Proposal: Natural Language Processing for Educational Games

Dung Le
Bennington College
Bennington, VT 05201
dungle@bennington.edu

Natural Language Processing, a subfield in Computer Science dealing with machine ability to understand and process human languages, has gained popularity in the recent years. It is predicted to be an imminent component for mobile and web applications that interact directly with users through natural languages. Game, despite its dominant mean of communication through computer graphics and scripted storytelling, does contain some certain components that deals with human languages. This project will concentrate on a particular type of educational games that is used to teach language.

The rest of the proposal is laid out as follow: section 1 gives an overview of the game - since I haven't seen the prototype of the game, this section is only a crude description of what I have heard about the game so far (Ursula will fill in the components that I missed); section 2 defines the nature of the problem (which components of NLP will this project involve?); section 3 outlines the input, methods and output of the system; section 4 discusses in details of where the system sits in a larger context and how it can be extended to fit my interest; and section 5 is the conclusion. Appendix A notes how the source-code should be kept while Appendix B is the project timeline given a span of seven weeks. All sections will be updated weekly.

1. Game Components

The user guilds its character through each level using command expressed in natural languages. In the beginning levels, this command may be as simple as "Open the door" (*öffne die Tür* in German) or "Go upstairs" (*nach oben gehen*). The objectives of the beginning levels are to learn the most rudimentary words and tense. Therefore, grammar and discourse are omitted. (Note that pragmatics is still taken into account, thus, "Push the door" (*drück die Tür*) won't open the door.) However, as the player advances to higher levels, grammar will be taken into consideration, such as "Is it possible to open the door on the left?" (*Ist es möglich, die Tür links zu öffnen?*) In any cases, the system processes users' input and replies in perfect grammar, i.e. "Yes, the door on the left can be opened" (*Ja, die Tür links kann geöffnet werden*) or "No, you must find the key first" (*Nein, du musst zuerst den Schlüssel finden*.)

This leads to an important question: how does the system know if "the door on the left" can be opened or not? For each level, the system is given a script that describes the mission of that level, of how each game component interacts with each other and with the player. This script can be a formal level instruction written by game designer or just some bullet points of how the level should be played and how to win. The system will then build a knowledge representation graph that can be used to answer users' questions/commands. This will be my starting point for this project. However, it is important to note that a script for each level can be expensive and difficult to maintain - especially when the game is scaled out to many levels. A semi-supervised approach to this problem is to use an amalgamation between computer vision (to detect game objects) and hand-coded rules (to specify components' interactions) within each level. This will reduce the bookkeeping tasks for the game developers.

2. NLP Components

2.1. Knowledge Representation and Reasoning

Knowledge Representation is a subfield of NLP dedicated to representing knowledge from text and/or conversation in a form that machine can utilize to understand the context. In this project, the game designer will input a paragraph or two that acts as a walkthrough for each level in the game. The system needs to perform two tasks: 1. Represent each action (step-by-step) and 2. Put these actions into temporal order from beginning to end. If we view the designer input as a narrative, the system, hence, needs to build what is called *narrative event chains* (Chambers and Jurafsky, 2008; 2009) By definition, "a narrative event chain is a partially ordered set of events related by a common protagonist" (in this case, the player.) I will rely on the two papers by Chambers and Jurafsky to build a similar system that captures the information needed to complete the level.

2.2. Commonsense Knowledge

Commonsense Knowledge is considered to be one of the hardest problems in NLP due to the enormous scope of commonsense knowledge. However, it is important to put the matter under consideration as this type of knowledge is prevalent in daily conversations. In this project, commonsense knowledge can be used to check for semantics and pragmatics of input sentence. For instance, in order to open the door, the common commands are "Unlock the door" or "Put the key inside the keyhole and turn it around to open the door." The unlikely, yet, grammatically correct input can be "Talk to the door", etc. The system with commonsense knowledge of how the door can be opened will prevent input that is pragmatically impossible in real life or game design. In this project, I will use Open Mind Common Sense database (OMCS) and ConceptNet developed by MIT Media Lab, in which more than a million English facts have been contributed by more than 15,000 contributors through crowd-source. (Minsky et al., 1999; Singh et al., 2002)

2.3. Question Answering

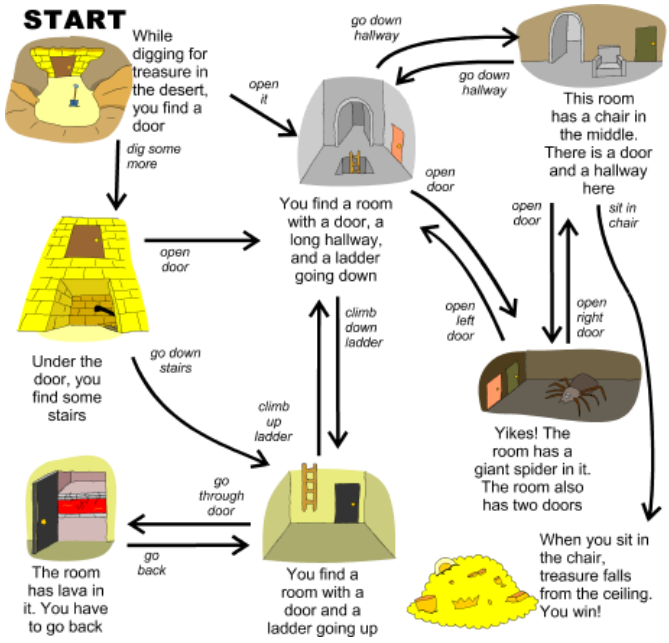
There are two major paradigms of question answering: *IR-based question answering* and *knowledge-based question answering*. While the former method relies on the knowledge extracted from a large number of documents using information retrieval, the latter answers natural language question by mapping the query over a structure database. This knowledge database can be built using knowledge representation as mentioned earlier. Since the game knowledge is unique for each level, this project approaches the issue using knowledge-based question answering method. (More on QA can be found in Chapter 28. Question Answering, *Speech and Language Processing (3rd edition)* by Daniel Jurafsky and James H. Martin)

3. Methodology

3.1. System Input

The system requires two types of input:

- *User Input:* an appropriate command/question (e.g. “Pick up the key”, “Open the door”, “Can I go upstairs?”, etc.)
- *Designer Input:* a paragraph or sentences that describe the objective and components (objects, interactions, controls, etc.) of the level. For the moment, I will write this input.



Example of designer input (for the above map):

“The objective of this level is to find the treasure hidden somewhere in the desert. You (the player), in the role of an explorer, start by digging for the treasure in the desert. While digging in the desert, you find a door. You need to input: i. ‘Open the door’ or ii. ‘Dig some more’. …”

Or the input can be as simple as:

- Level Objective: find the hidden treasure

- Level Objects: entrance door; stairs below entrance door; a room with a door on the right, a ladder going down in the middle, and a long hallway; etc.
  NOTE: It is important that the level objects have some indications of where the objects are in relation to others.
- List of possible commands (for interaction): 'Open the door', 'Dig some more', 'Go down the hallway', 'Climb down the ladder', etc.
- Others: advanced interactions or controls (if possible)

3.2. Processing Methods

From the designer input, the system first build a knowledge representation graph that captures the knowledge of the level (the objects and interactions). This will likely be a finite state machine that represents the map of that level. As mentioned earlier in section 2, Chambers and Jurafsky (2008, 2009) developed an unsupervised method for learning Narrative Event Chains from text. At the moment, I plan to apply a similar method, yet for a multi-scenarios event where the users' input is the central part of the narrative.

From the user input, the system first check for grammar and spelling. If the input is valid, it then compares to the list of possible commands and the knowledge representation model to check if this command/question is appropriate in this scenario. In addition to this, a commonsense knowledge database named *Open Mind Common Sense* (Minsky et al., 1999) will be used to check for pragmatics and semantics. This commonsense knowledge check is important because the user may input a grammatically correct command, yet, pragmatically and semantically ill-formed. (For example, the command "Commend the door" is grammatically correct, yet, will not open the door.) After checking for syntactics and semantics, the system uses the knowledge representation model to answer the users' command/questions.

3.3. System Output

- *Output:* a relevant and grammatically correct response (e.g. "The door is now opened", "A key needs to be found in order to open the door", etc.)
- ***Important:*** Even though the game aims to teach German, the currently supported language for input and output is English. Still, I will look for some quick translation program to test for German later.

4. Educational Games for teaching Programming Language

When I first brainstormed ideas for my senior work at Bennington College, I was passionate with the idea of creating a mobile app that teaches middle schoolers programming language. It was an attempt to make programming language less intimidating by teaching it the same way as children were taught English, Chinese or German. The project came forth as part of my belief that Scratch

and similar applications, despite their approachable graphics and content, was incompetent in teaching the language itself. The problems with Scratch are that the program focuses on logics and problem solving skills while abstracting away the core concepts and syntaxes of a programming language. As the results, children, despite having a good time playing around with the projects, struggle when learning the more 'serious' languages like Python or Java.

With that being said, I aimed to develop an application that utilized pseudocodes input by users to teach children both the problem solving skills and the syntactics of programming languages. However, after the discussion with my academic advisor and some professors at Middlebury College, I decided to postpone this idea to concentrate on NLP first. As I have been thinking through the methods for the language learning game, this idea becomes pertinent in many ways. For instance, a knowledge representation graph that I will build for the game can be modified so that it represents the users' purpose for each line of pseudocode, and from that, translates the pseudocode into appropriate programming language. If by any chance that I will be able to finish the system for the language learning game before the end of FWT, I will give this project some more thoughts and update the content of this section.

5. Conclusion

(to be written toward end of FWT)

REFERENCES

Nathanael Chambers and Dan Jurafsky. 2008. Unsupervised Learning of Narrative Event Chains. In *Proceedings of ACL-08,* Hawaii, USA.

Nathanael Chambers and Dan Jurafsky. 2009. Unsupervised Learning of Narrative Schemas and their Participants. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP,* Suntec, Singapore.

Push Singh, Grace Lim, Thomas Lin, Erik T. Mueller, Travell Perkins, Mark Tompkins and Wan Li Zhu. 2002. Open Mind Common Sense: Knowledge Acquisition from the General Public. *American Association for Artificial Intelligence*.

Daniel Jurafsky and James H. Martin. 2008. Speech and Language Processing.

APPENDIX A. List of all useful links

- I created a GitHub repository to store all the source-code and literatures used for this project at https://github.com/DungLe13/NLP-for-Game . I think there are many benefits for using GitHub, such as to do code review or for you to keep track of the progress. However, if you want to use a shared folder in Google Drive or anything, we can work this out.
- FWT 2018 Timesheet (to be filled in weekly) https://docs.google.com/document/d/1S3cy1op-0kfKNh9i0z0PP85Gre7VJEuJofD4p1GH__o/edit?usp=sharing

APPENDIX B. Project Timeline

WEEK	DATES	TASKS	STATUS
Week 1	01/02	Getting started on project proposal	Completed
	01/03	Continue working on proposal Sharing the document with Ursula	Completed
	01/04	Literature Reviews Finish proposal	Completed
	01/05	(**Ursula)** review the proposal, questions, etc. Dung: revise the proposal based on achieved consensus.	Need confirmation from Ursula
	Objectives	Complete the first 4 sections of the proposal Achieve consensus on the tasks and methods Literature Reviews	Completed
Week 2	01/08	Write sample for designer input Build the system that processes users' input First commit to GitHub	
	01/09	Build a grammar checker	
	01/10		
	01/11		
	01/12		

	Objectives		