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

Developer: Antonius Torode

Version – 1.001
Latest update: February 2, 2024

The Original Maintainer of this work is: Antonius Torode.

The Current Maintainer of this work is: Antonius Torode.

This document is designed for the sole purpose of providing a template for writing a manual/book in LaTeX.

This document is continuously under development.
Most Current Revision Date: February 2, 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.

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! Thereforethe 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 workin gand 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 plaers 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 compoents 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 thoguth 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 linkes 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 dissapear 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 flate 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.

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 (or S.T.O.C.K.P.I.L.E.S 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). 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 S.T.O.C.K.P.I.L.E. 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 S.T.O.C.K.P.I.L.E. 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 MMORPDND. 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 and hyperlink all words found in all html files to the appropriate html file whose name matches the words found.
7. Beautify the code.

4.1.2 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.

4.1.2.1 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.

4.1.2.2 Command Line

The MMORPDND application supports some command line features.

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

MMORPDND Tools and apps.

optional arguments:
  -h, --help    show this help message and exit
  -t, --test    Runs the test-all feature then exit.
  -u, --update  Runs the update-all feature then exit.
```

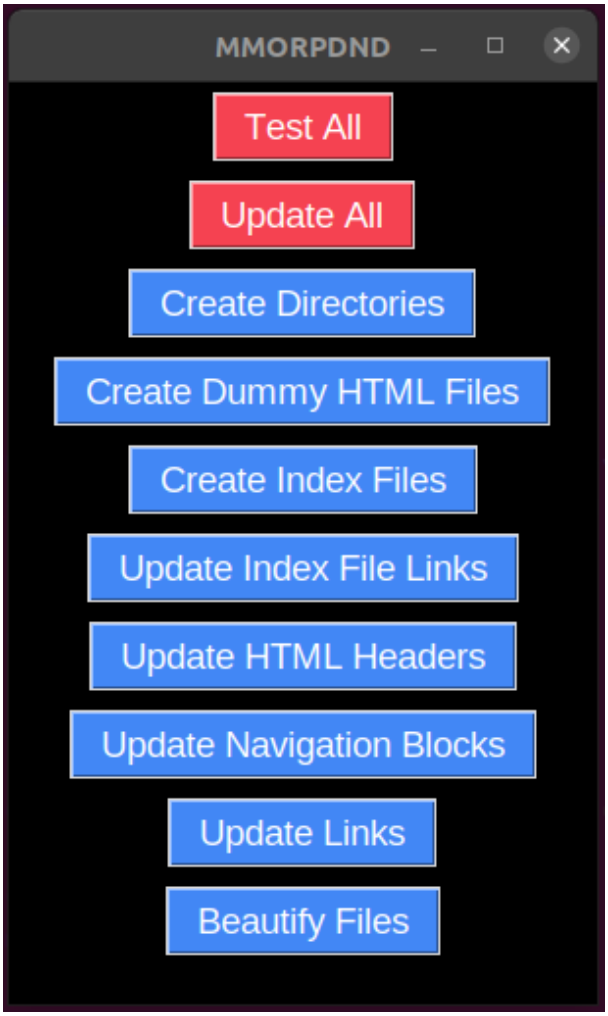

Figure 4.1: The MMORPDND Main GUI

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.

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.

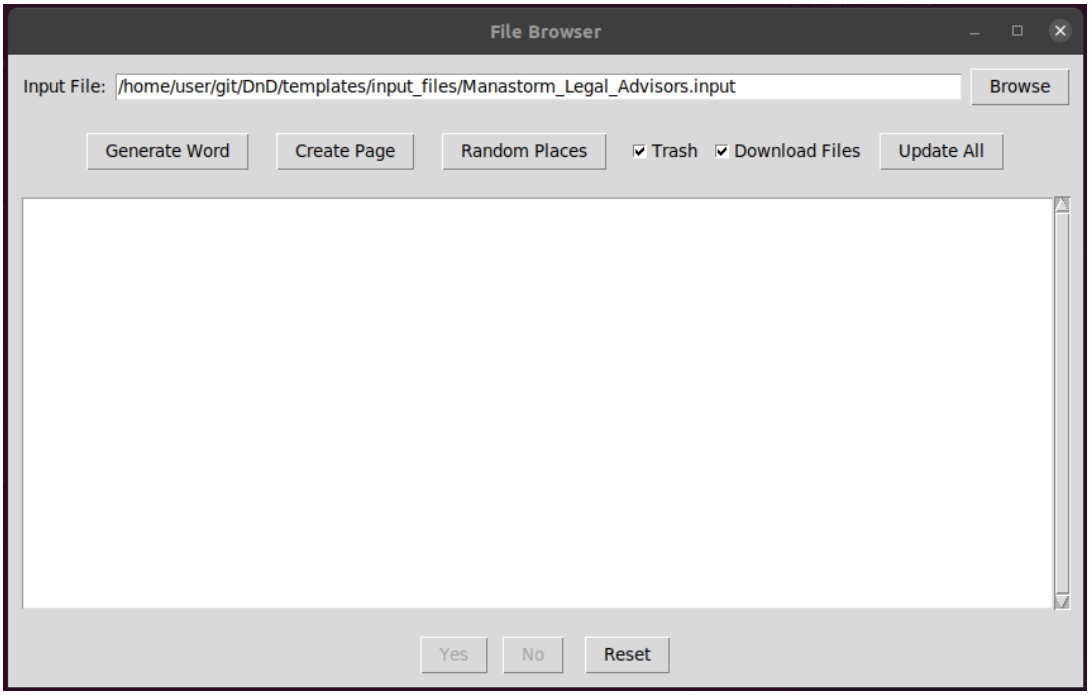

Figure 4.2: The MMORPDND Main GUI

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 4.2.2.13. 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).

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.
```

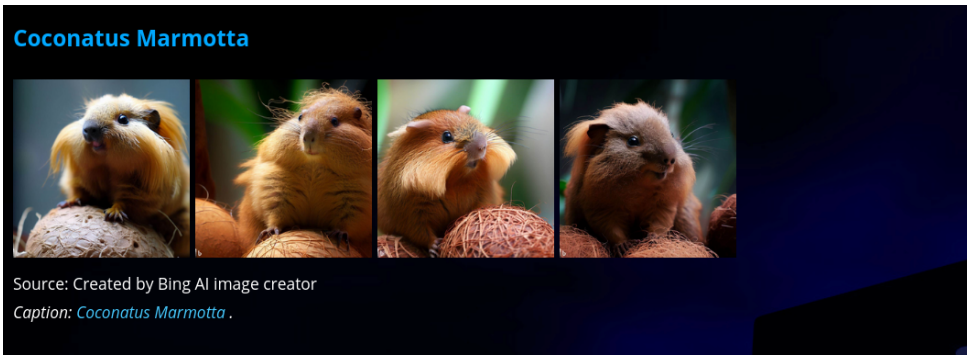

Figure 4.3: A dnd-image section generated using 4 images.

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.
```

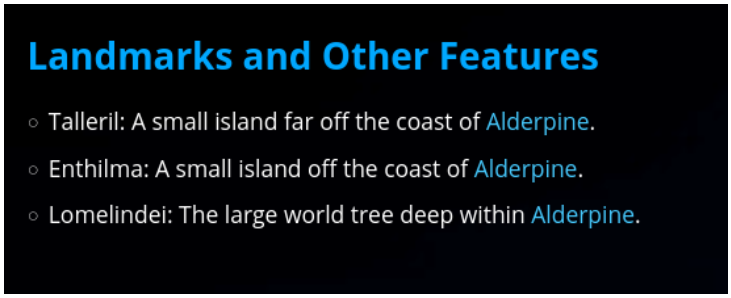

Figure 4.4: A dnd-list section.

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...
```

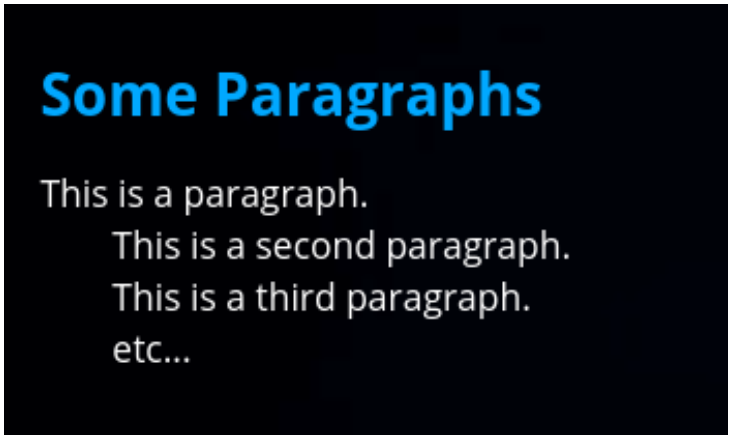

Figure 4.5: A dnd-info section with only paragraphs. Note that the first paragraph will not be indented but all subsequent ones will be.

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.
```

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

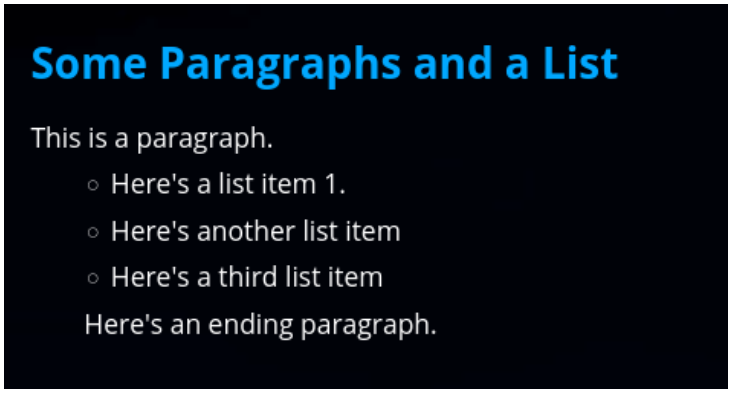

Figure 4.6: A dnd-info section with a paragraph followed by a list, followed by a paragraph.

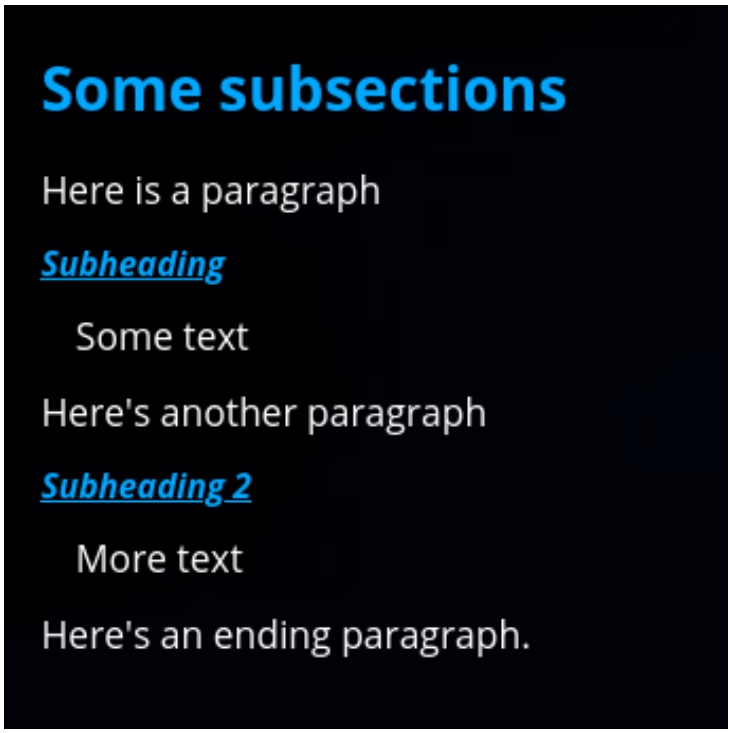

Figure 4.7: A dnd-info section with some paragraphs and two subsections.

work with youtube links. Given a simple youtube link, the code will find the name of the video and 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
```

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
```

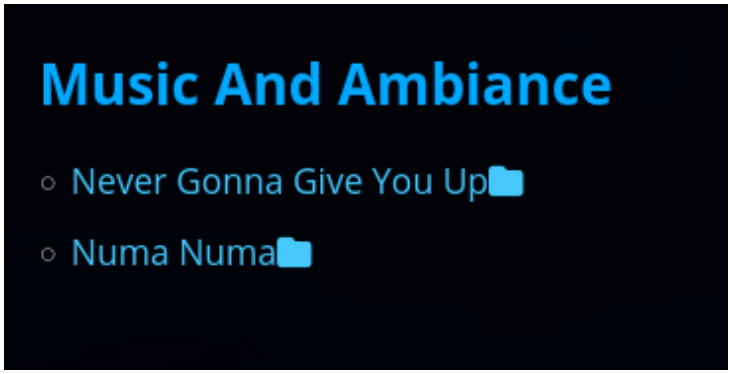

Figure 4.8: A dnd-music section with two songs linked to.

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

[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.

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 will search the templates/img folder for any image references within the character files.

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.

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

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.
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 Name Files (.names)

SECTION IN PROGRESS!

4.2.2.14 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.

4.2.2.15 Command Line

The creator application does NOT currently support command line features.

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

This tool is still in development and not yet functioning.

Programming Methods and Variables

4.9 MMORPPDND Programming Methods and Variables

class MMORPDND_VARS

```
"""
This is a class for storing variables used by MMORPDND* classes.

Methods:
   __init__(self)

Variables:
   # Define the directory structure
   self.directory_structure = { ... }

   # Define the number of HTML files to create in each subdirectory
   self.num_dummy_files_per_subdir = 3  # Stores all files created so far.
   self.all_index_files = []

   # Define the root directory
   self.root_dir = os.getcwd()
   self.script_dir = os.path.dirname(os.path.abspath(__file__))

   # Directories to exclude when processing.
   self.directories_to_exclude = ["templates", "css", ".git", ".idea", ".
       github", "scripts", "docs"]

   # Define the regular expression to match the header section
   self.header_regex = re.compile(r"<head>.*?</head>", re.DOTALL)

   # Define the regular expression to match the header section
   self.title_regex = re.compile(r"<title>.*?</title>", re.DOTALL)

   # Define the template file paths
   self.header_template_file = "templates/headerTemplate.html"
   self.nav_template_file = "templates/navTemplate.html"
   self.css_path = "css/mmorpdnd.css"
"""
```

global_vars = MMORPDND_VARS()

```
"""
Define a global variable containing the declared vars. Use this so they are
    all only defined once and can be updated/stored throughout the
    applications lifetime.
"""
```

is_image_file(file_name)

```
"""
   Checks if a file name is an image file based on its extension.

Args:
   file_name (str): The name of the file to check.

Returns:
   bool: True if the file name has an image extension, False otherwise.

Example:
   >>> is_image_file('myphoto.jpg')
   True
   >>> is_image_file('document.pdf')
   False
"""
```

get_relative_path(from_file, to_file)

```
"""
Returns the relative path from one file to another.

Args:
   from_file (str): The path of the source file.
   to_file (str): The path of the target file.

Returns:
   str: The relative path from the source file to the target file.

Raises:
   None.

This method takes two file paths, `from_file` and `to_file`, and calculates
    the relative path from `from_file`
to `to_file`. The relative path represents the path that, when followed from
     `from_file`, leads to `to_file`.

Example:
   Assuming from_file = '/path/to/source/file.html' and to_file = '/path/to/
       target/image.jpg',
   the method will return '../../target/image.jpg' as the relative path.

Note:
   The method uses the `os.path.relpath()` function to calculate the
       relative path.

Example usage:
   get_relative_path('/path/to/source/file.html', '/path/to/target/image.jpg
       ')
   print(relative_path)  # Output: '../../target/image.jpg'
"""
```

create_dummy_html_files(directory=global_vars.root_dir)

```
"""
Creates dummy HTML files in all directories and subdirectories for testing
    purposes.

Args:
```

```
    directory (str): The directory path to start creating dummy HTML files
        from. Defaults to global_vars.root_dir.

Returns:
    None.

Raises:
    None.

This method recursively walks through the directory structure, creates an
    index.html file in each directory,
and creates additional HTML files with random links in each subdirectory.

The method performs the following steps:
    1. Creates an index.html file in the specified directory with a basic
        HTML structure.
    2. Recursively walks through the directory structure using `os.walk`.
    3. For each subdirectory, excluding any directories listed in `
        global_vars.directories_to_exclude`:
       - Creates an index.html file in the subdirectory with a basic HTML
           structure.
       - Generates a specified number of dummy HTML files in the subdirectory
           , each containing a random link to another dummy file.
       - Prints a message indicating the successful creation of each HTML
           file.
    4. Creates additional dummy HTML files in the script directory (specified
         by the `directory` argument), each containing a random link to
        another dummy file.
    5. Prints a message indicating the successful creation of all HTML files.

Note:
    The content of the generated HTML files consists of a basic HTML
        structure with a header and body.
Each dummy file includes a link to two randomly chosen dummy files,
    facilitating testing scenarios.

Example usage:
    create_dummy_html_files()
"""
```

alphabetize_links(list_of_links)

```
"""
Alphabetizes the items in a list of links.

Args:
    list_of_links (str): A multiline string representing a list of links in
        the format
    "<li><a href="url">link_text</a></li>". Each link should be on a separate
         line.

Returns:
    str: A multiline string representing the alphabetized list of links.

Example:
    links = '''<li><a href="valen_shadowborn.html">valen_shadowborn</a></li>
    <li><a href="kaelar_stormcaller.html">kaelar_stormcaller</a></li>
    <li><a href="thorne_ironfist.html">thorne_ironfist</a></li>
    <li><a href="foobar.html">foobar</a></li>
    <li><a href="aria_thistlewood.html">aria_thistlewood</a></li>
    <li><a href="stoneshaper_golem.html">stoneshaper_golem</a></li>
    <li><a href="elara_nightshade.html">elara_nightshade</a></li>'''
```

```

    sorted_list = alphabetize_links(links)

    print(sorted_list)
"""
```

class MMORPDND

```
"""
A class for all the main MMORPDND features.

Methods:
    __init__(self)
    create_directories(self, path: str, structure: dict) -> None
    create_index_files(self, directory=global_vars.root_dir)
    move_dir_items_to_end(self, string)
    move_img_items_to_end(self, string)
    update_index_files(self)
    update_headers(self, directory=global_vars.root_dir)
    update_navigation(self, directory=global_vars.root_dir)
    beautify_files(self, directory=global_vars.root_dir)
    find_all_html_files(self, directory=global_vars.root_dir)
    update_html_links(self, directory=global_vars.root_dir)
"""
```

MMORPDND.__init__(self)

```
"""
Initialization method.
"""
```

MMORPDND.create_directories(self, path: str, structure: dict) -¿ None

```
"""
Recursively creates directories in the given path according to the structure
     specified in the dictionary.

Args:
    path (str): The root path where directories will be created.
    structure (dict): A dictionary representing the structure of the
        directories to be created.

Returns:
    None
"""
```

MMORPDND.create_index_files(self, directory=global_vars.root_dir)

```
"""
Creates index files for all subdirectories within the specified directory.

Args:
    directory (str): The directory path to start creating index files from.
        Defaults to global_vars.root_dir.

Returns:
    None.

Raises:
```

```
    None.

This method traverses through all subdirectories and files starting from the specified directory and creates an
index file named "index.html" in each directory that doesn't already have one.

The method performs the following steps:
    1. Loop through all directories and files using `os.walk` starting from the specified directory.
    2. Check if an index file named "index.html" already exists in the current directory. If so, skip that directory.
    3. Check if the current directory matches any of the excluded directories defined in `global_vars.directories_to_exclude`.
    If so, skip that directory.
    4. Create an index.html file in the current directory.
    5. Write the HTML content to the index.html file, including the directory name in the title and header.
    6. Print a message indicating the creation of the index file.

Note:
    The index.html file created contains a basic HTML structure with the title and header set to "Index of [directory_name]".

Example usage:
    create_index_files()
"""
```

MMORPDND.move_dir_items_to_end(self, string)

```
"""
Moves directory items (lines containing "/index.html") to the end of the input string.

This method takes a multi-line string as input and separates lines that contain "/index.html"
(directory items) from other lines (non-directory items). It then rearranges the lines by moving
the directory items to the end while maintaining the order of non-directory items.

Args:
    string (str): The input multi-line string to be processed.

Returns:
    str: A modified string with directory items moved to the end.

Example:
    input_string = "Line 1\n/index.html\nLine 2\nLine 3\n/index.html"
    result = move_dir_items_to_end(input_string)
    # result will be "Line 1\nLine 2\nLine 3\n/index.html\n/index.html"
"""
```

MMORPDND.move_img_items_to_end(self, string)

```
"""
Moves items containing "img/" to the end of the string while preserving their original order.

Args:
```

```
    string (str): A string containing items in the format '<li><a href="url">
        link_text</a></li>'.

Returns:
    str: A new string with items containing "img/" moved to the end while
        preserving their original order.

Example:
    >>> string = '''<li><a href="valen_shadowborn.html">valen_shadowborn</a
        ></li>
    ...             <li><a href="kaelar_stormcaller.html">kaelar_stormcaller</
        a></li>
    ...             <li><a href="thorne_ironfist.html">thorne_ironfist</a></li
        >
    ...             <li><a href="foobar.html">img/foobar</a></li>
    ...             <li><a href="aria_thistlewood.html">img/aria_thistlewood</
        a></li>
    ...             <li><a href="stoneshaper_golem.html">stoneshaper_golem</a
        ></li>
    ...             <li><a href="elara_nightshade.html">elara_nightshade</a></
        li>'''
    >>> new_string = move_img_items_to_end(string)
    >>> print(new_string)
    <li><a href="valen_shadowborn.html">valen_shadowborn</a></li>
    <li><a href="kaelar_stormcaller.html">kaelar_stormcaller</a></li>
    <li><a href="thorne_ironfist.html">thorne_ironfist</a></li>
    <li><a href="aria_thistlewood.html">img/aria_thistlewood</a></li>
    <li><a href="stoneshaper_golem.html">stoneshaper_golem</a></li>
    <li><a href="elara_nightshade.html">elara_nightshade</a></li>
    <li><a href="foobar.html">img/foobar</a></li>
"""
```

MMORPDND.update_index_files(self)

```
"""
Updates all index files in the directory and subdirectories to include links
     to other files in the same directory.

Returns:
    None.

Raises:
    None.

This method performs the following steps:
    1. Prints a message indicating that index files are being updated.
    2. Retrieves a list of all HTML index files in the current directory and
        subdirectories, excluding any directories listed in `global_vars.
        directories_to_exclude`.
    3. For each index file found:
        - Reads the file data.
        - Identifies the HTML files present in the same directory as the
            current index file.
        - If the index links div section does not exist in the file, it adds
            the div section just before the closing </body> tag.
        - Updates the index links by generating HTML code for each HTML file
            in the directory (excluding the index.html file) and appending it
            to the index links div.
        - Replaces the old index links section in the file with the updated
            index links div.
        - Writes the updated file data back to the file.
        - Prints a message indicating that the index file has been updated.
```

```
    4. Prints a message indicating that all index.html files have been
        updated.

Note: The method relies on regular expressions for searching and updating
    the index links section in each index file.

Example usage:
    update_index_files()
"""
```

MMORPDND.update_headers(self, directory=global_vars.root_dir)

```
"""
Updates the headers of HTML files in the specified directory and its
    subdirectories to match a predefined template.

Args:
    directory (str): The directory path to update the HTML files in. Defaults
         to global_vars.root_dir.

Returns:
    None.

Raises:
    None.

This method performs the following steps:
    1. Loop through all HTML files in the specified directory and its
        subdirectories.
    2. For each HTML file found that does not contain "Template" in its
        filename:
       - Read the contents of the HTML file.
       - Check if the file is in the list of directories to exclude.
       - Read the contents of the header template file.
       - Replace the header section in the HTML file with the contents of the
           template.
       - Generate a title based on the filename and replace the title section
           in the HTML file.
       - Determine the relative path to the CSS file.
       - Calculate the number of subdirectories between the HTML file and the
           CSS file.
       - Create the correct link path for the CSS file.
       - Replace the placeholder in the HTML file with the link to the CSS
          file.
       - Overwrite the HTML file with the updated contents.
       - Print a progress update indicating the file that has been updated.
    3. Print a message indicating that the headers and CSS have been updated
        in all relevant HTML files.

Note:
    The method relies on regular expressions for pattern matching and
        modification.

Example usage:
    update_headers()
"""
```

MMORPDND.update_navigation(self, directory=global_vars.root_dir)

```
"""
```

```
Updates the navigation block of HTML files in the specified directory and
    its subdirectories to match a predefined template.

Args:
    directory (str): The directory path to update the HTML files in. Defaults
        to global_vars.root_dir.

Returns:
    None.

Raises:
    None.

This method performs the following steps:
    1. Loop through all HTML files in the specified directory and its
       subdirectories.
    2. For each HTML file found that does not contain "Template" in its
       filename:
       - Read the contents of the HTML file.
       - Check if the file is in the list of directories to exclude.
       - Print a progress message indicating the file being processed.
       - Read the contents of the navigation template file.
       - Find the navigation block in the original HTML file using a regular
         expression.
       - If a navigation block is found:
       - Replace the navigation block in the HTML file with the contents of
         the template.
       - Write the modified HTML back to the file.
       - Print a message indicating that the navigation block has been
         replaced.
       - If no navigation block is found:
       - Print a message indicating that the navigation block was not found.
       - Insert the navigation contents at the start of the body tag in the
         HTML file.
       - Overwrite the file with the updated contents.
       - Print a message indicating that the navigation contents have been
         inserted.

Note:
    The method relies on regular expressions for pattern matching and
       modification.

Example usage:
    update_navigation()
"""
```

MMORPDND.beautify_files(self, directory=global_vars.root_dir)

```
"""
Beautifies HTML and CSS files in the specified directory and its
    subdirectories.

Args:
    directory (str): The directory path to beautify the files in. Defaults to
        global_vars.root_dir.

Returns:
    None.

Raises:
    None.

```

```
This method performs the following steps:
   1. Modifies the directories_to_exclude list to include template and CSS
       files.
   2. Loops through all files and subdirectories in the specified directory.
   3. For each HTML or CSS file found (excluding those in the modified
       directories_to_exclude list):
      - Constructs the file path.
      - Checks if the file is an HTML or CSS file, and skips it if not.
      - Reads the contents of the file.
      - If the file is an HTML file:
      - Uses BeautifulSoup to parse the HTML and prettify it.
      - If the file is a CSS file:
      - Uses cssbeautifier to prettify the CSS code.
      - Writes the prettified code back to the file.
      - Prints a message indicating that the file has been prettified.

Note:
   The method relies on BeautifulSoup for HTML parsing and prettifying, and
       cssbeautifier for CSS prettifying.

Example usage:
   navigator = Navigator()
   navigator.beautify_files()
"""
```

MMORPDND.find_all_html_files(self, directory=global_vars.root_dir)

```
"""
Finds all HTML files (not index.html files) in a directory and its
    subdirectories.

Args:
   directory: The directory to search. Defaults to the current directory.

Returns:
   A list of dictionaries containing the name (without extension), name (
       with extension), and full path
of each HTML file found.
"""
```

MMORPDND.update_html_links(self, directory=global_vars.root_dir)

```
"""
Update the links in the various HTML files to link to the appropriate file.

Args:
   directory (str): The directory to search for HTML files. Defaults to
       global_vars.root_dir.

Returns:
   None
"""
```

class MMORPDND_GUI

```
"""
Class to store GUI functions and operations. The majority of the methods in
    this class simply connect the UI elements to the associated MMORPDND
    class methods.

```

```
Methods:
    __init__(self)
    run(self)
    test_all(self)
    update_all(self)
    create_directories(self)
    create_dummy_html_files(self)
    create_index_files(self)
    update_index_links(self)
    update_headers(self)
    update_navigation(self)
    beautify_files(self)
    update_html_links(self)
"""
```

main()

```
"""
Main method to start the gui and take optional arguments/parameters for
    running via command line.
"""
```

4.10 Creator Programming Methods and Variables

output_text(text, option = ”text”)

```
"""
Print text in different colors based on the provided option.

Args:
    text (str): The text to be printed.
    option (str): The color option for the text. Valid options are "text", "
        warning", "error", "note", and "success".

Returns:
    None

Note:
    This function uses ANSI escape codes for color formatting. Colors may
        not be displayed correctly in all environments.
"""
```

get_youtube_video_name(url)

```
"""
Retrieves the title of a YouTube video based on the provided URL.

Args:
    url (str): The URL of the YouTube video.

Returns:
    str or None: The title of the YouTube video if it can be retrieved
        successfully,
                 None if there was an error.

Raises:
    None

```

```
Example:
    >>> url = "https://www.youtube.com/watch?v=dQw4w9WgXcQ"
    >>> title = get_youtube_video_name(url)
    >>> print(title)
    "Rick Astley - Never Gonna Give You Up (Official Music Video)"
"""
```

find_longest_and_shortest(words)

```
"""
Find the lengths of the longest and shortest words in a given list.

Args:
    words (list): A list of words.

Returns:
    tuple: A tuple containing the lengths of the longest and shortest words.

Raises:
    ValueError: If the input list is empty.

Examples:
    >>> word_list = ["one", "two", "three", "four", "five"]
    >>> longest_length, shortest_length = find_longest_and_shortest(
        word_list)
    >>> print("Longest word length:", longest_length)
    >>> print("Shortest word length:", shortest_length)
    Longest word length: 5
    Shortest word length: 3
"""
```

remove_numbers_at_start(string)

```
"""
Remove numbers at the start of a string.

Args:
    string (str): The input string.

Returns:
    str: The string with numbers removed from the start.
"""
```

append_to_file(file_path, string_to_append)

```
"""
Append a string to a file.

Parameters:
    file_path: The path to the file to append to.
    string_to_append: The string to append to the file.
"""
```

read_lines_from_file(file_name)

```
"""
Reads lines from a file and returns them as a list.

Parameters:
```

```
    file_name (str): The name of the file to read.

Returns:
    A list of strings, where each string represents a line from the file.
    Any leading and trailing whitespace is stripped from each line.

Raises:
    FileNotFoundError: If the specified file does not exist.
    PermissionError: If the specified file cannot be opened due to
        insufficient permissions.
"""
```

calculate_hp(class_type: str, level: int, constitution: int) -¿ int

```
"""
Calculate the hit points (hp) of a Dungeons & Dragons (DnD) 5th edition
    character
based on their class, level, and constitution modifier.

Args:
    class_type (str): The character's class (e.g. 'fighter', 'wizard', '
        rogue').
    level (int): The character's level, between 1 and 20.
    constitution (int): The character's constitution score, between 1 and
        30.

Returns:
    int: The character's hit points, based on their class and level,
        modified by their
        constitution modifier.

Raises:
    ValueError: If the given class_type is not recognized.
"""
```

calculate_proficiency_bonus(level)

```
"""
Calculate the proficiency bonus based on character level.

Parameters:
    level (int): The character's level.

Returns:
    int: The character's proficiency bonus.
"""
```

roll_4d6_drop_lowest()

```
"""
Rolls 4d6 and returns the sum of the highest 3 dice.
Returns:
    int: The sum of the highest 3 dice.
"""
```

get_stat_priority(character_class)

```
"""
Returns a list of attributes ordered by the stat priority for a given class.
```

```
Args:
    character_class (str): The class to get the stat priority for.
Returns:
    list: The list of attributes ordered by the stat priority.
"""
```

adjust_stats_for_level(assigned_stats, level)

```
"""
Adjusts the stats for a character based on their level.

Args:
    assigned_stats (dict): A dictionary representing the character's current stats, where keys are
        the stat names and values are the corresponding stat values.
    level (int): The level of the character.

Returns:
    dict: A dictionary representing the adjusted stats based on the character's level.

Raises:
    None.

This method takes the current stats of a character, represented by the `assigned_stats` dictionary,
and adjusts the stats based on the character's level. The adjusted stats are returned as a new dictionary.

The adjustment of stats is determined by the character's level:
 - For levels below 4, no adjustments are made, and the original stats are returned.
 - For levels 4 to 7, 2 bonus attribute points are awarded.
 - For levels 8 to 11, 4 bonus attribute points are awarded.
 - For levels 12 to 15, 6 bonus attribute points are awarded.
 - For levels 16 to 18, 8 bonus attribute points are awarded.
 - For levels 19 and above, 10 bonus attribute points are awarded.

The method prints the awarded bonus points and the original and updated attributes for informational purposes.

Note: The stats dictionary is assumed to have numeric values for each stat.

Example usage:
    assigned_stats = {'strength': 10, 'dexterity': 12, 'intelligence': 14}
    adjusted_stats = adjust_stats_for_level(assigned_stats, 8)
"""
```

generate_character_stats(character_class, level=1)

```
"""
Generates a list of six stats for a character based on their class and level.
Args:
    character_class (str): The character's class.
Returns:
    list: The list of six stats.
"""
```

calculate_modifier(attribute_value)

```
"""
Calculate the DnD attribute modifier based on the value of the attribute.

Args:
    attribute_value (int): The value of the attribute.

Returns:
    int: The modifier value for the attribute.

Example:
    >>> calculate_modifier(15)
    2
"""
```

copy_file_to_directory(file_path, directory_path)

```
"""
Copy a file to a directory.

Args:
    file_path (str): The path to the file to copy.
    directory_path (str): The path to the directory to copy the file to.

Raises:
    ValueError: If the file or directory doesn't exist.

Returns:
    None
"""
```

move_file_to_directory(file_path, directory_path)

```
"""
Move a file to a directory.

Args:
    file_path (str): The path to the file to move.
    directory_path (str): The path to the directory to move the file to.

Raises:
    ValueError: If the file or directory doesn't exist.

Returns:
    None
"""
```

print_prob_matrix(prob_matrix)

```
"""
Prints a probability matrix to the console in JSON format.

Parameters:
    prob_matrix (dict): A dictionary representing the probability matrix,
    where each key is an input character and the corresponding value is a
        dictionary
    of output characters and their probabilities.
"""
```

generate_prob_matrix(words)

```
"""
Generates a probability matrix based on a list of words.

Parameters:
    words (list of str): A list of words to use in generating the
        probability matrix.

Returns:
    A dictionary representing the probability matrix, where each key is an
        input character
    and the corresponding value is a dictionary of output characters and
        their probabilities.

Example:
    >>> words = ["cat", "dog", "cut", "cog", "cot", "caught"]
    >>> prob_matrix = generate_prob_matrix(words)
    >>> prob_matrix
{
    'c': {'a': 0.4, 'u': 0.2, 'o': 0.4},
    'a': {'t': 0.5, 'u': 0.5},
    't': {},
    'd': {'o': 1.0},
    'o': {'g': 0.6666666666666666, 't': 0.3333333333333333},
    'g': {'h': 1.0},
    'u': {'t': 0.5, 'g': 0.5},
    'h': {'t': 1.0}
}

Note that the probabilities for each output character are normalized so that
     they sum to 1.0.
"""
```

generate_word(prob_matrix, min_length=4, max_length=10)

```
"""
Generate a random word using a probability matrix.

Parameters:
    prob_matrix (dict): A dictionary representing the probability matrix for
         generating words.
    min_length (int): The minimum length of the generated word. Default
        value is 4.
    max_length (int): The maximum length of the generated word. Default
        value is 10.

Returns:
    A string representing the generated word.

Algorithm:
    1. Choose a random length between min_length and max_length.
    2. Initialize the word with a random input character.
    3. Generate the next characters based on the probabilities in the matrix
        .
        a. Get the output probabilities for the current input character.
        b. Create a list of possible next characters based on their
            probabilities.
        c. Choose a random next character from the list.
        d. Update the word and the input character for the next iteration.
    4. Return the generated word.
"""
```

move_file(source_file_path, destination_folder_path)

```
"""
Move a file from the source path to the destination folder.

Args:
    source_file_path (str): The path to the file to be moved.
    destination_folder_path (str): The path to the destination folder.

Returns:
    None
"""
```

class Variables

```
"""
A class to store app wide variables.

Variables:
   self.current_prob_matrix = None
   self.current_file = ""
   self.current_list = []
   self.output_file_folder = ""
   self.character_template_file = "characterTemplate.html"

   # Define directories to exclude
   self.directories_to_exclude = ["templates", "css", ".git", ".idea", ".
       github", "scripts", "docs"]

   # Define the root directory
   self.root_dir = os.getcwd()
   self.trash_dir = self.root_dir + "/trash"

Methods:
   __init__(self)
   trash_file(self, file)
   reset(self)
"""
```

Variables.__init__(self)

```
"""
Initializes the Variables class by setting initial variable status'.
"""
```

Variables.trash_file(self, file)

```
"""
Move a file to the trash folder.

Parameters:
    file: The file to move.

Returns:
    None
"""
```

Variables.reset(self)

```
"""
Reset the state of some objects to their initial values.

This method resets the state of the object by clearing the values of the current_file, current_list,
and current_prob_matrix attributes. After calling this method, the object is restored to its initial
state, ready for new data to be processed and stored.

Example:
    my_object = Variables()
    my_object.current_file = "data.txt"
    my_object.current_list = [1, 2, 3]
    my_object.current_prob_matrix = {"A": 0.2, "B": 0.3}
    my_object.reset()
    # After resetting, my_object's attributes are cleared and ready for new data.
"""
```

global_vars = Variables()

```
"""
Define a global variable containing the declared vars. Use this so they are all only defined once and can be updated/stored throughout the applications lifetime.
"""
```

get_character_fields(file)

```
"""
Read a file containing character fields and their values, and return a dictionary of the fields.

Args:
    file (str): The path to the file containing character fields.

Returns:
    dict: A dictionary mapping character fields to their corresponding values.

Raises:
    None.

The method opens the specified file and reads its contents. Each line in the file is expected to
represent a character field and its value, separated by an equals sign (=). The method parses each
line, extracts the field name and value, converts them to lowercase, and stores them in a dictionary.
The resulting dictionary is returned.

If the 'class' field is not found in the character fields, an error message is printed, and an
empty dictionary is returned.

Example usage:
    character_fields = get_character_fields('character_data.txt')
"""
```

split_list(lst, n)

```
"""
Split a list into n sublists of approximately equal size.

Args:
    lst (list): The input list to be split.
    n (int): The number of sublists to create.

Returns:
    list: A list containing n sublists.

Example:
    >>> my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    >>> sublists = split_list(my_list, 3)
    >>> print(sublists)
    [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]
"""
```

create_html_list(values)

```
"""
Create an HTML list from a string of semi-colon-separated values.

Args:
    values (str): The string of semi-colon-separated values.

Returns:
    str: The HTML list generated from the values.

Example:
    >>> create_html_list("Item 1; Item 2; Item 3")
    '<ul>\n<li>Item 1</li>\n<li>Item 2</li>\n<li>Item 3</li>\n</ul>'
"""
```

separate_header_and_info(string)

```
"""
Separates the header value and information from a string in the format "*
    header* Information here".

Args:
    string (str): The input string in the specified format.

Returns:
    tuple: A tuple containing the title value and the information.

Example:
    >>> separate_title_and_info("*header* Information here")
    ('header', 'Information here')
"""
```

create_html_info(values)

```
"""
Create an HTML info block from a string of semi-colon-separated values.

Args:
    values (str): The string of semi-colon-separated values.

```

```
Returns:
    str: The HTML info block generated from the values.

Example:
    >>> create_html_info("Item 1; - Item 2; - Item 3; Item 4")
    '<p class="first-paragraph">Item 1</p>\n<p><ul><li>Item 2</li>\n<li>Item
        3</li>\n</ul></p>\n<p>Item 4</p>\n'
"""
```

create_html_table(input_line)

```
"""
Convert a single line input to an HTML table format.

Args:
    input_line (str): Single line input containing semi-colon-separated
        values.

Returns:
    str: HTML table structure representing the input values.

Example:
    For a table of the form:

    -----------
    | a1 | a2 |
    |---------|
    | b1 | b2 |
    |---------|
    | c1 | c2 |
    -----------

    input: "2,a1,a2,b1,b2,c1,c2"
    output: '<table><tr><td>Value 1</td><td>Value 2</td></tr><tr><td>Value
        3</td><td>Value 4</td></tr><tr><td>Value 5</td><td>Value 6</td></tr
        ></table>'
"""
```

download_image(url, file_path)

```
"""
Download an image from a URL and save it to a file path.

Args:
    url (str): The URL of the image to download.
    file_path (str): The file path to save the downloaded image.

Returns:
    bool: True if the image was successfully downloaded and saved, False
        otherwise.
"""
```

add_number_to_filename(filename, number)

```
"""
Add a number to the filename before the extension.

Args:
    filename (str): The original file name.
    number (int): The number to add.
```

```
7
8  Returns:
9      str: The updated file name with a number added before the extension.
10
11 Example Usage:
12     >>> new_filename = add_number_to_filename("document.txt")
13     >>> print(new_filename, 3)
14     document (3).txt
15 """
```

is_image_file(file_name)

```
1  """
2  Checks if a file name is an image file based on its extension.
3
4  Args:
5      file_name (str): The name of the file to check.
6
7  Returns:
8      bool: True if the file name has an image extension, False otherwise.
9
10 Example:
11     >>> is_image_file('myphoto.jpg')
12     True
13     >>> is_image_file('document.pdf')
14     False
15 """
```

create_html_img(input_line)

```
1  """
2  Create an HTML block for an image section.
3
4  Args:
5      input_line (str): The input string of the form "image_file, image_source
           , caption".
6
7  Returns:
8      str: The HTML block representing the image section.
9
10 Notes:
11     - If the image file already exists, it will be used. Otherwise, the
           image will be downloaded from the provided image source URL.
12     - The image file, image source URL, and caption are extracted from the
           input line.
13
14 Example:
15     input_line = "image.jpg; https://www.example.com/image.jpg; A beautiful
           sunset"
16     html_block = create_html_img(input_line)
17 """
```

fix_image_extensions()

```
1  """
2  Updates all image extensions by running the fix_image_extensions.py script.
3
4  Example usage:
5        fix_image_extensions()
6  """
```

update_all()

```
"""
Updates all components of the MMORPDND system.

This function updates the MMORPDND system by performing the following steps:
   1. Retrieves the current working directory.
   2. Changes the current working directory to the parent directory.
   3. Constructs a command to update the system by running './mmorpdnd.py -u
       '.
   4. Executes the update command using the system shell.
   5. Changes the current working directory back to the original directory.

Note: This function assumes that the 'mmorpdnd.py' script is located in the
    parent directory.

Example usage:
   update_all()
"""
```

get_random_line(file_path)

```
"""
Return a random line from a file.

Args:
    file_path (str): The path to the file.

Returns:
    str: A random line from the file.

Raises:
    FileNotFoundError: If the file does not exist.
    IOError: If there is an error reading the file.
"""
```

extract_first_integer(string)

```
"""
Extracts the first integer from a given string.

Args:
    string (str): The input string.

Returns:
    int or None: The first integer found in the string, or None if no
        integer is found.

Example:
    >>> string = "'6 (barbarian 3, rogue 3)'"
    >>> first_integer = extract_first_integer(string)
    >>> print(first_integer)
    6
"""
```

class Creator

```
"""
This is the main class for the Crteator GUI and functionality.

```

```
Variables:
   self.last_user_input = None
   self.gui = tk.Tk()
   self.gui.geometry("850x500")
   self.gui.title("File Browser")
   self.gui.tk.call('wm', 'iconphoto', self.gui._w, icon)
   # Other GUI variables.

Methods:
   __init__(self)
   create_html_music(self, urls)
   download_youtube_video_as_mp3(self, url, output_path="../music")
   random_place(self, number=100)
   create_pages(self)
   create_page(self, file=global_vars.current_file)
   checkbox_changed(self)
   generate_char(self, file=global_vars.current_file)
   output_text(self, text)
   test(self)
   get_user_choice(self)
   yes(self)
   no(self)
   reset(self)
   browse_files(self)
   update_input_file(self)
   generate_word(self)
   run(self)
"""
```

Creator.create_html_music(self, urls)

```
"""
Creates an HTML list with links to the provided URLs and a folder icon.

Args:
    urls (str or list): The URL or list of URLs as a semicolon-delimited
        string or a list of strings.

Returns:
    str: The HTML list portion with links and folder icons.

Example:
    >>> urls = "https://www.youtube.com/watch?v=dQw4w9WgXcQ;https://www.
        youtube.com/watch?v=VIDEO2_ID"
    >>> html_list = self.create_html_music(urls)
    >>> print(html_list)
    <ul>
    <li><a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">Video 1</a><a
        href="local/path/video1.mp3"><i class="fas fa-folder"></i></a></li>
    <li><a href="https://www.youtube.com/watch?v=VIDEO2_ID">Video 2</a><a
        href="local/path/video2.mp3"><i class="fas fa-folder"></i></a></li>
    </ul>
"""
```

Creator.download_youtube_video_as_mp3(self, url, output_path="../music")

```
"""
Downloads a YouTube video as a high-quality MP3 file.

Args:
    url (str): The URL of the YouTube video.
```

```
    output_path (str): The path to the directory where the MP3 file will be
        saved.

Returns:
    str or None: The path of the downloaded MP3 file if the download and
        conversion
                 are successful, None if there was an error.

Raises:
    None

Example:
    >>> url = "https://www.youtube.com/watch?v=dQw4w9WgXcQ"
    >>> output_path = "/path/to/output/directory"
    >>> mp3_path = download_youtube_video_as_mp3(url, output_path)
    >>> print(mp3_path)
    "/path/to/output/directory/output.mp3"
"""
```

Creator.random_place(self, number=100)

```
"""
Generates random place names along with their corresponding types.

Args:
    number (int, optional): The number of random place names to generate.
        Defaults to 50.

Returns:
    None
"""
```

Creator.create_pages(self)

```
"""
Generate page files for a file or each file within a directory.

This method checks if the current file (global_vars.current_file) is a
    directory.
If it is a file, it calls the generate_char() or create_page() method based
    on the file extension.
If it is a directory, it iterates through each file within the directory and
     calls the generate_char()
or create_page() method for each individual file.

Returns:
    None

Raises:
    None
"""
```

Creator.create_page(self, file=global_vars.current_file)

```
"""
Create an HTML page based on the input file.

Reads the input file specified by global_vars.current_file and extracts the
    content to generate an HTML page.
The input file should have a '.input' extension.
```

```
The output HTML file is created in the specified destination folder or the
    current directory if not specified.

Returns:
    None
"""
```

Creator.checkbox_changed(self)

```
"""
This method is called when checkboxes are checked or unchecked.
It retrieves the values of the checkboxes and prints a message indicating
    whether they are enabled or disabled.
"""
```

Creator.generate_char(self, file=global_vars.current_file)

```
"""
Generate a character file based on the provided input file.

This method updates the input file, checks if it has the correct file type
    (.char),
generates character statistics and fields if they are not defined, replaces
    the fields
in the template file with the character information, and writes the new
    character file.

The process involves the following steps:
   1. Verifies if the input file has the correct file type (.char). If not,
       it displays an error message and exits.
   2. Retrieves the character fields from the input file.
   3. Determines the character class and level.
   4. If any fields are missing, generates default values for certain
       attributes and displays them.
   5. Calculates the character's hit points (hp) if not already defined.
   6. Generates the character file path and filename based on the character'
       s name.
   7. Reads the character template file.
   8. Processes and replaces the fields in the template with the character
       information.
      - Replaces general character fields.
      - Calculates and inserts modifier values for certain attributes.
      - Handles proficiencies and adds proficiency bonus to corresponding
          skills.
      - Populates information, notes, and image blocks.
      - Updates abilities and equipment lists.
   9. Writes the new character file at the specified filepath.
   10. If the option is selected, moves the input file to the trash.

Returns:
    None
"""
```

Creator.output_text(self, text)

```
"""
Output the given text to the GUI window and the large_text widget.

This method displays the provided text in the GUI window and appends it to
    the large_text widget.
```

```
It also ensures that the text is visible by scrolling to the bottom of the widget and updates
the GUI window to reflect the changes.

Args:
    text (str): The text to be displayed and appended to the large_text widget.

Example:
    gui_instance = MyGUI()
    gui_instance.output_text("Processing completed successfully.")
    # The text "Processing completed successfully." is displayed in the GUI window.
"""
```

Creator.test(self)

```
"""
Method for testing.

Returns:
    None
"""
```

Creator.get_user_choice(self)

```
"""
Displays a graphical user interface with yes/no buttons and returns the user's choice.

Args:
    None.

Returns:
    bool: The user's choice. True represents "yes" and False represents "no".

Raises:
    None.

This method displays a graphical user interface (GUI) with "yes" and "no" buttons and waits for the user to make a choice.
The GUI is implemented using a main event loop.

The method performs the following steps:
   1. Enables the "yes" and "no" buttons.
   2. Creates a BooleanVar to store the user's choice.
   3. Defines the callback functions that will be called when the buttons are clicked. These functions call the appropriate
      methods (e.g., `self.yes()`, `self.no()`, `self.reset()`) and set the user_choice variable accordingly.
   4. Configures the buttons to call the respective callback functions.
   5. Starts the main event loop using `self.gui.mainloop()`.
   6. Disables the "yes" and "no" buttons after the user has made a choice.
   7. Returns the user's choice as a boolean value.

Note: The specific details of the GUI implementation, such as the actual buttons and their configuration, may depend on
the underlying GUI framework used.

Example usage:
```

```
    get_user_choice()
"""
```

Creator.yes(self)

```
"""
Set last_user_input to "yes".

This method updates the last_user_input attribute to "yes" and displays a
    message indicating the change.

Example:
    gui_instance = MyGUI()
    gui_instance.yes()
    # The last_user_input attribute is updated to "yes".
"""
```

Creator.no(self)

```
"""
Set last_user_input to "no".

This method updates the last_user_input attribute to "no" and displays a
    message indicating the change.

Example:
    gui_instance = MyGUI()
    gui_instance.no()
    # The last_user_input attribute is updated to "no".
"""
```

Creator.reset(self)

```
"""
Reset last_user_input and provide status.

This method resets the last_user_input attribute to "reset", displays a
    reset message using the output_text
method, and confirms the change with a print statement.

Example:
    gui_instance = MyGUI()
    gui_instance.reset()
    # The last_user_input attribute is reset to "reset", and the GUI
        provides a reset status.
"""
```

Creator.browse_files(self)

```
"""
Open a file dialog to select a file path.

This method opens a file dialog to allow the user to select a file path. The
     selected file path is then
displayed in the editable box on the GUI.

Example:
    gui_instance = MyGUI()
    gui_instance.browse_files()
```

```
    # The user selects a file path using the file dialog, and the selected
        path is displayed in the GUI.
"""
```

Creator.update_input_file(self)

```
"""
Update the current input file and associated data.

This method updates the current input file based on the path entered in the
    GUI. If no file path
is provided, an appropriate message is displayed using the output_text
    method. If the provided
file path is different from the current file path, the global_vars object is
     reset, and the new
file path is set as the current file. Additionally, if the file's extension
    matches certain
predefined extensions (such as '.char', '.names', or '.list'), the lines
    from the file are read
and stored in the current_list attribute.

Example:
    gui_instance = MyGUI()
    gui_instance.path_text.set("data.txt")
    gui_instance.update_input_file()
    # The current input file is updated to 'data.txt', and associated data
        is adjusted.
"""
```

Creator.generate_word(self)

```
"""
Generates a word based on a probability matrix and offers the option to
    append it to a file.

Args:
    None.

Returns:
    None.

Raises:
    None.

This method generates a word using a probability matrix and prompts the user
     whether to append the generated word
to a file.

The method performs the following steps:
   1. Prints a message indicating that a word is being generated.
   2. Updates the input file.
   3. Generates a probability matrix based on the current list.
   4. Prints the generated probability matrix.
   5. Enters a loop that continues until the last user input is "reset".
       a. Generates a word using the current probability matrix.
       b. Outputs the generated word.
       c. Asks the user if they want to append the word to the file.
       d. Retrieves the user's choice.
       e. If the user chooses "yes", the word is appended to the file.
       f. If the user chooses "no", a message is outputted indicating that
           the word was not appended to the file.
```

```
        g. If the user's choice is neither "yes" nor "no", the loop continues.

Note: The specific details of how the probability matrix is generated and how the word is outputted may depend on
the implementation of the methods used within this method.

Example usage:
    generate_word()
"""
```

Creator.run(self)

```
"""
Runs the main GUI.
"""
```

References

[1]