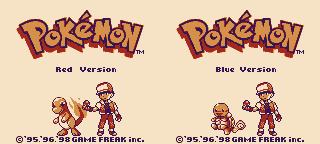
# Tutorial #3 – Bit Flags

## Overview

This tutorial will show a few examples of how to create some achievements for a game that has a lot of \*\*Bit Flags\*\*. [Pokemon Red|Blue](<https://retroachievements.org/game/724>) was chosen as the tutorial example since all the defeated Trainers, Item pickups, and caught Pokemon are tracked with bit flags.



![Pokemon Red|Blue Title Screen](Pokemon\_Red\_Blue.png)

## For Loops

The following examples use variations of \*\*For Loops\*\* to programmatically cycle through an \*\*Array\*\* of bit flags. \*\*For Loops\*\* are an extremely useful programming concept that allow you to execute a block of code any number of times. Using loops you can create very complex achievements in a few lines of code. If you unfamiliar with the concept of \*\*For Loops\*\* then please review the Khan Academy video [For Loops! A New Kind of Loop](<https://www.khanacademy.org/computing/computer-programming/programming/looping/pt/for-loops-a-new-kind-of-loop>)

## Arrays

\*\*Arrays\*\*\* are sequential data, of a single data type, that can accessed with a numerical index. In RAScripts arrays can be numbers, text, conditions, or another nested \*\*Array\*\*. \*\*Arrays\*\* work well together with \*\*For Loops\*\* since you can loop through an \*\*Array\*\* generating new code for each element in the \*\*Array\*\*. If you unfamiliar with the concept of \*\*Arrays\*\* then please review the Khan Academy video [Into to Arrays](<https://www.khanacademy.org/computing/computer-programming/programming/arrays/pt/intro-to-arrays>)

# Example 3A: Setting the Forest Ablaze

Pokemon Red|Blue is a Role Playing Game \*\*(RPG)\*\* where you collect and train Pokemon to fight other Pokemon Trainers around the Kanto region. You also collect or buy special Items that heal you Pokemon, remedy status affects, or help you Pokemon evolve. Both Trainer fights and Item pick ups are tracked in-game using \*\*Bit Flags\*\*.The following achievement is for when the player fights all of the Trainers and picks up all of the Items in the Viridian Forest. The Viridian Forest area was selected since it was one of the smaller sections of the game however, you can use the same snippet of code for any section of the game by passing a different \*\*Array\*\* of bit flags. If doesn’t matter if they are more or less bit flags than the example below since the code will adapt to number of elements in the Array.



![In the Viridian Forest](Pokemon\_Viridian\_Forest.png)

```

// Pokemon Red Version | Pokemon Blue Version

// #ID = 724

// An array on all the bitflags in the Viridian Forest

ViridianForest = [

bit0(0xd6f0), // Hidden Potion

bit1(0xd6f0), // Hidden Antidote

bit4(0xd5b2), // Antidote

bit5(0xd5b2), // Potion

bit6(0xd5b2), // Poke Ball

bit2(0xd7f3), // Bug Catcher 01

bit3(0xd7f3), // Bug Catcher 02

bit4(0xd7f3) // Bug Catcher 03

]

// Check if array of passed flags goes from previously all but one flag is true

// to all flags being true.

function FlagsComplete(Flags)

{

return sum\_of(Flags, f => prev(f)) == length(Flags) - 1 &&

measured(sum\_of(Flags, f => f) == length(Flags))

}

// Make an achievement for finding every item and defeating every trainer

// in the Viridian Forest

achievement(

title = "Example 3A: Setting the Forest Ablaze",

description = "Find every Item and defeat every Trainer in the Viridian Forest.",

points = 0,

trigger = FlagsComplete(ViridianForest)

)

```

## sum\_of

The above code utilizes the command \*\*sum\_of\*\* to add the value of each bit flag in the \*\*Array\*\*. Since a bit is 1 if on and 0 if off the summation of the bit flags will tell you how many bit flags are on. Using the length of 8 for the array in this example we test if the previous number of bit flags was 7 and the current number of bit flags that are on is 8. The advantage of counting bit flags like this is that the player can collect the Items and fight the Trainer in any order and still get the achievement. Additionally, we get save protection since the player must increment the bit flag count to get the achievement and it won’t trigger if they load a save file with the bit flags all on.

To help understand how to use \*\*sum\_of\*\* it might help to deconstruct an example. Consider the line:

```

sum\_of(Flags, f => prev(f))

```

The above line could be written as a \*\*For Loop\*\* where \_Flags\_ is the array we are looping through and \_f\_ is each element in the array. The following code is eqvilant to the above:

```

sum = 0

for f in Flags

{

sum = sum + prev(f)

}

// The value of sum if the number of bit flags that were on in the previous frame.

```

Note that the `f => prev(f)` part of the \*\*sum\_of\*\* means that the previous value of each element is counted. If we wrote `f => f` then the current value of each element is counted.

## Unrolling a Loop

To better understand how the \*\*For Loop\*\* is interpreted by the compiler you can unroll the loop into individual lines of code. Unrolling a loops is a technique used by programmers to optimize code since it avoids conditional jumps which take a small resources. The technique is more useful on embedded hardware where resources are limited however, it is a useful exercise when learning \*\*For Loops\*\*. Consider the loop:

```

sum = 0

for f in Flags

{

sum = sum + f

}

```

To unroll this For Loop we would copy the line of code inside of the eight times, one for each element in the \*\*Array\*\* and replace f with the element.

```

sum = 0

sum = sum + bit0(0xd6f0) // Hidden Potion

sum = sum + bit1(0xd6f0) // Hidden Antidote

sum = sum + bit4(0xd5b2) // Antidote

sum = sum + bit5(0xd5b2) // Potion

sum = sum + bit6(0xd5b2) // Poke Ball

sum = sum + bit2(0xd7f3) // Bug Catcher 01

sum = sum + bit3(0xd7f3) // Bug Catcher 02

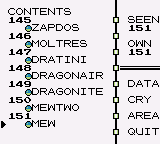
sum = sum + bit4(0xd7f3) // Bug Catcher 03

```

Both code snippets are equivalent however, the one using the \*\*For Loop\*\* is easier to write and there is less chance of a typo in the code. \*\*For Loops\*\* become even more desirable when the number of elements in the \*Array\* increases.

# Example 3B: Gotta ACTUALLY Catch 'em All

The main gimmick of Pokemon is that you can catch and train any of the 151 Pokemon in the game. This is emphasized by the games’ catch phrase “Gotta Catch ‘em All”. The game tracks both which Pokemon you have seen and/or caught in the Pokedex. When it was released Pokemon was sold as a Red and Blue version, each version with its own game exclusive Pokemon. So to actually catch all of the Pokemon you needed trade Pokemon with someone else who had a different version of the game. Additionally some Pokemon like Mew require glitches to catch them. Because of the requirement to trade with another person or use glitches the achievement this example is based is from the bonus set. The reason why it was chosen as one of the examples is because catching all 151 Pokemon is tracked in a consecutive bit flags which allows for some interesting optimizations.



!(Screenshot of the Pokedex)[Pokemon\_Pokedex.png]

```

// Pokemon Red Version | Pokemon Blue Version

// #ID = 724

// $C06E: Music Playng, title screen = 4105

function TitleMusic() => word(0x00C06E) == 4105

// Count each bit between $D2F7 bit 0 to $D309 bit 6

// these are the consecutive bitflags for each of the

// 151 Pokemon in the Pokedex

function PokedexCount()

{

count = 0

for address in range(0xD2F7, 0xD308) {

count = count + bitcount(address)

}

count = count + sum\_of(range(0,6), b => bit(b,0xD309))

return count

}

// Create an achievement for when the player goes from

// 150 Pokemon to 151 Pokemon in the Pokedex

achievement(

title = "Example 3B: Gotta ACTUALLY Catch 'em All",

description = "Catch all 151 Pokemon. Yes ALL of them.",

points = 0,

trigger = never(TitleMusic()) &&

prev(PokedexCount()) == 150 &&

measured(PokedexCount() == 151)

)

```

## bitcount

The above example uses the fact that the Pokedex uses consecutive bits to optimize the code with \*\*bitcount\*\*. Instead of adding the bits together with a \*\*For Loop\*\*, \*\*bitcount\*\* does the leg work for you. Consider line:

```

count = count + bitcount(address)

```

The bits could also be counted using the sum\_of function like:

```

count = count + sum\_of(range(0,7), b => bit(b, address))

```

Note that by using \*\*bitcount\*\* instead of \*\*sum\_of\*\* we drastically reduce that number of lines generated by the script. Instead of adding each bit per line we add eight bits per line. This results in 51 lines using \*\*bitcount\*\* instead of 303 lines using \*\*sum\_of\*\*, approximately 6x smaller.

## sum\_of bits

Another interesting trick in this example is the use of \*\*sum\_of\*\* to add the last seven bits of the Pokedex. Since we want to ignore the value of the eighth bit we can use \*\*sum\_of\*\* to add the values of bits zero to six. The use of the \*\*range\*\* function returns an \*\*Array\*\* of numbers `[0, 1, 2, 3, 4, 5, 6]` you could define the \*\*Array\*\* instead however the \*\*range\*\* function in more convenient especially for larger values. Consider the line:

```

count = count + sum\_of(range(0,6), b => bit(b,0xD309))

```

To help understand what going on we will unroll it like the previous example.

```

count = count + bit(0,0xD309)

count = count + bit(1,0xD309)

count = count + bit(2,0xD309)

count = count + bit(3,0xD309)

count = count + bit(4,0xD309)

count = count + bit(5,0xD309)

count = count + bit(6,0xD309)

```

By using \*\*For Loops\*\* we can make very complex achievements quickly and efficiently.

# Example 3C: Assemble the Dream Team

When battling other Trainers you can choose up six Pokemon to fight for you. One of the most enjoyable aspects of Pokemon is the ability to mix and match your team. By learning the strengths and weakness of each Pokemon type you can assemble your team to counter an opponent’s Team.



## Homework #3

1. Pick your six favorite Pokemon available in Pokemon Red | Blue.
2. Using the Pokedex memory of bitflags between $D2F7 bit 0 to $D309 bit 6 create an achievement for catching all six pokemon select in part A.

# Tutorial #3 Solution

# Solution A

Being a fan of the Pokemon Anime, I chose the team Ash used to fight the gym leader Drake to become the Orange League Champion in episodes 113 and 114. Ash selected: Pikachu, Squirtle, Tauros, Lapras, Bulbasaur, and Charizard.



![Screenshot of the Orange League Champions](Pokemon\_Dream\_Team.png)

# Solution B

Since the bits are likely non-consecutive we can’t use the \*\*bitcount\*\* optimizations from Example #3B. Instead we are counting the number of bit flags that are on like Example #3A since it can handle bit that are all over the place.

```

// Pokemon Red Version | Pokemon Blue Version

// #ID = 724

// $C06E: Music Playng, title screen = 4105

function TitleMusic() => word(0x00C06E) == 4105

DreamTeam = [

bit0(0xd2f7), // Bulbasaur

bit5(0xd2f7), // Charizard

bit6(0xd2f7), // Squirtle

bit0(0xd2fa), // Pikachu

bit7(0xd306), // Tauros

bit2(0xd307) // Lapras

]

// Check if array of passed flags goes from previously all but one flag is true

// to all flags being true.

function FlagsComplete(Flags)

{

return sum\_of(Flags, f => prev(f)) == length(Flags) - 1 &&

measured(sum\_of(Flags, f => f) == length(Flags))

}

// Create an achievement for when the player collects the entire Dream Team

achievement(

title = "Example 3C: Assemble the Dream Team",

description = "Catch Pikachu, Squirtle, Tauros, Lapras, Bulbasaur, and Charizard.",

points = 0,

trigger = never(TitleMusic()) && FlagsComplete(DreamTeam)

)

```