

Chatbot Report - jfa180002

System Description

I wanted this chatbot to be able to look up information on fighting game characters. The information was sourced from a popular website, dustloop.com. This site specializes in fighting games made by a certain developer, Arc System Works titles. Back when I still had passions and hobbies, I used to play fighting games casually. (Like VERY casually. I'm still quite bad at them.) So I thought a quick lookup tool in the form of a chatbot could be useful if you didn't want to wad through a bunch of webpages based on whatever game you wanted to look for. Ideally, this system would support a plethora of characters and games, but due to time constraints with the project and my job search I didn't get to fulfill my vision for this chatbot as I had hoped. Well at least not yet.

This prototype takes all the data from dustloop.com for the newest entry in the Guilty Gear series, Guilty Gear Strive. The data for the knowledge base was especially difficult to wrangle up because I wanted to use my own process for text generation for the chatbot using named entity recognition and tf-idf (more on that later) but this required that I add annotations to the raw frame data in the game. Thus I had to personally annotate and summarize over 70 individual moves and what their strengths and weaknesses are. This was a demanding task. Now [dustloop](http://dustloop.com) gives their own community descriptions, but I wanted this prototype to be a bit more lightweight. (Some of those descriptions are over three paragraphs long and are very technical, so I didn't want to scare off any users... yet.)

Moving on to the actual NLP techniques used here, NER was especially useful for finding key entities in the user input, because finding character or move names for example would allow us to find what the user was searching for really easily, thus allowing for more

personalized and accurate responses. Additionally, tf-idf was used to identify high importance terms based on the frequency of certain terms in a message. I applied this to my own corpus, and found the most important terms within the knowledge base and would transpose those importance values with what the user would input.

Additionally, we also added a context list so that over the course of a conversation, the bot would remember what character a user would have been asking about the whole time and would assume that if they wanted information on a jab for example, the bot would know what character the user wanted the data for.

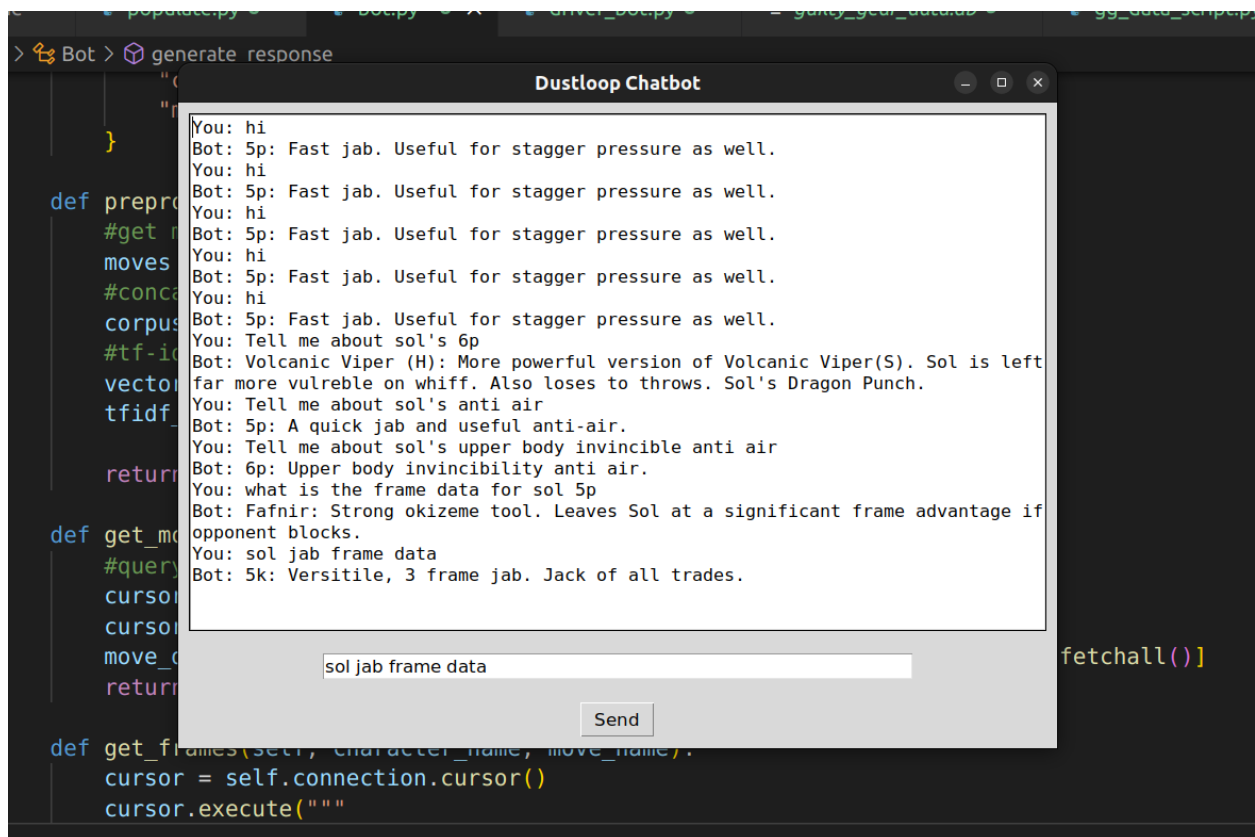
A final note about the system: the prototype is basically two python files connected to an SQLite database. The first python file houses all the logic, and the second allows for a client-facing interface/GUI thing that was easy to mockup and didn't need deployment.

Logic

The dialogue logic here was for the user to enter in a variety of inputs, based on movesets and what the user wanted. Ideally, we could generate responses based on the following sample inputs.

- "Tell me the frame data about Ky's anti-air."
- "What are some fast jabs for Sol Badguy?"
- "How negative is my frame advantage if my opponent blocks my c.S?"

Where we would have enough context to generate responses based on what the user listed, as we can find all the information in the knowledge base.



As you can see, we ran into some complications with our code. As we are only generating output on the move names and their descriptions but can't get the frame data out. But honestly, this is much farther than I thought I would get, so I'm almost stoked I got this close anyway. We have a lot of edits ahead of us should we want a viable product however.

Knowledge Base

SQLite

SQL ▾ < 1 / 1 > 1 - 2 of 2

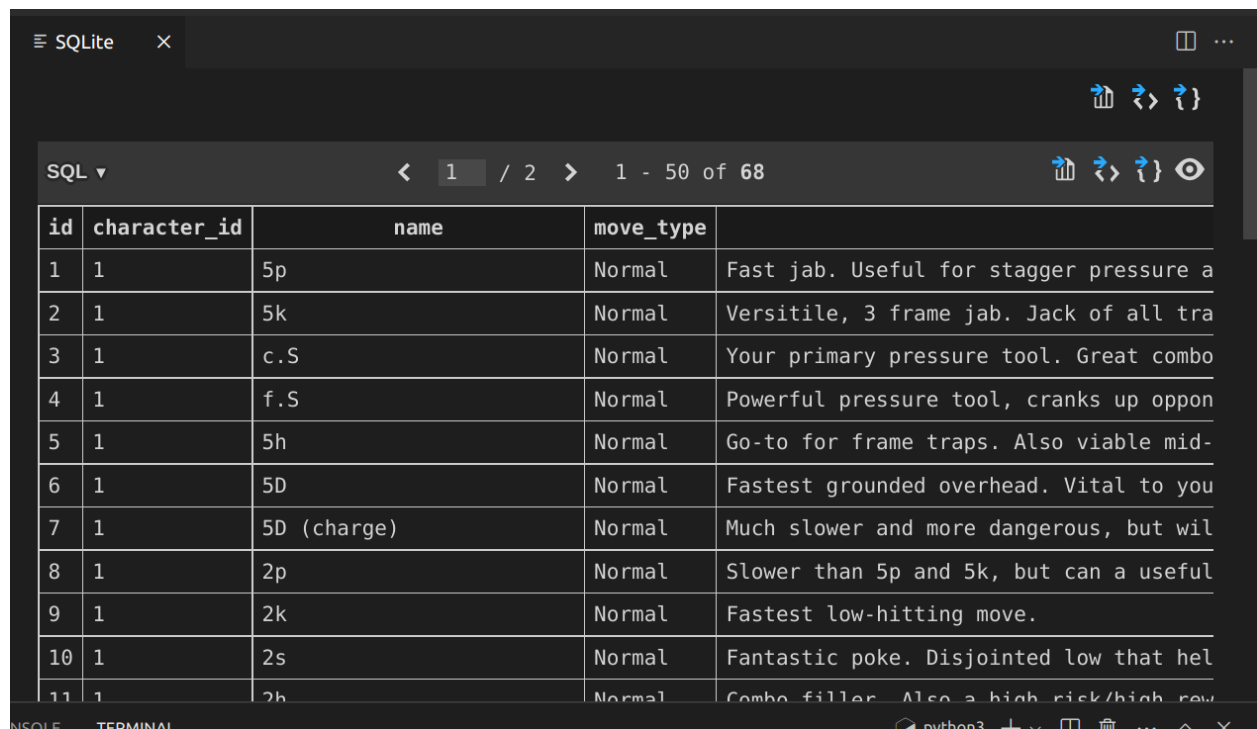
id	name
1	Sol Badguy
2	Ky Kiske

(The Character's table - reduced to two characters due to extreme data wrangling.)

id	move_id	startup	active	recovery	on_block
1	1	4	4	6	0
2	2	3	1	25	-16
3	3	7	6	10	3
4	4	10	2	13	2
5	5	11	4	20	4
6	6	20	3	26	-15
7	7	28	3	26	-10
8	8	5	4	8	-2
9	9	6	3	11	-2
10	10	10	6	15	-7
11	11	11	5	31	-17
12	12	10	3	18	-4
13	13	9	5	20	-11
14	14	15	6	20	-9

(A sample of the frame data table. Startup frames are the number of frames it takes to get a hurtbox in the game state after the animation plays. Active frames are how long the attack hurtbox is on the field of play for. Recovery frames are the number of frames it takes for the character to complete the ending animation of a move after the hurtbox disappears. The on block frame section is a race between which character can get to a neutral game state first, meaning there are no active or recovery frames and a character can perform an action. If a character has a negative on block value, it means that they have x amount of frames to wait longer than their

opponent before the attacker can act again. If positive, that means the attacker can perform an action that many frames ahead of the defender gets out of hitstun.)



id	character_id	name	move_type	
1	1	5p	Normal	Fast jab. Useful for stagger pressure a
2	1	5k	Normal	Versatile, 3 frame jab. Jack of all tra
3	1	c.S	Normal	Your primary pressure tool. Great combo
4	1	f.S	Normal	Powerful pressure tool, cranks up oppon
5	1	5h	Normal	Go-to for frame traps. Also viable mid-
6	1	5D	Normal	Fastest grounded overhead. Vital to you
7	1	5D (charge)	Normal	Much slower and more dangerous, but wil
8	1	2p	Normal	Slower than 5p and 5k, but can a useful
9	1	2k	Normal	Fastest low-hitting move.
10	1	2s	Normal	Fantastic poke. Disjointed low that hel
11	1	2h	Normal	Combo filler. Also a high risk/high rew

(A sample of the moves table, giving move names, types and a description of the move itself.)

```
7
8 class Bot:
9     def __init__(self, database_path):
10         self.connection = sqlite3.connect(database_path)
11         self.vectorizer, self.tfidf_matrix = self.preprocess_db()
12         #this way we remember what a user has to say thru-out a convo
13         self.context = {
14             "character_name": None,
15             "move_name": None,
16         }
```

This was the primitive user model we used, since it was a static popup for the chatbot that would terminate when closed, we don't have a persistent data model but have set variables so that during a conversation the bot can remember details and nuances of a conversation.

Reflections:

I'm not kidding anyone here, this bot leaves a lot to be desired, as we still need to get the frame data out and need to add the whole cast of characters. These are some improvements I plan to carry forward with the next version of the dustloop bot.

- Add all characters in Guilty Gear Strive
- Add a glossary section to the knowledge base for confusing terms within move descriptions.
- Allow for a more permanent memory structure for the user model
- Deploy on a flask web application and have the chatbot popup sit conveniently on the dustloop webpage. (Or at least mock up a design where it would look real professional.)