Compiler Optimizations 214

Tools 221

Mini Painter 222

Mini Tracker 224

Mini Notator 233

Bfxr 247

Flash ROM to Cart 253

Scripting 258

Event Glossary 261

- Actors 265
- Array 291
- Behaviours 295
- Camera 297
- Collisions & Overlaps 302
- Control Flow 304
- Dialogue, Text & Menus 310
- Engine Fields 316
- Graphics 318
- Input 324
- Math 328
- Miscellaneous 330
- Music & Sound Effects 333
- Particles 337
- Pathfinding 340
- Platform 344
- Save Data 348
- Scene 351
- Screen 363
- Temp. Triggers & Projectiles 372
- Timer 378
- Tween 380
- Variables 383

Writing Plugins 386

Script Search 400

Async Events 403

Custom Scripts 404

Dialogue Variables 407
Math Expressions 409
Script Values 411
Variables 413
Exporting the Game 415
Implementation Support 421

Home

MD Engine is a visual engine to create games compatible with the Sega Mega Drive / Genesis consoles.

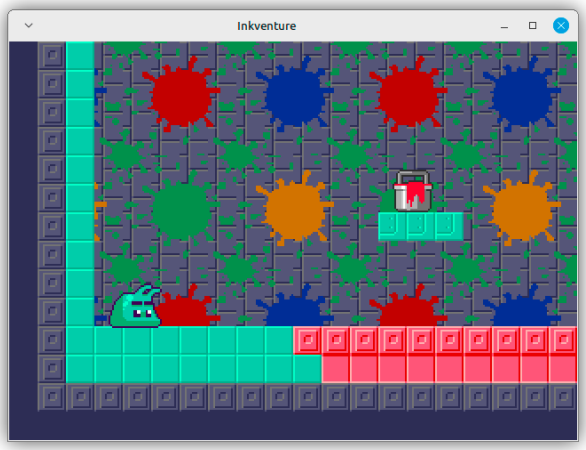
It's been created to enable users with minimal or no programming knowledge to make games. It uses a visual scripting system. The engine also features several tools to make it easier to convert images and fonts to be usable with the engine.

More advanced users can work directly with the engine C code and extend it with plugins.

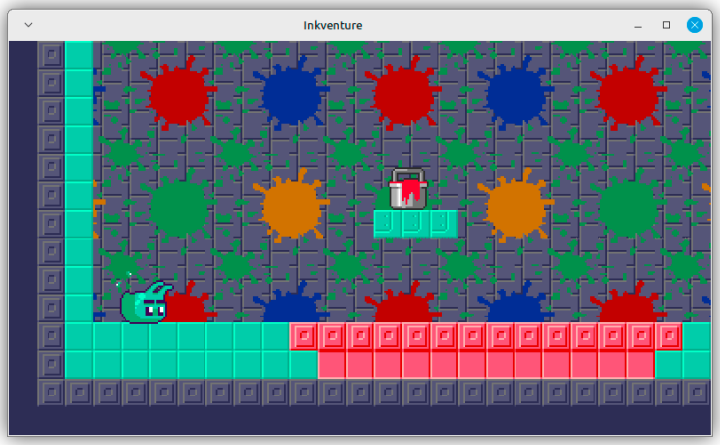
The engine can generate ROM files that can be run on an emulator or real hardware, as well as HTML5 web page versions, Windows Desktop versions, and Steam-enabled Desktop versions.

The engine also features several quality-of-life improvements if the games are run with the Desktop exports, such as wide-screen mode, removal of sprite limits, and a CRT effect, which can be enabled by the user at will.

Here you can see the Wide Screen mode (16:9 aspect ratio).



Normal



Wide Screen

MD Engine UI is based on the open-source engine GB Studio. You can find a full version of the licenses included in MD Engine here.

Finally, for GB Studio users, you can find an in-depth comparison of both engines here.

System Requirements

Minimum requirements for MD Engine on Steam:

Platform	OS and processor	Memory
Windows	Windows 10 (64-bit), Core 2 Duo	2 GB RAM
macOS	Apple silicon by default; Intel Mac build on a separate Steam branch	2 GB RAM
Linux	Kubuntu	2 GB RAM

Allow additional disk space for projects and exports.

Download & Installation

MD Engine is available on **Steam** and the **Two Black Cats store**.

- **Steam** — download it like any other software or game: MD Engine on Steam
- **Two Black Cats store** — buy a license and download the available builds from your account: MD Engine on twoblackcatsgames.com. This purchase does not include a Steam key; see the MD Engine EULA.

Standalone builds from the Two Black Cats store require license activation. Steam manages installation and access through the Steam client.

Platform Notes

Windows

Use Windows 10 or later (64-bit).

Linux

The standalone Linux build supports AppImage packaging. If your download includes an `.AppImage`, make it executable with `chmod +x` and run it. Replace `MD_Engine.AppImage` in the commands below with your downloaded filename.

Depending on your permissions and the path you are running the AppImage from, the system may not be able to mount or extract it. If you get errors like `AppImages require FUSE to run` or permission-related failures, you can extract the AppImage and run it directly:

```
./MD_Engine.AppImage --appimage-extract
./squashfs-root/AppRun
```

This extracts the contents into a `squashfs-root` folder, bypassing the need for FUSE or special mount permissions. This is also useful on systems where FUSE is not installed or when running from paths like network drives or `noexec`-mounted partitions.

If you are using a Wayland-based desktop and experience display issues, try launching with the `--ozone-platform-hint=auto` flag or setting the `GDK_BACKEND=x11` environment variable.

macOS

Steam installs the Apple silicon build by default. On an Intel Mac, select the Intel Mac branch in Steam. For a direct download, choose the build for your Mac. Open the `.app` bundle; if macOS blocks it, follow its instructions.

Troubleshooting

The editor launches but the screen is blank or white

Check that your system meets the published requirements and try updating your graphics drivers. If the problem persists, include your OS and editor version when reporting it.

Build fails with "compiler not found"

The build tools are bundled with MD Engine. On Steam, use **Properties > Installed Files > Verify integrity of game files**. For a standalone installation, reinstall the package from your account if bundled files are missing.

The Play Window doesn't open or shows a black screen

Check the **Build Log** for an export or emulator error, then restart the editor. Include the error text when reporting a problem.

General tips

- Check the System Requirements page to make sure your machine meets the minimum specs.
- If you experience performance issues with large projects, hiding unused scenes in the world view can help.
- The Build Log contains detailed information when something goes wrong during compilation.

Frequently Asked Questions

Here are some of the most common questions about MD Engine.

Is there a macOS version?

Yes. Steam has an Apple silicon build by default and an Intel Mac build on a separate branch. MD Engine can also export macOS apps (.app). See Exporting the Game for details.

> **INFO**
>
> macOS exports have not yet gone through Apple certification (App Store review). They are suitable for development and testing.

Can I export to mobile platforms?

Yes. MD Engine supports iOS and Android exports. iOS requires macOS with Xcode to build. Android produces an APK suitable for sideloading or distribution. See Exporting the Game for configuration details.

> **INFO**
>
> Mobile exports have not yet gone through Apple App Store or Google Play certification.

Can I export to handheld Linux devices?

Yes. MD Engine can produce PortMaster packages for handheld Linux gaming devices such as Anbernic, Miyoo, and others. See PortMaster Export for details.

> **EXPERIMENTAL**
>
> PortMaster support is experimental. The packaging format may change in future versions.

Will it support [new platform]?

MD Engine is 100% focused on generating Sega Mega Drive-compatible games, so we don't have plans to include any other retro console platforms in it. That said, we may be working on other retro engines! So it's best to check our website to know exactly what we're up to!

Will it support Sega CD / Sega 32X?

Right now, the focus is on the base system. The custom PCB we use allows for 8MB of unmapped access by disabling access to the Sega CD, so that's one of the reasons we are not focusing on the Sega CD.

If you want extra features, we recommend you export the project and use the generated source code with your preferred toolchain.

That said, we may add support for those later.

Can I sell games made with MD Engine?

Yes. Games you create with MD Engine are yours. With a standard license, you can distribute and sell them as physical cartridges or digital downloads. Educational licenses allow non-commercial use only. See the MD Engine EULA for details.

Can I run my game on real hardware?

Yes. MD Engine produces standard Mega Drive ROMs that work on real hardware. You can use a flash cart (such as an Everdrive or Mega Everdrive) or produce physical cartridges using the Two Black Cats custom PCB.

My game runs slowly / the CPU load is high

Open the Debugger and check the **CPU Profiler** tab. It shows exactly where CPU time is being spent. Common causes of high CPU load include:

- Too many actors with collision checks — try the **Gridding** collision mode or reduce the number of active actors.
- Complex scripts running every frame — add Idle or Wait events inside loops.
- Large pathfinding searches — use non-blocking mode and adjust the **Budget** setting.

My game doesn't build / I get a build error

Check the **Build Log** for the specific error. Common issues include:

- **Too many unique tiles** — Reduce the number of unique tiles in the scene or increase the VDP tile limit in Settings.
- **Too many actors or sprites** — Reduce the number of actors or sprite tiles in the scene.
- **Invalid event inside HBlank/VBlank** — Only specific events are allowed inside blank scripts.

How do I save and load game data?

Use the save data events. See the Save System documentation and the Save Data Event Glossary for details. There is also a Save Demo template.

How large can my ROM be?

The standard Mega Drive supports ROMs up to 4MB (32Mbit). With the Two Black Cats custom PCB, sizes up to 8MB are supported. You can monitor ROM usage in the Build Log.

Where can I find example projects?

MD Engine ships with several Templates & Demos covering platformers, top-down action, shooters, and individual feature examples. These are the best starting point for learning how the engine works.

MD Engine vs GB Studio

The UI for both MD Engine and GB Studio is pretty much identical, as MD Engine was forked from GB Studio with all target-platform limitations removed.

MD Engine received a full new compiler, new events, and new physics and audio systems.

The following table outlines some of the major differences.

Feature	MD Engine	GB Studio
Engine Core	SGDK	GBDK-2020
Runtime	Native C code	GBVM scripts
Palettes	9-bit RGB	15-bit RGB
Viewport Size	320 x 224 (H40) or 256 x 224 (H32)	160 x 144 pixels
Scene Backgrounds	Images or Tiled maps	Images
Scene Types	No	Yes
Global Player	No	Yes
Any Actor is "Player"	Yes	No
Triggers have several scripts	Yes	No
Scenes have Setup/Update/End scripts	Yes	No

Open Source Software Licenses

This page reproduces the component credits shown on the MD Engine splash screen. It is not an inventory of every package included in the editor. License files shipped with individual dependencies and build tools remain the source for their full terms.

```
Engine Core - SGDK
Copyright: 2025 Stephane Dallongeville
License: MIT
https://github.com/Stephane-D/SGDK


Original Sound Driver - MDSDRV
Copyright: 2019-2020 Ian Karlsson
License: Zlib
https://github.com/superctr/MDSDRV


Sound Driver improvements - MDSDRV Improvements
Copyright: homeless
License: Zlib
https://github.com/garrettjwilke/MDSDRV


Base Emulator - dgen-sdl
Copyright: 1998, 1999 Dave
License: 3-Clause BSD
https://dgen.sourceforge.net/


Motorola 68000 CPU - Musashi (M68000)
Copyright: 1998-2001 Karl Stenerud
License: MIT
https://github.com/kstenerud/Musashi


Z80 CPU - micro MSX2+
Copyright: 2023 Yoji Suzuki
License: MIT
https://github.com/suzukiplan/micro-msx2p


Debugger plane scrolling feature - Exodus
Copyright: 2024 Roger Sanders
License: MIT
https://github.com/RogerSanders/Exodus


Editor UI & Visual Scripting System - GB Studio
Copyright: 2019 Chris Maltby
License: MIT
https://github.com/chrismaltby/gb-studio


Sound Generator - bfxr2
Copyright: 2021 Stephen Lavelle
License: MIT
https://github.com/increpare/bfxr2


Mini Painter based on DPaint-js - DPaint-js
Copyright: 2023-2026 Steffest
License: MIT
https://github.com/steffest/DPaint-js


Image Converter - tiledpalettequant
```

```
Copyright: 2022 rilden
License: MIT
https://github.com/rilden/tiledpalettequant


References for the map editor - ascii-map-editor
Copyright: 2023 stmn
License: MIT
https://github.com/stmn/ascii-map-editor


UI for the Font Converter - font2bitmap
Copyright: 2023 stmn
License: MIT
https://github.com/stmn/font2bitmap


Emulator HTML file & reference - binjgb
Copyright: 2016 Ben Smith
License: MIT
https://github.com/binji/binjgb


Restart Icon - vscode-codicons
Copyright: Microsoft Corporation
License: MIT
https://github.com/microsoft/vscode-codicons


Import Icon - maryakveo
Copyright: Mary Akveo
License: Public Domain
https://maryakveo.com


Solid BG/FG Icon - maryakveo
Copyright: iconsax
License: Iconsax Free License
https://iconsax.io


Transparent BG/FG Icon - maryakveo
Copyright: iconsax
License: Iconsax Free License
https://iconsax.io


No FG Icon - phosphor-icons
Copyright: Phosphor Icons
License: MIT
https://github.com/phosphor-icons/phosphor-icons


Wall Icon - phosphor-icons
Copyright: Phosphor Icons
License: MIT
https://github.com/phosphor-icons/phosphor-icons


Wall Crossed Icon - phosphor-icons
Copyright: Phosphor Icons
License: MIT
```

```
https://github.com/phosphor-icons/phosphor-icons


Sprite Box Icon - pixelarticons
Copyright: Gerrit Halfmann
License: MIT
https://github.com/halfmage/pixelarticons


Sprite Box Crossed Icon - pixelarticons
Copyright: Gerrit Halfmann
License: MIT
https://github.com/halfmage/pixelarticons


Localization Icon - lucide
Copyright: Cole Bemis
License: Lucide License
https://lucide.dev


Build System - GNU Make
Copyright: 1988-2023 Free Software Foundation, Inc.
License: GPL-3.0
https://ftp.gnu.org/gnu/make/make-4.4.1.tar.gz


Build Shell & Utilities - BusyBox (busybox-w32)
Copyright: Erik Andersen, Rob Landley, Denys Vlasenko and others
License: GPL-2.0
https://github.com/rmyorston/busybox-w32


Build System Runtime (Windows) - libgcc (MinGW build of the GCC runtime)
Copyright: Free Software Foundation, Inc.
License: GPL-3.0 with GCC Runtime Library Exception
https://ftp.gnu.org/gnu/gcc/
```

Selected license texts follow. For other terms, use the component links above and the license files bundled with MD Engine.

MIT License

```
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE
```

3-Clause BSD License

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the authors nor the names of their contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

Zlib License

```
This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must
   not claim that you wrote the original software. If you use this
   software in a product, an acknowledgment in the product
   documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must
   not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source
   distribution.
```

Lucide License

```
ISC License

Copyright (c) for portions of Lucide are held by Cole Bemis 2013-2022 as part of Feather (MIT).
All other copyright (c) for Lucide are held by Lucide Contributors 2022.

Permission to use, copy, modify, and/or distribute this software for any purpose with or
without fee is hereby granted,  provided that the above copyright notice and this
permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.

IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT,
OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE,
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

GNU General Public License

GNU Make, BusyBox and libgcc are the build tools MD Engine invokes to compile a project, on every target. They are separate programs run as their own processes. They are not linked into the editor, nor into any game it produces, so they are aggregated with MD Engine rather than forming part of it.

`libgcc` here is the host runtime that GNU Make itself needs to run on Windows. It is not a compiler, and it is unrelated to the toolchain a project is built with: Mega Drive projects are compiled by SGDK's m68k GCC, while the other engines use LLVM.

They are distributed under:

- **GNU GPL version 2**: BusyBox
- **GNU GPL version 3**: GNU Make
- **GNU GPL version 3 with the GCC Runtime Library Exception 3.1**: libgcc

The full licence texts ship with the application under `buildTools/shared/LICENSES/`, together with a manifest naming the exact version, upstream source URL and SHA256 of every binary included.

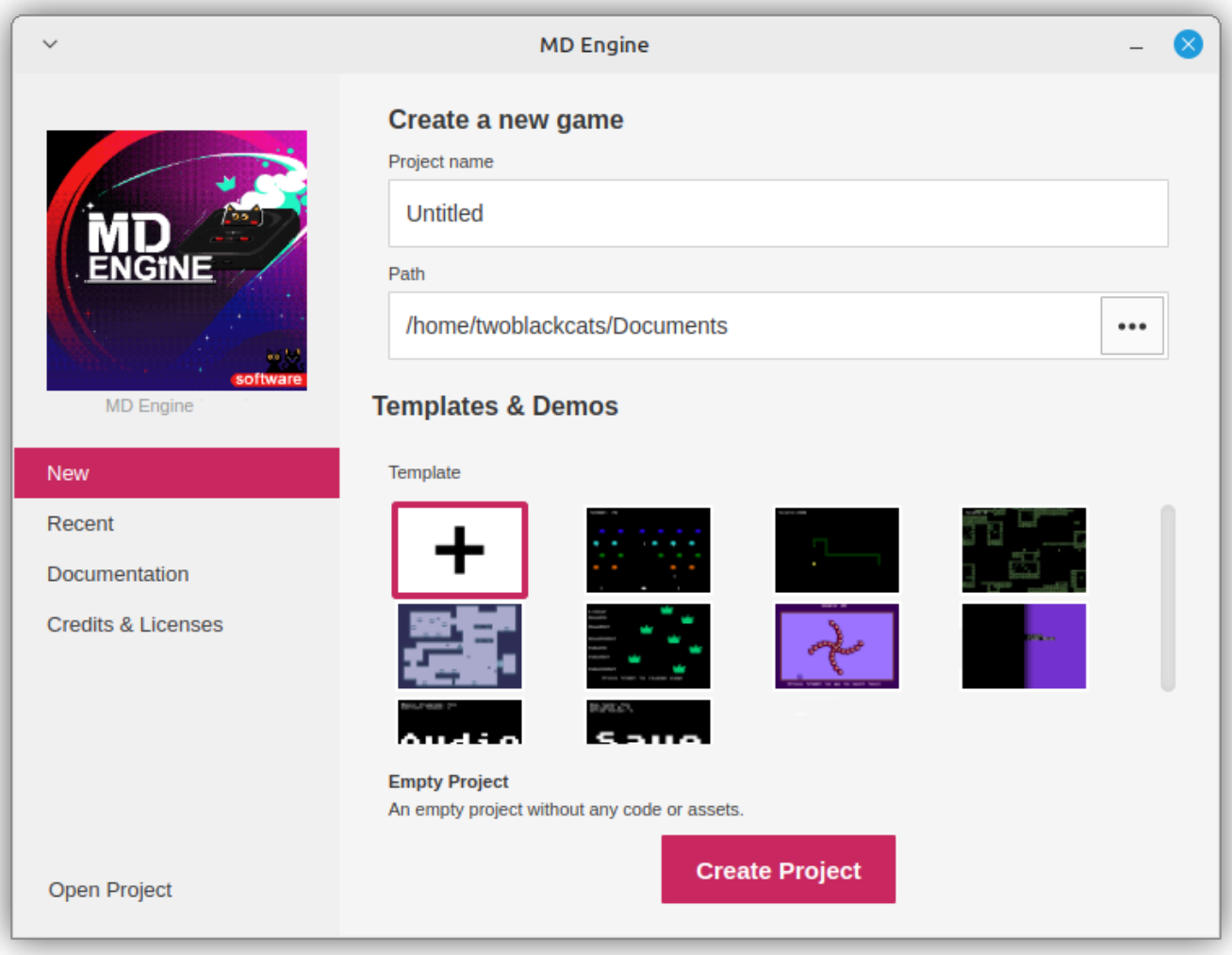
You are entitled to the complete corresponding source for these programs for at least three years from the date you received MD Engine. The upstream sources are:

- GNU Make 4.4.1: https://ftp.gnu.org/gnu/make/make-4.4.1.tar.gz
- BusyBox (busybox-w32): https://github.com/rmyorston/busybox-w32
- GCC (libgcc, host runtime for GNU Make): https://ftp.gnu.org/gnu/gcc/

Full texts: GPL-2.0, GPL-3.0, GCC Runtime Library Exception 3.1

Getting Started

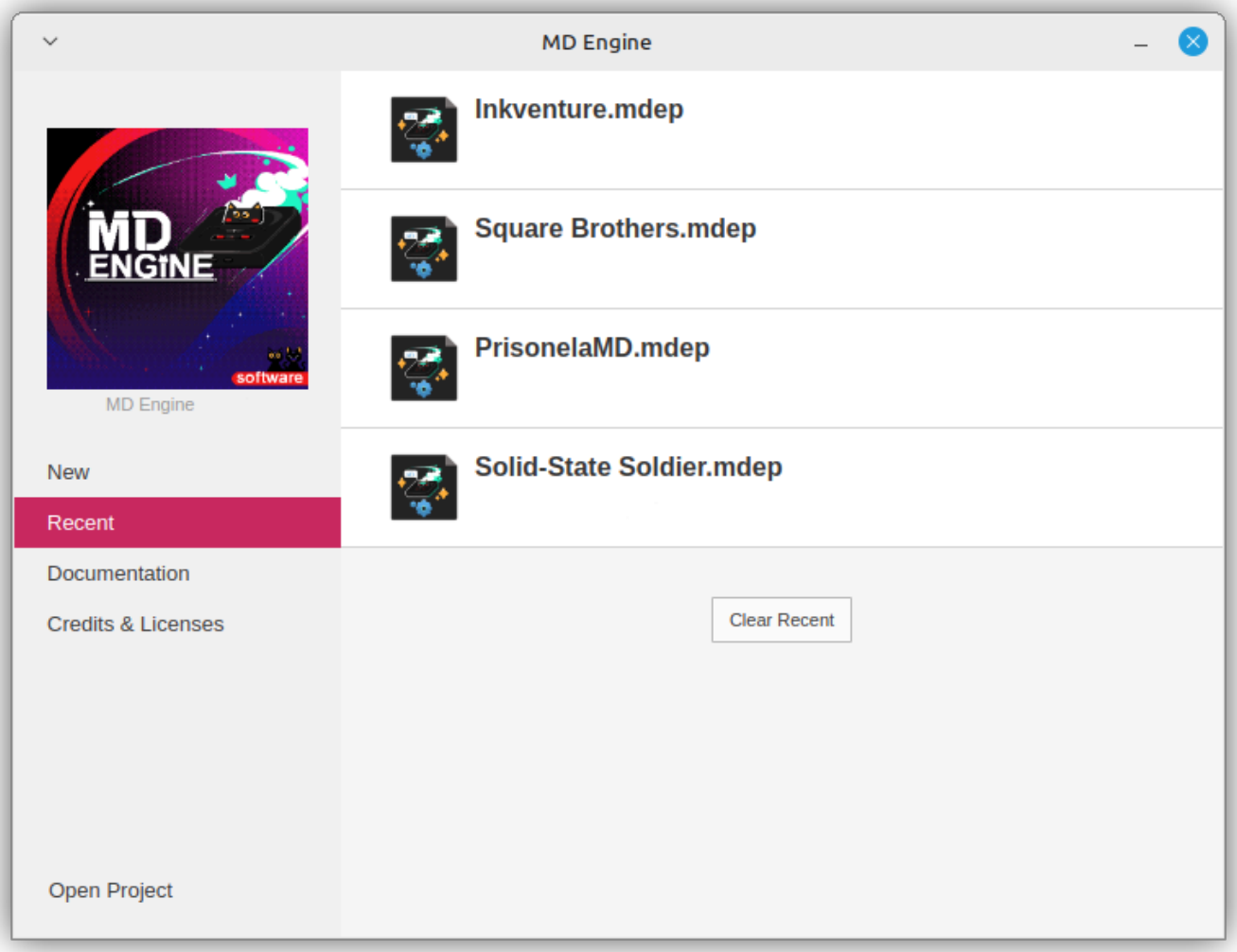
When you start working with MD Engine, the first time you open it, you will see the *New Project* window.



Here you can create an empty project or open any of the provided templates and demos to explore.

> **TIP**
>
> Before creating an empty project, we recommend you check our *First Project* guide as well as review a few templates and demos.

If you have created a project already, you will see the *Recent* menu; from there, you can open your past projects.



WARNING

Only the last 10 recent projects are kept.

If you press *Clear Recent* the list will be emptied.

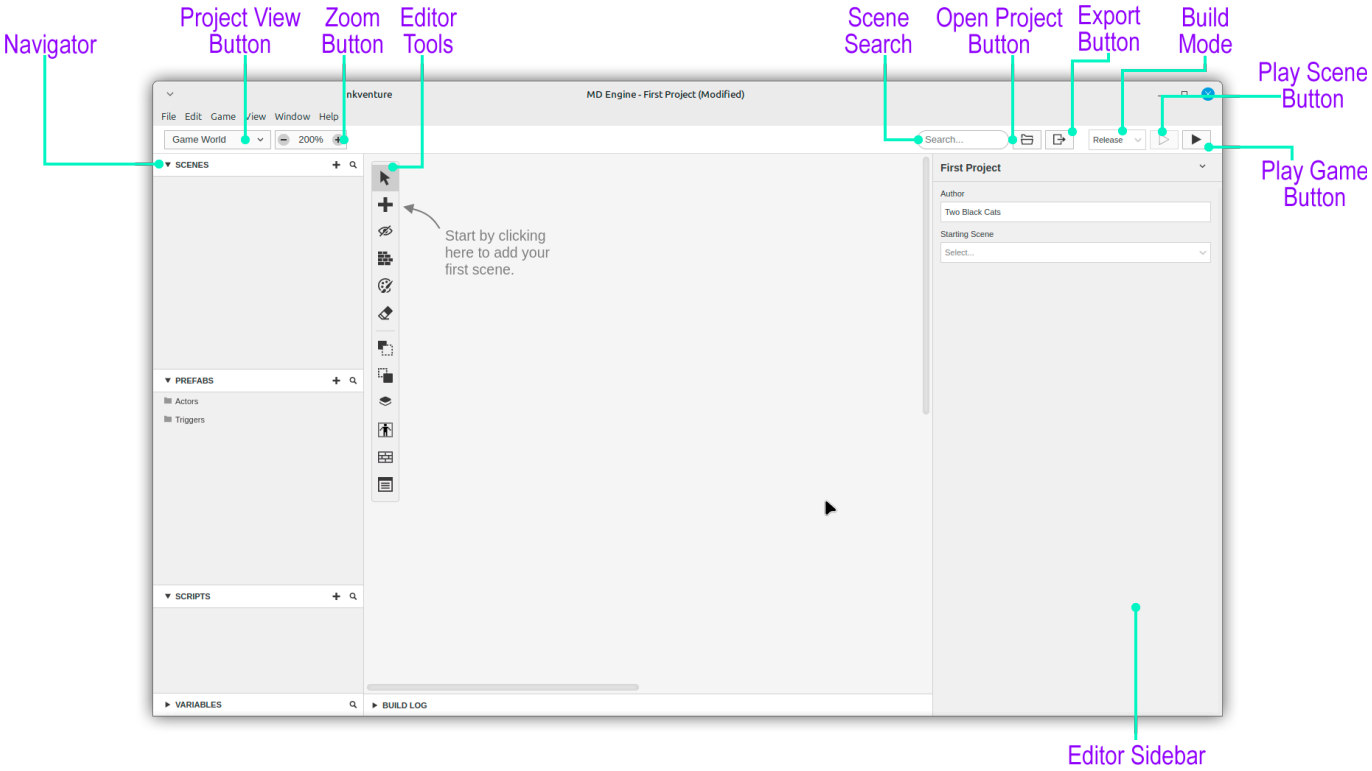
UI Overview

MD Engine UI is split into three different main areas:

- **Navigator** - Contains the list of elements on the current view.
- **Central Area** - Contains the element preview or main area of interaction.
- **Editor Sidebar** - Contains the list of all properties or settings we can change for the element selected on the *Navigator*.

All the MD Engine View areas share this UI layout.

Here you can see a summary of each button that you can see on all the areas.



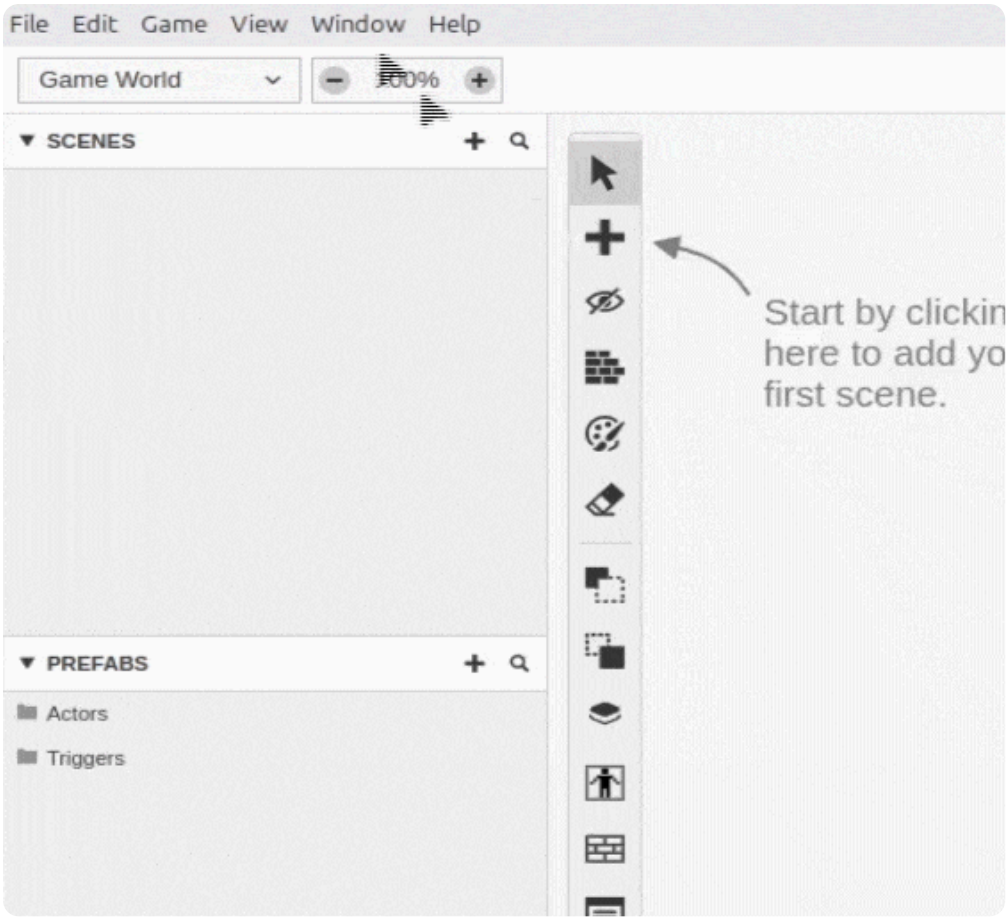
The *Navigator* area will show all the elements inside that specific View.
The Editor Sidebar will display the selected item.

When opening MD Engine, the first view for the *Project Editor* is the *Game World*.

Game World is where you can assemble your game with scenes, actors, and triggers.

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 preview animations.
- **Backgrounds & Maps:** Preview your background, maps and tileset assets.
- **Fonts:** Create and preview a font.
- **Music:** Preview your music files.
- **Sound Effects:** Preview your sound effects files.
- **Palettes:** Edit your palettes for games.
- **Localization:** Preview and edit all the text in your game, localizations, and all the used characters.
- **Debugger:** The game debugger, it will open when running a game.
- **Settings:** Change your project's settings such as maximum elements, VDP, collision settings, etc.

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

Detached Script Editor

You can detach the script editor into a separate window for more screen space when working with complex scripts. This is especially helpful on single-monitor setups where the sidebar can feel cramped.

To detach the script editor, select a scene, actor, or trigger and look for the **Detach Script Editor** button in the script editor header (it shows a detach/undock icon). Clicking it opens the script editor in its own floating window. Click the button again (now showing a dock icon) to re-attach it back into the main editor sidebar.

Editor Tools

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.

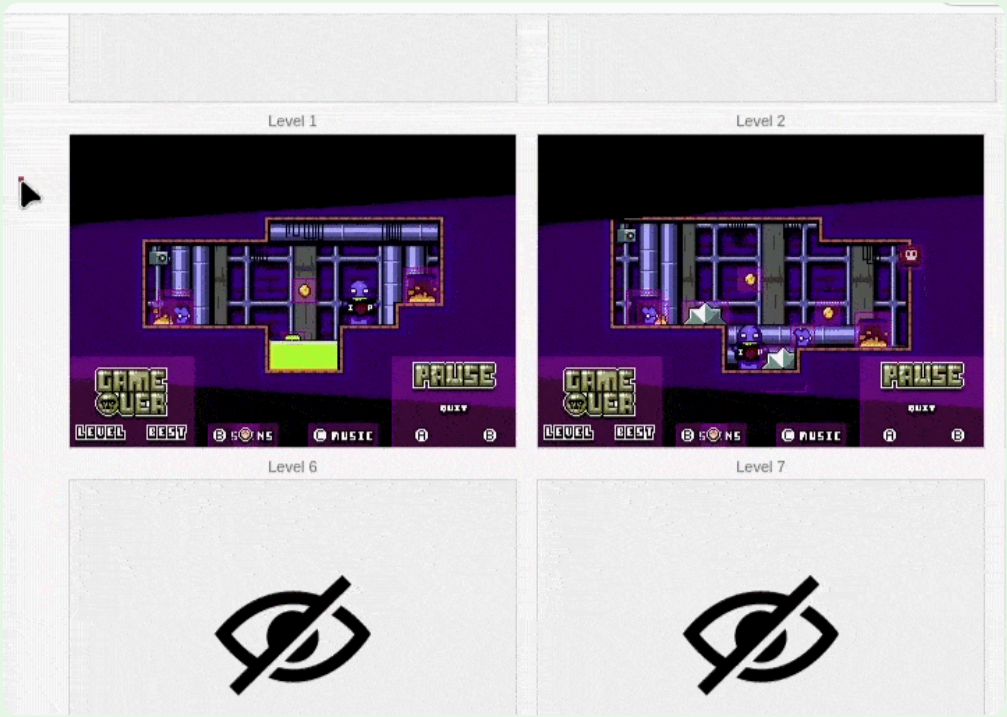
To look at project properties again from the *Editor Sidebar*, click on any empty space between scenes.

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.

SCENE MULTI-SELECT

If you hold `Shift` while clicking on scenes you can add and remove them from your selection. This allows you to drag multiple scenes together or right click to delete multiple at once. You can also hold `Shift` while dragging to create a box selection.



Multi-Select Actors & Triggers

You can select multiple actors and triggers within a single scene:

- **Ctrl/Cmd + Click** — Add or remove individual actors/triggers from the selection.
- **Shift + Drag** — Draw a selection rectangle to select all actors and triggers it intersects.

- **Ctrl/Cmd + A** — Select all actors and triggers in the focused scene.



With multiple entities selected you can:

- **Drag** any selected entity to move all of them together, maintaining relative positions.
- **Shift + Arrow keys** — Move all selected entities by one tile.
- **Delete** — Remove all selected entities at once.
- **Ctrl/Cmd + C** — Copy all selected entities.
- **Right-click** — Choose "Delete Selected (N)" to delete the group.

Selected actors show a dashed purple border, and selected triggers show a dashed orange border.

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. Erase actions can be undone by pressing `Ctrl / Cmd` + `Z`.

Scenes are not affected by *Erase mode*. To delete a scene either:

- Select the scene in *Select mode* then in the *Editor Sidebar* click the down arrow at the top and choose the "Delete Scene" menu item.
- Select the scene and press the `Backspace` key on your keyboard.
- Right click on the scene and click "Delete Scene" from the menu.

Collision tool

Allows you to add collisions to any type of scene using MD Engine's *Drawing mode*.

Palettes & Priorities tool

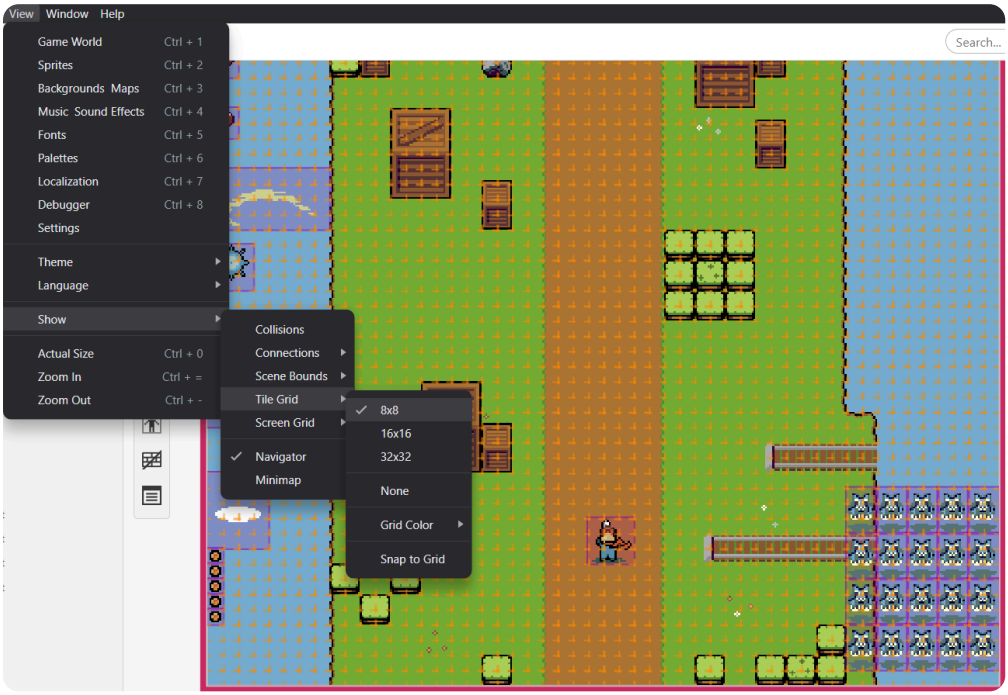
Allows you to change the priority or palette in your scene. The *Palettes & Priorities tool* also uses MD Engine's *Drawing mode*.

Tile Grid

The Tile Grid overlay displays a semi-transparent grid on the selected scene, helping you align actors, triggers, and collision tiles to the hardware tile boundaries.

To enable it, go to **View > Show > Tile Grid** and choose a grid size:

- **8x8** — Shows individual hardware tile boundaries.
- **16x16** — Shows metatile/block boundaries (useful for collision alignment).
- **None** — Disables the grid (default).

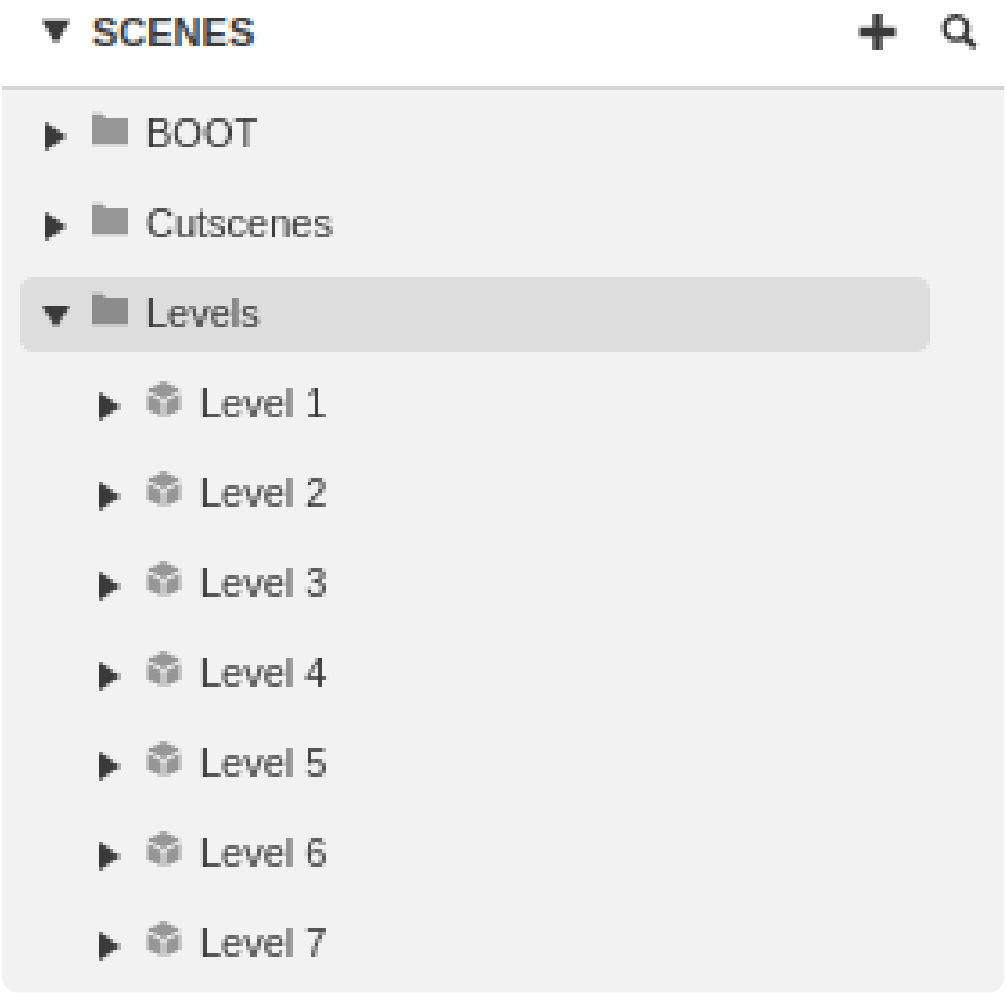


The grid appears only on the currently selected scene and does not affect gameplay or exports.

See the documentation on Keyboard Shortcuts for editor tool shortcuts.

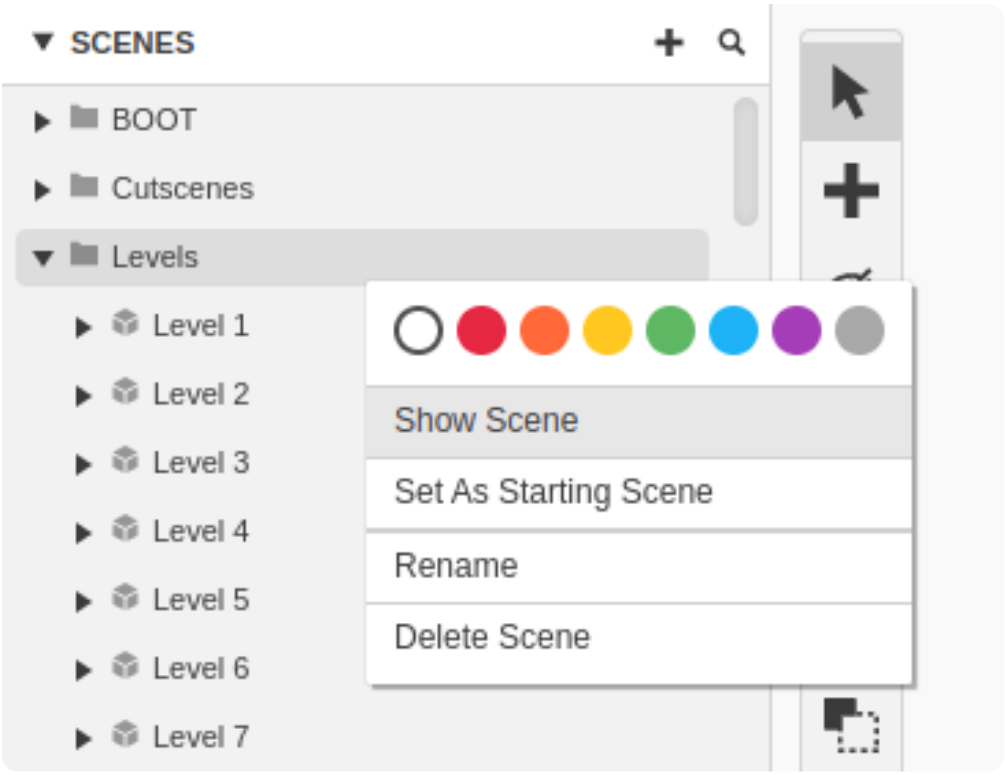
Navigator

The Navigator is the sidebar on the left which allows quickly navigating through your project.



Context Menu

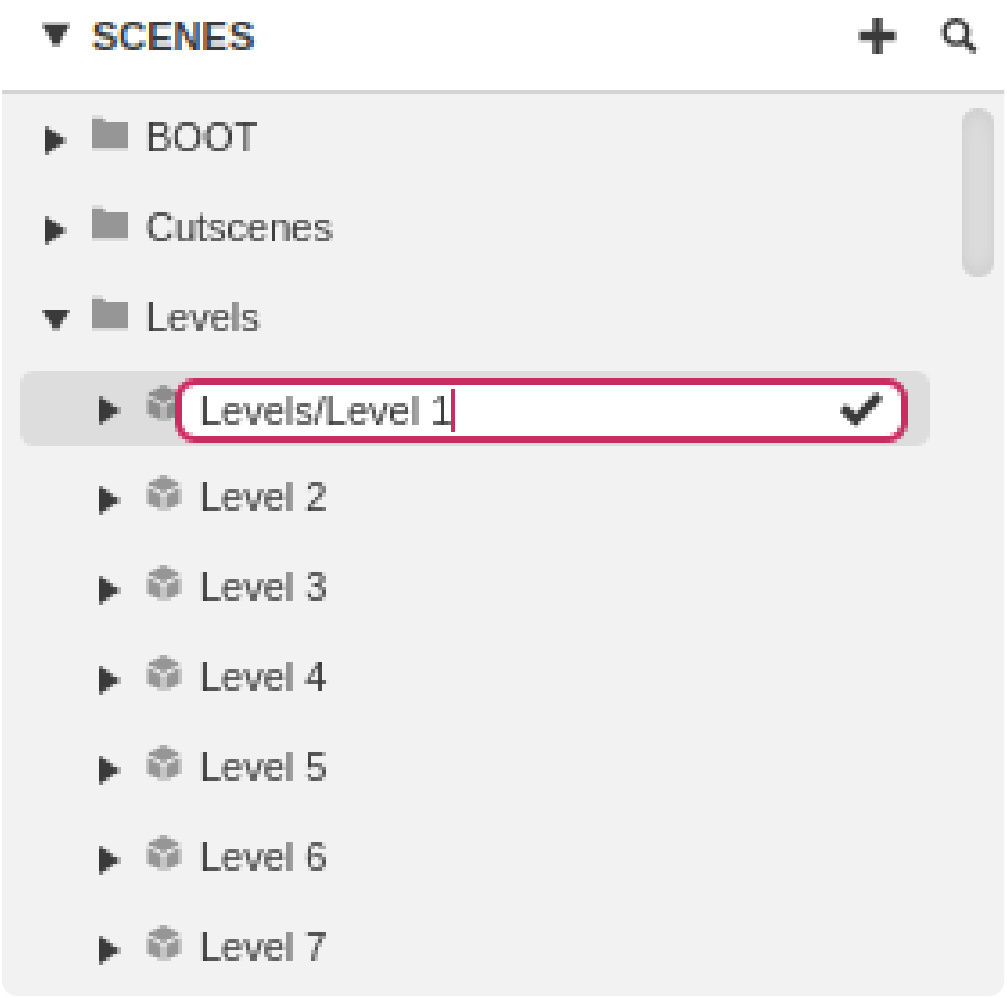
By right-clicking on items in the Navigator you can open a context menu.
The scene context menu shown below allows you to set color labels, set the scene as the starting scene for your project, rename and delete the scene.
You can also show or hide a scene, this will help on performance.



Folders

If your scenes, scripts, palettes or assets contain `/` or `\` characters in their names these will be treated as folders in the Navigator allowing you to organize your project.

e.g. Naming a scene `Levels/Level 1` as shown below:



will cause it to appear in the navigator as `Level 1` with a folder named `Levels`:

> **WARNING**
>
> When renaming assets like Backgrounds into folders using the Navigator, the physical files in your project (`.png` etc.) will also be moved to the newly specified folder structure.
> It's important to save your project after making changes like this as moving those files without saving could cause the application to be unable to locate assets when you next load your project.
>
> It is recommended that you use a version control system allowing you to revert mistakes.

Keyboard Shortcuts

Move selection up - `Arrow Up`
Move selection down - `Arrow Down`
Open folder - `Arrow Right`
Close folder - `Arrow Left`
Rename - `Enter`
Select multiple items (Scenes only) - `Shift` while clicking

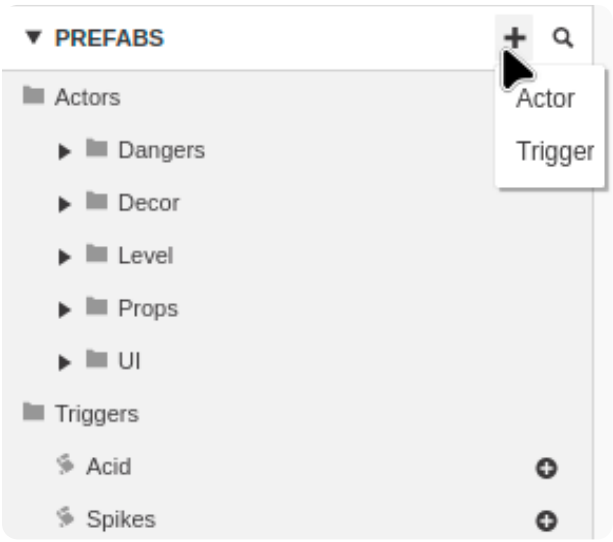
Prefabs

Prefabs are a useful feature to allow reusing actors and triggers across multiple scenes. Imagine them as templates.

Creating a Prefab

You can just click the + button, select *Actor* or *Trigger* depending on the game needs.

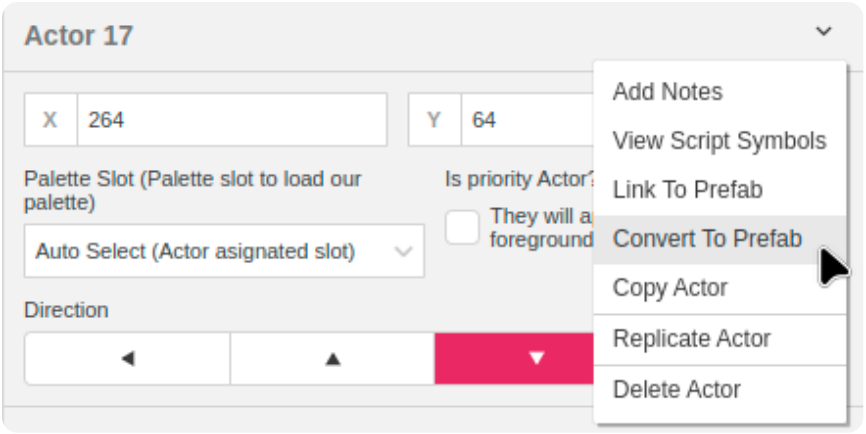
This will generate an empty prefab.



From Actor to Prefab

If there is an actor inside a scene that is not linked to a prefab, it can always be turned into a prefab later on.

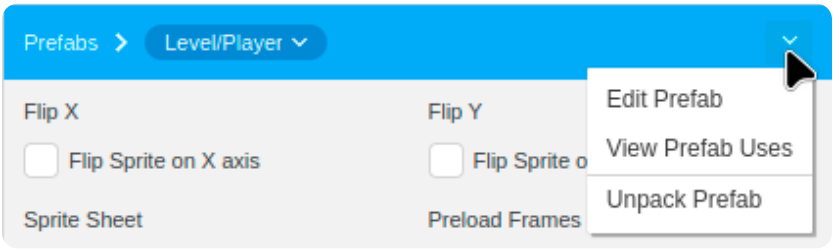
To do that, just go to the *Editor Sidebar* and select *Convert to Prefab*.



From Prefab to Actor

If there is a prefab on a scene and that prefab, for some reason, needs a lot of custom changes you can always unpack it from the prefab.

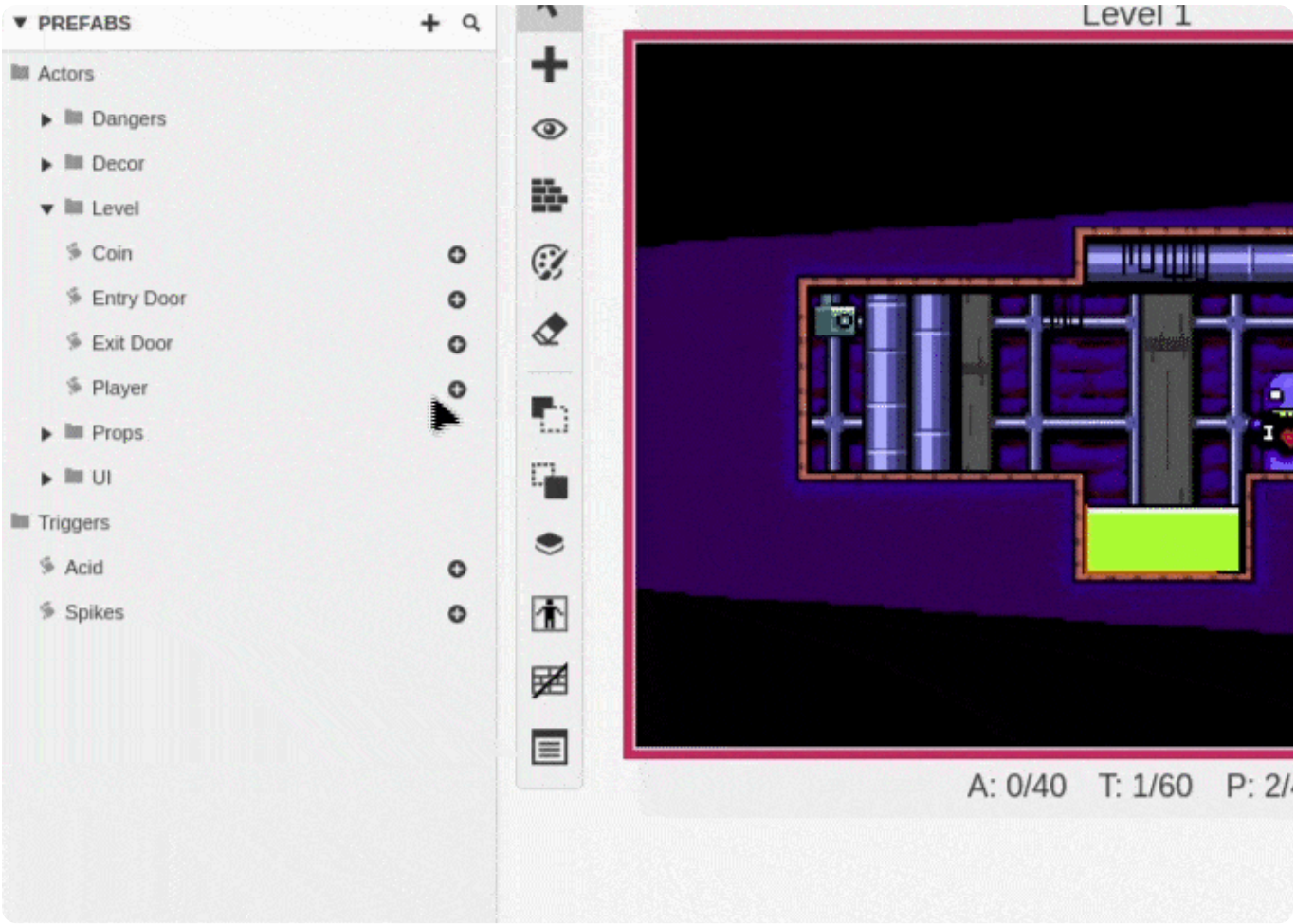
To do that, just go to the *Editor Sidebar*, go to the blue prefab area, and select *Unpack Prefab*.



Instancing a Prefab

To use a prefab inside a scene, just go to the *Navigator* and then click the  button of the desired prefab.

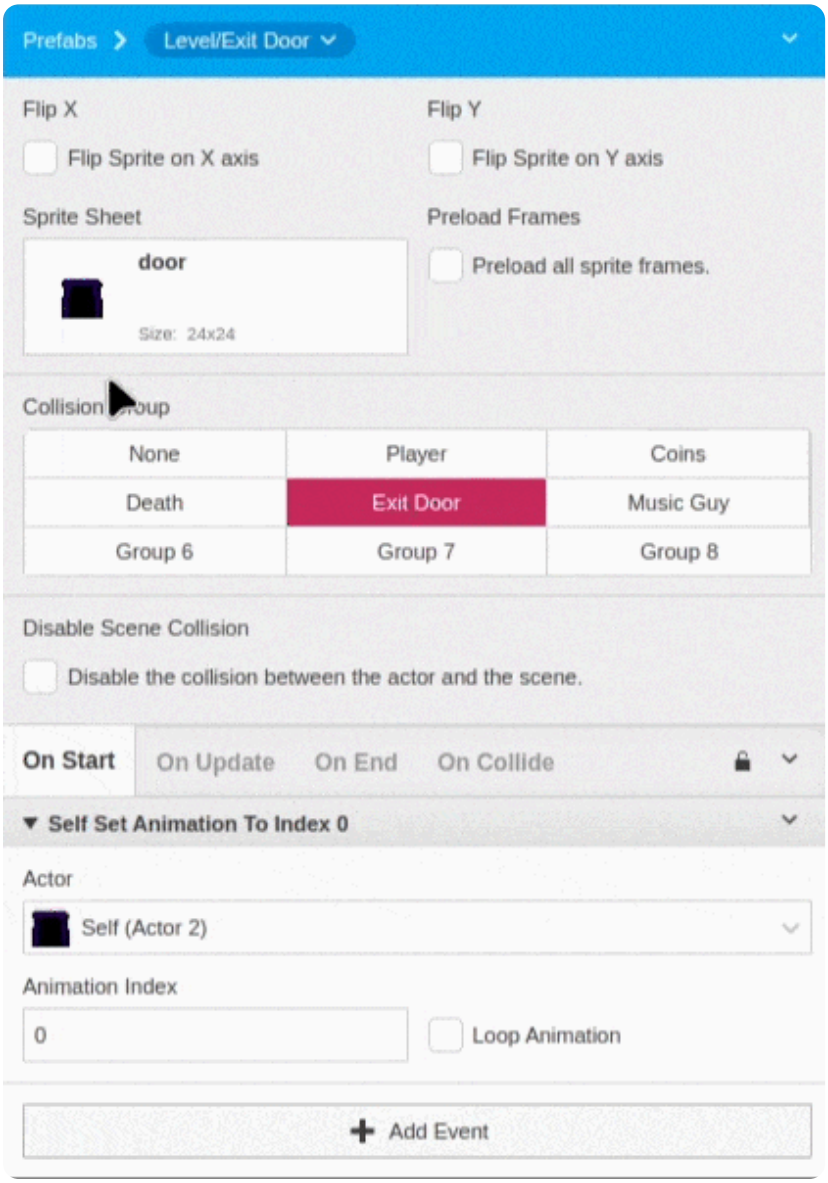
Then, the instanced *Actor* or *Trigger* can be placed on the scene.



Updating a Prefab

There are some instances when a prefab may vary slightly, like a different number on a starter value.

The values on the instanced prefab can be adjusted once it's instanced on a scene. The edited values will be displayed with an outer blue glow.

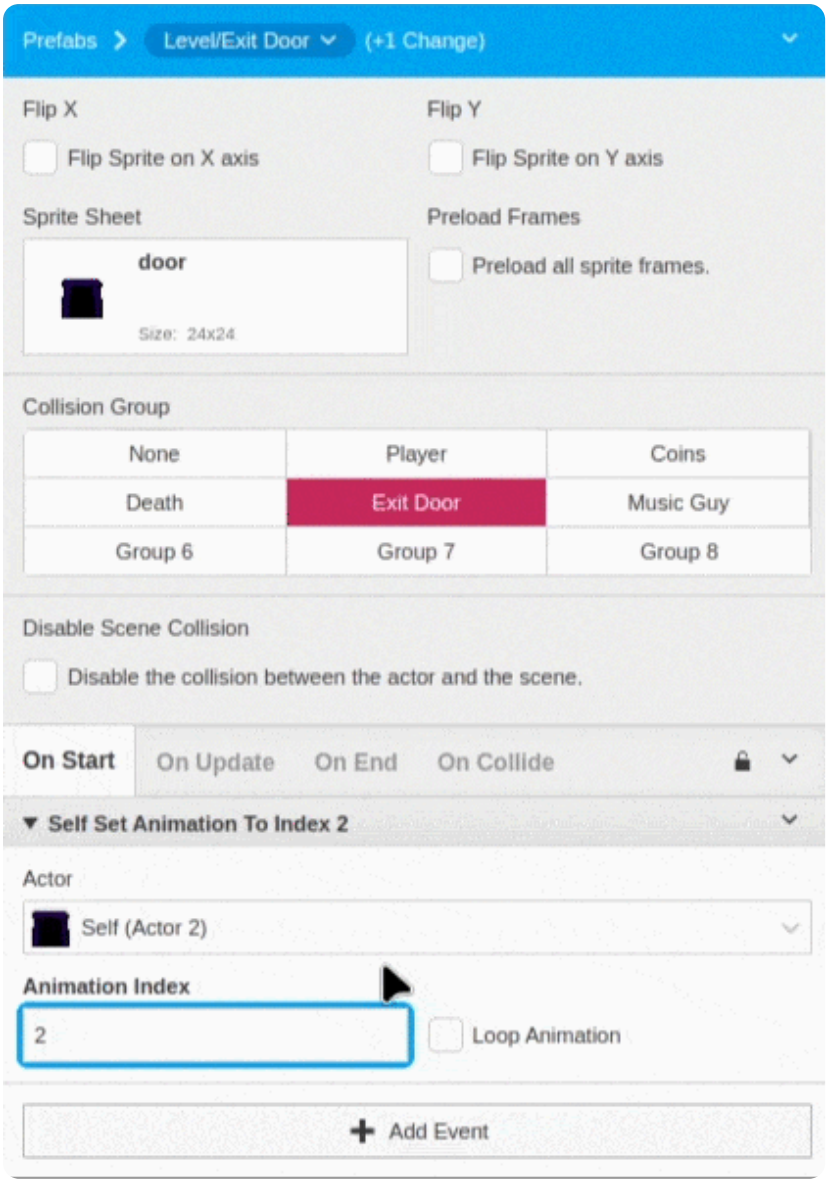


> **WARNING**
>
> Only values or references can be different, the events will **always** be shared.

Apply Changes

If you later want to apply your changes to the prefab, you can go to the *Editor Sidebar*, go to the blue prefab area, and select *Apply Changes*.

This will make the custom value the default one for this prefab and the outer blue glow will disappear.

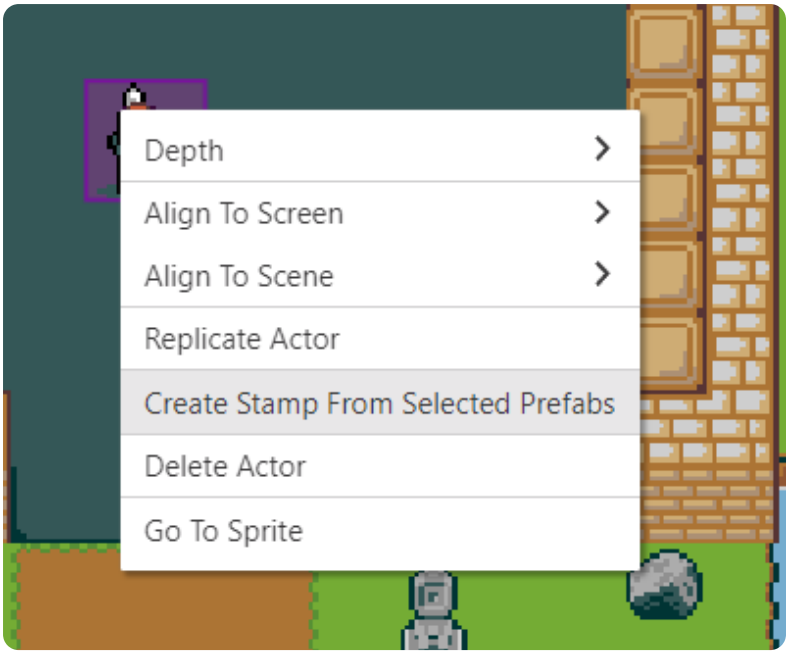


Stamps

Stamps are group templates that contain multiple actors and triggers arranged in a specific layout. Unlike individual prefabs, stamps capture the spatial arrangement and relationships between entities, letting you place pre-configured groups into any scene.

Creating a Stamp from Selection

1. Select one or more actors and/or triggers in a scene (using multi-select).
2. Right-click and choose **Create Stamp From Selection**.



The stamp captures each entity's relative position, sprite, scripts, and all settings.

Creating an Empty Stamp

In the *Navigator* sidebar under Prefabs, click the + dropdown and select **Stamp**. You can then configure the stamp manually in the Stamp Editor.

Placing a Stamp

1. In the Navigator, find your stamp under the Prefabs section.
2. Right-click and select **Place Stamp**.
3. Click on a scene in the world view to place all the stamp's entities.

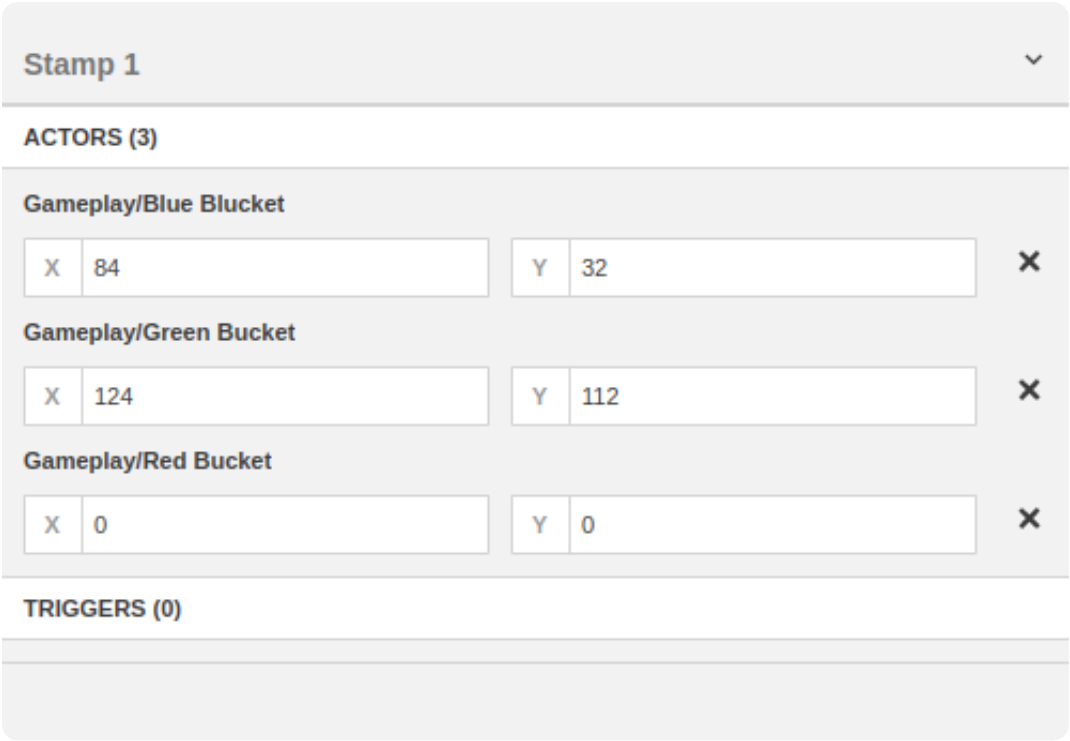


All actors and triggers are created at their relative positions from the click point. Actor references within scripts (e.g., "Move Actor" events referencing another actor in the group) are automatically remapped to point to the newly created copies.

Editing a Stamp

Click a stamp in the Navigator to open the **Stamp Editor** in the sidebar. Here you can:

- Rename the stamp.
- Add or edit optional notes.
- View and modify the list of contained actors (with positions).
- View and modify the list of contained triggers (with positions).
- Delete the stamp via the dropdown menu.



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 MD Engine will warn you giving you a chance to save your project first.

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 `.mdep` 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 `.mdep` 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 `.mdep.bak`. If you ever wish to roll back to the previous version in your project you can rename this file to have the extension `.mdep` and open this file instead.

Keyboard Shortcuts

Play Window

When playing your game inside MD Engine the controls depend on what's defined on the *Settings*, you can use both keyboard and gamepad.

Navigating The Menus

Much of the functionality of MD Engine 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` **Script Search** - `Ctrl/Cmd` + `Shift` + `F`

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`
Pan View - Hold `Space` (while clicking and dragging on *Game World*)
Zoom - Hold `Ctrl/Cmd` + `Scroll wheel`
Focus on Scene Search - `/` **Multi-Select Add/Remove** - `Ctrl/Cmd` + Click **Box Select Actors/Triggers** - `Shift` + Drag **Select All in Scene** - `Ctrl/Cmd` + `A` **Move Selected** - `Shift` + Arrow keys **Delete Selected** - `Delete` or `Backspace`

Navigator

The Navigator is the sidebar on the left which allows quickly navigating through your project.

Move selection up - `Arrow Up`
Move selection down - `Arrow Down`
Open folder - `Arrow Right`
Close folder - `Arrow Left`
Rename - `Enter`
Select multiple items (Scenes only) - `Shift` while clicking

Drawing Mode

Drawing mode is automatically enabled in the *Collision tool* and the *Palettes & Priorities 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`

Debugger

When the Debugger is enabled, you can use the following keyboard shortcuts in both the *Project Window* and *Play Window*:

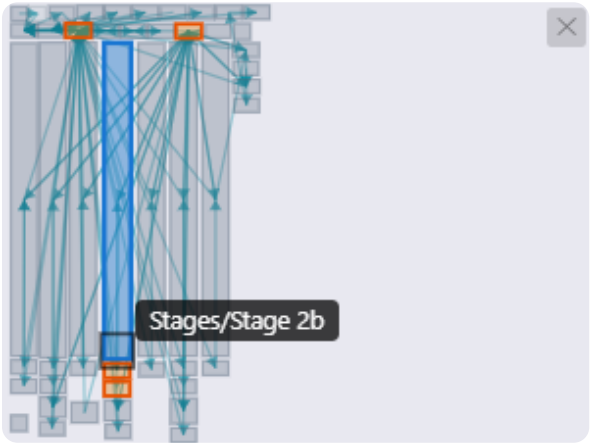
Pause/Resume - `F8`
Step Forward One Instruction - `F9`
Step Forward One Frame - `F10`

Minimap

The Minimap is a small overview panel in the bottom-right corner of the world view that shows all scenes in your project as colored rectangles. It lets you quickly navigate to any part of the world, which is especially useful for large projects with many scenes.

Enabling the Minimap

Go to **View > Show > Minimap** to toggle the minimap on or off.

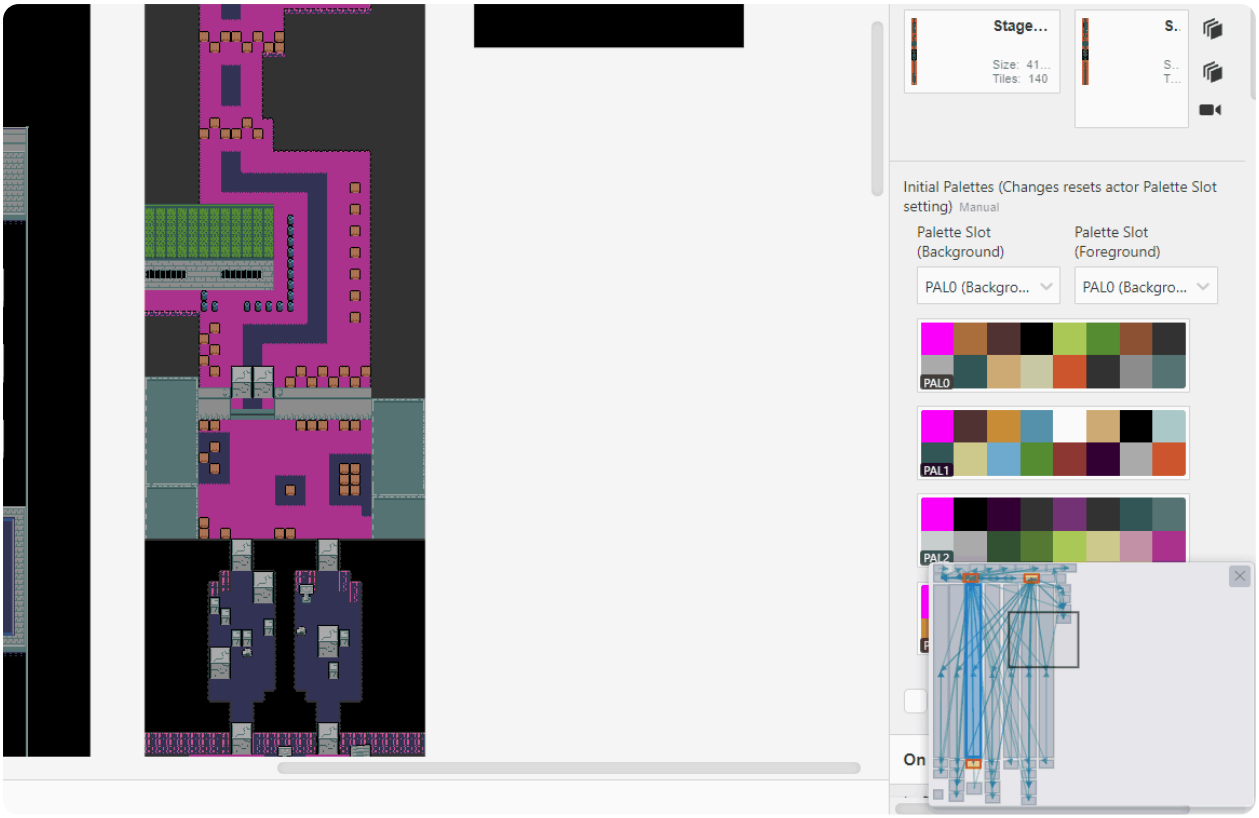


Reading the Minimap

- **Blue rectangle** — The currently selected scene.
- **Orange rectangles** — Scenes connected to the selected scene (via Change Scene events).
- **Gray rectangles** — All other visible scenes.
- **Dimmed rectangles** — Hidden scenes.
- **White outline** — Your current viewport position in the world.
- **Cyan lines** — Connection arrows between scenes that have scene-switch events.

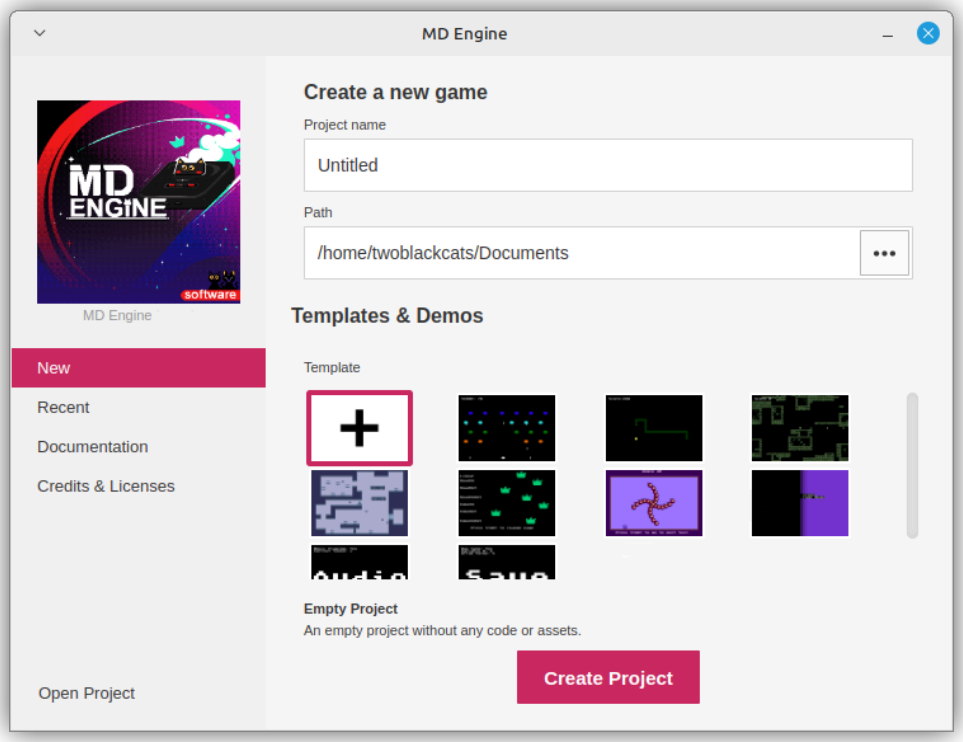
Navigating with the Minimap

- **Click** anywhere on the minimap to jump the viewport to that location.
- **Click and drag** to pan the world view in real time.


Stage...
Size: 41...
Tiles: 140
S..
S..
T...
Initial Palettes (Changes resets actor Palette Slot setting) Manual
Palette Slot (Background)
Palette Slot (Foreground)
PAL0 (Backgro...
PAL0 (Backgro...
PAL0
PAL1
PAL2
On

Your First Project

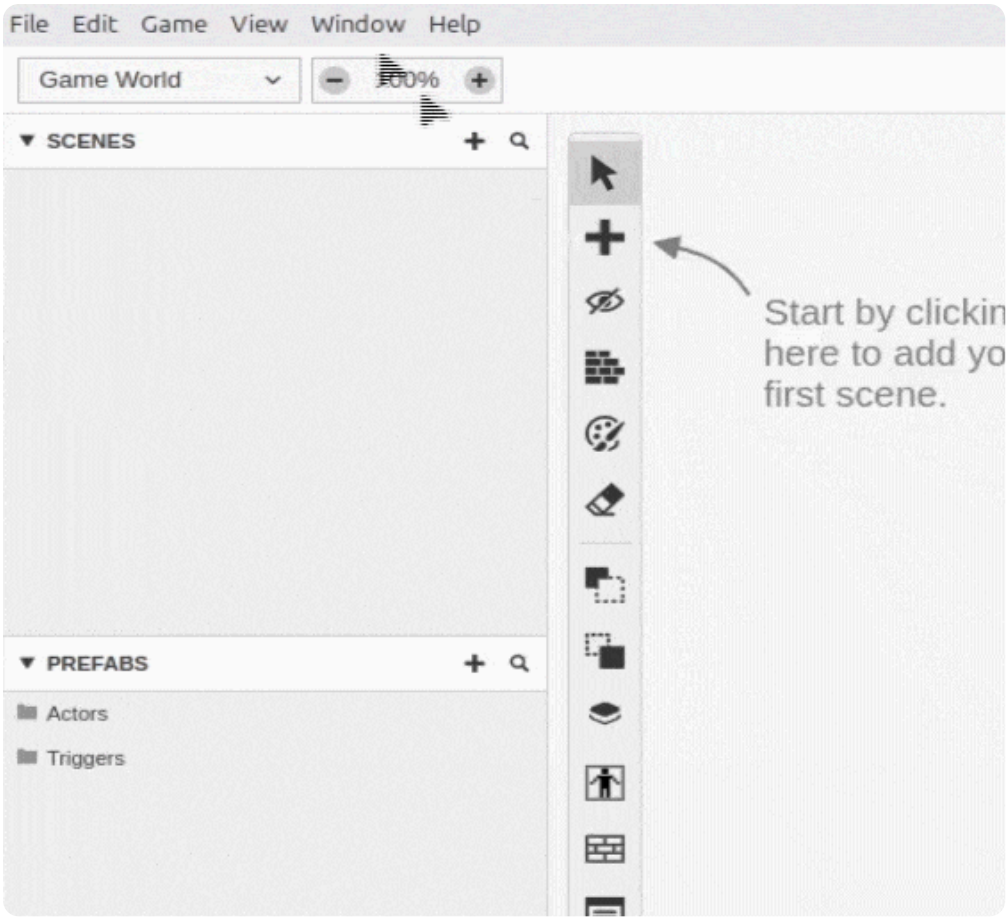
For our first project, we will start by selecting Empty Project as we do not want to edit any templates.



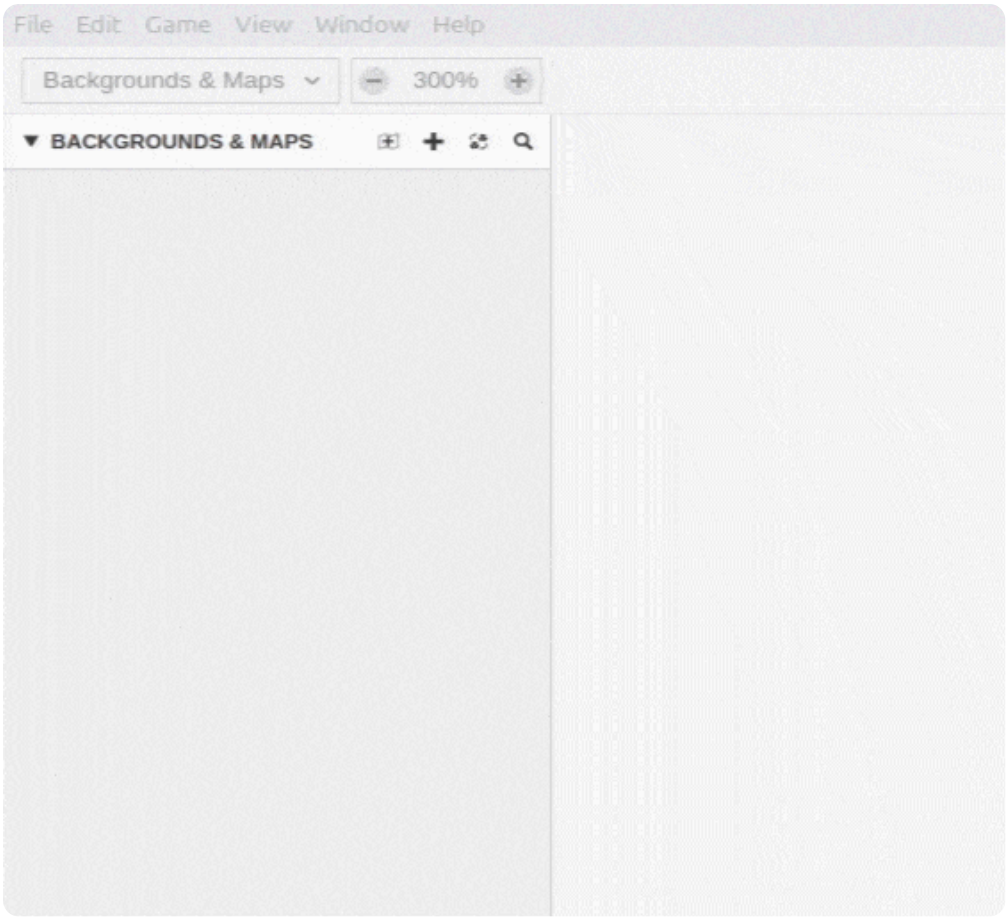
This will generate an empty project without any assets.

For the simplicity of this guide, you can find a background here and a sprite for our first project.

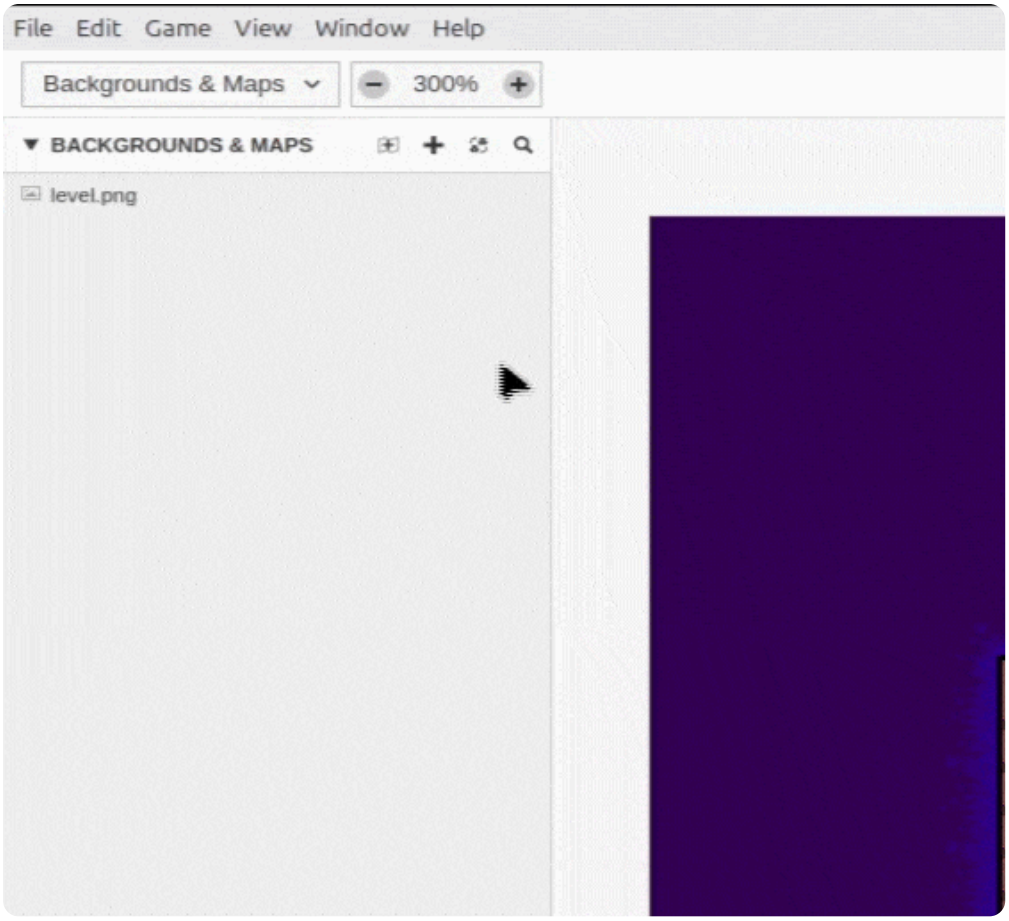
After downloading the zip, you can import the background by going to the *Project View Button* and selecting *Background & Maps*.



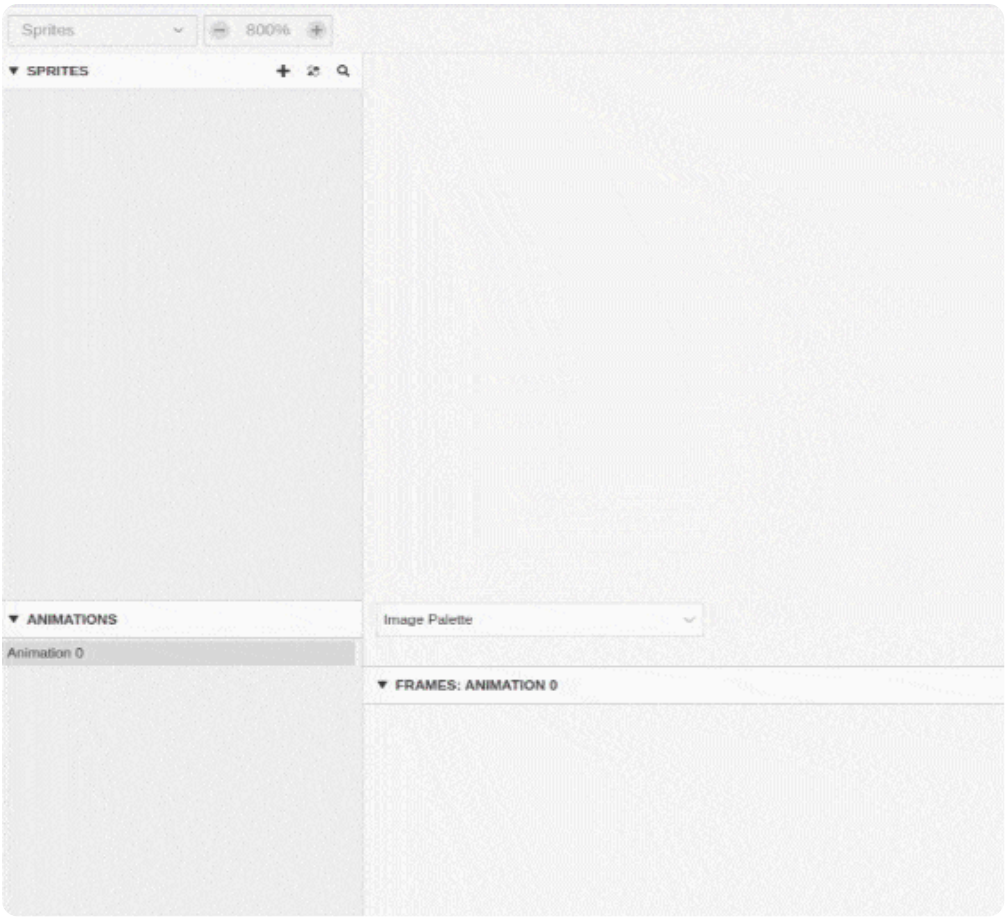
Once you are on the *Background & Maps* view, you can just drag the *level.png* file into the area to import it.



Now we can import our *Sprite*, so we go to *Sprite* view.



Once in *Sprites* view, you can just drag the *character.png* file into the area to import it.



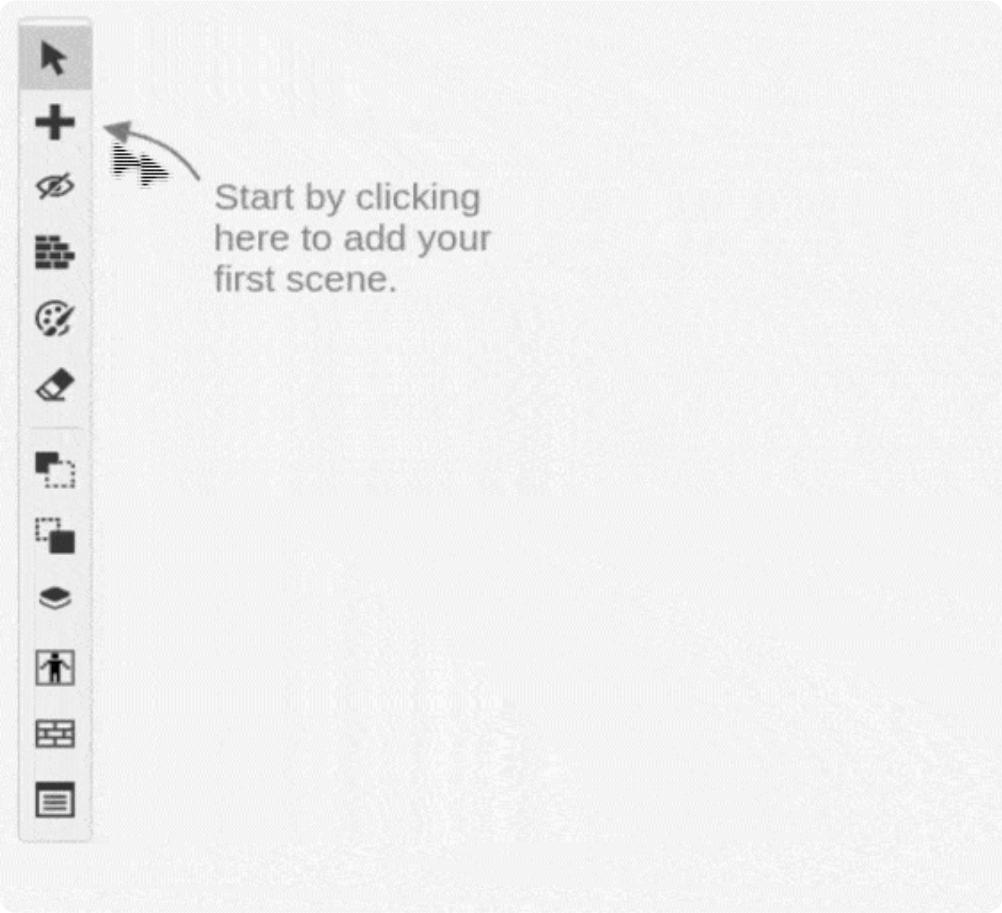
Now that we have all assets imported we can continue with making our first scene. Next, let's create our first scene.

Your First Scene

After creating our first project, we need to add the first scene.

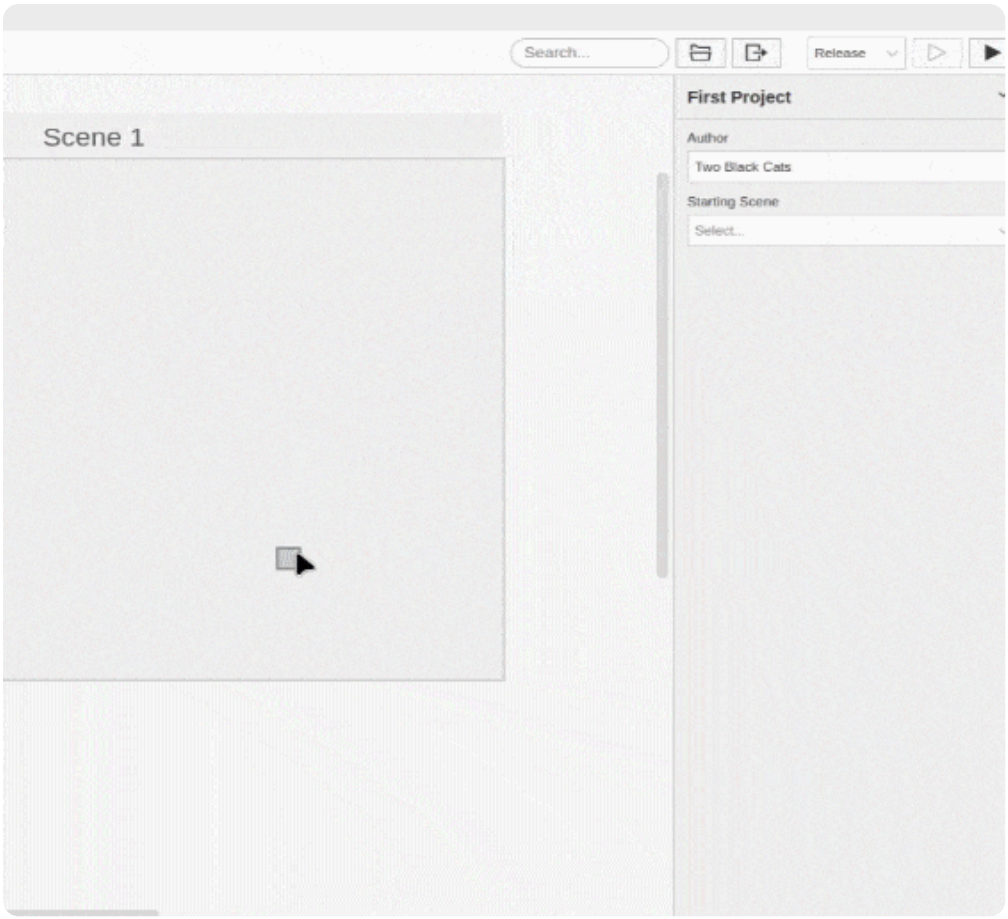
A more detailed explanation of Scenes can be read here.

For now, you can just click the + button, select *Scene* and place the new scene.



After you have your scene, it's time to set the background.

In order to do that, first you have to select the scene, then go to the *Editor Sidebar*. On the *Editor Sidebar* click inside *Plane (B) Background* and select *level.png*.



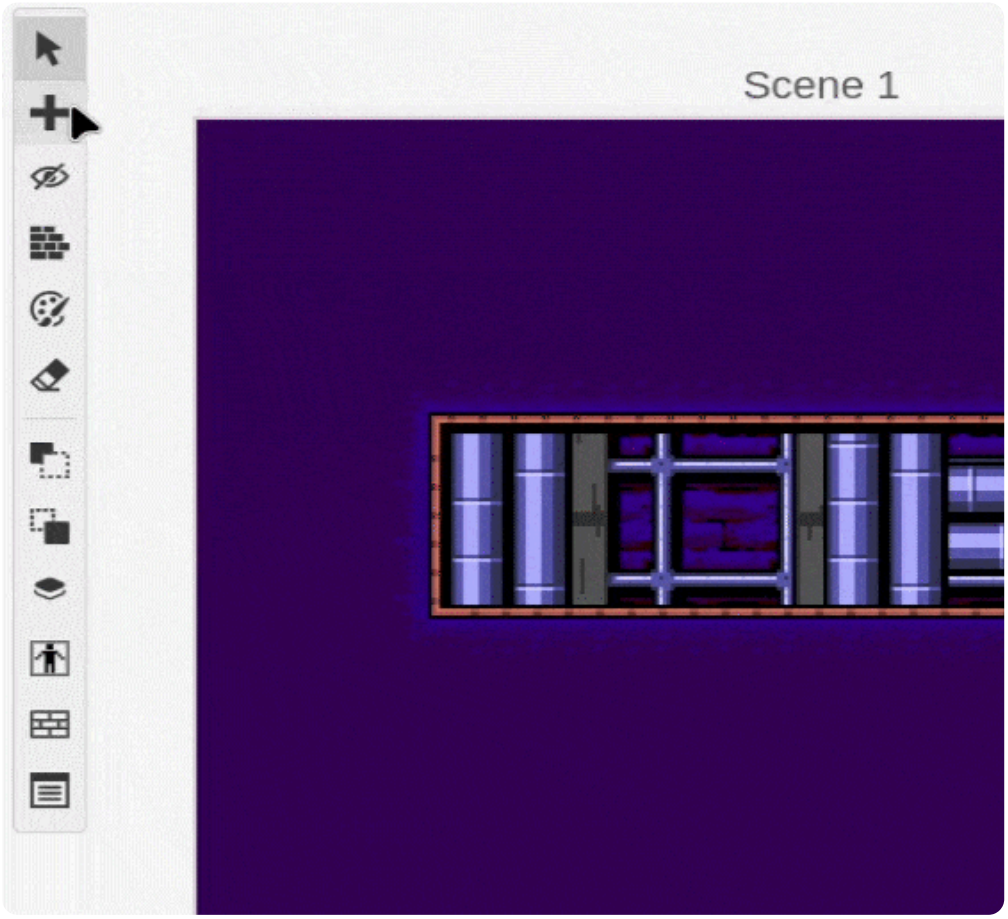
Now that we have our scene with the background, it's time to add our first Actor. Next, let's add our first actor.

Your First Actor

After creating our first scene, we need to add the first Actor.

A more detailed explanation of Actors can be read here.

For now, you can just click the  button, select *Actor* and place the new actor.



After doing that, it will have our initial sprite by default.

Now that we have our scene with the background and our actor, it's time to run everything. Next, let's test our game.

Templates & Demos

MD Engine includes a good number of templates and demos you can review.

Here you can find a small explanation of each template or demo.

The embedded web demos share browser storage. If it fills up, the demos keep running, but new saves will not persist. Free space for this site and reload the demo to resume saving. Clearing site data also removes existing demo saves.

Full Games

- **Game - Inkventure** — Full commercial game provided as an example
- **Game - Prisonela MD** — Full commercial game provided as an example
- **Game - Square Brothers** — Full commercial game provided as an example

Templates

- **Action Platformer** — Side-scrolling platformer with combat and enemies
- **Top-Down** — Explore a house and rescue your pet rats
- **Visual Novel** — Branching dialogue and character sprite expressions
- **Point & Click** — A point-and-click adventure sample: move the cursor with the D-Pad, pick a verb (Walk, See, Talk, Get, Use) and press A to interact. You lost your keys, find the spare one and get to bed!
- **Snake** — Classic snake game demonstrating grid movement and arrays
- **Breakout** — Classic breakout game example
- **Invaders** — Simple space invaders game
- **Vertical Shooter** — Project showcasing how to spawn and recycle enemies from a pool in a vertical shooter

Demos

- **Array** — How to create and use arrays
- **Audio** — Playing music and sound effects
- **Camera Follow** — Camera follow modes
- **Collisions** — Actor collision group options
- **Auto Scrolling** — Two ways to scroll a scene automatically
- **Endless Runner** — Project showcasing an endless runner with a seamlessly looping map
- **Co-op Play** — Co-op example, player 2 can join with START, the camera follows the midpoint of both players.
- **Wall Kick** — Project showcasing how to make a wall kick.
- **Tile Animation** — Project showcasing how to animate tiles.
- **PAL vs NTSC** — Region display differences and runtime detection
- **Dialogue Planes** — Dialogue layout across planes and anchors
- **Dialogue Codes** — Inline text controls across dialogue events
- **Pathfinding** — A* route finding around walls
- **MDSDRV** — Music and sound with the MDSDRV driver

- **HBlank - Palette** — Horizontal Blank script for palette changes
- **HBlank - Window Plane** — Horizontal Blank script for Window Plane changes
- **Palette Cycling** — Project showcasing palette cycling: a range of colors rotates in place every few frames, with start/stop, speed and direction controls.
- **Line Scroll Wave** — Project showcasing line-scroll waves: every scanline of the background scrolls by a moving sine, with underwater, heat haze and waterline presets.
- **Raster Gradient** — Project showcasing raster gradients: one palette color is rewritten as the screen is drawn, turning a flat sky into a dusk, dawn or night gradient.
- **Image** — Manual image loading and drawing
- **Parallax** — Parallax scrolling backgrounds
- **Particles** — Launching particles
- **Projectiles** — Launching projectiles
- **Save** — Saving and loading data
- **Shared Sprites** — Project showcasing the Shared sprite engine: 40 animated saws share 6 cached frames in VRAM instead of each keeping its own tiles.
- **Slopes** — Slope-based terrain for platformers
- **Temporary Triggers** — Using temporary triggers for attacks
- **Text** — Localization, dialogue, and drawing text to screen
- **Transitions** — Simple scene transitions
- **Tweens** — Sprite movement with different tween ease functions

Action Platformer



Play this demo online

Controls: You can move with the arrow keys, jump with Z key and shoot with C key.

An example showcasing a small action-platformer game, featuring shooting, ladders, enemies, a few levels, dialogues, and a camera.

Art created by ansimuz
License: CC0

Array



Play this demo online

Controls:
Z key adds a value to the front, X key removes a value from the end.

A small example showing how to create an array, and how to add / remove values from it.

It logs all the information into the debugger console as well as printing it.

Audio



Play this demo online

Controls:

Z key to play music, X key to pause music, C key to play a sound.

A small example showing how to play and pause a VGM music file and WAV sound file.

Auto Scrolling



Play this demo online

This sample shows two ways to scroll a scene automatically.

See Camera Events for camera movement controls.

Breakout



Play this demo online

Controls:
Move the paddle with arrows, D-Pad or Left Stick if you use a gamepad.

A basic breakout example that shows how to make a breakout game and features the actor bouncing off walls.

Camera Follow



Play this demo online

Controls:
Move the ball with arrows, D-Pad or Left Stick if you use a gamepad.

MD	Gamepad	Keyboard
D-Pad	Left Stick D-Pad	Arrow Keys WASD
A	A	Z
B	B	X
C	X	C
Start	Menu	Space

The example showcases how to use the camera, as well as lerp and follow modes.

Collisions

This sample demonstrates collision options between actors.



Play this demo online

See Collisions for how collision groups and masks work.

Co-op Play

Co-op example, player 2 can join with START, the camera follows the midpoint of both players.

Dialogue Codes



Play this demo online

This sample exercises inline text speed changes, waits, button prompts, cursor moves, fonts, variables, and characters. It uses Show Text, Show Choices, Draw Text, and a translation.

See Dialogue Variables for text formatting and commands.

Dialogue Planes

This sample shows dialogue on different planes and at different anchor positions. It also demonstrates offsets, clamping, scrolling, and closing the dialogue window.



Play this demo online

See Dialogue, Text & Menus for the available events.

Endless Runner

Project showcasing an endless runner with a seamlessly looping map

Game - Inkventure



Play this demo online

** Preview of the demo version.*

Controls:

ACTION	MD	Gamepad	Keyboard
Move	D-Pad	Left Stick D-Pad	Arrow Keys WASD
Jump / Confirm	A	A	Z
Camera	C + D-Pad	Y + D-Pad	C + Arrows
Pause	Start	Menu	Space

This is a full commercial game provided as an example. The game has been released on cartridge, modern consoles, and Steam.

The code and art can be used for learning purposes, but not commercially.

Game - Prisonela MD



Play this demo online

Controls:

ACTION	Mega Drive	Gamepad	Keyboard
Move	D-Pad	Left Stick D-Pad	Arrow Keys WASD
Jump / Confirm	A	A	Z
Skins	B	B	X
Music	C	X	C
Pause	Start	Menu	Space

This is a full commercial game provided as an example. The game has been released on cartridge, modern consoles, and Steam.

The code and art can be used for learning purposes, but not commercially.

Game - Square Brothers



Play this demo online

** Preview of the demo version.*

Controls:

ACTION	Mega Drive	Gamepad	Keyboard
Move	D-Pad	Left Stick D-Pad	Arrow Keys WASD
Jump / Confirm	A	A	Z
Swap Characters	B	B	X
Camera	C + D-Pad	Y + D-Pad	C + Arrows
Pause	Start	Menu	Space

This is a full commercial game provided as an example. The game has been released on cartridge, modern consoles, and Steam.

The code and art can be used for learning purposes, but not commercially.

HBlank - Palette



Play this demo online

Controls:
ARROWS to move, Z key to jump and X key to attack.

Example showing how to create palette effects using the HBlank callback script.

HBlank - Window Plane



Play this demo online

Controls:
ARROWS to move, Z key to jump and X key to attack.

Example showing how to create a double HUD using the HBlank callback script and the Window Plane.

Image



Play this demo online

Controls:
SPACE or Start / Menu / OPTIONS button to swap scenes.

Example showing how to manually load images and draw them to the screen.

Invaders



Play this demo online

Controls:
Move the ship with arrows, D-Pad or Left Stick if you use a gamepad.
Use Z key or A Button to shoot.

A simple invaders game, it shows how to make a moving ship and projectile shooting.
It also illustrates collision grinding to avoid performance issues.

Line Scroll Wave

Project showcasing line-scroll waves: every scanline of the background scrolls by a moving sine, with underwater, heat haze and waterline presets.

MDSDRV

This sample demonstrates music and sound playback with the MDSDRV driver.



Play this demo online

See Sound Driver for supported audio formats.

PAL vs NTSC



Play this demo online

This sample compares the PAL display area (240 pixels at 50 Hz) with NTSC (224 pixels at 60 Hz). It also shows how to detect the region at runtime. Change **Settings > Preview Region** and run it again to compare the two modes.

Palette Cycling

Project showcasing palette cycling: a range of colors rotates in place every few frames, with start/stop, speed and direction controls.

Parallax



Play this demo online

Controls:
Move the character with arrows, you can also jump with Z key.

Example showing how to make a parallax background with image streaming and 4 different layers at different speeds.

Particles



Play this demo online

Controls:
Z key or A button to emit particles.

Example showing how to launch particles in different angles.

Pathfinding



Play this demo online

This sample demonstrates A* pathfinding around walls. Use the directional pad to move the cursor and press **A** to send the NPC to that point.

See Pathfinding for path slots, algorithms, and settings.

Point & Click Adventure

A point-and-click adventure sample: move the cursor with the D-Pad, pick a verb (Walk, See, Talk, Get, Use) and press A to interact. You lost your keys, find the spare one and get to bed!

Projectiles



Play this demo online

Controls:
SPACE or Start / Menu / OPTIONS button to swap pages.

Example showing how to launch projectiles in different angles.

Raster Gradient

Project showcasing raster gradients: one palette color is rewritten as the screen is drawn, turning a flat sky into a dusk, dawn or night gradient.

Save



Play this demo online

Controls:

Z key saves the game, X key increments a variable, C key increments an array variable. SPACE key clears the save.

A small example showing how to save and load data to either Flash or SRAM memory.

Shared Sprites

Project showcasing the Shared sprite engine: 40 animated saws share 6 cached frames in VRAM instead of each keeping its own tiles.

Vertical Shooter

Project showcasing how to spawn and recycle enemies from a pool in a vertical shooter

Slopes



Play this demo online

Controls:
Move the character with arrows, you can also jump with Z key.

Example showing how to make simple slopes.

Snake



Play this demo online

Controls:
Move the snake with arrows, D-Pad or Left Stick if you use a gamepad.

A basic snake example, it shows how to make a snake game using only tiles to allow for a snake that is longer than the sprite limit.

Temporary Triggers



Play this demo online

Controls:
ARROWS to move, Z key to jump and X key to attack.

Example showing how to use temporary triggers for attacks on a hack'n'slash type of game.

Text



Play this demo online

Controls:
Z key or A button to advance dialogues.

This example shows how to display dialogue and choices, as well as variables inside the text.
It features Japanese localization after displaying text in English.

Tile Animation

Project showcasing how to animate tiles.

Top-Down

Explore the house, interact with objects, and rescue your pet rats. This playable sample is also available as the **Top-Down Game** template on MD Engine's **New Project** screen.



Play this demo online

Controls:

ACTION	Mega Drive	Gamepad	Keyboard
Move	D-Pad	D-Pad	Arrow Keys
Interact / advance text	A	A	Z

Transitions



Play this demo online

Controls:

Use Z key or A Button to change scene, Right arrow or Right direction to change fade mode.

A small example showcasing how to do some simple scene transitions with tiles and scripts.

Tweens



Play this demo online

Controls:
SPACE or Start / Menu / OPTIONS button to swap pages.

This example features a sprite moving from left to right using different tween ease functions.

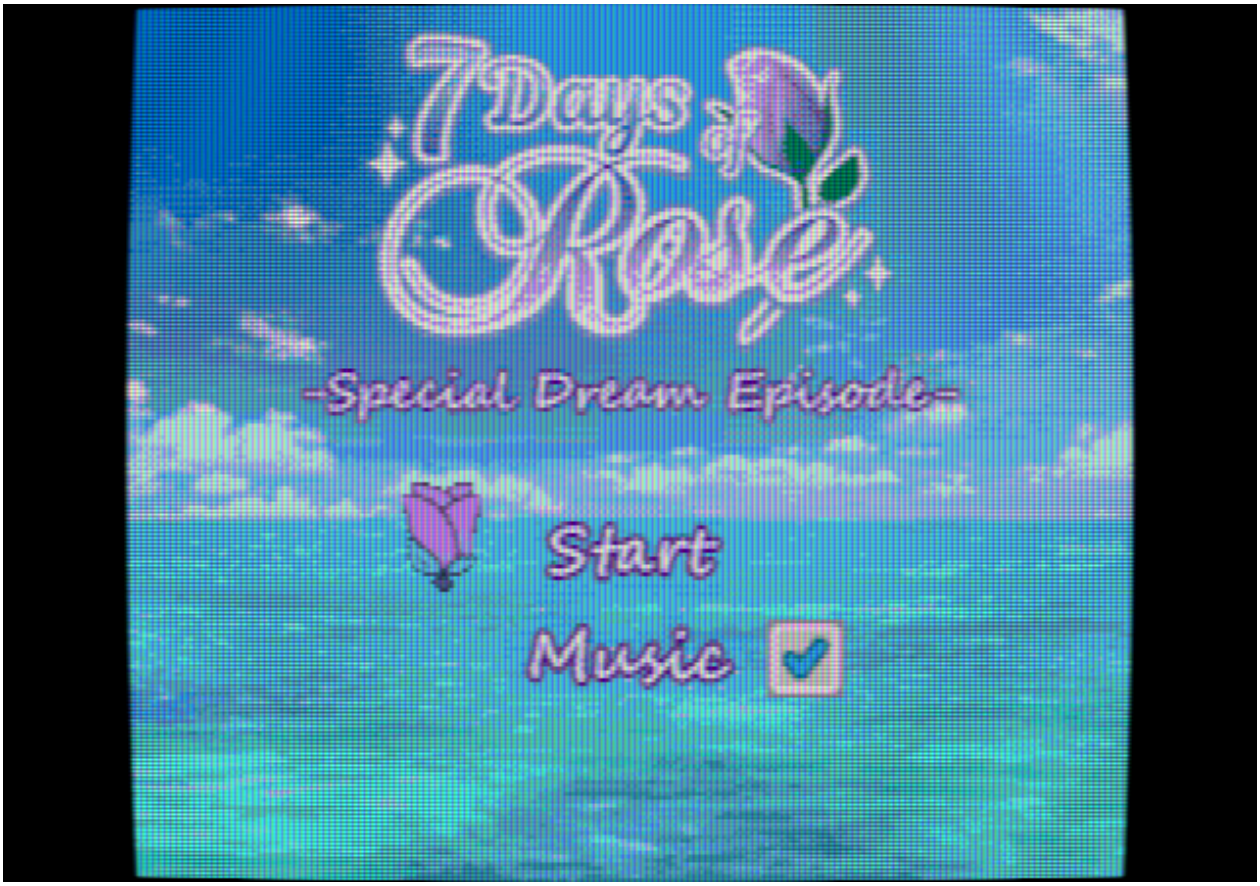
MD Engine includes a tweening system that can be used to interpolate numeric values over time.

This is the list of Easing functions you can choose from:

- linear
- smoothstep
- smootherstep
- quadIn
- quadOut
- quadInOut
- cubeIn
- cubeOut
- cubeInOut
- quartIn
- quartOut
- quartInOut
- quintIn
- quintOut
- quintInOut
- sineIn
- sineOut
- sineInOut
- bounceIn
- bounceOut
- bounceInOut
- circIn
- circOut
- circInOut

- expoIn
- expoOut
- expoInOut
- backIn
- backOut
- backInOut
- elasticIn
- elasticOut
- elasticInOut

Visual Novel



Play this demo online

This template shows branching dialogue and character sprite expressions in a visual novel scene.

See Dialogue, Text & Menus for the events used to build dialogue and choices.

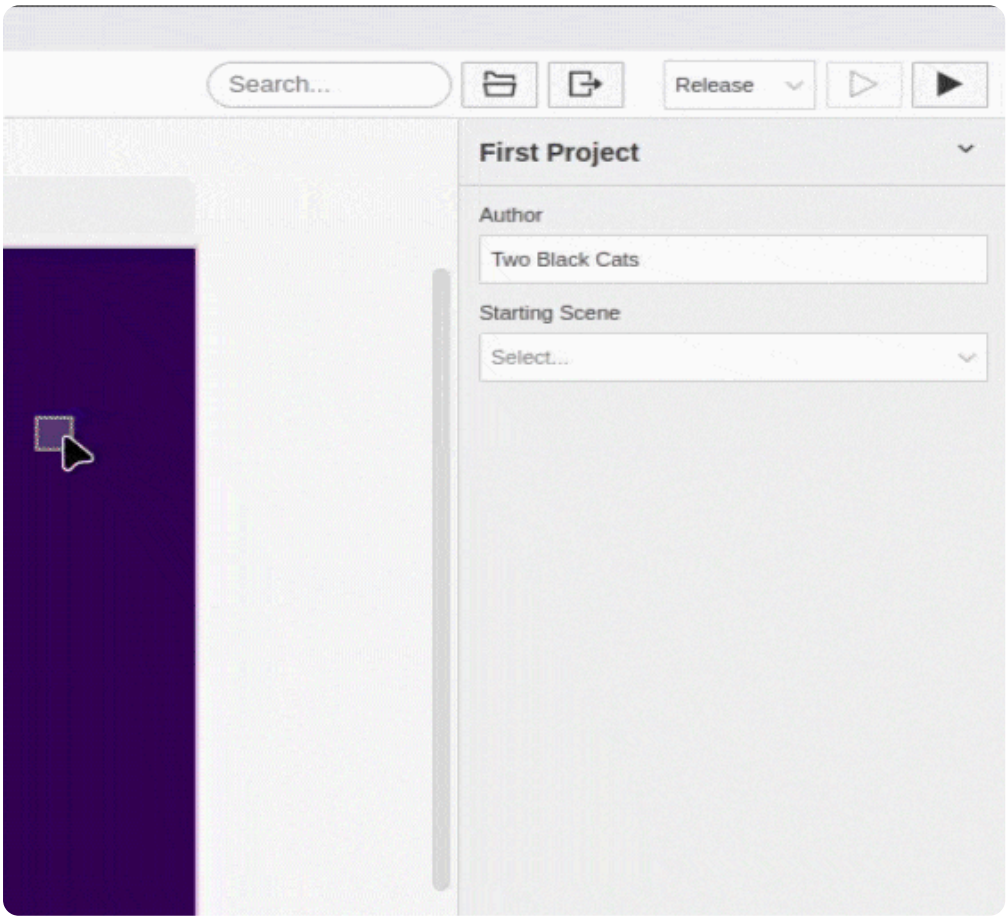
Wall Kick

Project showcasing how to make a wall kick.

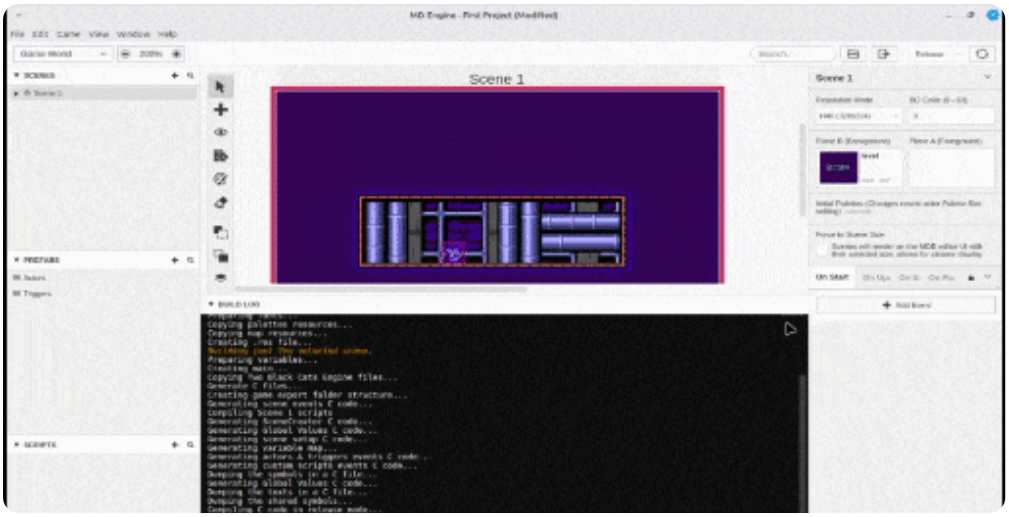
Testing Your First Game

You have two options when running MD Engine projects: you can either run the selected scene with the *Test Scene* button or run the entire game with the *Play* button.

> **TIP**
>
> A single scene game will build faster than a game with multiple scenes.



After pressing Play, the game will build and it will swap to the Debugger Window.



Now you can see your first game in action!

Now you should check other examples.

Importing Scenes & Projects

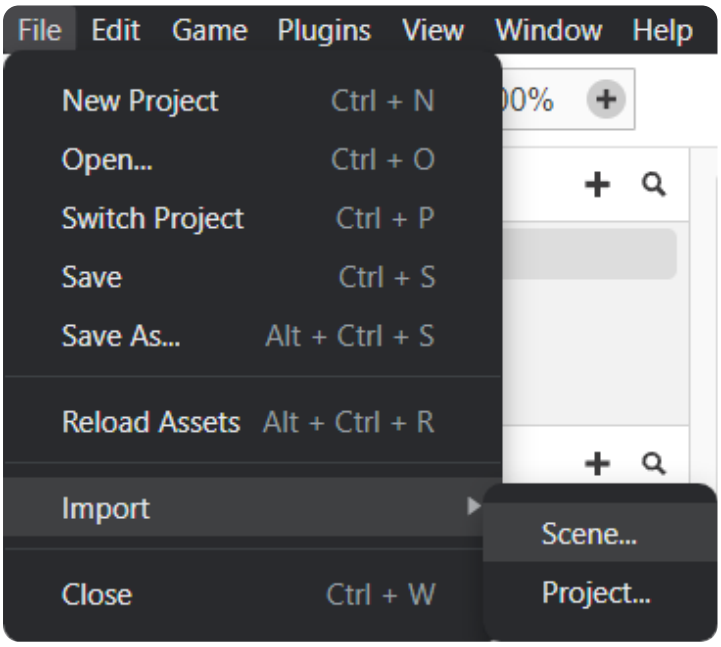
You can import content from other MD Engine projects into your current one — either a single scene with all its dependencies, or an entire project.

> **WARNING**
>
> Importing cannot be undone. Always save your project before importing.

Import Scene

Go to **File > Import > Scene...** and select the source project's `.mdep` file (or its `.json` project file). The importer reads scenes from the project containing that file.

A dialog lists all scenes in the source project. Select the scene you want and click **Import**.



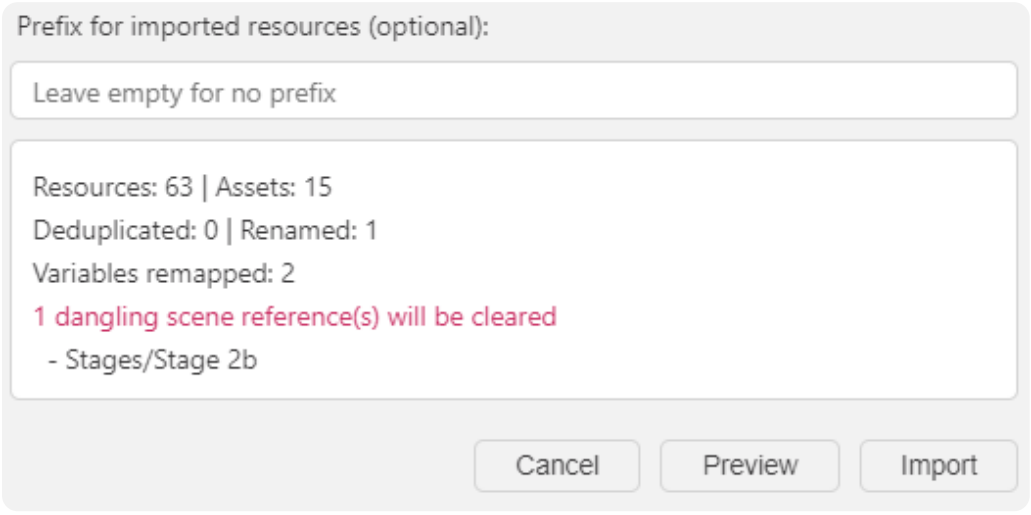
The importer will:

- Copy the scene and all its actors, triggers, and scripts.
- Copy all referenced assets (sprites, backgrounds, palettes, music, etc.).
- **Deduplicate** assets that already exist in your project (by filename and content hash).
- **Rename** assets that have the same filename but different content (appending `_imported`).
- **Remap** all UUIDs to avoid conflicts with existing entities.
- **Clear** scene references that point to scenes not included in the import (e.g., "Go To Scene" events pointing to non-imported scenes will be cleared).

> **TIP**
>
> Double-click a scene in the list to import it immediately.

Preview

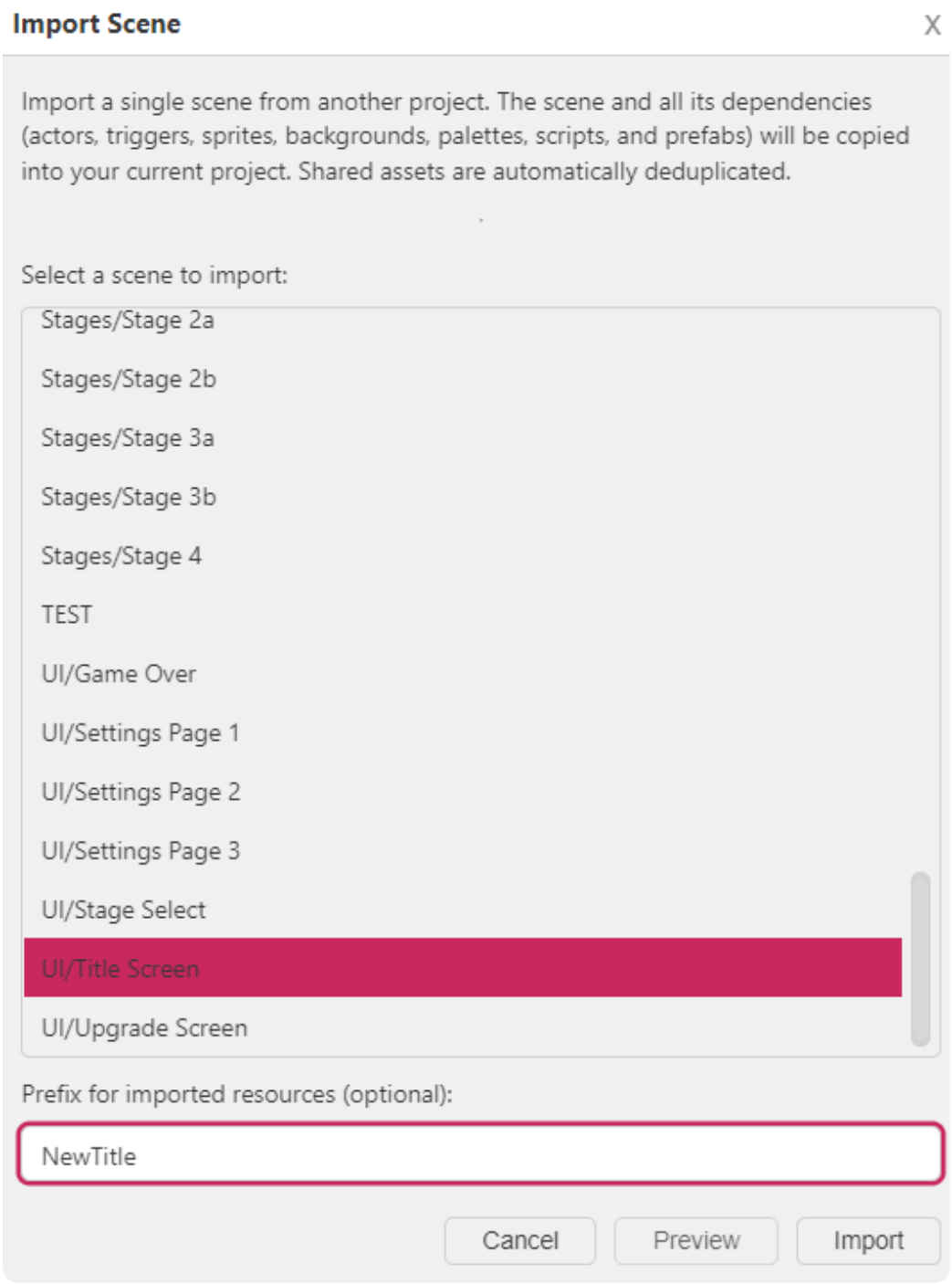
Click **Preview** before importing to see a summary of what will be imported — how many resources, assets, deduplication counts, and any warnings — without making changes.



Import Project

Go to **File > Import > Project...** and select the source project file.

Enter a **prefix** that will be added to all imported resource names and symbols to prevent naming conflicts.



All scenes, assets, variables, and collision groups from the source project are merged into your current project.

Asset Types

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.

MD Engine allows you to import assets directly from the engine window, this applies to all asset types. To delete an asset, right-click it in the project navigator and choose **Delete**.

Sprites and Background images can be imported as they are, but it is also possible to convert them to an allowed format. For more information on this conversion, check the Image Converter section.
Regarding Fonts, they can be loaded as-is, if you followed the font creation information outlined in the New Fonts section, or they can be converted from TTF using the Font Converter.

You can edit sprites and backgrounds in MD Engine with Mini Painter, or use an external image editor. If you edit a sprite or background PNG file externally and save it in your project's asset folder, the change appears in the Project Window when you switch back to MD Engine.

For background maps, you can do basic edits to them from inside the engine, for a better editing experience you can use the Tiled Map Editor, just be aware as stated here tile rotation is not supported, only tile horizontal and vertical flips.

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 maps.
Each image asset type has a different set of requirements detailed over the next few sections of this documentation. You can select the default application to open when clicking asset edit buttons in the MD Engine Preferences window.

Sound Driver

MD Engine supports three music drivers. You can change the active driver in Project Settings > Audio.

XGM2

Plays VGM music files. Supports music playback along with 3 PCM channels with adjustable volume (FM and PSG only). Compatible with VGM, XGM, WAV, and Mini Tracker (`.mtrk`) files.

You can create VGM files using Deflemask, Furnace, or any other tracker that supports VGM export. You can also compose directly inside MD Engine using the built-in Mini Tracker or Mini Notator.

XGM

Plays VGM or XGM music files. Compatible with VGM, XGM, WAV, and Mini Tracker (`.mtrk`) files. Does not support music fade in/out or volume control.

MDSDRV

Uses the MDSDRV sound driver by superctr. Plays MML music files compiled via mdslink. Compatible with MML files only. You can compose MML files using the built-in Mini Notator.

Sprites

Sprites are the graphics used by playable or interactive characters in your scenes. You can import sprites with the  button if you want to load the PNG as-is. You can also convert any PNG into MD Engine format with the  button. For more information regarding this, you can check the Image Converter section.

You can also add sprites to your game by including `.png` files in your project's `assets/sprites` folder.

Image Requirements

The image size must be a multiple of 8px in both width and height. If you add the sprite as-is, it must be an indexed `.png` file with a maximum of 16 colors. If you are unsure about the color requirements you can always use the Image Converter.

Animated Sprites

If you want to have sprites that play animations, you need to make your PNG image containing each animation on a separate row.



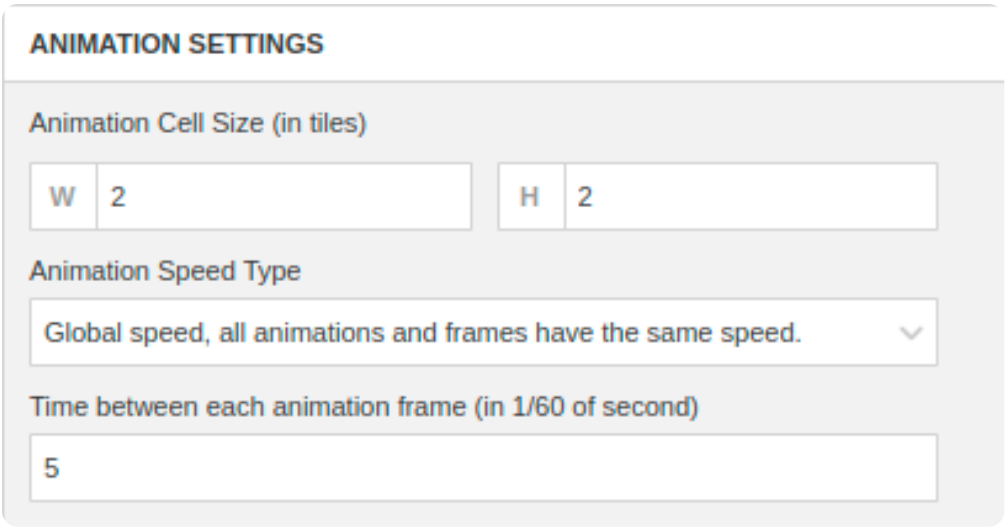
Each row will represent one animation, during compile time empty space will be optimized out, so it's not a problem if there are some rows that have more empty space than needed.

Settings

Sprites settings have two main sections *Animation Settings* and *Sprite Settings*.

Animation Settings

There are two main options to edit here, *Animation Cell Size* and *Animation Speed Type*.



Animation Cell Size

This represents the size of an individual frame of the image.

Animation Speed Type

This represents the type of animation setup.
You have three different options:

- Global speed
- Per Animation speed
- Per Frame speed

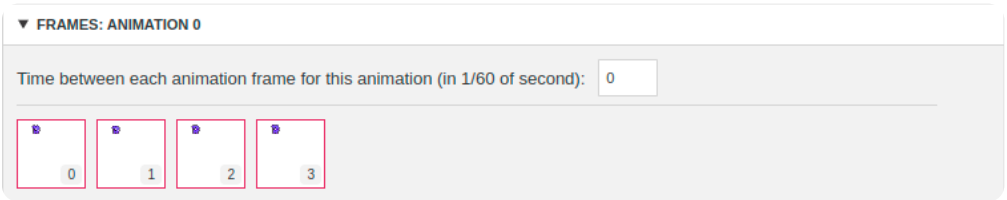
Global speed

With this option all the sprite animations and frames have the same speed.
It's the simplest option and works best for simple decorations and NPCs.

Per Animation speed

With this option, each animation can have different speed values.
This allows you to have faster animations and slower animations when both have the same number of frames.

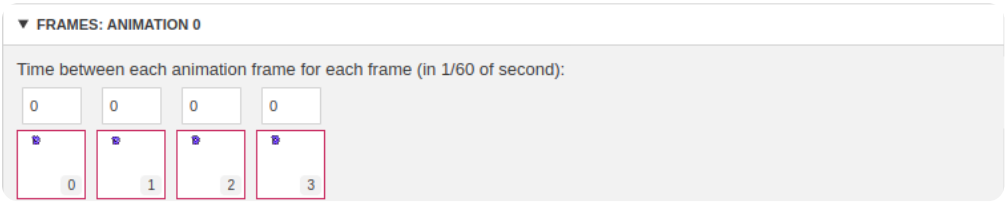
If you select this option you can edit the value in the *Frames* window.



Per Frame speed

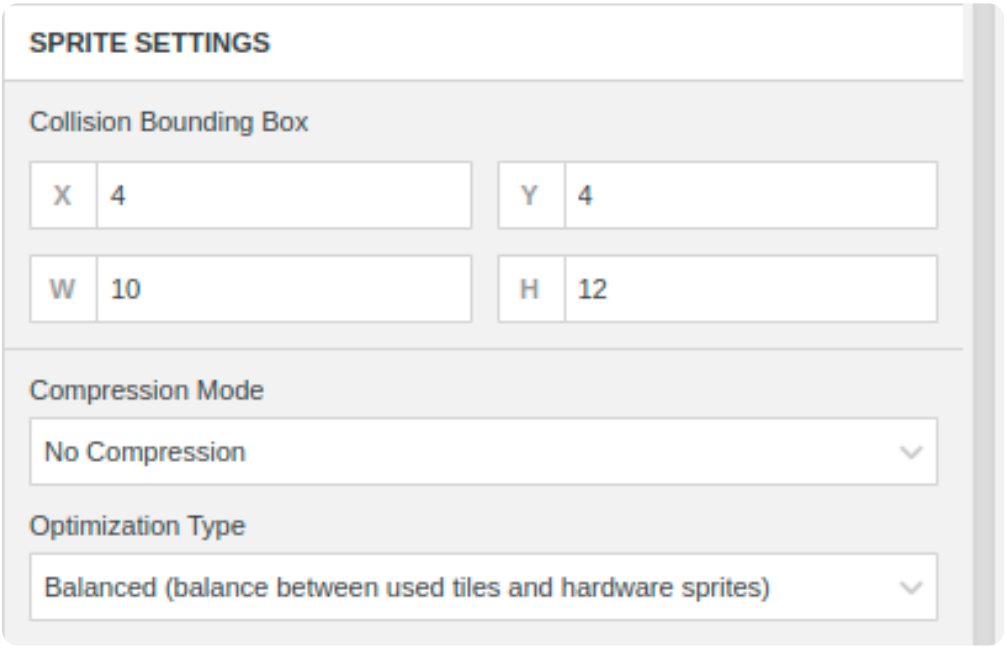
Each frame of each animation can have different speed values. This allows some frames to stay longer than others without manually duplicating them on the PNG image.

If you select this option you can edit the value in the *Frames* window under each frame.



Sprite Settings

The Sprite settings hold information related to how the sprite will be used when attached to an actor and how it will be stored in the ROM.



Compression Mode

This represents how the image is compressed when compiled into the ROM.

You have four different options:

- No Compression (No decompression cost, but produces larger files)
- Fastest (Prioritizes decompression speed, recommended for streamed assets)
- High Compression (Prioritizes smaller files, recommended for data decompressed only once)
- Automatic (Tests all supported methods and selects the smallest result, decompression speed may vary)

Use *Fastest* for animated sprites whose graphics are streamed while they run. Use *High Compression* for static or preloaded sprites that are decompressed only once. *Automatic* minimizes the compiled size, but may select a slower

decompression method.

Optimization Type

This represents how the image is optimized during the compile time.

You have four different options:

- Balanced: Balance between used tiles and hardware sprites.
- Sprite: Reduce the number of hardware sprites at the expense of more used tiles.
- Tile: Reduces the number of tiles at the expense of more hardware sprites.
- No Optimization: cover the whole sprite frame.

The best option if you are just starting is to use *Balanced*.
If your sprite is too big it will cause problems during compile time.

Hardware Sprites

Each sprite is composed of an internal type of object called **hardware sprite**.

They are a small, independently drawn graphic element that the video hardware can move, flip, or hide without redrawing the whole background.

There are a total of **80** hardware sprites available.
Each hardware sprite can be 8, 16, 24, or 32 pixels wide and independently 8, 16, 24, or 32 pixels tall. The largest is 32×32 pixels.

There is also a scanline limit, even though 80 sprites can exist, only 20 can appear on a single scanline (horizontal row).
If more than 20 overlap, the excess don't appear (they "drop out") — classic flicker behavior.
This limit can be removed for the *Desktop Export* with the Enhanced Mode.

Maximum Sprite Sizes

One MD Engine *Sprite* can have a maximum of 16 **hardware sprites**.

Sixteen 32×32 hardware sprites cover at most the same drawn area as a 128×128 image. A frame can be wider or taller if its occupied cells still fit within the 16-sprite limit.

> **WARNING**
>
> If a sprite uses more than 16 hardware sprites, the following error would be printed on build:
> **Sprite frame '<sprite_symbol>' uses 20 internal sprites, that is above the limit (16), try to reduce the sprite size or split it.**

Backgrounds & Maps

Each scene can have two layers, one is called Plane B (Background) and the other is called Plane A (Foreground).

They can be either image files (`.png`) or Tiled map files (`.tmx`).

You can import the background with the  button if you want to load the file as-is. You can also convert any PNG into MD Engine format with the  button, for information regarding this you can check the Image Converter section.

You can also add backgrounds or maps to your game by including `.png` or `.tmx` files in your project's `assets/backgrounds` folder.

Image files can be loaded as **Map** or as **Image**, there are some restrictions to each file type that are outlined below.

Maps vs Images

You can use any image file as a map. Tiled map files are only supported for maps though.

Action	Map	Image
Scroll Support	✔	✔*
Parallax Support	✘	✔
Wrapping	✘	✔
Streaming	✔	✘

* Images can scroll but it needs to be enabled by an event.

> **TIP**
> The **Map** resource type will be the preferred one in most cases.
> If you don't know what to use it's recommended to use **Map**.

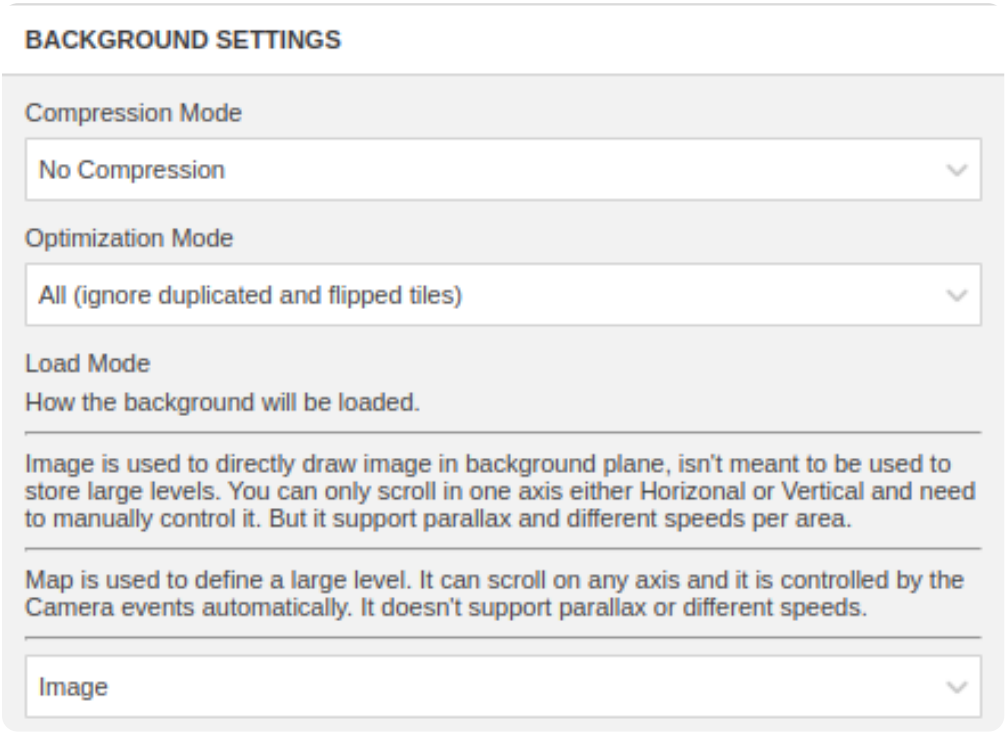
Images

Image Requirements

The image size must be a multiple of 8px in both width and height. If you add the image as-is, it must be an indexed `.png` file with a maximum of 64 colors.
Each 8x8 tile can only use colors from a single 16-color palette.
If you are unsure about the color requirements you can always use the Image Converter.

Image Settings

The image settings hold information related to how the image will be used when attached to a scene and how it will be stored in the ROM.



Compression Mode

This represents how the image is compressed when compiled into the ROM.

You have four different options:

- No Compression (No decompression cost, but produces larger files)
- Fastest (Prioritizes decompression speed, recommended for streamed assets)
- High Compression (Prioritizes smaller files, recommended for data decompressed only once)
- Automatic (Tests all supported methods and selects the smallest result, decompression speed may vary)

Use *No Compression* when avoiding decompression cost matters more than ROM size. Use *Fastest* for streamed or frequently loaded assets, and *High Compression* for data loaded once. *Automatic* minimizes the compiled size, but its decompression speed may vary.

Optimization Mode

This represents how the image is optimized during the compile time.

You have three different options:

- No Optimization
- All: ignore duplicated and flipped tiles.
- Duplicate: ignore duplicated tile only.

The best option if you are just starting is to use *All*.

Load Mode

This represents how the image will be loaded when the scene opens. You have two different options, Image and Map. You can check the main differences here.

Image Mode

This is used to directly draw images in the background plane and isn't meant to be used to store large levels. You can only scroll in one axis either Horizontal or Vertical and need to manually control it. But it supports parallax and different speeds per area.

Map Mode

This is used to define a large level. This option would be the same as using a Tiled map file.
It can scroll on any axis and it is controlled by the Camera events automatically. It doesn't support parallax or different speeds.

Maps

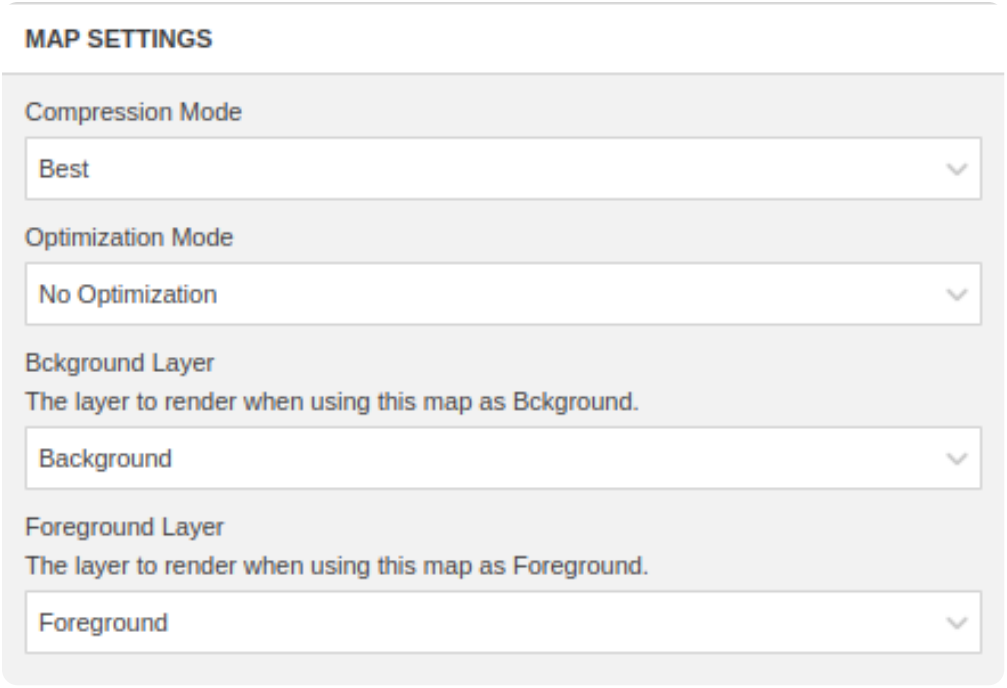
Map Requirements

For maps imported and edited in MD Engine, embed the tileset in the TMX file. The editor's TMX parser does not load external TSX files. The tileset must be placed on the `assets/tilesets` folder.
The path in the map must be relative to it and not absolute.
The tile size should be a multiple of 8, it's recommended to use `8px` or `16px`.

> **TIP**
>
> The best solution if you want to make new maps and edit them with Tiled Map is to create them inside MD Engine, then link the tilesets, save them and proceed to edit with Tiled Map.

Map Settings

The map settings hold information related to how the Tiled map file will be used when attached to a scene and how it will be stored in the ROM.



Compression Mode

This represents how the image is compressed when compiled into the ROM.

You have four different options:

- No Compression (No decompression cost, but produces larger files)
- Fastest (Prioritizes decompression speed, recommended for streamed assets)
- High Compression (Prioritizes smaller files, recommended for data decompressed only once)
- Automatic (Tests all supported methods and selects the smallest result, decompression speed may vary)

Use *No Compression* when avoiding decompression cost matters more than ROM size. Use *Fastest* for streamed or frequently loaded assets, and *High Compression* for data loaded once. *Automatic* minimizes the compiled size, but its decompression speed may vary.

Optimization Mode

This represents how the image is optimized during the compile time.

You have three different options:

- No Optimization
- All: ignore duplicated and flipped tiles.
- Duplicate: ignore duplicated tile only.

The best option if you are just starting is to use *All*.

Background / Foreground Layer

This represents what layer would be used when the map is attached on a *Scene* as either Plane B (Background) or Plane A (Foreground) layer.

Map Editor

MD Engine includes a simple map editor that can be used to create or edit Tiled map files, as well as to link tilesets to them.

More information about it here.

Using Tiled

> **WARNING**
>
> Be aware when using Tiled Map directly! There are some restrictions that apply to the maps:
>
> - You can't rotate tiles, only Horizontal or Vertical flips
> - All tiles on a layer must belong to the *SAME* tileset
> - Only two layers can be used per map, if you have more they will be ignored

Tilesets

Tilesets are used to render the maps, you can import tilesets with the  button if you want to load the PNG as-is. You can also convert any PNG into MD Engine format with the  button, for information regarding this you can check the Image Converter section.

You can add tilesets to your game by including .png files in your project's assets/tilesets folder.

Tilesets can also be loaded independently and used to manipulate the overall scene. You can load a tileset with Load Tileset event.

Image Requirements

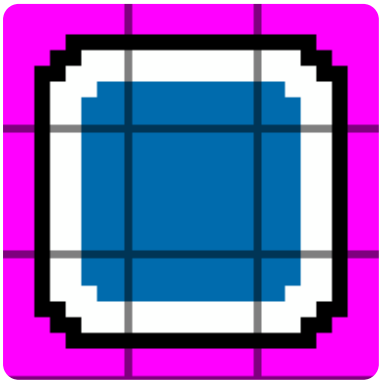
Tilesets follow the same image requirements as backgrounds. See Image Requirements for details.

UI Tilesets

There are special tilesets that can be used for the dialogue system.
They would be the dialogue frame and dialogue cursor.
They can be set with the Set Dialogue Cursor and Set Dialogue Frame or make them the default option in the Dialogue & Text settings.

Dialogue Frame

The *Dialogue Frame* should be an image of `24px` x `24px`, containing 3 tiles in both width and height.



Here you can see how it looks once loaded in-game.

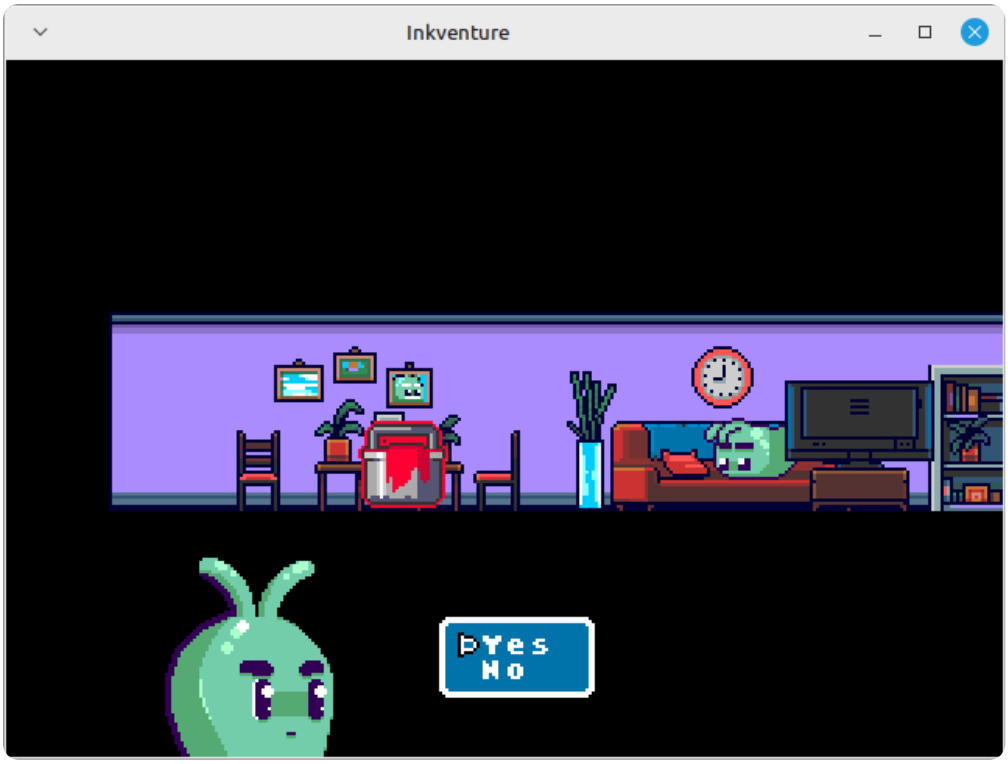


Dialogue Cursor

The *Dialogue Cursor* should be an image of `8px` x `8px`, containing one tile.



It will be used during dialogue choices. Here you can see how it looks once loaded in-game.


Inkventure
Yes
No

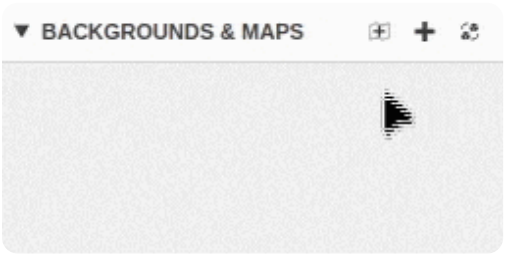
Map Editor

MD Engine includes a simple map editor that can be used to create or edit Tiled map files.

It has a small set of features, but it's best to use it to create the map as well as to link the tilesets.

Creating the Map

To create a map inside Background & Maps view, click the  button.



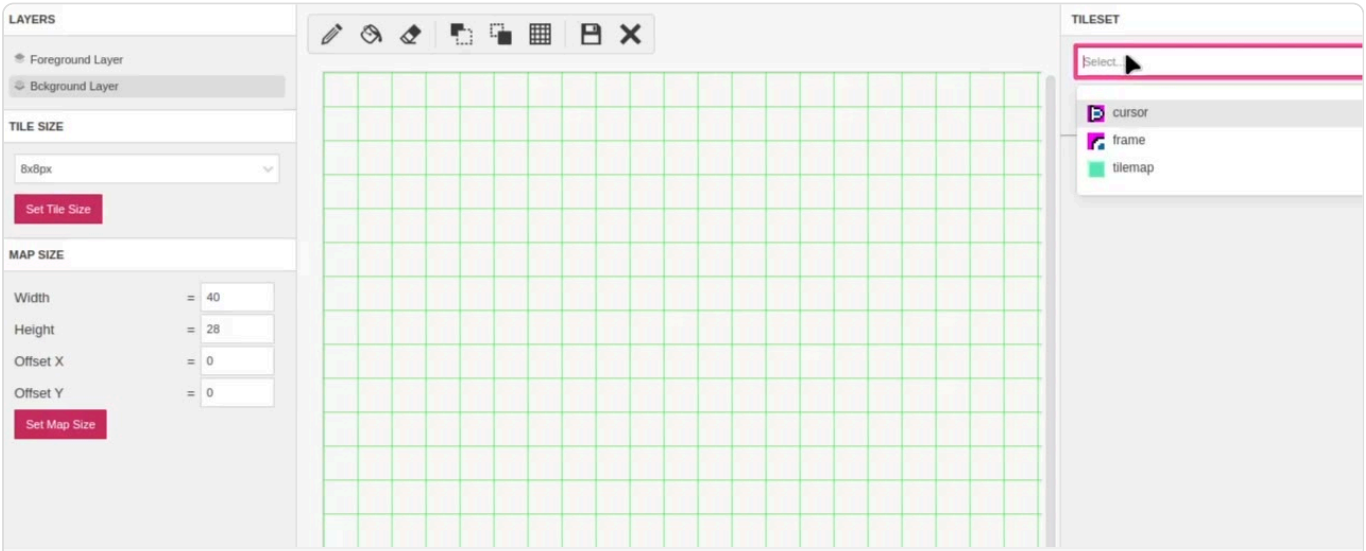
This will create a new empty map that can be edited by clicking the  button.



Linking Tilesets and Changing Tile Size

Once on the Map Editor, we go to the *Editor Sidebar*, then select the desired tilesets, after that click the *Use on Current Layer* button to confirm.
Accept the pop-up, this will link the tilesets to the map.

After that step you can safely edit the tile size on the *Navigator*.
Select the desired tile size, and then click the *Set Tile Size* button to confirm. Accept the pop-up; this will set the tile size.



Video: link-map.mp4 — view online in the web documentation.

> **NOTE**
>
> For detailed Tiled Map requirements and restrictions, see Using Tiled.

Fonts

The font assets are used to store the letters needed for the dialogue system.

The amount of letters you can use depends on the text system you use.

Simple Mode (English)

This mode loads the entire font into VRAM so the amount of characters you can use is limited.
This system only supports letters with a size of `8px` x `8px`.

The font should have the following structure, you can follow the file below and adjust it.



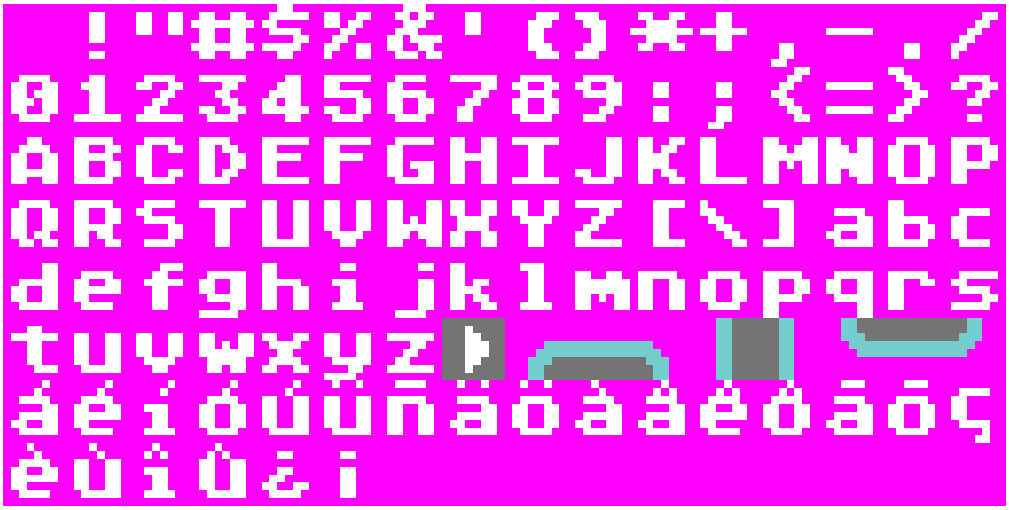
Be aware, there are 10 slots that get used for the dialogue box and arrow cursor.
They can be empty if you load them with their proper events or be directly included in the font.

> **TIP**
>
> The example displays transparent background for the letters.
>
> If you want the text to be inside the dialogue box all the time, make sure to change the pink with the dialogue box color.

Simple Mode (EFIGSP)

This mode loads the entire font into VRAM so the amount of characters you can use is limited.
This system only supports letters with a size of `8px` x `8px`.

The font should have the following structure, you can follow the file below and adjust it.



Be aware, there are 10 slots that get used for the dialogue box and arrow cursor.

> **WARNING**
>
> Be aware, the tiles after ¿¡ tiles won't be loaded into the VDP.

Advanced Mode

This system allows for any amount of letters you need as the font is not loaded into VRAM.

This is the best system for Asian languages due to the need for many characters and support for letter sizes over `8px` x `8px`.

For this system you need to convert TTF fonts with the Font Converter into PNG.

New Fonts

If you want to create new fonts, first you need to decide what text mode you are going to use as the process is slightly different depending on that.

Simple Modes

For the Simple Modes, you will need to create a single *.png* file if you want to create your own font, you just need to follow the previous samples.

> **WARNING**
>
> The letter order is very important as the engine knows what letter to draw based on that pre-defined order.

Advanced Mode

For the Advanced Mode, you will need a *.png* file and a *.json* counterpart.

The fastest way will always be to use the Font Converter though.

> **WARNING**
>
> Creating fonts manually is a complex procedure, using the Font Converter is recommended to create at least the file outline to then edit on top.

JSON File

The font *.json* file contains the font information as well as the codepoint order and size.

Here is a small example:

```
{
  "name": "<font name>",
  "size": "<number>",
  "gridWidth": <8 or 16>,
  "gridHeight": <8 or 16>,
  "codepoints": [
    {  "codepoint": <codepoint>, "width": <width in pixels>, "xadvance": <xadvance in pixels> },
    ...
  ]
}
```

PNG File

The font *.png* file should follow the order in which codepoints are noted in the JSON file.

Font Colors

The default font, as well as the fonts generated with the font converter for the Advanced Mode, use the last color (color 15) of the selected Text palette as the text color.

You can see an enlarged image with the respective palette on the Dialogue & Text section.

Font Converter

The font converter can be used to generate PNG files from TTF files.
This will help include fonts much faster into MD Engine than manually drawing them!

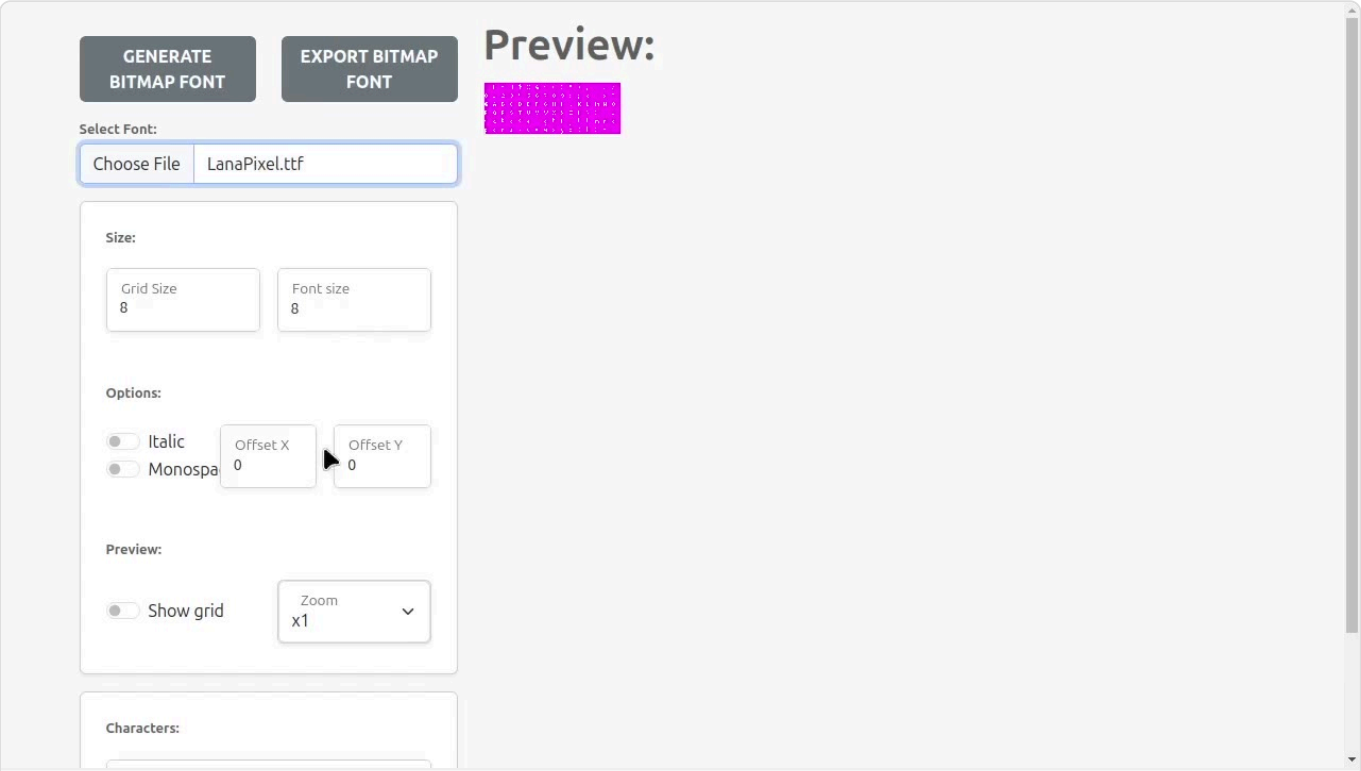
How to convert a TTF to PNG

First of all you need to select a TTF file.

After that, you can adjust the *Grid Size*, *Font Size*, and any other parameter until you get the desired result.

You can also add or remove characters in the *Characters* box to make sure all are included.
The exact list of used characters can be found in the *Unique Characters* area of the *Localization* view.

Here you can see a small video of how we generate a PNG from a TTF.



Video: font-converter-sample.mp4 — view online in the web documentation.

Music

Music can be imported with the + button or by placing files in your project's `assets/music` folder.

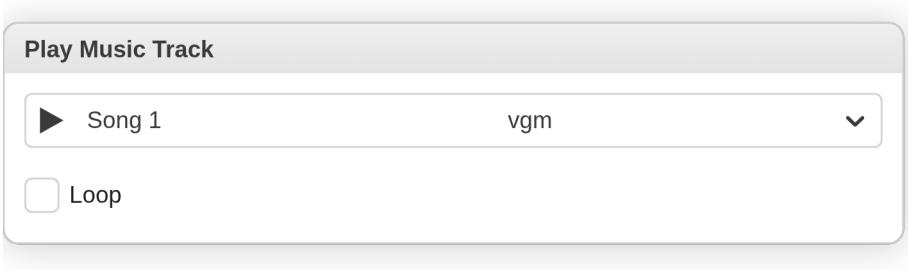
Supported formats depend on the active Sound Driver:

- **XGM2 / XGM** — `.vgm`, `.xgm`, `.wav`, `.mtrk` (Mini Tracker)
- **MDSDRV** — `.mml`

VGM is the recommended format for XGM2/XGM because it uses less ROM and has better quality. WAV files are converted to either 13.3 kHz or 6.65 kHz during build time — see Audio Settings for details.

You can also compose music directly inside MD Engine using the built-in Mini Tracker (grid-based tracker) or Mini Notator (text-based MML editor). Both support the full Mega Drive sound hardware and can export to VGM.

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



You can review the Sound Driver asset types section to find out more about the number of sound effects you can play at the same time.

Sound Effects

Sound Effects can be imported with the + button or by placing files in your project's `assets/sounds` folder.

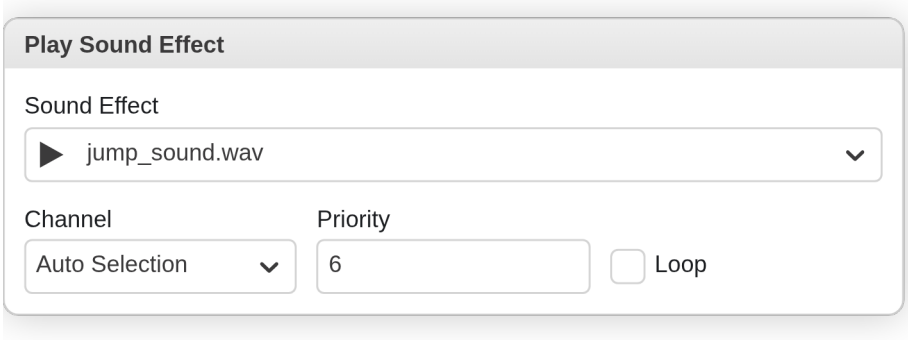
Supported formats depend on the active Sound Driver:

- **XGM2 / XGM** — `.wav`
- **MDSDRV** — `.mml`

WAV files are converted to either 13.3 kHz or 6.65 kHz during build time — see Audio Settings for details.

You can create sound effects directly inside MD Engine using the built-in Bfxr tool, which generates retro-style sounds (pickups, lasers, explosions, jumps, etc.) and exports them as WAV.

They can be played during the game with the Play Sound Effect event in your *Scene*, *Actor*, *Trigger* or *Custom* scripts.



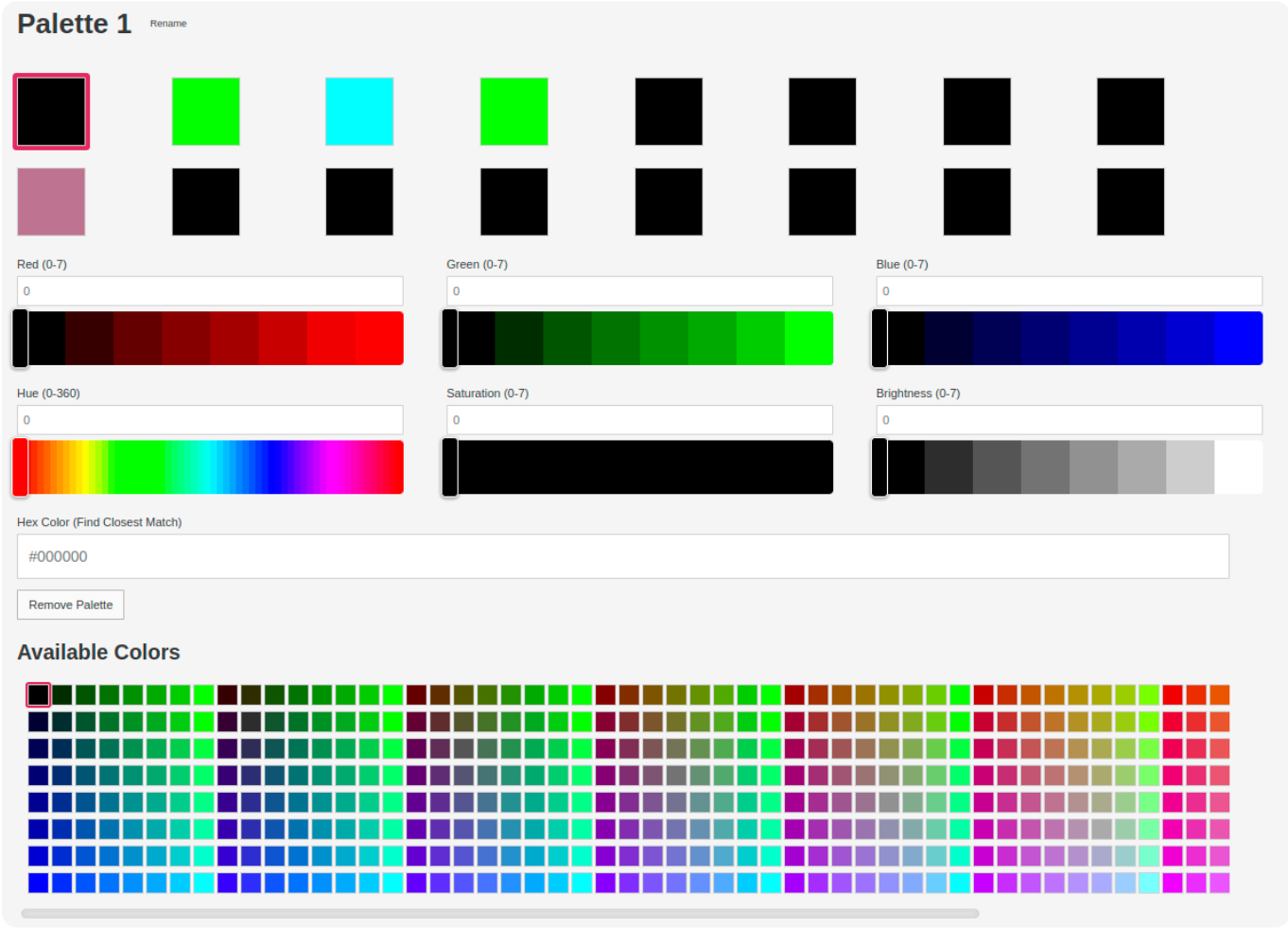
You can review the Sound Driver asset types section to know more about the number of sound effects you can play at the same time.

Palettes

The Palette asset contains the information about a specific game palette, consisting of 16 colors.

You can create a new empty palette with the button, import the palette from a specific PNG file with the button, or extract all the game palettes contained in the game PNG images with the button.

You can edit each palette color by clicking on it and then either shifting the color bars, entering a HEX value or by picking one of the available colors.
If you enter a HEX value, the color would be mapped to an available color automatically.



Palette Structure

The Mega Drive supports 4 palette lines of 16 colors each, for a total of 64 colors on screen at any time. Each color is 9-bit RGB (3 bits per channel), giving 512 possible colors.

- **Line 0** — Typically used for backgrounds. The first color (index 0) is the global background/transparent color.
- **Lines 1–3** — Used for backgrounds, sprites, or both, depending on your project's needs.

Each 8x8 tile (whether background or sprite) can only use colors from a single palette line. You assign a palette line per tile, not per pixel.

Sprites

Sprites can use any of the 4 palette lines, but each sprite tile is limited to the 16 colors of its assigned line. Sprites support a maximum of 16 colors per tile.

Backgrounds

Background tiles follow the same rule — each 8x8 tile references one palette line. A single background image can use all 64 colors across different tiles, as long as each individual tile stays within one palette line.

Palette Events

You can manipulate palettes at runtime using scripting events:

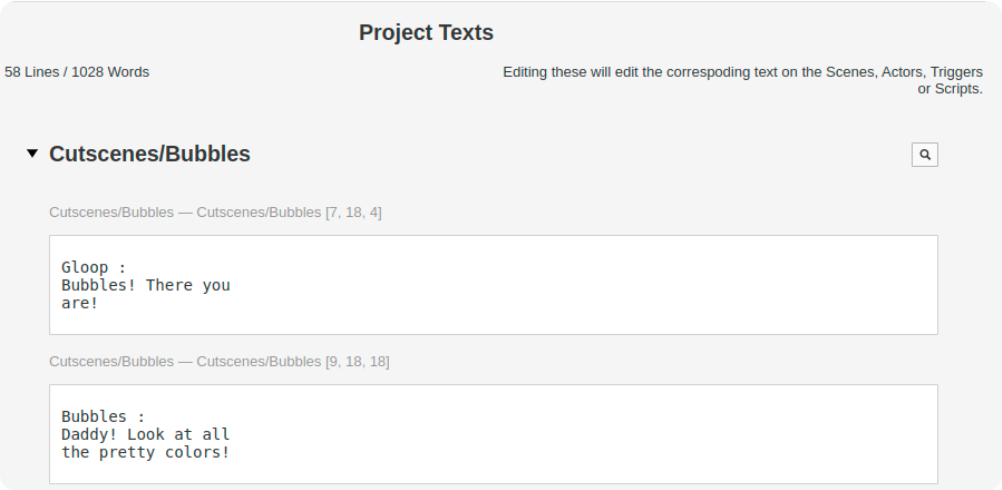
- Load Palettes — Load a palette asset into a palette line
- Preload Palettes — Preload palettes without applying them immediately
- Fade In / Fade Out — Fade palettes to/from black

Localization

The *Localization* view contains three different sections, *Project Texts*, *Unique Characters*, and *Localizations*.

Project Texts

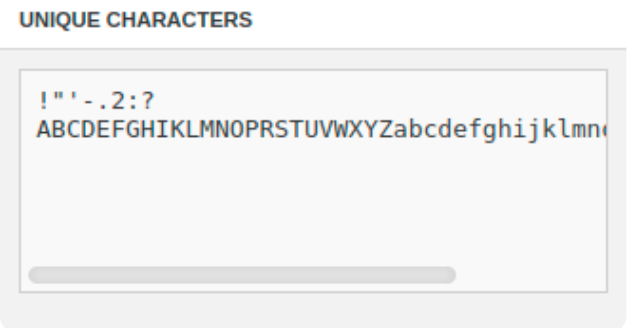
This contains information about all the text that is displayed on the game with the different text-related events.



Editing a text on this area will edit the corresponding text on the *Scene*, *Actor*, *Trigger*, or *Custom* script.

Unique Characters

This contains a list of all the unique characters used in the game. This is useful during the font import process, for information regarding this you can check the Font Converter section.



Localizations

This contains a list of all the localization assets in the game.



You can create a new empty localization with the ➕ button.
After that you can go to your project's `assets/localization` folder and edit the new text file.

You can edit the generated txt file with your preferred text editor, like Sublime Text or Notepad++.
Make sure to use something that supports UTF-8 encoding though.

The file has the following format:

```
// Localization file
//
// The file format is composed of pairs of original and localized text.
// Duplicated strings are removed.
// Upper line is original, bottom should be the localization.
// DO NOT MODIFY ORIGINAL STRING
// Lines that start with // are comments
// $XX$ represents variable replacements.
// \n represents a line break on the text
Froggy:\nHey, you haven't seen my pet fly\nbuzzing around, have you?\nPoor guy has gotten lost
again!

Gloop:\nNo, sorry! I'm looking for my kids\nmyself.\n

Froggy:\nLooks like we've both got our hands\nfull! Good luck with your quest!

Gloop:\nYou too, my fellow green friend!\nI'll holler if I spot your buzzing\nbuddy!

Welcome to INKVENTURE!\n\nUse the DPAD to move.\nPress A to jump.

(etc)
```

In order to translate the file, you need to place the translation under the original string.

```
// Localization file
//
// The file format is composed of pairs of original and localized text.
// Duplicated strings are removed.
// Upper line is original, bottom should be the localization.
// DO NOT MODIFY ORIGINAL STRING
// Lines that start with // are comments
// $XX$ represents variable replacements.
// \n represents a line break on the text
Froggy:\nHey, you haven't seen my pet fly\nbuzzing around, have you?\nPoor guy has gotten lost
again!
フロッギー：\nねえ、どこかで僕のペットのハエを見なかった？\nブンブン飛び回ってるやつなんだけど、\nまたどっか行っちゃっ
たみたいなんだ〜。
Gloop:\nNo, sorry! I'm looking for my kids\nmyself.\n
グループ：\nあー、ごめん！　こっちも自分の子どもたちを\n探してるところなんだ。
Froggy:\nLooks like we've both got our hands\nfull! Good luck with your quest!
フロッギー：\nそっか、お互い大忙しだね！\n探しもの、がんばろう！
Gloop:\nYou too, my fellow green friend!\nI'll holler if I spot your buzzing\nbuddy!
グループ：\nああ、緑の仲間よ！\n君のブンブン友だちを見つけたら、\nちゃんと呼ぶからね！
Welcome to INKVENTURE!\n\nUse the DPAD to move.\nPress A to jump.
INKVENTURE へ ようこそ！\n十字ボタンで移動。\nAボタンでジャンプ！
(etc)
```

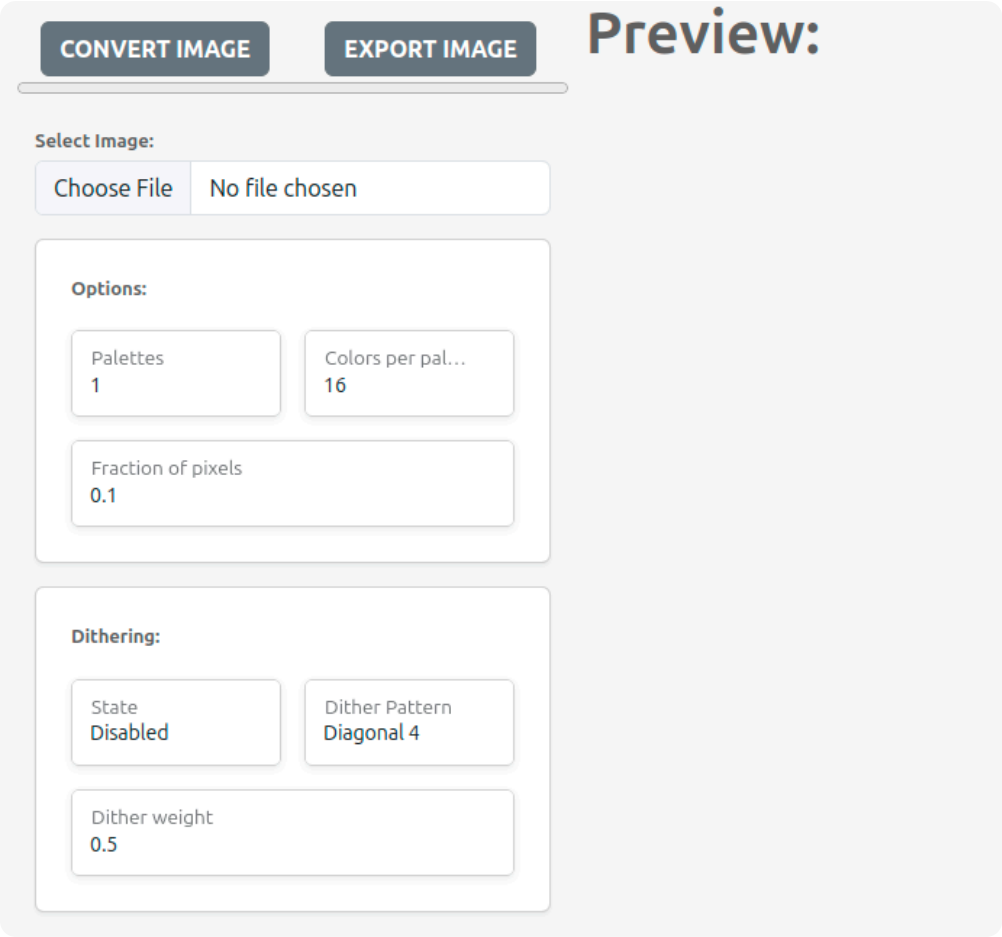
The engine during compile time will create the needed structures so you can swap languages with the Set Text Localization event.

Image Converter

The image converter can be opened from either the *Sprites* view or the *Backgrounds & Maps* view.

It allows the conversion of any PNG image into a format accepted by MD Engine.

This is done by basically reducing the number of colors in the image and making sure each tile, `8px` x `8px` has all its colors from the same palette.



How to convert an image

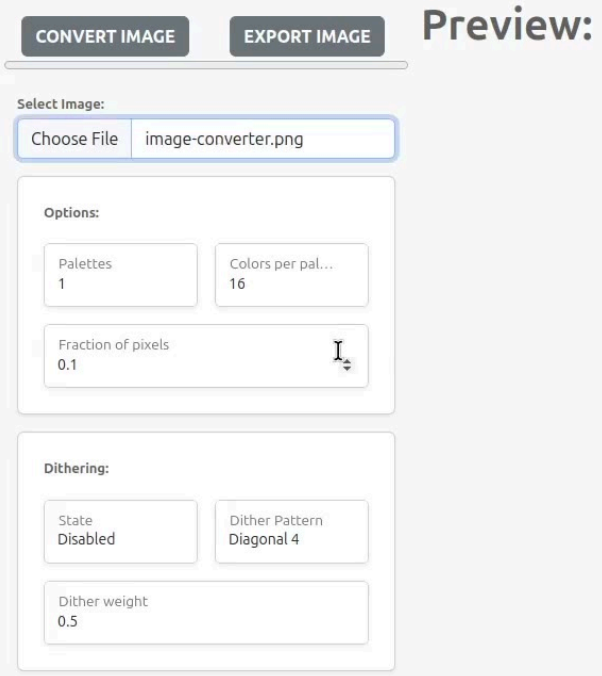
First of all you need to select a PNG file.
The image size must be a multiple of 8px in both width and height.

After that, you press *Convert Image*, this will generate the adjusted image.

You can tweak the number of used palettes, as well as number of colors, etc, until you get a result you like.
Remember, sprites can only have one palette.

Once you are happy with the result, you press *Export Image*, this will save the image in the relative `assets/` folder of your project.

Here you can see how we convert The Great Wave off Kanagawa image.
First of all we create a `320px` x `224px` image and then we load it with the converter.
After a few tweaks we finish with an indexed image compatible with MD Engine.



Video: image-converter-sample.mp4 — view online in the web documentation.

Finally, here you can see a before and after of the image we just converted.



You can always do adjustments of the parameters to meet your requirements and then edit the generated image with your preferred image editing tool.

Asset Usage Report

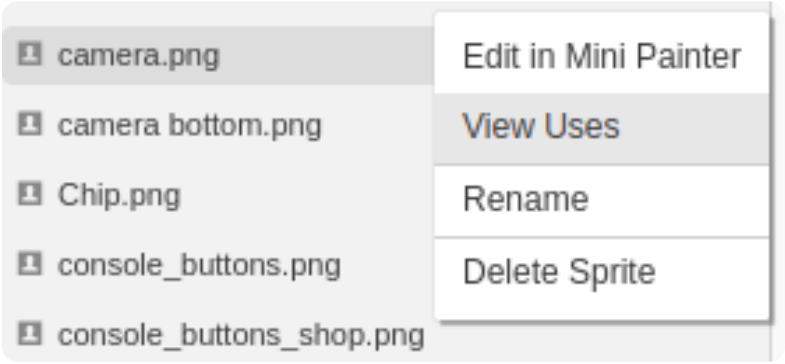
The Asset Usage Report shows you everywhere a specific asset is referenced in your project — which scenes, actors, triggers, and script events use it. This is useful for understanding dependencies before modifying or deleting an asset.

Viewing Asset Uses

There are two ways to open the usage report:

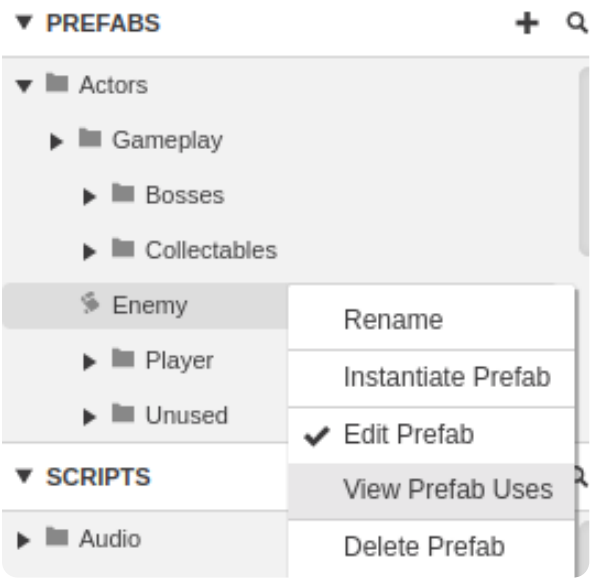
From the Asset Editor

In any asset editor (Backgrounds, Sprites, Palettes, Music, Sounds, Fonts, Localizations), click the **dropdown menu** and select **View Uses**.



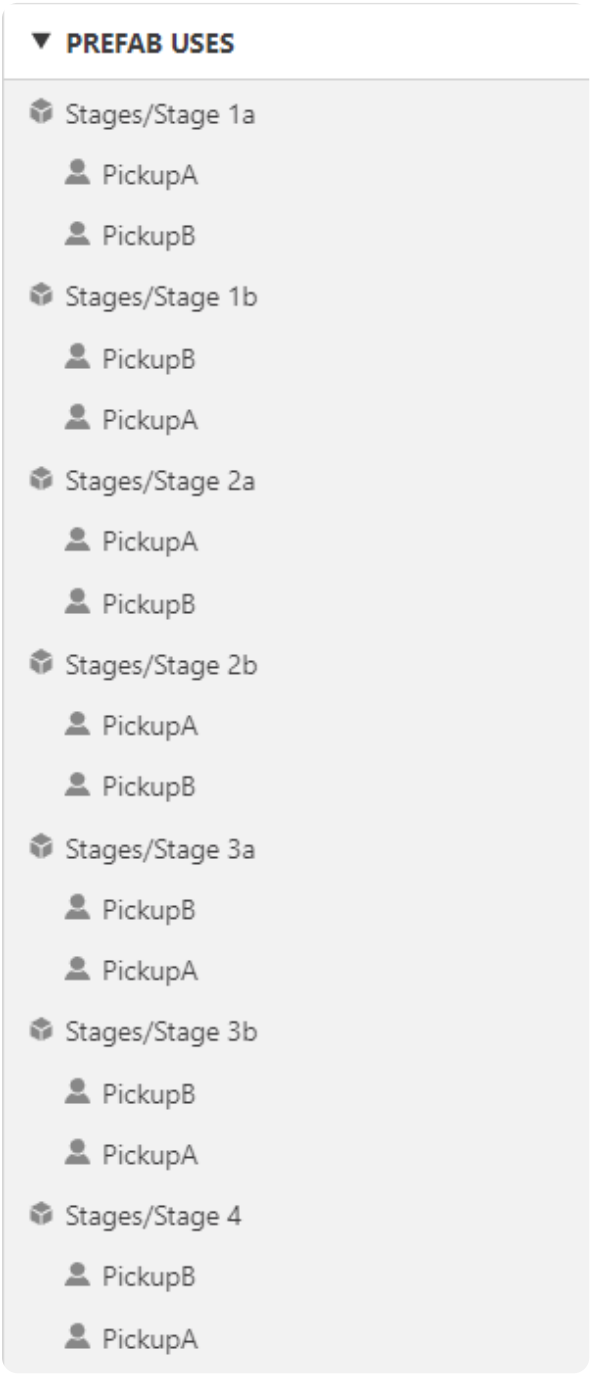
From the Navigator

Right-click any asset in the Navigator sidebar and select **View Uses** from the context menu.



Reading the Results

The usage list shows all entities that reference the selected asset:



- **Scenes** appear at the top level (e.g., a scene using the asset as its background).
- **Actors and triggers** appear nested under their parent scene.
- **Custom events** appear if any custom script references the asset.
- The total count is shown in the header.

If the asset is not used anywhere, a message indicates it is safe to remove.

Navigating to a Usage

Click any entry in the list to navigate directly to it:

- **Scene entries** — Selects and focuses the scene. If the match is in a script event, the correct script tab is opened and the view scrolls to the event.
- **Actor/Trigger entries** — Selects the actor or trigger within its scene and navigates to the matching script event if applicable.
- **Custom Event entries** — Opens the custom event editor.

> **TIP**
>
> The scan runs on a background thread and won't block the editor. For tileset assets, TMX map file references are also detected.

The Engine

MD Engine uses a custom-made compiler to generate code that can then be compiled with GCC into machine code.

The game then runs using TBCE (Two Black Cats Engine) which is built on top of the SGDK library.

The compiler enables the needed modules on the fly depending on the used events and engine settings.

Events to C process

When building the game, MD Engine will convert all the events into either C code or ASM code depending on the specific event.

Optimization process

When creating the final C functions from the scripts, the compiler will try to optimize the output to produce the smallest number of functions, so if you have several actors with identical events but different parameters for some of the values, the compiler will try to use a generic function with a *params* argument to match the needed arguments per actor.

This generates a smaller code footprint but introduces a small overhead due to parameter passing. In the vast majority of cases, this is preferred over having each actor in each scene have its own function.

Async Scripts

MD Engine has async events, these are events that start on one frame and finish some frames later.

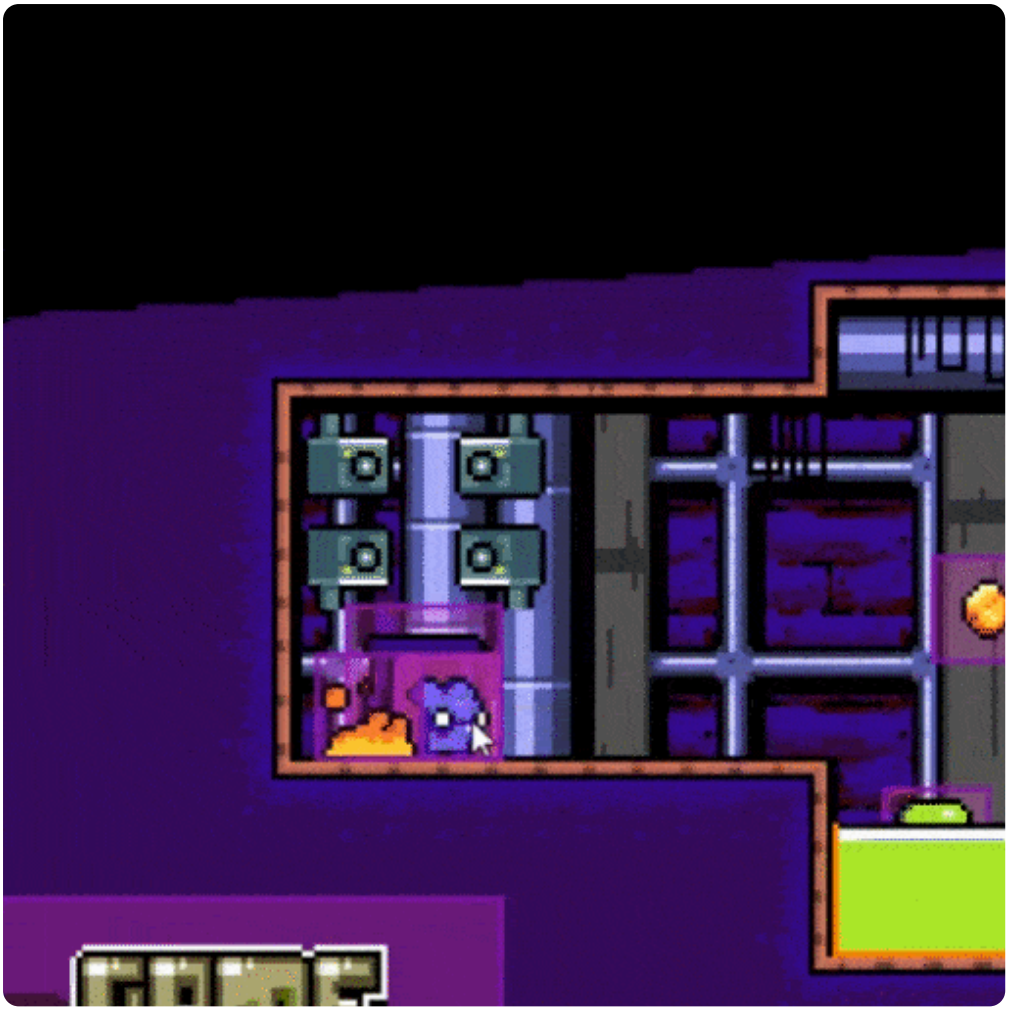
You can check more about async scripts on the Async Events section.

Scene Processes

Scenes follow a lifecycle of Start, Update, Exit, and Pause processes. For detailed information on each process, see Scene Processes.

Actors Depth on Scene

The order in which actors are added to a Scene reflects their current depth. You can change the depth by right-clicking and selecting *Depth* and the desired action.



This will also **change the order** in which their script functions are run.

Changing the depth during playtime won't change the function execution order though.

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 & Maps for more information on adding background images.

Scene Properties

- **Name** - The name of your scene. Useful for locating your scene with the search bar and scene navigator.

> **TIP**
>
> Using `/` characters in your scene's name will cause it to appear within folders in the scene navigator (e.g. `ui/title screen` will appear as `title screen` within the `ui` folder)

- **Resolution Mode** - Lets you choose the size of your scene, *H40* features `320px` width and `224px` height, *H32* features `256px` width and `224px` height. It's useful if you want to display splash screens but use less ROM space by having narrower images.

- **BG Color (0 - 63)** - Lets you choose the palette color index for the background, if you have automatic palettes it won't display on the editor.

- **Plane B (Background)** - Lets you choose a backgrounds or maps file from the `assets/backgrounds` folder for the Background Layer.

- **Plane A (Foreground)** - Lets you choose a backgrounds or maps file from the `assets/backgrounds` folder for the Foreground Layer.

- **Initial Palettes** - If automatic is selected, the palettes will be selected based on the background and foreground files as well as the actors, if you select manual, you can attach up to four different palettes created on the Palettes view.

- **Force to Scene Size** - This makes the scene render on the fixed resolution you selected on *Resolution Mode* instead of falling back to the Background or Foreground size, it's useful if you have scrolling backgrounds on the main menu or other UI scenes to have a cleaner editor.

Plane B (Background)

The Plane B (Background) is usually used for backdrops, distant scenery, or large static backgrounds.

It can also use the priority bit to appear in front of sprites if desired.

It can scroll independently of Plane A (both horizontally and vertically), make sure to review Backgrounds & Maps > Maps vs Images requirements, as parallax scrolling only works with images.

Plane A (Foreground)

Plane A (Foreground) is often used for HUDs, menus, text, or foreground scenery.

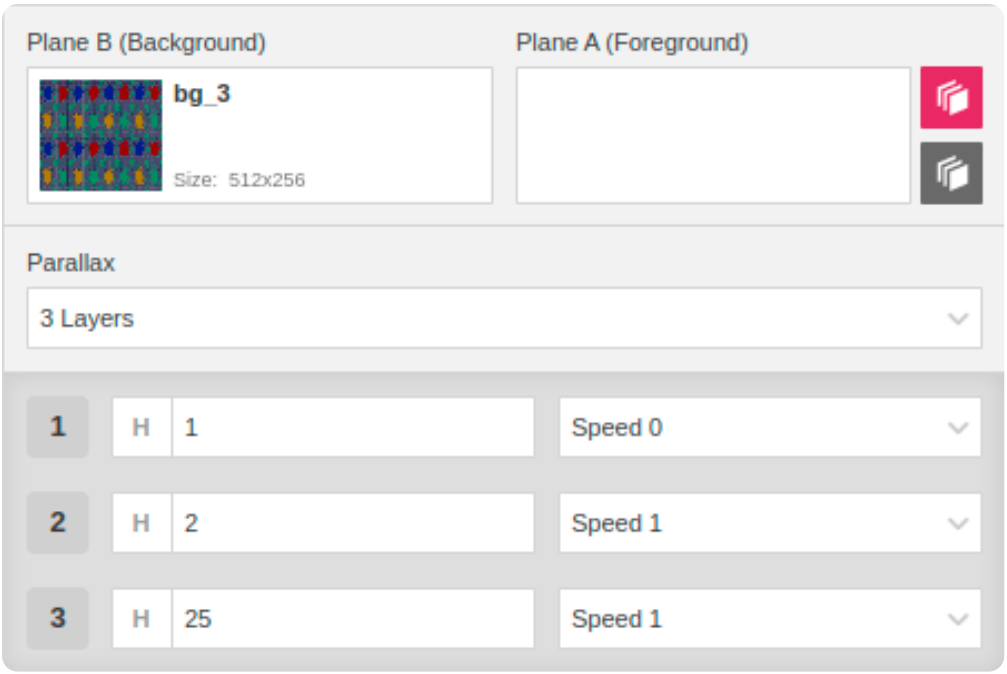
Plane A usually sits in front of Plane B, but sprites can appear in front of or behind depending on the priority bit.

It can also be used for parallax scrolling — giving the illusion of depth by moving at a different speed than Plane B, make sure to review Backgrounds & Maps > Maps vs Images requirements, as parallax scrolling only works with images.

Parallax Mode

Clicking the  button to the right allows you to turn on parallax mode (the top button is for *Plane B (Background)* parallax and the bottom button for the *Plane A (Foreground)* parallax) for the scene.

When parallax mode is enabled you can split the plane into up to three slices which can be modified to scroll at different speeds as the camera moves in game.
Be aware as noted on the Backgrounds & Maps > Maps vs Images section this only works with images and not with Tiled maps.



Palettes

The palette system has two different modes per scene.

- **Automatic** - The automatic mode picks the palettes based on the backgrounds or maps and actor sprites placed on the scene.
- **Manual** - The manual mode allows you to select four different palettes from the Palettes view that would be used on the scene.

Scene Scripts

Scenes can contain four different scripts.

- **On Start** - This script will be called as soon as the scene is loaded in the game. You can use this to do things like playing music as the scene loads, configuring events to happen on button presses, initializing actors based on the values of variables, and much more.
- **On Update** - This script will be called every frame before the *Actor* and *Trigger* **On Update** scripts, this can control most of your scene logic, like when to pause the game, or background effects, and much more.
- **On End** - This script will be called as soon as the scene is unloaded in the game. You can use this to pause music, do a fade out, save score if needed, and much more.
- **On Pause** - This script will be called instead of the *On Update* when the game is paused, you can place your pause update logic here.

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 default 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 (Tile type)** - Allows you to check if an actor is on top of a ladder to allow moving up and down, you can check the Action Platformer example for a detailed usage.

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 collisions when placed on top of another collision.

Additionally there is a dropdown menu allowing you to view additional tile types:

- **Tile Types** - These tile types can be used to mark a tile with a specific type, then you can check if an actor overlaps a specific tile type with the If Actor On Tile Type event. This is featured on the Game - Inkventure example.

Changing Palettes & Priorities on a Scene

Select the  from the *Editor Tools*. This allows you to change the default palette for the background and foreground layers as well as the tile priority.

For more information about the drawing mode used for the *Palettes & Priorities tool* and the *Collision tool*, see Keyboard Shortcuts.

Changing Palettes

You can select from *P0* to *P3* for either background or foreground. After that you can click on the desired tiles.



> **WARNING**
>
> This requires some testing as the final palette depends on the one you use on each layer.
>
> For example, if the *Plane A (Foreground)* uses Palette 1, and you tick to use Palette 0, the result would still be Palette 1!

Tile Priority

You can change the tile priority for either *Plane B (Background)* or *Plane A (Foreground)* from low-priority, the default one, to high-priority, these tiles will appear in front of actors allowing you to create depth in your scenes.

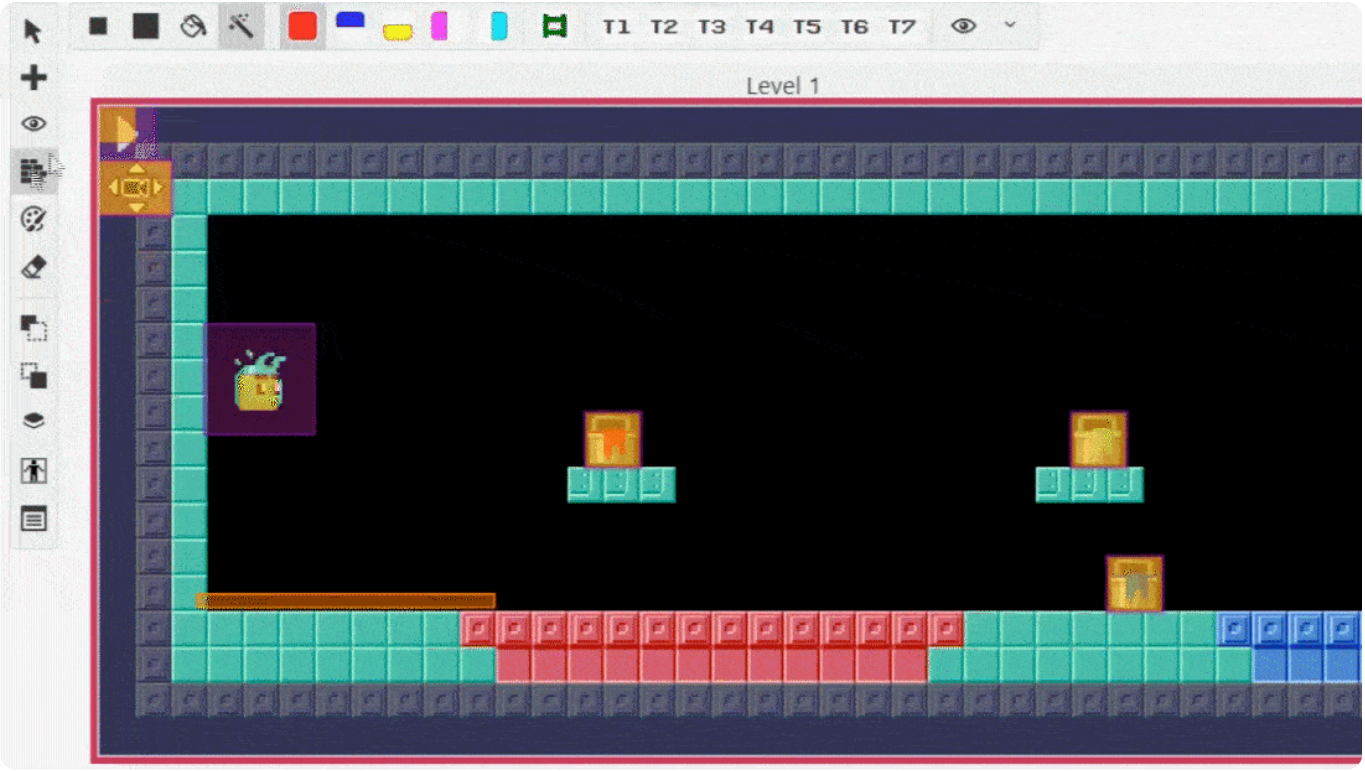
Select the  from the *Tool Bar* to change the *Plane B (Background)* priority, and select the  from the *Tool Bar* to change the *Plane A (Foreground)* priority.

The zones with high-priority will appear with a yellow or blue square respectively like in the next image.



Magic Brush

The Magic Brush is available when adding collisions, palette changes or priorities to a scene and can be used to paint every tile in the scene that matches the one you clicked instantly.



> **WARNING**
>
> The magic brush uses the information on the background plane first, if not present then on foreground.
>
> When the layer is a Tiled Map file it tries to use the information on the *Background Layer*.

Scene Limits

MD Engine uses per-scene limits to keep performance predictable.

Each scene can have a maximum of 40 actors, 60 triggers and 4 palettes.

You can check this information by selecting a scene and looking for the gray bar under your scene that reads: `A: 0/40 T: 0/60 P: 0/4`. The letters on this bar represent the following:

- `A:` represents the number of actors that the scene is using.
- `T:` represents the number of triggers that the scene is using.
- `P:` represents the number of palettes that the scene is using.

> **WARNING**
>
> The actor and trigger limit is a recommendation. The game does build even if you exceed it.
>
> But we can't guarantee performance or that it even fits on VRAM or RAM.
>
> Due to this we won't be reviewing support requests where the performance limits are by-passed.
>
> Use at your own risk!

Actor Limits

Each scene can have a maximum of 40 actors.

Make sure to check the Hardware Sprite information, clustering too many actors together in a scene will cause some actors to become invisible in-game on real hardware or emulators. For *Desktop Export* you can avoid this with the Enhanced Mode.

Trigger Limits

Each scene can have a maximum of 60 triggers.

Background and Sprite Tile Limits

The engine by default has 1004 Background & Foreground Tiles, 420 Sprites Tiles, and 96 Font Tiles.

You can change these settings from the Settings section or use the VDP Tiles Setup event on the scene *OnStart*, or *OnEnd*.

You can use the VRAM Viewer in the Debugger to get a visual representation of how tiles are being used in each scene in your game.

Replicate Actor

You can replicate an actor, this will copy the actor across all created scenes in the game.

This has been created to make it easier to replicate UI across different levels.

Scene Processes

MD Engine follows strict Scene processing steps. This ensures all functions are called at known times and avoids confusion when writing event code.

The Start Process

The **On Start** script is called before anything else, right before calling the script all the used modules get reset, except for the Global Arrays.

The scene start process is outlined in the following steps.

1. Setup important data such as total actors and triggers, scene size and collision map
2. Set scene mode, BG color index and reset camera module if present
3. Load the palettes and create a backup
4. Load background, then draw the background
5. Wait 1 VSync
6. Load the foreground, then draw the foreground
7. Wait 1 VSync
8. Initiate the actors
9. Scene **On Start** script
10. Actors **On Start** script
11. Preload sprites if needed

Be aware of the *Scene* **On Start** call position compared to the *Actors* **On Start** call position.

The start process is closely related to the exit process, as there is always a scene transition, even if you select no-splash build, there is a 2 frame black scene to set up the system. So it's recommended to review this closely with the Exit Process.

Setting Palettes inside On Start

Using the Load Palettes event can mess up fade-in effects. The best is to use Preload Palettes event instead.

The Update Process

The **On Update** script is called every frame, right before calling the function the engine runs the following tasks:

1. Pre update all actors
    - Sets the last X/Y position

    - Reset the acceleration X for actors with platformer behaviour enabled
    - Marks the hitbox as not updated
2. Update timers module if active
3. Update tweens module if active
4. Call the scene update process
5. Update camera module if active
6. Update the scroll module if active
7. Increase the game timer
8. Check if a pause request has been issued
9. Sync all actors positions

The scene update process is outlined in the following steps.

1. Run the scene **On Update** script logic
2. Run the actors **On Update** script logic
3. Update behaviours
    - Update actor position based on physics and update hitbox
    - Clear actors touching values
    - Check actor against groups overlaps
    - Check group against same group overlaps
    - Check actors against scene collisions
    - Run the async **On Collide** script pending logics
4. Check triggers overlaps
5. Update projectiles module if active
    - Update actor position based on physics and update hitbox
    - Check overlap with actors
    - Check projectile against scene collisions
6. Check temporary triggers overlaps
7. Update particle position based on physics

Be aware of the *Scene* **On Update** call position compared to the *Actors* **On Update** call position.

The Exit Process

The **On Exit** script is called when we change from one scene to another.

The scene exit process is outlined in the following steps.

1. Run the actors **On Exit** logic
2. Run the scene **On Exit** logic
3. Free preloaded sprites if needed
4. Release actors
5. Clear maps
6. Reset the scene

7. Clear loaded images
8. **On Debug mode** clear the VDP
9. Free projectiles module if active
10. Free all non-global arrays, if the module is active
11. Free particles module if active
12. Reset the image scrolling
13. Reset scrolling to affect entire plane
14. Free the thread module if active
15. Free the timers module if active
16. Free the tween module if active
17. Reset the camera module if active
18. Update VDP layout if requested during **On Exit**
19. New scene start process
20. Update camera after the start in case we moved the follow element on the start method

Be aware of the *Scene* **On Exit** call position compared to the *Actors* **On Exit** call position, as it's inverted from the other two scripts.

The Pause Process

The **On Pause** script is called every frame while the game is paused, right before calling the function the engine runs the following tasks:

1. Pre update all actors
    - Sets the last X/Y position
    - Reset the acceleration X for actors with platformer behaviour enabled
    - Marks the hitbox as not updated.
2. Call the scene **On Pause** script
3. Check if an un-pause request has been issued
4. Sync all actors positions

The pause process is very simple due to all systems being paused and it's expected that there will be very little or no animations or movements altogether. Also, there is no async script processing.

HBlank & VBlank

The Mega Drive draws the screen one horizontal line at a time, from top to bottom. After drawing each line, there is a brief pause called the **Horizontal Blank (HBlank)**. After all lines are drawn, there is a longer pause called the **Vertical Blank (VBlank)** before the next frame begins.

During these pauses the CPU is free to make changes (like swapping colors or adjusting scroll positions) that will take effect on the *next* line or frame. This is how many classic effects like gradient skies, water reflections, and split-screen UIs are achieved.

In MD Engine, you can attach scripts to these intervals:

- **HBlank** — Runs once per scanline (or every N scanlines). Use it for per-line effects like palette gradients or wavy scroll.
- **VBlank** — Runs once per frame. Use it for per-frame changes that need to happen during the blanking period.

You can enable these with the OnStart: Set Horizontal Blank Script event for HBlank and OnStart: Set Vertical Blank Script event for VBlank.

> **INFO**
>
> These events can only be placed inside a Scene *On Start* script.

There are some operations that can be done during these periods, such as changing palette colors, adjusting Window Plane or changing scroll values.

You can review the HBlank - Palette example for a palette effect, and HBlank - Window Plane for a Window Plane effect.

The HBlank script can be triggered every scanline if a *Scanline Counter* of 0 is used on the OnStart: Set Horizontal Blank Script event. Any other value will make the script trigger every (value + 1) scanlines. For example, setting 5 means that the script will occur each (5+1) scanline.

Allowed events inside HBlank & VBlank scripts

Inside these scripts there is a limited amount of events you can use, all the possible events start with *Blank:* if they can be used inside both scripts or with *HBlank:* for Horizontal Blank scripts only, and *VBlank:* for Vertical Blank events.

This is because the events inside these scripts are compiled to ASM instead of C code due to performance needs.

A full list of events can be seen below:

- Blank: If Scanline Is
- Blank: Palette Set Color
- Blank: Palette Set Color From Palette
- Blank: Scroll Set Value

- Blank: Window Plane Disable
- Blank: Window Plane Enable
- OnStart: Set Horizontal Blank Script
- OnStart: Set Vertical Blank Script
- Scene Disable Blank Scripts
- Scene Enable Blank Scripts
- Set HBlank Scanline Counter
- VBlank: Scroll Restore

> **WARNING**
>
> Using events that are not on this list will result in the game not building!

Scene Notes

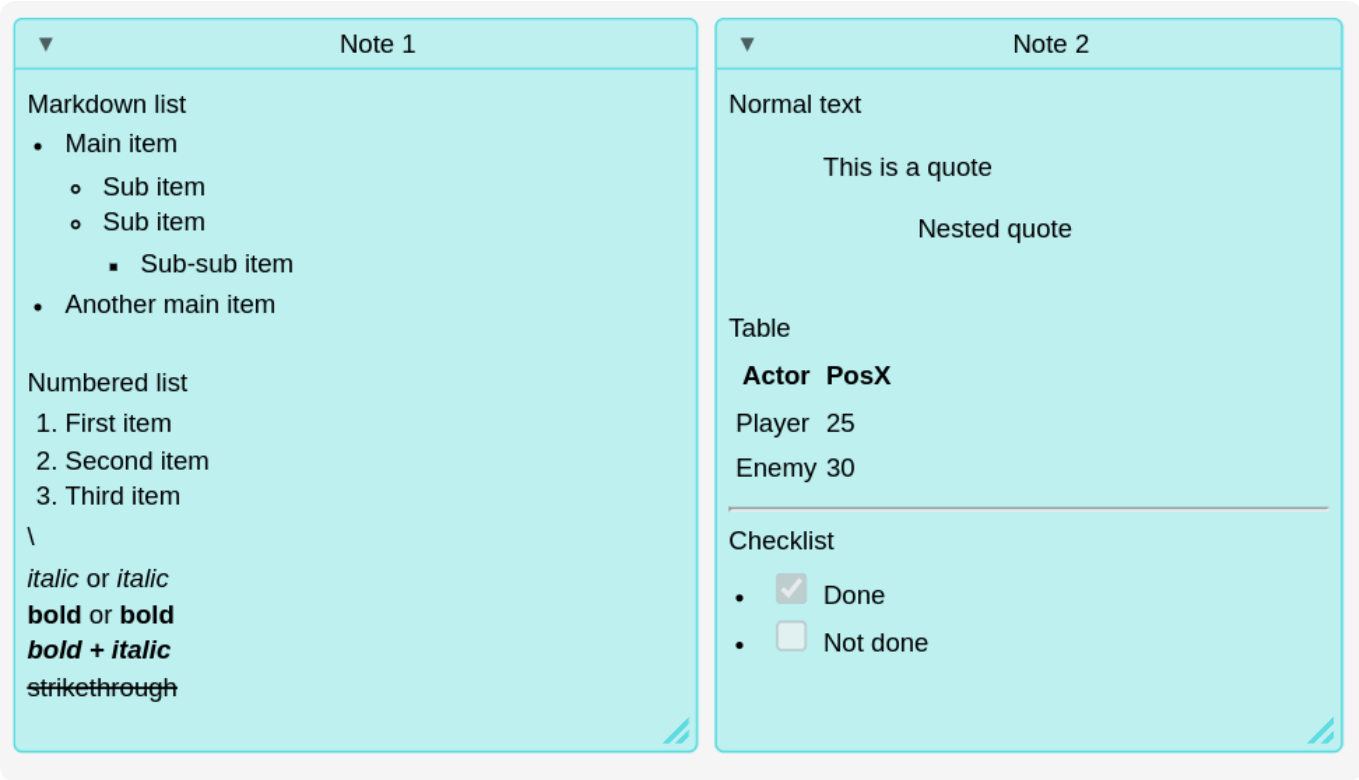
Notes are free-form text blocks that you can place anywhere on the world canvas. They are for your own reference and have no effect on gameplay.

Creating a Note

Right-click on empty space in the world view and select **Add Note**, or use the *Add* tool and choose *Note*.

Editing a Note

Double-click a note in the world view to enter edit mode. Type your content using **Markdown** syntax — headings, bold, italic, lists, and links are all supported. Press `Escape` or click outside to return to the rendered view.



Collapsing Notes

Click the **chevron** in the note header to collapse or expand the note body. This saves screen space when you have long notes. You can also right-click the note and choose **Collapse** or **Expand**.



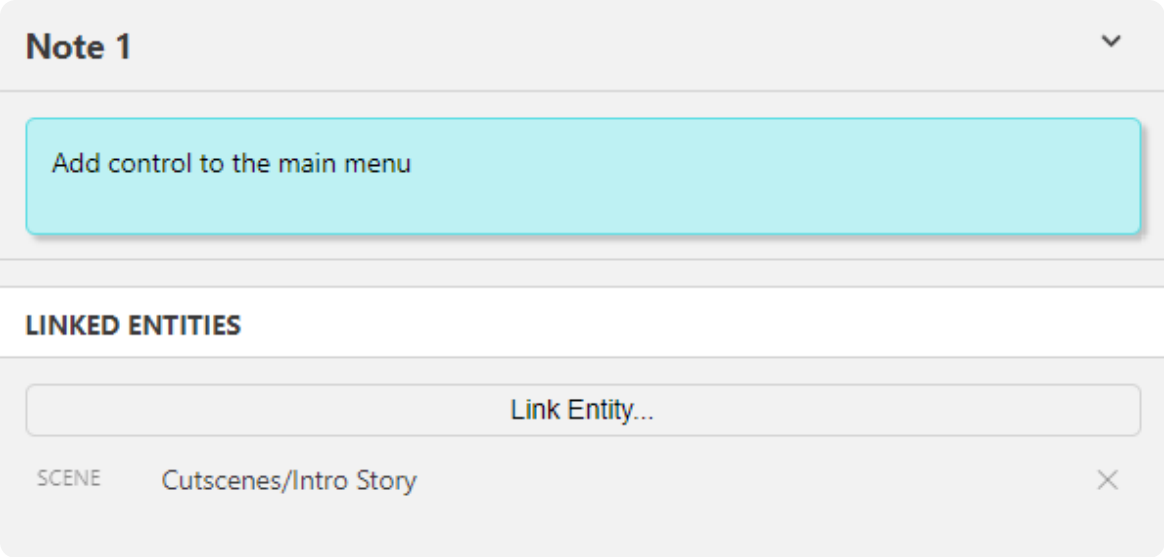
Linking Entities

Notes can be linked to actors, triggers, and scenes in your project. Linked entities appear as clickable tags — clicking one navigates to that entity in the editor.

To link an entity:

1. Select a note to open it in the *Editor Sidebar*.
2. Click **Add Link** to open the entity picker.
3. Browse the list (grouped by scene) and select the entity you want to link.

To remove a link, click the **x** next to the linked entity tag.



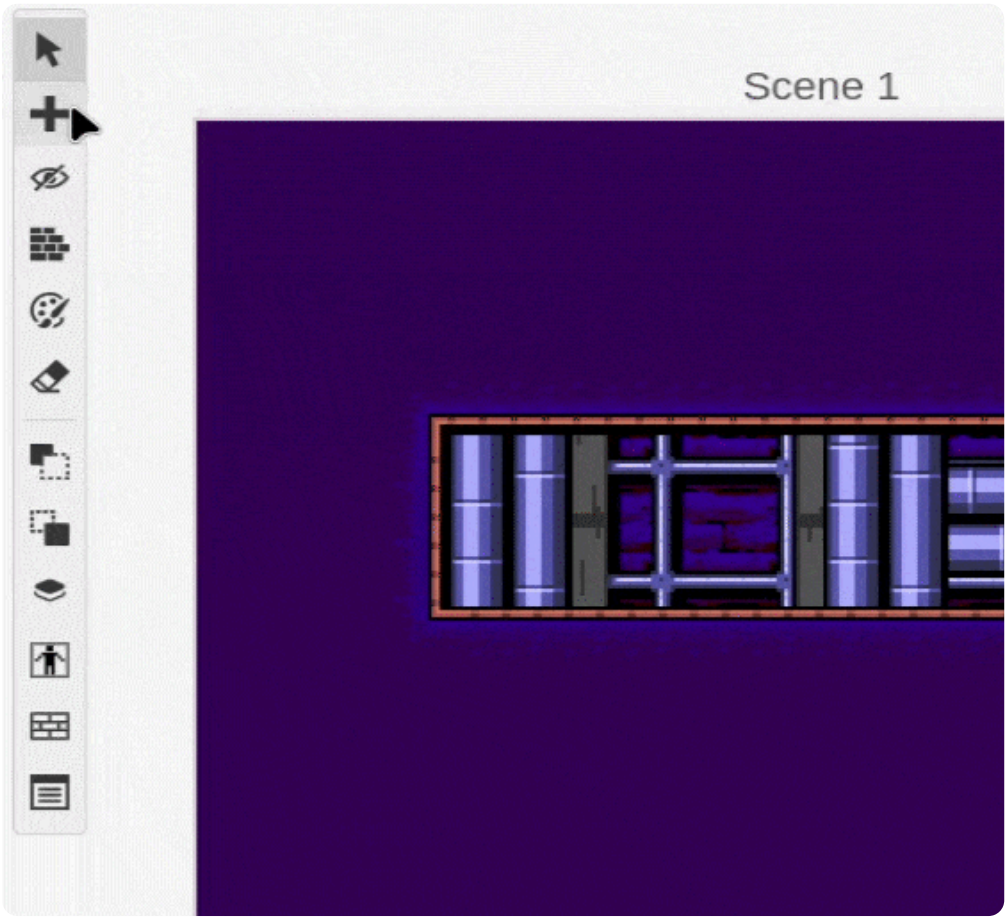
> **TIP**
> If a linked entity is deleted, its link is automatically removed from the note.

Actors

Actors are the player, 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

There are several settings you can adjust on the *Editor Sidebar* for your actors:

- **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 Hide Actor 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 *Game World*.
- **Pin to Screen** - Using the  button next to the actor position you can choose to pin the actor to the screen which causes it to not move as the game screen scrolls.

- **Palette Slot** - Selects the desired palette slot to use for this actor, if *Auto Select* is selected it will decide based on the spritesheet palette and the rest of the palettes in the scene.
- **Is Priority Actor** - If enabled it will make the actor appear over the foreground.
- **Direction** - The start direction of the actor.
- **Flip X** - If the actor sprite sheet should initially be flipped on the X axis when starting.
- **Flip Y** - If the actor sprite sheet should initially be flipped on the Y axis when starting.
- **Sprite Sheet** - Choose which sprite graphics should be used to display the actor.
- **Preload Frames** - Whether the sprite sheet will be fully preloaded into VRAM or streamed.
- **Collision Group** - Choose the collision group this actor belongs to.
- **Disable Scene Collision** - Fully disables the actor vs scene collision, this helps with performance.

Negative Coordinates

> **WARNING**
>
> Actors with negative coordinates will be treated outside screen, most logic discards actors with X or Y below 0.

Pin to Screen

When an actor is pinned it will appear in a fixed location as your game camera moves. The pinned 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 `320px x 224px` (H40) or `256px x 224px` (H32) 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.

A common example of usage would be UIs, such as health bar, ammo, etc.

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 Collide** scripts, see the documentation for Scripting.

Actor Scripts

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

- **On Start:** Called as soon as the scene is loaded in the game.
- **On Update:** (only if Update Mode is set to Parallel) Repeatedly called every frame, once the script finishes it will repeat. You can use this to create movement scripts.
- **On Interact:** (only if Update Mode is set to Interact) This is called if another actor stands in front of this actor and the *Actor Interact* event is called.
- **On End:** This script will be called as soon as the scene is unloaded in the game.
- **On Collide:** (only if a collision group is set) Called when this actor collides with an actor or projectile in a group enabled by the collision mask. This can include actors in the same group.

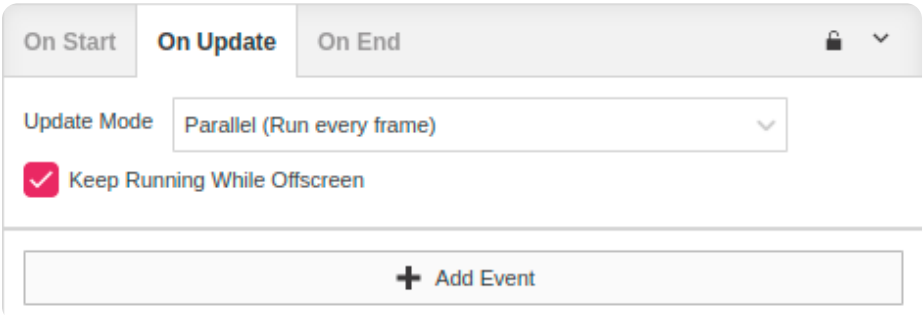
To start building a script, select an actor, click the script type you want to edit and click the *Add Event button* in the *Editor Sidebar* to open the event menu. Select an event to add it to the script.

For more information see the documentation for Scripting.

On Update Settings

Inside the **On Update** script there are two extra settings, **Update Mode** and **Keep Running While Offscreen**.

- **Update Mode:** If you select *Parallel* the script will run every frame, if you select *Interact* it will only run when something interacts with it.
- **Keep Running While Offscreen:** If ticked the OnUpdate script will keep running even if the actor is outside screen.



Interactions

Actor interactions can be invoked with Actor Interact event.

When this event is called, the X/Y position as well the actor direction would be used.
If there are other actors (such as NPCs) in front of the interacting actor (Player) their **On Interact** script will be called. (In other words, you must be facing the actor to interact. Set your direction using the Set Actor Direction event).

See the Top-Down example for a demonstration of usage.

Actor Scripting Properties

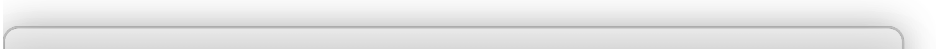
There are several actor properties you can use inside Script Values when creating scripts.

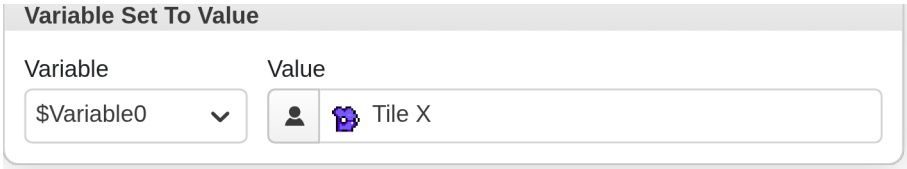
- **Tile X** - The *left* side position value in tile coordinates.
- **Tile Y** - The *top* side position value in tile coordinates.
- **Tile X (Right Side)** - The *right* side position value in tile coordinates.
- **Tile Y (Bottom Side)** - The *bottom* side position value in tile coordinates.
- **Tile X (Center)** - The *center* position value in tile coordinates.
- **Tile Y (Center)** - The *center* position value in tile coordinates.
- **Pixel X** - The *left* side position value in pixel coordinates.
- **Pixel Y** - The *top* side position value in pixel coordinates.
- **Pixel X (Right Side)** - The *right* side position value in pixel coordinates.
- **Pixel Y (Bottom Side)** - The *bottom* side position value in pixel coordinates.

- **Pixel X (Center)** - The *center* position value in pixel coordinates.
- **Pixel Y (Center)** - The *center* position value in pixel coordinates.
- **Width** - The actor bounding box *width* value in pixels.
- **Height** - The actor bounding box *height* value in pixels.
- **Velocity X** - The actor *velocity* in the *x axis* in pixels per second.
- **Velocity Y** - The actor *velocity* in the *y axis* in pixels per second.
- **Acceleration X** - The actor *acceleration* in the *x axis* in pixels per second squared.
- **Acceleration Y** - The actor *acceleration* in the *y axis* in pixels per second squared.
- **Is Touching Floor** - If the actor was, in the last frame, touching any collision tile/actor with the actor's *bottom* side of the collision box (i.e., standing on something).
- **Is Touching Left** - If the actor was, in the last frame, touching any collision tile/actor on the *left* side of the actor's collision box.
- **Is Touching Right** - If the actor was, in the last frame, touching any collision tile/actor on the *right* side of the actor's collision box.
- **Is Touching Up** - If the actor was, in the last frame, touching any collision tile/actor with the actor's *top* side of the collision box (e.g., hitting your head on a ceiling).
- **Current Animation Index** - The actor current *animation index* value as number (0 being the 1st animation row).
- **Direction** - The actor current *direction* value as number.
- **Frame** - The actor current *animation frame* value as number (0 being the 1st animation frame).
- **Is Animation Done** - If the actor current *animation is finished*.
- **Is Visible** - If the actor is *visible* (not hidden via Hide Actor).
- **Is Active** - If the actor is *active* (activated in the scene and receiving updates).
- **Is Pinned** - If the actor is currently *pinned to screen* (HUD-style, ignores camera scroll).
- **Is Flickering** - If the actor's *flicker effect* is currently counting down.
- **Is Flipped X** - If the actor's sprite is *horizontally flipped*.
- **Is Flipped Y** - If the actor's sprite is *vertically flipped*.
- **Is On Screen** - If the actor's position is *inside the visible viewport*.
- **Is Outside Screen** - If the actor's position is *outside the visible viewport* (opposite of *Is On Screen*).
- **Is Outside Scene** - If the actor's position is *outside the scene bounds* entirely.
- **Is Movement Enabled** - If the actor is allowed to *move* (its movement flag has not been disabled).
- **Is Immovable X** - If the actor is *immovable on the X axis* — collisions on its left or right will not push it.
- **Is Immovable Y** - If the actor is *immovable on the Y axis* — collisions on its top or bottom will not push it.
- **Is Immovable** - If the actor is *immovable on both axes* (X **and** Y).
- **Is Solid** - If the actor *blocks other actors* on collision (active AND collisions enabled).
- **Is On One Way** - If the actor is currently standing on a *one-way platform* tile.
- **Is Jumping** - If the actor has a *negative Y velocity* (moving upward).
- **Is Falling** - If the actor has a *positive Y velocity* (moving downward).

These properties can be selected in the Script Values area with the **Property** button.

This will create a selector like this:





Replicate Actor

You can replicate an actor, this will copy the actor across all created scenes in the game.

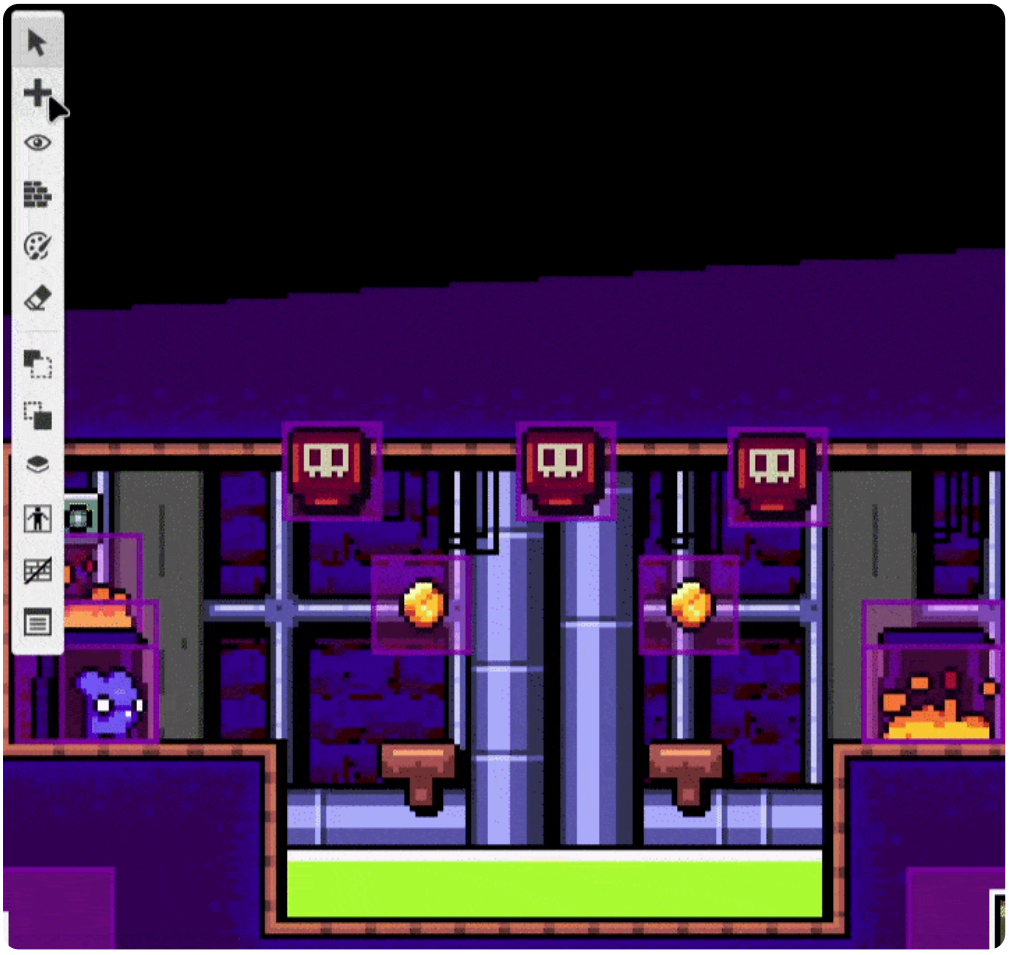
This has been created to make it easier to replicate UI across different levels.

Triggers

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

Adding a Trigger

To add a trigger to a scene click the  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 an actor walks on the trigger.

Trigger Properties

- **Name** - The name of your trigger. Giving your triggers a *name* helps organize them in your project.
- **Position** - Sets the X and Y position where the trigger will be positioned in a scene.
- **Size** - Sets the Width and Height of the trigger.
- **Collision Group** - Choose the collision group this trigger belongs to.

Trigger Scripts

Triggers can contain multiple scripts that will be called at different points in your game.

- **On Enter:** Called as soon as an actor enters into the trigger area.
- **On Update:** Repeatedly called every frame, while the overlap is still going on.
- **On Exit:** Called as soon as an actor exits the trigger area.

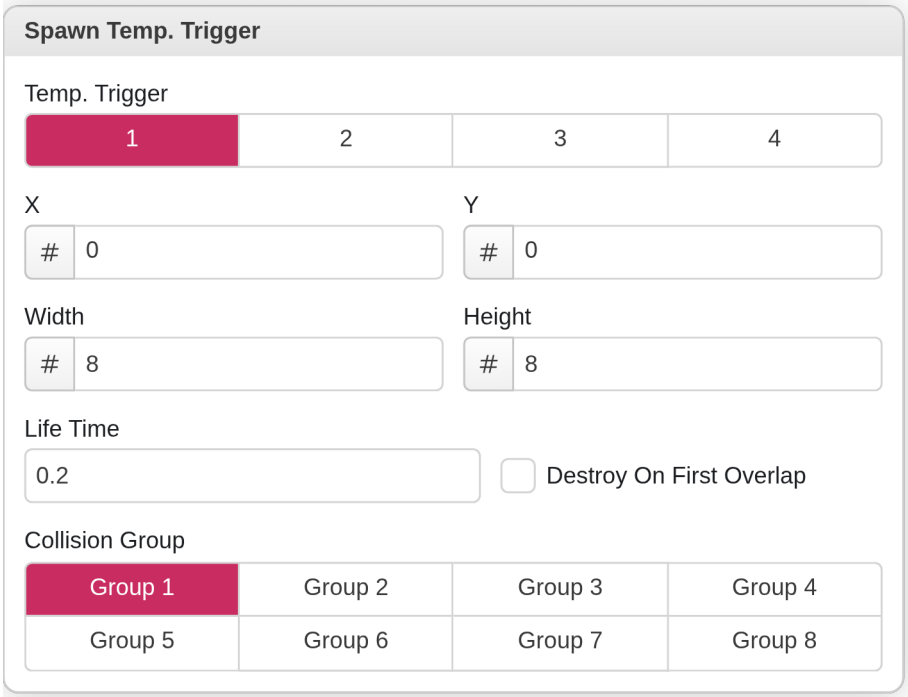
To start building a script, select a trigger, click the script type you want to edit and click the *Add Event button* in the *Editor Sidebar* to open the event menu. Select an event to add it to the script.

For more information see the documentation for Scripting.

Temporary Triggers

Temporary Triggers are a special type of trigger that exists only for a few frames in a scene. When an actor walks over them, the actor's **On Collide** script will be called. They are useful for creating temporary damage areas like attacks.

Temporary Triggers can be spawned with Spawn Temp. Trigger event.



Temporary Triggers Properties

- **Index** - The temporary trigger slot to use.
- **Position** - Sets the X and Y position where the trigger will be positioned in a scene.
- **Size** - Sets the Width and Height of the trigger.
- **Life Time** - The amount of time in frames or seconds that the temporary trigger will live for.
- **Destroy On First Overlap** - If the temporary trigger will be destroyed when the first overlap happens, it will ignore the lifetime if it's longer.
- **Collision Group** - Choose the collision group this trigger belongs to.

Modules

MD Engine has a number of optional modules that only get compiled into your ROM if your project uses them. This keeps unused features from consuming ROM space or RAM.

Below is an overview of each module and when you might use it.

Module	Description
Arrays	Store and manage lists of values — inventory, completed levels, high scores, etc.
Behaviours	Pre-built movement patterns: platformer physics and tile-based movement.
Camera	Follow actors, scroll, shake, flash, set deadzones and bounds.
Collisions & Physics	Collision detection, physics simulation, collision groups and masking.
Dialogue & Text	Text boxes, player choices, fonts, and text drawing.
Localization	Translate your game's text into multiple languages.
Math	Math functions (sine, cosine, atan2, sqrt) and unit conversions.
Particles	Lightweight visual effects — sparks, smoke, rain, explosions.
Pathfinding	Automatic route-finding between points on the collision map.
Projectiles	Spawnable attack objects — bullets, arrows — that interact with collisions.
Threads	Background scripts not attached to any scene, actor, or trigger.
Timers	Run scripts after a delay or at regular intervals.
Tweens	Smooth value interpolation with easing curves.

Many module limits (number of timers, threads, particles, etc.) can be adjusted in the Maximum Elements settings.

📄 Arrays

An array holds lists of values stored in a single place.

Behaviours

Platformer

Camera

The camera module controls what part of the scene is visible on screen. It can follow actors, scroll to specific pos…

Collisions & Physics

MD Engine provides a simple collision & physics system; it's not a fully featured physics engine like Box2D, but g…

Dialogue & Text

The dialogue system allows actors to display text boxes, choices, and conditional branching. It handles everythin…

Localization

The localization module allows you to translate your game's text into multiple languages and switch between the…

Math

The math module provides mathematical functions and unit conversions used by the engine and available to your…

Particles

Particles are small visual objects (like sparks, smoke, dust, explosions, rain, snow, etc.) that can be spawned to c…

Pathfinding

MD Engine includes an optional pathfinding module that automatically finds routes between two points on a scen…

Projectiles

Projectiles are small visual objects that overlap actors (like bullets, arrows, etc.) that can be spawned to create a…

Threads

The threads module allows you to run scripts that are not attached to any Scene, Actor, or Trigger. Threads are u…

Timers

The timer module allows you to set up a script that runs after a certain duration or repeatedly at regular intervals.

Tweens

The tween module handles the process of smoothly interpolating a value from one state to another over a specifi…

Arrays

An array holds **lists of values** stored in a single place.

An array lets you store and manage multiple pieces of related data — like player inventory, completed levels, or selected dialogue options, etc. All that without creating a separate variable for each piece of information.

What Is an Array?

You can think of an **Array** as a **row of boxes**, each holding one value.

Each position in the array is called an **index**.

Index	Value
0	10
1	20
2	30
3	40

Indexes start at **0**, so the **first element** is at position `0`.

Creating an Array

You can create an array with the Array Create event, it can be normal or global.

Global Arrays

Global arrays are not destroyed between scene changes, they exist on memory until they are released with Array Release event.

Debugging Arrays

You can view the created arrays in the Arrays Pane of the debugger window.

Saving / Loading Arrays

You can save / load arrays with the Game Data Process Array event.

See Also

- Array Event Glossary — Full reference for all array events
- Array Demo — Interactive example of array usage

Behaviours

Platformer

The platformer behaviour is useful for beginners and simple platformer games.

Enable Platformer Mode

The Enable Platformer Mode for an Actor event is a simplified call for the following events:

- Enable Movement for an Actor
- Enable Collisions for an Actor
- Set Actor Acceleration Y with the value of Gravity

During the actor pre-update if the behaviour is enabled the Set Actor Acceleration X event will be called with a value of 0.

Disable Platformer Mode

The Disable Platformer Mode for an Actor event is a simplified call for the following events:

- Disable Movement for an Actor
- Disable Collisions for an Actor
- Set Actor Acceleration Y with a value of 0

Simulate Control

The Actor Simulate Control Press event is a simplified call for the following events:

- If you Simulate to the **Left** it would be equivalent to Set Actor Acceleration X with the value of Acceleration * -1
- If you Simulate to the **Right** it would be equivalent to Set Actor Acceleration X with the value of Acceleration
- If you Simulate to the **Jump** it would be equivalent to Set Actor Velocity Y with the value of Jump

Tile-Movement

The Actor Simulate Tile-based Movement event is a simplified call for the following set of events:

- If Input Pressed Left
  - Set Actor Direction with Direction: *Left*
  - If the *Manage Animation* is enabled
    - Set Actor Animation with an *Animation Frame* of 0
    - Resume Actor Animation
  - Actor Move Relative with X: -*Tile* and Y: 0
  - Stop Script
- If Input Pressed Right
  - Set Actor Direction with Direction: *Right*
  - If the *Manage Animation* is enabled

        - Set Actor Animation with an *Animation Frame* of 0
        - Resume Actor Animation
    - Actor Move Relative with X: *Tile* and Y: 0
    - Stop Script
- If Input Pressed Up
    - Set Actor Direction with Direction: *Up*
    - If the *Manage Animation* is enabled
        - Set Actor Animation with an *Animation Frame* of 0
        - Resume Actor Animation
    - Actor Move Relative with X: 0 and Y: -*Tile*
    - Stop Script
- If Input Pressed Down
    - Set Actor Direction with Direction: *Down*
    - If the *Manage Animation* is enabled
        - Set Actor Animation with an *Animation Frame* of 0
        - Resume Actor Animation
    - Actor Move Relative with X: 0 and Y: *Tile*
    - Stop Script

If the *Auto-Interaction* is enabled

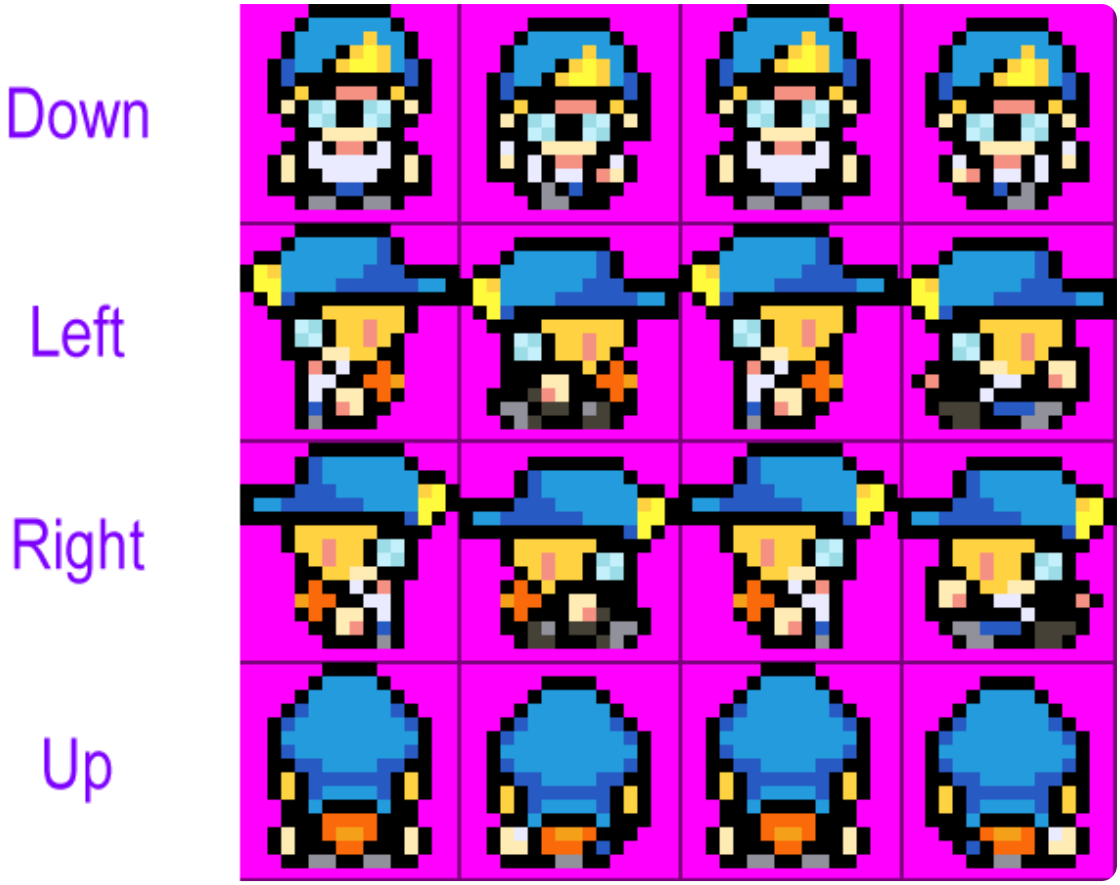
- If Input Just Pressed *Interact Button*
    - Actor Interact

If the *Manage Animation* is enabled

- Pause Actor Animation
- Set Actor Animation Frame with an *Animation Frame* of 0

Animation PNG Setup

To work with the managed animation, you need to adjust the PNG like the attached PNG.



See Also

- Collisions & Physics — Collision system configuration and modes
- Engine Fields — Physics values (gravity, acceleration, jump) used by behaviours

Camera

The **camera** module controls what part of the scene is visible on screen. It can follow actors, scroll to specific positions, shake, flash, and be configured with deadzones and bounds.

Following an Actor

The most common use of the camera is to follow an actor (typically the player). Use the Camera Follow Actor event to start following, and Camera Stop Follow to detach.

Follow Styles

The camera can follow an actor with different *styles*. The style adjusts the predefined deadzone setup — you can use Camera Set Deadzone if you don't like any of the presets.

- **Simple** — The camera is always centered on the actor.
- **Platformer** — The camera has small deadzones on the sides and a bigger one on the top area.
- **Top Down** — The camera has a big deadzone on all sides.
- **Screen By Screen** — The camera deadzone is the same as the screen, so the camera only moves when the actor reaches the edge.

The best way to understand this is to check the Camera Follow example.

> **TIP**
> The camera Deadzone is the area where the actor can move without causing the camera to scroll.

Shake and Flash

The camera can produce a **shake** effect to simulate impacts, explosions, or earthquakes using the Camera Shake event. You can stop it early with Camera Stop Shake.

The Camera Flash event produces a full-screen flash of a chosen color, useful for lightning, damage hits, or transitions.

Scrolling and Position

You can move the camera to an exact position with Camera Set Position (instant) or Camera Scroll To (smooth scroll over time).

Use Camera Set Lerp To to control how smoothly the camera catches up when following an actor. A lower lerp value makes the camera lag behind slightly, creating a more cinematic feel.

Bounds and Offset

By default the camera can scroll across the entire scene. You can restrict its range with Camera Set Bounds to prevent the player from seeing areas outside the playable space, or use Camera Copy Bounds to match another reference.

The Camera Set Offset To event shifts the camera's center point relative to the followed actor. This is useful for look-ahead effects or cut-scene framing.

You can also restrict which axes the camera is allowed to scroll on with Camera Set Allowed Scroll.

Tips

- For platformer games, use the **Platformer** follow style and adjust the deadzone to taste.
- Combine camera shake with a brief pause (using Wait) to add impact to attacks or explosions.
- When making a cutscene, stop following the actor, scroll to the target, and resume following when done.

See Also

- Camera Event Glossary — Full reference for all camera events
- Camera Follow Demo — Interactive example of follow styles
- HBlank & VBlank — Advanced scroll effects using blank interrupts

Collisions & Physics

MD Engine provides a simple collision & physics system; it's not a fully featured physics engine like Box2D, but good enough for arcade-style platformers, shooters, puzzles, etc.

There are several physics values, local to each actor:

- Position
- Velocity X / Y
- Acceleration X / Y

Also, some are global to the entire game:

- Maximum Velocity X / Y
- Deceleration X / Y
- Acceleration
- Gravity
- Jump

You can adjust the global ones on the Engine Fields settings.

The global acceleration, gravity, and jump values are used automatically for the Platformer Behaviour.

Collision System

There are three different collision systems available to choose, *Simple*, *Advanced (Fast)*, and *Advanced (Slow)*.

The major difference is that on the Advanced system, actors can push each other while on the simple they overlap but can't physically move one another.

Actor Overlap Behaviour

When two actors overlap each other you can adjust how the **On Collide** scripts are invoked, it can be changed on the Collisions settings.

- **First Time** - This option causes **On Collide** to be called only the first time it happens, and it won't be called again until they stop overlapping.
- **Continuous** - This option causes **On Collide** to be called every frame from when the overlap begins until there is no overlap.

> **WARNING**
>
> Be aware *First Time* option may behave the same as *Continuous* option if both actors are solid and you are using the Advanced system.
> Due to actors never fully overlapping, they separate and adjust their positions.

Collision Mode

This setting lets you choose different ways of calculating collisions.
You can choose between *All Actors*, *Only On-Screen Actors*, and *Gridding*.

All Actors

This is the default option, it will work fine for most games that have a reduced amount of actors and small levels.

Only On-Screen Actors

This is the best option if you have very large levels and a lot of actors spread, this way only actors that are on-screen are checked for collisions.

Gridding

This is the best option for shooters that feature a lot of bullets and actors, you can adjust the grid sizes, as well as how the objects are gridded to get the best possible performance.

Collision Special Options

Collision Tiles

Collision Tiles refers to the export size of the scene collisions information, you can set either 8x8 pixels or 16x16 pixels. This option helps reduce the amount of RAM used on collision information as well as the ROM size if all your collisions are placed on 16x16px squares.
During build time if you have any square with collision information that is not correct a warning will be printed.

Collision Bias

Collision bias refers to a small numerical adjustment (or tolerance) used to prevent actors from "sticking," jittering, or tunneling through each other.

Collision bias controls the simulation step size for physics resolution. It determines how far actors are pushed apart when overlapping and how aggressively the engine resolves penetrations.

- **Lower values** produce more precise collision detection and smoother separation, but require more processing time per frame. Use lower values when you need precise platforming or tight collision boundaries.
- **Higher values** are faster to compute but may cause the engine to miss thin collisions (tunneling), especially with fast-moving actors or projectiles.
- The **default value** works well for most games. You should only change it if you notice specific issues.

Practical advice:

- **Increase** the bias if you have fast-moving projectiles that need to detect thin walls, or if actors are jittering against surfaces.
- **Decrease** the bias for precise platforming where pixel-perfect landings matter.
- If actors get stuck in walls or floors, try adjusting the bias in small increments.

Collision Masking

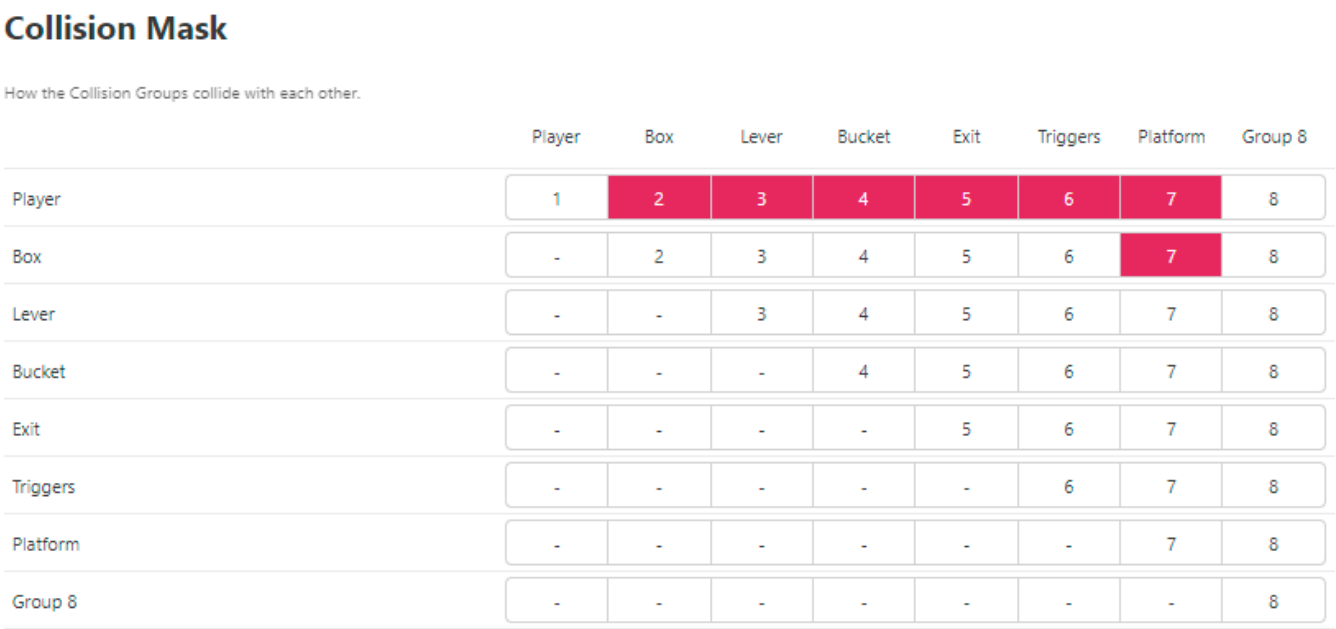
Collision masking is a technique used in physics simulations to control which actors can collide or interact with each other.

Instead of every actor checking collisions against every other actor (which can be inefficient or undesirable), collision masking lets you define groups that specify which collisions should be considered.

You can assign each actor to a specific group in the actor settings.



After that, you can select if each collision group overlaps another by just clicking the box.

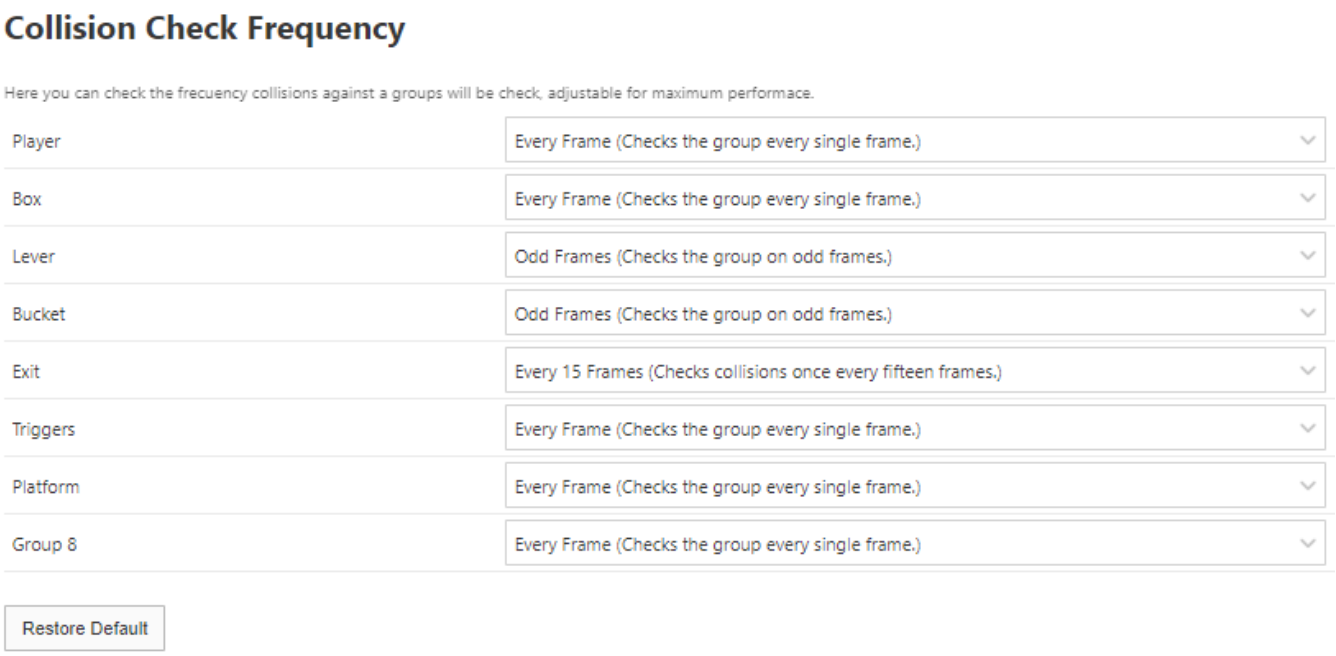


Collision Frequency

You can adjust the frequency each of the collision groups will be checked.
The options change depending on the collision mode.

All Actors

The *All Actors* mode allows you to check how each group is checked.
This helps increase the performance without having to change to a different collision mode.



Only On-Screen Actors & Gridding

These two modes allow you to check how each projectile group is checked. This helps improve the performance of shooters a bit more when using the gridding system.

See Also

- Behaviours — Platformer and tile-movement behaviours that use collision detection
- Engine Fields — Physics values (gravity, acceleration, jump) used by the collision system

Dialogue & Text

The dialogue system allows actors to display text boxes, choices, and conditional branching. It handles everything from simple one-line messages to multi-page conversations with player choices.

Displaying Text

Use the Show Text event to display a dialogue box with text. The text box appears at the bottom of the screen and waits for the player to press a button before continuing.

You can include variable values in your text using the `$XX$` syntax, where `XX` is the variable number. Line breaks are represented with `\n`.

Use Clear Text to remove the current text, and Close Dialogue to dismiss the dialogue box entirely. If you need the dialogue to remain open between events, use Force Dialogue to Stay Open.

Choices

The Show Choices event presents the player with selectable options. The result is stored in a variable that you can check with control flow events to branch your script.

Drawing Text Manually

For more control over text placement, use the Draw Text event. This lets you place text at a specific position on the screen without using the standard dialogue box. You can combine it with Draw Frame and Draw Cursor for custom menu layouts.

UI

The dialogue system uses two UI pieces — a **frame** and a **cursor**. You can change them in the Dialogue & Text settings window.

There are more details about the UI elements in the UI Tilesets section.

You can also swap the frame and cursor at runtime:

- Set Dialogue Frame — Change the dialogue frame tileset
- Set Dialogue Cursor — Change the cursor tileset

Fonts

Use Load Font to load a font into memory and Set Font to switch the active font. This lets you use multiple fonts for different characters or emphasis.

Colors

The default font, as well as fonts generated with the font converter for the Advanced Mode, use the last color (color 15) of the selected Text palette as the text color.

The default cursor and frame use colors 13 and 14 of the selected palette. You can have different palettes for the dialogue frame and the text:

- Set Dialogue Frame Palette — Change the frame palette
- Set Text Palette — Change the text palette





Tips

- Keep dialogue lines short — the Mega Drive screen is only 40 characters wide in H40 mode.
- Use Actor As Cursor to replace the default text cursor with an animated sprite for a more polished look.
- Set text priority with Set Text Priority to control whether text appears above or below sprites.

See Also

- Dialogue & Text Event Glossary — Full reference for all dialogue events
- Text Demo — Interactive example of dialogue features
- Localization — Translating your game's text into multiple languages
- Fonts — Font assets and the font converter

Localization

The localization module allows you to translate your game's text into multiple languages and switch between them at runtime.

How It Works

Localization works by replacing all in-game text strings with their translated equivalents. The engine does this automatically when you switch locales — every dialogue, choice, and drawn text event will display the translated version.

The workflow is:

1. Write your game's text normally using dialogue and text events.
2. Open the Localization asset view to see all the text in your project collected in one place.
3. Create a new localization file and translate each string in a text editor.
4. Use the Set Text Localization event in your game to switch to the translated text.

Switching Languages

Use the Set Text Localization event to change the active localization at any point during gameplay. A common pattern is to offer a language selection screen at the start of the game using Show Choices, then calling Set Text Localization with the selected language.

Once set, all subsequent text events will display the localized version until the localization is changed again or the game is restarted.

Tips

- If you are using the **Advanced** text mode, make sure your font includes all the characters needed for each language. Check the **Unique Characters** list in the localization asset view.
- Variable replacements (`$XX$`) and line breaks (`\n`) work the same way in all languages.
- You can create as many localization files as you need — one per supported language.

See Also

- Localization Asset Type — Detailed guide on creating and editing localization files
- Dialogue & Text — The dialogue system that localization applies to
- Fonts — Font assets, including support for non-Latin characters

Math

The math module provides mathematical functions and unit conversions used by the engine and available to your scripts.

Unit Conversion

The engine uses an internal fixed-point number format (FF32) for physics calculations. The math module automatically converts values you specify in **pixels per frame** or **pixels per frame squared** into this internal format, so you don't need to worry about the conversion yourself.

This means you can set velocity and acceleration values in intuitive units and the engine handles the rest.

Math Functions

The module provides implementations of common mathematical functions:

- **Sine and Cosine** — For circular motion, wave patterns, and angle-based calculations.
- **Atan2** — For calculating the angle between two points, useful for aiming or rotating toward a target.
- **Integer Square Root** — For distance calculations and other operations that need a square root.

These operations are based on **lookup tables** and favor speed over accuracy, which is appropriate for a game running on the Mega Drive hardware.

Scripting Events

The math module's scripting events let you evaluate expressions and use math in control flow:

- Evaluate Math Expression — Compute a math expression and store the result in a variable.
- If Math Expression — Branch based on a math condition.
- Loop While Math Expression — Repeat a block while a math condition is true.
- Math Functions — Access built-in math functions (sine, cosine, atan2, square root, etc.).
- Seed Random Number Generator — Set the random seed for reproducible results.

Tips

- Use sine and cosine to create circular or wave-based movement patterns for enemies or projectiles.
- Atan2 is useful for making an actor face toward the player or a target position.
- If you need reproducible randomness (e.g., for replays or seeded levels), use the seed event at the start of the scene.

See Also

- Math Event Glossary — Full reference for all math events
- Engine Fields — Physics values that use the math module's unit conversion
- Collisions & Physics — The physics system that uses these math operations

Particles

Particles are small visual objects (like sparks, smoke, dust, explosions, rain, snow, etc.) that can be spawned to create visual effects without using full actors.

They are managed in Particle Types. You have 5 particle type slots where you can set up your desired particle at any time, then spawn the particles when you need them.

You can check the Particles example for an in-depth usage example.

Related Events

- Preload Particle
- Emit Particle
- Destroy All Particles
- Particle Type Send in Front Actor
- Particle Type Send Behind Actor
- Particle Type Set Depth

See Also

- Projectiles — Similar spawnable objects that interact with actor collisions

Pathfinding

MD Engine includes an optional pathfinding module that automatically finds routes between two points on a scene's collision map. It supports multiple search algorithms with different movement styles and tradeoffs.

The module only gets compiled into your ROM if your project uses any pathfinding events.

Algorithms

You can choose the pathfinding algorithm in the Settings panel under **Pathfinding**.

Best-First Search

The default algorithm. Searches in 8 directions (including diagonals) and finds paths very quickly. Paths are usually good but may not always be the absolute shortest route, especially around complex obstacles. Best suited for most games.

A* (A-Star)

Searches in 8 directions and always finds the optimal path. Slightly slower than Best-First Search but guarantees the shortest route. Best suited for games where path quality matters more than speed.



How It Works

1. **Find Path** computes a route from a start position to a goal position, avoiding collision tiles.
2. The result is stored in a **path slot** (referenced by a variable). The path is a sequence of direction steps.
3. **Follow Path** makes an actor walk the path step by step at a given speed.
4. When the actor finishes walking, the path slot can be reused.

Path Slots

Paths are stored in a fixed-size pool. The number of simultaneous path slots can be configured in Settings (default: 4, range: 2-8). Each slot holds up to 64 direction steps.

A path slot is **allocated** when you use *Find Path* and must be **freed** when you no longer need it. If all slots are in use, *Find Path* will fail.

Each slot also records the status of its search, which *If Path Status* checks: **searching** while the search is queued or still running, **found** once the goal is reached, and **failed** when there is no path, when the start or goal is outside the search area, when no slot was free, or when the search was dropped. Searches share one search area, so they run one at a time: each *Find Path* joins a queue and starts when the searches before it have finished, so several actors can search at once. A scene change fails every queued search, and *Free Path* removes the slot from the queue. *Follow Path* on a slot that is still searching waits for the search to finish.

> **WARNING**
>
> Always free path slots when you're done with them. If you forget, the pool will fill up and no new paths can be computed.

Scripting Events

For a full reference of all pathfinding events with field descriptions and previews, see the Pathfinding Event Glossary.

Settings

The pathfinding settings are found in the **Pathfinding** section of the Settings panel.

- **Max Paths** - The maximum number of simultaneous path slots (default: 4, range: 2-8).
- **Budget** - How many search nodes the pathfinder processes per frame, in both modes (default: 64). Higher values find paths in fewer frames but take more of each frame. Set to 0 for unlimited: the search then finishes in the frame it starts, however long that frame becomes.
- **Algorithm** - Choose between *Best-First Search* (fast) and *A** (optimal).
- **Queue Size** - Controls how large an area the pathfinder can search. The default (*Small*) works for most games. Increase if pathfinding fails on very large open maps.

Tips

- Use the smallest actor footprint that makes sense for your game. Smaller actors are significantly faster to pathfind.

- With *Block Until Found* the script waits on the *Find Path* event while the engine searches with the selected **Algorithm**, a **Budget** of nodes per frame, so the game keeps running. Without it the script continues at once, the search runs as a 4-direction flood (the algorithm setting is not used), and *If Path Status* tells when it is done.
- Lower the **Budget** if searches slow the game down; raise it if paths take too many frames.
- Best-First Search is the fastest algorithm overall but may occasionally take a longer route around complex obstacles. If path quality matters, use A*.
- Make sure your collision map accurately represents walkable areas.

See Also

- Collisions - Collision system and collision tile settings
- Behaviours - Movement behaviours for actors

Projectiles

Projectiles are small visual objects that overlap actors (like bullets, arrows, etc.) that can be spawned to create an attack without using full actors.

They are managed in Projectile Types, you have 5 projectile type slots where you can setup your desired projectile at any time, then spawn the projectile when you need them.

You can check the Projectiles example for an in-depth usage example.

Related Events

- Preload Projectile
- Destroy All Projectiles
- Projectile Type Send in Front Actor
- Projectile Type Send Behind Actor
- Projectile Type Set Depth

See Also

- Particles — Similar spawnable objects for visual effects
- Collisions & Physics — Collision system that projectiles interact with

Threads

The threads module allows you to run scripts that are not attached to any *Scene*, *Actor*, or *Trigger*. Threads are useful for background processes or situations where no specific actor or trigger is appropriate.

How Threads Work

A thread is a standalone script that runs alongside the rest of your game while the current scene is active. It is not tied to an actor or trigger. Running threads are cleared when the scene changes.

When a thread's script finishes executing, the thread stops. If you want a thread that runs continuously, place its logic inside a Loop event.

> **WARNING**
>
> If you use a loop inside a thread, you **must** include an Idle or Wait event inside the loop body. Without it, the thread will consume all available processing time and the game will soft-lock.

Thread Slots

By default, you can create up to **5** threads. This can be increased in the Maximum Elements settings.

Each thread slot is independent — you can have multiple threads running simultaneously, each with its own script.

When to Use Threads

Threads are best suited for:

- **Background music control** — Changing music based on game state without tying the logic to a specific actor.
- **Counters or timers** — Tracking time or score across actor interactions in the current scene.
- **Monitoring conditions** — Continuously checking if a condition is met (e.g., "if all enemies defeated, open the door").
- **Cutscene orchestration** — Coordinating multiple actors from a single script that doesn't belong to any one of them.

Threads vs Timers

Both threads and timers run scripts independently, but they serve different purposes:

- **Threads** run a script once (or continuously with a loop) and are good for ongoing background logic.
- **Timers** fire a script at regular intervals and are good for periodic actions on a fixed schedule.

Tips

- Always include an Idle or Wait in looping threads to avoid freezing the game.

- Threads are a good way to handle logic that doesn't belong to any particular actor.
- Running threads stop when the scene changes. Start a new thread in the next scene if that logic needs to continue.

See Also

- Timers — Scripts that run after a duration or at intervals
- Tweens — Smooth value interpolation over time

Timers

The timer module allows you to set up a script that runs after a certain duration or repeatedly at regular intervals.

This is useful for controlling timed events, delayed actions, periodic animations, or any game logic that needs to happen on a schedule rather than every frame.

Setting Up a Timer

Use the Attach Timer Script event to create a timer. You specify:

- **Timer slot** — Which timer to use (selected by number).
- **Duration** — How long to wait before the script runs, in seconds.
- **Script** — The events to execute when the timer fires.

The timer fires repeatedly at the specified interval until you remove it or change scenes.

Timer Slots

By default, there are **4** timer slots available. You can increase this in the Maximum Elements settings.

Each slot is independent — you can have multiple timers running simultaneously with different durations and scripts.

Managing Timers

- Remove Timer Script — Stops a timer and clears its script.
- Restart Timer — Resets a timer's countdown back to the beginning.

Waiting

The timer module also includes two useful waiting events:

- Wait — Pauses the current script for a specified duration (in seconds).
- Idle — Pauses the current script for one frame, allowing other scripts and the game loop to continue.

> **WARNING**
>
> The timer script will be invoked recurrently even if the previous execution did not finish. If your timer script uses async events (like Actor Move To or Wait), multiple copies of the script may run in parallel. Keep timer scripts short and avoid async operations when possible.

Tips

- Use timers for spawning enemies at intervals, blinking UI elements, or triggering periodic environmental effects.

- If you need a one-shot delay (fire once, not repeatedly), attach the timer and remove it inside the timer script itself after it fires.
- Timers are cleared when the scene changes. Attach a new timer script in the next scene if the interval should continue there.

See Also

- Timer Event Glossary — Full reference for all timer events
- Threads — Background scripts without an actor or trigger
- Tweens — Smooth value interpolation over time

Tweens

The tween module handles the process of smoothly interpolating a value from one state to another over a specified duration.

For example, moving a sprite from x = 0 to x = 100 over 1 second, or fading a palette color from bright to dark.

Using the tween module for value interpolation allows for smoother or fancier results than manual linear calculations. The module handles timing, easing curves, and callbacks automatically.

How It Works

Use the Start Tween event to begin a tween. You specify the start value, end value, duration, and easing function. The tween runs in the background and updates the target value each frame until the duration is reached.

You can run multiple tweens simultaneously for different values — for example, tweening an actor's X and Y positions at the same time to create curved movement.

Managing Tweens

- Pause Tween / Resume Tween — Pause and resume a specific tween.
- Pause All Tweens / Resume All Tweens — Pause and resume every active tween.
- Stop Tween / Stop All Tweens — Cancel a tween before it finishes.
- If Tween Running — Check whether a tween is still active.

Easing Functions

Easing functions control the rate of change during the tween. A **linear** tween moves at constant speed, while easing functions create acceleration, deceleration, or bouncing effects.

- **In** — Starts slow and accelerates.
- **Out** — Starts fast and decelerates.
- **InOut** — Starts slow, speeds up in the middle, and slows down at the end.

The available easing functions are:

- linear
- smoothstep
- smootherstep
- quadIn
- quadOut
- quadInOut
- cubeIn
- cubeOut
- cubeInOut
- quartIn
- quartOut
- quartInOut
- quintIn
- quintOut
- quintInOut
- sineIn
- sineOut
- sineInOut
- bounceIn
- bounceOut
- bounceInOut
- circIn
- circOut
- circInOut

- expoIn
- expoOut
- expoInOut
- backIn
- backOut
- backInOut
- elasticIn
- elasticOut
- elasticInOut

The best way to understand these is to check the Tweens example.

Tips

- Use **bounceOut** for items landing or dropping into place.
- Use **backOut** for UI elements that overshoot slightly before settling — it adds a polished feel.
- Use **sineInOut** for gentle, natural-feeling movement.
- Pause all tweens when the game is paused to prevent animations from continuing during menus.

See Also

- Tween Event Glossary — Full reference for all tween events
- Tweens Demo — Interactive example of easing functions
- Timers — Timed and periodic script execution
- Threads — Background scripts for ongoing logic

Engine Fields

The *Engine Fields* are a set of global values that control the physics simulation for your game. They can be changed in the Settings panel and also modified at runtime using the Engine Fields events.

These fields are primarily used by the Platformer Behaviour and by the physics system in general. They can also be used manually if you are building your own movement logic.

> **INFO**
>
> These values are **global** — they apply to every actor in the game. You cannot have different gravity values for different actors using engine fields alone. For per-actor physics, set velocity and acceleration directly on each actor.

Physics Fields

Max Velocity X

The maximum horizontal speed an actor can reach, in **pixels per second**. Typical values: 60–180 for platformers, 30–80 for top-down games.

Max Velocity Y

The maximum vertical speed an actor can reach, in **pixels per second**. Typical values: 120–300. Set this high enough that falling feels responsive.

Acceleration

The horizontal acceleration applied when simulating left/right platformer movement, in **pixels per second squared**. Typical values: 200–600. Higher values make the character feel more responsive; lower values feel more floaty.

Deceleration X

The horizontal deceleration applied when acceleration is **not** active (i.e., the player releases the direction buttons), in **pixels per second squared**. Typical values: 200–800. Higher values mean the character stops more quickly.

Deceleration Y

The vertical deceleration applied when acceleration is **not** active, in **pixels per second squared**. This is rarely used in platformers (gravity handles vertical motion) but can be useful in top-down games.

Gravity

The global gravity value applied to actors with the platformer behaviour enabled, in **pixels per second squared**. Typical values: 400–1000. Higher gravity makes jumps shorter and snappier.

Jump

The initial upward velocity when a jump occurs, in **pixels per second**. Typical values: 150–400. This works together with gravity — higher jump with higher gravity feels fast and punchy, while lower values of both feel floaty.

Tips

- Start with the default values and adjust in small increments while testing in the Play Window.
- The feel of your game is determined by the **relationship** between these values, not any single one. Gravity vs Jump determines jump arc; Acceleration vs Deceleration determines how "slippery" movement feels.
- You can change engine fields at runtime using scripting events — for example, reducing gravity during an underwater section or increasing jump strength with a power-up.

See Also

- Engine Fields Event Glossary — Scripting events to read and modify engine fields at runtime
- Collisions & Physics — The collision and physics system
- Behaviours — Platformer and tile-movement behaviours that use these fields
- Settings — Where to configure engine fields in the editor

Save System

MD Engine supports saving on both the Flash memory and the SRAM memory, you can change the way of saving on the Settings.

For you there won't be any difference between using Flash or SRAM saving, as MD Engine manages that internally.

You only need to select what you want in the settings menu. Nonetheless, we have outlined a few important things about each system so you can make an informed choice.

Saving Procedure

The saving procedure consists of either Game Data Save or Game Data Load events.

Each of these events takes a process script.

Process Script

The process script has the responsibility of processing the variables and arrays to the save system.

- **Game Data Process** - Saves or loads the selected variable into the save system.
- **Game Data Process Array** - Saves or loads the selected array into the save system.

> **WARNING**
>
> If you need to patch the game, be aware removing variables from the process script is **very risky**.
>
> Adding a new variable is less risky, but make sure to test properly.

Saving Types

There are two types of saving systems: Flash Saving and SRAM Saving.

Flash Saving

Flash saving relies on modifying the end part of the flash chip where the game is stored.

MD Engine uses the SGDK flash saving module and save manager which supports the following features:

- **Wear leveling** - The module intelligently appends data in an efficient way to avoid flash chip wear as much as possible. The mechanism is specially efficient if you use small slots with big flash sectors.
- **Safe save** - The module avoids data loss even in the case of a power cut while data is being saved. In this case, the incomplete save condition is detected on initialization, and data is corrected, so you can keep the last data set properly recorded before the incomplete save was initiated.

You can back up and restore flash save data directly from hardware using the Flash ROM to Cart tool.

Flash Saving restrictions

Saving directly to the flash has one clear advantage: you don't need extra parts on your PCB like SRAM, a battery, or an FRAM chip. Your data will also last longer since there’s no battery to run out.

On the other hand, there are a few limitations you should be aware of:

- **Smaller ROM space** - The save data area takes space from your game. It's something to plan for if your game is going to be large.
- **No Music or Sounds while saving** - Most save functions stop the audio. This means you can't play music or sound during saving.

These are normally not a big deal, but sometimes they can be serious enough that you might prefer using an SRAM saving instead.

> **WARNING**
>
> **If you use our desktop or web exports, this won't affect you.** That's why this is not on the restrictions list, but it's still worth mentioning.
>
> There is **no emulator support** currently for flash saving!
>
> So if you were planning on using any other emulator besides our desktop or web export, be aware of this!

SRAM Saving

You can read and write SRAM save data directly from hardware using the Flash ROM to Cart tool. It exports emulator-compatible `.srm` files and can import them back.

SRAM saving falls back to the use of the classic static RAM (SRAM) chip + battery module or ferroelectric RAM (FRAM) chip, which does not need a battery but is more expensive.

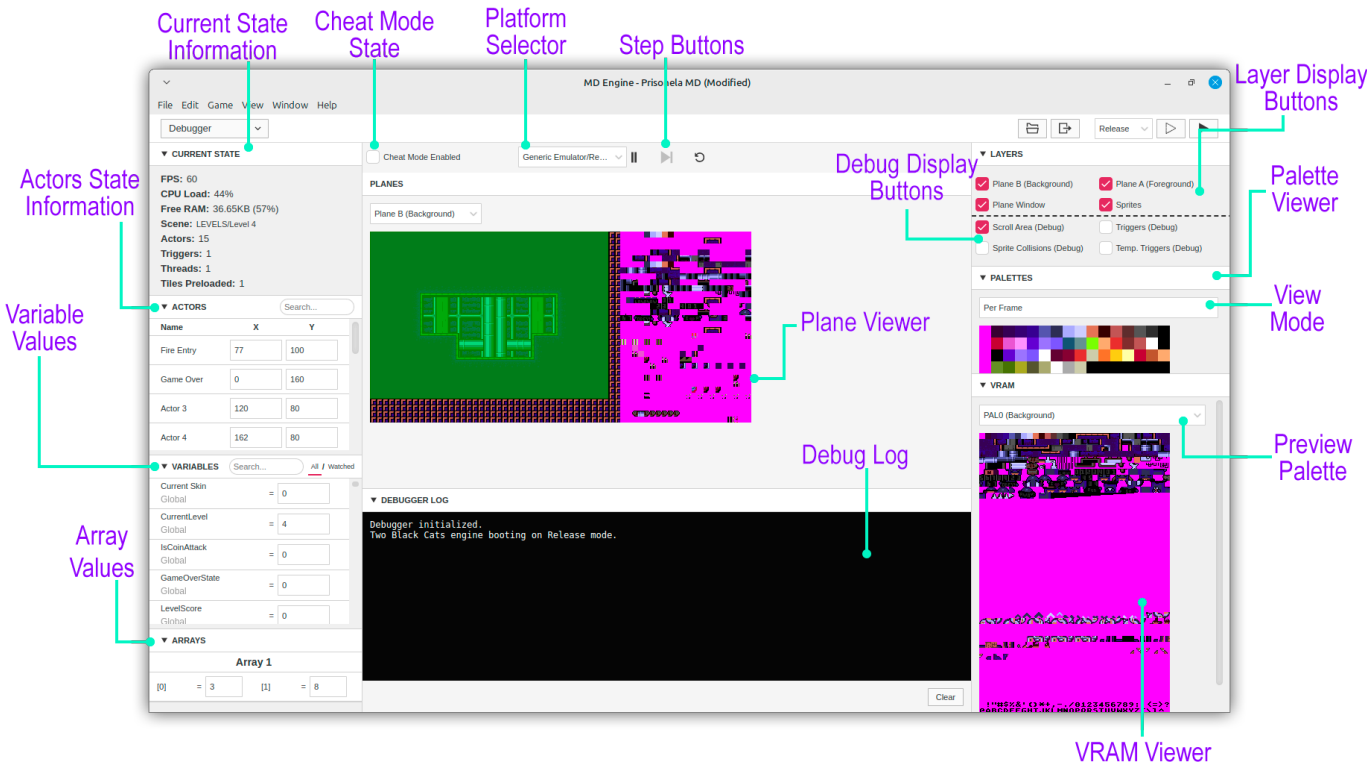
You need to be aware of this if you are planning to produce your own games and ship them to your players as SRAM normally relies on CR2032 lithium batteries, which may have complicated shipping procedures.

Beyond the shipping concerns, cartridge costs may be higher than when using regular flash saving, but you won't be limited to the previous restrictions.

Debugger

The **Debugger** is a built-in tool that lets you inspect and control your game while it runs. You can view actor positions, check variable values, step through scripts one instruction at a time, and profile CPU performance — all without leaving the editor.

To open the Debugger, use the menu or run your game in debug mode. The debugger connects to the running game and updates in real-time.



Current State

The **Current State** pane provides information about the game.

- **FPS** - Displays the current Frames Per Second when running.
- **CPU Load** - The current CPU load of the game.
- **Free RAM** - The current amount of free RAM of the game.
- **Scene** - The current scene running in the game.
- **Actors** - The current number of actors loaded on the game.
- **Triggers** - The current number of triggers loaded on the game.
- **Threads** - The current number of threads running on the game.
- **Tiles Preloaded** - The current number of tiles preloaded by Sprites.

Actors

The **Actors** pane provides information about the current actors in the scene and their X and Y positions. You can also type in new coordinates which will be **live updated** in your running game.

Variables

The **Variables** pane allows you to see the current value of the variables used within your game. You can also type in new values which will be **live updated** in your running game.

Arrays

The **Arrays** pane provides information about the current arrays in the scene and their values. You can also type in new values which will be **live updated** in your running game.

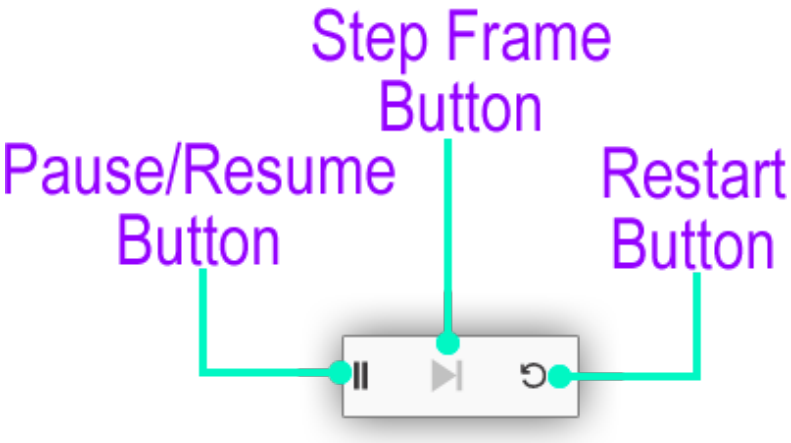
Cheat Mode & Platform

You can enable the *Cheat Mode* by ticking the *Cheat Mode Enabled* box, after that the If Cheat Mode is Enabled event would become True and run the code normally.

You can select the *Platform* you want on the selector, after that the If Running on Platform event would match the selected platform.

Step Buttons

When your game is paused you can use the *Step Buttons* or keyboard shortcuts to slowly step through your game to follow the progress of any running scripts.



- **Pause/Resume** - Toggles between playing and paused modes.
- **Step Forward One Instruction** - Run the game until the next instruction. This is useful for stepping through event scripts one command at a time.
- **Step Forward One Frame** - Run the game until the next frame of animation starts. This is useful for understanding what is happening every frame during events that take time such as Actor Move To.
- **Restart** - Restarts the game.

Keyboard Shortcuts

When the debugger is enabled, you can use the following keyboard shortcuts in both the *Project Window* and *Play Window*:

Pause/Resume - `F8` **Step Forward One Instruction** - `F9` **Step Forward One Frame** - `F10`

CPU Profiler

The **CPU Profiler** tab provides a detailed breakdown of how the Mega Drive's M68K CPU time is spent each frame. It helps you identify performance bottlenecks in your game.

The profiler is split into two columns: **Overall Timing** on the left and **Per-Script Timing (Update Loop)** on the right.

Overall Timing (Left Column)

Shows how many **scanlines** each engine section consumes per frame. The Mega Drive has a budget of **262 scanlines** (NTSC) or **312 scanlines** (PAL) per frame.

Main Sections

Section	Description
INP	Input processing
PRE	Pre-update logic
TMR	Timer callbacks
TWN	Tween animations
TAN	Tile animations
ULP	Update Loop — contains all script execution, collision, motion, etc.
CAM	Camera updates
SCL	Scroll updates
SYN	Actor sync (position/animation to sprites)
SPR	Sprite update (DMA preparation)
AUD	Audio driver update
VBL	VBlank Wait — idle time waiting for vertical blank (not real processing)
DBG	Debugger Update — debug-only overhead (CPU load, FPS, RAM queries)

Update Loop Sub-Sections

When the Update Loop (ULP) section is expanded, you can see its internal breakdown:

Section	Description
SCN	Scene script update
ACT	Actor script updates
ASY	Async script execution
MOT	Motion / physics
GRP	Grid preparation
COL	Collision detection
CHT	Collision hit callbacks
TRG	Trigger checks
PRJ	Projectile updates
TMP	Temporary triggers
PST	Post-update logic

Column Values

Each section row displays:

- **Lines** — Current scanline count for this section
- **Min** — Minimum scanlines observed across recent frames
- **Avg** — Average scanlines across recent frames
- **Peak** — Highest scanline count observed (resets on scene change)
- **%** — Percentage of total frame budget
- **+/-** — Delta from frozen snapshot (only visible when Freeze + Deltas are enabled)
- **Budget icon** (circle) — Click to set a max scanline budget; the row highlights red when exceeded and a budget line appears on the sparkline
- **Sparkline** — Visual history of recent frame values
- **Bar** — Visual bar showing relative cost

Total Row

The total row shows:

- **Scanline total** — Sum of all processing sections (excludes VBlank Wait and Debugger Update)

- **%** — Percentage of frame budget used by actual processing
- **Avg. CPU** — Average CPU load reported by the engine over the last 8 frames
- **p95** — 95th percentile: 95% of recent frames were at or below this value (shows your worst typical frame, filtering out rare spikes)
- **max** — Highest CPU load observed across recent frames
- **Spike** — Warning badge that appears when the current frame exceeds the average by more than 2 standard deviations

Other Row

Shows unaccounted time — the gap between the sum of all sections and the total frame budget. A small value (0–2 scanlines) is normal.

Per-Script Timing (Right Column)

Shows individual script execution times within the Update Loop. Scripts are split into **Sync Scripts** and **Async Scripts** sections.

Each script row displays:

- **Name** — Entity name and script key (e.g. `Player.onUpdate`, `Elevator.tween`)
- **Lines** — Current scanline count
- **Min / Avg / Peak** — Same as section columns
- **%** — Percentage of total frame budget
- **Cost/Call** — Average scanlines per call (only meaningful when a script is called multiple times per frame)
- **Call count** — Shown as a multiplier badge (e.g. ×3) when a script runs more than once per frame

Stack Depth

When scripts call other scripts (e.g. via custom events), the **Depth** indicator next to the header shows the maximum call-stack depth reached. A warning color appears at depth 5 or higher.

Parent Chain

When call-chain tracking is available, hovering over a script row shows its caller hierarchy in the tooltip (e.g. `← scene.onUpdate ← MainLoop`).

Toolbar Controls

Control	Description
Sort by cost	Sort sections by scanline cost (highest first)
Freeze	Freeze the display on the current frame's data. Press `F` as a keyboard shortcut
Normalize	Scale all sparklines to the total frame budget instead of auto-scaling
Timeline	Show a flame-style timeline of recent frames. Click a bar to freeze on that frame's data
Deltas	Show the difference between frozen and live values (requires Freeze)
Show All / Hide 0	Toggle visibility of sections with zero values
Copy	Copy current profiler data as JSON to clipboard
CSV	Copy frame history as CSV to clipboard (includes per-script columns)

Script Toolbar Controls

Control	Description
Sort	Sort scripts by Cost, Name, Type, Scene, or Calls
Filter	Filter scripts by entity type (actor, scene, trigger)
Group by Entity	Group scripts by their parent entity with expandable rows
Inclusive	Toggle between self-time (excludes nested calls) and inclusive time (full duration including nested calls)

Detached Window

The profiler can be detached into its own window using the detach button in the profiler tab header. The detached window closes automatically when the debugger disconnects.

Script Budgets

Like section budgets, you can set per-script scanline budgets by clicking the budget icon (circle) next to each script. When exceeded, the row highlights red and a budget line appears on the sparkline. Budgets persist in localStorage across sessions.

Navigation

Clicking on a script row navigates to the corresponding entity (scene, actor, or trigger) in the editor and opens the matching script tab.

Planes

The **Plane Viewer** lets you visualize the different planes in their entirety, as well as the current scroll position. You can view a single plane at a time based on the selected value.

Memory Viewer

The **Memory Viewer** lets you inspect the Mega Drive's 64KB of 68000 RAM in real time. Select a memory zone and starting offset to browse hex values. This is useful for debugging custom C plugins, checking variable values at specific addresses, or understanding the memory layout of your running game.

VDP Registers

The **VDP Registers** tab displays all 24 Video Display Processor registers with their current values and human-readable descriptions. This shows the active display mode, nametable addresses, scroll settings, sprite table location, auto-increment value, DMA state, and more. Values update in real time as the game runs.

Sprite Table

The **Sprite Table** viewer shows all active hardware sprites with their position, size, tile index, palette, flip flags, priority, and link chain order. Each sprite entry includes a small preview thumbnail. This is useful for debugging sprite limits, checking sprite ordering, and verifying sprite attributes.

CRAM (Color RAM)

The **CRAM** viewer shows the VDP's Color RAM — all 4 palette lines (64 colors total). Colors are displayed as swatches. When scanline palette mode is enabled, you can view per-scanline palette changes, which is useful for debugging HBlank palette effects.

Debugger Log

The **Debugger Log** showcases the log produced by the engine as well as the log you send with the Log Message event.

Layers

The **Layers** pane provides a way to show / hide the different layers.

Palettes

The **Palette Viewer** pane provides a way to consult the palettes per frame and per scanline.

VRAM

The **VRAM Viewer** shows the current video memory used by graphics in your running game.

You can use this pane to get a visual preview of how many additional sprite and background tiles are available in the current scene and to help debug issues when you have used too many unique tiles in the scene.

You can also choose the palette you want the tiles to use during render.

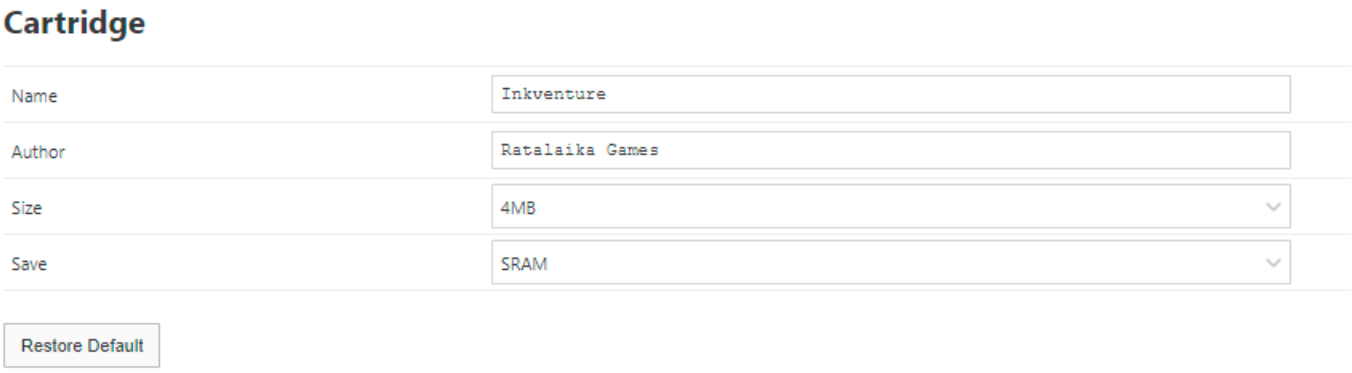
Settings

The *Settings* view will allow you to change your main project's settings.

Cartridge

This section allows you to change some of the basic information placed into the ROM metadata.

- **Name** - The game name.
- **Author** - The developer's name.
- **Size** - The target ROM size, you can check the Saving section to know more.
- **Save** - The preferred save mode, you can check the Saving section to know more.



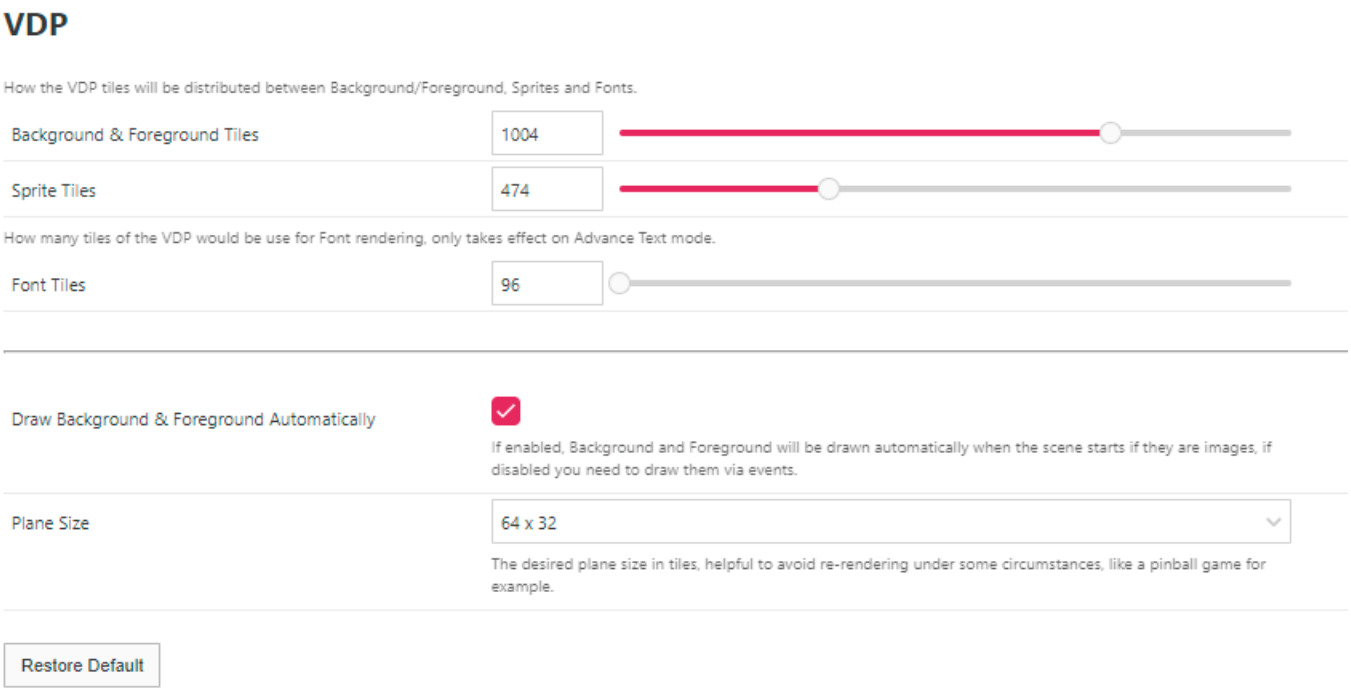
ROM Size Limits

The Mega Drive supports ROM sizes up to 4MB (32Mbit) by default. With the Two Black Cats custom PCB sizes up to 8MB are supported. Larger ROMs consume more space for assets (sprites, backgrounds, music, sound effects). You can monitor your ROM usage in the Build Log after building your project.

VDP

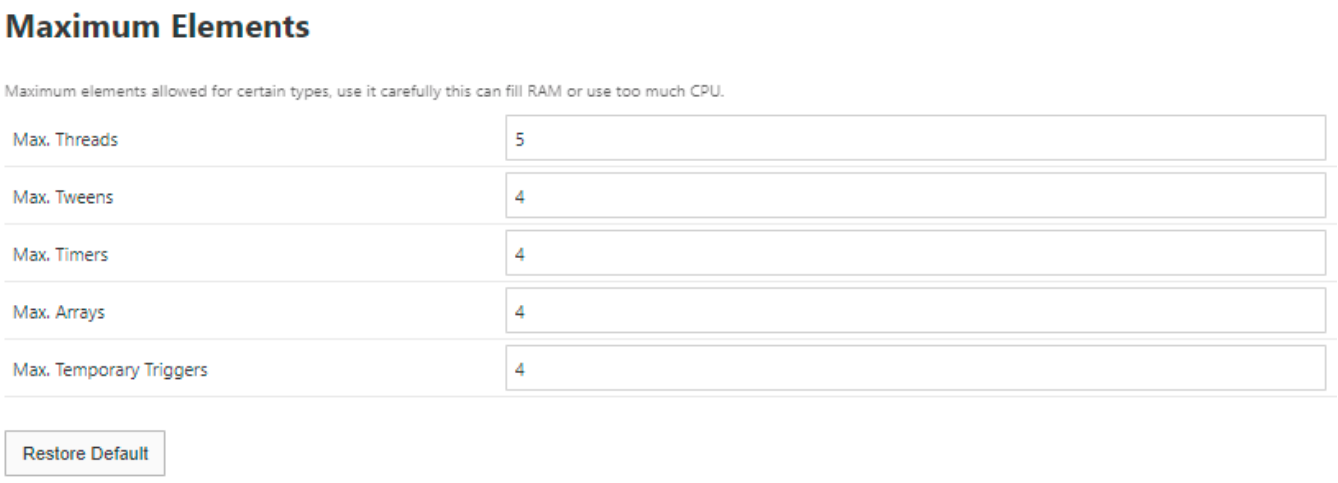
This section allows you to change the default VDP memory arrangement, you can also use the VDP Tiles Setup event on the scene *OnStart*, or *OnEnd* to change this during runtime.

- **Background & Foreground Tiles** - The number of tiles reserved for Backgrounds & Maps.
- **Sprite Tiles** - The number of tiles reserved for Sprites.
- **Font Tiles** - The number of tiles reserved for Fonts.
- **Draw Background & Foreground Automatically** - This allows you to disable the automatic rendering of Background & Foreground when a scene loads.
- **Plane Size** - This changes the default plane size.



Maximum Elements

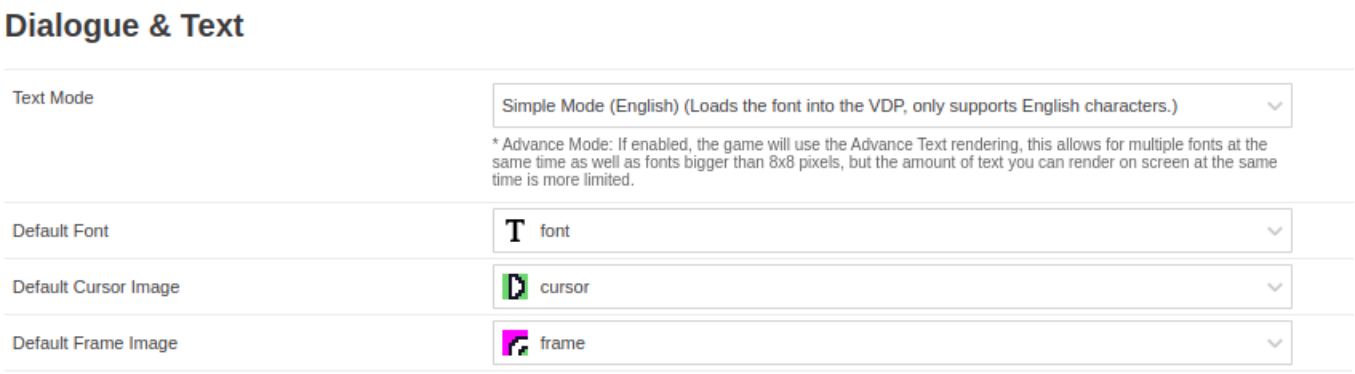
This section allows you to change the maximum amount of different module elements, some of these consume RAM if you have too many, so it is best to increase only if needed.



There is more information regarding what each of these are in the Modules section.

Dialogue & Text

This section allows you to change the text mode, the default font as well as the default Dialogue Cursor tileset and Dialogue Frame tileset.



A detailed explanation of each **Text Mode** can be found below.

Simple Mode (English)

This mode is enabled by default due to it being the one that uses the least VDP space as well as allows for fast rendering. It only supports a small sub-set of the ASCII character set, you can check the entire list on the section regarding font asset. It will need 96 VDP tiles.

Simple Mode (EFIGSP)

This mode can be enabled if you want to include extra language support without much VDP compromise, it supports the lower case special letters of French, Italian, German, Spanish, and Portuguese. You can check the entire list on the section regarding font asset. It will need 118 VDP tiles.

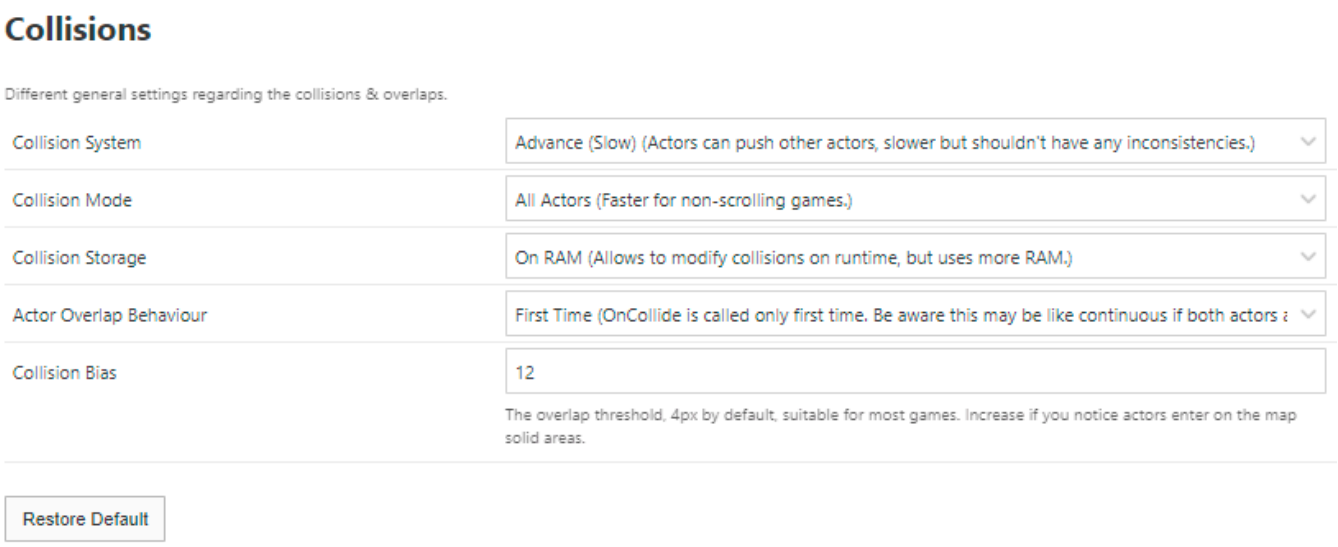
Advanced Mode

This mode allows support for Asian languages as well as the full ASCII character set, it also allows for higher quality fonts since it supports `16px` x `16px` characters. As a trade-off, it's slower on rendering as they render the text to the VDP instead of loading the entire font into it like on the simple modes. You can check some extra information on the section regarding font asset. The total amount of needed tiles can be set on the VDP section outlined before.

Collisions

This section allows you to change settings related to the Collisions module.

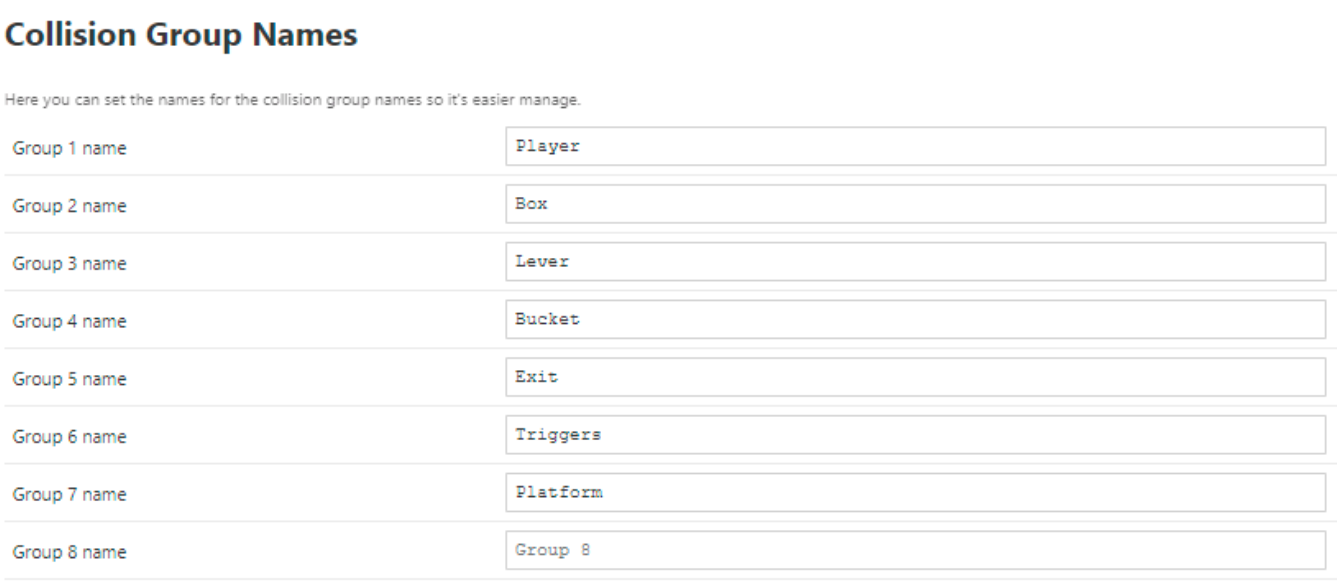
- **Collision System** - The game collision system.
- **Collision Mode** - The game collision mode.
- **Collision Storage** - How the scene collision information is stored.
- **Actor Overlap Behaviour** - The overlap behaviour between actors.
- **Collision Bias** - The collision simulation bias value.



There is more information regarding what each of these are in the Collisions module docs.

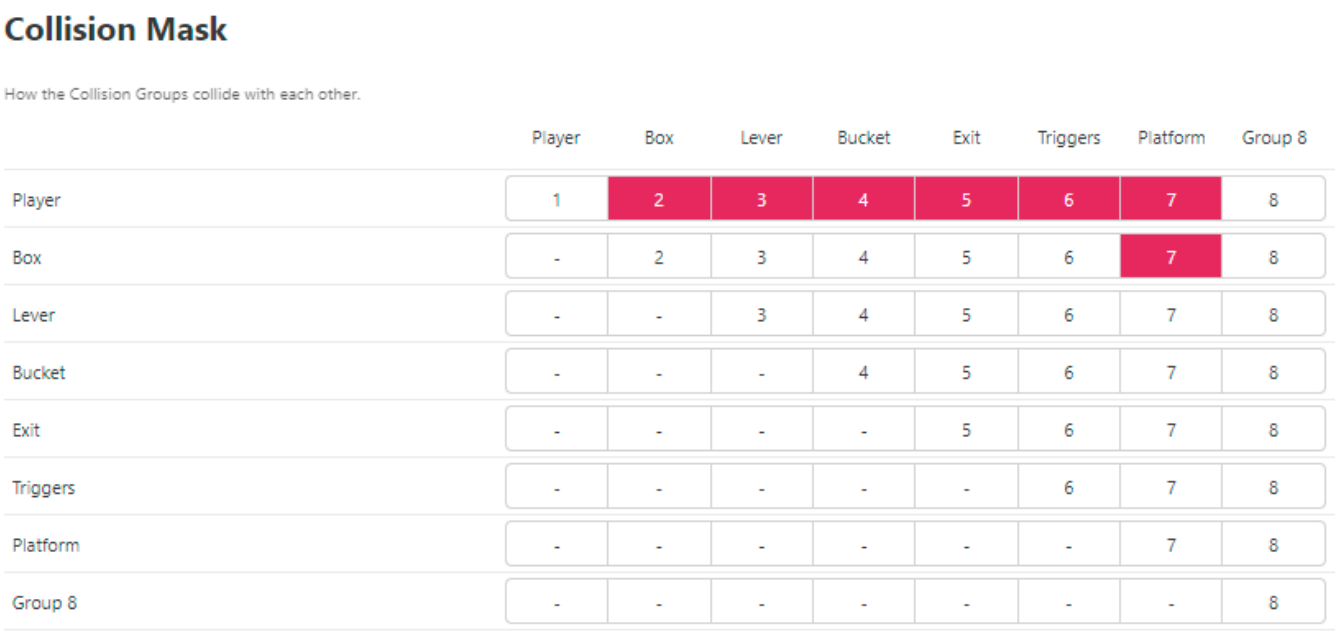
Collision Group Names

This section allows you to change the collision group names, so they are easier to recognize while working on your game.



Collision Mask

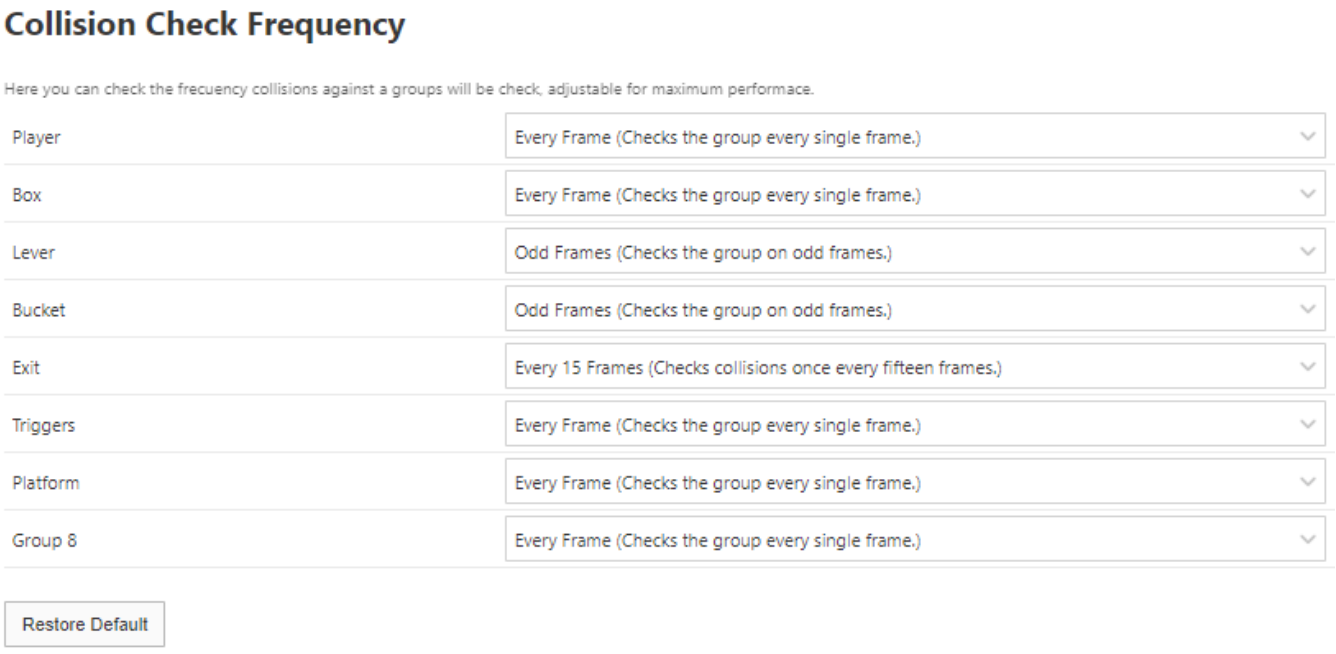
This section allows you to change how the collision groups collide with each other.



There is more information regarding what each of these are in the Collision Masking section.

Collision Check Frequency

This section allows you to change how frequently the collisions for each group are checked.



There is more information regarding what each of these are in the Collision Frequency section.

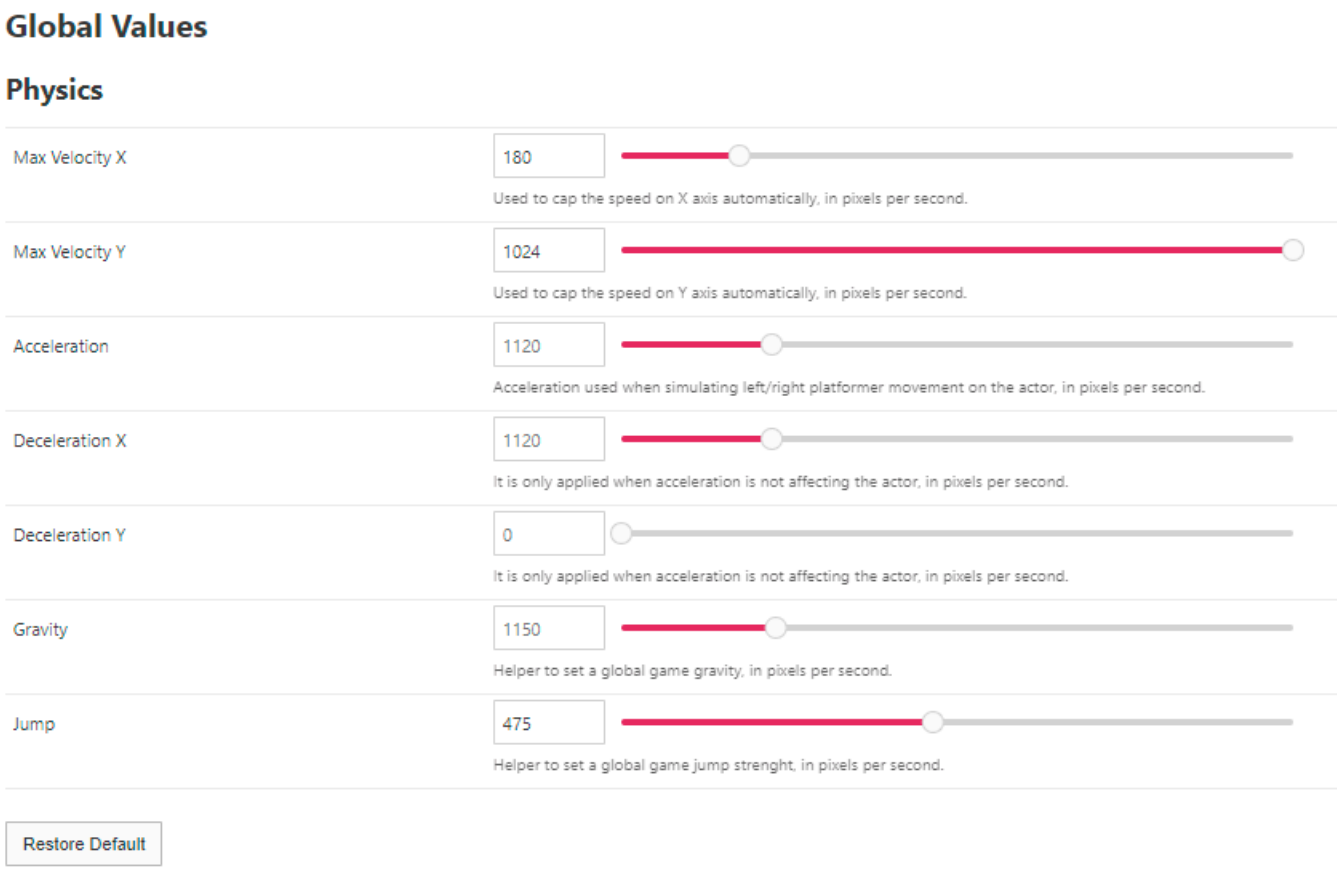
Pathfinding

These settings control the Pathfinding module:

- **Max. Path Slots** — The maximum number of paths available at once. The default is 4; the setting ranges from 2 to 8 in steps of 2.
- **Pathfinding Nodes/Frame** — The number of search nodes processed each frame. The default is 64; the setting ranges from 0 to 512 in steps of 16.
- **Pathfinding Algorithm** — **Best-First Search** is the default and searches quickly, but may produce a longer path. **A*** uses weighted distances to find an optimal path.
- **Pathfinding Queue** — The search queue size. **Small (256)** is the default; **Medium (512)**, **Large (1024)**, and **Huge (2048)** are available for maps that need a larger search area.

Engine Fields

This section allows you to change the engine fields, right now this only covers physics-related values.

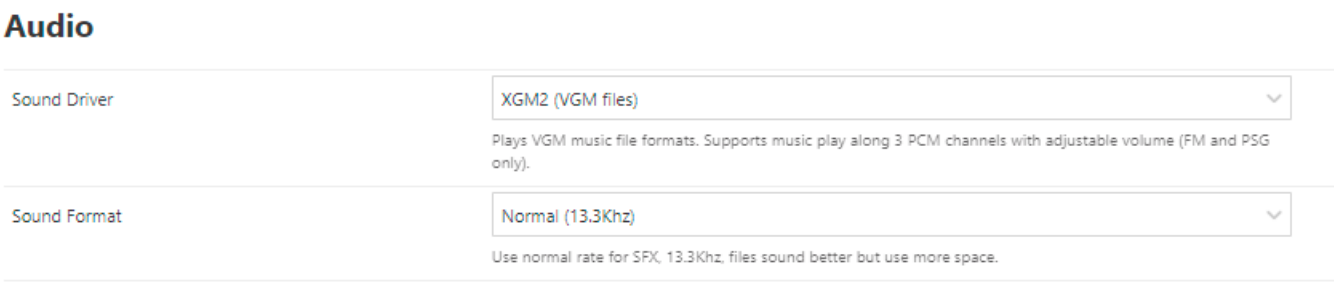


There is more information regarding what each of these are in the Engine Fields docs.

Audio

This section allows you to change the export Audio settings.

- **Sound Driver** — Select the music driver: **XGM2** (VGM files, 3 PCM channels with volume control), **XGM** (VGM/XGM files, no fade or volume control), or **MDSDRV** (MML files via mdslink).
- **Sound Format** — Export sound quality for WAV files: **Normal** (13.3 kHz) or **Half Rate** (6.65 kHz). Only available with the XGM2 driver.



There is more information regarding what each of these are in the Sound Driver section.

Build Options

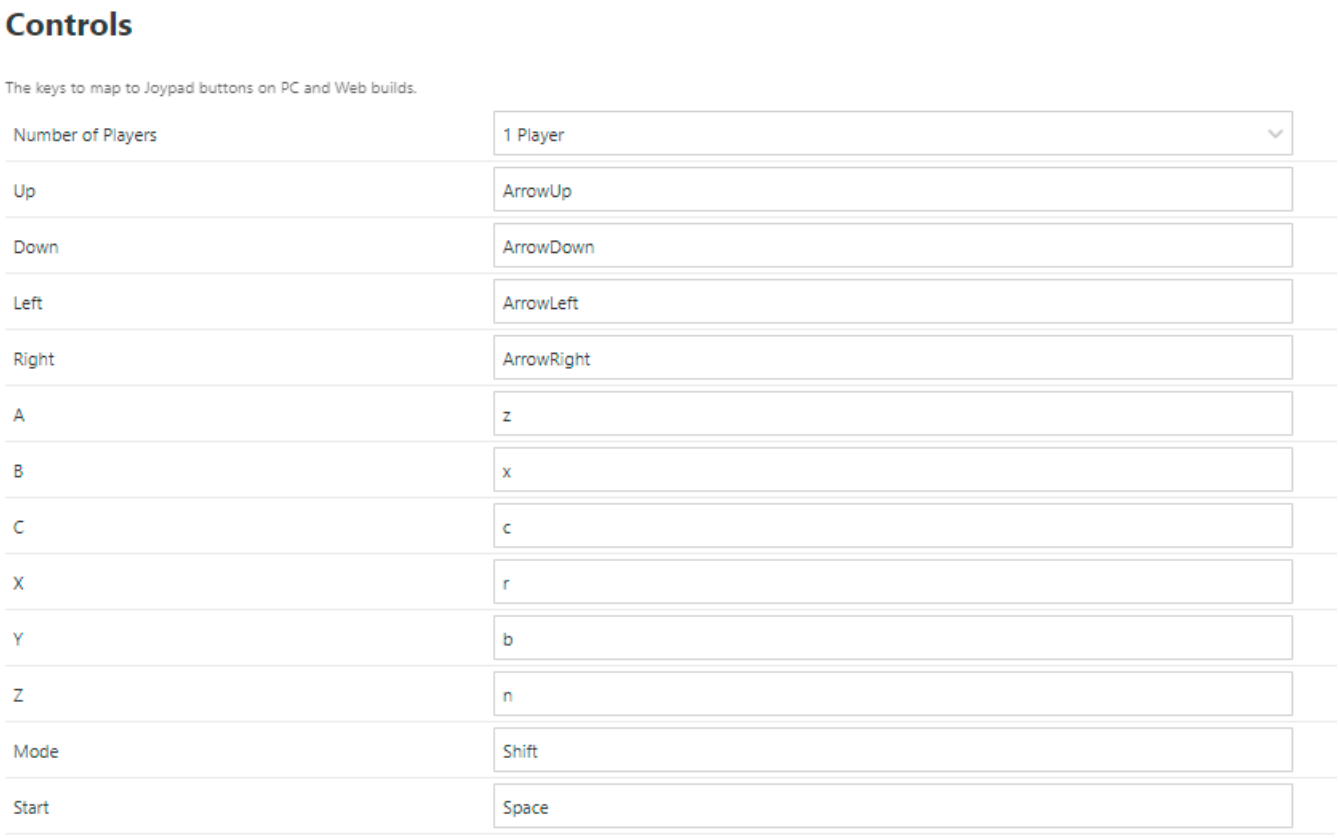
This section allows you to change the settings relating to building your project.

- **Open Build Log On Warnings** Choose if the `Build Log` should be displayed automatically when warnings are displayed such as "too many unique tiles or palettes used in a scene". If you're choosing to ignore these warnings it can be helpful to disable them, otherwise you'll be taken to the `Build Log` every time your game runs. Please note that *errors* building your project will always open the `Build Log` regardless of this setting.

This section also includes four **experimental compiler optimization settings** — Script Inlining, Advanced Script Inlining, CSE & LICM, and Parameterized Script Dedup. Advanced Script Inlining requires Script Inlining. For details on what each pass does and when to use them, see Compiler Optimizations.

Controls

This section allows you to change the default keyboard and gamepad controls used when playing your game from a web build, desktop build, and the *Play Window*.



To edit the keyboard 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.



To edit the gamepad controls for a button select the modern gamepad button that will be mapped to the MD button, several modern buttons can map to the same MD button.

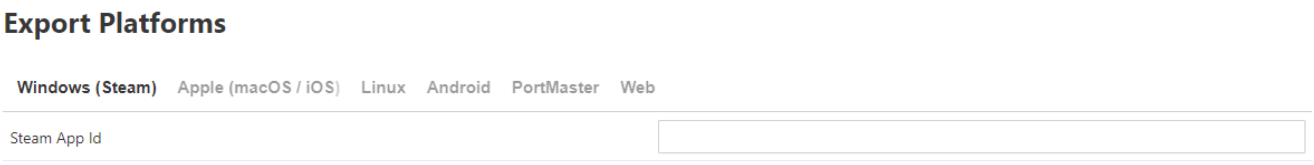
To reset to the original controls you can use the *Restore Default* button.

Export Settings

The *Export Settings* section allows you to configure platform-specific metadata for each export target. The settings are organized by platform tabs.

Steam

- **Steam App Id** — Your game's Steam application identifier. Required for the Windows/Linux Steam export to enable achievements and other Steam features.



Apple

macOS

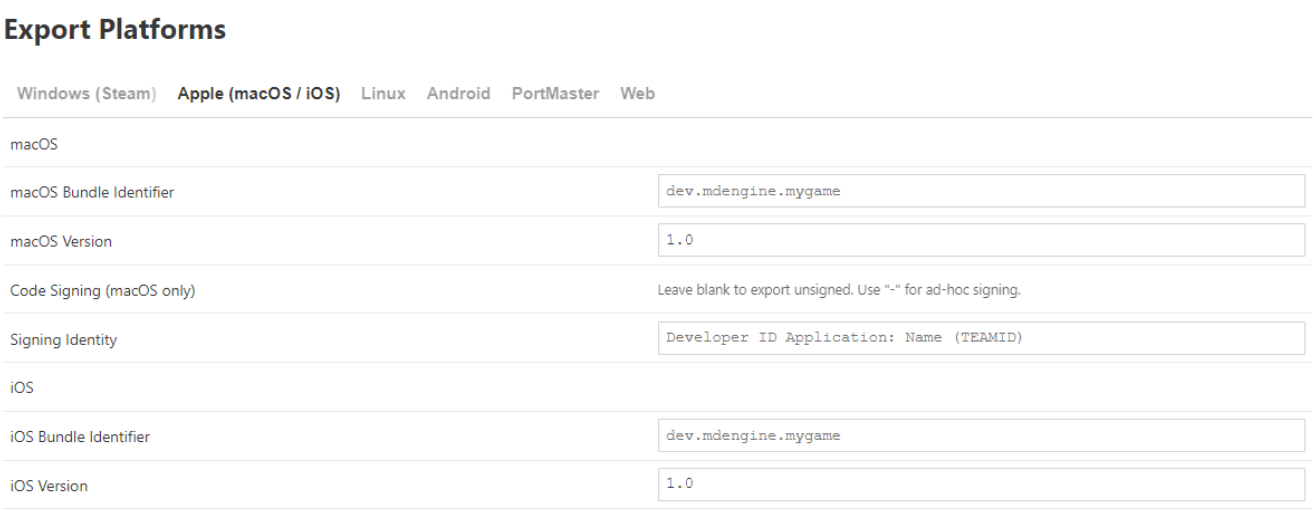
- **Bundle ID** — The macOS application bundle identifier (e.g. `dev.mdengine.mygame`). Required for distribution.
- **Version** — The version string displayed in the app's info panel.
- **Signing Identity** — Your Apple Developer signing identity for code signing and notarization (e.g. `Developer ID Application: Your Name (TEAMID)`). Optional but recommended for distribution.

> **WARNING**
>
> macOS exports have not yet gone through Apple certification (App Store review).

iOS

- **Bundle ID** — The iOS application bundle identifier.
- **Version** — The version string.

> **WARNING**
>
> iOS exports have not yet gone through Apple certification. Building requires macOS with Xcode.



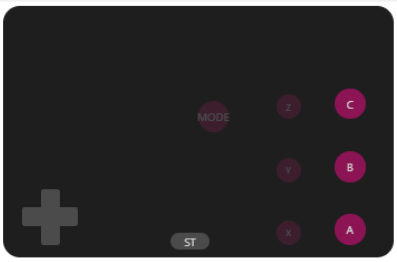
Android

- **Package Name** — Java-style package identifier (e.g. `dev.mdengine.mygame`).
- **Version Code** — An integer that must increase with each release.
- **Version Name** — A display version string (e.g. `1.0.0`).
- **Keystore Path** — Path to your `.jks` keystore file for APK signing.
- **Key Alias** — The alias of the signing key within the keystore.

> **WARNING**
>
> Android exports have not yet gone through Google Play certification.

Export Platforms

Windows (Steam) Apple (macOS / iOS) Linux **Android** PortMaster Web

Android Package Name	dev.mdengine.mygame
Version Code	1
Version Name	1.0.0
Release Signing (for Google Play)	Leave blank to use debug signing (sideload only)
Keystore Path (.jks)	/path/to/release.jks
Key Alias	my-key-alias
Layout Preset	Standard
Touch Opacity	80%
Preview	
D-Pad	▶ D-Pad
Button A	▶ Button A
Button B	▶ Button B
Button C	▶ Button C
Button X	▶ Button X

Linux

- **Version** — The package version string (e.g. `1.0.0`). Used by DEB and RPM exports.
- **Description** — A short description of the game. Used in DEB/RPM package metadata and desktop entries.

Export Platforms

Windows (Steam) Apple (macOS / iOS) **Linux** Android PortMaster Web

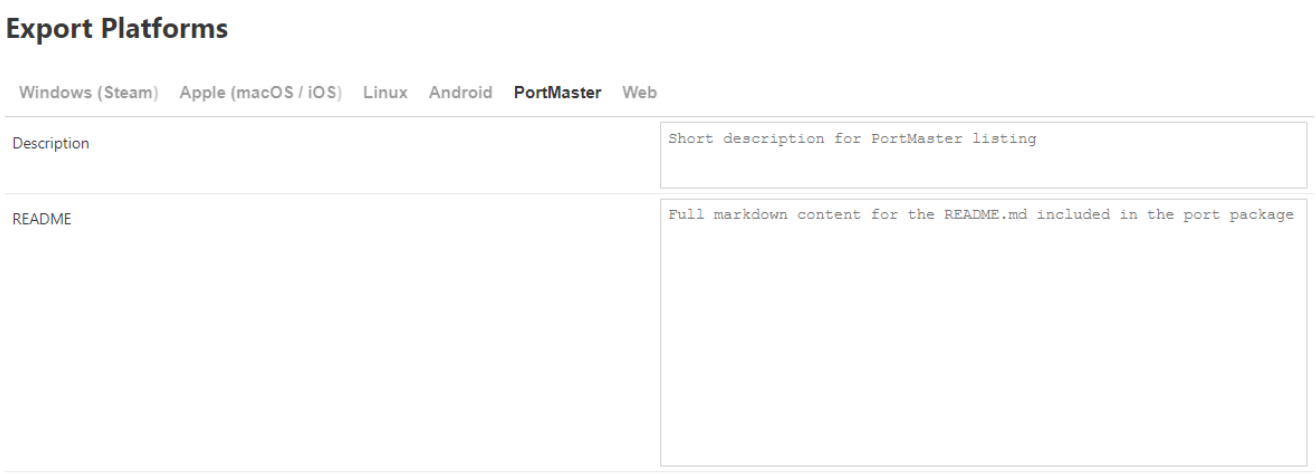
Version	1.0.0
Description	A game made with MD Engine

PortMaster

> **EXPERIMENTAL**
>
> PortMaster support is experimental. The packaging format may change.

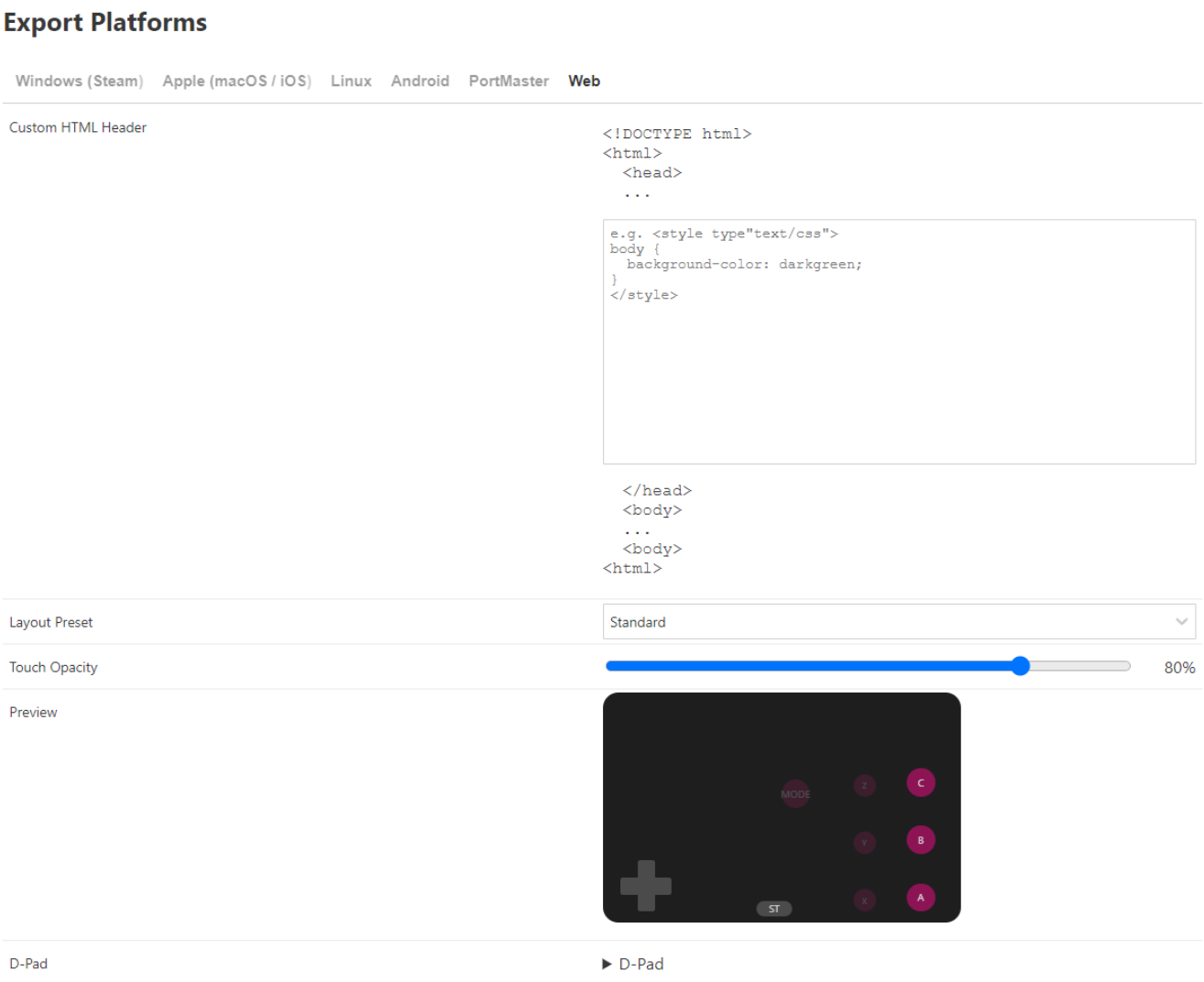
- **Description** — A short description included in the PortMaster package metadata.

- **Readme** — The full README content bundled with the PortMaster package.



Web

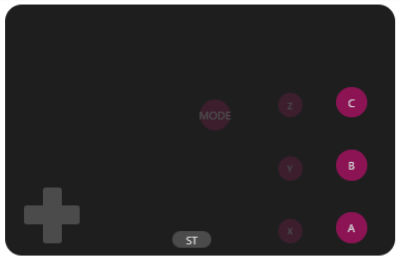
- **Custom HTML Header** — Custom content added to the HTML `<head>` section of the web build. You can use this to add CSS, JavaScript, or meta tags.



Touch Controls

Available for **Web**, **Android**, and **iOS** exports. Allows you to configure on-screen touch controls for mobile and tablet play.

- **Layout Preset** — Choose from Standard, Compact, Left-Handed, Diagonal Up, Diagonal Down, Six-Button, Minimal, or Custom.
- **Per-button configuration** — Toggle visibility, adjust scale and position for each button.
- **Phone Preview** — A live mockup showing the current button arrangement.
- **Enable/Disable** — Toggle touch controls entirely.

Layout Preset	Standard
Touch Opacity	80%
Preview	
D-Pad	▶ D-Pad
Button A	▶ Button A
Button B	▶ Button B
Button C	▶ Button C
Button X	▶ Button X
Button Y	▶ Button Y
Button Z	▶ Button Z

Build Process

Overview

When you press **Play** or **Export**, MD Engine compiles your project through several stages:

1. **Visual Scripts to C Code** — Your visual event scripts, scene data, actor definitions, and asset references are translated into C source files.
2. **C Code to 68000 Assembly** — The SGDK-based compiler processes the generated C code and produces Motorola 68000 assembly.
3. **Assembly to ROM Binary** — The assembler and linker combine all object files into a final `.bin` ROM file that runs on real hardware or an emulator.

This entire pipeline runs automatically. You can monitor its progress and diagnose issues through the Build Log.

Between stages 1 and 2, several compiler optimization passes run on the generated code to reduce ROM size and improve runtime performance. These can be toggled in **Settings > Build Options**.

Build Log

The **Build Log** window displays detailed output from the compilation pipeline. It shows:

- Asset processing progress (sprites, backgrounds, music, palettes).
- C compiler output, including any warnings or errors in generated code or plugin code.
- Linker output with the final ROM size.
- Timing information for how long each stage took.

To open the Build Log, check the output panel after pressing Play or Export. Warnings appear in yellow and errors in red.

Common Errors

Too Many Unique Tiles

Scenes have a limited number of unique 8x8 tiles available for backgrounds. The Mega Drive VDP can address a fixed amount of VRAM for tile data, and each unique tile pattern consumes one slot.

What causes it:

- Highly detailed or noisy background images with many unique 8x8 pixel patterns.
- Using multiple complex backgrounds in a single scene.

How to fix:

- Reduce the number of unique tiles by simplifying background art.
- Share tiles between layers — reuse the same 8x8 patterns where possible.

- Use simpler backgrounds with repeated tile patterns.
- Check the VRAM Viewer in the Debugger to see how many tiles are in use.

Palette Overflow

Each scene can use up to 4 palette lines (64 colors total). Sprites and backgrounds share this palette space.

What causes it:

- Too many sprites or backgrounds each requesting their own palette line.
- Assets that use colors not present in any existing palette line.

How to fix:

- Reduce the number of distinct palette lines by sharing palettes between sprites.
- Edit your assets to use colors from existing palette lines.
- Check the Palettes documentation for palette management tips.

ROM Size Exceeded

The ROM has a maximum size of 4MB by default (8MB with a custom PCB configuration). As you add assets, the ROM grows.

How to monitor:

- After each build, the Build Log shows the total ROM size.
- Keep an eye on this value as you add sprites, backgrounds, music, and scenes.

Tips to reduce ROM size:

- Compress or simplify assets (fewer unique tiles, smaller sprite sheets).
- Reduce the number of scenes or consolidate similar scenes.
- Use shared tilesets and palettes to avoid duplicating data.
- Remove unused assets from your project.

Build Timeout / Compilation Failed

If the build takes too long or fails outright:

- Check the Build Log for the specific error message — it usually points to the exact file and line.
- Try cleaning the build folder and rebuilding (the editor handles this automatically in most cases).
- If the error is in plugin code, check the plugin's C source files for syntax errors.
- Verify your project doesn't exceed the hardware limits listed in Settings.

ROM Size Monitoring

After each successful build, the Build Log reports the total ROM size in bytes. As you add assets (sprites, backgrounds, music, sound effects, scenes), the ROM grows. Regularly checking this value helps you catch size issues early rather than discovering them when you are close to the limit.

TIP

If your ROM is approaching the 4MB limit, review your largest assets first — background images and music tracks tend to consume the most space.

Compiler Optimizations

Overview

MD Engine includes experimental compiler optimization passes that can reduce ROM size and improve runtime performance. These passes analyze your visual scripts after they've been converted to an internal representation (IR) and apply transformations before generating the final C code.

The four settings in **Settings > Build Options** are enabled by default. CSE and LICM share one setting. Advanced Script Inlining extends Script Inlining and only runs when Script Inlining is enabled.

> **WARNING**
>
> These optimizations are marked as **Experimental**. If you encounter build errors or unexpected runtime behavior after enabling them, disable the passes one at a time to isolate the issue.

How It Works

When you build your project, the compilation pipeline works in stages:

1. **Visual Scripts → IR** — Your event scripts are converted into a structured Intermediate Representation (Stmt/Expr tree).
2. **Optimization Passes** — The IR is analyzed and transformed (inlining, CSE, LICM, dedup).
3. **IR → C Code** — The optimized IR is rendered to C source files.
4. **C → Assembly → ROM** — GCC compiles the C code with its own optimizations (-O2/-O3), then the assembler and linker produce the final ROM.

The compiler's optimization passes complement GCC — they operate on game-specific patterns that GCC cannot see (like repeated actor property reads or identical script bodies across actors).

Optimization Passes

Script Inlining

What it does: Replaces calls to small synchronous scripts with the script's body directly at the call site, eliminating function call overhead.

Example: If you have a utility script "Reset Temp Variables" that just sets two variables to zero, instead of generating a function call:

```
// Before: function call
res = script_70(&globalCtx);

// After: inlined body
TBCE_Temp[6] = 0;
TBCE_Temp[7] = 0;
```

When it helps:

- Projects with many small reusable scripts called from multiple places.
- Eliminates the overhead of parameter marshalling and function dispatch for trivial scripts.

Current limitations:

- Only inlines scripts with no parameters (`V_VAR_N`, `AC_PARAM_N`).
- Scripts must not use actor locals (`__self`, `__a0`).
- Scripts with early returns (inside if/switch) are skipped.
- Only synchronous scripts qualify (async scripts with suspend points cannot be inlined).

> **TIP**
>
> Enable **Advanced Script Inlining** to handle value parameters, actor locals, and early returns. The general inlining limits below still apply.

Setting: *Enable Script Inlining (Experimental)*

Advanced Script Inlining

What it does: Extends basic inlining to handle scripts that take parameters (`V_VAR_N`, `AC_PARAM_N`) and use actor locals (`__self`, `__a0`). Keeps the callee's declarations but substitutes their initialization with the actual call-site values, wrapped in a scoped block.

Constant specialization: When a parameter is a compile-time literal (e.g., SFX ID `11`), `switch`/`if` statements on that parameter are folded — only the matching branch is emitted.

Example (Play SFX with literal param):

```
// Before: 630-line function call
TBCE_scriptParams[0] = 11;
TBCE_variableParams[0] = 2;
res = script_12(&globalCtx);

// After: constant-folded inline (only case 11 emitted)
{
    if ((TBCE_Variables[53] > 0))
    {
        TBCE_SoundPlayPCM(sound_gun_slash, TBCE_SFX_CHANNEL_AUTO, 6, false);
    }
}
```

When it helps:

- Scripts called with literal values (SFX IDs, animation indices, constant flags).
- Eliminates parameter marshalling (global array writes + TBCE_UnpackVariable calls).
- Combined with constant folding, can reduce 630-line functions to a few lines at each call site.

Early return handling: Scripts with `return` statements inside `if`/`switch`/`for`/`while` blocks are supported — each early return is converted to a `goto` past the inlined block. For example:

```
// Before inlining: script_78 (Destroy Tile) has early returns
if (collision == DIR_DOWN) {
    setTile(0);
    return true;    // early return
}
setTile(0);
clearCollision();

// After inlining: returns become gotos
{
    if (collision == DIR_DOWN) {
        setTile(0);
        goto __inline_end_0;    // was: return true
    }
    setTile(0);
    clearCollision();
}
__inline_end_0:;
```

Current limitations:

- Scripts must be synchronous and below the size threshold (20 nested statements after body extraction).
- Scripts with reference parameters, recursive calls, or bodies that depend on the script context (`ctx`) are skipped.

Setting: *Enable Advanced Script Inlining (Experimental)* (requires **Enable Script Inlining**)

Common Subexpression Elimination (CSE)

What it does: Finds expressions that appear two or more times across different statements within the same scope and hoists them into a temporary variable, so they're computed only once.

Example:

```
// Before: TBCE_ActorGetX called 3 times
if ((TBCE_ActorGetX(__self) > 100)) { ... }
if ((TBCE_ActorGetX(__self) < 200)) { ... }
TBCE_Temp[0] = TBCE_ActorGetX(__self);

// After: computed once
s16 __cse0 = TBCE_ActorGetX(__self);
if ((__cse0 > 100)) { ... }
if ((__cse0 < 200)) { ... }
TBCE_Temp[0] = __cse0;
```

When it helps:

- Scripts that read the same actor property (position, size, animation) multiple times.
- Collision checks that compute the same map index repeatedly.
- Any script with repeated array accesses like `TBCE_Variables[N]`.

How it stays safe:

- Only hoists expressions whose dependencies aren't modified between uses.
- Writes to a variable invalidate any CSE that reads it.
- Uses a whitelist of known-pure functions (getters like `TBCE_ActorGetX`, `TBCE_ActorGetY`, `TBCE_ActorOnScreen`) that are safe to hoist.

Setting: *Enable CSE & LICM (Experimental)*

Loop-Invariant Code Motion (LICM)

What it does: Identifies expressions inside `for` and `while` loop conditions that don't change across iterations and moves them before the loop.

Example:

```
// Before: localVars[3] read every iteration
for (TBCE_Temp[5] = (localVars[3] - 1); TBCE_Temp[5] <= (localVars[3] + 1); TBCE_Temp[5] += 1)
{
    // loop body doesn't modify localVars[3]
}

// After: computed once before the loop
s16 __licm0 = localVars[3];
for (TBCE_Temp[5] = (__licm0 - 1); TBCE_Temp[5] <= (__licm0 + 1); TBCE_Temp[5] += 1)
{
    // ...
}
```

When it helps:

- For-loops that use actor properties or variable reads in their condition.
- Nested loops where the outer variable is used in the inner loop's bounds.

How it stays safe:

- Tracks all variables written inside the loop body (including nested if/switch).
- Uses precise alias analysis — writing `localVars[1]` does NOT block hoisting `localVars[3]` (different constant index).
- Skips hoisting if any raw code in the loop could affect the expression.

Setting: Shares toggle with CSE — *Enable CSE & LICM (Experimental)*

Parameterized Script Dedup

What it does: Finds scripts that are structurally identical except for a few differing constants (such as actor IDs). Instead of emitting 20 copies of the same function, it emits one shared implementation with the varying values as parameters, plus thin one-line wrapper functions.

Example:

```
// Before: 20 identical ~800-line functions
bool actor_167_update(TBCE_ScriptContext* ctx) { /* 800 lines with AC_ACTOR_167 */ }
bool actor_168_update(TBCE_ScriptContext* ctx) { /* 800 lines with AC_ACTOR_168 */ }
// ... 18 more copies

// After: 1 shared function + 20 one-line wrappers
static bool __shared_update_0(TBCE_ScriptContext* ctx, u16 __p0) { /* 800 lines with __p0 */ }
bool actor_167_update(TBCE_ScriptContext* ctx) { return __shared_update_0(ctx, AC_ACTOR_167); }
bool actor_168_update(TBCE_ScriptContext* ctx) { return __shared_update_0(ctx, AC_ACTOR_168); }
```

When it helps:

- Projects with many actors that share the same event logic (e.g., enemies, collectibles, NPCs with identical behavior).

- Significantly reduces ROM size — 20 copies of 800 lines becomes ~800 + 20 lines.

How it works internally:

- Compares scripts' IR trees structurally, replacing leaf values with placeholders.
- Groups scripts with identical "skeletons" (same structure, different constants).
- Diffs the IR to find exactly which values vary.
- Supports up to 4 varying parameters per group.

> **TIP**
> GCC will typically inline the thin wrapper functions at -O2/-O3, so there is zero runtime overhead from this optimization — only ROM savings.

Setting: *Enable Parameterized Script Dedup (Experimental)*

Troubleshooting

When to Disable Passes

If you experience issues after a build:

1. **Build fails with C compiler errors** (e.g., undeclared variables, type mismatches) — an optimization pass generated invalid code. Disable passes one at a time to find which one.
2. **Game behavior changes** (e.g., collisions not working, scripts not executing correctly) — a pass may have incorrectly transformed your logic. Disable and report the issue.
3. **Build succeeds but ROM is unexpectedly large** — the parameterized dedup might not be catching your scripts (check if they have raw code differences).

Known Limitations

Pass	Limitation
Script Inlining (Basic)	Only parameterless sync scripts without actor locals or early returns
Advanced Script Inlining	Handles value parameters, actor locals, and early returns; skips async scripts, reference parameters, recursive callees, `ctx` dependencies, and bodies over the size threshold
CSE	Won't hoist across raw code blocks or non-pure function calls
LICM	Only hoists from for/while conditions, not arbitrary loop body expressions
Param Dedup	Skips groups where differences are in raw code (e.g., preprocessor guards); max 4 varying params

Settings Reference

These settings are found in **Settings > Build Options**:

Setting	Default	Description
Enable Script Inlining	On	Inline small parameterless sync scripts directly at call sites
Enable Advanced Script Inlining	On	Extend inlining to parameterized scripts with constant folding and early return → goto conversion
Enable CSE & LICM	On	Hoist repeated expressions and loop-invariant code into temporaries
Enable Parameterized Script Dedup	On	Merge structurally identical scripts into shared functions

All four settings default to **On**. Use the **Restore Defaults** button in Build Options to re-enable them all.

Tools

MD Engine includes several built-in tools for creating and editing game assets directly within the editor. These tools are designed to work seamlessly with the engine's asset pipeline and hardware constraints.

- Mini Painter — Pixel art editor for sprites and backgrounds
- Mini Tracker — Music tracker for composing VGM music
- Mini Notator — Music notation editor
- Image Converter — Convert images to engine-compatible formats
- Font Converter — Convert TTF fonts to engine-compatible formats
- Bfxr — Sound effects generator
- Flash ROM to Cart — Flash ROMs, read/write SRAM and flash saves on FlashKit-MD hardware

Mini Painter

Mini Painter is the built-in pixel art editor for creating and editing sprites, backgrounds, and tilesets. It is designed around the Mega Drive's palette and tile constraints, ensuring your artwork is always hardware-compatible.

Features

- **Indexed color editing** — All drawing uses palette indices (0–63), not RGB colors. This guarantees compatibility with the Mega Drive's 4 palette lines of 16 colors each.
- **Drawing tools** — Brush, eraser, flood fill, spray, line, arc, rectangle, circle, and gradient.
- **Selection tools** — Rectangular, polygon, and flood select with cut/copy/paste support.
- **Layers** — Multiple layers with opacity and visibility controls.
- **Animation** — Frame-based animation with per-frame and per-animation speed controls.
- **Tile mode** — View and edit individual 8x8 tiles with automatic VRAM synchronization.
- **Palette editor** — Edit colors, sort, spread, and import palettes from images.
- **Dither support** — Built-in dither patterns for drawing and gradients.
- **Smudge, blur, and sharpen** — Pixel manipulation tools with adjustable strength.
- **Sprite sheet import** — Import sprite sheets with automatic frame detection and animation grouping.
- **Undo/redo** — Full history support for all operations.

Palette Constraints

Mini Painter enforces the Mega Drive's palette rules:

- 4 palette lines of 16 colors each (64 colors total).
- Each 8x8 tile can only use colors from a single palette line.
- Colors are 9-bit RGB (3 bits per channel, 512 possible colors).
- The first color of each palette line is treated as the background/erase color.

For more details on palettes, see Palettes.

Keyboard Shortcuts

Shortcut	Action
`B`	Brush tool
`E`	Eraser
`G`	Flood fill
`L`	Line tool
`R`	Rectangle tool
`O`	Circle tool
`S`	Selection tool
`I`	Color picker
`H`	Pan
`+` / `-`	Zoom in / out
`Ctrl+Z`	Undo
`Ctrl+Y`	Redo
`Ctrl+C`	Copy
`Ctrl+V`	Paste
`Ctrl+X`	Cut
`Delete`	Clear selection

Mini Tracker

Mini Tracker is the built-in music tracker for composing music compatible with the Mega Drive's sound hardware and the XGM2 sound driver. It provides a traditional tracker-style interface with full access to the YM2612 FM synthesizer, SN76489 PSG, and DAC sample playback.

Features

- **13-channel editing** — 6 FM channels, 3 FM3-extended operator channels, 3 PSG square channels, and 1 noise channel.
- **FM instrument editor** — Full 4-operator FM synthesis with algorithm, feedback, ADSR envelopes, and 11 built-in presets.
- **PSG instrument editor** — 16-step volume envelopes and arpeggio sequences with 4 presets.
- **DAC sample editor** — Load, edit, and loop audio samples (WAV, MP3, OGG, FLAC, AIFF) for playback on FM channel 6.
- **Pattern-based sequencing** — Up to 256 patterns with configurable row count (1–256), reusable via references or deep copies.
- **22+ effects** — Standard tracker effects (arpeggio, portamento, vibrato, volume slide) plus FM-specific effects for real-time operator control.
- **FM3 extended mode** — Per-operator frequency control for FM channel 3, enabling complex multi-voice patches.
- **Import/export** — Import from VGM, MIDI, DefleMask (.dmf), and Furnace (.fur). Export to VGM and MIDI.
- **MIDI input** — Connect a MIDI keyboard for real-time note entry.
- **Undo/redo** — Full history support for all operations.

Interface Overview

Toolbar

The toolbar spans two rows at the top of the editor:

- **File operations** — New, Save, Load, Import (VGM, MIDI, DMF, Furnace), Export (VGM, MIDI).
- **Transport controls** — Play from cursor (`F5`), Play from start (`F6`), Stop (`F8`).
- **Song parameters** — BPM (60–300), Speed (1–31 ticks per row), Rows per pattern (1–256), Octave (0–9), Edit step (0–16).
- **FX columns** — Number of visible effect columns per channel (1–8).
- **CH3 mode** — Toggle between Normal, Extended, and CSM modes.
- **Master volume** — Adjustable from 0% to 200%.

Order List

The left panel shows the song's pattern sequence. Each entry maps an order position to a pattern number.

- **Add (+)** — Append a new empty pattern.
- **Remove (-)** — Remove the selected order entry (pattern data is preserved).

- **Duplicate (Dup)** — Create an independent copy of the pattern.
- **Reference (Ref)** — Reuse the same pattern data (edits affect all references). A badge shows the reference count.
- **Up / Down** — Reorder entries in the sequence.

Pattern Editor

The central area displays the pattern grid. Each row contains cells for every channel, and each cell has columns for Note, Instrument, Volume, and up to 8 effect slots.

- **Color-coded channels** — FM (blue), PSG (green), DAC (orange), Noise (purple).
- **Edit mode** — Press `Space` to toggle. The cursor border turns red when edit mode is active.
- **Layout modes** — Vertical (default, channels side by side) or Horizontal (channels stacked).

Instrument Editor

The bottom panel has four tabs:

- **FM** — Algorithm/feedback selector, 4 operator panels with full ADSR and parameter controls, 11 presets.
- **PSG** — 16-step volume envelope editor and arpeggio sequence, 4 presets.
- **DAC** — Waveform viewer with loop markers, normalize/reverse tools, sample rate display.
- **Piano** — Interactive keyboard for previewing instruments.

Channels

Mini Tracker provides 13 channels that map to the Mega Drive's sound hardware:

Channel	Chip	Description
FM1–FM6	YM2612	6 FM synthesis channels. FM6 doubles as the DAC output for sample playback.
3-OP1–3-OP3	YM2612	FM3's individual operators when CH3 Extended mode is enabled.
SQ1–SQ3	SN76489	3 PSG square wave channels.
NOI	SN76489	1 noise channel (white or periodic noise).

CH3 Modes

FM channel 3 supports three modes:

- **Normal** — Standard FM channel, all 4 operators share one frequency.
- **Extended** — Each operator can have an independent frequency via the 3-OP channels, enabling multi-voice patches from a single channel.
- **CSM** — Carrier Suppress Modulation mode for special synthesis effects.

FM Instruments

FM instruments use 4-operator frequency modulation synthesis. The algorithm (0–7) determines how the operators are routed (series, parallel, or mixed), and the feedback (0–7) controls self-modulation on operator 1.

Each operator has:

Parameter	Range	Description
Detune (DT)	0–7	Fine frequency offset
Multiplier (MUL)	0–15	Frequency multiplier
Total Level (TL)	0–127	Output level (0 = loudest)
Rate Scaling (RS)	0–3	Key-dependent envelope speed
Attack Rate (AR)	0–31	Attack speed
Decay 1 Rate (D1R)	0–31	First decay speed
Decay 2 Rate (D2R)	0–31	Sustain/second decay speed
Release Rate (RR)	0–15	Release speed
Sustain Level (SL)	0–15	Level at which D1R transitions to D2R
AM Enable	On/Off	Amplitude modulation by LFO
SSG-EG	0–15	SSG envelope generator mode

Additional per-channel parameters:

- **AMS** (0–3) — Amplitude modulation sensitivity.
- **FMS** (0–7) — Frequency modulation sensitivity.

FM Presets

11 built-in presets: Default, Piano, Bass, Organ, Guitar, Strings, Brass, Synth Lead, Synth Pad, Chroma Perc, Reed.

PSG Instruments

PSG instruments define how the SN76489's square wave channels shape their sound:

- **Volume envelope** — A 16-step sequence controlling attenuation over time (0 = loudest, 15 = silent).
- **Arpeggio sequence** — Optional semitone offsets applied per tick for rapid pitch changes.

PSG Presets

4 built-in presets: Default, Staccato (quick decay), Soft (slow attack), Pluck (fast attack and decay).

DAC Samples

FM channel 6 can play back 8-bit PCM samples instead of FM synthesis. The DAC editor supports:

- Loading samples from WAV, MP3, OGG, FLAC, and AIFF files.
- Waveform display with zoom and scroll.
- Loop start/end markers for looping samples.
- Normalize and reverse processing.

Effects

Effects are entered as a type byte followed by a value byte (e.g., `01 40` = portamento up at speed 64). Each cell can have up to 8 effect slots.

Standard Effects

Code	Effect	Value
`00`	Arpeggio	`xy` — Alternates between note, note+x, note+y semitones
`01`	Portamento up	`xx` — Pitch slide speed
`02`	Portamento down	`xx` — Pitch slide speed
`03`	Tone portamento	`xx` — Slide to target note at speed
`04`	Vibrato	`xy` — x = speed, y = depth
`07`	Tremolo	`xy` — x = speed, y = depth
`08`	Set panning	`xx` — `<40` = left, `40–BF` = center, `>=C0` = right
`09`	Note retrigger	`xx` — Retrigger interval in ticks
`0A`	Volume slide	`xy` — x = slide up, y = slide down
`0B`	Jump to order	`xx` — Order position
`0C`	Set volume	`xx` — Volume (0–127)
`0D`	Pattern break	`xx` — Row in next pattern
`0E`	Extended	`Cx` = note cut after x ticks, `Dx` = note delay by x ticks
`0F`	Set speed/BPM	`xx` — Below 20 = speed (ticks/row), 20+ = BPM

FM-Specific Effects

Code	Effect	Value
`10`	Set algorithm	`00–07`
`11`	Set feedback	`00–07`
`12`	Set multiplier	Bits 7–6 = operator, bits 3–0 = value (0–15)
`13`	Set detune	Bits 7–6 = operator, bits 2–0 = value (0–7)
`14`	Set attack rate	Bits 7–6 = operator, bits 4–0 = value (0–31)
`15`	Set D1R	Bits 7–6 = operator, bits 4–0 = value (0–31)
`16`	Set D2R	Bits 7–6 = operator, bits 4–0 = value (0–31)
`17`	Set total level	Bits 7–6 = operator, bits 5–0 = value (0–63)
`18`	Set release rate	Bits 7–6 = operator, bits 3–0 = value (0–15)
`19`	Set sustain level	Bits 7–6 = operator, bits 3–0 = value (0–15)
`1A`	Set SSG-EG	Bits 7–6 = operator, bits 3–0 = value (0–15)
`1B`	Set AMS/FMS	`xy` — x = AMS (0–3), y = FMS (0–7)

PSG-Specific Effects

Code	Effect	Value
`1C`	Noise mode	Bit 2 = white (1) / periodic (0), bits 1–0 = frequency

Global Effects

Code	Effect	Value
`20`	LFO frequency	`00–07`
`21`	Timer A	High byte control
`22`	CSM toggle	`01` = on, `00` = off

File Format

Mini Tracker uses the `.mtrk` file format — gzip-compressed JSON with embedded DAC sample data as base64.

Import Formats

- **VGM** (.vgm) — Experimental, reverse-engineers register writes into patterns.
- **MIDI** (.mid) — Converts note events to tracker patterns.
- **DefleMask** (.dmf) — Full tracker data import.
- **Furnace** (.fur) — Full tracker data import.
- **Instruments** — TFI, TYI, Y12, OPM, DMP files can be imported into the instrument editor.

Export Formats

- **VGM** (.vgm) — Full register log for hardware-accurate playback.
- **MIDI** (.mid) — Note and timing data export.

Keyboard Shortcuts

Playback

Shortcut	Action
`F5`	Play from cursor
`F6`	Play from start
`F8`	Stop
`Space`	Toggle edit mode

File

Shortcut	Action
`Ctrl+S`	Save
`Ctrl+O`	Load
`Ctrl+Z`	Undo
`Ctrl+Y`	Redo

Navigation

Shortcut	Action
`Arrow keys`	Move cursor
`Tab` / `Shift+Tab`	Next / previous channel
`PageDown` / `PageUp`	Jump 16 rows
`Home` / `End`	First / last row
`Ctrl+Up` / `Ctrl+Down`	Jump by beat

Editing

Shortcut	Action
`Delete`	Clear cell
`Insert`	Insert row
`Shift+Delete`	Delete row
`Ctrl+C` / `Ctrl+X` / `Ctrl+V`	Copy / Cut / Paste
`Ctrl+A`	Select all (progressive: cell, channel, all)
`Ctrl+Delete`	Clear column
`Ctrl+Backspace`	Clear channel
`[` / `]`	Decrease / increase edit step

Transpose

Shortcut	Action
`Ctrl+F1`	Transpose down 1 semitone
`Ctrl+F2`	Transpose up 1 semitone
`Ctrl+F3`	Transpose down 1 octave
`Ctrl+F4`	Transpose up 1 octave

Channel

Shortcut	Action
`F9`	Mute/unmute channel
`F10`	Solo channel

Note Input (Edit Mode)

Notes are entered using a piano-style keyboard layout:

- **Lower octave** — `Z` `X` `C` `V` `B` `N` `M` (C D E F G A B), sharps on `S` `D` `G` `H` `J`.
- **Upper octave** — `Q` `W` `E` `R` `T` `Y` `U` `I` `O` `P` (C D E F G A B C D E), sharps on `2` `3` `5` `6` `7` `9` `0`.
- **Note off** — `1` in the note column.
- **Hex values** — `0–9`, `A–F` for instrument, volume, and effect columns.

Mini Notator

Mini Notator is the built-in MML (Music Macro Language) editor for composing music compatible with the Mega Drive's sound hardware. It provides a text-based approach to music composition with real-time audio preview, syntax highlighting, and visual feedback through an oscilloscope and piano roll.

Features

- **MML text editor** — Full code editor with syntax highlighting, auto-completion, find/replace, and real-time compilation.
- **13 sound channels plus 3 dummy channels** — 6 FM channels, 3 FM3-extended operator lanes, 3 PSG square channels, and 1 noise channel.
- **Real-time preview** — Hear changes as you type with WASM-based Genesis chip emulation.
- **FM instrument editor** — Visual algorithm diagram, feedback slider, and per-operator parameter controls.
- **PSG instrument editor** — Bar-based volume envelope editor.
- **DAC sample support** — Import WAV files for sample playback on FM channel 6.
- **Piano roll** — Visual note timeline color-coded by channel with playback tracking.
- **Oscilloscope and level meter** — Real-time audio visualization.
- **Import/export** — Import from Furnace, DefleMask, VGM, and FM instrument files. Export to VGM.
- **Undo/redo** — Full history support via the built-in code editor.

Interface Overview

Toolbar

The toolbar spans two rows at the top of the editor:

- **File operations** — Save and import from the editor; standalone mode also has New, Load, and VGM Export controls. Import supports Furnace, DefleMask, VGM, and FM instruments.
- **Transport controls** — Play (`F5`), Stop (`F8`).
- **Volume slider** — Adjustable from 0% to 200%.
- **Progress bar** — Shows playback position. Click to seek.

MML Editor

The central area is a full-featured code editor with:

- **Syntax highlighting** — Notes (teal), commands (green), directives (purple), channel letters (pink/bold), instrument numbers (blue).
- **Real-time compilation** — MML is compiled automatically as you type (300ms debounce). Errors appear as inline underlines with hover tooltips.
- **Standard editor features** — Undo/redo, find/replace, multi-cursor, bracket matching.

Bottom Panel

A resizable panel below the editor with four tabs:

- **FM** — Algorithm selector with visual routing diagram, feedback slider, and a 2x2 operator grid with sliders for all parameters.
- **PSG** — Bar-based volume envelope editor. Click bars to edit values.
- **DAC** — Load WAV files, preview waveforms, set loop points.
- **Visualizer** — Oscilloscope (left) and VU level meter (right), plus a piano roll showing note blocks color-coded by channel.

Channels

Mini Notator accepts channel letters A–P: 13 sound channels and 3 dummy channels.

Channel	Letter	Chip	Description
FM1–FM6	A–F	YM2612	6 FM synthesis channels. FM6 doubles as the DAC output for sample playback.
PSG1–PSG3	G–I	SN76489	3 PSG square wave channels.
Noise	J	SN76489	1 noise channel (white or periodic noise).
FM3 OP1–OP3	K–M	YM2612	Separate operator frequencies in FM3 extended mode.
Dummy	N–P	—	Dummy channels for macros.

Each channel is written on its own line, prefixed by its letter. Multiple channels can share commands:

```
A o4 l8 c d e f g a b > c4
G o5 l16 c e g > c < g e
ABC t120 v12    ; set tempo and volume on channels A, B, C
```

MML Syntax

Directives

Directives configure global song properties. They start with `#`:

```
#title "My Song"
#author "Composer"
```

Directive	Description
`#title`	Song title (used in VGM metadata)
`#author`	Composer name (used in VGM metadata)

To set tempo, use the inline `t` command on a channel (see Tempo and Timing).

Notes

Notes are lowercase letters `c d e f g a b` (`h` is an alias for `b`). Accidentals use `+` (sharp), `-` (flat), and `=` (natural).

```
A c d+ e f g- a b=
```

Symbol	Meaning
`c`–`b`	Note names (C through B)
`h`	Alias for B
`+`	Sharp
`-`	Flat
`=`	Natural (explicit)
`r`	Rest

Note Length

Note length is specified as a number after the note. Use `l` to set a default length. Append `.` for dotted notes (multiple dots are supported). Use `:N` for frame-based durations in ticks.

```
A l8 c d e f g4 a2. b1
A c:48 r:24       ; frame-based durations (48 and 24 ticks)
A c4.. d8.        ; double-dotted quarter, dotted eighth
A l:48 c d e      ; default length in ticks
```

Length	Duration
`1`	Whole note
`2`	Half note
`4`	Quarter note
`8`	Eighth note
`16`	Sixteenth note
`32`	32nd note
`.`	Dotted (1.5x duration, stackable: `..` = 1.75x)
`:N`	Frame-based duration (N ticks)
`lN`	Set default length to N
`l:N`	Set default length to N ticks

Octave

```
A o4 c d e > c < c
```

Command	Meaning
`oN`	Set octave (0–7)
`>`	Octave up
`<`	Octave down

Volume

```
A v12 c d v8 e f
```

Command	Range	Description
`vN`	0–15	Coarse volume
`VN`	0–255	Fine volume
`V+N` / `V-N`		Relative fine volume adjustment
`)` / `(`		Volume up / down by 1 (or `)N` / `(N` for N steps)

Note: `)` increases volume, `(` decreases volume (matching ctrmml convention).

Panning (FM Only)

```
A p3 c d p1 e f p2 g a
```

Value	Output
`p0`	Mute
`p1`	Right
`p2`	Left
`p3`	Center

Ties and Slurs

```
A c4 ^4      ; tie: extend C by another quarter note
A c4 ^:48    ; tie: extend by 48 ticks
A c4 & d4    ; slur/legato: connect C to D without re-triggering
```

Quantize

Controls how much of the note length is sustained before note-off.

```
A Q6 c d e f   ; sustain 6/8 of each note
A q2 c d e f   ; early release: cut 2 ticks before end
```

Command	Description
`QN`	Quantize (1–8, note length is N/8 of full duration; Q0 resets to full)
`qN`	Early release (1–8, ticks subtracted before note-off)

`Q` and `q` are mutually exclusive — setting one resets the other.

Portamento

```
A G10 c e g    ; slide between notes at speed 10
```

Command	Description
`GN`	Portamento speed (0 to disable)

Detune

```
A K5 c d e f   ; fine-tune up by 5
```

Command	Description
`KN`	Detune (-128 to +127)

Transpose

```
A _12 c d e f   ; transpose up 12 semitones (one octave)
```

Command	Description
`_N` or `kN`	Set transpose (-128 to +127)
`__N`	Set relative transpose (cumulative)

Key Signatures

```
A _{G} c d e f     ; G major (F is sharped)
A _{a} c d e f     ; A minor
A _{+fc} c d e f   ; sharpen F and C
A _{-be} c d e f   ; flatten B and E
```

Both ctrmml notation (`F+`, `B-`, `E-`) and standard notation (`F#`, `Bb`, `Eb`) are accepted.

Instruments

FM Instruments

FM instruments are defined with the `@N fm` directive followed by algorithm, feedback, and 4 operator rows:

```
@0 fm
  3 5
  31 10  5  7  2 25  0  1  3  0
  31 12  6  7  2 28  0  2  3  0
  31  5  5  7  1 35  0  1  7  0
  31  8  4  7  2 30  0  1  0  0
```

The first line is `algorithm feedback`. Each operator line contains: `AR DR SR RR SL TL RS ML DT SSG-EG`.

Parameter	Range	Description
Algorithm	0–7	Operator routing topology
Feedback	0–7	Self-modulation on operator 1
AR	0–31	Attack rate
DR	0–31	Decay rate
SR	0–31	Sustain rate
RR	0–15	Release rate
SL	0–15	Sustain level
TL	0–127	Total level (0 = loudest)
RS	0–3	Rate scaling
ML	0–15	Frequency multiplier
DT	-7 to +7	Detune
SSG-EG	0–15	SSG envelope mode (add 100 to enable AM)

Use `@N` in a channel line to select an instrument:

```
A @0 o4 c d e f
```

The FM tab in the bottom panel provides a visual editor for these parameters. Changes sync back to the MML source automatically.

2-Operator Chord Macro

Duplicates an FM instrument with modified operator multipliers and transpose:

```
@24 2op  2   5   5   4   4   0
;        ^   ^   ^   ^   ^   ^
;     base  ML1 ML2 ML3 ML4 TRS
```

PSG Instruments

PSG instruments define a volume envelope as a sequence of values (15 = max volume, 0 = silence):

```
@1 psg 1 4 6 8 10 12 13 14 15
```

Supports slides (`>`), holds (`:length`), sustain (`/`), and loop (`|`) markers.

The PSG tab provides a bar-based editor for the envelope.

DAC Samples

DAC samples are defined with the `@N pcm` directive:

```
@128 pcm "kick.wav" rate=18000 offset=4000
```

Parameter	Description
`rate=N`	Override sample rate (Hz)
`offset=N`	Start position offset (samples)

Envelopes

Define reusable pitch, volume, or pan envelopes with `@M`, `@E`, or `@P`:

```
@M1 -2>0:20 0         ; pitch slide from -2 semitones over 20 frames
@M2 0:20 | V0:1:5     ; vibrato with 20-frame delay
@E1 15 14 / 13>0:7    ; volume fade with sustain point
@P1 | 3 1 3 2         ; pan envelope cycling L/R
```

Command	Description
`MN`	Activate pitch envelope @M (0 = disable)
`EN`	Activate volume envelope @E (0 = disable)
`PN`	Activate pan envelope/macro track @P (0 = disable)

Loops and Flow Control

Repeat Loops

```
A [c d e f]4        ; repeat 4 times
A [c d e f / g a]3  ; last iteration plays g a instead of looping back
```

Song Loop Point

The `L` command marks where the song loops back to in VGM export:

```
A o4 c d e f
A L
A g a b > c
```

Macros (Subroutines)

Define reusable sequences on a numbered track with `*N` at the start of a line, and call them with `*N` inline:

```
*32 c8 d8 e8 f8
A o4 *32 *32 g4
```

Drum Mode

Maps note letters to macro calls for percussion:

```
A D20 c d e f    ; c=macro *20, d=*21, e=*22, f=*23...
A D0             ; disable drum mode
```

Echo

```
A \=2,3 c4 d \4 e \4    ; echo with 2-note delay, volume -3
```

Command	Description
`\=N,V`	Set echo delay (N notes back) and volume reduction (V). Positive N clears buffer; negative preserves it.
`\[duration]`	Insert echo note with optional explicit duration

Conditional Blocks

Apply different commands per channel when using multi-channel addressing:

```
ABC {c/d+/g} {d/f/a}    ; A gets c,d — B gets d+,f — C gets g,a
```

Grace Notes and Reverse Rests

```
A ~c16 d4     ; grace note C before D
A c4 R8 d4    ; borrow time from C for sync
```

Tempo and Timing

```
A t150 c d e f     ; set tempo to 150 BPM
A C96 c d e f      ; set measure length to 96 ticks (default)
A s6 c d e f       ; set shuffle offset to 6 ticks
```

Command	Description
`tN`	Set tempo in BPM (0–255)
`TN`	Set tempo using native hardware timer value (0–255)
`CN`	Set measure length in ticks (default 96)
`sN`	Set shuffle offset in ticks (signed, alternates +/- per note)

Platform Commands

Hardware-specific commands enclosed in apostrophes:

```
A 'lfo 1 5'         ; set AM sensitivity 1, PM sensitivity 5
A 'lforate 3'       ; set global LFO rate
A 'fm3 0011'        ; enable FM3 extended mode (operators 1-2)
A 'write 0xB0 0x3A' ; direct FM register write
A 'tl1 +5'          ; adjust operator 1 total level
A 'mode 1'          ; PSG noise: use tone channel 3 as source
A 'pcmrate 4'       ; set PCM pitch rate
A 'pcmmode 3'       ; set 3-channel PCM mixing mode
A 'carry'           ; macro track carry mode
```

Command	Description
`'lfo AMS PMS'`	Set per-channel AM/PM sensitivity (0–3, 0–7)
`'lforate N'`	Set global LFO rate (0 = off, 1–9)
`'fm3 MASK'`	FM3 extended mode (binary mask e.g. `0011`, or `1111`/`0` to disable)
`'mode N'`	PSG noise mode (0–1)
`'pcmrate N'`	PCM pitch rate (1–8)
`'pcmmode N'`	PCM mixing mode (2 = 2ch@17.5kHz, 3 = 3ch@13kHz)
`'write REG DATA'`	Direct FM register write
`'tl1-4 [±]VAL'`	Set/adjust operator total level
`'dtml1-4 VAL'`	Set operator DT/MUL register
`'ksar1-4 VAL'`	Set operator KS/AR register
`'amdr1-4 VAL'`	Set operator AM/D1R register
`'sr1-4 VAL'`	Set operator sustain rate register
`'slrr1-4 VAL'`	Set operator SL/RR register
`'ssg1-4 VAL'`	Set operator SSG-EG register
`'fbal VAL'`	Set feedback/algorithm register
`'carry'`	Macro track carry mode (don't reset on new note)

Import and Export

Import Formats

Format	Extensions	Description
MML	.mml, .txt	Direct text load
Furnace	.fur	Zlib-compressed tracker format. Supports Genesis 10-channel and Ext CH3 13-channel configurations.
DefleMask	.dmf	Zlib-compressed tracker format. Supports Genesis configurations.
VGM	.vgm	Imports register data into a song.
FM Instruments	.tfi, .tyi, .y12, .opm	Individual instrument files inserted into the song.

You can also drag and drop files onto the editor to import them.

Export Formats

Format	Extension	Description
MML	.mml	Human-readable text file
VGM	.vgm	Video Game Music binary with GD3 metadata and loop point support

Keyboard Shortcuts

Playback

Shortcut	Action
`F5` / `Ctrl+Enter`	Play from start
`F8`	Stop
`Left` / `Right`	Seek ±1 second
`Shift+Left` / `Shift+Right`	Seek ±5 seconds
`Home`	Seek to beginning
`End`	Seek to end

File

Shortcut	Action
`Ctrl+N`	New
`Ctrl+S`	Save
`Ctrl+O`	Open

Editor

Shortcut	Action
`Ctrl+Z`	Undo
`Ctrl+Y`	Redo
`Ctrl+C`	Copy
`Ctrl+X`	Cut
`Ctrl+V`	Paste
`Ctrl+A`	Select all
`Ctrl+F`	Find
`Ctrl+H`	Find and replace
`F1` / `?`	Toggle shortcuts help

Mini Notator vs. Mini Tracker

Both tools compose music for the same Mega Drive hardware, but take different approaches:

	Mini Notator	Mini Tracker
Input	Text-based MML code	Grid-based pattern editor
Workflow	Type commands, hear changes instantly	Fill cells in a pattern grid
Best for	MML-experienced composers, scripted music, fine control over commands	Visual composers, tracker-style workflows
Instruments	Text definitions with visual editor for parameters	Full GUI editor with presets
Effects	Inline MML commands	Dedicated effect columns per cell

Both share the same underlying audio engine and produce output compatible with the XGM2 sound driver.

Bfxr

Bfxr is the built-in sound effects generator for creating retro-style sound effects. It can generate a variety of sounds such as jumps, explosions, power-ups, and more, exporting them as WAV files compatible with the XGM2 sound driver. The current editor exposes one Bfxr synth tab with 12 waveform types and preset generators for quickly creating game-ready sounds.

Features

- **Preset generators** — One-click generation of common game sounds: pickup, laser, explosion, powerup, hit, jump, and blip.
- **Classic chiptune synth** — Bfxr synth engine for retro-style sound generation.
- **12 waveform types** — Square, sawtooth, sine, triangle, white noise, and more exotic options like FM synthesis and voice.
- **30+ parameters** — Fine control over pitch, envelope, vibrato, filters, bit crush, flanging, overtones, and pitch jumps.
- **Randomize and mutate** — Generate entirely random sounds or incrementally tweak existing ones. Parameter locks prevent specific values from changing.
- **Real-time preview** — Hear changes immediately with a waveform display.
- **Batch export** — Export all sounds as a ZIP file of WAV files.
- **File management** — Save/load individual sounds (.bfxr) or entire collections (.bcol).

Interface Overview

The interface is divided into three panels per synth tab:

Left Panel

- **Preset generators** — Buttons for each sound type (Pickup, Laser, Explosion, etc.) plus Randomize and Mutate.
- **Create New Sound** — When enabled, generators create a new file instead of overwriting the current one.
- **Sound management** — Apply (commit edits), Revert (discard edits), Duplicate (clone sound).
- **File list** — All saved sounds in the current synth tab. Double-click to rename. Modified sounds show an indicator.

Center Panel

- **Waveform selector** — Choose from 12 oscillator types.
- **Parameter sliders** — All synth parameters with labels, ranges, and lock buttons. Locked parameters are excluded from randomization and mutation.

Right Panel

- **Waveform display** — Visual representation of the generated sound.
- **Play on Change** — Automatically play the sound when parameters are modified.
- **Play button** — Manual playback.
- **Volume slider** — Master volume control.
- **Save button** — Export current sound as WAV.

Bfxr Synth

The synth supports all 12 waveform types and the full parameter set.

Preset generators:

Generator	Description
Pickup/Coin	Short upward pitch sweep
Laser/Shoot	Downward frequency sweep
Explosion	Low-frequency noise burst with flanging
Powerup	Ascending pitch with optional vibrato
Hit/Hurt	Noise-based impact with short decay
Jump	Upward square/saw burst
Blip/Select	Short bright tone for UI feedback
Randomize	Fully random parameters (respects locks)
Mutate	Small tweaks to the current sound (respects locks)

Waveform Types

#	Waveform	Description
0	Square	Classic chiptune sound. Supports pulse width (duty cycle) and duty sweep.
1	Sawtooth	Bright and buzzy with rich harmonics.
2	Sine	Pure tone with no harmonics.
3	White Noise	Full-spectrum noise for explosions and impacts.
4	Triangle	Clear and resonant, between sine and square.
5	Rasp	Periodic 1-bit noise with a digital buzz.
6	Tan	Hyperbolic tangent curve producing distorted, unpredictable sounds.
7	Whistle	Sine with a high harmonic overlay for a breathy quality.
8	Breaker	Quadratic waveshaper producing smooth, hi-fi tones.
9	Bitnoise	SN76489-style periodic noise for authentic console sounds.
10	FM Synth	FM synthesis using AKWF wavetables for dense, complex timbres.
11	Voice	Sampled vocal-like waveform using AKWF wavetables.

Parameters

Envelope

Parameter	Range	Description
Attack Time	0–1	Duration of the volume ramp-up
Sustain Time	0.01–1	Duration at peak volume
Sustain Punch	0–1	Extra volume boost during the sustain phase
Decay Time	0.03–1	Duration of the fade-out

Pitch

Parameter	Range	Description
Start Frequency	0–1	Base pitch of the sound
Frequency Slide	-0.5–0.5	Pitch slide up or down over time
Frequency Acceleration	-1–1	Acceleration applied to the pitch slide
Min Frequency	0–0.99	Floor frequency — sound cuts off below this

Vibrato

Parameter	Range	Description
Vibrato Depth	0–1	Strength of pitch modulation
Vibrato Speed	0–1	Speed of pitch oscillation

Pitch Jumps

Parameter	Range	Description
Pitch Jump Amount	-1–1	First pitch jump magnitude
Pitch Jump Onset	0–1	When the first jump occurs (fraction of sound duration)
Pitch Jump 2 Amount	-1–1	Second pitch jump magnitude
Pitch Jump 2 Onset	0–1	When the second jump occurs
Pitch Jump Repeat	0–1	How often jumps repeat (0 = single, 1 = rapid)

Square Wave

These parameters only apply when the Square waveform is selected:

Parameter	Range	Description
Square Duty	0–0.99	Pulse width (0.5 = standard square)
Duty Sweep	-1–1	Sweep the pulse width over time

Overtones

Parameter	Range	Description
Overtones	0–1	Number of harmonic layers added to the sound
Overtone Falloff	0–1	How quickly harmonics decay in amplitude

Repeat

Parameter	Range	Description
Repeat Speed	0–1	Rate of note retriggering (0 = no repeat)

Flanger

Parameter	Range	Description
Flanger Offset	-1–1	Phase offset amount
Flanger Sweep	-1–1	Sweep of the phase offset over time

Filters

Parameter	Range	Description
LP Filter Cutoff	0.01–1	Low-pass filter cutoff frequency
LP Filter Cutoff Sweep	-1–1	Sweep the low-pass cutoff over time
LP Filter Resonance	0–1	Resonance peak at the cutoff frequency
HP Filter Cutoff	0–1	High-pass filter cutoff frequency
HP Filter Cutoff Sweep	-1–1	Sweep the high-pass cutoff over time

Bit Crush

Parameter	Range	Description
Bit Crush	0–1	Sample rate / bit depth reduction
Bit Crush Sweep	-1–1	Sweep the bit crush effect over time

Dynamics

Parameter	Range	Description
Compression	0–1	Dynamic range compression for added punch

Randomization and Mutation

Randomize

Generates a fully random sound by resetting all unlocked parameters to random values. The randomization uses weighted distributions (power curves) to produce more musically useful results — for example, attack times are biased toward short values, and filter cutoffs are biased toward extremes.

Mutate

Applies small, incremental tweaks to the current sound. Each unlocked parameter has a 50% chance of being nudged by up to 10% of its range. The waveform type has a 10% chance of shifting to an adjacent type.

Parameter Locks

Click the lock icon next to any parameter to prevent it from being changed by Randomize or Mutate. This lets you explore variations while keeping specific aspects of a sound fixed.

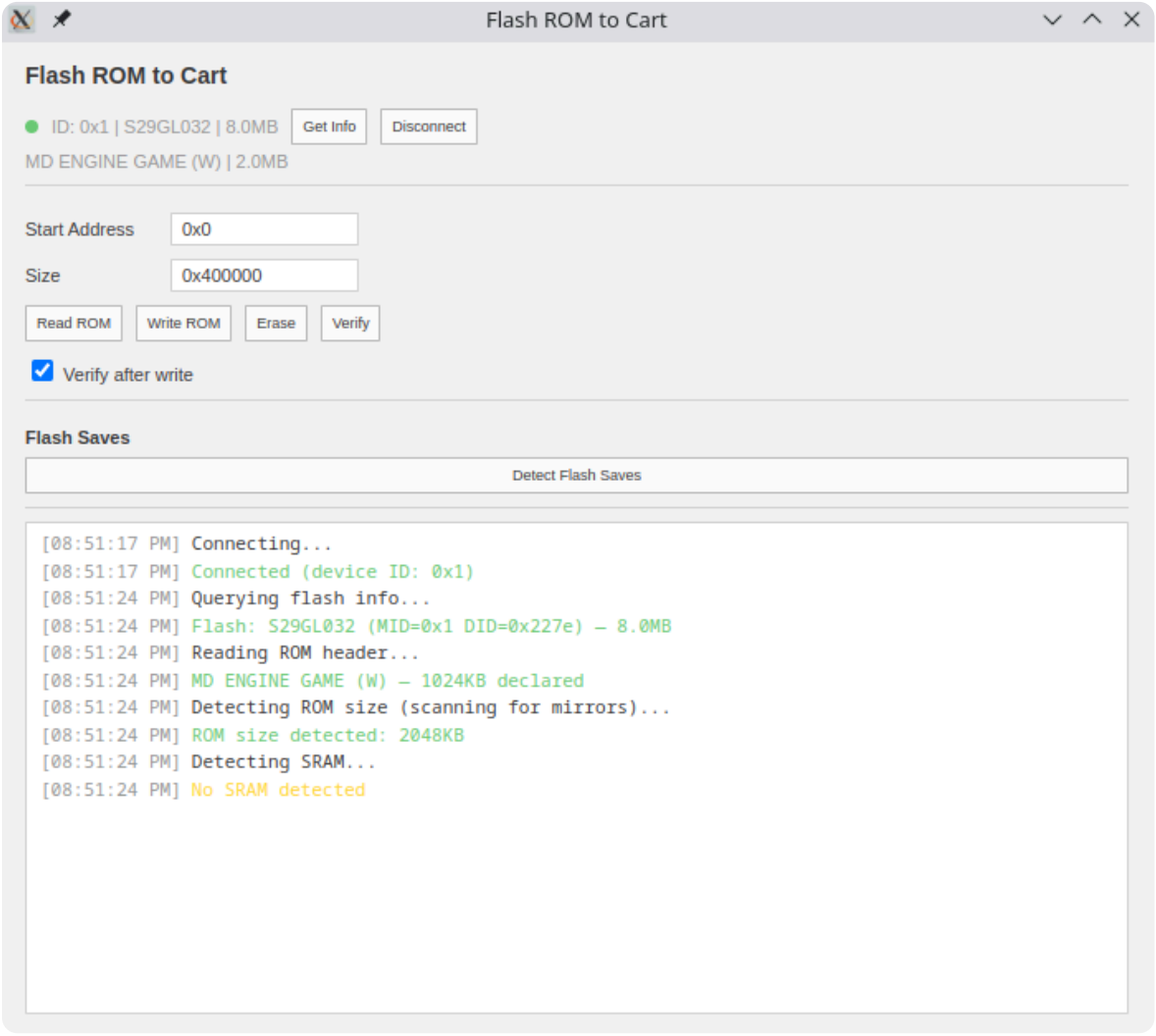
Keyboard Shortcuts

Shortcut	Action
`Space` / `Enter`	Play current sound
`Ctrl+S`	Save current sound as WAV
`Ctrl+C`	Copy sound parameters to clipboard
`Ctrl+V`	Paste sound parameters from clipboard
`L`	Toggle lock on all parameters

Flash ROM to Cart

Flash ROM to Cart is the built-in tool for programming FlashKit-MD cartridges directly from the desktop editor. It communicates with Krikzz's FlashKit-MD hardware over USB serial.

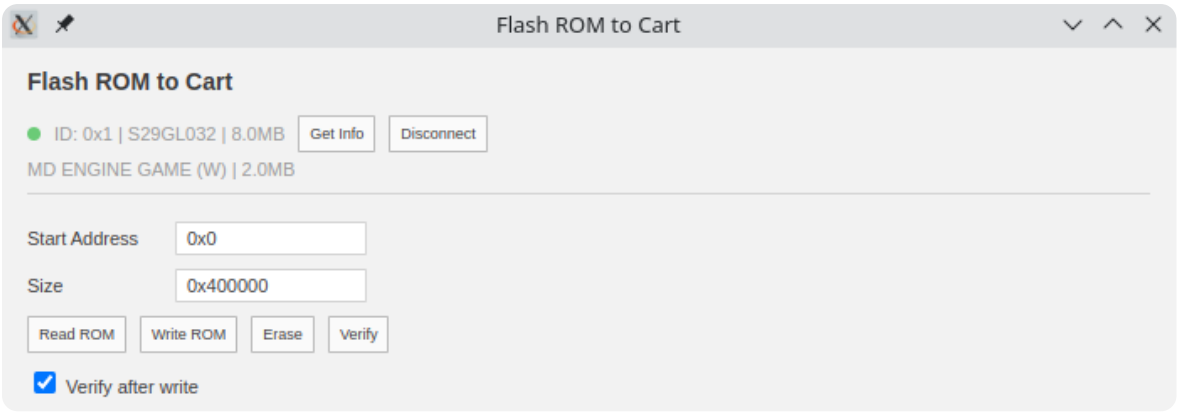
Use it to flash your game to a cartridge for testing on real hardware, read/write SRAM save data, and manage flash saves.



Getting Started

1. Connect a FlashKit-MD device to your computer via USB.
2. Open the tool from **Game > Flash ROM to Cart**.
3. Click **Connect** — the editor scans available serial ports and tries to connect to the FlashKit-MD.
4. Once connected, the connection bar shows the device ID. Click **Get Info** to show flash chip details.

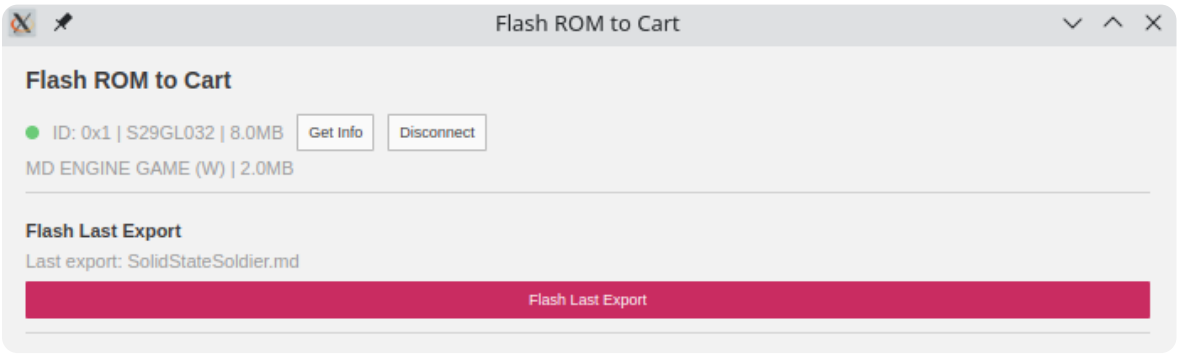
Click **Get Info** to query the flash chip. If the chip is in the known database, its name is shown (e.g. "S29GL032 — 4.0MB") instead of raw hex IDs.



Flash Last Export

After exporting a ROM from the editor (**Game > Export As > ROM**), the Flasher automatically detects the exported file. A **Flash Last Export** button appears showing the ROM filename — click it to flash the ROM to the connected cartridge in one step.

This is the fastest way to iterate: export → flash → test on hardware.



ROM Operations

The ROM operations section provides low-level access to the flash cart:

Operation	Description
Read ROM	Dump the cartridge contents to a `.bin` file. Specify the size and start address.
Write ROM	Flash a `.bin` file to the cartridge. Erases the target region first, then programs using AMD Unlock Bypass mode.
Erase	Erase a region of flash (rounded to sector boundaries).
Verify	Read back the cartridge and compare against a file byte-by-byte.
Cancel	Cancel any running operation.

The **Verify after write** checkbox (enabled by default) automatically verifies the data after a Write ROM operation.

Cart Info

- **Read Header** — reads the Mega Drive ROM header and shows the game name, region code (J/U/E/W), and declared ROM size.
- **Detect Size** — auto-detects the actual ROM size by scanning for address mirroring. This avoids dumping 4 MB of repeated data from a smaller ROM.

Flash Chip Identification

When you click **Get Info**, the tool queries the flash chip via CFI (Common Flash Interface) and looks up the manufacturer/device IDs in a built-in database. Recognized chips include models from AMD/Spansion, Macronix, Winbond, ST/Numonyx, SST, and others commonly found in flash cartridges.

SRAM

If the cartridge has battery-backed SRAM or FRAM, the Flasher can read and write save data.

1. Click **Detect SRAM** to check for SRAM and determine its size.
2. **Read SRAM** downloads the save data in **emulator-compatible format** (stripped — only the actual data bytes, not the raw 16-bit bus data). The resulting `.srm` file works directly with emulators.
3. **Write SRAM** accepts either format:
   - **Stripped** (emulator `.srm` files) — automatically converted to raw bus format before writing.
   - **Raw** (bus format, 2x the data size) — written as-is. The format is auto-detected based on file size vs. detected SRAM size.

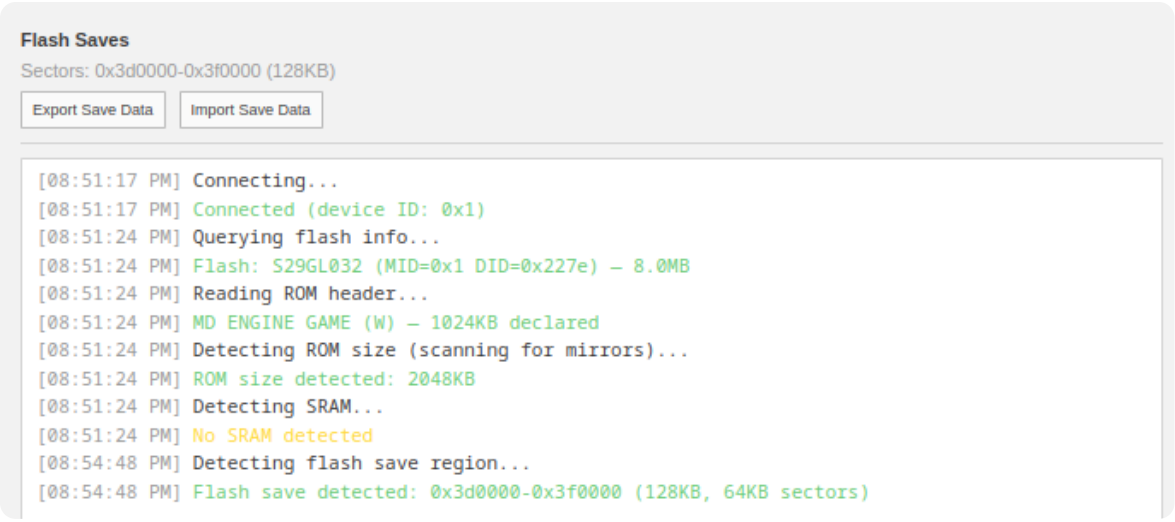
After writing, the data is automatically verified (only the SRAM data bytes are compared — the garbage high bytes on the 16-bit bus are ignored).

For more information on SRAM saving, see the Save System documentation.

Flash Saves

Games built with **flash saving** (configured in Settings > Cartridge) store save data directly in the ROM's flash chip. The Flasher can detect and manage these save regions.

1. Click **Detect Flash Saves** to read the ROM header and locate the save sectors.
   - ROMs with `"FL"` in the header use explicitly defined sector addresses.
   - ROMs with `"RA"` auto-detect the last two equal-size sectors from the flash chip's CFI data.
2. After detection, the tool shows the sector addresses and total save region size.



Export / Import Save Data

- **Export Save Data** — downloads both save sectors as a single binary file. Use this to back up save data before flashing a new ROM.
- **Import Save Data** — erases the save sectors and programs the data from a previously exported file. The file size must match the save region exactly.

Save Slots

The tool can parse the SGDK flash save format used by the engine:

- Walk the blob chain in the active sector
- List which slots have data and their sizes
- Read individual slot data

For more information on flash saving, see the Save System documentation.

Troubleshooting

Device not detected

- Make sure the FlashKit-MD is connected via USB and powered on.
- Try a different USB cable or port.
- On **macOS**, no additional drivers are typically needed for FTDI-based devices.

Serial port permission denied (Linux)

On Linux, the error `Failed to open serial port: FILE_ERROR_ACCESS_DENIED` means your user doesn't have permission to access the USB serial device. There are two ways to fix this:

Option 1: Add your user to the `dialout` group (recommended)

```
sudo usermod -aG dialout $USER
```

You must **log out and back in** (or reboot) for the group change to take effect.

Option 2: Install a udev rule (no logout required, applies system-wide)

Create a file at `/etc/udev/rules.d/99-flashkit.rules` with the following content:

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", MODE="0666"
```

Then reload the rules and re-trigger device detection:

```
sudo udevadm control --reload-rules && sudo udevadm trigger
```

This grants all users read/write access to FTDI USB serial devices (vendor ID `0403`), which is what the FlashKit-MD uses. The rule takes effect immediately for newly plugged devices — no logout needed.

> **FUTURE IMPROVEMENT**
>
> The udev rule could be bundled with the Linux DEB/RPM installer packages so users don't need to configure permissions manually. The rule file would be installed to `/etc/udev/rules.d/` as part of the package post-install step.

Erase or write errors

- Ensure the cartridge is firmly seated in the FlashKit-MD.
- If an operation fails mid-write, the flash may be in an undefined state. Try erasing the affected region before writing again.
- Check the log output for specific error codes.

Connection fails

Check the Flasher log for the serial ports it tried and any errors. If it reports that no serial ports were found, check that your computer recognizes the connected FlashKit-MD. On Linux, a permission error may require the serial port permission steps above.

Scripting

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 to 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 favorites, 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 Start** - This script will be called as soon as the scene is loaded in the game. You can use this to do things like playing music as the scene loads, configuring events to happen on button presses, initializing actors based on the values of variables, and much more.
- **On Update** - This script will be called every frame before the *Actor* and *Trigger* **On Update** scripts, this can control most of your scene logic, like when to pause the game, or background effects, and much more.
- **On End** - This script will be called as soon as the scene is unloaded in the game. You can use this to pause music, do a fade out, save score if needed, and much more.
- **On Pause** - This script will be called instead of the *On Update* when the game is paused, you can place your pause update logic here.

Actor Scripts

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

- **On Start:** Called as soon as the scene is loaded in the game.
- **On Update:** (only if Update Mode is set to Parallel) Repeatedly called every frame, once the script finishes it will repeat. You can use this to create movement scripts.
- **On Interact:** (only if Update Mode is set to Interact) This is called if an actor stands in front of this actor and the *Actor Interact* event is called.
- **On End:** This script will be called as soon as the scene is unloaded in the game.
- **On Collide:** (only if a collision group is set) Called when this actor collides with an actor or projectile in a group enabled by the collision mask. This can include actors in the same group.

Trigger Scripts

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

- **On Enter:** Called as soon as an actor enters into the trigger area.
- **On Update:** Repeatedly called every frame, while the overlap is still going on.
- **On Exit:** Called as soon as an actor exits the trigger area.

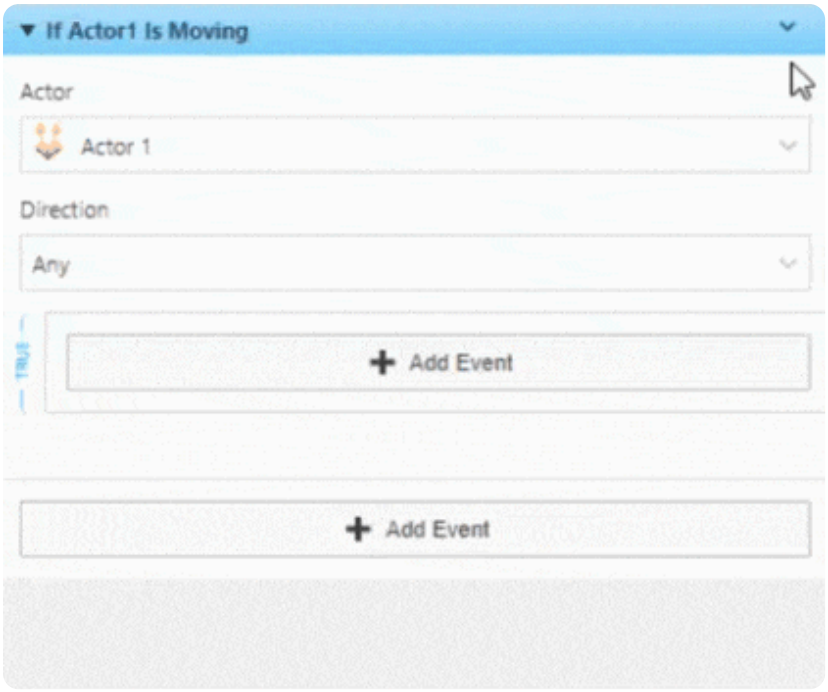
Control Flow

MD Engine has many events that allow you to control the flow of the script, such as check If Actor Is Moving, or If Actor On Screen, etc.

Event inversion

Most of the control flow events can be inverted to check for the opposite condition, such as *If Actor Is **NOT** Moving* or *If Actor **NOT** On Screen*.

To invert an event, check the down arrow and select *Invert Event* if present, that would invert the event and check for the opposite condition.



Else

If you want to have code for when a script condition is false, you can enable the Else part with *Enable Else* in the event drop down, which is right under *Invert Event*.

Event Glossary

Events are separated into different categories.

Here you can find a list of all of those categories.

Actors

Events for controlling actor movement, position, properties, sprites, and animations.

Array

Events for working with arrays, including reading, writing, and manipulating array data.

Behaviours

Events for enabling and controlling built-in actor behaviours such as platformer movement and tile-based movem…

Camera

Events for controlling the camera position, movement, and screen effects like shake and flash.

Collisions & Overlaps

Events for detecting and responding to collisions and overlaps between actors.

Control Flow

Events for controlling script execution flow, including conditionals, loops, and script calls.

Dialogue, Text & Menus

Events for displaying dialogue, text, and menu choices.

Engine Fields

Events for reading and updating engine field values at runtime.

Graphics

Events for controlling graphical elements such as palettes, overlays, and background layers.

Input

Events for detecting button presses and player input.

Math

Events for performing mathematical operations and evaluating expressions.

Miscellaneous

Events for miscellaneous operations such as comments, threads, and system functions.

Music & Sound Effects

Events for playing and controlling music and sound effects.

Particles

Events for setting up and spawning particle effects.

Pathfinding

Events for computing and following paths on the scene collision map.

Platform

Events for detecting the current platform and toggling platform-specific features.

Save Data

Events for saving and loading game data.

Scene

Events for managing scenes, transitions, scrolling, tiles, and scene properties.

Screen

Events for controlling screen-level settings such as VDP configuration, HBlank/VBlank scripts, and fade effects.

Temp. Triggers & Projectiles

Events for spawning temporary triggers and projectiles.

Timer

Events for setting up and controlling timers.

Tween

Events for creating smooth value transitions using tween animations.

Variables

Events for reading, writing, and manipulating variables and flags.

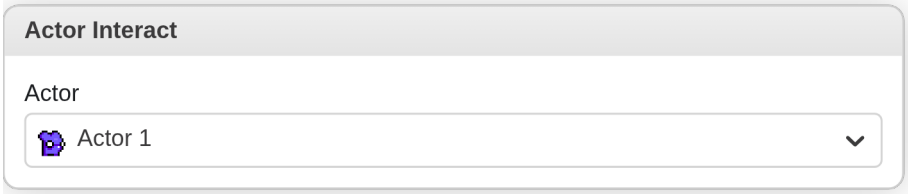
Actors

Events for controlling actor movement, position, properties, sprites, and animations.

Actions

Actor Interact

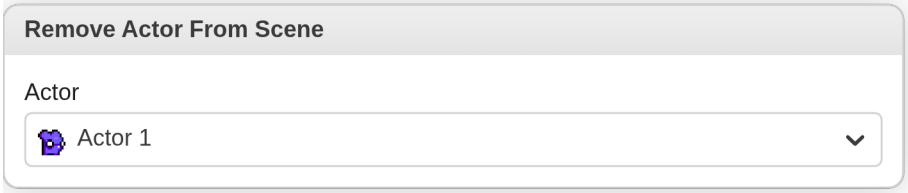
The actor tries to interact with what is in front of them, it uses actor direction.



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

Remove Actor From Scene

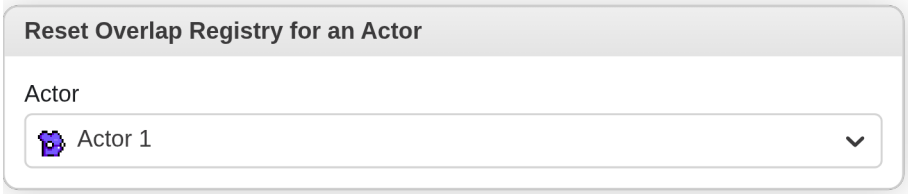
Removes the actor from the scene, this disables everything and make it not collide, but this DOESN'T free the RAM until scene is not finished.



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

Actor Overlaps Reset

Resets the first-time overlap tracking for an actor, allowing all On Collide callbacks to fire again even if the actor is still touching other actors.

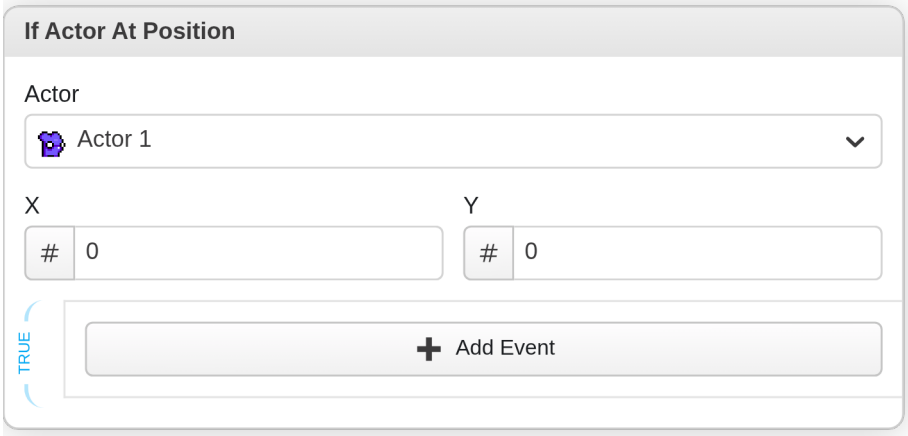


- **Actor**: The actor you want to update.

Control Flow

If Actor At Position

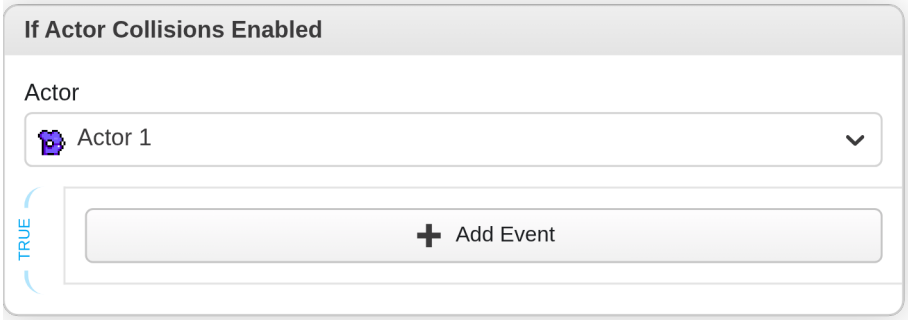
Conditionally run part of the script if an actor is at a specified position.



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

If Actor Collisions Enabled

Conditionally run part of the script if an actor is collisions are enabled.

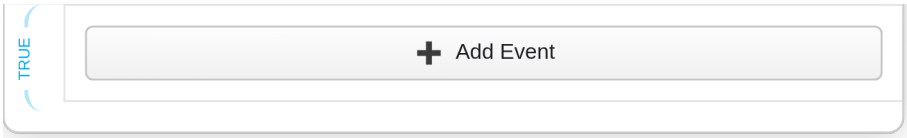


- **Actor**: The actor you want to check.

If Actor Facing Direction

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

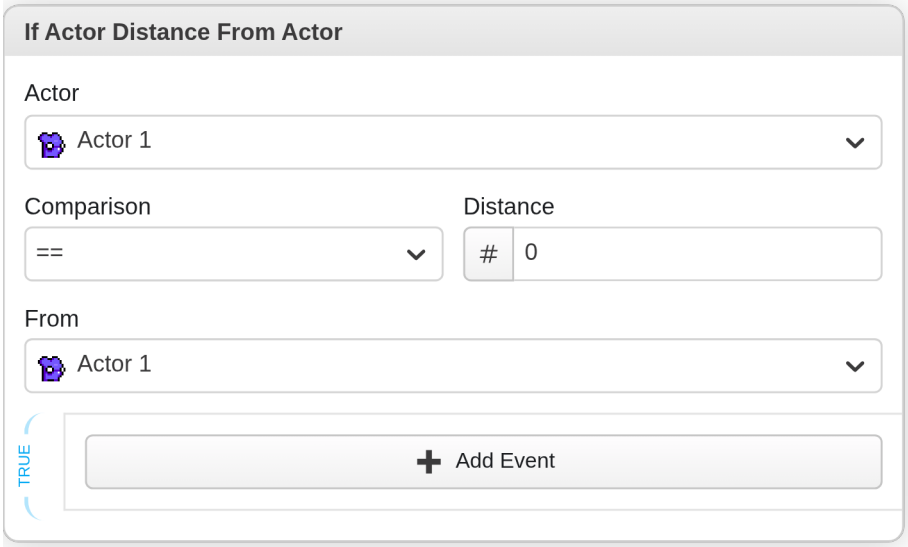




- **Actor**: The actor you want to check.
- **Direction**: The actor direction.

If Actor Distance From Actor

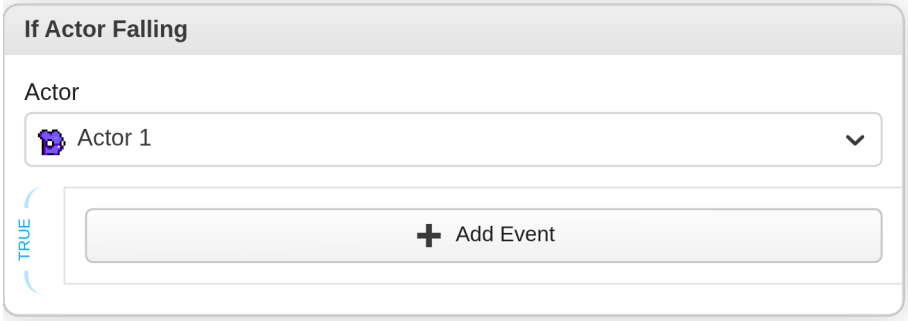
Conditionally run part of the script if an actor is within a certain distance of another actor.



- **Actor**: The actor you want to check.
- **From**: The actor to compare distance with.
- **Distance**: The distance value.

If Actor Falling

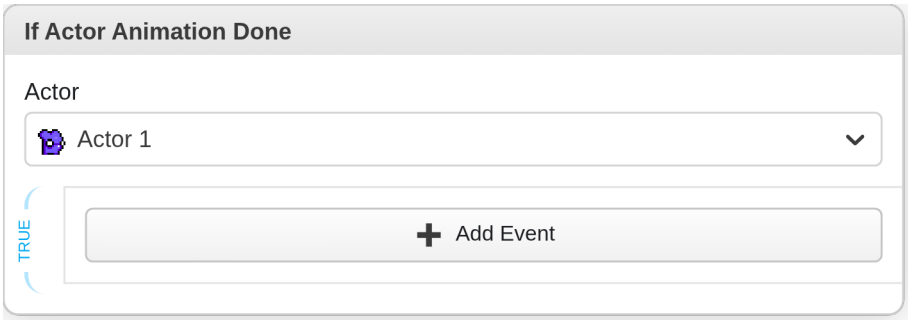
Conditionally run part of the script if an actor is falling (It's the same as velocityY > 0).



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

If Actor Animation Done

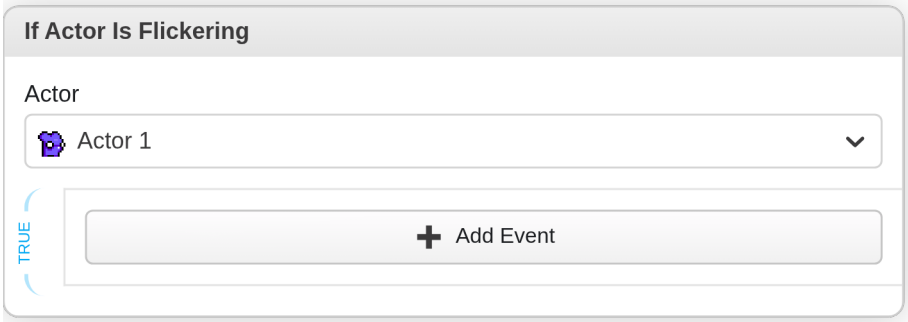
Conditionally run part of the script when an actor's non-looping animation has finished playing. Useful for chaining animations (e.g., play attack → wait for done → return to idle) or triggering events on death / spawn completion.



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

If Actor Is Flickering

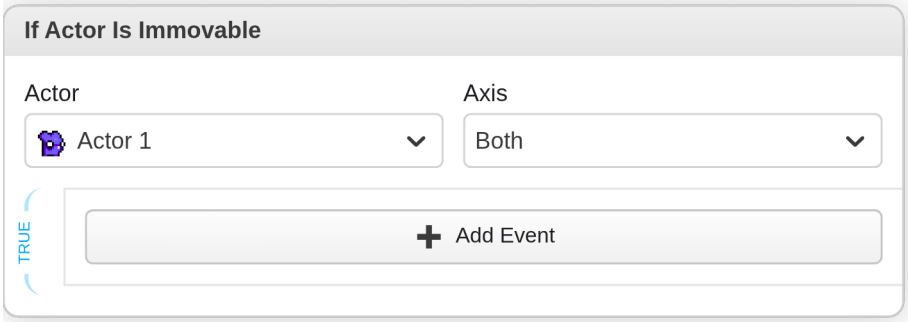
Conditionally run part of the script while an actor's flicker effect is still active.



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

If Actor Is Immovable

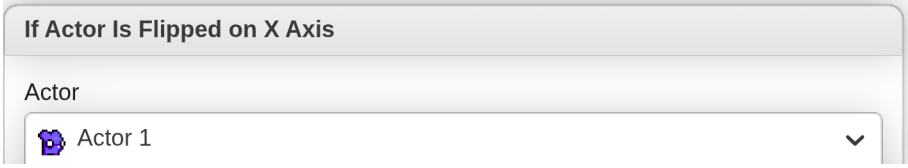
Conditionally run part of the script if an actor is immovable on the chosen axis (X only, Y only, or both axes must be immovable).

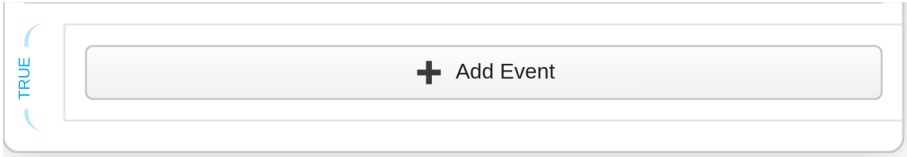


- **Actor**: The actor you want to check.
- **Axis**: Which axis to check (X only, Y only, or both axes must be immovable).

If Actor Is Flipped on X Axis

Conditionally run part of the script if an actor sprite is flipped on the X axis.

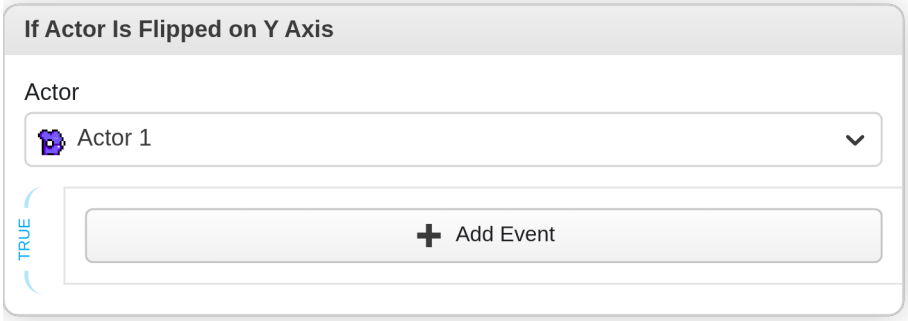




- **Actor**: The actor you want to check.

If Actor Is Flipped on Y Axis

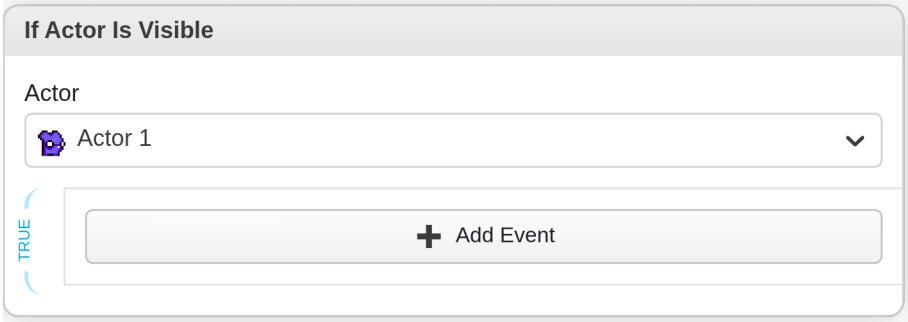
Conditionally run part of the script if an actor sprite is flipped on the Y axis.



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

If Actor Is Visible

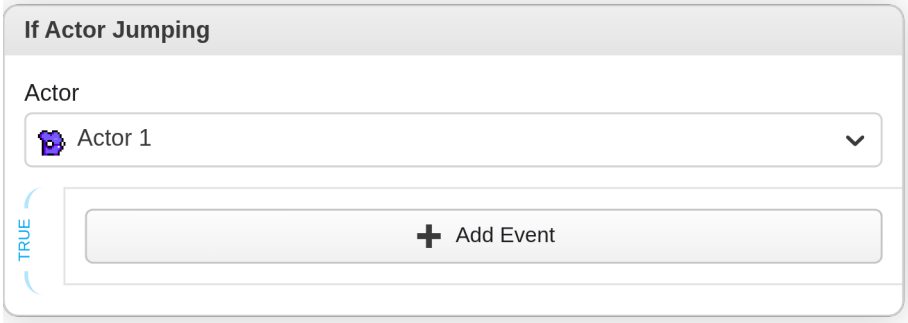
Conditionally run part of the script if an actor sprite is visible.



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

If Actor Jumping

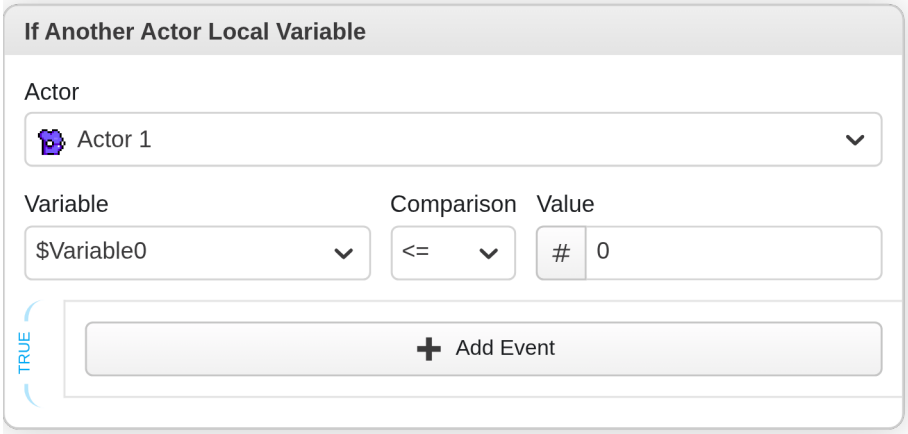
Conditionally run part of the script if an actor is jumping (It's the same as velocityY < 0).



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

If Another Actor Local Variable

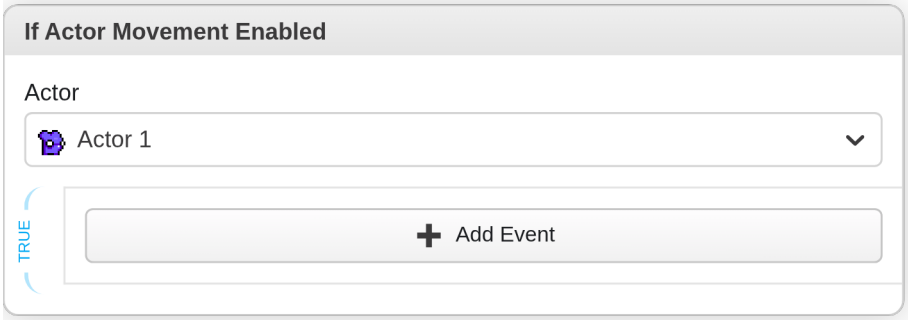
Conditionally run part of the script if another actor specified local variable is what we want.



- **Actor**: The actor you want to check.
- **Variable**: The variable to use.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **Value**: The value to set the selected variable to.

If Actor Movement Enabled

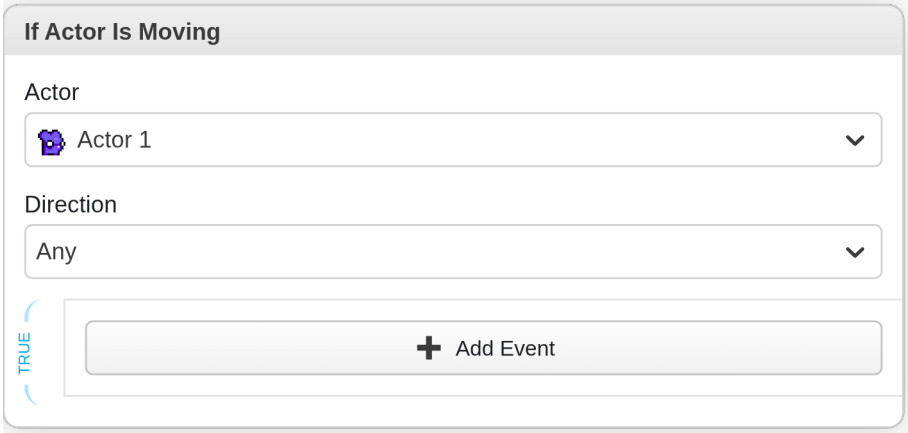
Conditionally run part of the script if an actor is movement is enabled.



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

If Actor Is Moving

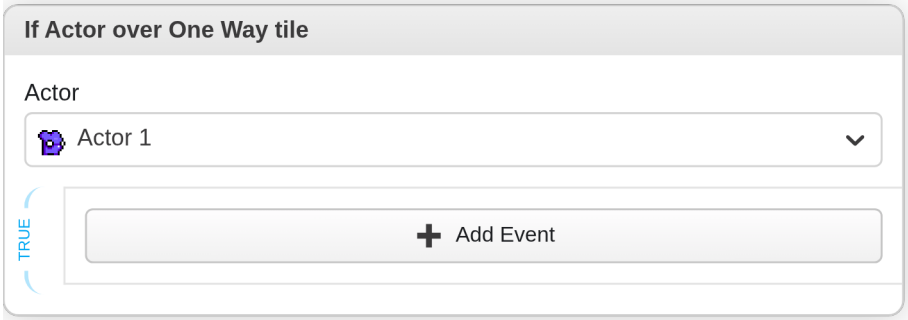
Conditionally run part of the script if an actor is moving.



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

If Actor over One Way tile

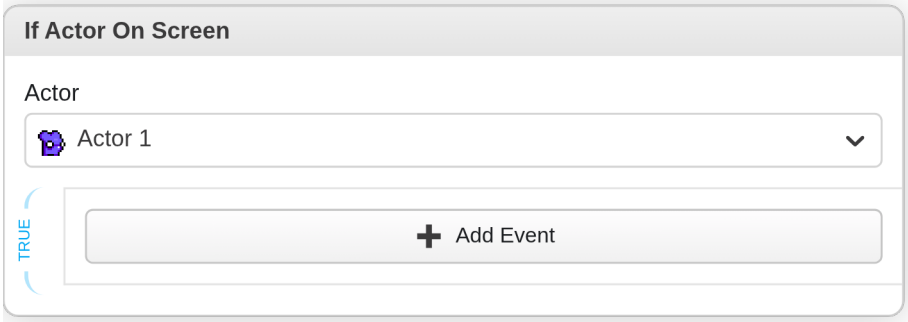
Conditionally run part of the script if the actor is on top of a ONE WAY tile (blue collision).



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

If Actor On Screen

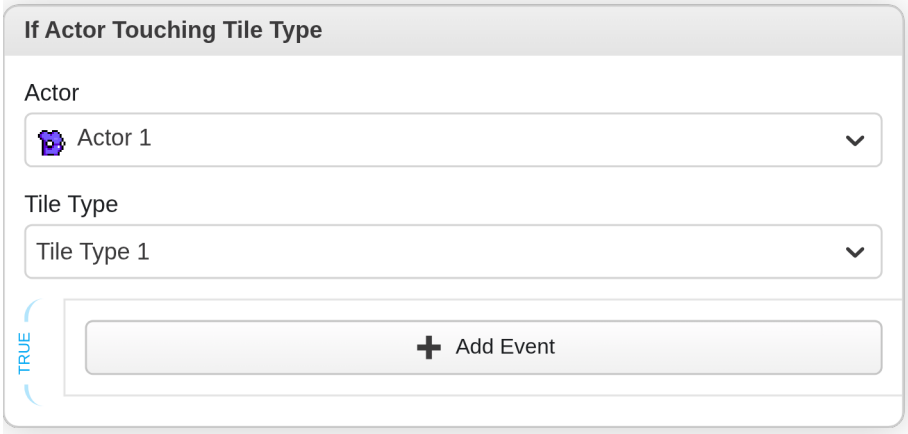
Conditionally run part of the script if the actor is on screen (accounts for scrolling if any).



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

If Actor On Tile Type

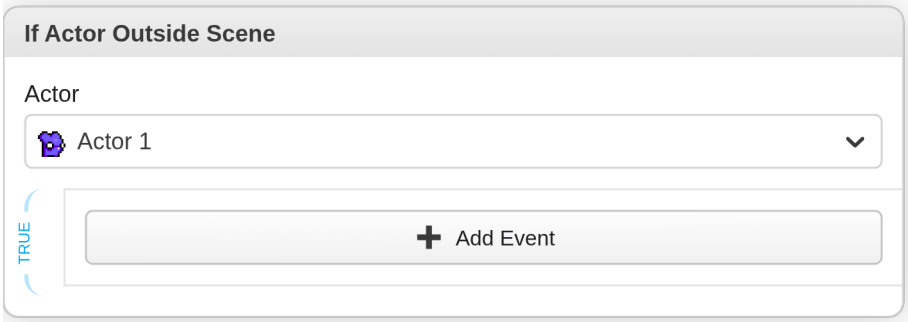
Conditionally run part of the script if the actor is touching a tile of the selected type.



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

If Actor Outside Scene

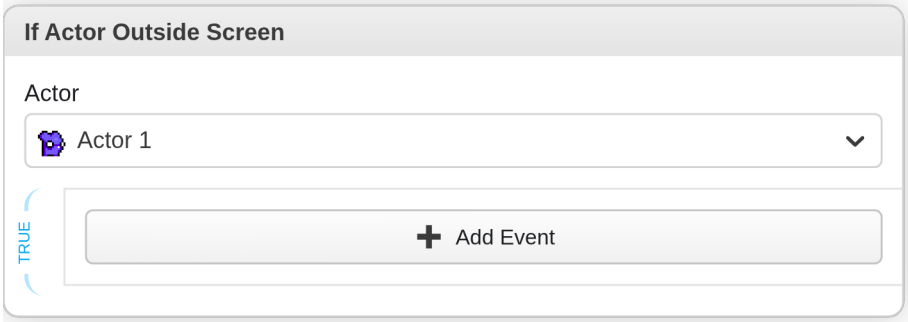
Conditionally run part of the script if the actor is outside scene.



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

If Actor Outside Screen

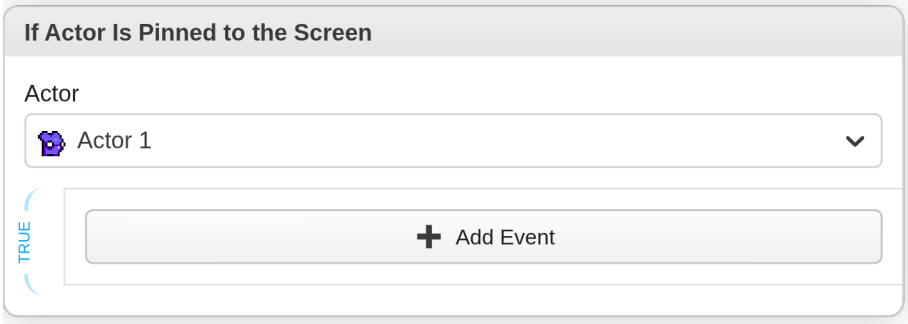
Conditionally run part of the script if the actor is outside screen (accounts for scrolling if any).



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

If Actor Is Pinned to the Screen

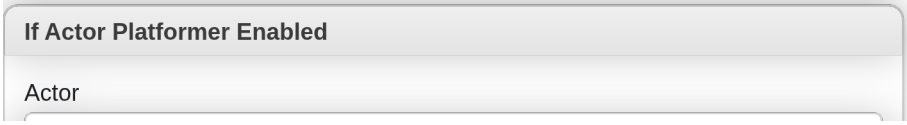
Conditionally run part of the script if an actor is pinned to the screen.

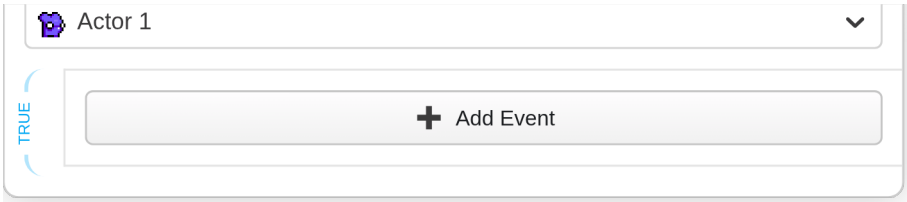


- **Actor**: The actor you want to check.

If Actor Platformer Enabled

Conditionally run part of the script if an actor is platformer mode is enabled.

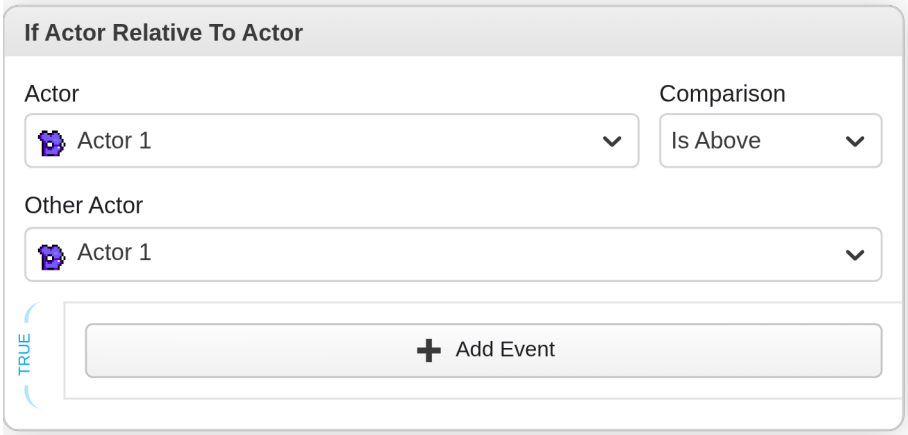




- **Actor**: The actor you want to check.

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.

Movement

Enable Movement for an Actor

Enable movement for an actor allowing gravity, velocity and acceleration to affect its position.



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

Disable Movement for an Actor

Disable movement for an actor stopping gravity, velocity and acceleration to affect its position.



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

All Actors Set Position Relative

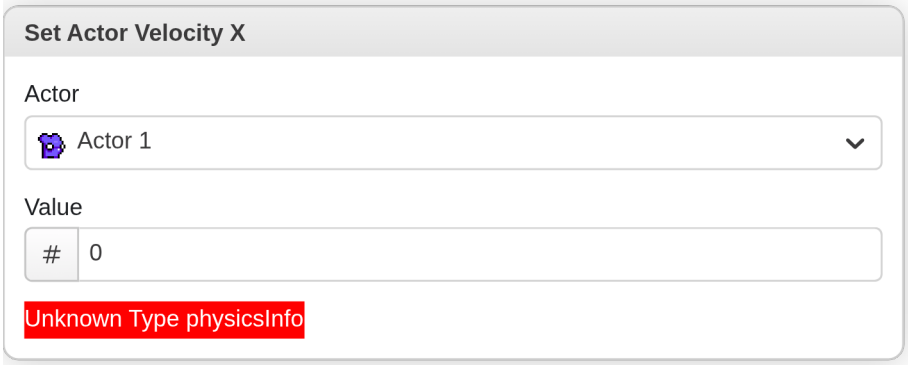
Set the position of all actors relative to its previous position, causing it to instantly move to the new location.



- **X**: The horizontal offset relative to the current position.
- **Y**: The vertical offset relative to the current position.

Set Actor Velocity X

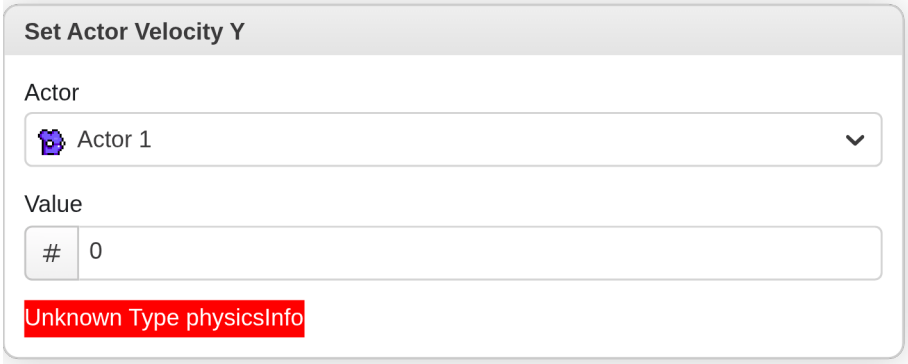
Set the Velocity X value of an actor, causing it to start moving if movement is enabled on it (in pixels per second).



- **Actor**: The actor you want to update.
- **Value**: The value you want to set.

Set Actor Velocity Y

Set the Velocity Y value of an actor, causing it to start moving if movement is enabled on it (in pixels per second).

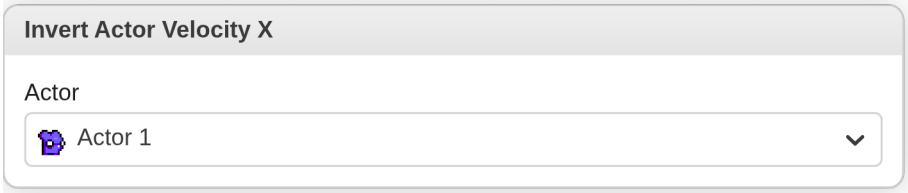


- **Actor**: The actor you want to update.

- **Value**: The value you want to set.

Invert Actor Velocity X

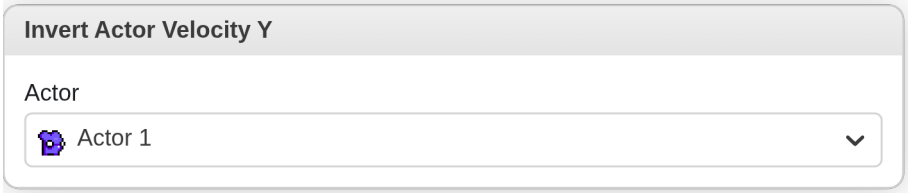
Invert the Velocity X value of an actor, much faster than a Set Velocity X event with Get Velocity X * -1.



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

Invert Actor Velocity Y

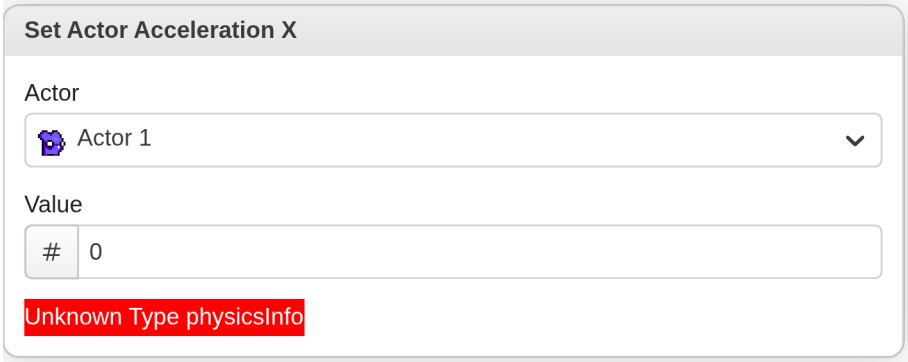
Invert the Velocity Y value of an actor, much faster than a Set Velocity Y event with Get Velocity Y * -1.



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

Set Actor Acceleration X

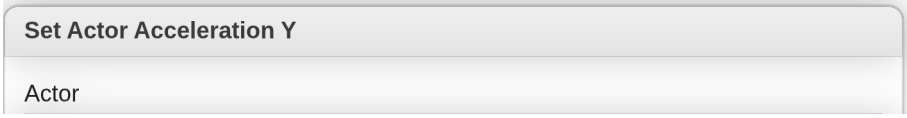
Set the Acceleration X value of an actor, causing it to start moving if movement is enabled on it (in pixels per second squared).

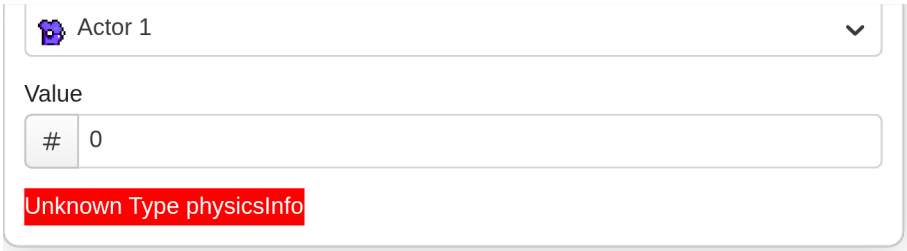


- **Actor**: The actor you want to update.
- **Value**: The value you want to set.

Set Actor Acceleration Y

Set the Acceleration Y value of an actor, causing it to start moving if movement is enabled on it (in pixels per second squared).





- **Actor**: The actor you want to update.
- **Value**: The value you want to set.

Actor Set To Target Position

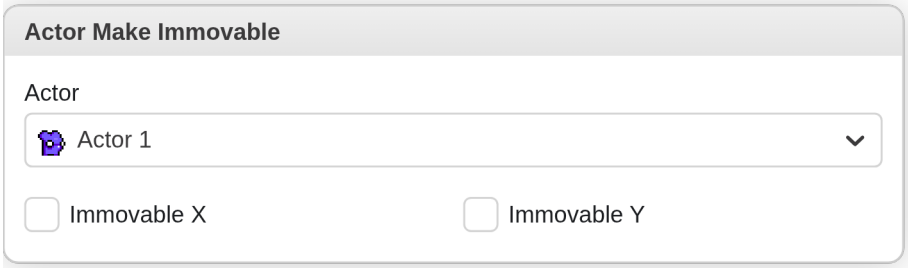
Set the actor position and direction to the one requested by the latest scene change call if any was set.



- **Actor**: The actor you want to update.
- **Set Direction**: Set the direction as well.

Actor Make Immovable

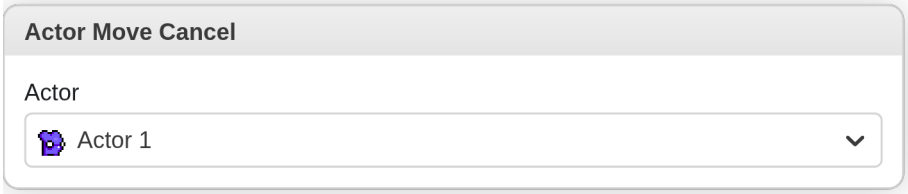
Make the actor immovable on the selected axis.



- **Actor**: The actor you want to update.
- **Immovable X**: If enabled, the actor won't move after a collision on the X axis.
- **Immovable Y**: If enabled, the actor won't move after a collision on the Y axis.

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

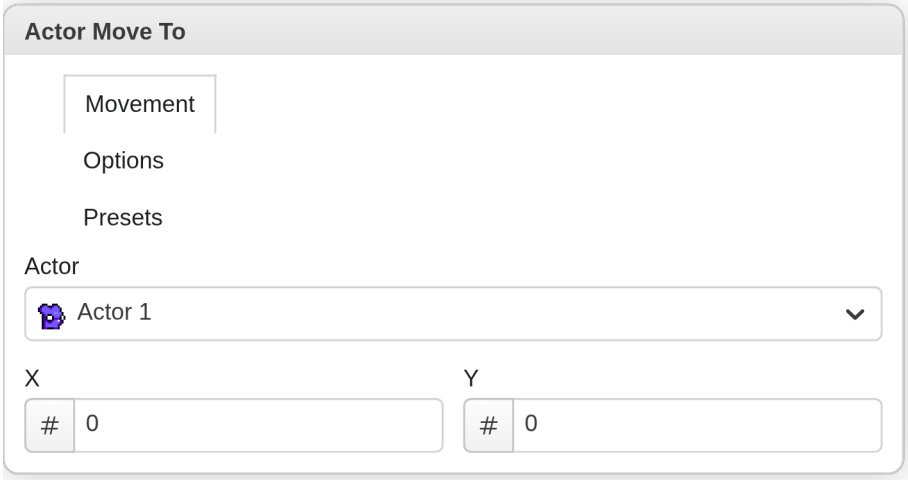
Move an actor relative to its current position.



- **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.
- **Speed**: The movement speed.
- **Stop On Touch**: Stop movement if we touch any solid surface in the movement direction, useful to avoid blocking the script.

Actor Move To

Move an actor to a new position.

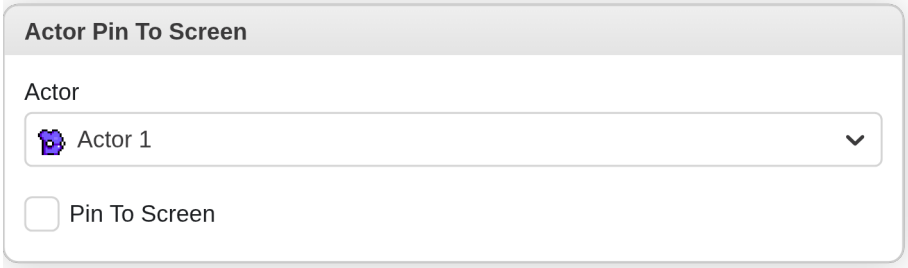


- **Actor**: The actor you want to move.
- **X**: The horizontal offset relative to the current position.
- **Y**: The vertical offset relative to the current position.
- **Movement Type**: Choose if should move in horizontal/vertical axis first or if it should move diagonally to destination.

- **Speed**: The movement speed.
- **Stop On Touch**: Stop movement if we touch any solid surface in the movement direction, useful to avoid blocking the script.

Actor Pin To Screen

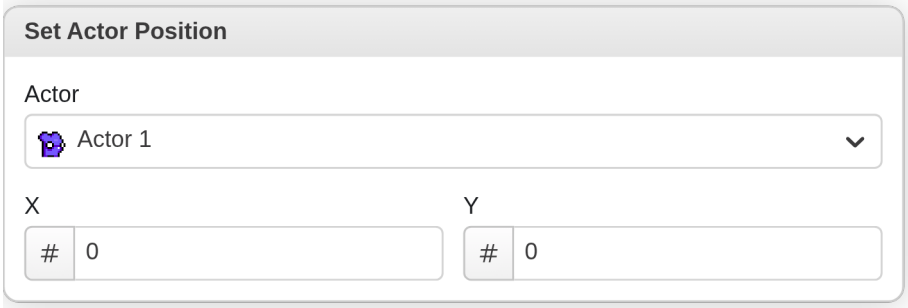
Pin the actor to the screen, will ignore camera scrolling.



- **Actor**: The actor you want to update.
- **Pin To Screen**: Pin actor to Screen.

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 its previous position, causing it to instantly move to the new location.



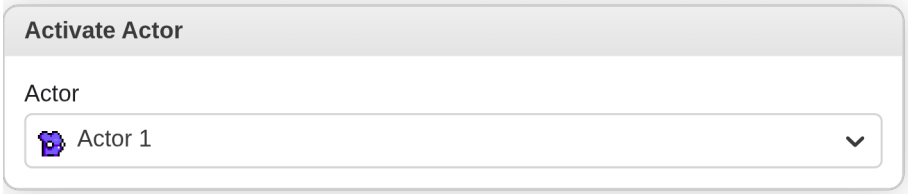
- **Actor**: The actor you want to update.

- **X**: The horizontal offset relative to the current position.
- **Y**: The vertical offset relative to the current position.

Properties

Activate Actor

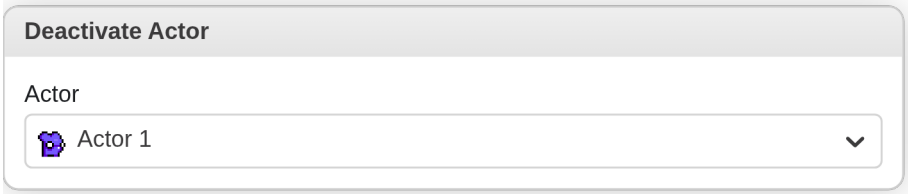
Activate an actor, causing its OnUpdate script to start.



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

Deactivate Actor

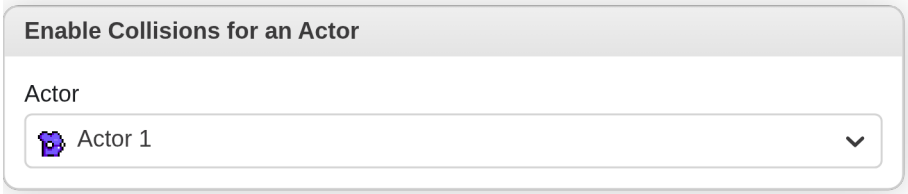
Deactivate an actor, causing its OnUpdate script to not run.



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

Enable Collisions for an Actor

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



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

Disable Collisions for an Actor

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



Note: This only makes the actor non-solid, so it passes through walls and other actors. Its OnCollide script still runs when actors or projectiles hit it. Use Disable OnCollide script to stop that.

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

Actor Flicker

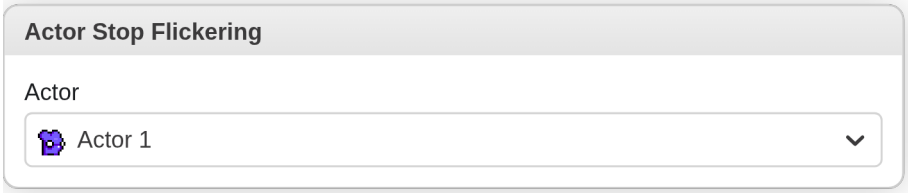
Change the visible / hidden value each frame to create an effect.



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

Actor Stop Flickering

Cancel an in-progress flicker effect on the given actor.



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

Enable OnCollide script for an Actor

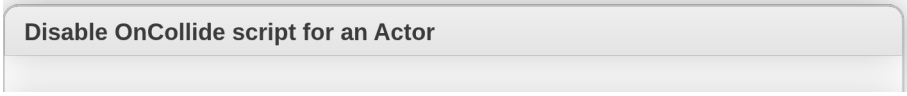
Enable OnCollide script for an actor, the script will be invoked when an overlap or collision happens.

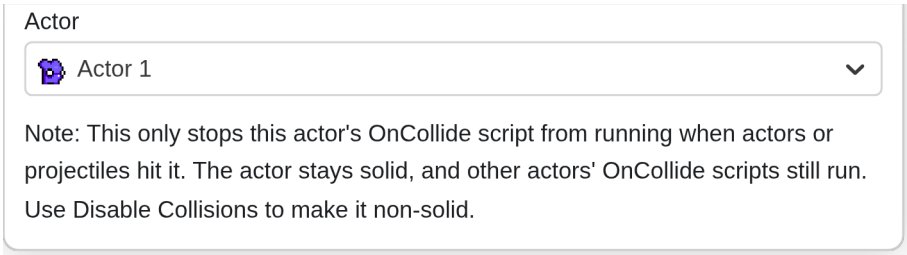


- **Actor**: The actor you want to update.

Disable OnCollide script for an Actor

Disable OnCollide script for an actor, the script won't be invoked when an overlap or collision happens.

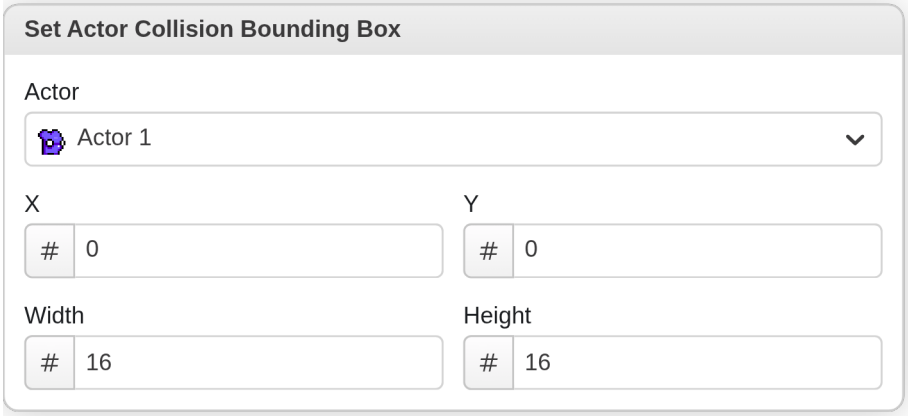




- **Actor**: The actor you want to update.

Set Actor Collision Bounding Box

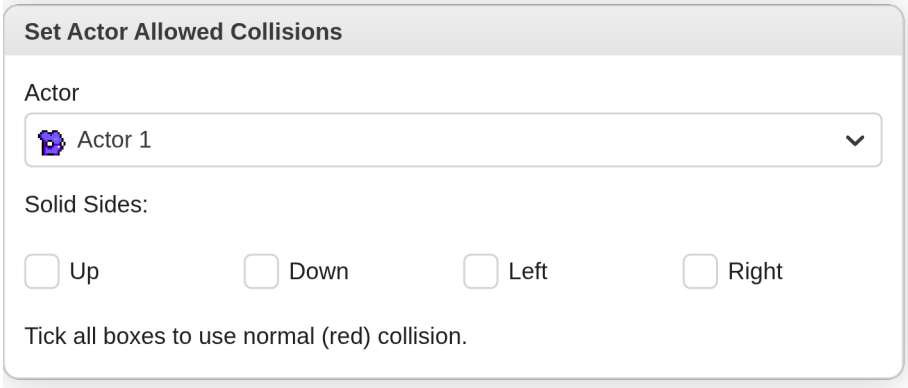
Set the collision bounding box of an actor to a position and size.



- **Actor**: The actor you want to update.
- **X**: The horizontal position.
- **Y**: The vertical position.
- **Width**: The box width.
- **Height**: The box height.

Set Actor Allowed Collisions

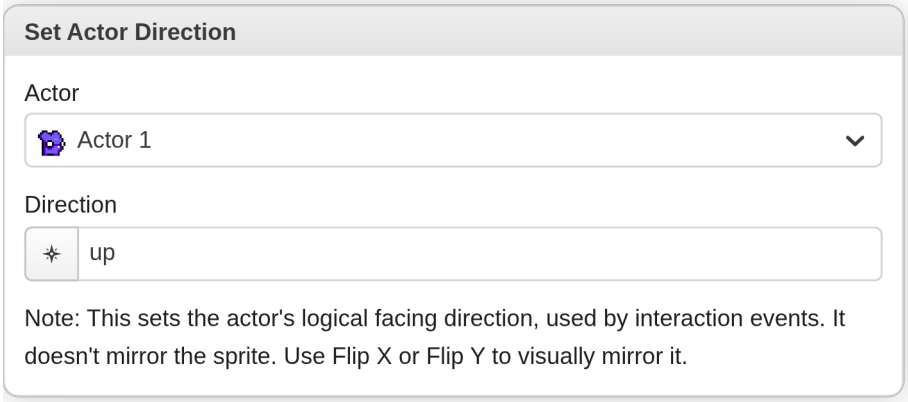
Enables specific sides for a collision, actor needs to have Enable Collisions called as well for this to work.



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

Set Actor Direction

Change the logical direction that an actor is currently facing. This does not mirror the sprite; use Flip X or Flip Y for that.



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

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.

Actor Set Palette

Set the actor palette.

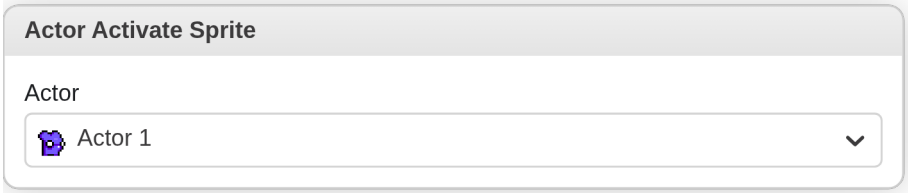


- **Actor**: The actor you want to update.
- **Palette Slot**: The palette slot you want to use.

Sprite

Actor Activate Sprite

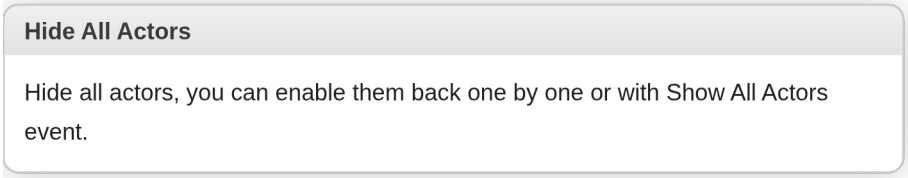
Activates an actor sprite, causing it to animate even during pause.



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

Hide All Actors

Hide all actors, you can enable them back one by one or with Show All Actors event.



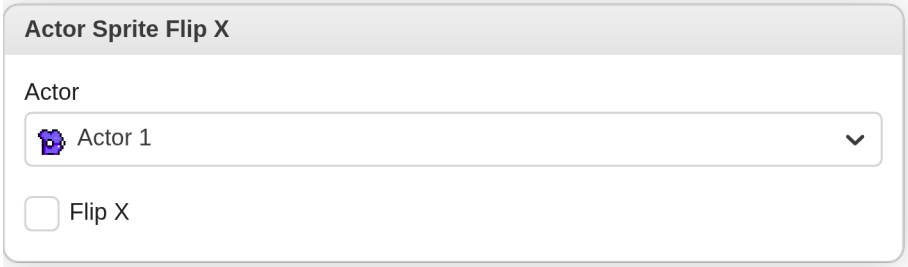
Show All Actors

Show all actors that were hidden.



Actor Sprite Flip X

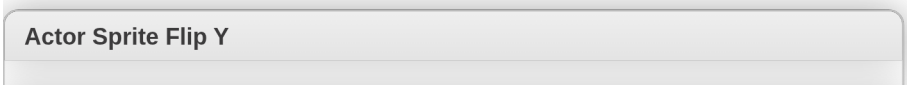
Flip the actor sprite on the X axis.

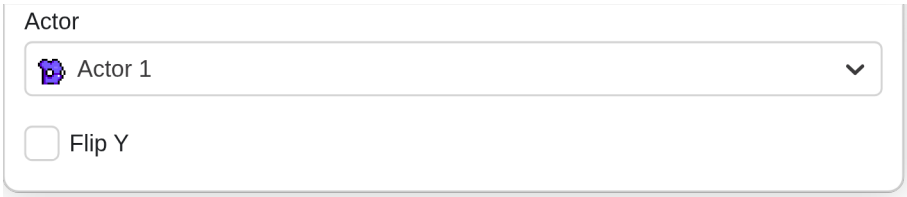


- **Actor**: The actor you want to update.
- **Flip X**: Flip Sprite on X axis

Actor Sprite Flip Y

Flip the actor sprite on the Y axis.

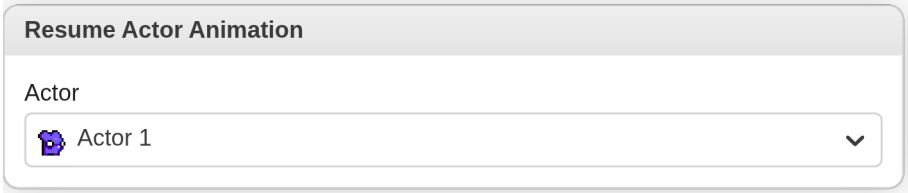




- **Actor**: The actor you want to update.
- **Flip Y**: Flip Sprite on Y axis

Resume Actor Animation

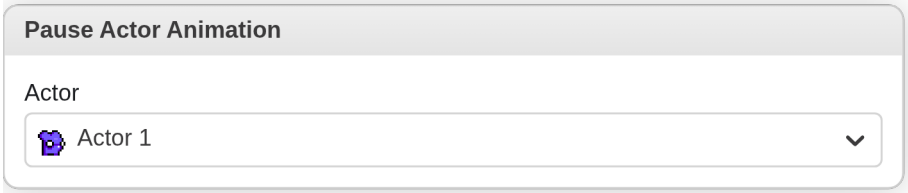
Resume the animation that should be playing.



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

Pause Actor Animation

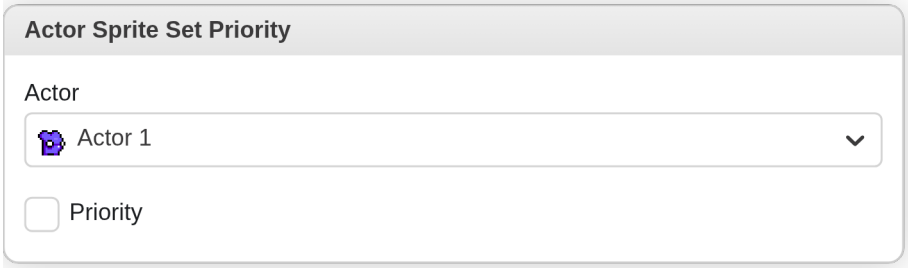
Pause the animation that should be playing.



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

Actor Sprite Set Priority

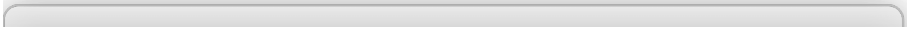
Set the actor priority flag.



- **Actor**: The actor you want to update.
- **Priority**: They will appear over the foreground.

Actor Send to Front

Send the actor to the top of the rendering queue so it renders on top of all other actors.

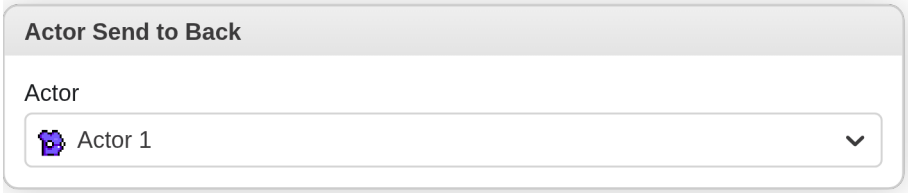




- **Actor**: The actor you want to use.

Actor Send to Back

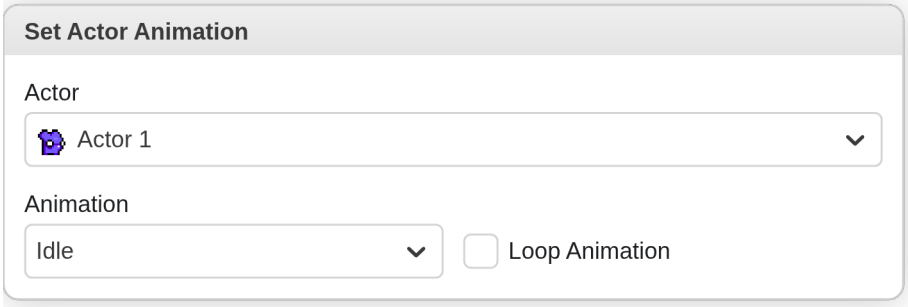
Send the actor to the back of the rendering queue so it renders below all other actors.



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

Set Actor Animation

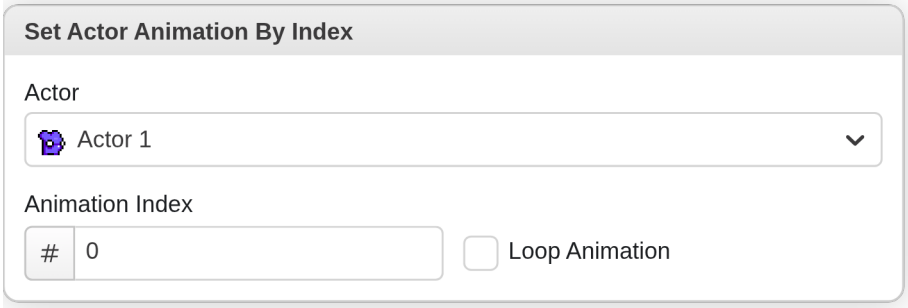
Set the animation that should be played by the actor.



- **Actor**: The actor you want to update.
- **Animation**: The animation to set
- **Loop Animation**: Set if animation should loop.

Set Actor Animation By Index

Set the animation that should be played by the actor by their numeric index.

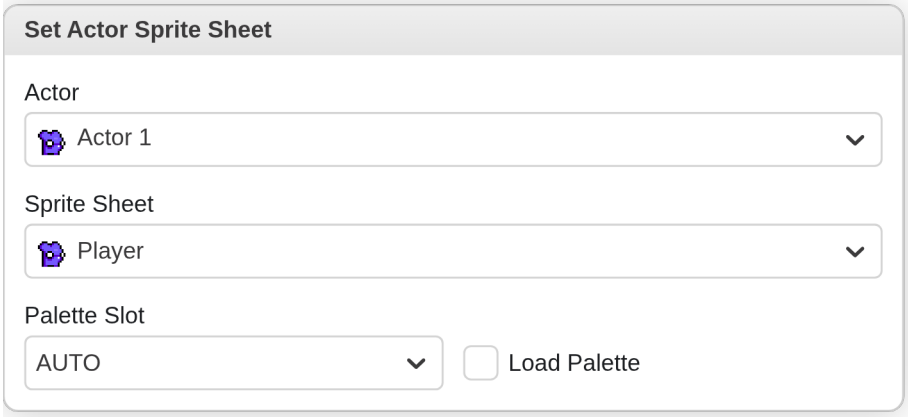


- **Actor**: The actor you want to update.
- **Animation Index**: The animation index from the sprite sheet.

- **Loop Animation**: Set if animation should loop.

Set Actor Sprite Sheet

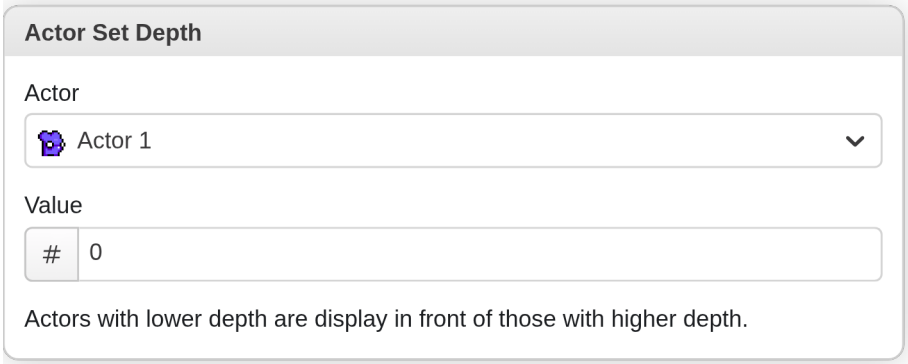
Set the sprite that should be used to render the actor.



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

Actor Set Depth

Set the actor depth on the rendering queue. From -255 to 255. Lower values means rendering higher on the queue.

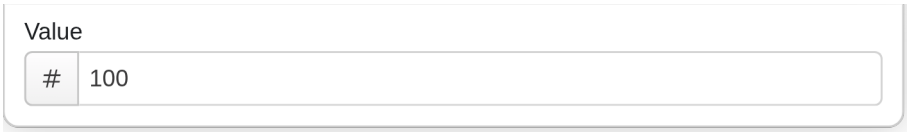


- **Actor**: The actor you want to update.
- **Value**: The value you want to set.

Actor Set Elasticity

Set the actor elasticity, the bounciness of the actor. Only affects advance collisions.





- **Actor**: The actor you want to update.
- **Value**: The value you want to set.

Variables

Increment Another Actor Local Variable

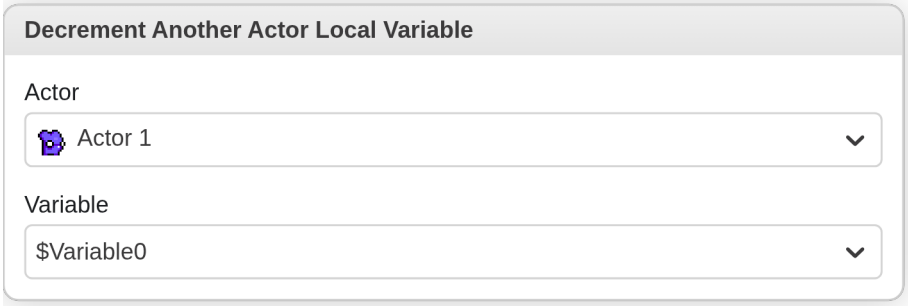
Increment a different actor Local Variable.



- **Actor**: The actor you want to update.
- **Variable**: The variable to use.

Decrement Another Actor Local Variable

Decrement a different actor Local Variable.



- **Actor**: The actor you want to update.
- **Variable**: The variable to use.

Store Actor Direction In Variable

Store the current direction of an actor within a variable. Down: 0 Right: 1 Up: 2 Left: 3





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

Store Actor Position In Variables

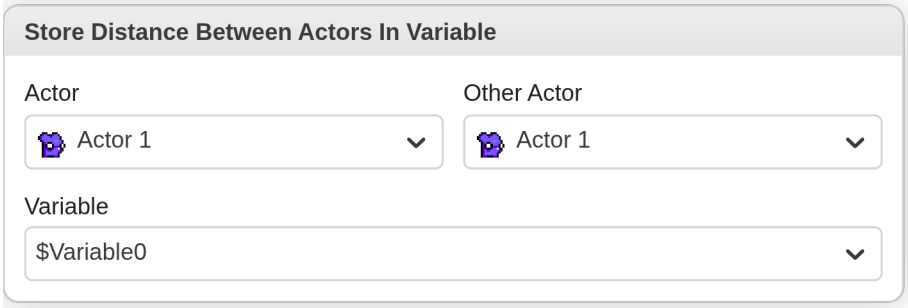
Store the current position of an actor within two variables, one to store the horizontal position and another to store the vertical position.



- **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.

Store Distance Between Actors In Variable

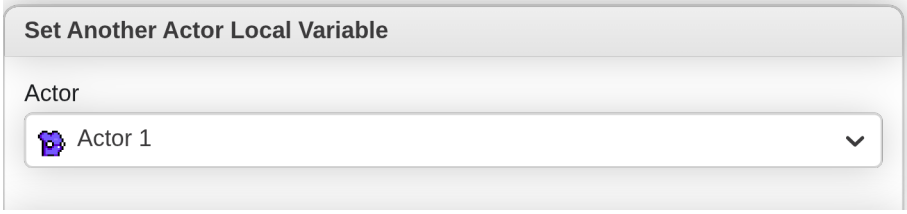
Compute the Euclidean distance between two actors and store the result in a variable. Units default to pixels; switch to tiles via the variable's unit toggle.

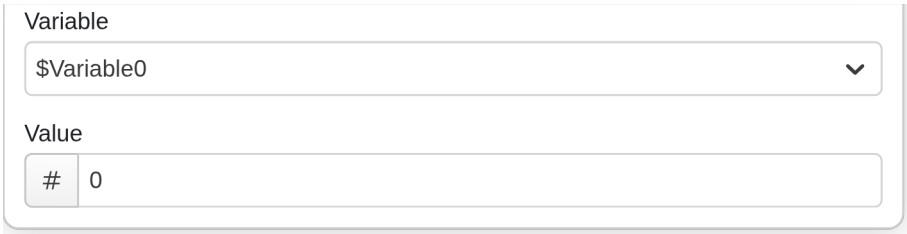


- **Actor**: The actor you want to check.
- **Other Actor**: The second actor to measure distance to.
- **Variable**: The variable to use.

Set Another Actor Local Variable

Set a different actor Local Variable to the selected value.



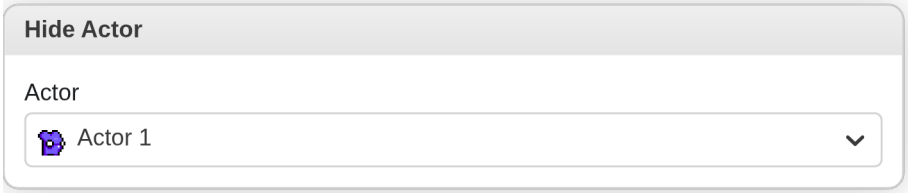


- **Actor**: The actor you want to update.
- **Variable**: The variable to use.
- **Value**: The value to set the selected variable to.

Visibility

Hide Actor

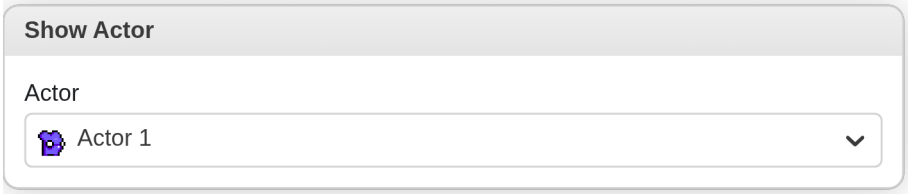
Hide an actor, causing it to become invisible. Its OnUpdate script will continue to run while hidden.



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

Show Actor

Unhide a previously hidden actor.

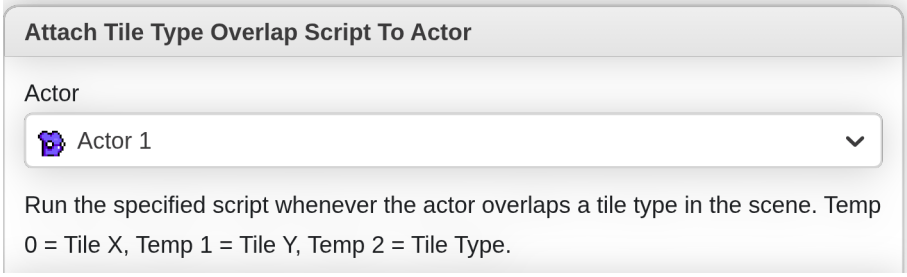


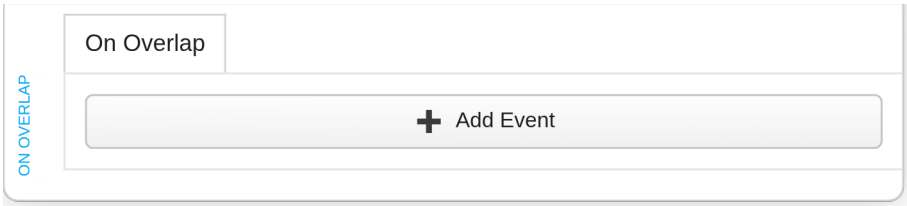
- **Actor**: The actor you want to show.

Scripts

Attach Tile Type Overlap Script To Actor

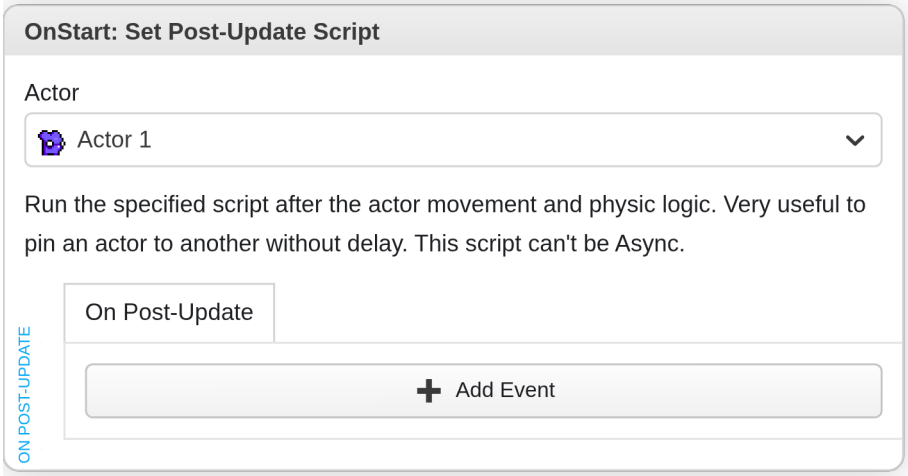
Run the specified script whenever the actor overlaps a tile type in the scene. Temp 0 = Tile X, Temp 1 = Tile Y, Temp 2 = Tile Type.





OnStart: Set Post-Update Script

Run the specified script after the actor movement and physic logic. Very useful to pin an actor to another without delay. This script can't be Async.



- **Actor**: The actor you want to update.
- **On Post-Update**: The script to run when after all update logic.

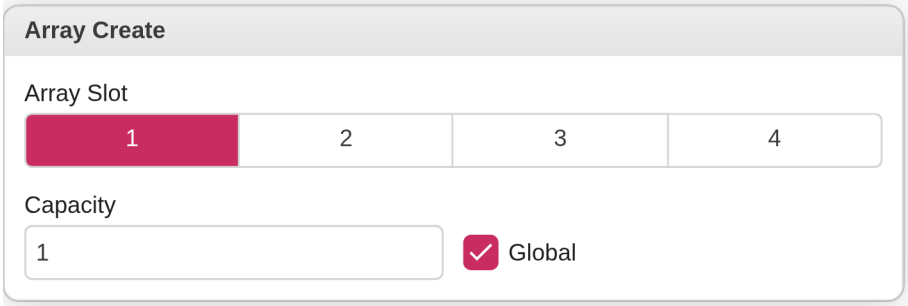
Array

Events for working with arrays, including reading, writing, and manipulating array data.

Setup

Array Create

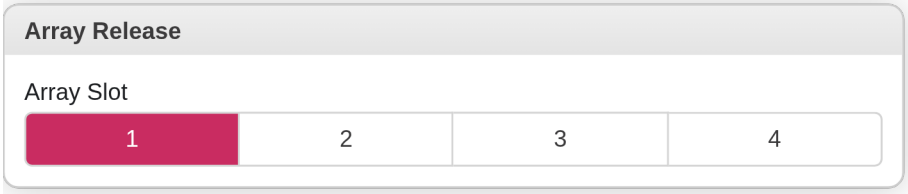
Create a new array for using in the game.



- **Array Slot**: The array slot to use.
- **Capacity**: The number of elements it can hold.
- **Global**: Global arrays dont get released on scene changes.

Array Release

Release the selected array and free all used memory.



- **Array Slot**: The array slot to use.

Set Array Size

Set the array size and fill all with 0. The value can't be bigger than the original capacity.



- **Array Slot**: The array slot to use.

Set Array Global

Set the array global flag.

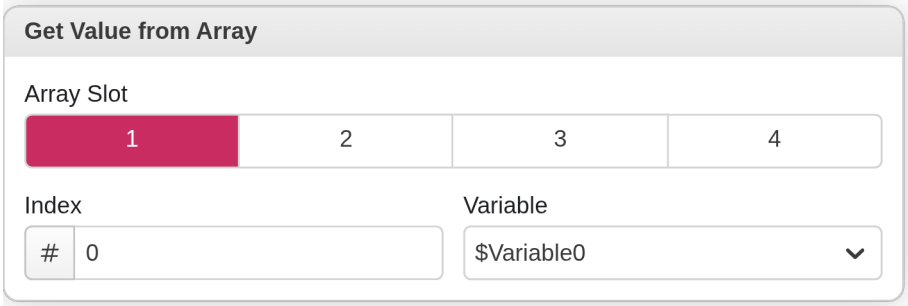


- **Array Slot**: The array slot to use.
- **Global**: Global arrays dont get released on scene changes.

Access

Get Value from Array

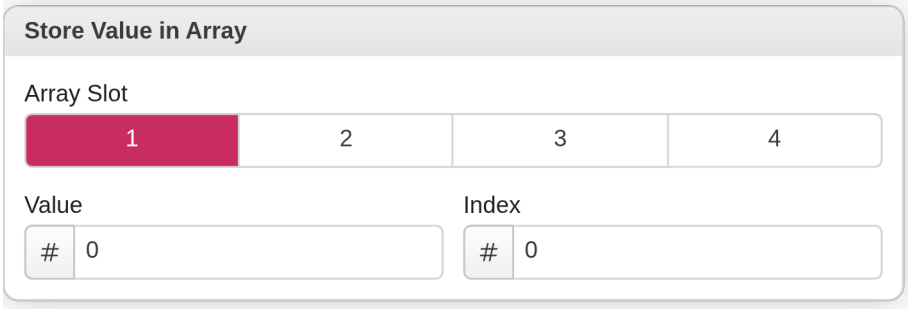
Get the value of the given position of the array into a variable.



- **Array Slot**: The array slot to use.
- **Variable**: The variable to use.

Store Value in Array

Set the value of the given position of the array from a variable.



- **Array Slot**: The array slot to use.
- **Value**: The value you want to set.

Actions

Push Value to Array

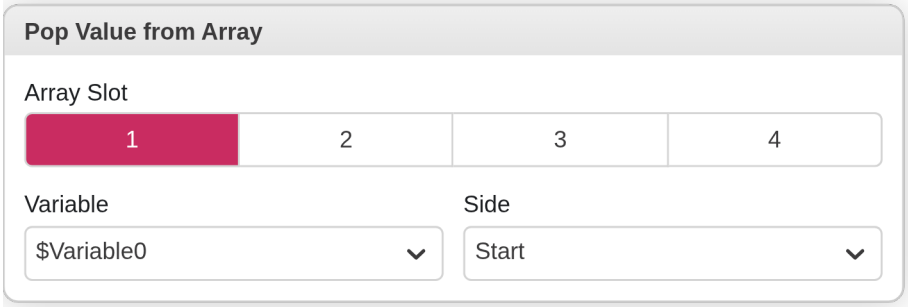
Push a new value into the array.



- **Array Slot**: The array slot to use.
- **Side**: The side do you want to perform the operation.

Pop Value from Array

Remove a value from the array and store it into a variable.



- **Array Slot**: The array slot to use.
- **Variable**: The variable to use.
- **Side**: The side do you want to perform the operation.

Delete Index from Array

Delete the given index from the array.



- **Array Slot**: The array slot to use.

Delete Value from Array

Delete the given value from the array.

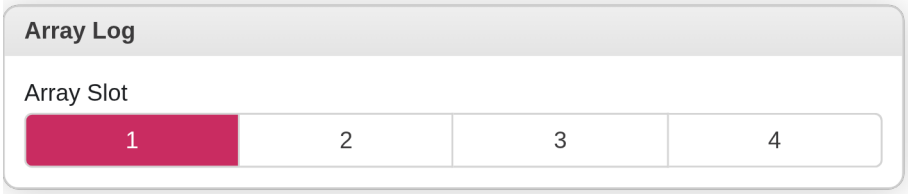


- **Array Slot**: The array slot to use.

Miscellaneous

Array Log

Log the selected array. Useful for debugging.

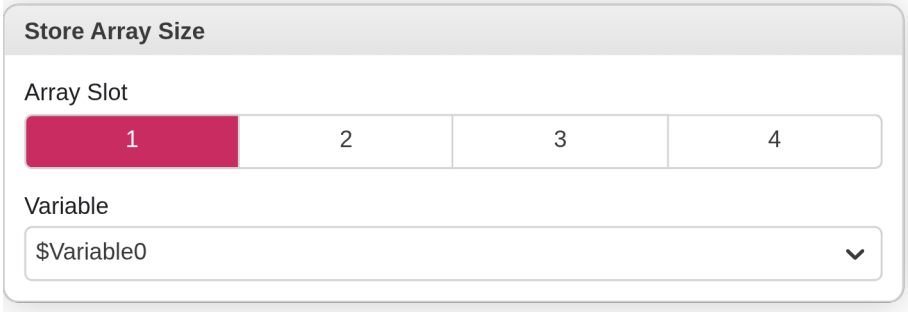


- **Array Slot**: The array slot to use.

Properties

Store Array Size

Store the array size in a variable.



- **Array Slot**: The array slot to use.
- **Variable**: The variable to use.

Behaviours

Events for enabling and controlling built-in actor behaviours such as platformer movement and tile-based movement.

Platformer

Enable Platformer Mode for an Actor

Enable Platformer Mode for an actor, it does for some automatic clears to speed up platformer development.



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

Disable Platformer Mode for an Actor

Disable Platformer Mode for an actor, it saves some CPU, but not much.



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

Actor Simulate Control Press

Simulate a control press on a platformer game for the selected actor.

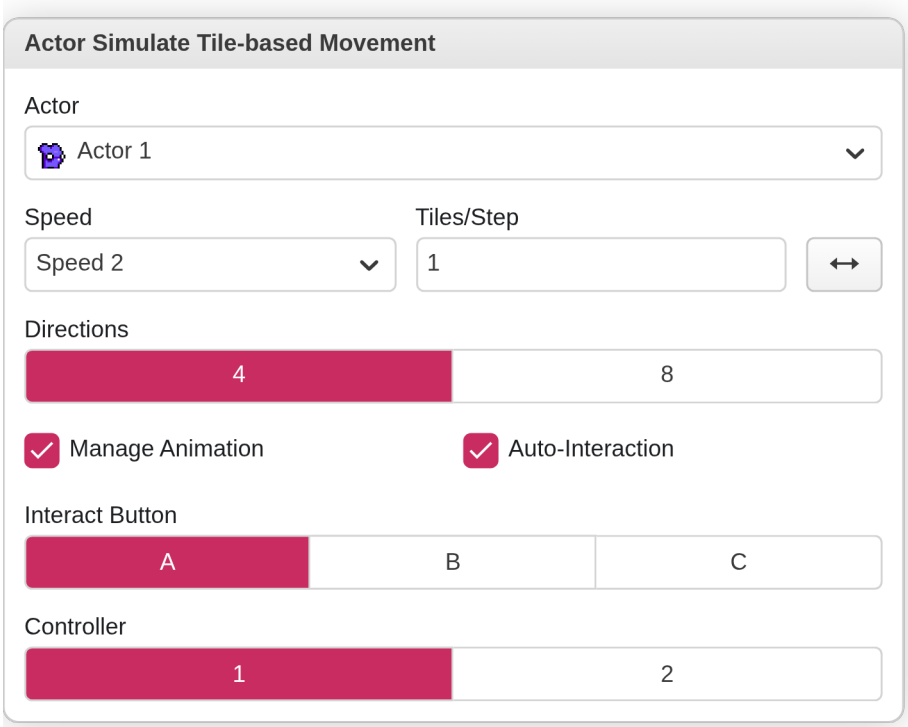


- **Actor**: The actor you want to use.
- **Control**: The direction to simulate pressing.

Tile Movement

Actor Simulate Tile-based Movement

Simulate the actor on a tile-based movement automatically, also interact automatically.



- **Actor**: The actor you want to update.
- **Movement Type**: Choose if should move in horizontal/vertical axis first or if it should move diagonally to destination.
- **Speed**: The movement speed.
- **Tiles/Step**: The number of tiles to move each time, normally character width / 8.
- **Controller**: The controller index to check.
- **Interact Button**: The joypad button to check.

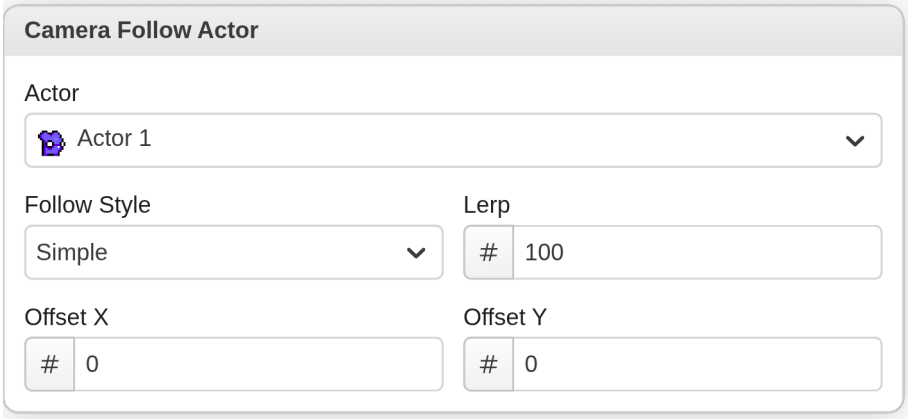
Camera

Events for controlling the camera position, movement, and screen effects like shake and flash.

Actions

Camera Follow Actor

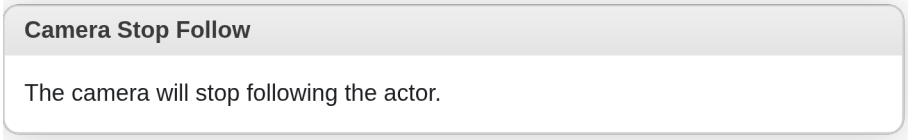
Follow the selected actor.



- **Follow Style**: The deadzone predefined setup, you can use Set Deadzone if you don't like any.
- **Lerp**: How smooth the camera follows. From 0 to 100.
- **Offset X**: The horizontal position.
- **Offset Y**: The vertical position.

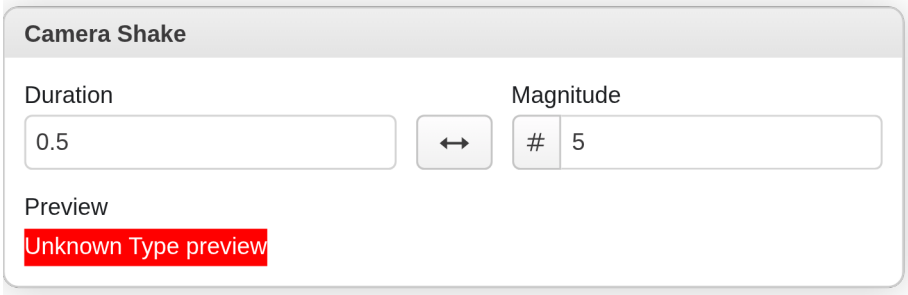
Camera Stop Follow

The camera will stop following the actor.



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.
- **Magnitude**: The maximum amount of pixels the camera will move during the shake.

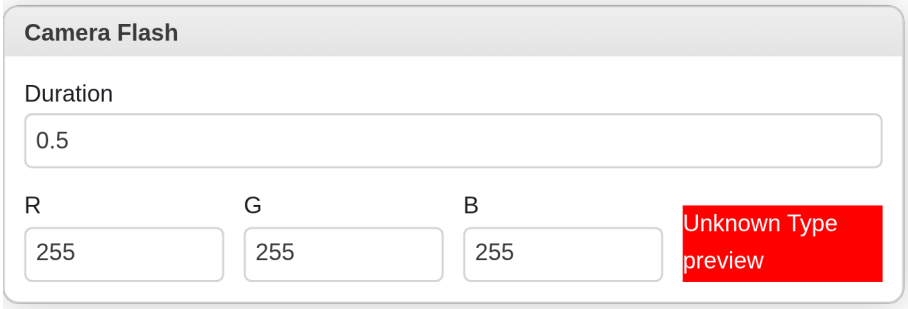
Camera Stop Shake

The camera will stop shaking if it is.



Camera Flash

Flash the camera with the selected color for a period of time.

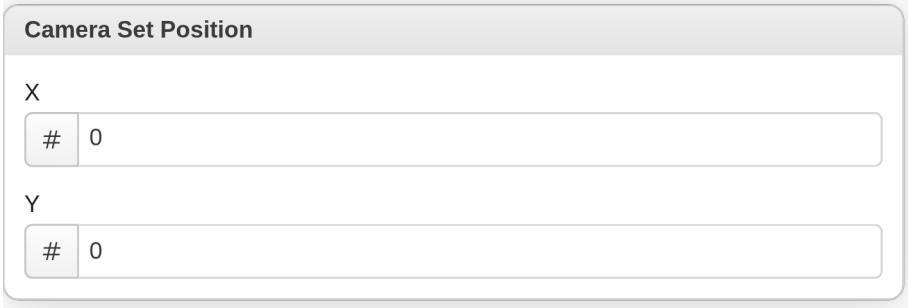


- **Duration**: The length of time to wait before the colors go back to normal in seconds or frames.

Movement

Camera Set Position

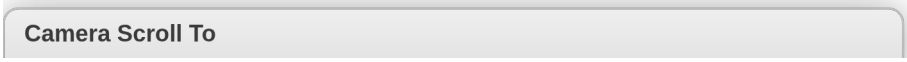
Set the position of the camera, causing it to instantly move to the new location.

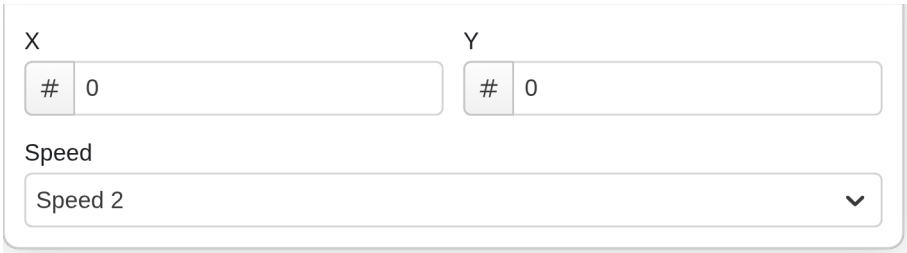


- **X**: The horizontal position.
- **Y**: The vertical position.

Camera Scroll To

Scroll the camera to a new position.



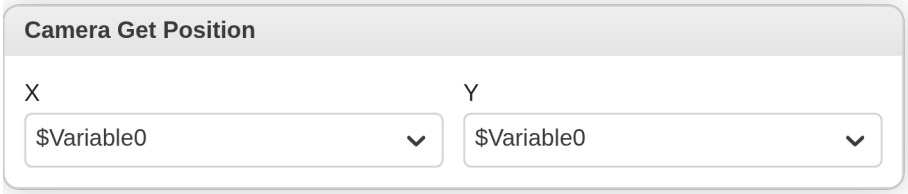


- **X**: The horizontal position.
- **Y**: The vertical position.
- **Speed**: The movement speed.

Variables

Camera Get Position

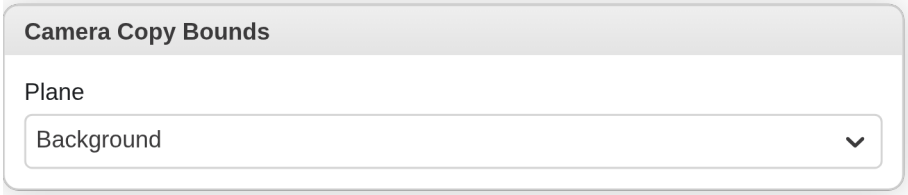
Store the camera's top-left X and Y position into two variables. Note: this returns the top-left corner, while Camera Set Position uses the camera center.



Properties

Camera Copy Bounds

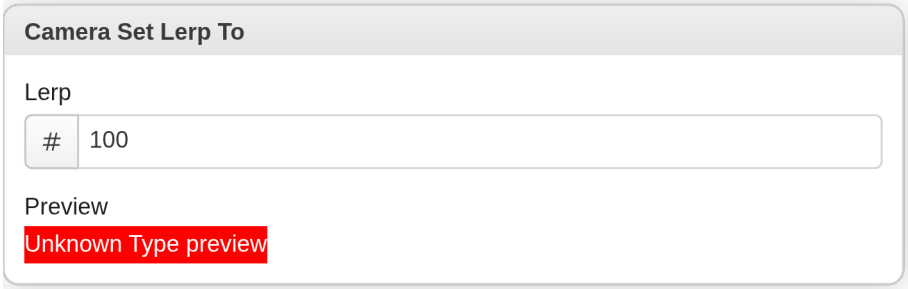
Copy the camera bounds from the selected plane image or map.



- **Plane**: The plane we want to use

Camera Set Lerp To

Set the camera lerp to the given value.



- **Lerp**: How smooth the camera follows. From 0 to 100.

Camera Set Offset To

Set the camera offset to the given value.



- **Offset X**: The horizontal position.
- **Offset Y**: The vertical position.

Camera Set Deadzone

Set the camera deadzone, actors movement inside the deadzone won't move the camera.



- **X**: The horizontal position.
- **Y**: The vertical position.
- **Width**
- **Height**

Camera Set Bounds

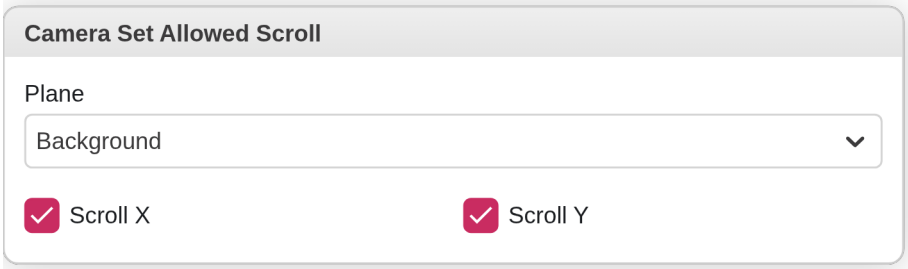
Set the camera bounds, the camera won't move outside these. Normally this would be the level size.



- **X**: The horizontal position.
- **Y**: The vertical position.
- **Width**
- **Height**

Camera Set Allowed Scroll

Enable or disable scroll axes for the selected layer.



- **Plane**: The plane we want to use
- **Scroll X**: Enable or disable horizontal scrolling for the selected layer.
- **Scroll Y**: Enable or disable vertical scrolling for the selected layer.

Collisions & Overlaps

Events for detecting and responding to collisions and overlaps between actors.

Actions

Collisions Enable Group

Enables the given collision group. This only affects Actor vs Actor, not actor vs map.

Collisions Enable Group

Collision Group

Group 1	Group 2	Group 3	Group 4
Group 5	Group 6	Group 7	Group 8

- **Collision Group**: The collision group that should be used when registering collisions with actors.

Collisions Disable Group

Disables the given collision group. This only affects Actor vs Actor, not actor vs map.

Collisions Disable Group

Collision Group

Group 1	Group 2	Group 3	Group 4
Group 5	Group 6	Group 7	Group 8

- **Collision Group**: The collision group that should be used when registering collisions with actors.

Collisions Enable All Groups

Enables all collision groups. This only affects Actor vs Actor, not actor vs map.

Collisions Enable All Groups

Enables all collision groups. This only affects Actor vs Actor, not actor vs map.

Collisions Disable All Groups

Disables all collision groups. This only affects Actor vs Actor, not actor vs map.

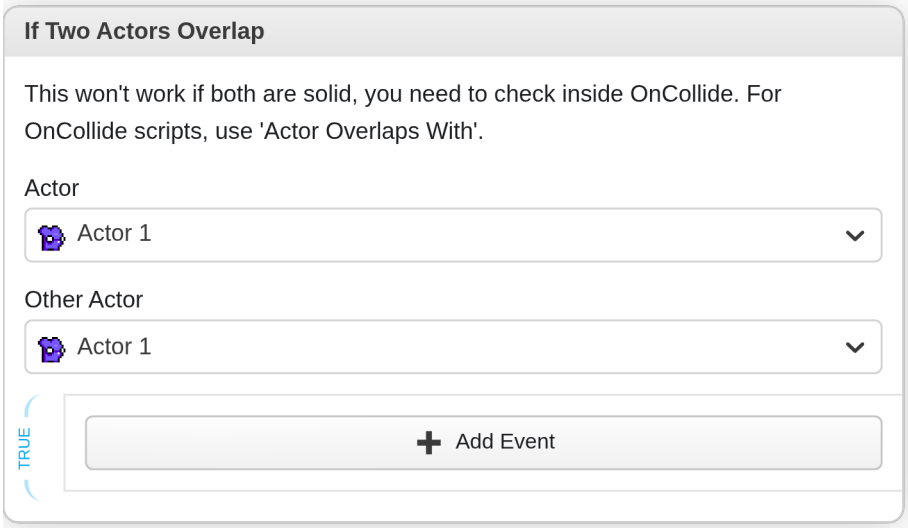
Collisions Disable All Groups

Disables all collision groups. This only affects Actor vs Actor, not actor vs map.

Control Flow

If Two Actors Overlap

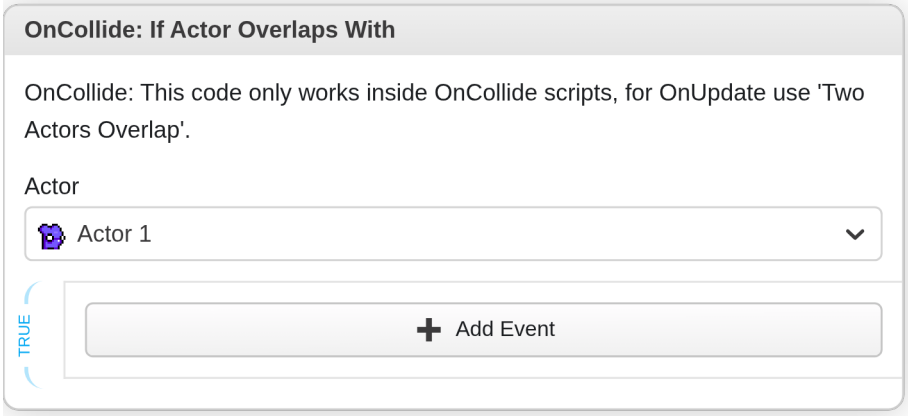
Conditionally run part of the script if the two selected actors overlap.



- **Actor**: The actor you want to check.
- **Other Actor**: The actor you want to check against.

OnCollide: If Actor Overlaps With

Conditionally run part of the script if the actor we overlapped with is the selected one. Only useful inside OnCollide scripts.



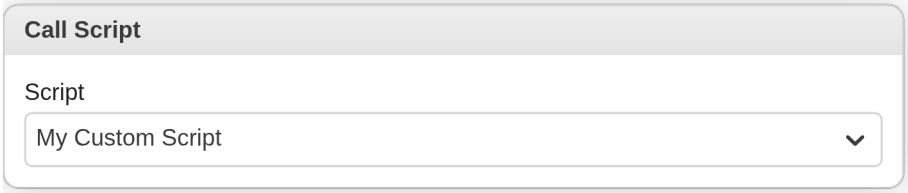
Control Flow

Events for controlling script execution flow, including conditionals, loops, and script calls.

Actions

Call Script

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

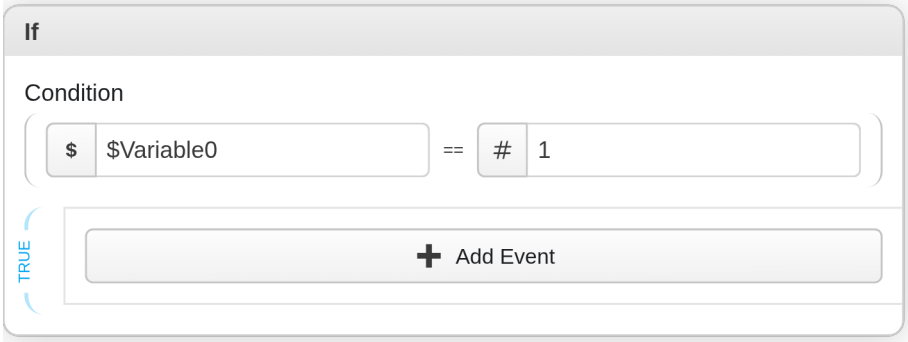


- **Script**: The script to run.

Control Flow

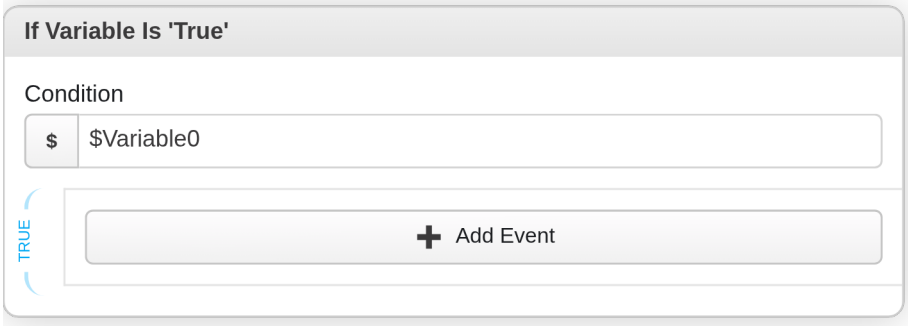
If

Conditionally run part of the script if the specified value evaluates to true.



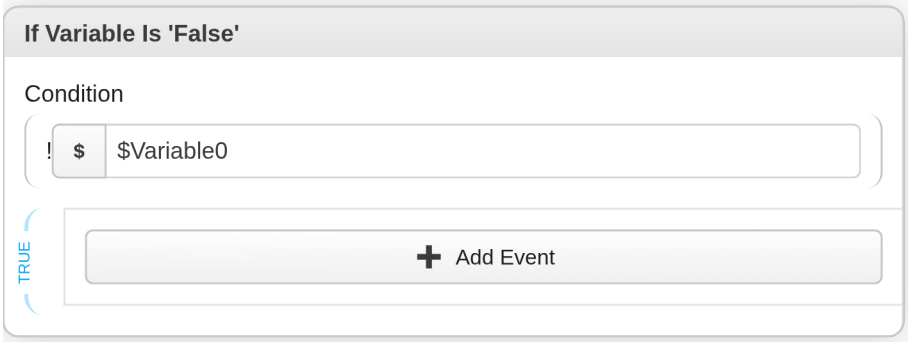
If True

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



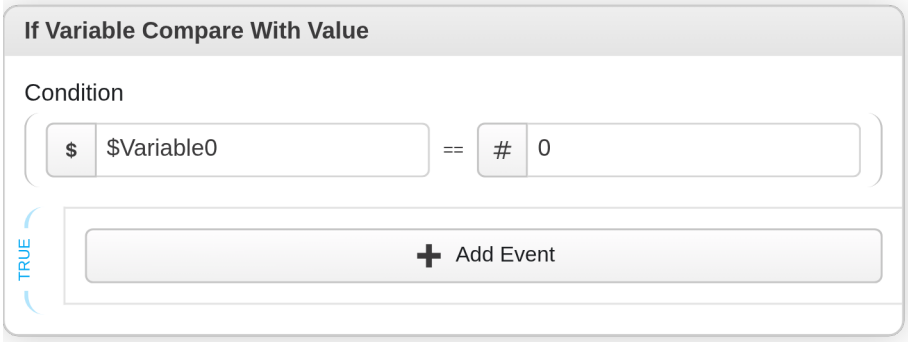
If False

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



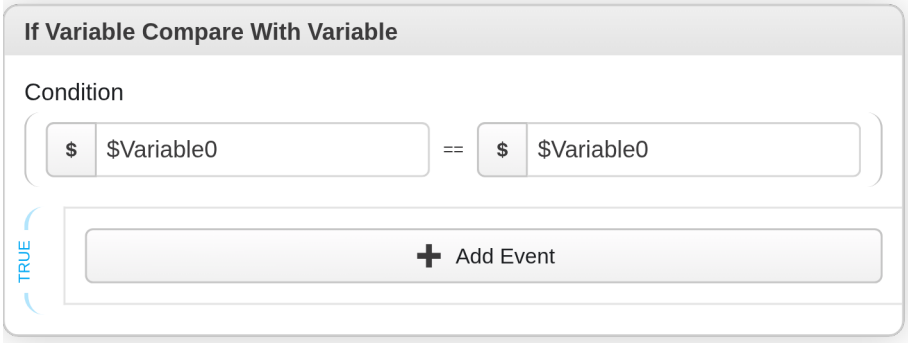
If Variable Compare With Value

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



If Variable Compare With Variable

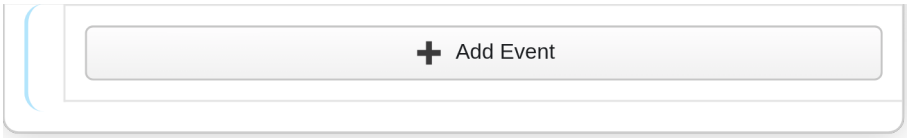
Conditionally run part of the script based on the value of a variable compared with another variable.



Loop While

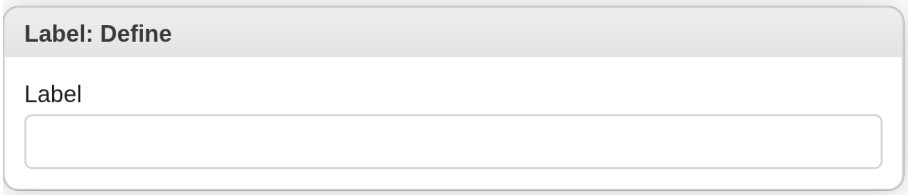
Run part of the script in a loop while an expression is true.





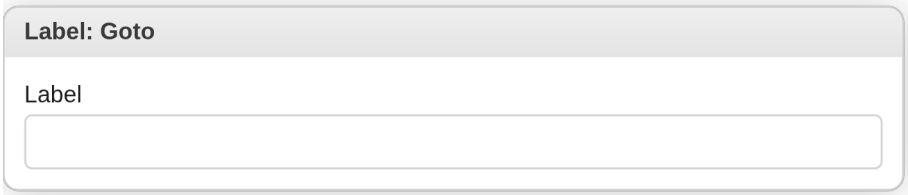
Label: Define

Define a new label to jump to.



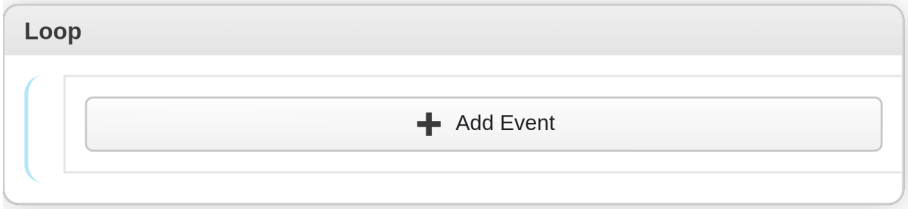
Label: Goto

Jump to the selected label inside the script.



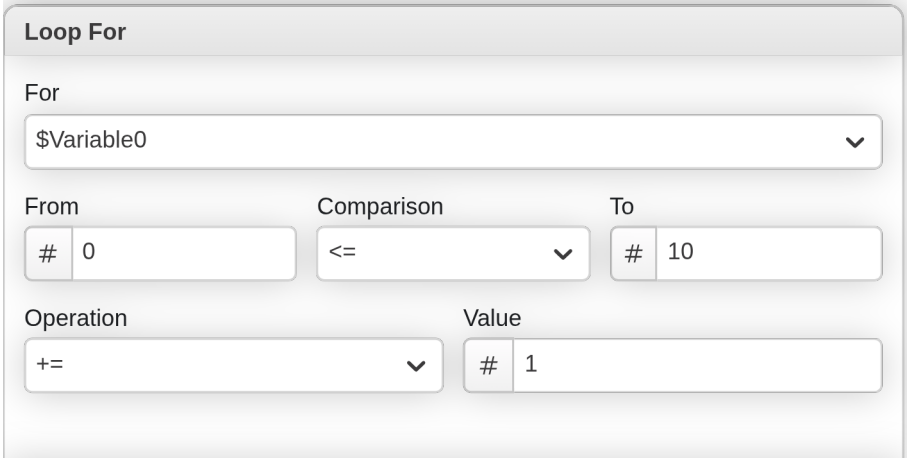
Loop

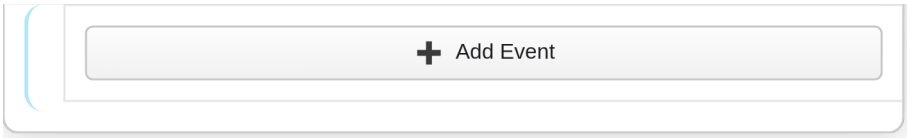
Run part of the script in a loop forever. Remember to break out of the loop otherwise the player will become stuck at this point. You can use a Stop Script or Change Scene event to stop the loop.



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.

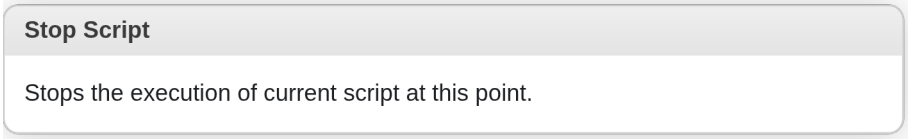




- **For**: The variable to use.
- **From**: The initial value of the counter variable.
- **Comparison**: The comparison operator to use e.g. 'Less Than' or 'Greater Than'.
- **To**: The end value of the counter variable.
- **Operation**: The operation to use for combining a value with the counter variable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

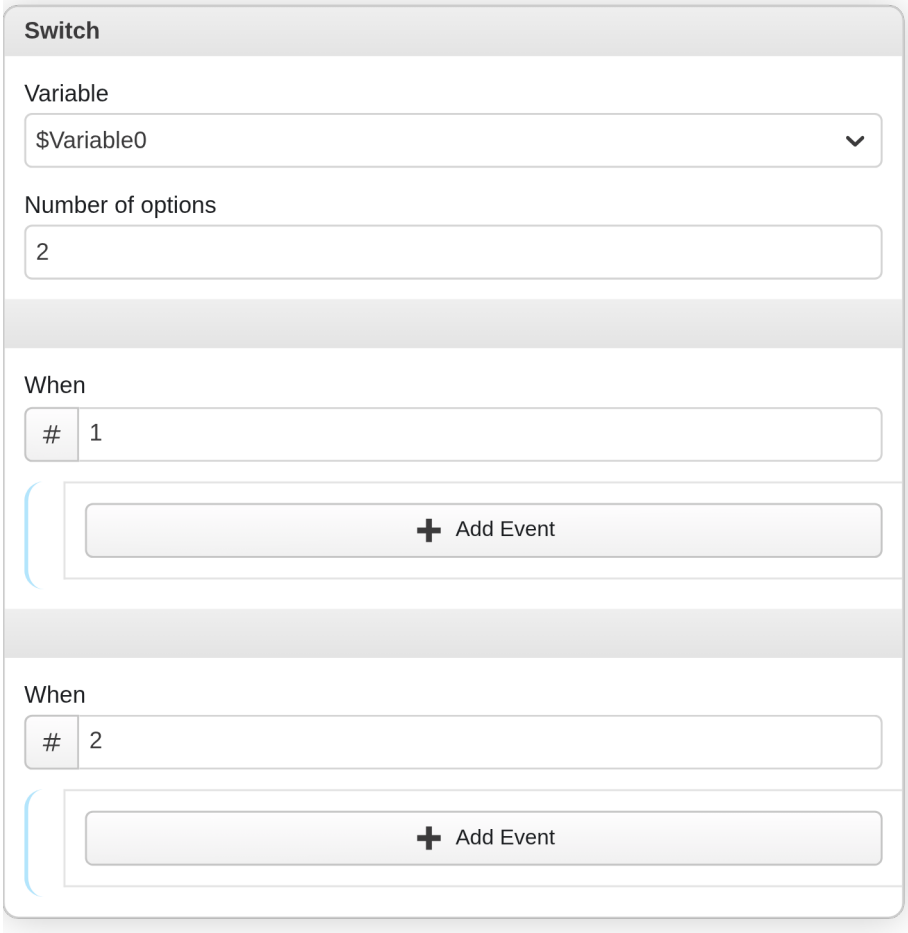
Stop Script

Stops the 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.

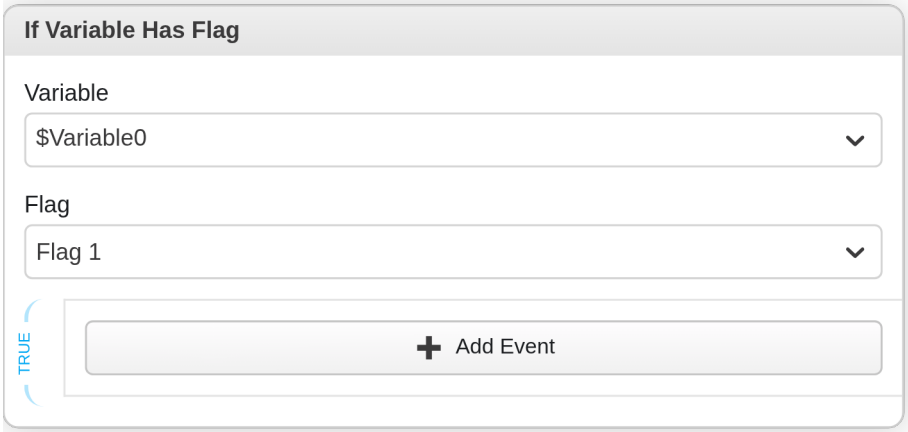


- **Variable**: The variable to use.
- **Number of options**: The number of options required.

Variables

If Variable Has Flag

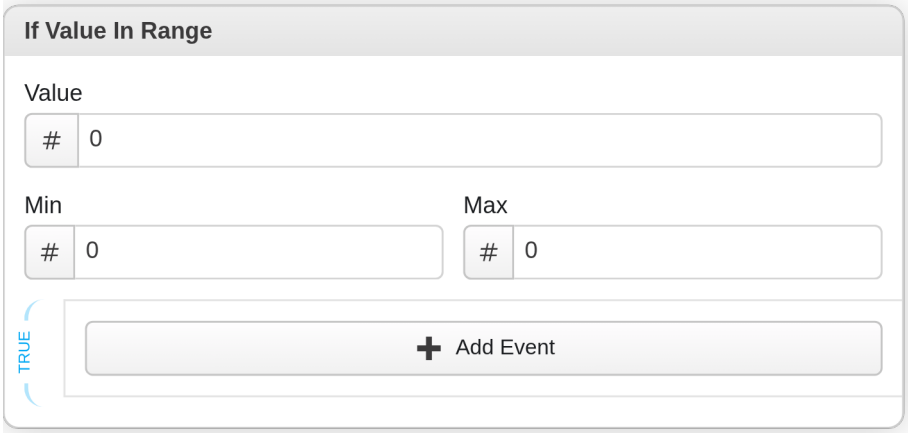
Conditionally run part of the script if the specified variable has the chosen flag set as true.



- **Variable**: The variable to use.
- **Flag**: The flag to check.

If Value In Range

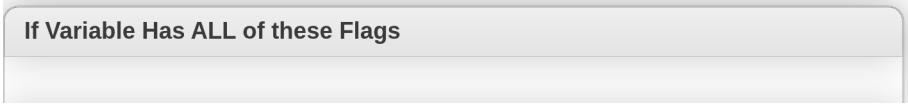
Conditionally run part of the script if the given value is on the given range.

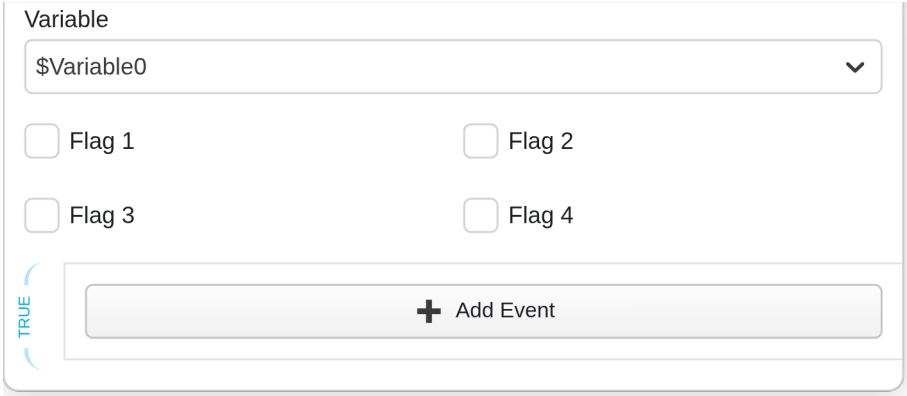


- **Value**: The value to check.
- **Min**: The minimum value (inclusive).
- **Max**: The maximum value (inclusive).

If Variable Has ALL of these Flags

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

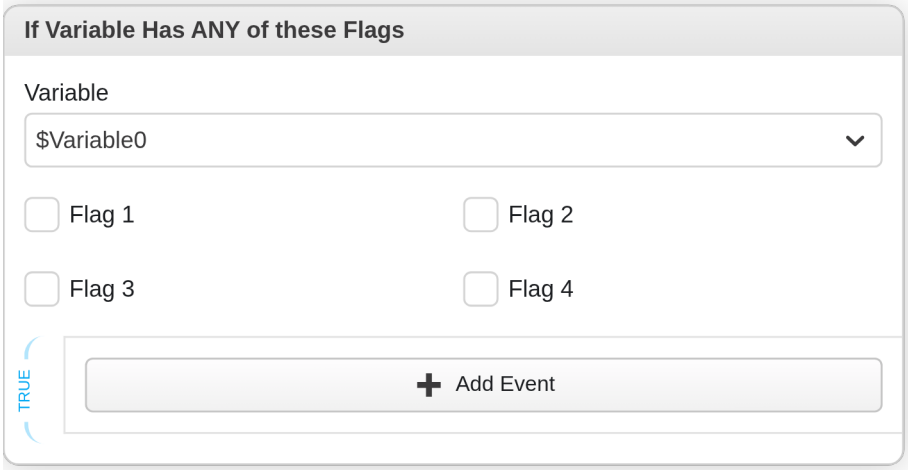




- **Variable**: The variable to use.

If Variable Has ANY of these Flags

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



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

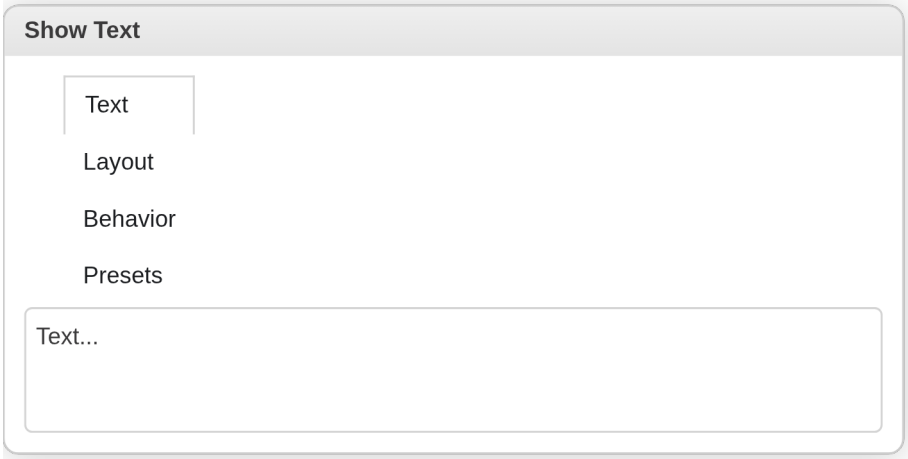
Dialogue, Text & Menus

Events for displaying dialogue, text, and menu choices.

Actions

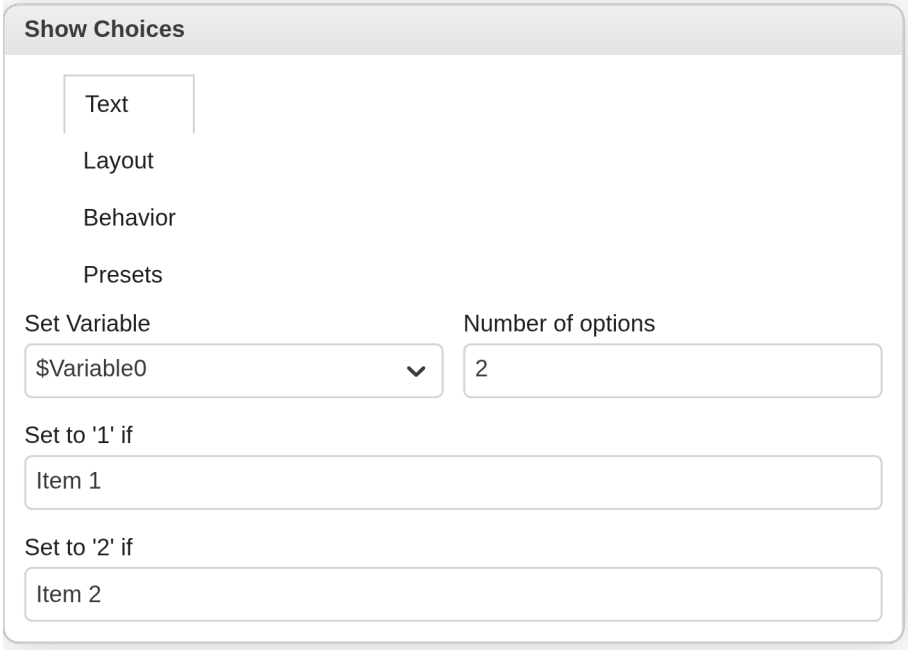
Show Text

Show a dialogue box on the screen. You need to use interact button for it to go away.



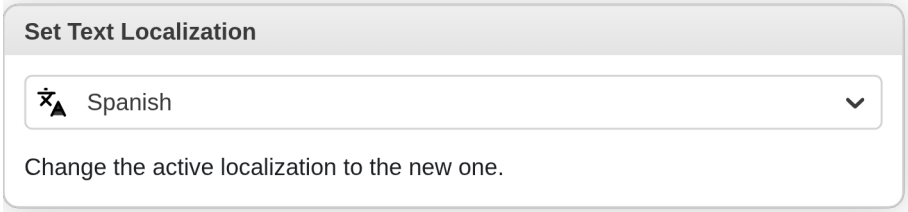
Show Choices

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.



Set Text Localization

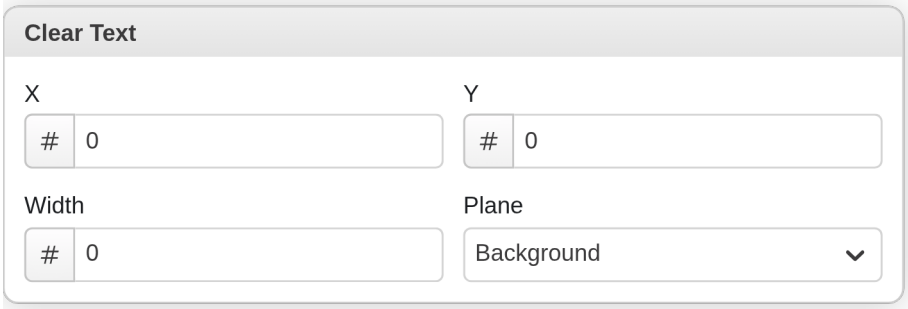
Change the active localization, any future text related calls will use the new localization.



- **Localization**: Change the active localization to the new one.

Clear Text

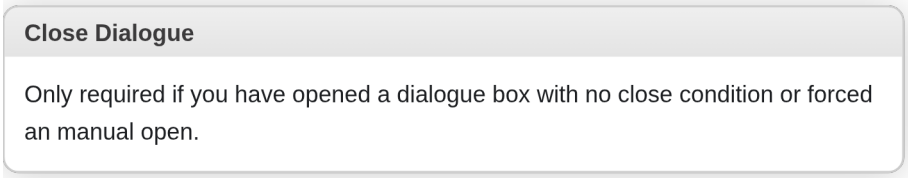
Clear a line of text at the given tile coordinates & size.



- **X**: The horizontal position.
- **Y**: The vertical position.
- **Width**: The total width.
- **Plane**: The plane we want to use

Close Dialogue

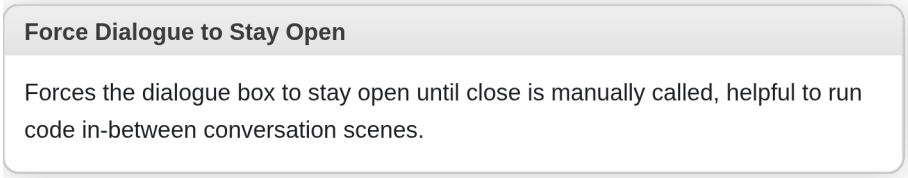
Close a dialogue box currently on screen.



- **Speed**: The movement speed.

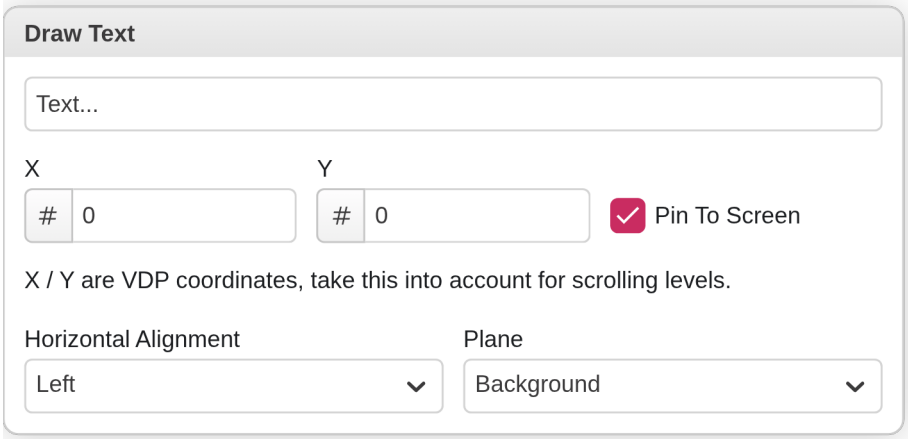
Force Dialogue to Stay Open

Forces the dialogue box to stay open until close is manually called, helpful to run code in-between conversation scenes.



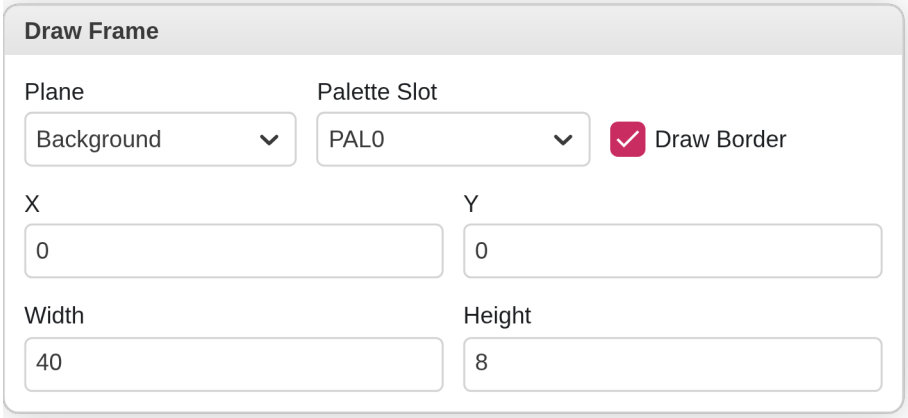
Draw Text

Draw a line of text at the given tile coordinates.



Draw Frame

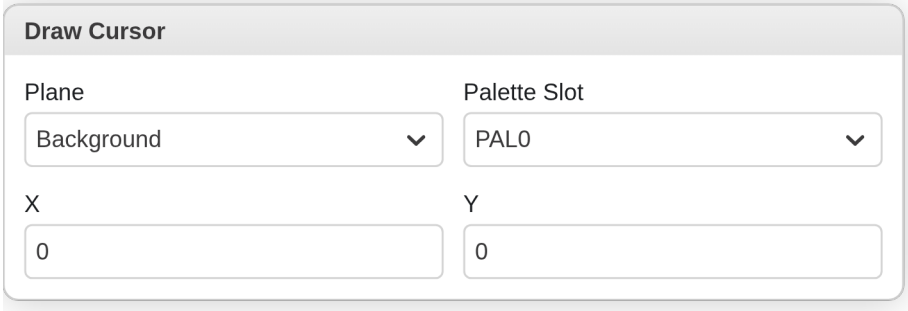
Draws the dialogue frame without content on the given plane, frame tiles have priority enabled.



- **Plane**: The plane we want to use
- **Palette Slot**: The palette slot you want to use.
- **X**: The horizontal position.
- **Y**: The vertical position.

Draw Cursor

Draws the UI cursor on the given plane, if drawn already it clears previous cursor as well.

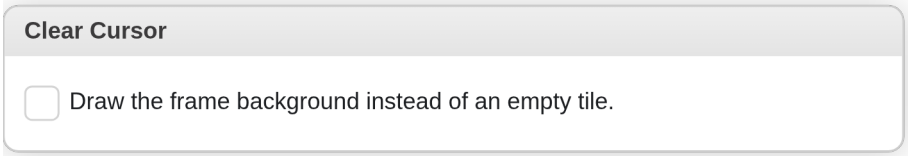


- **Plane**: The plane we want to use

- **Palette Slot**: The palette slot you want to use.
- **X**: The horizontal position.
- **Y**: The vertical position.

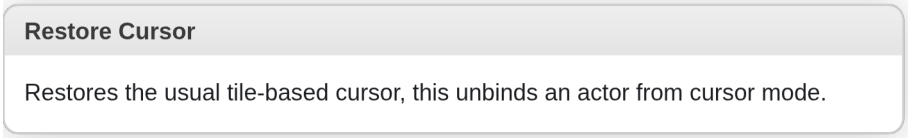
Clear Cursor

Clear the last cursor usage, it can draw an empty tile or the frame background.



Restore Cursor

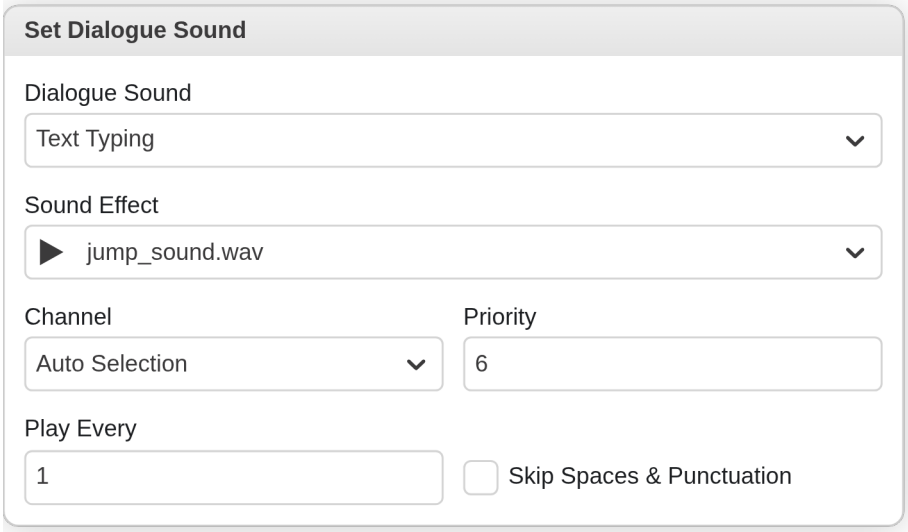
Restores the usual tile-based cursor, this unbinds an actor from cursor mode.



Properties

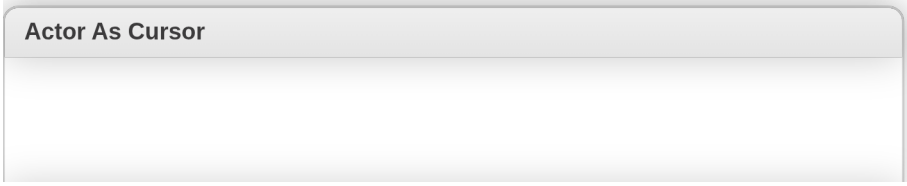
Set Dialogue Sound

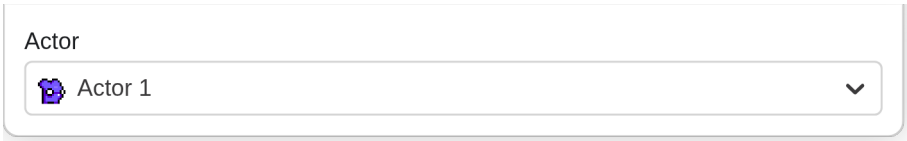
Set the sound effect a dialogue plays while typing text, opening or closing its box, moving to the next page or using a choice menu.



Actor As Cursor

Use the selected actor as cursor on choices display & draw cursor calls.





Load Font

Load the selected font to use rendering text, on Advance Text Mode, this replaces the default font.



Set Font

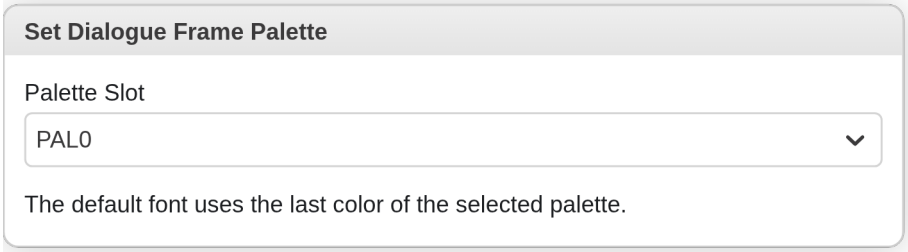
Set the currently used font for text, equivalent of Load Font event.



- **Font**: The font to use for drawing text.

Set Dialogue Frame Palette

Set the palette that will be used by the dialogue frame rendering.



- **Palette Slot**: The palette slot you want to use.

Set Text Priority

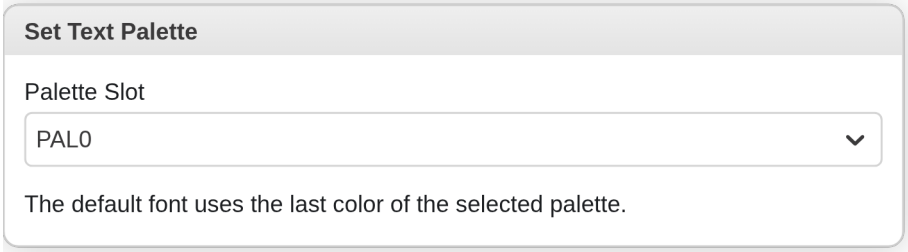
Set the priority that will be used by the text rendering.



- **Priority**: If the text will be rendered with priority.

Set Text Palette

Set the palette that will be used by the text rendering.



- **Palette Slot**: The palette slot you want to use.

Set Dialogue Cursor

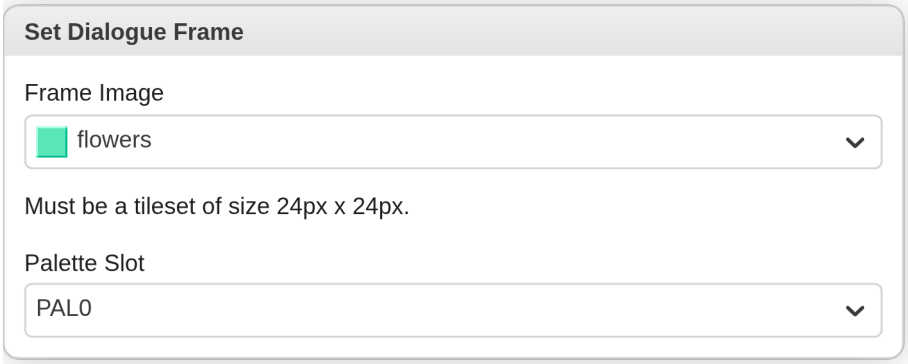
Set a tileset to replace the dialogue cursor image.



- **Frame Image**: The tileset to use as a dialogue frame.

Set Dialogue Frame

Set a tileset to replace the dialogue frame image.



- **Frame Image**: The tileset to use as a dialogue frame.
- **Palette Slot**: The palette slot you want to use.

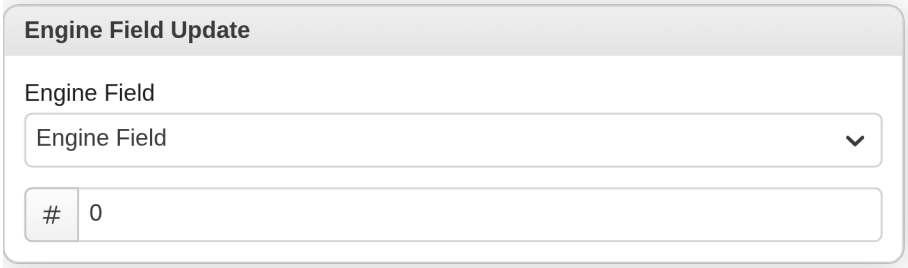
Engine Fields

Events for reading and updating engine field values at runtime.

Actions

Engine Field Update

Change the value of a Engine Field.



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

Variables

Store Engine Field In Variable

Store the value of an Engine Field in a variable.

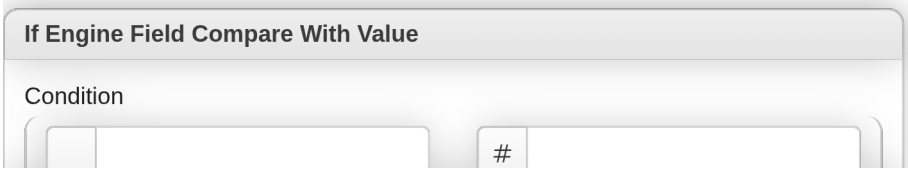


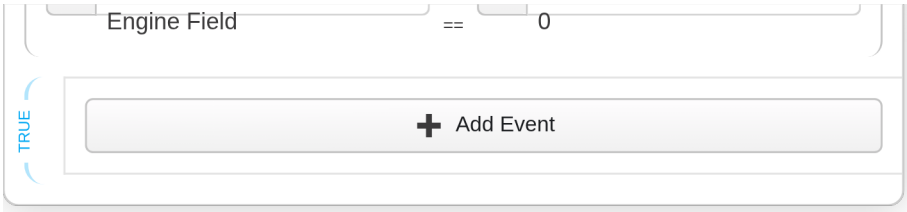
- **Engine Field**: The Engine Field to update.
- **Variable**: The variable to use.

Control Flow

If Engine Field Compare With Value

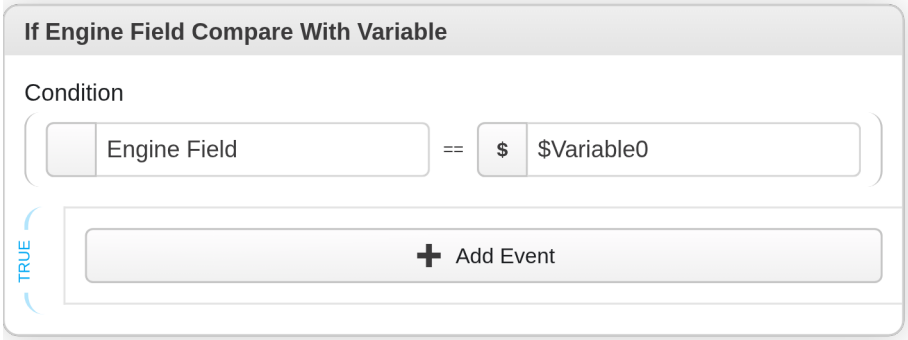
Conditionally run part of the script based on the value of an engine field compared with a value.





If Engine Field Compare With Variable

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



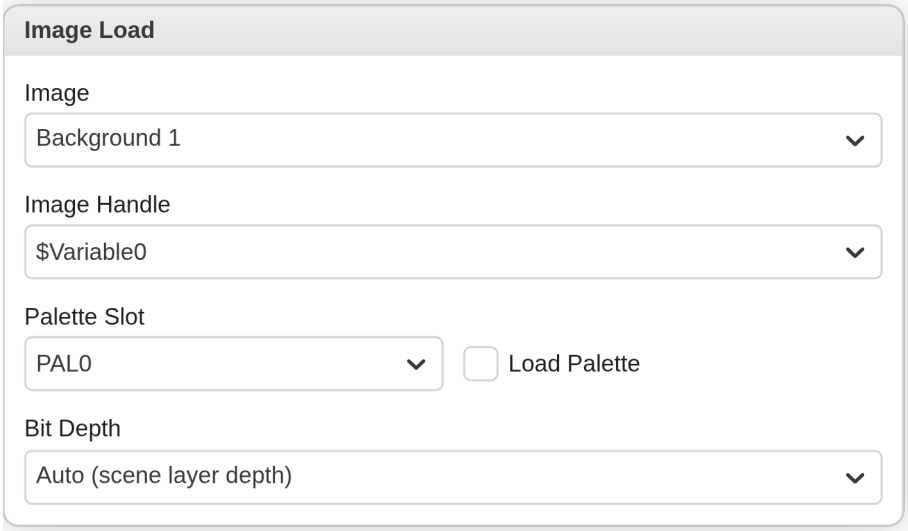
Graphics

Events for controlling graphical elements such as palettes, overlays, and background layers.

Image

Image Load

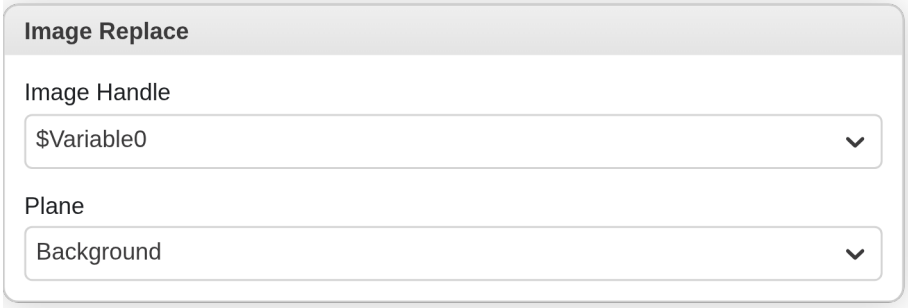
Load an image into the VDP for later usage.



- **Image**: The image to load into the VDP.
- **Image Handle**: The image handle to operate with the image. You can use this handle to draw or unload the image.
- **Palette Slot**: The palette slot you want to use.

Image Replace

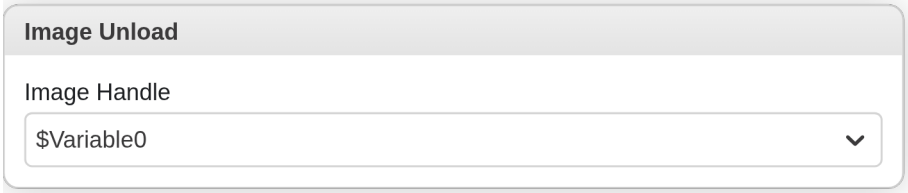
Replaces the initial image used on the scene with the new one, useful for scroll automatic processing as well as Scene Redraws, etc.



- **Image Handle**: The image handle to operate with the image. You can use this handle to draw or unload the image.
- **Plane**: The plane we want to use

Image Unload

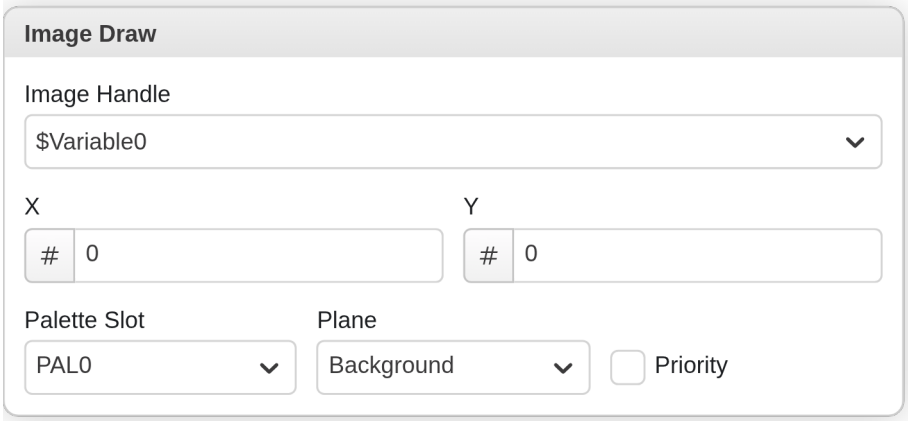
Unload an image from VDP.



- **Image Handle**: The image handle to operate with the image. You can use this handle to draw or unload the image.

Image Draw

Draw an image loaded into the VDP.



- **Image Handle**: The image handle to operate with the image. You can use this handle to draw or unload the image.
- **X**: The horizontal position.
- **Y**: The vertical position.
- **Palette Slot**: The palette slot you want to use.
- **Plane**: The plane we want to use

Image Enable Axis Streaming

Enables image axis streaming during scrolling for the selected axis, only one axis can be used at a time.



- **Image Handle**: The image handle to operate with the image. You can use this handle to draw or unload the image.
- **Direction**: The direction the streaming will be done.

Image Get Layer Handle

Load the selected layer's image handle into a variable to do draw calls.

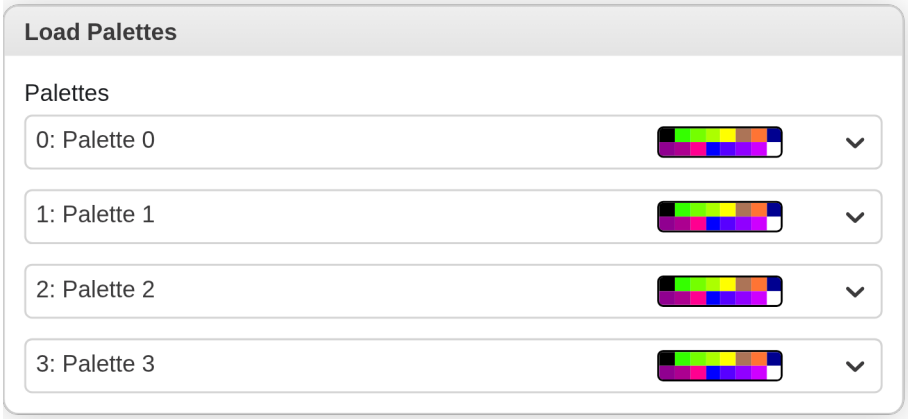


- **Image Handle**: The image handle to operate with the image. You can use this handle to draw or unload the image.
- **Plane**: The plane we want to use

Palettes

Load Palettes

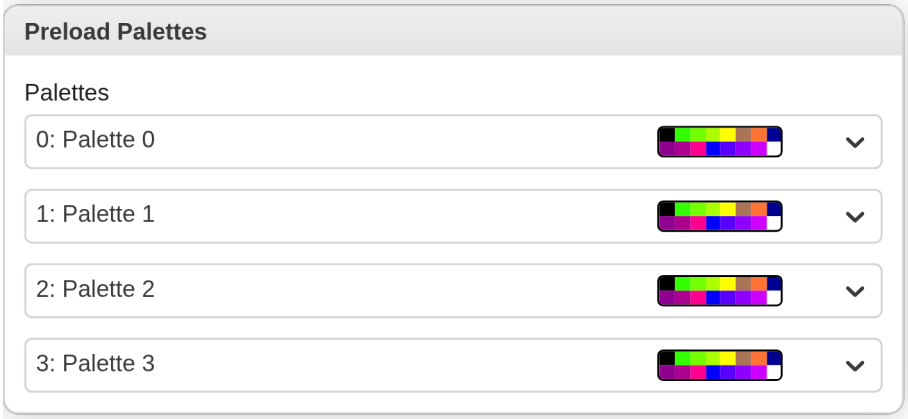
Replace some or all of the current scene's palettes instantly.



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

Preload Palettes

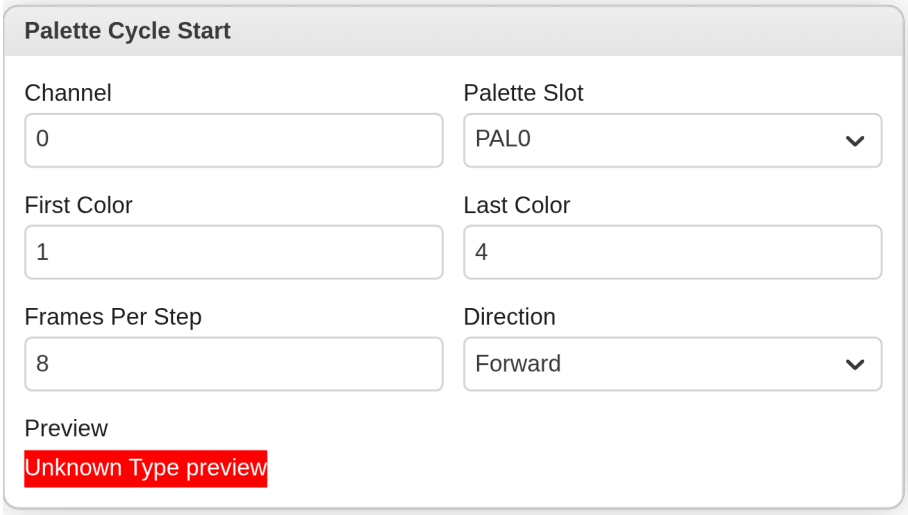
Preload some or all of the current scene's palettes, they won't be replaced into CRAM until you invoke a fade in.



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

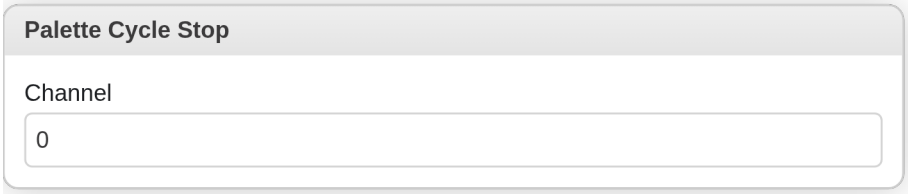
Palette Cycle Start

Rotate a range of colours in a palette slot every few frames on a cycle channel, for waterfalls, lava, conveyor belts or neon signs. Starting a channel that is already running replaces it. The cycle pauses while a palette fade runs.



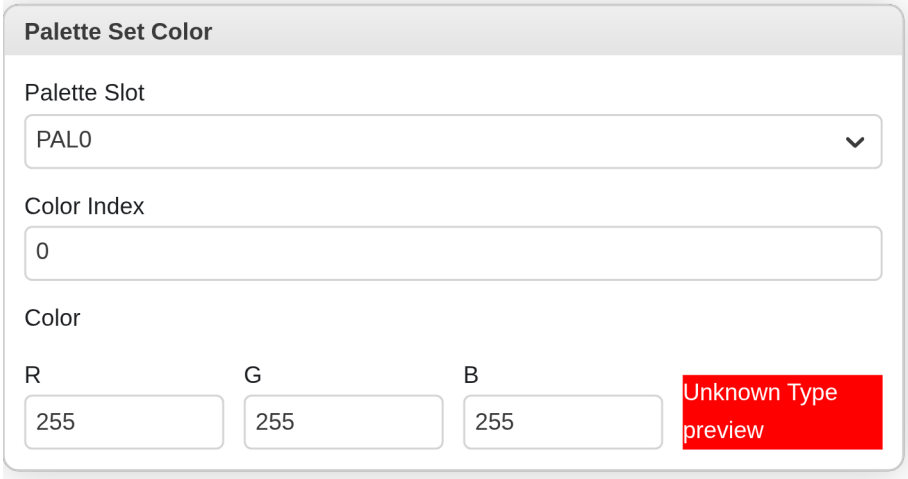
Palette Cycle Stop

Stop a palette cycle channel. The colours stay where the cycle left them.



Palette Set Color

Change the selected color from the selected palette to the values given. It's faster than RGB method due to colors are compiled and not calculated on runtime.

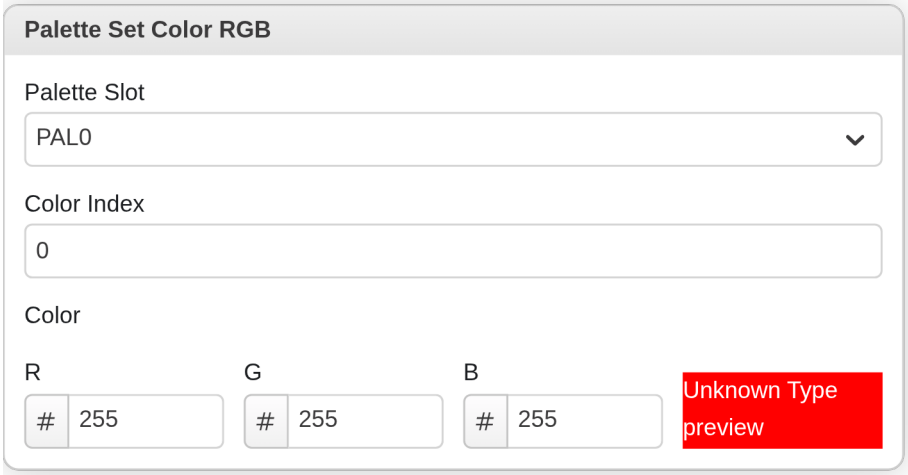


- **Palette Slot**: The palette slot you want to use.

- **Color Index**: The color index to use inside the palette.

Palette Set Color RGB

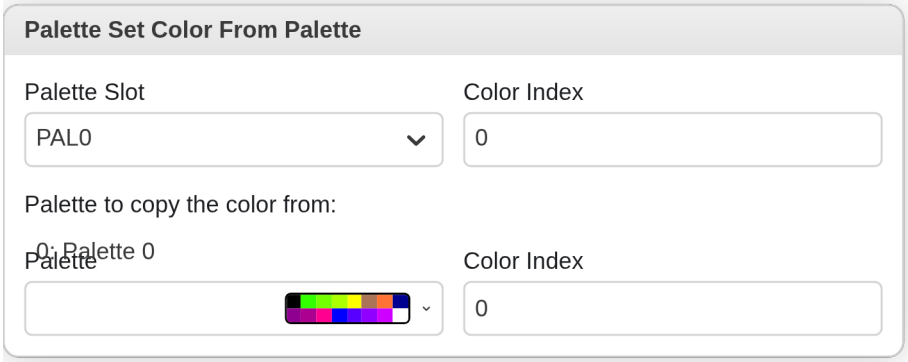
Change the selected color from the selected palette to the RGB values given.



- **Palette Slot**: The palette slot you want to use.
- **Color Index**: The color index to use inside the palette.

Palette Set Color From Palette

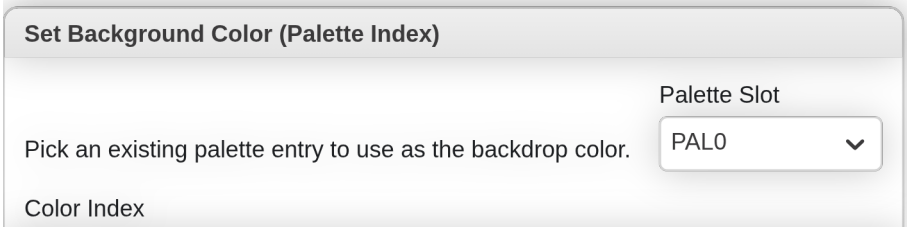
Change the selected color from the selected palette to the selected one.



- **Palette Slot**: The palette slot you want to use.
- **Color Index**: The color index to use inside the palette.
- **Palette**: The new palette to use.

Set Background Color (Palette Index)

Point the MD backdrop register at an existing palette entry. MD-only. The backdrop will match whatever RGB is currently loaded in that slot.





- **Palette Slot**: The palette slot you want to use.
- **Color Index**: The color index to use inside the palette.

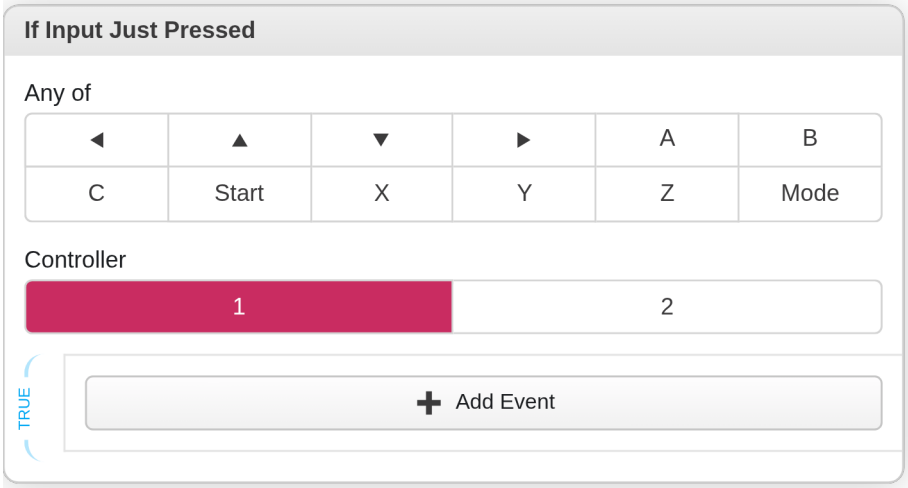
Input

Events for detecting button presses and player input.

Control Flow

If Input Just Pressed

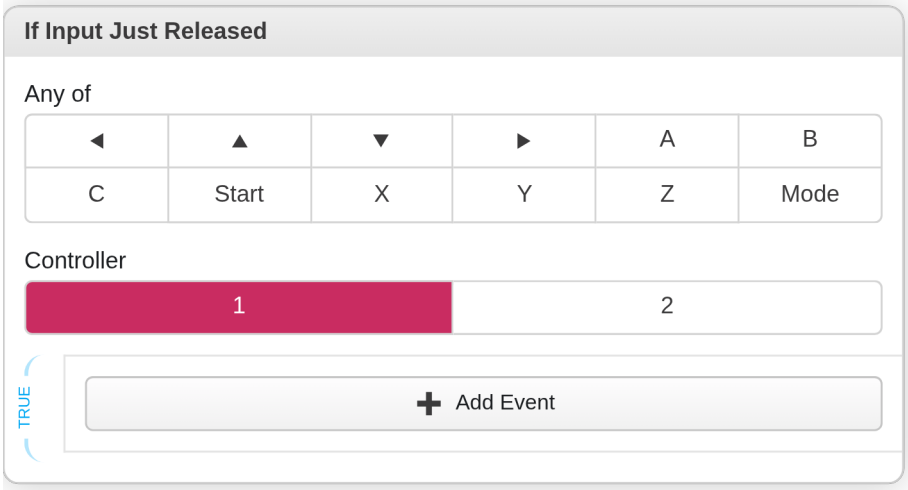
Conditionally run part of the script if the specified joypad button is just pressed.



- **Any of**: The input buttons to check.
- **Controller**: The controller index to check.

If Input Just Released

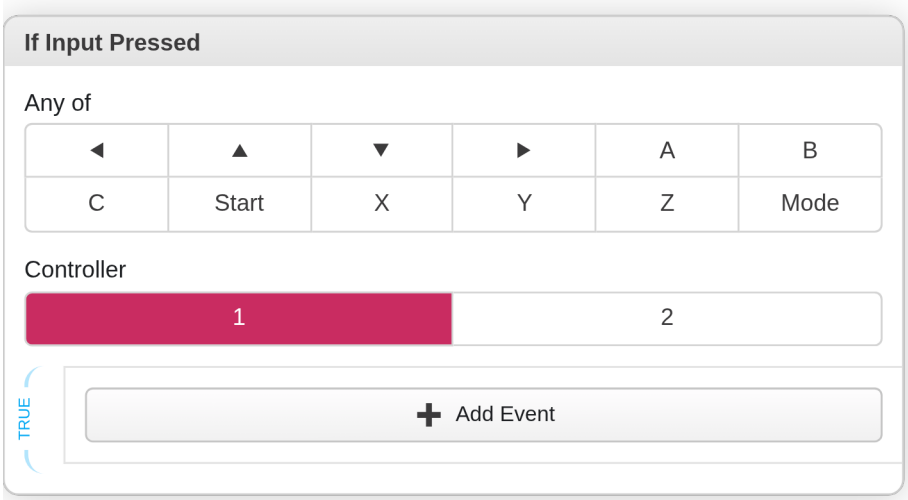
Conditionally run part of the script if the specified joypad button is just released.



- **Any of**: The input buttons to check.
- **Controller**: The controller index to check.

If Input Pressed

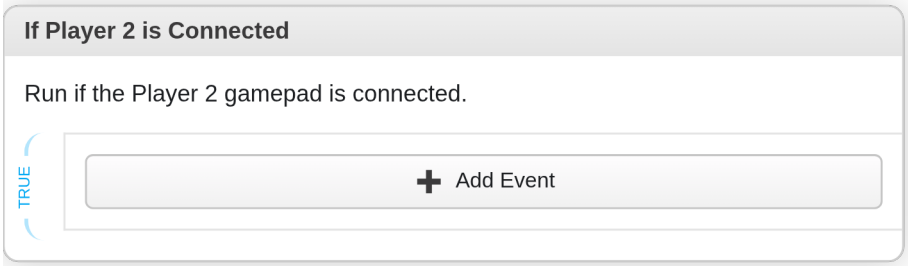
Conditionally run part of the script if the specified joypad button is currently pressed.



- **Any of**: The input buttons to check.
- **Controller**: The controller index to check.

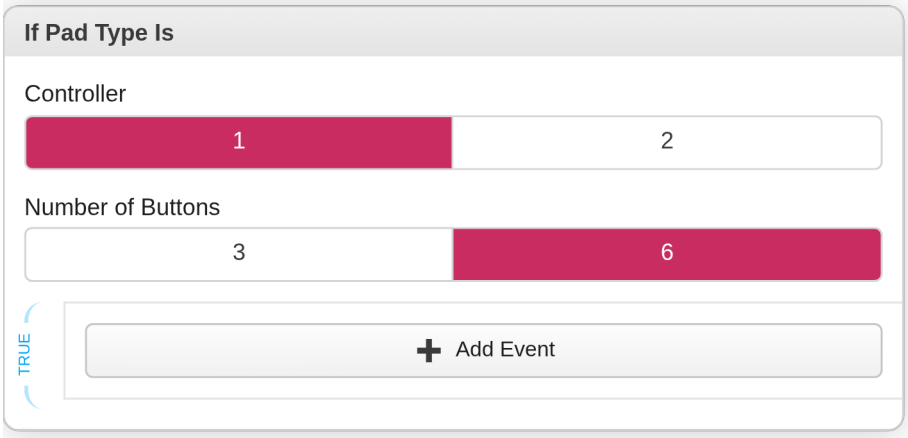
If Player 2 is Connected

Conditionally run part of the script if the gamepad for Player 2 is connected.



If Pad Type Is

Conditionally run part of the script if the gamepad connected to the given controller port is a 3 button or a 6 button pad.



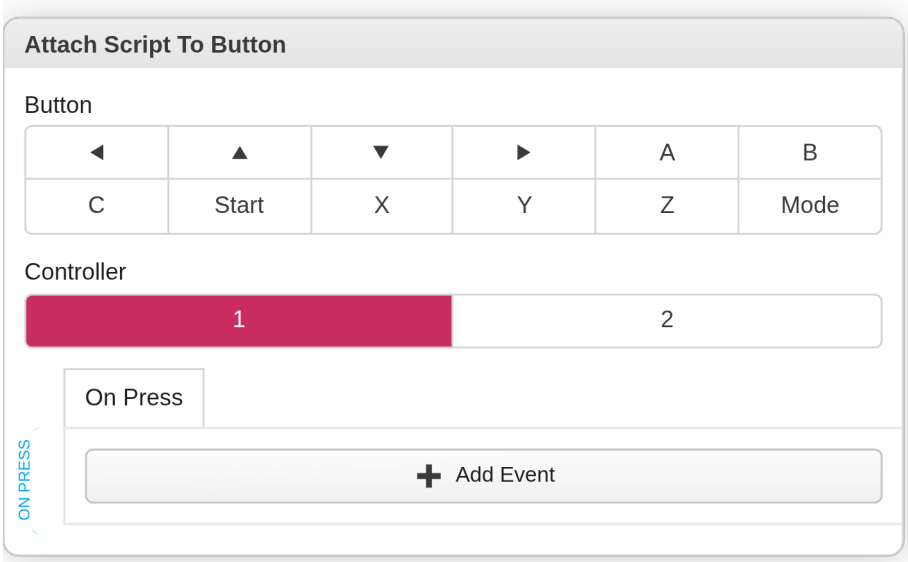
- **Controller**: The controller index to check.
- **Number of Buttons**: The number of buttons the gamepad must have.

Only available on Mega Drive / Genesis.

Script

Attach Script To Button

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



- **Button**: The joypad button to check.
- **Controller**: The controller index to check.
- **On Press**: The script to run when the button is pressed.

Pause Script Until Input Pressed

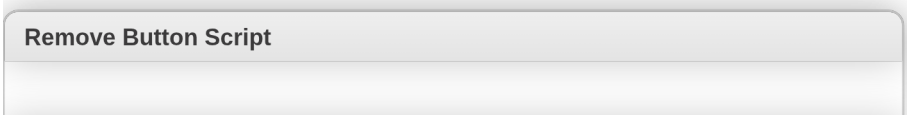
Pauses the script until one of the specified joypad buttons are pressed.

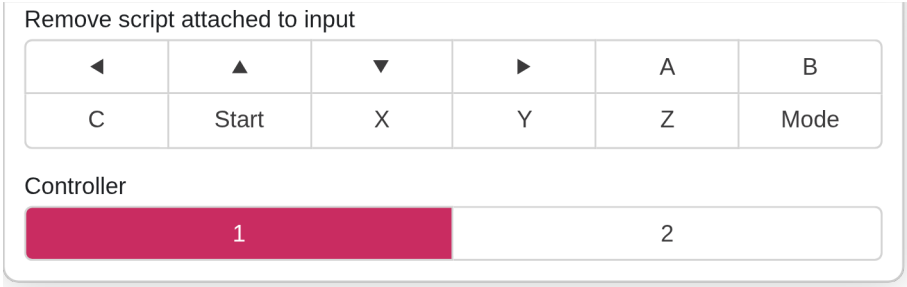


- **Any of**: The input buttons to check.
- **Controller**: The controller index to check.

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.
- **Controller**: The controller index to check.

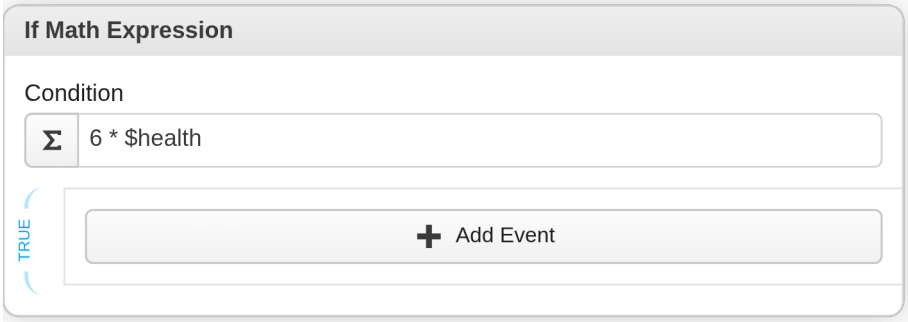
Math

Events for performing mathematical operations and evaluating expressions.

Control Flow

If Math Expression

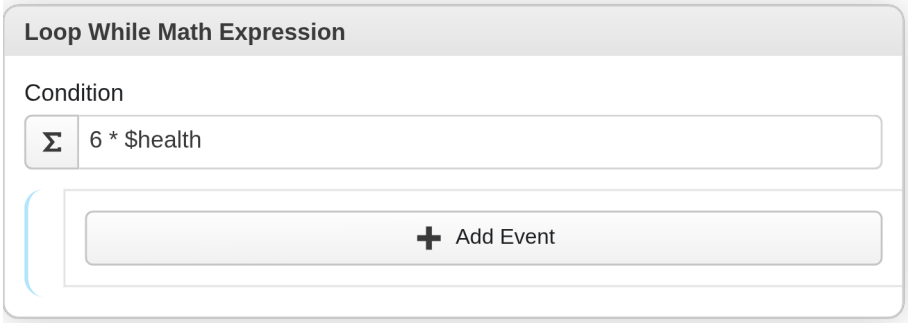
Conditionally execute part of the script if the specified math expression evaluates to true.



- **Condition**: The condition to evaluate.
- **True**: The script to run if the condition is true.

Loop While Math Expression

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

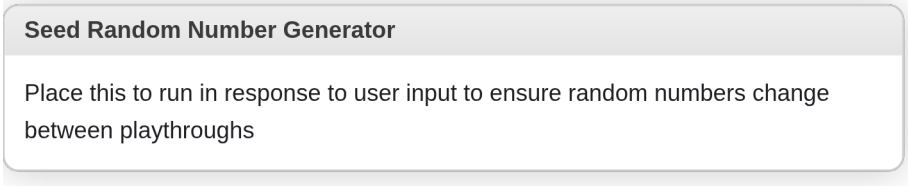


- **Condition**: The condition to evaluate.

Random

Seed Random Number Generator

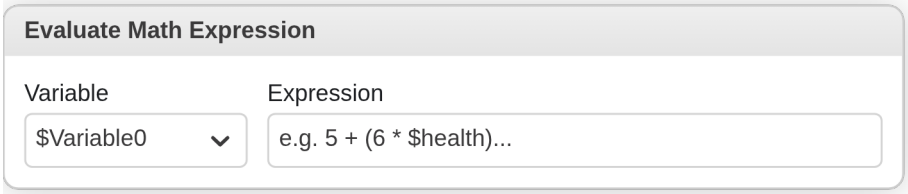
Place this to run in response to user input to ensure random numbers change between playthroughs.



Variables

Evaluate Math Expression

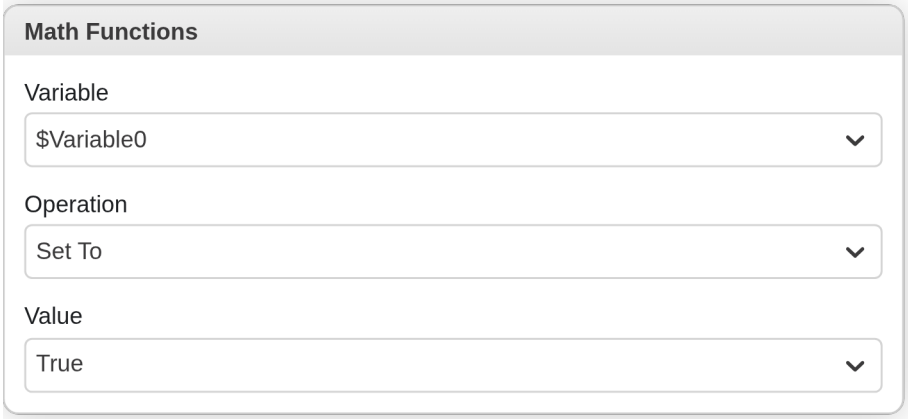
Set a variable to the result of evaluating a math expression.



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

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.

Miscellaneous

Events for miscellaneous operations such as comments, threads, and system functions.

Actions

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.



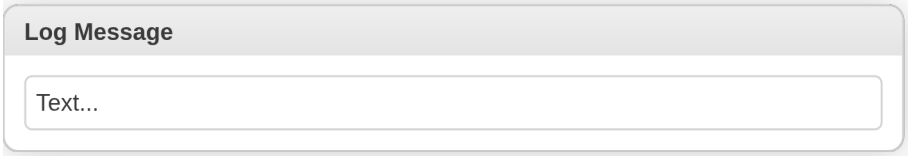
Display FPS

Display FPS on a corner of the screen on the text plane.



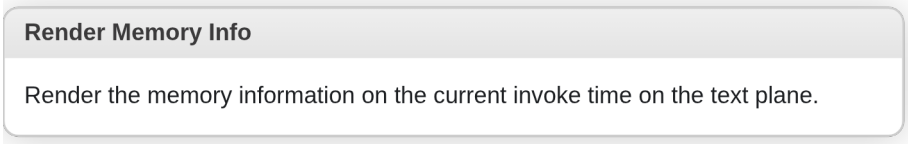
Log Message

Prints a debug log on the console. Useful for debugging.



Render Memory Info

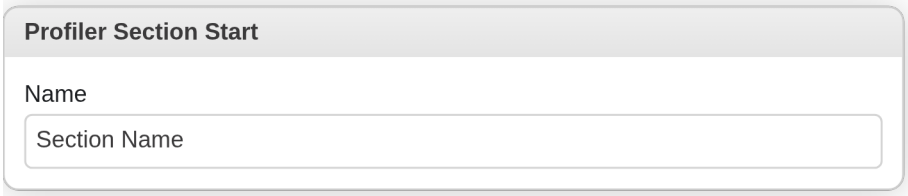
Render the memory information on the current invoke time on the text plane.



Debug

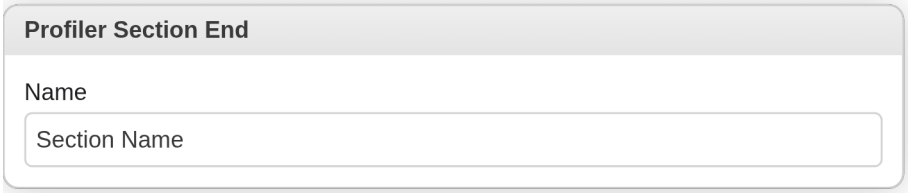
Profiler Section Start

Start measuring CPU cycles for a custom profiler section. Must have a matching Profiler Section End with the same name.



Profiler Section End

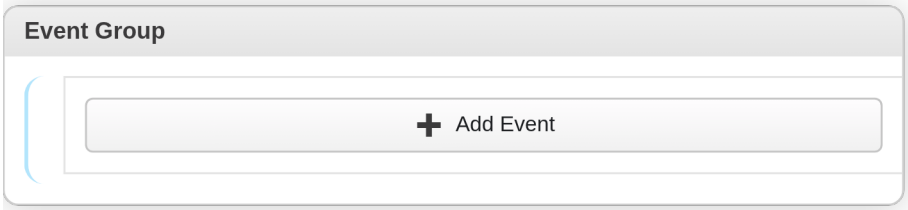
Stop measuring CPU cycles for a custom profiler section. Must have a matching Profiler Section Start with the same name.



Setup

Event Group

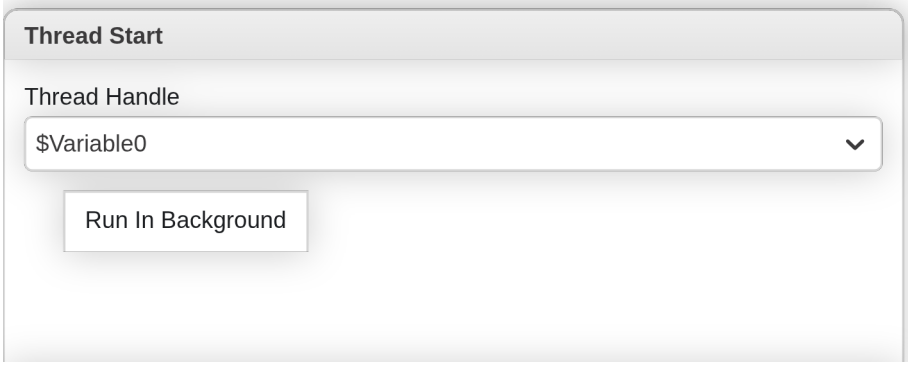
Allows you to group together parts of your script for organizational purposes.

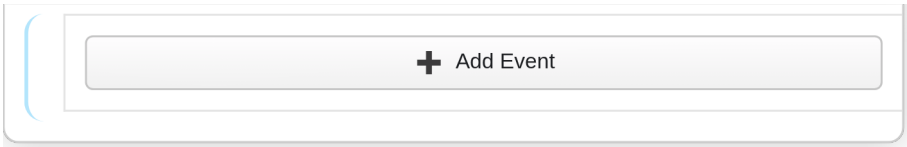


Threads

Thread Start

Run a script in another thread.

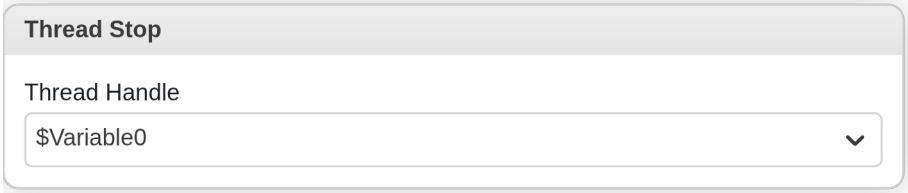




- **Thread Handle**: The handle for the thread you want to stop.

Thread Stop

Stop a thread based on the selected thread handle.



- **Thread Handle**: The handle for the thread you want to stop.

Advanced

Inline C Code

Insert raw C code directly into the compiled output. Use with caution — no validation beyond basic syntax checks is performed.



- **C Code**: Raw C code to inject into the compiled output.

> **CAUTION**
>
> This event injects raw C code directly into the compiled output. The editor performs basic syntax checks (mismatched parentheses, braces, brackets, and missing semicolons) but does not validate the code beyond that. Incorrect code will cause compilation errors. Use this event only if you are familiar with C and the engine's internal API.

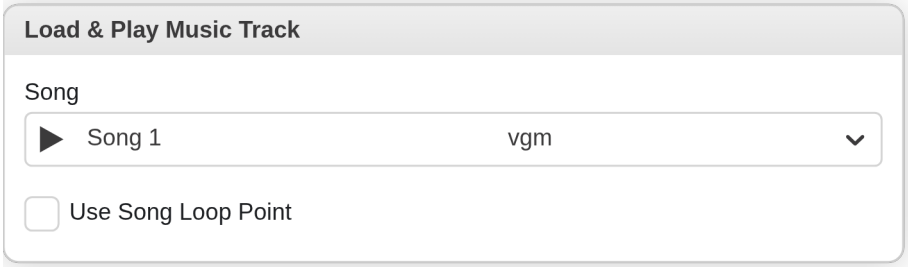
Music & Sound Effects

Events for playing and controlling music and sound effects.

Actions

Load & Play Music Track

Load & 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.
- **Use Song Loop Point**: Jump back to the song's loop point when it reaches the end. The loop point has to be set in the tracker; a song without one still stops at the end. PCM music repeats from the start.

Load Music Track

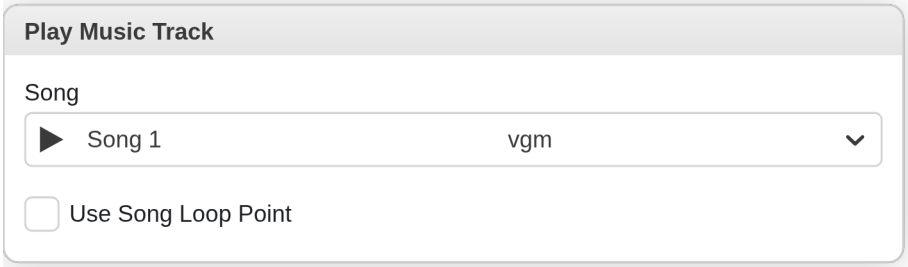
Load a music file. You can play it later with 'Play Music Track'.



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

Play Music Track

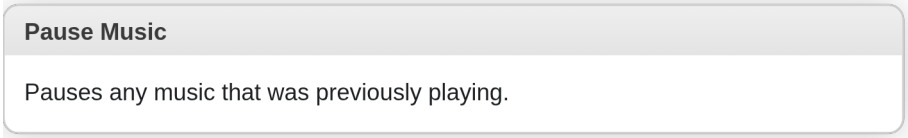
Play the loaded music file. If you play a new song while another song is playing, the old song will stop automatically.



- **Song**: The song to play.
- **Use Song Loop Point**: Jump back to the song's loop point when it reaches the end. The loop point has to be set in the tracker; a song without one still stops at the end. PCM music repeats from the start.

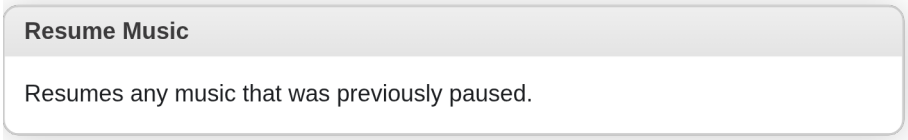
Pause Music

Pause any currently playing music, only works with VGM files.



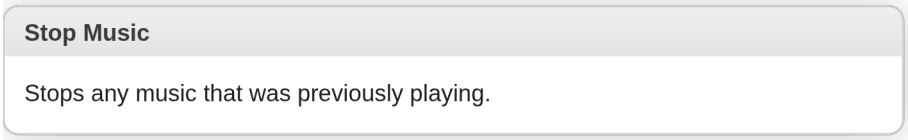
Resume Music

Resume any currently playing music, only works with VGM files.



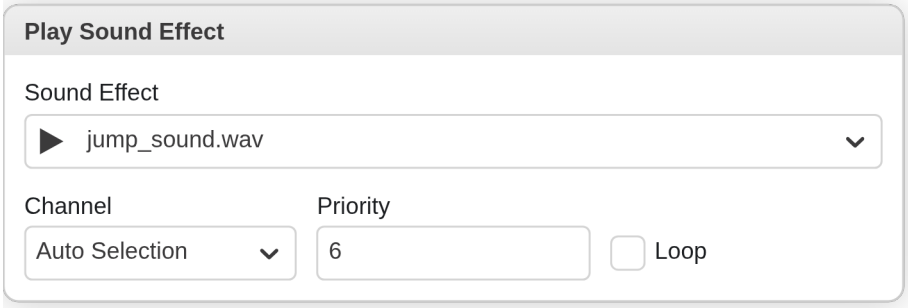
Stop Music

Stops any currently playing music.



Play Sound Effect

Play a sound effect, choose a .WAV file from '/assets/sounds'.

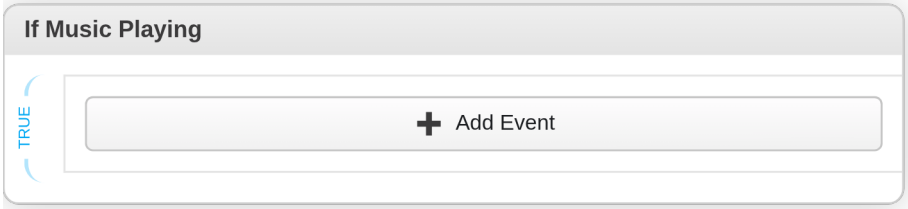


- **Sound Effect**: The sound effect to play. Can choose from files within '/assets/sounds'.
- **Channel**: Channel to use to play sample.
- **Priority**: The priority of the effect, from 0 to 15 where 0 is lowest priority and 15 the highest one. If two sound effects are playing at the same time then higher priority sound effects will take precedence.
- **Loop**: If the sound will loop or not.

Control Flow

If Music Playing

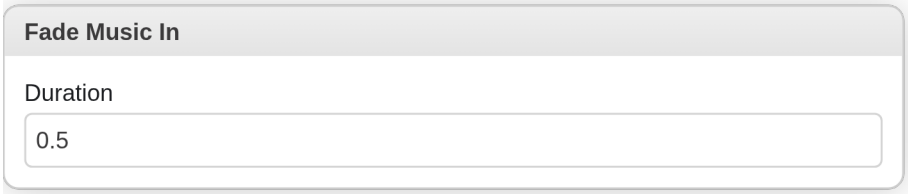
Conditionally run part of the script if music is playing.



Properties

Fade Music In

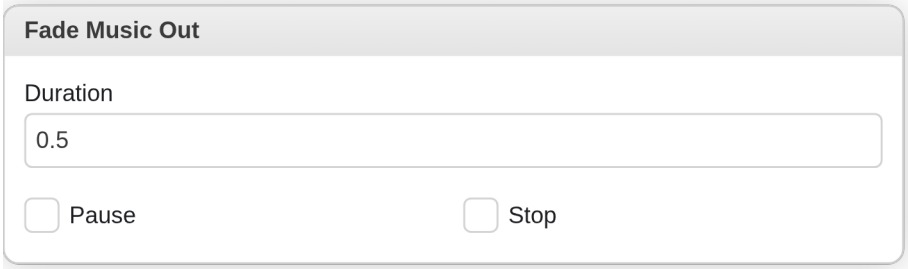
Fade the music from 0 up to the current volume, only works with VGM files. Must be called right after a play or resume.



- **Duration**: The length of time needed to perform the fade in seconds or frames.

Fade Music Out

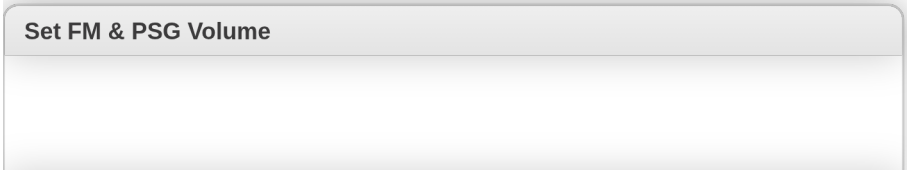
Fade the music to 0 from current levels, only works with VGM files.



- **Duration**: The length of time to pause the script for in seconds or frames.
- **Pause**: If we pause music after fading, only works with VGM files.
- **Stop**: If we stop music after fading, only works with VGM files.

Set FM & PSG Volume

Sets the FM & PSG Volume to the selected values.





- **FM volume.**: The FM volume.
- **PSG volume.**: The PSG volume.

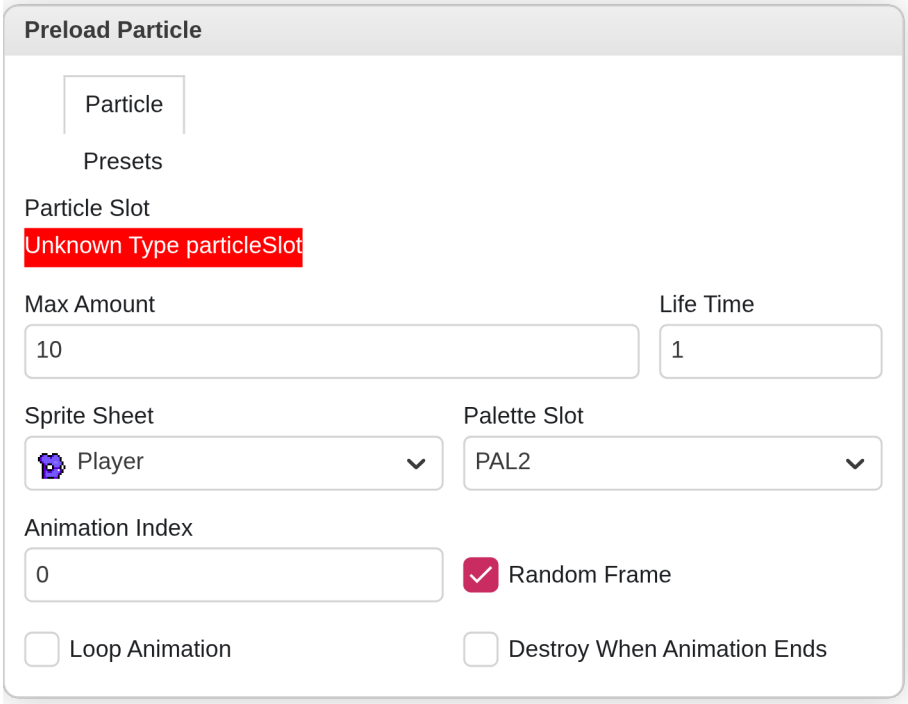
Particles

Events for setting up and spawning particle effects.

Actions

Preload Particle

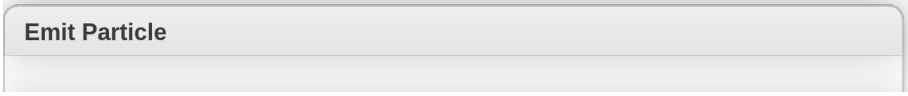
Setup a particle type to later use via emit particle.

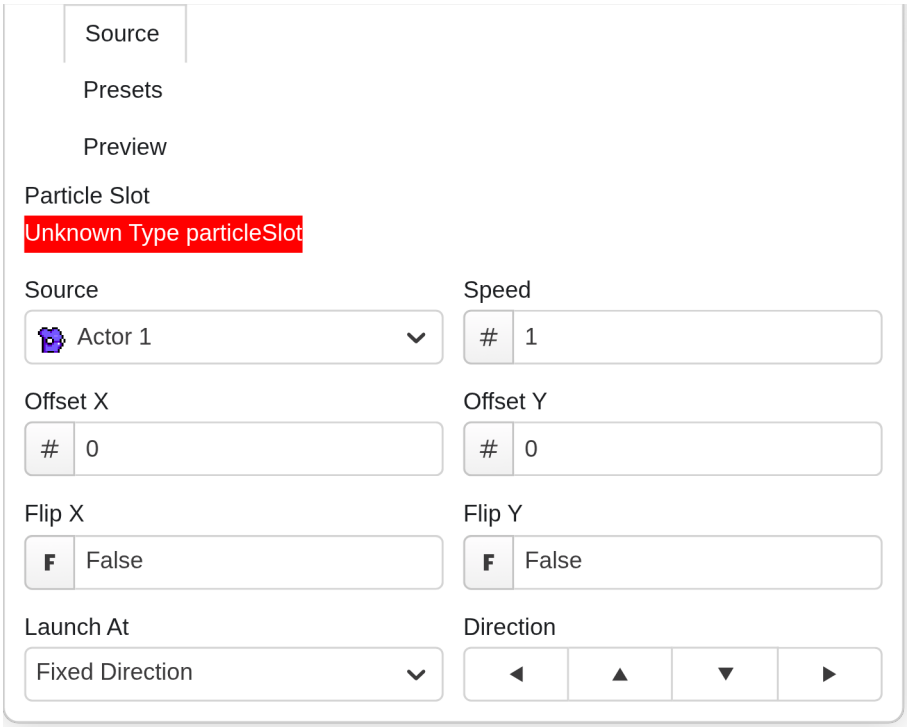


- **Particle Slot**: The particle slot to use.
- **Max Amount**: The maximum amount of this particles to preload, if you try to emit more, nothing will happen.
- **Sprite Sheet**: The sprite to use for rendering the particle.
- **Palette Slot**: The palette slot you want to use.
- **Life Time**: The amount of time in seconds that the particle will live for.
- **Destroy When Animation Ends**: If the particle will be destroyed when the animation ends, it will ignore the life time if it's longer.
- **Animation Index**: The animation index from the sprite sheet.
- **Random Frame**: If set, the sprite will get a random frame when emitted.
- **Loop Animation**: Set if animation should loop.

Emit Particle

Emit a particle from an actor in a specified direction.

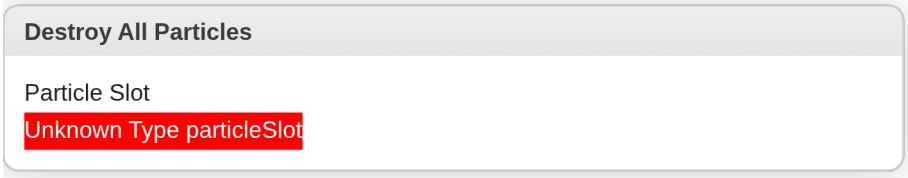




- **Particle Slot**: The particle slot to use.
- **Source**: The actor to launch the projectile from.
- **Speed**: The movement speed.
- **Offset X**: The horizontal offset from the source actors position to start launching the particle.
- **Offset Y**: The vertical offset from the source actors position to start launching the particle.
- **Flip X**: Flip Sprite on X axis
- **Flip Y**: Flip Sprite on Y axis
- **Direction**: The direction to launch the particle. Can either be a fixed direction or based on an actor's current direction.

Destroy All Particles

Destroy all particles on screen.

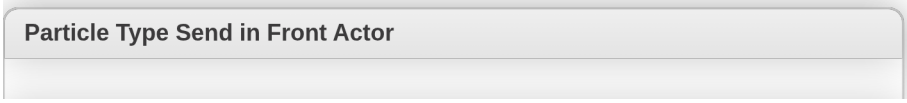


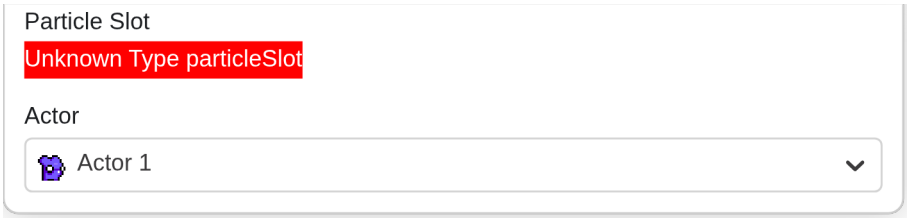
- **Particle Slot**: The particle slot to use.

Sprite

Particle Type Send in Front Actor

Send the particle sprites in front of the selected actor so it renders on top of it.





- **Particle Slot**: The particle slot to use.
- **Actor**: The actor you want to use.

Particle Type Send Behind Actor

Send the particle sprites behind the selected actor so it renders below it.



- **Particle Slot**: The particle slot to use.
- **Actor**: The actor you want to use.

Particle Type Set Depth

Set the particle depth on the rendering queue. From -255 to 255. Lower values means rendering higher on the queue.



- **Particle Slot**: The particle slot to use.
- **Value**: The value you want to set.

Pathfinding

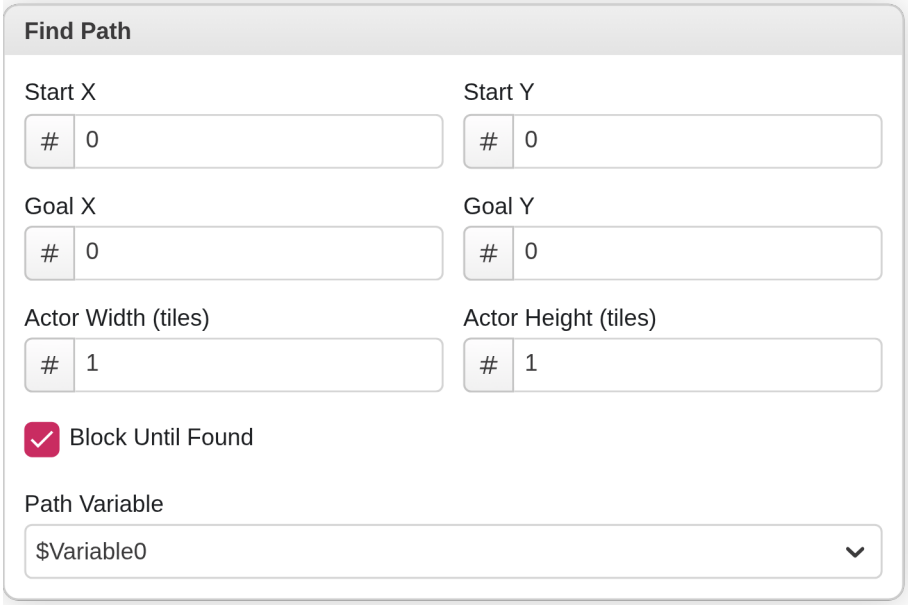
Events for computing and following paths on the scene collision map.

For a detailed overview of how pathfinding works, see the Pathfinding Module documentation.

Actions

Find Path

Find a path between two positions using pathfinding.

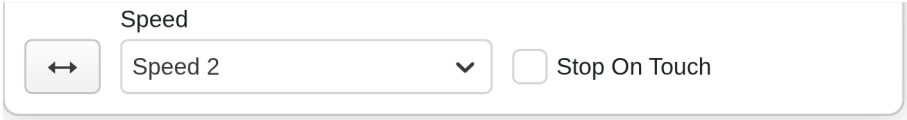


- **Start X / Start Y**: The starting X position for pathfinding.
- **Goal X / Goal Y**: The goal X position for pathfinding.
- **Actor Width (tiles)**: Actor footprint width in collision tiles (e.g. 2 for a 16px actor with 8px tiles).
- **Actor Height (tiles)**: Actor footprint height in collision tiles (e.g. 2 for a 16px actor with 8px tiles).
- **Block Until Found**: If enabled, the script will wait until the path is fully computed before continuing.
- **Path Variable**: Variable to store the path slot index.

Follow Path

Make an actor follow a previously found path.

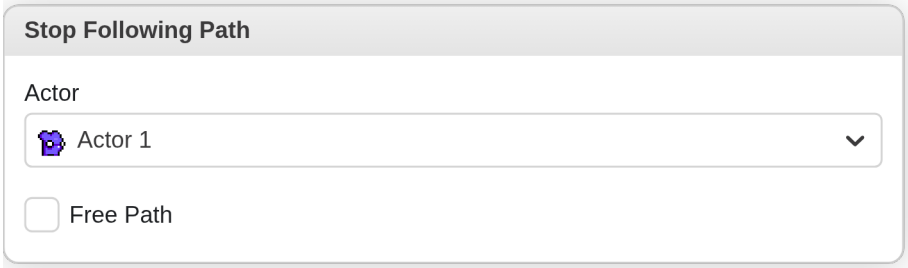




- **Actor**: The actor you want to use.
- **Path Variable**: Variable to store the path slot index.
- **Movement Type**: Choose if should move in horizontal/vertical axis first or if it should move diagonally to destination.
- **Speed**: The movement speed.
- **Stop On Touch**: Stop movement if we touch any solid surface in the movement direction, useful to avoid blocking the script.

Stop Following Path

Stop an actor from following a path.



- **Actor**: The actor you want to use.
- **Free Path**: Also release the path slot when stopping.
- **Path Variable**: Variable to store the path slot index.

Free Path

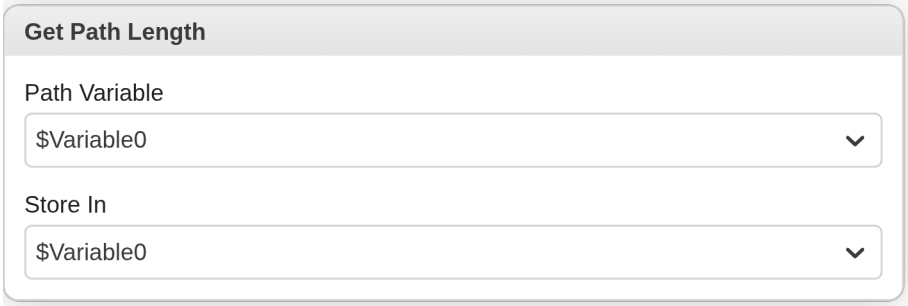
Release a path slot back to the pool.



- **Path Variable**: Variable to store the path slot index.

Get Path Length

Get the number of steps in a path.



- **Path Variable**: Variable to store the path slot index.
- **Store In**: Variable to store the result.

Get Direction At Index

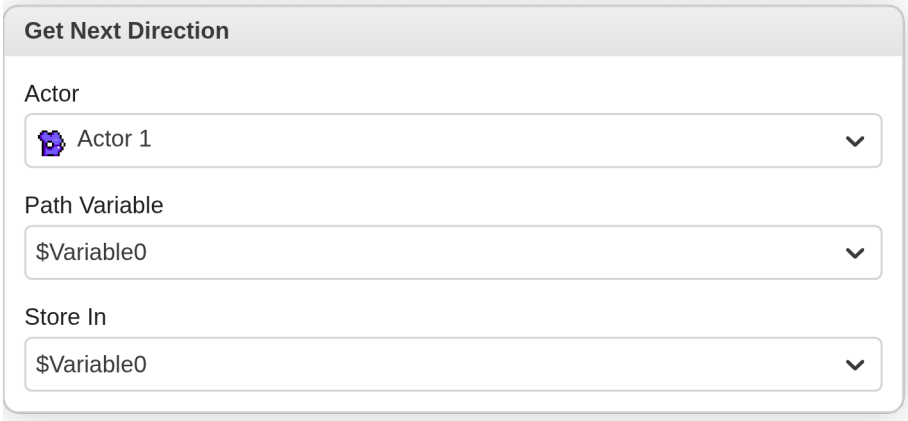
Get the direction at a specific index in a path.



- **Path Variable**: Variable to store the path slot index.
- **Index**: The step index in the path to read.
- **Store In**: Variable to store the result.

Get Next Direction

Get the next direction for an actor following a path.

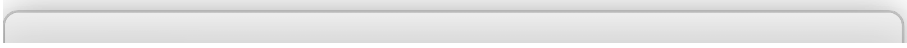


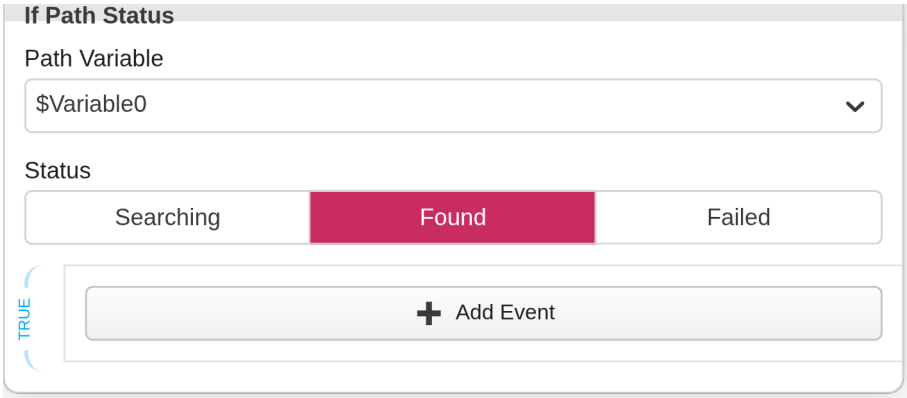
- **Actor**: The actor you want to use.
- **Path Variable**: Variable to store the path slot index.
- **Store In**: Variable to store the result.

Conditions

If Path Status

Conditionally run part of the script based on the status of a path search: still searching, found, or failed (no path, or no free path slot).





- **Path Variable**: Variable to store the path slot index.
- **Status**: The search status to check for.
  - **Searching**: the search for this slot is queued or still running.
  - **Found**: the search reached the goal. The path can be empty when the start is the goal.
  - **Failed**: there is no path, the start or goal is outside the search area, the search was cancelled by a scene change, or *Find Path* found no free path slot.

See Also

- Pathfinding Module - Detailed module documentation, settings, and performance information

Platform

Events for detecting the current platform and toggling platform-specific features.

Actions

Toggle Cheat Mode

Toggle Cheat Mode on or off.

Toggle Cheat Mode

☐ Enabled

Close Game

Closes the game if possible, only works on Desktop export.

Close Game

Closes the game if possible, only works on Desktop export.

Notify Game Completed

Notify that the game has been completed, this is needed for some specific platform exports due to their guidelines.

Notify Game Completed

Notify that the game has been completed, this is needed for some specific platform exports due to their guidelines.

Toggle 4:3 Window

Toggle 4:3 Window if the emulator supports it.

Toggle 4:3 Window

☐ Enabled

Toggle CRT Shader

Toggle CRT Shader if the emulator supports it.

Toggle CRT Shader



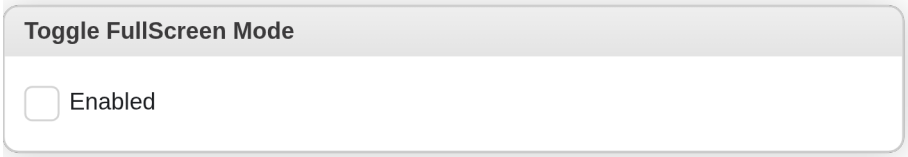
Toggle Enhanced Mode

Toggle Enhanced Mode if the emulator supports it. On TBCE emulator this disables sprite limits and renders foreground under window plane.



Toggle FullScreen Mode

Toggle FullScreen Mode if the emulator supports it.



Toggle Rewind

Toggle Rewind if the emulator supports it. On TBCE emulator this allows to Rewind the game in real time.



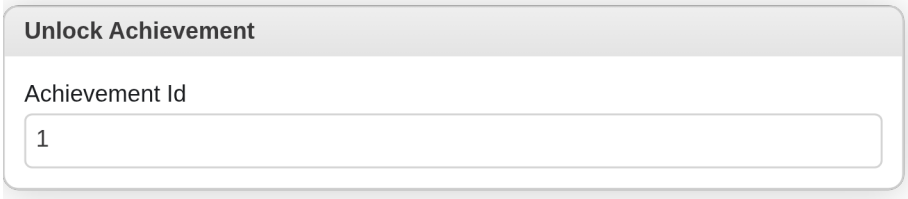
Toggle Wide Screen Mode

Toggle Wide Screen Mode if the emulator supports it.



Unlock Achievement

Unlock an achievement if the emulator supports it.

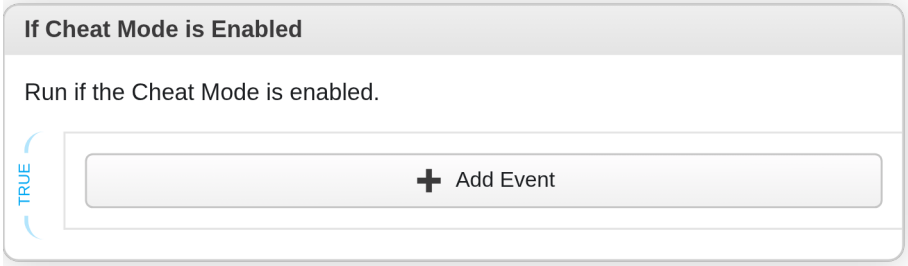


- **Achievement Id**: The numeric achievement id to unlock.

Control Flow

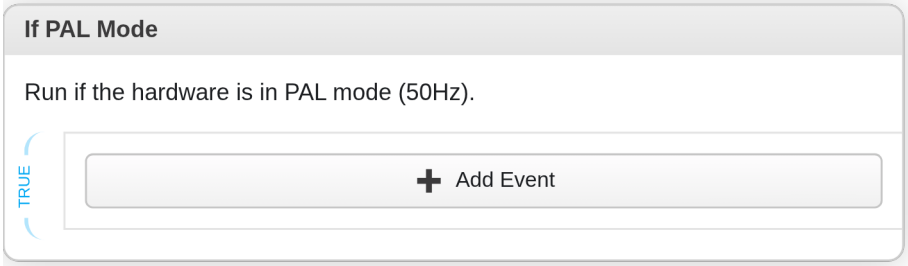
If Cheat Mode is Enabled

Conditionally run part of the script if the Cheat Mode is enabled.



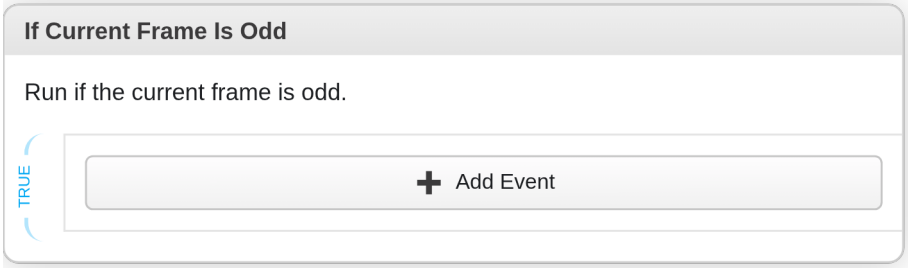
If PAL Mode

Conditionally run part of the script when the hardware is running in PAL mode (50Hz) instead of NTSC (60Hz).



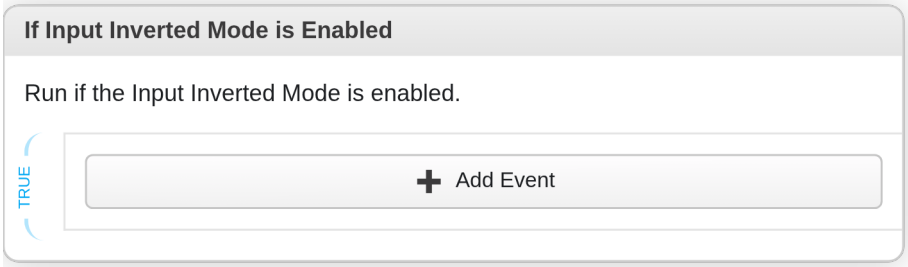
If Current Frame Is Odd

Check if the current frame is odd.



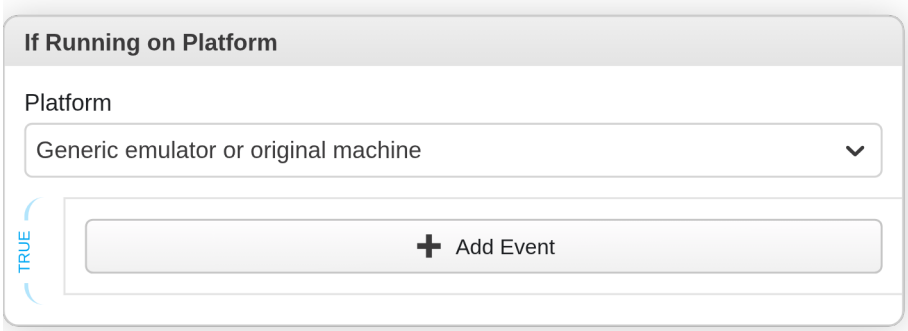
If Input Inverted Mode is Enabled

Conditionally run part of the script if the Input Inverted Mode is enabled.



If Running on Platform

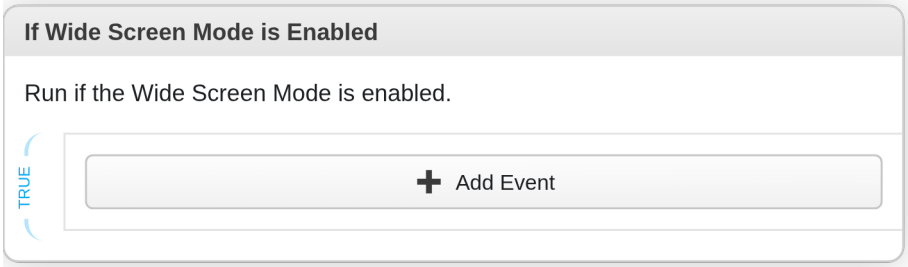
Conditionally run part of the script if the game is being played on the selected platform.



- **Platform**: The platform we want to check against.

If Wide Screen Mode is Enabled

Conditionally run part of the script if the Wide Screen Mode is enabled.



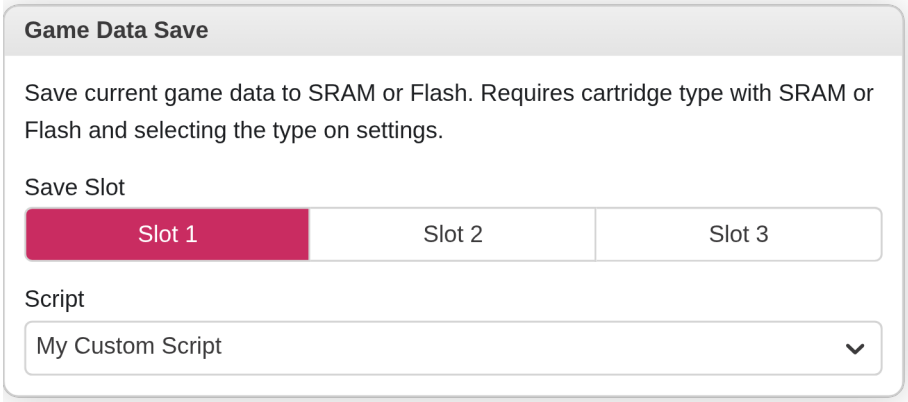
Save Data

Events for saving and loading game data.

Actions

Game Data Save

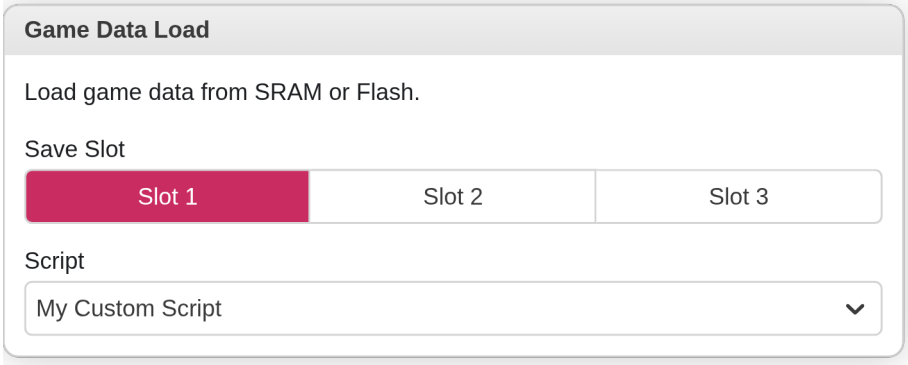
Invoke the script that will perform the data processing to save into the selected slot.



- **Save Slot**: The save slot to use.
- **Script**: The script that will perform the data processing.

Game Data Load

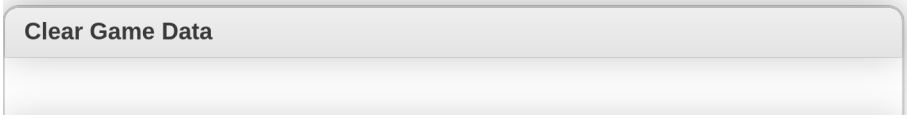
Invoke the script that will perform the data processing to load from the selected slot.

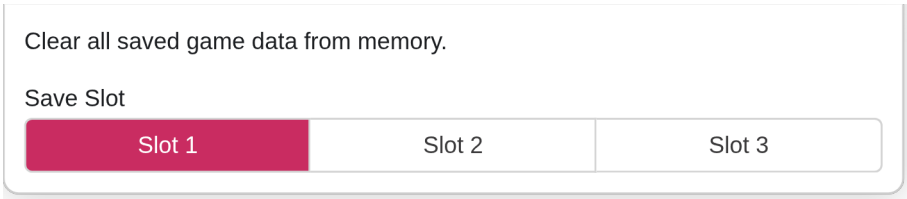


- **Save Slot**: The save slot to use.
- **Script**: The script that will perform the data processing.

Clear Game Data

Remove any previously saved game data in the selected slot.



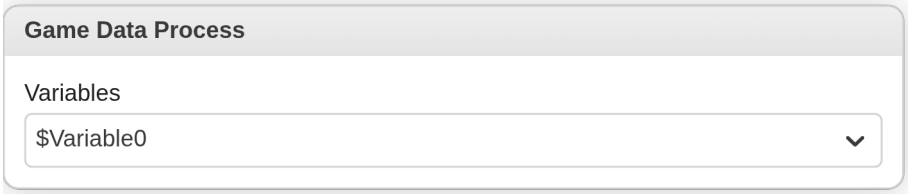


- **Save Slot**: The save slot to use.

Process

Game Data Process

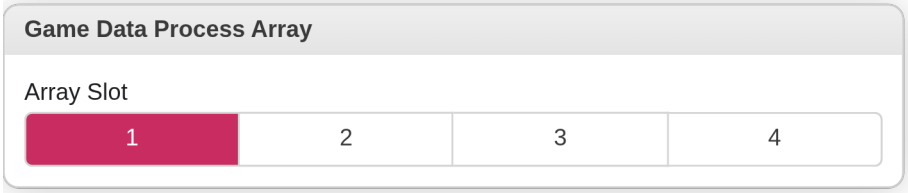
Process the selected variables for loading or saving.



- **Variables**: The list of variables to process, to either load or save.

Game Data Process Array

Process the selected array for loading or saving.

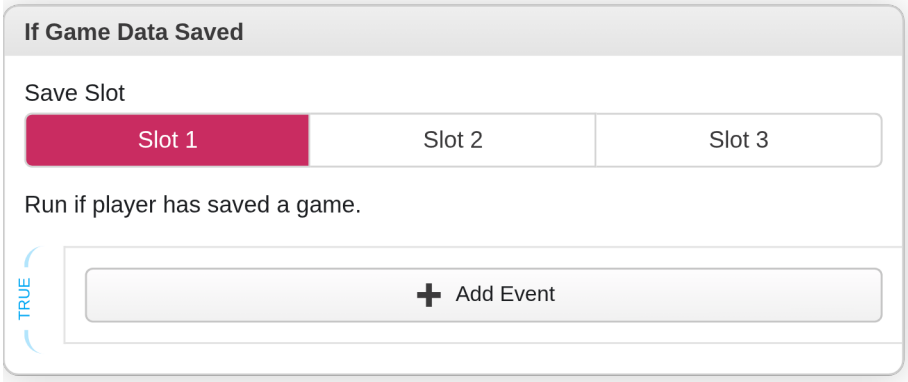


- **Array Slot**: The array slot to use.

Control Flow

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.

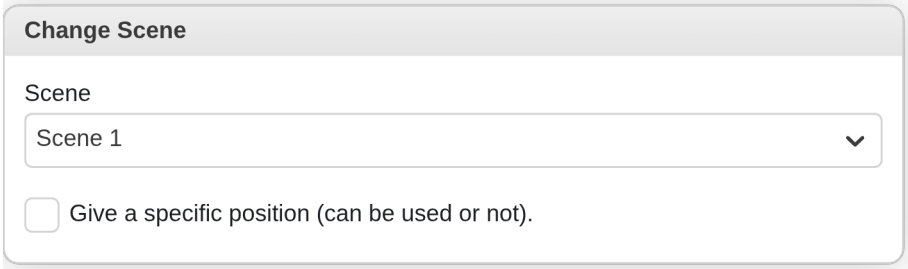
Scene

Events for managing scenes, transitions, scrolling, tiles, and scene properties.

Actions

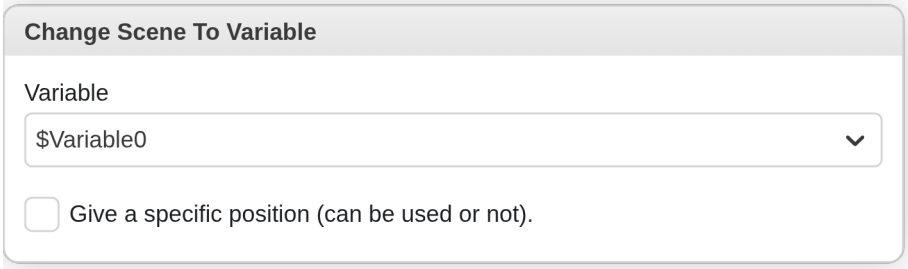
Change Scene

Transition to a new scene with player at a specified position and direction.



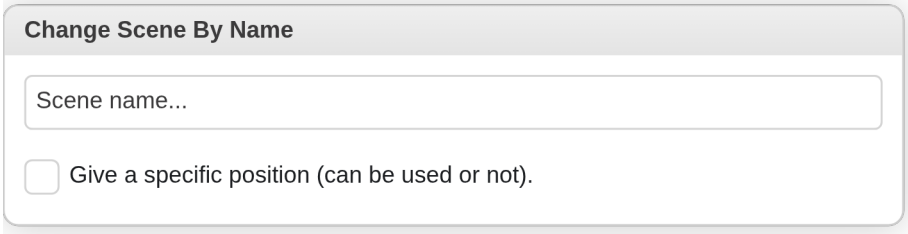
Change Scene To Variable

Transition to the scene index stored in a variable, with player at a specified position and direction.



Change Scene By Name

Transition to a new scene with player at a specified position and direction.



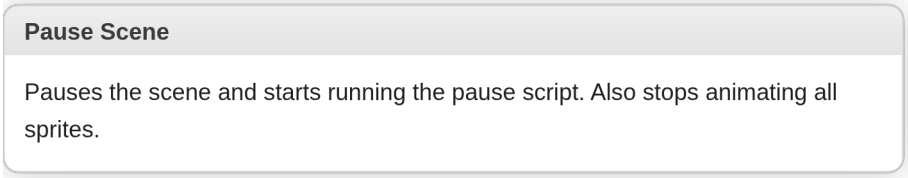
Restart Scene

Restart the current scene.



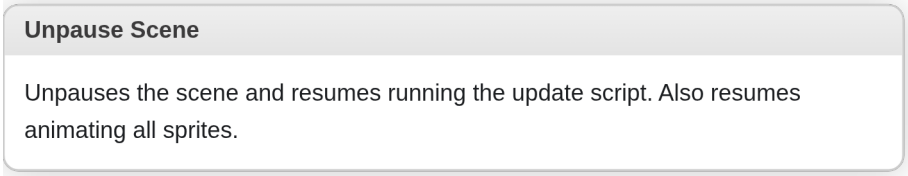
Pause Scene

Pauses the scene and starts running the pause script. Also stops animating all sprites.



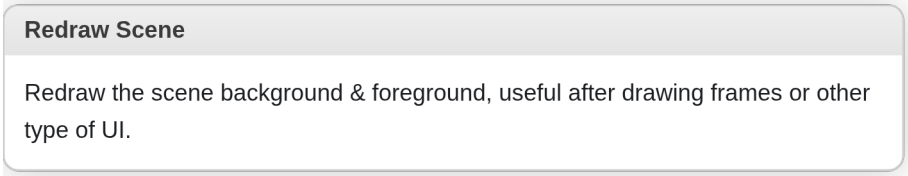
Unpause Scene

Unpauses the scene and resumes running the update script. Also resumes animating all sprites.



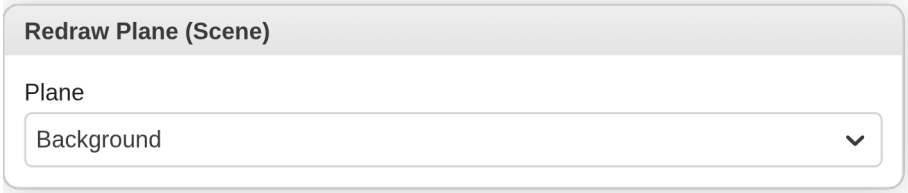
Redraw Scene

Redraw the scene background & foreground, useful after drawing frames or other type of UI.



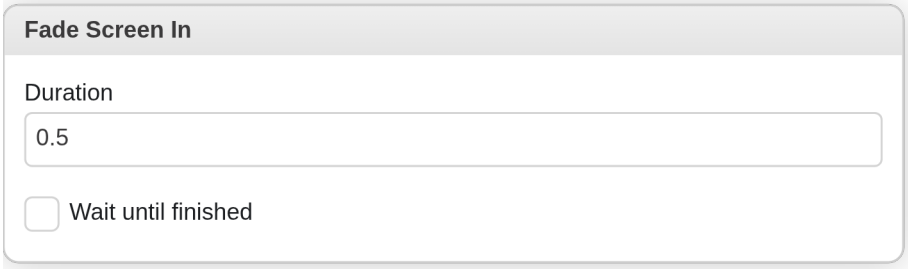
Redraw Plane (Scene)

Redraws a single plane, scrolling it to the camera position.



Fade Screen In

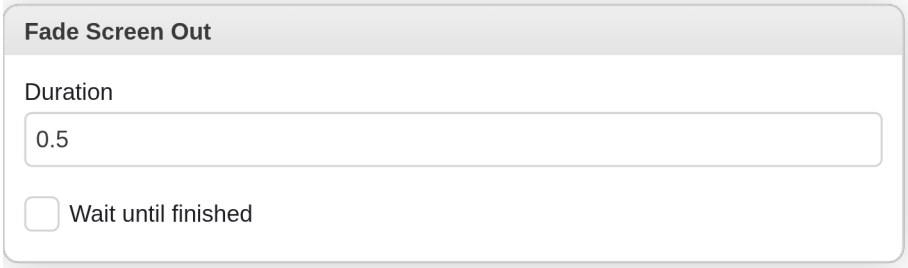
Fade the scene from a blank screen.



- **Duration**: The length of time needed to perform the fade in seconds or frames.
- **Wait until finished**: Set if script should pause until fade has finished.

Fade Screen Out

Fade the scene to a blank screen.

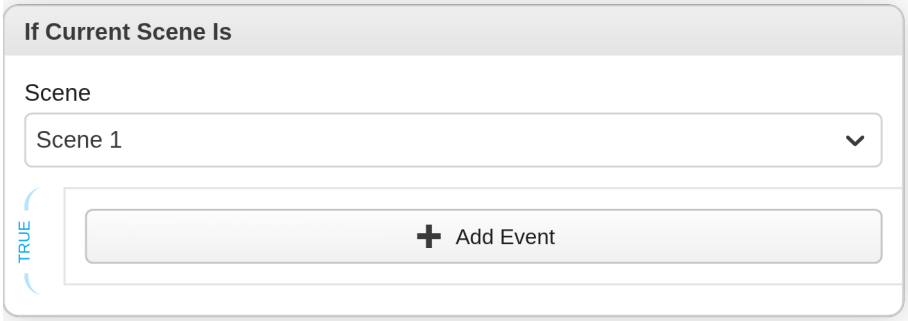


- **Duration**: The length of time needed to perform the fade in seconds or frames.
- **Wait until finished**: Set if script should pause until fade has finished.

Control Flow

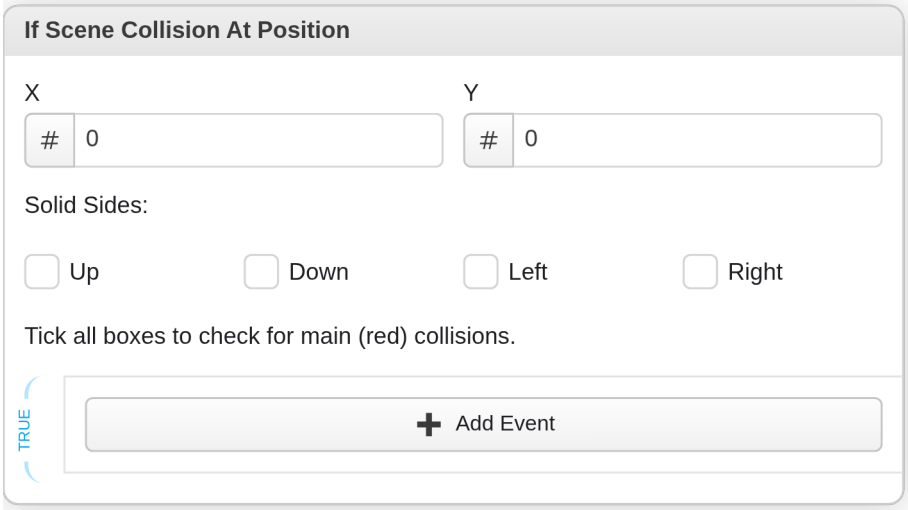
If Current Scene Is

Conditionally run part of the script if the current scene is the selected one.



If Scene Collision At Position

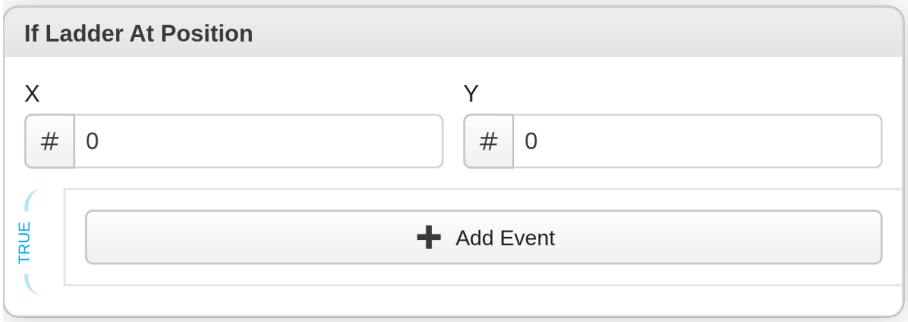
Conditionally run part of the script if the specified collision type is found at a specified position.



- **X**: The horizontal position.
- **Y**: The vertical position.

If Ladder At Position

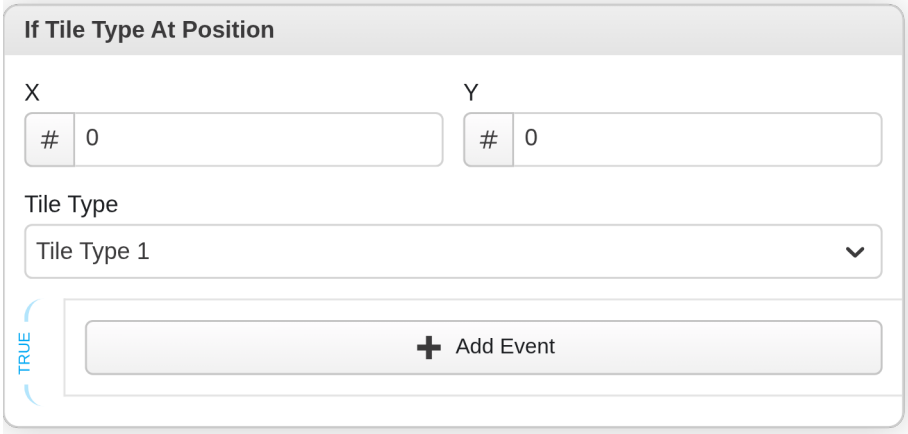
Conditionally run part of the script if a ladder is at a specified position.



- **X**: The horizontal position.
- **Y**: The vertical position.

If Tile Type At Position

Conditionally run part of the script if the selected Tile Type is at a specified position.

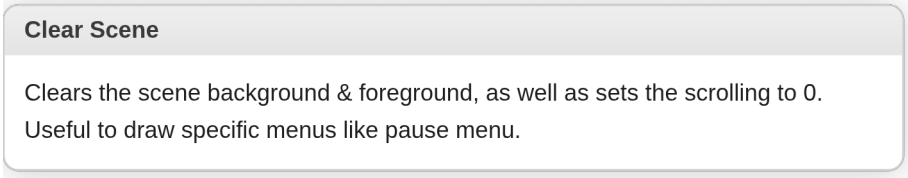


- **X**: The horizontal position.
- **Y**: The vertical position.
- **Tile Type**: The collision tile type to check for.

Graphics

Clear Scene

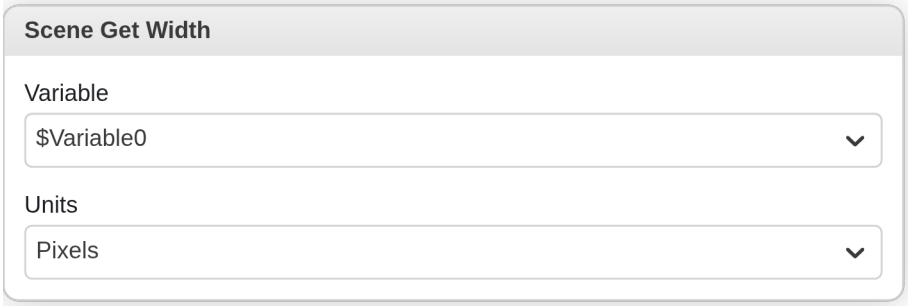
Clears the scene background & foreground, as well as sets the scrolling to 0. Useful to draw specific menus like pause menu.



Properties

Scene Get Width

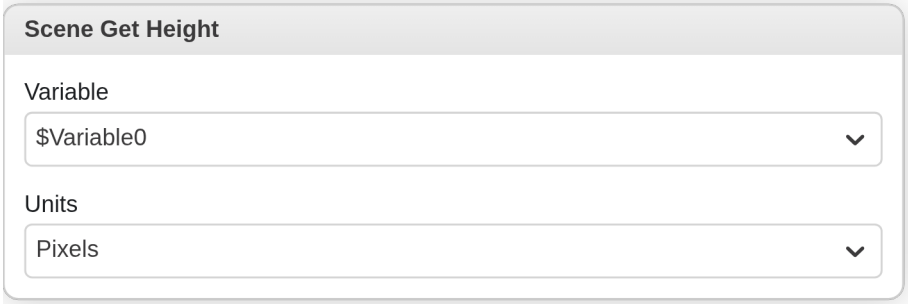
Load the scene width into a variable.



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

Scene Get Height

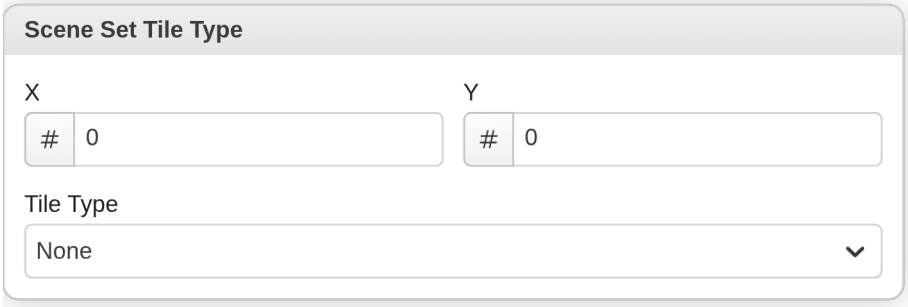
Load the scene height into a variable.



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

Scene Set Tile Type

Set the given tile type on the given tile coordinates.

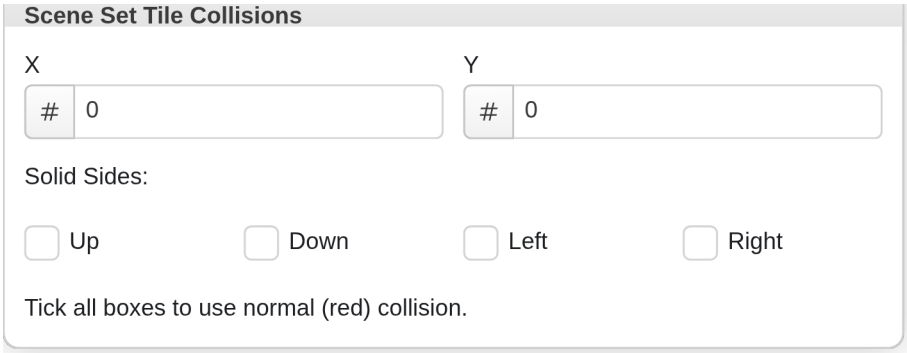


- **X**: The horizontal position.
- **Y**: The vertical position.

Scene Set Tile Collisions

Enables collisions on the given tile coordinates.





- **X**: The horizontal position.
- **Y**: The vertical position.

Scroll

Scroll Set Mode

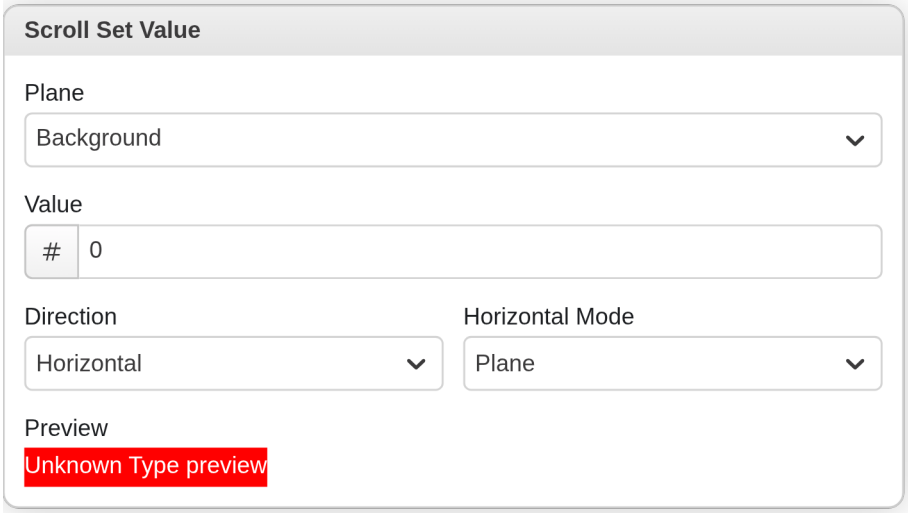
Set the scroll mode.



- **Horizontal**: The Horizontal scroll mode.
- **Vertical**: The Vertical scroll mode.

Scroll Set Value

Set the scroll to a new value.

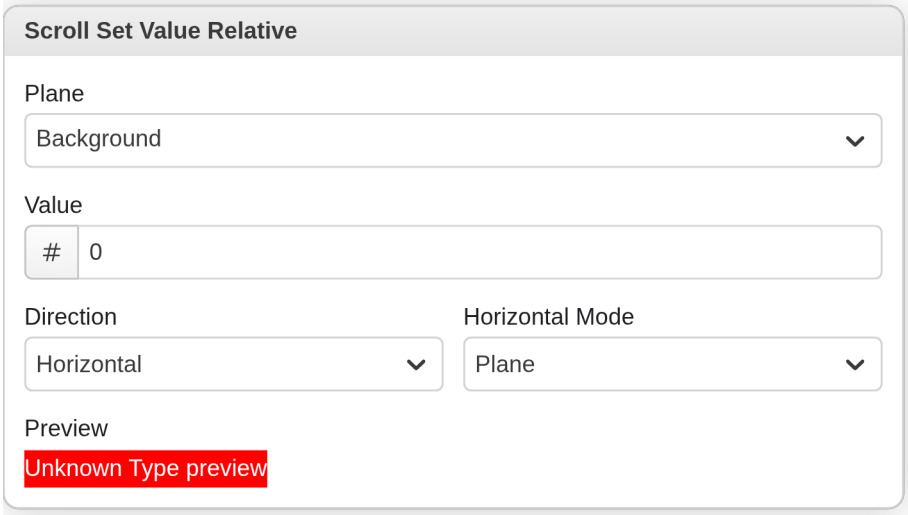


- **Plane**: The plane we want to use

- **Value**: The value you want to set.
- **Direction**: The scroll direction to set.
- **Horizontal Mode**: The Horizontal scroll mode.

Scroll Set Value Relative

Set the scroll to a new value relative to its previous value.

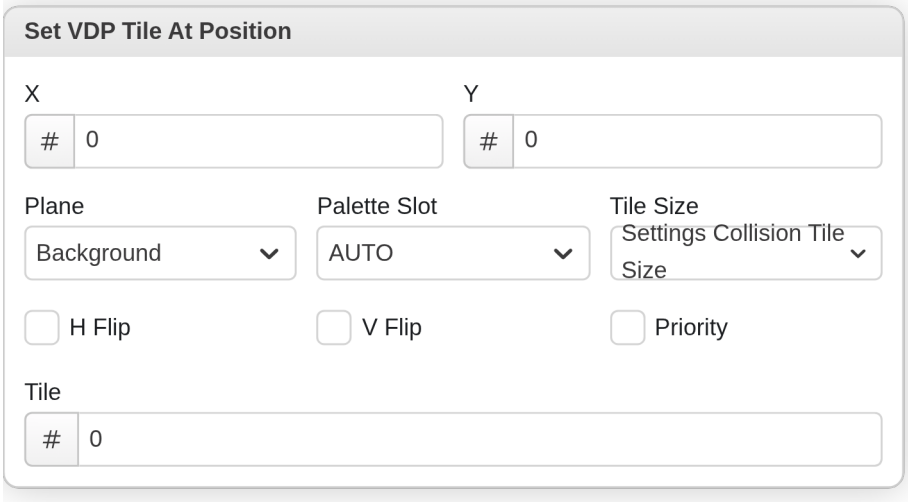


- **Plane**: The plane we want to use
- **Value**: The value you want to set.
- **Direction**: The scroll direction to set.
- **Horizontal Mode**: The Horizontal scroll mode.

Tiles

Set Tile At Position

Set a tile at a specified VDP coordinates with the given tile on the VDP directly. This change is temporary, it will be removed once the screen scrolls.

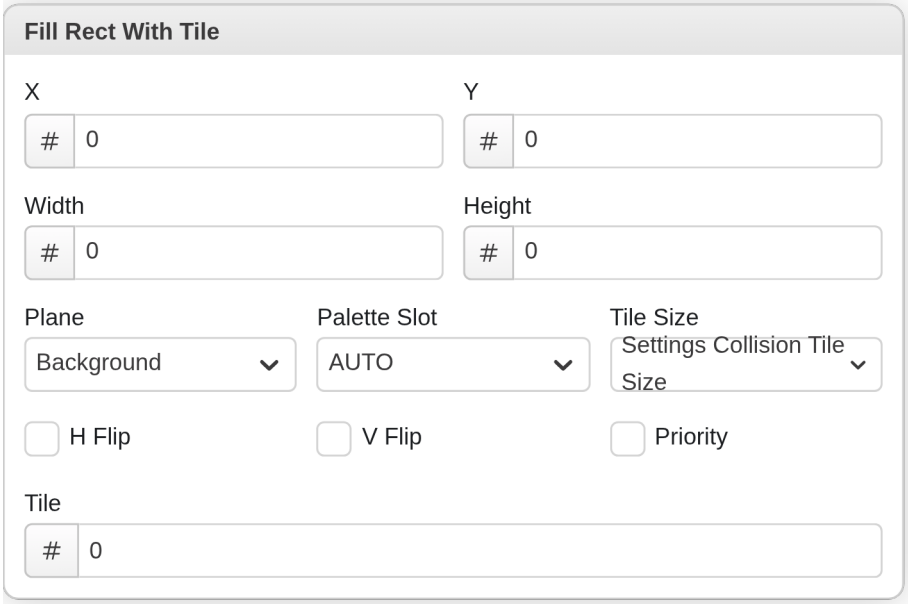


- **X**: The horizontal position.

- **Y**: The vertical position.
- **Plane**: The plane we want to use
- **Palette Slot**: The palette slot you want to use.
- **Tile**: The tile index in VRAM.

Fill Rect With Tile

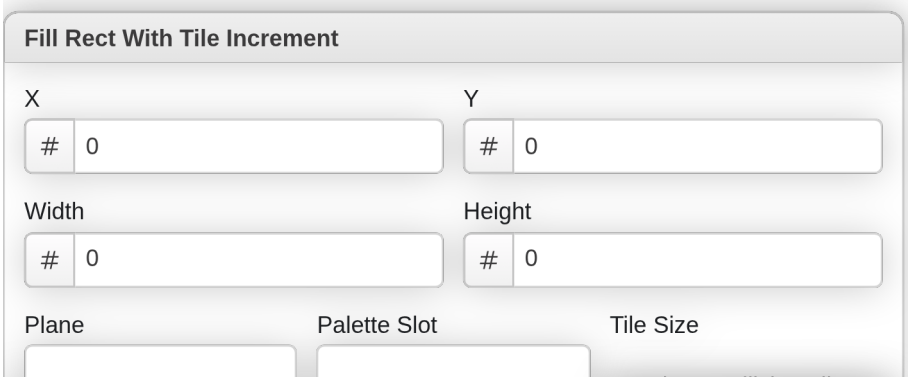
Fill the specified rectangle with the given tile on the VDP directly. This change is temporary, it will be removed once the screen scrolls.

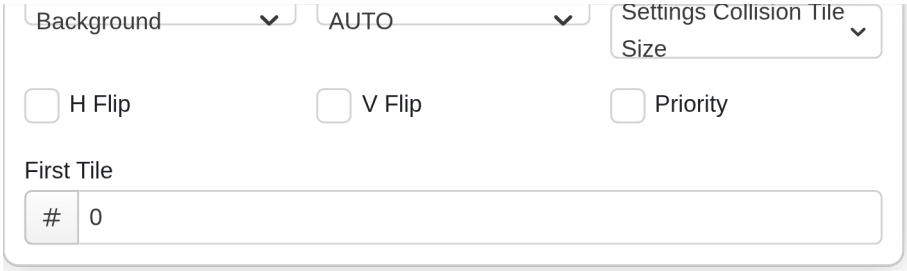


- **X**: The horizontal position.
- **Y**: The vertical position.
- **Width**: The total width.
- **Height**: The total height.
- **Plane**: The plane we want to use
- **Palette Slot**: The palette slot you want to use.
- **Tile**: The tile index in VRAM.

Fill Rect With Tile Increment

Fill the specified rectangle with the given tile on the VDP directly, for each tile of the rectangle base tile will be incremented by 1. This change is temporary, it will be removed once the screen scrolls.

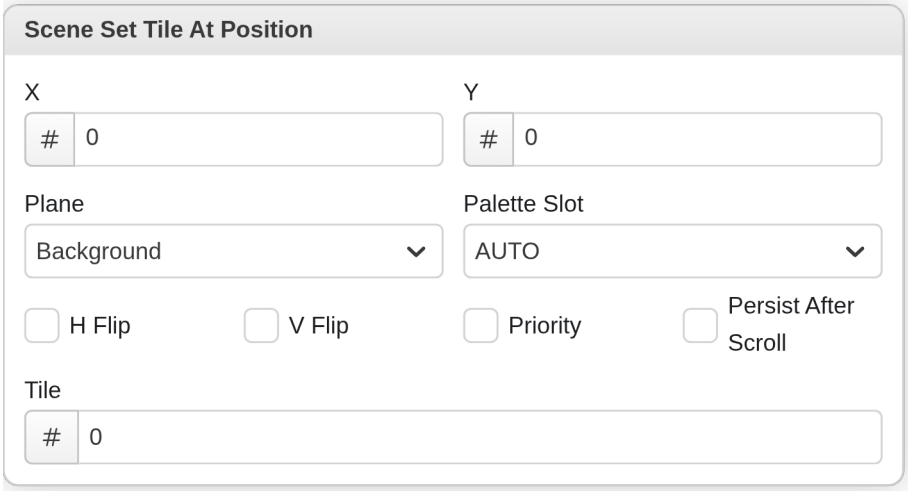




- **X**: The horizontal position.
- **Y**: The vertical position.
- **Width**: The total width.
- **Height**: The total height.
- **Plane**: The plane we want to use
- **Palette Slot**: The palette slot you want to use.
- **Tile Size**: The base tile size to use.
- **First Tile**: The first tile index in VRAM.

Scene Set Tile At Position

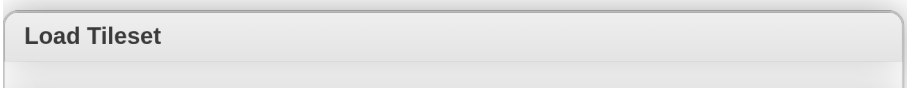
Set a tile at a specified scene coordinates with the given tile. This change can be persisted even after scrolling until the scene is restarted.

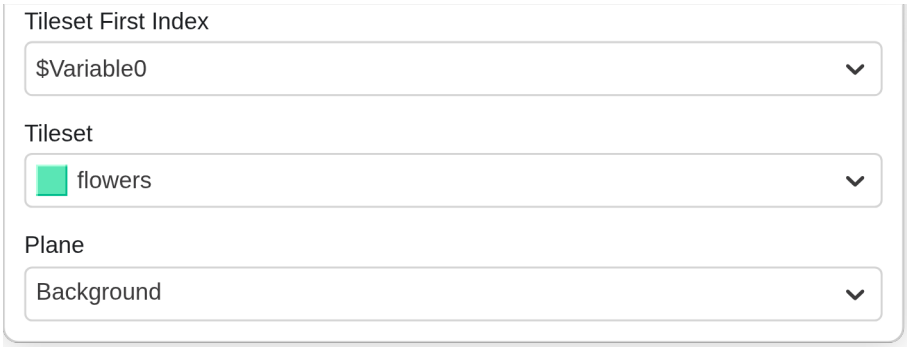


- **X**: The horizontal position.
- **Y**: The vertical position.
- **Plane**: The plane we want to use
- **Palette Slot**: The palette slot you want to use.
- **Tile**: The tile index in VRAM.
- **Persist After Scroll**: Persist tile changes even after scrolling.

Load Tileset

Load an entire tileset into VRAM and set variable to the first tile index to use on Set Tile calls.

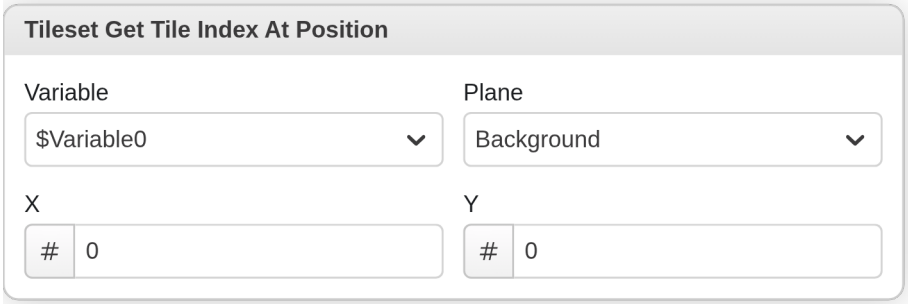




- **Tileset First Index**: The tileset first index will be stored on this variable. You can use this with Set Tile calls.
- **Tileset**: The tileset to fetch tiles from.

Tileset Get Tile Index At Position

Store the VRAM tile index used by a map cell of the selected layer, for example as the destination of a tile animation.



Variables

Store Current Scene Id In Variable

Store the Current Scene Id in a variable.



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

Scene Get Tile Type

Get the given tile type on the given tile coordinates.





- **Variable**: The variable to use for the Tile Type.
- **X**: The horizontal position.
- **Y**: The vertical position.

Tileset Get Layer First Index

Load the selected layer's first tile index into a variable to be able to set tileset.

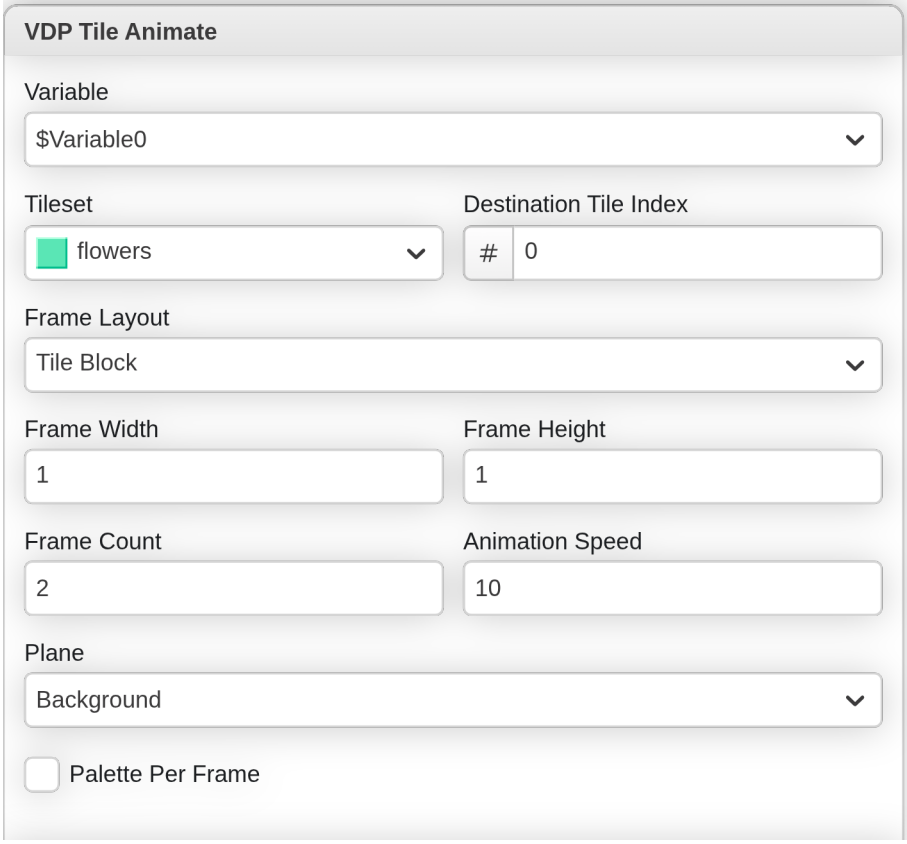


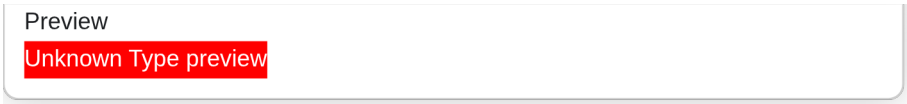
- **Image Handle**: The image handle to operate with the image. You can use this handle to draw or unload the image.
- **Plane**: The plane we want to use

VDP Tile Animation

VDP Tile Animate

Start a tile animation that periodically loads tile frames from a tileset into VRAM. Returns a handle to stop the animation later or get the current frame.

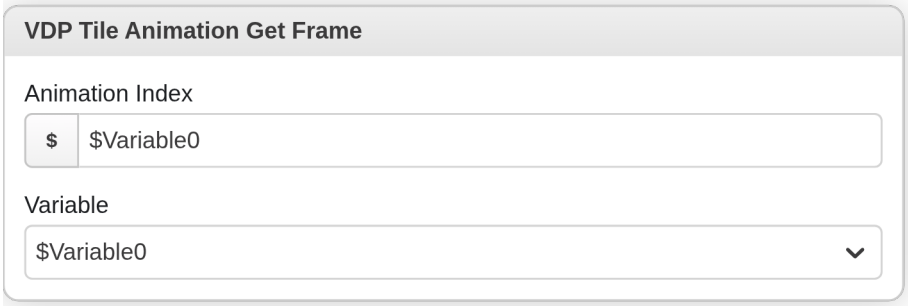




- **Variable**: Variable to store the animation handle (used to stop it later or check current frame).
- **Tileset**: The tileset to fetch tiles from.
- **Destination Tile Index**: VRAM tile index where the animation tiles will be written.
- **Frame Layout**: How frames are read from the tileset: blocks of width x height tiles, or a number of consecutive tiles in row order.
- **Tiles Per Frame**: Number of consecutive tiles, read in row order, that make one frame.
- **Frame Width**: Width of each animation frame in 8x8 tiles. Frames are cut from the tileset image left to right, top to bottom.
- **Frame Height**: Height of each animation frame in 8x8 tiles. The frame is written to VRAM row by row starting at the destination tile index.
- **Frame Count**: Total number of animation frames.
- **Animation Speed**: Number of vblanks between each frame change.

VDP Tile Animation Get Frame

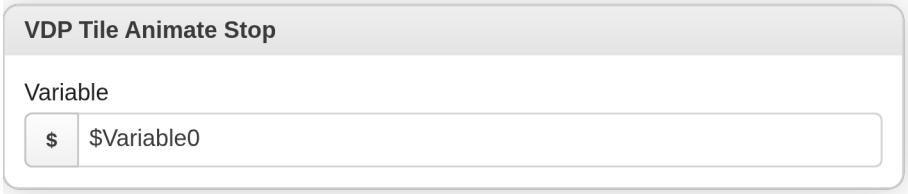
Stores the current frame of the given VDP Tile Animation.



- **Animation Index**: Variable containing the animation handle to check.
- **Variable**: Variable to store the current animation frame.

VDP Tile Animate Stop

Stop a running tile animation using the handle returned by VDP Tile Animate.



- **Variable**: Variable containing the animation handle to stop.

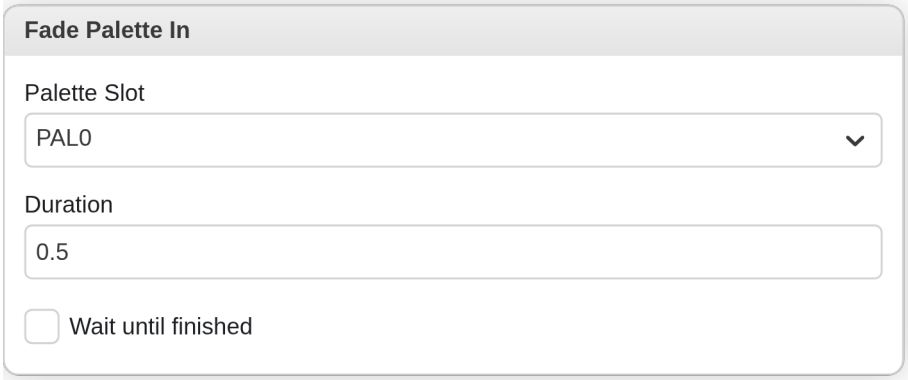
Screen

Events for controlling screen-level settings such as VDP configuration, HBlank/VBlank scripts, and fade effects.

Actions

Fade Palette In

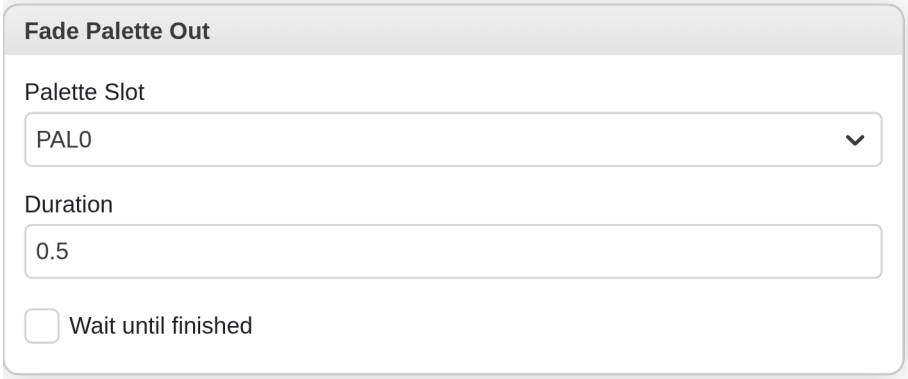
Fade the selected palette from blank.



- **Palette Slot**: The palette slot you want to use.
- **Duration**: The length of time needed to perform the fade in seconds or frames.
- **Wait until finished**: Set if script should pause until fade has finished.

Fade Palette Out

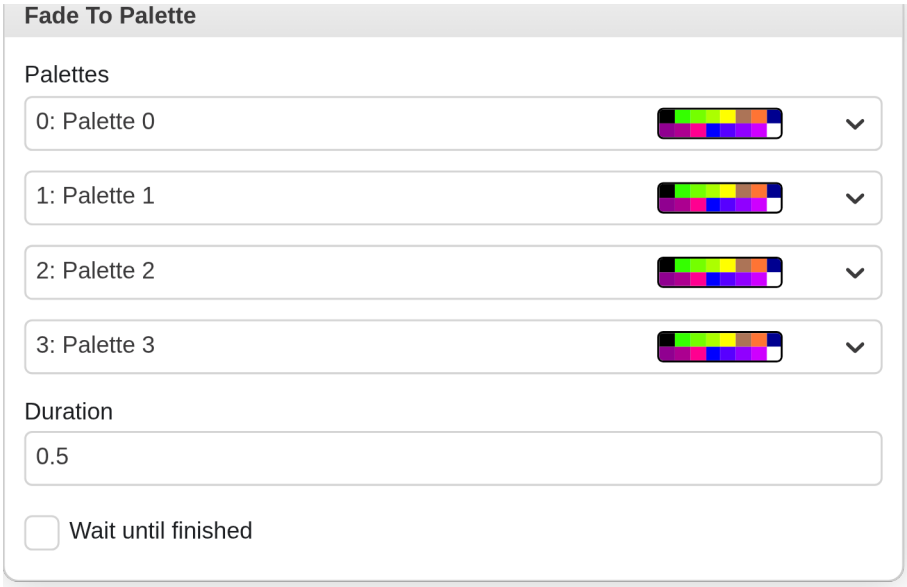
Fade the selected palette to blank.



- **Palette Slot**: The palette slot you want to use.
- **Duration**: The length of time needed to perform the fade in seconds or frames.
- **Wait until finished**: Set if script should pause until fade has finished.

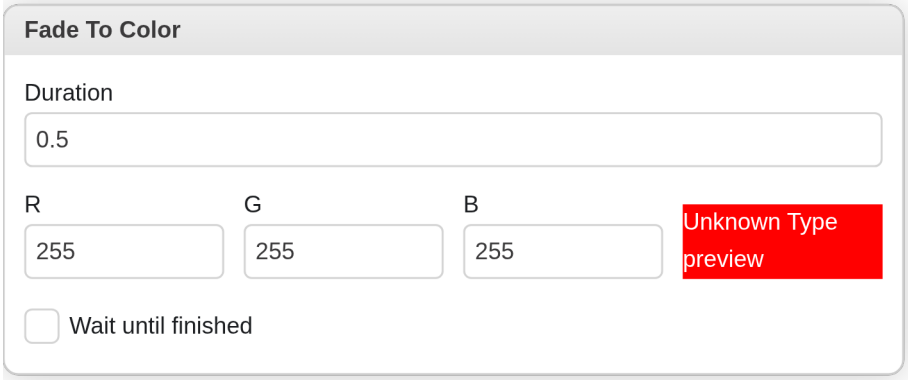
Fade To Palette

Fade the screen colours towards the selected palettes, for example an all-white palette.



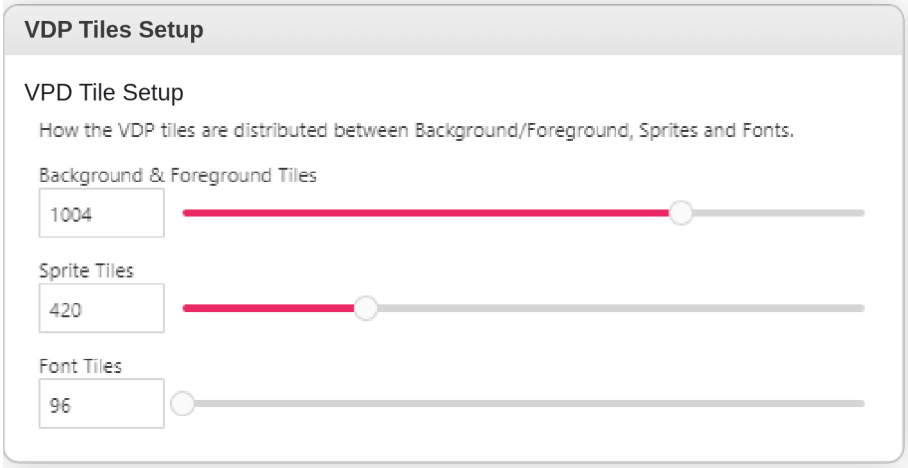
Fade To Color

Fade the whole screen towards a single colour, for example white.



VDP Tiles Setup

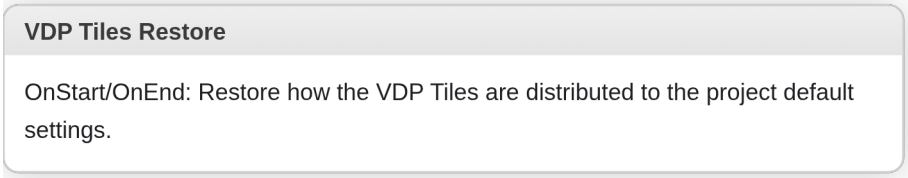
OnStart/OnEnd: Change how the VDP Tiles are distributed.



- **VPD Tile Setup**: How the VDP tiles are distributed between Background/Foreground, Sprites and Fonts.

VDP Tiles Restore

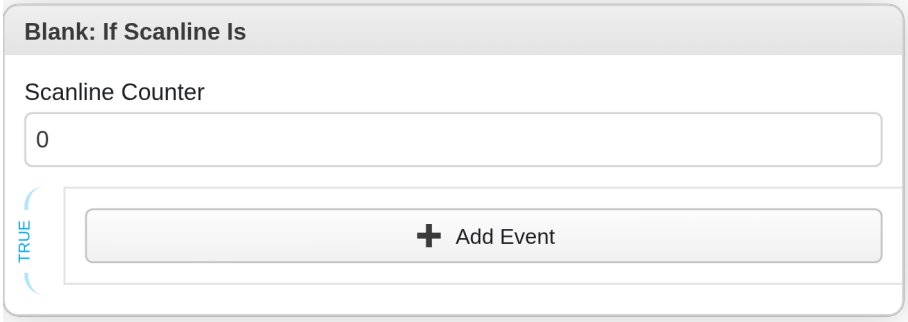
OnStart/OnEnd: Restore how the VDP Tiles are distributed to the project default settings.



Blanks

Blank: If Scanline Is

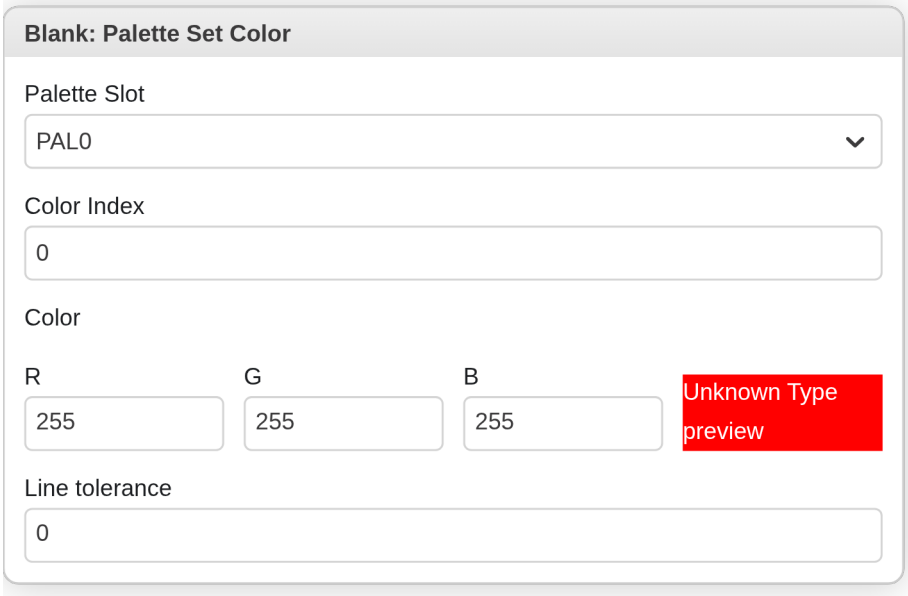
Conditionally run part of the script if the current scanline number match the given value.



- **Scanline Counter**: Number of scanlines between script calls.

Blank: Palette Set Color

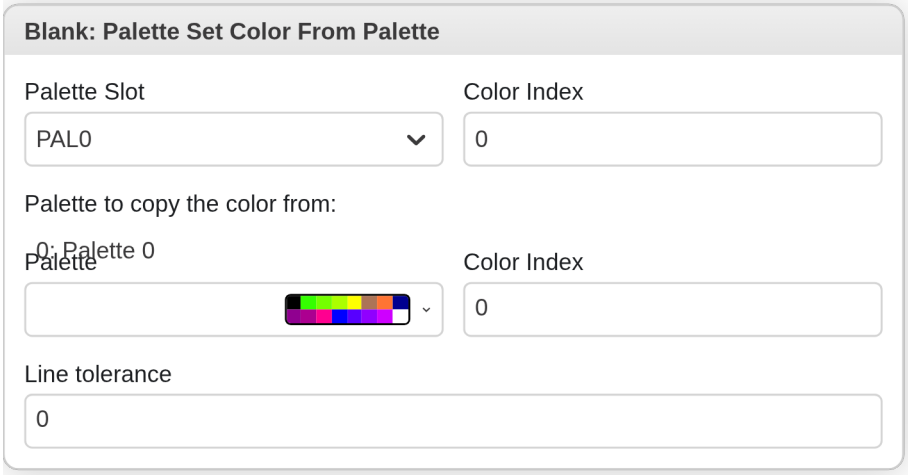
Change the selected color from the selected palette to the values given. This event is only usable for HBlank scripts.



- **Palette Slot**: The palette slot you want to use.
- **Color Index**: The color index to use inside the palette.
- **Color**

Blank: Palette Set Color From Palette

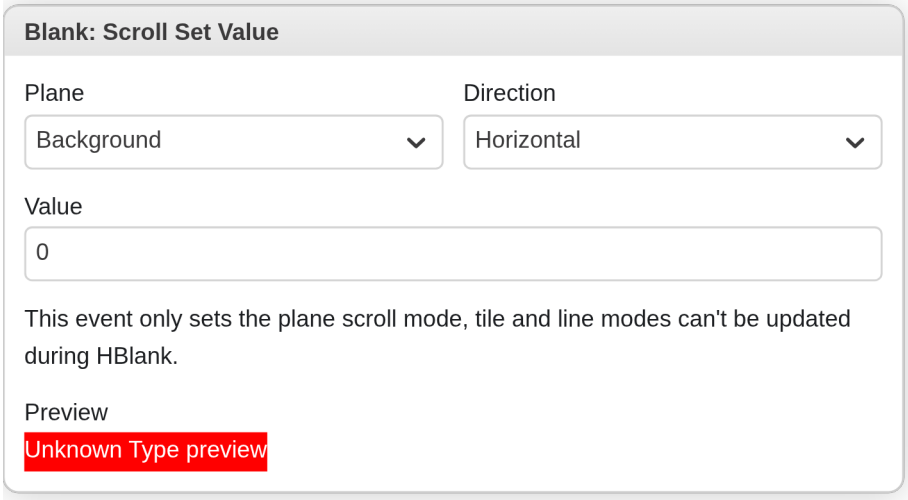
Change the selected color from the selected palette to the selected one.



- **Palette Slot**: The palette slot you want to use.
- **Color Index**: The color index to use inside the palette.
- **Palette**: The new palette to use.
- **Color Index**: The color index to use inside the palette.

Blank: Scroll Set Value

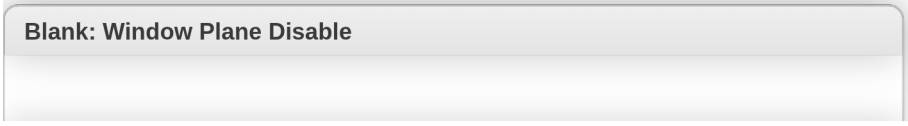
Set the plane scroll to a new value.



- **Plane**: The plane we want to use
- **Direction**: The scroll direction to set.
- **Value**: The value you want to set.

Blank: Window Plane Disable

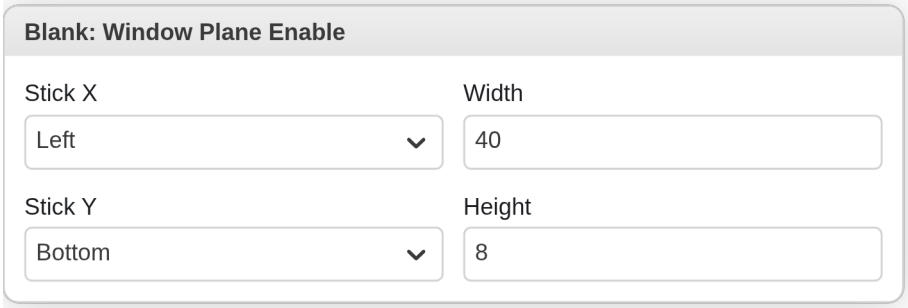
Disable Window Plane.





Blank: Window Plane Enable

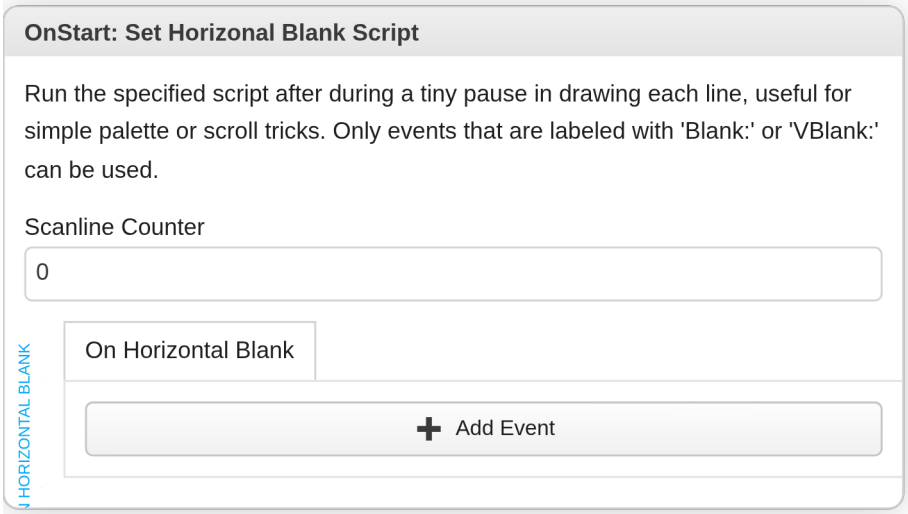
Enable Window Plane with the given parameters.



- **Stick X**: The side of the screen to stick on the X axis.
- **Width**: The width to draw the window.
- **Stick Y**: The side of the screen to stick on the Y axis.
- **Height**: The height to draw the window.

OnStart: Set Horizontal Blank Script

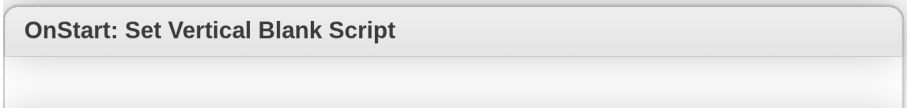
Run the specified script after during a tiny pause in drawing each line, useful for simple palette or scroll tricks. Only events that are labeled with 'Blank:' or 'VBlank:' can be used.

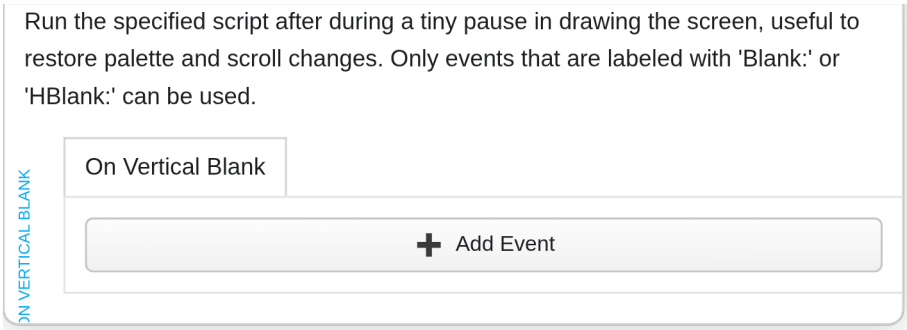


- **Scanline Counter**: Number of scanlines between script calls.
- **On Horizontal Blank**: The script to run when during the Horizontal Blank.

OnStart: Set Vertical Blank Script

Run the specified script after during a tiny pause in drawing the screen, useful to restore palette and scroll changes. Only events that are labeled with 'Blank:' or 'HBlank:' can be used.

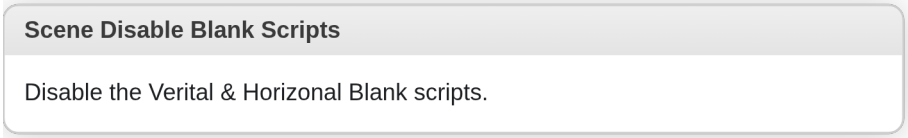




- **On Vertical Blank**: The script to run when during the Vertical Blank.

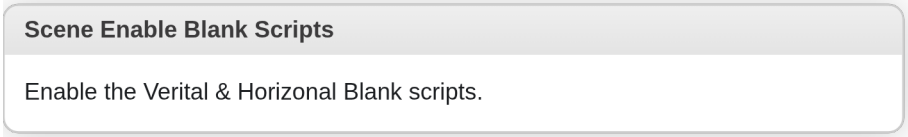
Scene Disable Blank Scripts

Disable the Verital & Horizonal Blank scripts.



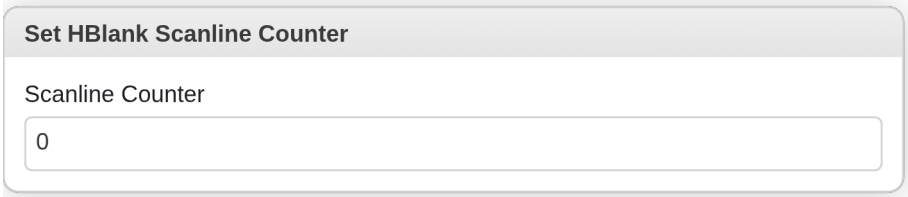
Scene Enable Blank Scripts

Enable the Verital & Horizonal Blank scripts.



Set HBlank Scanline Counter

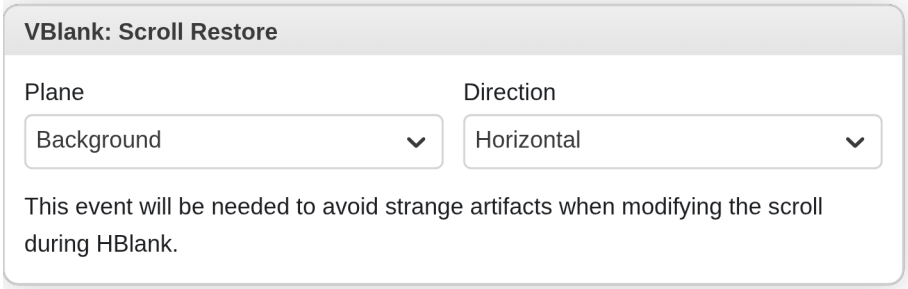
Set the number of scanlines between HBlank script calls.



- **Scanline Counter**: Number of scanlines between script calls.

VBlank: Scroll Restore

Restore the original camera scroll on the given plane.

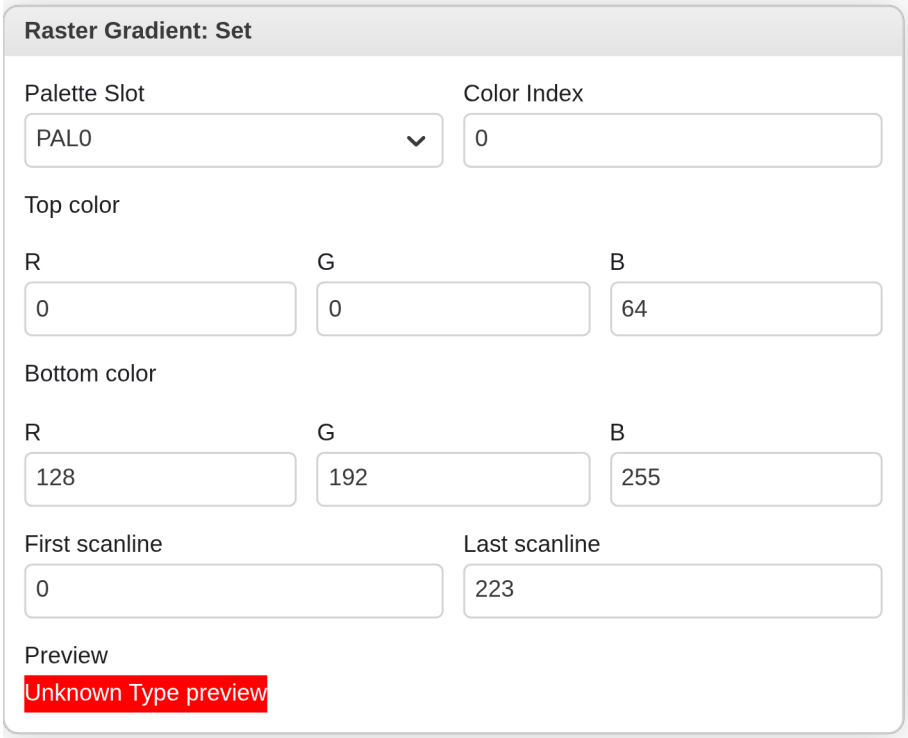


- **Plane**: The plane we want to use
- **Direction**: The scroll direction to set.

Raster Effects

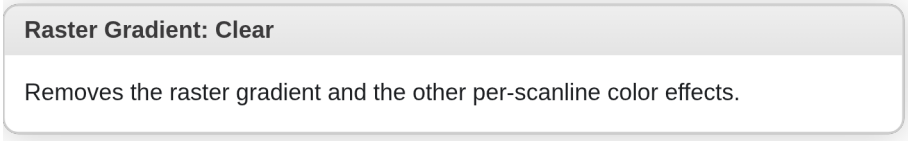
Raster Gradient: Set

Blends one palette color from a top color to a bottom color, scanline by scanline, between two scanlines. Outside those scanlines the color keeps its palette value. Color 0 of the first background palette is the backdrop, which makes a sky gradient. A new gradient replaces the previous one.



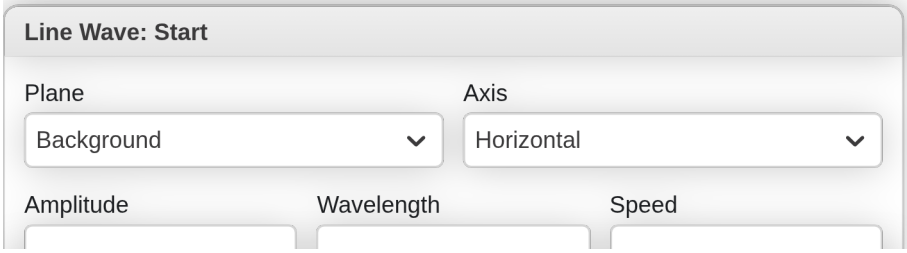
Raster Gradient: Clear

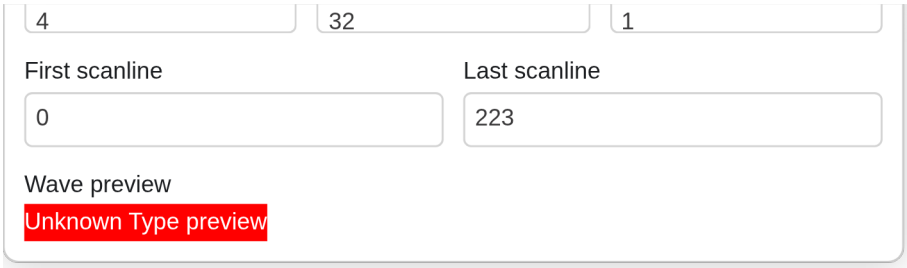
Removes the raster gradient and the other per-scanline color effects.



Line Wave: Start

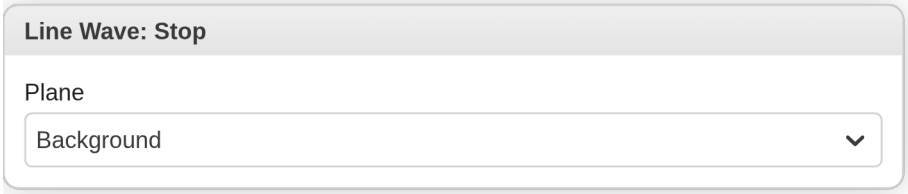
Offsets each scanline of a plane by a sine wave between two scanlines, for water, heat haze or dream effects. The wave moves by the speed in scanlines every frame and adds to the plane's scroll.





Line Wave: Stop

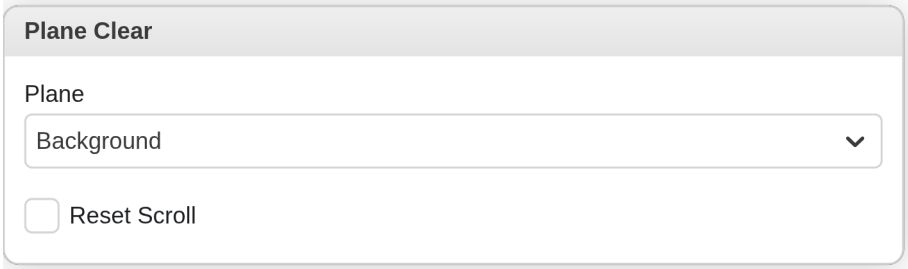
Removes the line wave from a plane and restores its plain scroll.



Planes

Plane Clear

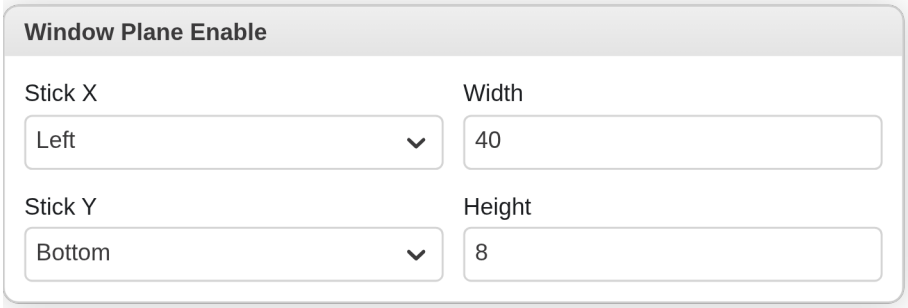
Clear the entire content drawn on the selected plane. Optionally reset its scroll values.



- **Plane**: The plane we want to use
- **Reset Scroll**: Also reset the scroll values and parallax state for this plane.

Window Plane Enable

Enable Window Plane with the given parameters.



- **Stick X**: The side of the screen to stick on the X axis.
- **Width**: The width to draw the window.
- **Stick Y**: The side of the screen to stick on the Y axis.

- **Height**: The height to draw the window.

Window Plane Disable

Disable Window Plane.

Window Plane Disable
Disable the Window Plane

Properties

Hilight / Shadow Enable

Enable Hilight / Shadow effect.

Hilight / Shadow Enable
Enable Hilight / Shadow effect

Hilight / Shadow Disable

Disable Hilight / Shadow effect.

Hilight / Shadow Disable
Disable Hilight / Shadow effect

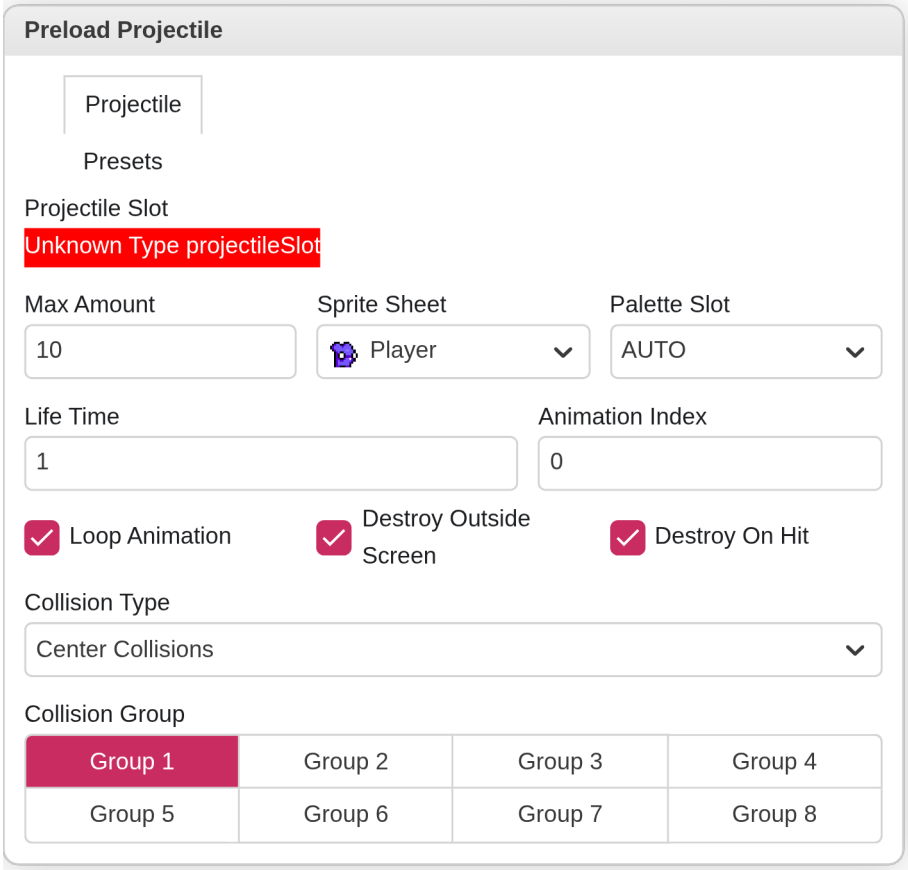
Temp. Triggers & Projectiles

Events for spawning temporary triggers and projectiles.

Actions

Preload Projectile

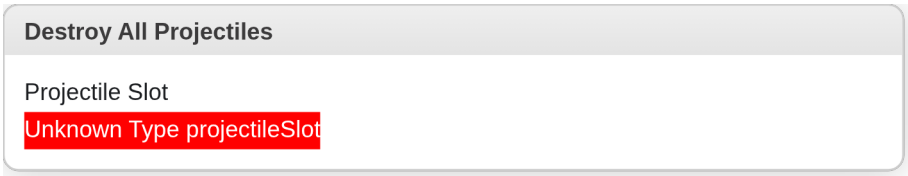
Setup a projectile type to later use via launch projectile.



- **Projectile Slot**: The slot that contains the projectile you want to launch.
- **Max Amount**: The maximum amount of this projectile to preload, if you try to shoot more, nothing will happen.
- **Sprite Sheet**: The sprite to use for rendering the projectile.
- **Palette Slot**: The palette slot you want to use.
- **Life Time**: The amount of time in seconds that the projectile will live for.
- **Animation Index**: The animation index from the sprite sheet.
- **Loop Animation**: Set if animation should loop.
- **Destroy Outside Screen**: Set if the projectile should be destroyed after its outside screen.
- **Destroy On Hit**: Set if the projectile should be destroyed after its first collision.

Destroy All Projectiles

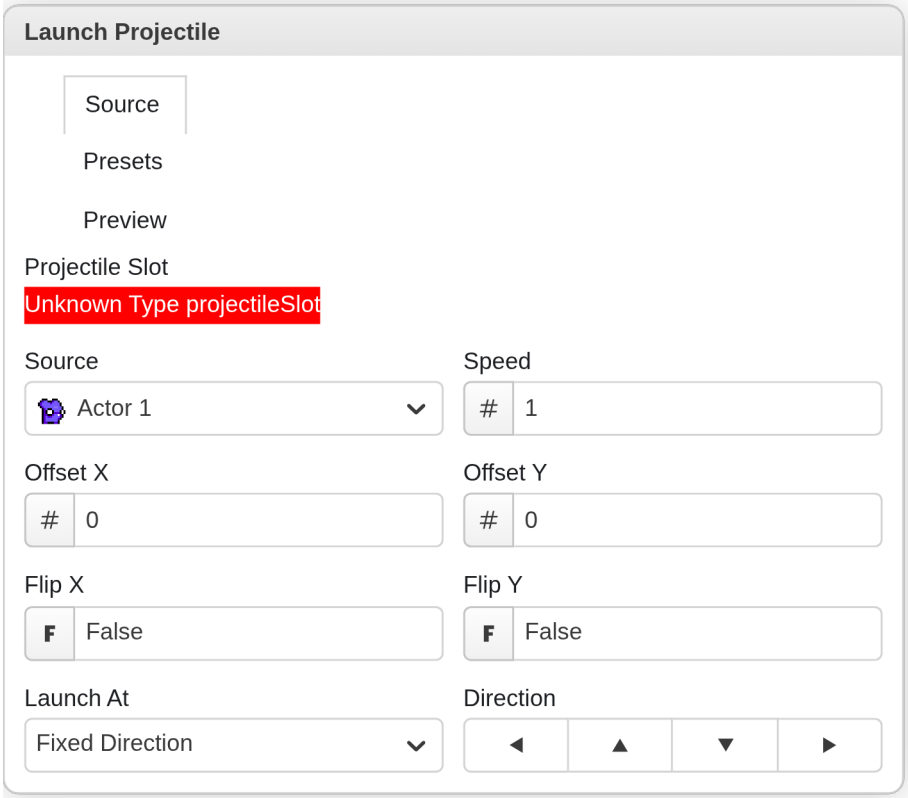
Destroy all projectiles on screen.



- **Projectile Slot**: The slot that contains the projectile you want to launch.

Launch Projectile

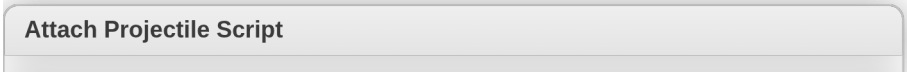
Launch a projectile from an actor in a specified direction. When a projectile collides with other actors it will trigger their OnHit scripts.

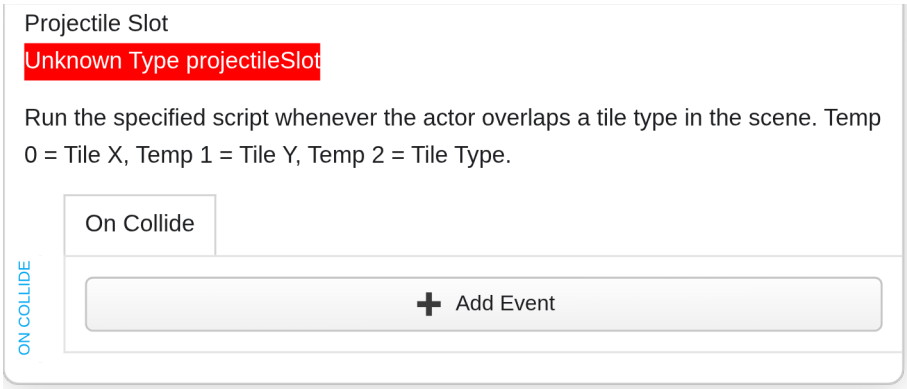


- **Projectile Slot**: The slot that contains the projectile you want to launch.
- **Source**: The actor to launch the projectile from.
- **Speed**: The movement speed.
- **Offset X**: The horizontal offset from the source actors position to start launching the projectile.
- **Offset Y**: The vertical offset from the source actors position to start launching the projectile.
- **Launch At**: Direction mode for the projectile launch.
- **Direction**: The direction to launch the projectile. Can either be a fixed direction or based on an actor's current direction.

Attach Projectile Script

Run the specified script when a projectile collides with the scene. Temp 0 = Tile X, Temp 1 = Tile Y, Temp 2 = Tile Type.



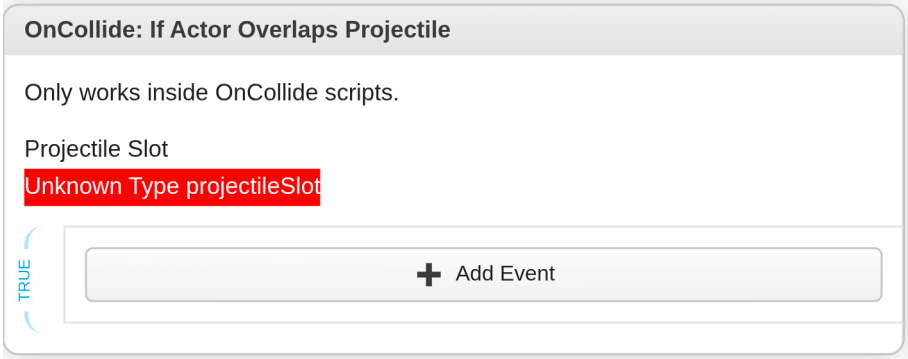


- **Projectile Slot**: The slot that contains the projectile you want to launch.
- **On Collide**: The script to run when the projectile collides.

Conditions

OnCollide: If Actor Overlaps Projectile

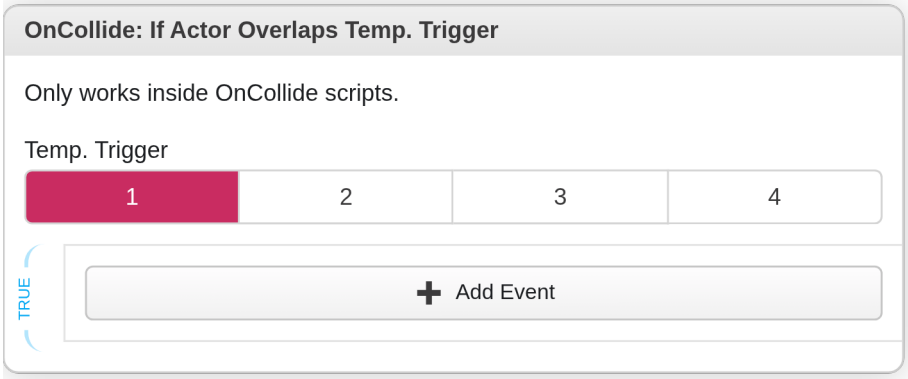
Conditionally run part of the script if we overlapped a projectile of the selected type. Only useful inside OnCollide scripts.



- **Projectile Slot**: The slot that contains the projectile you want to launch.

OnCollide: If Actor Overlaps Temp. Trigger

Conditionally run part of the script if we overlapped a Temp. Trigger of the selected type. Only useful inside OnCollide scripts.

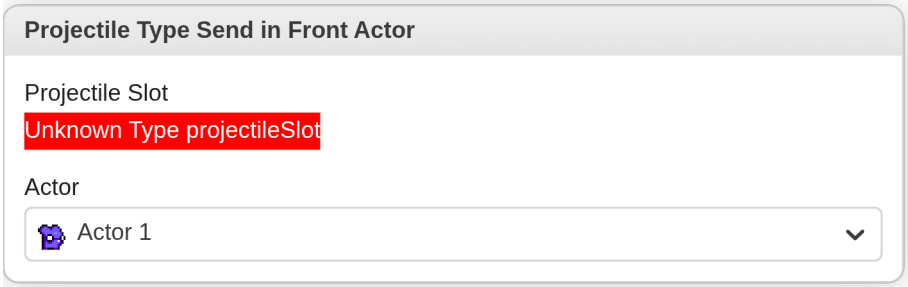


- **Temp. Trigger**: The temporary trigger slow to use, you can spawn up to four at once.

Sprite

Projectile Type Send in Front Actor

Send the projectile sprites in front of the selected actor so it renders on top of it.



- **Projectile Slot**: The slot that contains the projectile you want to launch.
- **Actor**: The actor you want to use.

Projectile Type Send Behind Actor

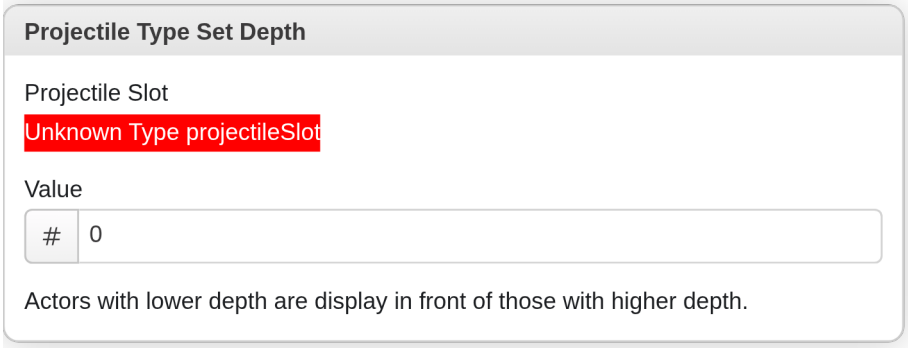
Send the projectile sprites behind the selected actor so it renders below it.



- **Projectile Slot**: The slot that contains the projectile you want to launch.
- **Actor**: The actor you want to use.

Projectile Type Set Depth

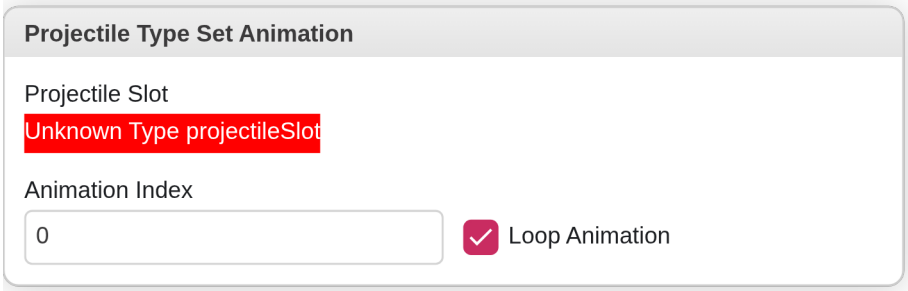
Set the projectile depth on the rendering queue. From -255 to 255. Lower values means rendering higher on the queue.



- **Projectile Slot**: The slot that contains the projectile you want to launch.
- **Value**: The value you want to set.

Projectile Type Set Animation

Set the projectile animation.

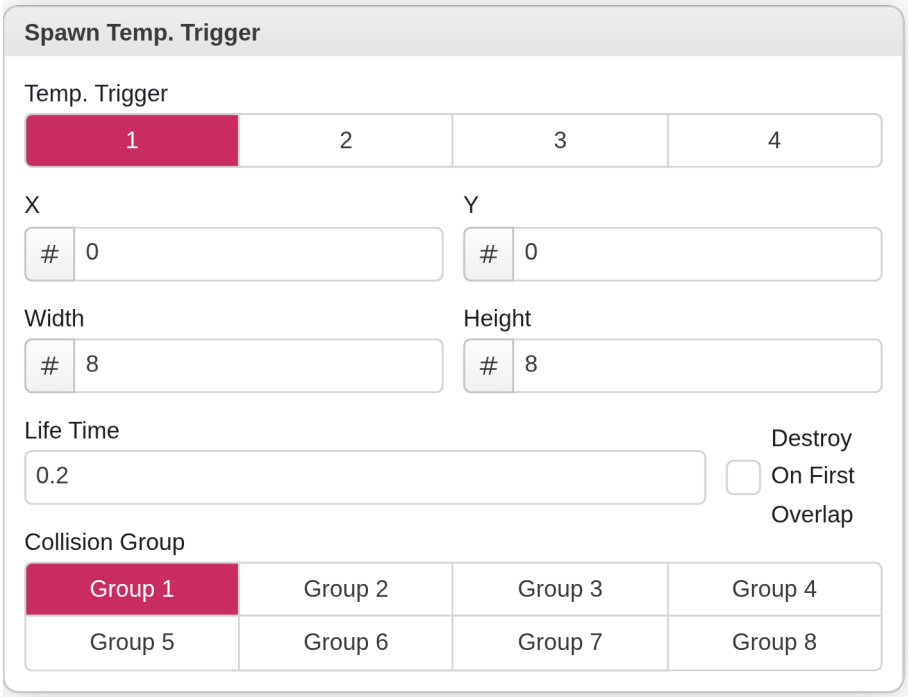


- **Projectile Slot**: The slot that contains the projectile you want to launch.
- **Animation Index**: The animation index from the sprite sheet.
- **Loop Animation**: Set if animation should loop.

Temporary Triggers

Spawn Temp. Trigger

Spawn a Temporary Trigger at the given location with the given size for the given amount of time.



- **Temp. Trigger**: The temporary trigger slow to use, you can spawn up to four at once.
- **X**: The horizontal position.
- **Y**: The vertical position.
- **Width**: The total width.
- **Height**: The total height.
- **Life Time**: The amount of time in seconds that the temporary trigger will live for.

- **Destroy On First Overlap**: If the temporary trigger will be destroyed when the first overlap happens, it will ignore the life time if it's longer.
- **Collision Group**: The collision group that should be used when registering collisions with actors.

Timer

Events for setting up and controlling timers.

Actions

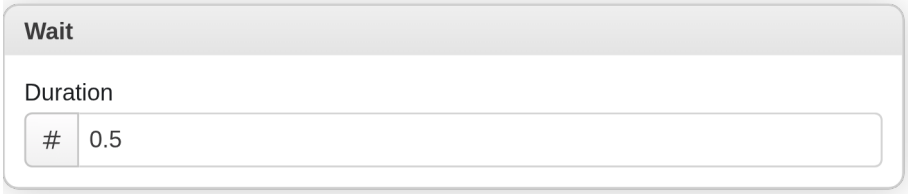
Idle

Pause the script for a single frame.



Wait

Pause the script for a period of time.

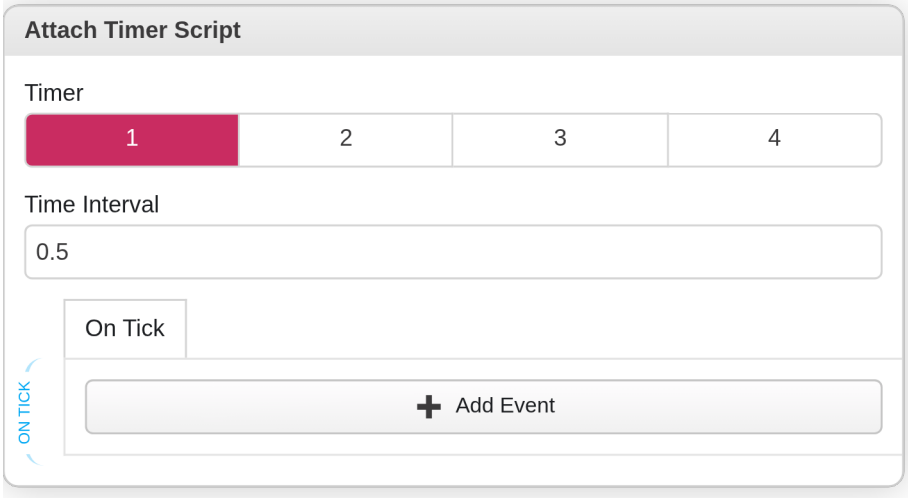


- **Duration**: The length of time to pause the script for in seconds or frames.

Script

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.



- **Timer**: The timer slot to use, each scene can use up to four timers at once (can be adjusted on Settings).

- **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.

Remove Timer Script

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



- **Timer**: The timer slot to use, each scene can use up to four timers at once (can be adjusted on Settings).

Restart Timer

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



- **Timer**: The timer slot to use, each scene can use up to four timers at once (can be adjusted on Settings).

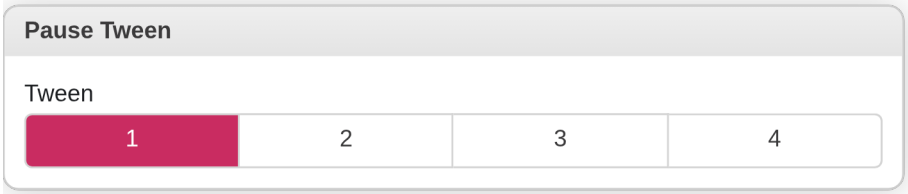
Tween

Events for creating smooth value transitions using tween animations.

Actions

Pause Tween

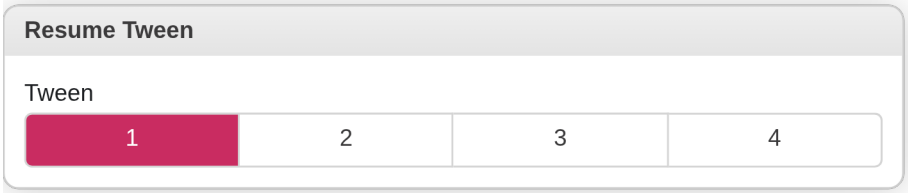
Pause the tween.



- **Tween**: The tween slot to use, each scene can use up to four tweens at once (can be adjusted on Settings).

Resume Tween

Resume the tween.



- **Tween**: The tween slot to use, each scene can use up to four tweens at once (can be adjusted on Settings).

Pause All Tweens

Pause all the tweens.



Resume All Tweens

Resume all the tweens.



Stop Tween

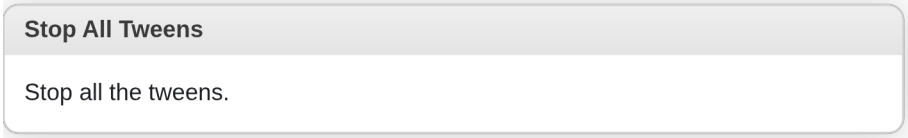
Stop the tween.



- **Tween**: The tween slot to use, each scene can use up to four tweens at once (can be adjusted on Settings).

Stop All Tweens

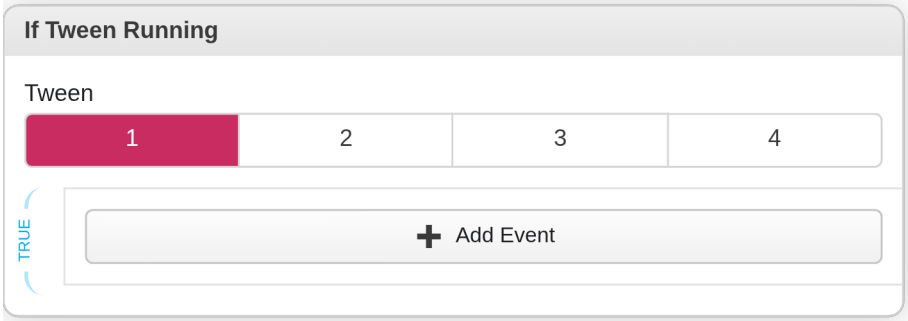
Stop all the tweens.



Conditions

If Tween Running

Conditionally run part of the script if the selected tween is running.

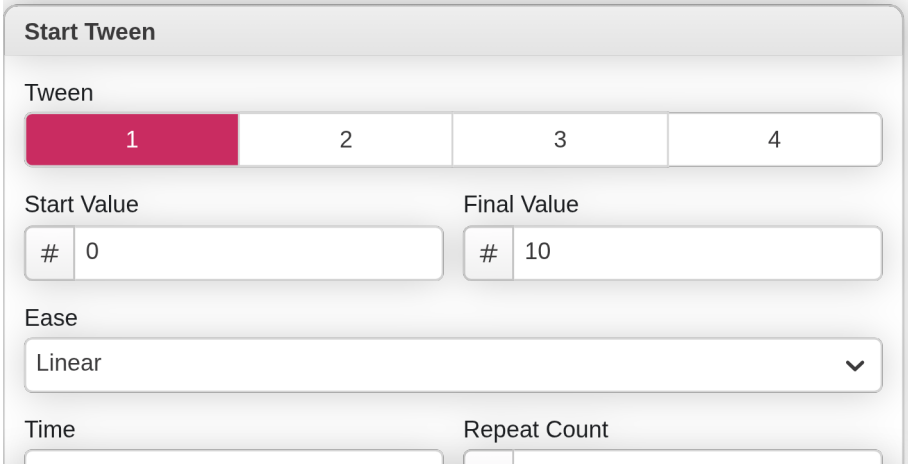


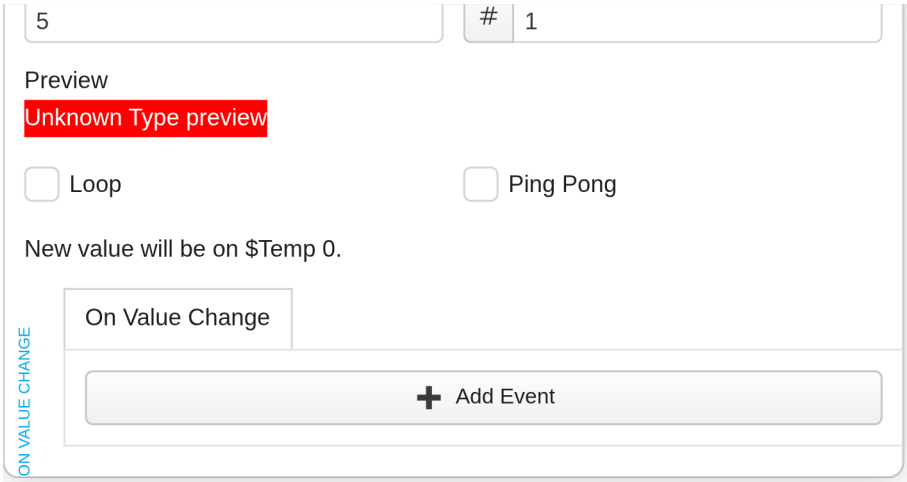
- **Tween**: The tween slot to use, each scene can use up to four tweens at once (can be adjusted on Settings).

Script

Start Tween

Start the tween. Tween a varible to the target value. The script will be called when the variable changes.





- **Tween**: The tween slot to use, each scene can use up to four tweens at once (can be adjusted on Settings).
- **Start Value**: The initial value we should use.
- **Final Value**: The final value we should get.
- **Ease**: The ease function to apply.
- **Time**: The length of time the tween should take.
- **Repeat Count**: The number of times the tween will repeat (max 255).
- **Loop**: If the tween will loop or not.
- **Ping Pong**: If the tween will play backwards before starting again.
- **On Value Change**: The script to run when the variable value changes.

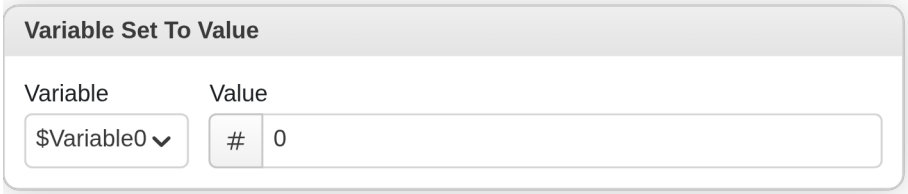
Variables

Events for reading, writing, and manipulating variables and flags.

Actions

Variable Set To Value

Set the specified variable to a defined value.

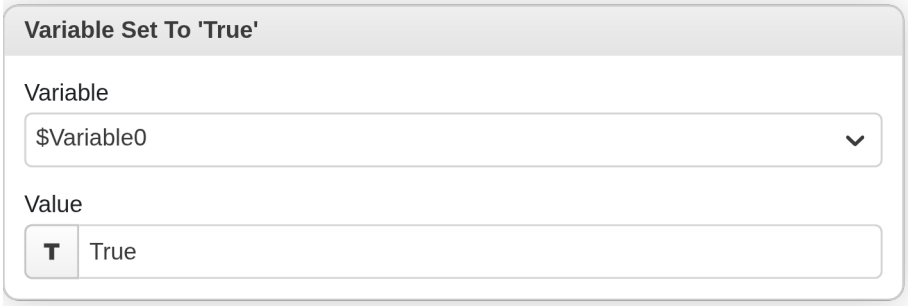


- **Variable**: The variable to use.
- **Value**: The value to set the selected variable to.

Boolean

Variable Set To True

Set the value of the specified variable to true.



Variable Set To False

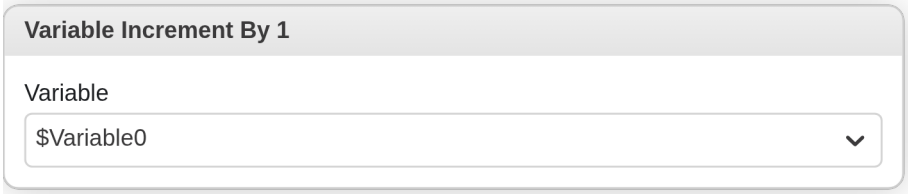
Set the value of the specified variable to false.



Counter

Variable Increment By 1

Increase the value of the specified variable by one.



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

Variable Decrement By 1

Decrease the value of the specified variable by one.

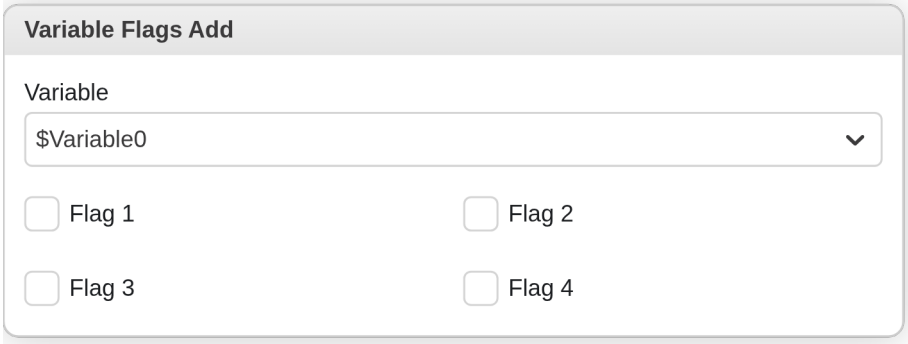


- **Variable**: The variable to use.

Flags

Variable Flags Add

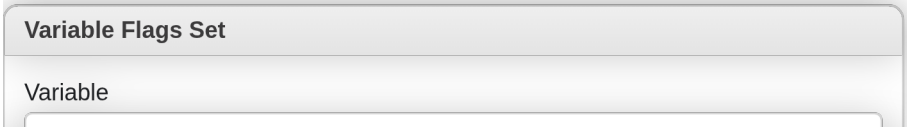
Set selected flags to true on a variable. All unselected flags will keep their previous value.

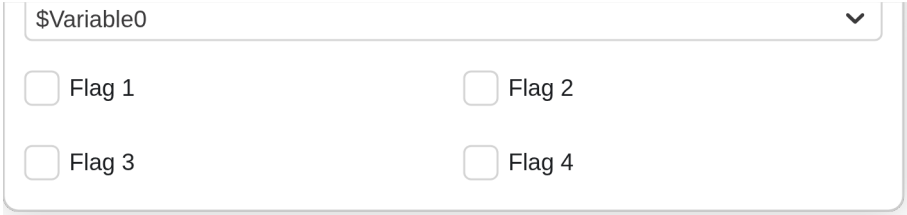


- **Variable**: The variable to use.

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.

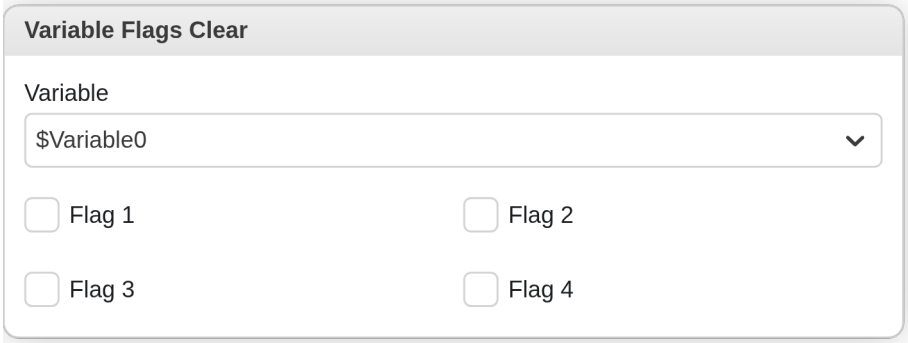




- **Variable**: The variable to use.

Variable Flags Clear

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



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

Reset

Reset All Variables To 'False'

Reset all global variables used by your project back to false.



Writing Plugins

Plugins let you add custom events to MD Engine. A plugin is a folder inside your project's `plugins/` directory containing an `events/` subfolder with one or more JavaScript files.

Folder Structure

```
your-project/
  plugins/
    MyPlugin/
      events/
        eventMyCustomEvent.js
      engine/            (optional: custom C engine code)
        src/
        engine.json
```

Event File Structure

Each event file exports a module with the following shape:

```
const id = "PLUGIN_MY_CUSTOM_EVENT";
const groups = ["EVENT_GROUP_PLUGINS"];
const name = "My Custom Event";

const fields = [
  {
    key: "someValue",
    label: "Some Value",
    type: "number",
    defaultValue: 10,
  },
];

const compile = (input, helpers) => {
  // Your compilation logic here
};

module.exports = {
  id,
  name,
  groups,
  fields,
  compile,
};
```

Required Exports

Export	Type	Description
`id`	`string`	Unique identifier, e.g. `"PLUGIN_MY_CUSTOM_EVENT"`
`name`	`string`	Display name shown in the event menu
`groups`	`string[]`	Event groups, typically `["EVENT_GROUP_PLUGINS"]`
`fields`	`object[]`	UI fields shown in the editor sidebar
`compile`	`function`	Compilation function `(input, helpers) => void`

Optional Exports

Export	Type	Description
`asyncEvent`	`boolean`	Set to `true` if the event uses async helpers (`emitAsyncModal`, `emitAsyncWait`)

The Compile Function

The compile function receives two arguments:

- **input**: the user-configured field values (merged `args` and `children` from the event)
- **helpers**: an object with all available compiler methods and properties

Available Helpers

Code Emission

Helper	Description
`emitStmt(stmt)`	Emits a statement IR node to the output
`cmnt(text)`	Emits a comment line
`startLine()`	Begins a new statement block (increments the program counter)
`endLine(async?)`	Ends the current statement block. Pass `true` to add an async resume label
`emitAsyncModal(expr)`	Emits an async modal check: `if (!expr) { return false; }` with an async label. The `expr` should be the function call. The negation and if-wrapping is handled automatically
`emitAsyncWait(setup, cond)`	Emits a two-step async wait: executes `setup`, then loops with `return false` while `cond` is true
`renderExpr(expr)`	Renders an Expr node to a string (useful for comments)

Registration

Helper	Description
`registerCDefine(symbol)`	Registers a C preprocessor define (e.g. `"NEEDS_DIALOGUE"`)
`registerSubScript(...)`	Registers a sub-script for compilation

Properties

Property	Description
`helpers.pc`	Current program counter. Capture it to create loop-back points
`helpers.depth`	Current indentation depth (rarely needed with IR nodes)
`event`	The current event object being compiled

IR Node System

The MD Engine compiler uses an intermediate representation (IR) to generate target-agnostic code. Instead of writing raw C strings, plugins build IR nodes: plain JavaScript objects that the compiler converts to the target language.

Expression Nodes (Expr)

Expressions represent values. All IR builders are available directly from the `helpers` parameter. Just destructure what you need:

```
const compile = (input, helpers) => {
  const { lit, sym, call, binop, unary, member, index, emitStmt, stmtCall } = helpers;
  // ...
};
```

Builder	Description	Example
`lit(value)`	Literal (number, string, boolean)	`lit(10)`, `lit("hello")`
`sym(name)`	Symbol reference (variable, constant, enum)	`sym("BUTTON_A")`
`call(fn, ...args)`	Function call	`call("TBCE_TextShow", lit(1))`
`binop(op, left, right)`	Binary operator (`+`, `-`, `<`, `==`, `|`, etc.)	`binop("<", sym("i"), lit(10))`
`unary(op, operand)`	Unary operator (`!`, `-`, `~`)	`unary("!", sym("done"))`
`member(obj, field, acc)`	Member access (`.` or `->`, defaults to `.`)	`member(sym("ctx"), "pc", "->")`
`index(array, idx)`	Array index	`index(sym("data"), lit(0))`
`cast(type, operand)`	Type cast	`cast("u16", sym("val"))`
`raw(code)`	Raw code string (avoid when possible)	`raw("customCode()")`

Values that depend on project data use resolver helpers rather than generic IR builders:

Helper	Description	Example
`variableAccessExpr(id)`	Variable reference by ID	`variableAccessExpr(input.variable)`
`propertyAccessExpr(actorId, property)`	Actor property reference	`propertyAccessExpr(input.actor, "pxpos")`
`directionForValueExpr(value)`	Direction constant	`directionForValueExpr(input.direction)`
`compileScriptExpr(value)`	Compile a Script Value field to an expression	`compileScriptExpr(input.value)`

The first argument to `propertyAccessExpr` is an actor resource ID, normally obtained from an `actor` field. It is not the actor's display name. Use `"$self$"` when the event runs in an actor script and should refer to that actor. Position property names include `pxpos`/`pypos` for pixels and `xpos`/`ypos` for tiles.

Statement Nodes (Stmt)

Statements represent actions. Also available from `helpers`:

Builder	Description	Example
`stmtExpr(expr)`	Expression as statement	`stmtExpr(call("fn"))`
`stmtCall(fn, ...args)`	Function call statement (shorthand)	`stmtCall("TBCE_Init", lit(1))`
`stmtAssign(target, value)`	Assignment	`stmtAssign(sym("x"), lit(0))`
`stmtAssignOp(target, op, value)`	Compound assignment (`+=`, `-=`, etc.)	`stmtAssignOp(sym("x"), "+", lit(1))`
`stmtReturn(value)`	Return statement	`stmtReturn(false)`
`stmtIf(cond, body, elseBody?)`	If / else	`stmtIf(sym("done"), [stmtReturn(true)])`
`stmtWhile(cond, body)`	While loop	`stmtWhile(sym("running"), [...])`
`stmtFor(init, cond, step, body)`	For loop	`stmtFor(...)`
`stmtForInfinite(body)`	Infinite loop	`stmtForInfinite([...])`
`stmtSwitch(expr, cases, default?)`	Switch statement	`stmtSwitch(sym("mode"), [...])`
`stmtComment(text)`	Comment	`stmtComment("setup")`
`stmtBlock(...stmts)`	Statement block	`stmtBlock(stmt1, stmt2)`
`stmtRaw(code)`	Raw code (avoid when possible)	`stmtRaw("custom;")`
`stmtBlank()`	Blank line	`stmtBlank()`
`stmtGoto(label)`	Goto	`stmtGoto("loop")`
`stmtLabel(name)`	Label definition	`stmtLabel("loop")`
`stmtBreak()`	Break	`stmtBreak()`

Why IR Nodes?

IR nodes are **target-agnostic**. The same plugin code works regardless of whether the compiler outputs C, assembly, or any future target. The compiler's emitter handles all formatting details (braces, semicolons, indentation, register allocation).

> **WARNING**
>
> Avoid using `raw()` in new plugins. It embeds target-specific strings that will break if the output target changes. Use the structured IR nodes instead.

Example 1: Display All Project Texts

Here is a complete plugin that displays all text strings in the project sequentially:

```
const id = "PLUGIN_DISPLAY_ALL_TEXTS";
const groups = ["EVENT_GROUP_PLUGINS"];
const name = "Display All Texts";

const fields = [
  { label: "Loops through all texts in the project, displaying each one." },
];

const compile = (input, helpers) => {
  const {
    cmnt, startLine, endLine,
    emitAsyncModal, emitStmt, registerCDefine,
    lit, sym, call, member, index, binop, unary,
    stmtAssign, stmtAssignOp, stmtCall, stmtIf, stmtReturn,
  } = helpers;

  const waitHelper = member(sym("ctx"), "waitHelper", "->");

  // Enable the dialogue module
  registerCDefine("NEEDS_DIALOGUE");

  // Async label setup
  startLine();
    cmnt("Pre Async event Label");
  endLine(true);

  // Start the text display system
  cmnt("Start the show text module");
  emitAsyncModal(call("TBCE_TextShowStart", lit(1), sym("WINDOW")));

  // Open a dialogue frame (bottom of screen, 40x8 tiles)
  cmnt("Show Frame");
  emitAsyncModal(
    call("TBCE_TextShowOpenFrame", lit(0), lit(20), lit(40), lit(8), lit(3), lit(1))
  );

  // Initialize the loop counter to 0
  startLine();
    emitStmt(stmtAssign(waitHelper, lit(0)));
  endLine(true);

  // Capture the program counter for the loop-back point
  const loopPC = helpers.pc;

  // Show the current text
  cmnt("Show Text");
  emitAsyncModal(
    call("TBCE_TextShow",
      index(sym("currentLocale"), waitHelper),
      lit(1), lit(21), lit(0), sym("BUTTON_A"))
  );

  // Wait until the player presses A, then consume the input
  cmnt("Wait For Input BUTTON_A from Player 1");
  startLine();
    emitStmt(stmtIf(
      unary("!", call("TBCE_MasterInputJustPressed", lit(0), sym("BUTTON_A"))),
      [stmtReturn(false)],
      [stmtCall("TBCE_InputUpdate")]
    ));
  endLine(true);
```

```
  // Increment counter and loop back if more texts remain
  cmnt("Loop");
  startLine();
    emitStmt(stmtAssignOp(waitHelper, "+", lit(1)));
    emitStmt(stmtIf(
      binop("<", waitHelper, sym("TOTAL_TEXT_LINES")),
      [
        stmtAssign(member(sym("ctx"), "pc", "->"), lit(loopPC)),
        stmtReturn(false),
      ]
    ));
  endLine(true);

  // Close the dialogue frame
  cmnt("Close text frame");
  emitAsyncModal(call("TBCE_TextShowCloseFrame", lit(3)));

  // Finish the text display system
  cmnt("Finish the show text module");
  startLine();
    emitStmt(stmtCall("TBCE_TextShowFinish", lit(1)));
  endLine(true);
};

module.exports = {
  id,
  name,
  groups,
  fields,
  compile,
  asyncEvent: true,
};
```

Example 2: Draw an Actor's X Coordinate

`propertyAccessExpr()` creates a numeric runtime expression. It does not return a JavaScript number or a C string. Likewise, a C `char` stores one character, while the TBCE HAL function `TBCE_TextDraw()` expects a null-terminated `char *`. Convert the number to text before drawing it.

The event below lets the user choose the actor and the screen position. Put it in `events/eventDrawActorX.js`:

```
const id = "PLUGIN_DRAW_ACTOR_X";
const groups = ["EVENT_GROUP_PLUGINS"];
const name = "Draw Actor X";

const fields = [
  {
    key: "actor",
    label: "Actor",
    type: "actor",
    defaultValue: "$self$",
  },
  {
    key: "x",
    label: "Screen X (tiles)",
    type: "number",
    min: 0,
    max: 63,
    defaultValue: 12,
  },
  {
    key: "y",
    label: "Screen Y (tiles)",
    type: "number",
    min: 0,
    max: 31,
    defaultValue: 12,
  },
];

const compile = (input, helpers) => {
  const {
    cmnt,
    emitStmt,
    endLine,
    lit,
    propertyAccessExpr,
    startLine,
    stmtCall,
  } = helpers;

  cmnt("Draw the actor's pixel X coordinate");
  startLine();
  emitStmt(
    stmtCall(
      "DebugPosition_DrawX",
      propertyAccessExpr(input.actor, "pxpos"),
      lit(input.x),
      lit(input.y)
    )
  );
  endLine();
};

module.exports = { id, name, groups, fields, compile };
```

The small engine helper performs the numeric conversion. Put these files under `engine/src/plugins/`.

debugPosition.h

```
#ifndef DEBUG_POSITION_H
#define DEBUG_POSITION_H

#include <types.h>

void DebugPosition_DrawX(s16 actorX, u16 screenX, u16 screenY);

#endif
```

debugPosition.c

```
#include "debugPosition.h"
#include "TBCE/TBCE_Inc.h"

void DebugPosition_DrawX(s16 actorX, u16 screenX, u16 screenY)
{
    // 7 bytes is enough for all the size a pos X can be lol
    char text[7];
    intToStr(actorX, text, 1);

    // We use fill in case values go lower :P
    TBCE_TextDrawFill(text, screenX, screenY, 6);
}
```

Finally, add `engine/engine.json`. Its version must match the version in the editor's `appData/mde/engine-schema/engine.json`:

```
{
  "version": "1.1.0"
}
```

To display tile coordinates instead, change `"pxpos"` to `"xpos"`. Add this event to an update script if the value should follow the actor every frame.

Async Events

If your event needs to wait for something (user input, animation, timer), set `asyncEvent: true` in the module exports and use the async helpers.

emitAsyncModal

Used for operations that block until a condition is met. Pass the function call expression. The compiler wraps it in `if (!call) { return false; }` and inserts an async resume label.

```
// Wait for a text frame to open
emitAsyncModal(call("TBCE_TextShowOpenFrame", lit(0), lit(20), lit(40), lit(8), lit(3), lit(1)));
```

startLine / endLine

Wrap statement blocks that need an async resume point. Calling `endLine(true)` inserts a label that the script runner uses to resume execution on the next frame.

```
startLine();
  emitStmt(stmtIf(
    unary("!", call("TBCE_InputJustPressed", lit(0), sym("BUTTON_A"))),
    [stmtReturn(false)]
  ));
endLine(true);
```

Loop-Back Pattern

Capture `helpers.pc` after an `endLine(true)` call to get a resume point, then assign it to `ctx->pc` and return false to jump back:

```
startLine();
  cmnt("Setup");
endLine(true);

const loopPC = helpers.pc;

// ... do work ...

startLine();
  emitStmt(stmtIf(
    binop("<", counter, sym("MAX_COUNT")),
    [
      stmtAssign(member(sym("ctx"), "pc"), lit(loopPC)),
      stmtReturn(false),
    ]
  ));
endLine(true);
```

Engine Plugins

If your plugin needs custom C engine code, add an `engine/` folder with an `engine.json` manifest and a `src/` directory:

```
MyPlugin/
  events/
    eventMyEvent.js
  engine/
    engine.json
    src/
      myModule.c
      myModule.h
```

The `engine.json` must declare the engine version it was built for:

```
{
  "version": "1.1.0"
}
```

Engine source files in `src/` are copied into the build output and compiled alongside the main engine. If the plugin's engine version doesn't match the project's engine version, a warning is shown during compilation.

Field Types

The `fields` array defines the UI controls shown in the editor sidebar. Each field object supports:

Property	Type	Description
`key`	`string`	The key used to access this value in `input`
`label`	`string`	Display label (also used for info-only text fields with no `key`)
`type`	`string`	Field type: `"number"`, `"text"`, `"checkbox"`, `"select"`, `"variable"`, `"actor"`, `"scene"`, `"direction"`, `"sprite"`
`defaultValue`	`any`	Default value for the field
`min`	`number`	Minimum value (for number fields)
`max`	`number`	Maximum value (for number fields)
`options`	`[value, label][]`	Options for select fields

Example Fields

```
const fields = [
  {
    key: "speed",
    label: "Speed",
    type: "number",
    defaultValue: 3,
    min: 1,
    max: 10,
  },
  {
    key: "direction",
    label: "Direction",
    type: "select",
    defaultValue: "left",
    options: [
      ["left", "Left"],
      ["right", "Right"],
      ["up", "Up"],
      ["down", "Down"],
    ],
  },
  {
    key: "target",
    label: "Target Actor",
    type: "actor",
    defaultValue: "$self$",
  },
  {
    key: "enabled",
    label: "Enabled",
    type: "checkbox",
    defaultValue: true,
  },
];
```

Building Your First Plugin

Here is a step-by-step walkthrough for creating a simple plugin that adds a custom event:

1. **Create the plugin folder structure**: Inside your project's `plugins/` directory, create a folder for your plugin (e.g. `MyFirstPlugin/`), then add an `events/` subfolder inside it.

2. **Define the event**: Create a JavaScript file in `events/` (e.g. `eventSayHello.js`). The filename must start with `event`.

3. **Write the event handler**: Export the required fields (`id`, `name`, `groups`, `fields`, `compile`) as shown in the Event File Structure section above. Start with a simple `compile` function that emits a comment or a single function call.

4. **Test in the editor**: Restart MD Engine (or reload the project). Your custom event should appear in the event menu under the Plugins group. Add it to a script, press Play, and verify it works.

C Engine Plugins

If your plugin includes custom C code in the `engine/` subfolder, it will be compiled alongside the TBCE engine during the build process. This allows direct hardware access, custom assembly routines, and tight integration with the engine's internal systems. See the Engine Plugins section for the required folder structure and `engine.json` manifest.

Debugging Plugins

- **Check the Build Log** for compilation errors from plugin code: both JavaScript event errors and C engine errors appear here.
- **Use the Debugger's Variables tab** to inspect plugin variable values at runtime.
- **Test with the Play Window** before exporting to catch issues early.
- If your plugin uses `raw()` expressions, verify the output in the generated C files to ensure correctness.

Publishing & Sharing

Plugins can be shared as ZIP files containing the plugin folder. Users install them by extracting the ZIP into their project's `plugins/` directory. Make sure to include:

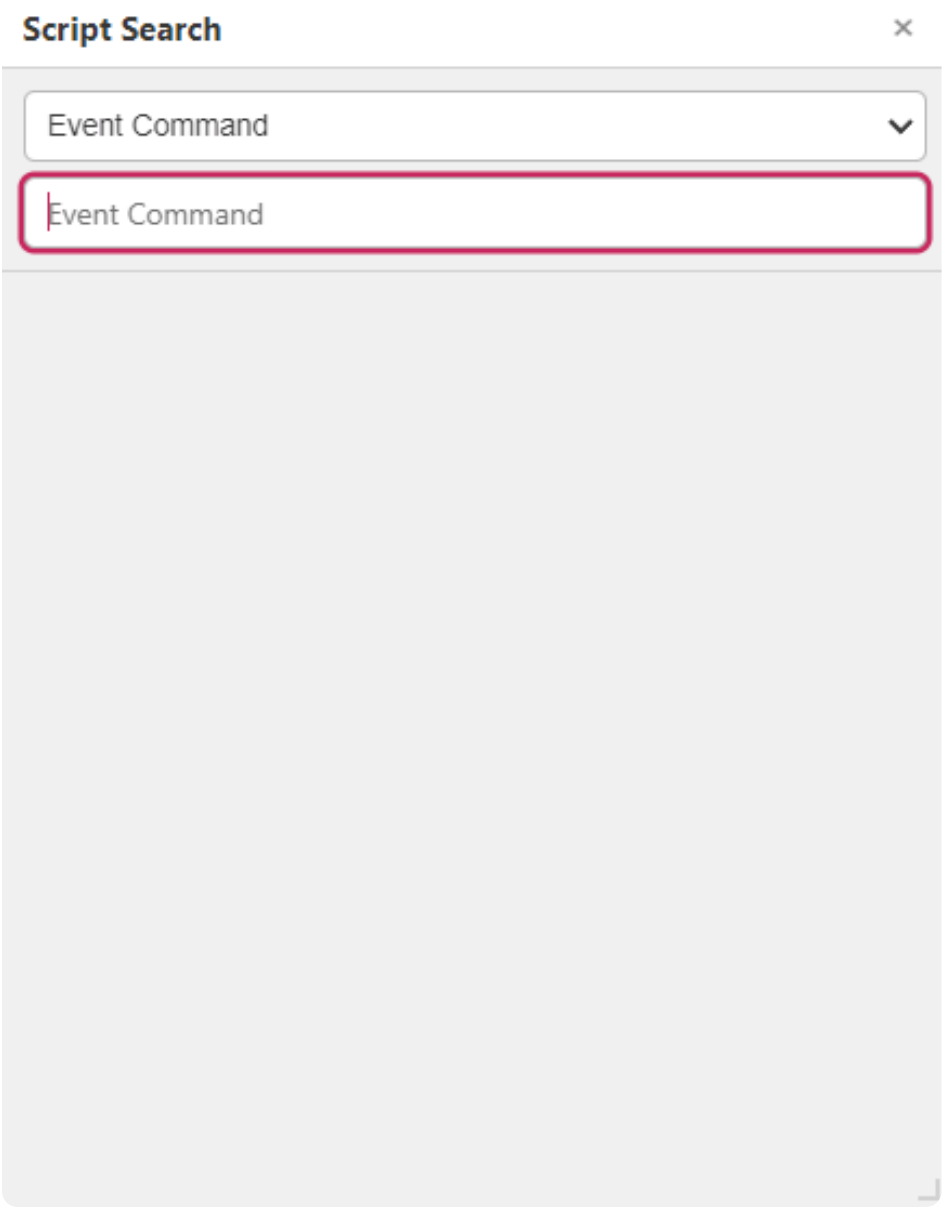
- All event files in `events/`.
- Any engine code in `engine/` (if applicable).
- A brief README or comment header explaining what the plugin does and how to use it.

Script Search

Script Search lets you find specific events across every script in your project — scene scripts, actor scripts, trigger scripts, and custom events.

Opening Script Search

Open it from **Edit > Search** or press `Ctrl/Cmd` + `Shift` + `F`.



Search Types

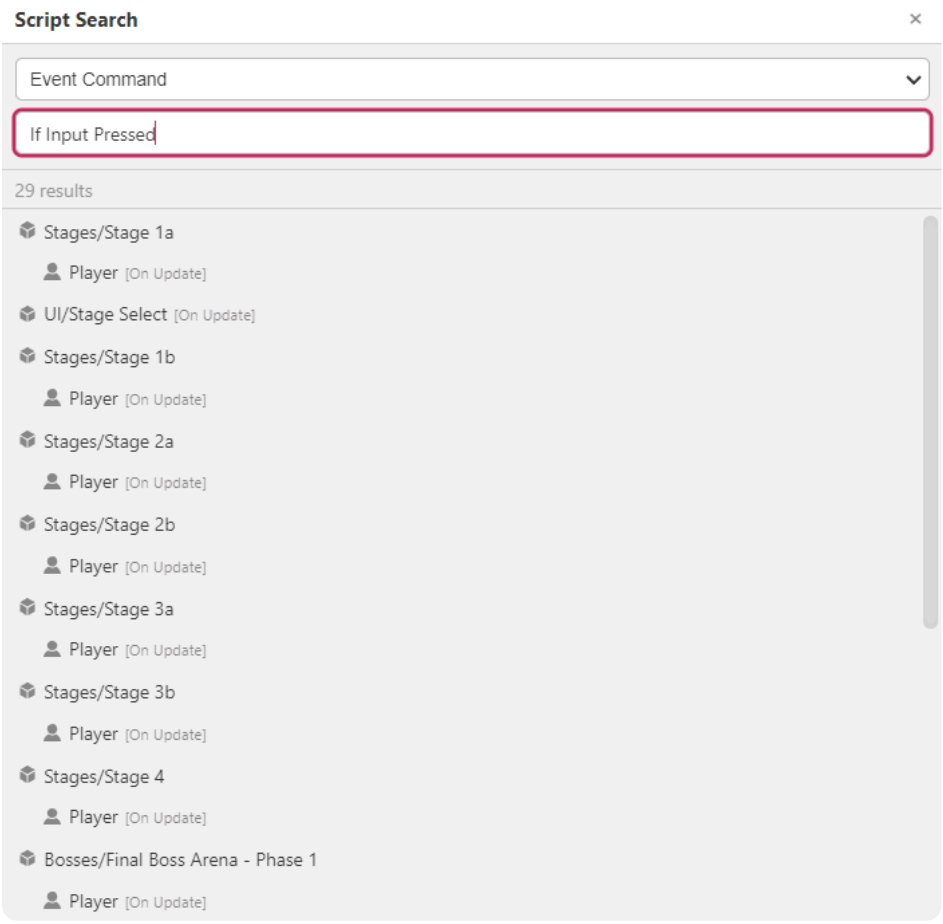
Use the dropdown to choose what to search for:

Type	What it finds
Command	Script events by command name (fuzzy matching)
Variable	Events that reference a specific variable
Actor	Events that reference a specific actor
Custom Event	Events that call a specific custom script
Constant	Events that reference a specific constant
Text	Events containing specific text in dialogue, menus, etc.

Search Results

Results are grouped by scene, actor, trigger, or custom event. Each result shows:

- The entity name and type icon.
- A **script key tag** (e.g., `[On Start]`, `[On Update]`, `[Hit 1]`) indicating which script tab the match is in.



Navigating to Results

Click any result to navigate directly to the matching event:

1. The editor selects the entity (scene, actor, trigger, or custom event).
2. The correct script tab is activated automatically.
3. Any collapsed parent events are expanded.
4. The view scrolls to the matching event and highlights it.

> **TIP**
>
> The search runs on a background thread so it won't freeze the editor, even in large projects.

Panel Controls

The Script Search panel is **draggable** and **resizable** — position it wherever is convenient for your workflow.

Async Events

MD Engine has some script events that don't finish execution on the exact same frame they are called. They are considered asynchronous events.

These are the events that are asynchronous:

Actor

- **Actor Move To**
- **Actor Move Relative**

Camera

- **Camera Flash**
- **Camera Scroll To**
- **Camera Shake**

Input

- **Pause Script Until Input Pressed**

Dialogue, Text & Menus

- **Show Text**
- **Show Choices**
- **Close Dialogue**

Behaviours

- **Actor Simulate Tile-based Movement**

Pathfinding

- **Follow Path**

Timer

- **Idle**
- **Wait**

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*.

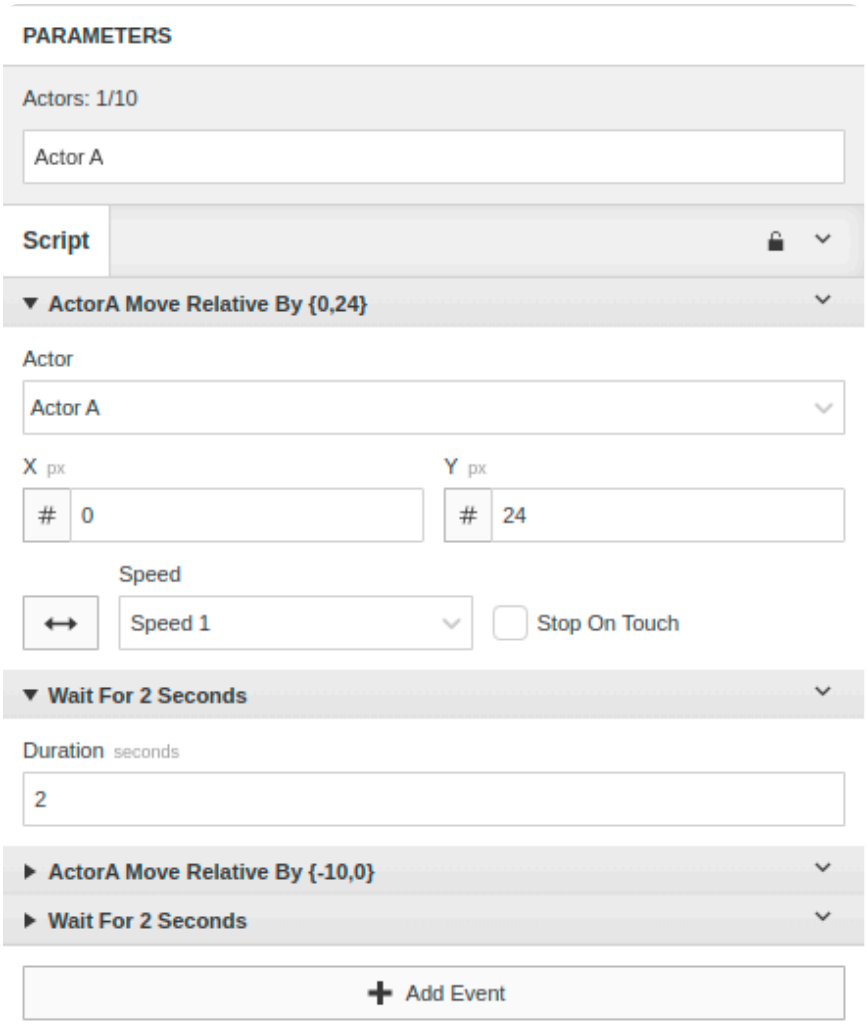
Click the  button to create a new *Custom Script* or select one to edit from the list.

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

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 *Actor A* parameter, but if you don't have any it will be added to the parameter list.

For example the following custom script makes `Actor A` move down, wait and move left.



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.

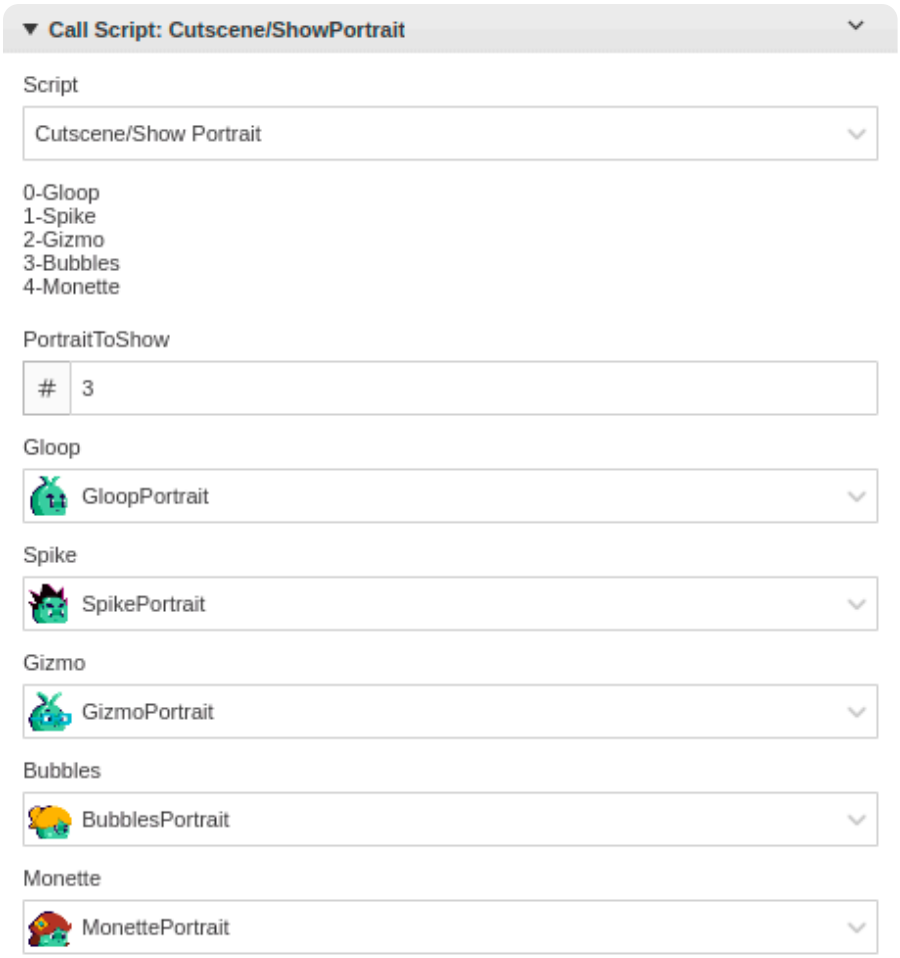


- **Pass By Value** Copies the current value of the variable at call time so that any changes to the variable parameter from within the script will not affect the variable that was passed in. Use this if you want the custom script to **NOT** be able to change the variable that was passed in.

- **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.

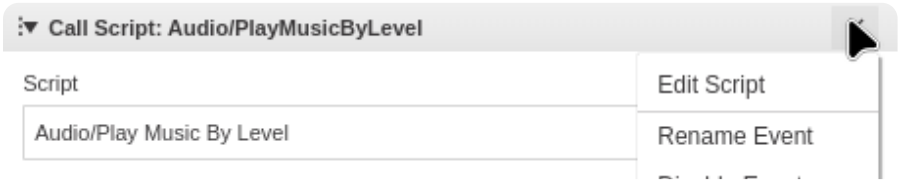
Calling a Custom Script

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.

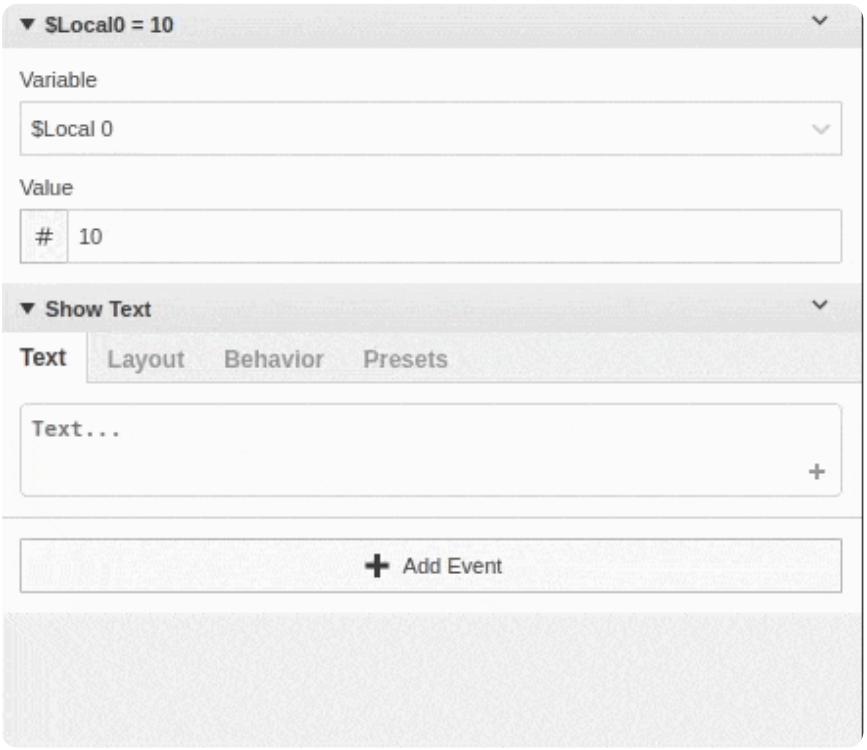
If you ever want to edit the *Custom Script* you can return to it using the list on the *Navigator* or by selecting *Edit Script* from the event dropdown menu.



Dialogue Variables

Script events that allow displaying text, such as Show Text also allow you to display the value of variables.

Typing `$` followed by a variable's name will allow you to search through variables to display in your text. Once a variable has been added you can click it to open a menu allowing you to change the variable.



Formatting Variables

It is possible to add prefixes to dialogue variables allowing them to be used differently within your text.

- **Fixed Length**: Adding `%D` followed by a number between `1` and `9` before a variable causes it to be displayed as a fixed length value with leading zeros. e.g. `%D5$Variable` would display as `00042`.
- **Character Code**: Adding `%C` before a variable causes it to be displayed as an ASCII character code. e.g. `%C$Variable` when the variable is `65` will display as `A`.
- **Text Speed**: Adding `%T` before a variable causes it to be used to set the text speed from that point onwards in the dialogue.
- **Font**: Adding `%F` before a variable changes the font to the index stored in that variable. To find the index of each font, export project data using **Game > Advanced > Export Project Data**, then inspect the `FN_...` definitions in `src/game/<project>_texts.h`. Their indices match the font table in `<project>_texts.c`.

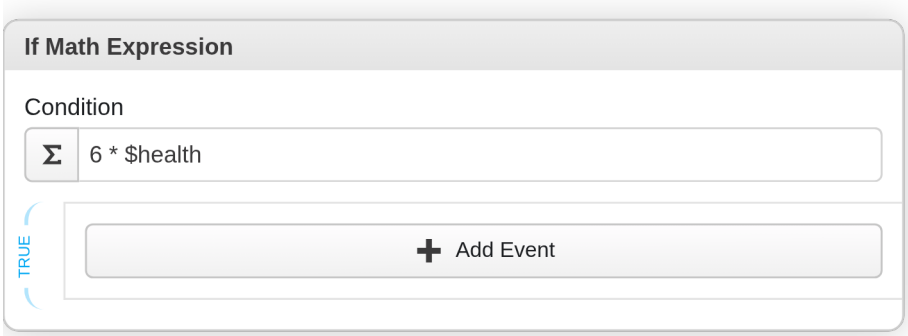
Text Commands

Typing `!` followed by one of the commands below allows you to alter how the dialogue is displayed from that point onwards, allowing you to use multiple fonts and speeds in a single dialogue event.

- `!Font` Allows you to choose a font to switch to mid dialogue, this option only works with advanced text render mode.
- `!Speed` Allows you to choose from the different text speeds.
- `!Instant` Allows you to set the text to appear instantly.
- `!Cursor` Allows you to move the text cursor to a new position. Click the `P(0,0)` or `M(+1,+1)` token to open a popup allowing you to change the position to move to.

Math Expressions

The Evaluate Math Expression and If Math Expression events as well as any events supporting Script Values 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
- `<=` less than or equal to
- `>` greater than
- `<` less than

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`
- `atan2(a, b)` return the 2-argument arctangent of `a` and `b`
- `isqrt(a)` return the square root of `a` rounded down to nearest integer
- `rnd(a)` return a random number greater than `0` and less than `a`
- `arr(a, b)` return the value on index `b` for the array with number `a`, first array has value of `a` = 1

When you want to combine multiple conditions in your code, you can use boolean logic operators:

- `||` **or**: This operator is used when you want to check if at least one of the conditions is true.
  For example, if you write `$Local0 == 0 || $Local1 == 0`, it means "if either variable Local0 equals 0 or variable Local1 equals 0".

- `&&` **and**: This operator is used when you want to check if all conditions are true. For example, if you write `$Local0 == 0 && $Local1 == 0`, it means "if both variable Local0 equals 0 and variable Local1 equals 0".

- `!` **not**: This operator is used when you want to invert the truth value of a condition.
  For example, if you write `!($Local0 == 0)`, it means "if variable Local0 does not equal 0".

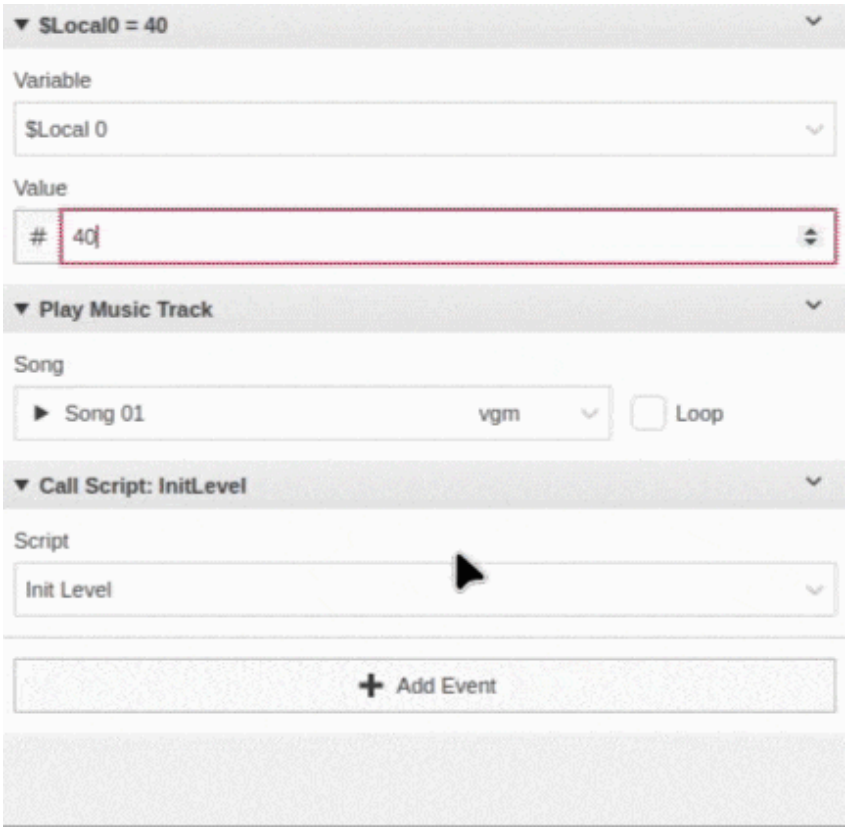
You can use variables in expressions by typing `$` and searching for the variable's name.

You can also use engine fields in expressions by typing `#` and searching for the engine field's name.

> **TIP**
>
> Don't forget the variable access `$` nor the engine field access `#`.

Script Values

Many script events now support using **Script Values** as inputs. These allow you to combine number values, variables, mathematical operations and functions to create complex values.



You can use Script Values in your scripts anywhere that you see a button to the left of an input field (as shown below) by clicking the button to open the menu.

$ $Variable0

Math Expressions

Script Values are very similar to Math Expressions with a more visual interface, the same math functions are available for both value types. Anywhere you can use Script Values you can also use the menu to change to using Math Expressions instead, allowing you to type your expressions. You're also able to combine Script Values and Math Expressions if needed.

Actor Properties

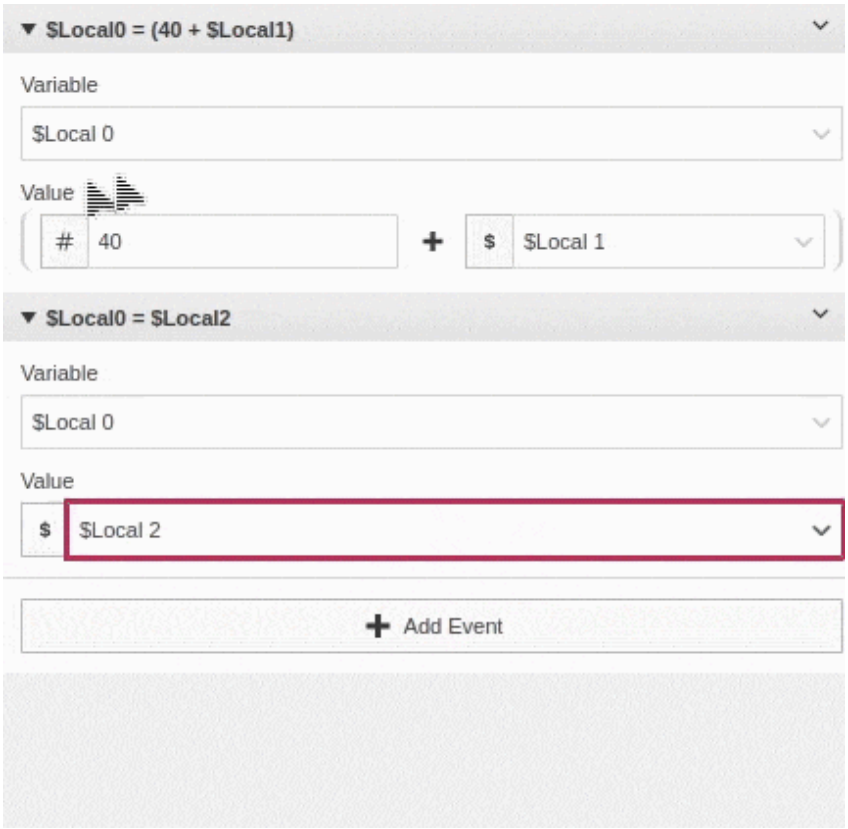
Inside Script Values you can also select different Actor Properties to access key information of the selected actor. Anywhere you can use Script Values you can also use the menu to change to *Property* instead which will allow you to select an actor property.

Engine Fields

Inside Script Values you can also select different Engine Fields to access key information of the engine. Anywhere you can use Script Values you can also use the menu to change to *Engine Field* instead which will allow you to select any engine field.

Drag and Drop

By dragging the Script Value buttons you can quickly move values between different parts of your script.



If you need to move values between different scripts, opening the menu and selecting "Copy Value" will allow you to select "Paste Value" when opening the menu from another script.

Variables

Variables are small pieces of data that your game uses to remember things, like whether a door is unlocked or how much health the player has, etc.

You can think of a variable as a labeled box that holds a number.

The variables are what is called a *s16* or *Short 16*, which can hold values from -32,768 to 32,767.

All variables have a default value of 0 or **false**, any value different than that becomes **true**.

Variable Types

You have three types of variables, *Globals*, *Locals*, and *Temporary* variables.

All types of variables share the same value limits.

Global Variables

You can have 1024 different global variables on your game.
They can be accessed from all *Actor*, *Trigger*, and *Scene* scripts.

Global variable values are persisted across scene changes.

Local Variables

You can have 16 different local variables per *Actor*, *Trigger*, or *Scene*.

Local variables values are reset to 0 when changing scenes. This is done to reduce the amount of used RAM in the game.

Temporary Variables

You can have 16 different temporary variables.

They are useful to do loop indexing, or get return information from different events.

Their value can't be assured and should always be reset before using, even inside a loop if you have any async events their value may have changed.

Flags

Flags are a special type of information that you can store in a variable.
They can only be **TRUE** or **FALSE**.

Flags are stored in each of the bits of the variable, so there are 16 flags you can set per variable.

You can set the flag names to make them easier to use.

WARNING

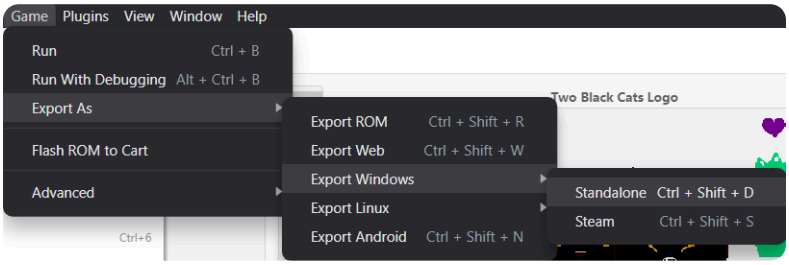
If you manage a variable using their flags, but then use a *If Compare Variable* or *Log* event, the variable will hold a numeric value.
That's the representation of the binary flags stored, so make sure to use only flag checking events to avoid problems unless you know what you are doing.

Exporting the Game

MD Engine can export your game to multiple formats, from a raw ROM file for flash carts and real hardware to standalone desktop executables, web builds, mobile apps, and handheld Linux packages.

How to Export

Go to **Game > Export As** and choose one of the available formats.



ROM Export

The ROM export allows you to burn your game on a flashcart and test it on the original compatible hardware. Once exported, you can flash the ROM to a cartridge using the built-in Flash ROM to Cart tool.

Custom ROM extension

You can change the generated ROM extension on the *Preferences* in the *ROM Extensions* section. The selected extension needs to start with a dot.

Web Export

The web export allows you to play the game from a website.

Custom Web Header

You can change the website Custom Header in the Settings > Export Settings > Web section.

Windows Export

Standalone

The standalone Windows export bundles your game with an embedded emulator into a single executable.

Steam

The Steam export adds Steam integration including achievements support. Configure the Steam App Id in Settings > Export Settings > Steam.

Custom Icon

You can change the icon used on the Windows version by placing a file called *icon.ico* inside your `assets/` folder.

Linux Export

Standalone

A standalone Linux executable with an embedded emulator.

AppImage

> **INFO**
>
> The AppImage export option is only available when running MD Engine on Linux.

Packages the game as a portable AppImage that runs on most Linux distributions without installation.

DEB Package

Creates a Debian package (.deb) installable on Ubuntu, Debian, and derivatives via `dpkg -i` or your package manager.

- Installs desktop entry and icon (256x256)
- Dependencies: libc6, libsdl2-2.0-0
- Configure version and description in Settings > Export Settings > Linux

RPM Package

Creates an RPM package installable on Fedora, openSUSE, and Red Hat-based distributions via `rpm -i` or `dnf`.

- Same features as DEB (desktop entry, icon)
- Configure version and description in Settings > Export Settings > Linux

PortMaster Export

> **EXPERIMENTAL**
>
> PortMaster support is experimental. Packaging format and launcher behavior may change in future versions.

PortMaster is an open-source game launcher for handheld Linux gaming devices such as Anbernic, Miyoo, and others.

The PortMaster export creates a `.zip` package containing:

- A launcher script with automatic input mapping and A/B button swap support
- The game binary and encoded ROM data
- SDL2 fallback for device compatibility
- A `screenshot.png` (640×480) used by the PortMaster launcher
- Package metadata (`port.json`), game info (`gameinfo.xml`), and a `README.md`

Screenshot

The package includes a `screenshot.png` displayed in the PortMaster launcher. To provide a custom screenshot, place a `portmaster_screenshot.png` file in your project's `assets/` folder. If no custom screenshot is found, the project icon (`assets/icon.png`) is resized to 640×480 instead.

Configuration

Configure the package description and README in Settings > Export Settings > PortMaster.

Limitations

- **Always fullscreen** — PortMaster games run in fullscreen mode at the device's native resolution. The aspect ratio is preserved with automatic letterboxing/pillarboxing.
- **CRT shader disabled** — The CRT shader effect is not available on PortMaster builds due to hardware constraints on handheld devices.

Installation

Copy the exported `.zip` file to the `ports/` directory on your handheld device. The game will appear in the PortMaster launcher.

macOS Export

> **WARNING**
>
> macOS exports have not yet gone through Apple certification (App Store review). Suitable for development and testing.

Creates a macOS application bundle (.app). Requires **macOS 10.13 (High Sierra)** or later. The exported app is a universal binary supporting both Intel and Apple Silicon Macs.

macOS 10.13 is the minimum because it is the oldest version that supports Metal graphics, C++17, and the libc++ standard library — all three are required by the engine.

- **Bundle ID** — Required for distribution (e.g. `dev.mdengine.mygame`)
- **Version** — Version string shown in the app info
- **Code Signing** — Optional signing identity for notarization (e.g. `Developer ID Application: Name (TEAMID)`)

Configure in Settings > Export Settings > Apple.

iOS Export

> **WARNING**
>
> iOS exports have not yet gone through Apple certification (App Store review). Suitable for development and testing. Requires macOS with Xcode to build.

- **Bundle ID** — Required (e.g. `dev.mdengine.mygame`)
- **Version** — Version string

Configure in Settings > Export Settings > Apple.

Android Export

> **WARNING**
>
> Android exports have not yet gone through Google Play certification. Suitable for development and sideloading.

- **Package Name** — Java-style identifier (e.g. `dev.mdengine.mygame`)
- **Version Code** — Integer that increases with each release
- **Version Name** — Display version string (e.g. `1.0.0`)
- **Keystore** — Path to a `.jks` keystore file for APK signing (optional for debug builds, required for release)
- **Key Alias** — The alias of the signing key within the keystore

Configure in Settings > Export Settings > Android.

Touch Controls

For **Web**, **Android**, and **iOS** exports, you can configure on-screen touch controls for mobile/tablet play.

The Touch Layout Editor (in Settings > Export Settings > Touch Controls) provides:

- **Layout presets**: Standard, Compact, Left-Handed, Diagonal Up, Diagonal Down, Six-Button, Minimal, Custom
- **Per-button configuration**: Toggle visibility, adjust scale and position
- **Live preview**: A phone mockup showing the current button layout
- **Enable/disable**: Toggle touch controls on or off entirely

Exporter Features

There are several advanced features included in the MD Engine exporter that can be enabled with some events, you can check all on the Platform events section.

This includes the following:

- Achievements
- Platform Detection
- Input Inverted Mode
- Wide Screen Mode (16:9 aspect ratio)
- Toggle Fullscreen Mode
- Toggle 4:3 Window
- Toggle CRT Effect
- Close Game Request

Achievements

This is supported on the *Desktop Export* only when enabling Steam.

You can check the Unlock Achievement event information to know more.

Platform Detection

This is supported on the debugger, *Web Export*, and *Desktop Export*, it helps check the platform the game is running on.

You can check the If Running on Platform event information to know more.

Input Inverted Mode

This should be ON only on Japanese PS4 consoles to invert the UI button displays of Cross and Circle.

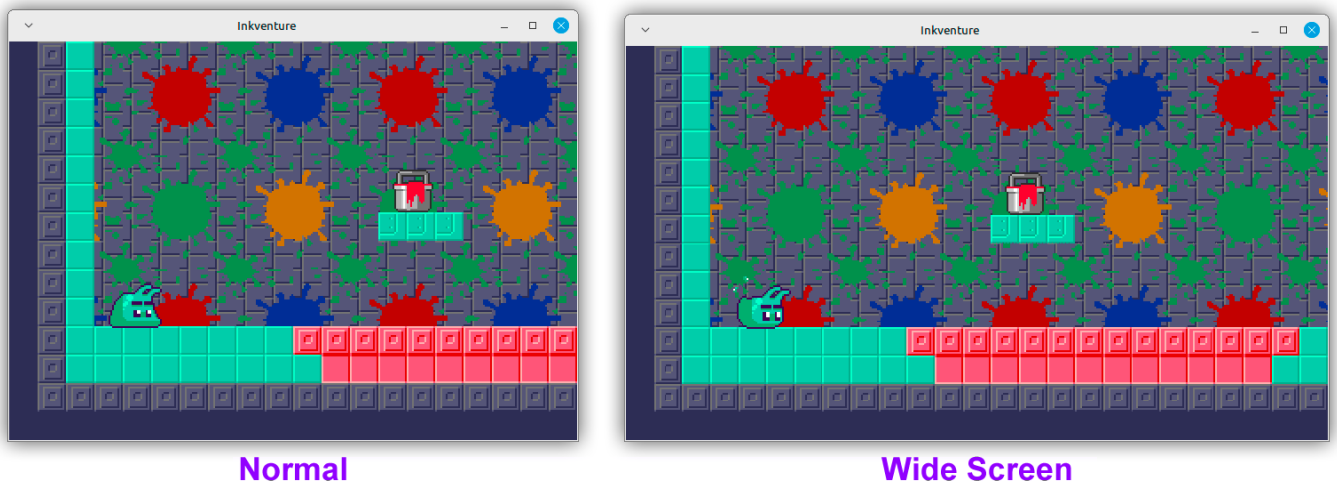
You can check the If Input Inverted Mode is Enabled event information to know more.

Toggle Wide Screen Mode

This is supported on the debugger, *Web Export*, and *Desktop Export*, it makes the game render in 16:9 aspect ratio.

If you are using the H40 mode, instead of having a `320px` x `224px` screen area you will have a `400px` x `224px` screen area. This works by rendering more content on the right side of the screen.

Here you can see an example:



You can check the Toggle Wide Screen Mode event information to know more.

Toggle Fullscreen Mode

This is supported on the *Desktop Export* only, it toggles the window fullscreen mode.

You can check the Toggle FullScreen Mode event information to know more.

Toggle 4:3 Window

This is supported on the *Desktop Export* only, it makes the window occupy a 4:3 area instead of a usual 16:9 area with black borders.

You can check the Toggle 4:3 Window event information to know more.

Toggle CRT Effect

This is supported on the *Desktop Export* only, it toggles a CRT shader effect.

You can check the Toggle CRT Shader event information to know more.

Close Game Request

This is supported on the *Desktop Export* only, it notifies the game close has been requested.

You can check the Close Game event information to know more.

Enhanced Mode

This is supported on the *Desktop Export* only, it toggles the enhanced mode. This mode disables the sprite limit and allows you to render the Plane A (Foreground) under the Window Plane so dialogue windows look better.

You can check the Toggle Enhanced Mode event information to know more.

Toggle Rewind

This is supported on the debugger, *Web Export*, and *Desktop Export*, it toggles the rewind support.

You can check the Toggle Rewind event information to know more.

Notify Game Completed

This is supported on *Desktop Export* only, it notifies the runtime the game has been completed. It allows for special achievement unlocks and such if desired, depending on the runtime implementation.

You can check the Notify Game Completed event information to know more.

Implementation Support

> **ADVANCED CONTENT**
>
> This page is intended for **emulator developers and runtime implementers** who want to integrate MD Engine's export features into their emulator or platform runtime. It references VDP registers, memory addresses, and other low-level details that are not needed for general game development.

All the export features can be added to any emulator runtime. Here is a brief overview of how each of these is implemented in the engine.

Reading / Writing from tbceState

This is the simplest method. It is adjusted so it can work with pretty much every single emulator that can read/write from memory.
It also allows for implementations using RetroAchievement toolchains without direct access to the emulator source code.

Achievements

Unlocking an achievement sets a specific variable in memory that is set with the achievement ID.
The best way to find it is getting the symbol address for *tbceState*, then read 7 bytes after it. This variable is u8-sized, so the only thing needed is to read the position and when it's not zero, unlock the achievement that matches that number. Once it has a non zero value, we recommend setting it back to zero.

Platform Detection

The emulator runtime sets a variable every frame depending on the state.
The best way to find it is getting the symbol address for *tbceState*, then write 6 bytes after.

The possible options you can write are:

- 0: Generic emulator / Real hardware
- 1: Steam
- 2: Android
- 3: iOS
- 4: Nintendo Switch
- 5: PS4
- 6: PS5
- 7: Xbox One
- 8: Xbox Series X/S

It's recommended to implement this so the tutorials can easily be swapped depending on the target platform.

VDP registers

This method is more low level for more advanced features that require work editing the emulator itself and / or the runtime.
They rely on writing different values to the VDP register 26.

Toggle Wide Screen Mode

This is managed by the values 0 and 1.
VDP register 26 will get a value of 0 when it needs to be set to *OFF* and a value of 1 to be set to *ON*.

Toggle Fullscreen Mode

This is managed by the values 2 and 3.
VDP register 26 will get a value of 2 when it needs to be set to *OFF* and a value of 3 to be set to *ON*.

Toggle CRT Effect

This is managed by the values 4 and 5.
VDP register 26 will get a value of 4 when it needs to be set to *OFF* and a value of 5 to be set to *ON*.

Close Game Request

This is managed by the value 6.
VDP register 26 will get a value of 6 when a closing request has been issued.

Toggle 4:3 Window

This is managed by the values 7 and 8.
VDP register 26 will get a value of 7 when it needs to be set to *OFF* and a value of 8 to be set to *ON*.

Toggle Enhanced Mode

This is managed by the values 9 and 10.
VDP register 26 will get a value of 9 when it needs to be set to *OFF* and a value of 10 to be set to *ON*.

Toggle Rewind

This is managed by the values 11 and 12.
VDP register 26 will get a value of 11 when it needs to be set to *OFF* and a value of 12 to be set to *ON*.

Notify Game Completed

This is managed by the value 13. VDP register 26 will get a value of 13 when a *Game Completed* notification has been issued.