Massively Multiplayer Online Role Playing Dungeons and Dragons (MMORPDND)
Manual & Programming Documentation

Developer: Antonius Torode

Version – 1.015
Latest update: July 25, 2024

This document is continuously under development.
Most Current Revision Date: July 25, 2024

Torode, A.
Michigan State University –
Department of Physics & Astronomy.
2017, Student.
Includes Source Code and References
ISBN: NONE

Contents

Introduction

Imagine having a toolkit that makes crafting an entire Dungeons and Dragons world an effortless adventure. The Massively Multiplayer Online Role-Playing Dungeons And Dragons (MMORPDND) system is a collection of tools and scripts that have been designed to simplify the process, aiding in the creation of visually appealing HTML files and automating the linkage of files in a user-friendly wiki-style layout.

At its core, this project serves as a collaborative haven for D&D enthusiasts. Within this dynamic universe, players are invited to claim their own territories, assuming control and adding their creative essence to the ever-expanding lore. Moreover, the existing components are at the disposal of all participants, allowing for a seamless integration of established content into their unique narratives and imaginative designs. In essence, this endeavor mirrors the essence of a massively multiplayer online role-playing experience, with a D&D twist. It's an invitation to embark on an interconnected journey, where each player becomes a co-author of a sprawling and vibrant MMORPDND universe.

1.1 Front-End

For players and enthusiasts, anyone can access the MMORPDND database by visiting the front end website hosted at `MMORPDND.github.io`. The 'Database' button in the navigation bar will take the user to the database of MMORPDND information and pages. This page is not guaranteed to be the most up-to-date version of the MMORPDND database, but should always be close. Once there, the user can simple click on an entry and explore the sub-directories for detailed information about creatures, spells, items, and much more.

1.1.1 Front-End Development

The `MMORPDND.github.io` site is the main front-end website which is hosted in the Github repository at `https://github.com/MMORPDND/MMORPDND.github.io`

1.2 Back-End

The bulk of the database information and main DnD repository is located in a separate repository at `https://github.com/torodean/DnD/`. This repo is used as a sub-module within this main `MMORPDND.github.io` repo which serves as a front end for sharing and users. This documentation mainly focuses on the back-end features for developers.

Installation And Setup

In this chapter, we delve into the essential technical aspects of getting your MMORPDND project up and running. From tool installation to environment configuration, this section provides the necessary groundwork for a seamless initiation into the world of D&D universe creation.

2.1 Windows

TODO - Write this section.

2.2 Linux

2.2.1 Semi-Automated Installation with Setup Script

To expedite the installation and setup process, a convenient setup script has been provided. The provided setup script currently supports Linux environments. This script automates the installation of required dependencies and prepares your environment for the MMORPDND project. Follow these steps to get started quickly:

1. Open a terminal window.
2. Install Git if it's not already present on your system. Run the appropriate command for your operating system:

   ```
   sudo apt-get install git
   ```

3. Clone the MMORPDND repository by running the following command:

   ```
   git clone https://github.com/torodean/DnD.git
   ```

4. Navigate to the cloned repository directory:

   ```
   cd DnD/
   ```

5. Make the setup script executable by running:

   ```
   chmod +x mmorpdnd-setup.sh
   ```

6. Execute the script with administrative privileges to initiate the installation process:

   ```
   sudo ./mmorpdnd-setup.sh
   ```

The script will automatically install the necessary dependencies.

7. Once the script completes, your system will be set up with all required components for the MMORPDND project. If there are any dependencies that did not install via the script, you may need to follow the manual installation guidelines in the below section.

By utilizing Git and the setup script from the cloned repository, you can significantly streamline the installation and setup process, ensuring a swift start to your MMORPDND adventure.

2.2.2 Manual Installation

Before using the MMORPDND project, it's crucial to ensure that your system meets the necessary requirements. The following prerequisites must be satisfied to facilitate a smooth installation and setup process:

- **Git:** Install Git if it's not already present on your system. Run the appropriate command for your operating system:

  ```
  sudo apt-get install git
  ```

- **:** Clone the MMORPDND repository by running the following command:

  ```
  git clone https://github.com/torodean/DnD.git
  ```

- **Python 3:** Ensure that Python 3 is installed on your system. If it's not already installed, you can use the following commands to install it:

  ```
  sudo apt-get update
  sudo apt-get install python3
  ```

- **Python Package Installer (pip):** Install pip, the package installer for Python, using the following command:

  ```
  sudo apt-get install python3-pip
  ```

- **Python Tkinter:** This package is used for creating graphical user interfaces in Python. Install it with:

  ```
  sudo apt-get install python3-tk
  ```

- **Python Libraries:** The MMORPDND project relies on the following Python libraries:
  - `bs4`, `BeautifulSoup`: These libraries are used for web scraping and HTML parsing. Install them with:

    ```
    pip3 install bs4
    pip3 install BeautifulSoup
    ```

  - `regex`: This library provides advanced regular expression capabilities. Install it with:

    ```
    pip3 install regex
    ```

  - `cssbeautifier`: This library is used for formatting CSS files. Install it with:

```
pip3 install cssbeautifier
```

- `tqdm`: This library provides a fast, extensible progress bar. Install it with:

```
pip3 install tqdm
```

- `pytube`, `moviepy`: These are used for managing audio files. Install it with:

```
pip3 install pytube
pip3 install moviepy
```

To ensure a streamlined installation and setup experience, make sure these prerequisites are met before proceeding with the MMORPDND project.

Design

3.1 Main Features

The MMORPDND system at it's core relies on creating a D&D universe. Within a D&D universe are players, characters, stories, regions, maps, items, and more! Therefore, the MMORPDND system must accompany all of these. This project was initially conceived as a personal learning project and personal tool for managing D&D campaigns (as a Dungeon Master tool). Because of this, much of this design was done AFTER many of the tools were already working and in development. This design will therefore correlate to that existing setup and explain some of the choices that were decided upon.

It is my hope that I can maintain good design documentation and outline thoughts in design decisions to be a helpful reference or helpful for anyone who chooses to make additions or improvements. At the time I've started writing this, many of the tools have already been created, but some are still only in their contemplation phase.

3.1.1 Mapping

At the heart of MMORPDND lies very in depth and large maps (initially starting with the Matella region). These maps represents only portions of the starting planet and can therefore easily be expanded upon in terms of new regions, continents, etc. Each map is to be created in sufficiently high resolution so that creators can zoom in and expand upon the finer details of the area needed for a campaign.

For each campaign, a creator can select an area of the larger map, zoom into a smaller region, and develop the towns, cities, or surrounding areas that they will need for their campaign. Having the larger detailed map serves as a guide to keep locations and areas consistent as well as provide options for outside lore that is taking place simultaneously to the players campaign.

within a large world, there are typically many regions, cities, towns, rivers, mountain ranges, etc. Each of these can get lost and forgotten if they are not used. Within the MMORPDND system, all of these areas (once designated and designed) will have an HTML file generated for it. Upon running the main tools of the system, every named place will automatically be linked to for quick reference - providing any already created information to anyone who needs it.

3.1.1.1 Wonderdraft and Dungeondraft

Wonderdraft is a mapmaking tool that has many different features for building larger map areas such as regions, islands, cities, etc. It has a plethora of tools such as custom symbol additions, brushes, paths, etc. The tool creates clean looking maps that can be converted into different styles with the click of a button. One nice feature of the Wonderdraft tool is the ability to zoom into a region and create a more focused map from a larger one. For all these reasons (especially that last one), Wonderdraft was chosen as the intitial map making software for all MMORPDND maps and areas.

Dungeondraft is a close relative to Wonderdraft. It is designed primarily for smaller area's and building layouts. It is handy for making small battle maps and puzzle maps. It also has many unique features and supports custom symbols and objects. This was chosen as the MMORPDND battle map creator for its ease of use and large community containing many resources.

3.1.2 Characters and NPCs

A large part of D&D is having characters and npc's. The MMORPDND system provides a quick and easy way to create nice looking NPC *character sheets* (in the form of HTML pages) for quick reference or use. Many of the necessary parts of creating a character or NPC so that it is ready for gameplay can be automated. These include determining stats, health, bonuses, etc. The MMORPDND system can automate much of this process so that the user can create characters with only a fraction of the work normally needed.

It is often the case that NPCs have set locations, jobs, duties, names, families, or more. It's not always easy to keep track of these things. Within the MMORPDND system, each character will have a generated HTML page with all this information on it. The system will automatically search for an NPC name and automatically link all references to this page for quick reference!

3.1.2.1 Other Game Glements

Outside of maps and characters, there are many other elements to a D&D game that can bring life to the world. These include items, lore, quests, factions, and much more. The MMORPDND system will manage all of these similar to how maps and characters are managed. With each different object gettin gits own HTML file to store information, the system will parse and link all files together appropriately.

3.1.3 mmorpdnd.py

This mmorpdnd.py script was the first tool created in the MMORPDND system. It was full created before any of this design documentation. It performs the bulk of the *behind the scenes* automation such as file linking, formatting, etc. This tool is designed to do all the main features that makes the MMORPDND system complete and connected without any user input. The tool can be run via command line but also has an optional GUI that provides more specific options to run the individual components of the tool for time saving convenience.

Since the MMORPDND system relies on HTML files to store its data, there were a few considerations. since new files would be added constantly (as new data is created or added to the world), a mthod for keeping these files consistent was devised. The mmorpdnd.py script will use a template header and footer, scan for all html files (omitting some folders and files), and update these headers appropriately. This also makes modifying the headers and footers very simple as you only need to modify one file to then propogate the changes throughout the database.

It was decided that having an index file in each directory would be a convenient way of listing all files within that directory (html files, images, folders, etc). This process was also automated so that an index file is automatically created in each folder which has links to all desired sub-files. This makes navigating through the database simple.

Finally, the linking components were added to the mmorpdnd.py script. The idea behind this is to be able to quickly reference anything that is mentioned throughout the documents. The MMORPDND system will gather a compilation of all HTML file names. It then searches all the HTML files for references to those names. Upon finding them, the references are linked to the appropriate HTML files. It has been considered that this system will not function properly if multiple files with the same name are created. It is currently up to the creators to make sure file names are unique enough that this does not occur. As the world and system is still rather small, this is not an issue. A solution to this will likely be added as/if the system expands.

3.1.4 creator.py

The creator.py tool was the second tool in the MMORPDND system to be developed. This tool was also developed before these design documents. This tool is the one to take user-defined data and turn it into HTML files within the database. It was decided that simple text input would be best as input (as anyone will have access and ability to edit these) without any extra knowledge. These text files should be formatted properly for the system to understand the data being entered and properly turn it into the appropriate HTML files.

Items of all sorts were created in a desired format that would appear appropriate and nice. These formats were then broken down into their base and common components. These components were turned into elements the user can define within the data to determine how it will appear (list, item, table, image, audio, etc). These are further elaborated on in the below sections and each has a simple and similar way of inputting data.

The *character sheet* or character page is somewhat unique as it appears different than the other files. The .char file extension was used to generate these character files. This was the first type of input file to be created by the system. The user simply defines the needed parameters for a character and then the system wil fill in the rest and try to adapt appropriately given missing information. After this was completed, the approach turned to a more generic and versatile input type (.input files), which have much mroe flexibility.

These generic input files for the system allow the user to have control over what types of data to input. These generic files will format text in blocks that look nice based on the type the user sets. For objects that have accompanying images (creatures, towns, maps, etc), a system was also devised to automatically detect multiple images. When naming images on linux or windows, the OS typically adds a number at the end of each similarly named file ('(1)', '(2)', ...). This is automatic on Windows and optional in Linux based machines. This format was chosen as it is easy for both operating systems to modify files in this way. When images have multiple files with these numbers, the system will automatically detect the images and display them all.

The creator is setup and thought of as being the primary tool for new data/file creation. Therefore it is expected that new features will be added. It is a GUI application as it requires regular user input.

3.1.5 Stockpile.py

At the time of writing this document, this is currently the first undeveloped script/app that will have its design discussed. This application/script will be responsible for automating the S.T.O.C.K.P.I.L.E.S (see section 3.3.4). Since the players will need access to the S.T.O.C.K.P.I.L.E.S (the list of items in the S.T.O.C.K.P.I.L.E Bazaar), this will have to be public and published somewhere for the players to view.

One option is to use a Google sheets, although this is not as custom/versatile as a self contained solution and relies entirely on a third party system. The benefits to this is that all of the features available to Google Sheets would be available. the downside would be that integration with the Google Sheets would need accomplished in order to automate this.

Another option would be to host a page in the MMORPDND database for the players to access. This option would have the benefit of easy automation. It can be tied into a system that regularly pushes updates through Github actions. One downside is that only the features of HTML or that programmed into the system would be available. Another downside to this would be that the players would have a direct link to the MMORPDND database (and thus potentially DM notes). This could be somewhat remedied by removing the usual heading which links back to the other pages and simply being a self contained static page (although it would not be hard to go back a directory in the url).

The main functionality of the S.T.O.C.K.P.I.L.E.S is the ever changing nature of the available items. There are two main types of items that are contained within the S.T.O.C.K.P.I.L.E.S. The first is a collection of items (collection A) that are typically found in most places and widely available. This would be D&D items such as simple weapons, food, general supplies, etc. The second is a collection of items (collection B) that are typically trade goods brought from outside areas. This would be a collection of rare artifacts, unique items, specialized trade goods, etc. These items would primarily depend on the traders coming and going and these items would have limited availabilities. Although both of these lists should technically differ from area to area based on natural resources, locations, and other things, it would not be a far stretch of the imagination to assume that the general supplies for most areas are similar and then the trade items coming and going could also be similar. For this reason, one collection of items can be used for all areas when using the S.T.O.C.K.P.I.L.E.S. These two collections are also distinct, exclusive, and therefore may be best as entirely separate lists.

Collection A will contain items that are typically available. Every now and then you can find an item out of stock based on random chance. The items in this list are typically well traded and common items that are traded often. For this reason, their prices are somewhat well established and only vary slightly. In contrast, collection B will contain items that are traded and have worth determined by whomever is willing to buy or barter for it. These items may have largely varying prices based on the merchants sources and perception of worth. These items will vary in price drastically and have a significantly more random stock status.

For both collections A and B, the lists can be stored as a text file. Each line will represent an item. Each item should have a name, base price, and description. When the S.T.O.C.K.P.I.L.E.S are updated, A small percentage of items from collection A will be deemed out of stock (say 5-10%). The remaining items will be in stock. Of the items in stock, the prices will be randomly selected based on a bell curve around the base price (some slightly more than usual, some slightly less). A variance of 80-120% can be used. For collection B, we will start with a number of items in stock (say 10-20% the size of collection A's list). After each update, some items will disappear from the list, and new items will be added. This can be based on the cadence of the refresh and the amount of time that passes in game (values to be adjusted later). The variance of price for these items can vary much greater and range from 60-150%. To simplify sell prices, they can be made a simple flat rate of whatever the current sale price is (say 75-80%). Upon each update of the S.T.O.C.K.P.I.L.E.S, the current list can be compared to the master lists (collection A and B) and the items can be updated accordingly. The stockpile.py should do all the appropriate work and randomization.

A configuration file can be used to determine the desired values for whatever cadence is chosen. For example, if the list is being updated once a month, the changes should be more drastic than if the list is updated daily. The configuration file can contain basic settings such as cadence, the sell percentages, price variations, standard deviations, and more. This will be configurable to allow the user to experiment and test for desired amounts.

To allow for testing for experimenting with the desired values, upon each update, the prices of the values can be stored/saved and plots can be automatically generated to show price trends over time. It can be decided if the players would have access to this information, but it would primarily be used for testing and deciding on the system parameters to use for whatever setup the DM is using.

3.1.6 char_maker.py

The character maker (char_maker.py) is an app/script that has plans for development but is still in the design phase. The purpose of the character maker is to utilize AI to quickly generate character files for the MMORPDND database. The output files will be the .char files used by the creator.py application.

3.2 Coding Choices

Although many programming languages could have been selected for this project, Python was chosen for a few reasons. The primary reason was the familiarity and opportunity to further learn Python that was presented to the original creator. Another was it's simple to use and modify nature. With Python scripts, end users can open the code, make changes, and run them seemlessly without having to worry about proper setups and build systems (simple just having the correct python environment installed). Python provides easy to use string and file manipulation which was very relavent for the plans in this project.

3.3 Gameplay Features

Since this system is ultimately for designing a D&D universe, there are various new ideas relating to the gameplay itself that have also been developed alongside it. These include different styles of play, different ways to interact with characters, different managing techniques for Dungeon Masters, and more. I will attempt to outline some of these here but they may progress faster than this documentation does.

Personally, my largest hiccup when it comes to D&D campaigns is finding people who have the time commitment. I do not necessarily mean once every week or even once every month, but rather willingness to meet one some regular cadence for an extended period of time. My personal DM style is to have a ton of depth and a slow overall story progression (not to be confused with a lack of action, adventure, or plot). I tend to plan for a campaign that will never end and let it play out as it does. The story will tend to change when players are not able to show up, but this is not always easy depending on where they left off. Unfortunately, scheduling is a nightmare and occasionally you can never find time to meet. Even though the players say they enjoy things well, there's always a player who drops out due to it taking too much time (even if you've only met 4 times in a year), which inevitably leaves to others dropping out too. To remedy this, I created what is known as the O.R.B.I.T System (outlined below), where players can come and go and not have to meet for every session. I have done something similar to this in the past and it worked out rather well, but this is a more robust version.

3.3.1 O.R.B.I.T System

One-shots, Risky Business, and Intriguing Tasks (O.R.B.I.T) is a system meticulously crafted to enhance the Dungeon Master's ability to run engaging campaigns in a flexible and dynamic manner. Tailored for Dungeons and Dragons (or other RPG) sessions, O.R.B.I.T. revolves around the idea of encapsulating complete adventures within a single session while seamlessly weaving an overarching narrative of covert operations, daring risks, and mysterious tasks.;Designed with the ever-changing availability of players in mind, O.R.B.I.T. allows for a revolving cast of characters, making it ideal for those seeking episodic gameplay with diverse and flexible party compositions. The acronym captures the essence of the campaign system: One-shots represent the self-contained missions or adventures. Risky Business involves the diverse array of financial and downtime activities and Intriguing Tasks form the backbone of an unfolding narrative that keeps players hooked and invested. In the game world, this system is embodied by the company Operations, Risky Business, and Intriguing Tasks, commonly known as Orbit. Orbit is a clandestine organization operating within Matella, masquerading as a reputable consultancy and security firm in the public eye. The city, rich with factions, secrets, and evolving events, serves as a dynamic backdrop to the adventures orchestrated by Orbit.

Whether players are pursuing high-stakes heists, unraveling political mysteries, or delving into ancient ruins, Orbit provides a modular and adaptable framework. Orbit Adventures shines in its flexibility, accommodating different playstyles and player schedules. The incorporation of downtime activities, risk management, and intriguing narratives creates a rich and immersive gaming experience. The game's structure empowers both Dungeon Masters and players to collaboratively shape a campaign that unfolds seamlessly, with each session contributing to the overarching story.

Between high-stakes missions, Orbit operatives engage in usualy life as well as some focused downtime activities to enhance their skills, manage their finances, and expand their inventory.

- Training for Advancement: The players can dedicate downtime to training and honing skills, facilitating character advancement. Training directly impacts the ability to level up, gaining new abilities, and improving existing ones. The Orbit system relies on the E.N.G.A.G.E system as the primary means of character enhancements for the players.

- Financial Endeavors: The players (throught usual life activities) engage in various financial endeavors during downtime, including working regular jobs, making wise investments, or even taking calculated risks such as gambling. The outcomes directly affect the operatives' financial standing, influencing their resources for future missions. The Orbit system relies on the P.R.O.F.I.T system as the primary means of financial management for the players.

- Inventory Management: The players can allocate time to inventory management, exploring markets, forging alliances, and discovering unique opportunities to acquire new items. This includes purchasing or crafting equipment and magical items, ensuring operatives are well-prepared for upcoming challenges. The Orbit system relies on the S.T.O.C.K.P.I.L.E Bazaar as the primary means of commerce interactions for the players.

Each operative's choices during downtime activities have a tangible impact on their character sheet. Whether it's gaining new skills, accumulating wealth, or expanding their arsenal, downtime is a crucial period where operatives invest in their personal growth and readiness for the next operation. Choose wisely, for the city's shadows hold both risks and rewards, and every decision shapes the trajectory of an operative's journey in Orbit Adventures.

3.3.2 E.N.G.A.G.E. System

The Earning New Gains through Active Growth Endeavors (E.N.G.A.G.E.) System is designed to infuse character progression with dynamic and active elements, allowing players to shape their characters through earned gains achieved via active growth endeavors that the characters live during game downtime. In this system, characters accumulate 'Gains' through successful sessions, missions, creative role-playing, and personal accomplishments. At the end of each session, players are rewarded with Gains, reflective of their active engagement and achievements. These points can be strategically allocated across various categories, such as leveling up, acquiring new skills, enhancing attributes, or unlocking special abilities.

The core philosophy of the system is the name of the system "Earning New Gains through Active Growth Endeavors," emphasizing that character development is not solely a passive process but a reflection of the character's active engagement and contributions to the unfolding narrative. Whether through mastering new skills, leveling up, or unlocking unique abilities, players have the agency to actively shape the trajectory of their characters' growth. Additionally, the E.N.G.A.G.E. System encourages players to participate actively in downtime activities, training, and quests, creating a synergistic relationship between in-game actions and character advancement. This dynamic approach ensures that characters evolve organically, reflecting the choices, actions, and endeavors of the players as they navigate the rich and immersive world of the campaign.

3.3.3 P.R.O.F.I.T System

In the Passive Returns and Opportunities for Financial Investment and Treasure (P.R.O.F.I.T) System (or simply P.R.O.F.I.T.S for short), characters have the opportunity to engage in various financial endeavors, each tier representing a different level of risk. This system is designed to be straightforward, providing characters with options to pursue low, medium, or high-risk financial activities during campaign downtime.

- Low Risk: Characters opt for stable employment, taking on a regular job in the city or performing services for a reliable employer. The type of employment can be anything but should be something fitting for the individual character. Characters receive a steady income, providing a reliable but moderate financial growth. Minimal loss potential, as the stability of the job ensures a consistent income, but the potential for significant wealth accumulation is limited.
- Medium Risk: Characters decide to invest their funds in businesses, properties, or ventures that offer potential returns over time. Medium to high potential returns, with profits increasing as the investment matures. There's a chance of losing a portion or the entirety of the invested capital if the venture faces challenges or fails.
- High Risk: Characters engage in high-stakes games of chance, placing bets or participating in gambling activities. High potential returns, with the possibility of substantial wealth gained through luck or skill. Significant loss potential, as characters risk losing their entire wager or investment in the unpredictable world of gambling.

At the end of each downtime period, characters declare their chosen financial activity (Low Risk, Medium Risk, or High Risk). The Dungeon Master determines the outcome based on the chosen tier, taking into account the associated risks and potential gains or losses. For simplicity, you can only pick one risk category per downtime period. A dice roll or a predetermined probability can be used to simulate the unpredictability of financial activities, especially in medium and high-risk scenarios. Characters can accumulate wealth over time, and their financial decisions may influence their lifestyle, ability to purchase items, or engage in other significant activities. The Financial Growth System aims to provide a balance between simplicity and engagement, offering characters diverse opportunities to navigate the financial landscape in the campaign. The specifics of this system are further detailed in the database files and will likely adapt and evolve over time.

3.3.4 S.T.O.C.K.P.I.L.E System

The S.T.O.C.K.P.I.L.E System (called S.T.O.C.K.P.I.L.E.S or Stockpile system for short) represent all of the market areas within the region that the players have access to during a period of time. Within the bustling market district, the characters will encounter a dynamic array of stores collectively referred to as the Special Treasures, Ordinances, Collectibles, Knickknacks, Paraphernalia, Items, Limited Ephemerals (S.T.O.C.K.P.I.L.E.) Bazaar (or sometimes simply called S.T.O.C.K.P.I.L.E.S or Stockpiles). It's important to note that this term isn't assigned to a single establishment but is rather a convenient label used by the Dungeon Master.

To simplify and enrich your shopping experience, the Dungeon Master will manage a master list or spreadsheet encompassing all the items scattered across the diverse stores within the Stockpile Bazaar. This central document acts as a comprehensive resource, allowing you to effortlessly browse through the available Special Treasures, Ordinances, Collectibles, Knickknacks, Paraphernalia, Items, and Limited Ephemerals. During downtime, your characters are free to explore the various stores, each boasting its own unique offerings. The master list serves as an organized overview of all items accessible in the expansive market district. Prices are subject to fluctuation as stocks rise and fall, and items can seamlessly appear or disappear from the list based on availability.

It's crucial for players to manage their inventory and money diligently. Each item in the Stockpile Bazaar will be accompanied by both a purchase price and a sell price. Players must keep track of their current money, as it serves as the sole limit to what they can purchase. As you navigate this vibrant market district, your choices in acquiring and selling items will directly impact your character's resources and potential for future acquisitions. Embrace the excitement of discovery and bartering in the S.T.O.C.K.P.I.L.E. Bazaar, where each visit promises a unique journey through a world of wonders, and your astute financial decisions shape the course of your character's adventure.

Tools And Scripts

4.1 mmorpdnd.py

4.1.1 Purpose

This script serves as the backbone for an array of automatic linking and logistical features within MMOR-PDND. These functionalities typically operate on all files, encompassing crucial elements such as the generation of directories, HTML index files at the directory summits, establishment of uniform headers and navigation for all HTML documents, seamless interlinking between files, refinement of both HTML and CSS code, and an array of additional capabilities. When ran, this script will perform the following actions in the following order:

1. Create Directory Structures if not already existing.
2. Create index files for each directory.
3. Update each index file to link to all files and images within the directory.
4. Update the headers of all HTML files to the template HTML header.
5. Update the navigation of all HTML files to the template navigation HTML.
6. Search for invalid hyperlinks in HTML files and removes them.
7. Search and hyperlink all words found in all HTML files to the appropriate HTML file whose name matches the words found.
8. Beautify the code.
9. Publicize the HTML files specified in the templates/lists/public_files.list file.

4.1.2 Features

4.1.2.1 Create And Update index.html Files

TODO

4.1.2.2 Update All HTML Navigation And Headers

TODO

4.1.2.3 Update All HTML Hyperlinks

TODO

4.1.2.4 Beautify Files

This feature will take all css and HTML files and ‘beautify’ them. This is done by the BeautifulSoup Python library and will format all the files in a similar manner. Because this feature exists, the formatting of the HTML files when editing them is not needed, as it will be overwridden.

4.1.2.5 Publicize Files

In some cases, links to a page may be desired for the players to access. In these cases, it is desired to remove the navigation header on the HTML pages as well as links to other pages within the MMORPDND database. This feature will facilitate doing just this. It is a simple feature to create pages for 'public' (meaning links that the players in a campaign can access without linking them to the rest of the database) release. The script takes an input file called 'templates/lists/public_files.list'. The script will create duplicate files of all files listed in the 'public_files.list' with an appended '_public' label on the file names. It will then parse all the public files and remove all links within the files that are not links to other public files. If a public file exists for one of the links, the link will instead be updated to the appropriate public file.

The publicize files feature was originally it's own standalone script that was later integrated into the 'mmorpdnd.py' script. This feature uses the 'templates/lists/public_files.list' file as input. This file should have relative file paths to all files that are wished to be made public (relative to the 'mmorpdnd.py' script. The script will begin by creating a copy of all files (specified in the list file) with an appended '_public' attached to the file name. These public files will have all header navigation data removed, and all links removed. In the case that a public file exists for what was previously linked to, the link will be updated to the public file rather than removed.

4.1.3 Use

The mmorpdnd.py script can be ran as a gui window or using the command line options. The two main features are 'test' and 'update'. The 'test' feature will create fake files and then perform the update command on them. This is useful for testing if the features and newly added features are working properly. The 'update' feature will perform a combination of tasks which updates all appropriate files to do the steps mentioned in the list above.

4.1.3.1 GUI

The GUI window is somewhat self explanatory. The two main features are represented by red buttons - the 'test' and 'update' features. There are also other buttons to simple perform and of the individual tasks using the appropriately associated button.

4.1.3.2 Command Line

The MMORPDND application supports some command line features. The update feature will run the updates for all HTML files.

```
usage: mmorpdnd.py [-h] [-t] [-u] [-r]

MMORPDND Tools and apps.

options:
  -h, --help     show this help message and exit
  -t, --test     Runs the test-all feature then exits.
  -u, --update   Runs the update_all feature then exits.
  -r, --remove   Runs the remove_broken_links feature then exits.
```

4.2 templates/creator.py

4.2.1 Purpose

The creator tool is a practical solution for converting basic template text files into functional HTML pages. It takes care of the technical aspects by automatically creating HTML files and filling in missing details, like character stats or other content. The tool understands the structure of your template files, recognizes placeholders, and replaces them with accurate data. Whether you're building character profiles, story summaries, or any content with consistent formatting, this tool ensures your HTML documents are correctly formatted and ready for use. It simplifies the process, letting you focus on content creation while it handles the conversion from templates to HTML.

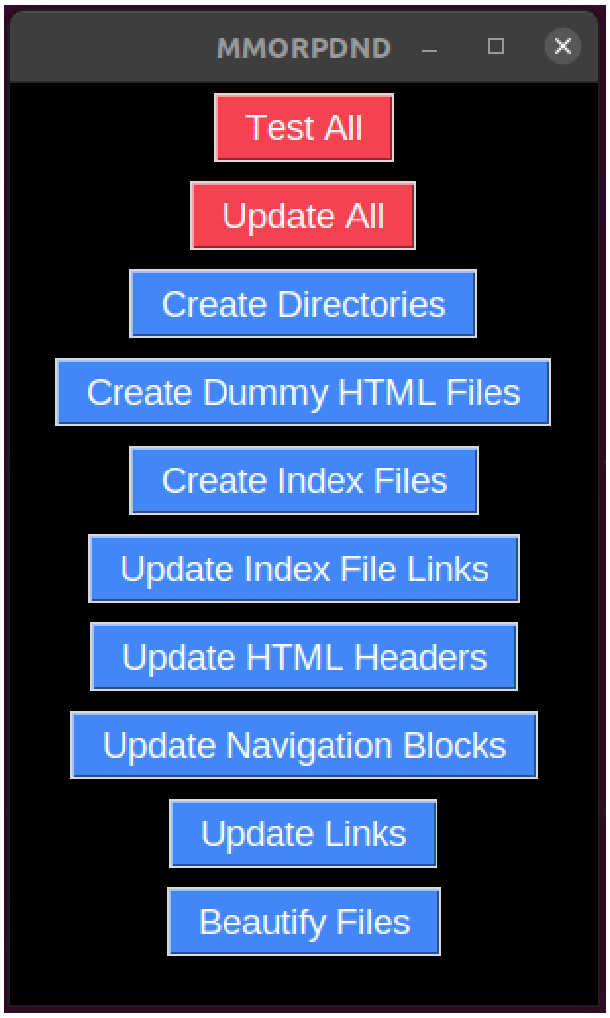


Figure 4.1: The MMORPDND Main GUI

4.2.2 Use

The creator tool is a powerful tool that contains many subtle features. It is important to understand these subtleties before using the tool and therefore I encourage the reader to read this enture section before use.

4.2.2.1 GUI

The main GUI of the creator tool has a few components. These include text boxes, buttons, and check boxes.

The “Input File” text box field is used to specify the input file that is to be processed by the tool. This file can be an ‘.input’, ‘.char’ or ‘.names’ file. Ensure the file is in the proper format (see below sections) or unexpected behavior may occur. The ‘Browse’ button is used to open a file explorer window for selecting a file and will automatically populate the “Input File” text field when a file is selected. The large text box in the center of the GUI is where output generated from the tool as well as status messages are displayed.

The “Generate Word” button is used for random word generation. This feature takes a list of words (whether names, places, or other) and generates a similar random word using the data in the input ‘.names’ file. This is explained in more detail later in section 4.2.2.14. The “Create Page” button is used to generate an HTML file based on a ‘.input’ or ‘.char’ file and is explained in more details in the following sections. the “Random Places” file is used to generate place names based on an input ‘.names’ file along with a ‘.list’ file. These are explained mroe in later sections. The “Yes”, “No”, and “Reset” buttons are used when user input is asked for. The “Update All” button performs the same functions as the mmorpdnd.py update function (see the previous sections).

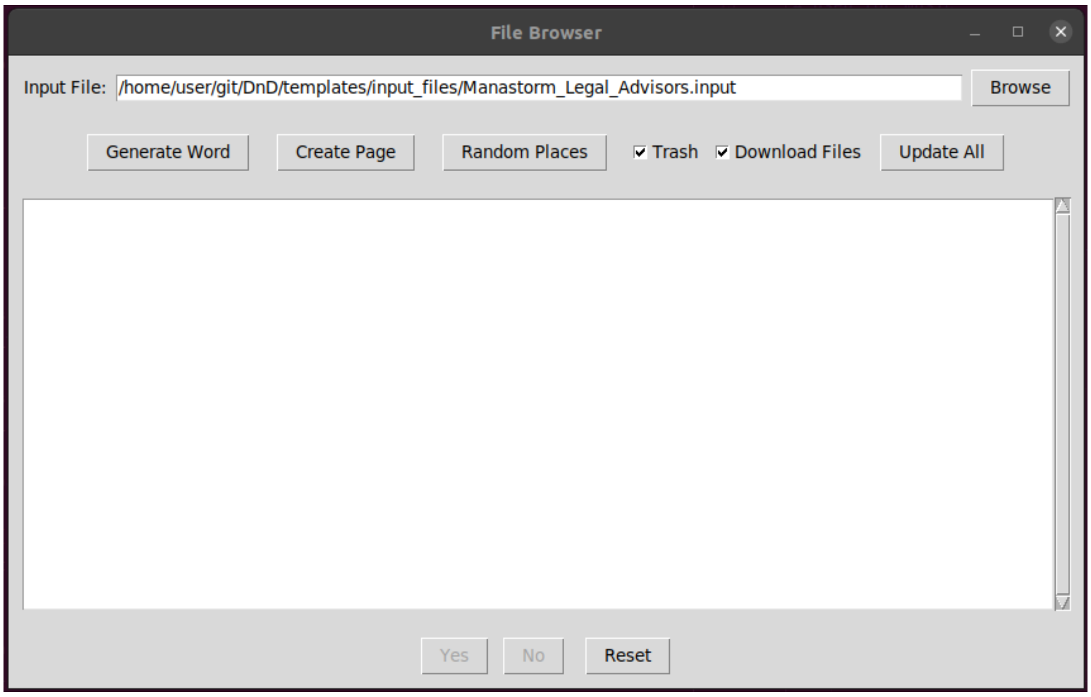


Figure 4.2: The MMORPDND Main GUI

The “Trash” checkbox will determine if processed files will be moved to the trash folder after processing and if duplicate files are to be deleted. Typically, the user will want this to be checked, but it is useful to leave unchecked for testing and if files will be processed multiple times. The “Download Files” button determines if music files should be downloaded when ‘.input’ files containing a dnd-music section are processed (this is explained mroe in later sections).

4.2.2.2 Input Files (.input)

The input files are essentially just text files with the file extension ‘.input’. The main feature of the creator.py tool is to read in these input files, parse them, and generate the appropriate HTML files from them. These input files must follow a strict formatting but offer a few various helpful features. The name and main heading of the generated HTML file will be created based on the name of the input file (underscores will be replaced by spaces for the heading). Each line of the input file represents a ’section’ of the HTML file that will be generated. Each line follows the following format:

```
Heading Text[type]=Information text
```

The ”Heading Text” is what will appear as the header for that HTML section. The ”type” value is what determines the properties of this section and how it is interpreted by the creator tool. The ”Information text” is the actual information for that section. The only exception to this rule is the first line which defines the ”folder” that the HTML file generated from the input file will be placed into. This ”folder” line is special in that it has no type and the creator tool will recognize the folder keyword and store this information separately. This folder line must be formatted as follows:

```
folder=path/to/where/I/want/my/generated/file
```

The various types determine the formatting and decoding of the input file, and each type has small subtleties. The various types are

1. dnd-image: This type is used to display an image or multiple images.
2. dnd-list: This is used to display a list of items.
3. dnd-info: This is used for general information and sections. It also supports listing information between paragraphs.
4. dnd-music: This is a section for storing various music for the regions.

4.2.2.3 dnd-image type

The **dnd-image** type is used to display images specific to the file. The information text must contain three parameters separated by a semi-colon delimeter. These parameters are **img/image-name.jpg;image source;image caption**. The first parameter is simply the location of the image and the image name. These are typically placed in 'templates/img'. The second is the source of the image, this is simply a string but is there to keep all images properly sourced. The third is a caption that will be displayed with an image. This caption is also a string and can be whatever the user desires. As an example, here's a proper dnd-image block (see figure 4.3):

```
Coconatus Marmotta[dnd-image]=img/coconatus_marmotta.jpg;Created by Bing AI image
    creator;Coconatus Marmotta.
```

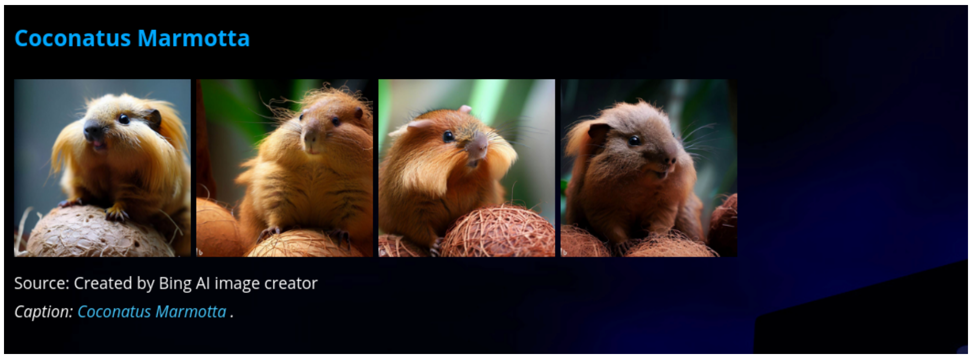

Coconatus Marmotta
Source: Created by Bing AI image creator
Caption: Coconatus Marmotta .

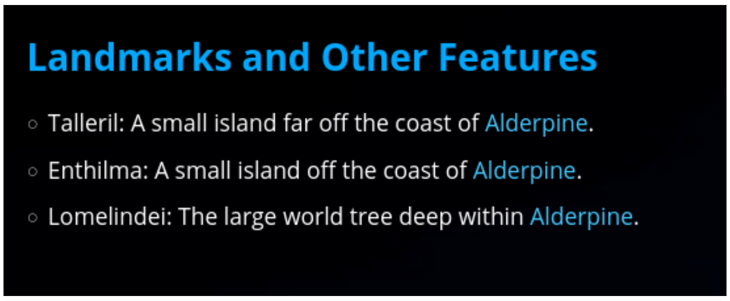
Figure 4.3: A dnd-image section generated using 4 images.

One unique feature of this is that if the image name does not exist (i.e. 'img/coconatus_marmotta.jpg'), the tool will append a ' (1)' to the name and search again (i.e. 'img/coconatus_marmotta (1).jpg'). If this new name is found, the tool will search for image names with the index incremented and include all such images (i.e. 'img/coconatus_marmotta (2).jpg', 'img/coconatus_marmotta (3).jpg', etc) in the output HTML block.

4.2.2.4 dnd-list type

The **dnd-list** type is used to display a simple list of items. The information text must contain one or more parameters separated by a semi-colon delimeter. As an example, here's a proper dnd-list block (see figure 4.4):

```
Landmarks and Other Features[dnd-list]=Talleril: A small island far off the coast of
    Alderpine.;Enthilma: A small island off the coast of Alderpine.;Lomelindei: The
    large world tree deep within Alderpine.
```



Landmarks and Other Features

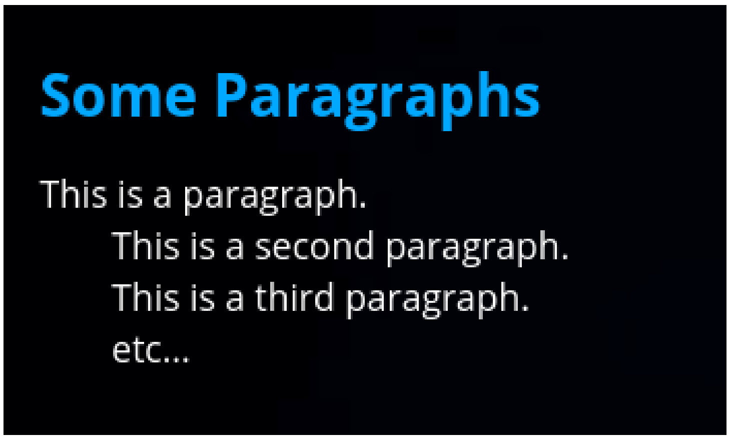
- Talleril: A small island far off the coast of Alderpine.
- Enthilma: A small island off the coast of Alderpine.
- Lomelindei: The large world tree deep within Alderpine.

Figure 4.4: A dnd-list section.

4.2.2.5 dnd-info type

The **dnd-info** type is used to display a section of text. The information text must contain one or more parameters separated by a delimeter. The delimeter for this type will define the format of the sections within this text. For writing normal paragraphs, simple use a semi-colon (';') delimeter. The semi-colon delimeter will make the next parameter appear as a normal paragraph. As an example, here's a proper dnd-info block (see figure ??):

```
Some Paragraphs[dnd-info]=This is a paragraph.;This is a second paragraph.;This is a
    third paragraph.;etc...
```


Some Paragraphs

This is a paragraph.

This is a second paragraph.

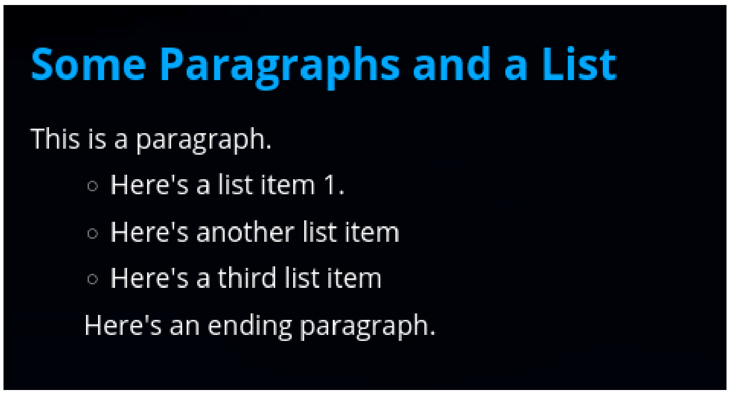
This is a third paragraph.

etc...

Figure 4.5: A dnd-info section with only paragraphs. Note that the first paragraph will not be indented but all subsequent ones will be.

To add in a list element between paragraphs, you can use a semi-colon and dash (';-') delimeter. The ';-' delimeter will tell the creator tool that a list has began and the it will use all subsequential parameters with a preceeding ';-' delimeter as the list items until a different delimeter is entered or there is no more parameters to read in. As an example, here's a proper dnd-info block (see figure ??):

```
Some Paragraphs and a List[dnd-info]=This is a paragraph.;-Here's a list item 1.;-Here's
     another list item;-Here's a third list item;Here's an ending paragraph.
```


Some Paragraphs and a List

This is a paragraph.

- Here's a list item 1.
- Here's another list item
- Here's a third list item

Here's an ending paragraph.

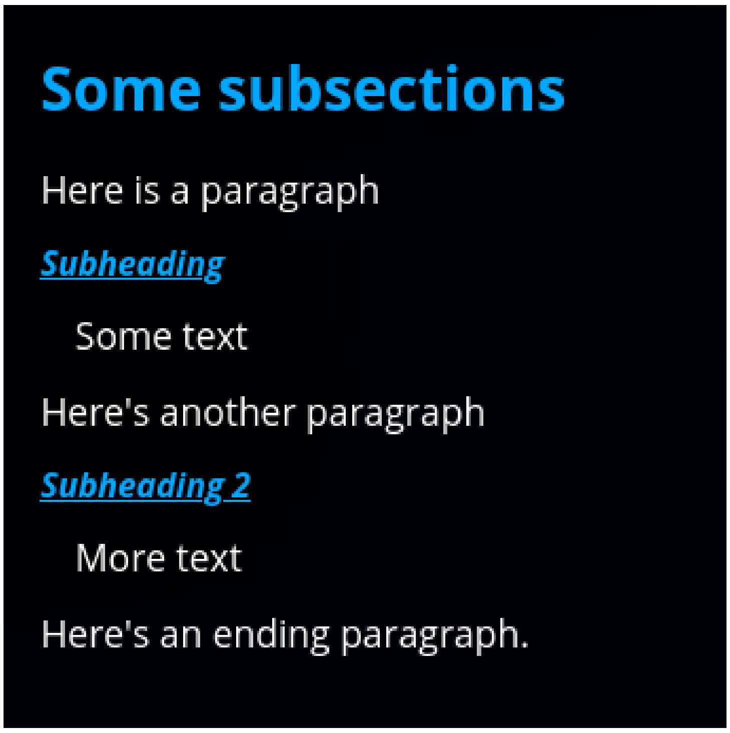
Figure 4.6: A dnd-info section with a paragraph followed by a list, followed by a paragraph.

To add a subsection, you can use the semi-colon followed by an asterik (';*') as the delimeter. This delimeter has a somewhat special formatting in that there needs to be a second asterik to mark the end of the subsection heading. For example, ';*heading text*normal content text'. The 'heading text' will display as a subheading, while the 'normal content text' will be the main content under that subsection. As an example, here's a proper dnd-info block (see figure ??):

```
Some subsections[dnd-info]=Here is a paragraph;*Subheading*Some text;Here's another
    paragraph;*Subheading 2*More text;Here's an ending paragraph.
```

4.2.2.6 dnd-music type

The **dnd-music** type is used to display a simple list of music items. The information text must contain one or more parameters separated by a semi-colon delimeter. Currently, this is primarily designed to work with youtube links. Given a simple youtube link, the code will find the name of the video and



Some subsections

Here is a paragraph

Subheading

Some text

Here's another paragraph

Subheading 2

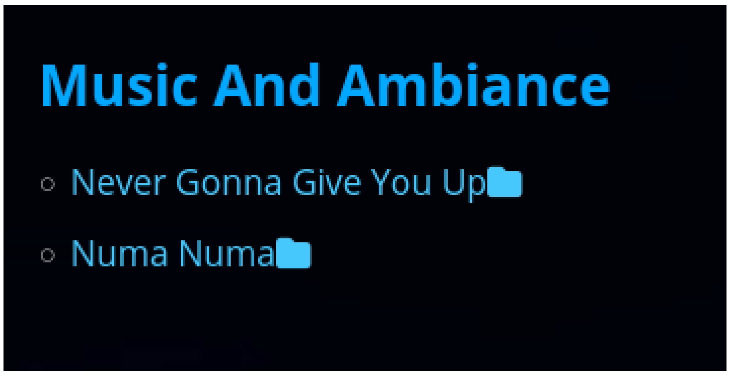
More text

Here's an ending paragraph.

Figure 4.7: A dnd-info section with some paragraphs and two subsections.

use that as the list information as well as keeping a link to the video itself. If the "Download Links" checkbox in the GUI is enabled, the files will be downloaded as mp3 files and the folder next to the list item will be hyperlinked to the local file. These files will be located in a music folder. As an example, here's a proper dnd-music block using fake URL's (see figure 4.8):

```
Music And Ambiance[dnd-music]=https://youtube_link_1;https://youtube_link_2;https://
    youtube_link_3
```



Music And Ambiance

- Never Gonna Give You Up
- Numa Numa

Figure 4.8: A dnd-music section with two songs linked to.

4.2.2.7 dnd-table type

The **dnd-table** type is used to display a simple table. The information text must contain one or more parameters separated by a semi-colon delimeter. The first parameter determines how many elements there are in each row. The rest of the parameters will then be parsed and placed into the table from left to right (top to bottom). As an example, here's a proper dnd-table block which creates a 3x2 table:

```
Attributes[dnd-table]=3;Strength;10;+0;Dexterity;12;+1
```

Figure 4.9: A dnd-table section with 3 columns and 2 rows.

4.2.2.8 Example Input File

For example input files, you can look inside the templates/trash folder. This is where input files get placed after they are finished processing. The following is an example input file with an example of many different elements. The associated image files 'foobar.jpg', 'foobars (1).jpg' and 'foobars (2).jpg' should be located in the templates/img folder for the below file to process correctly.

```
folder=items

A Single Image[dnd-image]=img/foobar.jpg;Created by Someone;A foobar image.

Information Section[dnd-info]=Here is a paragraph of text.;Here is a second paragraph.

Information Section With Smaller Headers[dnd-info]=Here is a paragraph of text.;*Small
    Header* Here is a second paragraph.

A List Of Items[dnd-list]=item A; Item B; Item C; Item D

Multiple Images[dnd-image]=img/foobars.jpg;Created by Someone;Multiple foobar images.

Information Section with List[dnd-info]=Here is a paragraph of text followed by a list;-
    List_item_A;-List_item_B;-List_item_C;Followed by a paragraph.

Music And Ambiance[dnd-music]=https://youtube_link_1;https://youtube_link_2;https://
    youtube_link_3
```

The above file will create a file in the 'items' folder found in 'campaign/items'. The file will start with a header "A Single Image", where the foobar.jpg image is displayed. A caption of "A foobar image." will be under the image. Next will be a heading "Information Section" That has two paragraphs. Next a similar section with a header "Information Section With Smaller Headers", a paragraph, then a smaller header "Small Header", followed by another paragraph. Next, another heading "A List Of Items" with 4 items in the list. Following this, a "Multiple Images" heading with 2 images 'foobars (1).jpg' and 'foobars (2).jpg'[1] with a caption "Multiple foobar images." Next, another heading "Information Section with List", with a paragraph, a list, and then another paragraph. Finally, a heading "Music And Ambiance" with links to three youtube videos. If the "download links" checkbox is enabled, these video's will be downloaded as mp3 files and the fodler icon will link to the local files.

4.2.2.9 Character Files (.char)

The character files are text files with the ".char" extension. These are a special type of input file that generates an HTML page specific for DnD character information. This was actually designed before the more general input file formatting outlined in the sections above and therefore lacks some of the versatility and extra features that can be included in the input files. Nevertheless, the character format creates an HTML page that should more than suffice for any npc characters and has many of its own unique features and capabilities. The .char file contains lines that define the various elements of a character. These lines to not currently support the types that are supported with the input files. Each line has a specific output format in the generated HTML file based on a template and the information itself. Some lines in the file are required while others are optional. The required lines must be present for the character file generated to be completed, while the optional lines will automatically be randomized and generated in the event that they are missing. Eahc line of the character file must be formatting in the following way:

```
Keyword = Value
```

The "Keyword" is used by the creator tool to determine which type of information is being input, and the "Value" is the value of the information. For example "level = 3" would let the creator know that the character level is 3, and it will use that information appropriately.

Once the character file is appropriately filled out and created, it should be saved/placed in the templates/input_files folder for processing by the creator tool just like the .input files. The creator tool

[1] If an image is not found, it will automatically search for files with that image name with an appended (1), (2), (3),... If '(1)' is found, this image will be used and the number incrememted until no other images are found.

will search the templates/img folder for any image references within the character files. This is different than the behavior of the input files (which rely on the user to specify the 'img' folder in the image path).

4.2.2.10 Required Keywords

The required keywords for a character file are the following:

1. folder: This defines the folder that the output HTML character file will be placed. This should be a valid file path to a folder.
2. name: This defines the name of the character.
3. image: This is an image for the character. This should be just the image file name and extension (no path) and the image itself should be placed in the "templates/img" folder for automatic detection.
4. class: This defines the DnD class. This must be a valid class from the following list:
   - barbarian
   - bard
   - cleric
   - druid
   - fighter
   - monk
   - paladin
   - ranger
   - rogue
   - sorcerer
   - warlock
   - wizard
5. abilities: The abilities of the character. This field is a list and should be entered using a comma (',') as the delimeter.
6. equipment: The equipment of the character. This field is a list and should be entered using a comma (',') as the delimeter.
7. proficiencies: These are the proficiencies of the character. This field is a list and should be entered using a comma (',') as the delimeter. The proficiency bonus will automatically be calculated based on level and applied to proficiencies.
8. information: General information and description of the character.

The above list of keywords are all required and if they are missing or ill-formatted, unexpected behavior can occur.

4.2.2.11 Optional Keywords

The optional keywords for a character file are the following:

1. level: This is the level of the character. If the level is not defined, it will default to 1. This must be an integer.
2. hp: This will be generated automatically if not included based on the character class, level, and constitution[2] value.
3. ac: The Armor class value.
4. size: The size of the character.
5. type: The type of the character (creature, humanoid, animal, etc)
6. alignment: The alignment of the character.
7. speed: The speed of the character.
8. resistances: The resistances the character has (if any).
9. immunities: The immunities the character has (if any)
10. senses: The senses. A ", Passive Perception: #" will automatically be appended to this value based on a calculated passive perception.

[2]The constitution value is automatically generated based on the character class and level if not included.

11. languages: Languages of the character.
12. race: The race of the character.
13. background: The background of the character.
14. strength: This represents the character's physical strength and ability to exert force. This must be an integer.
15. dexterity: This represents the character's agility, reflexes, and fine motor skills. This must be an integer.
16. constitution: This measures the character's health, stamina, and resistance to illness and fatigue. This must be an integer.
17. intelligence: This reflects the character's mental acuity, memory, and problem-solving skills. This must be an integer.
18. wisdom: This represents the character's perception, intuition, and common sense. This must be an integer.
19. charisma: This measures the character's charm, persuasion, and ability to influence others. This must be an integer.
20. notes: Notes to add about the character that aren't contained in the 'information' section.

It is recommended to include all values in the character file and just use "None" for any string values that are not needed. This will prevent any un-expected bugs. Many of the values replace a string in a template file based on the values input, so if they are not specified, a correlated 'search string' will appear in the final HTML file that serves as a placeholder for the information.

4.2.2.12 Example Character File

For example character files, you can look inside the templates/trash/chars folder. This is where input files get placed after they are finished processing. The following is an example character file. The associated image file foobar.jpg should be located in the templates/img folder for the below file to process correctly.

```
folder = characters/player
name = fooBar
ac = 16 (natural armor)
hp = 84
level = 13
size = small
type = humanoid
alignment = neutral good
speed = 30 ft.
resistances = None
immunities = None
senses = Darkvision 60ft
languages = Common, Elvish
image = foobar.jpg
race = Fooling
class = wizard
background = Barber
strength = 15
dexterity = 13
constitution = 18
intelligence = 22
wisdom = 4
charisma = 14
abilities = basic attack: A basic attack with nothing special; prestidigitation
equipment = torch; small sword, a potato
proficiencies = strength, acrobatics, history
information = This is some info for foobar, the template character.
notes = No notes needed for template dude.
```

Once created, the file should be placed in the templates/input_files folder and then processed via the creator.py tool.

4.2.2.13 Optional Appended Input Types

The character files can have optional input file type (specified in section 4.2.2.2) blocks specified after the normal character file information. This will generate sections identical to that of the input files at the bottom of the character file page. See the example below:

```
... # other character stats here
proficiencies = strength, acrobatics, history
information = This is some info for foobar, the template character.
notes = No notes needed for template dude.

A new section[dnd-info]=Some more information.

Some more images[dnd-image]=img/images.jpg;images;More images of this char.
```

4.2.2.14 Name Files (.names)

The '.name' files are essentially lists of names to be used. The file extension '.names' is simply to differentiate between lists used as other things and lists that are specifically for name generation. These files should each have a common theme (i.e. a file for human names). Within the files, there should be one name per line. The "Generate Word" button is used for random word generation. This feature takes a list of words (whether names, places, or other) in the form of a names file and generates a similar random word using the data in the input '.names' file. After generating a word, it will also ask the user if they would like to append that word to the list for future generation/use.

The algorithm used to generate these names is quite straight forward and outlined in detail below. At first, AI name generation was thought of as an option, but this tended to give very cliche, repetitive, and similar sounding names - it wasn't very creative. The algorithm added here generates entirely new words that follow the patterns and structure of the words in the input lists. The more words that are contained in the names list, the better the name generation will be. With small lists, the name generation may not be that good.

The algorithm used for name generation begins by reading in all words in the names file. The words are sorted into a probability matrix that determines the probability of which letter should follow another based on all the occurrences of those letters within the words of the list. For example, the word 'hello' would generate the below probability matrix.

```
{
    "e": {
        "l": 1.0
    },
    "h": {
        "e": 1.0
    },
    "l": {
        "l": 0.5,
        "o": 0.5
    },
    "o": {}
}
```

Once generated, a small number of corrections and adjustments are made, and then this probability matrix is used to construct a new word. The length is randomized based on the smallest and largest words within the names file.

4.2.2.15 List Files (.list)

The '.list' files are simple lists of information. These are similar to the '.names' files, with the difference that these list files are typically used internally within the various tools.

4.2.2.16 Command Line

The creator application supports limited command line features. The file option will run the creator for a single input file and then update all files.

```
$ ./creator.py -h
usage: creator.py [-h] [-f FILE]

```

```
MMORPDND Creator Tool.

options:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Run the creator for a single input file and update all files.
```

4.3 templates/lists/check_link_validity.py

This script will simply check the validity of the *links.list* file.

4.4 templates/lists/organize_random_places.py

This script will sort the location names in the *random_place.names* file based on the last word in each item. The output will be saved to *random_place_sorted.names* so that the original file will not be disturbed.

4.5 templates/lists/remove_duplicates.py

This is a simple command line tool that will search through a file and remove any duplicate entries. This is useful when dealing with large name files or list files.

```
usage: remove_duplicates.py [-h] [-f FILE] [-a]

Remove duplicate lines from files

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Path to the file
  -a, --all             Run against all files in the current folder
```

4.6 templates/img/fix_image_extensions.py

This tool will simply search through all files in the templates/img folder and replace all '.jpeg' extensions with '.jpg'. This is both for consistency and functionality of the other apps.

4.7 templates/languages/translator.py

This tool is still in development and not yet functioning.

4.8 templates/reset_all_files.py

The 'reset_all_files.py' script provides functions for restoring all files to a state where they are prepared to be processed by the 'creator.py' application. It performs three main operations: copying image files to the 'tempaltes/img' folder, deleting HTML files in the campaign directory structure, and moving specific input files back to the 'templates/input_files' folder. The script includes the following functions:

1. **'copy_images(source_dir, dest_dir)':** This function copies image files from a specified source directory to a destination directory. It identifies images based on common file extensions such as '.png', '.jpg', '.jpeg', and '.gif'. The function traverses the source directory and its subdirectories, specifically looking for folders named 'img', and copies the detected image files to the destination directory.

2. **'delete_html_files(directory)':** This function deletes all HTML files within a specified directory and its subdirectories. It traverses the directory tree, identifies files with a '.html' extension, and removes them. This is useful for cleaning up directories from unwanted HTML files.

3. **‘move_input_files(source_dir, dest_dir)‘:** This function moves files with ‘.input‘ or ‘.char‘ extensions from a source directory to a destination directory. It ensures that the destination directory exists, creating it if necessary. The function traverses the source directory and moves the matching files to the destination directory.

   The script is configured with paths relative to its location:

- The `campaign_dir` points to the ‘../campaign‘ directory relative to the script directory.
- The `dest_img_dir` is set to the ‘img‘ directory within the script’s directory for storing copied images.
- The `trash_dir` is designated as the ‘trash‘ directory within the script’s directory.
- The `input_files_dir` is set to the ‘input_files‘ directory within the script’s directory for moving input files.

  The script performs the following operations when executed:

1. Copies image files from the ‘campaign‘ and ‘trash‘ directories to the ‘img‘ directory.
2. Deletes all HTML files from the ‘campaign‘ directory.
3. Moves ‘.input‘ and ‘.char‘ files from the ‘trash‘ directory to the ‘input_files‘ directory.

Ensure that the directories specified (e.g., ‘campaign‘, ‘img‘, ‘trash‘, ‘input_files‘) exist and are accessible. The script creates directories as needed but assumes the necessary permissions are available.

4.9 templates/stockpile/

This directory contains the various elements of the Stockpile system (see 3.3.4).

4.9.1 stockpile.py

The stockpile.py script is the main application controlling the Stockpiles. The stockpile script takes an optional config file where the various parameters for stockpile updating and generation can be modified (see /refstockpile config).

```
usage: stockpile.py [-h] [-c CONFIG] -g GENERAL -t TRADE [-b BUY] [-s SELL] -o OUTPUT [-
    i]

S.T.O.C.K.P.I.L.E System Interface and Database Updater.

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        The config file containing key-value pairs representing
    configuration parameters, each line formatted as 'variable=value.'
  -g GENERAL, --general GENERAL
                        The file containing general store items.
  -t TRADE, --trade TRADE
                        The file containing random and trade items.
  -b BUY, --buy BUY     The file containing the buy price history of the various items.
    Including this option will update this.
  -s SELL, --sell SELL  The file containing the sell price history of the various items.
    Including this option will update this.
  -o OUTPUT, --output OUTPUT
                        The output file to compare to and update. This should be a .input
    file with the lists formatted in the dnd-table format (see documentation).
  -i, --initial         Generates an initial .input file for processing.
```

4.9.1.1 Item Lists

The item lists are the master list that contains all of the general and specialty items for the Stockpiles. These lists are fed into the stockpile app with the '-g' and '-t' options. The lists should be formatted such that each line represents a single item. Each item will contain 3 parts separated by a semicolon delimiter - A name, a price, and a description (see below for example file).

```
Abacus;2g;2 lb.
Acid (vial);25g;1 lb.
Alchemist fire (flask);50g;1 lb.
Arrows (20);1g;1 lb.
Blowgun Needles (50);1g;1 lb.
Crossbow bolts (20);1g;1.5 lb.
Sling bullets (20);4c;1.5 lb.
Antitoxin (vial);50g;
Backpack;2g;5 lb.
```

The distinction between these two lists is in the types of items they contain. The general list should be easily obtainable and typically available items for an area. In contrast, the Trade/Specialty items are rare items, or things that may or may not be available - primarily carried by traders to a region.

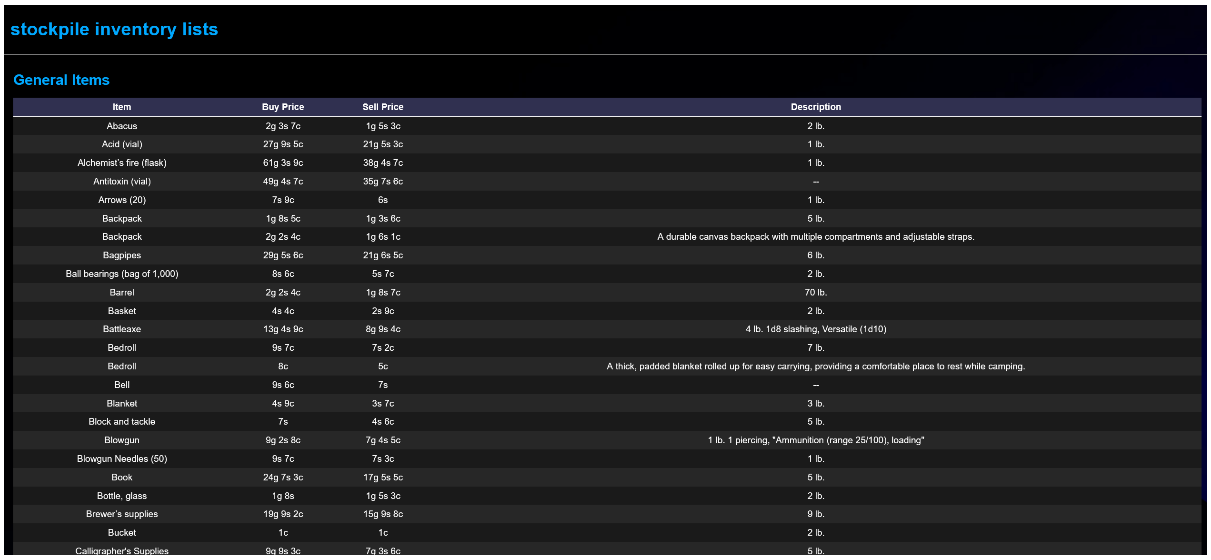
4.9.1.2 Buy and Sell Price History Lists

The buy and sell price history files can start blank. The application will create these by adding items into the list that are not already found. The files are formatted so that each line represents a single item. The lines will contain a number of elements separated by a semi-colon delimiter. The first element is the item name. All later elements are the various prices of the items as they have changed (thus capturing the price history).

```
Abacus;1.9;1.95;2.03;2.16;2.08;2.07;2.13;1.99;1.96;2.03;2.06;2.17;2.27;2.31
Acid (vial);27.32;26.57;28.75;28.61;28.42;28.7;28.01
Alchemist fire (flask);50.52;52.08;54.07;55.43;57.35
Alchemist supplies;48.03;46.5;45.58;47.4;45.4;44.22;42.28;40.12;40.93
Antitoxin (vial);55.09;54.65;57.01;58.46;61.02;64.23;68.13;68.74;66.47
Arrows (20);1.01;1.02;0.97;1.01;0.98
```

4.9.1.3 Output list

The output of the stockpile program will be a '.input' file that matches the correct format to display all items in a list. The file will have two sections, one for general items and one for specialty/trade items. See Figure 4.10 for the example html file that is produced from an input file from the stockpile output.


stockpile inventory lists

General Items

Item	Buy Price	Sell Price	Description
Abacus	2g 3s 7c	1g 5s 3c	2 lb.
Acid (vial)	27g 9s 5c	21g 5s 3c	1 lb.
Alchemist's fire (flask)	61g 3s 9c	38g 4s 7c	1 lb.
Antitoxin (vial)	49g 4s 7c	35g 7s 6c	--
Arrows (20)	7s 9c	6s	1 lb.
Backpack	1g 8s 5c	1g 3s 6c	5 lb.
Backpack	2g 2s 4c	1g 6s 1c	A durable canvas backpack with multiple compartments and adjustable straps.
Bagpipes	29g 5s 6c	21g 6s 5c	6 lb.
Ball bearings (bag of 1,000)	8s 6c	5s 7c	2 lb.
Barrel	2g 2s 4c	1g 8s 7c	70 lb.
Basket	4s 4c	2s 9c	2 lb.
Battleaxe	13g 4s 9c	8g 9s 4c	4 lb. 1d8 slashing, Versatile (1d10)
Bedroll	9s 7c	7s 2c	7 lb.
Bedroll	8c	5c	A thick, padded blanket rolled up for easy carrying, providing a comfortable place to rest while camping.
Bell	9s 6c	7s	--
Blanket	4s 9c	3s 7c	3 lb.
Block and tackle	7s	4s 6c	5 lb.
Blowgun	9g 2s 8c	7g 4s 5c	1 lb. 1 piercing, "Ammunition (range 25/100), loading"
Blowgun Needles (50)	9s 7c	7s 3c	1 lb.
Book	24g 7s 3c	17g 5s 5c	5 lb.
Bottle, glass	1g 8s	1g 5s 3c	2 lb.
Brewer's supplies	19g 9s 2c	15g 9s 8c	9 lb.
Bucket	1c	1c	2 lb.
Calligrapher's Supplies	9g 9s 3c	7g 3s 6c	5 lb.

Figure 4.10: A dnd-table generated by the stockpile output file.

4.9.2 Stockpile Config

The configuration file consists of key-value pairs, where each line follows the ‘variable=value’ format. Below is a list of the available configuration parameters, their types, and descriptions.

- **cadence (int)**: The frequency in seconds at which the configuration parameters are updated. Default: 86400.
- **general_price_variance (float)**: The variance factor for general item prices. Default: 0.05.
- **trade_price_variance (float)**: The variance factor for trade item prices. Default: 0.2.
- **general_items_percent_in_stock (float)**: The base percentage of general items that are in stock at any given time. Default: 0.9.
- **general_items_percent_in_stock_variance (float)**: The variance on the *general_items_percent_in_stock* value. Default: 0.05.
- **trade_items_percent_in_stock (float)**: The base percentage of trade items that are in stock at any given time. Default: 0.2.
- **trade_items_percent_in_stock_variance (float)**: The variance on the *trade_items_percent_in_stock* value. Default: 0.1.
- **sell_price_percentage (float)**: The base percentage of the buy price to determine sell prices for items. Default: 0.75.
- **sell_price_percentage_variance (float)**: The variance of the sell prices. Default: 0.1.
- **general_items_input_tag (str)**: The tags that appear as the header for the dnd-table lines in the .input files for general items. Default: ”General Items”.
- **trade_items_input_tag (str)**: The tags that appear as the header for the dnd-table lines in the .input files for trade items. Default: ”Specialty/Trade Items”.

Below is an example of a configuration file with custom values:

```
cadence=43200
general_price_variance=0.1
trade_price_variance=0.15
general_items_percent_in_stock=0.85
general_items_percent_in_stock_variance=0.04
trade_items_percent_in_stock=0.25
trade_items_percent_in_stock_variance=0.08
sell_price_percentage=0.7
sell_price_percentage_variance=0.15
general_items_input_tag=Basic Items
trade_items_input_tag=Trade Goods
```

When the application starts, it attempts to load the configuration parameters from the specified file. If no file is provided or the file is not found, default values are used, and a warning is displayed. If the configuration file is not found, the application will output an error message and continue using the default values. Ensure the file path is correct and the file is accessible. For constructing a configuration file, the application provides a help message detailing the available variables and their descriptions. Use the ‘variable=value’ format.

4.9.3 stockpile_plot.py

The ‘stockpile_plot.py’ script will create plots of all the buy and sell prices of the generated stockpile prices (see section 4.9.1.2). The input files should be formatted by the stockpile.py script or match the format of the files output by that script.

```
usage: stockpile_plot.py [-h] -b BUY -s SELL [-i ITEM] [-o OUTPUT]

S.T.O.C.K.P.I.L.E.S price history plotting.

optional arguments:
  -h, --help            show this help message and exit
```

```
-b BUY, --buy BUY      The file containing buy price history for the stockpile items.
-s SELL, --sell SELL   The file containing sell price history for the stockpile items.
-i ITEM, --item ITEM   The item to search for and plot. This will display the plot as
  well. If not specified, all plots will be generated.
-o OUTPUT, --output OUTPUT
                       The folder to output all plots to. This option will be ignored if
  '-i' is specified.
```

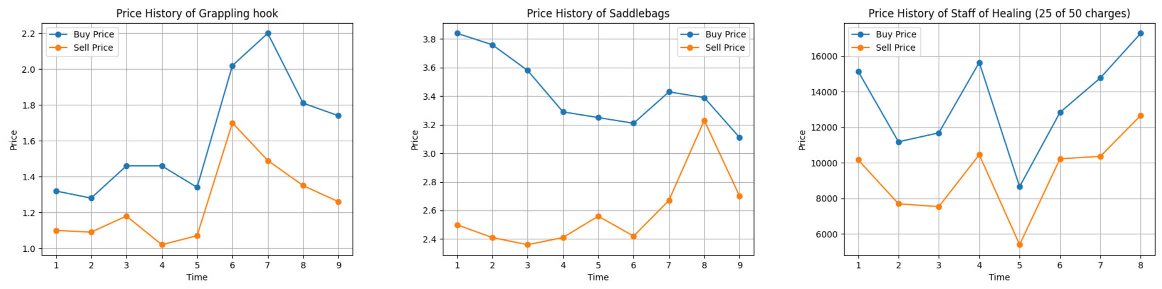


Figure 4.11: Three sample plots created by the stockpile_plot.py script for the stockpile system.

4.9.4 templates/add_external_links.py

The ‘add_external_links.py’ script processes HTML files to replace specified words with corresponding hyperlinks. It is designed to automate the embedding of links within a set of HTML files based on a predefined list.

To use the script, ensure the list of words and links is saved in a file (e.g., lists/links.list), with each line formatted as ‘word,link’. Place the HTML files to be processed within the target directory (e.g., ../campaign) - which is the default place they should be. Then run the script. It will load the word-link pairs, find all HTML files in the specified directory, and replace the specified words in each HTML file with the corresponding hyperlinks (in the appropriate HTML format).

4.9.5 templates/visualize_nodes.py

This script generates and visualizes a network graph of HTML files within a campaign (or a specified with some modification) directory. The script begins by defining the directory to scan for HTML files and then proceeds to build and visualize the graph. The process consists of the following steps:

1. **Extracting Links:** The script uses the ‘extract_links’ function to parse HTML content and extract links to other HTML files. This function utilizes the BeautifulSoup library to find all anchor tags with an ‘href’ attribute that ends with ‘.html’.

2. **Building the Graph:** The ‘build_graph’ function constructs a directed graph using NetworkX. It scans the given directory and its subdirectories for HTML files, excluding ‘index.html’ and files containing ‘_public’. For each HTML file, the function adds a node to the graph and creates directed edges between nodes based on the links found in the files.

3. **Visualizing the Graph:** The ‘visualize_graph_plotly’ function visualizes the graph using Plotly. It computes the layout of the graph with ‘nx.spring_layout’ and generates traces for edges and nodes. The node sizes are dynamically adjusted based on their degree, and the final interactive visualization displays the network of HTML links.

4.9.6 templates/trash/img/cleanup_trash_images.py

This script is designed to identify and delete image files from the campaign (or a specified with some modification) directory if matching files are found in a database folder. The script starts by defining the current directory as the script folder and calculates the relative path for the campaign folder. It then proceeds to find and remove matching image files from the script folder. The script operates as follows:

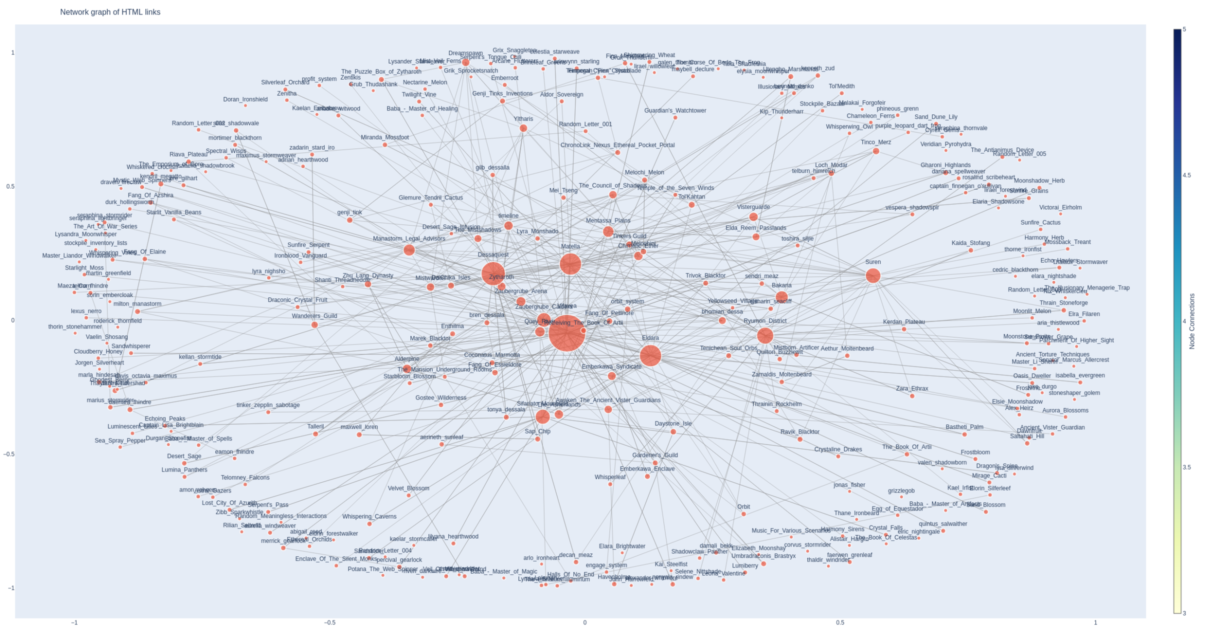


Figure 4.12: A visualization of the nodes for all HTML files and their connections. The larger nodes represent files that have more links within them.

1. **Getting Image Files:** The script uses the 'get_image_files' function to generate a list of image files from a given directory. It filters files based on common image file extensions including '.png', '.jpg', '.jpeg', '.gif', '.bmp', and '.tiff'.
2. **Finding and Deleting Images:** The 'find_and_delete_images' function searches for images in the database folder that match the names of images found in the script folder. For each matching image file, it deletes the file from the script folder.
3. **Execution:** The script sets the paths for the script folder and the database folder. It then calls the 'find_and_delete_images' function to perform the matching and deletion operation.

Github Integration And Automation

5.1 Github Integration

Both the front-end and back-end MMORPDND projects are hosted on github at the following links:

- Front-end: `https://github.com/MMORPDND/MMORPDND.github.io`
- Back-end: `https://github.com/torodean/DnD`

The front-end repository is a static github.io page which contains the back-end repository as a Git sub-module. This sub-module is referenced as a 'database' for front-end users to interact with. The 'database' is essentially only the 'campaign' folder of the back-end repository and does not publicly link to the scripts or back-end specific tools.

5.2 Github Automated Features

The GitHub repository contains a few automated features. The 'automated-updates' branch is dedicated to these features and all of the workflows and automated features will run and update this branch. It is then up to the developers to merge these into main. The front-end website will link directly to the 'automated-updates' branch instead of main.

5.2.1 Automated Python Tests

This section provides an overview of the GitHub Actions workflow used to run automated tests. Its purpose is to ensure that the codebase remains stable by running a suite of tests. The workflow '.github/-workflows/tests.yml' ensures that any changes made to the repository are automatically tested, providing immediate feedback on the impact of those changes. The primary goal of this workflow is to streamline the process of testing and validating code changes, ensuring that new commits do not introduce regressions or break existing functionality. It automatically triggers on pushes to the 'main' branch and pull requests targeting the 'main' branch, ensuring continuous integration and testing. By integrating this workflow, developers can maintain a high level of code quality with minimal manual effort, leveraging the automation capabilities provided by GitHub Actions. By integrating this workflow, developers can maintain a high level of code quality and quickly identify and fix issues introduced by new changes. The key workflow steps follow:

1. Repository Checkout: The workflow begins by checking out the repository to access the code and test files.
2. Python Setup: It sets up the specified version of Python required for running the tests.
3. Dependencies Installation: Installs necessary dependencies, specifically 'pytest', which is used for running the tests.
4. Test Execution: Changes the directory to where the tests are located and runs the specified test files using 'pytest'.

Developers using this workflow should ensure the following:

- All necessary dependencies are listed and can be installed via pip.

- Test files are located in the specified directory and are correctly named.
- The 'main' branch is used consistently for integration and testing purposes.
- Required secrets, such as GitHub tokens, are set up in the repository settings to allow the workflow to push changes if needed.

5.2.2 Automated README Update And Generation

This document describes the GitHub Actions workflow configured to automatically update README files with image details. The workflow, defined in '.github/workflows/run_script.yml', is triggered on pushes to the 'automated-updates' branch that affect the 'templates/img/' and 'campaign/characters/' directories. The purpose of this script is to provide a quick visual reference (the README.md files) for images contained within the folders. This allows the developers to quickly see what the images in Github are without having to select them when in a browser. For users developing in a desktop environment, this feature may not be needed. It performs the following key steps:

- Repository Checkout: Checks out the repository to access the code and directories.
- Node.js Setup: Sets up the specified version of Node.js needed to run the script.
- Script Execution: Runs the 'scripts/update_readmes.js' script, which processes image directories and updates the README files. The script generates or updates a 'README.md' file in each directory with a table of images, including thumbnails and filenames.
- Commit and Push Changes: Commits any changes made by the script and pushes them back to the 'automated-updates' branch.

Developers should ensure the Node.js script 'update_readmes.js' is correctly placed in the 'scripts' directory and has execution permissions. The workflow configuration must be accurate, with required GitHub tokens set up to allow the workflow to push changes. This workflow automates the process of keeping README files up-to-date with image information, thereby enhancing documentation consistency and reducing manual updates.

5.2.2.1 scripts/update_readmes.js

The 'scripts/update_readmes.js' script automates the generation of README.md files within specified directories by creating HTML tables of images. It processes directories listed in TEMPLATE_DIR, NON_PLAYER_DIR, NON_PLAYER_DIRS, and PLAYER_DIR (all four defined below), using the glob package to handle nested directories. For each directory, the script reads all image files (excluding any existing README.md), formats them into a table with thumbnails and filenames, and writes this content into a README.md file within the same directory. This automation helps maintain up-to-date documentation for image assets across various project directories.

- TEMPLATE_DIR = './templates/img/';
- NON_PLAYER_DIR = './campaign/characters/non-player/img/';
- NON_PLAYER_DIRS = './campaign/characters/non-player/*/img/';
- PLAYER_DIR = './campaign/characters/player/img/';

5.2.3 Automated LaTeX PDF Building

This section describes the purpose and usage of a GitHub Actions workflow designed to automate the generation and updating of LaTeX documents. The workflow '.github/workflows/build_latex.yml' ensures that any changes made to the LaTeX source files within the repository are automatically compiled and the resulting PDF is committed back to the repository. The primary goal of this workflow is to streamline the process of maintaining LaTeX documentation and ensure the documentation is properly generated when browser-based Git changes are added or changes on a host machine that does not have LaTeX generation capabilities. It automatically triggers on pushes to the 'automated-updates' branch and changes within the documentation directory, ensuring that the LaTeX documents are always up-to-date without manual intervention. By integrating this workflow, developers can maintain consistent and up-to-date LaTeX documentation with minimal manual effort, leveraging the automation capabilities provided by GitHub Actions. The key workflow steps follow:

1. Repository Checkout: The workflow begins by checking out the repository to access the LaTeX source files.
2. TeX Live Setup: It sets up a TeX Live environment to compile the LaTeX documents.
3. Document Compilation: The LaTeX source files are compiled into a PDF.
4. File Renaming: The generated PDF is renamed for clarity or versioning purposes.
5. Commit and Push: The updated PDF is committed and pushed back to the repository, ensuring that the latest version is always available.

Developers using this workflow should ensure the following:

- The LaTeX source files are located in the specified directory.
- The workflow file is correctly configured with the appropriate branch and paths.
- Required secrets, such as GitHub tokens, are set up in the repository settings to allow the workflow to push changes.