



**Coropata English translation for Nintendo DS**  
**Translation/localization**  
**theory, modification techniques**  
**and team management**

**Kendall Price**

**2023**

## Introduction

This document exists to provide a detailed breakdown of the translation process for the Coropata (Nintendo DS) English patch by Demon Network.

I will outline and explain the tools/programs used, the way in which we dealt with various types of images/text in the game, and the reasoning for decisions made related to the localization.

I hope this document serves as a combination of a how-to guide for aspiring ROMhackers/translators/localizers, but also as a showcase on what we at Demon Network have achieved as a freshly formed translation group. Hopefully it will also serve as a personal portfolio for myself going forward.

I will be posting this document publicly to my website ([kendalls.garden](http://kendalls.garden)), but Demon Network does have its own website that I encourage you to check out yourself! ([demonnetwork.co.uk](http://demonnetwork.co.uk)).

I'd also like to make sure that anyone reading knows that we as a translation group understand that Coropata as a copyright and an IP is fully owned by Lukplus.,LTD. We don't intend to ever distribute any copyrighted materials and will only ever distribute patch files that can be used on an original and legal copy of the game. All of our hard work has been done out of passion for the game, video-game preservation, and a dedication to bringing lesser-known Japanese titles to an English-speaking audience.



# Table of Contents

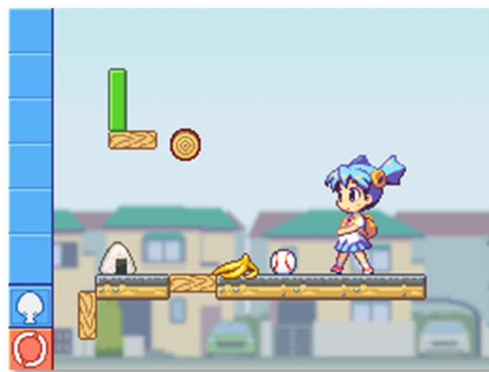
1. <a href="#">Picking a Game – Why Coropata?</a> .....	4
2. <a href="#">Re: The Steam Release</a> .....	5
3. <a href="#">Demon Network</a> .....	7
4. <a href="#">What is Coropata?</a> .....	8
5. <a href="#">Tools, Sites and their Uses</a> .....	9
5.1 <a href="#">CrystalTile2</a> .....	9
5.2 <a href="#">Tahaxan</a> .....	10
5.3 <a href="#">wxMEdit</a> .....	11
5.4 <a href="#">MelonDS</a> .....	13
5.5 <a href="#">Notepad++</a> .....	14
5.6 <a href="#">xDelta</a> .....	15
5.7 <a href="#">GitHub</a> .....	16
5.8 <a href="#">Google Sheets</a> .....	17
6. <a href="#">Translating the Game</a> .....	18
6.1 <a href="#">Text Access</a> .....	18
6.2 <a href="#">Image Access</a> .....	19
6.3 <a href="#">Arm9.Bin</a> .....	20
6.4 <a href="#">Cutscenes</a> .....	21
6.5 <a href="#">Levels, Items and Descriptions</a> .....	24
7. <a href="#">Stylistic Choices</a> .....	26
7.1 <a href="#">Honorifics/Name Nomenclature</a> .....	26
7.2 <a href="#">Oscar/Geotail</a> .....	26
7.3 <a href="#">Geotail’s Speech</a> .....	27
7.4 <a href="#">Hinotori</a> .....	27
8. <a href="#">Post Translation</a> .....	28
8.1 <a href="#">Reflection</a> .....	28

## Picking a Game – Why Coropata?

When picking a game for translation, speaking exclusively from the perspective of a fan project that expects no compensation, there's a few different factors that determine whether or not a game gets picked. For a lot of translators or manga scanlators, the decision to start a project is based on a passion for the source material; perhaps this is something the translators have wanted to see in their target language for a long time, or it is something that is highly requested by a community of similar people.

For us, Coropata was, on the surface, an extremely cute and entertaining game that caught our attention, but outside of its immediate appeal there were many things that made it the ideal choice for a translation team of our calibre.

Firstly, the game being on DS meant that we'd have years of well documented and developed PC programs that allow us to extract, edit and inject all the assets we would need very easily. Nested in that is the way that the Japanese text, when viewed in a hex editor, wasn't compressed or encoded in any confusing algorithm, and was able to be manually edited (with some care) after the text was ready for re-injection.



Previously we had floated the idea of translating PC-98 games, specifically a game titled “Doki Doki Pretty League”; A large part of this game was accessible to us, but due to the extremely proprietary nature of a lot of PC-98 file types (For example, images being in a “.LOG” format that no decompression algorithm available to us would be able to decipher), it was very difficult if not impossible to translate without a skill in reverse engineering that was frankly very far removed from our level of expertise.



There's also the issue that the hardware required to see these patches in action would be relatively hard to come by, compared to the ease of accessing any of the Nintendo DS line in 2023. On top of this, the PC-98, outside of being rare and ancient, was only available in Japan, as opposed to the global success of the DS.

## Re: The Steam Release

Before embarking on a translation we, as a team, have to research the source material and see if there are any previous attempts to the translate the game, successful or not. Of course, most of the time, if a successful English translation already exists then there is no need for us to interfere and waste time creating our own.

There is however, an official English translation released alongside the PC/Steam version of the game, but for two reasons I felt it was still necessary that we create our own patch. The first is that, simply put, I felt the official translation was not of a high enough quality; I felt we could do better. To demonstrate this, I'll use a specific example that first tipped me off that there was some improvement to be made on the official translation, and that was as follows:



Compare the above two screenshots, the left from the official PC release and the right from the original Japanese version. Both of these feature the “Kigen” and “Tairyoku” meters, which are status indicators during the gameplay. The problem is that both of these are the same, where they should be translated in the English version to “mood” and “energy” respectfully.

These kinds of issues, where it is assumed the English player base will inherently understand these romanized Japanese words, is among a litany of issues that made us feel confident in starting our own localization. Outside of this, much of the translation feels rather direct and unnatural, as if it was either done by a machine translation platform, or someone who had little experience with the way people actually speak and use English in real life.



The other reason is that there is simply a novelty in having the translation/patch be available for the original hardware; yes, you can play this game on a PC now, but is it the original experience that the developers intended? A player might not personally care about the way they experience the game, but having it be an option is a something we wanted to fulfil.

Coropata is first and foremost, a DS game, a platform which was fully portable and used the touch stylus to its' advantage. Something about having it played on what is essentially a glorified emulator on PC feels inauthentic, and those who want to play it in its original form that do not understand Japanese, needed our help.

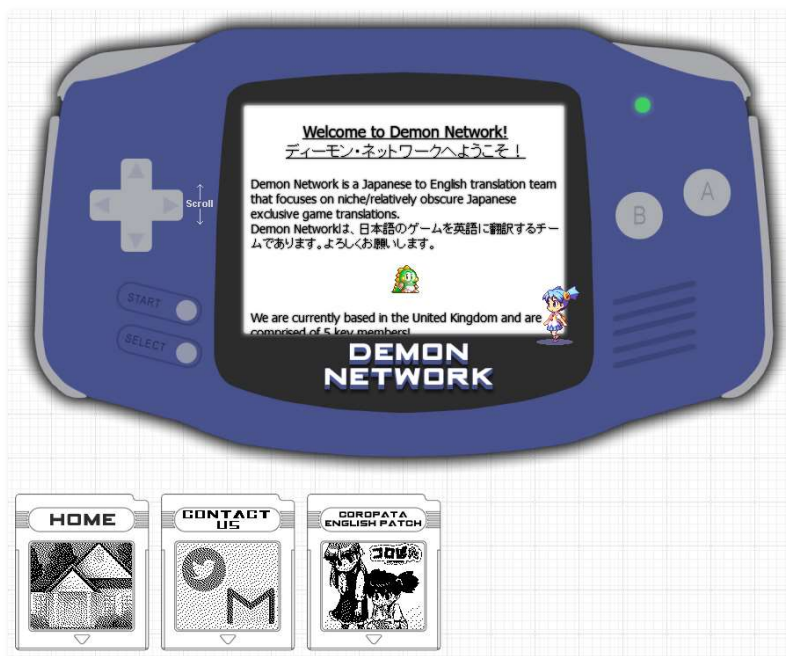


## Demon Network

“Demon Network” is the name of my translation team/group, newly formed to tackle the translation of Coropata, making this our first project. I had personally worked as part of a team translation team before, but wanted to work with close friends I have that share a similar interest in the types of sources I wanted to take a stab at. We are a small, close-knit group that enjoys Japanese language content and want to bring it to a wider audience by making it available in English.

My goal with Demon Network was paradoxically opposite to the reasons why a lot of English patches are even produced in the first place; usually it's a result of high demand, but taking a look at Coropata and other games we either looked at or have lined up in the future, this doesn't seem to be the case.

For us, the obscurity of the game makes it desirable to be translated. It's all well and good that a game like “Mother 3” was given an in-depth and passionately crafted translation, but that was inevitable given the cult status of the earthbound/mother series. Games like Coropata might have been left without the proper attention indefinitely if we don't dig it up from the trenches of time.



Our team consists of 3 translators, an artist/image editor and a member familiar with ROMhacking/programming; we have our own personal website at <https://demonnetwork.co.uk>

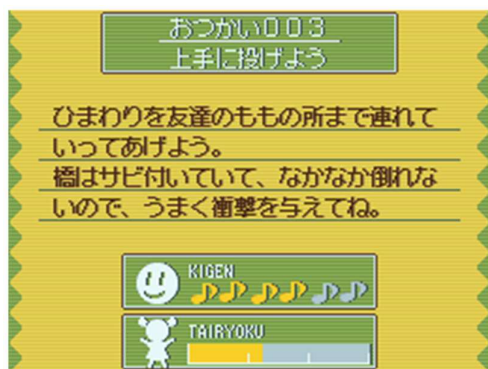
## What is Coropata?

Perhaps a little bit of context as to what the game actually is would help for unfamiliar readers.

Coropata is a Japanese puzzle game that was developed and published by “LukPlus” in 2009 for the Nintendo DS. It functions as a “Rube Goldberg” style of puzzle, where items are placed that interact with each other in a long series of events in order to create a finalized result. This kind of thing is easier to show visually than to write about, so I’ll use these images from LukPlus’ official Coropata website to illustrate.



Each stage presents you with an objective, for example “Get Himawari over to Momo (her friend)”. When the play button is pressed, a “simulation” of sorts begins, and Himawari will typically start moving forward in the direction she starts facing. The player needs to place objects in various locations to make sure that Himawari safely achieves her objective; this simple premise increases in difficulty over the course of the story, covering 6 chapters with their own unique storylines and cutscenes.



The aforementioned “Kigen”/”Tairyoku”, aka “Mood” and “energy” must be paid attention to, as Himawari’s behaviour can radically shift based on how tired or happy she is, to the point of giving up and resorting to sitting down and crying.

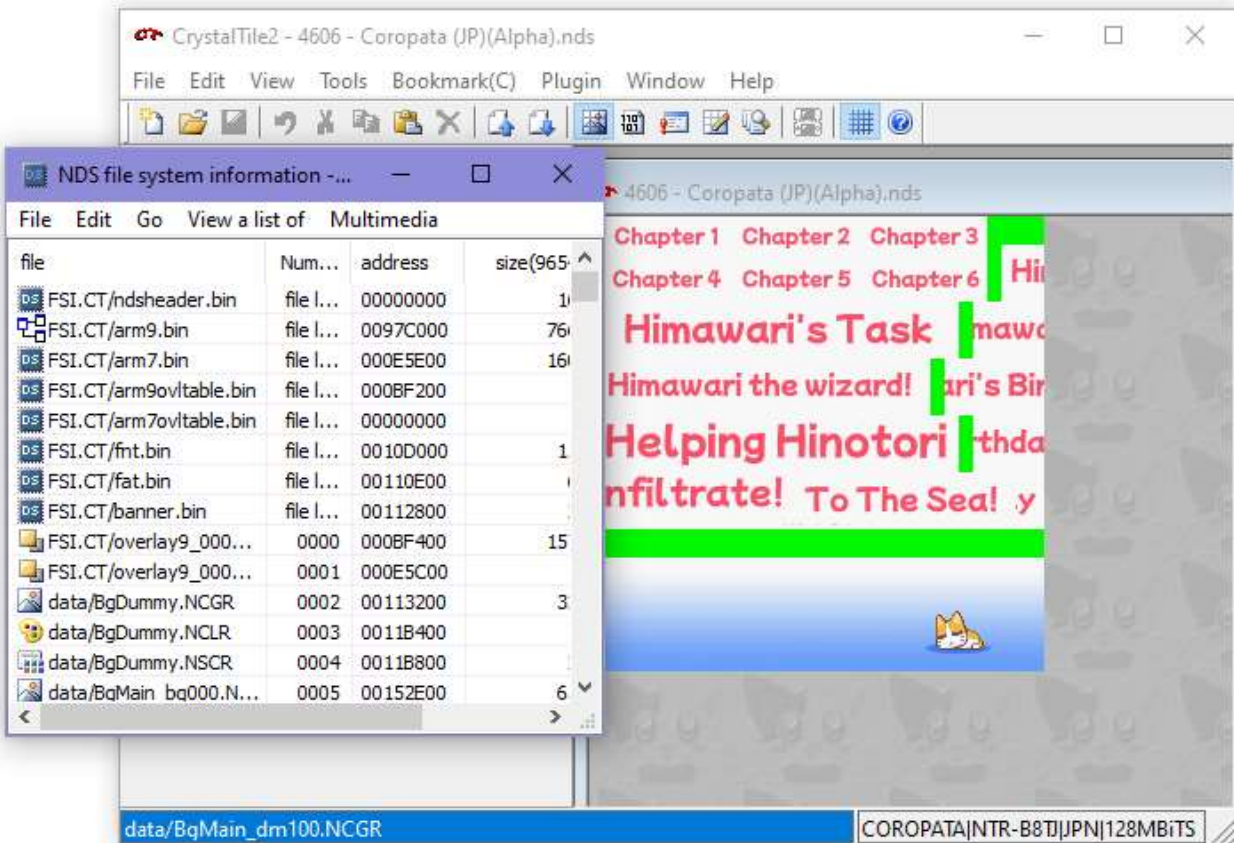


## Tools, Sites and Their Uses

In this section I will give an overview of each of the tools and websites that were used in the translation, editing and management of the game; some of these had much more use than others but each had unique and useful reasons to be included in the program arsenal.

### CrystalTile2

CrystalTile2 is a multifunction editor/extractor that works for the .nds format that DS games are packaged inside.



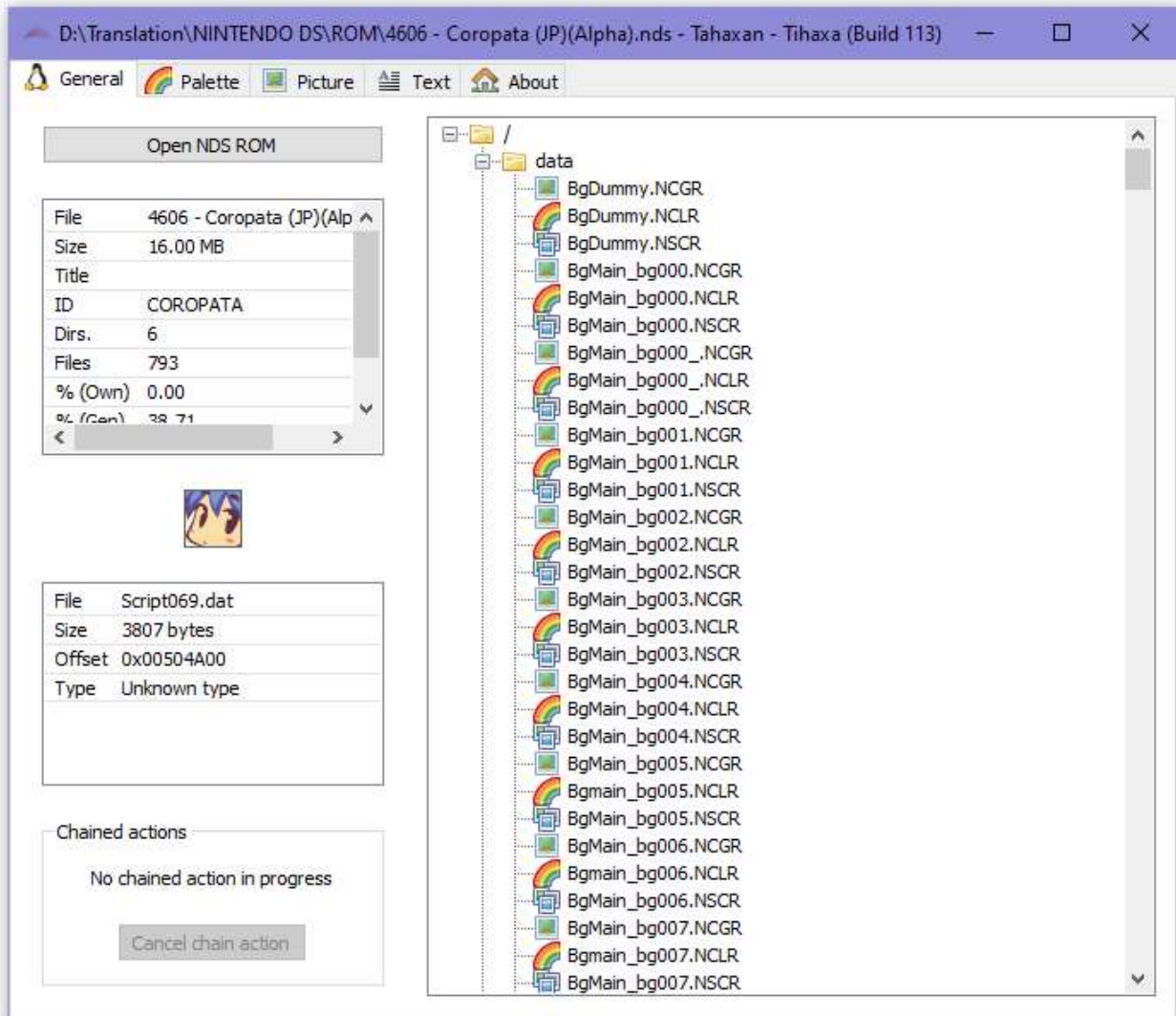
When we open Coropata in this program, we have immediate access to all the relative .bin files that run the game, the level .dat files and the script .dat files, as well as tilesets, colour palettes and music files. We can use crystaltile for essentially any form of file dumping or reinjecting that we please, with the added benefit of being able to view tilesets graphically.

Script .dat files that are extracted with crystaltile can be edited in a hex editor of your choice, then reimported into the file system and saved as an .nds file.

## Tahaxan

While functionally similar in a lot of ways to CrystalTile, and also being superseded by it, Tahaxan came in useful at the start of the translation process when we were just trying to see what types of things were hidden inside the .nds wrapper.

This program lets you view tilesets as full images, explore folders and extract a full clean rip of the contents of your DS game



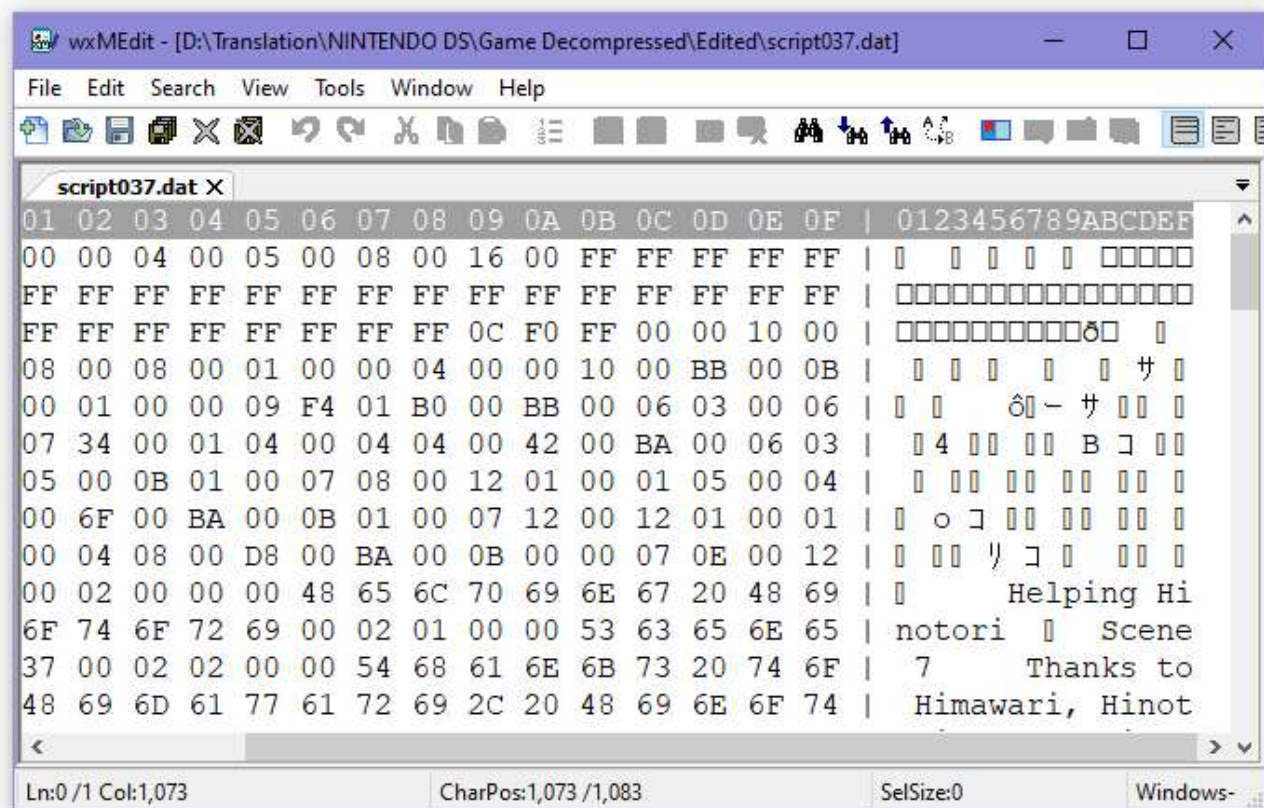
## wxMEdit

This serves as the main hex editor for the entirety of the translation. Opening the previously extracted script .dat files in this editor and changing the encoding to Shift-JIS allows us to view the cutscenes as hex, which I can then extract and save into a separate text document for ease of distribution with other members of the team.

After the text has been translated I can save these script files and also arm9.bin (which I will talk about in more length later), then using the aforementioned crystaltile2 I can reinject them into the game.

Using the buttons at the top right we can switch between hex mode and script mode; the difference not only being organization on the screen but also how deleting and adding text works in context. When in hex mode, you interact directly with the hex, which is useful for inputting certain hex codes like "0A", which indicates in a cutscene to create a line break for a box of dialogue.

When in script mode, deleting or lengthening text becomes easier (for cutscenes at least, this shouldn't be attempted when editing arm9.bin since it will crash on startup), and text starts to follow a more traditional display style.



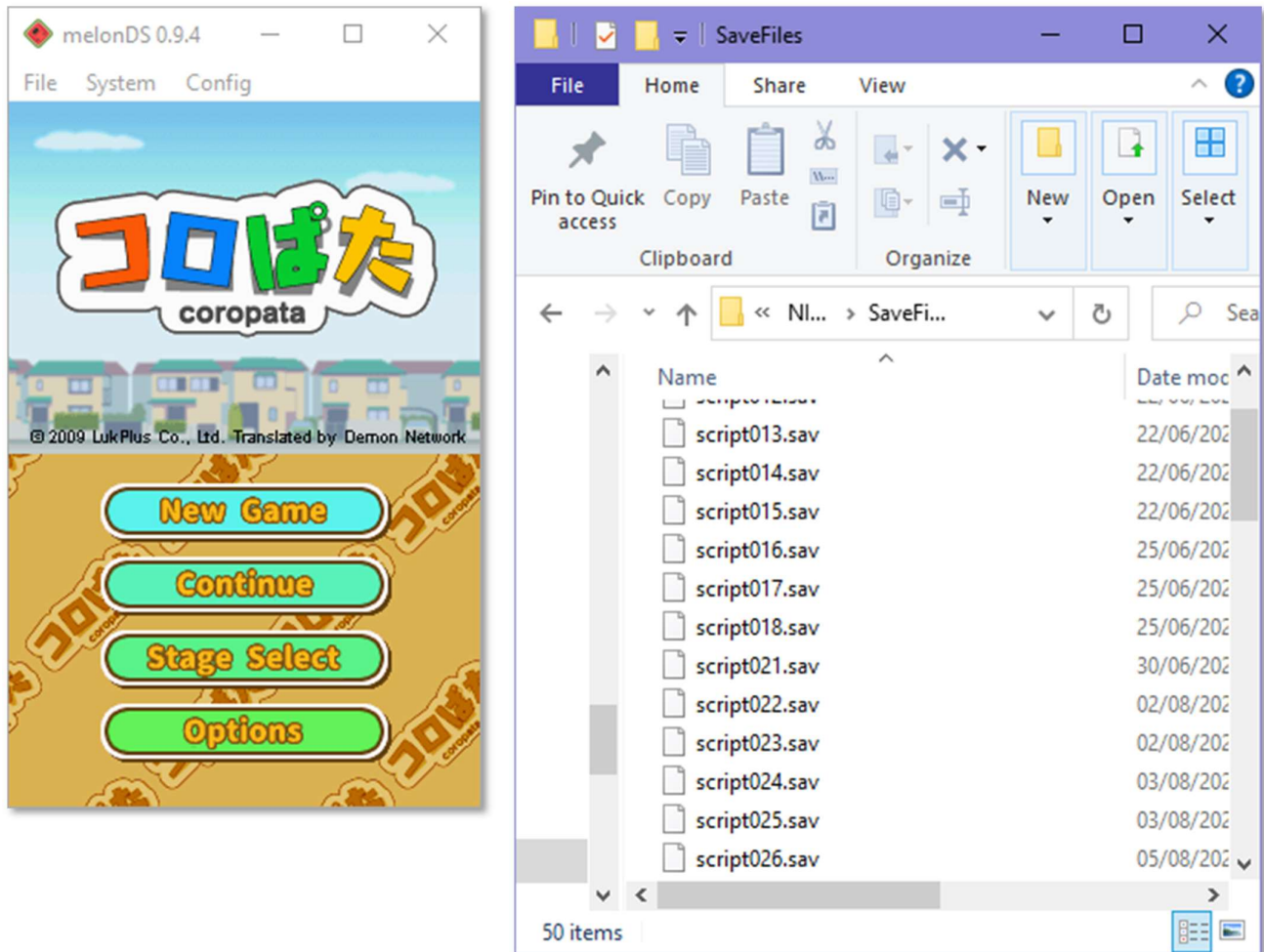
```
wxMEdit - [D:\Translation\NINTENDO DS\Game Decompressed\Edited\script037.dat]
File Edit Search View Tools Window Help
script037.dat X
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F | 0123456789ABCDEF
00 00 04 00 05 00 08 00 16 00 FF FF FF FF FF | 
FF FF FF FF FF FF FF FF FF FF FF FF FF FF | 
FF FF FF FF FF FF FF FF 0C F0 FF 00 00 10 00 | 
08 00 08 00 01 00 00 04 00 00 10 00 BB 00 0B | 
00 01 00 00 09 F4 01 B0 00 BB 00 06 03 00 06 | 
07 34 00 01 04 00 04 04 00 42 00 BA 00 06 03 | 
05 00 0B 01 00 07 08 00 12 01 00 01 05 00 04 | 
00 6F 00 BA 00 0B 01 00 07 12 00 12 01 00 01 | 
00 04 08 00 D8 00 BA 00 0B 00 00 07 0E 00 12 | 
00 02 00 00 00 48 65 6C 70 69 6E 67 20 48 69 | 
6F 74 6F 72 69 00 02 01 00 00 53 63 65 6E 65 | 
37 00 02 02 00 00 54 68 61 6E 6B 73 20 74 6F | 
48 69 6D 61 77 61 72 69 2C 20 48 69 6E 6F 74 | 
Helping Hi
notori Scene
7 Thanks to
Himawari, Hinot
Ln:0 /1 Col:1,073 CharPos:1,073 /1,083 SelSize:0 Windows-
```





## MelonDS

MelonDS is a well optimized DS emulator that allows us to run the DS version of Coropata on PC hardware. Using this, the ability to quickly refresh the game to see new changes, speed up the game, pause and dump memory, becomes invaluable in the modification and debugging of the game.



Every time that the emulator saves the game, we can repurpose the save file as a checkpoint for various stages and cutscenes. Using GitHub uploads (which will be mentioned later), save files are distributed among translators which gives us easy access to any piece of dialogue without requiring everyone to complete their own copy of the game.

Of course, it does require one person to achieve this task, so I have personally played through and created save files for each important part of the game in order to streamline the process for other people working on the project.

Savefile names correlate to a scene in the game, with the first number referring to the chapter, and the second the particular scene.

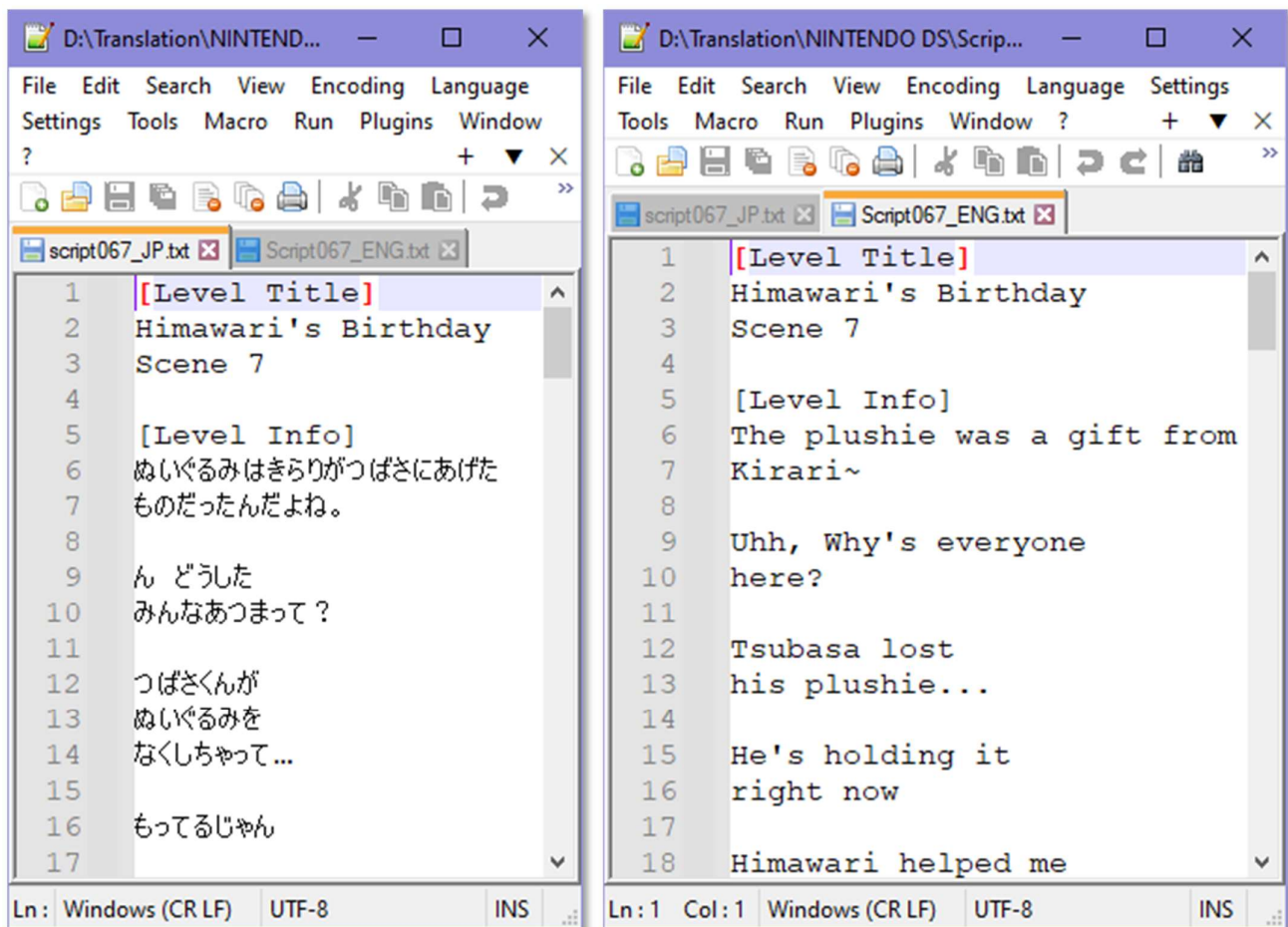


## Notepad++

Notepad++ is, as the name suggests, an upgrade and extension on the classic notepad program, usually used for its ability to deal with programming script of various types. Typically, this lies in colour coding elements of text to make the reading and organization of code simple.

For purposes of the translation, however, the most important function is actually the column count at the bottom of the screen, which shows your cursors position on the line of text. This is extremely useful when met with restrictions on the length of single strings of text, as we often see in Coropata.

For most of the cutscenes, dialogue in speech bubbles should be kept to no longer than around 25/26 characters per line, ideally short (for aesthetic purposes). For level and item descriptions, a hard limit of 30 characters exists before text is cut off the screen; This is due to the characters not exhibiting any form of word wrap as they do in the dialogue boxes.



The image shows two side-by-side Notepad++ windows. The left window is titled 'D:\Translation\NINTEND...' and has tabs for 'script067\_JP.txt' and 'Script067\_ENG.txt'. The right window is titled 'D:\Translation\NINTENDO DS\Scip...' and has tabs for 'script067\_JP.txt' and 'Script067\_ENG.txt'. Both windows display a script with line numbers on the left. The status bar at the bottom of each window shows 'Ln: Windows (CR LF) UTF-8 INS'.

```
1 [Level Title]
2 Himawari's Birthday
3 Scene 7
4
5 [Level Info]
6 ぬいぐるみはきりりがつばさにあげた
7 ものだったんだよね。
8
9 ん どうした
10 みんなあつまって？
11
12 つばさくんが
13 ぬいぐるみを
14 なくしちゃって...
15
16 もってるじゃん
```

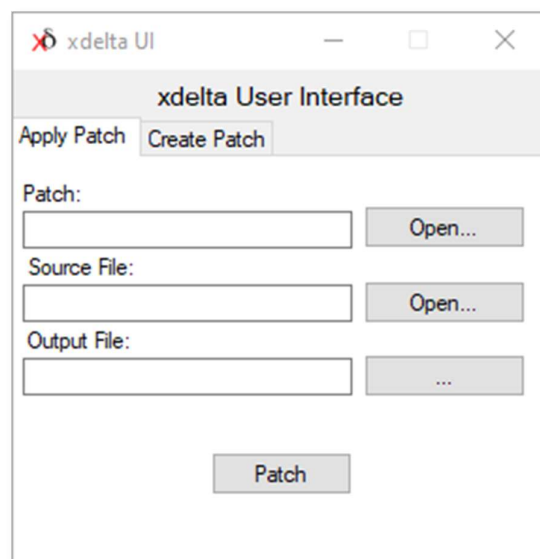
```
1 [Level Title]
2 Himawari's Birthday
3 Scene 7
4
5 [Level Info]
6 The plushie was a gift from
7 Kirari~
8
9 Uhh, Why's everyone
10 here?
11
12 Tsubasa lost
13 his plushie...
14
15 He's holding it
16 right now
17
18 Himawari helped me
```

## xDelta

xDelta is a popular patcher tool that we can use to create our patch for distribution.

Of course, distributing a Nintendo DS game, translated/modified or not, would be illegal and as a group we do not condone piracy at all.

Using a patcher such as xDelta allows us to take the source .nds file and create a patch, which is simply all the differences in code between the original file and the fully translated version. This patch can then be distributed, applied to the retail copy of someone's game, and played without any illegal distribution of files.

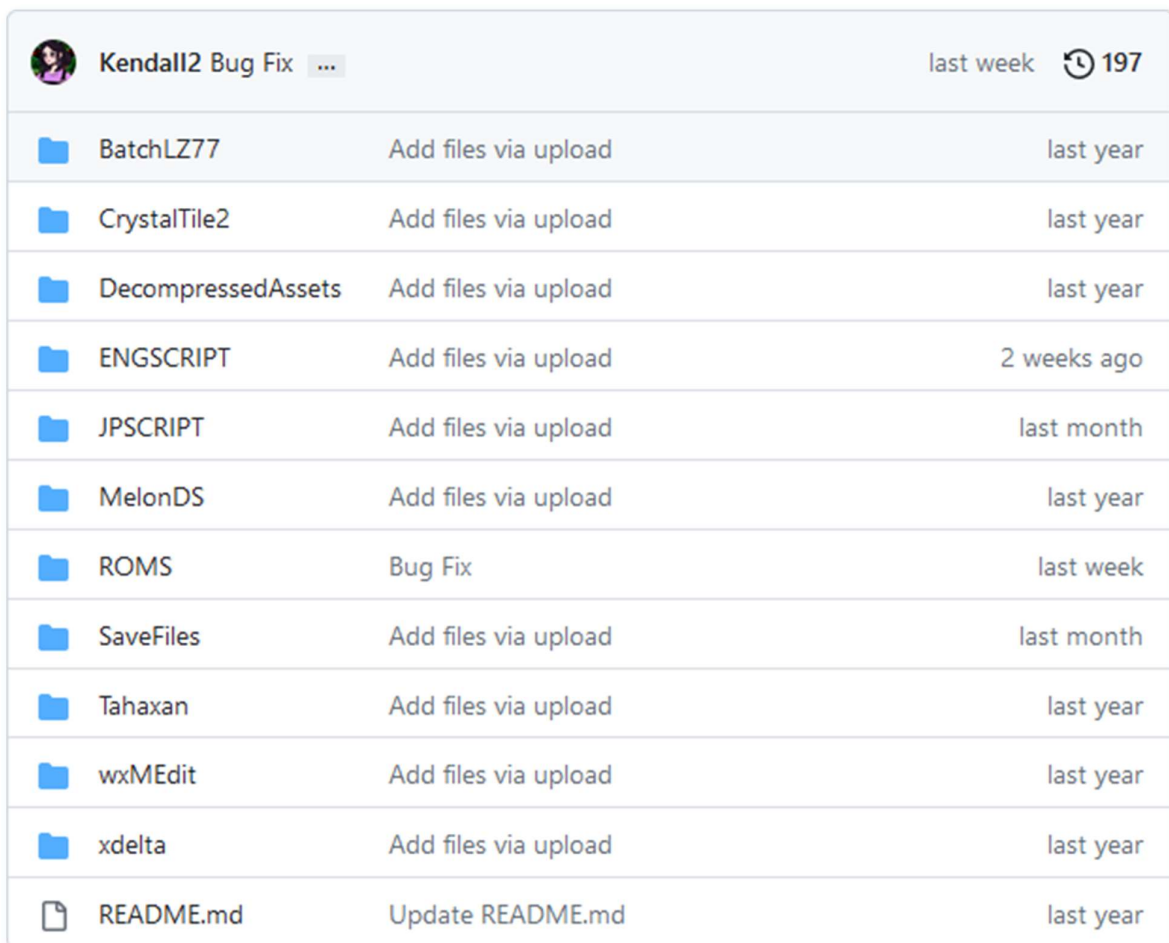


## GitHub

GitHub is a well-known and popular website that allows developers and programmers to create repositories for their code, and designate “collaborators” who can download from and “push” files to that repository.

Using GitHub, I can distribute all of the necessary files for the translation to other members of the group. When other translators have completed a scene/piece of dialogue, they can upload the file to a section of the GitHub repo labelled “ENGSCRIPT”. From here I can download their edited files and reinject them into the game.

This is also where any new save/script files will be uploaded as I create them, as well as any new builds of the game that would be convenient for others to have access to.



The screenshot shows a GitHub repository page for a user named Kendall2. The repository is titled "Bug Fix" and has 197 commits. The repository contains several folders and one file. The folders are: BatchLZ77, CrystalTile2, DecompressedAssets, ENGSCRIPT, JPSCRIPT, MelonDS, ROMS, SaveFiles, Tahaxan, wxMEdit, and xdelta. The file is README.md. The repository was last updated last week.

Folder/File	Description	Last Updated
BatchLZ77	Add files via upload	last year
CrystalTile2	Add files via upload	last year
DecompressedAssets	Add files via upload	last year
ENGSCRIPT	Add files via upload	2 weeks ago
JPSCRIPT	Add files via upload	last month
MelonDS	Add files via upload	last year
ROMS	Bug Fix	last week
SaveFiles	Add files via upload	last month
Tahaxan	Add files via upload	last year
wxMEdit	Add files via upload	last year
xdelta	Add files via upload	last year
README.md	Update README.md	last year

# Google Sheets

Filename	Status	Being worked on by	Notes	Arm9 bin	Status	% Complete
Script000_JP.txt	IN PROGRESS	Kendall				
Script000_JP.txt	DRAFT COMPLETE	Hazy				
Script000_JP.txt	COMPLETED	Mistyhands				
arm9 bin	COMPLETED	Kendall	Large file that contains level info/item info etc.	Task 01	COMPLETED	100
Script011_JP.txt	COMPLETED	Kendall	Very first intro sequence after pressing start.	Task 02	COMPLETED	
Script012_JP.txt	COMPLETED	Hazy	Cutscene 2 (C1C2)	Task 03	COMPLETED	
Script013_JP.txt	COMPLETED	Kendall	Cutscene 3 (C1C3)	Task 04	COMPLETED	
Script014_JP.txt	COMPLETED	Hazy	Cutscene 4 (C1C4)	Task 05	COMPLETED	
Script015_JP.txt	COMPLETED	Hazy	Cutscene 5 (C1C5)	Task 06	COMPLETED	
Script016_JP.txt	COMPLETED	Hazy	Cutscene 6 (C1C6)	Task 07	COMPLETED	
Script017_JP.txt	COMPLETED	Hazy	Cutscene 7 (C1C7)	Task 08	COMPLETED	
Script018_JP.txt	COMPLETED	Hazy	Cutscene 8 (C1C8)	Task 09	COMPLETED	
Script021_JP.txt	COMPLETED	Hazy	Cutscene 9 (C2C1) (First cutscene of chapter 2)	Task 10	COMPLETED	
Script022_JP.txt	COMPLETED	Hazy	Cutscene 10 (C2C2)	Task 11	COMPLETED	
Script023_JP.txt	COMPLETED	Mistyhands	Cutscene 11 (C2C3)	Task 12	COMPLETED	
Script024_JP.txt	COMPLETED	Mistyhands	Cutscene 12 (C2C4)	Task 13	COMPLETED	
Script025_JP.txt	COMPLETED	Mistyhands	Cutscene 13 (C2C5)	Task 14	COMPLETED	
Script026_JP.txt	COMPLETED	Mistyhands	Cutscene 14 (C2C6)	Task 15	COMPLETED	
Script027_JP.txt	COMPLETED	Mistyhands	Cutscene 15 (C2C7)	Task 16	COMPLETED	
Script028_JP.txt	COMPLETED	Mistyhands	Cutscene 16 (C2C8) (Final cutscene of chapter 2)	Task 17	COMPLETED	
Script031_JP.txt	COMPLETED	Mistyhands	Opening Cutscene of Chapter 3	Task 18	COMPLETED	
Script032_JP.txt	COMPLETED	Mistyhands	Chapter 3 Scene 2 (C3C2)	Task 19	COMPLETED	
Script033_JP.txt	COMPLETED	Mistyhands	Chapter 3 Scene 3 (C3C3)	Task 20	COMPLETED	CHAPTER 1 COMPLETE!
Script034_JP.txt	COMPLETED	Mistyhands	Chapter 3 Scene 4 (C3C4)	Task 21	COMPLETED	
Script035_JP.txt	COMPLETED	Mistyhands	Chapter 3 Scene 5 (C3C5)	Task 22	COMPLETED	
Script036_JP.txt	COMPLETED	Mistyhands	Chapter 3 Scene 6 (C3C6)	Task 23	COMPLETED	
Script037_JP.txt	COMPLETED	Mistyhands	Chapter 3 Scene 7 (C3C7)	Task 24	COMPLETED	
Script038_JP.txt	COMPLETED	Mistyhands	Chapter 3 Scene 8 (C3C8) C3 Finale	Task 25	COMPLETED	
Script041_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 1 (C4C1) Intro	Task 26	COMPLETED	
Script042_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 2 (C4C2)	Task 27	COMPLETED	
Script043_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 3 (C4C3)	Task 28	COMPLETED	
Script044_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 4 (C4C4)	Task 29	COMPLETED	
Script045_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 5 (C4C5)	Task 30	COMPLETED	
Script046_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 6 (C4C6)	Task 31	COMPLETED	
Script047_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 7 (C4C7)	Task 32	COMPLETED	
Script048_JP.txt	COMPLETED	Kendall	Chapter 4 Scene 8 (C4C8)	Task 33	COMPLETED	
Script051_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 1 (C5C1) Intro	Task 34	COMPLETED	
Script052_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 2 (C5C2)	Task 35	COMPLETED	
Script053_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 3 (C5C3)	Task 36	COMPLETED	
Script054_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 4 (C5C4)	Task 37	COMPLETED	
Script055_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 5 (C5C5)	Task 38	COMPLETED	
Script056_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 6 (C5C6)	Task 39	COMPLETED	
Script057_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 7 (C5C7)	Task 40	COMPLETED	
Script058_JP.txt	COMPLETED	Kendall	Chapter 5 Scene 8 (C5C8) Outro	Task 41	COMPLETED	

For any project of this scope, organization is key; Using google sheets (a free alternative to Excel), every bit of progress we make can be tracked, and sections of the game can be marked as either “IN PROGRESS”, “DRAFT COMPLETE” or “COMPLETED” to indicate their status, and make sure people aren’t working on the same scene at the same time.

The “DRAFT COMPLETE” stage indicates that something has been translated, but before we have had a chance to look over the text, workshop any changes and then place into the game. Once a piece of text has been checked and injected into the ROM, the status is changed to “COMPLETED”.

As seen in the image, a section was used to keep track of cutscene progress, and a separate area was used for the progress of the arm9.bin file, since it contained all the level and item data and thus was exceedingly large.

Functions in sheets/excel also allowed us to calculate real time percentages of our progress, and create bar graphs which give us a visual representation of our efforts.



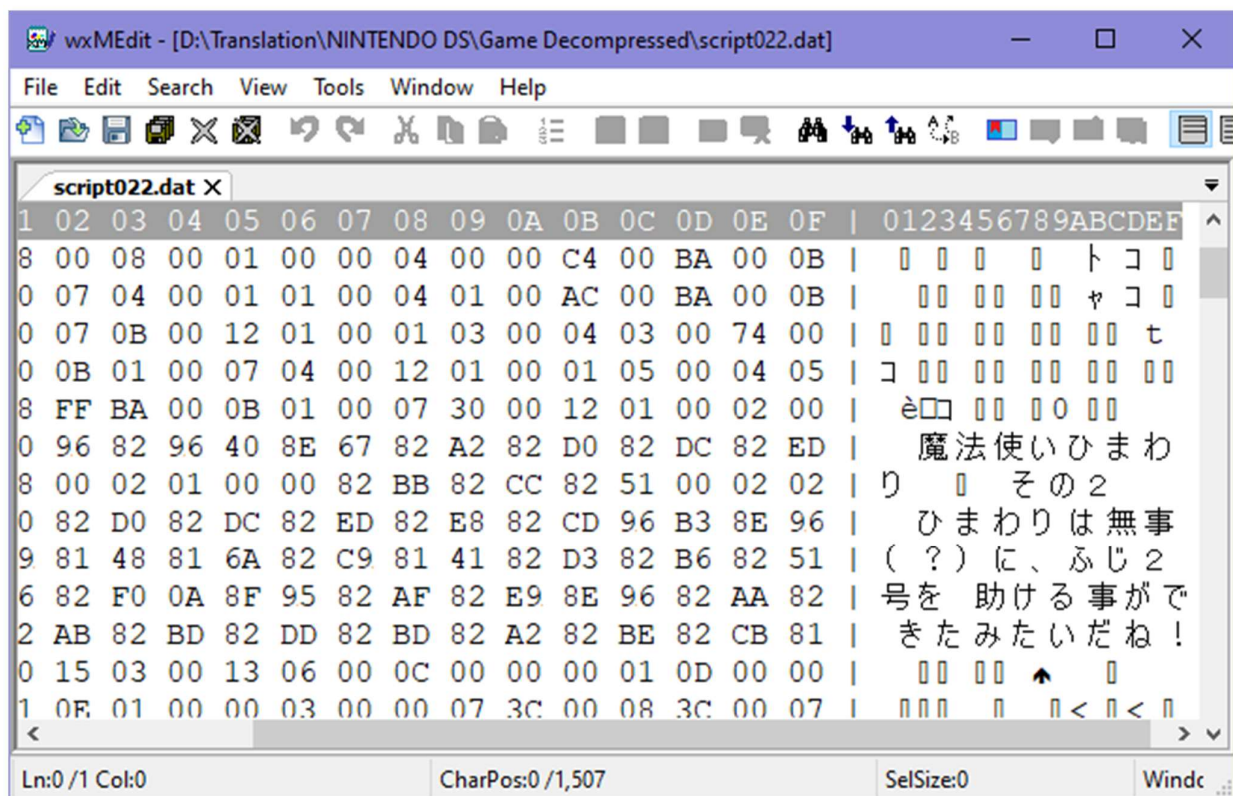
## Translating The Game

As I have mentioned before, viability is the name of the game when it comes to translation in a personal project capacity. All of the elements that need to be changed into English must also be available for us to edit and modify without any coding or major changes to the game outside of our scope of knowledge.

In this section I will describe the process of translating the game, which involves checking all assets are accessible to us, extracting the text, translating/localizing to English and then reinjecting into the ROM.

### Text Access

At first, access to the game's text seemed trivially easy; I could simply load up any of the "ScriptXXX.Dat" files into wxMEdit, change the encoding to SHIFT-JIS, and full strings of uncompressed Japanese text would be shown to me.



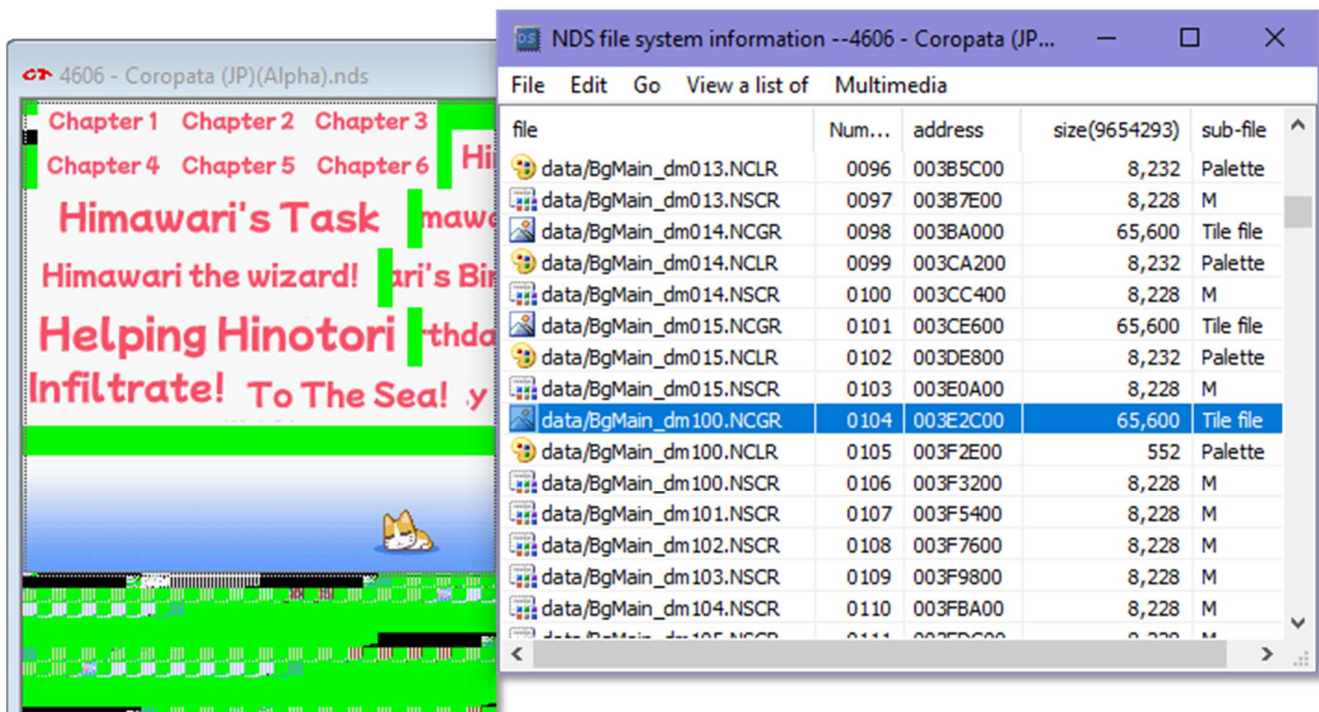
Of course, there is a large amount of hex that points to programming for the game, which we can use to learn how certain functions are called within the hex editor (e.g. line breaks, special characters etc.)



## Image Access

Using crystaltile, access to images is fairly simple. With the Nintendo DS, much like previous Nintendo handhelds, images are actually stored as “Tiles” or “Tilesets” rather than just some kind of compressed image; This allows the DS to store its graphics in a very low file-size manner, but means that each image is actually a split of 3 different files

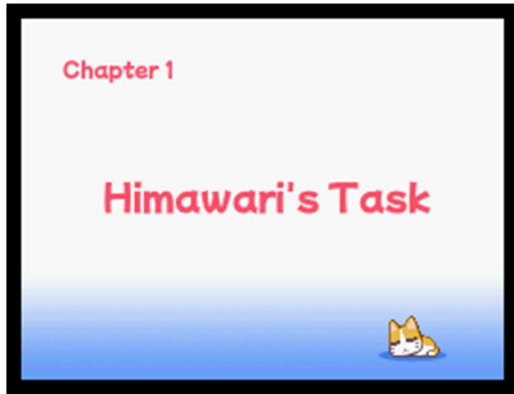
These are: A palette file, which tells the game which colour set (AKA the palette) to change to, a tileset file, which is a collection of possible tiles for that palette, and then for individual images a file which arranges the tiles in specific ways to create a final image. For those unfamiliar with this technique, it’s probably better to show this visually than try to describe it through text.



Above is the tileset “BgMain\_dm100.NCGR”, this shows tiles for every single chapter intro screen in the game; Whenever a new chapter is started, tiles will be taken from this image, using the currently selected palette “BgMain\_dm100.NCLR”, and reconstructed into the final image.

The useful thing about crystaltile is that it allows us to copy the bitmap of this tileset or a finished image, then paste that into a preferred image editor and directly manipulate the pixels, before letting us copy it back into crystaltile and save into the ROM. This makes editing images trivial, but I would recommend only editing tilesets directly; Editing of finalized images often creates errors as it’s difficult to tell if the tiles you edit aren’t going to be used elsewhere and create a mess.

You can see some of the finalized images underneath the selected tileset and palette. They appear as .NSCR files, such as “BgMain\_dm100.NSCR”, which when selected will combine the tileset and palette files into this formation:



Which, as you can see, is the chapter introduction screen for the first chapter of the game. By using tilesets in this fashion, every single chapter introduction screen in the game can be created using “arrangement data” rather than actual pixels. Each image is made of the same tileset and palette swap, and so creating a new image requires a much lower amount of memory.

### Arm9.bin

Arm9.bin is a very large file inside the game’s ROM that includes most of the programming for the game, but also includes text for item names, item descriptions, level names and level descriptions. This was a troubling issue at first because I was unaware that the .bin file contained said text, but it was soon found and able to be edited like the other cutscene .dat files.

The difference between arm9.bin and the normal cutscenes is that there is absolutely no room for change in the size of the file. Cutscenes, as we will get into later, can vary in size from the original and have specific hex codes to add in extra line breaks etc. However, if arm9.bin differs from its original size, the game will crash when started.

To circumvent this, sticking to strict limits, making sure that text doesn’t “overflow” its original byte size, and also adding blank “junk” data, when necessary, is extremely important. I kept track of the progress of arm9.bin with an individual section of the progress spreadsheet, as it was such a large file and took a substantial amount of time to translate.

Arm9.bin	Status
Task 01	COMPLETED
Task 02	COMPLETED
Task 03	COMPLETED
Task 04	COMPLETED
Task 05	COMPLETED
Task 06	COMPLETED
Task 07	COMPLETED
Task 08	COMPLETED
Task 09	COMPLETED

This was split into 128 sections for the 128 levels of the main game.

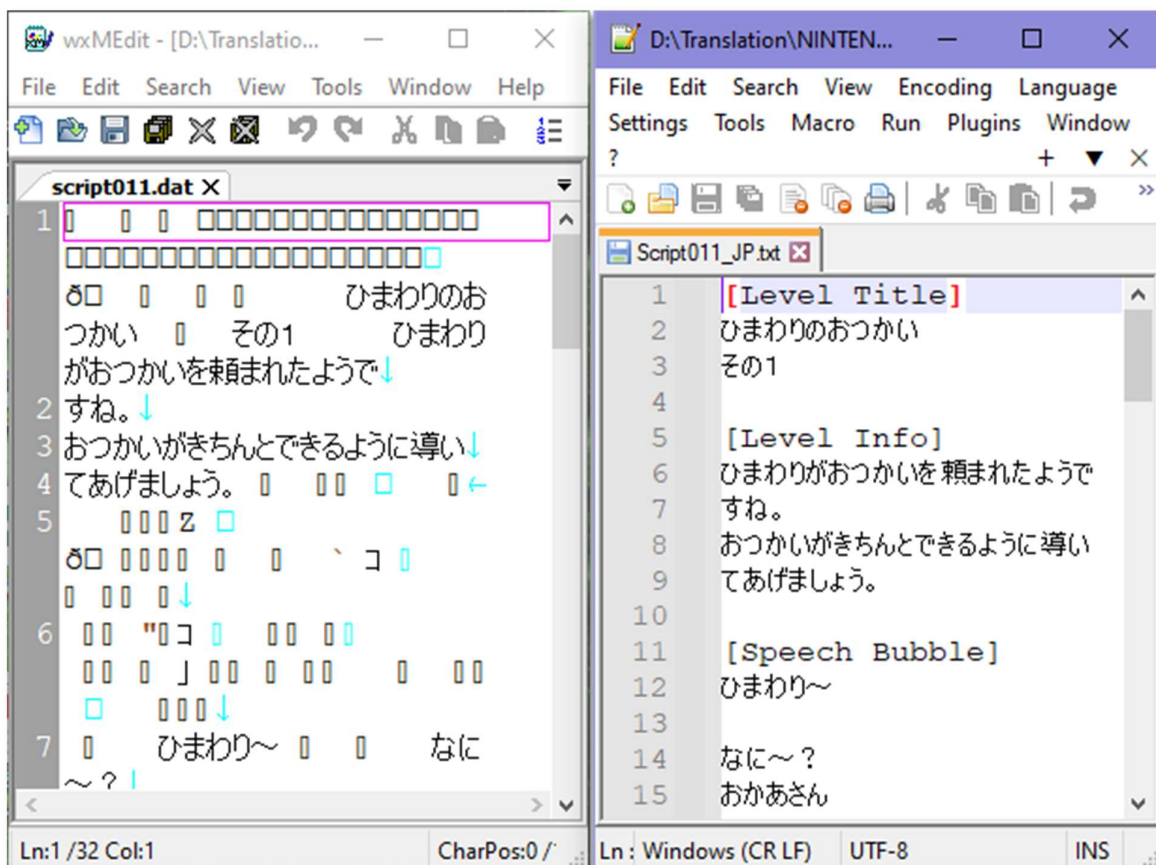
## Cutscenes

In Coropata, every 3 levels a new cutscene is shown to us. These scenes introduce characters, have them communicate and advance the story. For the purpose of translation, they are the easiest and most forgiving things to tackle.

The first thing I will do is play the game until the cutscene appears, and as soon as the “saving” logo is shown at the bottom right of the DS screen, I will extract the current save file before renaming it to “Script[current scene number].sav”; From here I can share this file to GitHub for anyone else who wishes to jump to that point in the game.

The first scene of the game will be script011.dat, with 11 pointing to chapter 1, scene 1. The first thing I’ll do is open up this .dat file in wxMEdit, then switch to the “script” view instead of the hex.

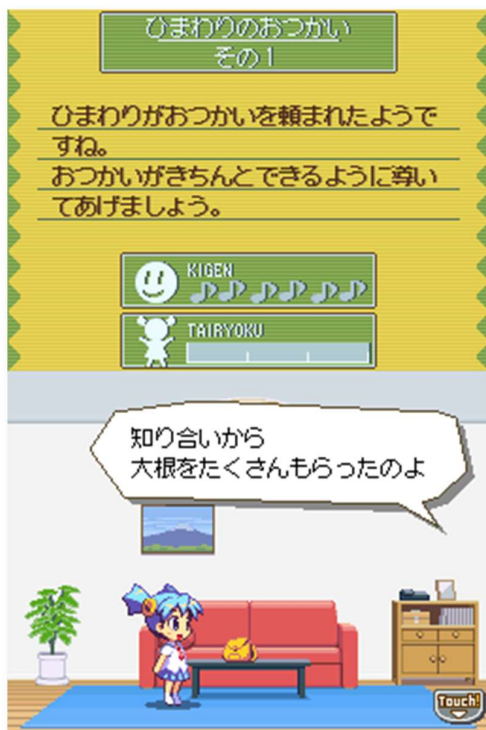
In the script view, lines of text can easily be copied over into a notepad++ document, which I then save and distribute on GitHub also.



From here, the typical workflow involves clicking through the cutscene in the emulator, and translating the Japanese into English! Everyone has a different preference for how they do things, but for me I like to duplicate the file and have both the Japanese and English versions side by side on the screen.

Coropata's cutscenes do make it easier for us by having each line require the player to tap the screen to progress dialogue. This means we don't have to abuse the pause feature of the emulator and can move along at our own pace.

There are a few ground rules and important quirks when working on cutscenes, but generally they are very flexible. I say this because the dialogue boxes in these scenes are actually dynamic and malleable to custom amounts of text, as well as line numbers.



In general, translators will want to stick to under 25~ characters or so for each line, and no more than 4 lines in a single dialogue box. In theory there can be more characters per line before it cuts off the screen, but aesthetically it starts to look a bit rougher as the edges of the speech bubble clip off of the screen etc.

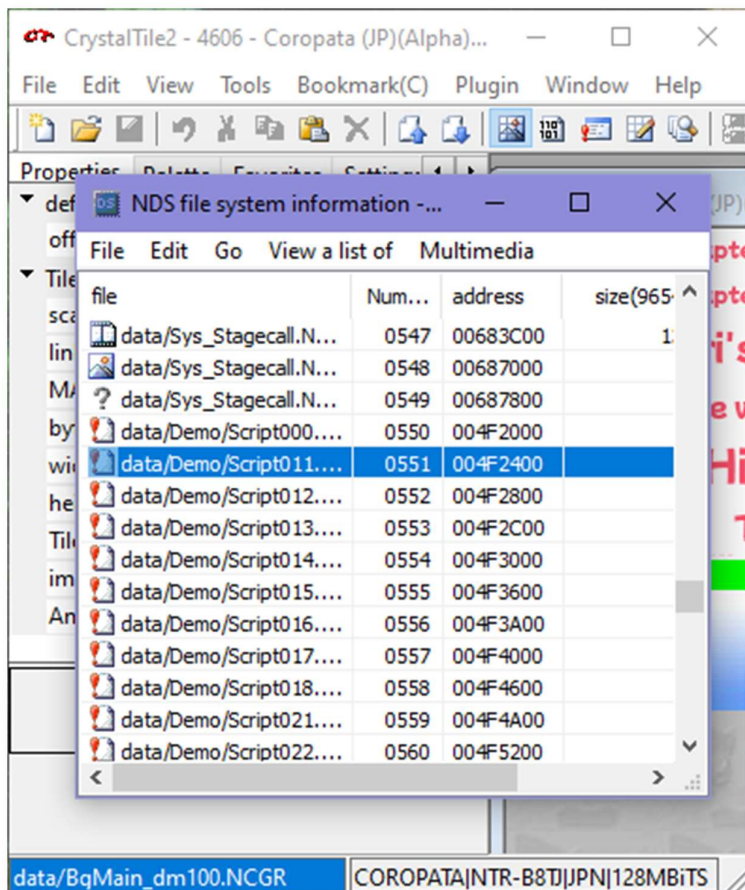
Using the hex code "0A", a new line is formed in the speech bubble. In the images on this page, you can see a bubble that was converted from 2 lines of text to 3.



In terms of consistent style, of course the title will remain the same in every scene, the “Scene” will increment by one for each cutscene, and we opted to stick to a “no full stops” punctuation style. What this means is that the use of “!”, “?”, “...” and “~” to end a sentence are acceptable, but no full stops. This has two functions, the simplest is that it saves us a valuable bit of space that isn’t taken up by an unnecessary character, the second is that it feels less harsh in tone, which we felt matched better with the more playful and aloof attitude that the game generally has.

The title in each of these scenes also has a character limit; any more than 15 characters will have the ends of it start to become distorted and chopped up. Some chapters have names longer than 15 characters, like “Himawari’s Birthday”, so it is important to come up with appropriate abbreviations that don’t appear too odd. For this example we settled on “Himawari’s B.Day”.

After the scene is done, the completed English script is uploaded to GitHub in a folder labelled “ENG Script”, and the status of the scene is set to “DRAFT COMPLETE” on the progress spreadsheet. Once the draft script has been double checked and, if necessary, amended/cleaned up for visual appeal, we can reinject it into the games ROM using crystaltile.



Before this can take place, it needs to be remade as a .dat file, just like the original script file was, so I would manually go through the translated text and insert it into a duplicate of the original .dat file in the correct locations, making sure to insert any additional line breaks with the 0A hex code.

Then, I can take the modified .dat file and import it into the location of the original using crystaltile. From here it's as simple as saving the ROM.



## Levels, Items and Descriptions

This is where things start to get tricky in terms of keeping an adequate and accurate translation which preserves the original meaning of the Japanese, whilst sticking to very harsh space constraints.

As mentioned before, arm9.bin is the file that contains the data for all the level titles and their descriptions, as well as the in-game item names and their descriptions. Due to the way arm9.bin works, we cannot add any additional line breaks or extra characters without breaking the way the game functions.

There is a silver lining in the way that Japanese is handled in hex, which is that one Japanese character of any type (Kana or Kanji), is represented by 2 hex numbers, while English characters are represented by only 1 hex number. This effectively gives us 2 English characters for every single Japanese character found in arm9.bin.

There are many situations where an item is labelled with only 2, sometimes just 1 Japanese character, which often happens with Japanese nouns, especially if they use kanji. When this happens, it's time to brainstorm what the most appropriate solution would be.

For a handful of items, the best course of action is to abbreviate the name and add further explanation in the description underneath, since there's typically a lot more space available for use. A good example of this would be the reflecting mirror item.

The original Japanese used here is “鏡”, which of course only gives us 2 English letters of space to translate with. I opted for just using the abbreviation “LM”, which is then explained below as a “Lightreflect Mirror™”, which sticks to our strict spacing but also gives all of the information to the player about what this item is and its function.

The description also had a relatively low amount of space, but using the trademark symbol as a tongue-in-cheek method to give it a product-esque name, it's possible to convey all the necessary information in a very cramped form factor.



There are some elements that cannot be done perfectly due to the file's nature. For example, the level titles are perfectly spaced and hard coded to contain the name of the level that the developers originally wrote. This means that in the English version, if the translated title is shorter in hex size than the original, we have to fill the space with junk data.



In order to format the titles so that they appear centred underneath the task number at the top of the screen, this data had to be blank spaces, using the hex code "20". When in the level itself, the title looks good, is positioned correctly, and the blank data allows us to keep the file size. However, a known issue is that in the stage select screen, the level titles can appear disjointed and strange because it actually reads and displays the blank spaces in a different way.

Of course, we could just start the level titles from the original starting point, which would fix the appearance in stage select, but would make the in-game titles look odd and have them appear randomly across the topmost green box. We decided the former method was more appropriate.

Just like the aforementioned "Lightreflect Mirror" situation, there are also level titles that are written with so few characters that it makes fitting English words into the small space very impractical. For these cases, the closest possible approximation has been used to convey the same or similar meaning, whilst staying within the sizing limit. As a result, some of the level titles might seem a tad uninspired compared to their original Japanese counterparts, but without a major change in the games code, this isn't something that can be helped.

Also of note is that the general line limit for arm9.bin, since everything is contained within the top screen, is around 27-28 characters before it starts to become cut off. It does allow for more room per line than the cutscenes, but we cannot add any new lines should the space run out. Formatting is an issue here since lines are not dynamically word-wrapped like in dialogue boxes, so workshopping of each line has to be done to make sure it doesn't have any large and obvious gaps.

## Stylistic Choices

As mentioned before, we chose an approach for grammar that seemed more light-hearted, in order to fit the cheerful attitude of the game; Sentences can end in “!”, “~”, “...” and “?”, but periods are not used to end a sentence.

There are a few other decisions relating to character naming, honorifics, etc, that I will detail here.

### Honorifics/Name nomenclature

Coropata, and many other Japanese games, use honorifics common to Japanese speech, including -san, -chan, -kun, -tan when a character is referring to other people. There are a few different ways to approach translating these; Some opt to convert them into nicknames or alter the way that the rest of the sentence feels in order to reflect said suffixes. Other translations will leave the suffixes untouched, which can work but does presume that the audience understands the suffix conventions of the Japanese language.

For Coropata, we chose the simplest path of just omitting the suffixes; It doesn't alter the rest of the content of the sentence and handily saves us space for other, more important elements of text.



### Oscar/Geotail

Oscar and Geotail refer to the cat and dog characters that Himawari regularly interacts with. The original names for these two are “Fuji-2-gou” and “Jiote-ru” respectively. These might seem like odd choices for animal names at first, but they are actually references to two different Japanese space satellites.



Geotail's name was left as-is, since it closely resembles a “normal” name, with the “tail” part feeling somewhat related to household pets. Fuji-2-gou feels a lot more alien when directly translated, so we opted for the original satellite's alternative name: Oscar.

### Geotail's speech

Geotail is a dog that doesn't have the ability to speak to humans, and therefore his dialogue is reduced to various onomatopoeia that represent dog barks, pants, whimpers etc. We tried our best to use sounds that would be instantly recognized in text form, such as "Arf!" for barking noises, but resorted to the old-fashioned method of using asterisks to represent sound effects in place of his whimpering.

For these scenarios, the phrase "\*whimper\*" is used, simply because any other onomatopoeia felt like they could cause confused or uncertainty in the type of sound that was being expressed; something that wasn't ambiguous in the original Japanese.



### Hinotori

During the course of the entire game, there is a running joke that Hinotori tries to guess the underwear colour of any girl he talks to, in order to really drive home that he's a perverted old man.

When translated into English, and given the age of some of the characters he does this to, including Himawari herself, it seems unnecessarily inappropriate. Hinotori has plenty of opportunities to get across the point that he is a bit of a pervert, but I think this one in particular is seen as appropriate in the Japanese idol culture space; something that doesn't really have a direct parallel in the English-speaking world.



As a result, we've felt that it isn't needed in the game and doesn't really add to the character of Hinotori in a necessary way, so we've opted to remove this running joke for the English audience. There are still plenty of ways in which Hinotori's character is shown as gross and creepy, just not this one in particular.



## **Post Translation**

Finally, after translating all of the levels, items, cutscenes and images; After compiling the ROM into a fully English version, we can use xDelta to create a patch.

After almost an entire year in development, with me and my friends working together as Demon Network to get this patch finished, we've finally done it!

I've done some bug tests and a few updates to the patch to amend a few word wrap issues, but with patch v1.2 I think everything is up to snuff. The patch has been tested and works on the following physical hardware: Original DS, DS lite, DS LL, DSi, DSi LL and 3DS.

The next steps going forward are to distribute the patch on romhacking websites, and then the demon network site ([demonnetwork.co.uk](http://demonnetwork.co.uk)), and finally my own personal website ([kendalls.garden](http://kendalls.garden)).

I'd like to thank everyone at Demon Network for their dedication and hard work over the year; it's a really rewarding feeling seeing something you've put so much time and energy into finally come to fruition and be realized as something tangible.

## **Reflection**

This project isn't the first translation patch I've done, but it is the first time I've created and been at the helm of a team. It's the first time that I've had this much input into the extremely technical side of things that fall outside the realm of translation, but I've gained a lot of useful knowledge about programs used in the deconstruction of DS ROMs, and going forward feel much more confident with similar projects.

Something that would've been useful is a much deeper knowledge of a coding language for creating macro scripts. I used to study java in school and enjoyed it very much, but have since forgotten a lot of the framework that would've helped me a lot when it came to the slow and tedious parts behind the scenes.

A lot of the time, I would be manually copying and pasting lines of text in order to create shareable files for the rest of the team, but being able to code my own program that would automate this program would've been very helpful. Still, given the circumstances I think we did the best we could.