Encryption for Complete Quality Preserving Reversible Data Hiding Method in Animated GIF

Team Members: Lee Jian Hui, Naavin Ravinthran, Low Jun Jie

Table of Content

End User Guide

Environment

The user needs to have a python interpreter installed on his machine.

Prerequisite Installation Guide

Our program is already bundled as a python package, so that relative imports are able to work. It is strongly recommended to install it on a python virtual environment, but it is optional.

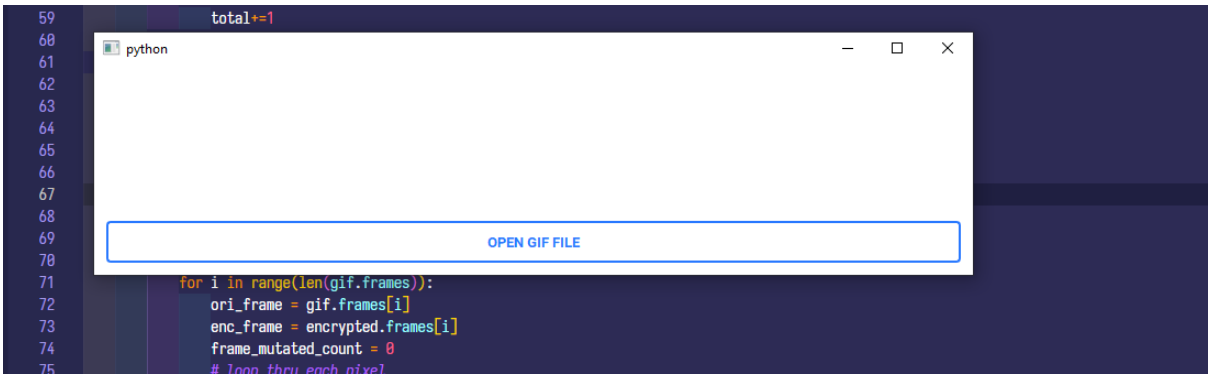
For a developer that wishes to run the project, they have to perform the following steps:

1. Create a python virtual environment using **python -m venv [name of env]** OR other equivalent methods
2. Activate the virtual environment, go to your folder where the virtual environment pyenv is installed and then activate the script: ./Scripts/Activate.ps1 on Windows, or ./bin/activate for Linux.
3. Run **pip install git+https://github.com/GIF-ME-HD/gif_me_hd_proto.git** (Or **pip install git+ssh://git@github.com/GIF-ME-HD/gif_me_hd_proto.git**) in order to configure and install the project as a python package.
4. Run the GUI with **python -m gif_me_hd.gui**
5. Use the library in your scripts by using **import gif_me_hd**
6. To uninstall run **pip uninstall gif_me_hd_proto**

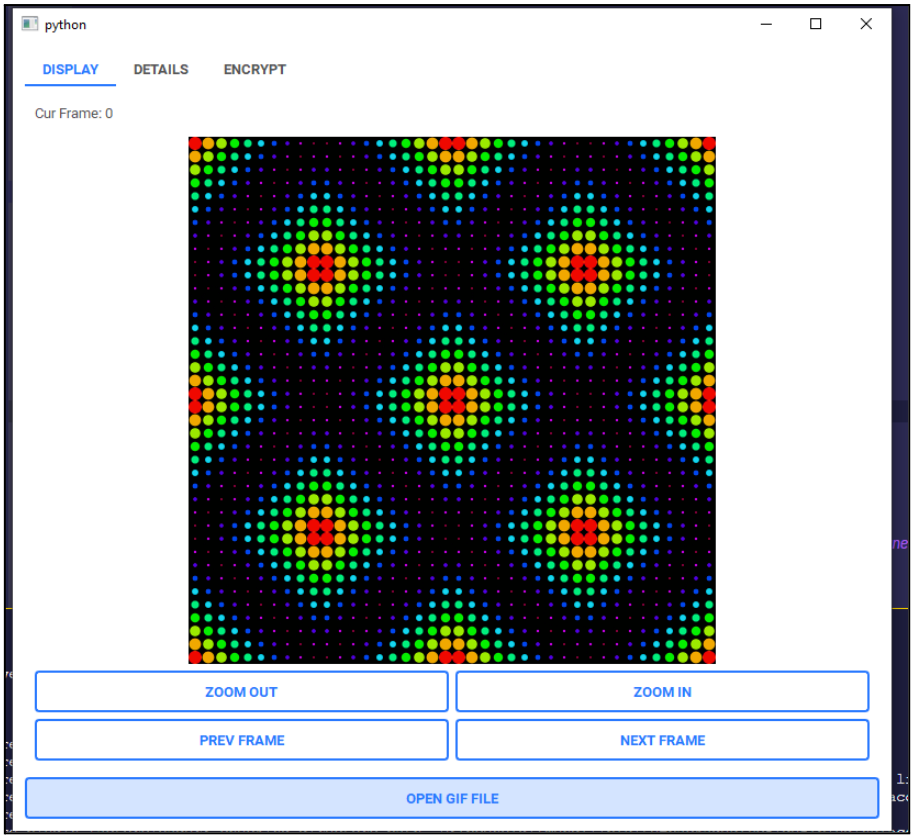
GUI

Start Up: Choose GIF to parse & load

Upon booting up the GUI application, the user will first need to choose a GIF file to load into the application which will be parsed under the hood.

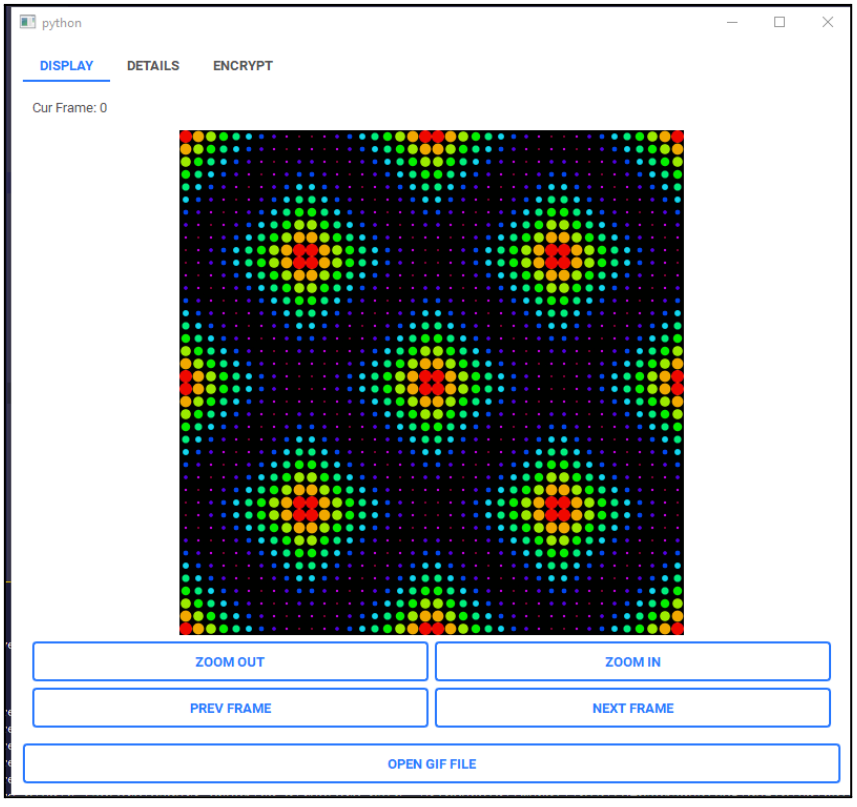


After that, the chosen Gif image will be loaded into the application, and then the user can analyse the contents of the GIF file

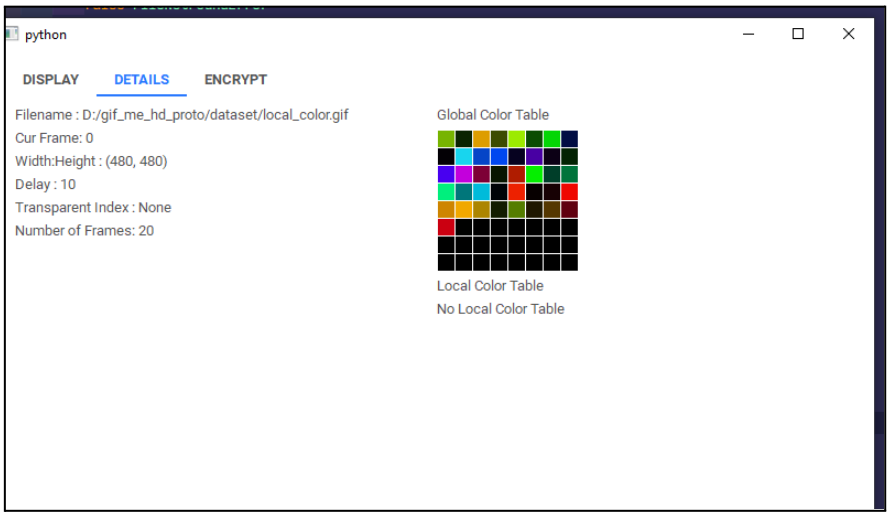


Display, detail and encrypt tabs

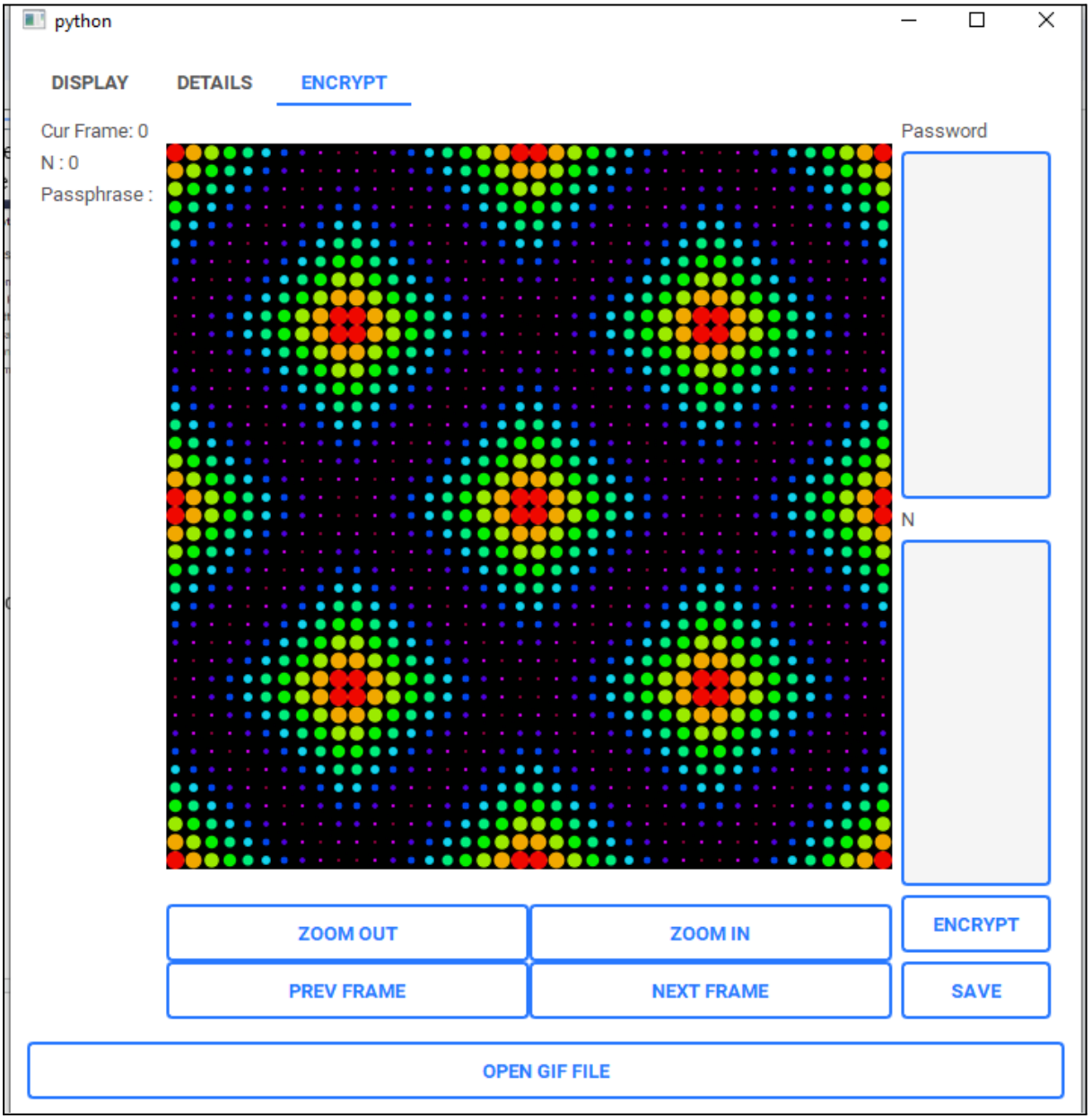
The display tab showcases the original unencrypted gif file and the user can zoom in and out as well as navigate in between different frames in the entire gif sequence.



The details tab will display the meta data of the Gif file as well as the global color table and the local color table for the current gif frame if it exists.



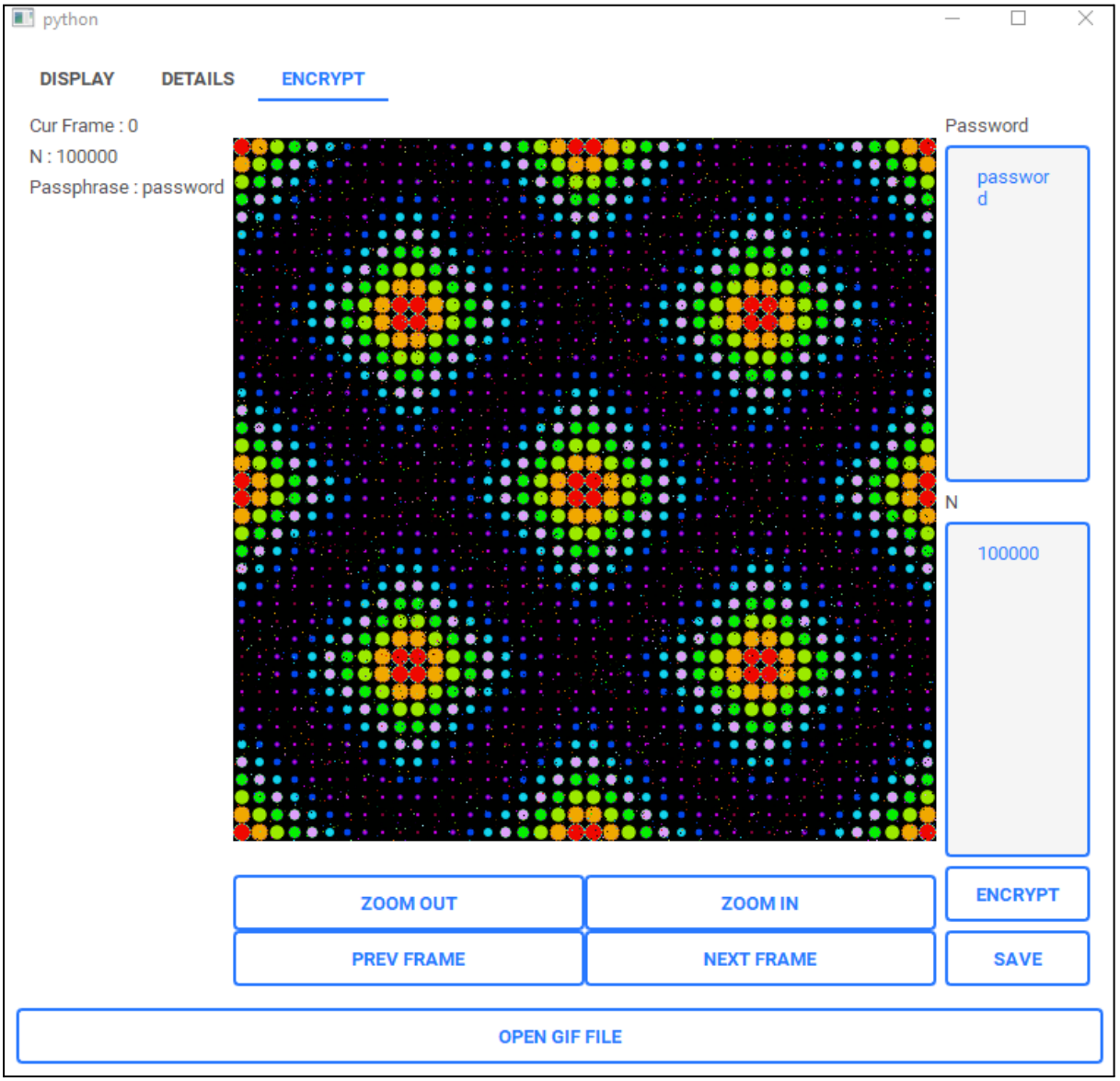
Encrypt window tab lets the user be able to encrypt the parsed gif file by providing a suitable password and N threshold value.



Encrypt by providing an N and a password (in the encrypt window tab)

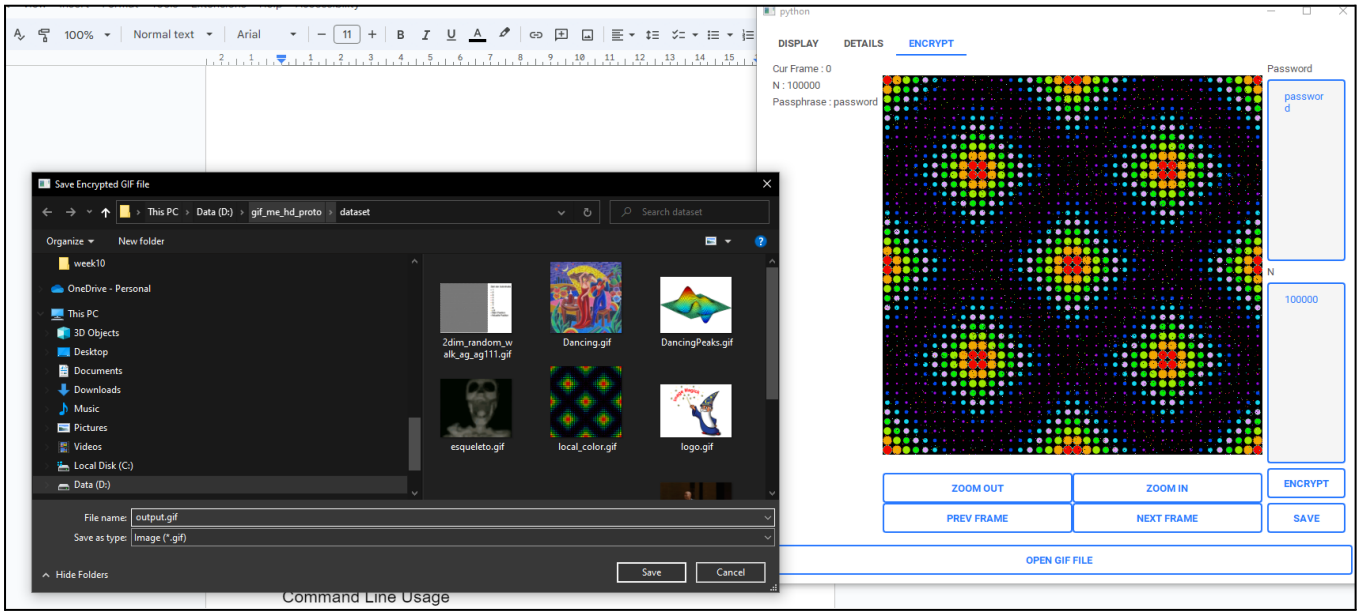
After providing the password and N value, in this case it will be “password” and 100000 respectively. After pressing the encrypt button, the user will be greeted with the encrypted Gif Data object presented in the UI.

This is the frame with index 0 of the encrypted GifData object as shown in the screenshot below. We can see there are mutated pixels in this frame.



Saving the encrypted gif file

The user can export the encrypted GifData into a gif file by providing the appropriate file path.

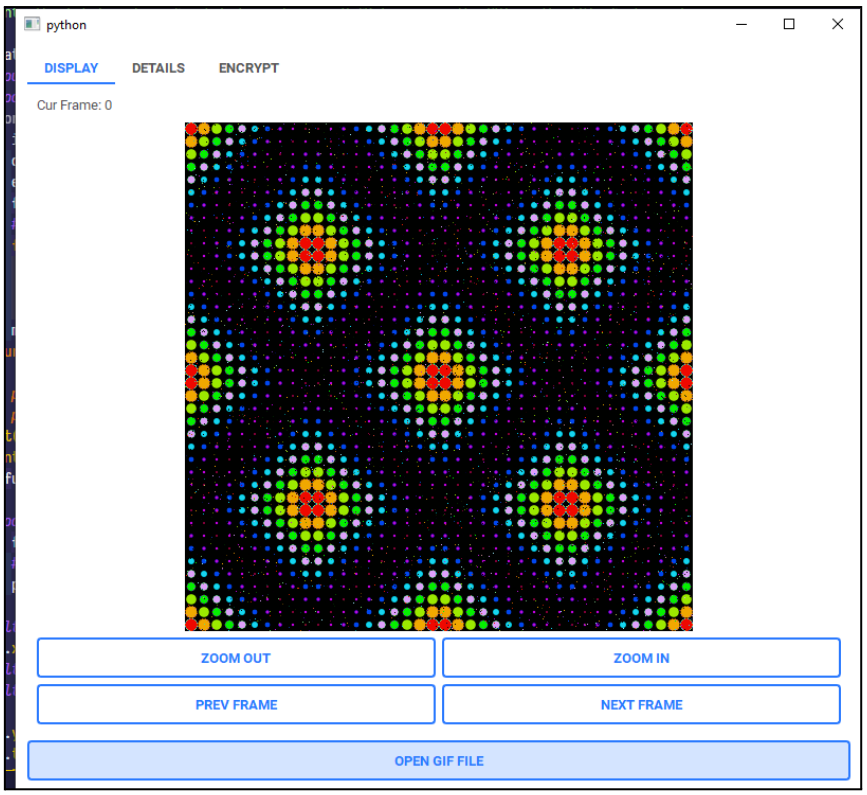


Command Line Usage

Decrypt an encrypted gif file

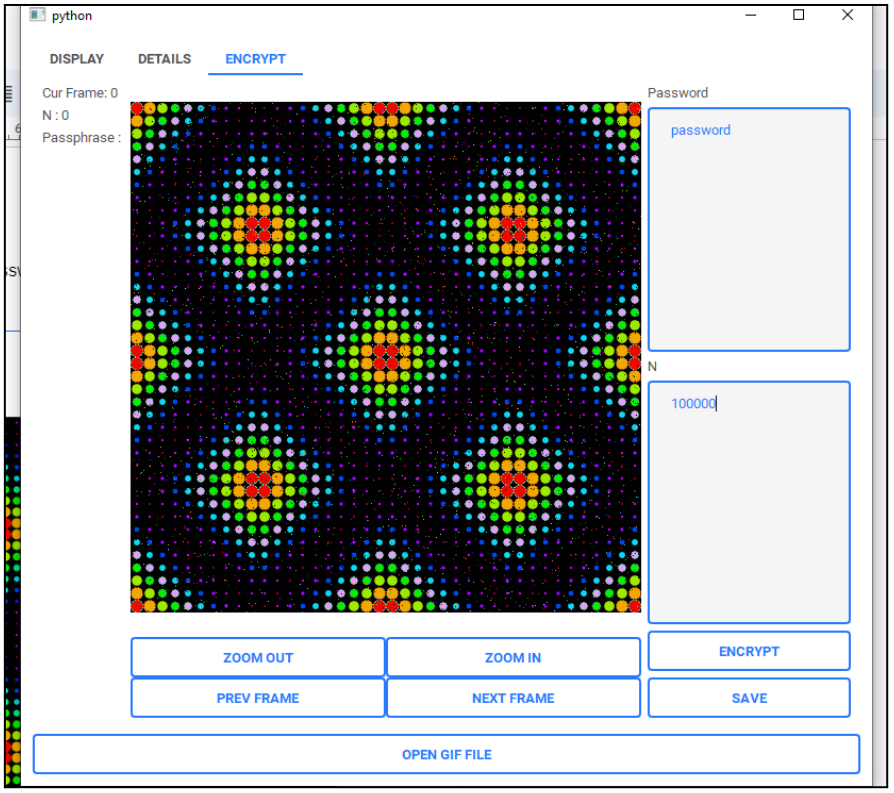
To decrypt an encrypted gif file, the user has to have the password and N value used in the encryption process which will normally be stored in the comment extension of the GIF file.

This screenshot below shows the earlier encrypted gif file that was exported, now parsed and loaded into our application again.

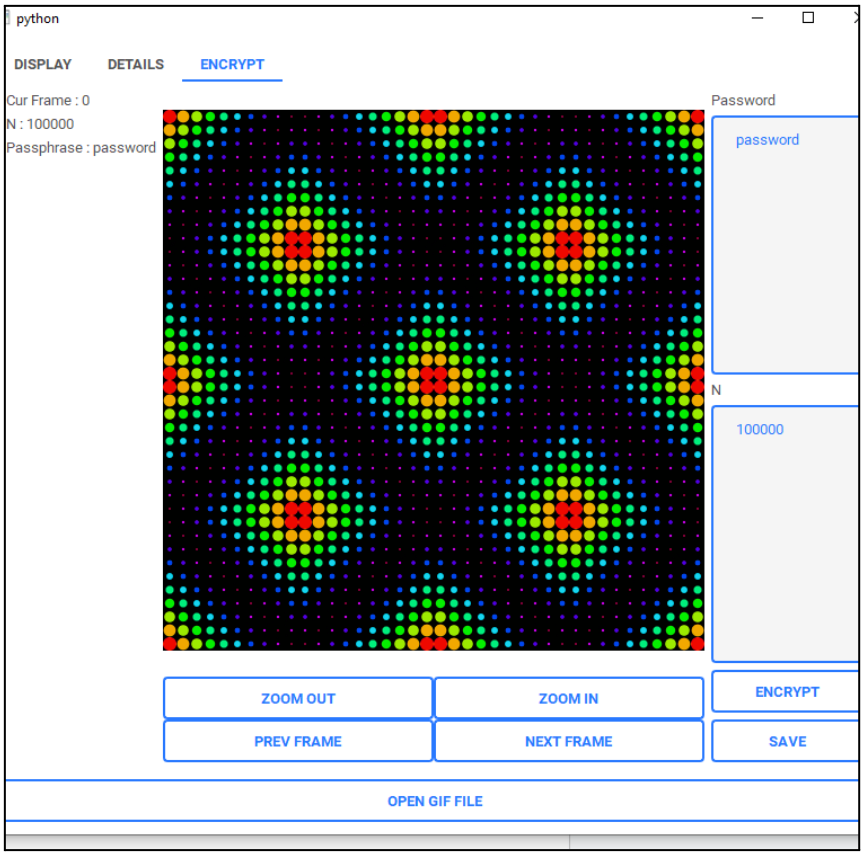


The user now uses the same password and N value, in order to decrypt the encrypted file to restore the original gif file.

Before decryption:



After decryption:

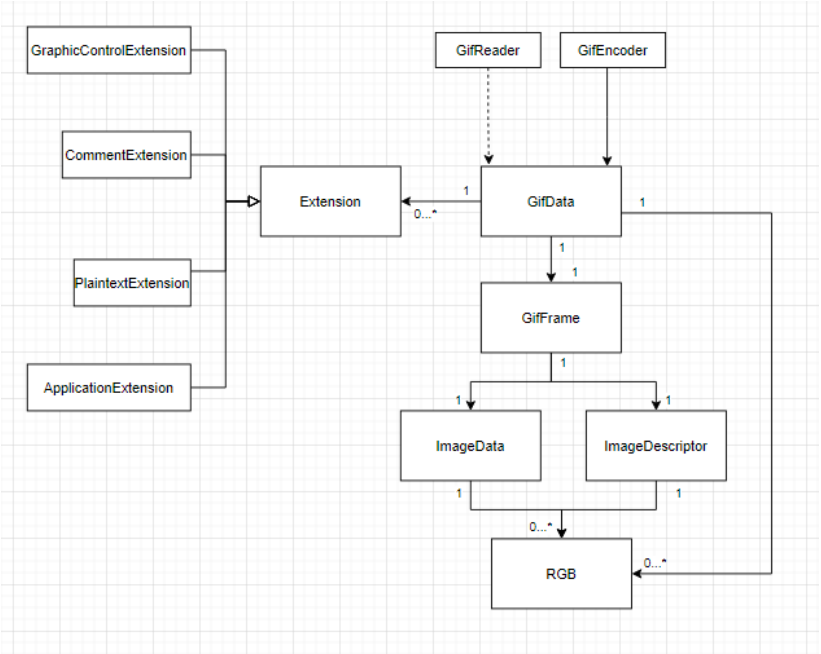


Technical User Guide

GIFData class

The pre-requisite step of performing encryption on a Gif file in order to output a valid encrypted gif file, is to first parse the contents using a Gif file parser first.

The GifData class represents all of the contents of a gif file after the parser parses the Gif file. The class diagram below shows the hierarchical structure of the GifData class.



Library Usage

Note: You can see the code for this section in the `examples/basic_example.py` file.

Reading an Image

```
#!/usr/bin/env python
from gif_me_hd.parse import GifReader

# Download GIF
def download_gif(url, filename):
      import requests
      req = requests.get(url)
      with open(filename, 'wb') as f:
      f.write(req.content)

sample_gif = 'sample_gif.gif'
```

```
download_gif('https://media.tenor.com/eupJDAG479cAAAAC/infinity-trai
n-dance.gif', sample_gif)

# Create the reader object
reader = GifReader(sample_gif)

# Parse the image
gif_image = reader.parse()

# Print out various details about the GIF Image
print(f"Loaded GIF {sample_gif}")
print(f"GIF width: {gif_image.width}\tGIF height:
{gif_image.height}")
print(f"Number of frames: {len(gif_image.frames)}")
print(f"Global Color Table Size: {len(gif_image.gct)}")
print(f"First 10 RGB triplets in Global Color Table:
{gif_image.gct[:10]}")


gf = gif_image.frames[0]

print(f"First frame info")
print(f"Transparent Color Index:
{gf.graphic_control.transparent_color_index}")
print(f"Uses Local Color Table: {gf.img_descriptor.lct_flag}")
print(f"Delay: {gf.graphic_control.delay_time*10} ms")
list_of_indices = gf.frame_img_data
print(f"First 10 indices in image: {gf.frame_img_data[:10]}")
# …
```

Encrypting/Decrypting an Image

Continuing from the code in the previous section, add in the following

```
#…
from gif_me_hd.encrypt import encrypt
from gif_me_hd.encrypt import encrypt as decrypt
n = 100000
print("Encrypting Image...")
encrypted_image = encrypt(gif_image, "password", n)
print(f"Encrypted Image Info")
print(f"First 10 RGB triplets in Global Color Table:
{gif_image.gct[:10]}")
# Should see it is different from the previous color table. The
frame_img_data should also be different now.
```

Writing an Image

```
from gif_me_hd.encode import GifEncoder
encoder = GifEncoder("output_file.gif")
encoder.encode(encrypted_image)
encoder.to_file()
# There should be a file called output_file.gif in your working
directory now.
```

C++ python bindings

The implementation of the LZW compression for the gif frame data in GIF files is extremely slow if written in python. Hence, we have decided to write the most time-consuming part of our program (which is the compression of the GIF data) into the C++ language.

`pip install`-ing the project will automatically compile the C++ bindings, with no extra things to be done by the user.

Class Documentation

Besides providing a Graphical User Interface, this also exposes a library to use for reading/parsing GIF files, encrypting it, and encoding it back to be saved to a GIF file. The main aspect of our project is the encrypt function.

GifReader (gif_me_hd.parse.GifReader)

Attribute	**Description**
__init__(filename)	Takes in the name of the file to parse.
parse()	Returns a GifData object that contains info about the Image

GifData (gif_me_hd.data.GifData)

Attribute	**Description**
bg_color_index	Background color index

color_resolution	Resolution of the Colors in the GIF Image
extensions	List of gif_me_hd.extensions.Extension objects that define the GIF file format extensions
frames	List of gif_me_hd.data.GifFrame objects that are the frames in the GIF file in sequence.
gct	Global Color Table. Is None if GCT is disabled, otherwise, is a list of gif_me_hd.data.RGB objects.
gct_flag	Is True if the global color table is enabled, False otherwise.
gct_size	Size of the Global Colour Table. In the format of 2 ** (1+gct_size)
height	The height of the image
width	The width of the image
pixel_aspect_ratio	For modern devices, this value is unused
sort_flag	True if the global color table is sorted.

GifFrame (gif_me_hd.data.GifFrame)

Attribute	**Description**
frame_img_data	Returns a list of indices. Each index is then used with the color table (which can be the global color table or a local color table, See: graphic_control), to get the colour of the pixel. It is arranged from the top left of the image, scanning to the right, then going one row down.
graphic_control	Returns a GraphicsControlExt object, which gives some object about the frame.
img_descriptor	Returns an ImageDescriptor object, which gives some object about the frame.

GraphicsControlExt (gif_me_hd.extensions.GraphicsControlExt)

Attribute	**Description**
delay_time	The delay (in centiseconds) for the frames.
disposal_method	Not used in modern systems.
hidden	Not part of the GIF file. Frames are associated with a Graphics Control Extension, but the graphics control extension when encoding, the extension should only be written once. So, the hidden

	value is added and is false for the first frame after the extension is introduced, and is true for every subsequent frame using that extension.
size	Size of the extension
transparent_color_flag	True if transparency is enabled
transparent_color_index	If transparency is enabled,the index in the color table that is this value will be the “transparent” color.
user_input_flag	Not used on modern devices.

ImageDescriptor (gif_me_hd.extensions.ImageDescriptor)

Attribute	**Description**
sort_flag	True if the color table is sorted
interlace_flag	Not used on modern devices
lct	Local Color Table. Returns the same type of Global Color Table
lct_flag	Similar to gct_flag
lct_size	Similar to gct_size
left	The x coordinate that it should start drawing on, relative to the GIF’s width/height.
top	The y coordinate that it should start drawing on, relative to the GIF’s width/height.
height	The height of the frame that is to be drawn (it can be less than t he GIF’s height stated in GifData, so only a sub-portion is rendered)
width	The width of the frame that is to be drawn (it can be less than t he GIF’s width stated in GifData, so only a sub-portion is rendered)

GifEncoder gif_me_hd.encode.GifEncoder

Attribute	**Description**
__init__(filename)	Prepares to create an output GIF with the argument filename.

encode(gifdata)	Takes in a GifData object and encodes it to bytes.
to_file()	Outputs to the filename named in the constructor
bytes	Bytes of the encoded GIF file after calling encode(...)

(Function) gif_me_hd.encrypt.encrypt

Arguments	**Description**
gifdata	A GifData object that you wish to encrypt. It will be modified in-place.
password	A string password to use for the encryption. It will be passed to a key-derivation function internally.
n	How much you wish to “encrypt” the image.