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 guides 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.

UPDATED: Game Prototype
There will be two scenes in this prototype:

1. The first scene simulates a trending hidden objects game, where given a room full of objects, the player needs to find a list of objects described in the game. I think the game dynamic works perfectly to learn new words in a different language.
2. The second scene will focus on learning grammar and reviewing new words learned earlier. This scene will align with the game description mentioned in the earlier paragraph.

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

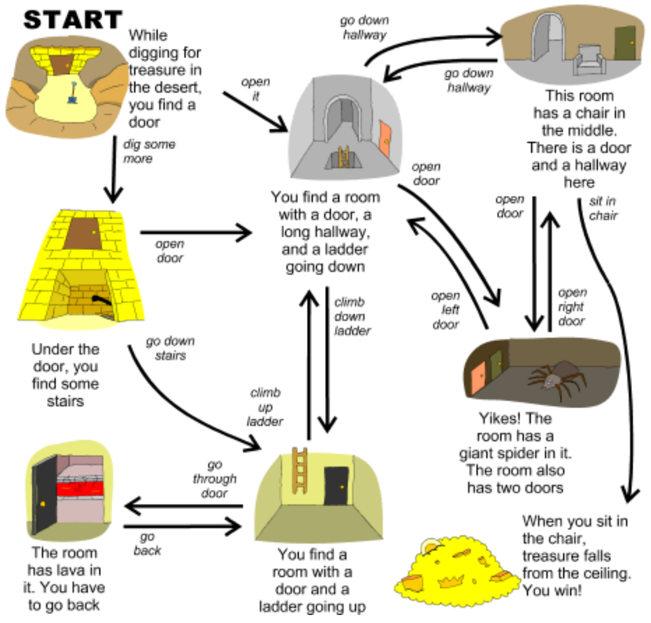
There are two major paradigms of question answering: *IR-based question answering* and *knowledge-based question answering*. While the former method relies on the knowledge extracted from a large number of documents using information retrieval, the latter answers natural language question by mapping 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.


START
While digging for treasure in the desert, you find a door
dig some more
open it
go down hallway
go down hallway
This room has a chair in the middle. There is a door and a hallway here
open door
You find a room with a door, a long hallway, and a ladder going down
open door
open door
sit in chair
climb down ladder
open left door
open right door
Under the door, you find some stairs
go down stairs
climb up ladder
Yikes! The room has a giant spider in it. The room also has two doors
go through door
go back
The room has lava in it. You have to go back
You find a room with a door and a ladder going up
When you sit in the chair, treasure falls from the ceiling. You win!

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.
(**UPDATED:** Two approaches for processing designer input and building world (level) model

- *Approach 1*: The designer provided a paragraph that described every step in details. I built a Python script to process this text and extract information into a dictionary of events and possible actions. (Note: Since the script is partially rule-based, this paragraph must meet certain criteria in syntaxes and format.)
- *Approach 2*: I provided a generic script (templates) in the form of an array of dictionaries. The designer, without knowledge of programming language, just needed to fill in the objects and possible actions within each level.

Certainly, there were both pros and cons for the two approaches. However, in this prototype, for better precision, I would go with the second approach. Still, I kept the script of the first approach, in case the game scales quickly and approach 2 is no longer appropriate.)

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. Side Projects

5.1. Side Project 1: Machine Comprehension using Commonsense Knowledge

Is it possible for an artificial intelligence system to understand commonsense knowledge through reading comprehension? And if yes, how can this knowledge be represented and interpreted in the task of question answering? These are the questions that this project, formally described in *SemEval-2018 Task 11: Machine Comprehension using Commonsense Knowledge,* aims to answer. As mentioned in section 2.2, commonsense knowledge is considered one of the most difficult topic in natural language processing, and despite the fact that commonsense knowledge is not the central aspect of this project, I believe that this side project, if being taken with care, can assist the educational game quite nicely.

In the following section, I will describe my approach on the shared task. Details on the task, including motivation, description, and data sets can be found at https://competitions.codalab.org/competitions/17184. The dataset contains multiple instances, with each having one scenario and a series of questions related to the scenario.

For each scenario:

- Step 1: Represent the scenario (paragraph) into a list of narrative events: verb(arg_1, arg_2) (e.g. [went(I, None), flipped(I, light switch)])

OR complete phrase of event (e.g. [I went into my bedroom, I flipped the light switch])

- Step 2: Use Glove model to obtain distributed representations of words.
- Step 3: Represent each event as vector of real numbers.

For each question in the set of questions:

- In the training set, for each correct answer, replace the the answer into the question to form an affirmative sentence. These correct answers will be used later to train the model of natural language inference.
  e.g. Q: Where did they put the picked up toys?
  Corrected A: in their containers
  => They put the picked up toys in their containers.
- Represent the obtained sentence in the same formats as step 1 above.
- Obtain vector representation of event using the same model as in step 3 above.

MODEL 1: Distributed Representation of Narrative Event

A simplified version of neural network will be built. The task is to find the correct order of narrative events given the events (the dataset is taken from DeScript gold-paraphrase dataset, with each event being manually labelled.) However, the ultimate goal for this subtask is not the order task, but the embedding parameters {R, C, T, A}.

- *Approach 1*: (Modi and Titov, 2014) The hidden layer is computed by summing linearly transformed predicate and argument embeddings and passing it through the logistic sigmoid function. (Different transformation matrices for arguments and predicates, T and R, are used.) The event representation f(e) is then obtained by applying another linear transformation (matrix A) followed by another sigmoid function. (NOTE: for the complete phrase of event, the same transformation matrix is used for every word in the phrase.)

- *Approach 2*: (Baroni and Zamparelli, 2011; Socher et al., 2012) To be added later

MODEL 2: Natural Language Inference using LSTM

6. Results and Conclusion

For seven weeks, I have accomplished the following:

- A fully working user processing system. This includes checking for user's spelling, grammar, and pragmatic input.
- A fully working knowledge representation system that takes the level description (given by game designer) and builds a world knowledge about the level (objects and possible interactions.)
- A simple question answering system that takes the user command/question and compares it against the knowledge about the level to provide an answer.
- A REST endpoint for the whole system.
- A simple 3D game prototype, titled Language Explorer, made using Unity3D.

Evaluations on each component:

1. *User processing system*
   Unlike the spelling and grammar correction provided by human, an automated system is certainly not perfect, which in this case can be a disadvantage given that the ultimate goal of this application is to learn a language. However, my system for processing users' input achieves a very high accuracy.

   The grammar checker at the moment is quite simple, and can be made better using Hidden Markov Model (basically, the idea is that given a sequence of words and their POS, the system calculates the probability of sequence and compares the score against a threshold.) The pragmatic checker at the moment checked whether or not the user's input "made sense" using commonsense knowledge database. The accuracy of this approach depends on the size of the database and its information. For now, this is the best method that I can come up with; if given the opportunity, I will look deeper into this area.

2. *Knowledge representation system*

The knowledge representation system is partially rule-based. As I mentioned earlier, there are two approaches in representing knowledge about the level. Here, I will discuss its pros and cons:

- *Approach 1*: This approach does not restrict game designer about the format of the game description. As the game grows larger and more levels are added, this approach might be preferred since it can scale well with the game. One problem, however, is that this approach might have a lower accuracy than the second.
- *Approach 2*: Since this approach asked game designer to provide the level knowledge in certain format and grammar, it has very high precision. However, as the game scaled up, this approach might not be sustainable.

This is a common trade-off in many model, which is referred as the precision-recall ratio.

3. *Question answering system*
The question answering system for the moment is also partially rule-based. It can answer the simple command from the user. The word embedding method, despite taking quite some time to load the model, works well in comparing if the two actions (one as input by the user and the other as provided in the level knowledge) are semantically equivalent.

4. *Game Prototype*
The game prototype was initially not a part of this project. However, I managed to build a simple 3D game to demonstrate the NLP system. As mentioned earlier, the animation was omitted, yet, all the game objects were put together. (Camera and lighting can be better though). For the moment, I have not linked the two components together. However, with the REST endpoint being set up, this should not be a problem.

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.
- Side Project 1: Machine Comprehension using Commonsense Knowledge. GitHub repository can be found at https://github.com/DungLe13/commonsense .
- 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	Completed

		Finish proposal	
	01/05	**(Ursula)** review the proposal, questions, etc. Dung: revise the proposal based on achieved consensus.	Need confirmation from Ursula. Completed (01/16)
	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	Completed
	01/09	Build a grammar checker Finish processing users' input	Completed
	01/10	First attempt at knowledge representation	Completed
	01/11	Continue working on knowledge representation	Completed
	01/12	Finish works on knowledge representation	Incomplete (due to other application deadlines)
	Objectives	Processing user's input (spelling, grammar, and pragmatic check) Knowledge representation	Work in progress
Week 3	01/15	Continue works on knowledge representation	Completed
	01/16	Continue works on knowledge representation	Completed
	01/17	Review abstract and reviews of the paper sent by Ursula	Completed
	01/18	Minor changes to knowledge representation Knowledge representation (commit to GitHub)	Completed (not commit to GitHub yet)
	01/19	Continue working on game prototype Wrap-up on knowledge representation Approach 2: knowledge representation template (need to update this approach into the	Completed (commit to GitHub)

		documentation)	
	Objectives	Knowledge representation (cont.) Build the first game prototype (using Unity) (It turns out that a generic knowledge representation is difficult than I expected. Thus, no game prototype was done this week. Despite this, I have some ideas for the game design, and will update the 'Game Components' section.)	Work in progress (game prototype needed)
Week 4	01/22	First attempt building narrative representation (narrative event chain - based on Chambers and Jurafsky, 2008)	Completed
	01/23	Side project literature reviews (narrative representation, question answering, natural language inference)	Completed
	01/24	Finish working on narrative representation Get Stanford CoreNLP to work Commit to GitHub	Completed
	01/25	Second attempt building narrative representation (distributed representation of predicate and its arguments - based on Modi and Titov, 2014)	Completed
	01/26	Continue working on building the second approach for narrative representation	Completed
	Objectives	Side Project 1: Machine Comprehension using Commonsense Knowledge (Examine the dataset, start building data pre-processing script and narrative representation)	Completed
Week 5	01/29	Third attempt building narrative representation (make sure that each sentence contains only one narrative event)	Completed
	01/30	Continue working on narrative representation (third attempt)	Completed
	01/31	Literature reviews on distributed representation of words and events as real-numbered vector. GloVe (installation + testing on dataset)	Completed

	02/01	Update the documentation on the approaches for Machine Comprehension using Commonsense Knowledge Exploration on the dataset used to learn distributed representation of events	Completed
	02/02	Day Off	Day Off
	Objectives	Continue working on side project 1 - refine the models for narrative representation	Completed
Week 6	02/05	Getting back to the NLP for game project (Start linking the two scripts, one processing users' input and the other building world (level) model) Finding images and assets for game prototype Updating the game parts in documentation	Completed
	02/06	Start building walls and rooms in Unity3D Continue finding 3D models and game assets	Completed
	02/07	Adding 3D character + scripts for basic movements and exploration Refine 3D models (ground, lights, and game objects)	Completed
	02/08	Dialog box added + basic scripts for interaction	Completed
	02/09	Return to side project 1: fourth attempt in building narrative representation (actually, a refinement of the third attempt)	Completed
	Objectives	Side Project 2: Build a game prototype using Unity3D and C# (Note: this game prototype is meant to test the NLP component. As a result, the animations of game objects were ignored.)	Completed
Week 7	02/12	Python scripts for reading and extracting template level description + question answering Refine the script for processing user's input	Completed
	02/13	Continue with question answering Game prototype revisited	Completed
	02/14	Change script into functions + adding REST	Completed

		endpoints First attempt in calling REST endpoints from C# script	
	02/15	Tet Holiday @ noon	
	02/16	Final updates to the documentation	Completed
	Objectives		