

A Multi-User Dungeon, MoD (MUD on Discord)

Platform: Discord

Hosting: AWS EC2, or any OS with a running a database instance

Technologies used: Discord Bot API, DynamoDB, EC2 (maybe)

If you are familiar with the concept of a MUD or Discord, feel free to skip those sections.

Description of a MUD, and the Problem with them.

A MUD is a **non-graphical/text-based** online multiplayer game. Traditionally, MUDs are accessed through the telnet protocol, and feature role-playing and stat-based gameplay. Even today, there are MUDs which still see service and active userbases (*Achea, Aardwolf, BatMUD, Discworld*). **These are not collectivized text-adventure games, but rather worlds that players explore with no definite end goal, but have many activities that the player may choose to engage in, often facilitated by in-game systems.**

Therein lie some problems. Telnet is, by default, not a secure protocol, and while there are extensions that may provide security with telnet use, they are not widely supported. MUDs are also a more technical-to-access sort of system, and this may dissuade users from engaging with them.

Thus this project, to make a MUD that is more secure and accessible to modern users.

Description of Discord

Discord is a Slack-like chat service, host to over 130 million users, whose target demographic is the gaming community. It features the ability to host server instances on Discord's personal cloud (much like how Slack's hosting works), and is used for a wide variety of communication purposes(text, voice, video chat). The following capabilities of Discord are key to this project:

- Supports having multiple chat rooms with designated uses
- API that handle user login/verification(thus maintaining security without needing to implement a login system)
- It's free to use and users are free to add bots to their servers
- Discord servers are capable of handling tens of thousands of users.

What's a bot?

In the context of Discord, a bot is an automated user profile that has controlled access to user chats. Bots are capable of parsing pre-defined commands, and can be given a variety of permissions to act in proxy of an administrator for a server. Bots operate through the use of publicly facing API endpoints that Discord's dev team has made for parsing server data, and users interact with them through commands.

Description of the Idea (MoD)

Use the secure authentication of Discord (which uses HTTPS, 2FA, and IP locking), as well as the platform itself, to give users an interface with which to interact with a Discord bot. This bot will facilitate the same functionality of a traditional MUD, with commands for players to use and interact with. The project will involve learning how to interact with public API, setup and maintenance of a database, and quite a bit of writing.

Why Discord? Why make a MUD bot? Why any of this?

Discord is a very fresh (3 years old) platform, and is constantly growing with a massive userbase. Interacting with, and learning how to use APIs is integral to modern programming.

This is a good opportunity for students to learn how to parse API documentation effectively, and integrate it with their own code.

AWS is at the forefront of cloud technology, and any experience with it prior to graduation is extremely valuable. DynamoDB is within the scope of AWS, and is a very modern non-relational database system, meaning that students would be at the forefront in learning how to use it.

The main demographic of Discord is users interested in gaming. Because MUDs are oriented (generally) around role-playing and other RPG (role-playing game) elements, it is very likely that many users would be interested and will use MoD.

The following features should be considered necessary for a *minimum viable product*:

- multiple rooms (chat rooms in the discord server) which will be cordoned according to the in-game location of the player. It is possible to prevent a user from joining, or even seeing rooms based on roles (i.e. permissions) on Discord. These rooms should each have a separate functionality.
 - a room in which players spawn
 - a room in which a shopkeep NPC(non-player character) is present and interactable. Players may acquire items from this NPC, such as weapons or potions
 - a room in which a type of monster(like a goblin or slime) may spawn for the purpose of demonstrating player combat.
- command system that players may use to interact with the Bot. Like a traditional text-based game, they would have options such as “Look”, “Talk”, “Attack”, “Help”. These commands would begin with an identifier symbol that indicates it is intended for the bot(a ‘\$’ or ‘?’ or ‘!’ are all common identifiers).
 - example of two players in a room(chat), interacting with the bot:

User1337: \$Look room

Bot: Motes dance in the air, illuminated by rays of light. The rays bounce around revealing a library of ancient stature, and a lone figure sitting in an armchair by a crackling fireplace. There is 1 other player present.

xXUser2077Xx: Yo

User1337: Sup

User1337: \$Talk figure

Bot: the old figure turns and looks at you, “Hello there, blahblahblah”.

- A player stat system (simply level/strength/defense/health for now)
- A small database for storing player inventory, as well as descriptions/attributes of players, objects, rooms, NPCs, and monsters.

Stretch Goals:

- Implementing a quest system
- Implementing more monster types, developing a wider world to explore
- Developing more general lore for the world players explore

- Flesh out player stats more, and by association, monster stats
- Record player logs to determine behavioral heuristics
- Make the system modular/extensible so that others may use it as a foundational tool for creating their own MUDs
- Add a subscription service for players who want extended functionality, or micro-transactions.

In summary:

The goal is to create a more secure, easier to host version of a traditional MUD, using Discord as the player client. Discord was chosen due to the existing userbase being most likely to be receptive to RPG elements, as well as its relevance as a modern platform. DynamoDB was chosen for its newness/relevance, as well as it being non-relational system.