LING5300Project

mkamerath

April 2024

Background

For the LING5300 final project, I decided that I would do a text based adventure game. The general problem statement looks like this. "Given some text, can the program guide the user into the final state based on their text responses?" Less formally, a text based adventure game is essentially a rudimentary chat bot with some story added to it.

I approached this problem by mostly using regex and keeping track of the player through some global variables. Whenever the player provides a response, a regex attempts to match to what they said. If there is a match, then the story continues in one way. If there is not a match, then the story continues in a different way.

Throughout the adventure, there will be linguistics based puzzles that the user will have to solve in order to demonstrate their linguistics knowledge and earn their rewards at the end of the story. These puzzles are based one three different things that we learned in class.

1. Bigrams/Word Frequency
2. Finite State Automata (with regexes)
3. Context Free Grammars

My approach for solving this problem involves coming up with a set of regexes to capture user responses, and then branch according to which regex they match. It also involves the invention of several linguistics puzzles based on skills I learned this semester. This is done throughout the program with the game loop which is described in the Methodology section.

A successful program will be one that captures a myriad of user input and, if instructions are followed, allow the user to progress through the story and eventually be told that they have achieved victory.

Because this is a text based adventure, it will be hard to evaluate how to evaluate the program. In the end, I settled on doing some run-throughs of the game myself. Knowing that I would be biased towards the results I wanted, I also got my wife and brother in law to go through the program a few times and give feedback.

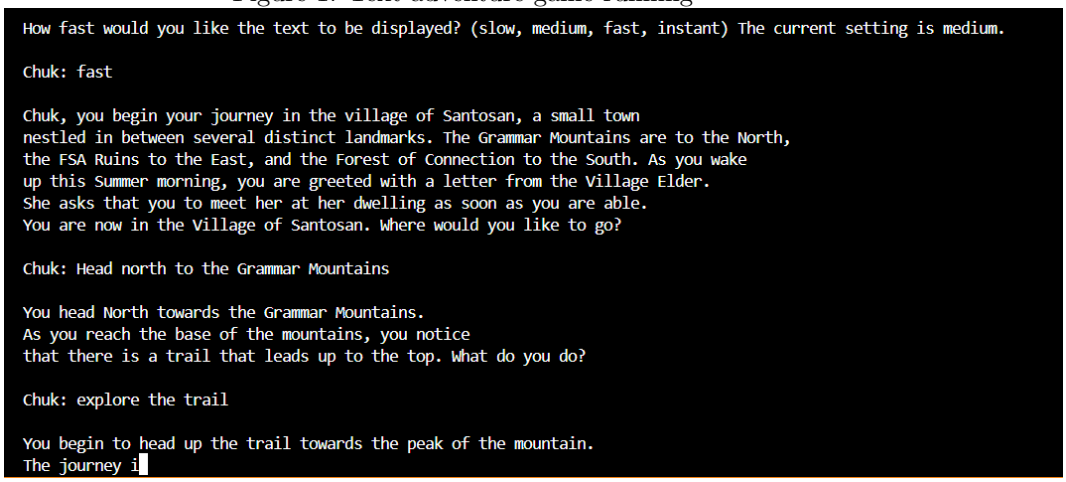
Methodology

The basic loop of the game looks like this:

$$Present\ Info \rightarrow Capture\ Input \rightarrow Match\ regex \rightarrow Decide\ Story\ Branch$$

Presenting the info seemed trivial at first. The info can just be displayed using print statements for the user to see in the console. However, after designing the first section, just print statements seemed to be underwhelming and dissimilar from text based games I had previously played. I decided on creating a method that would print letters individually and then the program sleeps for an extremely short amount of time, ≤ 0.01 seconds. This let's the text be presented in a way similar to other text based adventure games and feels more like a narration. Below is a screen cap of what the console looks like at a point during the adventure. Capturing the input was similar to presenting. Solving

Figure 1: Text adventure game running



the problem seemed trivial at first, but the same problem of displaying text was present. Fortunately, this problem was solved similarly to presenting the info and I incorporated the sleep timings into any place that required user input.

Regex matching is the core of the decision making process after the user has entered their input. Every possible thing the user could do must be captured in one of the matching regexes before a decision can be made. These regexes were

created using the context of the information presented to the user. Using this context, each regex was created to capture different ways that the player could progress in the game.

Depending on which regex is matched, different output is presented to the player. Generally, there are three different cases covered whenever a decision needs to be made. The first is when the player says something that matches the regex. In this case, the story progresses, usually the puzzle is displayed or completed. The second case is when a player attempts to give up. This happens at puzzles where they might not know the answer. In this case, the puzzle stops and they return to the main hub. Finally, if the input is not understood, the player wanders around confused, neither making progress nor returning to the hub. The player then has the opportunity to try whatever part of the journey they are on again.

Code Organization

The files in the project are organized in the following way. The puzzles are described in the Data section of this document.

start.py

This is the entry point of the program. It contains the code that gets the player's name, their desired text speed, and the main hub for the adventure. Here, the player is presented with 4 choices of places to go, the old lady's house, The FSA Ruins, The Grammar Mountains, and the Forest of Connection. Players can then choose where they would like to go.

old_lady.py

The old lady is the initial quest giver and the final stop of the game. When you first get there, she explains your quest and tells you to explore the other places. If the player has completed all puzzles, then she gives them a prize and the game is over.

fsa_ruins.py

Here, the player has to look around before they find the ruins. Once they have found the ruins, they are presented with a series of regexes one by one. After the player has entered a string that satisfies each regex, they have completed the area.

grammar_mountains.py

Here, the player ascends a trail that leads them to the peak of the mountain. On top, they will solve a puzzle that deals with CFG. Once they have come up with a sentence that satisfies the grammar rules, they have completed the area.

bigrams.py

Here, the player goes into a forest that leads them to a man who gives them information about their journey. He tells them the amount of words they have said throughout their journey and asks them about bigrams. If they answer his question correctly, the player completes the area.

helpers.py

This file contains constants and functions that are used throughout the program. Here is where the text speed is set and definitions for printing and getting input from the player. It also includes helper methods for dealing with the bigrams part of the journey. Finally, this file keeps track of whether or not the player has solved all of the puzzles and are able to complete their journey.

cky.py

An implementation of the cky parsing algorithm is found in this file. It also includes the set of grammar rules for this specific implementation which are in Chomsky Normal Form. In addition to that, there is a method that returns true if a given sentence is in the grammar or false if the sentence is not a part of the grammar.

test.py

Tests for the methods contained in helpers.py and cky.py are contained in this file. Running it should demonstrate that the methods work correctly for different types of input.

Data

There was no real data that I used for this project. So instead, this suggestion this section will be a discussion on each of the puzzles necessary to get through the adventure.

Bigram Puzzle

This puzzle is pretty simple. After the player has said at least 25 words, they can start this puzzle. The player is shown how many words they have typed. Then, they must answer how many bigrams are made from those responses using digits. If they answer it correctly, the puzzle is done.

FSA Puzzle

This puzzle is a selection of five regexes that are increasingly hard to understand. The list of regexes is:

1. `r"[Ww]hy[\w\s]+\?$"`
2. `r"[Ww]hat[\w\s]+ you are doing here\?$"`
3. `r"^[Tt]he[\w\s]+\bis [0-9]+$"`
4. `r"[\w\s]+ \$(?:[0-9]{3},)+[0-9]{3}\.[0-9]{2}$"`
5. `r"^[Ii]{3}[\w\s]+[qvr].*[xyz][\w\s]+\w*m$"`

The solutions to these puzzles can be described in the following way:

1. A string starting with Why or why, some word characters and ending with a question mark
2. A string starting with What or what, some word characters and ending with "you are doing here?"
3. A string starting with The or the, some word characters, the word is, and then any number of digits.
4. A string of any word characters followed by a dollar amount that must be at least 3 digits long. Cents must be included.
5. Starts with some combination of I's and i's, then any word characters, then either qvr or xyz, some more word characters, and then the string must end with the letter "m".

CFG Puzzle

The CFG Puzzle shows the player the following grammar:

1. S $\rightarrow$ NP VP
2. NP $\rightarrow$ Det N | NP PP
3. VP $\rightarrow$ V NP | V PP
4. PP $\rightarrow$ P NP
5. Det $\rightarrow$ a | the
6. N $\rightarrow$ thing | person | hill | man | woman
7. V $\rightarrow$ walked | saw | ate
8. P $\rightarrow$ up | on | in

They will then have to come up with a sentence that the above grammar will accept. An example sentence that is covered in the grammar would be "the woman ate the man on the hill"

Results

After being satisfied with the results of me running the text adventure, I handed it off to my wife for more testing. My desire was that she would be able to complete the adventure on her first run through if I helped her through some of the linguistics puzzles. She was not able to complete the adventure because there were a few minor bugs causing her confusion. After fixing these bugs and starting over, she breezed through everything and completed adventure.

Next was my brother in law. I was worried about his ability to get through everything. However, he went through the entire game without any sort of issue with some help on the linguistics puzzles.

In its current state, the text adventure game seems to be easy enough for somebody with an understanding of the linguistics principles learned in Computational Linguistics

Discussion

Improving Results

As always, more testing can be done with more users to determine if more coverage needs to occur in the different regexes. In addition, adding more functionality could be added to the CKY parsing to cover more possible sentences. More exact text and handling of edges cases can help in the case where users get confused.

CKY Parsing

This implementation can handle any grammar where words do not belong to more than one part of speech class. For example, the word "fine" can be used in a verb sense or a noun sense. This implementation would not allow for both. A modification of having every cell in the array contain a set of rules would allow it to parse any grammar in Chomsky Normal Form. If a sentence contains an S symbol in the top right cell. The images below show the results of the cky algorithm on different sentences.

Figure 2: Running CKY on the sentence "the woman ate the man on the hill"

```
the woman ate the man on the hill

None Det  NP   None None S    None None S
None None N    None None None None None None
None None None V    None VP   None None VP
None None None None Det  NP   None None NP
None None None None None N    None None None
None None None None None None P    None PP
None None None None None None None Det  NP
None None None None None None None None N
```

Figure 3: Running CKY on the sentence "the person walked on the person"

```
the person walked on the person

None Det  NP   None None None S
None None N    None None None None
None None None V    None None VP
None None None None P    None PP
None None None None None Det  NP
None None None None None None N
```

Figure 4: Running CKY on the sentence "the person saw the man saw the woman"

```
the person saw the man saw the woman

None Det  NP   None None S    None None None
None None N    None None None None None None
None None None V    None VP   None None None
None None None None Det  NP   None None S
None None None None None N    None None None
None None None None None None V    None VP
None None None None None None None Det  NP
None None None None None None None None N
```