

The Dynamic Conversation Engine

The Dynamic Conversation Engine is a new design for video game interactions between players and Non-Player Characters (NPCs). It stresses individualized responses and dynamically generated content.



Why is a new system needed? There are several current methods of processing interaction between players and NPCs and they all work just fine, right?

No. They don't. They all have their advantages and disadvantages, but none of them are even close to the level of immersion the Dynamic Conversation Engine can provide. Graphical processing technology has increased leaps and bounds since the advent of gaming, but player/NPC interaction technology has barely progressed at all. It's time for this to change.

Note: this document is intended to provide a very general overview of the Dynamic Conversation Engine. It is not intended to be a comprehensive design document that allows the Dynamic Conversation Engine to be directly implemented in a game.

Contents

Brief History of Player/NPC Interaction	2
Designing Something Better: the Dynamic Conversation Engine	4
Information Storage in Databases	9
Syntax in Databases	10
The News Learning Engine	11
Needs for the Dynamic Conversation Engine to be Successful	13
Implications of the Dynamic Conversation Engine and the News Learning Engine.....	14
A Possible Business Model for the Dynamic Conversation Engine.....	15
Building on the Dynamic Conversation Engine: Adding even more Depth	16



Brief History of Player/NPC Interaction

Before discussing the specific details of the Dynamic Conversation Engine's design, first a brief history of current methods of dealing with player interaction with NPC's is needed, if only to outline the differences between the Dynamic Conversation Engine and current methods. The history presented here is a basic review – it isn't comprehensive. It focuses on games that make player/NPC interaction an important aspect of their gameplay. For the most part, this means the Role Playing and Adventure genres, since the First Person Shooter and MMO genres usually offer a very thin and one-sided level of interaction. However, a slimmed down version of the Dynamic Conversation Engine could be used in those genres as well.

Direct written conversation. This system is used in text-based story games and old role-playing games. There interaction is only one-sided, coming from the NPC to the player, but without input. This is the least immersive of the systems, but is better than nothing, and can still provide the player with a deep story.



Predetermined conversation trees. This is similar to direct written conversation, but gives the player choices as to which conversational path he would like to take. This is often used in adventure games, such as FunCom's *The Longest Journey*. It is more interactive, but it is much like taking a multiple choice test: both the questions and their answers are written by a writer. This system gives the player a small amount of choice – he can ask questions about aspects of the story he is interested in, but if the option is not given, he cannot ask.



Keyword or journal-based. In this system, each NPC is given access to a database of knowledge and the player asks about relevant keywords. The responses are written beforehand by writers and often are repetitive, due to the amount of content that must be hand-written. Bethesda's Elder scrolls series, specifically *Morrowind*, uses a system like this. There are several problems with this system: the unlocks many keywords as the game progresses causing



each NPC to present too many options. This can lead the player to feel overwhelmed. The interface is also text-based, and messy. NPC responses are often repetitive, making the conversations also feel unnatural as NPCs repeat the same thing over and over. Bethesda's *Oblivion* and *Fallout 3* use a similar system to this, though they clear up many of the interface issues by attempting to only present relevant responses. However, the player protagonist is often silent in these systems, and although this silence is intended to increase the immersion by not imposing a character's personality on the player, it has the danger of making the player feel as though his has no personality.



Adapted conversation trees. Examples: Bioware's *Mass Effect*. This type of system is currently the best and most immersive conversation engine. This system uses vast amounts of hand-written dialogue, serving up multiple choice options. This system is a mix of predetermined conversation trees and the keyword based system, allowing the player a much more immersive experience. However, the topics are limited and players can still feel frustrated by limited options. The conversation trees are also updated and rearranged based on news from other sources, such as the player completing quests. This system can feel immersive at first, but eventually the player learns its shortcomings and is able to see through the system to the underlying mechanics.



Next, we'll look at a new type of conversation system, the Dynamic Conversation Engine.



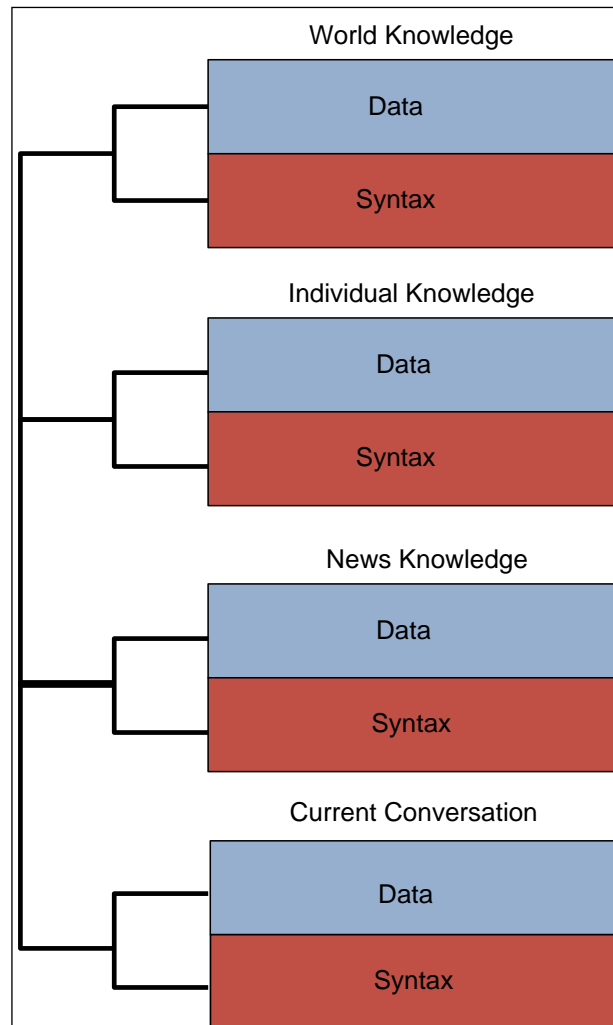
Designing Something Better: the Dynamic Conversation Engine

The Dynamic Conversation Engine is a new conversational system. It is designed to dynamically generate conversational content in real time, much like modern graphics engines generate lighting and graphics. Conversation is generated by a series of language algorithms and knowledge databases tied to each NPC. The information each NPC could access is stored in a series of tiered databases or libraries, shaped by their language and syntax algorithms, and returned to the player in the form of dialogue.

Because of the complicated nature of this system, in order to optimize speed and also help determine priority, each NPC would have access to four separate databases, arranged in a tiered structure.

Each of the four databases would be separated into two specific parts, knowledge and syntax. The knowledge half would contain data and the syntax section would contain language formation algorithms and speech idiosyncrasies, as well as basic language and structure.

1. The World Knowledge database would contain basic knowledge of the world, things that all members of the world would know: cultural ideas, state capitals, basic history, and so on. The World Knowledge syntax would contain the basic tenants of the language – sentence structure, grammar, etc, for the language. The World Knowledge database is the same for all NPCs in the same world.
2. The Individual Knowledge database would contain specific information related to that NPC: personal history, local knowledge, likes and dislikes, relationship information, etc. The Individual Knowledge syntax would contain idiosyncrasies of language related to that character – accents, word preferences (perhaps set by an “intelligence” level), etc.
3. The News Knowledge database would contain news that the character would be aware of that would shape its responses: knowledge of the player,



knowledge of the player’s previous deeds (quests, reputation, etc.), changes in the world since its inception. The News syntax section contains the language that deals with those new events, such as player created labels or names. The News database can receive updates from the News databases of other characters.

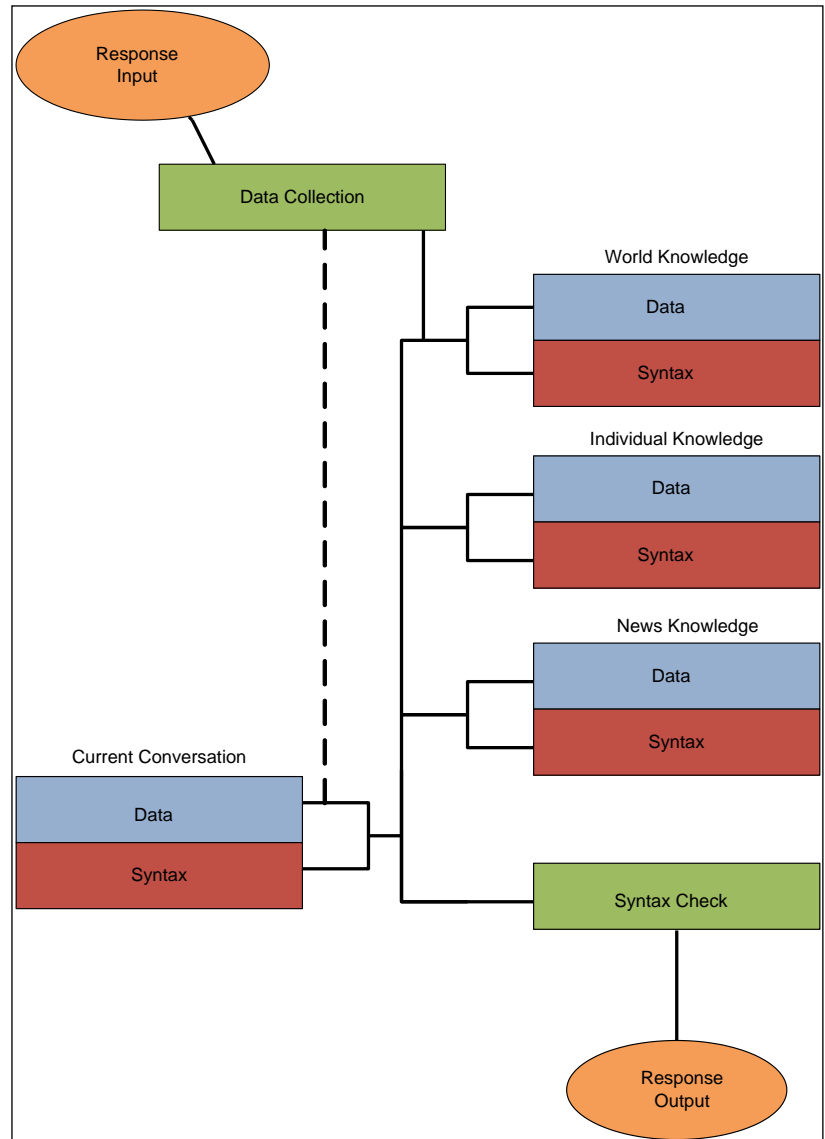
4. The Current Conversation Knowledge database contains information related to the current conversation the NPC is engaged in with the player. It functions as a short-term memory for the NPC, allowing the player to interact with the NPC by asking about a recently discussed topic. The Current Conversation syntax allows the player to label a concept in a certain way in the conversation, and have the NPC, once it learns the player’s labels, to adopt them.

All four databases are tiered in priority for the amount of influence they have on the responses, with 4 taking preference over 3, 3 over 2, and 2 over 1.

Each time a comment or question is asked of an NPC, a data package, called a “response” is created. The response initially contains several things: the question asked, the time of day, the NPC’s present location, the time of day and any other relevant information.

The response then moves to Data Collection, which sorts the question into relevant keywords. The keywords are referenced from database 4 (Current Conversation), which stores the lingo of the current conversation. This allows the player and NPC to build a shared diction together, using standard pronouns for previously discussed concepts. For instance, if the player and NPC are currently involved in a conversation about the “king”, using the data sorting and the referenced information from database 4 (Current Conversation) “king” can be stored as “he” for both input and output. This allows for a more natural conversation.

After sorting and definitions, the response moves up to database 1



(World Knowledge) and gathers the information relevant to its response. This database contains general background information, as well as language syntax. The response will then move up to database 2 (Individual Knowledge), repeating the process and narrowing down the response to more specific information and syntax. Then database 3 (News Knowledge), then 4 (Current Conversation), each time refining its response with more specific information.

The last database accessed takes precedence, meaning that if the response finds relevant information in database 4, it will use that, but if not, it will use what it got from 3, or 2, or 1. This means that the NPC should always have something to say on the subject, even if it is just general world knowledge. After accessing all databases, the response passes through a final syntax check to make sure the grammar is correct, and reports the response back to the player.

Here is an example of the Dynamic Conversation Engine in action:

The player is talking to an NPC, Garoth Nardic, a traveling trader. The player encounters the traveling trader on the road, after just wandering into a new country. The player is curious about the place and stops the trader to ask some questions.



Player: What country is this?

Garoth Nardic now creates a “response” data package. The data package takes the query from the player and also grabs any other important information that might be relevant to the answer (current location, time of day, etc).

After the response is created, the response moves to data collection. Here the player’s query is sorted into relevant keywords, and those keywords are also assigned to conversational variables (if applicable) using information stored in database 4 (Current Conversation).

After the response is sorted and defined, it is sent to the top of the NPC’s database tree. In database 1 (World Knowledge), Garoth Nardic’s response finds that the name of the country he is currently in is called “The Dotesian Empire.” Inside database 1 Garoth Nardic’s response also learns how to relay this information by accessing the language syntax section of database 1. His response becomes, “This is the Dotesian Empire.” Garoth Nardic’s response now leaves database 1 and goes to database 2.

In database 2 (Individual knowledge) Garoth Nardic’s response learns that Garoth Nardic doesn’t like the Dotesian Empire. It also learns that Garoth Nardic is open about that fact, so would be willing to tell the player that, even though the level of trust for the player is low, since they just met. The syntax section of database 2 tells the response that Garoth Nardic is more likely to use adjectives than separate sentences. Thus, Garoth Nardic’s response becomes, “This is the accursed Dotesian Empire” and moves up to database 3.

In database three (News Knowledge) Garoth Nardic’s response learns that the Dotesian Empire was recently invaded by goblins. The syntax section of the news database tells the response that this goblin invasion is being called “the brown plague.” Thus, Garoth Nardic’s response becomes “This is



the accursed Dotesian Empire. What's left of it, anyway, after the brown plague." and moves up to database 4.

In database 4 (Current Conversation) Garoth Nardic's response learns that this is the first question asked by the player, so nothing new needs to be added to the response. Garoth Nardic's response stays the same and moves up to the final syntax parser. However, before moving up to the syntax parser, the response updates the information in the Current Conversation database to include what it learned. The current topic is the Dotesian Empire and the brown plague was recently mentioned.

The syntax parser does a final check to make sure the grammar is correct, then returns Garoth Nardic's response to the player.



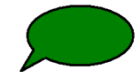
Garoth Nardic: This is the accursed Dotesian Empire. What's left of it, anyway, after the brown plague."



Player: Who is the King?

Although the player does not ask what county he is referring to when asking for the king, the response collects data from the previous question in the data collection phase. This means that the Dotesian Empire is assumed as the answer to the question: "what country?"

Including the previous question as part of the initial data package means that in database 1, the response uses the previous question's keyword's, specifically "Dotesian Empire" and returns the king of the Dotesian Empire up to the next level of the tree. If Garoth Nardic happens to have a personal or news-based opinions on the current king, that information will also be included. Maybe Garoth Nardic hates the king, or the king was recently killed in the brown plague. These bits of information can be added to Garoth Nardic's response as it moves up the database tree. The Current Conversation database is expanded to include that this information has been relayed to the player.



Garoth Nardic: The king is of Dotesian Empire was Jero Caesar, but he is dead.



Player: How much is that sword?

The response to this question finds nothing to relate to in the data collection phase, since the question is not related to the previous lines of conversation. Hence, it begins down the tree with no included data. Nothing is found in database 1 except for general information about swords, thus it creates a response based on what a sword is.

However, when moving up to database 2, the response finds the specific pricing information on Garoth Nardic's wares, and alters it's response to include that information. Depending on the level of trust the response holds for the player, this may also alter the price.



Database 3 holds no relevance to the price of the sword, unless Garoth Nardic is charging more for weapons because of the brown plague. Database 4, the current conversation, also holds no relevance, since the player has not pursued this line of questioning before.

 **Garoth Nardic: That sword is 100 gold pieces.**

 **Player: I'm looking for Gerald Niplock. Do you know where he is?**


The response captures the data it needs, then starts at the beginning of the tree. Again, this is an out of the blue question, so no previous information is relevant.

Database 1 tells the response nothing. It finds no information on the name, so its response is “I don't know who that is.”

However, in database 2, the response finds the name listed, meaning that Garoth Nardic knows Gerald Niplock and thinks that Gerald Niplock's current location is in Dotensas, the capital of the Dotesian Empire. Depending on the level of trust Garoth Nardic has for the player, or how much he cares about Gerald Niplock, his response chooses how much information to give the player.

Garoth Nardic doesn't care about Gerald Niplock, so his response chooses to tell the player what he knows, shaping his words as defined by his particular syntax patterns, “Last time I heard, Gerald Niplock is in Dotensas.”

Database 3 and 4 yield no alterations, so the response passes through syntax checking and returns to the player.

 **Garoth Nardic: Last time I heard, Gerald Niplock is in Dotensas.**

The player ends the conversation there and heads down the road. This is a simple example, but demonstrates the power of the conversation engine. The inclusion of current data allows for more fluid conversations, a conversation that swings from point to point based on previous conversation topics, mimicking real conversation. Also, it allows for NPC's the easily tailor their responses to local variables: time of day, location, etc, without needing a writer to rewrite a line for each possible contingency.

This is a good start, but only a small part of the power of the Conversation Engine's design. In the next section, the News Learning Engine will be detailed.

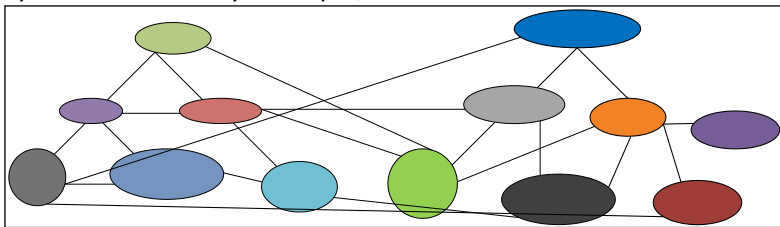


Information Storage in Databases

As previously mentioned, information stored in the databases is split into two sections, knowledge and syntax. Information is stored in a three dimensional cross-referenced web structure, allowing for keyword based access, but also connections between different sections of the web. Much in the way that memory is interconnected within human brains, information in the knowledge section of the database is cross-referenced. This is difficult to show on a 2D surface, so please bear with the humble graphics and examples.

Information in the database is stored based on keyword association, allowing any types of information to be included, with any amount of depth. For an information node that an NPC only had a small amount of knowledge about, only a few associations would need to be included. Although a standard template could be used for regularly used nodes (counties, characters, etc), it would be easy to adapt the templates with more information simply by adding more keyword associations. Because the system is keyword based, the information included need not be just statistic-like information, because keywords can convey concepts, such as attitudes or emotions.

Each keyword association would function both as a word-unit and a link to other concepts, much like a hyper-link in a document or a pointer. Thus, the web can be built by connecting concepts to each other and allowing the NPC to make connections it would otherwise not be able to make. Using a synonym seeding system would allow a quick and versatile way to fill multiple keywords and tags into the system. Once the system is varied enough, more connections can be added.



Field	Data	Keyword Tags
Country Name	Dotesian Empire	country, land, place
Founding Date	Jerd 16 1156	date, began
Enemies	Taluria, Yoberland	enemies, enemy
Friends	Kloveria	friends, ally, allies,

Nodes for important storyline elements could also be used to include phrases, though these should be used very sparingly if they are not to become recognizable by players very quickly.



Syntax in Databases

As for the syntax section, standard grammar includes a system of language that is already based on rules, and those rules can be adapted to the language creation algorithms. There are several ways to handle this, based on an increasing complexity.

Each keyword will need to be labeled with as a certain part of speech, which can be done by importing words from a standard dictionary. The simplest method is to use a very simple “mad libs” like sentence generation, beginning with the simplest type of sentence: Noun Verb.

Once that level of complexity is established and the engine can reliably develop two word sentences based on a Noun Verb model, adjectives, adverbs, conjunctions, and other more complex syntax rules can be added until sentences feel natural enough to replicate human speech.

Although interesting and dynamic methods of syntax information is preferred, standard sentence templates can be added if necessary. A system could be created that selects the level of sentence complexity needed based on the number of keywords that need to be returned back to the player. If the template method is used, many templates should be created and used randomly, that way the language still feels naturally generated.

Ideally, however, the syntax algorithms should be created to dynamically arrange their grammatical components into templates themselves, thus allowing for more random speech patterns. An analysis of sentence structure could also be used to determine percentages of different types of syntax patterns and generate varying types of sentence structure based on those percentages.



The News Learning Engine

The News Learning Engine is an aspect to the Dynamic Conversation Engine that allows NPC's to dynamically update their database files through conversations with the player or outside news sources (read: other NPC's). As NPC's converse with the player, they may change their attitudes or opinions on certain topics. They must also be able to react to news of the player from other sources, such as the player completing an important quest, or the player deposing a noble, etc.

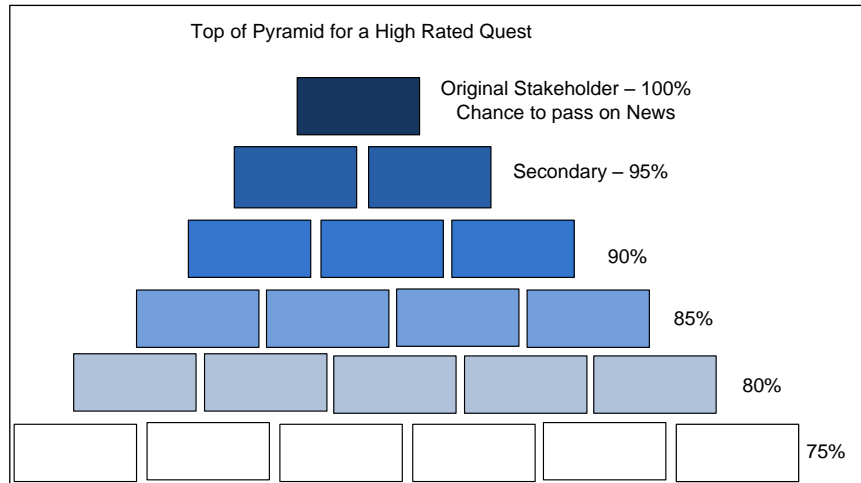
The Learning Engine is a percentage-based system that allows data in a certain database to move to other databases, both within a singular NPC or to other NPC's.

For example:

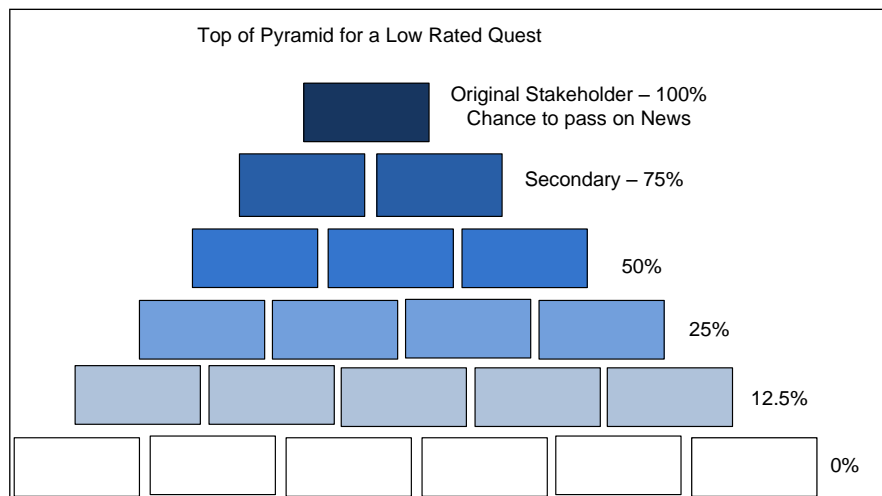
This example assumes mobile NPCs or a relationship system built into database 2 (Individual Knowledge) of each NPC.

The player completes an important quest, such as exposing a plot to murder the king. The player is declared a hero in the country. All the databases of the NPC's in the country are seeded with

knowledge of the player and his heroic deed. They now have access to that information, via their News Knowledge databases (database 3).



That's a big example and not very interesting example, so let's look at a smaller one. Say a player completes a small, low-rated quest. He rescues a cat from a tree for a little girl. She's happy. Because the quest is low-rated,



all the NPC's in the country are not seeded, but the girl's parents (the relationship information stored in her individual database) are seeded with knowledge of the deed. Other NPCs that come in contact with the girl or her parents have a chance of being seeded with knowledge of the deed, based on the impact rating of the quest. Because the impact of the quest is small, only a small percent of other NPCs will be seeded with knowledge of the deed. Each time the knowledge is seeded to a new NPC, the rating is further lowered: decreasing the chance it will be passed on again. As time goes on and NPCs spread throughout the world, knowledge of the deed will spread, slowing down as it get further away from its original stakeholder, the little girl.

The interesting thing about this for the player is the spread of news from his deeds in an organic and interesting fashion. It also means that knowledge of the player's moral choices can be used to affect other NPCs, but only those who have heard the news of his deeds. NPCs that have little interaction with other NPCs would have little knowledge of the player, but those in busy cities which the player frequents would be well informed.

Meta-informants could also be created, NPCs who News Knowledge databases were seeded not through contact with other knowledgeable NPCs, but through the news simply existing in the world in any fashion. This would be useful for quest-givers, wizards, internet hackers, and other important information-dealing characters.

Obviously the system would need to be set up for variable quest rating percentages, so different quests can be seeded at different priorities.

In a robust implementation of this system, the player could create a new concept or term, impress upon an NPC the important of that term or concept (via a high trust rating, a bribe, or something else) and allow the information to slowly seed to the rest of the NPC population. The possibilities for a new type of quest, spreading misinformation or defamation, would be amazing and fun.



Needs for the Dynamic Conversation Engine to be Successful

A linguist needs to assist with the conversation engine's dynamic algorithm creation. The language must sound natural and be grammatically correct or it will break the immersion.

The Dynamic Conversation Engine needs to use voice recognition technology to take the input. Typing will not keep up with player standards, though type would be fine for an initial prototype. However, for game usage, typing questions is tedious and breaks the immersion. Most players will not put up with it, even if it allows for deeper conversations.

Synthesized voice technology would also need to be at a higher standard than it is now, in order for the voices to be dynamically generated back to the player. Although text output back to the player is an option, players have gotten used to voiced responses. However, recording all possible combinations would be completely unfeasible and limiting. Voices need to be generated through synthesis.

It's possible, depending on the hardware demands on the processor and hard drive, that current technology is not up to par with what the Dynamic Conversation Engine requires. However, the time it will take to develop the Dynamic Conversation Engine, prototype it, and utilize it within a game, should give more than enough time for the technology to catch up, or even be pushed forward by demand for engine support, much as graphical engines push forward graphical processing technology.



Implications of the Dynamic Conversation Engine and the News Learning Engine

The Dynamic Conversation Engine creates the possibility for a level of game immersion for players the likes of which have never been seen.

The Dynamic Conversation Engine changes the role of the game writer from a direct writer of dialogue to an idea-smith who seeds NPC databases with information relevant to their characters, rather than directly crafting the dialogue. Since the Dynamic Conversation Engine is the entity actually producing the dialogue, all that's needed to create an interesting, versatile new NPC is to seed the NPC's database with information relevant to that character. This seeding can be as detailed or as sparse as the writer wants it to be, because each NPC will always have information from the general World Knowledge database to back up any holes in its knowledge and syntax.

The Dynamic Conversation Engine, because it is expandable and updateable, also offers an interesting way to enhance AI technologies. Because the information is sorted in a way to prepare it for communication, it gives an interesting look at ourselves and the way we sort information. The Dynamic Conversation Engine could be used for other things, not just games, but also as a searchable database of any type of knowledge.



A Possible Business Model for the Dynamic Conversation Engine

Depending on the ability of home processor technology and data storage, and because speed is always a major concern, the amount of storage needed or processing power needed to process the Dynamic Conversation Engine may not be available for home users. However, this does not mean that the Dynamic Conversation Engine is not yet possible. Another way is feasible.

Cloud computing, that is, data processing on a central server, using the home user's computer as a thin client, could be an option for the Dynamic Conversation Engine. If the home user's machine was not feasible (due to data storage and processing power constraints), a group of central servers could process the backside of the Dynamic Conversation Engine and stream the response over the internet back to the user's home machine.

Aside hardware constraints on the user's end, this model has two distinct and important advantages over running the code on the home user's machine:

1. The code is not stored on the home user's machine at all, thus keeping the technology that runs the Dynamic Conversation Engine secret and away from the hands of competitors.
2. The Dynamic Conversation Engine could be licensed to many different games and processed from the same location, without including the code in those games. This means that the company that owned the Dynamic Conversation Engine could license their conversation service to other companies, much in the way that companies license their graphics engines.



Building on the Dynamic Conversation Engine: Adding even more Depth

This section doesn't have much in the way of explanation, but offers some quick and dirty ideas for making the Dynamic Conversation Engine even more robust. It assumes that a working version of the Dynamic Conversation Engine was already implemented and working.

Integrating a better resource (NPC ownership) and motivation system, so NPC's could dynamically create quests based on the resources they had to offer as rewards and their particular motivations. "I'll give you a cow to spread misinformation about my neighbor" or "I'll give you 100 gold pieces to assassinate my rival." These motivations would be changed and altered through News Learning Engine updates. An NPC's country goes to war with a neighboring country and a particularly patriotic noble has a pile a gold pieces. Patriotism would be an aspect of the NPC's personality that was tied to country aggrandizement and killing the enemy, so when the player asks about killing the enemy, the NPC offers gold for killing. Because the quests are tied to an NPC's resources and motivations, any NPC with something to trade can become a quest-giver, given the right circumstances. It's up to the player to decide if that quest is worth it or not. Obviously, this system would need more thought and an already running Dynamic Conversation Engine to plan the specific details.

Other integrations are possible, as well, building off the initial functionality of the Dynamic Conversation Engine.

