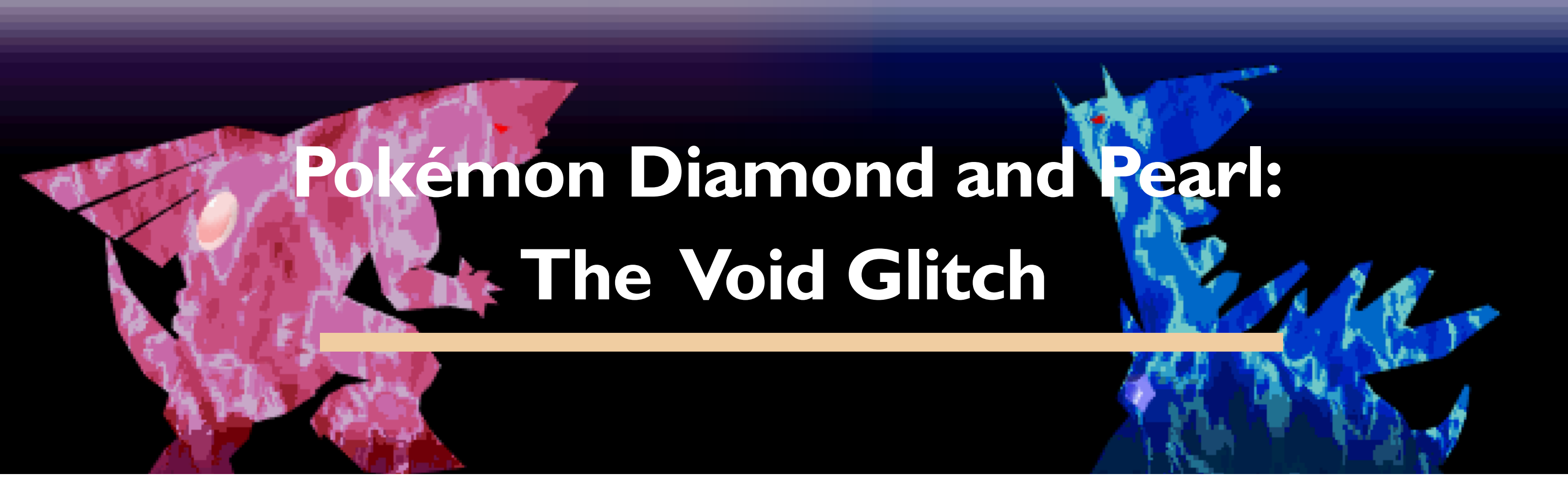


Pokémon Diamond and Pearl: The Void Glitch

Table of Contents

1. Introduction

1.1 The void Glitch

The Void Glitch is the most infamous glitch in the gen 4 Pokémon games Diamond and Pearl. By exploiting glitches it is possible to reach the out of bounds areas of the game. This area appears black[1] and was therefore dubbed the Void. Through the years, this glitch has been researched thoroughly and has allowed players to capture unreleased event Pokémon like Darkrai, Shaymin and almost a decade later even Arceus.

After the hunt for Arceus was over, the goal became to break open the gen 4 games as much as possible. Voidroutes became more optimized and new glitches continued to be found. In this paper I will attempt to explain how the Void behaves and how it can be exploited in an attempt to preserve research.

2. Resources

2.1 Scripts, Dumps and documentation

Since 2017 the research in gen 4 games has become increasingly more technical. As a result of this scripts have been written by multiple members to assist in the finding of exploits. In the following section a list of useful scripts, dumps and similar useful resources are listed along with the authors of the works.

1. Routing Script For Dummies. / *RSFD.lua*
   This script has combines all the most important resources from previous scripts, and has added several new applications. It is compatible with DPPt and HGSS.

   Author:
   RETIRE
   Co-Authors:
   MKDasher: *Void.lua* and *Loadlines.lua*
   Ganix: *VET.lua*

[1] This is not true if the camera loops or is desynced, in which case it shows a designated area.

2. *RNG.lua*

   This script shows the possible encounters of wild Pokémon in the area. It also shows the stats of your party Pokémon and opponents.

   Author: MKDasher

3. Mapdata Dump

   This is a dump of NPC and Warprelated data that is loaded when entering a new Map ID. It is heavily utilized in Voidrouting.

   Author: Ganix

4. Mapscripts Dump

   This is a dump of mapscripts, and what their id in runtime is. It is an essential document for understanding Script Execution.

   Author: Ganix

5. RAM documentation

   Author: MAP
   Co-Authors:
   Martmist
   RETIRE
   FlederKiari
   Ganix

6. Script opcodes

7. Diamond Decompilation

3. Entrance Points

3.1 Tweaking Glitch

The tweaking glitch is the only currently known glitch that allows the player to enter the void without prior setup on all versions of Diamond and Pearl. It abuses the loading of new chunks in the game. At any given time, exactly 4 chunks should be loaded. By crossing the loadlines the game uses in quick succession, it is possible to misplace chunks.

You cross a loadline to write a chunk into a temporary buffer. You then cross more loadlines to delay that buffer from being written to the designated section in memory. After you stop moving the game will catch up and write the buffer to memory. However, it will overwrite the last loaded chunks as a result. In the video you see the most commonly used tweak. If you slow the footage down, you can see the left area loaded correctly for a split second before being overwritten by the chunks that were still in the buffer.

This tweaking then allows you to move through this irregularly located chunk, which is called an Abyss. To fix this glitch, you simply have to reload the chunks. This can be done by opening and closing menus, save resetting or crossing new loadlines. To go out of bounds, you want to move to where a door should be located under normal circumstances. You then open and close a menu to load the correct chunks, and you can enter the door from above. Doing this will load the player in the building, but will automatically move you 1 step down, into the out of bounds area called the Void.

3.2 Surf Glitch (JP exclusive)

The Surf Glitch is an oversight that is present in the Japanese Games but was patched out from the international releases. In the chunk data of all Japanese Elite 4 rooms of Diamond and Pearl, a single water tile is present on the bottom door. You can’t directly interact with it by pressing A. This is because the bottom door is an NPC, and it has priority over interacting with the tile it stands on. However, you can still open the Pokémon Menu and select surf there, which will allow you to surf into the door and hop out of bounds.

There is one more entrance point for the Japanese game which is explained under
5. Wrong Warps

4. Void Glitch Basics

4.1 Traversing through RAM

The Void is essentially the game's RAM. It interprets the game's memory as Map IDs, and therefore any action you take directly influences the Void. By loading the correct data, we can manipulate what Map IDs appear and therefore we can route to essentially anywhere in the game.

Every Map ID is a 2 byte value. The valid Map IDs range from 0 to 558. Everything beyond that automatically gets converted to Map ID 3, which is Jubilife City.

In order to understand movement through the Void, you can say it is similar to an Excel document. Cells hold values, which in this case would be a 2 byte value called the Map ID. Every 32 steps you take, the game will read from a new cell. If you move left or right you'll move a single cell, or 2 bytes.

When it comes to moving up or down however, the game has to use an extra variable. For this, it checks the size of your Matrix. The Matrix is the inbounds area of the game. For example, in a house, there is only one 32x32 area, or one cell, that is used. Therefore the Matrix Width will be 1. If the size of the Matrix increases, like in the Safari Zone, it can go up to 4. For Sinnoh, by far the largest area with a size of 960x960 tiles, this will be 30.

It will multiply the Matrix Width by 2 bytes, or in other words, by one cell. Every 32 steps up or down you would get moved by 1 cell, 4 cells or 30 cells depending on these factors.

In other words, in a house going 1 cell down is the same as going 1 cell right. In a bigger area, going 1 cell down equals moving 4 cells right. In Sinnoh, going 1 cell down equals moving 30 cells right. Because of this, you should prioritize routing with a Sinnoh Matrix, because you'll move 30 times faster vertically.

In order to change Matrix, you simply have to save reset in any Map ID that is part of that Matrix. For Sinnoh, that includes any City or route, but for most other Matrixes, there is only 1 Map ID that can be used.

If this was a lot information, luckily this is all visualized when using the RSFD.lua script. It will show the current position in RAM, and allows you to colorcode the Map IDs that are important to you.

4.2 BSOD, crashes and softlocks

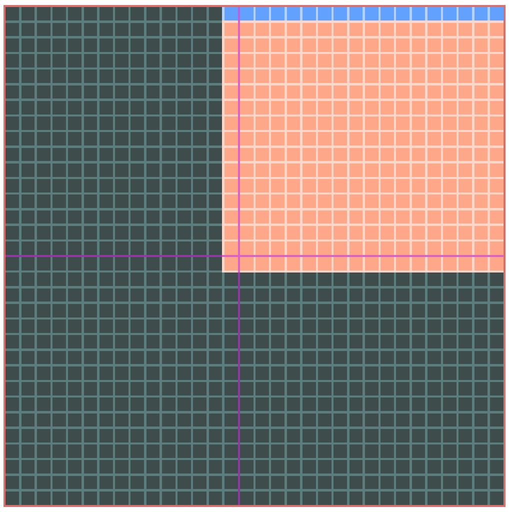
Before we start routing there are still some things we need to address. Most importantly, different causes for crashes and softlocks. A BSOD, or Black Screen of Death, is an effect that can permanently make a save file unplayable. Luckily, it is entirely preventable. A BSOD is caused by entering Map IDs known as 'BSOD Maps' that load 3D models that do not get removed when loading new Map IDs. If you attempt to load graphics after such a model has been loaded, the game will crash. The graphics are loaded when selecting a save file, therefore saving with these models loaded can make that save file unplayable. To prevent a BSOD from happening, simply do not enter any of the 'BSOD Maps'. They are colored dark red in the RSFD lua.

A loadline crash can occur when moving over a loadline in a void with a small Matrix, in specific directions. For example, moving left in a Poketch Co. Void is perfectly fine, but moving right over a loadline will crash the game. This is not always easy to get around.

You also should look out for cutscenes. If an NPC should be moved that is not loaded, you crash. This can occur if a part of the cutscene moves you out of your Map ID, and the one you enter does not have enough NPCs to support the rest of the cutscene.

Softlocking can occur when the player moves into tiles that have a collision ID above 0x80. In the void, luckily, the 4 chunks that are always loaded only have a single one that doesn't read 00's. By default it is the top right subchunk in a Cell.

4.3 Basic routing techniques



In the next section we put this knowledge into play by writing some simple routes. For this, you will need the RSFD.lua aswell as the Mapdata.docx dump.

In the first basic routes, we will be utilising Mapdata. The game however does not only read Map IDs from memory, but also the chunks.

In any cell we will pass, the left and bottom section are walkable, but the top right chunk will be filled with garbage. Later on in the guide, you will learn how to manipulate these tiles. For now, simply avoid the chunk.

To toggle seeing the loadlines, press shift+L while the RSFD.lua is active.

4.3.1 Mapdata

Mapdata is a commonly used section of memory to manipulate. It is the closest section of memory in the void that can be fully manipulated and it is not too hard to grasp for a beginner. In this section of memory the game loads NPCs, warps, triggers and their corresponding data when you enter a new Map ID. Things such as sprite IDs, flags, events, x and y coordinates are all loaded here.

This data does not get cleared when you enter a new Map ID, it will simply load in the NPC's and Warp's of the new Map ID. Remember, the Void is RAM, so you can read this data as Map IDs. And by doing that, you will load even more NPC's and Warps. This results in a chain of writing data, and then interpreting this data back as Map IDs.

First, to reach this section of memory you want to follow these steps from the Poketch Co. entrance. We use the North, South, East and West as directions.

1 S
17 W
14 N
446 W

save reset
2111 S
147 W
320 S

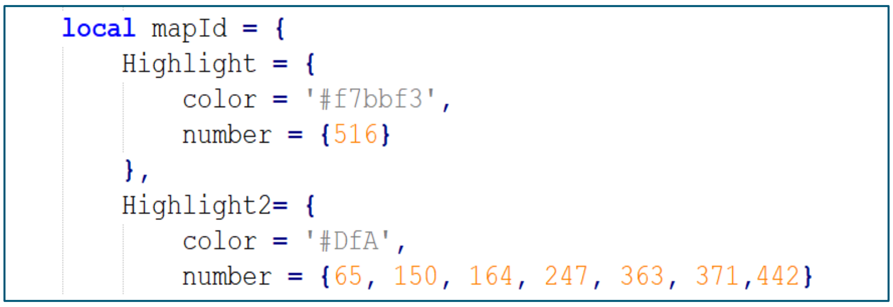
You might notice that while going 2097 S, you go through what looks like Sinnoh. Well, that is because you are 30 cells left of where Sinnoh is. If you remember from earlier, moving 1 cell down is the same as going 30 cells right. This also means that if you go 30 cells left, you are at the same place in RAM as if you just went 1 cell up. So for all the game cares for, you are just reading from the section of RAM that is used for Sinnoh.

Under that, you go through a different Sinnoh, but this time they're the chunks of Sinnoh. After that we reach Mapdata.

What we will do now is route to a Map ID. As an example, we will be routing to an elevator. Elevators generally use a mapscript to teleport you to where you should go once you enter their Map Id. This makes it simple as a first route. We will be going to Map ID 516, the elevator used in Lighthouse Vista, in Sunyshore City.

First, we need to find out what Map IDs load 516, as a warp, coordinate or flag of some sort. For this, open the *Mapdata.docx* and press CTRL + F, then put 516 with a space in front and after it. If you select 'Headers' you will find the Map IDs that load that value into memory. They will be colored yellow.

The Map IDs that should pop up are 65, 150, 164, 247, 363, 371, 442. What you do next is you open *RSFD.lua* in any prefered text editor and insert 516 as 'Highlight' and the other Map Id's as 'Highlight 2'. This will change the colors in which they're displayed in the lua script.

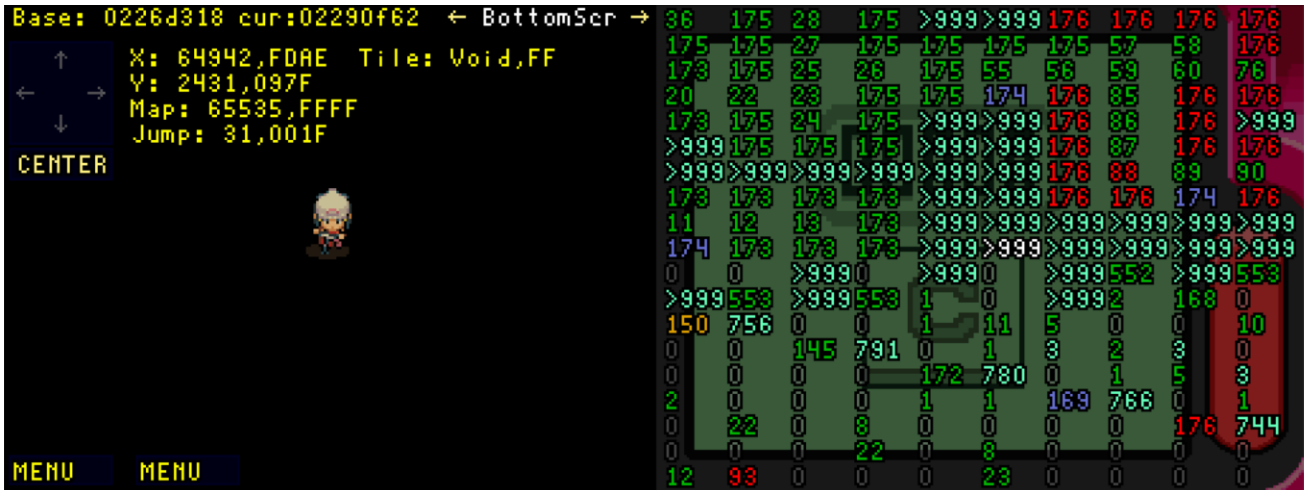


```
local mapId = {
    Highlight = {
        color = '#f7bbf3',
        number = {516}
    },
    Highlight2= {
        color = '#DfA',
        number = {65, 150, 164, 247, 363, 371,442}
```

This is how the Map IDs should be implemented in the script.

You can also edit the colors. By default, Highlight is pink. Highlight 2 is orange.

With this set up, run the script in DeSmuME (or Bizhawk, if future versions are updated to work on that). If it asks you for lua51.dll and lua5.1.dll, simply add those in the directory of DeSmuME. It should look like the following picture.



```
Base: 0226d318 cur:02290f62 ← BottomScr →
X: 64942,FDAE  Tile: Void,FF
Y: 2431,097F
Map: 65535,FFFF
Jump: 31,001F
```

```
36   175  28   175  >999 >999 176  176  176  176
175  175  27   175  175  175  175  57   58   176
173  175  25   26   175  55   56   59   60   76
20   22   23   175  175  174  176  85   176  176
173  175  24   175  >999 >999 176  86   176  >999
>999 175  175  175  >999 >999 176  87   176  176
>999 >999 >999 >999 >999 >999 176  88   89   90
173  173  173  173  >999 >999 176  176  174  176
11   12   13   173  >999 >999 >999 >999 >999 >999
174  173  173  173  >999 >999 >999 >999 >999 >999
0    0    >999 0    >999 0    >999 552  >999 553
>999 553  >999 553  1    0    >999 2    168  0
150  756  0    0    1    11   5    0    0    10
0    0    145  791  0    1    3    2    3    0
0    0    0    0    172  780  0    1    5    3
2    0    0    0    1    1    169  766  0    1
0    22   0    8    0    0    0    0    176  744
0    0    0    22   0    8    0    0    0    0
12   93   0    0    0    23   0    0    0    0
```

5. Wrong Warps

6. Battletower Void

7. Tilewriting Basics

8. Ledgecancelling

9. Advanced Tilewriting

10. Script Execution

11. Word Index

Abyss	A chunk that has been dislocated by performing a tweak or instant tweak.
ASLR	ASLR stands for Arbitrary Space Layout Randomization. This is a way to change the layout of RAM by adding a small offset. It is seeded by mac address, inputs and the time of booting the game. It only changes when booting and soft-resetting the game.
Battletower Void	When you warp or save reset in any of the Battletower Map IDs, you will end up in a Battletower Void. When inside these voids, a RAMshift occurs that can have influence on the loading of chunks or executing scripts, as well as disabling the stepcounter.

BSOD	BSOD stands for Black Screen of Death. This occurs when the player has travelled through a list of Map IDs (referred to as BSOD maps) that load maprelated 3D models that do not get unloaded when simply loading a new Map ID. These models will cause a crash when the game attempts to apply textures by loading graphics. If the player save resets after this has occurred the game may crash each time you boot the game, and simply hangs on a Black Screen of Death.
Buffer	
Chunk	
Ledgecancel	A Ledgecancel is a glitch that occurs when a player is able to walk onto a bikeramp and open the sidemenu within a specific timeframe. It has many applications, but it notoriously annoying to set up on Hardware.
Matrix	The Matrix is the inbounds section of an area. It can be as small as a 32x32 area for houses up to 960x960 for Sinnoh's outdoor section. When saving and resetting inside a Map ID, you will load the game with the inbounds area turning into the one of that Map ID. In some cases such as Sinnoh, multiple Map ID are part of the same Matrix Center, and therefore any of them can be used to load the same inbounds area.
Map ID	Every Map ID is it's own ingame area. It is a 2byte value, meaning it can range from 0 to 65535. The used indexes range from 0 to 558, every index above that gets converted to being index 3 by an Error Handler. Map 3 is Jubilife City.
Matrix Width	The Width of a Matrix center is a value used to determine your position in RAM. It is dependent on the size of your Matrix. The bigger the Matrix Center, the bigger the Matrix Width. And as a result, the more bytes you travel when going up or down.