Customizing Pikachu with Arbitrary Code Execution

Rebecca Brunner

July 20, 2024

Abstract

This guide will provide a full walkthrough of how to setup *Arbitrary Code Execution (ACE)* within *Pokémon Yellow* in order to customize the game loop. We will alter the game loop to insert custom sprites, alter music, add custom text and so on. All codes are written for English. All codes should be compatible with original hardware, accurate emulators, and *virtual console (VC)*. Codes will provide technical explanation as well as step by step instruction. It is recommended but not required to have beginner's background with assembly. For original hardware it is recommended to have a method for save injection. In order to demonstrate how to use ACE to modify your game, we will be implementing a save file to play as Team Rocket's Jessie and James.

Contents

1 Initial Setup and Environment

This section will go over initial setup of an environment capable of triggering ACE on demand. This section will provide resources to exploit the game and trigger ACE, extend ACE to setup a friendly developer environment, and eventually install a full on RAM editor and script selector.

1.1 4F

To start with you're going to want ot setup ACE within your game. To do so you can follow the excellent set of guides provided by Discord GCRI member Timo. See the guide listed in the references [2]. If you are more of a visual learner you may refer to Timo's video guides [1].

1.2 RAM Editor

Once you have finished your 4F install, install the RAM writer and script selector *(TimOS)*. See the reference section for a link to the code [3]. Once you have the RAM writer, grab your pikachu and proceed through the game as normal until you get the pokédex to avoid softlocks. Enter the PC in Viridian so we have access to a PC.

1.3 Mass Clear a Box

We're going to write our exploit to the current active box since it has an extremely large amoung of free space to work with. Before we begin, make sure to switch your active box to one that you want to use for your exploit. After that we're going to clear it. We only need to run this code once, so we're going to use a temporary free space to write a quick script to clear the box for us.

The trainer buffer is a temporary space in RAM that is overwritten whenever you have an encounter with an opponent trainer. It's a great place to write one-time codes and begins at d89d. Use the freshly installed RAM writer and use the following code to clear the box:

```
AF          ; xor a         ; zeros a register
01 60 04    ; ld bc,0460    ; load bc with the number of bytes
                              to clear
```

```
21 7F DA    ; ld hl,da7f    ; address where active box begins
C3 6E 16    ; jp 166e       ; Jump to FillMemory
```

2 Script Selector (TimOS)

As part of the setup for this guide, you should have installed a script selector. This selector comes with the RAM writer and nickname writer pre-installed, however, we can extend the selector to be able to easily bind our own script. In this section, we will be binding a code to allow us to steal opponent's pokemon to a new slot on the selector.

2.1 Overview of TimOS

As part of the setup you should have installed the RAM Writer along with a script selector the GCRI community has affectionally dubbed *TimOS* after its creator Timo. *TimOS* allows us to quickly execute stored codes. Two codes are installed by default - the first being the RAM writer in slot 1 and the second being the Nickname Writer in slot 2. The number of scripts can be extended to support however many scripts you can fit within the operating range from C6E8 to CB49. Let's quickly review how this script works.

2.2 Adding new scripts to TimOS

To start with, the number of selectable scripts is stored at C6E9. This is an index for a jump table with the list of scripts. The jump table is located at C7C0. So for example if we select to execute the 2nd script, the jump table will be indexed for the 3rd byte or 2nd address since a single address is two bytes. So to add an additional script, we first increment our selector by 1, then append the address of our new script to the jump table. The address should be within the operation range of *TimOS* as mentioned above in the overview.

2.3 How to Steal Pokemon

Taking this approach, let's extend *TimOS* to add a script to steal the opponent's pokémon during a battle. Pop open the RAM writer and navigate to value C6E9 (Refer to the install instructions for controls [3]). Increase the number of scripts from 2 to 3. Next pop over to the jump table at C7C0. Find the last value which should point to the nickname writer at D669. We want to add our new script to a place *TimOS* will save. Everything below the jump table up to CB49 is free, but let's try and be mindful of our jump table growing. For this guide I will decide to start our custom codes at C800. To specify this I put 00 at C7C4 and C8 at C7C5 in our jump table. At C800 I put C9 which is the opcode for return. Let's test our setup.

Save your game, then view your trainer card to navigate your SRAM bank off your save data in case we crash [4]. Use 4F to launch *TimOS* and trigger script 3. If nothing happens, our setup was successful. If not, double check your jump table and addresses align. Once you have confirmed your setup is working, let's adjust the code at C800 to allow us to steal pokémon.

2.3.1 Battle Type

There are three types of battles in pokémon games: wild battles, trainer battles and safari battles. What type of battle you are in is determined by a variable in RAM and can easily be adjusted live in a battle. Simply changing a trainer battle to a wild battle enables us to steal pokémon by throwing a pokéball at it. This will also automatically end the battle since wild pokémon are one off encounters. It's ideal to have this as a toggle so we can toggle it on whenever we see a desirable pokémon in a fight. To do this, put this code at C800:

```
3E 01          ; ld a,01          ; Load register a with 1
                                    which is the value for a
                                    wild battle
EA 56 D0       ; ld (D056),a      ; Load the value of address
                                    D056 with a.  D056 is the
                                    battle type
C9             ; ret              ; End and return to normal
                                    gameplay.
```

It's time to test our code! Get yourself into any trainer battle and try using script 3. It will consume one turn of battle, but you should be able to throw a pokéball at your opponent and attempt to catch their pokémon. This is our first step towards making our *Play As Rocket* save file, and we will be extending *TimOS* further as we go along.

3 OAM Hijack

The OAM routine is a routine called every animation frame of the gen 1 and gen 2 pokemon games. It is responsible for updating sprites, allowing them to move and be drawn to screen. As the function runs every animation frame, if we insert a custom jump point into the routine we can, ”hijack,” the routine to run ACE every frame. This is ideal for modifying the game’s loop and adding, ”event listeners,” which will trigger whenever a certain action happens. In this section we will be going over the OAM routine and how to hijack it. For a practical example, we will be porting a gen 2 code developed by Timo to add a run button to the generation 1 games.

3.1 The OAM Routine

TODO

3.2 Self installing Hijack

TODO

3.3 Preventing Crashes

TODO

3.4 Adding a run Button

TODO

3.5 Adding a toggle to our script selector

TODO

4 Persistence

At this point, every time you reboot your game you will have to re-toggle your hack. In this section we will be making our OAM hijack persistent - as in the game will automatically restore our hijack after resets. We will end the section by overriding the encounter music that plays when walking up to a trainer after being spotted.

4.1 Map Scripts

TODO

4.2 Creating a Persistent Hijack

TODO

4.3 Modifying Encounter Music

TODO

5 Sprites

In this section we are going to be overriding sprites as part of our OAM hijack. We will start by briefly going over how sprites are processed and end by overriding the overworld sprite, backsprite, and front sprites. For those on original hardware without access to save injection, we will provide alternative options that should be more accessible than entering 300 or so bytes in by hand.

5.1 Decompression

TODO

5.2 Interlacing

TODO

5.3 Overworld Sprite

TODO

5.4 Trainer Card

TODO

5.5 Backsprite

TODO

6 Custom Text

This final section will go over how to create custom text. We will be overriding the popup window for Pikachu. We will take advantage of Pikachu's friendship as well as emotes to fully customize your travelling companion. In this case, we will be turning Pikachu into James and giving him custom dialogue.

6.1 Text Scripting

TODO

6.2 Textboxes

TODO

6.3 Custom Text

TODO

References

[1] TimoVM. *Gen 1 Setup Video Guide*. `https://youtu.be/fCJsr6UQv7E`.

[2] TimoVM. *Gen 1 SRAM Setup*. `https://glitchcity.wiki/wiki/Guides:TimoVM%27s_gen_1_ACE_setups`.

[3] TimoVM. *TimOS and RAM Writer Install*. `https://glitchcity.wiki/wiki/Guides:Nickname_Writer_Codes#Installing_a_RAM_writer_environment_(TimOS)`.

[4] ZZAZZ. *Pokémon Red/Blue - analysis of basic crash types*. `https://www.youtube.com/watch?v=YneEmX8J5zA&ab_channel=TheZZAZZGlitch`.