Using Natural Language to Manage NPC Dialog

Gary Kacmarcik

Natural Language Processing Group
Microsoft Research
Redmond, WA
garykac@microsoft.com

Abstract

In this document, we describe our work applying natural language (NL) technologies to improve non-player character (NPC) dialog interactions in games, specifically role-playing games (RPGs). Our approach is to adapt the standard dialog menu interaction so that the menu items are dynamically-generated during game runtime rather than scripted during development time. In our system, menu items are constructed by manipulating abstract semantic representations stored in the NPC knowledgebase, converting them into NL text, and then ranking them so that the most relevant items are placed at the top of the menu. We demonstrate our approach in the context of a small RPG.

Introduction

Despite the oft-expressed interest in using natural language (NL) interactions in gaming worlds, a number of serious issues relating to language ambiguity – notably paraphrase handling and coreference resolution – need to be addressed before these techniques become practical. These problems are evident even in the ideal case where we have a co-operative player and assume that the speech recognizer or parser is working flawlessly. Real-world situations make these issues even more problematic.

It is thus unsurprising that commercial games rely primarily on scripted dialog interactions, often presented as a menu of choices. While this approach is flexible and expressive, the cost of authoring each interaction separately limits its ability to scale to large, complex, and dynamic interactions.

In this work, we explore the ground between scripted menu and full NL interactions. More specifically, we seek to merge these two approaches to avoid the NL interface problems while creating a system that is more capable of supporting dynamic interactions.

Related Work

While considerable work has gone into creating interactive dialog systems, the game industry has until recently expended relatively little effort making systems that work outside of predetermined scripts. Interactive storytelling (IS) strives to move away from scripted storylines by constructing a framework in which the stories emerge from the data. While recent work has reported success, these systems still rely primarily on scripted or templatic utterances as their building blocks.

NL interfaces are commonly used in these systems because they can help reinforce the dynamic nature of the interaction. However, one of the touted advantages of NL interfaces, the expressiveness granted to the player, is actually an illusion that the game developer must expend considerable effort to maintain. All possible input variations must be anticipated and encoded so that they can be handled correctly, and out-of-domain input must be managed robustly. Failure to handle this properly can frustrate the player and can break the immersion, thus defeating the purpose of the providing a rich interaction.

Our approach supports dynamic interactions and avoids NL input problems by creating dialog menus dynamically (and directly) from the contents of the NPC's knowledge-base (KB). To address the cognitive overload problem that can occur with a complex menu, we rank all of the menu items based on their relevance to the current game state so that the top-most items are always the most interesting.

Dynamic menus with NL underpinnings were first described in (Tennant at al. 1983), where a predictive left-corner parser was used to construct NL input using a cascading series of menus. We differ significantly from this approach in that we produce a single menu of complete utterances that are presented in ranked order of relevance.

The goals of the present work have much in common with that of (Szilas and Kavakli 2006) in that we both seek to improve interaction in dynamic environments without relying on NL input. Our approaches differ in that their work provides a two-tiered menu with no ranking of the items and context sensitivity only on the second tier. Their first tier is also comprised of a trace of all game events, which is unmanageable for all but the shortest games.

Dialog Model

For this project, we were interested in creating a dialog interaction that would support a dynamically changing

environment without incurring the cost and problems associated with a NL interface. We employ a variant of the standard dialog menu where the menus are populated dynamically with context-sensitive options that are automatically extracted from the NPC's KB.

In essence, we have the game engine simulate both the player and the NPC side of each dialog. For the NPC, the options are evaluated and the best option is selected. For the player, the options are ranked and an *n*-best list of options is presented to the player to select from.

Knowledgebase Creation

Rather than create our own special purpose representation for the NPC's knowledgebase, we opted to use the logical form (LF) structures produced by an existing parser (Heidorn 2000). The advantages of this approach are that it defines a truly general representation (Iwańska 2000) and it also provides a straightforward mechanism for creating these forms (via the parser).

Parsing the text for the KB does introduce some risk to our system. We address the problem associated with the inevitable parse errors by parsing only during development time and providing feedback about questionable parses. This permits any errors encountered to be fixed before the game is released. We also require that all coreferences within the text are appropriately tagged.

Interaction Candidates

Interaction candidates (ICs) are the items used to populate the dialog menus that are presented to the player when interacting with NPCs. An IC consists of two parts: a stimulus string and a response string. During gameplay, the player chooses from the set of available stimulus strings and is then shown the corresponding response.

Rather than scripting the ICs, we create them dynamically from the KB of the NPC. As a simplification for the current implementation, we limit ourselves to a question-answering (QA) interaction, so the stimulus is a question being asked and the response is the answer.

The list of potential ICs for an NPC is created by examining the entries in the KB and applying tree regular expression (*trex*) patterns to identify common structures and extract the interaction pairs. These trex templates are designed to be as general as possible so that once they are developed they can be reused among different games.

For example, given the sentence: "She will be OK if you can get a healing potion" our parser produces the following tree structure that is stored in the KB:

```
be [Futr]
└─Tobj─OK
        ├─Tsub───she
        └─Conditn─get [if]
                  ├─Tsub──you
                  └─Tobj──potion
                          └─Attrib─heal
```

On this tree, the following trex extraction pattern can be applied:

```
*0 be [Futr]
 └─Tobj─*1
         └─Conditn──*2
```

Applying this pattern extracts the subtrees that correspond to the *'ed nodes. (Note that when discussing trex patterns, we make use of a concise inline notation. E.g., "{*0 Lemma "be" Futr Tobj {*1 Conditn {*2}}}" for the above tree.)

The matches obtained from the above extraction pattern are then inserted into a construction pattern: "{*0 YNQ Tobj {*1 -Conditn}}" to create the following tree:

```
be [Futr YNQ]
 └─Tobj─OK
         └─Tsub──she
```

From which the generation component produces the stimulus string "Will she be OK?". The matches are also inserted into a second trex pattern ("{*2}") to produce the corresponding response string ("If you can get a healing potion."). This process is repeated for all the extraction patterns across all KB entries to create the initial set of ICs.

Managing Game State

While it is a straightforward process to create the list of ICs from the KB, it is necessary to present the items to the player in a manner that is dependant on the current game context. This context is managed by the introduction of *game tokens* into the NPC's KB. These tokens are abstract entities used to track game events and chunks of knowledge that are relevant to the gameplay. In our implementation, the game tokens correspond to specially-marked nodes or tuples (node-relation-node) in the KB.

Accumulating Tokens. Tokens are accumulated by the player in one of two ways. The first way is by participating in some event (like visiting a location), which will result in the token being given to the player.

The second way is by interacting with NPCs in the game to get the tokens that are encoded in that NPC's KB. For example, given a KB that contains "She got sick by eating a poisonous berry":

```
get
├─Manner──eat [*1]
│         ├─Tsub──she
│         └─Tobj──berry [*2]
│                 └─Attrib──poisonous
└─Tobj────sick [*3]
          └─Tsub──she
```

The game designer could attach tokens to each of the nodes marked with an * depending on the requirements of the game:

[*1]: she ate a poisonous berry
[*2]: poisonous berries exist
[*3]: she is sick

When ICs are extracted from KB entries, the tokens that exist in the KB are copied onto the stimulus and response structures of the IC. So, the "How did she get sick?"

question would be associated with the *3 token and the response "By eating a poisonous berry" would have the *1 and *2 tokens. In this fashion, the game can keep track of what the player has already learned and make appropriate adjustments to subsequent interactions.

Enabling ICs. The primary use for the tokens is to control the game story by enabling and disabling ICs based on the current game context. ICs are enabled if the player has obtained all of the tokens that correspond to the stimulus and is lacking at least one token in the response.

Player Goals. For our current implementation, we use a token dependency graph to encode the storyline for the game. Given the player's current set of tokens and this graph, we can trivially calculate the player's current goals. This goal calculation is done by identifying the set of tokens in the graph that have all of their preconditions met but have not yet been obtained by the player.

Ranking ICs. The primary purpose of the goal tokens is so that we can properly rank the ICs that are presented to the player. This is a critical component of the interaction because it prevents the player from being overwhelmed by a large number of "uninteresting" options.

When ranking the ICs, they are first categorized into bins based on whether or not they contain a goal token, repeat information that the player has already obtained, or are disabled. ICs may be disabled if they are not yet askable by the player or are duplicates of other ICs.

Within each bin, we rank each IC based on the TreeRank for each IC's stimulus and response and summing the weighted values. The TreeRank is computed by summing the TextRank (Mihalcea and Tarau 2004) values of each node in the tree. Beyond simply moving the most relevant results to the top of the candidate list, this approach also gives each NPC its own set of personal IC biases based on the overall contents of its KB.

Evaluation Metrics

There are three aspects of our system that require some sort of evaluation: Authoring, Gameplay, and Generation. Authoring is a development time metric and the other two are runtime evaluations.

Authoring. For this approach to be viable, we need to show that the authoring cost is reduced when compared against a scripted system that provides equivalent player choice. Given that we can extract multiple ICs from a single KB entry, we have a potential advantage over scripted systems which produce one interaction per authored entry. This fact, coupled with our ability to share KBs between NPCs and apply deictic substitutions to localize the content for each NPC, gives an advantage which should be apparent when applied to larger game scenarios. Of course, the parsing and coreference annotation requirements for each KB entry make them more costly to author than dialog tree entries. This cost delta is hard to quantify, but it is clear that our approach would be cost-effective only for non-trivial games.

Gameplay. The gameplay risk in a dynamic system is that the game may have dead-ends where the game is difficult to solve or unsolvable. We address this problem by running the game through a simulator that visits all available locations and automatically chooses each of the top 3 menu options, continuing as long as a new token is acquired. Non-game final situations where no new tokens can be acquired are problem areas that need to be addressed.

Generation. Being the only runtime NL component, the generation engine requires its own evaluation. Given the closed domain defined by the union of all KBs in the game, we can generate all possible utterances by producing all combinations of allowable tree manipulations. These utterances can either be reviewed manually or the perplexity of the stimulus and the response strings can be compared relative to an *n*-gram language model.

Conclusion

Our system for managing dialog allows the dialog text to emerge from an underlying NL-based representation. Through the creation of dynamic dialog menus, we extend a proven interaction technique and make it useful for dynamically generated game content. By virtue of ranking the content in this menu, we also provide a mechanism for guiding for the player through the game.

The current system presents many opportunities for future enhancement. A full goal-driven dialog model with a sensible personality, mood and emotion model would offer obvious advantages over the currently implemented QA interaction. Menu item ranking can be improved by incorporating recent dialog context and topic tracking. We also anticipate replacing the static KBs and token graphs with dynamic structures to create more compelling dialog interactions.

References

Heidorn, G. 2000. "Intelligent Writing Assistance". In R. Dale, H. Moisl and H. Somers (eds.) *A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*. Marcel Dekker, Inc. New York.

Iwańska, Ł. 2000. "Natural Language Is a Powerful Knowledge Representation System: The UNO Model". In Ł. Iwańska and S. Shapiro (eds.), *Natural Language Processing and Knowledge Representation*. AAAI Press/ MIT Press. Menlo Park CA. 7-64.

Mihalcea, R. and Tarau, P. 2004. "TextRank: Bringing Order into Texts", *EMNLP '04*, Barcelona.

Szilas, N. and Kavakli, M. 2006. "PastMaster @ Storytelling: A Controlled Interface for Interactive Drama". *IUI '06*. Sydney, Australia. pp. 288-290.

Tennant, H., Ross, K., Saenz, R., Thompson, C., and Miller, J. 1983. "Menu-based Natural Language Understanding". *ACL '83*. pp. 151-158.