Understanding the RoarTV2.pyw program

This guide explains how the RoarTV2.pyw script works. It is designed for streamers who want to automatically cycle through short .mp4 videos from different folders, occasionally inject “holiday” or “pet” videos and optionally report what’s playing to Discord. You don’t need to be a programming expert to follow along – the explanations focus on the concepts so that you can customise the behaviour later.

For background, Python comes with a variety of built-in modules. The tkinter module provides a standard way to create graphical user interfaces (GUI) in Python; it wraps the Tcl/Tk toolkit and is available on most operating systems 【417276314140738†L70-L83】. The os module contains functions for interacting with the file system, such as os.walk(), which **recursively generates directories and files** underneath a given folder 【134201306840430†L3317-L3327】. The random module supplies pseudo-random numbers; for example, random.randint(a, b) returns an integer between a and b, inclusive 【698501478696497†L195-L199】. The shutil module provides high-level file operations such as copying files 【809944871430822†L64-L66】. You’ll see these modules used throughout the program.

1. Imports and constants

At the top of the script, several modules are imported:

- **random** – used to pick videos and folders at random.
- **shutil** – used to copy or move files.
- **time** – provides functions such as sleep() to pause execution.
- **tkinter** (aliased as tk) and **ttk** – allow you to build the graphical user interface. The script uses frames, labels, buttons, check-boxes and a drop-down menu to let you select folders and configure settings.
- **discord** and **asyncio** – used to talk to the Discord API. asyncio provides an *event loop*, which runs asynchronous tasks and callbacks 【237400218830360†L84-L88】. The Discord library uses this loop to connect to Discord servers and send files without freezing the GUI.
- **requests** – a third-party library for HTTP requests; here it is used to send files via a Discord webhook.
- **json** and **logging** – used for saving presets and recording application events.
- **os** – used for directory operations. The program changes its working directory to the folder containing the script (os.chdir), and defines several file paths (for example LIVE_FOLDER, UPDATE_TXT_PATH, BAD_FILE) that are used throughout the program.

There are also a few constants you might want to change:

- **DISCORD_CHANNEL_ID** and **DISCORD_WEBHOOK_URL** – identify the Discord channel and webhook used to send notifications. If you run your own bot, replace these values with your own channel ID and webhook URL.
- **LIVE_FOLDER** – the directory where current files (such as roarTV2.mp4) are written. If you store your videos elsewhere, update this path.
- **PETS_FOLDER** – the root of the "pet cameo" videos. During each cycle the program will pick one random pet folder containing .mp4 files.

2. The DiscordUploader class

Sending files to Discord requires an *asynchronous* connection so that the GUI remains responsive. The DiscordUploader class wraps the discord.py client and exposes methods for uploading a file:

1. **Initialisation** – when an instance is created, a new Discord client is set up with default permissions (discord.Intents.default()). The code creates a separate event loop using asyncio.new_event_loop() and runs the client in a background thread. When the client logs in (on_ready), it sets an internal flag to signal that uploading is possible.
2. **_upload_file()** – an asynchronous coroutine that waits until the client is ready, obtains the channel by ID and sends the file. If the channel cannot be found (for example if the ID is wrong), it prints an error.
3. **upload_file(file_path)** – a regular method you call from the GUI. It schedules the coroutine on the event loop using asyncio.run_coroutine_threadsafe().
4. **upload_to_discord(file_path)** – a simplified method that uploads files via a webhook using the requests library. It checks that the file is smaller than the 50 MB limit and posts a multipart message containing the video and some context. If the webhook is rate limited (HTTP 429), it waits before retrying.

If you have your own Discord bot token you can modify the class to authenticate directly instead of using a webhook. To disable Discord notifications completely, you can avoid calling upload_file() and upload_to_discord().

3. The RandomMP4Mover class – user interface and state

The bulk of the program lives in the RandomMP4Mover class. When you launch the script, the __init__() method constructs a window with controls for choosing folders, adjusting settings and starting the playback loop. The class also manages state variables such as currently enabled folders, rotation intervals and history.

3.1 Building the GUI

The GUI is built using tkinter widgets. Frame widgets group related controls; Label widgets display text; Button widgets allow you to trigger actions; Checkbutton widgets let you enable or disable individual channels; and Combobox from ttk provides a drop-down list of presets. Even if you have never used tkinter before, remember that it is the standard GUI toolkit for Python applications 【417276314140738†L70-L83】.

The main elements include:

- **Preset selection** – a frame at the top of the window displays a drop-down list of preset names (e.g. “Normal”, “Christmas” or “OPEN”). Buttons next to the list allow you to **load** a preset (restore previously saved settings) or **save** your current configuration to a preset file. Presets are stored as JSON files in the presets subfolder under LIVE_FOLDER.
- **Rolls and cycle count** – below the presets are labels showing the results of a 1–256 and a 1–8 random roll (used for some custom behaviour) and the number of active folders in the cycle. These labels are updated whenever a video is played.
- **Folder selection and priority** – a row of controls lets you pick your main directory (the folder containing sub-folders of videos). Next to the “📂 Main” button is a **Priority** drop-down. Here you can set a specific folder as the priority channel or ask the program to choose one at random. A **No Priority** check-box allows you to ignore the priority channel entirely.
- **Priority insertion** – the **Priority Channel (every Nth)** field lets you specify how often a video from the priority channel is inserted into the rotation. For example, if it is set to 4, then after playing three normal folders the fourth slot will be a priority video.
- **Holiday chance** – a similar setting called **Holiday Chance (1 in N)** controls how frequently a “holiday” channel is inserted. A holiday channel is a random folder chosen from E:\\Holiday. Only one holiday video can be played per full cycle of normal folders.
- **Playback controls** – “▶️ Start” and “⏹️ Stop” buttons start or stop the main loop. A **History** menu allows you to view or export recently played videos, and you can upload the previous video to Discord by pressing F16.
- **Cycle settings** – you can adjust the **Rotation Interval** (the number of seconds each video plays) and see an **Estimated Cycle Time** label. The estimated cycle time takes into account the number of enabled folders, any extra slots for priority videos and your chosen interval.
- **Channel toggles** – the bottom section lists every sub-folder beneath the main folder. Each entry has two check-boxes: one to **enable/disable** the channel and one to indicate whether it should be played **twice** in a cycle (x2). The list is scrollable so it can handle dozens of folders.

3.2 Internal variables

To keep track of its state, the class stores numerous attributes:

Name	Purpose
self.folder_path	The main directory containing sub-folders of .mp4 files.
self.channel_vars	A dictionary mapping each sub-folder path to

Name	Purpose
	a pair of
BooleanVar objects (one for enabled, one for double).	
self.rotation_interval	A tk.IntVar representing how many seconds each video
plays. Changing this updates self.interval.	
self.priority_folder	The folder chosen as the priority channel.
self.priority_every_n	An integer (stored in a tk.IntVar) controlling how
often a priority video is inserted.	
self.holiday_chance	A tk.IntVar controlling the probability of a holiday
video (1 in N).	
self.full_subfolder_cycle	A list of currently enabled leaf folders.
self.all_files	A mapping from folder paths to the list of .mp4 files
contained within.	
self.used_files	A set of filenames that have been played during the
current cycle, so they aren't immediately repeated.	
self.played_history	A list of the most recent videos played, used for
history and Discord uploads.	

Additionally, various counters (like self.cycle_index, self.priority_counter and self.bad_count) keep track of progress through a cycle and how many times the "bad" button has been used.

3.3 Presets and saving state

The **load/save preset** methods allow you to remember your favourite configuration. When you save a preset, the program writes a JSON file containing:

- which folders are enabled and whether they play once or twice;
- the path to the main folder and the current priority folder;
- your chosen values for priority_every_n, rotation_interval and PETS_FOLDER.

When you load a preset, the program reads this JSON file, updates the GUI controls and repopulates the channel list. Loading a preset also invokes refresh_file_list() to rebuild the channel list (described below).

3.4 Selecting the main folder – select_folder() and refresh_file_list()

When you click the 📂 **Main** button, the program calls select_folder() which opens a file dialog so you can choose a directory. The chosen path is stored in self.folder_path and displayed in the label at the top of the window. The method then invokes refresh_file_list() to discover sub-folders.

Discovering sub-folders

refresh_file_list() clears the existing channel list and uses os.listdir() to list the immediate contents of self.folder_path. For each entry that is a directory (and doesn't contain "bad" in its name), it calls find_leaf_subfolder() to locate the **deepest folder** that contains no sub-directories. This ensures that even nested structures (e.g. MainFolder/Category/Channel1) are flattened so that Channel1 appears as a selectable channel. This uses os.listdir() and os.path.isdir() repeatedly until a folder with no subdirectories is found.

For each leaf folder, the method creates two tk.BooleanVar objects to represent whether the folder is enabled and whether it should be played twice. It then constructs a row in the scrollable frame containing checkboxes for these options. The entry is added to self.channel_vars, and the list of files in the folder is cached in self.all_files. If the folder belongs to the PETS_FOLDER tree, the "enabled" box is forced on and cannot be unchecked.

Updating enabled folders and the cycle estimate

After building the list, refresh_file_list() calls update_enabled_folders() which constructs self.full_subfolder_cycle, a list of the currently enabled folders. It also updates the Cycle Folders label and resets counters. The method uses os.listdir() and os.path.isdir() again to ensure each folder still exists and to populate self.all_files with the names of .mp4 files.

Finally, update_cycle_time_estimate() calculates how long it will take to play one full cycle of normal folders. It counts the number of enabled folders and adds extra slots for priority videos depending on self.priority_every_n. The estimate (in seconds and minutes) is displayed so that you know roughly how long a full rotation will take.

3.5 Rolling numbers – update_shantae_number()

This method updates a special file called shantaeNumber.txt and displays the results of two random rolls. First it chooses a random number between 1 and 256 using random.randint(). If the result is exactly 256, it writes "nu" (a special marker) to the file and resets the second roll. Otherwise it displays the 256-roll result and then picks a random number between 1 and 8, making sure it isn't the same as the previous roll. The chosen numbers appear in the GUI.

If you don't care about these numbers, you can ignore this method. Alternatively, you could repurpose it: for example, you might roll a die to decide which theme or overlay to use for your stream.

3.6 Monitoring triggers – monitor_trigger_files()

The program uses two special files (bad.txt and early.txt) to receive external signals. monitor_trigger_files() runs in a background thread and checks these files once per second while the main loop is running. If it detects that bad.txt was modified and contains a 1, it starts a new thread to handle the "bad" channel (see below). If early.txt changes, it calls early_change() which interrupts the current waiting period so that the next video starts immediately. This mechanism allows you to create hotkeys or other programs that control when the video advances or when you want to discard the currently playing file.

3.7 Choosing the next video – play_next_video()

play_next_video() implements the core logic of selecting which folder and file should play next. The algorithm works as follows:

1. **Priority mode:** If a priority folder is defined, contains .mp4 files, and you haven't disabled priority with the "No Priority" checkbox, the method decides whether to play a priority video. It keeps a counter (self.priority_counter) and compares it to the setting self.priority_every_n. When the counter reaches n, the next video will come from the priority folder; otherwise a normal folder is used.

2. **Normal mode:** To pick a normal folder, the method calls get_next_folder(), which cycles through self.full_subfolder_cycle and returns the next folder with available files. If a folder runs out of unused files, it refreshes its list from disk. The code subtracts files you have already watched (self.used_files) so that videos are not repeated until every file in that folder has been played once. After selecting a folder, it picks a random file from that folder using random.choice().

3. **Common tasks:** Once a file is chosen (whether from the priority folder or a normal folder), the method copies it to LIVE_FOLDER/roarTV2.mp4 using shutil.copy2(). Copying the file rather than moving it preserves the original file in its source folder. It then updates update.txt (a text file listing all played files), writes the current channel and file name to channel.txt (and backs up the previous entry to lastchannel.txt), and updates the history list for the GUI and Discord.

4. **Waiting:** The method looks up whether the current folder has the "x2" option enabled. If it does, it doubles the waiting period so the video plays twice as long. It then sleeps in one-second increments, monitoring self.change_event so that the wait can be interrupted by early.txt or bad.txt events.

When the method finishes, it returns the folder that was played so that move_random_mp4() can update cycle counters.

3.8 Playing special channels – play_special_video()

Special channels are used for **holiday** and **pet cameo** videos. play_special_video() takes a root folder, finds a leaf sub-folder, picks a random .mp4 file and copies it to the live video location. It then updates the same state variables as play_next_video(). A holiday video resets self.holiday_sent_this_cycle so that only one holiday can occur per cycle. This method returns the folder used so that move_random_mp4() can skip counting it towards the normal cycle.

3.9 The main loop – move_random_mp4()

Once you press the **Start** button, a separate thread runs move_random_mp4() continuously until you press **Stop**. The loop follows this sequence:

1. **Holiday injection:** If there are normal folders and a holiday has not yet been played this cycle, it draws a random number between 1 and the value of self.holiday_chance. If the result is 1, it chooses a random folder under E:\\Holiday and plays a holiday video using play_special_video().

2. **Normal playback:** Otherwise, it calls play_next_video(). The returned folder is added to self.used_subfolders to prevent the same folder from being used twice in the same cycle.

3. **Cycle completion:** When the number of played normal folders matches the number of enabled folders (len(self.full_subfolder_cycle)), the cycle resets. It clears self.used_subfolders and self.holiday_sent_this_cycle, so a new holiday can occur in the next cycle. It then plays a **pet cameo** video by calling get_random_pet_folder() and play_special_video().

This loop continues until you click **Stop** (which sets self.running to False).

3.10 History, export and uploading

The program keeps a history of the last 100 videos played in a file called history_paths.txt inside LIVE_FOLDER. The **History** menu exposes two actions:

- **View History** – opens a new window listing the last ten videos in reverse order (most recent at the top). You can left-click a filename to open the video directly in your operating system, or right-click to upload that video to Discord using the DiscordUploader. If the file is missing (for example because you deleted or moved it), the entry appears in grey.

- **Export History** – prompts you to save the entire history_paths.txt file elsewhere on your computer.

You can also send the previous video to Discord at any time by pressing the **F16** key. The method send_last_channel_to_discord() opens history_paths.txt, retrieves the second-to-last line (the previous video) and calls upload_to_discord().

3.11 Handling “bad” videos – bad_channel()

If you decide that the current video should never be played again, you can trigger the “bad” channel by creating or editing bad.txt to contain a 1. The background monitor will call bad_channel(), which moves the currently playing video to a bad sub-folder within your main folder. It also increments a counter stored in badnumber.txt. The method removes the file from self.all_files and self.used_files, resets the current file variables and writes “0” back to bad.txt so that the monitor stops triggering. After a short pause, normal playback resumes.

4. Customising the program

The script is meant to be flexible. Here are some common customisations you might consider:

- **Changing default folders** – adjust LIVE_FOLDER, PETS_FOLDER and LOG_FOLDER near the top of the script to point to your preferred locations. Make sure these directories exist.
- **Adding or renaming presets** – update self.preset_names in __init__() if you want different preset names. Preset files themselves live in the presets sub-folder.
- **Adjusting holiday and priority behaviour** – the holiday chance defaults to 1 in 64; you can change the initial value of self.holiday_chance or call the **Set to 1/3** button during testing. Similarly, the priority insertion defaults to every 4th slot (but is automatically adjusted to match the number of enabled folders for one pass). You can modify these values in the GUI or change the default values in the code.
- **Disabling Discord** – if you don’t want any Discord integration, remove or comment out calls to self.discord.upload_to_discord().
- **Supporting other file types** – currently the program only looks for .mp4 files. If you want to support .mkv, .avi or other formats, modify the checks that test for f.lower().endswith('.mp4').
- **Extending history** – the history window shows the last ten entries, and the file stores the last 100. Adjust these numbers in show_history() and the code that truncates history_paths.txt.
- **Changing the GUI appearance** – you can change fonts, colours and icons by editing the parameters to Label, Button and other widgets. Remember that ttk themes offer a modern look on most systems.

5. Summary

The RoarTV2.pyw program is a feature-rich tool for automatically rotating through folders of videos, injecting special content and interacting with Discord. Behind the scenes it uses standard Python modules like tkinter for the interface, random for selecting elements, os for traversing directories【134201306840430†L3317-L3327】, and shutil for copying files 【809944871430822†L64-L66】. It also shows how to integrate a graphical application with asynchronous networking via asyncio【237400218830360†L84-L88】. By understanding each section of the code and the purpose of its variables, you can confidently customise the program – whether by changing folder paths, tweaking the probability of holiday videos or disabling Discord uploads – and adapt it to your own streaming workflow.