### **Preface**

This tutorial will go over some game specific processes that allow objects to interact with mario. It will also go over mario himself and how to setup behavior scripts from the point of view of the object struct.

### **Mario struct**

Like objects, mario has a struct that determines how he functions in game. Mario’s struct is located at 0x8033B170 in ram always. If you want to control mario you will go through the same process that you use to control objects. Access the struct, load, edit then store variables inside it.

Most struct values are self explanatory and I will not go over but I will focus on 0xC, mario’s action. The action is just the state mario is currently in. The game controls most of mario’s properties by managing his action. For each action the physics of movement and possible controls will change. Each action has its own routine that will usually write to the other values in the struct based on controller input. This can be seen with jumping/running actions having different effects on speed with the same controller input.

If you change mario’s action, he will go into the new routine associated with that action. Changing action in the wrong scenario will cause a softlock while other times it will do nothing, but doing it correctly will allow precise control over mario. You will have to experiment with the values to figure out what is possible.

Ex. 1 Mid air jump

* LUI T9, 0X8034
* LW T0, 0XB17C (T9)
* ANDI T1, T0, 0X1C0
* ORI AT, R0, 0X80 //IN AIR
* BNE T1, AT, END
* LH T2, 0XAFA0 (T9) //BTN INPUT
* ANDI T2, T2, 0X8000 //A BTN
* BEQ T2, R0, END
* ORI T3, R0, 0X20
* LWC1 F0, 0XB1BC (T9) // y spd
* MTC1 T3, F2
* CVT.S.W F2, F2
* ADD.S F2, F2, F0
* SWC1 F2, 0XB1BC (T9) //y spd
* END:
* JR RA
* NOP

Here I add y speed to mario when you press A in the air. First I check mario’s action. Actions are separated into groups. You can find the group by *AND’*ing the action with 0x1C0 (0x80 is the “in air” group). Next I check the current button input which is at 0x8033AFA0, and if it is not A (0x8000), then I don't increase y speed. Afterwards its as simple as loading, adding then storing a float.

### **Interaction**

Interaction is how objects can affect mario without directly editing his struct. By setting the interaction type you can have objects do many different things very easily. For example you can set an interaction to burn mario on contact, or to be collected like a coin. Setting the interaction will be done usually in the behavior script of an object.

### **Behavior scripts**

Behavior scripts setup how an object behaves, but now that you understand the object struct I can give a more in depth description. The behavior script is a quick way to initialize or manage the object struct every frame. You tell the game which struct variables will be used and which functions that object will use.

Being proficient at behavior scripts is as simple as understanding the object struct, and knowing what values you need. Use this reference to learn the cmds.

[behavior cmds](https://hack64.net/wiki/doku.php?id=super_mario_64:behavior_commands)

Ex. 1 Running a function every frame

* 00 04 00 00
* 08 00 00 00
* 0C 00 00 00 80 40 10 00
* 09 00 00 00

This starts a script with a 0x00 cmd. Loops every frame with a 0x08 and ends with a 0x09. 0x0C just calls a function at 0x80401000. In all this script will tell the object to run 0x80401000 every frame

Ex. 2 Enabling interaction

* 00 04 00 00
* 11 01 00 01
* 2F 00 00 00 00 00 00 10
* 23 00 00 00 00 64 00 64
* 08 00 00 00
* 10 05 00 00
* 10 2b 00 00
* 09 00 00 00

I start with 0x00, then set obj flags with 0x11. If you check the reference, the struct value is dictated by: 0x1\*0x4+0x88 -> 0x8C which is labeled on the obj struct as obj flags. Obj flags themselves aren't written down but 0x1 is good to remember as it is the one that updates the model with the position. Otherwise the model will not match the location of the object.

Next I set interaction with 0x2F. Use the reference to check which interaction it is. Next I set the collision cylinder size. This is the cylinder that if mario collides with, will make the interaction happen.

Next I set 0x9C and 0x134 to zero with my two 0x10 cmds. 0x9c is the col timer. It means how many frames until the collision is active. I set it to zero every frame so it is always active. Next I set the interaction state to zero. The interaction state is the object's way of keeping track of what interaction has happened. By default most are in state zero. After an interaction objects usually go into a state of 0x1 or higher, which will make the interaction no longer happen. Our interaction is the type of a coin. This means we can collect the coin an infinite number of times because we reset the interaction and collision every frame.

Ex. 3 Solid object

* 00 09 00 00
* 11 01 00 11
* 2A 00 00 00 0E 01 05 00
* 08 00 00 00
* 0D 07 00 50
* 0C 00 00 00 80 38 39 CC
* 09 00 00 00

Here I start with a 0x00 cmd as usual. For solid objects you always use 0x09 as the second byte. If you don't, it will not be solid. Next I set the collision pointer with the 0x2A cmd. The collision pointer is just the location in the game where the collision data is, I may go over collision in a later tutorial but not here. I use 0x0D070050 to change the y pos every frame. I use 0x0D because y pos is a float and I want to offset not set. Finally 0x803839CC is a function that makes the object solid. If you don't use this function the object will not be solid. You will use this function a lot so it is best to remember it.

Ex. 4 Using everything together

* 00 04 00 00
* 11 01 00 11
* 2F 00 00 00 01 00 00 00
* 23 00 00 00 00 64 00 64
* 10 05 00 00
* 08 00 00 00
* 0F 13 01 00
* 0C 00 00 00 80 40 10 00
* 09 00 00 00
* .ORGA 0X1201000
* LUI T0, 0X8036
* LW T0, 0X1160 (T0)
* LW T1, 0X134 (T0)
* BEQ T1, R0, END
* LW T5, 0X154 (T0)
* ORI T9, R0, 0X30
* BNE T9, T5, END
* NOP
* SW R0, 0X134 (T0)
* END:
* LW T1, 0X134 (T0)
* JR RA
* SW T1, 0X14C (T0)

Like the last example, this sets up an interaction for an object. What's new is that I added an 0xF cmd and removed the cmd that set 0x134 to zero each frame. The 0xF cmd changes 0xD4 every frame, you should be able to figure this out by checking the reference. What I do now is check if the interaction happened, a.k.a. if the interaction is not zero. Then I wait 0x30 frames and reset it to zero. I store the interaction state to the action *0X14C* so that the timer get reset and I can measure 0x30 from the moment *0X134* changes.

### **Conclusion**

All together these three tutorials should be enough to do general object editing. All that remains is to make your own code and experiment with it. You can also use the hack64 wiki to find global variables that may help for your coding needs.