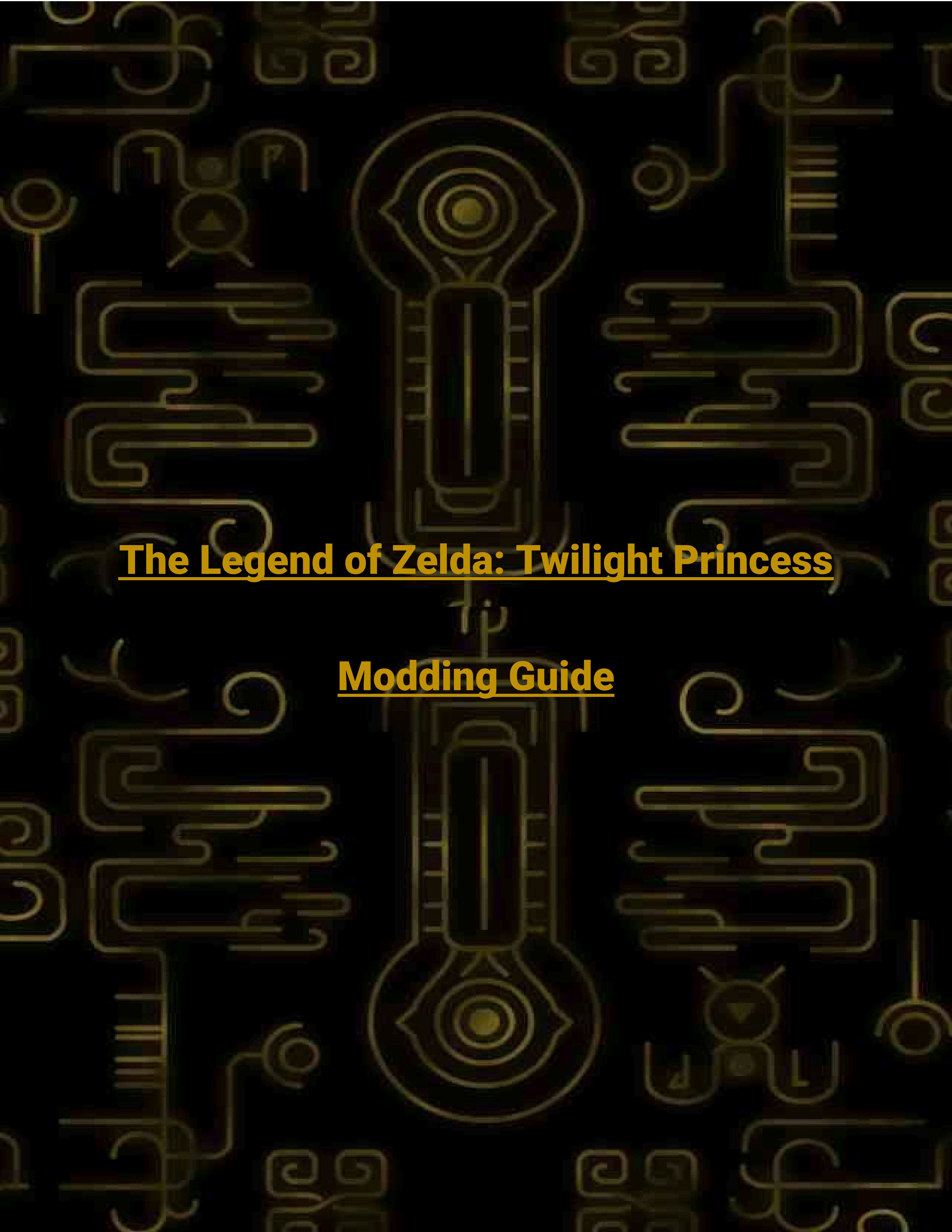

# The Legend of Zelda: Twilight Princess

# Modding Guide

# Table of Contents

# What You'll Need

## Tools

- VS Code
- Latest version of Python
- Git

## Decomp Project Fork

1. You'll need to first have a fork of the Git repository for the Twilight Princess decompilation project.
    - This can be done by opening a terminal and running:
        - **git clone [https://github.com/zeldaret/tp.git](https://github.com/zeldaret/tp.git)**
2. Temporarily place your .iso file of Twilight Princess in the folder called "GZ2E01" (Or the folder corresponding to your iso's region code, GZ2E01 is NA) inside the "orig" folder.
3. Now, in the terminal, run:
    - **python configure.py** [1]
4. Then, run:
    - **ninja**
5. Once ninja has finished running, feel free to move your iso back to wherever you'd like.

## LagoLunatic's GameCube File Tools

Download GCFT from the Git repo below:

- [LagoLunatic/GCFT: GameCube File Tools, a GUI multitool for modifying some common file formats used by GameCube games]()

## LagoLunatic's Rebuild Script

You will also need a python script created by LagoLunatic in order to rebuild the decompiled source code back into a .iso file (to put it incredibly simply).

1. Firstly, in your terminal, run:
    - **pip install "gclib[speedups] @ git+https://github.com/LagoLunatic/gclib.git"**
2. Then, in the "tools" folder of the decomp project that was just cloned, created a new python file called "rebuild-decomp-tp.py".
3. Finally, copy and paste the following code block from the following two pages into the file you just made.

[1] Or **python3 configure.py --non-matching --map**

- The script also works for The Wind Waker!

```
import argparse
import os
from pathlib import Path

if os.name == "nt":
  DEFAULT_DOLPHIN_CONFIG_PATH = Path(os.getenv('APPDATA')) / "Dolphin Emulator"
else:
  DEFAULT_DOLPHIN_CONFIG_PATH =
Path("~/.var/app/org.DolphinEmu.dolphin-emu/data/dolphin-emu").expanduser()

def expanded_path(path_str: str):
  path = Path(path_str)
  path = path.expanduser()
  return path

parser = argparse.ArgumentParser()
parser.add_argument(
  "vanilla_iso_path",
  type=expanded_path,
  help="Path to a vanilla Twilight Princess ISO to use as a base.",
)
parser.add_argument(
  "output_iso_path",
  type=expanded_path,
  help="Path to put the modified ISO.",
)
parser.add_argument(
  "decomp_repo_path",
  type=expanded_path,
  help="Path to the root of the git repository containing the tp decompilation.",
)
parser.add_argument(
  "--map",
  type=expanded_path,
  default=DEFAULT_DOLPHIN_CONFIG_PATH / "Maps",
  help="Folder to place the symbol map for the modified ISO (defaults to Dolphin's maps
directory).",
)

args = parser.parse_args()

decomp_build_path = args.decomp_repo_path / "build/GZ2E01"

import sys
import os.path
sys.path.insert(0, os.path.join(os.path.dirname(__file__), "gclib"))
```

```python
from gclib.gcm import GCM
from gclib.rarc import RARC
from gclib.yaz0_yay0 import Yaz0
from io import BytesIO
import shutil

gcm = GCM(args.vanilla_iso_path)
gcm.read_entire_disc()

rels_arc = RARC(gcm.read_file_data("files/RELS.arc"))
rels_arc.read()

for rel_name in os.listdir(decomp_build_path):
  if os.path.isfile(decomp_build_path / rel_name):
    continue
  if not os.path.isfile(decomp_build_path / rel_name / (rel_name + ".rel")):
    continue
  decomp_rel_path = decomp_build_path / rel_name / (rel_name + ".rel")
  with open(decomp_rel_path, "rb") as f:
    decomp_rel_data = BytesIO(f.read())
    rel_file_entry = rels_arc.get_file_entry(rel_name.lower() + ".rel")
    if rel_file_entry:
      rel_file_entry.data = Yaz0.compress(decomp_rel_data)
    else:
      gcm_rel_file_path = f"files/rel/Final/Release/{rel_name}.rel"
      assert gcm_rel_file_path in gcm.files_by_path, f"Invalid REL path: {gcm_rel_file_path}"
      gcm.changed_files[gcm_rel_file_path] = decomp_rel_data

rels_arc.save_changes()
gcm.changed_files["files/RELS.arc"] = rels_arc.data

with open(decomp_build_path / "framework.dol", "rb") as f:
  gcm.changed_files["sys/main.dol"] = BytesIO(f.read())

with open(decomp_build_path / "framework.elf.MAP", "rb") as f:
  gcm.changed_files["files/map/Final/Release/frameworkF.map"] = BytesIO(f.read())
shutil.copy(decomp_build_path / "framework.elf.MAP", args.map)

for _ in gcm.export_disc_to_iso_with_changed_files(args.output_iso_path): pass

print("Done")
```

# Modifying a REL

## Finding what REL corresponds to your desired mod

Much of the game's logic (such as npc or boss mechanics) is housed within various .rel files.

The decomp project contains those .rel files, but separated into .cpp and .h files that are actively attempting to be reverse engineered. As more code is decompiled, more logic contained within the .rel files is discovered.

And that means more aspects of the game that can be modified!

- The best way that I have found to find what REL you should modify, is to simply search the codebase using keywords in VS Code.
    - For example, if I search "Ganondorf", I will find the .cpp and .h files likely associated with Ganondorf's in-game behavior.

## Making changes

This one is pretty self-explanatory, but more tips and common scenarios could be put here.

- You can pretty much edit the .cpp and .h code however you want, as long as it doesn't cause compilation errors.

## Compiling the source code into a REL

Compiling your modified .cpp and .h files is a somewhat complex process but you can follow along with the steps below for a streamlined method:

1. Open up the file called **configure.py**. It is located in the root of the decomp project repo.
2. Go to line 374 (or wherever the line **Equivalent = config.non_matching** is located).
3. Change the line to:
    - **Equivalent = False**
4. On the line right below the one you just changed, put:
    - **Modded = config.non_matching**
5. A few lines below this, you'll see a line that says:
    - **config.libs = [**
6. The following blocks then essentially contain all the source code in the codebase. You will need to find the file corresponding to the .cpp and .h files you have modified.
    - The names are the same so you can just search the file for the name of the .cpp file you modified.

7. Once you find it, you will replace the status of that file's matching condition.
    - i.e. NonMatching, MatchingFor, or Equivalent
    - Take out out whichever of those variables that is next to the file you have modified, and replace it with:
        - **Modded**
    - ***Note: If the variable is "MatchingFor", take out the game version in parentheses to the right. This does not need to be put back in after putting the Modded variable in its place.***
8. Save the changes.
9. Now, in the terminal, run:
    - **python configure.py --non-matching --map** [2]
10. Then run:
    - **ninja**
11. And then run:
    - **python tools\rebuild-decomp-tp.py "(path to your original game iso)" "(path to wherever you want to build the new iso *including the resultant file name - for instance, rebuild_iso.iso*" "(path to the root of the decomp project repo)"** [3]
12. Once that finishes running, you should have a fully playable iso containing the changes you made, ready to be tested out!

[2] Or **python3 configure.py --non-matching --map**
[3] Or, again, using **python3**