NAME

go-gma-server – GMA battle grid map server

SYNOPSIS

go−gma−server [**−h**]

go−gma−server [**−help**]

go−gma−server [**−init−file** *path*] [**−log−file** *path*] [**−mysql** *database*] [**−password−file** *pass-file*] [**−port** *port*] [**−save−interval** *mins*] [**−sqlite** *path*]

DESCRIPTION

The individual **mapper**(6) clients used by players in a game may keep in contact with one another so that they all display the same contents. A change made on one client (moving a creature token or adding a room, say) appears on all the others. This is accomplished by starting a **go−gma−server** process and having all of the **mapper** clients connect to it via their **−−host** and **−−port** options.

Once connected, the server will send an initial greeting that may define a list of player character tokens to appear on the **mapper** context menus, or any other useful information the clients need to have at startup time. It may, at the GM's option, even initialize the client to show the full current game state.

From that point forward, the server is a simple echo relay between the clients, so they communicate with each other via the service. The server also tracks the commands it sees, so that it maintains a notion of the current state of the game. Clients may re-sync with the server in case they restart or otherwise miss any updates so they match the server's state. The server may respond directly to some client queries (e.g., "**AI?**") if it knows the answer rather than referring the query to the other clients.

To guard against nuisance or malicious port scans and other superfluous connections which don't proceed with the normal activity, the server will automatically drop any clients which don't authenticate within a few poll intervals. (In actual production use, we have observed some automated agents which connected and then sat idle for hours, consuming server resources.)

This is a re-implementation from scratch of the GMA server in the Go programming language. The original Python implementation is documented in **mapservice**(6).

OPTIONS

The following options control the behavior of **go−gma−server**.

−h,**−help**

Print a usage summary and exit.

−init−file *init-file*

Each line in *init-file* will be sent verbatim to each client upon connection to the server. The lines of this file must therefore be valid mapper protocol commands as documented in **mapper**(6). It is permitted for the JSON data portion of the command to span multiple lines. In this case, all lines after the first must be indented, and the final closing brace unindented on a line by itself. For example, the line

AC {"Name":"Fred","ObjID":"PC123","Color":"red"}

could also be written in the file as:

```
AC {
  "Name": "Fred",
  "ObjID": "PC123",
  "Color": "red"
}
```

The original version of the server only read this file at start-up, repeating its contents from memory to all clients; this version reads the file every time a client connects, allowing changes to be made to the initialization file which take effect immediately.

The following special commands may appear in this file. These are not sent to the clients directly, but trigger other setup events in the server itself:

SYNC Enables sync-on-connect mode. With this set, the server will automatically perform the effects of a **SYNC** command for each client as it connects (after authentication if that is required).

LOAD *filename* This directive is deprecated. It is not supported by the Go version of the server.

Any comments (i.e., lines starting with "**//**") are sent immediately. All other commands are sent after successful authentication, if authentication is enabled.

−log−file *log-file*
Append a record of server actions and diagnostic messages to the specified file. By default, this log is sent to the standard output.

−password−file *pass-file*
If this option is given, the server will require clients to authenticate with a valid password. The first line of *pass-file* is the shared password expected to be used by all players. If a second line is present, it provides a GM password which will grant GM privileges to a user authenticating using it.

Subsequent lines, if any, give specific passwords for individual users. These have the form
*username***:***password*
Any login for the given *username* must use the individual password for that user to successfully authenticate to the server. Any *username* not listed in the password file (other than "**gm**") will successfully authenticate if the general-use shared password (the first line of the password file) is given.

−port *port*
The service will accept incoming connections on the specified TCP port. The default is 2323.

−save−interval *mins*
If the **−mysql** or **−sqlite** option is given, this sets the frequency (in minutes) at which the game state is saved to the database. The default is every 10 minutes.

−sqlite *path*
Open the sqlite3 database *path* as the persistent record of game state and history. Any saved game state is read in from this database when the service starts, allowing the game to continue from where it was when the server was stopped. The service will periodically save its current state to this database.

−mysql *database*
Analogous to the **−sqlite** option, this uses a MySQL database for persistent storage. *This is not currently implemented.*

SECURITY

The authentication system employed here is simplisitic and not sufficient for general application security, but for the sake of inconveniencing cheating at our little game amongst friends, it will suffice for our narrowly-defined purposes where the stakes are negligibly low.

It is intended just to discourage cheating at the game by looking at spoilers or direct messages intended for other users, not for any more rigorous protection.

The main weakness of the system is that passwords are stored in plaintext on the server, which means it is critical to secure the password file and the system itself. Caution your players to use a password for the mapper that is different from any other passwords they use (which should be the password practice people observe anyway). A breach that reveals passwords from the server's file would then only allow an imposter to log in to your map service, which admittedly is more of an inconvenience than a serious security issue, assuming you use your map server just for playing a game and not for the communication of any sensitive information.

Don't use the GMA mapper server for the communication of sensitive information. It's part of a game. Just play a game with it.

SIGNALS

The map service responds to the following signals while running. Note that this is different from the behavior of the original server.

HUP This signal causes the server to save state if needed and shut down gracefully. No new connections will be accepted, but the server will wait for existing ones to terminate before shutting down.

INT Emergency shutdown. Just like the graceful shutdown caused by a HUP signal, but forces all existing connections to immediately terminate.

USR1 This signal causes the server to dump a human-readable description of the current game state on its standard output.

USR2 This signal causes the server to save its current state if needed.

SEE ALSO

gma(6), **mapper**(5), **mapper**(6), **mapservice**(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

COPYRGHT