Smithy: A *Monster Hunter: Rise* Discord Bot

1 Introduction

1.1 Discord

In 2015, the game development studio Hammer and Chisel released Discord under the domain name *discordapp.com*. The software's small computation footprint and robust combination of text- and voice-based communication caused it to quickly grow in popularity among video-game focused groups. By the end of 2016, the company reported over 25 million users worldwide, which has since ballooned to over 140 million active users in 2021.[1]

1.2 Monster Hunter

In 2004, the first *Monster Hunter* game was released for the PlayStation 2 as a bid by Capcom to make use of the new features of the console. The series saw gradually increasing popularity until 2018, with the release of *Monster Hunter: World*. Offering somewhat simplified mechanics, *Monster Hunter: World* saw a dramatic rise in popularity for the franchise, selling over 16.8 million units over its lifetime.[2] The most recent release, *Monster Hunter: Rise* has already shipped over 6 million units worldwide since its release in March.[3]

2 Goals and Prior Work

In *Monster Hunter*, players very naturally hunt monsters. After hunting a monster, the players gather parts that can then be used to craft improved equipment and consumables. This equipment is then used to improve the player's success at hunting other monsters, in a repeating game cycle. However, upon progressing through the game, the equipment one crafts often becomes dominated by the skills it provides. These skills are a "perk" system in *Monster Hunter*, and most end-game builds are focused upon optimizing their armor sets to attain certain values of each skill. However, this is easier said than done; the game features five different armor pieces, and each can be outfitted with some set of decorations which add additional skill points. Beyond this, the game also features randomly-generated charms, which are distinctly generated for each player and are frequently the deciding factor in what kind of builds a player can produce.

Naturally, having a computer program to assist in producing such builds is extremely helpful. Prior games have had programs such as *HoneyHunter* and *MHaG* to facilitate this, but these tools begin and end with simulation; they provide no optimization options.[4][5] Additionally, none of these tools have yet been updated to include *Monster Hunter: Rise*.

Monster Hunter has long been a community-oriented game, with parties of up to 4 teaming up to hunt monsters. As a natural consequence of this, many people now use Discord to find communities with whom to play and discuss the game. Thus, if one were to develop an armor set builder for *Monster Hunter: Rise*, it would make a great deal of sense to integrate that tool with Discord itself. This is exactly what Smithy seeks to do, via a Discord bot.

3 Design

3.1 Discord Bot

In addition to allowing users to communicate via text and voice chat, the Discord client offers developers a way to add their own functionality to the program via a *bot*. A bot is simply a piece of software that communicates with users in the same fashion as a user would; in our case, this corresponds to reading chat messages and responding in kind. Each bot is given an OAuth2 URL which it uses to connect to servers; a user wishing to add the bot to their server need only follow the link and approve the bot's access. Once connected, bots communicate with users via the Discord server, sending and receiving messages over HTTPS just like a standard user client.

Bots use a specific character, set by the developer, to denote the start of a bot command. Text immediately following this character tells the bot what method to run, and any other text is treated as arguments to the method. In addition, Discord bots include built-in helper commands that inform the user on how to use the bot.

3.2 Armor Set Optimization

The specific goal of this project is to produce optimized armor sets. The user should be able to provide the program with any number of skills, and a desired max value for that skill. Then, the program should find permutations of the available armors and provided charms which

meet those specifications as best as possible. While the set of possible armor pieces is fixed, charms are generated on a user-by-user basis. Thus, the bot must provide methods to add, remove, and modify charms in the database. These charms should be assigned to each Discord user individually for a very natural, simple user experience.

3.3 Accuracy and Performance

One of the most important considerations for this program is that it is a bot; it should simulate user-user interaction. This means that in almost all circumstances, commands need to be executed extremely quickly. For the complex query that is armor set optimization, we will often prefer results that are quickly generated and fairly optimized over results which are perfectly optimized but have long turnaround times.

4 Implementation

4.1 Packages and Interfaces

The entire program is implemented in Python, using the *discord.py* and *mysql.py* packages. The program primarily works as a liaison between the Discord server and the MySQL database; the commands given by the user are translated into queries, then the query results are formatted as text and printed to the Discord text interface.

Commands given to the discord bot are formatted by prefacing the text with an “!”. The bot offers a variety of methods the the user, falling into 3 main categories: building methods, display methods, and charm management. All of the commands are implemented in the *bot.py* file, which handles initiating the bot as well as establishing its commands.

4.1.1 Building Methods

The first and most important command is the ”!build” command, which produces optimized builds for a given set of skills. The build command makes use of the methods provided by the *queryconstructor.py* file to dynamically produce queries. The program cannot know aprior how many skills the user will want to query by, so the program must adjust the contents of the

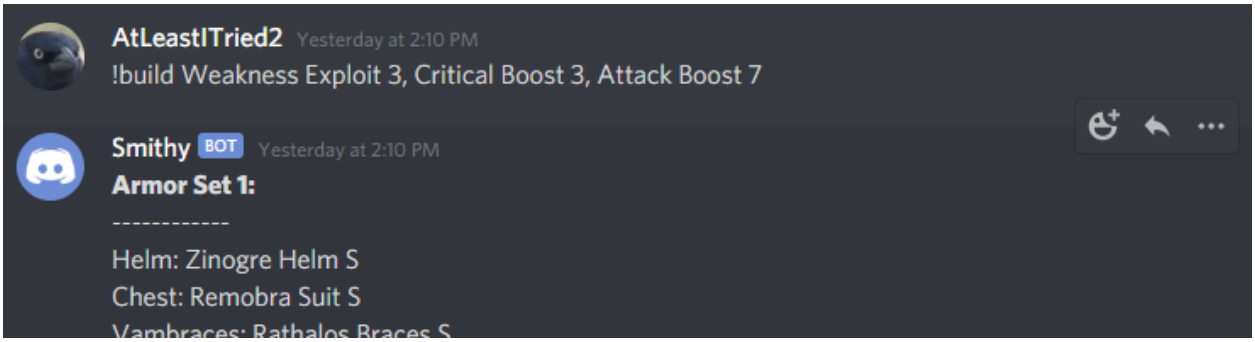

Figure 1: An example of the !build command

query at runtime. Additionally, the "!build" method offers a variety of flags to improve query results. The "-csv" flag causes the program to output the query results as a .csv file, uploaded to Discord. Additionally, the user can employ "-depth" to modify the search depth or "-many" to display additional build results. These flags can allow users to produce better results when the most basic search methods seem insufficient.

4.1.2 Display Methods

The second set of commands handle displaying information to the user. These include listing armor pieces and their properties via the "!show" command, as well as displaying skill information via the "!skill" command. The "!show" command in particular takes a variety of flags

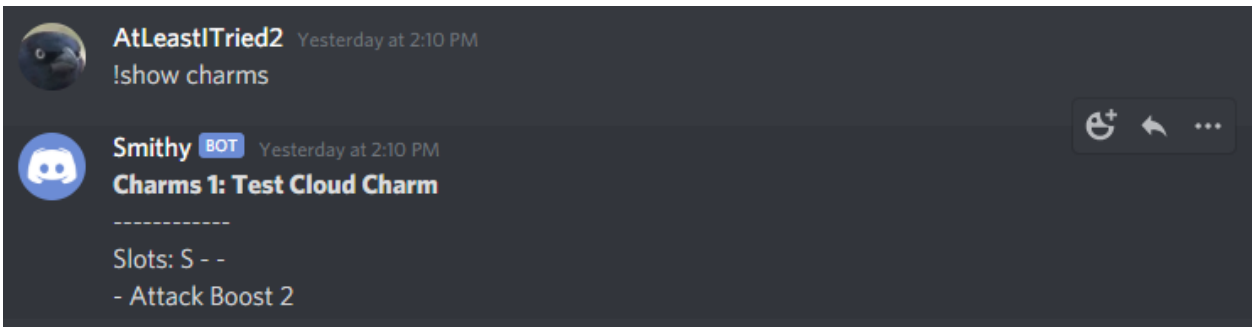

Figure 2: An example of the !show command, used for the charms

to control what results are displayed. The "-count" and "-full" flags allow the user to control the number of results returned, while the "-skill" flag allows the user to limit results to those containing a specific skill.

4.1.3 Charm Management

Finally, all users should be able to easily manage their own set of charms for producing builds. To accomplish this, 3 methods are included; ”!addcharm”, ”!deletecharm”, and ”!modifycharm”. These three operations form the core of the CRUD structure of the charm database, and allow the users to control what charms are used for their build optimization.

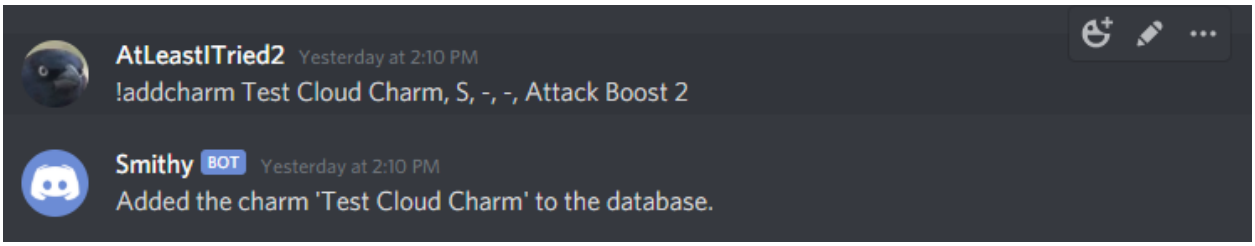

Figure 3: An example of the !addcharm command

4.2 Query Optimization

In general, a query like the one performed by the ”!build” command is very expensive. The problem at hand is a form of combinatorial optimization, specifically a constraint satisfaction problem. Due to the hefty scaling of such problems, we employ a variety of methods to improve query performance.

Firstly, the program only returns armor pieces from each subquery which contain either a point of a desired skill or a decoration socket of the correct size for a decoration of a desired skill. This ensures that the program only evaluates armor pieces which directly contribute to satisfying the requirements for the overall build. Additionally, the program sorts the results by skill points granted, then by number of slots and only returns the first 4 entries. This heuristic decreases the search space to a near-constant amount once joins between tables are being considered, and thus substantially improves performance. The results in this case are still quite accurate, as the armor sets in the game tend to be best built from armor pieces that already have the desired skills, not requiring the use of decorations.

Once each subquery is completed, the resulting tables are jointed together and the armor

pieces are filled out with decorations. Then, the program once again sorts by skill values and returns the results which performed the best. Any armor set which goes over the maximum skill value specified by the user, or by the game itself, is also ignored. Overall, this seems to produce highly accurate results while maintaining the desired performance.

4.3 Database Design

The database is designed primarily with the build-command query in mind; skill and slot values are indexed for fast searching, and each armor piece is given its own table. Each of the armor

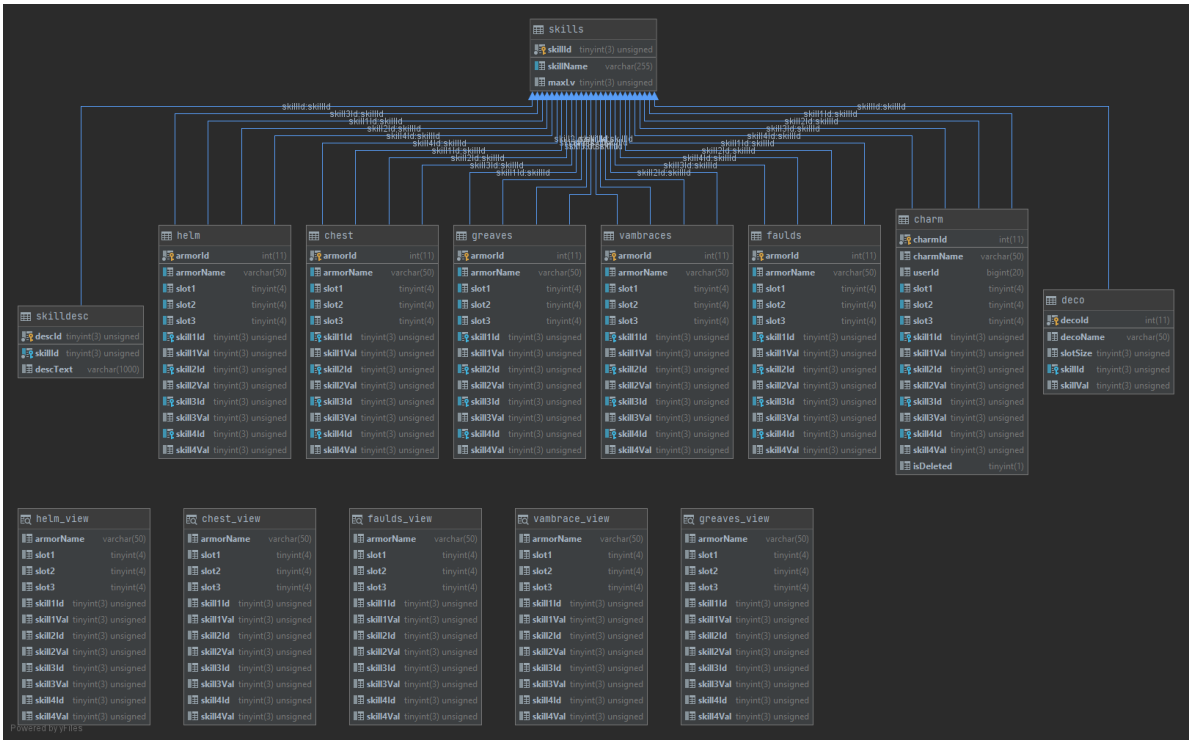

Figure 4: Database Schema

skills has a foreign key binding it to the skills table, so each armor piece can be found quickly and consistently by simply knowing what skills to search for.

References

[1] Discord Revenue and Usage Statistics (2021). Business of Apps. (2021, May 6). https://www.businessofapps.com/data/discord-statistics/.

[2] CAPCOM: Platinum Titles. CAPCOM IR. (n.d.). https://www.capcom.co.jp/ir/english/finance/million.html.

[3] MONSTER HUNTER RISE: CAPCOM. Monster Hunter Portal. (n.d.). https://www.monsterhunter.com/rise/us/topics/itempack/.

[4] Honey Hunters World, 20 Sept. 2020, honeyhunterworld.com/.

[5] "Welcome to MHAG Online." Mhag.info, mhag.info/.