|  |  |
| --- | --- |
| **Programs:** | **Where to Download** |
| Emuhawk/Bizhawk (You’ll use Bizhawk) | <http://tasvideos.org/Bizhawk.html> |
| Toomai’s SSB64 Moveset Data File | [http://n64vault.com/ssb-tools:toomai-s-moveset-spreadsheet](http://n64vault.com/ssb-tools%3Atoomai-s-moveset-spreadsheet) |
| HXD | <https://mh-nexus.de/en/hxd/> |
| The GE Editor | <https://github.com/carnivoroussociety/GoldEditor> |
| Madao’s PSA 64 (Optional) | [http://n64vault.com/ssb-tools:psa-64](http://n64vault.com/ssb-tools%3Apsa-64) |

**Step 1: Animation Selection**

The first step in modifying a move, is to determine if you want to change the animation, as this will dramatically change the next steps in your process. You can easily swap most animations a character has for another animation that character has. However, sometimes animations for other characters work as well (but many will crash, fail, or look absurd). To change an animation, you must first determine what animation you are currently on. Most hack testing is done in Dreamland Training Mode, that’s where we have the most documentation. The Training Dreamland “current animation” RAM address is: 80190F62 (Bizhawk cuts the 80 and just does 190F62).

Add this address to our ram watch list (it’s in tools in Bizhawk). It will show the ID of whatever animation you are currently in. So, use the attack that you want to swap out, write down that number. Let’s take Ganondorf’s forward air from Smash Remix for instance. I swapped Captain Falcon’s forward air animation for DK’s aerial punch. Find out what animation number is the original forward air (667). Then discover what DK’s animation ID is (3AB).

**Step 2: Animation Swapping**

Now that you figured that out, you can swap them out in Bizhawk’s hex editor (in tools). Search for the forward air animation ID (00000667) within the RAM. It should be in a list of each animation ID in hexadecimal order (A-F comes after 1-9), it may not be the first 667 you run across, but it will be obvious when you get there. Now, write in the animation address the animation you want. Do this by poking the “word” (a word is 8 characters of hex) in which 667 is in. This is done by right clicking and selecting poke. Write in the ID you want instead (000003AB). Does it work like you wanted? If so, let’s make a save state. The changes from here on out will require testing and you may want to come back to this point. Make save states often for different steps in your process.

To make such an edit permanent, you’ll need to open your Smash 64 ROM in HXD. Search for the area of code around the original ID. Then replace the ID with the new one you prefer.

NOTE: Many moves are animation dependent. Moves like falcon kick only work with certain moves and generally don’t work well with swapping.

NOTE: It’s a good idea to pick a bunch of potential animations, as most will not work right.

**Step 3: General Moveset Editing/Hitboxes**

You have your new animation, but the substance of your attack is still the same as before. This is going to require considerable changes. Please open the SSB64 Moveset file. This is file is insanely handy. If you make changes to the code, it will show you how it affects your attack. Go to the character you are editing on the tab list (Captain Falcon). Then go down to forward air. It’s at 9494, this plus 802F, will give you the RAM address in Dreamland Training (802F9494). Ignore the 202F on the chart, no idea why it says that. And once again, Emuhawk cuts off the 80 in its RAM addresses, so it’s just 2F9494.

You need to think about how you want your attack to work. It’s often a good idea to look at attacks currently in the game that work similar to yours in order to get yours to work appropriately. You can also look to later games for things like damage, knockback and otherwise. Though the games function differently, the base calculations are often very similar. Another fantastic tool for calculating hitboxes and other commands is Madao’s PSA 64, which automatically generates the desired code and effects).

It's important to know that the moveset file is written in a sort of microcode, with commands (there’s a list in the lookup table and PSA can generate some). The first two numbers of a word dictate the command (commands require different lengths of code, so using the spreadsheet can be tricky if you need to place a GFX where SFX was or similar). Oftentimes it may be best to write out your code in HXD.

In my hack, I knew Ganondorf generally has higher knockback and damage, so let’s increase those. Double click damage and it will show which box changes that. You’ll have to experiment with each number to see what affects what. Do the same with knockback.

One of things you’ll notice quickly is that your attacks aren’t connecting right with your character’s model. The reason is, each attack is connected to a bone in the characters model. The original attack is connected to the leg bone. We’re punching now so that’s no good. We need a fist bone. You’ll see bone is one of the categories in the spreadsheet as well, click it to discover the box/word that changes it. You could trial and error this to figure out which ID stands for which bone, but it’s better to skim through the spreadsheet and look for an attack connected to the same bone as you want. Falcon’s jab is connected to his fist. It’s number (14/15). Let’s put that in, one for each hitbox (You’ll have to experiment again to say what numbers make bone ID 14 or 15 or look to jab for guidance). Okay, now it’s connected to the fist. Does your attack hit where you want it? You might need to move the x, y, and z locations if not. A good way to truly see it is to turn on the hitbox Gameshark code (800D81BD 0049). This will show your character’s hurt box and hit boxes. Very helpful to truly see it.

Now we have other problems. This attack hits twice originally, because Falcon swings both legs. Not so with the new animation. We can delete the second hitbox pair of hitboxes. This gives us space to breath if we want to add special effects. We don’t in this situation though. Make sure you always end your hitboxes with an 18 command (18000000). The rest of the code should be 0’s until the next move.

You can now paste this into Bizhawk’s hex editor starting at 802F9494. Test your new move in RAM, tweak as you like it.

NOTE: if you start one of your words with 00, it will probably end your move and everything after it will be ignored. However, part of the hitbox commands, GFX and other commands are often all 0’s, don’t worry about those and don’t remove those (you can modify their effects if desired, but do not remove that word).

NOTE: This microcode is actually in binary, but we’re working in Hex. That means if you alter one number in the code, you can alter other parts of the attacks effect. This is particularly noticeable with knockback angle (if you were to have the 3rd digit be anything but 0, 4, 8, or C you would start changing the highest bits of the knockback scaling). Change damage and you also change clang (every number goes up 1 clang, then goes up one damage). The spreadsheet doesn’t always capture this, so you’ll have to take it slow and playtest.

NOTE: The number or order of the moveset ID will dictate which hitbox will override the other hitbox if two hitboxes overlap.

**Step 4: Timing**

You need to make sure your hitboxes windows open and close at the appropriate time. Two commands control this, the After Command (08), which comes at the start and the Wait Command (04), which comes anywhere else. You’ll want to increase or decrease these so they time up correctly with how you envision your attack. A hitbox will only be open for as long as the wait command after it lasts. Commands won’t start until after the After command ends.

**Step 5: Advanced Changes**

The most challenging move change I made was the Volcano kick (Fray mostly did this one).  You should get comfortable with Toomai’s SSB64 spreadsheet before you try something like this. It does no damage until it’s explosion, so this kick has to be altered significantly. However, since we’re cutting out hitboxes by a huge amount, that gives us plenty of space to breath as we are only limited to the space before the next attack begins (though there are go to commands if there is space in another location (90), but these are a bit tricky to use). Remove all your hitboxes but one. There’s only one for Volcano Kick. Now that we’ve deleted that…

**Step 6: Sound and Special Effects**

Adding sound effects is a bit tricky. There’s probably a better way to do this (there’s always a better way). But you can always refer to other attacks as reference or experiment to find the right sound. There are a few commands that create sound effects, one that stops upon connection (4C) and one that always goes through to completion (38) and one that generates character voice effects (44). We don’t want our sounds to stop at any part. So, we’ll use (44) because we want our boy to give a nice grunt. So, let’s move the After command and just start with the grunt (44 00 01 52). You’ll have to experiment or look at other moves to determine sound effects and voice effects. Now let’s put the After command right after the voice clip (08 00 00 0D). We want a whoosh type sound to signify the moving of his leg (38 00 00 2A). Then let’s wait to time up the explosion/hitbox (04 00 00 08).

Let’s add our explosion sound we want a 38 command because we want it to sound off regardless of connection (38 00 00 2A). Now let’s paste our hitbox after that.

We’ll need to make it a fire type move, so click that box to see which word modifies that. Experiment till you start changing the effect. Let’s also ramp up the damage and knockback.

We don’t want this connected to a bone like normal, so set to bone 0. We’re positioning it by the X, Y, and Z positions. Y=125 Z=400.

Now let’s add a graphic effect. I found mine by looking at our boy Starfox. I knew his up special had a nice explosion effect, but you have got to make sure it’s in the right position (98 00 74 00 00 00 00 00 01 A0 00 00 00 00 00 00). This will take experimentation, because the spreadsheet doesn’t capture positioning. Know that some parts of the GFX codes will affect size and location of an effect. Depending on where you edit, it will either be in a specific location or within a random area within a range. It’s frustrating. Now we need a sound effect that booms, let’s go with (38 00 00 01). Lastly, this is a tremendous attack, we need it to shake the earth. Let’s add a shockwave GFX effect. (9B F8 84 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 06). This type of effect will shake the screen to make it seem epic!

At the end of everything your code should look something like:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Voice | 44 00 01 52  |  |  |  |  |
| After | 08 00 00 0D  |  |  |  |  |
| SFX | 38 00 00 2A  |  |  |  |  |
| Wait | 04 00 00 08  |  |  |  |  |
| Hitbox | 0C 00 02 D1  | 01 5E 00 00 | 00 7D 01 90  | 5A 46 E0 03  | 05 42 19 00  |
| SFX | 38 00 00 01  |  |  |  |  |
| GFX | 98 00 74 00  | 00 00 00 00  | 01 A0 00 00 | 00 00 00 00  |  |
| GFX | 9B F8 84 00  | 00 00 00 00  | 00 00 00 00  | 00 00 00 00  |  |
| Wait | 04 00 00 06  |  |  |  |  |
| End Hitbox | 18 00 00 00 |  |  |  |  |
| End Move | 00 00 00 00 | 0 to repeat until end of attack |  |  |  |

NOTE: Certain commands require more codes than others. Keep this in mind when you write out your code, because it must be equal to or larger than the original code (unless you utilize go to commands). This can make using the moveset tricky if you’re switching out a lot of commands of different lengths in the spreadsheet. The more complex the code, the better to just write it out in HXD instead.

You should test out your changes in RAM, like we did with animations. Search for the relevant code and replace. As you continue to successful change your code, make new save states so you can always go back.

NOTE: All moves must end with “00000000”

**Step 7: Inserting your Code**

Once you iron out your code. Open up GE Editor. Go to Preferences in the tools tab and set your Smash 64 Rom as your default game. Now go to Game Configuration in the Tools tab. Click the files arrow and scroll till you see EB, this is Captain Falcon’s moveset file (Moveset File list below). Save this to a folder. Open it with HXD. Copy the original version of the code, either from an unedited version of the SSB64 moveset file or from RAM. Now search (ctrl+f) and paste in the code in the search bar. Make sure you are searching for hex values, not text. Find the beginning of your attack code. Now copy your new code and paste (write paste, not insert paste. We cannot expand the length of the code!). Save a new file. Go back to Game configuration in the GE Editor and click inject file. Select your new file. Click write rom and save as [name].z64. You’ll now want to give your ROM a new CRC so that it can be playable. Open the GE Editor and go to tools, then Trim/Flip Rom Tools, select recalculate checksum, then select your file. Now you’re done!

**Step 8: Finding Space**

You may struggle to find space for code if you want it to do more than it previously did. An attack cannot continue past where it originally ends or it will conflict with the next attack. However, there are ways to circumvent this. One of these is a GOTO command. You can use the GOTO command to direct to an area of code that you no longer use. For instance, in my Ganondorf mod, I no longer used the huge amount of space dedicated to his jab combo commands (this was done by modifying his combo attribute). So, I used that space for the rest of my code. A GOTO command looks like this if accessed in RAM:

90000000 [insert RAM Address here]

However, that’s not how it looks when you open the file. You need to write it in a kind of shorthand that the game’s code uses so it can place the code we’re it needs to be in any situation. This is a good time to note that the code moves around depending on stage, characters, and mode. A ram address will not stay the same in each match, it will move. The game needs to have a frame of reference. The way the code deals with this is by looking at the location of the code within the moveset file you edited in Step 7.

The method of short hand works like this:

90000000 XXXXYYYY

XXXX \* 4 = offset of next pointer from start of file

YYYY \* 4 = offset of data from start of file

Now, let’s break this down.

Determine where in the file you want your code to continue. Let’s say that’s at address 1DF0 of the file. You will divide that number by 4 (in hex). You should get 77C, that goes at the end of the word next to your command so…

90000000 XXXX077C

But that’s not all you need to do. The way these commands work is connective. That means that the GOTO or SUBROUTINE command immediately before refers to the next code. So, you will need to find both of those locations (not always adequately captured in the spreadsheet, so you’ll have to scan the file yourself). Look for the address of the next GOTO or SUBROUTINE. You’ll need to point to it as well. Let’s say the next GOTO command is at 2040. You should divide by 4, that will give you 810. IMPORTANT: You are directing to the location AFTER the command. Do not place the pointer to the “90000000”, direct it to the word after the “90000000”. So, our code should be:

90000000 0810077C

Now that you’ve done that, you’ll have to edit the previous SUBROUTINE or GOTO command as well. Just like you did for yours, you’ll need to place some new digits in that command. Let’s say you placed your GOTO command at 1F70 (once again this is the address of our “0810077C”, not “90000000”. Divide that number by four to get 7DC and place that at the beginning of the previous GOTO or SUBROUTINE command.

Another option is simply expanding the size of the file. This is especially easy for the move that is at the end of the file. This shouldn’t cause issues as long as you are not trying to expand it to a great degree. However, this option has not be experimented with to a great degree. To do this, simply keep writing code after the end of the file, it is as simple as that.

The third option is to just move the start of the move to another location, see step 9.

**Step 9: Redirecting a Move**

Right next to every animation for a moveset (as we discussed in Step 1), there is a pointer to the moves functionality in the file we just edited. Unlike Step 8, it is not in shorthand. So, if your code begins at 1DF0, you’ll just write 1DF0 in the spot. This can be useful to give a different hitbox to different moves. For example, falcon kick uses the same hitbox for aerial and ground, however it need not be the same since they are both different moves. You can redirect this move by going to its location in the ROM itself (not the file we’ve been working on, the actual Smash 64 ROM now). Find the location of the animation. The last four digits of the next word will direct it to the portion of your ROM. Overwrite those last for digits to wherever you placed your new or moved location within the moveset file. It will start reading the attack from that location.

**Step 10: Action/Function Table Editing**

Every action in the game has a function table which affects the nature of the move and the flags used within them. It is a list of four words, that are organized thusly:

FUNCTION TABLE:

0x???????? // main subroutine/ending transition

0x???????? // interruptibility/other(often unused)

0x???????? // movement/physics subroutine

0x???????? // collision subroutine

To find the location of the functions for your move, you will need to set a “breakpoint.” A breakpoint is a more advanced hacking technique which freezes the game and shows the currently executing portion of the code and what information is stored in its registers. This can be done in Nemu and Project64. In Project64 you will need to enable Debugger in the advanced options of the options menu.

You’ll need to set a breakpoint at the RAM location 800E7448. This can be done by clicking debugger🡪breakpoint🡪then pasting that address in the search bar. Double click that address so it becomes red. Now whenever a new action is performed by a character, it will “break”. On the right side you should see a list of registers. The one labeled S0 will have the address of the function in question (ignore the first word of the register, which is probably FFFFFFFF).

These functions within the table you just located can be swapped to a variety of effects. This can be done to allow a move to transition to another action, if you correctly swap the main subroutine/ending transition from another function. Similarly, you can swap the second function to cause interesting effects. These effects are associated with moveset flags, such as the 57000001 flag used by Link during his Up Special. For example, in order to give link the ability to be able to change his orientation with using his Up Special (flip from left to right or right to left), you can swap in the interruptibility function from Captain Falcon’s Up Special (80160370). This will have other effects that cause Link to fly higher during his Up Special. You will have to position the aforementioned flags in order to work appropriately with the move you are modifying. More advanced modders can use these function tables to create custom functions through the use of assembly (not covered here).

**Step 11: Playtesting**

You’ll need to playtest this a lot to see how it works. Get the opinions of others, play competitive matches against friends.

**OTHER:**

There’s still plenty of other things to mess with and experiment with in the spreadsheet, including things like severity, knockback angle and the different types of knockback. You’ll just have to experiment and see how they work.

There also plenty of other commands, which are listed in the Lookups tab in the spreadsheet. I’ve not come close to covering them all. For example, a loop command can be used to open and close hitboxes repeatedly, like Mario’s down B functions. There are tons of things to learn and do with this spreadsheet. Many interesting options are available, you’ll need to experiment to learn how they work. This website has a list that may provide explanation for commands: [http://opensa.dantarion.com/wiki/Events\_(64)](http://opensa.dantarion.com/wiki/Events_%2864%29)

**ATTACK SPEED:**

You may be wondering how to change attack speeds, that’s not feasible here. I did that through assembly hacking. That will be covered in another tutorial.

**PROJECTILES:**

There is far less documentation on projectiles. They are located in other compressed files in the ROM. There is a similar, though far less expansive spreadsheet for projectiles. They do not function under the exact same principles as normal moves, but have some similarities.

**Moveset Files:**

CA - Mario hitbox data

D0 - Fox hitbox data

D4 - DK hitbox data

D8 - Samus hitbox data

DC - Luigi hitbox data

E0 - Link hitbox data

E4 - Kirby hitbox data

E8 - Jigglypuff hitbox data

EB - Falcon hitbox data

EE - Ness hitbox data

F2 - Pikachu hitbox data

F6 - Yoshi hitbox data