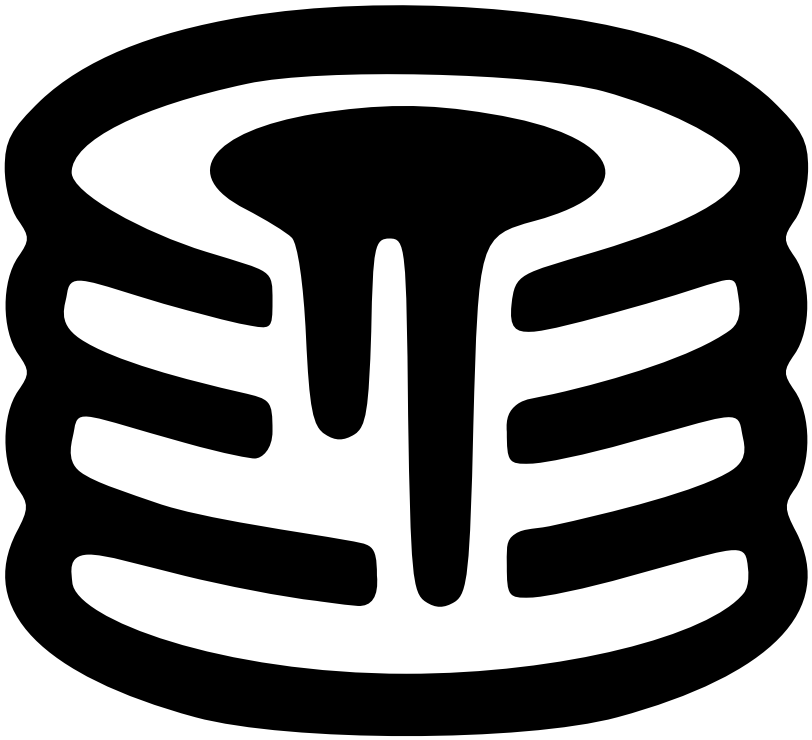
THE COMPLETE STROOP DOCUMENTATION

BY SUPERM789

With the additional help of:
FramePerfection, Pannenkoek2012, Crackhex, Marbler & Danebou



Designed for SM64-TAS-ABC STROOP v1.0.5

Document version 1.2

Chapter 0: What is STROOP? 4
Chapter 1: Installing and Running STROOP 5
1.1: Downloading & main menu 5
1.2: Connecting 5
1.3: Viewing the data 6
Chapter 2: Global Features 7
2.1: Right clicking the version number 7
2.2: Standalone buttons 8
2.3: Lock and gear icons 8
2.4: Remaining version number buttons 9
Chapter 3: Object grid 11
3.1: The object slots 11
3.2: The top options 12
Chapter 4: Variables 14
4.1: The right click menu 14
4.2: Controllers 16
4.3: Generic right click menu 17
4.4: Extended right click menu 18
4.5: Shortcuts 20
Chapter 5: Object tab 21
5.1: Quick action buttons 21
5.2: Object controllers 22
5.3: Object variables 23
5.4: Hidden variables 25
Chapter 6: Mario tab 27
6.1: Left panel 27
6.2: Variables 27
6.3: Hidden variables 28
Chapter 7: HUD tab 30
7.1: Variables 30
Chapter 8: Camera tab 32
8.1: Left panel 32
8.2: Variables 32

Chapter 9: Triangles tab **34**
9.1: Left panel **34**
9.2: Variables **36**
9.3: Hidden variables **38**
Chapter 10: File tab **40**
10.1: Left panel **40**
10.2: Variables **41**
10.3: Hidden variables **42**
Chapter 11: M64 tab **43**
11.1: Piano roll **43**
11.2: Left panel **44**
Chapter 12: Actions tab **47**
Chapter 13: Input tab **48**
Chapter 14: Water tab **49**
Chapter 15: Miscellaneous tab **51**
15.1: Variables **51**
15.2: Hidden variables **53**
Chapter 16: Custom tab **55**
Chapter 17: TAS tab **57**
17.1: Self & point **57**
17.2: Position angle **57**
17.3: Variables **60**
17.4: Schedule **61**
17.5: Hidden variables **61**
Chapter 18: Map tab **62**
18.1: Navigating the map & options **62**
18.2: Navigating the third dimension **62**
18.3: Trackers **63**
18.4: Adding your own trackers **66**
18.5: The plus button **75**
18.6: Controllers & data **81**
18.7: Variables **83**
Chapter 19: Memory tab **85**

19.1: Variables **86**
Chapter 20: Options tab **87**
Chapter 21: PU tab **89**
21.1: Variables **89**
Chapter 22: Area tab **91**
Chapter 23: Model tab **92**
Chapter 24: Gfx tab **93**
24.1: Graph nodes **93**
24.2: Display lists **95**
Chapter 25: Debug tab **96**
25.1: Left panel **96**
25.2: Variables **98**
Chapter 26: Hacks tab **100**
Chapter 27: Cam hack tab **102**
27.1: Be your own lakitu **102**
27.2: Variables & panning **103**
Chapter 28: Q frames tab **105**
Chapter 29: Var hack tab **106**
29.1: Variable box **106**
29.2: Buttons & formatting **107**
Chapter 30: Coin tab **109**
Chapter 31: Disassembly tab **110**
Chapter 32: Testing tab **111**
Chapter 33: Snow tab **114**
Chapter 34: Main save tab **115**
Chapter 35: Painting tab **116**
Chapter 36: Sound tab **118**
Chapter 37: Search tab **119**
Chapter 38: Cells tab **121**
Chapter 39: Music tab **122**
Chapter 40: Script tab **123**
Chapter 41: Warp tab **124**
Conclusion **125**

Chapter 0: What is STROOP?

Hello and welcome to the complete manual, documentation and tutorial for the Super Mario 64 Technical Runtime Observer and Object Processor, referred to as STROOP. This document will go over everything STROOP has to offer, from how to install STROOP to hidden niche features. This document will attempt to explain everything it can, but it cannot guarantee that everything will be 100% correct. Mistakes do happen, and for some extremely niche bits, their use may be unknown. It is recommended to look out for updates to this document to improve its accuracy in the future. You are not expected to retain all the information in this document or even a quarter of it, as STROOP is a program that has had many features repeatedly stitched onto it over the course of nearly a decade, resulting in what we have today. Many of STROOP's features are hidden in menus or just generally hard to grasp, so don't be afraid to ask in the SM64 TASing and ABC discord server if you still have questions **after** reading a section of this document.

To aid with navigating the document and finding answers quickly, the vast majority of buttons, textboxes and variables will be used once or more in "quotation marks" in this document. These can be used to quickly find explanations or documentation with Ctrl+F when you have a specific thing you are looking for. If you can't find something with this method, simply find the right chapter or sub-chapter. They're not THAT long, and it never hurts to learn more stuff while looking for something else.

Now, let's start from the beginning: what even is STROOP? Why is it so useful for TASing? Well, put simply, STROOP is a RAM watch program. It gives read-write access to an emulator's memory. But unlike other RAM watch tools like cheat engine, STROOP is tailor made for exactly one program: Super Mario 64 (SM64). And this is useful, because extensive research has been done on where everything in SM64 is stored in memory and what it's used for, so we can skip the tedious locating process that has to be done in general purpose RAM watch programs.

Chapter 1: Installing and Running STROOP

The first choice to make when downloading STROOP is to choose which build to use. The main one featured on the official GitHub account and developed by Pannenkoek2012 and others is the original STROOP branch. However, after years of unplanned add-ons and new features, it runs rather slow, uses more memory than it needs to, and lacks functionality useful to top-tier TASing. In 2021, FramePerfection decided to refactor the whole program to improve speed, memory and user experience. However, this came at the cost of cutting some of STROOP's less used features. This is the STROOP found in the resources channel of the discord server. Nowadays, both STROOPs are used depending on personal preference, and as such which one you decide to download is up to you. This guide, however, will cover the original STROOP version, as it contains a more complete feature set.

Before downloading, it's very important to note that STROOP is a WinForms application and is thus heavily reliant on Windows libraries. If you're on Linux, it's recommended you download FramePerfection's STROOP and run it with WINE to get it to work, but reliability is not fully guaranteed. If you're on Mac, you're out of luck. Your best bet is to run STROOP in a virtual machine or dual boot Windows if your Mac is old enough.

1.1: Downloading & main menu

https://github.com/SM64-TAS-ABC/STROOP/releases/download/vDev/STROOP.zip is the download link to the latest version. The version in the releases page of the GitHub repo is older, so specifically use the above link. Once you've extracted the contents of the ZIP file, run STROOP.exe. Both your web browser and Windows may flag STROOP as a malicious program, and that's because it's a cookie stealer. Nah I'm kidding, it's a false flag. Select the options to keep the program and run it anyways. STROOP will fail to load if you run the program from the windows search menu, so open the exe directly from your file explorer then pin it to your taskbar for later. Once it loads, you will be presented with a big white screen with stuff in the middle.

This is STROOP's main menu. The large box above the buttons is a list of all currently running Nintendo 64 emulators STROOP was able to detect. If you've opened the emulator after STROOP, you need to hit the refresh button for STROOP to update the list. Before it can do anything, it needs to find an emulator and connect to it. STROOP is developed around the assumption that you're using modern Mupen64 with it. Not Mupen64Plus, that's a different emulator. While you can get other emulators to connect, it can take some manual work and some features might not work properly. This tutorial will continue under the assumption that you're using Mupen64, because it's the emulator of choice for TASing SM64. It will go over the process and possible issues for connecting it with STROOP. So if you haven't already, download the modern Mupen64 along with the basic tools needed for TASing at this link: https://mupen64.com. Don't try to find Mupen on Google, because you might end up with an old version that has a remote code execution glitch in it. And this time I'm not kidding.

1.2: Connecting

Now if you open Mupen64, run Super Mario 64 on it and hit refresh, you should find Mupen64 listed in the box. If it's not there, that's a common issue and here's how to fix it. STROOP is looking for a program named "Mupen64", but recently Mupen64 has started changing the name of the exe file to match the version. So what you want to do is go over to where your Mupen64's executable is and

rename the file to "mupen64.exe". After doing this, re-open Mupen and click refresh on STROOP, it should be there.

To connect to the selected emulator in the list, you simply click connect. This will open STROOP in your SM64 instance. The other buttons found on the menu are also helpful, and you should take note of them. "Refresh & Connect" will refresh the list then automatically connect to the first emulator found. "Open Saveste" will let you open a savestate in Mupen64's savestate format (.st, .savestate or .st1 to .st9) to read and edit that savestate's memory just as if you had the emulator paused. It's useful when you want to make changes to a savestate without opening an emulator. Finally, "Bypass" will open STROOP with no emulator or savestate connected. STROOP has a few features that don't require a connection to an emulator or a savestate, and this allows you to bypass this screen and go directly to the application with no SM64 data. Most features will be unusable in this mode.

1.3: Viewing the data

Once in, it's likely you either can't see any data or the data is corrupted and incorrect. If this happens, there's another easy fix for this. STROOP reads the emulator's memory, but it needs to know where in the emulator's memory the game's memory is. The value it's looking for is called "RAM start". Most emulators don't provide it, which is why you're likely to have trouble using them. But with how terrible STROOP is at automatically finding the RAM start, Mupen64 allows you to retrieve its RAM start to perform the following fix. First, go to Mupen, and depending on your Mupen version, either under "Utilities" or "Help", you will find an option called "Show RAM start". A dialogue box will pop up with the value you need and a request to send the entire STROOP configuration line into your clipboard. Either copy the text shown or click yes. Next, you want to go to the folder where your STROOP exe is located. Navigate to the Config folder, and open Config.xml in a text editor. Near the top, you should see a line indicated with text that it's the one that needs modifying. If you only copied the text starting with "0x" in Mupen, then replace the ramStart value between the quotation marks. If you clicked the option to copy the whole configuration line, you want to select the entire line of interest and press ctrl+V to overwrite it with a new value. Note that Mupen's automatically generated configuration line also fixes the bug from earlier with the exe name, so make sure that in the line you copied, the value for "processName" matches the name of the exe file. When you're done, remember to save the file, then close and re-open STROOP. If you did everything correctly, STROOP has opened and when connected to an unpaused emulator running SM64, you should see data updating live. Be careful, it's normal to not see anything if you're on the title screen. Wait until you get to file select before confirming that things are working.

Chapter 2: Global Features

Now that you've successfully connected STROOP to your emulator, you need to learn to navigate STROOP. STROOP is divided into tabs. Tabs are the main categories of STROOP, and each tab provides different services to the user. The bulk of this document will be going over every tab and everything you can do in each one. However, tabs only change the top two thirds of the program, so I'll start by explaining everything there is around them. Each chapter of this document will go over a specific tab or section, and information within the chapter will be somewhat sorted from most useful to least. Once you think I'm starting to talk about stuff you'll never need, feel free to skip to the next chapter or sub-chapter.

Starting off at the top, we have the disconnect button. This is how you disconnect from the emulator or savestate and go back to the connection menu you saw earlier. Directly next to it is an FPS count for STROOP, showing how often it's updating with info from the game. Next to it, the name of the process STROOP is currently attached to. Below are the tabs themselves.

2.1: Right clicking the version number

Of all the options in the top right of STROOP, the version number (v1.0.5) actually contains some of the most useful ones. You can get this menu by right clicking the version number. This opens a long list of buttons and toggles, but only some of them are very useful. This document will cover the useful options here and keep the rest for later.

"Open mapping" and "clear mapping" are for working with ROM hacks. A ROM hack may have a corresponding map file, which is a file with a .map extension that lists where everything is laid out in memory. Having this file allows you to use STROOP on certain SM64 ROM hacks. You load it in with the top option and clear it with the other. Not all ROM hacks are useful with STROOP, as custom objects get shown as invalid/unknown and custom physics changes can make STROOP's math incorrect.

The next option is one that is very useful and good to know about. "Inject Hitbox View Code" will inject into the game the custom code used by Pannenkoek in many videos starting with Wall, Floors and Ceilings Part 3. This shows the geometry type of triangles, all cylindrical hitboxes and even the orientation axis of walls. Although note that there's no way to remove the hitbox view code without loading a savestate without it or resetting the emulator. It gets injected into the game's memory, so any savestates made with it will also contain it. Furthermore, if you move around the static collision triangles with STROOP, you need to re-click this option to re-apply the patch and make it accurate once more. Note that you shouldn't use this patch while recording TASes meant to be played back without it, as it can cause desyncs due to the way it's implemented.

Next button is the "Free Movement Action" button. This will set Mario's action state to be that of the built-in debug movement action left in the game by the developers. For the controls, you move laterally with the joystick, vertically up or down with the D-pad, hold B to move faster, hold L to move slower, and press A to exit back to normal movement.

Clicking on "Everything in File" will set your currently selected save file to be 100% complete, giving you all stars and unlocking everything that is obtainable without hacks. "Save as Savestate" will allow

you to save whatever is loaded into STROOP as a Mupen format savestate. If you're using STROOP to edit a savestate, this button is crucial.

I'll skip over unimportant buttons in this list and cover them later in this chapter, so we'll skip directly to "Download Latest STROOP Release", which when clicked will download from your default browser the latest version of STROOP found on the project's GitHub as a separate zip file. STROOP doesn't have any automatic update system, to get a new version you have to delete the old version and set up the new one. "Copy Download Link" simply puts the download link the previous option uses into your clipboard so you can use it yourself.

"Documentation" will open up the Ukikipedia page for STROOP, which contains the beginning of a complete list of all its features. But at the time of writing, all of the information for the tabs isn't there, which is why this document was made. However, this is still useful as a quick list of useful things you can do with variables.

2.2: Standalone buttons

Back to the left, we have three buttons to modify the tabs. The left and right arrows move the placement of the currently selected tab either to the left or right. This is so that you can organize the order of your tabs. Next is the plus button. Clicking on this will bring down a menu of hidden tabs not shown to you initially. Click on a specific tab to have it added at the end of your tab list. Clicking on "Restore All Tabs" will add all hidden tabs to your tab list. To hide a tab, click on it while holding control. Right clicking either of the arrow buttons will display a hidden option for restoring the recommended tab order. However I advise not using this, as there's a glitch where it empties your hidden tabs list before manually adding the tabs it removes, making it so the list has duplicates. Also, the recommended tab order is different from the one you get when you open STROOP for the first time.

Next to the buttons is a drop-down menu for which version of the game you're currently playing. There are 4 versions of the game, and you can select which version STROOP should use as a basis on where to find stuff. It's automatic by default, and you really should never have to change this. After this is the drop-down menu for write protection. This is for when you want to really make sure you don't accidentally overwrite something, and you're just using STROOP to read data. Setting this to "Read Only" will prevent you from modifying any data in SM64's memory.

The next 6 buttons are for hiding and showing different sections of STROOP. You should think of these buttons as 2 groups of 3 buttons. The first three are for toggling stuff in the tab-specific zone. Most tabs have a left section and a right section, well these buttons allow you to show either only the left section or only the right section, or both which is how it normally is. The same goes with the next 3 buttons, but this is for the top tab-specific section and the object list below which I'll get to next chapter. You can show either only the top, only the bottom, or both. If the current tab has more than 2 panels, you can hold a number key while clicking the buttons to be more specific with which panels are shown or hidden.

2.3: Lock and gear icons

Clicking on the lock will open an options menu for locking variables. I'll explain locking later, but for now just think of it as variable-specific write protection. When clicked, this menu allows you to see how many locks you have applied on, the option to remove all of them at once, and has another option

for removing a specific lock, which lets you select which lock you want to remove. The lock icon also closes the moment you have at least 1 lock active and opens up when you have none. The next option is a toggle to disable all locks. All lock icons become blue to indicate they're inactive. Untick the option to re-enable locks. Finally, "See Lock Info" will open up a new window with a table showing information on all currently active locks. This window is by default way too small to see anything or even recognize it as a table, so you have to stretch it wider to see everything. And indeed, this gives information on each lock you have applied, along with the variable it's locking. Because of the way this table is made, if a variable's name is too long, it can desync the values from their respective columns, shifting everything over by one.

Next up is the settings icon. Note that everything found here can also be found on the Settings tab, and the icon in the top right even has an option to send you to the settings tab. So I'll hold out on these options for the moment, I'll explain them in the settings tab chapter.

2.4: Remaining version number buttons

And now for the less useful version number options skipped over earlier. "Go to closest floor vertex" will teleport Mario to his closest floor vertex on the floor he's currently above, which can be better seen with the hitbox view code active. "Show MHS vars" creates a pop-up window with specific variables listed in a configurable XML file. MHS refers to an old memory watch program some people used to use before STROOP, so this is for people still used to that.

"Copy Mario state" puts into your clipboard the values of Mario's main variables formatted as C# code.

"Show all helpful hints" pops open a new window listing every loading screen tip you are able to see when opening up STROOP and waiting for it to load. You can also get this list by right clicking the tip shown on the loading screen, but that window will close as soon as STROOP loads. Don't trust this list completely, because some of the things on it are outdated and don't exist anymore or were moved somewhere else.

"Enable TASer settings" is a shortcut for sending you to the TAS tab and hiding the left half of the tab window. Clicking this hidden option takes the same number of clicks as setting it up yourself, so I don't know why this exists.

"Add gfx vertices" adds coordinates as variables in the Triangles tab. The coordinates will be of the graphics triangle whose coordinates match that of the currently selected collision triangle in the triangle tab. If you click this option while holding control, it will search through groups of 4 vertices, and holding a number key will search for groups of that many vertices. This function scans through all of RAM, so it's very inefficient and lags STROOP for a long time. If it succeeds, the variables will be added to the start of the triangle tab.

"Show skribblio words" is really cool, clicking on this opens a window containing a list of every star name, object name, level name and thing found in SM64 and separates them with commas. This is so that you can copy the list and paste it into any game that accepts a custom word list. As the name implies this was originally made for skribbl.io, but it also works with other games like codenames.

"Show Image Form" opens a window with no background. With it, you can open any image on your computer and view it here. The slider changes the image's transparency. However, terrible form design causes the transparent background to become pink, even though it's supposed to be transparent. Also, another coding error makes the pink turn into black when you bring the slider back to normal.

"Show Coin Ring Display Form" opens a window that shows 5 coin diamonds with the number 8 below them. It's meant to show which coins and secrets are currently collected from the floating rings in BOB, but the hardcoded addresses in the code don't point to the objects they're supposed to point to, so it doesn't work.

"Copy TTC XML" this button puts into your clipboard XML data containing the results of a TTC simulation. The code doesn't actually run the simulation, so you just get the initial values of everything, but it does use the current RNG and frame number values to get the results.

"Test Something" and "Test Something Else" are used for running test code, and as such what they do changes wildly depending on the version. As of writing, the first one is an algorithm to help with a bruteforcer. It takes values from the custom tab and outputs a sequence in the dev console, raising an error if the values entered are 0. The other is used to find wall gaps, outputting in a pop-up window the number of float increments between the two common vertices of Mario's current floor and wall triangles. Needless to say, you shouldn't click these. They might crash STROOP one day.

"Format Subtitles" takes the text currently in your clipboard, assumes it's in an RTF format, and processes it to remove RTF tags and remove duplicate sections. It outputs the result in a pop-up window.

Finally, "Update Cam Hack Angle" and "Update Floor Tri". These two are special because they're toggles and not buttons. They were used by Pannenkoek when recording the "koopa running around" videos. Both only do something when selecting an object with the hacked camera from the cam hack tab. The first one makes the camera move around the koopa depending on its position, and the second updates Mario's room so that the koopa is always loaded even if he enters a different room.

Chapter 3: Object grid

STROOP, as its name implies, is first and foremost an object processor. When it was first designed by Pannenkoek, it was meant to be used to help with cloning. Everything else came afterwards. This can be seen in the design of STROOP, where at all times (unless hidden by the user), the bottom of the application contains the game's object list. This is the same list that's mentioned in some of Pannen's older cloning videos, and as such it keeps the same format. There's a fixed number of object slots, and unloaded objects get moved to the end of the list, turned into vacant slots. The slots go from left to right top to bottom and are indexed starting at 1. And so are the vacant slots (abbreviated to VS), they're indexed with VS#.

3.1: The object slots

The object grid allows you to see all currently loaded objects, ways in which they're interacting with Mario, their category and also what data remains in the vacant slots. Clicking on an object will open that object up in the Object tab, unless you're on a tab where clicking on the object opens it in that specific tab. Holding control when clicking will allow you to select multiple objects at once, and holding shift will let you select a range. Selecting multiple objects at once does let you perform actions on all of them at once as long as you hold control while doing so.

The color of the slot the object is in indicates its processing group, which is how the objects are sorted in STROOP by default. Objects are processed in the order you see, but the categories help define their purpose. Here are the meaning of each color category, in order: pink is for some but not all object spawners, red is for tangible objects, orange is for objects that are "useable", yellow-orange is for Mario, yellow is for pushable objects, green is for actors or important unique objects, light blue is for objects that respawn, dark blue is for everything else that's part of the level, purple is for anything that doesn't fit in the other categories, and finally brown is for unimportant objects. Unimportant objects have the unique property of being able to be unloaded automatically if there are no vacant slots remaining.

Different icons can appear on top of an object slot to indicate its current relation to STROOP, Mario or the camera. The following is every icon along with its meaning. The four red corners indicate the object is selected and being showcased in the object menu. The black or colored border icon indicates an object is marked. The blue parentheses icon indicates that this object is currently the closest one to Mario (that isn't dust or Mario himself). The two hands icon means the object is being held by Mario. The Mario feet icon means the object is being stood on by Mario. The Mario head looking right icon is for an object that's being interacted with by Mario, and the Mario head looking left is for objects that are being used by Mario. Those are two different things. The shell icon is for objects Mario is currently riding. The camera icon is for objects being focused on by the camera that aren't Mario. The blue line, green line and red line are for when Mario's referenced floor, wall or ceiling is from this object. For floors and ceilings, Mario does not have to be touching the triangle for this to appear, just that it's currently above or below him. The four vertical parentheses are for showing the parent and children of an object. They can be seen when holding down the P key over an object slot, and will appear on other object slots. The pink one indicates the object's parent, the light blue one indicates the object's children, the gray one indicates the object doesn't need a parent and the black one indicates an object doesn't have a parent. The red dot indicates that the object's hitbox is touching Mario's hitbox. It's disabled by default in the options menu. Finally, the yellow dots note

objects colliding with Mario. Objects are only capable of colliding with up to 4 objects, so the position of the dot indicates in which slot it's in. If you hover over an object slot with the C key pressed, the yellow dot will move to show which objects are colliding with it instead of Mario.

You can perform some actions directly on these object slots by right clicking on a slot. "Select in object tab" opens the object in the object tab. This is useful when you're currently in a tab where clicking the object doesn't do that. "Select in memory tab" opens the object in the memory tab. "Select in current tab" is broken and doesn't work. Except for the object tab, but that already has a button. All the options from "Go to" to "Ukikipedia" are shortcuts for buttons that can be found in the object tab, so I'll explain what each does in that chapter. "Mark" and "Unmark" are for adding a thick black border to the object slot. This helps keep track of it in the object list or for performing certain operations with them. You can hold alt when clicking an object slot to mark or unmark it.

The next options are all for copying or pasting parts of the object's data. Pasting data works as you'd expect, if the text in your clipboard is formatted correctly, the data will be inserted. "Copy address" gives where the object is in memory, "copy label" is the object slot's current label, "copy position" is the object's position in XYZ separated with commas, "copy graphics" is the address of the object's visual 3D model in memory, "copy object" will copy the entirety of an object, saved in a custom STROOP format. This one actually doesn't go to your clipboard, but rather STROOP's memory. This means that when you restart STROOP, it also forgets what object(s) you have copied.

There's a final hidden menu that you can get when right clicking the space in between the object slots, revealing a few more options. "Select marked slots" selects slots that are marked. "Select specific marked slots" selects only the marked slots marked with a specific color. Specific color? Yes, that's right, because if you click an object slot while holding a number key, the marking color will be different. The marked overlay is normally black, but there are 10 additional colors for the 10 number keys. "Select copied address" will read your clipboard, and if it's the address of an object, it will select the object with that address. "Clear marked slots" removes all marks. "Clear selected slots" removes all selections. "Unload all but marked slots" does exactly what it says, it unloads every single object that isn't marked. Finally "Display as row" when toggled on will display objects slots as a single gigantic row instead of a grid.

As a reminder, for all of these right click options, you have to hold control when selecting them if you want them to apply to all currently selected objects. This is also how you open multiple objects at once in the object tab.

3.2: The top options

Most of these top options aren't important, but the size slider lets you modify the size of the object slots. It's very slow and buggy, but it'll work eventually. You can right click it to get an option to restore the original slot size, but this causes an error when shrinking the slots in this way, making some slots error out when used.

For the remaining top options, the slot indices update automatically, but you can freeze them in place by activating the "Lock labels" checkmark. "Selection Method" allows you to choose how you select objects to highlight them in the object menu. By default you click on them, but you can also set it so that objects are automatically selected when a specific interaction happens, like Mario standing on

them, or them being the closest object to Mario. This also means it updates live, so you don't have to focus on STROOP too much.

Next is the Labeling Method, which affects how the slot labels work. This only affects the labels, not the objects themselves or the order they're in. Recommended will pick the option best matching the Sort Method, which is the next setting. But when that's not changed, recommended picks the "SlotPosVs" option, which is what was described earlier. "SlotPos" is the same but without differentiating vacant slots, so the indexing continues all the way to the end. "SlotIndex" makes the number be the order of the slots in memory, which means the numbers don't follow eachother with the default object sorting. Finally, "RNGusage" will remove the squares and colors, turning every index into an indicator for how many times that object called RNG. However, without a settings from the hacks tab, it doesn't work.

Finally, there's the "SortMethod". This affects the order the objects are shown in. By default it's "ProcessingOrder", which is the order the objects are processed in, there's "MemoryOrder" which orders the objects by their position in memory. When this option is selected along with having the labelling method be automatic, the labels will switch to slot index, keeping the indices consecutive. "DistanceToMario" will order objects by how close they are to Mario, and obviously this will make Mario always be in the first slot, because Mario is the closest to Mario. Finally, there's "LockedLabels" sorting, which orders objects by the value of their label, so it's useful when you have locked labels enabled.

Additionally, using the left and right arrows at the very top of STROOP, you can move the object slots left and right through the object list. Hold down control when clicking the arrows to move the selected object slot(s) left or right by one. Hold down a number key instead to move the slots by that number of positions. If the sorting method is processing order, STROOP will automatically change the background color of the object as it moves from one processing list to another. It prioritizes doing this before moving the object left or right. If sorting method is set to memory order, it moves the object in memory. You can see this by its memory address changing. Neither of the other sorting methods have something happen when using the arrows.

Chapter 4: Variables

Before moving on to things each tab can do, it's very important to first cover something many tabs have in common, and that's variables. Variables are the containers for each of the values in the game and those made by STROOP, and there are many, many things you can do with them. They almost always appear on the right side of the tab panel, with each variable having a name and a value. These values update in real time to match what the emulator currently has in memory. But memory watching isn't limited to reading information, you can also add your own! Double click on the value of a variable to edit it, typing in the new value you want it to have, then press enter. If what you're trying to do is impossible or invalid, STROOP will flash the variable red for a brief second to indicate that. Not all variables come from the game, many are generated by STROOP. Some STROOP-made variables can be modified. STROOP has code to modify the in-game variables it uses to generate its values. But not all STROOP-made variables have these custom setting functions, and those will flash red to show they can't be modified. If successful, your modification will instantly be sent to the emulator, although it will take a frame for the visuals of the game to update accordingly.

Variables come in many different types, which you will be familiar with if you've ever programmed in a C style language. There are bytes, shorts, integers, signed variants of the three, floats, doubles, booleans and pointers. SM64 doesn't use 64-bit integers, so long isn't supported as a type in STROOP. Luckily you don't have to manually keep track of a variable's limits, as STROOP will automatically truncate or convert invalid numbers.

When modifying positions and angles, remember that in SM64, the Y axis is the vertical axis, and there are 65536 degrees in a rotation instead of 360.

Double clicking on the name of a variable will bring up a new window containing information about it. "Class" is what the variable is used for, such as number, angle or memory address. "Type" is that variable's type, along with the size in bytes in parentheses. "Basetype + offset" says which SM64 memory structure the variable belongs to, and what its memory offset is from the start of that structure. If it says "relative", that's because the variable is a global variable, and thus the offset will actually just be its memory address. "N64 base address" and "emulator base address" give the address of the base structure of this variable, both on the N64 and for the emulator. "N64 address" and "emulator address" give the real address of the variable, for both the N64 and the emulator. If a tool in STROOP asks for the memory address of a variable, it will always be referring to the N64 address unless specified. If a STROOP variable has no N64 or emulator address and its offset value is text instead of a number, it means the variable isn't actually in the game but is generated by STROOP. This method is the quickest way to tell if a variable comes from the game or STROOP.

4.1: The right click menu

More options on manipulating variables can be found by right clicking a variable. This first right click menu is obtained by simply right clicking a variable without having it selected. "Highlight" adds a red rectangle around a variable, making it easier to find at a glance. "Lock" puts a lock icon next to a variable and makes it so that every STROOP update frame, it sets the value of that variable to be whatever it was when it was locked. Depending on processing order, locking variables can more or less look jittery in game, but it works. The lock settings described in the global features chapter affect

these. "Copy" copies the value of the variable to your clipboard, and "Paste" attempts to put that value into the variable.

Now for some options, whether they appear depends on the type of variable you have clicked. If it's part of a triplet of variables like XYZ or width/height/depth, you'll see two options called "Copy coordinates" and "Paste coordinates". This is for copying the entire triplet of values instead of just a single variable. As seen by the arrow, there are multiple ways to copy the coordinate triplet. These sub-options are simply for choosing how the values are separated, whether that be by commas, spaces, tabs, new lines or commas and spaces. Pasting coordinates accepts all of these formats, so you don't have to worry. Note that STROOP incorrectly parses lists of data if your computer's regional settings have the decimal character as something other than a period.

The next two options are always there. The "Round to ..." sub-menu allows you to choose how many digits of precision you want to show in STROOP. This appears even if the value is an integer or a boolean, but this only affects floats and doubles. Selecting 0 rounds to the nearest integer, selecting a greater number will display that many digits or less if more aren't required, and selecting "No rounding" will display all of the digits, up until it displays the value in scientific notation. "Display as hex" will show the value in hexadecimal and can be toggled on and off.

The next options only appear if the selected variable is an angle. "Signed" is a toggle to display the value as a signed short instead of an unsigned one, changing the range to -32768 to 32767. "Units" allows you to select the units the angle is displayed in. By default it's SM64's in-game 65536 degrees, but you can change this to several other options. Note that for all of these, the result will be rounded to the nearest integer in STROOP, but the actual value will remain the same in memory if left unedited. "In-game units" is the default and what was described, "HAU" stands for Hexadecimal Angle Units, it's simply the In-game units divided by 16. It's useful for Mario movement, where the game truncates his angle to a multiple of 16. "Degrees" will display the angle in the standard range of 0 to 359 degrees, "Radians" will show the angle in radians, although this is less useful in low quantities due to the rounding. Finally, "Revolutions" displays the number of full rotations the angle represents. "Truncate to multiple of 16" will round the displayed value down to the smallest multiple of 16. This rounding takes place before the conversion to the units selected from earlier. "Constrain to one revolution" is enabled by default on some variables. It forces the angle's value to always remain within 1 full rotation, performing a modulo operation on it if it exceeds that range to bring it back in. "Reverse" displays the angle that would be exactly half a rotation away from the real value.

This next option only appears if the selected variable is a pointer, and it's simply called "view address". All it does is open that pointer in the memory tab. Unless that pointer is null (zero), in which case nothing will happen.

The next two options only appear if the selected variable is an object. "Display as object" toggles between showing the object's index in the object list or the underlying pointer being referenced. "Select object" will select the object referenced by the value. If the selected variable is a pointer to a triangle, the option "select triangle" appears, which selects the triangle in the triangles tab.

"Display as checkbox" and "display as inverted" only appear if the variable is a boolean. If the first one gets untoggled, the variable will display the entire bitfield it's a part of as an integer. If the second one is toggled, 0 means activated and 1 means not activated.

The two controller options have a lot to them, so they will be left for the next sub-chapter.

"Add to custom tab" adds that variable to the custom tab, which will be explained in its chapter. "Fix address" means that a variable will keep track of its base value as well as its offset to it. By default, these variables will show the value for the currently selected thing, but fixing a variable means it keeps referencing that value for the thing that was selected in STROOP when it was fixed, and not the currently selected thing. "Rename" allows you to modify the variable's name and "Remove" removes that variable from your view. Obviously that variable still exists in-game, but you just can't see it now. To bring back removed variables, you reset STROOP or use an option talked about in the generic right click menu sub-chapter.

4.2: Controllers

There are two options for controllers in the variable right click menu. Controllers are a set of controls for modifying one or more specific variables, often in a dedicated window or on the side of a tab. By selecting "open controller", we can showcase the basic controller. A small window will pop up showcasing the name of the variable, its current value and some other controls. The plus and minus buttons will either add or subtract the value found in between them to the value of the variable. Holding control while holding down left click on either the plus or minus buttons will add or subtract the value every frame until let go. This middle value defaults to 100 but can be changed to be whatever integer you want. Real numbers don't seem to work, even if the variable is a float. Below are get and set buttons along with a third value. This is a sort of "save state" for the variable. When you press the get button, the value of the variable will be transferred to the middle value. When you press set, the middle value will become the variable's new value. You cannot edit the variable's current value in the controller, but you can modify the savestate value and press enter, which does the same thing. Finally, the fix and lock toggles. Lock activates the lock for the variable, but the controller bypasses the lock and modifies the value anyway. As for the fix toggle, it fixes the variable. Furthermore, right clicking the plus and minus buttons gives a small menu. You can start and stop continuous add or subtract, which simply adds or subtracts the value in the middle every STROOP update frame much like if you were to hold control and left click. The last option is a toggle to switch the position of the plus and minus buttons.

Clicking on "Open bit controller" will open a different looking controller to the first one. The bit controller, as the name would suggest, is designed for bitwise low-level manipulation of the values. At the top you still have the variable name and value, but now just below it are its value in hexadecimal and below that its value in binary, with a space separating each byte. Note that N64 memory is little endian, so the bytes of multi-byte values are ordered from right to left in memory. All three of these values cannot be edited. Below is the main attraction of the controller, a table displaying a checkbox for every bit in the value. The number of rows shown correspond with the type of the variable. The first column indicates the index of the byte in memory relative to the variable's address. It's in descending order due to the endianness. The second column shows the value of that specific byte in decimal, then the third column that value in hexadecimal, the fourth column that value in binary, and finally, the 8 bits of the byte, ordered from most significant to least significant. If the variable being controlled is a float, special background colors will be added to the checkboxes to indicate the location of the sign, exponent and mantissa bits. You can manipulate each of these checkboxes to change the value in real time, bit by bit. Finally, there exists a secret right click menu you get by right clicking anywhere

in the bit table that allows you to toggle between showing the value and showing float components. If the controlled variable is not a float, this won't do anything. But if it is, selecting the second option will change the top 3 values to showcase the real numbers that make up the float. The float is calculated by doing A times C times 2 to the power of B.

It is worth being noted that bitwise controllers cannot be used on variables that are generated by STROOP and not found in the game.

4.3: Generic right click menu

There is a second right click menu for when you click on a space in a variable panel not occupied by a variable. This is where the more generic options for the variables panel hide. "Reset variables" will undo any variable status modification or deletion. This will NOT modify the actual value of the variables, just STROOP-specific stuff like the name and whether it was removed. There's a glitch where specific variables won't re-appear, but this is only visual and can be fixed by clicking on a vacant object slot then clicking back where you were. "Clear all but highlighted" will remove all variables except the ones that are highlighted. This also unhighlights the remaining variables. "Fix vertical scroll" removes and re-adds all variable controls, presumably as a fix to a glitch of some sort.

"Add custom variables" allows you to add your own variables to the variable list. You select its name, its type from the available types, which memory structure it's based off of and what its offset into that struct is. Clicking "add variable" in this window won't close it, allowing you to add another with only slight modifications. The new variable will represent the selected location in memory and can be used like any other. Right clicking the add variable button in this window lets you enable or disable mapping. If disabled, STROOP will not bother checking the map file you inserted into the program when looking for addresses.

"Add mapping variables" adds variables to STROOP based off of the provided .map file for ROM hacks. "Add dummy variable" will create a new variable of the specified type that doesn't represent any part of the game's memory. It does absolutely nothing. If control is held when selecting a type, a new window will show up allowing you to specify how many dummy variables you want to add at once.

"Open / Save / Clear" opens a sub-menu containing a list of actions mostly pertaining to saving or loading the current variable setup. STROOP allows you to save your current variable setup in a custom .stv format and load it later. "Load" will add the contents of an stv file onto whatever you currently have. "Open as pop out" will open the variables in a new window. This new window contains its own options in its right click menu. One for toggling window borders, one for toggling the window always being on top of other windows, and finally one to close the window. "Save in place" will overwrite STROOP's default settings to have the current setup become the default setup. As the pop up warning tells you, this action cannot be undone, and going back would require re-downloading STROOP. "Save as" lets you export your current setup as an .stv file. Finally, the clear button, which removes all variables regardless of if they're highlighted or not.

"Do to all variables" allows you to perform a right click menu action on all variables at once. This opens the secret larger right click menu that will be explained in the next sub-chapter.

"Filter variables" allows you to view secret variables that big variable doesn't want you to see. The most necessary categories are visible by default, but there can be some hidden categories only

accessible through this menu. These depend on the current tab, some have many hidden categories while most have none. The vast, vast majority of these variables are simply a mathematical expression using other variables, or a component of another variable before it's been turned into something useful. 99.9% of STROOP users will never need these. They will be covered in this document, but with much less detail or care and in their own respective sub-chapters.

4.4: Extended right click menu

There exists an even larger right click menu for variables. It can be accessed by first selecting one or more variables. To view this menu for a single variable, left click its name to select it then right click it. You can select variables while holding control and shift in the same way you can select object slots, which means that this is the only menu that shows up when selecting multiple objects. It's the same no matter what and features mostly things found in the simplified menu but in a more verbose way, so this document will only be covering new things. Menu options that used to only appear for one type of variable always appear but only affect that type of variable.

In the highlight menu, you can specify what color you want your highlight to be, changing it from the default red. There is a new third option when fixing an address, called "fix address special". It's a quick hack made for coin rings, where it fixes on a slot instead of an object.

Copying variables now lets you choose the formatting of the separator. Most of the options were already seen in the option for copying triplets, but there are some new ones. Copy with names, copy as table and copy as code. "Copy with names" adds the name of the variable with a tab before the value, "copy as table" is similar to copy with names but adds a row before the values, saying "vars" and the address of the first selected variable. And finally, "copy as code" copies the variables as if you were initializing a new C# variable. If you click on this last one while holding control, a window will pop up with a dollar sign symbol. This lets you set the formatting for the variable names. The dollar symbol will be what's replaced with the variable's original name.

You can no longer automatically copy an XYZ triplet, but selecting multiple variables at once in this menu allows you to group anything together in your copy.

When it comes to pasting the copied data, the code for pasting in the data of multiple variables has a glitch that only affects you depending on your computer's region settings. No matter which format you chose to copy the variables, pasting always expects floats to have their decimal portion separated with a period. Once again, if your computer's default language is that of a region where this isn't the case, the data will be corrupted, and non-integer floats will not be pasted correctly. This also messes up the data that comes after the float.

Angles now have their own "display as hex" option, on top of the pre-existing display as hex option for all variables. Meaning that in this menu there are two different ways to display an angle as hexadecimal.

"Show variable XML" will open a window displaying the variable's metadata in XML format. "Show variable info" will display it as a table instead, but the window is small by default so you have to stretch it out for it to look right.

"Add variables..." this option allows you to create new variables whose value is a mathematical operation on one or multiple other variables. The order you select variables in is what matters, and if you select more than required, it will only consider the first ones. This new value updates in real time along with its sources. You have a great variety of operations to choose from, but most of them require a certain number of variables to be picked at once. In most contexts, when modifying the value of variables made with this menu option, holding control will modify the original variables in a different way. This is also the case for STROOP-made variables involving distances between things.

"Addition", "subtraction", "multiplication", "division", "modulo", "non-negative modulo" and "exponent" require two variables. For those who don't know, the difference between the two modulos is that regular modulo is the standardized C style modulo that acts different with negative numbers, while non-negative modulo acts the same with positive and negative. "Mean", "median", "min", "max" and "sum" take any number of variables you want; "2D distance" and "3D distance" take 4 and 6 variables respectively, they interpret their inputs as points in 2D or 3D space. "Real time" interprets the inputted number as a number of frames, and outputs that number of frames converted to real time, assuming SM64's standard 30 frames per second. For floats, it rounds to the nearest integer. "Action description" interprets the input as an action bitfield and gives the action name if that results in a valid action. Finally, "dereferenced" interprets the value as a pointer, and it shows the value located at that pointer, interpreting it as the type you choose in the sub-menu. If you hold control when clicking this option, a window will appear allowing you to add an offset in bytes to that pointer.

"Set cascading values" will ask you for two inputs. An initial value and offset value. What this will do is it will go through all variables you've selected and set the first one to the initial value, the second to the initial value plus one times the offset, the third to the initial value plus two times the offset, and so on. When one variable is selected, the offset value does nothing.

"Set background color" lets you modify the background color of that variable slot. The default color options provided are all rather light, but you can select custom color to open a color picker and pick your own color. Control color is white, and default is whatever the variable was before you changed it. "Last custom color" sets it to the last custom color you've chosen.

The "move" menu allows you to rearrange the order of variables. To move a variable, you select "start move", then go to the extended right click menu of another variable and select in the move menu "end move". The starting variable will then move one slot past the variable you used as the finish line. To cancel a move, select "clear move" in the move menu. "Rename" now works differently in the extended right click menu. It opens a form prompting you to enter a value, with a dollar sign already in place. This lets you create a format for variable names, where the dollar symbol will be replaced with the variable's original name. This is useful for when you're renaming many variables at once and want them to all start with the same thing. Additional note, most times in STROOP when a window opens up and prompts you to enter a value, you can right click the OK button to get an option called "use clipboard". This retrieves the text from your clipboard and uses that as the value entered.

"Open triplet controller" is a secret third type of controller which is actually taken from the side bar for some tabs, so I'll refrain from explaining it for now, but it'll be in the object tab chapter. Just know that for this one to work, you have to select at least 3 variables at once. If you select more than 3, your first 3 selections will be the ones to be modified as X, Y and Z in that order.

"Open pop out" duplicates all selected variables to a new window. "Add to tab" duplicates all selected variables to a different tab of your choosing. There are 4 ways to transfer the variables. "Regular", which just sends them over as they are without fixing the memory address. "Fixed", which fixes the memory address of all the variables, "grouped by base address" and "grouped by variable" which are variations of fixed. The first one sorts the variables by their base address, and the second sorts them by the offset of the variable. "Add to custom tab" is just the previous option with its sub-menu but it sends the variables directly to the custom tab instead of asking you.

4.5: Shortcuts

STROOP has many, many shortcuts for variables. Most of them are mentioned in the loading screen tips, but others not. Here are all of them.

Clicking on a variable while holding down a number key will highlight that variable with a color corresponding to the held key. 1 is red, 2 is orange, 3 is yellow, 4 is green, 5 is blue, 6 is purple, 7 is pink, 8 is brown, 9 is black and 0 is white.
Clicking on a variable while holding both control and a number key will color that variable depending on which key is held. 1 is red, 2 is yellow, 3 is beige, 4 is green, 5 and 6 are blue, 7 and 8 are purple, 9 is gray and 0 is white.
Clicking on a variable while holding S will send it to the custom tab.
Clicking on a variable while holding T will send it to the TAS tab.
Clicking on a variable while holding I will send it to the script tab.
Clicking on a variable while holding M will send it to the memory tab.
Clicking on a variable while holding F will fix that variable.
Clicking on a variable while holding H will highlight that variable.
Clicking on a variable while holding L will lock or unlock it.
Clicking on a variable while holding D will toggle displaying it as hexadecimal.
Clicking on a variable while holding R will rename it.
Clicking on a variable while holding C will open it in a variable controller.
Clicking on a variable while holding B will open it in a bit controller.
Clicking on a variable while holding P will paste your clipboard into it.
Clicking on a variable while holding delete, backspace or escape will delete it.
Clicking on a variable while holding the backtick key will move it to the hacks tab.
Clicking on a variable while holding Z held will set its value to zero.
Clicking on a variable while holding Q will let you pick a custom color for its background.
Clicking on a variable while holding N will open it in the memory tab if it's not STROOP-made.
Clicking on a variable while holding O will set its background to the last custom color you picked.
Clicking on a variable while holding plus(equals) or minus will add or subtract one from its value.
Clicking on a variable while holding X will start its moving process. Then clicking on another variable while holding X will move that first variable one spot past it.

When selecting add to custom tab in the standard right click menu, click the option while holding G or A to have the variable be sent fixed instead of normally.

Chapter 5: Object tab

The object tab is STROOP's main tab. It displays all relevant information on the currently selected object(s). In the top left corner, you get the icon seen in the object list below, and that object's name. If multiple objects are selected the name will be replaced with the number of selected objects and the icon will contain the icons of all selected objects. There's information besides it such as "bhv", which is the latter 2 bytes of the pointer to the object's behavior script, which can be used as a sort of ID for that object, even if it isn't necessarily unique. Then, "slot pos" for its current slot position in processing order, "slot index" for its current slot index in memory order, and "add", a pointer to its address in memory.

In general, the object tab functions well when multiple objects are selected. The tools on the left will act on all of them and variables will display their value if it's the same or "(multiple values)" if they vary. Setting these variables will act upon all objects.

5.1: Quick action buttons

The following 10 buttons on the left panel are the most useful for messing around with objects and is also what we saw earlier in the object slots chapter where you could access these buttons remotely from an object slot's right click menu. Those shortcuts access the default option for these buttons, not any of their right click menu options.

"Go to" sets Mario's position to 300 units vertically above the object. When right clicking this button, you have more options. "Goto laterally" will set Mario's X and Z but not Y, "goto X", Y and Z will set that axis but not the others. "Goto center top" will set Mario's position to 300 units above the median of all that object's collision triangles. If the object has no collision or is too far away, this won't do anything. "Goto center laterally" does the same but without changing Mario's Y coordinate. If multiple objects are selected, it will move Mario to an average of the positions.

"Retrieve" sets the object's position to 300 units vertically above Mario. When right clicking this button, you can select "retrieve laterally" to set the object's X and Z but not Y, or "retrieve X", Y or Z to set only a specific axis. If multiple objects are selected, it will move them all at once.

"Go to home" sets Mario's position to 300 units vertically above the object's home. Right clicking this button will once again bring up more options for choosing to go to the home laterally or on one axis, but also give a duplicate of every option with the format of "object goto home". What this does is move the selected object to its own home. If multiple objects are selected, goto home will send Mario to 300 units above the average of all the homes and object goto home will send each object to 300 units above its respective home.

"Retrieve home" sets the object's home position to 300 units vertically above Mario. Right clicking this button will bring up options very similar to the ones for going to the home, and it works the same. For "object retrieve home" options in the right click menu, it has the object's home go to 300 units above it. If multiple objects are selected, for "retrieve home" and its derivatives you retrieve all of their homes at once and for " retrieve home to object" each object retrieves their own home.

"Release" tells the object that it's been released by Mario. This doesn't affect much. "Unrelease" does the opposite. You can right click this button to select either option or choose whether you want to

have the fake release call use the values for throwing or dropping with "release by throwing" or "release by dropping".

"Interact" tricks the object into thinking Mario has successfully interacted with it. "Uninteract" removes the interaction flag manually. You can right click this button to choose specifically one of the two.

"Clone" tells Mario that he's now holding this object, allowing him to clone it. "Unclone" removes the object from Mario's hands without disturbing the original object. You can right click this button and select "clone without action update" or "unclone without action update" to hold the object hands free.

"Unload" forcefully unloads the object from the game. "Revive" forcefully brings the object back to life. These actions take an extra frame to fully complete. You can right click the button to specifically select one of the two.

"Ride" tells Mario that he is now riding on that object, as if it were a shell. Visually, the object will do nothing, Mario will be riding on nothing, but Mario will always act like he's on a shell. "Unride" brings Mario back to normal. You can right click this button and select "ride without action update" and "unride without action update" to send the signal to the object that it's being ridden or unridden but without making Mario enter or exit a shell action.

"Ukikipedia" opens the Ukikipedia page for that object. Not all objects have Ukikipedia pages, so it's likely that it will open a blank or missing page.

5.2: Object controllers

The remainder of the left panel for the object tab are object controllers. These allow for easier control over the object's most important variables.

First, we have the controller for the object's position. This is our first example of a 3D controller. These buttons will move the object in either X, Y or Z, both positive and negative. For the lateral axes, you can move diagonally. The amount moved for X and Z will be this number in the middle, while vertically the amount moved will be this number, which both default to 100 units. This checkbox up here labeled "relative" will change the meaning of the directions. Instead of X, Y, Z, the axes will now become forwards/backwards, left/right and up/down. This takes into account the object's current facing angle (yaw), moving it relative to that instead of the world's axes. For this specific position controller, you can click on the buttons while holding control to move Mario as well as the object. You can also hold alt while clicking to move the object's home along with the object.

3D controllers appear in many places across STROOP, and they all look similar to this, and share common features. For example, right clicking a 3D controller will open up more options. By default, Z- is the direction of the upmost button of the controller. However, the first 8 options of this right click menu let you rotate the buttons around so that any direction can be the up button, including diagonals. Selecting "inverted" will flip the meaning of the buttons horizontally. "Pop out" will open the controller in a new window. "Pop out fixed" doesn't appear all the time, but when it does, it's to have a controller for that one specific thing instead of whatever is selected. If you right click the Y part of the 3D controller, your only option is to invert the buttons. In general, all controllers have an invert

option, including the smaller ones. By default, the diagonal lateral buttons (like X+Z+, etc.) will change the lateral variables together by the middle number of units, meaning that each of the two axes have their values changed by the middle number times the square root of 2 divided by 2. This can be changed in the options tab to make it so each axis is modified by the middle number individually.

Next is the object's angle. When working with angles in SM64, you'll have to be familiar with yaw, pitch and roll, as that's the terminology used here. And also remember how one full turn is 65536 degrees. So for the angle controller, it's simpler. You have yaw, pitch and roll, buttons to add or subtract from the angle, and the number in between each for how much you add or remove, which defaults to 1024, or 1/64th of a turn. The angles will automatically wrap around when passing either 0 or 65535, so you don't have to worry about overflows. You can right click these smaller controllers as well to invert the position of the + and – buttons. For these three controllers, you can hold control while clicking the buttons to have Mario rotate that many angle units around the object. If multiple objects are selected, you can also click while holding alt to have all those objects rotate around the average of their lateral positions.

Then there's the object's scale, which modifies its size. You've got width for left and right size, height for up and down size, and depth for forwards and backwards size. 1 is the standard size, and the value represents a multiple of that default size. They're the same buttons for increasing and decreasing, but this time with two checkboxes. Aggregate will combine all the values into one, letting you perform operations on all 3 axes at once, keeping the aspect ratio of the object the same. Multiple will change the addition and subtraction buttons to become multiplication and division. Scaling up or down an object will also scale up or down its collision, but not its activation and tangibility radii or hitbox. Division by 0 will not cause a crash.

At the bottom is one last triplet controller, this time for an object's home. It's a standard 3D position controller, but it affects the object's home. There's nothing more to add.

5.3: Object variables

This document will now cover all of the variables that can be found on the right panel in the object tab. All of the variables with a white background are variables that can be found in all objects, whether they are used or not. Variables with a color matching that of the object's processor group color (the color in the background of the object's icon), are variables unique to that object type. Because explaining object-specific variables would require going over nearly every object in the game, this document won't be covering those. Besides, they are usually rather obvious as to what they are meant to be used for. However, one exception to the rule of variable colors is a Mario spawner's ID variable. It's shown as white, but it should be purple. It's a mistake, and it's not a default variable.

"X", "Y" and "Z" represent an object's position in 3D space. "X Speed", "Y Speed" and "Z Speed" represent an object's speed in each axis. It represents by how much their position in that axis will change by the next frame if left uninterrupted. "H speed" is the object's horizontal speed. It's the distance that's going to be moved laterally.

"Yaw facing", "pitch facing" and "roll facing" represent the angle the object is currently facing in on this frame. "Yaw moving", "pitch moving" and "roll moving" represent the angle the object is currently moving with on this frame. This is generally the same as the facing angle but can differ sometimes. "Yaw velocity", "pitch velocity" and "roll velocity" represent the speed of rotation for that object. Each

value representing by how much the object plans on turning by the next frame. Note that this value isn't the rotation acceleration, but rather the rotation speed.

"Ydist obj to Mario" and "Hdist obj to Mario" provide the vertical and lateral distances between the object and Mario. "Dist obj to Mario" is the combination of those 2, resulting in the 3D distance between the object and Mario. "Ydist home to Mario", "Hdist home to Mario" and "dist home to Mario" offer the same concept but for the object's home instead of its position.

"Angle obj to Mario" is the angle the object would need to be facing to be looking directly at Mario. "Dangle obj to Mario" is the object's yaw angle minus the Angle obj to Mario variable, clamped to the range of -32768 to 32767. "Angle Mario to obj" is the angle Mario would need to have to be looking directly at the selected object. "Dangle Mario to obj" is Mario's yaw angle minus the Angle Mario to obj variable, clamped to -32768 to 32767.

"Mario hitbox away" gives the closest distance laterally between Mario's hitbox and the object's hitbox. "Mario hitbox above" gives the difference between the bottom of Mario's hitbox and the top of the object's hitbox. "Mario hitbox below" gives the difference between the bottom of the object's hitbox and the top of Mario's hitbox. "Mario hitbox overlap" indicates if Mario's hitbox is currently overlapping with that of the object.

"Tangible dist" is the distance Mario needs to be from the object for the object to be tangible. Exceeding this distance will make Mario able to walk through the object. "Draw dist" indicates the distance Mario needs to be from the object for the object to be rendered in-game.

"Native room" is used in places like HMC and BBH where the area is split in many sections that load and unload depending on where Mario is. It gives an index into a list of rooms, and it says which room it belongs to.

"Parent object" points to the object this object belongs to. If the object doesn't have or doesn't need a parent, there are multiple things it can point to. Usually it's either itself, null, or an unused object.

"Graphics" is a pointer to a 3D model in memory that contains the graphical model for that object. "Model" is similar, but this points towards the collision model rather than the visual model.

"Visible" indicates whether the object can be seen. "Active" indicates whether the object should be executing its behavior code. "Face camera" indicates whether the object should act as a billboard and always face towards the camera.

"Animation" is a pointer to animation data. "Animation frame" is the number of frames into the current animation the object is in. "Animation timer" is the total number of frames that the object has been animating for. "Animation speed" is the speed of the animation. This is rarely used. "Graphics timer" is a timer used for graphics. It's usually used by effects and particles.

"Scale width", "scale height" and "scale depth" are floats representing the scale of an object. As you saw, they can be manipulated in the left panel, but here they are as variables.

"Home X", "Home Y" and "Home Z" represent the position of the object's home. "Release status" is a pointer to the code that gets executed when the object is released by Mario. "Interaction status" is a bitfield containing all information pertaining to how the object is interacting with Mario and the world.

"Subtype" indicates which subtype of that object it is. As an example, butterflies can either be ones with bombs or without. "Action" represents an index into the list of possible actions that object can currently have. Specifically, the one currently being acted upon.

Finally, "timer" acts as a general purpose timer that increments once per frame. Objects can use them however they want, or not at all.

5.4: Hidden variables

The object panel has the most hidden variables of any panel by far. The categories are "advanced", "Processgroup", "flags", "collision", "movement", "transformation", "coordinate", "floorcoordinate" and "RNG".

For advanced, "IG dyaw" is the delta yaw calculated with in-game logic. "Graphics X", Y and Z are the position of the object's graphical model. "Graphics yaw", pitch, roll are the rotation components of the object's graphical model. "Camera view X", Y and Z are the object's position relative to the center of the screen in pixels. Z is the object's distance from the camera. Next are four distances from the object to Mario, ""dist to Mario"" is a fake evil distance to Mario that's only updated for some object but not all.

Then the X and Z distances from the object's home to Mario, many distances from the object to its home. ""Angle to Mario"" is only updated by some objects, but not all. "Dangle Mario to obj mod 512" is the delta angle from Mario to the object, but modulo 512. "Pitch Mario to obj" and "dpitch Mario to obj" are the pitches and delta pitches from Mario to the object, then more angles from the object to its home, then the angle from the home to the object.

"Hitbox pointer" is a pointer to the object's hitbox data. Below it is the height, radius and effective radius of said hitbox. Effective radius means the radius but with Mario's collision radius added onto it. After that is the hurtbox height, radius and effective radius, then finally the downwards offset of the hitbox. Next is distances between the object and Mario's hurtbox, including a boolean for if they're overlapping, and the angle distance from Mario's punch checking point. "Wall radius" is the radius used for wall checking, "floor height" is used by some objects for the distance of the floor below it, "gravity" is the object's gravity, "water buoyancy" is the object's buoyancy when in water, "bounce coefficient" is what gets multiplied with the vertical speed for the new one after a bounce, "friction" is the object's friction when rubbing against the ground, "drag strength" is another friction variable, and "damage" is how much damage the object gives out.

"Health" is a health value used by some objects, "opacity" is also used by some, "talking state" is the state of talking objects, "active during timestop" is if the object should still be active when time is stopped, "platform object" is the object this object is standing on, "process group" is the process group ID, "process group desc" is the text description of the ID, "prev memory obj" is the slot number of the previous object in memory, "next memory obj" is the next one. "Prev processed obj" and "next processed obj" are the previous and next objects to be processed, with its own group name show if it's the first or last.

"Behavior script" is a pointer to the behavior script. "Shadow radius", "shadow opacity" and "shadow type" are the aspects of the shadow, explained further in the gfx tab chapter. "Visual pos updates" and "visual angle updates" toggle the object's visual model updating, "forward movement" forces

some objects to move forward, "Mario close" is a flag used by some for Mario being close, "tangible" is 0 for tangible and -1 for no, "use relative pos" puts the object at the origin, "relative X", Y, Z are used with the previous variable, "collision type" is a bitfield for the collision type, "num collided objects" is a number from 0 to 4, and the four variables after it are the object slots of each of those collisions. "Movement flags" is a bitfield for movement information.

"Don't loop animation" forces the animation to not loop, "reverse animation" reverses the object's animation, and "freeze animation" freezes its animation. "Animation max" is one above the maximum number of the animation frame, "animation frame 2" is a copy of animation frame, "initial release status" is the code for when the object is released, "stack index" is the index for the object's personal stack, "active flags" is another bitfield, "action 2" is a copy of the action variable and finally "sub action" is the ID of the object's sub-action.

For processgroup, these are all points to the start and end objects of each process group, which are the colors used for the object slot backgrounds and boxes. Changing any of these is pretty much guaranteed to crash the game. "Vacant next" points to the first vacant slot.

The flags section covers all the object flags objects can use to specify exception on how they act. All of the ones with a specific name are used in the game. These can be found in decomp as oFlags.

The collision section is for the collision flags of the objects. These show all flags objects can use to specify how or with what they interacted with. "Collision type" is simply the total value of this bitfield. They can be found in decomp as oInteractType.

The movement section is even more flags, this time having to do with how or where the object is moving. "Movement flags" is the total value of this bitfield. They can be found in decomp as oMoveFlags.

The transformation section displays all 16 floats found in the object’s transformation matrix. Because this matrix is recomputed every frame before rendering, modifying it doesn't do anything.

The coordinate section is for objects with collision triangles, and only displays values when the collision is active. You get the smallest and largest values found in the vertices for all 3 axes. X, Y and Z range are the maximum value minus the minimum value of that axis. X, Y and Z midpoint are the value half way in between the minimum and maximum. "Farthest dist" finds the vertex that's the furthest from the object’s position and returns its distance from the object’s position.

The floorcoordinate section does the same thing as the previous one but only considers the floors of the object. It also only displays the minimums, maximums and nothing else.

Finally, in the RNG section, you have 3 variables. The first one points to the "object RNG value" in memory, and the other two supposedly give the index of that RNG value and that index’s difference from the current index. However, that’s a lie because there’s no such thing as an object RNG value. The variable doesn’t exist, and STROOP is reading the object’s moving yaw. So, ignore these.

Chapter 6: Mario tab

Next is the Mario tab. This tab contains all of Mario's variables. This is not to be confused with the Mario object found in the object list. The Mario object is simply the object being puppeted by the Mario logic, which is what this is for. If you want to control Mario, go to this tab and not the Mario object. This tab is similar to the object tab, but it has some slight differences.

6.1: Left panel

The left panel contains many similarities to the object tab, but also some new stuff. At the top, you simply have an icon of Mario with no additional information, then two big buttons below. "Toggle visibility" toggles Mario's rendering. This helps with getting screenshots of the level if Mario is in the way. "Toggle handsfree" makes it so that if Mario is holding an object, he will now be holding it handsfree. If you've seen Pannen's videos on handsfree, then you know how that works.

The position controller is the same as in the object tab, but the controller below that is different. Instead of 3 angle controllers, you have a yaw controller and 2 other controllers for horizontal speed and vertical speed.

Below that are four more controllers for sliding speed in the X axis, the Z axis and just horizontally in general. But the last one is marked as another yaw controller. The difference between this one and the last one is that the previous one modified Mario's facing yaw, whilst this one modifies Mario's moving yaw.

Finally, there's controls relating to Mario's HOLP. You can click "goto HOLP" to set Mario's position to exactly where the HOLP is and click on "retrieve HOLP" to have the HOLP be set to exactly where Mario is. You can right click both of these buttons to go to or retrieve the HOLP in not all axes. Laterally will only use the HOLP X and Z, and you can also go to or retrieve an individual HOLP axis. Below that is a standard 3D controller for the HOLP. Enabling relative mode on it will use Mario's facing angle.

6.2: Variables

Mario's variables share some with that of an object, so I'll only go over the variables not found in all objects.

"De facto speed" is something you'd remember if you've seen Pannen's rolling rocks video. It's Mario's speed after taking into account the steepness of the floor he's on. So this value is more accurate to how fast Mario is moving than his H Speed. "Sideways speed" is the speed at which Mario is moving sideways relative to the direction he's facing in.

X and Z sliding speed are used for calculating speed in each axis when sliding, and there's a very nice Pannen video explaining the concept. Sliding yaw is the same, but as a short for working with Mario's yaw when he's sliding.

"Yaw intended" is the direction Mario is aiming for. It's where he ideally wants to move towards. "Dyaw intend-face" is Mario's intended angle subtracted by Mario's facing angle. "Dyaw intend-back" is the previous variable but rotated 32768 angle units.

"Twirl yaw" is Mario's visual angle when he's twirling. "Floor yaw" is the yaw direction in which the floor triangle Mario is currently on is pointing towards. This value is 0 if the floor has no slope. "Yaw velocity" is Mario's turning speed. This is only used when twirling, flying, swimming or spinning bowser.

Pitch and roll are Mario's facing pitch and roll. They each have their own velocity and are only used for Mario model rotations happening when airborne. Even then, the velocity variables are only used when flying.

"Flying energy" gives a general indication on how maneuverable Mario can be when flying. Specifically, it uses the formula "$Hspeed^2 + 4/\pi * Yposition$", which calculates how high Mario can fly from his current state, barring some other constraints. In general, low energy means little possible movement, high energy means being able to change direction easily.

"Stood on object" is a pointer to the object Mario is standing on, "Ridden object" is a pointer to the object Mario is riding, "Interaction object" points to the object last interacted with by Mario. "Used object" points to the object last used by Mario. "Held object" points to the object Mario is holding.

HOLP X, Y and Z are the coordinates to Mario's held object's last position. "HOLP type" changes whether you're holding a light object, a heavy object or bowser. Those 3 things have different positions relative to the HOLP for how they get released from Mario.

"Hat on head", "Hat in hand", "Wing cap", "Metal cap", "Vanish cap" and "Should have hat" are all booleans governing the status of Mario's hat. They are self explanatory for the most part. Checking the cap effect tick boxes will have the effect last with no time limit, and "Should have hat" is used to indicate that the game thinks Mario's hat is on his head. It doesn't necessarily mean that Mario's cap IS on his head, which is why this variable is should have hat and not does have hat.

"Cap timer" gives the remaining number of frames until Mario's cap turns back to normal. "Hitstun timer" gives the number of remaining frames until Mario is vulnerable again. This timer pauses when Mario is inside an enemy. "Unsquish timer" gives the number of remaining frames before Mario becomes unsquished. This starts counting down when Mario can walk and is set to -1 during the rest of the squish animation. "Squish timer" gives the number of consecutive frames Mario has been squished for. If this reaches 300, he dies. "Water shell timer" gives the number of consecutive frames Mario has been using the water shell. After 240 frames, the water shell disappears. Some of these timers are used as temporary variables for other actions, but their given name is their main purpose.

"Peak height" represents the highest height Mario achieved since he last touched the ground. "Floor height" gives the height of the point on the ground directly below Mario. "Fall height" is the peak height minus the floor height. This is obviously used for fall damage calculation. "Current room" is Mario's current room index in levels with multiple rooms such as HMC or BBH.

"Effective Mario object" and "Visual Mario object" point towards the Mario object. The difference between the two is that one is responsible for handling Mario's movement, and the other handles Mario's visuals. Under normal gameplay, these 2 variables never desync.

6.3: Hidden variables

This tab also has many hidden variable categories. There is "advanced", "holpmario", "holppoint" and "trajectory".

For advanced, "speed multiplier" is the current speed multiplier that affects Mario's speed to slow him down. "Rot disp X", Y and Z indicate how fast Mario is moving around from standing on a rotating object. "IG dyaw" is Mario's delta yaw calculated the way it would be in-game. "Twirl yaw mod 2048" is Mario's twirl yaw mod 2048. "Torso yaw", pitch and roll are the angles of Mario's torso relative to the lower half of his body. "First person yaw", "first person pitch", "first person yaw 2" and "first person pitch 2" all have to do with where Mario is looking at when in first person mode. The first two variables are for the camera's angle, and the second pair is the angle for Mario's head. "Quicksand height" is how far below quicksand Mario is. "Floor footstep type" is an ID for which sound effect to play when Mario walks on this floor triangle. "Just teleported" is the flag Mario uses to know if he can teleport when standing in a teleporter. Because if you just teleported, you don't want to be sent back immediately. "Blinking state" is the ID for which eye texture Mario is using. 0 for open, 1 for half closed and 2 for closed. "Blinking timer" is the timer for Mario's blinking. It's like the global timer but it pauses when Mario shouldn't blink.

For the holpmario section, you get measurements between Mario and his HOLP. The X, Y, Z, lateral and total distances. "Angle Mario to HOLP" is the angle Mario would need to face the HOLP. "Dangle Mario to HOLP" is Mario's angle minus the previous variable. "Angle HOLP to Mario" is the angle the HOLP would need to be facing Mario. HOLPs don't have an angle, so not literally.

For holppoint, these variables are an extra positionangle controller, as seen in the TAS tab, where the full explanation on these is. Everything from point pos type to point angle are copied from the TAS tab. Everything after compares the point positionangle from the TAS tab to Mario's HOLP. You once again have X, Y, Z, lateral and full distances along with angles, just like the holpmario section. Curiously, you also get two more variables which I believe are mistakenly here and should've been in the trajectory section. "Bob-omb traj frames" and "cork box traj frames" both calculate the number of frames it would take for a thrown object to go from the HOLP to the positionaangle in point.

For trajectory, you get trajectory information for all of Mario's areal maneuvers. "Traj remaining height" is the peak height of Mario's current areal maneuver minus his current height. "Traj peak height" is the expected maximum height achieved by the current areal maneuver. "DJ V speed" is the vertical speed Mario would get if he were to double jump with his current speed. "DJ height" is the total height Mario would gain if he were to double jump with his current speed. "DJ peak height" is what Mario's peak Y value would be if he double jumped with his current speed.

Chapter 7: HUD tab

The HUD tab controls the game's heads-up display. In the top left is an icon showing a full health bar, with some buttons below it allowing for some quick options.

"Full HP" fills Mario's HP, "Die" sets Mario's HP to enough needed to die, "Game over" is the same as die but also sets your life count to 0, "99 coins" sets your coin count to 99, "100 coin star" will spawn a 100 coin star without changing your coin count, "100 lives" sets your life count to 100, "Standard HUD" will set your life count to 4, your coin count to 0, your health to full and your star count to 120.

"Turn off HUD" will allow you to toggle between the HUD being visible or invisible. This is great for screenshots. You can even right click on this button to get two different ways to hide the HUD. The only difference between the two is that "disable HUD by removing function" will leave the coin display on.

Finally, the "Full HP" checkbox at the bottom is for locking your health to maximum, preventing you from taking damage.

7.1: Variables

"Life count" is your number of lives remaining in memory. "Life display" is your life count as seen in the HUD.

"HP" is displayed both in hexadecimal and decimal, and represents your exact HP value, beyond what the game shows you. Think of it this way: Mario's number of HP slices is a real number, and the game simply rounds downwards to know how many to show. Mario starts off with 8½ slices worth of HP, and dies the moment he has less than one full slice's worth left. The "HP (hex)" variables gives a better representation of this, as the first two digits can be read as Mario's number of slices shown in the game, and the last two digits are the remainder, shown in hexadecimal. "HP (dec)" is the decimal version of this number, and "HP (remainder)" and "HP (integer)" gives the decimal values that could be seen in the hex value above. A slice can be divided into 256 pieces, so the remainder value is out of 256. "HP display" is the number of slices you have left as seen in the HUD.

"HP increase counter" and "HP decrease counter" are both used when Mario takes a set amount of damage. For every frame these numbers are positive, they will decrease by 1 and Mario will either gain or lose 64 HP points, or one quarter of a slice.

"Coin count" is the number of coins you have collected in memory, "coin display" is the coin count as shown on the HUD, and "coin counter" is used for the coin total shown after you collect a star. "Red coin count" is the number of red coins you have collected.

"Star count" is the number of stars you have collected in memory, "star display" is the star count as shown on the HUD, "previous star count" is a star count that lags behind the real count a bit, waiting until you pass the save menu to update to the current total, and "stars in file" is the number of stars you have saved in your current file.

The next 4 variables are purple and relate to the in-game timer. "Time counting" determines if the timer should currently be active and counting up, "Time (number)" gives the current time in frames,

and "Time (Text)" formats that frame count to the in-game text. "PSS slide race" is for whether the timer is currently being used for PSS.

The next 6 variables are yellow and have to do with turning on or off certain portions of the HUD. You can toggle on and off the life count, the HP and camera icons, the coin count, the star count and the timer. You can also modify the level index to change what level the HUD thinks you're in.

The next 2 green variables are the raw bytes of assembly instructions injected into RAM for hiding or displaying the HUD. Change these only if you know what you're doing.

The final 4 blue variables modify the state of the HP HUD element. Its state represents its current action, like if it's moving or where it is on screen. The HP timer is the timer used to time these states correctly. The HUD X and Y are its position on the screen in 2D. The origin is in the bottom left.

Chapter 8: Camera tab

The camera tab covers the game's camera and everything it's able to do. You won't get much use out of this tab because the game fights very hard to keep control over the camera. Most things you do will be immediately undone by the next frame. If you actually want full control over the camera, check the Cam Hack tab chapter.

When working with the camera, you have to know some terminology and how the camera in SM64 works. So there's the camera, but also the focus. The focus is a point in 3D space, and the camera always looks directly at the focus, no matter where it is. It doesn't slowly move over towards the focus, it's the focus' job to be the one to move slowly from location to location.

8.1: Left panel

Whilst the focus determines where the camera must look, the camera is free to control its own position in 3D space. This first 3D controller is for moving the camera in 3D space, relative to the world or itself.

Camera spherical represents rotating the camera around the focus. Both XYZ and yaw/pitch/roll don't really work as terms for this, so the symbols used are theta, phi and R. However, you can also think of this as left/right, up/down and forwards/backwards. Think of a sphere around the focus, and you're moving a point, in this case the camera, on the surface of the sphere. And forwards/backwards increases or decreases the sphere's radius. Whilst you can also get the same results with the position controller, this is better for rotating around the focus.

"Disable FOV functions" is a button that toggles between removing and re-adding the function pointers for the routines that modify the camera's FOV. Disabling this will prevent the FOV from changing in most situations. When using Mupen with dynamic recompiler, you need to savestate after making the change and load that savestate to have the changes apply.

At the bottom, there is another pair of triplet controllers, this time for the focus point of the camera. The top one works as expected and moves the focus in 3D space, but the bottom one is the inverse of the other spherical controller for the camera. Instead of the camera rotating around a sphere with the focus as its center, this last controller is for moving the focus around a sphere with the camera as its center.

8.2: Variables

The first seven variables are flags for camera status. You have one indicating if the player is allowed to enter Mario cam, one for if Mario cam is on or off, one for if fixed cam is on or off, one for the close-up view, another for the further out view, and two more for the left and right camera views. Close and far cam can be controller from here, but left and right cam won't do anything, as the C-left and C-right do other stuff as well.

The camera X, Y and Z are its position in 3D space, the camera goal X, Y and Z are the positions the camera attempts to reach, the camera yaw and pitch indicate the camera's angle. Roll isn't shown because outside of shaking, it's never used.

The next 3 variables are for the centripetal angle, shortened to cent angle. The cent angle is a more useful representation for the camera's direction, as it's the value Mario uses to know where forwards is. Target cent angle is the forwards direction of the angle the camera is targeting, and fixed cent angle is only used in areas where you can only rotate the camera in 45-degree angles, like the vanish cap stage. It represents the cent angle that should be applied right now.

Camera focus X, Y and Z are the focus' position in 3D space. Camera goal focus X, Y and Z are the position that the focus is aiming for. The one it's moving towards.

"Focus veer dist" is the distance between the focus and Mario, showing how much it has veered off from where it's supposed to be. This is most pronounced when Mario is looking to the left or right, as the focus will move away from him to show more in the direction he's looking.

"Dist cam to focus" is the 3D distance between the camera and its focus, and "dist cam to Mario" is the 3D distance between the camera and Mario.

"Camera mode" is an ID representing the current state of the camera, and second object is for whenever the camera needs to show Mario as well as another object at the same time. Like for example king bob omb on the mountain.

"FOV" is the camera's field of view. By default it's 45 degrees, but it can change slightly on certain situations in game. "Near dist" and "Far dist" are the distances from the camera for the near clip plane and far clip plane. If you don't know what that means, it's the minimum and maximum distance something on screen has to be from the camera to be rendered. By default, that's minimum 100 units away and maximum 20000 units away.

Then there's a second copy of FOV, but this one is more important, since modifying it actually does stuff. However, the other FOV is also the only one that gets changed during zoom-in camera shakes. Below are 5 variables that are raw bytes for assembly instructions used for modifying the FOV. These are what are used with the "Disable FOV functions" button.

After that are 3 more function pointers for routines handling camera shaking.

Finally, 16 variables for shaking the camera. There are 4 shaking directions: yaw, pitch, roll and zoom. Each has 4 variables for controlling it. The first variable controls the intensity of the shaking, the speed controls the frequency of the oscillation, the decrease variable by how much the intensity gets decreased each frame. Finally, the oscillation value is where the current oscillation is in its period, mapped onto a signed short.

Chapter 9: Triangles tab

The triangle tab is for watching over and interacting with the collision triangles that make up the game. Specifically the collision triangles, and not the visual triangles. With this tab, you're only able to read from and modify things that are tangible to Mario. Everything else is outside the scope of this tab. That's an important distinction to make. If you're using the injected hitbox view code while messing with the triangles, know that the hack won't update live. If you want to refresh what's on screen, you need to re-inject the hack every time you want to update what's on screen.

In SM64, not only are triangles either walls, floors or ceilings, but there are also two types of collision triangles: those that belong to the level and those that belong to objects. Object triangles are tricky, because they only exist if Mario is close enough to the object, and they're recalculated each frame, meaning that they can't be modified in STROOP.

9.1: Left panel

To get information from a triangle, you first need a way to select the triangle. Which is obviously not as easy as selecting an object. You can select from these 6 options on how you want to select your triangle. Floor, wall and ceiling refer to having the currently selected triangle be the one Mario is currently interacting with. For walls, they will only be selected if Mario is touching that wall. For floor and ceiling, it will be the one currently above or below Mario, if there is one. "Custom" will let you input the memory address of a triangle. If this option is not selected, the memory address shown will be that of the currently selected triangle. "Map hover" will select the last triangle that you had selected in the map tab with enable object hover on, and "map accum" selects all of the triangles you have accumulated by holding shift on the map tab triangles with enable object hover on. When on map accum mode on this tab, press escape to delete the list of accumulated triangles.

You can also right click the triangle text above the radio buttons to get an option called "paste triangles", but the formatting is not what you would expect, and I wouldn't exactly call it pasting triangles. This option is for changing triangle coordinates on a large scale. The text you need to put in your computer's clipboard is the following: 10 lines of text, where the first is pointers in hexadecimal format to the triangles whose vertex coordinates you want to change. The nine following lines are for the 3 axes of all 3 coordinates of each triangle, with each triangle getting its own column. Once executed, this function will go over every column of text and set the positions of that triangle. The other option in this menu is "update based on coordinates", and all it does is go through every currently selected triangle and updates its values to what they should be.

It's also possible to select multiple triangles at once. You can either do this by putting multiple addresses in custom and separating them with a comma, or map accum. Not all of the controls or variables work when multiple triangles are selected, but most do. Once you've selected a triangle, everything else in this tab becomes available. Everything below is for actions on either the selected triangle or all triangles. You can also right click the custom text to get "paste addresses", which retrieves a list of triangle addresses from your clipboard.

Every triangle has 3 vertices. In SM64, those vertices have a specific order for every one. With the "Goto V1", V2 and V3 buttons, you can teleport Mario on top of those 3 vertices.

"Goto closest" will teleport Mario above the vertex of the currently selected triangle that's the closest to him. You can right click this button to add an offset to Mario's teleport location, either having the position be off the vertex by 0.5 units or 0.999 units. This will add the value to the axis if it's positive and subtract it if negative.

"Retrieve" will move the selected triangle up or down to match Mario's current height. If the triangle is directly below Mario, it will move until the point on the triangle directly below Mario is at his position. If Mario is not directly above the Triangle, the triangle will move until its highest vertex is a few units below Mario's current Y position.

"Neutralize" forces a floor to change its type to that of a regular, safe, non-slippery floor. By right clicking this option, you can select whether you want to neutralize the triangle by changing its type to hex 15 or 0. The difference between these two is that hex 15 is an intentionally non-slip floor that's easy to walk up on, while 0 is a default floor that takes more time for Mario to walk upwards on.

"Annihilate" deletes the selected triangle. It will cease to exist. If this is the floor below Mario, Mario will go out of bounds and die. Right clicking this option will give two more options. The first one deletes all triangles except for the death barrier, if there is one. The second deletes all ceilings. These don't delete object triangles, only the ones from the level geometry.

Checking "Vertex misalignment offset" adds half a unit whenever you teleport to a triangle vertex, similar to the menu option for go to closest.

You can move the triangle around with the position controller and also move the triangle in the direction of its normal. A triangle's normal is basically an indicator for which direction the triangle is pointing in, so you can use this to move the triangle "forwards" or "backwards". For the position controller, you can hold control when clicking the buttons to move every loaded triangle at once in that direction. This lags STROOP for a short while when you do it. Additionally, when the relative checkbox is active, you can hold shift while clicking the buttons to use Mario's angle instead of the triangle's facing direction.

Checking the "Neutralize triangle" checkbox will automatically neutralize any triangle that you select.

"Show coords" will create a pop-up window displaying the X, Y and Z values for all 3 vertices of the currently selected triangle. "Show equation" will display the equation formed by the normal values and the offset. It's explained in more detail in the next sub-chapter with the normal variables.

This next section is for recording triangle data. By checking the "record triangle data" checkbox, the game will start adding every selected triangle to a list. The number shown beside in a box is the current length of that list. To stop recording data, uncheck the checkbox. No data will be deleted from the list until you click the "clear data" button. To access this data, there are 3 buttons. The first is "show data". This will list the X, Y and Z positions for all 3 vertices of each triangle in the list. By default, the checkbox down here called "Repeat first vertex" is checked, meaning that every triangle will have 4 coordinate triplets, but the 4th ones will always be equal to the first ones. This is useful, because the first vertex is used to determine the priority of the triangle, but then again you can already see it, so this is a stupid feature, and even stupider that it's on by default. "Show vertices" will show all unique vertices used by the triangles in the list. No duplicates, no spacing. Just one long list. Finally, "Show addrs" will list the addresses of all the triangles in the list.

The type conversion section allows you to search and replace the types of all triangles of a specific type. First, a drop-down menu allows you to choose whether you want to search through all triangles, all floor, all walls or all ceilings. None of these options affect object triangles. This is for level geometry only. From there, you enter the type index for the triangle type you're searching for, and what you want to replace them with. Be warned that both of these values will be interpreted as hexadecimal, whether you add the 0x prefix or not. When you click on the convert button, STROOP will search through all triangles matching your settings and replace their type accordingly.

The 5 buttons at the bottom are for modifying all triangles at once. "Neutralize all triangles" will neutralize all triangles. You can right click this button to only neutralize certain types of triangles. Most of these options are self-explanatory, but I'll cover the ones that aren't. "Neutralize sleeping" activates the flag for all triangles that they come from an object rather than the level. This doesn't affect much, but one of the things it does affect is Mario's ability to sleep, because Mario cannot sleep on object triangles. This option will thus prevent him from sleeping. "Neutralize loading zones" will neutralize triangles that cause invisible loading zones. These invisible loading zones can be found in DDD, WDW, the TTM slide and inside the pyramid in SSL.

"Disable all cam collision" will make it so all triangles don't have collision with the camera. You can right click this option to specify only walls, floor or ceilings.

The last 3 buttons at the bottom create a pop-up window showing a table containing all the notable variables for that triangle. To better view these values, copy the contents and paste them in a spreadsheet program like excel.

"Show level tris" displays all triangles from the level geometry, and right clicking this button allows you to simply copy the values directly as well as filter for floors, wall or ceilings. "Show obj tris" displays all currently tangible object triangles. Right clicking this button allows you to copy directly and filter for walls floors and ceilings, but also select the triangles for the objects you currently have selected in the object grid. "Show all tris" displays both the level geometry triangles as well as tangible object triangles in one big list. Right clicking this button allows you to copy directly and filter for walls floors and ceilings, but doesn't have options for currently selected objects.

9.2: Variables

"Classification" indicates whether the triangle is a wall, floor or ceiling. "Surface type" is the ID in hexadecimal for the triangle type. A triangle's type can do a lot of things, like be lava, vertical or horizontal wind, a warp trigger, ice, etc. "Description" provides a text description of the triangle's type.

"Slipperiness" gives an ID for the slipperiness of that triangle type, "Slipperiness description" is the text description for that, "friction multiplier" is that triangle's friction. Mario's speed gets multiplied with this value every frame, so the closer it is to 1, the slipperier this triangle is.

Next is a bunch of flags for the triangle. The first variable is the flags represented as a number, so what's after that is what's important. "X projection" and "Z projection" are for walls. To simplify collision, walls either only have their hitbox extend in the X direction or Z direction. This is what determines that. "Belongs to object" is if the triangle belongs to an object. "No cam collision" means the triangle doesn't collide with the camera. "Room" is the room the triangle is in for levels with rooms.

"Exertion" is used by special floor triangles like flowing water and horizontal wind that push mario around when he's on them. The first exertion variable simply indicates that the triangle is capable of exerting a force onto Mario. "Exertion angle" is the angle the triangle pushes you in, but represented as a single byte instead of 2. To convert an in-game angle to an exertion angle, simply divide it by 256. Multiply an exertion angle by 256 to get the real angle back.

"Y min – 5" is the smallest of the 3 vertex Y values, minus 5. "Y max + 5" is the largest of the 3 vertex Y values, plus 5.

Next are the X, Y and Z positions for all 3 vertices of the triangle. Remember that triangle vertices can only be on integer coordinates, and not in between.

After that is a copy of Mario's position for reference, the index of the closest vertex of the selected triangle to Mario, and its position.

"Steepness", gives the angle of the floor relative to pointing upwards. This ranges from 0 for completely flat to 16280 for the steepest possible floor. "Uphill angle" is the yaw angle Mario would need to have to be going directly uphill, same thing for "downhill angle", "left angle" and "right angle", but for those directions, relative to the triangle's orientation.

Then there's more copied Mario variables, this time the intended yaw and the facing yaw. "Uphill dangle" is Mario's facing direction minus the triangle's uphill angle, clamped to the range -32768 to 32767. Every other delta angle variable is the same but for the triangle's downhill angle, left hill and and right hill angle. "Hill status" is whether Mario is facing uphill or downhill. "Wall kick angle away" represents whether you currently have a valid wall kicking angle with a wall. If this number is less than 0, Mario is able to wallkick on the selected wall. "Wall kick post angle" is the angle Mario will have if he wallkicks on the selected wall at his current angle.

Then there's the triangle "normal" variables. In theory, the sum of the squares of the X, Y and Z components of the normal should add up to 1, and this vector would point in the direction the triangle is pointing. "Normal offset" is different to the other three. To help visualize it, imagine if the flat plane created by the triangle extended out infinitely, using the angle of the triangle's surface. The three normal variables will give the orientation of this plane, but not its position. If you were to interpret the normal values in an equation, this plane would always intersect the world's origin. To have the plane actually intersect the triangle, you need to add a constant to its position, and that's what the normal offset is. It's the height of this plane created by the triangle above the origin relative to the position of the origin. For example, a flat floor 100 units above Y0 would have a normal offset of 100. By the way, the equation to create this theoretical plane is what you see when you click the "Show equation" button on the left panel.

"Associated object" gives the slot number of the object Mario is interacting with, if there is one.

"Floor triangle", "wall triangle" and "ceiling triangle" are pointers to the floor below Mario, the wall Mario is touching (if any) and the ceiling Mario is under (if any). This floor pointer right here, it being null is the definition of out of bounds. After is the floor height copied from the Mario tab and ceiling height, the height of the ceiling above Mario. If there is no ceiling above Mario, it uses the default ceiling that's 20000 units high. "Dist above floor" is Mario's distance above the floor below him, and "dist below ceiling" is the distance below the ceiling above him.

"Normal dist away" is Mario's distance from the triangle's theoretical infinite plane mentioned earlier with the normal variables. "Vertical dist away" is Mario height above that same theoretical plane. "Height on triangle" is the difference between Mario's height and the lowest point of the triangle.

The rest of the variables are part of the "advanced" variable category, and even though they're visible by default, they're so complex and plentiful that I'll consider them hidden variables. In fact, I recommend disabling the advanced variables in the generic right click menu to have this tab lag less.

9.3: Hidden variables

Other than "advanced", hidden variable categories include "self" and "extendedlevelboundaries" for this tab.

For advanced, there's "syncing speed" which is the same as in the PU tab. "Max hspd uphill" is the maximum speed you could have while facing uphill on this triangle to not hit OOB. "Max hspd uphill at ang" is the same but takes Mario's yaw angle into account. "Max hspd downhill" and "max hspd downhill at ang" is the same but for how fast you can go while snapping down to the floor. These only work for non-flat floors.

Most of the grey variables are better understood with the context of the cells tabs. "Cells" and "Mario cell" are taken from the cell tab. "Tri list pointer" is a pointer to the start of the triangle pool. "Tri list capacity" is the maximum size of this pool. "Level tri count" is the number of geometry triangles in the level. "Object tri count" is the number of currently active object collision triangles. "Total tri count" is these two numbers added up. "Current tri index" is the index of the currently selected triangle in the triangle pool. "Current tri obj index" is the same but for object collision triangles. "Current tri address" is the current triangle address. "Node list pointer" is a pointer to the list of triangle nodes. "Level node count" is the number of triangle nodes in the level. "Object node count" is the same for objects. "Total node count" is the last two variables added together.

"X min", "X max", "Y min", "Y max", "Z min" and "Z max" are the biggest and smallest X, Y and Z values of all 3 vertices of the selected triangle. "X range", "Y range" and "Z range" are the max value of that axis minus the min value. "X midpoint", "Y midpoint" and "Z midpoint" are the middle X, Y and Z values between the max and min value of each axis.

The next 15 variables provide the X, Y, Z, lateral and total distance between Mario and each of the three vertices of the selected triangle. "Dist to line12", "dist to line23" and "dist to line 31" are Mario's distance to each of the edges of the triangles. Each edge being labeled with the IDs of its two vertices concatenated.

The next 9 variables are the angle from Mario to each vertex, the delta angle from Mario to each vertex and the angle from the vertex to angle. These work just like the angles and delta angles found in the object tab.

Next is the angles from each vertex to every other vertex. After that, the delta angle of each edge between both its vertices. Every edge is represented twice, for both forwards and backwards.

For self, "self normal dist away" is the distance between the self positionangle from the TAS tab and the plane containing the selected triangle. "Self vertical dist away" is the same but only vertical distance. "Self height on triangle" is the Y position of the point on the triangle directly below the self

positionangle from the TAS tab. "Self dist to line12", "self dist to line 23" and "self dist to line 31" are exactly the same as the previous distances to edges, but these use the self positionangle from the TAS tab instead of Mario.

For extendedlevelboundaries, all that is provided are the coordinates of each vertex along with the normal offset assuming that a ROM with extended level boundaries enabled is being played. When extended level boundaries is not on, these values are the same as the real ones.

Chapter 10: File tab

If you want to hack yourself a completed save file in STROOP, the fastest way to do so is by right clicking the version number and selecting "everything in file". But what if you want something more precise? The file tab gives you complete control over every single thing that the game saves to your save file for later.

10.1: Left panel

This is the first tab to have a non-standard left panel. It's much wider to account for the completion grid. However, before anything, you have to select a save file to work on. You can choose any of the 4 save files, as well as whether you want to save their current state, or the values actually saved outside of the game's memory. Updating your current state is what you want to do to affect the game live, but if you don't save, the changes won't apply to your save file once you reset. And if you only make changes to your saved file, nothing will change in the game until you leave the file without saving. To make things less confusing, I'll refer to the left options as the working file and the right options as the saved file. Note that saving to the saved file still saves to game RAM, and not to your emulator's EEPROM file, so resetting the game after setting these will still erase the data if you don't save in-game.

Along with selecting a file, you have buttons to the side allowing you to save, copy and erase files outside of the main menu. "Save" will take the current data of the save file pair you selected and copy it over the saved file. In other words, it moves the data from file X to file X saved, where X is the currently selected file, no matter whether you picked its working file or saved file.

"Copy" will copy the state of your currently selected option, and "paste" will paste it to the currently selected option. Unlike the save button, these buttons consider the working file and saved file to be separate, so you can copy and paste anything to anywhere.

"Erase" will delete everything for the currently selected file. Both the working file and the saved file.

"In-game copy and paste" will make copying and pasting work differently. With this option checked, copying will always select the working file of the pair, even if the saved file is selected, and pasting will paste the contents to both the working and saved file of the file you selected.

After this, you can finally start modifying the save files themselves. The top section here is for controlling the location of Mario's hat. You can select one of the images below to set its location. It can be on Mario's head, in Klepto's claws, on the ground in SSL, on a snowman's head, on the ground in SL, on Ukiki's head or on the ground in TTM. If the hat is set to be on the ground somewhere, you can set its position with an X, Y and Z.

Next up are all the miscellaneous flags for unique stuff in the game. Here, you can toggle whether or not the save file is marked as actually existing, the state of the three cap switches, if either of the castle key doors are unlocked, if the moat is filled or empty and the position of the DDD painting relative to fire sea. Also, the key door buttons have 3 states, because you can also have where Mario has the key but hasn't opened the door yet.

The next buttons are for toggling on or off a bunch of stuff at the same time. You can have all stars, have no stars, open all cannons, close all cannons, open all doors, close all doors, get all the coin

high scores or erase the coin high scores. Then you have the 2 ultimate buttons, everything and nothing. "Everything" does the same thing as "everything in file" from the version number menu, and "nothing" is the same as erasing the file.

At the bottom is a selection menu for defining what "all coins" actually means. It can either be 100 coins, 255 coins, the maximum achievable in game without glitches, or the maximum obtainable in game but with glitches. This will only affect what the "all coins" button does.

At the bottom right is a button labeled "update HUD to X stars", with X being the current number of stars in the selected save file. Modifying the stars in this tab doesn't affect the HUD, so this button allows you to update it manually. Note that nothing in the game actually checks the HUD value for the star count, so this is only a visual discrepancy.

Now we can finally address the giant table to the right of the buttons. The leftmost column is an abbreviation representing every single stage, secret stage and star source in the game. The next 7 columns store up to 7 stars each stage or source can have, and then a column for a possible cannon, a possible locked star door before the stage, and finally the coin score for main stages. The star order is based on the order in-game, with the 100 coin star interpreted as the stage's 7th star. Hope you remembered your star numbers, in general you should familiarize yourself with them! The only non-obvious order is for PSS, so I'll just note that here, the second star represents the under 21 seconds star.

10.2: Variables

In this tab, the variables provide very little additional control and are mostly just a more complex way of displaying the information found on the left. However, some stuff is only found here.

"Current file" is the currently loaded file in the game. This is 1 indexed, so A is 1, B is 2 and so on. "Current star count" is the count in memory, "Star display" is the star count on the HUD, "previous star count" is the star count that only updates after you pass through a save prompt, whether you end up saving or not. "Stars in file" is your real star count.

The next 15 variables seem to be a star count for each stage, but their name and presentation are misleading. They're actually bitflags represented in decimal. The lowest 7 bits of the variable represent the 7 stars in the stage. They're ordered in such that the least significant bit is the first star, and so forth to the left.

After that, are things we've seen before. Bools for all the secret stars and cannons, then the coin scores, the states of the cap switches and the states of the bowser keys and locked doors.

The section for the hat is interesting because we get a bit more control. Here of course we can specifically say if the hat is possessed by someone, but also where it is in the world if it isn't. You can set the exact level, area and position for the hat, even outside of the levels it's supposed to be in.

After more things we've seen before are the variables for the checksum. SM64 does a checksum on a file to be able to verify later that it is valid and uncorrupted. Every change to the file you make through STROOP will change the checksum accordingly to validate the file, but you can mess with it here. "Checksum constant" is the constant used by the game for the checksum formula. Changing it is an

easy way to corrupt save file. "Checksum" is the current checksum saved on the file, and "checksum calculated" is what your checksum would be if it’s up to date.

Finally, there’s 105 flags for every main stage star in the game, this time with the star name for those who just can’t bring themselves to remember numbers for a game they spend so much time on.

10.3: Hidden variables

The only hidden variable category here is "advanced", and it contains the bitflags for all of the castle’s secret stars. "Castle stars" is a bit field for the toad and MIPS stars, and every other stage has a single bit for if their star is collected or not, except for PSS, which has 2 bits. "Cake stars" is for the number of stars you have collected in the ending screen of the game with an image of a cake. That picture is technically a level in the game’s code. (But obviously it was never meant to have stars.) Setting all of the save file bitfields in memory should give a star count of 182. You shouldn't set any of the values past their max values, as that can corrupt other bits in the file data and actually remove stars.

Chapter 11: M64 tab

This is the tab you're most likely to visit if you bypassed STROOP's connection menu. This tab has nothing to do with your game and is instead a complete M64 file editor. For the uninitiated, M64 files are the TAS files used by Mupen64. They're a sequence of inputs with a header containing information about the TAS. In this tab, you can quickly modify the file without needing SM64 or even Mupen open.

To open an M64 file and use this tab, click the "open" button in the top left and select a .M64 file from the file explorer. From there, the file's name will appear above with a [MODIFIED] tag if it's been modified, a counter indicating the number of inputs you have currently, the number of inputs in the original file and the difference between the two.

To close this file without saving you click "close" or open a new file, click "reset changes" if you've messed up really badly, as this will bring the file back to how it was when you last saved it. Finally, you can save the file or save a new file with these last two buttons.

11.1: Piano roll

Unlike the previous tabs, the section on the right is the less complicated and the more useful one. When you load a TAS, it will be displayed here in a piano roll format. You can scroll through every frame of the TAS, modifying the inputs for each frame. When an input is different from the value in the saved file, it will be given a red background.

The "frame" column indicates the index of that frame in the current TAS, and the "ID" column indicates its index in the order of when the frames were added. Meaning that new added frames will be given new indices, rather than ones that fit to keep the order.

X and Y represent the control stick. The values are stored as a signed byte, meaning that they can range from -128 to 127. For X, negative is left and positive is right, and for Y, negative is up and positive is down.

After that are all the buttons, grouped by type then ordered from most important for SM64 to least important. So A, B, Z, Start, R, The C buttons up, down, left and right, then finally L, D-pad up, down left and right. The C buttons have a yellow background, and L and the D-pad buttons have grey buttons because they don't do anything in SM64 under normal circumstances.

To check or uncheck a button for a specific frame, you click on the checkbox. If you click outside the checkbox, you will select the input, but not change it. This will be important for later. You can select ranges of inputs with shift and multiple with control. You can also copy the buttons and values with control+C, and paste them in a separate program as a table of values.

You may have noticed by now, but the piano roll section also has three tabs at the top. What I've been showing is the "inputs" tab, but you can also switch tabs to either modify the header or view statistics about the TAS.

In the header tab, you can find general information about the TAS stored in the header. To not corrupt a TAS, make sure you know what you're doing. Some of these values need to not be touched for the TAS to remain playable.

"Num inputs" is the length of the TAS, by the number of input frames it has. "Num VIs" is the number of vertical interrupts in the TAS. This is the number of times during the TAS that the game will send data out of the N64 to be displayed on screen. In general this should be around twice as large as the number of inputs for 30fps games like SM64, but it can be larger to account for lag. There's nothing wrong with having this number be larger than it should be, but it being too small can cause problems, which is why some programs like STROOP will set the VI count to the largest possible number to not have to think about it.

"Num rerecords" is the number of times a savestate was loaded during the making of the TAS. It being easily editable right here is the best example for why you shouldn't trust or care about re-record counts.

"Movie start type" tells Mupen how to start playing back the TAS. You can see all the options by clicking the drop-down arrow all the way on the right or by double clicking to go to the next option. "From start" will have the movie play from the very start of the game on a fresh save file. "From savestate" will have Mupen attempt loading a savestate with the same name of the M64 in the same directory as it and start the TAS immediately after loading the savestate. "From EEPROM" is similar to from start, but this will save all save files within the M64 so that instead of starting from nothing, you keep the data stored to the cartridge.

"FPS" should always be 60, unless the TAS is on a European ROM, then it should be 50. "ROM name" is the internal name of the ROM file the TAS was made on, "county code" is an ID representing the region code, and "CRC32" is the checksum for the ROM file.

"Author" is the name of the author of the TAS, and "description" is the description of the TAS. Feel free to edit these two values as much as you want.

"Num controllers" is the number of controllers "plugged into" the emulator when the TAS was made. Then there are 12 true or false values indicating whether each of the 4 controllers was plugged in, had a memory pak or had a rumble pak.

The "plugin" section contains the names of the video, sound, input and RSP plugins used when creating the TAS. "Signature" is the hashed signature of the Mupen executable used to make the TAS, followed by an ID representing the version number and a UID as well for good measure. That's all for the header.

The stats page contains STROOP-made statistics about the TAS. These cannot be edited, as they aren't part of the file. First is a press count for every single button on the N64 controller. These count individual presses, not the number of held frames.

"Lag VIs" shows the number of extra vertical interrupts added in due to lag, "num unused inputs" is always 0, "num joystick frames" gives the number of frames that the control stick was not neutral, and "num input changes" is the number of times the inputs changed from one frame to another.

11.2: Left panel

Now that the most basic form of M64 editing has been covered, we'll move back over to the left panel for the more advanced controls. The drop-down menu on the top right lets you choose between "frame of input", "frame after input" and "frame observed". These modify the starting number for the

frame indices, and it helps with changing the frame of reference for what the number means. Whether it be the frame the input is made on, the frame it's registered or when the input modifies the game state. This will set the starting index to either -1, 0 or 1.

The "max out VI count" checkbox will set the VI count to the 32-bit integer limit when saving the file. This doesn't affect playback, but it does prevent some possible errors, so it's on by default. However, you don't need it to be on for the vast majority of things.

The "go to" button takes the number written in the textbox on the left and navigates the piano roll to that frame number. Not ID, frame number. The four buttons below are for changing which ROM the file expects. You can set the TAS to be either a US version TAS or a JP version TAS, but you can also copy and paste specific ROM IDs and names if you copy, load a new file then paste.

To the left of that are 3 textboxes. Normally, what they do is provide quick information about which rows and columns you have selected. "Start frame" gives the smallest frame number row selected, "end frame" gives the largest frame number row selected, and "inputs" lists the columns selected by which inputs they correspond to. The number of rows selected will be displayed in parentheses in the text above. Once your range is selected, you can do things with the buttons below.

"Turn off row range" will remove all inputs for the entire row of the range entered above, regardless of what's in the inputs section. It always does the entire row. For the control stick, it will set it to neutral. "Turn off input range" does the same thing, but this time while only considering the inputs listed in the inputs section. "Turn on input range" does the opposite and turns on all the inputs there. However, this will not do anything with the control stick, unless a value is put in over here in the "on value" textbox. The number put in here will determine what on means for the joystick. "Delete row range" deletes all rows selected above, regardless of the listed inputs. This removes the entire rows all at once.

"Turn off cells" and "turn on cells" don't use the text listed above as reference, but rather what's actually selected on the piano roll, and only that. The turning on and off acts the same as the other on and off buttons.

This list down here is a local clipboard. Every time you copy the inputs using one of the copy buttons, that entry will be added to this list, so you can select which copied entry you want to paste later with the paste buttons. By default, the clipboard is given a single default entry of one empty frame. This is indeed one empty frame. To delete entries from the clipboard, select one and press the delete, escape or backspace key on your keyboard.

There are three copy buttons. Two of them are just above, "copy row range" and "copy input range". They select their inputs just like both turn off options. The entry will be listed in the clipboard as the number of frames selected, the inputs selected, which rows specifically, and then the name of the file. The last one is up here to the right of the other buttons, labeled "copy clipboard for joystick" reads your clipboard to look for a list of comma separated values for the joystick, interleaved X and Y in that order, and adds those values as a set of inputs to paste later. When this button is right clicked, you can choose to have it interpret every value as X or every value as Y, in which case the data does not need to be interleaved.

To paste inputs, select which set of inputs you want to paste, and select in the piano roll where you want to insert. You then have two insertion options. "Paste insert" or "paste overwrite". Insert will move the selected rows down and insert the selection above into the newly created frames. Newly added frames are marked with a green background, and similar to the red background for new inputs, they disappear when the file is saved. The second paste option is overwrite, which will overwrite the values at and below where you selected. This will only overwrite what's necessary, so if your selection wasn't an entire row, the rest of the row will be kept intact. The "multiplicity" textbox below the paste buttons acts as a multiplier for the pasting process. It takes in a positive integer, and multiplies you selected entry by that number. If the given number is invalid or not a positive integer, pasting won't work. This is very useful when you want to paste a sequence that's repeated for a long time.

"Quick duplication" allows you to quickly duplicate a section of the file as many times as you want. It will take everything from frame number "1st iter start" to just before "2nd iter start" and copy it "total iters" times after the end of the copied range. Except not exactly, because the code stupidly removes 1 from all 3 of these indices, so what it ACTUALLY does it take everything from the first number minus 1 to the second number minus two and copy it the third number minus 1 times. This tool inserts frames and doesn't overwrite.

"Add pause buffer frames" will add 3 new rows after every single selected row in the piano roll, as well as press start on all of those selected rows. The added rows will always follow the format of blank, pause, blank, to perform the pause buffering. This makes creating pause buffering easy, as it can be added over regular gameplay.

Chapter 12: Actions tab

The action tab is for viewing and analyzing Mario's current action and animation. At the top of the tab, Mario's current action is written in giant black letters, and his current animation is written in giant blue letters.

The "action" variable is not a pointer but rather a bitfield. It's the hexadecimal representation of a 9-bit ID combined with the bits seen on the right with a white background. They're labeled 9 through 31 and thus represent the bits 9 through 31 of the action variable. Because there's no control over the lower 9 bits, modifying any of these on their own will likely create an invalid action. "Action description" is the same as the giant black text above. "Previous action" is what Mario's action before the current one was. "Prev action desc" is the text description of the previous action. "Action phase" is the phase of the action, when an action takes multiple animations. "Action timer" is a general-purpose timer each action can use as they please, and thus it represents different things depending on the action. "Action parameter" is a general-purpose number used by the actions. It's not used very often. "Wallkick timer" gets set to 5 when you hit a wall and counts down to 0. It represents the number of frames you have to wallkick, and on which frame you did. Don't believe the propaganda, there are 5 frames to wallkick, not 4. "Landing timer" gets set to 5 when you land on the ground and counts down to 0. It represents the number of frames you have to upgrade your jump to a double jump or your double jump to a triple jump. "A press timer" counts the number of frames since you have last pressed the A button, capped at 255. "B press timer" does the same thing with the B button. "Animation" is the animation ID for your current animation. "Animation description" is the text representation of that ID. "Animation timer" is the number of frames since the animation has started. "Action ID" is the ID of your action that was mentioned earlier. Its values will be the 9 lowest bits of the action variable. Furthermore, "action group" is the action ID with the 6 lowest bits masked to 0, leaving bits 6 to 8 representing the group of the action, like whether it's airborne, moving, etc. "Action group desc" is the text description of that action group. After this it's only the bitfield left, but I've already explained how it's just the highest 23 bits of the action variable. However, here you get to see roughly what they represent about the action.

You can right click the action name at the top of the tab to get some cool options. "Select action" opens a window with a list of all actions, and selecting one will put Mario in that action. "Free movement action" will put Mario in the free movement action, just like in the version number right click menu. "Open action form" will open a new window with a table giving information about every single action in the game. In this form, every action in the game has its own row, and the columns are the info for that action. Name is the action name, value is the full bitfield of that action, group is the ID of the action group it belongs to, group name is the name of the group it's in, ID is the ID of that action, and after that is the values given to the bitfield that describe the action's capabilities. For the animation name in blue text just under the action name, you can also right click it for more. "Select animation" opens a window listing every animation in the game, and you can select one to give it to Mario. Animations that go unused in the game are prefixed with a ~. "Replace animation" lets you replace an animation with another in the game's code. The first window will make you choose which animation to replace, and once selected another window will ask you which animation to replace it with. Once done, if you try performing that animation in the game, Mario will instead awkwardly perform the animation it was replaced with.

Chapter 13: Input tab

Upon entering the tab, you're greeted with a giant N64 controller. This controller will update live with the inputs you're pressing, but only if the game is running. It will freeze in place if the emulator itself is paused, this is simply the controller from the SM64's perspective.

The giant controller's stick will move around to emulate the movement of the stick, even if it means going into the corners where it would be impossible to reach on a regular N64 controller. Every other button gets highlighted in bright red to indicate it's being pressed. Z, being on the back side of the controller, has its indicator off to the left side of the controller next to the stick. If a giant controller doesn't fancy you, you can right click it to select from 2 other input display methods. One being a sleek input layout, and the third one being a vertical version of the same thing. This right click menu also allows you to record the input window, however this feature doesn't work, and saving the file will fail and raise an error.

On the right, you have a bunch of variables for the state of the controller. The first 4 are special for SM64, because it shows how it processes the raw stick input. What it first does is scale the inputs down using trigonometry so that the magnitude of the stick never exceeds 64. This means that in SM64, moving the stick halfway in a direction and pushing it all the way will be registered as the same thing by the game. "Effective X" and "effective Y" are the X and Y results of the stick after this operation, and to double check, "magnitude" is shown below, and it indeed never exceeds 64. "Scaled magnitude" takes the magnitude and performs some non-linear operation on it, making movements near the edge of the 64-magnitude limit count as more than movements near the center. This also caps at 32 instead of 64.

After that are the raw values for the X and Y of the stick, as signed bytes ranging from -128 to 127. What follows next are 3 copies for every other button on the controller. One for if it's currently being held down that frame, one for if it wasn't pressed last frame but now is, indicating that it was just pressed down, and finally one for "buffered" inputs. Buffered here means that it was held since before the controller was polled. This doesn't work well when frame advancing with the emulator, so it's hard to analyze.

The last few things to note are that the raw stick values don't have a "Just" state, only a current and buffered state. Also, there are two mysterious variables for buttons called U1 and U2. Sadly, these aren't secret hidden buttons Nintendo is hiding from you. When the controller sends the input data to the N64, it's packaged in 4 bytes per controller. The first 2 are for the stick's X and Y, so this leaves 16 bits for the remaining buttons. But there are only 14 remaining buttons, so 2 of those bits go unused. U1 and U2 simply stand for unused 1 and unused 2. Activating these buttons still shows them on the controller graphic as non-existent buttons above the start buttons, but only with the classic input display.

Chapter 14: Water tab

Water is special in SM64. It's not a type of floor or a special kind of barrier, it's its own thing. Technically, water exists everywhere in SM64. It's just that whenever it's not used, it's given a default Y value of -11000. And also, water is invisible. The water surface you see in game are intangible graphical triangles given a moving water texture and transparency. "Water" really is just a set of zones defined in the XZ plane given a specific Y value to say at what height the water is. This is why waterfalls in SM64 aren't water, and you can't swim in them. The zones that define water can only be flat.

The water tab doesn't just control the water, but many other things as well, such as toxic gas or fog. This tab should technically be called the "environment regions" tab, but this mechanic is mainly used by water, so it still makes sense to call it the water tab and I'm going to refer to this as water specifically even if it can be other stuff.

"Water level" is the current water level below Mario, "water level median" is only used in WDW, and indicates what the water level should be fluctuating around, "water level above median" gives the difference between the water level and the median.

"Mario above water" gives the difference between Mario's Y position and the water, and "Mario Y" is a copy of Mario's Y position for reference.

"WDW main WL" and "WDW town WL" are the current water levels for both halves of WDW. "WDW painting level" is the last Y position Mario had when entering the WDW painting. WDW is special, it's the only level in the game to have a slightly fluctuating water level and also a water level that moves around. (Other than inside the ship in JRB and the water in the castle's basement.)

The next couple dozen variables are every single non-default water level in the game. These are what you should modify if you want to change the water level. In here you have all of the water levels of the game you would expect, but also some for other environments.

The fog that appears over JRB when you first enter the level is listed here. It only appears in this state, however. You can't use this tab to bring it back when it's not meant to be there. Still in JRB, the water level inside the ship is listed, along with the start and end position of the water for when it drains. Strangely, the end value for this draining appears twice. The difference is that "JRB ship end WL" is the point at which the water stops draining, and "JRB ship goal WL" is the water level that gets set once the draining is complete. Normally these values are the same, but now you can make them different. Note that the goal is also used for detecting when to stop draining, but only as a failsafe, so it will go a bit below it before ignoring the other value.

The haze in HMC also gets classified as water, just like JRB's fog. It says that there's a real and fake value, but they're both visually real. The haze is actually 2 layers of textures on top of each other. It's simply that the real one is what gets used to determine whether Mario should be suffocating.

LLL volcano water level yes you heard that right and no it's not referring to the lava or mist of any kind. The inside of the volcano in LLL has water you can touch. You can't swim in it due to how close the real lava is to the surface, but if you get to the very edge, you can see Mario make splashes. And yes, you can swim in it if you were to move the lava triangles out of the way with the triangle tab.

The brown mist in the SSL area with the tox boxes gets classified as water as well. Furthermore, it's actually 2 water sections, and they meet up right here where the seam is visible. TTM's water levels are simply numbered because of how hard it is to describe them, but in STROOP they're simply numbered from bottom to top. TTM is the only level that actually attempts diagonal water, and you can see how bad it is. THI has an extra variable for indicating whether the water at the top of the mountain has been drained.

Every other area in the game is either well described right here or has the default water level of -11000.

All of the purple variables are for the exact details of how the water is laid out. You see, water is by default at Y -11000, and to have it not be that you need to designate a axis-aligned rectangular section using 2 X values and 2 Y values. This boundary will define a water section, where you are free to set the height to whatever you please. Whilst the limit in SM64's code for the maximum number of water zones is 20, the game (and thus STROOP) never use more than 4.

"Current water" is an index to which of the water zones you're currently over. It doesn't matter if you're in the water or not, only that you're in the XZ boundary. If you're overlapping none of them, this value will be -1. If you're overlapping multiple, the lowest index will take priority.

"Water pointer" is the pointer to the data for the water zones for the current area. "Num water levels" is the number of water zones this area is using. If this value is less than the maximum, the water code will ignore any values past the limit, meaning that even if you're within the boundaries of an unused zone, it won't register you as in it.

After that is the current water boundary data for all zones. They consist of the Y coordinate for the water level, the minimum and maximum X coordinate and the minimum and maximum Z coordinate. Finally, the constant for the default water level, which is normally -11000. You can change this to modify the default water level.

If you plan on moving the water zones, remember this neat trick. The code that checks if Mario is within any of the zones always uses less than or greater than for comparing positions. This means that if Mario has a float perfect position that lands exactly on the water seam, he won't count as in it. Most places in the game with two touching water zones don't keep this in mind, so you can do funny stuff like the following: go to the castle front yard with the moat filled up. Pause the emulator. Use STROOP to set Mario's position to exactly X 4000, Y 0, Z -58. Make sure Mario has exactly zero speed when you teleport him. Unpause and have fun!

Chapter 15: Miscellaneous tab

The miscellaneous tab is for important game things that don't really fit in anywhere else. They aren't part of an object, they aren't meant solely for Mario, it doesn't have to do with triangles, the files or the camera, no. This is just... stuff. Most of what's in here is useful and can be hard to locate in STROOP if you don't know about this tab.

There's barely anything on the left panel. At the top is an icon with the game's text saying "misc", with 3 things below it. The first is a checkbox to turn off the music. Checking this will turn off the in-game music. Only music, not sound effects. Unchecking this will not immediately bring back the music, but the next time a song loads in it will be played. "RNG index tester" takes the number in the textbox below and sets it as the RNG index when the "set & increment" button is pressed. This will also automatically increase the number in the textbox by 1. Note this increases the RNG index, not the value. The difference will be explained shortly. Also, the inputted number can be any signed 32 bit number, despite the RNG index limit being 65113. STROOP will simply take the modulo of the inputted number to set the index.

Finally, "go to course". This button is really useful, and it should've been made way bigger and more obvious. Clicking this button will open a menu listing every single stage, secret stage and place in the game. Clicking on these will warp you to that location. The stages and secret stages will warp you to either the star selection menu or spawn position if there is none, you can go directly to the fights of the bowser stages to skip the level itself, you can go to the castle lobby, front yard and back yard, and finally ending cutscene, which is mislabeled, because it takes you to the cake screen. This button doesn't work all the time, only in places where Mario exists. So not on the start screen, file select menu or cake ending screen.

15.1: Variables

"Global timer" is a timer that starts at 0 game the game boots up and counts up by one every frame. It's used by many objects to keep track of the time. "Global timer mod 64" is simply the global timer modulo 64.

"RNG value" is the last RNG value generated by the game. It's also the number that'll be used to generate the next RNG value. The RNG value starts at 0 when the game boots up, and it generates 65113 additional numbers pseudorandomly until repeating and going back to 0. This means that the sequence of numbers will always be the same. And because we know this sequence, STROOP provides the "RNG index" variable to give an index as to where in the sequence we currently are. "RNG index mod 4" is the RNG index modulo 4. This is useful, because Mario's dust calls RNG 4 times, and dust is the biggest way to manipulate RNG in TASes without wasting time. So if the RNG is on the multiple of 4 you want it to be on, you can manipulate it to your desired value easily.

"Num non-obj RNG calls" and "num RNG calls" gives the number of times in the last frame the RNG function was called. The first one filters out all object calls, whilst the second only gives the true total. However, much like the RNG object sorting from chapter 3, these won't do anything unless a specific hack from the hacks tab is enabled.

"Play time" converts the global timer into real time. It shows the time in a years, days, hours, minutes, seconds format. Because it relies on the global timer, it wraps around to 0 once it does too. "Num

loaded objs" is the number of currently loaded objects, "currently processed obj" is the last processed object, indicated with its slot number. This value can show up as VS1 if the last processed object was in the process of unloading.

"Animation timer" is similar to global timer, but it only counts up when objects are being processed, because it's meant to be used when objects on screen need to be animated. "Max obj render dist" is the minimum distance objects need to be to the camera to be rendered, but negative. By default it's at -20000, to 20000 units.

"Music on" is a checkbox that gets unchecked naturally when the game is paused or a selection menu is presented to the player that affects the music. In stages, this will mute the music but in the castle this will only lower the volume. Unlike the turn off music checkbox on the left tab, this one will continue playing the music when unchecked. If the result of unchecking this is that the music gets muted, it continues playing where it left off. "Volume" is the volume for the music. It can be set to any real number, but it's not meant to be above 1. Putting it above 1 can cause distortion to start appearing in the music, getting worse the higher the volume is.

Checking "special triple jump" will give Mario the special triple jump he normally gets from talking to Yoshi. "Talked to Yoshi" is the value the game uses to know whether to spawn Yoshi on the castle when Mario loads the area. "Never entered castle" is the value the game uses to know whether the game should show bowser's "no one's here" message when entering the castle. It's not a part of the save data but rather set to be true if you load into a new save file.

"Current file" is the currently selected file. "Demo counter" is the index into the demo sequence. It indicates which demo to play next. "Demo description" is a text description on which demo will play next. "Demo timer" is a timer that counts up for every frame the big Mario head is on screen, and it loads up the next demo when the timer reaches 800.

"Timestop enabled" is a checkbox that determines whether time should be "frozen". Normally, this happens whenever there's dialogue on screen. All objects not relevant to the dialogue will freeze in place and wait until timestop is disabled. Activating this effect outside of a textbox is known as the timestop glitch, and this checkbox right here is an easy way to trigger it on command.

"TTC speed setting" is the index of the current TTC speed. 0 is slow, 1 is fast, 2 is random and 3 is still. "TTC speed desc" provides these descriptions. "TTC save state" is interesting. It's a giant piece of text encoding the current state of every moving object in TTC. By copying this text, you can paste it later into this variable to set the state of everything in TTC to how it was when you copied it. However be careful, because it can crash Mupen if you enter a TTC savestate encoding a different number of objects than what you currently have loaded.

"Mission selected" is the index of the star you select in the star selection menu, indexed starting at 0. "Mission layout" is that plus one. "Mission name" doesn't give the name of the star, but instead just shows the same value as mission layout. "Last star" and "last star course" indicate that last star you've obtained. Last star is the index of the star, and last star course is the course ID.

"Selection index" is used for keeping track of the highlighted option in pause menus, save menus and character dialogue options. "Stage index" is an index indicating the current area of the game Mario is in. This value has nothing to do with the stage number seen in game. "Selection timer" is the timer

used to manage the delay when selecting between options in any menu of the game. It dictates how long you have to wait until you can move the cursor once again.

"Level index" is the level index you're familiar with. It's what appears at the bottom of star selection menus, although it also contains values for secret stages as well. "Level" contains the exact same value as stage index, but updates quicker. "Area" indicates which area of the current level/place you're in. As an example, if you're in the castle lobby, the value is 1. If you're upstairs, the value is 2. If downstairs, it's 3.

"Loading point" is similar to area, but more precise. Some places in the game have invisible loading seams, and this variable keeps track of that as well. As an example, TTM is loading point 1, the start of the slide is loading point 2, but the slide contains 2 additional invisible area transitions, so later sections of the slide have loading points 3 and 4.

The next three variables keep track of the last level sub-area you were in. This is done so that when you re-enter the painting after dying in a sub-area, you re-enter it immediately. "Last area" is the pointer to that area, "last area entrance" is an ID pointing to which entrance you used to enter that area, and "last death course" indicates which level this happened in. Last death course gets set to 0 when you collect a star, so that's how the game knows to have you enter the level normally once again.

The last few variables are tools to convert segmented memory addresses to virtual memory addresses and vice versa. To convert segmented addresses to virtual, you enter the value up here then press enter to get the result just below, and to convert a segmented address to virtual, you enter the segment ID here, the segment address here, press enter to get the result just below. If you're smart enough to have a need for this, you probably didn't need a tutorial anyway.

15.2: Hidden variables

This tab has two hidden variable categories: "advanced" and "coin".

For advanced, "hand X" and "hand Y" refer to the position of the Mario hand on the file select screen. Furthermore, these variables are messed up because they're mistakenly reading the floats as shorts. To create accurate variables, use the custom variable creator, and from a relative base, create floats with offsets 0x801A7D1C and 0x801A7D20.

"Mario cam selected" and "fixed cam selected" are for knowing which option is selected in the pause menu of a level. These can be manipulated to move the cursor out of bounds, but it doesn't do anything.

"Destination" and "destination 2" are the IDs for the location on where you're going during a warp and why you're going there. Stuff like getting a star, dying or getting a game over. They point to sSourceWarpNodeId and sDelayedWarpOp respectively in decomp. "Door direction" is an ID representing which way you went through a door. It points to sDelayedWarpArg in decomp.

"Transition state" is the current game mode. Curiously, setting this value to 5 enables a built-in debug frame advance mode, and pressing D-pad down advances the game by 1 frame. The camera goes out far in the distance when the game is in between frames, but it zooms in to its normal position when frame advancing. Pressing start exits this mode.

"Transition type" is the shape and direction of the fade cutout. With 256 added when the game is performing a transition, because STROOP is reading 2 bytes as a short. "Transition progress" is the fade timer. "Transition black fade" is a 2 frame timer for when the screen is completely black. "Transition white fade" is a 1 frame timer for when the screen is completely white. "Transition shrinking" is a timer for when the cutout is shrinking.

"Gfx buffer start" and "gfx buffer end" are the start and end of the graphics buffer. "Gfx buffer space" is the amount of remaining space. It's what's show as BUF when classic mode is activated in the debug tab. "ROM version tell" is an ID for the ROM you're using. It's not a checksum or hash, so it's not a 100% unique ID.

For the "coin" category, "coin RNG value" and "coin RNG index" are the first RNG value and index used by the last mobile coin to be spawned in the game. "Coin RNG index diff" is the difference between the goal index and the index used by the last mobile coin. Full explanation in a few sentences from now. "Coin H speed", "coin Y speed" and "coin angle" and the random speeds and angle the last mobile coin generated using its RNG value. "Goal RNG value" and "goal RNG index" are variables you can set yourself to tell STROOP which RNG value or index you're trying to hit. These will adjust eachother and "goal RNG index diff", which is the difference between your goal and the current index.

Chapter 16: Custom tab

This tab is a designated place to dump all of the custom variables you create. On the right, you have a few default ones representing Mario's most important values, but this is intentionally rather blank so that you can add whatever you want here using STROOP's variable creation tools. If you missed them, they're covered in chapter 4. Because variable creation and manipulation has already been explained in the variable chapter, the remainder of this chapter will cover the left panel.

The top controls are for saving and loading the variables. There already exist controls for saving and loading variables in the generic right click menu of the variable section, but due to the purpose of this tab, these buttons help outline that feature. You can open and save the variables and their values in a custom STROOP format. "Clear vars" just wipes the variable panel clean, deleting all variables.

Each of the 5 buttons in the top left (except paste vars) have right click menus. Right clicking "open vars" reveals an option to open Mario state data, which is just the default Mario variables but with a few more added in. Right clicking "save vars" will allow you to save all current variable pop-out windows created using the "pop out" option in a variable's extended right click menu. "Clear vars" has its right click menu offer an extra option for only deleting the variables added by default.

"Copy vars" will copy all variables currently in the tab into your clipboard. By right clicking this button, you can choose out of many possible copy formats and even set which one you want to be the default. This default setting is not saved and will reset when the program is closed. These formats are the same found in the extended right click menu for variables, so they've been explained before. "Paste vars" will take the data from your clipboard and put it into the variables currently in the tab. This acts as a sort of partial savestate, and you get to control what gets saved. Note that this button uses the same code as the one for pasting multiple variable values at once from the extended variable right click menu, so it also expects English decimal punctuation format.

Much like in the triangle tab, you can record data here. When you check the "record values" checkbox, STROOP will start recording data. It uses the global timer to keep track of in-game time and record data for each frame. To stop recording, simply uncheck the checkbox. The number to the side will indicate the length of the data list in frames. To view the data, click "show values". This will open a pop-up window with the table of values for every variable in the tab, along with the global timer as reference in the first column. The erase the data, click "clear values". The data will not be erased until this is clicked, so you can record, stop, and record again later.

"Use value at start of global timer" will prevent entries from being overwritten. With this unchecked, modifying values while the game is paused will overwrite the entry for that frame with the new data.

When recording data, STROOP uncaps its FPS from 30 to whatever it's capable of to keep up with the game. The loop that checks the values to record data runs asynchronously to the game, so STROOP increases its FPS to make it much more likely that every frame of data is captured correctly. To analyze this performance, two additional statistics are given at the bottom. "Freq" is the frequency of how many times per in-game frame is STROOP reading the value. You want this value to always be at or above 1 to not skip any frames of data. If frames do happen to be skipped because the emulator is in fast forward mode or some other circumstance, the number of gaps in the data will be indicated in "gaps". This just measures the number of missing global timer entries since the first one, so pausing recording, advancing the game a bit then resuming the recording will also increase this value.

The second panel on the left below the first one is simply a bunch of controllers to modify the look of the variable panel. These all function like standard variable controllers, with the plus, minus, get and set buttons. In order, the controllers control the width of the bar containing the variable's name, the width of the bar containing the variable's value, the height of the variable rectangle as a whole, the font size and the text's size relative to the top of the rectangle. Above all these controllers is a button to reset these values to their default. Important note, these controllers don't just affect the custom tab, but all variable across the entirety of STROOP. This is where you can customize their look a little. Changes made here do not get saved and thus do not persist when STROOP is closed and reopened.

Chapter 17: TAS tab

The TAS tab has a similar concept to the custom tab, but it contains variables and tools more likely to be used when TASing the game.

17.1: Self & point

The main features of this tab are actually not located on the left panel, but rather hidden on the right among the variables. You have copies of Mario variables, global variables, camera variables and input variables, but then you have things like "self" and "point".

Well, all of these blue variables provide calculations between two sets of values. Those values being "self pos" and "point pos" for positions and "self angle" and "point angle" for angles. By default, self is set to be Mario's position and angle, and point is a custom point that doesn't map to any memory. You can change these to map to the position or angle of other stuff, but that will be covered later in the chapter. I'll start with the basic operations found on the left panel.

"Store position" copies Mario's position to the point coordinates. You can right click this option to only store the lateral coordinates X and Z, or any single one of the coordinates. "Take position" copies the values from point X, Y, Z to Mario's position. You can right click this option to only take the lateral coordinates or any single one of the coordinates.

"Store angle" copies Mario's yaw facing to the point angle variable. "Take angle" copies the point angle to Mario's yaw facing. "Goto vertex" snaps Mario to the closest vertex of the triangle he's currently on. The final button is special and will be left for later in the chapter.

Below the buttons are 2 3D controllers, one for the self position and another for the point position. Checking the relative option will consider each position's associated angle variable for calculation. If its corresponding angle is NaN, it defaults to using Mario's yaw. You can also use Mario's facing yaw as the relative direction by clicking on the buttons and holding shift.

17.2: Position angle

Indeed, you can change what these positions and angles are referring to by changing the value in the "type" variables. These are special kinds of variables called "positionangles", because they encode a position and an angle in one piece of data behind the scenes. To change them, you simply select one of the four "type" variables and insert the name for what you want to track, with specific formatting if needed. Here is a complete list of every single thing you can track and the codeword/formatting for each. Capitalization does not matter for entering the terms. Not all of these entries can provide angles, and some can't provide positions. Non-existing or error values will show up as NaN. Get ready, cause there's many.

- "Custom" doesn't track anything specific. It lets you set a specific position or angle, and STROOP stores it in memory for use in other places.
- "Custom2" is a second distinct custom position and angle. Same goes for "self2" and "point2". This is required in order to have two separate custom points, as otherwise the two would point to the same values.
- "Mario" uses Mario's position and angle.

- "Holp" uses the HOLP position. Interestingly, if the Mario positonangle gets its position changed while the alt key is pressed, it moves his graphical object instead of Mario himself. This has little effect on the game, however.

- "Cam" or "camera" will use the camera's position and angle.
- "Camfocus" or "camerafocus" will use the position of the camera's focus.
- "Camgoal" or "cameragoal" will use the camera's goal position.
- "Camgoalfocus" or "cameragoalfocus" will use the camera's focus' goal destination as position.
- "Camhackcam" or "camhackcamera" will use the position and angle of the hacked cam. This camera will be covered in a later chapter. "Camhackfocus" is the position of the focus of the hacked camera.
- "Mapcam" or "mapcamera" will use the position and angle of the camera used in the 3D map. This camera will be covered in a later chapter. "Mapfocus" is the map camera's focus position.

- "Selected" will use the position and angle of the currently selected object.
- "Obj" or "object" followed by a space and the object's address will use the position and angle of that specific object.
- "Objhome" or "objecthome" followed by a space and the object's address will use the position of that object's home.
- "Objgfx", "objectgfx", "objgraphics" or "objectgraphics" followed by a space and the address of the object will give the position of the graphics for that object.
- "Objscale" or "objectscale" followed by a space and the object's address will interpret the object's scale parameters as an X, Y and Z.
- "First" or "last" followed by a space and the name of an object as seen in STROOP, with spaces where needed. These will return the first or last of that object in a list of all of them sorted by their processing order and use its position and angle.
- "Firsthome" and "lasthome" do the previous thing but with the positions of the object's home.

- "Goombaprojection" with a space and the object address of a goomba will give the theoretical position of a goomba if it keeps running towards its target angle for the rest of its countdown.
- "Bullypivot" with a space and the object address to a bully will return the position of the pivot of a bully. This is only useful when performing bully battery.
- "Pyramidnormal" with a space and the object address of a fire sea pyramid will return the normal vector of the pyramid.
- "Pyramidnormaltarget" with a space and an object address to a fire sea pyramid will return the target normal vector of the pyramid.
- "Koopathequick" will set the position and angle to hardcoded values based off of the global timer. The range of global timer values this works on is 23692 to 25981, all other values will provide invalid results. This was made by Pannen for Plush display a koopa the quick ghost on the map tab.
- "Ghost" will use the position and angle of the current ghost. This will be covered in a later chapter.

- "Tri" or "triangle" followed by a space, the address of the triangle another space and a number from 1 to 7 will give a position relating to the triangle depending on the number put at the end. 1, 2 and 3 will use the position of the first, second or third vertex of the triangle respectively, 4 will use the vertex closest to Mario, 5 will use the vertex closest to self (this also means this option cannot be used when putting this on self), 6 will use the point on the triangle under Mario, and 7 will use the point on the triangle under self (this can also not be used with self).
- "Objtri" followed by a space then the object address of the object you want the triangle from then another space then the index in the list of all geometry triangles for that object to choose your specific triangle then a space and finally a number from 1 to 4. This is an alternate way of selecting a position relating to an object triangle, this time by using the triangle's index in the list of collision triangles for that object. The number at the end works the same as with tri, but options 5, 6 and 7 are unavailable.
- "Floor", "wall" and "ceiling" followed by a space then a number from 1 to 4. These will use a position relating to Mario's currently referenced floor, wall or ceiling. The number at the end works the same as in tri.

- "Snow" followed by a space then a number will use the position of the snow particle with the ID of the given number.
- "Qframe" followed by a space and an integer will give a prediction for the position of Mario's next quarter-steps, with the specific quarter step being chosen by the number added at the end. However, the code doesn't work because it forgets to divide Mario's speed by 4, so you get incorrect values. Also, there's no bounds checking to make sure the number is below 5.
- "Gframe" is the same thing as qframe with the same syntax, but instead it first subtracts the value of the global timer from the added integer before using it for calculations. Because this uses the same code as qframe, it's also broken.

- "Marioprojection" gives the predicted position and angle for Mario on where he will be once he reaches 0 speed, but only when assuming that he is on flat ground, will not perform any additional movement until then and that he's in an action that reduces his speed by 1 each frame, such as moonwalking.
- "Rolloutpeak" will give the position and angle of Mario when he was at the peak of his last rollout. This only updates once Mario starts going down, but it sometimes doesn't register.
- "Previouspositions" gives the position of the median of the previous positions found in the first previous position tracker in the map tab.
- "Nextpositions" does the same but with the next positions tracker.

- "Pos" or "position" followed by a space, and up to 4 real numbers all with spaces between them for a hardcoded position angle. The first number is X, the second is Y, then Z then the angle. If the angle is not an integer, it will be rounded down. If fewer than 4 numbers are added after pos, then the remaining numbers will become NaN.
- "Ang" or "angle" is the same as the above, but it only takes one additional number, and that number is the angle.

- "Trunc" followed by a space then any other thing listed on here will truncate the position of that thing to an integer. And by any other thing, I mean you can take any entry on this list, add trunc before it to truncate it. Hell, you can even add trunc multiple times even if it won't do anything.
- "Offset" followed by a space, 2 real numbers with spaces between them, another space with a boolean value, another space then in brackets, any other entry in this list. The first number is the magnitude of the offset vector, the second number is the yaw angle, once again truncated to an integer, and the boolean value can either be written as true or false, or 0 or 1. This boolean value determines whether the offset is relative to the facing direction of the positionangle entered within the brackets. This is a very powerful entry and can be used in clever ways. Say you want the position exactly 100 units in front of Mario. Then, you would enter "offset 100 0 true [Mario]", and it will give the coordinates for the point 100 units forward at angle 0, relative to Mario's angle.
- "Yoffset" does the same thing, but it only takes one number before the brackets instead of three. This is for adding a vertical offset. Be careful, because both offset and yoffset don't check for the number of segments in your message.

Finally, the last possible entry is "none". It does absolutely nothing, and modifying any of the components will completely break STROOP, forcing you to close it with the task manager and restart the application. So uh, don't use it.

17.3: Variables

As discussed near the start of the chapter, all the red or green variables are copied from either the Mario tab or some of the other basic tabs. All the red ones come from Mario, global timer and the RNG variables come from the miscellaneous tab, cent angle comes from the camera tab and current X and Y come from the input tab, and they specifically refer to the raw inputs from the control stick before they've been treated by the game.

Then, you have the positionangle variables, self pos, self angle, point pos and point angle, and after that all of the mathematical operations made on those two. After that is the distance between those two point, for the X, Y and Z coordinates, and also lateral and total distance. "Angle point to self" is the yaw angle you would need to face towards if you wanted to go from point to self. "Dangle point to self" is the point angle minus the angle from point to self. "Angle self to point" and "dangle self to point" are the same as the previous two, but this time from self to point instead of point to self. "Dangle self to angle" is the self angle minus the point angle, clamped to -32768 to 32767. "Fdist point to self" and "Sdist point to self" provide the forwards and sideways distance from point to self, taking into account point's angle. "Fsdist point to self" is the last 2 values coupled together in parentheses, without rounding.

"Pitch self to point" is the pitch you would need to have to be facing Point if you were at Self, as a signed value ranging from -16384 to 16384. "Walking distance" is similar to the Marioprojection positionangle, but this gives the distance Mario has left to travel assuming he's on flat ground. "Walking distance diff" is just walking distance minus the horizontal distance from point to Mario. The final variable, "Schedule offset" makes a great segway into the last feature of the TAS tab.

17.4: Schedule

There was actually one last positionangle type that was intentionally left out of the list: "schedule". And that's because it connects to many other parts of this tab. A schedule is a sequence of positions and angles that can be played back and shown in either self or point for reference. It can be created in the custom tab by having the first 4 variables be an X, Y, Z then an angle, and recording those values. Once done recording, you can select everything in the pop-up window, copy it to your clipboard and paste it into the TAS tab with "paste schedule". By default the values in the schedule are treated as double precision, but in the right click menu for the button you can force it to use single precision floats.

If a valid schedule is set using this method, the positionangle types for point position and point angle will automatically be set to "schedule", and if your schedule contained any extra variables after the angle, they will be added as extra variables at the end, unlabeled. Schedules use the global timer to anchor themselves to time, so you need to set the global timer to the value it was when you recorded the schedule to have the values line up.

If you don't want to change the global timer value, you can use the last default variable in the tab "schedule offset" to add an offset to the schedule's global timer on playback. The value here will simply be subtracted from the current global timer to get the value used to look for the corresponding schedule entry. If a schedule is missing data for a specific global timer value, the variables will show up as NaN.

Note that the schedule feature will not work if your computer's regional settings has the fractional separator for real numbers set to a character other than the period. If the separator is a comma, the parsing scripts will fail to extract or display the data from the schedule.

17.5: Hidden variables

There are two hidden variable categories: "TAS" and "Point2". The point2 category is very simple. It's simply a second set of self and point position angles. This time labeled self2 and point2. All the variables work in the exact same way as self and point.

For the TAS category, all the variables highlighted in red are not only just copied from the Mario tab, but they're all also found in the basic variable category. Except for "H movement", which finds the position between values at hardcoded memory addresses.

The variables highlighted in blue will give the state of the currently active ghost when the ghost hack is enabled. There's the ghost's position, speed, angle, action and the current frame of the ghost recording being played.

The purple variables are more ghost information. The dist variables are Mario's position minus the ghost's position, the forwards and sideways distance between Mario and the ghost, the delta speeds and delta yaw angles, which are also just the Mario variables minus the ghost variables.

Green variables are copied from miscellaneous tab, pink are more Mario tab variables, and "lag" is a number representing Mupen's lag, which is calculated as the number of VIs in the current TAS, minus the number of frames times two. This number is meaningless when not playing back or recording a TAS.

Chapter 18: Map tab

The Map tab is the magnum opus of STROOP. It's by far the largest and most complicated tab. It's used very frequently in high-skilled TASing, and mastering it is the key to using STROOP to its fullest TASing potential.

18.1: Navigating the map & options

The map tab is composed of three sections. On the right is the map. On the top left is the sub-menu and on the bottom left are the trackers. To start, we're just going to learn how to navigate the map and learn what the options that affect this navigation do.

Navigating the map is as simple as holding down left click and dragging. To zoom in, you scroll the mouse wheel. By default, the map is a simple orthographic top-down view of the currently loaded area, with ceilings and upper floors removed for better visibility. To rotate the map around the current center of the panel, hold down the right click and move your cursor in a circle around the map. This works better when zoomed out but is always finicky. Double clicking any mouse button will bring the map back to a centered, non-zoomed and unrotated state. You can also hold shift while holding left click to rotate the map instead of using the right click button. Holding control while rotating the map will snap it to multiples of 45 degrees.

The options panel on the right contains a few options that are worth mentioning this early. "Enable PU view" has the map draw parallel universes when you zoom out far enough. This also helps getting a feel for the scale of parallel universes, how far away they are from the original map. They are only visible in the map's top down mode. "Enable reverse dragging" will inverse the direction of all moving and rotating when dragging with the mouse. This works in all map modes.

By default, the map will display the place where Mario is currently in along with its corresponding background, but if you want it to be something else, the "level" and "background" drop-down menus let you select which map or background is currently being shown. "Recommended" will update the map and background the next time you load a new area or a savestate. And not all backgrounds are available, some like the WDW background are missing from the list. You can choose "transparent" in either to remove that layer from the map entirely, and there are several custom single color or gridline backgrounds available to choose from.

18.2: Navigating the third dimension

Top down orthographic mode isn't the only map mode available. If you click "enable orthographic view", the view mode switches to showing a 3D orthographic view of the map. To help see better, triangles are rendered without textures, translucent and tinted a blue, green or red color depending on their type. You can move the map around with left click, and rotate the map with right click or shift, but now in 3D. It's an orthographic projection of the level, meaning that everything maintains a consistent size no matter how far it is from the camera. You can also hold control in this mode while rotating the map to snap to 45 degree angles, but this will also bring the camera down to ground level.

By checking "use cross section" while in orthographic mode, the map will only display the wall, floor and ceiling hitboxes along the cross section created by taking the current angle of the map camera and creating a horizontal plane where Mario is. Wall hitboxes will attempt to align themselves to either

X or Z as they do in the game, but the code that displays this is a bit off, so the wall hitboxes can appear way wider and thinner than they are at certain map camera angles. Once again, double clicking any mouse button will bring the camera back to a fixed point where the origin is in the middle and is looking directly forwards.

The last 3D camera mode is for full 3D, and is enabled by checking "enable 3D". This setting has priority over the orthographic modes. This turns the map into a real 3D map using proper perspective projection. In this mode, triangles are not opaque, but are rendered with a gradient to help distinguishing them from eachother. Wall triangles can be two different shades of green depending on if they're using an X projection for wall collision or a Z projection. If you don't move the map camera after first enabling this mode, its default behavior will be to match the position, angle and FOV of the game's camera in real time. To move around, hold left click and drag to move the camera along the plane created by its angle. So, from the camera's perspective, directly up, down left or right. Holding right click will let your rotate the camera in place, without changing its position. You can also hold shift to use left click for this. Scrolling the mouse wheel will move the camera directly forwards or backwards. Holding down both left and right click at the same time will stack the movement and rotation effects.

If you hold shift, hold left click then let go of shift, you enter a mode where simply moving your mouse around turns the camera. This might be a glitch, and it's easy to do accidentally, you can simply exit this weird mode by right clicking or shift clicking. This can also happen with camera movement by holding down left click, holding down shift, letting go of left click then letting go of shift. This time you exit this with a left click. This also works with holding left + right click.

When in any of the 3D modes, checking the "disable hitbox tris" option will hide all triangle hitboxes. This has the effect of making the level invisible if you have no trackers currently active, so this isn't useful for now. Right clicking this option will reveal two new buttons. "Toggle show arrows" will switch between showing and hiding the arrows in the cross-section map mode. These arrows and symbols are the same as seen in Pannen's walls floor and ceilings videos. "Reset" simply resets the level and object triangle lists in case they ever get desynchronized for some reason.

18.3: Trackers

Trackers are the real heart of the map tab. Map trackers are things you can add to the map to track anything in the game. They can range from points, triangles, shapes, arrows and so on. When you first open the map tab, a single tracker is added by default: Mario. As you can see, moving Mario around in the world will also move his corresponding icon on the map.

You can add more default trackers using the left row of checkboxes in the panel above. Mario is selected by default, but you can also track the HOLP, the camera's position, the camera focus' position, the ghost currently active with the ghost hack, the self positionangle from the TAS tab, the point positionangle from the TAS tab, Mario's currently referenced floor triangle, wall triangle and ceiling triangle, and finally unit gridlines, which don't appear until you're zoomed in far enough. You can right click on the label of all of these trackers to add more of that kind.

As well, you can click on objects in the object slots to add them as trackers. Object slots will have a rainbow box around them indicating that they are currently being tracked in the map tab. This rainbow highlight takes precedence over the indicator for marked objects.

Every single tracker you currently have on the map will appear in the bottom left panel, the tracker list. They each have their own box, with a variety of info and controls. As a note of precaution, many of the controls accept scrolling your mouse wheel over them as a way to modify their values, so I strongly recommend moving your cursor over to the right column of icons when scrolling through the tracker list.

The rightmost column of icons are the main controls for the tracker. Click the big red X to delete the tracker, click the eye icon to hide the tracker from view in the tab, and click it again to make it visible again. You can right click this eye to toggle which map modes you want this tracker to be visible in. The up and down arrows allow you to rearrange the order of the trackers in the tracker list. Click on these arrows while holding control to move the tracker to the very top or very bottom of the list. You can also hold a number key while clicking to move the tracker X spots in the list instead of 1, where X is the held number key. The gear icon allows you to access additional controls for that tracker, if there are any. The plus icon allows you to attach a tracker to this tracker. The gear and plus icons will be covered extensively later.

On the left side of the tracker, you have that tracker's icon on a red background. You can right click the icon to choose whether you want to use the default one, the one found in the object slots (if it's available for this tracker) or a custom image from your computer with "use custom image". Below that is the tracker's name. You can set it to whatever you want, and you can return it to its default name by right clicking and selecting "reset custom name". Setting a custom image doesn't work for all trackers, but sometimes it only appears under certain conditions, or can also be put onto trackers that don't have an icon of their own normally.

The three sliders to the side control the appearance of the tracker on the map. "Size" is for the size of the tracker. If the tracker has an icon on the map, this refers to the number of pixels the image will take up on your screen, divided by two. But it can also refer to other aspects of the tracker if it does not have an icon. Like for example, a floor triangle hitbox will have its size be 78 by default as its height, which is the size of its collision when Mario is walking. In the options panel above, there is a slider labeled "global icon size", and this slider allows you to change the size of all icons on the map at the same time. This will not affect size sliders for trackers that don't use that slider for icon size. If the size slider affects something other than an icon size, there is usually an option in the gear menu (covered later) to manually set the size of a custom icon.

"Opacity" is the tracker's opacity on the map. 100 is fully opaque and 0 is invisible. "Line" refers to the line width of trackers that are things drawn to the map rather than icons placed on top. The value represents the width of the line in pixels on your screen. All three of these values can be popped out as controllers by right clicking on the text label, and right clicking on the slider for either the size and line size will let you select its range, with powers of 10 up to one billion and 65536 being available. These limits only apply to the sliders. However, you can manually set the values to very very large numbers.

The topmost drop-down menu is used to indicate if the tracker should appear on top of the others or below. By default, trackers use their Y position in the world to determine what appears on top of what. You can change this to "order on top" or "order on bottom" to either have the tracker always be on top or always be on the bottom. Ties are broken with the tracker's position in the tracker list, with higher up trackers being drawn on top.

The second drop-down menu is for when the object should be visible at all. By default, trackers are set to be visible only when loaded, but this can be changed to "visible always" or "visible when this bhvr is active" (the text is cut off). Visible always makes the tracker always visible even when the tracked thing unloads, and the behavior option only makes the tracker visible if it's currently running its behavior code.

Next are two color pickers. The color is represented as a 24 bit RGB value, where three numbers from 0 to 255 represent the red, green and blue channels respectively. You can modify the value directly or click on the colors to the side to open a windows default color picker. The top color picker is for the fill color of the tracker and the bottom one is for the line color. If the tracker is an icon, these don't do anything. Unless you're in 3D mode, where the top color picker color gets multiplied with the icon image to tint it.

The last three checkboxes are more options for how tracker icons should be drawn. "Rotates" determines whether icons should rotate on the map to match the angle of what they're tracking. Right clicking this option shows "reset custom rotates", which resets the angle of the tracker back to where it was, and sets the rotates flag to its default value for that tracker. "Scales" changes the meaning of the size slider. When off, the value of the size slider represents half the size of the icon in pixels relative to the screen it's being displayed on. When active however, size now represents a half the size of the icon in in-game units. Meaning that changing the size of the map now also changes the size of the tracker, instead of keeping it a static size. "Rel" being checked means that trackers located in parallel universes will be drawn in the home universe instead of their actual location.

Additionally, in the options menu panel above, checking the "enable object hover" option will activate a mode where hovering over the map or certain trackers gives more info on them. This mode does not work with 3D enabled and you're unable to rotate the perspective in orthographic mode when this option is on. All the way at the top of STROOP above the tab list, you will see information text pop up whenever your cursor is over the map. When it doesn't see any trackers with hover info, it simply gives the position below the cursor that intersects with the plane Mario is on. In top down mode, this means that the Y value in the info is Mario's Y value. Right clicking anywhere on the map will open up the hover mode options. By default, there will always be 2 options. The first one lets you copy the position under the cursor found at the top of STROOP, in the format of your choice and with the option of it only being XZ instead of XYZ. The other option is "go to clicked position" and teleports Mario to that position. This doesn't perform any OOB checks, so be careful. For all the default trackers, they simply give their position as information, which you can copy, paste and go to in the right click menu for these. Except the unit gridlines which can't be hovered over and the triangles which have their own menu covered in the next sub-chapter. When in hover mode, you can hold control to ignore trackers with hover information, pretending that they aren't there. You can right click the enable object hover checkbox to toggle on and off the object hover detecting certain types of trackers. This is useful for quickly reaching something hidden under another tracker.

Just above that is another option labeled "enable object drag" that will activate a mode where you can drag certain trackers around by holding down left click over them. This doesn't only affect their position on the map, but this also modifies the in-game position of the tracked object in real time. This only works in top down or orthographic perspective modes, and so because of that you can only move the object along a plane. In top down mode, the Y value is what will stay constant. Dragging

objects also doesn’t make collision or OOB checks, so be careful. When dragging around most positionangle trackers like Mario, camera, etc. You can hold control while dragging to lock the Z position of the tracker and hold shift to lock the X position of the tracker. There’s something similar in orthographic mode, where holding control will lock the Y position of the tracker and holding shift will lock the lateral position of the tracker.

18.4: Adding your own trackers

So far, the only trackers shown are the default ones listed in the options menu that are a simple checkbox away. The 4 other buttons provide tools for managing trackers. "Clear" deletes all trackers in the tracker list. Right clicking this button opens more options. "Reset to initial state" deletes every tracker and adds back the Mario tracker, bringing the tracker list back to how it is when you first open it. The next three options are presets, and so they’ll be explained in the presets category of the add button. "Save" allows you to save your current tracker list as a custom ".mapdata" file. This file can be read later with the "open" button, and opening a STROOP mapdata file will add every tracker from that file to your tracker list without deleting the existing trackers. By right clicking the save button, you can choose to have your mapdata file save more than just trackers. It can also contain map tab settings and STROOP settings.

The "add" button allows you to add many more trackers to the map. This document will go over every tracker in the add button menu and their additional options in the gear menu if available. If the tracker contains data viewable with the hover mode, that’ll be mentioned as well.

The first tracker category is "objects...", and it gives more options on how to add objects as trackers. You could already add individual objects by clicking on their slots, but using the options here, you can add them in bulk and have them all share a single tracker slot in the list. "Add tracker for all objects" will add a tracking that tracks every object slot, both used and empty. "Add tracker for marked objects" will put all marked object slots into one tracker, no matter their marking color. "Add tracker for specific marked objects" does the same but lets you select a specific marking color. "Add tracker for all objects with name" will open a new window asking you to enter an object name. The name has to be exact (but capitalization doesn’t matter), and match the name given to the object by STROOP. This name can be seen in the object tab for that object. When the name is entered for an existing object, STROOP will collect every object with that name and put all of them under one tracker. This tracker category is the only way to group objects together under a single tracker, as adding object using the object slots, even with shift held, will simply add one tracker at a time. Additional info for all object trackers, right clicking on them in hover mode gives you an option to open up that object in the object tab.

The "triangles..." tracker category is for adding collision triangles. "Add tracker for level floor tris", wall tris or ceiling tris will add a tracker for the entirety of the level’s collision triangles of that type. "Add tracker for all object floor tris", wall tris and ceiling tris will add a tracker for the collision triangles of the selected type of all tangible objects. "Add tracker for custom floor tris", wall tris and ceiling tris will open up a window and prompt you to enter a list of memory addresses for triangles of that type separated by commas. If you don't enter anything, it will automatically add Mario’s currently referenced wall, floor or ceiling. In hover mode, all these triangle trackers including the default ones give their tracker name followed by the address of the hovered triangle, the index of that triangle within the group that the tracker controls, and finally a position calculated by taking the median of all vertex

positions for each axis individually. When right clicking a tracked triangle in hover mode, you get new options. "Select in triangles tab" opens that triangle in the triangles tab. "Copy address" will put the memory address of the triangle in your clipboard. "Copy coordinates" will put the triangle's coordinates in your clipboard in this format: "X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3". "Annihilate" deletes the triangle from memory. "Unload associated object" will unload the object this triangle belongs to. If this isn't an object triangle, nothing happens.

All triangle trackers also contain a very extensive options menu accessed with the gear icon on the tracker. The options you get depends on the type of triangle tracked. The options common to all triangles are the following: "set min height" and "set max height" open windows asking for a number. This number will set a minimum or maximum Y level for where triangles can be drawn on the map. Entering no value will default to Mario's Y position. Triangles tracked by that specific tracker that fall outside of this vertical range will not be drawn on the map. If the triangle falls on both sides of either the minimum or maximum, it will be cut off partially and only the part of the triangle that fits within these limits will be shown on the map. This has the effect of possibly turning triangles into quadrilaterals. Instead of adding a constant value, you can also put in the name of a positionangle using the syntax found in the TAS tab. This allows for a more dynamic minimum or maximum. Bear in mind that putting in "none" will crash the map tab, so don't do that. "Clear min height" and "clear max height" will remove this limit. These options have a different name and function for walls. On walls, it's "set relative height" and "set absolute height". These two have you set a Y position, and the Map will only show wall hitboxes that affect Mario at a certain height. Relative height will set that height relative to Mario's position while absolute height won't be relative to anything. These can be cleared with "clear relative height" and "clear absolute height". "Show arrows" will show the arrow symbols for walls, floors and (the not symbol for) ceilings when displaying the cross section of the triangle, the same as seen in the walls floor and ceilings videos. "Use cross section" will display the cross section of the triangle even when in regular orthographic mode. "Color by height" will apply a gradient to the triangles to turn them into a height map. This doesn't work on walls or 3D. "Set icon size" is there because setting a custom image on a triangle tracker will put that image on every vertex. This option will let you change the icon size because the size slider is used for the triangle's hitbox size. "Set within dist" will only show triangles if Mario's Y position is at least a certain number of units away from the triangle's vertical range, with the number being set with a window that opens up when the option is clicked. This value can be negative, in which case Mario has to be far enough within the triangle's vertical range for it to appear. "Set within center" will let you set your own custom Y value for the previous option instead of Mario's Y value. Both these options can be cleared with "clear within dist" and "clear within center".

On both floors and ceilings, "show tri units" will show the individual unit bounds for the edges when zoomed in far enough. This is crucial for unit-precise movement or analysis. This also works in cross-section mode, but only when use cross section is also enabled in the gear menu. "Truncate bottom of hitbox" Will only affect floor or ceiling triangles in cross-section mode with show tri units active. It truncates the bottom of the unit-columns to a whole number. There are also three additional options exclusive to floor triangles. "Exclude death barriers" will not draw triangles with the death barrier floor type. "Distinguish sliding classes" will color floor triangles based on how slippery they are with their floor type. There's many different properties these floors can have, but in general, normal slipperiness will be green, no-slip will be light blue, slippery floors will be yellow and very slippery floors will be

red. "Enable quarter frame landings" will do many things. First, it will only show floor sections with a Y value obtainable by Mario within the next 10 frames if only his position, speed and gravity were to be taken into account. Next, it color codes each section of the triangles either red, yellow, green or blue depending on which quarter-step Mario would land on the triangle with. Red means first quarter step, then yellow, green and finally blue for the 4th. When TASing for speed, you'll want to use all 4 of your quarter steps, so you should aim for blue sections.

The remaining triangle tracker options can only be found on the ones located in the add button category. They all share "auto update", which toggles the option to have the triangle change their position on the map when they move in the game. This is enabled by default except for level triangle trackers. Custom triangle trackers will have the option "add more tris", which lets you add more triangle addresses to this tracker. Level and object triangle trackers both share "use current cell tris", which only draws the triangles located in Mario's current cell. All remaining options can only be found in level triangle trackers, no matter the triangle type. "Update on level change" toggles whether the triangle tracker should swap out old data for new data whenever the level differs from the one it's drawing. "Reset" will clear the tracker's triangle data and force it to fetch what's currently in the game's memory, overwriting any other options meant to pause its tracking. "Remove current tri" toggles whether Mario's currently referenced triangle should be removed from the tracker's rendering list. It doesn't bring triangles back when Mario leaves them, so the only way to get them back is to first disable this option then reset. "Show tri data" will open a new window containing a table for the data for all triangles under that tracker. This is the same type of window and table as the show tris buttons from the triangle tab. "Open form" opens a new window containing data about the triangle tracker. It displays the number of tracked triangles within this tracker, along with 4 buttons and each triangle's address along with the distance between its center and Mario. On this window, "sort" will refresh the list and sort them so that the closest triangles to Mario are first, "annihilate" does the same as the button from the triangle tab, "inject" injects the hitbox code seen in the version number's right click menu and "remove" removes that triangle from the list, the same way that remove current tri option works. "Include object tris" makes the level tris tracker also track object triangles. And that is finally everything for triangle options.

The "points..." tracker category lets you track custom point and shapes of your liking. Each one of these trackers will start by opening a window and asking you for a list of numbers. All you need to do is list either pairs or triplets of coordinates/values, separated by commas and nothing else. Once done, you click pairs if you listed 2D coordinates and triplets if you listed 3D coordinates. For each of these buttons, you can right click them and select "use clipboard" to use the text in your clipboard as the value for that button. "Add tracker for custom unit points" will track exact unit points with a yellow square. No icon, so you really have to zoom in quite a lot to be able to see it. If these units align with 0 on any lateral axis, they will double in width to cover both 0 and -0. In hover mode, you get its index within the tracked points of that tracker, and its position. Right clicking allows you to copy its position. "Add tracker for custom cylinder points" lets you add a cylinder to the map. The coordinates you enter represent its center position. To adjust the dimensions of the cylinder, change the size slider to modify its radius size in units, and in the gear menu, "set relative min Y" and "set relative max Y" represent the top and bottom Y levels of the cylinder relative to its own Y coordinate (which is 0 if you didn't enter any). By default, the cylinder has a 100 unit radius and stretches from its base Y to 100 units above. "Use cross section" forces the cylinder to display its cross section in orthographic mode, and

"set icon size" is to set the size of a custom icon if one is assigned. In hover mode, you get the index of that cylinder within the cylinders tracked by its tracker and the position of the current bottom middle of the cylinder. Right clicking lets you copy that position. "Add tracker for custom sphere points" lets you add spheres to the map. In top-down mode, spheres are shown as an intersection of the sphere using Mario's height. If Mario's Y position is not between the top and bottom of the sphere, it's invisible. You change the sphere's radius using the size slider, and the gear menu lets you toggle viewing its cross section and setting the size of a custom icon. Hover mode gives the sphere's index within that tracker, the position of its center and right clicking lets you copy that center position. "Add tracker for custom icon points" is another way of tracking specific points, but now with an icon. By default, the icon is green Mario, but ideally this is changed to a custom image. Its gear menu lets you copy the point's position, or just its lateral position. In hover mode, you get that point's index within the tracker and its position. Right clicking will let you copy its position, go to its position, go to its position laterally and "remove point" will remove that individual point from the tracker.

The "gridlines..." category lets you add trackers that divide the level at different scales. "Add tracker for unit gridlines" is the exact same tracker as seen in the default trackers. It's the same thing. In the gear menu, "override extended level boundaries" will set the unit gridlines to appear as if extended level boundaries is not activated, even if it is. That's because if extended level boundaries is enabled in STROOP options, units will expand their size by 4 times, so by default the unit gridlines compensate for this. "Set icon size" lets you set the size of a custom icon. "Add tracker for float gridlines" works similarly to the unit grid, but you have to zoom in way, way further in to see the grid. The closer you are to the map's origin, the more tightly packed the float grid is and the further you'll have to zoom. These visualize all the possible positions that Mario or any object can exist on. The gear icon only lets you set the custom icon size. "Add tracker for cell gridlines" adds gridlines for the collision cells of the game. To learn more on how they work, visit the cells tab chapter. The only gear menu option is to set the custom icon size. "Add tracker for PU gridlines" shows gridlines indicating where parallel universes are located. By default, the intersections of the lines show the local origins of universes. In the gear menu, you can switch between 3 settings on how to display the lines. "Setting 1" is what's on by default, "setting 2" puts the lines half way in between the PUs, and "setting 3" puts lines around the areas where collision is able to exist. In 3D, these lines will always match their Y position with that of Mario. When setting 1 is enabled, the size slider dictates how many PUs it requires for the lines to appear again. The option "use Mario as origin" doesn't do anything, surprisingly. You can also set the custom icon size. "Add tracker for custom gridlines" will add gridlines around the level. You can use the size slider to choose in how many section the level should be divided into. The only option is setting the custom icon size.

The "level..." category lets you add an overlay to the map displaying an entire level or background. "Add tracker for custom level" adds an entire level image as an overlay. To change the displayed level, click on "select level" in the gear menu, then a window will pop up with the same list as seen in the map options above. This image only appear in the top down map mode. "Add tracker for custom background" does the same, but with the background list. Interestingly enough, adding a custom icon to either of these will let you set your own level image or background. Just make sure to add both and layer them correctly so that the map is still usable.

The "Mario positions..." category is where you can add trackers that predict or analyze Mario's positions. "Add tracker for iwerlipses" adds a tracker for iwerlipses. Iwerlipses are named after SM64

ABC contributor Iwer Sonsch, and the shape represents the range of positions a normal airborne Mario could be in on subsequent frames depending on his stick position. This means that when not in the air or when in a butt slide animation, this tracker is meaningless. By default, it shows the positions of the next 12 quarter steps, but this can be changed with the size slider. In the gear menu, "lock positions" will stop updating the iwerlipses, keeping them wherever they were, in their same position and shape. "Show quarter steps" will toggle whether all quarter step iwerlipses should be shown or only the ones for full frames. "Add tracker for next positions" will display Mario's next predicted positions, assuming nothing unexpected blocks Mario or changes his action and Mario keeps his current speed. This almost never happens, so this tracker is not very useful. Quarter frame positions are marked with an orange Mario icon and full frame positions are marked with a light blue Mario icon. For each of these, in hover mode you can see the index of that position and its value. In the gear menu, toggling "use colored Marios" will switch between the colorful Mario icons or only standard red Marios for everything. "Show quarter steps" toggles displaying the orange Mario quarter steps. "Set num frames" lets you pick the number of full frames to run the simulation for, and thus also how many icons there are. "Use pitch" accounts for Mario's current pitch in the position prediction. "De facto speed..." lets you choose between auto, force enable and force disable. By default, it's on auto, which only accounts for defacto speed when Mario is touching the surface of a triangle. Forcing it on or off will have either have it always be affected by the floor below Mario or never affected.

"Add tracker for previous positions" will display the positions of Mario's 4 previous quarter steps and what happened to them. Because this information isn't stored in the game's memory, this can only work if a custom hack is activated in the hacks tab or by clicking the gear menu and selecting "enable ROM hack". This hack can only be enabled with the emulator's CPU mode set to pure interpreter. When enabled, Mario icons of various colors will depict your last frame's history. The full color code can be accessed in the gear menu under "see key", but the most common colors are pink for the starting position, orange for the next intended position and light blue for a wall push. "Show each point" will play a neat animation showing every single position change that happened in the movement code, one by one by hiding and showing the different colored Marios. "Track history" will start keeping track of all previous position histories, even for previous frames. It will keep recording until this option is unchecked. It does this using the global timer, meaning you have to be careful to not override data or have STROOP lag enough to skip a frame. "Pause history" will pause the position recording. "Clear history" will erase everything that's been recorded so far. Having history recording turned on also disables the ability to view the show each point animation. In hover mode, you get the index of that Mario icon, with older icons getting higher numbers, and its position. You can copy this position in the icon's right click menu. "Add tracker for slide positions" creates a simulation of the next frame assuming Mario is sliding and adds a Mario icon for every single possible position he could be in depending on the next control stick position. It's sort of like iwerlipses but for sliding movement. This obviously creates a lot of icons, so I recommend not having too many of these active at once. In hover mode, hovering over any of the Mario icons will give that specific simulation's index, its position, and all of its other simulation results. X and Y are the stick inputs generating that position, and everything else is the result. When right clicking on any of these, you can not only copy the position and go to it, but also copy any of the other result variables from that simulation. In the gear menu, "pause" will freeze the current simulation in place and not run any more. To resume the simulations, toggle pause once more. "Exclude 0 hspeed cases" will not render any simulations where the result

of that simulation left Mario with 0 horizontal speed. "Exclude turn-around angle" will not render any simulations whose control stick position would result in the absolute value of Mario's intended angle minus his facing angle being greater than 18204. "Add tracker for OJ position" tracks where Mario would end up if he were to perform an overflow jump. It appears as a single orange Mario icon whose position depends on Mario's current speed and angle. This tracker has no options, and hover mode simply gives the index and position of the OJ spot. Right clicking lets you copy the position.

The "Mario move..." category is for trackers that would aid with planning out or analyzing Mario's movement. "Add tracker for C-up floor tris" adds a custom floor triangle tracker that automatically adds to itself every single level geometry triangle with a floor type that allows for C-up hyperspeed. The options and hover menu details are exactly the same as a custom floor tris tracker. "Add tracker for punch floor tris" adds a custom floor triangle tracker that automatically adds to itself every single level geometry triangle with a floor type that allows for hyperspeed punching. The options and hover menu details are exactly the same as a custom floor tris tracker. "Add tracker for punch detector" adds a custom cylinder tracker always located 50 units in front of where Mario is facing with a radius of 5 units and a height ranging from Mario's position to 80 units above that. This is the area that the game checks for walls in when Mario is punching to know if he punched a wall. The gear and hover menus are exactly the same as a regular cylinder tracker. "Add tracker for ledge grab checker" adds a tracker that draws 2 lines to the map in all modes except top-down to show where the ledge grab checks is performed. The thicker purple line represents the wall checks. Walls are not checked across the entire line, but rather at each end of the purple line. 30 units above Mario and 150 units above Mario. If a wall is found at the bottom position but not the top position, it passes on to the second step, which is finding a floor to grab onto. This is what the skinnier orange line is for. The games scans from the top of the orange line, which is 238 units above Mario, to the bottom of the orange line, which is 100 units above Mario. Once it finds a floor, it makes Mario ledge grab onto it. If Mario is not currently touching a wall, the orange line will be right above him. However, its position when Mario is touching a wall is its accurate position. Changing the size slider changes the diameter of the orange line and changing the line slider changes the diameter of the purple line. In the gear menu, you can click "set custom wall triangle", which fixes the position of the orange line relative to Mario in place, ignoring any other walls. You can clear this with "clear custom wall triangle".

"Add tracker for HOLP displayer" displays many green star icons packed very closely together near Mario. You notice how when Mario is standing still with an object in his hands, the object moves around slightly? Well that can be crucial when you need a precise HOLP, so this tracker shows every single position the HOLP goes through during the object's slight movement animation, along with a thin line connecting the path. You can visualize this better by activating the HOLP tracker at the same time while Mario is holding something. As you'll see, the HOLP tracker pass through every green star. This tracker follows Mario around, even if the HOLP isn't with him. In hover mode, you get each icon's index and position, and right clicking lets you copy it. "Add tracker for squish cancel spots" will draw red and green unit dots when possible squish cancel spots are found. These can only be seen when zoomed in far enough that the unit grid is visible, and it performs a check for every pixel of the map on your monitor, so it can really lag the map. The algorithm checks every pixel currently visible on your map and starting from Mario's height looking downwards, it checks to see if that position is within a squish cancel spot. Successful spots will be highlighted in red and green. If it's red, it means that squish cancel can be performed but it will hurt Mario, and green means that it can be performed

without hurting Mario. In the gear menu, "copy base points" will put into your clipboard a table containing all coordinates of found valid locations. "Copy midpoints" will put into your clipboard a table containing all midpoints of found valid locations. Midpoints being the middle of the unit square found. Selecting either of the previous two options when no points are found yet will raise an error. "Set custom height" lets you set a custom height to start looking down to find squish cancel spots. "Clear custom height" clears this. In hover mode, valid unit squares will give you their index and the coordinates of their midpoint. Right clicking will let you copy their base position or midpoint.

The "misc..." category gives a bunch of other trackers that don't fit in any of the other categories. "Add tracker for waters" will add purple rectangles showing the boundaries of all water sections of the level. In 3D camera modes, these rectangles will be drawn at the proper heights. This tracker ignores the default water level at -11000. In hover mode, you get the index of that water zone but no information on position. However, when right clicking on a zone in hover mode, you can copy the minimum and maximum X and Z, as well as the Y level of the water for that zone. The values are even displayed before you copy them. "Add tracker for hitbox tris" gives you a second copy of the collision mesh you see when in orthographic or 3D map modes. This is useful because you have more control over it, such as its opacity and line width. An example use case would be being in cross section map mode and having the second set of hitbox triangles at 0 opacity and 1 line width to make it easier to know what the cross section you're looking at represents. Because this is sort of like a custom set of triangles, the gear options and hover menus only contain stuff shared with triangle trackers and thus have been explained prior in this document. "Add tracker for aggregated path" is a tracker that combines all existing path trackers into one so that only the aggregator needs to be visible for all paths to be drawn, but the aggregator can only be viewed in top down map mode. Path trackers are explained in the next sub-chapter.

"Add tracker for compass" adds a compass to the top left of the map when in top down mode. You can change its opacity and line thickness with both sliders, but to customize the compass further, you select "open settings" in its gear menu. This will open a variable panel window containing the compass settings as variables. "Position" is where on the map the compass will be. The possible values for this are: TopLeft, TopRight, BottomLeft, BottomRight and Center. All measurements are in monitor pixels. "Line height" is the distance from the arrows to the center of the compass, "line width" is the width of the lines from the arrows to the center, "arrow height" is the height of the triangles making the tip of the arrows, "arrow width" is the width of the triangles at the tip of the arrows, "horizontal margin" and "vertical margin" are 5 less than the distance from the horizontal and vertical edges of the map screen to the tip of the compass, "direction text size" is the font size of the axis name text times 7ish, "angle text size" is the same but for the angle text, "direction text position" is the distance from the center of the triangle that the axis text should be at, "angle text position" is the same but for the angle text, "show direction text" toggle showing the names of the axes, "show angle text" toggles showing the angle an object needs to face that direction, and "angle text signed" makes the angle text range from -32768 to 32767. "Add tracker for coordinate labels" adds blue text in top down mode indicating vaguely the rows and columns of X and Z positions, and another label next to your mouse showing where you are pointing without having to be in hover mode. You can change the size of the text with the size slider, but do not set it to 5 or less, or else the map itself will crash, making it unusable and unviewable until the application is closed and reopened (except 3D mode, that’ll still work). Just like the compass, its settings are mostly in a variable window retrieved with "open settings"

in the gear menu. All values are in monitor pixels. "Custom spacing" lets you specify a custom constant frequency for how often labels should appear. Putting it at 0 will set it to its default changing value, but you can have it be always every 100 units. Setting this value to a negative number will crash STROOP. "Margin" is the distance between the center of the text labels to the edge of the screen. "Label density" is how close labels have to be to disappear when custom spacing is 0. "Use high X" toggles the Z labels being on the side with positive X values, and "use high Z" toggles the X labels being on the side with positive Z values. Yes, those variable names are confusing. "Bold text" toggles the boldness of the text.

"Add tracker for cork box tester" will place a grid of polka dots on the screen, and zooming in enough will replace the dots with more precise positions. These dots represent the predicted lifetime of a cork box placed at the highest floor found at that position. The color of the dot is the category of how long the cork box lasted before being destroyed. There's no gear menu options, but the hover menu reveals a lot more. When hovering over a dot, you get its index, its position and most importantly the result of the simulation at that location. The simulation runs until either 1001 frames have passed or the box was destroyed. The color of the dot is an indicator of that simulation's lifetime. Turquoise means the box lasted the full 1001 frames, dark red means it lasted 500 frames or less, red means it lasted 501 to 800 frames, pink means it lasted 801 to 900 frames, white means it lasted 902 to 950 frames, and cyan means it lasted 951 to 1000 frames. Interestingly, if the box's lifespan was exactly 901 frames, the dot gets colored yellow. This helps uncover the purpose of this tracker, which is to find cork box positions where it explodes automatically on the last frame of its existence, which means the cork box will both explode and respawn. Right clicking in hover mode lets you copy a dot's position. "Add tracker for bounds" adds a tracker that is simply a yellow rectangle with light blue vertices. This tracker has no gear menu options and hover mode simply gives the index and position of each vertex. You can move around the vertices in object drag mode to change the position and shape of the bounds. "Add tracker for camera view" adds a tracker for a giant yellow trapezoid matching the camera's position and angle. However, this is not a tracker for the camera's frustrum. The yellow trapezoid only represents the area that Mario must be within to be rendered in the game. This is useful for performing BCA. This tracker works in all camera modes but is inaccurate in top-down mode when the camera's pitch isn't straight forward. In the gear menu, "set radius" lets you set the height of the trapezoid.

The "custom..." category contains miscellaneous customizable trackers that give the user more control over their meaning. "Add tracker for custom positionangle" lets you add a custom positionangle tracker. A window will pop up asking you to enter the positionangle you want to track. This has the exact same syntax as the positionangle types from the TAS tab, so you can read all the possible types in that chapter. You can only set this once, changing the positionangle type will require adding a new tracker. Its options are the same as any other point/positionangle tracker. "Add tracker for line segment" is a tracker that draws a line between two positionangles of your choosing. Two windows will open one after the other for you to set first your starting point then your end point for the line. Once entered, a line will appear connecting those two positionangles. If your positionangles have the same position, no line will be drawn. The size slider determines how far past the end in units the line should continue. In the gear menu for the line, "use fixed size" means that the line will use the size slider to determine its length. It will start at the starting point then move that many units towards the end point before stopping. You can set this length to be longer than the length between the two

points to have the line go past the end. "Set backwards size" lets you input a number, and the line will be extended behind the starting point by that many units. This option ignores the fixed size option. "Show midline" will draw an infinitely long line perpendicular to the existing line at its midpoint. The midpoint takes into account both forwards and backwards extensions. "Set icon size" lets you change the size of the custom icons. Important notice: setting any position angle to "none" in the last two trackers will crash the whole map tab, making it unusable until STROOP is closed and reopened. "Add tracker for drawing" adds a tracker that at first, does nothing. However, in the gear menu, toggling "enable drawing" freezes the map in place and now holding down left click will let you draw on the map. Once done drawing, untoggled the enable drawing option, and now you can move around the map with your drawing fixed in world space. This drawing also appears in 3D, but you're only able to draw on the XZ plane at Y 0. Back in the gear menu, "clear drawing" is how you erase the drawing and "undo last stroke" deletes only your last stroke. You can keep clicking undo indefinitely until the whole drawing is gone.

The "preset..." category contains two presets of map tracker collections. These are existing map trackers all added at once with specific options and parameters pre-configured. "Add preset for walls, floors, ceilings" will add trackers for wall, floor and ceiling collisions. This preset includes a level floors tracker named "floors" with show tri units enabled and within distance set to 500 units, a level ceilings tracker named "ceilings" with show tri units enabled and within distance set to 500 units, and 5 level walls trackers, 2 for the upper and lower hitboxes of grounded collision, 2 for the upper and lower hitboxes of air collision, and one for water collision. The upper ground walls have their relative height set to -60, the lower ground walls have their size set to 24, their color set to cyan and their relative height set to -30. All non-ground wall trackers are set to be invisible by default, with the assumption that you switch which are visible depending on Mario's state. The upper air walls have their color set to cyan and their relative height set to -150. The lower air walls have their relative height set to -30. Finally, the water walls have their size set to 110 and their relative height set to 10. After that, all trackers have their opacity set to 20, their order type set to order on bottom, auto update set to true and include object tris set to true. This preset adjusts the hitboxes to be accurate to all possibilities, as the regular wall tracker only accounts for the general case. "Add preset for movement" will add trackers to aid with Mario's movement. The first two trackers are named "floor snap up" and "floor snap down". They're both level floor trackers with their order set to order on bottom, show tri units set to true and include object tris set to true. Floor snap up has its color set to dark cyan, its minimum height set to Mario's position and its maximum height set to 78 units above Mario's position. Floor snap down has its color set to orange, its minimum height set to 100 units below Mario's position and its maximum height set to 0.001 units below Mario's position. These represent the area that Mario will receive a floor snap up and snap down respectively. Along with those, a next positions tracker is added with no changes, three Mario facing direction arrow trackers are added with their size set to 2000 and line width set to 2, with the second one also having its color set to red and angle offset set to 32768, and the third one actually being a sliding direction arrow and having its color set to magenta. Arrow trackers will be covered in the next sub-chapter. Finally, a PU gridlines tracker is added with its line width set to 2 and mode set to setting 3.

However if you remember, right clicking the clear button right next to the add button also gives some more presets. "Surface triangles white" sets the map background to pure white, the level to transparent, toggles the unit gridlines tracker checkbox and adds two additional trackers, level floor

tris and level wall tris. "Surface triangles black" is the exact same thing as the previous one, but the background is black instead of white. "Enable TASer settings" first clears all trackers, then adds a Mario tracker with its size set to 15, a ghost tracker with its size set to 15, a previous positions tracker with its size set to 10, a unit gridlines tracker with its order type set to order on bottom, a current floor triangle tracker with its opacity set to 8 and its order type set to order on bottom, and finally, a level walls tracker with its relative height set to -30 and its order type set to order on bottom.

The final option in the add button menu is "add map pop out", which opens a new window containing a live copy of the map and nothing else. The map will have its mode be whatever the current map mode is and stick with it forever. Unless the map mode is 3D, in which case the pop out map will ignore that checkbox. The pop out map will also react live to the cross section mode being activated if it is in orthographic mode. Hover mode does not work and double clicking to reset the map position and angle doesn't work in the pop out map. This pop out makes the map more convenient to access, especially when working with another tab in STROOP.

18.5: The plus button

Oh yeah, there's more. Of course there's more. On each map tracker in the list is a green plus icon that opens a menu for many more trackers. All of these trackers are here because they're attached to that tracker like a parent. Although, the child tracker is able to stay in the list and function correctly even if the tracker used to create it leaves. Almost all of these additional trackers are intended to be added to object or position trackers, usually even specific types of those. Attempting to add these trackers onto trackers they don't work with can produce wildly varying results. Depending on the combination, it can function surprisingly correctly, function in a broken state, not add the tracker at all or even crash the map tab or STROOP if you're unlucky. This document will describe all of these trackers when added using a tracker they were meant to be attached to, usually Mario or another game object.

The first category is the "cylinder..." category, which covers different types of cylinders representing aspects of the object, usually collision. All of these cylinders have the same gear and hover options as the cylinder trackers mentioned in the previous sub-chapter. "Add tracker for hitbox cylinder" adds a tracker for that object's hitbox using the dimensions specified in its memory. This is the cylinder used for general interactions between objects or Mario. "Add tracker for effective hitbox" is just like the previous tracker, but the cylinders have their radius and height extended by the radius and height of Mario's hitbox. This means that whilst the hitbox trackers represents collision when two hitboxes collide, effective hitbox represents collision when Mario's position enters it instead of his hitbox. Effective hitboxes also have an additional gear menu option called "use interaction status as color", which colors the cylinder red if there's currently no interaction and cyan if there is. Interaction being defined as Mario's interaction pointer being set to that object, and not a hitbox collision. "Add tracker for hurtbox cylinder" and "add tracker for effective hurtbox cylinder" do the exact same thing as the previous two trackers, but for the object's hurtbox instead of the hitbox. This is the hitbox used for determining if the object should hurt Mario. "Add tracker for effective hitbox/hurtbox cylinder" combines the previous two effective hitboxes under one tracker that draws two cylinders. This also has the use interaction status as color gear menu option. "Add tracker for push hitbox cylinder" is a tracker for objects that push Mario outwards when he tries to get near them, like cork boxes or chests. They're colored orange and show where Mario's position has to be to be affected. Last is "add tracker

for custom cylinder", which lets you attach a cylinder to something while also specifying its properties. By default, the cylinder is 1000 units in radius, and ranges vertically from the Y position of its parent to 100 units above that. You can change its radius with the size slider and its top and bottom relative to its parent with "set relative min Y" and "set relative max Y" in the gear menu.

The "sphere..." category covers different types of spheres representing aspects of the object, usually how close you have to be to it for something to happen. These trackers have the same gear and hover options as the sphere trackers mentioned in the previous sub-chapter. "Add tracker for tangibility sphere" adds a tracker for a sphere that Mario has to be within for the object to have its collision be active. This sphere is usually rather small for objects, which is why you have to be so close to them to see their collision triangles in STROOP. For some objects like the bullet bill blaster, this radius is so small the collision can unload while Mario is standing on the object. "Add tracker for draw distance sphere" adds a tracker for a sphere that Mario has to be within for the object to be rendered. Some objects like enemies and signs turn invisible if Mario is far enough away from them. This sphere shows exactly what the rendering radius is. "Add tracker for custom sphere" adds a tracker for a sphere centered at the parent tracker. You can change the radius of this sphere in units by changing the size slider.

The "home..." category tracks various aspects of an object's home position. The home can be used for many thing depending on the object, or it can also be completely unused. "Add tracker for home" adds a pink house icon on the object's home position. Object homes are usually static but can also move like in the case of scuttlebugs. The hover menus options are the same as any position tracker. "Add tracker for custom cylinder for home" is a custom cylinder tracker centered at the object's home instead of its position. Some things happen when a certain distance from an object's home, so this is for tracking that. However you have to provide the radius, relative min Y and relative max Y yourself with the size slider and gear menu. "Add tracker for custom sphere for home" is the same as the previous tracker, but as a sphere instead of a cylinder. It's centered on the object's home, and you set the radius in units with the slider.

The "triangles..." category lets you view the triangles of an object. Mostly the collision ones again. "Add tracker for floor tris", "add tracker for wall tris" and "add tracker for ceiling tris" are just like their earlier all object counterparts from the previous sub-chapter, but these focus on a specific object and are also missing the use current cell tris option from the gear menu. Everything else is the same. "Add tracker for gfx tris" adds a trackers for the graphical triangles of the object. Not the collision triangles, actually the visible triangles this time, including the shadow beneath it if there is one. However, this tracker is only usable if the "object graphics triangles" hack is enabled in the hacks tab. Without it, STROOP has no way of accessing the data and storing it to display on the map. If both the hack is active and the object is visible, select "use this for gfx tris" in the gear menu. This sets up the object to be the one whose graphical triangles show up. The object's graphical triangles should then appear on the map where the object is as yellow triangles. These are not visible in the 3D map mode. You can only view a single object's graphical triangles at a time, and you select which is active by selecting "use this for gfx tris" to set STROOP's hacked memory to which object should be checked. If the hack "object graphics triangles cam POV" is enabled, the map graphics triangles will use their view matrix as their world matrix, making them appear near the map's origin, as if the camera was exactly at the origin looking forward. So, make sure that it's not enabled, unless that's what you want for some reason. If the graphics triangles ever disappear from the map and can't be found at the origin, try re-

selecting the option in the gear menu to select that object's graphical triangles and advance the game by at least one frame.

The "arrows..." category contains various arrows that display aspects of an object as a vector in XZ space. Arrows can have their length changed with the size slider, which is 300 by default. All arrow trackers come with five options by default in the gear menu. "Use recommended arrow size" will change the size of the arrow based on what it's tracking. Usually, it'll be the magnitude of the vector. "Use truncate arrow" will convert the angle of the arrows to in-game angle units if the angle is stored as a float within STROOP. This is enabled by default. "Set arrow head side length" sets the length of the two lines poking out the end of the arrow to from its head. By default it's always 100 units long, but this can be too long for shorter arrows. There is no way to have the size of the lines correlate to the length of the arrow. "Set angle offset" will add an additional rotation to that arrow. The inputted number in the window will be interpreted as in-game angle units and that many will be added to the arrow. For example, adding 32768 would inverse the direction of the arrow. "Use pitch" will account for the pitch angle of the object to shorten the arrow's magnitude. When in hover mode, you can hover over arrows by specifically placing your cursor over their tip, where all three lines meet. You will see their position at the top of the screen, and when right clicked, you will be able to copy the position of the arrow's head. When object dragging is enabled, you can grab the head of the arrow where the lines meet to drag around its position. This also has the effect of setting the value of whatever it's tracking, including the thing it uses for its recommended distance if that's enabled. When dragging an arrow, you can hold control to lock the magnitude of the arrow and hold shift to lock the angle of the arrow.

"Add tracker for Mario facing arrow" adds an arrow representing Mario's facing yaw. Activating its recommended size sets its magnitude to Mario's horizontal speed. "Add tracker for Mario moving arrow" will add an arrow for Mario's moving yaw. Activating its recommended size sets its magnitude to Mario's horizontal speed. "Add tracker for Mario intended arrow" will add an arrow for Mario's intended yaw. Activating its recommended size sets its magnitude to Mario's horizontal speed. "Add tracker for Mario speed arrow" will add an arrow for Mario's X and Z speeds represented as a vector. Activating its recommended size sets its magnitude to Mario's horizontal speed. "Add tracker for Mario sliding arrow" will add an arrow for Mario's sliding X and Z speeds represented as a vector. Activating its recommended size sets its magnitude to Mario's horizontal sliding speed. "Add tracker for Mario twirl arrow" will add an arrow showing Mario's current twirling angle, that is the angle his model looks towards during his twirling action. Activating its recommended size sets its magnitude to Mario's horizontal speed. "Add tracker for Mario floor arrow" will add an arrow facing in the direction that would be immediately downwards for the floor that Mario is currently above. Activating its recommended size sets its magnitude to Mario's horizontal speed.

"Add tracker for object facing arrow", "add tracker for object moving arrow" and "add tracker for object speed arrow" are the exact same as the Mario equivalents seen above, but meant for use with any object. "Add tracker for object graphics arrow" will add an arrow representing the yaw used by the object's 3D model. Its recommended size is the object's horizontal speed. "Add tracker object angle to Mario arrow" adds an arrow that points towards Mario's position. If the arrow's length is longer than the distance between the object and Mario, the arrow will simply keep going past Mario. The recommended size option for this tracker uses the object's built-in variable for its distance to Mario, but not all objects initialize this value, so the arrow can end up being something like 20000 units long on some objects. "Add tracker for object target arrow" first scans all of the object's STROOP variables

for any containing the word target (with a capital T) in the name. This isn't a default variable, so only some objects such as goombas have a target variable. If none are found, the arrow will just point towards 0. If one or more are found, the first one found will be what the arrow tracks. And yes, this does mean that you can trick the code by renaming one of the existing variables to target. The recommended arrow size option sets the magnitude to the object's horizontal speed. "Add tracker for object custom arrow" lets you add your own custom arrow for the object. However, it's not done by setting a custom angle and magnitude. Two pop up windows will appear one after the other asking you first the offset of the tracked angle and its size in bytes. This is because custom arrow means you get to pick which variable in the object's memory is getting tracked. For this, you'll need to know the offset of the variable you want to track. You can get this in the object tab by double clicking the variable, it will appear in the info windows as "Object + 0x##", with that ## part being what you want to enter in the first window. For the size, it seems as if you can enter any number here, but STROOP interprets any number that isn't 2 as 4, so you just need to check to see if the value you want to track is a short or an int. Once this is done, you're able to track any data in the object struct, or even past that if you're feeling funny. The recommended size option for this arrow is the object's horizontal speed.

"Add tracker for swooper effective target arrow" is a preset custom arrow for a swooper's unique effective target yaw variable, indicating in which way it's really moving as it wiggles back and forth in a direction. The recommended size option sets the magnitude to the swooper's horizontal speed. "Add tracker for scuttlebug lunging arrow" is a custom arrow for a scuttlebug's lunging direction. However, a scuttlebug doesn't have a lunge direction variable, so this just reads the scuttlebug's facing yaw. The difference between this arrow and the facing yaw arrow is that this one has a custom recommend yaw. If that option is activated, the arrow's magnitude will be set to the time remaining of the scuttlebug's lunge timer multiplied by the value on the size slider. "Add tracker for custom positionangle arrow" will let you enter a positionangle using the same syntax as in the TAS tab. The angle of this position angle will be the angle used for this arrow. This arrow has no recommended size and just uses the size slider value. Warning: entering the "none" positionangle crashes the map tab.

The "misc..." category contains various different trackers that can be attached to objects. "Add tracker for current unit" tracks with a pink 1 by 1 unit square the exact unit that the object is currently located in. In hover mode, you can see and copy the position of this unit. "Add tracker for current cell" tracks the current cell that the tracker is checking for collision (or not) in with a yellow square. In hover mode, you can see the X and Z range of the cell and copy either the minimum or maximum X or Z of the cell. "Add tracker for angle range" draws a line starting from the parent object outwards for every single possible angle, by default with jumps of 16 angle units. The size slider determines the length of these lines. In the gear menu, "use relative angles" will have the lines start at the parent tracker's angle instead of 0. "Set angle diff" is the jump in angle units between each line. "Use in-game angles" will override the two previous options and use in-game trigonometry logic to calculate the lines, and in the process also set the diff value back to 16. "Add tracker for sector" adds a sector of a circle around the parent tracker, with the sector pointing in the parent tracker's angle. The size slider controls the radius, which is 1000 by default. By default, the sector goes 4096 angle units on each side of the facing angle, but this can be changed in the gear menu with "set angle radius". It's useful for objects that only do something if their target is within a certain angle of their facing direction. This tracker is only visible in the top-down map mode. "Add tracker for facing divider" will add a red line that goes through the

parent tracker at an angle perpendicular to the parent tracker's angle. You can change the length of the line with the size slider, and its length will be that value but for each side of the center. By default, it's 1000 units on each side. "Add tracker for home line" will draw a line from the parent tracker to its home. "Add tracker for render table" will draw a giant table on the map in top-down mode. You cannot change the size, opacity or position of this table. The table shows a 2 by 2 grid displaying whether the object it's tracking is visible or not and if it's active or not. They are 2 different flags in an object that both make it invisible to the player, but they have key differences, so it's important to know the difference. However, the static size and position of the table indicate that it was used by Pannen for recording a video, not for general use. Activating the scales checkbox on the tracker will show a second arrow on the left, to differentiate the object blinking from regular invisibility. In the gear menu, you can set images to fill in the squares. "Select rendered object image" will be put in the top left square and "select not rendered object image" will be put in the remaining squares. There is no way to change the size of these images in the tracker.

"Add tracker for path" will start recording on every frame the position of its parent tracker and draw a line showing its position over time. It starts recording data from the moment it's added and stores the position of its parent on every new global timer value. The line is drawn from every value to the next, so sudden jumps or teleports will draw a straight line across the map. By default, the line has its color very by how long ago the parent tracker passed over that spot. Fresh data is red, and slowly fades to yellow over time. These colors can of course be changed with the color picker. This line can be viewed in any camera mode. The controls for the line are found in its gear menu. "Reset path" will clear all line data, erasing what's currently on the map. Toggling "reset path on level change" will reset the line whenever the current area changes. Passing through a seamless loading zone or loading a savestate will both reset the line when this is on. "Use blending" toggles the color gradient on the line. Changing this tracker's size slider also indicates the span of that gradient in frames of data before it just sticks to yellow. "Pause" pauses the data recording. Toggle it back off to resume the recording. "Truncate points" toggles truncating all of the values to integer coordinates when displaying them on the map. "Use value at start of global timer" is toggled on by default and makes it so that values written to the data table for the latest global timer value cannot be overwritten. So by default if you were to pause the emulator then change Mario's position in STROOP, the line would stay where Mario was. With this option off, the line would instead go to Mario's new position. "Set modulo" is for skipping values in the data when drawing the line to the map. By default every point is used, by if say you wanted to use every 12th point of data, you'd enter 12 in the pop-up window. "Deduplicate points" does not add values to the list if they have the same position as the last values added. This creates holes in the data, but it's better for memory usage when only using the path visually. "Show quarter steps" is used when you have a custom icon set for the path, which puts the icon on every point of data. Toggling this on will add 3 smaller icons with interpolated positions between each point of data. Toggling on icons is also how you access the hover data for a path, each point with a custom image will give you its index, position, and right clicking it will let you copy that position. "Set icon size" lets you set the custom icon size, with quarter step icons being half of the regular size. "Copy points" will put into your clipboard a table with the global timer, X position, Y position and Z position as columns. Clicking this option while holding control will also copy quarter step positions by interpolating them, although this incorrectly changes the global timer on every quarter step instead of every full step. You can use "paste points" to paste these values in later, or go back to the TAS tab to paste this in as a schedule without an angle. By doing that, you can then set up a positionangle tracker to follow the path in real

time. "Add tracker for branch path" is a path tracker that acts differently. First off, it only has three options in the gear menu, those being reset path, pause and use blending, all of which are already covered in the path tracker explanation. The key difference with this tracker is its behavior when attempting to record data for a global timer value that already contains data. Instead of overwriting the data and everything that comes after it, the branch path tracker will instead show the existing path be drawn in real time using the global timer, and as well record any new data you might be forming. You can repeat this as many times as needed to draw as many branches either using savestates or editing the global timer value. This tracker does not support having a custom icon like the previous one. If you notice gaps forming in your paths, it's because STROOP is running too slowly to catch the data in time. A quick fix for this is to go in the options tab and change STROOP's FPS to be much higher. I strongly recommend not keeping this on however, as STROOP running continuously puts unnecessary strain on your CPU usage.

"Add tracker for offset positionangle" will create a tracker that's a positionangle with a horizontal offset from its parent. The size slider controls the magnitude of the offset, and the line slider truncated to a multiple of 16 becomes the angle offset. The rotates flag becomes whether the offset angle is absolute or relative to in front of its parent tracker. In the gear menu you can change the icon size and in hover mode you can see its position, and copy, paste or go to it with a right click. When dragging this tracker around, you can hold control to keep its distance from its parent tracker constant and hold shift to keep its angle from its parent tracker constant.

The "object specific..." category is for trackers meant to be used only on one kind of object. "Add tracker for coffin box" is a tracker for the moving coffins in BBH. It marks the area Mario has to enter for them to rise up. The tracker is shown as a rectangle and not a box, even though the coffin's trigger box has a definitive height. "Add tracker for fly guy zone dividers" will draw 2 giant horizontal lines in orthographic mode. They represent the dividers for the three zones that affect a fly guy's AI. Because these are meant to cut the Y axis, they only make sense when viewing the orthographic map from its default head-on angle. Mario being in either the low, middle or high zone is what decides fly guy's action, so it's crucial for fly guy manipulation. "Add tracker for pyramid platform normal" is a tracker for the tiling upside down pyramid near the start of bowser in the fire sea. The shapes represent places Mario has to be in to tilt the pyramid in a certain way. There are always at most 3 pink rings. The middle one represents the positions Mario would need to be in to have the pyramid's Y normal be the same as it is now. The inner ring is for positions Mario would need to be in to have the pyramid have the Y normal be 0.01 larger, and the outer ring is for having it be 0.01 smaller. The red and green parabola triplets offer the positions Mario would need to be to rotate the normal in either the X or Z directions as fast as possible. With the middle line being for keeping the current X or Z normal value and the outer ones for deviating it by 0.01 in either direction. The green parabolas are for the X axis and red ones for the Z axis. You can additionally set a custom normal value in the gear menu to use that instead of the pyramid's current normal. When using a set normal, only the center lines are drawn. "Add tracker for pyramid normal" and "add tracker for pyramid normal target" represent as Mario icons the current and target normal as points above the pyramid's flat top in the direction of its normal. The first tracker will show its current normal with an orange Mario and the second will represent the target normal with a blue Mario. Both of these can be interacted with in hover mode to get the address of the pyramid object, their position, and in the right click menu, options for copying and pasting that position.

The "preset..." category offers collections of trackers to add all at once with pre-set settings configured for a specific purpose. "Add trackers for chuckya map objects" adds a set of trackers that help with manipulating a chuckya's AI. It's meant to be applied onto a chuckya object. This preset will add a tracker for the chuckya's home, an arrow tracker for the chuckya's facing direction set to be green and 3000 units long, an effective hitbox cylinder tracker, a sector tracker with a new size of 187 and angle of 10912 on each side, another sector tracker with a size of 3000, a facing divider tracker with a size of 3000, a draw distance sphere tracker, a custom cylinder tracker set to be cyan with a radius of 1900 and a relative minimum Y of -5000, another custom cylinder with a radius of 500 and one last custom cylinder with a radius of 2000. All of this accounts for most if not all of the checks the chuckya AI performs in its routine. "Add trackers for fly guy map objects" adds a set of trackers for manipulating fly guys. This preset will add a cylinder tracker for the fly guy's effective hurtbox, a moving direction arrow tracker with a length of 10000 and a line width of 2, a home line tracker with a line width of 2, a custom sphere tracker with a radius of 400, 20 percent opacity and cyan color, another custom sphere tracker a radius of 2000 opacity of 20 percent and green color, two draw distance sphere trackers, both with 20 percent opacity one red one yellow and the yellow one has use cross section set to false, another custom sphere tracker centered on the fly guy's home with a radius of 2000, 20 percent opacity and pink color, and finally a fly guy zone divider tracker.

Thus ends the list of every tracker in the map tab.

18.6: Controllers & data

The map tab has its own tabs, hidden above the options. By default, you're put in the options sub-tab, which is where most of what you will ever need is. However, there are more tabs containing additional information, controls and settings.

The "controllers" sub-tab gives you controllers for the map camera in both the top-down and orthographic map modes. It's divided into 3 sections. "Scale" is for how zoomed in the camera is, "center" is for the position of the camera laterally and "angle" for the camera's yaw rotation and also pitch rotation when in orthographic mode. In the scale section, the radio buttons let you select a scale preset. "Course default" sets the scale to a value associated with that course to have it fill in the map as well as possible while keeping everything in. "Max course size" will force the entire main map to fit within the map even if it contains a large section without geometry. "Custom" is for anything else. The value next to the custom option is the current map scale. A scale of 1 means that every pixel on your monitor corresponds to 1 unit in the game. In general, that number is an approximation of how many pixels on your screen it takes to correspond to 1 in-game unit. The controllers on the right with the math symbols lets you add or subtract a custom value from the scale, and multiply or divide the scale by another custom number. Right clicking anywhere in the scale area lets you select more scale presets. They're all very zoomed in, with varying sizes for the unit grid. "Very small unit squares" sets the scale to 6, "small unit squares" sets the scale to 12, "medium unit squares" sets the scale to 18, "big unit squares" sets the scale to 24 and "very big unit squares" sets the scale to 40.

In the center section, you can select from more preset options. "Best fit" centers the map for the visuals to fit neatly on screen, "origin" puts the origin point of the map in the center and "Mario" centers the map on Mario. This last one updates continuously, meaning that if left on this mode, the map will continue to follow Mario. Custom is for everything else, and the number side it is 3D coordinates in the world for the center of the map separated with semi-colons. In top-down mode,

the Y value here doesn't do anything, but in orthographic mode this 3D point becomes the center of rotation. The controller to the right lets you move the camera in 3D relative to its current facing direction. The up and down arrows move the camera up and down, left and right move it left and right, the diagonal arrows are a mix of both (always one after the other, even if the option in the options tab to not do this is enabled), and the buttons with an X in the middle of a circle move the camera immediately forwards and backwards. The one on the top with a bigger X is forward. The customizable number in the middle is the number of monitor pixels you're moving the map by. "Change by pixels" determines whether the controller will move the map in monitor pixels or in-game units. It's pixels by default. "Use Mario depth" is also enabled by default and makes the current depth value of the map center be at Mario's position relatively. For example, in top-down mode the camera Y value will be Mario's Y value. Turning this off also makes rotating in the orthographic mode less extreme. If you right click anywhere in this section, you get extra options. "Center on Mario" will move the center of the camera to Mario's position, but unlike the option on the left, not continuously. Only once. The three remaining options "can drag horizontally and vertically", "can only drag horizontally" and "can only drag vertically" act like radio buttons and allow you to lock the camera movement in either the up/down or left/right directions.

In the angle section, you can choose from the radio buttons which cardinal direction is up. By default, it's 32768, but you can set it to either of the three other cardinal directions. Other than those, you get more dynamic options. "Mario" will set the map's up direction to Mario's facing angle, "camera" will set the map's up direction the camera's yaw (map camera's pitch is not affected), "centripetal" will use the camera's centripetal angle as seen in the camera tab, and "Mario side" will have the map's up direction be Mario's left. Custom contains a set of two values separated with a semi-colon, the map camera's yaw and pitch. The pitch is only used in orthogonal mode. The controller on the right lets you rotate the camera with a customizable value. The left and right rotating arrows rotate the map image clockwise and counterclockwise, and the up and down arrows pitch the camera up and down. When right clicking in this section, you get more options. "Use Mario yaw", "use camera yaw" and "use centripetal yaw" will set the yaw of the camera exactly once to match their respective angle. "Use 0 pitch" sets the pitch to 0. The last three options are named the same as in the center section, but here horizontally represents the camera's yaw and vertically represents the camera's pitch.

In the "data" sub-tab, you get a brief summary of Mario's general location. At the top in bold you have the name of the current level or place. The text below it doesn't always appear, but when it does it makes the location more precise, like saying that you're specifically in the volcano of LLL. It can also make remarks on the state of the level, like specifying that you're in the version of WF with the tower there. It can also say which floor of a level you're on. It's not a 1:1 correlation with background maps, but it's close. "PU [X:Y:Z]" says Mario's current PU in the X, Y and Z axes. "QPU [X:Y:Z]" does the same for Mario's QPU. "Location" retrieves 4 ID values directly from memory for more info on where Mario is. The first two get the ID for the level and area from the sWarpDest struct in decomp, the third gets the area ID from the gCurrAreaIndex variable, and the fourth is an ID for the mission layout from gCurrActNum. "Floor Y norm" is the Y normal of the floor below Mario.

Skipping the vars sub-tab (for the next sub-chapter) to the "3D controllers" sub-tab, here we have the controllers for the full 3D camera mode. Here, you simply have 5 different 3D controllers and an FOV slider. The first four 3D controllers are also found in the camera tab and perform the same action as there, but on the map camera instead of the game's camera. "Camera & focus" is for moving the

camera's position and its focus' position at the exact same time. The FOV slider simply changes the field of view of the 3D camera. Some of the controllers could not be functioning, read about Mode in the next sub-chapter to learn why. By default it's 45, which is the same as in the game, but this can be changed if you want a more zoomed in experience or you're a quake pro.

18.7: Variables

Finally, the "vars" sub-tab. This contains all of the advanced map options, laid out as STROOP variables. The first group of variables consist of all of the ones highlighted in green, white and sometimes red. These are advanced controls for the camera in the map's 3D mode. "Mode" sets the current mode for specifically the 3D map mode and its camera. The two defaults are "ingame" and "cameraposandangle". In-game means that the camera will follow the in-game camera move for move, including the FOV. It's what's set when you first open the 3D map mode or double click while in it. Camera pos and angle is what it's set to the moment you move the camera around with the mouse. This mode determines the camera's position and rotation through controlling its position and angles with your mouse. Other modes that exist are "cameraposandfocus", which determines the camera's position with two positions, one for the camera and another for its focus, "followfocusrelativeangle" will have the camera follow the focus from a set distance and a relative angle, and "followfocusabsoluteangle" will have the camera follow the focus from a set position and set angle. Most of these won't do anything interesting when first set, and that's because they need to be controlled with the variables below. Depending on the camera mode, the ones used to control the current mode will be highlighted in red. For example, the in-game mode only uses variables from memory, so none of the variables are highlighted. Modifying white variables will sometimes do something, sometimes not. Camera X, Y, Z are the camera's position, camera yaw, pitch, roll are its angle. Roll is not modifiable outside of this variables, so if you want a non-0 camera roll, set it here. Focus X, Y, Z are the focus' position. FOV down at the bottom is the map camera's FOV. The rest of the variables are for controlling the other non-standard camera modes. All of the "PA" variables let you enter a positionangle type to each of the position and yaw angles for both the camera and focus. The camera and focus will both update their position and angle according to these position angles in real time. For the following modes, you have three "following" variables. "Following radius" is the distance the map camera will be from its focus, "following Y offset" is the value the camera will add onto the focus' Y position to determine its own and "following yaw" is the angle added onto the positionangle for the focus' angle. In absolute angle mode, this variable will be the yaw angle the camera constantly looks towards.

The variables highlighted in blue are simply the values seen in the controllers sub-tab, but with rounding applied to them by default. The grey variables are the additional options for the map tab. "2D zoom speed" is the multiplier applied to the current scale value for every tick of your mouse wheel. This is 1.1 by default. "Orth H rotate speed" and "orth V rotate speed" is how fast you can rotate the maps in orthographic mode. The value is the angle rotated for every pixel your mouse moves. "3D zoom speed" is the number of units forwards the camera moves in 3D mode when you scroll one mouse wheel tick. "3D translate speed" is the number of units the camera moves sideways when moving the mouse by 1 pixel in 3D mode. "3D rotate speed" is the number of angle units the camera rotates when moving the mouse by 1 pixel in 3D mode. "Num circle points" is the number of vertices used when drawing a circle. "Num sphere points" is the number of vertices used around the equator of a sphere drawn in 3D mode. These let you balance accuracy and performance. "Unit precision

threshold" is how far zoomed in you have to be to have the unit grid appear. Your scale value has to be larger than the value of this variable. "Sort orthographic tris" will sort the triangles in orthographic mode before rendering them to the screen. This makes it so that triangles closer to the camera will appear on top of ones that are further. The render sorting algorithm used by STROOP isn't perfect, but it's good enough for the use case. "Use not for ceilings" and "use X for ceilings" change what symbol is used when looking at ceilings in cross section mode will arrows turned on. Instead of using arrows like walls and floors, ceilings by default use a not symbol to show how Mario doesn't get pushed by them, but rather his quarter step gets cancelled completely. By unchecking the first of the two options, you can switch this back to being arrows. Or by switching on the second option, the ceilings will display an X. The first option overrides the second, so activating both will show not symbols.

"Allow keyboard controls" activates being able to use the keyboard to move the map camera in top down mode and orthographic mode. Keyboard controls read your keyboard for inputs even when STROOP's window is not in focus, so be careful. The controls are that you use the arrow keys to move around laterally, E and D to zoom in and out and S and F to rotate clockwise and counterclockwise. And yes, that does mean STROOP uses not WASD but the keys one over to the right. Due to them not working in 3D mode, this means you can move the camera for the top-down or orthographic maps even when they're hidden due to the 3D map being shown. The last 3 variables let you pick custom values for the translation, rotation and zoom speed of the keyboard controls. These values work just like the speed values above.

Chapter 19: Memory tab

The memory tab is a hex viewer for the entirety of the N64's memory, displayed as raw values. At the top are the controls for the tab, and below you have 3 sections. On the left is an indicator for the addresses to the right, and this is where you have the values. Annoyingly, these two halves are not linked together, so scrolling down one doesn't scroll down the other, rendering the left bar useless most of the time. Further to the right, you have this blank section which is actually a variable panel. There are no variables here by default.

N64 memory is mapped from 0x80000000 to 0x803FFFFF. Attempting to view memory outside this region will fail. Because SM64 doesn't use the expansion pak, the bonus memory mapped from 0x80400000 to 0x807FFFFF cannot be viewed by default. However, there exists an option in the options tab to change this.

Clicking on an object slot with this tab open will by default open the object in this tab rather than the object tab. Opening an object here will automatically set the base address and memory size to the address of the object and its size.

At the top you have "base address", which is the address of the first value in the list below. You can right click the text to get a menu to bookmark addresses. You can add bookmarks to go back to an address later, you can delete those bookmarks and you can select which bookmark you want to go to. "Memory size" is the amount of data visible to you at once. The larger this value is, the longer it will take to refresh the values. "Little endian" is whether to display the data as little endian or big endian. Note that STROOP acts weirdly with endianness for 16bit and 32bit types. For short and ushort it switches odd and even values around instead of the bytes themselves, and for int and uint it just doesn't do anything. "Rel addresses" will index the addresses on the left bar from 0 when checked, displaying the address relative to the base address found at the top. This option is checked by default, and unchecking it will display the complete address of the value directly to the right of it.

"Update continuously" will update the values live when checked. This can make it hard to select or view specific data, so unchecking this "freezes" the values. However, this also freezes every other control. The only way to update the visuals with the option unchecked is either to modify the little endian checkbox or change the data type.

"Copy object" and "paste object" work exactly like the options found in the right click menu of object slots. This means they also don't modify values that could cause the game to crash if disturbed. However, these buttons do not check to make sure the data you're copying or pasting is valid object data. Meaning if you copy data from a non-object address and paste it onto an existing object, you're going to crash the game instantly. Right clicking the paste object button will bring up a right click menu with two options. "Paste object without primary variables" is the default, and "paste object without secondary variables" will not overwrite the object's position, home position, facing angles and moving angles. "Use obj address" is only used for variables, so it'll be covered later in the chapter. Clicking paste object with control held, valid data copied in this tab and selected object slots will automatically paste the copied data into every selected object slot.

"Highlight object vars" will highlight all data used by objects. A red highlight indicates it's part of an object variable, and a green highlight indicates that it's part of an object-specific variable. The selection box here lets you choose the format to display the data in. By default it shows everything as

signed shorts, but you can choose to show it as bytes, signed bytes, unsigned shorts, integers, unsigned integers, single precision floats and double precision floats. However no matter what is chosen, the width of each row will always be exactly 16 bytes worth of data. The "hex" checkmark lets you view the data as hexadecimal rather than decimal. The "obj" checkmark displays the data as the object slot the data is pointing to. This turns all non-object slot pointers to dots, and this only works when "uint" is selected in the type selection box. Any other type will not show anything.

The arrows on the right of the top panel are for moving the data window forwards and backwards by 16 bytes. The amount of data in the panel will not change, but it will start 16 bytes earlier or later and end 16 bytes earlier or later. The only difference between the white and black arrows are that for the white arrows you have to click every time you want to move, but for the black arrows you can hold them down to move at a steady pace.

19.1: Variables

The variable panel is slightly hidden on this tab, but it has its own features to connect it with the stuff on the left. To add a variable to the variable panel, you simply click on any of the values while holding control. All data that contains memory represented in the corresponding variable panel will be highlighted in blue. The variable will be given a default name simply stating the type it was when it was added, and its memory address. If rel address is checked, then the address will be the one relative to the start of the viewed memory block, and it will stay that even if the memory block moves around.

However, if you hold control and alt when clicking a variable, it will be given the name of the variable it represents for that object. This will only work if you click data highlighted in red. If the selected value contained multiple variables, they will all be added at once. If the selected value contained only a portion of a variable, then all the other values corresponding to that variable will be highlighted as well.

If the "use obj address" checkbox is checked, then added variables will set their base address to the start of the object’s address, with the offset indicating where they are instead of simply being a relative address into memory as a whole. However, due to the way the code is structured, retrieving the name of a variable with alt will always give the variable an offset, even when use obj address is unchecked.

If you right click the list of types just above the hex and obj checkboxes, an option will appear called "add all vars". Clicking this will add every single variable of the selected object as a variable. Unfortunately, this can't be combined with holding alt to find the variable names within the object. So doing this will only add as many variables as it takes to reach 0x260 bytes, depending on the currently selected type. If the selected type is rather small, doing this can lag STROOP for a while.

When adding a variable with control, you can hold additional keyboard keys to set the type of the variable. A makes it an angle, B makes it a boolean, Q makes it an object slot and T makes it a triangle.

Chapter 20: Options tab

This is where you get a complete overview of STROOP's settings. Many of these can be accessed through the cog at the top of the application, but this tab contains all of those and then some. This is also the final tab that's visible by default when you first open STROOP. Every chapter after this one will cover a tab that's hidden by default, and thus less likely to be useful.

"Use night mode" will activate night mode but STROOP's night mode implementation is so bad you're going to want to not use it. "Display yaw angles as unsigned" will display yaw angles as unsigned shorts instead of signed shorts, meaning they range from 0 to 65535 instead of -32768 to 32767. "Variable values flush right" is for whether you want variable values to be aligned to the right side of their box or the left side of their box. "Start slot from index 1" makes the object slot list start at index 1. Turning this off will start from index 0 instead. "Offset goto/retrieve functons" makes it so using goto and retrieve on objects has no offset. By default, going to an object will place Mario 300 units above the object, and retrieving it will place it 300 units above Mario. This turns that to 0.

"PU controller moves camera" this refers to the hidden PU tab. By default the camera gets moved along with Mario when moving with the PU controller but unchecking this disables that. "Scale diagonal position controller buttons" makes it so moving something diagonally with a 3D controller moves it by the distance written in the middle in both axes one after the other instead of moving that amount diagonally. For example, with the option checked, moving 100 units diagonally in X+Z+ direction would add around 70.7 units to both the X and Z axes, because the hypotenuse would be 100 units. With the option unchecked, it would add 100 to each instead, even if that means the total distance covered is around 141 units.

"Exclude dust from closest object" prevents dust particles from being given the label of closest object to Mario in the object list. Dust being close to Mario is trivial, so it's on by default. "Use misalignment offset for distance to line" will make it so by default, calculations that involve checking the distance from a point to a line will be off by 1 in certain scenarios to account for misalignments. "Don't round values to 0" will never round floating point values down to 0, even if the precision would be greater than the desired precision set in the variable menu. Unchecking this will display 0 in variables whose value would be rounded to 0 given the rounding settings set on that variable.

"Display as hex uses memory" is poorly named but also makes it so hexadecimal values represent the bytes in memory instead of the values found in that memory as hex. For example, with this option checked, floating point numbers would appear as the raw bytes that make up the float, but with the option unchecked, it would round the float down the nearest integer and convert that integer to hexadecimal.

"Neutralize triangles with 0x15" changes the default behavior of neutralizing triangles. The difference between triangle type 0 and 0x15 was already explained in the triangle tab chapter, so just know that unchecking this defaults to neutralizing with triangle type 0. "Cloning updates HOLP type" determines whether using STROOP's clone buttons will also update the HOLP type to match the object being cloned. "Use in-game trig for angle logic" makes STROOP use SM64's trigonometry math functions to calculate the angle between 2 positionangles, instead of the accurate measurement.

"Use extended level boundaries" makes STROOP assume the ROM you have loaded has extended level boundaries enabled. For those who don't know, extended level boundaries is a ROM patch

commonly found in old ROM hacks, and it uses cheap tricks do extend maximum level size. This also messes up geometry a bit and is the cause behind the fake BitDW door clip you might've seen before. "Use expanded RAM size" makes STROOP assume that you have an N64 expansion pak loaded into the emulator, giving you access to the latter 4MB of the N64's memory.

"Do quick startup" makes STROOP boot up 4x as fast. That's it. It does this by skipping loading the variables of every single tab, but that only saves time if you plan on using every single tab. And yes, you are seeing this correctly, it's OFF by default. There is no reason to not have this activated. It increases the loading time when you first click a tab by half a second, how is that a problem? For reference, on my setup, normal STROOP boot time is about 8.7 seconds, and with quick startup enabled it goes down to 2.5 seconds.

The "reset saved settings" button below will set every option in the leftmost box back to their default settings. Except do quick startup. It doesn't touch that one at all.

The second column of checkboxes are for which indicator overlays to show by default on the object slots. You can disable or enable any of the overlays. If you forget what each one does, it's covered in the object slots chapter. Right clicking in this section will give options to either turn all of them on or all of them off. However, these options will not affect the last 3 overlays: hitbox overlap object, parent object and child object. Additionally, those 3 are the only ones to be unchecked by default. As a reminder, the parent object and child object overlays will only appear when hovering over an object slot, and you're still able to see them even when they're disabled in the settings by holding P when hovering over an object slot.

The remaining options are listed as variables on the right panel, and it contains all the options whose values is not on or off. "Goto above offset", "goto infront offset", "retrieve above offset" and "retrieve infront offset" are for setting the offsets to place the objects for the goto and retrieve shortcuts for objects. By default, both are set to be 300 units directly above Mario. However, you can set this to instead be at Mario's height, in front of him. Infront being relative to the direction Mario is facing. These options won't do anything unless the "offset goto/retrieve functions" option is set to true.

"FPS" refers to STROOP's FPS. By default, this is 30 to match the game's FPS. However, you can make this slower or faster to change how frequently STROOP checks Mupen's memory for updates. Setting this high enough will cause STROOP to run at an effectively uncapped framerate, although this isn't recommended, as it causes unnecessary CPU usage. "Pos controller relativity" will change what the positionangles in the TAS tab are moving relative to when moved with the 3D controllers on the left panel in relative mode. By default this is None, meaning that they move relative to themselves, but this can be changed to any positionangle type, such as Mario.

"Object slot size" is a number representation of the slot size slider found above the object slots. Unlike the slider, any number can be entered here. However, setting this value to below 1 can often crash STROOP. And just like the size slider, it's laggy and can take multiple seconds or attempts to get the size set correctly. "Custom release status" will override the default release status for objects when releasing or dropping them with the value found here.

There exist options for changing the shape and appearance of variables in STROOP, but these are found in the custom tab, likely because they aren't saved and reset to default when the application is closed.

Chapter 21: PU tab

The PU tab provides easy PU navigation tools as well as info for those willing to traverse through them without cheats. On the left panel is located a unique-looking PU controller. The buttons around the house allow you to traverse through lateral PUs in all 4 cardinal directions. The singular arrow buttons move Mario one PU over, while the double arrows move Mario by one QPU, or 4 PUs. Clicking the house button will teleport Mario back his home universe. None of these buttons modify Mario's position within the universe, unless you move far enough where float coarseness starts affecting Mario, at which point he will lose precision on his original position.

The text below the controller tells Mario where he is in the multiverse. Indicated with his current PU and current QPU, with QPU-misaligned PUs being marked as fractional location in the QPU grid.

At the bottom is a 3D controller for moving through PUs, similar to the one above. However in this one, you're able to move diagonally. This second controller exists so you can make use of the number in the middle to move an arbitrary number of PUs as well as move through vertical PUs or VPUs. The 3D controller is further made unique by the lack of the relative checkbox, now replaced with the QPU checkbox, moving Mario by QPUs instead of PUs. However curiously, the relative checkbox can still be accessed by popping out the controller using the right click menu. Unfortunately, this is only a visual error, the checkbox still acts like the QPU checkbox.

21.1: Variables

All of the red variables are copied over from Mario to be used for reference on where he is, what's his speed and what's his angle.

For the blue variables, "syncing speed" is the speed you would need to move over one PU while taking into account the steepness of the ground you're on. "QPU speed" is the number of QPUs you would move over with the speed you currently have. "PU speed" is the same thing but for the number of PUs. "QPU speed comp" and "PU speed comp" are the integer components of your current PU and QPU speed. In other words, it's your PU and QPU speed rounded to the nearest integer. "Relative speed" is the speed you're going at relative to your spot on the level map. So if you're at your syncing speed minus 1000, your relative speed would be -1000. "Relative X speed" and "relative Z speed" give the relative speed but only in either the X or Z axes.

Note that for all of the blue variables, setting the value of one of them will set Mario's speed and all of the other accordingly. Furthermore, there is an adjustments variable highlighted in yellow called "PU params". You enter a value by entering two integers separated by a comma. By default, the syncing speed, PU and QPU speed and QPU and PU speed component variables assume that you're trying to move 1 QPU over, but modifying PU params can modify this. You enter the number of QPUs you're trying to move over, and the variables will modify to the values needed for that.

The green variables give information about your current floor triangle. "Dist above floor" is Mario's distance above the floor below him, "normal Y" is the Y normal of that floor triangle and "defacto multiplier" is the effective defacto speed multiplier Mario is currently subjected to. This differs from the normal Y if Mario is in the air.

The purple variables indicate Mario’s position in the multiverse. "QPU X", Y and Z give Mario’s current QPU position, "PU X", Y and Z give Mario’s current PU, and "relative X", Y and Z give Mario’s relative position to the level map.

Finally, there’s information for all 4 quarter steps. Each quarter step has 4 variables for predicting the next Z and Z relative speed and the next intended relative Z and Z position. These predictions do not account for change in slope angle or something being in the way. They assume that the slope stays constant for all 4 quarter steps and that all of them succeed. It’s not an advanced algorithm, it’s just interpolation.

Chapter 22: Area tab

The area tab controls variables relating to Mario's currently loaded area. These areas are used in levels such as DDD and WDW to load and unload different sections of the map to make the level feel larger. On the left bar, you can select any area from 0 to 7 to view its contents in the variable panel. Activating the "select current area" checkbox will simply automatically select Mario's current area and update that selection live.

As for the variables, there's a pointer to the current area, and the index of that area for the level. These two variables exist in two places, one for Mario and one for the game (the Mario ones being highlighted in red and indicated with an M). These 4 variables are the only one to not update as you change your area selection, as they're specifically referring to the currently loaded area. The remainder of the variables reflect the area selected on the left panel.

"Area ID" is the area you currently have selected. "Is current area" tells you if the selected area is the one currently loaded. "Terrain type" is an ID for the type of terrain of the area. This determines what sounds to play when walking on the ground. "terrain description" is a text description of that terrain ID. "Geo layout ptr" is a pointer to the area's visual triangles. "Collision ptr" is a pointer to the area's collision triangles.

"Mystery 1" and all 4 other mystery variables laid throughout this tab imply they're unused variables, but they're not. Whoever added them in was simply too lazy to rename them once decomp found their purpose, even if they are unused. The real mystery is where is mystery 5? It's missing. Anyway, mystery 1 is actually a pointer to data for the rooms of the area. Rooms like in BBH or HMC.

"Mini objects ptr", confusingly referred to as macro objects in decomp, is a pointer to a list of important objects in the area that are spawned with priority. "Warp links head ptr" is a pointer to the start of the linked list for the teleporters of the area. "Mystery 2" is a pointer to the list of painting warps in the area. "Mystery 3" is a pointer to the list of instant warps in the area, like the ones on the TTM slide. "Objects head ptr" is a pointer to the list of the remainder of the objects in the level. "Level camera ptr" is a pointer to the camera. "Mystery 4" is the ONLY one of the mystery variables to actually be unused.

"Whirlpools" is a pointer to the 2 whirlpools the area can contain. Their details are further seen below. There can be up to 2 whirlpools per area and they both have a position and a strength. The position is their center, and their strength is by how much they pull Mario towards their center each frame. Interestingly, the strength can be negative, in which case Mario gets pushed away from the center. This is used in the game for both jet stream stars, or any strong current. "Mystery 6" is simply a pointer to the second of the 2 whirlpools.

"Level dialog index" is the index of the dialog to show when entering an area you haven't beaten yet. "Always 0xFF" is always 0xFF. The structural integrity of our reality as we know lies upon this variable forever being 0xFF. Changing its value could have cataclysmic consequences unconceivable to any human being. A rapture unlike any other would tear open the sky as- oh wait nevermind it's just an unused second dialog index.

"Music param 1" is the music ID for this area, and "music param 2" are the arguments passed into the music playing function as a bitfield. "Unused" is unused.

Chapter 23: Model tab

The model tab lets you view the collision models of objects or the level. To view the level geometry, click the "view level" checkbox, which actually acts more like a button, since you can't uncheck it manually. To view an object, click it in the object slots. Instead of bringing you to the object tab, this will instead add a blue selection marker to the object and open it in this tab. If the object has any collision data, you will see it. This also means that you cannot select multiple objects at once while in this tab.

On the right panel is a view of the 3D model selected, rotating counterclockwise by default with all edges drawn in dark blue. This view can be interacted with just like the 3D camera from the map tab, however the controls are different. You move around with either the arrow keys or WASD, you move vertically up or down with Q and E, you look around by clicking and dragging. To increase your turning speed, hold shift. To slow it down, hold alt. Holding control will lock all movement. Scrolling the mouse wheel seems like it moves the camera forwards but be careful. It doesn't. It changes the FOV, meaning it can mess up your movement.

The address of the collision data of the object will show at the top next to "model address". Unfortunately, the model address number is read-only, you can't enter your own pointer. Furthermore, when displaying the level geometry, it just says (level) instead of the address.

Below the address are 2 tables. One for vertices and another for triangles. By default, the shitty layout of STROOP makes them crunched up to the left and unreadable, so you have to drag the middle bar between the left and right panels of the tab further to the right to extend the tables and view the data. For the vertices, you see the total indicated at the top and a table containing the index of that vertex in the data, and the X, Y and Z values. For the triangle table, you once again have the total and the index, but also other data. "Type" refers to the type of the triangle. This is used to mark the special properties of floors. "T1", "T2" and "T3" are simply indices into the vertex table, indicating which indices make up that triangle.

Selecting vertices marks them with a yellow square, and selecting triangles fills them in with their respective color for if they're a floor, wall or ceiling. By default when opening a model, the first vertex and all triangles are selected.

The shader used for the 3D view can look really messed up when it comes to filling in triangles the further away you are from them. However, the closer you get, the more accurate the filling becomes.

Chapter 24: Gfx tab

This tab is for managing and viewing the data for everything that gets rendered to the screen (except HUD elements). It's one of the most complicated and tech-heavy tabs, so I recommend skipping this chapter entirely if you're even somewhat new with STROOP or technical SM64 knowledge as a whole.

The render scene is composed of a giant tree of graph nodes. Each node gives a tiny bit of information about its child nodes, and each node can have as many children as it needs to. A node can give information on how to render its children, the position of their children, the level of detail for that object, etc.

On this tab, there are 3 sections. The left one, the middle one and the right one. The left section is the graph node tree. Once you've grown the tree, you will be able to navigate through the nodes and their children here. The center section contains the controls and variables. The right section is a text editor. You can modify the text in there, even when later on stuff will appear there, but there is zero use to typing in there. I believe that might've even been left in as a mistake.

To be able to view and modify the nodes, there are 2 different buttons to do so. "Build from root" will create the tree from the root node, thus giving you access to the entire tree. However, this becomes very cluttered for any actual work. If you're only targeting one or a few objects, you can select those objects in the object slot section, move back to the gfx tab and click "build from selected objects". This will create the tree but only show the data for the selected objects. If the object has no model, it will have no child nodes. The "inject hitbox view code" is the exact same as seen in the version number's right click menu. That one was just a shortcut to the button in this tab. It implements the hitbox view code hack.

Once the tree is built, click on the + next to a node to reveal its children if it has any. Click on the node itself to inspect its data. All nodes contain the same 20 byte header, shown here with yellow variables, but they often also contain their own data depending on the node type, shown as blue variables.

24.1: Graph nodes

The header contains the variable's type, represented with an ID in hexadecimal. Some IDs have hex 100 appended to them to signify that they execute code, and that their first custom variable is a function pointer to said code. "Active" is a flag saying if the node is active or not. If inactive, the node and all its children pretend to not exist. "Bit 1" indicates whether a node should process all of its children first before processing itself. "Billboard object" indicates whether an object should always face the camera. "Bit 3" indicates if a master list node should incorporate a Z-buffer. "Invisible object" indicates that it is invisible. "Is animated" indicates that it or its children are meant to animate. "List index" indicates in what order the node and its children should be drawn. The possible values are 0 for nothing special, 1 to render as regular opaque geometry, 2 to be drawn as an opaque decal, 3 to be drawn as an opaque inter, 4 to be drawn with full transparency when needed, 5 to be drawn with working partial transparency, 6 to be drawn as a decal with transparency, and 7 to be drawn as an inter with transparency. What does inter mean? I don't know, but water has it. "Previous" is a pointer to the previous sibling node, "next" is a pointer to the next sibling node, "parent" is a pointer to the parent node and "child" is a pointer to the first child node.

There are the 22 types of nodes, and their IDs are mostly sequential with a few skipped numbers. Here are roughly what they do. ID 1 is the "root" node. It's where the graph node tree starts. There's only one, and it contains on screen dimensions and positions.

ID 2 is for orthographic projection rendering or "screenspace" nodes, so stuff rendered directly to the screen instead of the world. The HUD doesn't use the graphics tree in this tab, so the only thing using this is the level's background or sky. The two custom variables are unnamed. The first one doesn't do anything, and the second one shouldn't even exist. Orthographic projection nodes are only supposed to have one custom variable, so this second one actually overruns the memory into the next node, usually the background image. So don't touch it. ID hex 103 is for perspective projection, or "projection 3D" nodes. This is for rendering stuff in a 3D world. It has a pointer to its update function, an FOV, a near clip plane and a far clip plane.

ID 4 is for "master list" nodes. They contain 8 lists, one for each render list type. The ones from the list index variable in the yellow section. Pointers 0 through 7 point to the head of each list, while pointers 8 through 15 point to the tail of the list. Pointer 0 goes with 8, 1 with 9 and so on. ID hex A is for "group" nodes. These are simply used for grouping stuff together under one parent. ID hex B is for "level of detail" nodes, and these check for how far from the camera something is and act accordingly. "Min cam distance" and "max cam distance" are the distance from the camera the children have to be to be affected, and pointer 1 doesn't exist, and once again simply overruns memory into the next node. ID hex 10C is for a "switch" node. It can have multiple names describing what it switches, but the name always starts with switch. It runs its selection function and outputs the selected value in "selected child".

ID hex 114 is for the "camera" node. It contains its update function, its current position labeled with "from" coordinates and its focus labeled with "to" coordinates. Fun fact for the people brave enough to read this far into the GFX chapter, you can make your own hacked cam by setting the function pointer to null and manually changing the from and to positions. The real camera will still exist, making it hard to control Mario, but tases will play back the same, meaning you can get some sick footage. Or better yet, if you know how to hack savestates, you can record tases, then later modify the savestate to add the better camera and manually move it around with custom code.

ID hex 15 is for performing a translation and a rotation at once and is labeled "debug transformation" despite being used in the game. It takes in an XYZ for translation, an XYZ for rotation, and performs it. ID hex 16 is for performing only a translation. It's only used in one place in the game, and that's the debug level select. "Segmented address" is a pointer to a display list. If null, it doesn't render anything. XYZ offset are the translation offsets. ID hex 17 is a "rotation" node. It acts just like the translation node, except that XYZ are used for rotating.

ID hex 18 is for a "game object" node. It represents all or part of an object. If the object is split into multiple graph nodes, the "shared child" pointer indicates the object's main display list. ID hex 19 is an "animated node". It handles animating it and its children. It retrieves the animation data from data stored in its parents and moves accordingly. It can also apply an XYZ translation. ID hex 1A is a "billboard" node. It automatically turns it and its children into billboards. This is useful for when you want only a portion of something to be a billboard, like a goomba body. ID hex 1B is a "display list" node. Its sole purpose is to draw a display list containing rendering info. ID hex 1C is a "scaling node". It simply linearly scales it and its children. The scale variable is how much to scale by, with 1 being

normal size. ID hex 28 is a "shadow" node. It handles drawing the shadow for its children. "Radius" is the size of the shadow, "opacity" is the shadow's opacity, with 0 being invisible and 255 being fully opaque. "Shadow type" is a value indicating the shadow's type. 0 means a nine vertex circle, 1 means a four vertex circle, 2 is an unused flat circle, 10 is a static square, 11 is a scalable square, 12 is a toggleable square shadow, 50 + an ID for a square with hardcoded parameters and 99 for the player's shadow. ID hex 29 is the "object parent" node, which contains all of the objects. Its sole variable is a pointer to the groups it contains.

ID hex 12A is for "script" nodes. These come under many names indicating what they draw, but they all start with script. These nodes render things that are generated dynamically by a function, such as water or wobbling paintings. The pointer points to the function the generates the display list, and both parameters can be used by the node however it wants to pass data onto the function. ID hex 12C is for the "background image" node. It contains a function to draw the background. However, this node is meant to have another variable, but STROOP doesn't show it. This is common in this tab, but this one especially sucks because it would've let you set the background to anything you want, or even have it be a flat color of your choosing if the pointer is null. If this bugs you a lot like me, go to STROOP's github page, download the source code, navigate to GfxManager.cs, and in the GfxBackgroundImage class, add the line "precursors.Add(gfxProperty("Background", "int", 0x1C));" after the line that's similar to it. Then build and launch your custom STROOP to have this one new feature available! Unfortunately only one background image is in memory at a time, but the color thing still works.

ID hex 12E is for a "held object" node. It renders the object that Mario is holding, in Mario's hands. The function pointer is the code that renders it, "Mario offset" is the offset from Mario, likely a holdover from when Luigi was planned to be in the game. "Held object" is a pointer to the held object, and "position x", y and z are a translation offset. ID hex 2F is for a custom culling radius. Unfortunately, it's not implemented in STROOP and thus shows up as a default "GFX node" with no parameters, but it's meant to give a specific distance for how far something needs to be from the camera to be rendered. Usually used for very large objects.

24.2: Display lists

When selecting either a display list node, an animation node, a translation node or a rotation node, you will be able to click the "export display list" button to have STROOP read the contents of the display list and show in the rightmost panel its contents, formatted to be slightly more readable by humans. The display list info cannot be edited, only read. And even then, it's very hard to read. It contains instruction on how to render the display list formatted with abbreviations of the commands, it contains vertex data that formatted like this: "(368, 146, 7) 0x0000 (990, 0) (190, 101, 218, 255)". In order, that's vertex position, flags, UV coordinates and finally either the normal or color depending on the settings. The triangle data looks like this: "BF000000 0046283C G_TRI1 indices (7, 4, 6)". The data at the end are indices in the vertex list for which vertices make up the triangle. There's not much else I can say about these. To go any further, I'd recommend hopping off STROOP and reading decomp.

Chapter 25: Debug tab

The debug tab is for accessing the debug resources built into the game itself. STROOP is not creating any of these, it's all in the game and simply needs an external resource like STROOP to turn it on. These were used by the developers of the game to test stuff.

25.1: Left panel

On the left panel, you can choose one of 7 options for on-screen test info labeled advanced mode, you can pick one of 3 options for the resource meter and toggle on or off 4 different debug features.

The "advanced mode" radio buttons let you pick which of 6 text displays you want, or if you want it off. Note that these debug features were created by the developers of the Japanese version of the game and that many of the letters in the colorful text graphics were removed when porting to the US, so garbled letters are actually just letters that are missing data on the US version. Also, an M at the start of a number indicates that the number is negative.

"Object counter" shows the number of objects currently loaded in memory on screen. This is very useless if you have STROOP open, but it's useful when making a TAS or a recording where you want the object count to be visible without editing.

"Check info" provides statistics for the collision data currently in memory for that frame. "Area" represents the current collision cell you're in with a hexadecimal byte. SM64 levels are divided up into square cells to make collision detection faster, only checking for the collision triangles within that cell. "DG", "DW" and "DR" are for currently loaded floors, walls and ceilings respectively. The number directly next to the two letters is the number of currently checked collision triangles, both static and dynamic for that triangle category, and the second number a bit further away is the number of times within the previous frame a call was made to check for collision for that type of triangle. "Listal" gives the number of surface nodes currently allocated in memory. "Statbg" is the geometry triangle count for the level itself, not counting any of the objects. "Movebg" is the geometry triangle count for all the currently tangible objects. Finally, you get the object count as a bonus.

"Map info" gives info about the whatever the last processed object is, which for STROOP it'll always be Mario. "Area" acts the same as area from the check info menu, "WX", "WY" and "WZ" are Mario's position truncated to an integer, "BGY" is the height of the floor below Mario, "AngY" is Mario's angle converted to be represented as degrees from -180 to 179, with 0 being 0 and 32768 being -180. "BGcode" is the type ID of the floor under Mario. "BGstatus" is the decimal representation of the flags of the floor under Mario. "BGarea" is the current room the floor under Mario is set to be in. The game is then meant to print the water level at Mario's position and the object counter again, but they overflowed the debug print code, causing it to say "DPrint over" instead.

"Stage info" prints the current TTC speed setting and the object counter. That is it. For info on what each number represents for TTC speed settings, check the miscellaneous tab chapter.

"Effect info" shows a list of 8 list entries named "A0" to "A7", and an A to the left of A0 used as a cursor. This menu does nothing in the release build of the game. Sometimes objects read from the table, but there's no way to make any of the values not 0, so nothing happens. You also get an object counter.

"Enemy info" is the exact same as effect info but with a B instead of an A. It also has the object counter. Interestingly, despite there being no vanilla way to modify the contents of this list, B7 being equal to 1 is one of the conditions for activating spawn mode, which will be covered later in this chapter. So if you ever select this option and see B7 set to 1, STROOP did that, not the game.

"Resource meter" is for activating bars near the bottom of the screen indicating performance. Both modes comprise of 3 bars. The top one measures the CPU tasks, the second one measures the RCP tasks (the RCP is basically the N64's GPU), and the final static bar at the bottom is for reference.

"Meter 1" draws all sections of the bars one after the other to provide a cleaner looking measurement. The shorter the top 2 bars are, the better performance is. This also makes the bottom bar more useful, since you can quickly gauge performance. Blue is ideal, yellow is okay, orange is bad and red is unplayable. In this mode, for the top bar, red is for sound updates, yellow is for level script execution and orange is for rendering. For the bottom bar, orange is the RDP duration (graphics), yellow is RSP duration (shading and sound mixing) and red is Vblank. However, the second bar for the RCP is usually not drawn at all. I don't know if this is simply a common emulation inaccuracy or the result of a glitch in the profiler. Sometimes a little bit of red shows up though.

"Meter 2" provides mostly the same information, but that data is shown in the order the tasks are performed in. Meter 1 grouped all colors with each other, but the CPU doesn't complete a task before moving onto another. It sometimes jumps from one to the other, and this is what we see here. The meter at the bottom is no longer used to represent how well the game is performing, it represents frame time. The left side is the start of the frame, and the right side is the end of the frame. All of the colors mean the same thing as in meter 1, but a new color has been added. Cyan, which represents the time it takes for the CPU to send display lists. As with meter 1, the second bar rarely appears. However, there is a correlation between the small red bit appearing with the two sections of the top bar coming into contact with each other.

The "misc debug" section provides four checkboxes for additional debug tools. "Classic mode" prints info to the bottom right corner. "ANG" is the steepness of the floor below Mario, measured in degrees from 0 for flat to 90 for vertical. "SPD" is Mario's speed truncated to an integer. "STA" is Mario's action ID. "MEM" gives the number of free bytes in memory for display list allocation and "BUF" gives the number of free bytes in memory for the graphics pool.

"Spawn mode" lets you spawn objects with the D-pad. D-pad left spawns a crazy box, D-pad right spawn a koopa shell and D-pad down spawns a water shell. Spawn mode can only be activated if your advanced mode selection is either off or stage info. If it's not one of those two, stage info will automatically be activated. Those two are somewhat linked in the code, so they rely on each other.

"Stage select" overrides the normal title screen of the game with the debug title screen of the game. The next time either the file select screen or title screen of the game loads, you will instead be sent to the stage select screen. The music and background are the same as the real title screen, but Mario's head is gone, and in its place is a giant prototype logo for the game in solid red, green and blue letters. No demo will ever play, and the text below will ask you to select a stage then press start. You select a stage with D-pad up and down, with the stage index being able to go from 1 to 38. The index will be set to whatever the last area you were in was.

However, when entering a stage, you're sent in a special way. Stage select will always use file D for save data, it will bypass the star select screen and always send you into star number 1 in multi-star areas, and most importantly, exiting the stage in any shape or form, whether that be by dying, grabbing a star or exiting the level in the pause menu, you will always be brought right back to the stage select screen, if you entered a non-stage area like the castle through the stage select menu, you will be able to go to different places, but those 3 things listed above will still send you back and also still no star select menus. To exit level select mode, either uncheck the mode in STROOP or press C left + C right + Z + Start on the level select menu.

These are the values of the 38 selections and where they send you. 1, 2 and 3 are invalid courses. All invalid courses in this list will simply default to sending you to the real title screen. However, the game is not reset so all values in memory stay intact, including the stage select debug flag, so pressing start on the real title screen will send you back to the stage select screen. 4 is "TERESA OBAKE" and sends you to BBH. 5 is "YYAMA1 % YSLD1" and sends you to CCM. 6 is "SELECT ROOM" and sends you into the castle lobby. 7 is "HORROR DUNGEON" and sends you to HMC. 8 is "SABAKU % PYRMD" and sends you to SSL. 9 is "BATTLE FIELD" and sends you to BOB. 10 is "YUKIYAMA2" and sends you to SL. 11 is "POOL KAI" and sends you to WDW, and the water will be the same as what it was the last time you entered the painting. 12 is "WTDG % TINBOTU" and sends you to JRB. 13 is "BIG WORLD" and sends you to THI's large island. 14 is "CLOCK TOWER" and sends you to TTC on whatever was the last setting it was on. 15 is "RAINBOW CRUISE" and sends you to RR. 16 is "MAIN MAP" and sends you to the castle grounds. 17 is "EXT1 YOKO SCRL" and sends you to BITDW. 18 is "EXT7 HORI MINI" and sends you to VCUTM. 19 is "EXT2 TIKA LAVA" and sends you to BITFS. 20 is "EXT9 SUISOU" and sends you to the secret aquarium. 21 is "EXT3 HEAVEN" and kills you in real life, sending you to heaven... or if you're lucky it just sends you to BITS. 22 is "FIREB1 % INVLC" and sends you to LLL. 23 is "WATER LAND" and sends you to DDD. 24 is "MOUNTAIN" and sends you to WF. 25 is "ENDING" and sends you to the cake screen. 26 is "URANIWA" and sends you to the castle courtyard. 27 is "EXT4 MINI SLID" and sends you to PSS. 28 is "IN THE FALL" and sends you to COTMC. 29 is "EXT6 MARIO FLY" and sends you to TOTWC. 30 is "KUPPA1" and sends you to the BITDW boss fight. 31 is "EXT8 BLUE SKY" and sends you to WMOTR. 32 is an invalid level. 33 is "KUPPA2" and sends you to the BITFS boss fight. 34 is "KUPPA3" and sends you to the BITS boss fight. 35 is an invalid course. 36 is "DONKEY % SLID2" and sends you to TTM. 37 and 38 are invalid courses. After that, the index wraps around back to 1.

Finally, "free movement" puts you into the free movement action the moment you touch the analog stick. It's already covered because of its entry in the version number right click menu, so this document won't repeat itself here. However, this toggle is inconsistent. It doesn't always work due to the lazy way it's been implemented. If you do get it to work, Mario will not be completely in the debug action, as he is still able to interact with some objects.

25.2: Variables

"Advanced mode" is 0 when off and 1 when any of the other values. "Advanced mode sttg" is the index of which advanced mode setting chosen, ranging from 0 for object counter to 5 for enemy info. "Resource meter" is 0 when off and 1 when on, and "resource meter sttg" is 0 for meter 1 and 1 for meter 2. "Classic mode", "Spawn mode" and "stage select" are 0 when disabled and 1 when enabled. "Free movement" is a value that gets used for the free movement patch in this tab. Its value depends on the current version of the game. In the American version, it's 0x5FAB when activated and 0x98D5

when disabled. In the Japanese version, it's 0x5F0D when activate and 0x97D1 when not. A0 to A7 and B0 to B7 represent the values found in effect info and stage info. You can manually set their values here. Not many things in the game read those values however, so to see what gets affected, check decomp. The spawn values indicate the object IDs and behaviors that get spawned for each corresponding button in spawn mode. However, modifying these variables does nothing.

Chapter 26: Hacks tab

The hacks tab provides simple hacks to quickly patch into the game. Kind of like a gameshark menu. The tab has 2 panels and controls at the bottom. The right panel is simply the menu for the bottom controls, they're a single thing. The left panel is for miscellaneous hacks.

However, as stated by the text at the top, the hacks require the pure interpreter to be enabled. This applies to all hacks in this tab. What's the pure interpreter? It's a setting in Mupen64 to improve accuracy in the CPU. However, it slows the game down a lot, so it's disabled by default. To change the CPU core in Mupen, selection options->settings->general, scroll down to the section named either "Core" or "CPU core" and click on the "type" option until pure interpreter is selected. Click OK then restart the game. When you're done using the hacks tab, I strongly recommend changing back the CPU interpreter to dynamic recompiler for performance. Luckily, by creating a savestate after enabling the hack then leaving and changing back the CPU settings, you can load the previously made savestate to use any hack without being in pure interpreter. It's only really required when first enabling the hack.

Toggling "PU visibility" will make PUs visible. Of course, objects will be missing and water won't exist, but that's just how they are. "Camera 45 degrees" forces the camera to be like the one from special stages like the vanish cap stage. This means that instead of C-left and C-right moving the camera left and right up to a certain point, it simply rotates the camera around Mario by 45 degrees, giving you more control. "Camera control" enables the hacked camera in the cam hack tab. "Quarter frame info" allows the Q frames tab to be used. "Display variable" adds back the missing rainbow text characters to the US version of the game, and might do other things as well. "Display variable 2" seems to exactly the same as the last option but with a little bit more. It's hard to tell what the hacks do from lines of hexadecimal in a text file.

For the next 4 hacks, there's a (U) version and a (J) version. This is because the American and Japanese versions of the game have different RAM mappings, and as such the hacks need to be placed in different locations. Be careful when selecting, because clicking the patch for the wrong version is very likely to crash the game or cause weird things to happen. For example, enabling quarter frame info first and the previous positions then disabling quarter frame info, Mario becomes unable to land on the ground.

"Re-enable debug controls" re-enables the debug controls, maybe. It seems to be patching code to call unused functions in the "debug.c" decomp file, but I don't have the patience to manually trace bytecode for this hack that not even Pannen knew what it did. "Previous positions" stores previous position data in STROOP so that it can be used in the map tab and Q frames tab. Note that the previous position hack modifies code also modified by the quarter frame info hack, so weird behavior can happen when you try removing one when the other is enabled.

"Num RNG usages" allows the RNG usage labeling method for the object slots to be used. The number of times the object calls RNG in a frame will determine the color of the background of the slot. "RCP crash" correctly emulates the RCP crashing when too many triangles are drawn to the screen at once, instead of corrupting memory like emulators normally (and inaccurately) do.

"Object graphics triangles" enables the use of the object graphics triangles tracker in the map tab. "Object graphics triangles cam POV" does the same thing, but from the camera's POV.

The right panel and the controls below are a tool to spawn any object in the game. Any of them. Even unfinished prototype objects that never appear in the game like Blargg. This tool does not require pure interpreter mode enabled on the emulator. Simply select an object, click the "Set spawn type" button then press L in the game to spawn the object in front of Mario. However, be careful! Most objects are only designed to be in the levels they naturally appear in. Attempting to spawn certain objects in levels they don't belong in may crash the game. Prototype objects are marked with a tilde and the level they're meant to be in. Spawning them is usually safe. None of them move or are tangible, the ones in vanish cap stage made the HUD and some objects disappear and the "blue thing" from wing cap stage is the only real unstable one. It corrupts graphics memory and makes the game very unstable. Spawning it is likely to cause the entire emulator to crash after a few seconds.

The "behavior" value will be the behavior of the object when spawned. "GXF ID" is the graphics ID the object will use when spawned. "Extra" is for extra parameters used by the object to dictate facts about itself. For example, the size of the koopa. Pressing the "Reset (turn off)" buttons removes your ability to spawn objects until you click on set spawn type once again. Note that many behavior values will not work properly without a specific graphical model associated to it, so be careful when freestyling.

Chapter 27: Cam hack tab

The cam hack tab allows you to operate the hacked camera STROOP provides. With the vanilla ROM of the game however, this tab does nothing. To enable the hacked camera, you need to activate the camera control hack in the hacks tab (with pure interpreter enabled on the emulator of course) or load a savestate with the hack already implemented.

Alternatively, you can download a ROM of SM64 with the hack pre-baked into it. You can obtain a link to this tab by very intuitively right clicking the "camera mode:" text and selecting "download camera hack ROM". This will open a MediaFire link in your default browser to download the z64 file. With this hack loaded, this tab's controls will be available to you.

27.1: Be your own lakitu

The left panel of this tab contains all the controls for the hacked cam. At the top are the 5 modes. "Regular" disabled the hacked cam, returning the view of the game back to the real camera. Which by the way, the real camera still exists even when the hacked cam moves around, and Mario uses the position of the real camera to determine which direction is up. That's good because it means you can play back TASes while moving the camera around, but it also means that playing the game regularly will be very difficult.

"Fixed position, fixed angle" will fix in place the position and angle of the camera. It can only be controlled with the 3D controllers below. It's like fixed cam but without turning.

All 3 remaining options need to have an object selected to work. If no object is selected, it defaults to Mario (not the Mario object, the Mario state). To select an object, simply click its object slot. A red camera icon will appear next to the object to indicate that it is currently selected by in the cam hack tab, even if the camera mode isn't one that follows an object. "Fixed position, watch object" will fix the camera's position but continuously set the focus to the position of the selected object so that it's always at or near the center of the screen. "Follow object, relative angle" will have the camera follow the object from a set distance and always adjust the camera's angle to match the yaw angle of the watched object. "Follow object, fixed angle" will also follow the selected object, but unlike the relative angle option, this one will always maintain the exact same positional offset from the object, never changing the angle of the camera.

Once you've selected a non-regular camera mode, you can move the camera around with the 5 3D controllers. However, the first 4 of the 5 controllers are identical to the ones found in the camera tab. So to get an explanation on how to move the camera around in 3D space, go check the chapter for the left panel of the camera tab.

The last controller at the bottom middle is the only one not found in the camera tab. All it does is simply move the camera and the focus at the same time and the same distance, meaning you can move the camera around without changing its angle. The controllers will always work on non-regular camera modes, even if its seems like an aspect of the camera is fixed in place. However, moving the focus when the focus is the position of an object will move the object instead of the camera's focus. So be careful with that. Activating the relative option when moving the object focus around will use the object's angle as reference for forwards.

For both the controllers that move the hacked camera's position, if you're currently in either of the follow object modes and the relative checkbox is active, you can hold shift while clicking the buttons to use Mario's angle as forward instead of the camera's angle.

27.2: Variables & panning

"Mode" is the mode you selected at the top left of the tab. Note that both follow object options share the same mode, with the only difference between the two being whether the "absolute angle" variable is checked or not. All the other modes ignore this checkbox. "Object to follow" is the slot number of the currently followed object. After that is the camera position, yaw and pitch, and the focus position.

"Radius", "theta" and "relative height" are only used when in either of the follow object modes. Radius displays the distance between the object and the camera, and theta displays the camera's yaw relative to the object's yaw, with 0 meaning that the camera is looking in the same direction as the object. Relative height is the difference in height between the camera and the followed object. "FOV" is the camera's current field of view.

The remainder of the variables refer to something called "panning". This is one of the coolest features of the cam hack tab, and it's hidden right here. Panning allows you to set a custom camera path for the camera to follow, helping record clean movement in the game.

To start, simply set the "num pans" variable to 1. 20 or so variables should appear when you do this. These are the inputs for the pan motion. Think of pans as sort of keyframes. They have a start point and an end point, and a bunch of other options. The number you set in num pans is the number of keyframes you will have available to you. However, be careful. Each added pan will add 20 or so variables to the tab and so setting the value too high can make STROOP run out of memory and crash.

Pans only work if your cam hack mode is set to either of the "fixed position" modes. It can't be in normal mode or following an object. "Current pan" is the index of the previous pan, or the next pan if the next one is the first one.

"Pan cam pos", "pan cam angle", "pan cam rotation" and "pan cam FOV" toggle whether or not a certain aspect of the camera should be panned. Pos enables changing the camera's position in a pan, angle enables changing the camera's angle in a pan, but only if the fixed position or fixed angle mode is chosen. The camera can't look at an object and look in a specific direction at the same time. FOV changes the camera's FOV or at least tries to because the game fights hard to set the FOV to what it wants. Finally, rotation, which is special. Pan cam rotation is the only one of the modes to only work when the cam hack mode is set to either of the follow object modes. This is because the given positions and angles in the pans will be relative to the followed object. This also means that pan cam rotation cannot be set at the same time as either pan cam pos or pan cam angle. Turning it on will force the other two to disable, and vice versa. "Buffer" does nothing.

For each pan, you get 22 variables. The only non-unique one is global timer, which is just there for reference and quick access. These pans work off of the global timer for timing, so it'll come in handy. Curiously, you get a copy of the global timer for every pan variable group, despite them all referencing the same value in memory.

"Start time" and "end time" are the start and end values for your pan. The last frame of camera movement will be the frame after either of these barriers are reached. "Duration" will give the difference between the start and end frames. You can directly modify this value to also modify the end time at once. If the duration is negative, the camera will immediately jump from the beginning point to the end point once the start time is reached.

"Ease start", "ease end" and "ease degree" allow you to add easing motions to the start or end of your movement. This will gradually start or stop the camera to add more life to the motion. Ease degree dictates how strong the easing is. By default it's 3, 1 means no easing, and the higher the value the faster the acceleration and deceleration is. In practice, you shouldn't ever need an easing value above 20. Setting the easing degree below 1 causes weird things to happen. Setting the ease degree to 0 causes it to immediately jump from the start to the center point, then from there to the end. Setting the value to a negative number is even weirder. It makes the camera start at a far-out point way past the end point, come quickly rushing to the real world, passing the mid-point exactly halfway through the pan only to go past the start, and go wildly overboard in the other direction. The lower the negative number inputted, the more distance it travels. And this distance grows exponentially, so a value of -20 can already be enough to have the camera start at an infinite distance.

"Rotate CW" has the camera rotate from the starting angle to the end angle in a clockwise motion. By default the camera will always turn counterclockwise no matter the angles, so this is how you change it. The starting positions and angles are highlighted in green, and the ending positions and angles are highlighted in red.

"Cam radius start" and "cam radius end" are for controlling the start and end distances from an object when in cam rotation mode, and "FOV start" and "FOV end" are the FOVs the pan will try to change between.

Each of these variables are prefixed with "PanX", with X being a zero-based index for which pan they belong to. This tool might not be intuitive or have a good UI, but it allows for some very cool shots when used correctly.

Chapter 28: Q frames tab

This tab allows you to view information on all 4 of Mario's quarter steps processed in the previous frame. This tab contains only variables, and they only work if the "quarter frame info" hack is activated in the hacks tab. If working correctly, once Mario moves, the variables will update their values. Note that they will freeze when Mario is swimming because Mario does not move in quarter steps during water movement.

"Movement action" shows Mario's previous action variable from the actions tab. It's to help you know what Mario was doing before the 4 quarter step were calculated. Each quarter step has 11 variables associated with it. The first one highlighted in purple is the result. The result is the value returned by the quarter step function to indicate if something happened, and what to do from there. The numbers have different meanings depending on whether Mario is in the air or on the ground.

When Mario was grounded when the frame started, 0 means that he became airborne, 1 means nothing happened, 2 means Mario hit a ceiling or out of bounds and 3 means Mario hit a wall. When Mario was airborne when the frame started, 0 means nothing happened, 1 means Mario landed, 2 means Mario hit a wall, ceiling or out of bounds, 3 means Mario has grabbed a ledge, 4 means Mario has grabbed a ceiling, and 6 means Mario hit a lava wall.

For every quarter step, "water Y" was the height of the water at that point, "floor Y" was the height of the floor below Mario at that point, "ceiling Y" was the height of the ceiling at that point, "floor tri" is a pointer to the floor triangle that was below Mario, "ceiling tri" is a pointer to the ceiling triangle that was above Mario, "low wall tri" is a pointer to the wall that was detected in Mario's lower wall hitscan, if any, "high wall tri" is a pointer to the wall that was detected in Mario's higher wall hitscan, if any. Finally, Mario's position on that quarter step.

This tab's only hidden variables category is quite interesting. It's called "Previouspositions", and it's 23 sets of positionangles meant to represent Mario's 23 previous positions. These can be enabled by activating the previous positions hack in the hacks tab. Unfortunately, these don't seem to work. They only show 5 or so previous positions at a time, with many duplicates. So these aren't actually that useful. They don't even show every frame of movement.

Chapter 29: Var hack tab

The var hack tab allows you to display your own custom debug text to the game's screen, formatted similarly to the in-game debug menus found in the hack tab. The variables are listed in custom boxes in a similar fashion to the list of trackers in the map tab. This tab can be hard to get working, since you need to coordinate multiple things at once. First, you need the "display variable" hack enabled in the hacks tab, then click the "enable ROM hack" button on the var hack tab. You can right click this button to get a version where the variables update faster. It's listed as experimental, but from what I've seen it works fine. Only then are you ready to use the tab. If it doesn't seem to work, try restarting the game without loading a savestate. You can mess up the hacks set up by STROOP by loading savestates with a different hack state to what STROOP expects.

29.1: Variable box

To create a new variable, simply click "add new variable". By default, the variable will track Mario's horizontal speed. If you keep adding variables, they will display other Mario values in a specific order. It goes horizontal speed, yaw angle, precise health, the room of the floor triangle below, X, Y and Z position, and finally X, Y and Z HOLP position. Adding more variables after this will simply keep adding HOLP Z. By right clicking on the add variable button, you get more options. You get the RNG index, Y normal of the floor below, defacto speed, horizontal sliding speed, action description, animation description, Dyaw facing and Dyaw facing but in hexadecimal angle units.

But what if you want to choose your own variable? Well after creating a variable, it will appear on the right panel with a bunch of info. Below the name, you will have the address of the variable. That's what you want to change. To get a variable's address in STROOP, double click on it. The address you want is "N64 address". This means that variables generated by STROOP are not compatible with the var hack. This also means that the variables obtainable through the add variable button's right click menu are special and are the only STROOP-made variables that you can add.

After that, you'll also want to change the name. The name is what will appear in-game as text. Luckily, the var hack ROM codes also add back the missing letters from the alphabet in the American version of the game. This also fixes the text for the game's debug menus found in the debug tab. However, make sure that your name isn't too long and to add a dollar sign to the end of the variable name. As that's what gets replaced with the value of the variable. Without it, only the name will appear. There's a lot when it comes to formatting the name, so that will come later.

Now you've given STROOP the memory address of your variable, but STROOP can't know what it's supposed to be reading at that location. You need to provide the type of the variable using the radio buttons to the right. You can select between all types used by SM64 and display them as hex if needed with the "use hex" checkbox.

After the type is chosen, you need to position the variable on the screen. The given X and Y positions will apply to the first printed character, so it's your job to make sure there's enough room to print it. When it comes to SM64 screen coordinates, (0, 0) is the bottom left corner. Printed text only gets put on screen from 10 X, 3 Y to 300 X, 220 Y. Going out of these bounds will simply either not move the text or cap the position.

To have your variable appear on screen after modifying its properties, simply click the "apply variables to memory" button. If you want to update the on-screen positions, click the "set positions & apply variables to memory" button. Alternatively, pressing enter after typing in any of the variable's text fields will automatically run the update function. Similarly to the map tab, you can remove an entry by clicking the giant red X and move it up or down through the list with the blue arrows.

However, the variable still has more options on its left side. Checking the "use pointer" checkbox will have the address value be used as a base address instead, and the value will be fetched from the base address plus the "pointer offset" below.

"No num" in red text looks ominous, but checking it simply prevents the code that replaces the dollar sign with the real number to run. This parameter is automatically checked with the special variables, as the dollar sign replacement code is also responsible for their special properties.

You can right click in most places on a variable slot to get two extra options. The first one allows you to duplicate a variable and all its parameters, and the second is a quick macro to convert a variable to two unsigned shorts displayed side by side in hex that give the value of two sequential places in memory.

29.2: Buttons & formatting

Near the top of the left tab, you have 3 additional buttons underneath the add variable button. These allow you to save, load and clear var hack tab variables. They're saved in a custom format that you can open later to keep all the parameters. The "show variable bytes in little endian" and "show variable bytes in big endian" will pop up forms listing 32 bytes in text form for each existing variable panel. The endianness of the bytes simply refers to the order they're in. They're in little endian format when in N64 memory, but can be in big endian format when being manipulated elsewhere. This is what's being stored in memory by STROOP to be used by the hack. It contains in order the address to read, the X and Y positions on screen, the name, whether it uses an offset, the offset, whether it's using a signed type and finally the ID of the type.

The 3 controllers below are for determining what the automatically given position of the variables will be when creating them. By default, variables start at X 10, Y 192 and go down by 17 units for every variable. Even if you change the position values of the variables that come before, STROOP will simply use the number of existing variables to calculate the next position, not checking where they actually are. Here you can change the initial position of the first variable with X and Y, and change delta Y for how many units downward each variable should move by.

Both apply variables were already explained, same with enable rom hack, which simply leaves "clear variables in memory". It erases all variables from SM64's screen. The ROM hack is still enabled and variables can quickly be added back, but this is a way to show nothing on screen without touching the variables already on the tab.

Now, for the formatting. Here is what happens to the variable name when it's sent to memory. "\c" gets replaced with a coin icon, "\m" gets replaced with a Mario head icon and "\s" gets replaced with a star icon. Well, they actually get replaced with a plus sign, a comma and a dash, but SM64's text engine uses those characters to signify those icons. In fact, the only advantage to using the backslash

macro is that they’re easier to remember. You can simply add the raw symbols yourself (and this goes for all text replacements here, really).

After that, if the "no num" checkbox is unchecked, all dollar signs get replaced with "%x" if displaying the value as hex and "%d" if displaying the value as decimal. This is what causes all floats to be rounded. After that, if the variable is one of the special ones created with the add variable button’s right click menu, it replaces dollar signs with that variable’s custom string replacement function that automatically bakes in the value. This means that these special values are not injected into the game's code and their update rate is tied to STROOP's FPS. Next, the name is capped to a maximum of 19 characters, and that’s it for the name formatting before the values are sent into memory.

Once in there, the hacked code calls SM64’s formatting function, which functions similar to C’s printf but lacks most of the functionality. The only thing that works is specifying the minimum length of the printed number, along with an additional flag telling to fill the empty number space with leading zeros. To specify the minimum number of characters, simply add the desired number of characters between the percent symbol and X or D ("%x" -> "A"; "%4x" -> " A"). To fill that empty space with leading zeros, add a 0 in front of the number ("%04x" -> "000A"). That’s the only formatting that exists. Even attempting to use specifiers other than D or X fails.

Chapter 30: Coin tab

This tab is used to calculate the speeds and trajectories of coins released from enemies depending on RNG. This allows you to search for RNG values that satisfy criteria for the speed or angle of enemy coins.

First, you're going to want to set the coin parameters of the coins. You can select from an existing preset by scrolling through the list of objects in the top left and selecting the enemy or object you're seeking to collect the coins of. Selecting one will automatically set the five parameters. "Hspeed scale" is the variation of the coin's horizontal speeds, "Vspeed scale" is the variation of the coin's vertical speeds, "Vspeed offset" is the minimum vertical speed of the coins, "param order" is the order in which the coin's parameters are retrieved from the RNG function, and "num coins" is the number of coins that are dropped at once.

If you want, you can set your own custom values. For the param order, your options are "HAV", "HVA", "AHV", "AVH", "VAH" and "VHA" (caps required). V is for vertical speed, H is for horizontal speed and A is for angle. You can see the ranges of horizontal and vertical speeds just below. Note that the closing parentheses indicates that the coin can go up to but not including that value.

Next, you filter for the desired results. STROOP will search through all 65 thousand possibilities, so you want to automatically pull out the ones you want. You can enter a minimum and maximum horizontal speed, a minimum and maximum vertical speed, as well as a minimum and maximum angle. "Req'd num of qualified coins" determines how many of the coins need to satisfy the given conditions to pass the test. Setting this to the same value as the number of coins parameter ensures that all coins need to pass the test for a given RNG value to enter the table.

Next, the customization parameters. "Display non-qualified coins of a qualified coin group" determines whether a successful RNG seed should display the values of all of the coins dropped or only the ones that pass the test. If more than the minimum needed pass the test, all of the ones that pass will be shown. "Num decimal digits" is the precisions shown of the floating point values of the coins. "Starting RNG index" is to set a custom current RNG index for the table. By default, it picks the current value.

Now that that's all set, you're ready to generate the table. Simply click the "calculate" button and wait a few seconds for STROOP to calculate and draw the table. This can take a short while if there's a lot to process and generate. To erase the current table, either generate a new one or click the "clear" button. The size of the table is indicated at the bottom of the left panel.

The table has 6 columns. "RNG index" indicates which of the 65114 RNG indices the row represents. "RNG value" is the value returned by the RNG function on that index. If you've forgotten exactly how these two work, it's explained in the miscellaneous tab chapter. "RNG to go" is the number of RNG calls remaining until your current RNG index or the one inputted in the customization section will be the one for that row. Finally, the last 3 columns are all of the coin horizontal speeds, vertical speeds and angles.

You can change the width of each of the columns and click on the header row to change how the rows are sorted, but that's pretty much it.

Chapter 31: Disassembly tab

The disassembly tab is the simplest tab in STROOP. To use it, enter a memory address in the "start address" text field. The memory you can search through ranges from 0x80000000 to 0x8FFFFFFF, but N64 memory only ranges from 0x80000000 to 0x803FFFFF, or 0x807FFFFF if extended RAM is enabled in the settings. Your inputted memory address will be truncated to a multiple of 4 and addresses outside the range will have the first nibble forced to be 0x8. Press the "go" button to view the contents of the first 160 bytes of memory at that location. Once loaded, click the "refresh" button to update the memory and the "more" button to load and add on the next 160 bytes in memory.

The disassembly tab interprets all memory as MIPS instructions, so this cannot be used for viewing data, only code. On the left side of each row of text will be the address of the CPU instruction in blue followed by the 4 bytes making it up in big endian order and light grey text, and finally the instructions and text representation of the instruction in red text. Registers will be highlighted with green text and pointers in blue, even being able to click on them to jump to that address in memory.

Unless your name is Kaze and you're jacked as shit, it's unlikely you'll get a lot of use out of this tab. If your goal is to understand the game's code, decomp's C is much better.

Chapter 32: Testing tab

The testing tab is the least user-friendly tab, because it's not really meant to be used by users. It's a dump for any quick hacks, testing equipment and tools used by Pannen or a small number of people. This isn't as much of a tab for testing STROOP as much as it is a tab for testing stuff in SM64 with some sort of GUI to do so. The many sections are haphazardly laid around with groupboxes, so I'll be referring to each section by the text above it.

"Conversion" converts N64 addresses to emulator addresses and vice versa. You simply enter the N64 address in the "address" textbox, the size in bytes of the value located there in the "bytes" textbox, and once you click the "convert" button, the corresponding emulator address will show up in the "result" textbox. When entering an emulator address as the input, the bytes textbox doesn't get used for the calculation. However, you still need a valid number to be in there for the code to run.

"Memory reader" simply shows the bytes in memory at a given address, in the selected format. You select the value type you want to read the memory as, the address to look at, the number of variables to return, and if you want the values displayed as hexadecimal or not. Then when you click "read", a window will pop up listing all of the values found. There's little use to checking the hex checkbox, because that's exactly what the disassembly tab lets you view. Although, it is useful for copying larger amounts of hexadecimal values and storing them elsewhere.

"State transfer" is the very first thing in STROOP that comes with a set of instructions, which is honestly very sad considering how far in we are and where it's found. By clicking the "instructions" button, you get the following text:

"This is a tool for having one m64 file start with the same state (i.e. RNG, global timer, HOLP, etc.) as another m64 file. This assumes the m64 starts from the mission select screen. To use it, just follow these instructions:
(1) Pause the emulator.
(2) Open the m64 that you would like to copy state from.
(3) Advance 1 frame.
(4) Press the Save button in the State Transfer box in STROOP.
(5) Go to the mission select screen where you would like to paste the state to (possibly on a different ROM).
(6) Make sure you're selecting the correct mission.
(7) Pause the emulator.
(8) Press the Apply button in the State Transfer box in STROOP.
(9) Start a new m64 from snapshot."

There's not much to add. Checking the "offset timers" checkbox will simply subtract one from the global timer and animation timer when pressing the "apply" button.

"Obj at HOLP" will continuously set the position of an object to Mario's HOLP. Simply copy the address of the desired object, put its address in the "obj" textbox and check the "on" checkbox. Unchecking the checkbox will stop putting the object there. "Obj at home" is similar, but this time you provide the addresses of two objects, and the first one will have its position set to the home of the other while the checkbox is activated. "Obj at obj" works the same way and sets object 1's position to object 2's position while the checkbox is activated.

"Scuttlebug stuff" is a tool for manipulating scuttlebugs and the room they belong to. First, you select with the radio buttons the star you're working in. Clicking on "lunge at home" will retrieve some currently loaded scuttlebugs using hardcoded memory addresses (meaning you shouldn't be using this if you have dead scuttlebugs) and set their values so that they lunge towards their home. This indicates that the tool is meant to be used for the A button challenge in these levels. "Get tris" will collect all geometry triangles in the level and store them in a list. "Clear tris" will empty this list. The floor buttons to the right only do things if a BBH mission is currently selected. Even then, most of them don't do anything. But when they do something, they first set every triangle in a specific room to a dummy non-existent room. Next they take all triangles for outside the mansion and convert their room to the room it just erased. Finally, it sets the native rooms of all selected scuttlebugs to the erased room. Needless to say, you shouldn't ever use this tool.

"Tri rooms" is a tool to convert the native rooms of triangles to a specific value. In the "from" textbox, you enter one or more room numbers separated by commas, enter the room number you want to convert to in the "to" textbox, then click the "convert" button. All level geometry triangles belonging to a room listed in the from textbox will have their room set to the new value.

"Goto" is a simple Mario teleporting tool. You set the desired coordinates and click the "goto" button to set Mario's position to there. "Get current" will retrieve his position to set the values, and "paste & goto" will paste the contents of your clipboard into the values then teleport to that location. This only works if you have 3 valid numbers separated by commas in your clipboard.

"Control stick" gives a centralized location for information about the control stick and its effect on Mario's angle. "Raw X" and "raw Y" are the inputs for the rest of the section. You can set them to custom values or check the "use input" checkbox to have the current control stick position be used as input. "Effective X" and "effective Y" are the same as in the input tab, "angle" is Mario's facing yaw, "angle guess" is STROOP's guess for the direction Mario should be moving in relative to the camera's destination angle, "angle intended" is Mario's intended angle assuming no loss of joystick accuracy and "diff" is Mario's intended angle.

"TTC simulator" performs a simulation of the entirety of tick tock clock on its random setting. You set the number of frames you want to run the simulation for in "end frame", and a list of frame numbers separated by commas from 0 to the end frame where Mario generates dust in "dust frames". Click on "calculate" and a new window will pop up after a short while with the results of the simulation using the current RNG value in your Mupen instance. If your Mupen instance is currently recording or playing back a TAS, the starting frame will actually use the current frame in the TAS playback instead of 0, so you need to set the inputs relative to that. The output window will give the current state of every single moving object in TTC, whether that be its position, angle or state. After that, you get the ending RNG value and index, the number of simulated frames and a long string of hexadecimal digits that represent a TTC state. If you remember from the miscellaneous tab, this number can be copied into the TTC state variable to set every object in TTC at once.

"Pendulum manipulation" is a tool to know on which frames you have to generate dust to manipulate a TTC pendulum to go further than it's supposed to. You simply enter an object address to a pendulum in the "pendulum" textbox, enter the number of swings you want the pendulum to successfully do in "iterations" then click "calculate". After a few seconds, a window will pop up listing every frame you need to generate dust on for each swing.

"TTC logger" logs TTC states in a list and tells you if the state you're currently in is in the list. When "log states" is checked, every frame the game is running it will add the TTC state to the list. "Logs" gives the current size of the list, "clear" will wipe the list and text under the checkbox will say "NEW" if the state doesn't already exist in the list and "OLD" if it's already there.

"Schedule" lets you go through a list of hardcoded information packages accompanied with a description. These follow the path for collecting the star watch for rolling rocks in 0.5 A presses. Considering STROOP's history, this wasn't used for the original 12 hour performance, but likely the 5 hour version done a few years later. "Current" indicates the current global timer value and is the only information not provided by the hardcoded values. "Frame" is the number of frame ID of when that moment takes place, relative to some time around when Mario first touches an elevator. Next is his position, vertical speed which is always 0 and his horizontal speed which is rarely set in the data, but when it is, it's -100000 and it stays that value. Clicking the "set" button will advance the ID by one set Mario's values to the ones in the data, and even some other stuff at certain hardcoded frame IDs. The "previous" and "next" buttons allow you to increment or decrement the index in the data without touching Mario. The "reset" button sets the index to the default value of 47 and moves Mario there. WARNING!!!: There is no bounds checking for retrieving the data from the array, so attempting to change the index to below 0 or above 75 will silently crash STROOP, rendering the program frozen in time. Furthermore, you will not be able to close the window through traditional means, being forced to close it with the task manager.

"Invisible walls" is a tool for locating invisible walls with a section of a level. You enter a minimum X, minimum Z, maximum X and maximum Z and STROOP will perform a downwards raycast at every single unit point to check for an invisible walls. You also need to provide a Y value to know from where to start looking downwards, which is important for finding object collision invisible walls or for places with two sets of floors on top of eachother. However if the sky is above where you're looking for invisible walls, you can simply set the Y value to something big like 10000. Checking "only lone points" will filter out any invisible walls point with another invisible wall point directly 1 X or 1 Z away. Diagonally touching doesn't count as touching. Once all that is done, click the "calculate" button to have STROOP find the ceilings or OOB points in that area and give a list in a pop-up window. On the window will be a title indicating how many points were checked and how many ceiling or OOB points were found, and a list of every single found X,Z location. The bigger the checked area, the more time it takes STROOP to search and the more memory the application uses. Be careful with this tool, because setting the search location in a giant out of bounds or ceiling area will cause every raycast to succeed, eating up your computer's memory so fast with the output text it can cause other applications or even your operating system to crash.

Chapter 33: Snow tab

This is a tab to manipulate snow or any floating particles used by the snow system. Snow can obviously be found in CCM and SL, but water particles sharing the same system can also be found in the secret aquarium and inside the ship in JRB.

On the left panel, you can select a specific snow particle using its index in the particle list and clicking "retrieve" to gain control over it. Once selected, you get to use a 3D controller to move the particle around.

On the variable panel, you get copies of the global timer, global timer mod 64, RNG value and index, RNG index mod 4 highlighted in yellow. In green, you get the snow types. Snow type 0 is no snow, type 1 is regular snow, type 2 is water snow and 3 is an unused blizzard mode that looks really cool. All 4 areas' snow types are accessible, but only 1 is used at a time. "Snow array address" is a pointer to the start of the array of snow particle positions. In blue, you get "snow counter" which is the number of currently active snow particles, "snow max" which is the current particle limit, "snow center" X, Y and Z for the center of all the snow. It usually puts itself somewhere near Mario and the center of the screen. "Snow function" is a pointer to the function responsible for updating snow positions.

After that, depending on how many particles are active, you get an X, Y and Z for each particle. These variables come and go as needed. When all 140 snow particles are loaded in-game, it can lag the tab a bit.

Chapter 34: Main save tab

The main save tab is like the file tab, but for the save data not for any specific save file. It's comprised mostly of coin rankings. Just like the file tab, you can select between what's currently in memory and what's actually saved. Here, it's the options in the "struct" section in the top left. Clicking the "save" button will transfer the regular data to the saved data. "Sound mode" lets you choose which sound mode is selected in the settings: mono, stereo or headset. "Language" only applies to the European version, and it lets you select which language to display the text in.

The table on the right side of the left panel holds the coin ranking placements for all the files. In this, the higher the number the better. You can only set these number to 0, 1, 2 or 3, but ties are allowed, in which case the file with the lowest letter takes priority.

For the variables, "current file" is a copied variable from the file tab for reference. The variables highlighted in blue "file A coin ranks" and so on are hex representations of the bit fields containing the placement data for all 15 courses. The 32 bit number is divided into 16 2-bit chunks where the first 15 bit pairs denote a number from 0 to 3 representing the placement, each pair corresponding to a level. "Sound mode" is the number ID for the selected sound mode. Stereo is 0, mono is 1 and headset is 2. "In-game sound mode" is the ID some different parts of the game use. With this, stereo is 0, mono is 3 and headset is 1. The next 3 variables are for the checksum and work similarly to the ones in the file tab, but this time for the main save data. The last 60 variables are just variables representing the file coin ranking placements.

Chapter 35: Painting tab

This tab contains data used for the paintings in the game and their rippling effect. It's a very special effect that has its own rendering code to generate dynamically. On the left panel, you have a list containing an entry for every single rippling painting or wall in the entire game. Even ones you might forget about like the TTM slide entrance. You select the one you want to see the variables of. You should only ever select paintings that are currently in the loaded area, as selecting one from a different area or level will have the variables point to irrelevant data.

For each painting, you have the "ID", which is that specific painting's ID, "face count" is the number of textures the painting uses. Lowering this will remove textures, but increasing it pass its intended size will crash the game. "Ripple type" is whether the painting is using a special texture. This is only 1 for the HMC and COTMC paintings, as their texture changes depending on the angle it's looked at.

"Last floor", "current floor" and "floor entered" all store some form of the ID of the floor Mario is above. You see, the painting doesn't check for its distance to Mario to know when to ripple, but rather there's special floor types on the inside and outside of the painting that indicate what kind of rippling to do and where to do it. Last floor is the special floor type Mario was above on the last frame, current floor is the one he's above right now and floor entered is the floor ID Mario just entered, and 0 if it's been 2 or more frames that he's been above it.

"Ripple status" is 0 when the painting is not rippling, 1 when it's doing a passive rippling because the painting is loaded or Mario is near it and 2 when Mario has entered or exited the painting. "X rotation" and "Y rotation" are the rotation in 360 degree angles around each of the 2 axes. X, Y and Z are the position in 3D space. Because the special floors are what determines when Mario enters or touches a painting, moving or rotating the visuals of the painting does not change its tangibility, but it does change where the camera will go when zooming into it.

"Current ripple magnitude" is the current magnitude of the ripples on the painting. When a ripple is triggered, it gets the value from "passive ripple magnitude" if it's for a passive ripple and "entry ripple magnitude" if it's for an entrance or exit. When the current ripple magnitude goes below 1, the painting stops rippling.

"Current ripple rate", "passive ripple rate" and "entry ripple rate" are the numbers that the current ripple magnitude multiplies itself with on each frame to know how it will decay. The system works just like the ripple magnitude where the current values copies the value from passive or entry depending on the situation. If the ripple rate is less than 1, the magnitude will decrease over time. If it's exactly 1, the magnitude will stay constant. If it's larger than 1, then the magnitude increases exponentially and causes weird things to happen with the painting if you let it grow too long.

"Dispersion factor", "passive dispersion factor" and "entry dispersion factor" control the rate of the ripples. The lower the value, the wider the ripples. Although, due to the low number of triangles making up the painting visuals, you run into a kind of moiré effect at higher values, because the small number of triangles can't convey the larger number of ripples, so you end up with seemingly larger ripples.

Next is the "ripple timer", which the number of frames since the painting started rippling, or at least it would've been that if STROOP didn't completely mess up half of this tab. Here's the problem: In the

game’s code for the painting struct, there are 4 groups of 3 variables for painting parameters. STROOP displays three of them as ripple magnitude, ripple rate and dispersion factor, but it SKIPS OVER ripple decay. Which actually comes BEFORE ripple rate. To make things worse, even though the variable labels skip over these 3 variables, the address offsets don't! Meaning that "current ripple rate" onwards, every variable is actually pointing to an address 12 bytes lower than what it actually should be pointing to. This is why the names earlier didn't make sense. Ripple rate isn’t giving the ripple rate, it’s actually the ripple decay! And dispersion factor is the real ripple rate! So what does that make the ripple timer, "horizontal ripple spot" and "vertical ripple spot"? Well, they’re the dispersion factors! The dispersion factor simply says how fast the magnitude should decay depending on how far away a ripple is from the ripple center. The bigger the number, the further ripples can go. Luckily, STROOP intentionally skipping over some graphics pointers syncs the variables back up correctly for ripple trigger onwards, but that's still nine variables incorrectly labeled and 3 variables that are supposed to be accessible that are missing. If you want to mess with the ripple timer and ripple X and Y, go to the generic variable right click menu by right clicking on empty space in the variable panel, select add custom variables, and in the new panel, change type to float, change base to painting and set the offsets to 0x4C for the timer, 0x50 for the ripple X position and 0x54 for the ripple Y position.

"Ripple trigger" is whether or not the painting should be continuously rippling. It’s 10 if not, 20 if yes. "Brightness" is the opacity of the painting. 0 is invisible and 255 is fully opaque. "Last Mario under painting", "current Mario under painting" and "new Mario under painting" act similar to last, current and floor entered, but this time for if Mario’s Y position is below the bottom of the painting. 0 for below and 1 for above. "Size" is the size of the painting along one axis in units.

Chapter 36: Sound tab

This is a music and sound effect player. On the left tab, you can play any music track in the game by selecting it in the list then clicking the "play music" button below, half covered by the bad UI. To the left of the button will be the song ID of the currently selected song.

On the right panel is every single sound effect in the game. Due to how many there are and how hard it is to describe most of them, they're labeled with their hex ID. Just like the left tab, when selected its full ID will appear at the bottom next to the button "play sound effect", which plays it. Note that the ID is not an index, but rather a bitfield of information about the sound, like what sound bank it comes from, how to play it and any effects it should have. SM64 sound effects are divided into ten banks. Only one sound effect can play per bank at once. If a sound from a bank is playing and another from the same bank with a higher priority gets called, the previous sound effect stops playing. You can see which bank these sound effects belong to by checking the first 2 digits of the ID. They range from 0 through 9.

Furthermore, there are two ways that sound effects can be played. Most play the way you would expect; when the code calls for the sound to be played, it plays. But many other sound effects are for events that the game doesn't know how long they will take. Like, how long should a bob-omb's fuse sound effect last for? Mario kicking it can blow it up any time, and Mario re-grabbing it can prolong its life! The game's solution to this is that certain sound effects play continuously in a loop, and the caller will call the sound effect every frame or so, and when the caller stops calling, the sound effect will stop. SM64's default behavior to these and how STROOP does it is that the sound effect just goes on forever until it gets interrupted. So for many sound effects like a bob-omb's fuse you'll have a hard time getting rid of when playing them this way. Some days you just can't get rid of a bomb!

Chapter 37: Search tab

Yo dawg, I heard you liked memory watch programs, so I put a memory watch program inside your memory watch program! This tab functions like a more general memory watch application like cheat engine, and lets you experience the fun of finding variables yourself.

To search through memory, start out by selecting what variable type you're looking for with the drop down menu in the top left, just below that, you select which comparison type you're making for each memory value. Most options are self-explanatory, but here are the more interesting ones: "changed" is used for secondary scans, where you want to only keep the values that have changed since the last scan. "did not change" is the opposite. "Increased" and "decreased" also are used for secondary scans, where they only keep values that have increased or decreased respectively. These last 3 do not use the parameter below in their comparison. "Increased by" and "decreased by" are similar to the previous ones, but these take the parameter into account to know by how much your target has changed by. "Between exclusive" and "between inclusive" take 2 values separated by commas and search for values between those 2 values, exclusive not including exact matches and inclusive doing so. "Is integer values" succeeds for integer types and only succeeds for floating point types as well if they represent a whole number. "Has sensible magnitude" succeeds for integer types but only succeeds for floating point types if their absolute value is between 0.001 and 1000000 non-inclusive. This is to filter out reading non-floats as floats, which often produces crazy scale numbers. "Everything passes" does as it says. Every value passes. This is simply here as a quick way to refresh all currently captured values to a new value without performing a new search. Those last three search types ignore the parameters.

After having selecting your search type, you're very likely to need to enter one or more search parameters in the textbox below. To search with a hexadecimal value, have the start of your parameter text with "0x". If your search type requires two values, separate them with a comma. You can search for multiple values at once by separating them with a semi-colon, but unfortunately this technique only searches for places where those values appear consecutively in memory, so you can't search for two completely different things at once.

To perform a new scan once all the search parameters are set up, click the "first scan" button. This will scan all of the N64's memory. 0x400000 bytes normally, 0x800000 if you have extended memory enabled. Depending on the parameters of the search, the first scan can take anywhere from 2 seconds to 15. STROOP will be unresponsive during that time, but luckily you're provided with a progress bar showing progress through memory and how many bytes are remaining to search out of the total. Once done, the number of results will appear under the buttons. If this value is above 10000, nothing will be displayed in the list below and you will have to refine your search to be able to see its contents. To undo the scan and return to your previous candidate table, press the "undo scan" button. You can only go back one step this way. To empty your table, click the "clear results" button. This action alone cannot be undone. Clicking the undo button will simply bring you back to the previous table, not the one you just deleted.

The key to memory searching is to perform multiple scans. Once you've cast a wide net, move around a little in the game, then change the parameters accordingly and click the "next scan" button instead of the first scan one. This button lets you search only within the variables currently in your table, refining your results. If you keep searching while tweaking various different things in the game, you

will eventually lock in on the value you're looking for. Depending on how easy your target is to manipulate and how easily you can know its value, the difficulty of searching for variables can be tedious. But once you’re gotten your one or multiple candidates in the list below, you can move on to analyzing them manually.

Once your table is 10000 results long or less, you will be able to see all of their values individually. Each row will contain that variable’s address in memory along with its value. To get more control over them, select the ones you want to look at closer, using control and shift to select multiple at once, and click the "add selected as vars" button to convert those memory addresses to variables. You can do this to all variables in your table with "add all as vars". This obviously only works if you have no more than 10000 variables. However, I recommend you don't add more than 300 variables to your variable panel, as STROOP will happily eat up your computer’s memory until your pagefile bloats up like a tumor or you simply blue screen. All added variables will have the default name of "[type] [memory address]".

Chapter 38: Cells tab

It would be quite silly to have to check for collision with every single triangle in the level at once when checking to see if Mario has hit something. SM64's solution to this is to divide levels into cells. Each cell covers a 1024 by 1024 unit area. Each cell contains a list of every floor, wall and ceiling contained within it. Triangles that span over multiple cells are found in both, meaning the cell graph contains vastly more triangles than the level model itself. There are always exactly 256 cells covering all positions from -8192 to +8192. This limitation is the reason why in levels like SSL, even if the collision triangles extend outside of this boundary, they're still treated like out of bounds. There's no cell data past 8192 units in each lateral axis.

To build the cell graph, simply click the "build tree" button. A tree will be built containing all cell data. The data is divided into two categories: static triangles and dynamic triangles. Every non-triangle node contains a number in square brackets indicating how many triangles it contains. Dynamic triangles only contains the triangles of currently tangible objects. Both trees are built the same, but for the purposes of explanation I'll be focusing on the static triangles tree, since it's more populated.

The root contains 16 children, one for every row of cells along the Z axis. All 16 Z groups are labeled with an index and the ranges of positions they represent. Expanding one of these gives all 16 cells layed out on that Z axis range. Expanding one of these cells will separate the walls, floors and ceilings. If those lists are empty, that branch ends there. But if they contain triangle data, you can go one level deeper to list all walls, floor or ceilings for that cell. The triangles are labeled with their memory address and the vertical height of their first vertex, which is how they're ordered in in the cell. If you've recently seen Pannen's video on invisible walls, this might ring a bell. Select any of these triangles to see the variables on the right side come to life.

"Current address" is the memory address of the selected triangle. The variables highlighted in yellow are copied over from the Mario tab and are Mario's currently referenced wall, floor and ceiling. The variables highlighted in grey are the ones unique to the cell tree. "Level node count" is the number of static triangle nodes, "object node count" is the number of dynamic triangle nodes, "total node count" is those two numbers added together. "Mario cell" are the indices of the cell Mario is currently in. "Cells" contains the ranges of cells that the currently selected triangle is located in. As noted earlier, triangles that span over cell boundaries are located in both. Triangles being in multiple cells at once is the reason Mario doesn't have trouble colliding with stuff on cell boundaries. Cells check within 50 units of their borders for extra triangles. This usually catches weird collisions shenanigans, but the way this was implemented leads to some issues. For example, there's a wall in the water near the tiny piranha plant in small THI that is positioned in such a way that the 50 unit buffer is not enough. Try and find it yourself!

As for the remaining variables, they're just copied over from the triangle tab and give info on the selected triangle. Nothing new.

Chapter 39: Music tab

Not to be confused with the sound tab, this tab actually lets you analyze the variables that go into controlling the sound that's playing. SM64 operates using three music tracks. 0 is for the main music, 1 is for additional music tracks like a success jingle, and 2 is for combining the sound together. 1 is usually inactive and 2 works in weird ways and I'd recommend staying away because if you poke it wrong the sound just stops. So I'll simply be assuming the use of the sequence player with ID 0. SM64 uses a MIDI-esque format to store music to save on cartridge space, so this tab controls that.

Unfortunately, many of these variables cannot be updated after they've been initialized, so they're more or less in read-only mode. If changing a variable does nothing, don't be surprised.

"Enabled" is whether or not the track is currently playing. Turning it off pauses the music. "Finished" doesn't do anything. It's never read by the game. "Muted" works in a similar way to enabled, but in the code it's used when the music plans on coming back. It doesn't mute the track, it pauses it. "SeqDmaInProgress" and "bankDmaInProgress" indicate that music data is being fetched from memory. When these are activated, music is automatically paused. "SeqVariation" is used by exactly one music track: the game over theme. It allows it to reuse the notes from the regular title theme and simply perform an audio effect on them. "State" indicates the state of the music. Most IDs are undocumented, but 1 indicates a fade out. "NoteAllocPolicy" indicates to the code where to look for the next notes in memory using IDs. "MuteBehavior" tells the song how to behave when it's muted. 128 will stop the music script, 64 will stop playing notes and 32 will simply change the volume, as seen when pausing the game in the castle lobby. "SeqID" is the ID of the track as seen in the sound tab. Modifying this value while it's running will mess everything up and corrupt the sound system in some way, so don't touch it. "LoadingBankNumInstruments" and "loadingBankNumDrums" indicate the number of instruments and drums loaded from a song file. These are usually 0.

"Tempo" is the speed of the music. On the American version, it's roughly equal to the BPM of the song times 50. You can change this variable to change the speed of a song. "TempoAcc" is the remainder of the tempo beats for a frame. This just keeps track of how many tempo beats were left to go when the last music frame finished. "FadeRemainingFrames" is the timer used when the music is fading out, like exiting a level. "Transposition" is interesting. It's rarely if ever used by the game, but changing this variable pitch shifts the song up or down. The pitch shifting is quite intense, so I recommend not going past the range of -20 to +30. "Delay" is the number of audio ticks left until the song resets or moves on. It's sort of a timer ticking down. It doesn't control the position within the song, just how much time is left before something happens. "FadeVolume" is the music's volume. This is the same variable that could be seen in the miscellaneous tab. "FadeVelocity" is the speed at which the music fades when fading out. "Volume" is a fake volume variable that doesn't do anything. "MuteVolumeScale" is the number the volume multiplies itself with to know by how much to lower down when the music is set to lower its volume when muted, like when pausing in the castle lobby. You can set this variable to 0 to have it be muted, 1 to have no change or even above 1 to have the music get louder when muted.

Chapter 40: Script tab

This tab lets you run your own JavaScript code to interact with variables yourself. Luckily for me, this tab comes with instructions accessed by pressing the "instructions" button. They read the following:

"The Script Tab can be used to set variables in a custom way defined by you. Specifically, you write JavaScript code on the left, which can both read from and write to the variables on the right. So if you want to read from or write to a variable, you must first add it to this tab.

Within your JavaScript code, there are 2 implicit objects that you have access to. The first of these is INPUT, which can be used to read from the variables. For example, to get the value for Mario's X position, just write INPUT["Mario X"]. The second of these is OUTPUT, which can be used to write to the variables. For example, to write a value v to Mario's X position, just write OUTPUT["Mario X"] = v.

You can run your script continuously (runs once per STROOP frame) or just once. To see some examples of scripts you can write, click on the Examples button."

I highly recommend going through each example the tab offers by clicking the "examples" button and selecting each option one after the other. It helps give a feel for how the tab works. However, these instructions and the examples leave out multiple important things. They tell you how to use the INPUT and OUTPUT maps, but they leave out two other keywords.

LOADFILE can be used to load code from a file on your computer. It works very similarly to "#include" from C and C++, where before the code runs, a pre-processor replaces the statements with the text from the given file. This keyword's syntax is "LOADFILE" followed by a space then simply the name of the file if it's in STROOP's local folder, or an absolute file address. Examples: "LOADFILE test.txt" "LOADFILE C:\Users\name\Documents\test.txt". A LOADFILE statement MUST be on a line of its own. There can't be any text before the LOADFILE statement, and any text after the file path on the same line will be overwritten.

Second, the textbox located below the code box is the console box. Console.log doesn't work here, so you need to interact with the console another way. CONSOLE is a keyword that acts like a list variable, where you can retrieve a specific line from the console with the syntax "var x = CONSOLE[index];". You can equally print to the console with the syntax "CONSOLE.push(x);". This will print whatever X is to a new line in the console. Because CONSOLE is just secretly a STROOP-made variable, you can also get properties from it such as CONSOLE.length to do stuff such as retrieve the last line written to the console. Keep in mind that the console can be written to before running a script to pre-load data.

For some reason, you're also able to paste images in both the code and console boxes but uhhh don't do that, that shouldn't even be possible.

Chapter 41: Warp tab

The final tab in STROOP as of writing is used to control the teleporters sometimes found within a level. More specifically, it offers a tool to hook up teleporters you have duplicated from existing teleporters in the level or that you have brought yourself from other levels. This tab also contains instructions on the operations accessible by clicking the "instructions" button, and they are the following:

"The "Hook Up Teleporters" button can be used to enable teleporters in courses without any teleporters. To use it properly, follow these instructions:

First, go to a course with working teleporters, like BoB. Select 2 teleporters that link to each other. Right click on one of the slots, and click "Copy Object" with control held. Go to the course where you would like to place the new teleporters. Select the first 2 vacant slots. Right click on one of the slots, and click "Paste Object" with control held. Press the left arrow button (on the top right) with control held until the teleporters are at the end of the dark blue slots. Click the "Hook Up Teleporters" button.

Now the teleporters should link to each other, and you can place them wherever you want."

This tool is not quite refined, so it's very possible for the hooking up tool or simply loading this tab to crash either STROOP, SM64 or both. STROOP scans the game for teleporters or warps when the tab loads or a new area is entered, so this tab very often lags STROOP for several seconds.

As for variables, the ones highlighted in blue are the global ones. "gAreas" is a pointer to a list of all areas and their data, "current area index" is the current area index, "warp nodes address" is a pointer to the first warp node in the linked list. "Num warp nodes" is the current number of loaded warp nodes.

After that, every single teleporter or warp or Mario spawner in the level has 6 variables associated to it. Every warp has their variables starting with "warp X" with X being the index in the warp data. "ID" is the single byte ID of that node. "Dest level" is the destination level, marked with a level ID. "Dest area" is the destination area in that level, marked with an ID. "Dest node" is the ID of the destination node. If the warp takes you to another level, the code expects that area to contain a warp node with this ID. "Object" is the associated object of the node. "Next" is a pointer to the next node. A null pointer indicates that this is the last node.

Oh and to help with setting up the teleporters in this tab, clicking on object slots while in this tab will not send you to the object tab.

Conclusion

A word from the author:

I started this project not knowing how long it would take or what it would be, but still took at least three hundred hours' worth of testing, re-testing, searching through undocumented STROOP code, searching through decomp code, running STROOP through a debugger to catch hidden features or mistakes and finally writing down the results and observations. STROOP was designed to not have everything known by everyone, but I made this document because I didn't like that. STROOP is the most powerful tool I've ever seen for a single game, and it didn't deserve to go undocumented. I still feel like even after these seventy thousand words, I missed hundreds of hidden features. This document will have constant room for updates, and a fork will have to be done for the Frame refactor quickly rising in popularity. (Not by me though, I'm tired.) I sincerely hope that this causes many more people to use STROOP to mess around with SM64 or even helps established TASers be more efficient at doing what they do. I'd like to thank Pannen for answering all the questions I bombarded him with, making some of these sections much easier or even possible to write. I'd also like to thank Crackhex for motivating me to start writing this and for helping guide the project with feedback. Thank you as well for taking the time to read this document and this note.

- SuperM789

Full credits:

- **SuperM789**: Main writer, researcher & designer.
- FramePerfection: Verifier and fact-checker for the entire document.
- Pannenkoek2012: Provided generous help with questions, lead developer for STROOP.
- Crackhex: Provided feedback, contributed writing adjustments to intro.
- Marbler: Answered question about meaning of BITFS pyramid map tracker.
- Danebou: Answered question about meaning of behavior value in object tab.

Version history:

- 22nd of May 2025: Document created
- 6th of July 2025: Main content finished, double checking for hidden features
- 8th of July 2025: Content complete, commencing verification
- 20th of July 2025: Chapters 0 to 16 verified
- 5th of August 2025: Chapters 17 and 18 verified
- 16th of August 2025: Chapters 19 to 41 verified
- **16th of August 2025: Version 1.0 released**
- 27th of August 2025: Fixed mistake in Debug tab chapter
- **27th of August 2025: Version 1.1 released**
- 13th of Novem. 2025: Updated Mupen link
- **13th of Novem. 2025: Version 1.2 released**