**TagOS Documentation**

Hello! I’m Aston (President of RHUL HvZ in the UK from 2021-2022) and I’d like to present you with TagOS (previously known as Tagger but now expanded with a bunch more functionality); the solution to all your HvZ needs. I’ve developed this solution to address the need for tracking tags, providing an easy space for teams to communicate as well as allowing moderators easy tools to solve problems. It uses the Discord.py API to edit roles and write to a database so that each game is contained within your Discord server for your event. In this documentation, I’ll go over the server architecture needed and how TagOS works, including the commands list. Beyond any specifications laid out here, feel free to organise your server as you see fit.

This solution is entirely open source and available on GitHub, so if you need any further updates for your games don’t hesitate to add them!

**Section 1: Setting Up A Bot.**

The source code for TagOS is included in this package, however the token that it needs to register as a bot is not. This is because the token is unique to the instance of TagOS that we use at RHUL to run our games. You **MUST** set up your own bot token for TagOS to work in your games. One instance of TagOS can cover many games, but only one game at one time. This is because the nature of the player database means that an instance of TagOS with the same token writes to the same database – which will cause a total nightmare if it’s used in two games at once.

To set up a Discord bot, see the guide here: <https://discordpy.readthedocs.io/en/stable/discord.html>

You will need to place the token into the file token.txt in the TagOS folder. In terms of permissions, you will need to give Tagger the “Administrator” permissions from the Discord developer portal and whatever the highest role on your server is (Administrator / Committee / Moderator etc).

You will also need to make sure that your server is set up to allow bots to function, as some servers are not. If you encounter any problems with commands not being read, this may be why.

**Section 2.1: Server Architecture.**

TagOS needs to be able to recognise certain channels to work properly and these channels need to be exactly as specified. I’ll cover what channels, categories and roles need to be called, as well as the permissions they need here. I advise you to have general channels, announcements etc but I won’t cover those here as they are not essential to TagOS function.

**#join:**

#join is a public channel that everyone who wishes to play needs to be able to see. It allows people to enter themselves into the game using the .join command (Only usable in this channel and covered in the command section).

**Game Channels Category:**

The first category contains general game channels. The only two that TagOS uses to work are bounty-set and bounty-wall. Bounty-set must be public (restricted to humans and zombies), and bounty-wall must be read-only for players.

In bounty-set, the commands .bounty and .reward can be used. The output from these will be reposted in the bounty-wall channel that is read-only so that they do not get buried under other messages. The bounty functionality in general is optional and you can skip this step if you wish.

**The “Humans” Category:**

This is where all the human announcements as well as general voice and text chat occur. The only channel TagOS needs here is a text channel called “human-chat”. This is where tags will be announced to their other humans, letting them know when other humans get tagged.

Graphical user interface, text, website

Description automatically generated

**The “Zombies” Category:**

This is where zombies get their announcements, chat and strategize. Much like the above category, the only channel TagOS is concerned with here is “zombie-chat”. This is because it is the only place that zombies can run .tag to convert a human they have tagged into a zombie.

**Graphical user interface, application

Description automatically generated**

**Miscellaneous Category:**

This category is non-essential but recommended for housekeeping. It contains a channel for testing the bot and diagnosing any issues as well as a channel to run commands in that are private to eyes of players. I recommend nominating the most tech savvy member (or members) of your team to take ownership of these channels.

**Section 2.2: Server Roles.**

TagOS requires a number of roles to exist in the server to function properly. Namely, “Human”, “Zombie”, “Moderator”, “Committee”, “Bot Dev”, “Spectator”. These roles must be made exactly as set out here, otherwise TagOS will not recognise them. All these roles need the “Use application commands” permission in the role settings on your server.

**Human:**

The default role when people join the game. Represents a human with a braincode (covered later) that is yet to be tagged. This role should be able to see the human category and not the zombie one.

**Zombie:**

Zombie is the inverse of human, it represents a person who is out to add to the horde. Their braincode has already been redeemed. This role should be able to see the zombie category and not the human one.

**Moderator:**

Moderators run our games and as such have enhanced permissions. They are not admins, but they do have more power than players. It is up to you how much power you feel the moderators should have. Members with this role can run more commands than humans and zombies.

**Committee:**

Committee (at RHUL) are our elected leaders of the student group, and as such have the highest amount of power (Administrator rights) on our server. Give this role to the highest authority on the server. Members with this role can run ALL the commands that TagOS has. Even if you don’t have a committee, some commands require this role to work. You can either change the name of the role in the code or give this role to your version of a committee.

**Bot Dev:**

Bot Dev is a role given to me and is for debugging or maintenance / improvement of the bot. It allows you to run all the commands in the bot as well as access the test and debug channels.

**Spectator:**

The lowest priority role, this is just for people that wish to watch the game through the tag logs / text chats etc.

**Section 3.1: Player Commands.**

TagOS exists as a library of commands that manipulate a database. These commands require roles to run, as some (if misused) could wreck a game. Player commands are the most frequently used as well as the least powerful.

The command prefix for TagOS is “.”; for example to join the game you would run .join X Y Z

**.join:**

The most commonly used command, .join puts you into the game. It generates you a braincode (a 3 word unique code representing your humanity ,when tagged you must give this to a zombie to tag you), assigns you human and opens up human chat for you. This command can **ONLY** be used in the #join channel. The command syntax is as follows:

.join Firstname Lastname StudentNumber

Practical example: .join John Smith 12345678

Note that if you do not have a student number (e.g in a game not held in one university) then you can use any number, we only use student number to keep track of members.

In relation to braincodes, humans must keep them secret until tagged or they can be turned into a zombie from the comfort of their own home.

**.check\_braincode:**

Privately messages you your braincode. Used if a braincode is not sent or is lost and needs to be sent again.

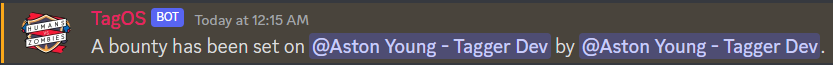
**.bounty:**

This command sets a bounty on a player which is announced publicly. The output is posted in #bounty-wall.

Practical example:

Graphical user interface, website

Description automatically generated



**.reward:**

Used in tandem with .bounty. Sets a reward for the bounty, should that player be tagged / stunned. Output is also posted in #bounty-wall.

Practical example:

Graphical user interface, text

Description automatically generated

Graphical user interface, website

Description automatically generated

**.how\_many\_zombies:**

Outputs the current number of players registered as a zombie in the database.

**.how\_many\_humans:**

Outputs the current number of players registered as a human in the database.

**.tag:**

Used by a zombie to turn a human into a zombie. Once a human is tagged, the human will give their 3 word braincode to the zombie who tagged them. The zombie will then go to #zombie-chat and type the following (if the braincode is onetwothree):

.tag onetwothree

This will then convert the human with the braincode onetwothree into a zombie, remove their access to human chat (unless they are a moderator / admin or have elevated perms) and announce their demise to both human and zombie chat.

This command will not work outside of zombie chat.

**Section 3.2: Moderator Commands.**

The commands in this section require elevated permissions in the form of the “Moderator” / “Committee” / “Bot Dev” roles to function. Without the role, the commands will not execute.

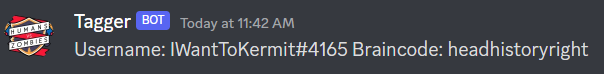
**.get\_braincode:**

Retrieves the braincode of a single player using their username.

Practical example:

Text

Description automatically generated

****

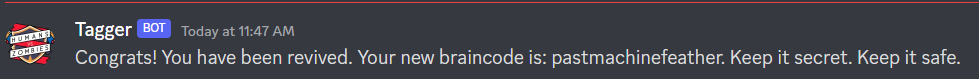
**.revive:**

Revives a zombie - restoring them to humanity, generating a new braincode and removing them from zombie chat whilst adding them to human chat again.

Practical example:

Text

Description automatically generated

****

**.revive\_all:**

Revives every zombie, restoring everyone to human. Useful for restarting after a wipe.

**.delete\_player:**

Removes a player from the game entirely using their username.

Practical example:

Graphical user interface, text, application

Description automatically generated

**Section 3.3: Admin Commands.**

These are the most powerful commands and require either the “Committee” or “Bot Dev” roles to prevent misuse.

**.check:**

Sends all data about all players in the database to the channel you use it in. This is a LOT of output and so I highly recommend only using this command in the debug channel.

**.deroll\_all:**

Removes all human, zombie and spectator roles from everyone in the server.

**.reset\_database:**

Resets the player database entirely to a fresh copy.

**.end\_game:**

Ends the game by resetting everything. Runs .reset\_database and .deroll\_all. Only use this at the end of a game, not after a wipe!

**.k:**

Logs TagOS out. You will have to restart TagOS to run another game. Run this after .end\_game.

**Section 4: Core Components.**

This section covers the individual components of TagOS, should you be curious or want to make adjustments / alterations / improvements (go nuts with it!).

**Folders:**

backupdatabase – Contains unused template for the player database. Do not delete or you won’t be able to start a new game.

cogs – Contains the files for player, moderator and admin commands.

logs – Outputs the logs of each time TagOS is run. Useful for debugging.

**Folders / Files:**

bot.py: Contains most of the code to initialize and run the bot. Establishes permissions, loads stored commands, and puts the bot online.

death\_messages.txt: Contains a store of strings that are used to announce tags. One is picked at random to be put at the end of each tag announcement in human chat, for example “John Smith has been tagged. No one saw that coming!”. You can edit these or add more if you have any particularly good ones!

player\_database: Contains all the data needed for Tagger to run. Names, roles, IDs, braincodes etc for each player.

Backupdatabase: Folder that contains a fresh database to be cloned into the used one at the start of every game. Do not edit!

TagOS\_Documentation.docx: This documentation!

token.txt: Your unique bot token.

words.txt: Contains around 1000 words, 3 of which are randomly selected to form each braincode. You can add or remove these as you wish.