Collin Nelson
Fall 2021
CPTS 315 Final Project

TERRARIAnalyzer

-Project Statement-

TERRARIAnalyzer is a project intending to data mine world saves from the game Terraria for useful and otherwise inaccessible information.

What is Terraria?

Terraria is a tile-based 2D sandbox game released by the company Re-logic in 2011. The game is similar to the iconic game *Minecraft*, though it features fewer survival and crafting mechanics, with a greater emphasis on combat and exploration. Importantly to this project, the game's world is procedurally generated, and consists of over 20 million tiles. In addition, most of the tiles are not visible to players in-game during the normal course of play. This makes any real information about the nature of ore distributions or item locations very difficult to discover. As it is a game that I have devoted hundreds of hours to in my lifetime, I thought it would be fascinating to try and mine some interesting insights about my world from the save files of the game.

Goals & Challenges

With this context, I set the goals of this project as the following.

- Write C# code which can decrypt the proprietary Terraria world save binary files into an object-oriented data structure.
- From this data, find a way to efficiently uncover the following information in roughly this order of importance…
    - The frequency of certain tiles (which include useful tiles like in-game ores and structure blocks) plotted with respect to depth.

    - The locations of in-game chests which contain particular items.
    - Other interesting statistics that are normally inaccessible, like amounts of different liquids in the world map, most common chest loot items, relative prevalence of various tile types in the world, etc.
- Create a user interface that can display this information in a way that is human-readable and intuitive.

While there is perhaps a little more to be gleaned from the data, these were the pieces of information that I found to be the most important to me, and out of the things that can be learned from the world file these were the pieces of information that I deemed to meet the criteria of being both distinctly useful, and distinctly difficult to discover for oneself using in-game or available online resources.

This set of goals implies several significant challenges, foremost among which are the following…

- Terraria's world files are stored in a proprietary binary format. While some resources exist as reference for how to decrypt this data there is no easy out-of-the-box solution for reading this data in C#.
- Large-sized Terraria worlds consist of >20 million tiles. This means that while the dataset doesn't preclude a linear search or similar algorithms, one does have to be careful what operations one performs in a linear time algorithm to avoid expensive operations.
- The resulting data is somewhat abstract, especially for instance a list of chest locations. I need to provide some way of displaying this kind of information, however, that is immediately useful and insightful for a human viewer.

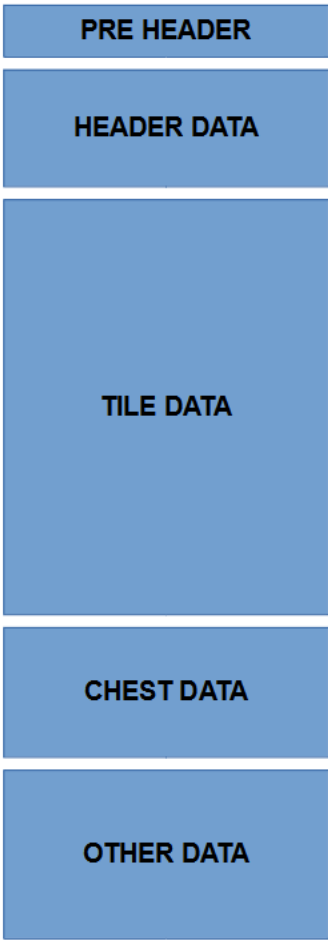
Collin Nelson
Fall 2021
CPTS 315 Final Project

-Methodology-

This project from an algorithmic standpoint roughly breaks down into 2 major parts. The first part is the problem of how one gets data out of the binary .wld file type and puts it into an object-oriented layout. The second part is how one efficiently searches the resulting objects in order to glean useful information. The latter is relatively simple as long as the former is done properly in a way that sets the project up for success. That first problem, however, is also the more difficult one, and is the one that took most of my time to get working on this project.

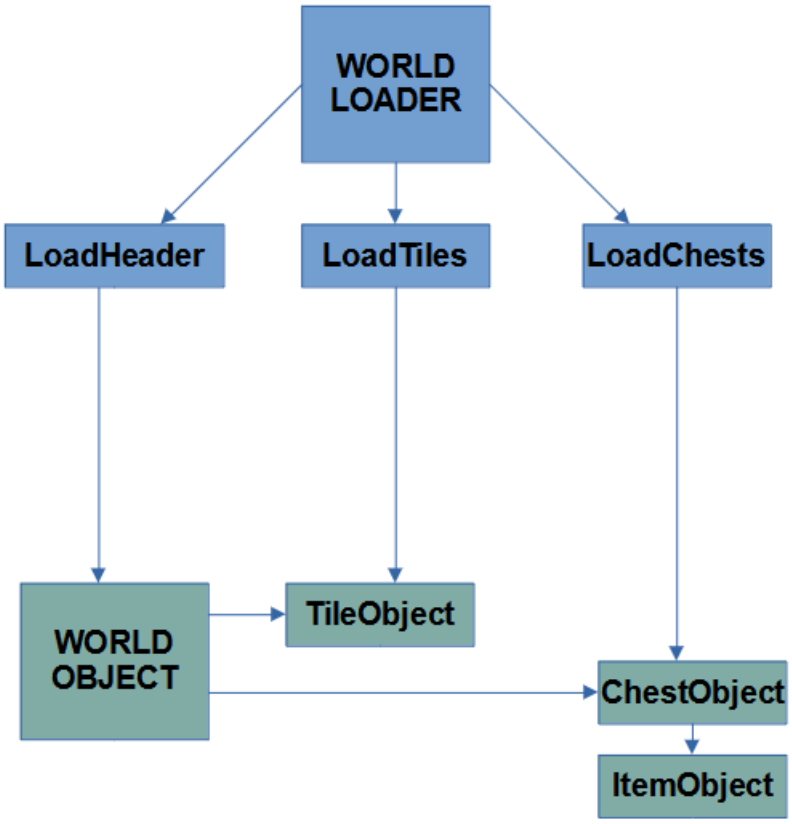
Decoding the. wld

A *simplified* diagram of the data within a Terraria world save file looks like the figure on the right. These data sections more or less have to be read in in the order in which they appear, from top to bottom. In addition, the content of each data section must be read *very* carefully, as in an unrestricted binary file allowing the head of your memory reader to become offset by even a single bit could completely change the values you read for the rest of the file.



The sections which I cared most about were the **TILE DATA,** and the **CHEST DATA** sections, since they contain the most valuable information available in the save file, although I also need access to certain fields found in the **PRE-HEADER** and the **HEADER DATA.**

In addition, the internal structure of each of these sections is quite complex, containing dozens of fields, and (especially in the case of the tile data) they sometimes contain optional fields, or fields of variable length defined by other fields in the save file.



To make this process less painful, I started by building a number of custom binary reader utilities that normalized and restricted my ability to read in data. This ensured safe data reading, and limited opportunities for mistakes in the code to only errors in the order of the fields I'm trying to read.

I then subdivided my code into more manageable sections so that I could further limit the scope of mistakes. Each of these subdivisions maps to a part of populates the data for a part of the class structure I designed to store this data. The above diagram shows the structure of this code.

In the diagram above, squares in blue represent components of the reader code. The objects in green represent the class objects that they read in from the save file to produce the resulting WorldObject data structure.

Reading the data

Luckily, actually pulling useful information from the result of the previous step is far easier than decoding the save file in the first place. The only consideration that has to be considered is that any action in the code that is operating for each tile needs to be relatively inexpensive or the number of tiles will make the code painfully inefficient.

For instance, for retrieving the frequency by depth for a tile I'm able to use straightforward iteration through the tiles array like so…

```
public static int[] CountTilesByDepth(WorldObject world, int id)
{
	//Foreach row in the worldObject
	int[] output = new int[world.WorldHeight];
	for(int i = 0; i < world.WorldHeight; i++)
	{
		//Foreach tile in the column
		for(int j = 0; j < world.WorldWidth; j++)
		{
		    //Increment count
		    if(world.Tiles[i + j * world.WorldHeight].TypeID == id)
		    {
		        output[i]++;
		    }
		}
	}
	return output;
}
```

Note that while I start with a string representing the item's name, this function is designed to use only the ids stored on the objects. This reduces the data footprint of the WorldObject and also makes this iteration much faster. Instead of storing the "user friendly" item name, it is retrieved from an xml manifest only when needed, and the id is used otherwise.

-Evaluation-

Having produced data, I require 2 things to confirm that my model is producing useful and reasonable results. The first is a set of world saves from the game to load, the second is a set of tiles or items with predictable distributions based on simple game knowledge. From these we can compare our outputs to what we would expect the distributions of these tiles to be to ensure that our code is capturing accurate data.

Datasets

For databases I included 3 files:

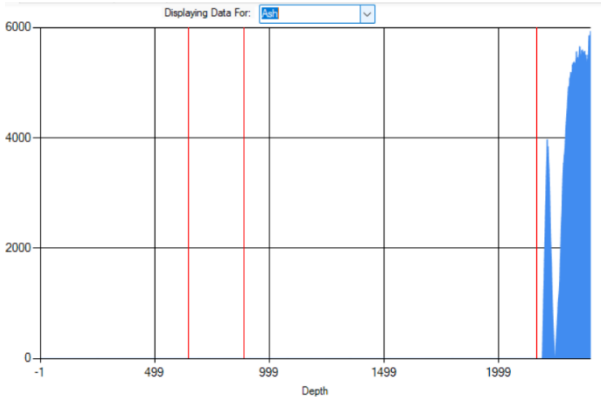
- **The Sundered Land:** A world save that I've played a couple hundred hours in. This world save includes things like user created structures, late-game biomes, and user-placed chests which create interesting behavior in the data.
- **untouched_world_1:** This world was generated but I intentionally never loaded into it. It is completely untouched world generation, and I selected the "Corruption" option which means that somewhere in the world there is a Corruption biome

- **untouched_world_2:** This is the same as the previous file (different random world gen but also untouched), but I selected the “Crimson” option which replaces the Corruption biome with a crimson biome.

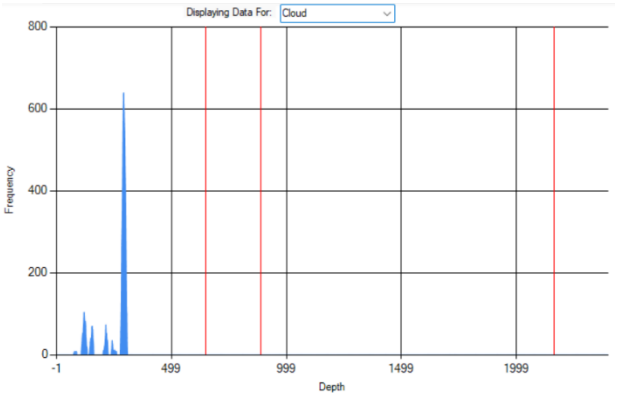
Metrics for success

Each of these datasets allows some slightly unique patterns to emerge, and is distinctly different, but also help to confirm the general patterns common across all Terraria worlds. In order to confirm that the data is accurate I would look at the frequency by depth charts for the following tiles with the following expectations based on in-game world-generation rules.

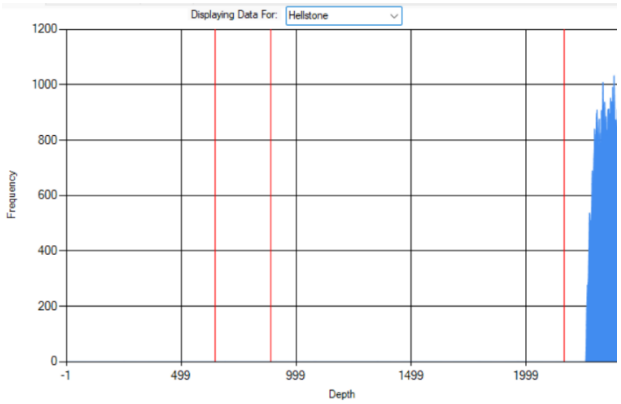
- **Ash:** Only spawns in the “Underworld”. Should have no concentration at any depth above the extreme right of the graph beyond the line indicating the start of the Underworld.



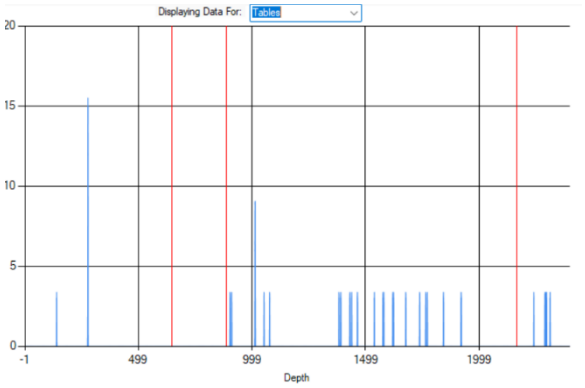
- **Cloud Block:** Only spawns in floating islands. Should have several spikes early in a depth map and then none beyond that.



- **Hellstone:** Checking for the same thing as Ash

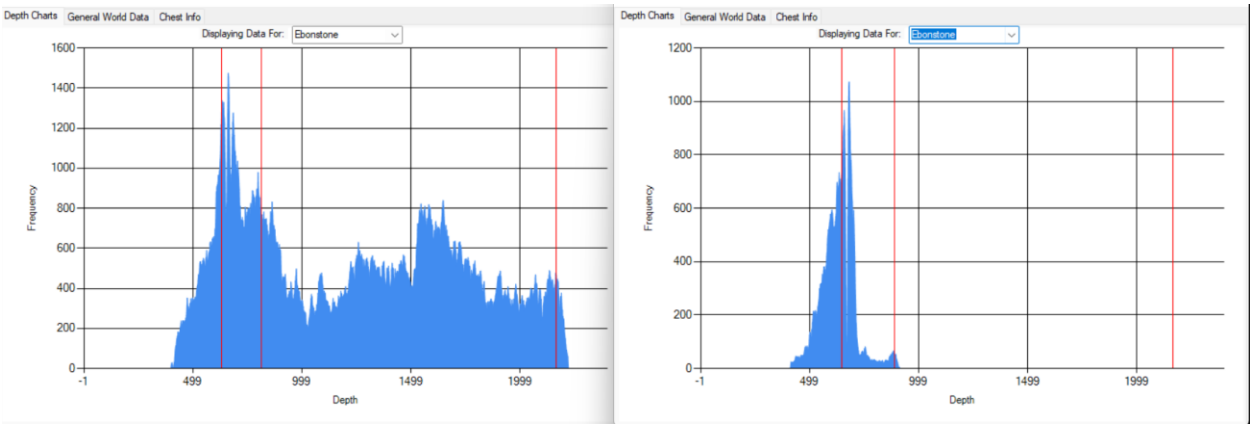


- **Structure Markers:** In a naturally generated world we can use the presence of tables to detect natural structures. Doing so we would expect to see a few structures in the sky (floating islands), a scattering of structures in the caverns layer in the middle of the map and a spike in structures in the Underworld where there are a number of buildings.



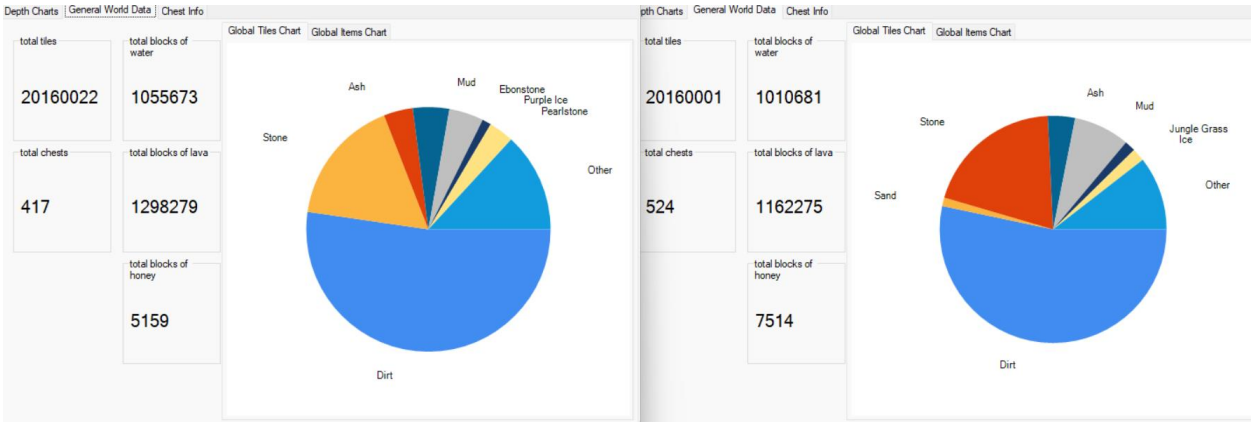
-Conclusion-

Ultimately the project unveiled some really interesting insights about my Terraria world. For instance, one mechanic of the game is that part of the way through the game's progression, the dangerous "Corruption" biome begins spreading. Normally it's difficult to conceptualize this spread, by it becomes stark when you view the untouched graph of corruption blocks next to a graph of corruption blocks in my long-time world.



I also found it interesting to see how many of the world's chests had been looted. By plotting all occurrences of something commonly found in chests, like gold coins you can see the blank patch in the middle of the plot for my long-time world, which is the area I've fully looted.

It was also interesting to compare some of the more general stats, where for instance, the spread of the corruption and a different biome called the hallow makes blocks associated with those biomes one of the most common blocks in the world in my played-in file, but they barely show up in my untouched file.



Note the "pearlstone" and "ebonstone" in the left chart but not in the right.

Overall, this project was incredibly interesting, and I feel that the resulting tool reveals some truly interesting insights about the Terraria world save and the composition of a Terraria world.