1. What do I have?

After going through this question you will have a list of files that you need to deeply investigate and may also have a diagram of the flow of the piece. You will begin to identify the rough age and state of the piece. This is where you lay the groundwork for the rest of your inquiry and start to gather material that will allow you to distinguish between artistic intent and technical convenience.

1.1 Literally what files do I have?

How do I answer this question?

Use the following bash command to make a file tree.

```
tree -a
```

This will create a new .txt file with content similar to the file tree seen below. Use this as the basis for your report..

1.2 What file types are present?

How do I answer this question?

Make a list in your report of all of the file types, use this to build out your understanding of "relevant" files in the next step.

1.3 Which of these files are important?

How do I answer this question?

Break the files in the tree down into four categories.

- Not Relevant.
- Documentation
- Relevant
- Unsure

Identifying Not Relevant Files

“Not Relevant” files are files that offer no insight into the piece and can be ignored. Mark these files by coloring their entry on the file tree red.

Look through the file tree you generated in the last step. .DS_Store files, bagit files with "bag" or "bagit" in the name ie. bag-info, bagit.txt, transfer related files, and hidden files (files that start with a .) can be marked as 'not relevant'

Here is an example of a file that is just the record of a transfer.

```
Transfer completed: Fri Dec 17 09:59:55 EST 2021
Transfer name: 2014-41-2_important_Application.zip
Target:M:\Conservation\TBMA-DigitalConservation\Collections\2014-41-2_important
\Assets for Dams\2014-41-2_important.zip

Application used: Exactly
User: null
Total File count: 11
Total Bytes: 6528270
```

Identifying Documentation Files

These are files that offer instructions on how to run the code, context or diagrams related to the code. Mark these files by coloring their entry on the file tree orange.

Look for .rtf, .txt, .pdf and .jpg, .png or other image files that have human readable information. If you're able to open a file, read it and it seems like it's giving instructions or perspective on how the piece functions, mark it as documentation.

Identifying Relevant Files

These are files that are directly related to the functionality of the piece. Mark these files by coloring their entry on the file tree green.

If your piece contains HTML

Start with the index.html file. This is the file that will be first loaded by browsers. This file is definitely important. Think of this file as the front door to a home. You need to enter through the front door in order to enter any other rooms in the house. You probably won't be able to know the number of bedrooms the house has when you're standing at the front door. But if you go from the front door to the entry way and walk from room to room you'll eventually have been in every room in the house. And once you've been in every room in the house you'd be able to tell someone else how many bedrooms there were.

So when looking at the index.html file make note of what files it references. If a .jpg file appears in the index file it's an important asset. Make sure to mark it in green when you encounter it. Similarly any html file linked to from the index file is an important file. Not only should you mark it in green but you should also treat it as a new "room" in your theoretical house and investigate it from top to bottom just as you did the index file, marking assets mentioned in green along the way.

It may be helpful for you to create a visual sitemap as you go. You might need to edit it later but recording this initial impression will only help to make your report more detailed. Below is an example of such a sitemap.

If your piece doesn't have HTML

Identify everything that would be able to be run as a standalone piece of code as important. This might require a bit of technical knowledge but there's a few good rules of thumb.

- .app files can be run as standalone files, these are important.
- .exe files can be run as standalone files, these are important.
- A .js file needs HTML in order to be executed in the browser, if it's alone that's unexpected and it should be marked as a question file rather than an important file.
- C, C++ and other C derivatives can be run as standalone files and are important.
- class files need HTML in order to be run in the browser, if it's alone that's unexpected and it should be marked as a question file rather than an important file.
- .SWF files can be run on their own and should be marked as important.

Identifying Unknown or Question Files

These are any files that don't fit neatly into any of the other three categories. Eventually you should be able to explain every one of these files. But for now they're open questions.

Don’t be afraid to revise this list later on in the report. This is simply a starting point.

1.4 What do we know?

How do I answer this question?

This is where you start building out an understanding of what the technical context of your code is. This will help to tease out artistic choice and expression from what might be mere technical convenience.

This means identifying what type of HTML you're dealing with, or tracing where boilerplate java code came from. If you're dealing with a legacy media format why did that media become deprecated? Where was it usually used and for what purpose when it was popular? For example, if you're looking at code designed to interact with a laserdisc player that was written in 1992 then the choice to use laserdisc isn't necessarily artistically significant. At that point laserdisc was a pretty common media format. However if you're looking at code designed to interact with a laserdisc player written in 2022 that is almost surely an intentional artistic choice.

Furthermore if you are dealing with code that touches something like a specific microcontroller or dvd player or laserdisc player find the manual and include that in your documentation. This can help you answer some of the finicky technical questions in part four.

Use these rules of thumb to get some general info on the files you've marked as relevant.

HTML

- **Age markers**
    - Do tags have style markings in them? If so, this is HTML3 or earlier. In tab style declarations, which look like <body color='white'> were done away with in HTML4 in favor of css.
    - Is there a <style> tag? This indicates CSS is present which means this was make after CSS was invented in 1996
    - What is the <DOCTYPE> declaration at the top of the file? This can explicitly date the file
- Does the page link to style sheets or JavaScript? If so you'll want to look at those as well.
- Are there any <meta> tags? These tags are used to indicate key words and help the site be indexed so they give insight into how the developer thought about the site.
- Are there any old or non standard tags? This requires a bit of knowledge about HTML or some googling but it can be incredibly helpful. Uses of tags like <blink> or <marquee> dat eteh piece to a specific time period and the creation of unique tags like <my_personal_tag> would indicate a purposeful artistic choice.

All Code

- Are there any comments? Comments are lines of documentation left by the developer They aren't executed and often appear grayed out in text editors. These are generally helpful and informative.
- Are the variable names informative or obfuscating? Do the variable names tell a story or are they perfunctory things like, x,y,z.

Media Files

- Is the format recent or legacy?
- Is it corrupted or are you able to view/play it?
- Can you run a media conch or similar metadata analysis report on it?
- Is the name informative?

2. What am I missing?

After this question you should know if there exists an artist hosted version or “ground truth” of the piece. You should also be able to tell if you are missing any assets, i.e are there scripts referenced in the code that aren’t present in your file tree. If there are external assets referenced you should have a record of them and a copy if possible.

2.1 Are there any files or assets missing?

How do I answer this question?

Go through the code on a granular level, checking specifically for references to file paths and images. Is there a path mentioned that doesn’t show up on your file tree? Is the piece linking out to something externally hosted? This counts as “missing” since you don’t have control over such elements. If and when the external site goes down that part of the piece will be lost if you don’t make an effort to download it and include it in your report.

2.1 Is there an artist version of the piece that you can still access?

How do I answer this question?

This is your chance to do some archival research, check the way way back machine, use google, ask your peers, use whatever is at your disposal to find out if you can view the artist hosted version of the piece. If you can’t find it then skip this whole section.

If the artist hosted version of the piece is different in any way you need to understand how and why. It could indicate that the version of the piece that you’ve been given is missing important files. Or that the artist continued to make edits after handing over some of the code. Both of these scenarios give important context to an analysis of the piece.

Assuming there is an artist version of the piece you can see.

2.2 What is the same/What is different?

How do I answer this question?

Repeat the first question and all of its steps on the artist version. What files does the piece have in common? Identify any files that seem to be identical and compare their hashes. This can be done by using the md5 command line function on macs and linux.

What does this give me?

If nothing is different than you can be assured that you have all of the code that you need to move forward to question 3. However if you find something that doesn't match then you can dig into why. Did the artist change something after providing code to your institution? Did they give you a different version intentionally? This will require a bit of research but can be very valuable background when discussing artist intent.

3. Does it work?

This question will help you or other technicians when developing a treatment or conservation plan for the piece. Any failure to run the code indicates a potential vulnerability. Even if you are able to run the code without issue it's valuable to have a clear timestamped walkthrough of the piece in case running it becomes impossible in the future.

3.1 Can I run my code without any intervention?

How do I answer this question?

This is going to require a bit of technical know how. By "without intervention" I simply mean can you run the code in the way that such code is meant to be run. Whether you're able to run the code or not record your initial observations of what you can see.

It's valuable at this point to make a screen recording in addition to writing out your observations. If you make a recording be sure to record information on what machine and OS you're using, how you're viewing the code and what the date is. In addition to what you can see, take note of what you can't see. Are there elements of your "relevant" files that you would expect to see but can't? For example if you're looking at HTML with a javascript file attached do you see evidence that that file is being executed?

Even if you can’t run the code at all it’s useful to write down the conditions under which you attempted to view the piece. This can save you and future researchers valuable time when trying to figure out what will or won’t work.

3.2 Can I run my code with an intervention?

How do I answer this question?

If you were unable to run your code in the previous step, now is the time to dig in and see what your options are with emulators. Some formats have free and commonly used emulators associated with them, like Ruffle as an emulator for Flash. Other deprecated formats might require getting creative, find old computers, spin up virtualboxes, whatever it takes to be able to view *something*.

Once you’re able to see the piece run, document exactly what methods you used, how you decided upon that method and why. Make a screen recording as you did in the last step.

4. How does it work?

This is where you really dig into the technical aspects of the code. Use what you’ve learned from the previous steps and the questions you’ve had along the way to guide your breakdown of the piece.

4.1 Is the code human readable?

In order to do a granular analysis of the code you need to be able to read it. If you’re dealing with something like Flash or a .class file you will need to decompile the code first. Be sure to note how you decompiled the code or otherwise gained insight into its functionality.

4.2 What does the piece do?

How do I answer this question?

You want to have a general summary of the functionality of the piece. For example “the html page loads a non-interactive jpg”, or “the flash animation opens a second window when the user clicks the smiley face icon”. For complicated pieces it’s tempting to immediately dive into the code at a variable level, but it’s helpful to start big and work your way down. It keeps you focused on the material outcome of the code and the artist's intent.

4.3 How does the piece accomplish its goals?

How do I answer this question?

This is when you begin to note granular details. For example if the piece uses a complicated case statement you should make a diagram of the resulting decision tree. If case 1 happens what is the material effect, what variables are changed and what in turn changes for the viewer.

You should be able to write a comprehensive explanation of how each part of the piece is executed and what its effect is.

If the piece is written in HTML you would inspect the network tab and console in the browser to see what assets are loaded and how those are used. If the piece is sending or receiving any data you should be able to explain where it's getting information from, what it's doing to that information once it is received and whether the format and or content of that information is artistically relevant. Is it a custom format the artist created? Or is it something standard like JSON?

4.4 What if anything is preventing this piece from executing correctly?

How do I answer this question?

This is a particularly difficult question because it requires that you catch something that might be invisible. If you were able to run the piece without intervention it's tempting to think that everything is operating exactly as it was intended. However there could be subtle errors happening that you're not aware of. The only way to catch things like this is to understand the code at a granular level. Go back to step 2, if there are references to external links make sure that site is actually still online. Understand the potential failure of the emulator to properly display something, for example don't assume that because something appears to be non interactive that it is supposed to be non interactive. Check the code

4.5 What could go wrong with this piece in the future?

How do I answer this question?

You want to gain an understanding of what might affect this piece's health long term. What vulnerabilities are associated with the languages used in the piece? Are any deprecated or in danger of being deprecated? If there are any external assets there present a vulnerability in two ways. First if that site ever goes down, which it inevitably will, then the piece will no longer function as intended. Second if the viewer isn’t connected to the internet then the piece won’t be functioning as intended.