Question-Answering in Role-Playing Games

Gary Kacmarcik

Microsoft Research
One Microsoft Way
Redmond, WA 98012 USA
garykac@microsoft.com

Abstract

In this paper, we give a general description of the issues associated with performing basic Question-Answering (QA) tasks against non-player characters (NPCs) within a simple role-playing game (RPG) or virtual world environment. We describe the aspects of this kind of QA system and provide an overview of our initial explorations into the implementation and evaluation of a QA system that is appropriate for RPG environments. We also introduce Keystone, a simple RPG environment that provides a small virtual world that allows different NPC controller backends to be plugged in thus allowing these systems to be evaluated in a more realistic game environment.

Introduction

For more than 40 years, researchers have been developing a variety of Restricted Domain Question-Answering (RDQA) systems that have been designed to cover a wide range of specific domains. The earliest systems (e.g., BASEBALL [Green61], LUNAR [Woods73]) used restricted domains out of technological necessity, but recently there has been a renewed interest in RDQA. This resurgence has been caused in part by a recognition of the difficulties of general open domain QA, but also by a desire to create complete working systems that address specific problems.

In this paper, we discuss the application of QA techniques to a particular restricted domain that has not been explored before in this context - that of computer role-playing games (RPGs). As part of this discussion, we describe the special properties of RPG environments and outline the techniques and approaches that are required.

As we envision it, QA in an RPG environment is a special case of the *interactive dialog* type of QA problems. Certainly the most well-known example of this system is [Winograd72]'s SHRDLU or "blockworld". The SHRDLU system supported a dialog that allowed the user to control a robotic arm in a virtual world of blocks on a table. The user could issue commands that affected the world state and also query the current state of the system.

Toys & Games

One problem with SHRDLU is that it is clearly a toy. We mean this not in the pejorative academic sense (i.e., a system designed using a tiny or contrived data set with limited scope or scalability), but rather to draw attention to the distinction between toys (like a train or a jump rope) and games (like chess) where a toy has no defined goal or winning conditions. Unless a toy can be made part of a larger game, the players are likely to lose interest or get distracted. Without a method for determining the player's goal in interacting with the system, there is no easy way to anticipate which type of questions will be asked.

The interactive dialog QA approach described here is designed specifically for games. By investigating the problem in a game environment, we ensure that the user has a clearly defined purpose, i.e., to win the game. This narrows the scope of the interaction and permits us to make simplifying assumptions.

Clearly, not all games are appropriate for QA interaction. For reasons the become clear below, we decided to work within a role-playing game framework.

Role-Playing Games (RPGs)

A *role-playing game* (RPG) is one of the more popular genres of video games being released for PC and game console systems. In an RPG, the player controls the actions of a character in a virtual environment, typically from a third-person over-the-shoulder or overhead view. RPGs are distinguished from other genres by the increased amount of interaction that they allow with the inhabitants of the virtual world.

The goal of the player in an RPG is to complete minimally one main quest and optionally some smaller or "side" quests. Whether or not the player is ultimately successful in the quest(s) is determined by the player's actions in response to interactions with the game environment and its inhabitants.

These game world inhabitants are termed *non-player characters* (NPCs) since they are controlled by the game system rather than a player. The quality of the interaction between the player and the NPCs in a game is crucial to the success of an RPG.

In most currently available RPGs, the player-NPC interaction handled by a *finite state machine* (FSM) of scripted dialog that is pre-written by the game designers. Some games introduce a bit of variability by using Markov models (commonly called *fuzzy state machines* (FuSMs) in game

development literature) to help vary the interaction. At the core, however, each state in the model still corresponds to a scripted interaction. In addition, some NPCs require custom interfaces. For instance, a shopkeeper NPC that allows the player to purchase or trade items. Our goal in creating more interactive NPCs is not to replace these current techniques, but rather to provide additional methods that can be used alongside existing technologies as deemed appropriate by the game designer.

QA in RPG Environments (RPG-QA)

There are many characteristics of RPGs that pose interesting or unique challenges for QA. In this section, we describe the aspects of RPG-QA and introduce our proposed architecture for applying QA techniques to the player-NPC interaction problem.

Aspects of RPG-QA

Even though there is considerable overlap in technologies and approaches between QA in an RPG and general QA, there are many significant differences. This section summarizes these aspects and describes where appropriate how they contrast with more typical QA applications.

Closed Domain. A significant feature of RPG environments is that they are completely closed domains. In most cases the entire virtual game world is constructed by the game designers specifically for that particular game.

Dynamic Domain. While the domain for an RPG is closed, it need not be a static collection of documents. The documents can be changed or augmented as a result of player-NPC interaction.

Document Preselection. In most QA systems, either all of the available documents are used, or a document preselection process based on the query is applied to reduce the number of documents to be evaluated. For an RPG, a trivial document preselection process is employed that depends not on the query but rather on the NPC being queried. In this process, only the documents that correspond to the selected NPC are used to answer the query. This is an important issue because it is appropriate and desirable for the system to return "no answer found" even if the answer exists elsewhere in the document collection.

NPCs Are "People". In an RPG, the query is not being addressed to an abstract entity or an oracle, but to a particular NPC in the world selected by the player. This carries with it certain expectations on the part of the player since the personality of the NPC needs to be reflected both in the answer and how the answer is presented. Approaches to this problem can range from simple lexical substitution to customized generation grammars.

Input Restrictions. Another significant difference is that RPGs tend to have considerable input restrictions that affect how the player can pose a query. Many game systems don't have keyboards available at all and those that do tend to avoid their use in games since text input is error prone and interrupts the flow of the game.

Single Answer. Just as in QA problems like story understanding, we know beforehand which documents contain the answer. Additionally, these documents do not contain redundant information and an answer typically occurs in exactly one document (if an answer exists at all). Thus, the precision issues commonly associated with restricted domain QA (where the answer appears in a small number of documents) apply here as well.

Queries. In a goal-driven environment like a game, there is a restricted set of query types that are commonly employed. In addition, there may also be default or null queries which are used to initiate the dialog with the NPC.

Coreferences. In a closed domain like an RPG, all of the entities mentioned in the document collection either have virtual representations in the game environment, or they are abstractions of entities in the environment. This is important for reference resolution since it means that all coreferences that matter for gameplay can (and should) be resolved.

User Model. When formulating the answer, the NPC should take into account its model of the listener (in this case, the player). This model is where the NPC records previous interactions and keeps track of assumptions that are made about the listener.

Entertainment. The primary goal of any game is to be entertaining. An interesting side effect of this is that accuracy and relevance, while still clearly important, need to be weighed against other issues like entertainment value.

Demanding User. In a game setting, the person posing the query can be considered a demanding user. While they are interested in getting an appropriate answer, they are less willing to tolerate system failures and idiosyncrasies. Anything that breaks the illusion of an intelligent person answering the question is undesirable.

Computational Cost. While some QA systems are designed as a shared resource with multiple users accessing the system simultaneously, a single-player RPG environment has a dedicated system available for use. This luxury permits the use of more advanced linguistic techniques for answering the user's query.

Document Authority. As with many other RDQA systems, document authority is not an issue for RPG-QA environments. Since all documents are authored specifically for the game, they are all of equal weight and trustworthiness.

However, this issue is not to be confused with the player's

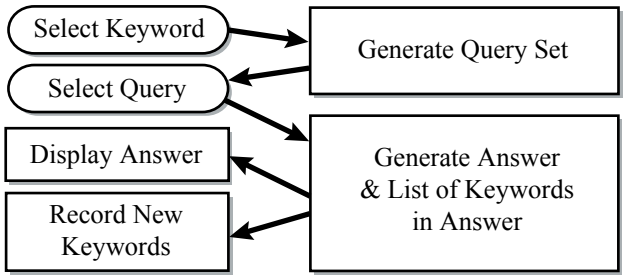

Figure 1 : Proposed QA process for an RPG.

confidence or trust in the NPC. In an RPG, a clear distinction is drawn between *document authority* (i.e., can we trust the documents to contain an accurate representation of NPC's knowledge?) and *answerer authority* (i.e., can the player trust the information that the NPC gives?). It is perfectly acceptable for an NPC to lie or withhold information if that is consistent with its personality, and it is also acceptable for the NPC's knowledgebase (KB) to contain information that is incorrect (from a more global perspective), but neither of these situations reflects poorly on the authority of the documents that comprise the NPC's underlying KB.

Proposed Architecture

Given the aspects mentioned in the previous section, we adopted an architecture that has some significant differences from a traditional QA system. These differences are caused primarily by the input requirements of the RPG environment.

No Keyboard. Most significantly, we do not rely on a keyboard for player input. Instead, we rely on a keyword-based approach where the player specifies keywords through game objects or keyword lists. These keywords are used to generate appropriate questions based on the context. In their most basic usage, keywords provide at least a "Who/What is <*x*>?" question. Other more detailed questions may be available depending on the keyword and the contents of the NPC's knowledgebase.

While this simplification severely limits the type of queries that can be asked, it also avoids non-trivial input validation problems such as typos, paraphrases, reference resolution and out-of-domain input. In addition, it provides a performance baseline: keyword matching is a standard fallback position when more advanced interaction techniques (like speech recognition or parsing) fail.

Automatic Question Formulation. Because players are only able to specify keywords, we must provide a mechanism for automatically generating a set of questions from the keyword. While generating the default "Who/What is <*x*>?" query is trivial, more interesting questions are more challenging. Fortunately, because we know that the player's goal is to solve the game, we are in a reasonable position to generate useful and relevant queries.

Overview. Figure 1 provides a high-level overview of the interaction that we propose for an RPG-QA system. In this figure, the interaction starts with the player selecting a keyword in the UI and sending it to an NPC. From this keyword, a set of NPC-specific questions is generated and presented to the user. The player then selects one of these "pre-answered" questions and receives an answer.

As shown in the figure, these answers may also be associated with keywords that the player can use in future interactions. For example, if the answer mentions a key location in the game, the keyword associated with that location would now be available to the player in future interactions. In addition to being useful for interaction, the keywords are also useful for evaluating the QA system, as discussed in the next section.

Evaluation

Since the goals of game QA systems differ from the goals of more typical QA systems, it is not surprising that the standard evaluation techniques are not sufficient for this application.

Evaluation Criteria

We divide the RPG-QA evaluation criteria into two distinct groups. The first group contains the standard criteria that can be applied to most QA systems. The second group contains criteria that are specific to a gaming environment. In addition, we also briefly discuss the issue of "entertainment value" and how this impacts the evaluation.

Standard QA Criteria. Most QA evaluations involve creating a random sample of question/answer pairs and using this to judge the overall accuracy of the system. As discussed in the next section, this approach is not well-suited for evaluating QA in an RPG environment.

However, there are two measures that are common to all QA systems that are appropriate for RPGs: the *speed* and the *fluency* of response. The speed of the response is of obvious importance: we must ensure that the system does not take an inordinate amount of time to respond. Fluency is equally important: if the response is unintelligible or of low quality, the user is likely to be dissatisfied. To measure fluency, we can use the perplexity of the response relative to an *n*-gram language model. Eventually, as the interaction evolves toward a more dialog-like interaction, a *dialog fluency* evaluation may also need to be considered.

RPG Criteria. A more interesting set of evaluation criteria are those that are specific to the RPG environment.

As mentioned earlier in this paper, current RPGs rely almost exclusively on scripted interactions. Even though there are obvious disadvantages to the scripted approach since the NPC cannot act outside of the scripted dialog, there are many advantages to the game creators that outweigh

these problems. The most notable of these are (1) ability to ensure the game is solvable, (2) consistent behavior, (3) ease of testing, (4) ease of localization, and (5) reliance on proven, efficient technology. These advantages form an interesting base from which we can develop an evaluation system for RPG-QA systems.

The most important constraint on an RPG-QA system is that it must result in a game that is solvable. We address this through the use of key queries and keywords.

Key Queries. Key queries are those queries that are crucial to the player's goal to solve any of the quests in the game. In the proposed architecture, a *key query* will always produce a *keyword* that is required to progress to the next stage of the quest. Given this, we can evaluate the solvability of the game by tracking the keywords and ensuring that the system is never placed in a state where a required keyword is not attainable.

In practice, we can evaluate the overall accuracy of the QA system by exhaustively testing all of the key queries at each point in the game and verifying the results. For RPGs that update the KBs during gameplay, a sampling of non-key queries are also evaluated to ensure that they do not interfere with the key query results.

This combination of key queries and keywords gives us an automated method for evaluating the game's solvability requirement.

Entertainment Criteria. As mentioned earlier, the main purpose of any game is to entertain the player. As long as the results produced by the QA system are reasonable and entertaining, they can be acceptable even if they are wrong from a strict QA point of view. However, quantifying entertainment value and balancing this with system accuracy is a difficult proposition.

To address the entertainment value issue, we therefore assume that the game designer has created an entertaining game and then we identify aspects of the QA system that might make the game less entertaining. By identifying elements that frustrate the player we can create evaluations to ensure that we don't inadvertently make the game less entertaining. The key query evaluation is a good example of this: creating a game that cannot be solved can be very frustrating for the player, so we created an evaluation to ensure that this cannot happen. The speed evaluation is another example since a slow system can also be quite frustrating.

Our System

The next two sections provide a brief overview of the current version of our system, which implements a subset of the architecture described in the previous section.

The system is divided into two major components: the user interface (UI) and the backend. The UI consists of either a graphical or a command line interface that allows the player to interact with the backend processor responsible for producing the NPC responses and managing the KBs.

An advantage of this division is that it facilitates experimentation with various backend systems. It also allows us to make the front-end UI available to other researchers and thus provide a common environment for evaluating different approaches to this problem.

KEYSTONE: The User Interface

KEYSTONE is the name for the UI component that handles all of the interaction with the player. We created two versions of the UI, a graphical interface and command line interface. The former is appropriate for situated evaluations and demonstrations, while the latter is more appropriate for automated evaluations and debugging. These interfaces are interchangeable since they both use the same mechanisms to communicate with the backend.

The primary motivation for creating this interface tool is the lack of an RPG creation toolkit that is suitable for QA researchers. Existing tools for creating RPGs provide adequate support for customizing the appearance and the structure of the game but make the crucial assumption that all NPC dialog interaction will be scripted in a dialog tree. Because of this limitation, these tools are inadequate for researchers interested in experimenting with different approaches to NPC interaction and control.

Plug-in API

There are two primary motivations for creating separate UI and backend components. The first is to make it possible to replace the back-end and experiment with alternate NPC control schemes. The second is to facilitate automated evaluations. Both of these goals are enabled by an application programming interface (API) that defines how the front-end UI and the backend can communicate with each other.

The API is primarily focused on supporting the keyword manipulation and query generation architecture described earlier. It also supports optional full-text queries and provides support for debugging commands. A full description of the complete API is given in [Kacmarcik05].

Backend: The NPC Controller

The backend is the part of our system that performs all of the interesting QA tasks. Everything that is not part of the UI falls into the domain of the backend.

We describe here the two most relevant parts of our backend implementation: the knowledgebase, and our approach to the question and answer generation tasks.

Document Pre-processing : Creating KBs

The relatively small number of documents in the collection makes it reasonable to perform extensive pre-processing to convert the data into a more suitable representation.

Source Text. Since a long-term goal of our system is to make it accessible to game designers without requiring specialized training in a formal knowledge representation language, we use natural language text as the source representation for the KB documents. We use a parser to convert this text into a more abstract semantic representation.

Reference Resolution. Before we parse the source text, we require that all of the anaphoric references have been resolved. Beyond the nominal/pronominal distinction, there are two types of anaphora present in our system: *endophora* (references that can be resolved within the text) and *exophora* (references to objects in the game environment). While we need to rely on the game developer to provide the referents for the exophoric references, we can partially automate the resolution of the endophoric references using standard techniques. However, since we require 100% accuracy, we manually review and correct the references to ensure that the KB contains valid information.

Parsing. The parser creates a predicate-argument style tree representation of the original text. These trees, called *logical forms* (LFs), are the representations that are stored in the KB.

This tree structure has many advantages. First, since it is based on our broad coverage grammar ([Heidorn00]) it provides a reasonable representation for all of the things that a player or NPC is likely to want to talk about in a game. In addition, it is easy to manipulate by splicing subtrees from one tree to another. Finally, we can also readily generate output text from this representation using our generation component (described in [Aikawa01]).

Question Generation

Since we only allow automatically generated queries, we need to ensure that we generate a set of interesting queries to avoid frustrating the player. Beyond the straightforward "Who/What is this?"-style of questions, we also use a Wh-question generator (originally described by [Schwartz04] in the context of language learning) to produce a set of answerable questions about the selected object.

Once the player selects a query, the final step in query generation is to create the LF representation of the question. This is required so that we can more easily find matches in the KB. Fortunately, because the queries are either formulaic (e.g., the "Who/What" queries), or extracted from the KB, the LF is trivially created without requiring a runtime parsing system.

A problem with our current implementation is that we have no mechanism for sorting or prioritizing these queries. This leads to a situation where we have an overwhelming number of possible queries, many of which are trivial and uninteresting to the player. We partially address this problem with heuristic filtering, but more work needs to be done in this area.

Answer Generation

After a question is posed to an NPC, the KB is scanned to find an appropriate response. This is done by matching a combination of LF subtrees and coreference identifiers between the query and the KBs associated for this NPC.

After finding an appropriate match, the response is expanded using any deictic relations that are stored in the NPC's KB. This allows us to individualize the responses for each NPC.

One limitation in our current implementation is that we always return the minimal subtree match. This is clearly not an optimal solution since the minimal match may not be the most informative, but we opted for this approach to avoid the problems associated with generating answers that are too long, as might happen in the case where the NPC has a lot of knowledge about the object in question. As with question generation, this is an area that requires more work.

Conclusions

We have given a general description of the issues associated with creating a QA system within an RPG environment and also provided an overview of our initial explorations in this area. We have also proposed some testing and evaluation methodologies that can be used to ensure that the system is providing a level of accuracy that is appropriate for a gaming environment.

References

[Aikawa01] T. Aikawa, M. Melero, L. Schwartz and A. Wu. "Multilingual Sentence Generation". In *Proceedings of 8th European Workshop on Natural Language Generation*. Toulouse, France. 57-63. 2001.

[Green61] B.F. Green, A.K. Wolf, C. Chomsky, and K. Laughery, "BASEBALL, an automatic question answerer", in *Proceedings of the Western Joint Computer Conference* 19, pp. 219-224, May 1961.

[Heidorn00] G. Heidorn. "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. 2000.

[Kacmarcik05] G. Kacmarcik. "The Keystone API". Forthcoming.

[Schwartz04] L. Schwartz, T. Aikawa, and M. Pahud. "Dynamic Language Learning Tools". In *InSTIL/ICALL Symposium 2004: NLP and Speech Technologies in Advanced Language Learning*. Venice, Italy. June 2004.

[Winograd72] T. Winograd. *Understanding Natural Language*. Academic Press. 1972.

[Woods73] W.A. Woods. "Progress in Natural Language Understanding - an application to lunar geology". *American Federation of Information Processing Societies (AFIPS) Conference Proceedings*, 42. pp.441-450. 1973.