SM64 Hacking Tutorial Part 5 - Custom Objects

By Arthurtilly

Link to part 4.5

Welcome to part 5 of this tutorial. You may have noticed that this is a document rather than a slideshow; this is because slides don't provide enough room for a long string of ASM code.

This tutorial will be looking at how to begin creating custom objects with ASM and behaviour scripts. Make sure you've read my tutorial for behaviour scripts before reading this!

We'll start with our custom behaviour script we created in part 4.5, which replaces the Kickable Board behaviour script.

```
00 04 00 00
11 01 00 01
10 05 00 00
10 2B 00 00
23 00 00 00 00 50 00 64
08 00 00 00
   0C 00 00 00 80 2A A3 F4
09 00 00 00
```

To add custom functionality to this behaviour script, we'll put our ASM at ***0x802AA3F4*** in RAM. Subtracting the offset of ***0x80245000*** gives us a ROM address of ***0x653F4***.

Let's start with the basic function wrapper:

```
.orga 0x653F4
ADDIU SP, SP, 0xFFE8
SW RA, 0x14(SP)

LW RA, 0x14(SP)
JR RA
ADDIU SP, SP, 0x18
```

Now let's think about the kind of object we want to create. Let's create an object that bounces you up in the air when you are touching it and are pressing A. First, we should outline our code in pseudocode:

```
If touching Mario:

    If A is pressed:

        Set state to Double jump

        Set Mario's Y speed
```

(The state doesn't matter too much as long as it's an aerial state.)
The best way to tackle this is to turn each line of pseudocode into ASM at a time.

Let's start with the code that detects if Mario is touching the object. For this, we need to use the *Detect Collision* function. I actually mentioned this function in part 2; it is located at ***0x802A1424***. The function takes the RAM addresses of the two objects that are being testing for collision as **`A0`** and **`A1`**, and returns the result of the test as **`V0`**. If there is collision then it sets **`V0`** to 1, otherwise it is set to 0.

First, we should look at two very important RAM addresses for working with custom objects: ***0x80361158*** and ***0x80361160***. These are both pointers, meaning that the value at their address is another RAM address, and it is these RAM addresses we need to pass to the function. Later we will be looking at these addresses again when we look at object structs. ***0x80361158*** is the RAM address of Mario's object, and ***0x80361160*** is the address of the current object (the one whose behaviour script is currently being run). Here is some simple code to pass these arguments to the function:

```
LUI T0, 0x8036
LW A1, 0x1158(T0)
JAL 0x802A1424
LW A0, 0x1160(T0)

BEQ V0, R0, !colliding
NOP

…

!colliding:
```

Note that sometimes using this function will result in crashes if you interact with other objects. If this happens, switch **`A0`** and **`A1`**.

Now we want to write code that detects if A is pressed or not. For this, we'll use the address ***0x8033AFA0***, which gives us the currently pressed inputs this frame. Another useful address is

0x8033AFA2, which gives us all the newly pressed inputs this frame. So for example, if A is pressed and wasn't pressed last frame, then it will be "active" in both of these values, but if A was also pressed last frame, it will be active in the former value but not the latter.

You might be wondering what "active" means in this case. Each button has a bit associated with it. If that button is active, the value will have that bit set to 1, but if it is not active it will be set to 0. For example, the button "A" sets the bit ***0x8000***. I'll include a full table of inputs and bits at the end of this tutorial. To detect if a bit is active, we'll use the **`ANDI`** command (AND Immediate) and AND the value with ***0x8000***. If the bit is set then the result will be ***0x8000***, but if it is not set the result will be ***0x0***.

```
LUI T0, 0x8034
LH T1, 0xAFA0(T0) // the value is only 2 bytes so use LH!
ANDI T1, T1, 0x8000

BEQ T1, R0, !aPressed
NOP

…

!aPressed:
```

Note that we have to reload **`T0`** as we called a **`JAL`** in the previous code, which means we cannot guarantee that **`T0`** will not have changed.

Now we'll set Mario's state to a double jump. We can use the list mentioned in tutorial 4 to find that the action value for a double jump is ***0x03000881***, and we found out that the RAM address of Mario's state is ***0x8033B17C***. We can use our previous value of **`T0`**, as we haven't used a **`JAL`** yet:

```
LUI T1, 0x0300
ORI T1, T1, 0x0881
SW T1, 0xB17C(T0)
```

The only thing we need to do now is set Mario's Y speed. We'll set it to 60. We can use this handy float converter: https://www.h-schmidt.net/FloatConverter/IEEE754.html to figure out the hexadecimal representation of 60, which turns out to be ***0x42700000***. Now, I haven't gone over floats yet, but for now just know that we need to set Mario's Y speed to this value. Now, the RAM address for Mario's Y speed is ***0x8033B1BC***. So let's write some code for this:

```
LUI T1, 0x4270
SW T1, 0xB1BC(T0)
```

Now, there's one more handy thing to know, and that's that you can add the behaviour script into your ASM file! Simply put this at the top of your file:

```
.orga 0x21A46C
hex {00 04 00 00 11 01 00 01 10 05 00 00 10 2B 00
00 23 00 00 00 00 50 00 64 08 00 00 00 0C 00 00
00 80 2A A3 F4 09 00 00 00}
```

and you won't need to change it in the hex editor!

Let's put all our code together now:

```
// set behaviour script
.orga 0x21A46C
hex {00 04 00 00 11 01 00 01 10 05 00 00 10 2B 00 00 23 00 00
00 00 50 00 64 08 00 00 00 0C 00 00 00 80 2A A3 F4 09 00 00 00}


// main code
.orga 0x653F4
ADDIU SP, SP, 0xFFE8
SW RA, 0x14(SP)

// check if colliding with Mario
LUI T0, 0x8036
LW A1, 0x1158(T0)
JAL 0x802A1424
LW A0, 0x1160(T0)

BEQ V0, R0, !colliding
NOP
        // check if A is being pressed
        LUI T0, 0x8034
        LH T1, 0xAFA0(T0)
        ANDI T1, T1, 0x8000

        BEQ T1, R0, !aPressed
        NOP

                // set Mario's state to Double jump
                LUI T1, 0x0300
                ORI T1, T1, 0x0881
                SW T1, 0xB17C(T0)

                // set Mario's Y speed to 60
                LUI T1, 0x4270 // 60.0
                SW T1, 0xB1BC(T0)

        !aPressed:

!colliding:

LW RA, 0x14(SP)
JR RA
ADDIU SP, SP, 0x18
```

Now, compile this with Armips and add an object with behaviour ID ***0x66C*** to a vanilla ROM. You'll see that if you're touching the object and are pressing A, Mario will shoot up into the air!


Thanks for reading my tutorial! In the next part, I'll go over object structs, what they are and how they can be used to make custom objects.

Appendix: List of bits set by each button:

C-Right	***0x0001***
C-Left	***0x0002***
C-Down	***0x0004***
C-Up	***0x0008***
R	***0x0010***
L	***0x0020***
D-Right	***0x0100***
D-Left	***0x0200***
D-Down	***0x0400***
D-Up	***0x0800***
Start	***0x1000***
Z	***0x2000***
B	***0x4000***
A	***0x8000***