January 2015

Realistic Dialogue Engine for Video Games

Caroline M. Rose
The University of Western Ontario

Supervisor
Mike Katchabaw
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

REALISTIC DIALOGUE ENGINE FOR VIDEO GAMES

(Thesis format: Monograph)

by

Caroline M. Rose

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

The concept of believable agent has a long history in Artificial Intelligence. It has applicability in multiple fields, particularly video games. Video games have shown tremendous technological advancement in several areas such as graphics and music; however, techniques used to simulate dialogue are still quite outdated. In this thesis, a method is proposed to allow a human player to interact with non-player characters using natural-language input. By using various techniques of modern Artificial Intelligence such as information retrieval and sentiment analysis, non-player characters have the capability of engaging in dynamic dialogue: they can answer questions, ask questions, remember events, and more. This conversation system is highly customizable, so the types of responses that non-player characters give can be modified to fit within a game's storyline. Although the system only currently allows for simple dialogue, it illustrates the potential for a more robust way to simulate believable agents in video games.

Keywords

Video games, non-player characters, conversation, dialogue, autonomy, believability, believable agent, intelligent agent, illusion of intelligence

Acknowledgments

I wish to acknowledge the tremendous help given to me by my supervisor, Dr. Michael Katchabaw. Without his guidance, this thesis would not have been possible.

I also wish to acknowledge Andrew Kope for his help in creating the episodic memory module.

Table of Contents

List of Tables

List of Figures

List of Plates

Chapter 1

1 Introduction

Video games are a popular form of entertainment and a multi-billion dollar industry [1]. In particular, role-playing video games (RPGs) have quite a large fan following [2]. This specific genre allows the player to assume the role of an adventurer with the purpose of progressing through a storyline. What separates this genre from others is that the player must usually navigate through a large game world and interact with non-player characters (NPCs) along the way [3].

Currently, role-playing video games excel at creating a convincing game world in multiple ways. Firstly, advances in hardware have allowed for high quality graphics and realistic physics. Secondly, the storylines are often quite immersive. Finally, the use of sound effects and music helps set the mood at any given point in time. Where these games still fail, however, is in creating believable artificial intelligence [4]. Specifically, the non-player characters do not seem like autonomous beings when the player must engage in dialogue with them. Instead, it is obvious that they are entirely scripted. Given that these non-player characters can be an integral part of the game's immersive experience, there is a need to make them act more realistically [5].

1.1 Artificial Intelligence in Games

The purpose of artificial intelligence (AI) is to make the computer perform the same thinking tasks as a human or an animal [6]. Some examples of such tasks would be the ability to make decisions and engage in realistic dialogue. Because of the complexity of human thought, creating a robust artificial-intelligence system is a difficult problem. It has been a growing field of study for several decades, but more research is still needed to reach the desired level of sophistication.

In the context of video games, the goal of artificial intelligence (just like the other elements of the game) is to entertain the player. The object is to create an immersive game world that causes the player to suspend disbelief for a period of time. In order for

video-game AI to accomplish this goal, non-player characters must have the illusion of being intelligent [7]. That is, they must be believable agents. Depending on the purpose of the NPC, different techniques are often needed to achieve this notion of appearing intelligent.

1.1.1 Movement

One way that NPCs can seem intelligent is by making sensible movements. For example, if a character without a ranged weapon needs to attack the player, it should first move to be within melee range of the player. Otherwise, it would end up hitting the air. Another example of movement would be navigating around obstacles to reach the desired destination, such as the location of an alarm if the player is caught trespassing.

1.1.2 Decision Making

Another way that NPCs can appear intelligent is to make decisions regarding what to do next. An NPC can potentially have several different behaviours to choose from, such as attacking the player, hiding behind cover, standing still, eating, etc. Consequently, appropriate decisions need to be made at different points in the game as to what the next course of action should be.

1.1.3 Strategy

The third way NPCs can display intelligence is through the use of strategy. While movement and decision-making concerns one NPC, strategy governs the overall behaviour of all the NPCs. An example of strategy would be for the NPCs to surround the player before attacking, thus ensuring that escape is difficult [6].

1.1.4 Dialogue

The fourth and potentially most difficult way for NPCs to show intelligence would be to engage in a dialogue with the player. As will be discussed further in Chapter 2, dialogue in the vast majority of games is scripted. The consequence of this is that either NPCs will repeat themselves unnecessarily or they will ignore the player entirely. Once they run out of new things to say, they will no longer seem like autonomous beings.

Given that the expectation of an NPC is to act like a human in a convincing manner, then ideally, an NPC should be able to pass the Turing Test. It can pass the Turing Test "if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer". To be able to accomplish such a feat, the computer needs the following:

- natural language processing (to understand what the human said)
- knowledge representation (to store its knowledge)
- automated reasoning (to use information to answer questions as well as derive new information)
- machine learning (to find patterns in order to adapt to new situations) [8].

1.2 Research Questions

This thesis will explore the following research questions:

- Is it possible to integrate current AI research results in a real-time conversation system?
- Is it possible to create a general-purpose conversation system that could be used in several different video games?
- To what extent does such a system function and perform?

1.3 Motivation for Research

As mentioned in Section 1.1, having an NPC be able to engage in realistic dialogue with a human player would add to the impression of it being an intelligent agent. Currently, NPC responses are entirely scripted in a large number of games, which breaks the illusion of autonomy. If there could be a way to allow the human player to interact with NPCs in a similar manner as how they interact with other human beings, then the NPCs would be highly credible as believable agents. Some common ways that humans interact with each other are face-to-face, on the telephone, and via instant messaging. Therefore, it is sensible to try to simulate similar forms of interaction between a human and an NPC.

Given the desire to allow the human player to interact with NPCs in a freeform way (as opposed to them always being required to select possible utterances from a menu like in current video games), the use of natural language processing and information extraction will be required. It should be stated, however, that the purpose of this thesis is not to invent new techniques in either field. There is already research being done on how to solve various tasks such as part-of-speech tagging and sentiment analysis. Instead, this thesis assumes these modules as given (and also makes the assumption that they work perfectly even though they do not necessarily) and develops a framework for natural game dialogue by integrating these elements into believable agents.

Tremendous research has already been done in Artificial Intelligence, yet video games continue to use outdated techniques for NPCs to interact with human players [9]. There are a few reasons why game designers are hesitant to improve dialogue AI. Firstly, it is a difficult research problem, and they would rather not have to solve it themselves. Instead, they would prefer for such a system to be readily available rather than create it in-house. Secondly, they are not confident that the current techniques are sophisticated enough to simulate realistic conversations, so they prefer to approximate dialogue with scripted techniques. Finally, they are not convinced that such high-level AI is actually necessary because they believe that video games are entertaining enough with their current AI [3].

If a system could be developed that could make NPCs act like human beings in a credible way, then it would add a new dynamic to video games. As mentioned previously, the purpose of video games is to entertain. Having this new element of gameplay would add to the entertainment value of those games where interacting with NPCs is necessary to progress through the storyline. Since the NPCs would act reasonably autonomously, the game world would be more immersive as a consequence, and the suspension of disbelief would not be broken as easily.

1.4 Proposed Method

The purpose of this thesis is to create an innovative new dialogue engine for games, leveraging existing results of research in computational linguistics, as well as to illustrate the potential level of sophistication that could be offered. The aim is for game developers

eventually to have an easy-to-use, plug-and-play system that could either be used as is or could be customized by them as desired. Such a system would allow them to enrich the game experience without requiring too much effort on their part.

The player should be able to interact with an NPC by inputting either questions or statements via a keyboard. The NPC would then parse the query and provide an appropriate response based on the nature of the input.

1.4.1 Types of Input

The NPC should be able to recognize the following types of input:

- Questions
    - "W6" questions (i.e. "Who", "What", "Where", "Why", "When", and "How" questions)
    - Yes/No questions
- Assertions (i.e. the human player giving seemingly factual information to the NPC)
- Compliments/Insults directed at the NPC

While this list is certainly not exhaustive, it offers a good foundation from which to build a framework because these are standard speech acts [10]. However, a framework should still be flexible enough to expand this list in the future if required.

1.4.2 Expected Types of Responses

In the event that the human player asks one of the W6 questions, then the NPC should reply with either a phrase or sentence that answers the question in its entirety. If the human player asked a Yes/No question, then the NPC should reply with either "Yes" or "No". In the case of a "No" answer, it might also be helpful for the NPC to add some extra information to clarify why the answer is "No". If the player has made an informative statement, then the NPC should acknowledge that something was said and

store the information in its knowledge base. If the player has made either a compliment or insult towards the NPC, then a response that illustrates that the NPC recognized the statement as a compliment/insult should be made (e.g. “Thank you” for a compliment or “That wasn’t very nice” for an insult).

1.4.3 End of Conversation

A dialogue should continue until the player has given an indication that s/he wishes for the conversation to end (e.g. by saying “Goodbye.”).

1.5 Structure of Thesis

This first chapter discussed some of the ways in which non-player characters can appear intelligent. It also mentioned why they may cease to appear autonomous when engaging in dialogue with the player, as well as giving an outline to a proposed solution to this problem. Chapter 2 highlights the state-of-the-art techniques currently used in games to simulate dialogue. Chapter 3 discusses some of the state-of-the-art techniques in Artificial Intelligence. Chapter 4 proposes a model for conversation. Chapter 5 presents an architecture for the model proposed in Chapter 4. Chapter 6 discusses an implementation of the framework presented in Chapter 5. Chapter 7 mentions the methods used to test the implementation discussed in Chapter 6 and also highlights the results of the testing. Chapter 8 presents a conclusion and discusses future work.

Chapter 2

2 State of the Art in Video Games

As briefly mentioned in Chapter 1, most dialogues in current video games are simulated using scripted techniques. “Scripted”, in this sense, means that the response of an NPC was pre-written by a human author instead of being generated dynamically by the video game. Scripted techniques do not require a sophisticated knowledge of artificial intelligence to use them, making them easier to implement than more advanced AI techniques. However, the major downfall is that NPCs can cease to appear autonomous after a while due to the highly deterministic nature of their responses.

The types of dialogue systems used in video games could be classified into three main types: cutscenes, branching dialogue trees, and simple natural language processing. Each of these types has advantages and disadvantages to their use, as will be explored in the next few sections.

2.1 Cutscenes

The easiest way to simulate dialogue is to give the human player no choice (or virtually no choice) in what can be said to an NPC. Video games of this type use cutscenes, which are scripted sequences that occur at different stages of the game. The sole purpose of cutscenes is to advance the narrative of the video game. Due to the nature of cutscenes, they are “passive and non-interactive, as opposed to the interactive nature of gameplay” [11]. Cutscenes do not require any Artificial Intelligence to generate the dialogue because the script was written out entirely beforehand by a human author. Consequently, cutscenes are akin to movie clips being inserted in the video game. Two popular video games notable for their use of cutscenes are *Halo* and *Call of Duty*.

Cutscenes are regularly used in several AAA game titles, but their use is considered controversial, and many gamers and even some game developers oppose them. Chet Faliszek, a Valve employee who worked as a writer for *Half-Life* and *Portal*, believes that gamers do not have the patience to sit through cutscenes. Thomas Grip, the lead programmer for the popular horror game *Amnesia: The Dark Descent*, believes that

"There is a big difference in our relationship to a protagonist when you are a passive observer compared to playing as that character" [12]. Thus, the inclusion of cutscenes in games is becoming increasingly undesirable.

2.1.1 Classifications

Cutscenes can be organized into three main classifications: live action, pre-rendered, and in-game cinematics. The following subsections will explain these three types in more detail.

2.1.1.1 Live Action

Live action cutscenes are similar to films because, like films, they are prerecorded and feature human actors. Due to this similarity, they would often be created by third-party production companies since they required the same resources needed to make a motion picture. Occasionally, they would even feature Hollywood actors. Consequently, they are rather expensive to make.

Live action cutscenes were quite popular in the early 1990s. Some notable titles that used them were *Wing Commander III: Heart of the Tiger*, *Command and Conquer*, and *The Horde.* However, they are rarely used today because most modern-day game developers consider them too costly to produce.

2.1.1.2 Pre-Rendered

Pre-rendered cutscenes are created using higher-quality versions of character models and environments than what is seen during gameplay. They are able to feature superior graphics due to the fact that they do not need to be rendered in real time. Instead, as the name implies, they are rendered in advance.

2.1.1.3 In-Game Cinematics

In-game cinematics use the same character and environment assets as what is seen during gameplay. What distinguishes them visually from actual gameplay is the use of a cinematic camera. Thus, the visual style of the cutscene will still resemble a film, but it is actually being rendered in real time [13].

2.1.1.3.1 Quick Time Events

Often, like with the other types of cutscenes, the player must passively watch the story unfold. However, a recent trend has been the use of quick time events. Quick time events (QTE) are essentially in-game cinematics that allow for some player interaction. Prompts appear at scripted intervals indicating that the player should press a button within a short period of time. If the player succeeds, then the cutscene continues; otherwise, the player may be forced to repeat that segment again. Their purpose is to keep the player actively engaged, but players can find them distracting and restrictive [14].

2.1.2 Advantages

The use of cutscenes has the following major advantages:

- They are easy to implement.
    - Since cutscenes are entirely scripted and require little to no player interaction in a dialogue, they do not require knowledge of Artificial Intelligence to create.
- Total control over what an NPC will say.
    - Due to the linear nature of cutscenes, there is virtually no risk of an NPC saying something nonsensical or inappropriate.

2.1.3 Disadvantages

The use of cutscenes has the following major disadvantages:

- NPCs are not autonomous
    - Since NPCs are not autonomous, the player may find it difficult to believe they have emotions, yet “the emotional dimensions of NPCs is usually seen as key to preserve the immersive quality of a virtual world” [15].
- Players are becoming disinterested in them

- Players are less likely to sit through cutscenes because they would prefer to be actively playing instead of passively watching [12]. Although quick time events attempted to remedy the issue of being passive observers, their use is not popular [14].

2.2 Branching Dialogue Trees

Dialogue trees are a common technique to simulate dialogue in video games. They allow the player to choose from a menu of options of what to say to an NPC. In some games, certain branches of the tree will be inaccessible unless a particular event in the game has occurred or if the NPC perceives the player in a certain way. For example, the player may be unable to ask an evil NPC to join him or her if the NPC's perception of the player is that s/he is good. Despite the name, dialogue trees are not necessarily tree structures. Instead, they are often graphs that can have cycles. Two notable game series that use dialogue trees are *The Elder Scrolls* and *Fallout*.

Figure 2.1 illustrates a simple example of a dialogue tree. Observe that two arcs can lead to the same node. Furthermore, note that one node's sibling can also be its successor.

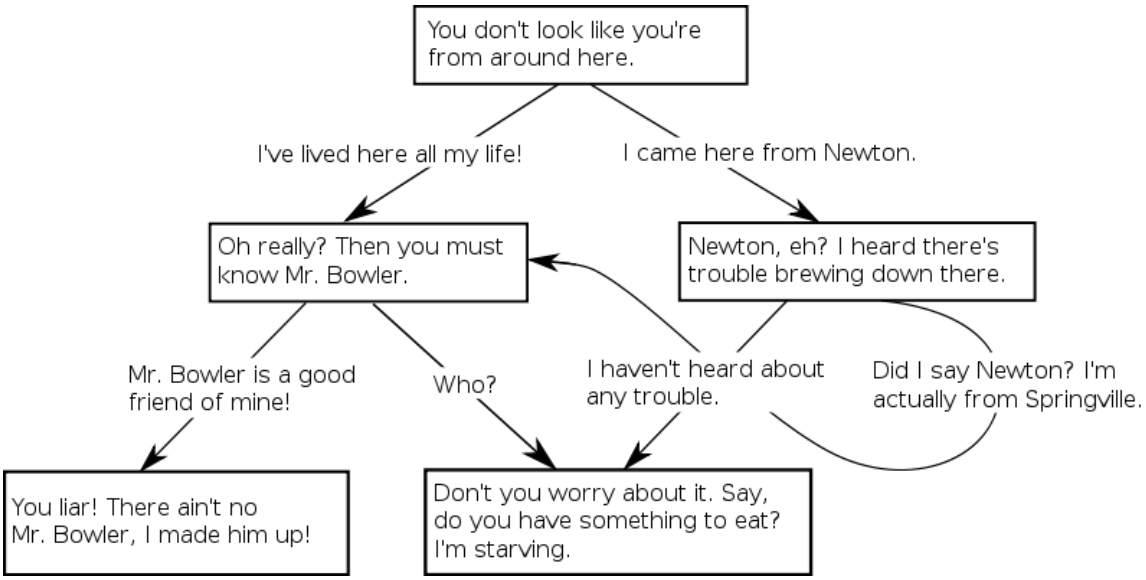

Figure 2.1 An example of a dialogue tree [16].

2.2.1 Advantages

The use of branching dialogue trees has the following major advantages:

- The author controls what the player is allowed to say
    - Since the player is forced to choose from a menu of possible utterances, there is no risk of the player saying something that the NPC is unable to respond to.
- Player has choice
    - Unlike in cutscenes where the player either passively watches or at most presses a button when prompted to do so, the player is actively able to choose what to say to an NPC. The choice of utterance has an obvious effect on the type of response the NPC will give.

2.2.2 Disadvantages

The use of branching dialogue trees has the following major disadvantages:

- Labour-intensive to write
    - The author needs to anticipate different utterances the player may want to say to an NPC at any given time in the dialogue. Since a dialogue tree grows exponentially, and since the player would likely need to interact with several NPCs in the game, the amount of writing an author would need to do is tremendous.
- Consistency can be difficult to maintain
    - If the author decides to edit a dialogue tree, then s/he needs to ensure that subtrees remain consistent with the edit. A small change early in the tree can cause a ripple effect that leads to numerous more changes being required later in the tree [17].

- Dialogue will eventually be exhausted
    - After a while, the bottom of the dialogue tree will be reached. When this occurs, the NPC will either repeat itself or simply ignore the player. Both circumstances break the illusion of intelligence.

2.3 Simple Natural Language Processing

Simple natural language processing is a technique that only a few video games employ. Nevertheless, it is worth mentioning because it offers an interesting alternative to the status quo that gives players more control over their actions. Using this technique, the player can input something to say to the NPC, and then the NPC responds in an appropriate manner. Although keyboard input is most common, it is not always necessary to use it; games can also use speech-to-text technology to convert microphone input into text. Some notable games that use simple NLP are *Façade*, *Bot Colony*, *Nintendogs*, *Photopia*, and *Galatea*.

2.3.1 Classifications

Video games that use simple natural language processing can be organized into two main classifications: simple commands and simple dialogue. The following subsections will explain these two types in more detail.

2.3.1.1 Simple Commands

In some games, the player can only interact with NPCs via direct commands, rather than engage in dialogue. Only commands that the system recognizes will cause the NPC to perform the desired action. Unrecognized commands will yield either no response or a nonsensical one because all responses are scripted [18]. A classic example of this type of game is *Nintendogs*, where the player could give a command like “High five!” and the virtual dog would put its paw up in response.

In other games, the player can indirectly converse with NPCs by inputting commands indicating the topic to discuss. Often, these types of games are text-based like *Photopia* and *Galatea*. As before, only recognized commands result in something useful

happening; otherwise, an error message will usually be displayed such as “I do not understand that sentence”. The following example from *Galatea* illustrates what may happen after the player inputs a recognized command:

> *Ask about spotlight*

“So does that get in your eyes?” you ask, gesturing at the spotlight.

“A bit. It's also too hot.” She shrugs. “But what can you do.”

2.3.1.2 Simple Dialogue

Some games offer the player a more sophisticated way of interacting with NPCs than merely giving commands. Rather, these games allow the player to engage in simple dialogue. Two examples of this type of game are *Façade*, which lets the player input either single words or sentences via keyboard, and *Bot Colony*, which lets the player input well-formed English sentences via either keyboard or speech recognition.

Façade is divided into various beats (or segments). For these different beats of the game, the author tried to anticipate what the player might want to say to an NPC. If the player inputs one of these predicted utterances, then the NPC follows the rule of how to act for that beat. If the player inputs something else, then the NPC acts in a way prescribed for the entire game. However, if no rule exists for how the NPC should respond, then the game moves on to the next beat and the NPC starts a new topic of conversation. It should be noted that the game is in real time, so a few seconds of silence is treated like player input [19].

Unlike *Façade*, *Bot Colony* uses automated dialogue. It is one of very few games that attempts such a feat. The player can ask questions, make assertions, and issue commands. Then the system tries to understand the meaning of what the player is saying with the intention of giving an appropriate response. Currently, the player can only interact with robot characters in such a manner because “a human character not understanding what he or she is told would immediately dispel the suspension of disbelief.” However, in order to

ensure that human characters could understand everything they are told, “the level of language understanding must be exceptionally high.” [20]

2.3.2 Advantages

The use of simple natural language processing has the following major advantages:

- Player has total freedom in what to say
    - Since players can input a statement rather than choosing from a menu, they will be more immersed in the game compared to using a dialogue tree because “when a player is faced with obvious choice points consisting of a small number of choices (e.g., being given a menu of three different possible things to say), it detracts from the sense of agency” [19].
- NPCs can appear to be intelligent
    - Assuming that the NPC’s response to what the player wrote is appropriate, the NPC will seem credible as a believable agent because it is able to respond sensibly to unscripted input.
- Game seems more realistic
    - Since these types of games often impose a time limit for the player’s response, they seem more lifelike due to the fact that the NPCs can notice unusual pauses in the dialogue and comment on them.

2.3.3 Disadvantages

The use of simple natural language processing has the following major disadvantages:

- NPC responses are usually not automated
    - Most of the time, NPC responses are prewritten by a human author; it is very rare for them to be automated. Thus, lots of time and effort must go

into writing them because “the only way to increase interactivity is to author extraordinary amounts of content by brute force” [19].

- NPC may ignore what player says
    - If no rule exists on how to respond to the player’s input, an NPC will either ignore the player entirely or suddenly change the topic of conversation. Either scenario breaks the illusion that the NPC is intelligent because it becomes obvious that it does not know how to respond.

2.4 Summary

This chapter discussed the state-of-the-art techniques used to simulate dialogue in video games. The first technique is the use of cutscenes, which are scripted sequences that the player must often passively observe. Cutscenes are generally either live-action, pre-rendered, or in-game cinematics. The second technique is the use of branching dialogue trees, which allow the player to choose from a menu of possible utterances. They tend to be “clumsy, difficult to write and unrealistic” but are considered to be “the best we’ve got” [21]. The third technique is the use of simple natural language processing, which consists of either simple commands or simple dialogue. These different techniques all have the risk that the player will eventually cease to view the NPCs as believable agents, which can break the immersion of the game.

Chapter 3

3 State-of-the-Art in Artificial Intelligence

As briefly mentioned in Chapter 1, the techniques for handling dialogue in video games are quite outdated. There has been a great deal of research done in Artificial Intelligence that could be used to make dialogue more dynamic, but for the various reasons discussed in Section 1.3, it is ignored. These advances in Artificial Intelligence are more sophisticated than the techniques mentioned in Chapter 2; thus, they do require more in-depth knowledge to employ. Nevertheless, they are highly robust and can greatly aid in making dialogue more automated. Consequently, they can assist in making non-player characters seem more autonomous. This section will cover some of the core techniques in Artificial Intelligence that should be utilized in video games: Information Extraction, Sentiment Analysis, and Question Answering.

3.1 Information Extraction

The purpose of Information Extraction is to automatically extract structured information from unstructured sources. The advantage of having structured information is that players can make richer queries than simply searching for individual keywords [22]. Consider the following example sentence: *"Paris is the stylish capital of France"*. An information extraction system should recognize from this sentence that there is a relationship between Paris and France, namely that one is the capital of the other; thus, it may generate a relational tuple such as *capital-of(Paris, France)* [23].

Extracting information from noisy, unstructured sources is not trivial. This area has drawn lots of attention from researchers for over two decades. During this time, various techniques for information extraction have been developed. While original systems were rule-based, the use of statistical methods such as Hidden Markov Models, conditional models based on maximum entropy, and Conditional Random Fields has become more favourable [22].

Information extraction can be divided into several subfields. This section will focus on the following three: Named Entity Recognition and Classification, Coreference Resolution, and Relationship Extraction.

3.1.1 Named Entity Recognition and Classification

Named Entity Recognition and Classification, often abbreviated as NERC, is a subfield of Information Extraction where a system automatically recognizes and classifies units of information such as the names of people, organizations, and locations, as well as numeric expressions about time, date, money, and percentages. Originally, Named Entity Recognition and Classification systems were entirely rule-based, meaning they used handwritten rules to identify named entities. However, most modern systems prefer to use machine learning techniques instead such as supervised, semi-supervised, and unsupervised learning.

3.1.1.1 Supervised Learning

Supervised learning is the most common technique for Named Entity Recognition and Classification systems. The way it works is that the system reads a corpus of text annotated by a human, memorizes a list of entities, and then creates a set of disambiguation rules using discriminative features. Several methods can be used to implement supervised machine learning such as Hidden Markov Models, decision trees, maximum entropy models, support vector machines, and Conditional Random Fields.

3.1.1.2 Semi-Supervised Learning

The use of semi-supervised (also known as “weakly supervised”) learning for Named Entity Recognition and Classification is rather recent. In semi-supervised learning, a technique called “bootstrapping” is used, which means that the system is provided only a small amount of labelled data and a fairly large amount of unlabelled data. The purpose of the small amount of labelled data is to allow the system to identify contextual clues common to all the examples. Afterwards, the system looks for similar contexts to try to find other named entities. By recursively applying this process, the system is able to find a large number of named entities appearing in a large number of contexts.

3.1.1.3 Unsupervised Learning

Clustering is the most common unsupervised learning technique for Named Entity Recognition and Classification. Using this method, the system creates clusters of terms based on how similar their contexts are, and then tries to identify named entities from these clusters. In general, the use of lexical resources (e.g. WordNet), lexical patterns, and statistics computed on an unlabelled corpus are required for unsupervised learning to be able to discover named entities.

3.1.1.4 Features for Machine Learning

The choice of which machine learning technique to use is not the only important factor in creating a Named Entity Recognition and Classification system. Finding appropriate features is equally as important. Features are distinctive attributes or characteristics of a word. They are required for the system to be able to create classification rules. There are several kinds of features that can be used in a machine-learning system such as word-level features (e.g., case, punctuation, morphology), list lookup features (e.g., list of entities), and document and corpus features (e.g., corpus frequency) [24].

3.1.1.5 Potential Use in Video Games

The ability to recognize and classify named entities would likely be an integral part of an information extraction system present in a video game. In several games, especially role-playing ones, important non-player characters and locations usually have unique names. It is quite likely that when players interact with an NPC, they might mention either another NPC or a location in the game world. Therefore, the system should recognize and correctly classify these named entities to ensure that the NPC can give an appropriate response to a player's query.

For example, consider the following two sentences: "*Lily is a kind of flower*," and "*Lily is my best friend*". In the first sentence, "*Lily*" refers to the plant, while in the second sentence, it refers to a person. Thus, the NPC's response should match the meaning of "*Lily*" that the player was referring to.

3.1.2 Coreference Resolution

A coreference occurs when different words or phrases refer to the same entity. The purpose of coreference resolution is to identify these portions of text so that the system knows the player is referring to the same object. A common example of coreference resolution would be anaphora resolution.

Coreference resolution is important because the meaning of a sentence is difficult to determine if the system is not aware that the same entity is being referred to by different expressions. Consider the following sentence: *"Joshua really likes cornflakes, but he gets them all over his face."* In this particular sentence, *Joshua*, *he*, and *his* all refer to the same object. Moreover, *cornflakes* and *them* also refer to the same object. Thus, the system needs to be able to recognize that the above sentence is equivalent to, "*Joshua really likes cornflakes, but Joshua gets cornflakes all over Joshua's face.*"

3.1.2.1 Difficulties

Originally, attempts at coreference resolution used rule-based approaches. Recently, however, machine learning is the more standard. Nevertheless, machine learning is still not a perfect solution (or even close to perfect). As mentioned in the previous section on NERC, the choice of features is very important. Unfortunately, commonly used features such as the distance between coreferences, string matching, and linguistic form are inadequate to recognize many coreferential relationships. Instead, commonsense and encyclopedic knowledge are generally required to be able to accurately resolve coreferences. Unfortunately, it is not a trivial problem to provide the system with all the commonsense and encyclopedic knowledge that humans have. Determining how to use lexical and commonsense knowledge to improve coreference resolution on unrestricted text is still an active area of research. The following two examples illustrate why coreference resolution is such a difficult problem.

3.1.2.1.1 Example 1

Consider the following example: *"A new report reveals more problems at the **Internal Revenue Service**. A broad review of **the agency** found it used improper tactics in*

*evaluating **IRS** employees at many **IRS** offices across the country."* The words in bold all refer to the same government agency, the Internal Revenue Service. Unless the system has a way to know that the IRS is a government agency, it could not be expected to recognize that *the agency* is a coreference.

3.1.2.1.2 Example 2

Consider another example: *"**Israel** will ask the United States to delay a military strike against Iraq until **the Jewish state** is fully prepared for a possible Iraqi attack with non-conventional weapons, the defense minister said in remarks published Friday."* In this sentence, the words in bold both refer to the state of Israel. As in the previous example, the system needs some way of knowing that Israel is Jewish state for it to correctly identify that a coreference is present [25].

3.1.2.2 Potential Use in Video Games

When people write complex sentences, they will generally use coreferences to avoid repeating themselves unnecessarily. They use pronouns, synonyms, hyponyms, hypernyms, and other expression types to convey their ideas. Consequently, rather than limiting players by requiring the use of only one identifier per entity, they should have the freedom to input more natural-sounding sentences. Thus, coreference resolution would be beneficial in video games because players would still be able to write in a way that they are accustomed to, and the system could better understand the meaning they are trying to convey. Thus, although coreference resolution technology is still imperfect, its use could still allow for a more immersive gameplay experience.

3.1.3 Relationship Extraction

Often, different entities will be related to each other in some way. The purpose of relationship extraction is to extract these semantic relations among entities. A relation can be defined as a tuple such that the entities within a text have a predefined relationship. Relationship extraction systems usually focus on seeking out binary relations. For example, in the sentence, "*CMU is located in Pittsburgh*," the system might generate the tuple *located-in(CMU, Pittsburgh)*, and in the sentence, "*Manuel Blum is the father of*

Avrim Blum," the system might generate the tuple *father-of(Manuel Blum, Avrim Blum)*. Relations do not have to be binary however; they can be a higher order as well. Consider the following example: *"At codons 12, the occurrence of point mutations from G to T were observed"*. This example has a quaternary relation and can be represented as the tuple *point-mutation(codon, 12, G, T)*. Like other areas of information extraction, the technique of choice for relationship extraction is machine learning, particularly supervised and semi-supervised learning.

3.1.3.1 Supervised Learning

Supervised learning is a common technique for relationship extraction. All that is required is to provide enough labelled examples of entities that are related and that are not related. Given a sentence *S*, define *T(S)* as the features extracted from *S*, and e_i and e_j as entities present in *S*. A mapping function f_R can be defined as follows:

$$f_R(T(S), e_i, e_j) = \begin{cases} +1 & \text{If } e_i \text{ and } e_j \text{ are related according to relation } R \\ -1 & \text{Otherwise} \end{cases}$$

This mapping function can be used to generate a labelled set of training data that can be used to train a classifier like Neural Nets or Support Vector Machines. If enough training data and appropriate features are provided then a classifier could potentially perform quite well.

In practice, supervised learning has some drawbacks. Firstly, it is difficult for the classifier to find new entity-relation types not included in the training data, so the data must have examples of all the entity-relation types that the classifier is expected to find. Secondly, it is not trivial to extend a classifier to higher-order relations, so a classifier for binary relations will have a difficult time finding non-binary relations. Thirdly, it is computationally expensive to work with large input, so it does not scale well as input sizes increase. Finally, input data needs to be preprocessed (e.g. form a parse tree), but preprocessing can already be quite error-prone, so errors may propagate.

3.1.3.2 Semi-Supervised Learning

The other common machine-learning method for relationship extraction is semi-supervised learning. As mentioned before, semi-supervised learning takes advantage of a technique called "bootstrapping". The major advantage semi-supervised learning has over supervised learning is that there is generally an abundance of unlabelled data available, but only a limited amount of labelled data. The reason for the low amount of labelled data is because it is expensive to create in large amounts.

Although semi-supervised learning has some advantages over supervised learning, it still has a significant disadvantage: it is difficult to use. Algorithms like Snowball, KnowItAll, and TextRunner require a large number of input parameters, but they do not explicitly specify how to select optimal parameters. Therefore, it can be quite difficult to tune the parameters to get optimal results. On the other hand, an algorithm like Dual Iterative Pattern Relation Expansion relies on hard pattern matching; consequently, it considers two patterns to be different even if they only differ by punctuation. Thus, more research is necessary to ensure that semi-supervised learning for relationship extraction is easier to use and more powerful [26].

3.1.3.3 Potential Use in Video Games

Relationship extraction is still far from perfect in unrestricted domains. However, it should be possible to use within a restricted domain like a video game. Since video games have a narrow scope, some of the downfalls of supervised learning that were previously mentioned should not cause too many issues. In a standard video game, there would likely only be a limited number of entity relationship types (e.g., father, friend, hairdresser, etc.); consequently, it would not be difficult to make exhaustive training data that includes examples of all the relationship types. Furthermore, a player is likely only going to input a few sentences at a time, so large input sizes are unlikely to occur during actual gameplay. Therefore, using supervised learning for relationship extraction may be practical in a game.

The ability to correctly extract relations would allow players to query the system about relationships in the game world (e.g., "*Who is Steven's father?*"), as well as give the

system information about relationships between entities (e.g., *"Henry is the King of England."*). As a result, the incorporation of relationship extraction would permit more complex dialogue between players and non-player characters.

3.2 Sentiment Analysis

Sentiment analysis is defined as "the computational study of people's opinions, appraisals, attitudes, and emotions toward entities, individuals, issues, events, topics and their attributes". It is also commonly known as opinion mining.

Let e_i represent an entity's name, a_{ij} represent an aspect of the entity e_i, oo_{ijkl} represent the orientation of the opinion about the aspect a_{ij} (the orientation can be positive, negative or neutral, and it may have different intensity levels), h_k represent the person who holds the opinion, and t_l represent the time when the person h_k expressed the opinion. An opinion can then be expressed by the following quintuple $(e_i, a_{ij}, oo_{ijkl}, h_k, t_l)$. The term *GENERAL* is used as the aspect in the special case where the opinion is about the entire entity rather than a specific part.

Consider the following example sentence written by a player named bigXyz on November 4, 2010: *"The voice of my Moto phone was unclear, but the camera was good."* A sentiment analysis system should recognize that bigXyz dislikes the voice quality on the Motorola but likes the camera. Thus, it would convert the sentence into the following two tuples: *(Motorola, voice_quality, negative, bigXyz, Nov-4-2010)* and *(Motorola, camera, positive, bigXyz, Nov-4-2010)*. The benefit of using this particular definition of "opinion" is that it facilitates converting unstructured text into structured data [27].

3.2.1 Potential Use in Video Games

When players interact with NPCs, they may potentially mention their opinion on some aspect of the game. For example, they might write something like, "*I do not like my sword. It is not sharp enough.*" With sentiment analysis, the system could recognize that the player has a negative sentiment towards the sword's sharpness. In response, it could perhaps recommend either a place to get it sharpened or a shop to purchase a new sword.

Furthermore, players may be either friendly or hostile towards an NPC. Sentiment analysis would allow the system to recognize the player's positive or negative sentiment about the NPC and have it respond in a sensible manner. For example, if a player insults the NPC, then the NPC could respond aggressively, while if the player compliments it, it could respond kindly. By adjusting the NPC's reaction based on how the player is perceived to be feeling, the dialogue will seem more natural and the NPC will seem more autonomous.

3.3 Question Answering

A question-answering system is a type of information-retrieval system that answers a natural-language query. Its purpose is to give concise answers to players' questions. Question-answering systems have recently become quite advanced, and research in this area is constantly growing. In recent years, there has been increased interest in researching interactive question-answering systems, which allow players to ask follow-up and clarification questions. This ability is useful because question-answering systems are only rarely used for just one question. Instead, they tend to be used for several questions related to a similar topic of interest (e.g. William Shakespeare). As a result, players should be able to ask related questions in a manner that is natural for them, rather than being forced to word queries in a contextually-independent way. Nonetheless, giving players this sort of flexibility is a non-trivial task and can prove to be difficult.

3.3.1 Difficulties

Two common issues that arise in interactive question-answering systems are ellipsis and anaphoric references. These problems and their potential solutions are explored further in the following subsections.

3.3.1.1 Ellipsis

Ellipsis occurs when part of a sentence is omitted in such a way that there is no verb phrase. They are generally used for follow-up questions. For example, the player may ask the system the following question, "*When was Shakespeare born?*" Afterwards, the player may ask, "*Where?*" The system must recognize that the player is actually asking,

"*Where was Shakespeare born?*" because the latter part of the sentence is implied. To determine the implicit verb phrase, it can use previous questions that were asked to fill in the missing words.

3.3.1.2 Anaphora

An anaphora is a linguistic form whose meaning depends on the context being known. Usually, anaphora present themselves as third-person pronouns. For example, the player may ask, "*Whom did he marry?*" Without context, it is impossible to determine who "*he*" refers to, and thus, it is impossible to answer the query. To resolve anaphoric references, the system can replace them with entities mentioned in previous questions [28]. Continuing with the example from Section 3.3.1.1, the system should replace "*he*" with "*Shakespeare*".

3.3.2 Potential Use in Video Games

Due to its ability to answer a natural-language query, a question-answering system would be beneficial in games where players need to ask questions to NPCs in order to proceed. In most modern role-playing games, players must choose a question from a dialogue tree. As mentioned earlier, the drawbacks of dialogue trees are that NPCs will run out of new things to say and that the author cannot anticipate every question the player may want to ask. A question-answering system would allow players to ask questions to an NPC as though they were conversing with a human.

3.4 Summary

This chapter discussed some state-of-the-art techniques in Artificial Intelligence that could be used to improve dialogue in video games. The first technique is information extraction, which automatically extracts structured information from unstructured sources. It would allow a system to better understand what a player is trying to say. Three areas of information extraction that would be highly relevant in the context of a video game are named entity recognition and classification (NERC), coreference resolution, and relationship extraction. The second technique is sentiment analysis, which extracts people's opinions about various entities. It would allow a system to formulate responses

that are relevant to how a player feels about something in the game. The last technique mentioned is question answering, which retrieves information based on a natural-language query. It would allow the system to answer questions the player may have about various aspects of the game. Combined with information extraction and sentiment analysis, question answering could potentially offer very sophisticated replies. While none of these techniques have been perfected yet, their implementation in a dialogue engine could greatly add to the believability of an NPC compared to traditional methods.

Chapter 4

4 Conversation Model

This chapter proposes a high-level model for conversation that will serve as the basis for the rest of this thesis. The goal of this model is to make NPCs appear as human as possible in order to cause the player to suspend disbelief. Although actual conversations between humans can vary significantly due to individual as well as social factors, this model attempts to dissect features that are common in many kinds of social situations. Furthermore, this model assumes that the player and the NPC are not close acquaintances.

Since there is no agreed upon model of conversation in the literature, this thesis offers one that could be used as the foundation for a framework. It is highly likely that not all conversations could fit this model; however, although it may be possible that a better model exists that could better simulate a wider range of conversations, this model was created because it is complex enough to allow for a variety of different conversations while simple enough that it could be implemented on a computer. The conversation architecture discussed in Chapter 5 requires that this particular model be used, but it could be modified to accommodate other models later on without great difficulty.

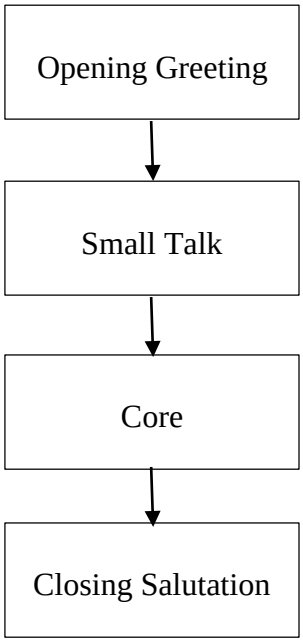

Figure 4.1 High-Level Model of Conversation

Figure 4.1 illustrates a high-level version of this conversation model. Here, a conversation begins with an opening greeting. Afterwards, the participants engage in small talk, which is meaningless chatter about neutral subjects. Once a certain level of comfort is reached, they progress to discuss more substantial topics. Finally, once the conversation is finished, they conclude with a closing salutation. Although not explicitly shown, this model uses the notion of turn-taking: the NPC and the player take turns during the conversation, so it is not possible for one to interrupt the other (i.e. the player inputs a complete thought, and then the NPC outputs a complete thought).

The purpose of Figure 4.1 is to provide a general overview of the stages in a conversation without focusing too much on the details. In particular, "Core" is essentially a shorthand that means any part of the conversation that is not a greeting or small talk, and this diagram treats it like a black box. The following sections will elaborate on each of these stages further.

4.1 Opening Greeting

Greetings are a frequent component of social interaction. Their appropriate use is vital to establish and maintain interpersonal relationships between people. Under Austin's Speech Act Theory, greetings are considered to be expressive illocutionary acts, meaning that they are not supposed to be taken literally. For example, when people say, "Good morning," it does not necessarily mean that they actually care if someone's morning goes well. Instead, these types of expressions are meant to show politeness [29].

While a large number of potential greetings exist, not all are appropriate in every situation. In deciding which greeting to use, one must consider the following factors:

- Social distance between speaker and listener
- Relative power of the listener over the speaker
- The absolute ranking of the imposition for that particular culture.

There are other factors that can affect the choice of greeting as well, such as emotional state or the personality of the individual. However, these factors are beyond the scope of this thesis and are left to future work.

4.1.1 Social Distance

Social distance is a symmetrical relationship. Assuming all else equal, the level of formality chosen by the speaker and the listener should be somewhat similar. Figure 4.2 illustrates an example of a social exchange between two close friends, while Figure 4.3 shows an interaction between two acquaintances in a formal setting. Notice how the level of formality is much higher in Figure 4.3 compared to Figure 4.2.

Friend 1: Hey there, Jim.
Friend 2: Howdy, Bill. Good to see you again, buddy.

Figure 4.2 Example of exchange where participants have short social distance.

Acquaintance 1: Good evening, Mrs. Smith.
Acquaintance 2: Good evening, Mr. Jones.

Figure 4.3 Example of exchange where participants have long social distance.

4.1.2 Relative Power

The relative power of the listener over the speaker is an asymmetrical relationship. Consequently, the participant with higher power can afford to be less formal than the participant with lower power. Figure 4.4 is an example of an exchange between a student and her teacher. In this example, the student is being formal by addressing her teacher by her title and last name, while the teacher is being less formal by addressing her student by only her first name.

Student: Good morning, Mrs. White.
Teacher: Good morning, Susie.

Figure 4.4 Example of exchange where one participant has power over the other.

4.1.3 Absolute Ranking of Imposition

Certain greetings can be considered highly imposing. As the imposition increases, the level of required politeness increases as well. A greeting is deemed to be imposing if it demands something of the listener that may interfere with his or her wants, self-determination, or approval (e.g., asking for goods, services, or time) [29, 30]. Figure 4.5 illustrates an example of a panhandler asking for money, Figure 4.6 shows an elderly person asking for help to carry groceries upstairs, and Figure 4.7 demonstrates a volunteer asking someone to hear about a fundraiser. These requests are highly imposing, particularly since they are all demanding something from a stranger. Thus, they counteract the imposition by acting politely.

Panhandler: Excuse me, ma'am, can you spare some change?

Figure 4.5 Example of person asking for money.

Elderly Person: Excuse me, could you please help me carry these upstairs?

Figure 4.6 Example of person asking for assistance.

Volunteer: Hello. Could you please spare a few minutes to hear about our fundraiser?

Figure 4.7 Example of person asking for someone's time.

4.1.4 Justification for Inclusion in Video Games

Different NPCs can have different social characteristics. These characteristics should influence how they behave. For example, some NPCs may portray close friends of the player, while others will initially be strangers. Furthermore, some NPCs may portray aristocrats, while others may portray paupers. Finally, some NPCs may have requests/demands for the player, while others do not.

Given that NPCs are portraying people, their choice of greeting needs to be appropriate for what a human would say in the same situation. As mentioned before, greetings are important for establishing and maintaining interpersonal relationships. Since the goal is for an NPC to act like a human, it needs to follow social conventions for conversation. Otherwise, it will cease to be a believable agent.

4.2 Small Talk

When people have never met or do not know each other well, they will generally engage in light conversation about neutral topics. This light conversation is what is known as small talk. The main purpose of small talk is to establish rapport and trust, which in turn reduces social distance. It provides participants the opportunity to set up an interactional style, as well as establish their reputations. Often, they will chat about topics like the weather, their physical environment, personal experiences, and preferences [31]. Figure 4.8 illustrates an example of small talk between two individuals who have never met. Initially, they discuss the weather, but then they proceed to talk about barbeques once they have established some rapport.

Person 1: Nice weather we're having.
Person 2: Yes, it's been gorgeous out lately. I heard there's supposed to be a storm coming soon though.
Person 1: Oh, really? I'll have to put my patio furniture away then. We just had our first barbeque of the season last night.
Person 2: Oh, that sounds fun! My family loves barbeques. We had them all the time last summer.

Figure 4.8 Example of small talk

In social interactions, there is the notion of *face*. Speakers wish to maintain positive face by getting the approval of their listeners, and they wish to maintain negative face by being unhindered in their autonomy [30]. According to Bickmore and Cassell, small talk mitigates any threat to face in two ways:

1. It provides an interactional style where it is easy to continue a conversation and establish comradery, which maintains positive face

2. It establishes that other conversation participants are non-hostile, which maintains negative face.

4.2.1 Justification for Inclusion in Video Games

There have already been documented cases of players establishing on-going relationships with chat-bots that were able to engage in small talk. Even when players were made aware that the systems could not actually understand what they were typing, they still maintained these relationships. It appears that small talk has a profound ability to alter players' perceptions of conversational agents, which permits them to suspend disbelief [31].

When a player interacts with various NPCs for the first time, they will be unfamiliar with each other, unless the storyline has explicitly established that an NPC has known the player since before the game actually started. Since strangers rarely have profound conversations due to lack of rapport, it would be bizarre if all NPCs acted comfortably enough around the player to discuss highly personal or controversial issues. Thus, it is sensible to include the ability to engage in small talk so that the player and NPC can gain familiarity and build trust.

4.3 Core

The core of the conversation could be considered as the most important part. At this point, social formalities have already occurred, so the player and NPC can discuss more significant issues. The specifics of what they can converse about is very much game-dependent, but it could include the player seeking information about something in the game world, the NPC introducing a quest, or even the NPC mentioning something that may prove useful later on at another stage of the game. Once again, there is no standard, agreed-upon model for conversation, so this thesis chose to include certain features that would be important to a number of games and scenarios. While not all possible situations that could occur during the core are necessarily covered here, this section covers many situations that are likely to occur and could be implemented. However, other features could be added later on to cover other possible situations. The following sections will explore this stage in more detail.

4.3.1 Player Asks Questions

A likely reason that the player would wish to speak with an NPC is to ask it for information. This information may assist with completing a specific quest (e.g., where to find the key to a castle), or it may be more general information that will be useful for the entire game (e.g., how to restore health). There are different kinds of questions the player may ask, so the NPC's answer must make sense to maintain the illusion of intelligence. The answers do not, however, always need to be correct. In fact, it could be interesting if different NPCs offered different answers to the same question, which is a phenomenon that can occur in real life. Furthermore, the NPC is allowed to admit to not knowing the answer, which can also occur in real life.

4.3.1.1 "W6" Questions

In English, there are question types that begin with the words *who*, *what*, *where*, *when*, *why*, and *how*. It is important that when an NPC retrieves information, it answers the correct question. Figure 4.9 is an example of an NPC giving an appropriate response to a player's question. The player wishes to know when someone named Jane will come home, and the NPC answers that she will be home at 9:00 PM. It could very well be true that Jane will be home at 9:00 PM, or it could be incorrect; nevertheless, it is irrelevant because the answer makes sense and is believable. On the other hand, Figure 4.10 is an example of an NPC giving a potentially inappropriate response to a player's question. Once again, the player wishes to know when Jane will be home. However, the NPC's response does not directly answer the question. Rather, it answers the question of where Jane is located. Unless the fact that Jane is currently located at Billy's house has some hidden meaning (e.g., she will be home quite late because she is with him, but it is not clear when), the response is nonsensical. This particular example highlights the problem of semantics versus pragmatics, so it would be up to a human author to determine whether the answer, "Jane is at Billy's house," is appropriate in this context or not.

Player: When is Jane coming home?
NPC: Jane will be home at 9:00 PM.

Figure 4.9 Example of NPC Giving a Sensible Answer to Player Query

Player: When is Jane coming home?
NPC: Jane is at Billy's house.

Figure 4.10 Example of NPC Giving a Nonsensical Answer to Player Query

4.3.1.2 Yes/No Questions

Yes/no questions are another question type that the player may ask. These often begin with conjugated forms of "to be" or "to do". While not strictly necessary for the NPC to answer a question with the words "Yes" or "No", often people do include one of these words in their answer. Figure 4.11 illustrates an NPC answering a yes/no question with a descriptive answer. Although it does provide the information the player was likely seeking (i.e. when Jane will actually come home), it comes across as somewhat artificial because it does not acknowledge that the player was incorrect. Figure 4.12 illustrates an NPC replying with only "No". It answers the question correctly, but it also raises the question, "When is Jane coming home?" Figure 4.13 illustrates a hybrid of Figure 4.11 and 4.12. Here, the NPC acknowledges that the player was incorrect and also provides clarification as to why. Consequently, the answer is both natural-sounding and complete.

Player: Is Jane coming home at 5:00 PM?
NPC: Jane is coming home at 9:00 PM.

Figure 4.11 Example of NPC answering a Yes/No Question with a Descriptive Sentence

Player: Is Jane coming home at 5:00 PM?
NPC: No.

Figure 4.12 Example of NPC answering a Yes/No Question with "No"

Player: Is Jane coming home at 5:00 PM?
NPC: No, Jane is coming home at 9:00 PM.

Figure 4.13 Example of NPC answering a Yes/No Question with "No" and a clarification

4.3.1.2.1 "Did You Know…" / "Did You Hear…" Questions

Occasionally, the player may try testing an NPC by asking if it knows or has heard something. Although worded as a question, the player is essentially making an assertion, which is a different kind of speech act. This question is still a yes/no type, but the NPC should clarify what it means if it says "No". Figure 4.14 illustrates an example of an NPC replying "No" to the player's question. The reply is ambiguous because it could mean, "No, I was not aware of that," or "No, that statement is false." Figures 4.15 and 4.16 are examples of unambiguous replies that the NPC could offer.

Player: Did you know that Jane coming home at 5:00 PM?
NPC: No.

Figure 4.14 Example of NPC replying with just "No" to a "Did You Know..." Question

Player: Did you know that Jane coming home at 5:00 PM?
NPC: No, I didn't know that.

Figure 4.15 Example of NPC replying that it did not know an asserted fact

Player: Did you know that Jane coming home at 5:00 PM?
NPC: That's not true. Jane is coming home at 9:00 PM.

Figure 4.16 Example of NPC disagreeing with an asserted fact

4.3.2 NPC Initiates Topics

To make the NPC appear more autonomous, as well as add an interesting dynamic to the dialogue, the system should not rely solely on the player to initiate topics of discussion. Instead, it should also bring up topics at various points to carry on the conversation. The following two sections propose ways it could achieve this task.

4.3.2.1 NPC Asks the Player Questions

Either to demonstrate curiosity or to further the plot, having the NPC be able to ask the player questions can make it seem more genuine. The types of questions it should ask depend heavily on factors such as the game's storyline and the NPC's personality, so it is difficult to suggest particular questions that all NPCs should ask. Nevertheless, it is important that no matter what it asks, it should remember the player's answers in case it needs to discuss them later on. Figure 4.17 is a simple example illustrating how the player may test the NPC to see if it remembers an earlier part of their dialogue.

NPC: Halt. Why are you here?
Player: I wish to see the King.
NPC: Very well.
…
Player: Do you know why I'm here?
NPC: You wish to see the King.

Figure 4.17 An example of an NPC remembering a player's response

Having the NPC ask questions is also a natural way to introduce quests if there are any. For example, the NPC could say something like, "*I have something that needs to be done, but I can't do it on my own. Will you help me?*" While the model of the core does not explicitly have rules on what to do if the player were to answer "*Yes*," an implementation of the model could check that the player gave a positive response and then implicitly query the system with a question such as, "*What do you need help with?*" By automatically using a question such as this one as the player's input, the system would be able to find a fact in the knowledge base similar to this one, "*I need help finding three rare rubies. Please bring them back to me.*"

4.3.2.2 NPC Recalls Episodic Memories

There are two types of memories that people can have: semantic (i.e., factual) and episodic. Episodic memory "pertains to knowledge of one's experienced life events and is used for the formation of autobiographical memories" [32]. So far, this chapter has

focused mainly on the importance of NPCs recalling *facts*. This section will serve as a complement and highlight the usefulness of recalling *events*.

Since NPCs are portraying people, they should be able to discuss their life stories. This kind of conversational storytelling is "often used to relate interesting events or humor and thus contribute to the rapport-building function". Ideally, the NPC should be recalling events related to the overall dialogue. However, even if they happen to be unrelated, the player may be able to permit them if they are entertaining or informative enough [31]. Thus, their inclusion can aid in making the NPC seem more believable while also strengthening its relationship with the player. Figure 4.18 shows how an NPC might mention an episodic memory during a conversation.

Player: Do you know where I could go fishing?
NPC: A good fishing spot is at Lake Orin. I remember when I used to take my son out fishing. It feels like just yesterday.
Player: Does your son still like fishing?
NPC: No. My son hates fishing now. He would rather drink mead all day.

Figure 4.18 Example of NPC recalling episodic memory

4.3.3 Player Makes Statements

The player might make statements at certain points in the conversation with the purpose of either changing the topic, volunteering information, or giving an opinion. Statements can be either objective or subjective, and these different types should be treated in different ways.

4.3.3.1 Objective

Objective statements are supposedly factual, so the NPC should treat them as the player volunteering information. Consequently, it should remember what the player said for future reference. It may be that the player provided incorrect information, whether deliberately or accidentally, so if the NPC is able to identify the contradiction, it should mention it unless there is a game-related reason why it would keep its knowledge a

secret. Objective statements should be treated in a similar way to the "Did you know…" questions mentioned in Section 4.3.1.2.1.

4.3.3.2 Subjective

Subjective statements are the player's opinions on different issues. These opinions may be either positive or negative. The NPC should be able to recognize if the player is complimenting or insulting it and respond appropriately, as illustrated in Figures 4.19 and 4.20. If the player is giving an opinion about something else, then the NPC should mention its own opinion on the subject if it has one (and if there is no reason not to), as can be seen in Figures 4.21 and 4.22.

Player: You're ugly.
NPC: How rude!

Figure 4.19 Example of NPC responding to an insult

Player: You're really smart!
NPC: Why, how kind of you to say that. Thank you!

Figure 4.20 Example of NPC responding to a compliment

Player: I love chicken pot pie.
NPC: I love it too. It's delicious.

Figure 4.21 Example of NPC agreeing with the player's sentiment

Player: The Count is cruel.
NPC: I disagree. He is a kind man.

Figure 4.22 Example of NPC disagreeing with the player's sentiment

4.4 Closing Salutation

The closing salutation occurs at the conclusion of a conversation. It follows the same rules as the opening greeting, so it is important that the level of politeness matches and that the greeting makes sense for the situation. For example, an NPC should not say, "*Goodnight*," during the day.

4.5 Summary

This chapter introduced the conversation model that serves as the basis for the rest of this thesis. In this model, a conversation is divided into four main stages: opening greeting, small talk, core, and closing salutation. Greetings are a social formality, and different levels of politeness are required depending on the situation. Small talk is light conversation whose purpose is to allow speakers to build rapport and trust. The core of the conversation represents the most important part of it. It is in the core that the player and NPC can ask each other meaningful questions, share information and opinions, and recall events. The closing salutation ends the conversation and must follow the same rules as the opening greeting. By having a realistic conversation model, an NPC can seem more believable and autonomous.

Chapter 5

5 Conversation Architecture

This chapter proposes an object-oriented architecture for the conversation model from Chapter 4. It is object-oriented in the sense that it is modular, extensible, and designed in such a way that the modules could be instantiated in an object-oriented programming language. This architecture is designed with ease-of-use in mind, so it is very "plug-and-play". Most of the work required in using it is simply to create an initial knowledge base of facts and some episodic memories for the NPCs. The rest is handled in the background by the conversation module.

5.1 Conversation Handler

The only module in the conversation architecture that the video game communicates with directly is the conversation handler. The conversation handler is a black box responsible for overseeing the entire conversation. Its purpose is to ensure seamless switching between the various stages of conversation mentioned in Chapter 4: opening greeting, small talk, core, and closing salutation.

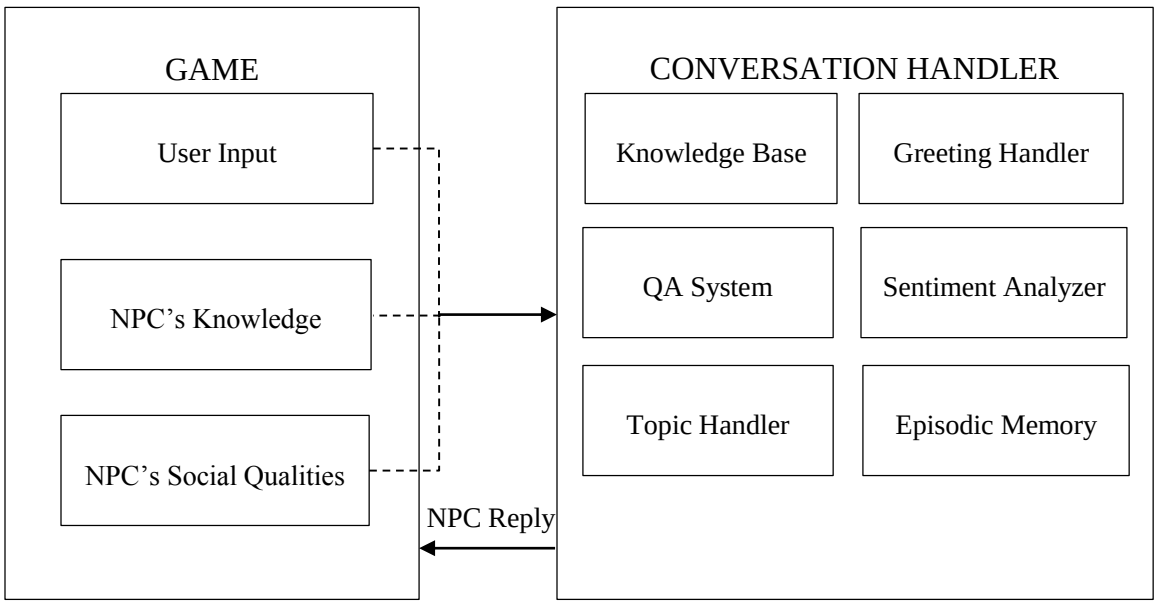

Figure 5.1 The Conversation Handler

Figure 5.1 illustrates how the conversation handler works. Here, it is depicted as separate from the game to highlight the fact that it could be used in a variety of different games. It accepts player input, information about the NPC's knowledge (e.g., factual and episodic memories), and information about the NPC's social qualities (e.g., its relative power over the player) from the game. Then, this information gets delegated to the following modules in order to generate an NPC response: knowledge base, greeting handler, question-answering system, sentiment analyzer, topic handler, and episodic memory module. Afterwards, the conversation handler outputs the NPC's reply back to the game. The details concerning how it accomplishes this task are hidden from the rest of the game.

In the event that a game wants to trigger a conversation without requiring the player to initiate contact, it could easily send a greeting to the NPC on the player's behalf without the player's knowledge. Then it would appear as though the NPC is initiating contact. In this particular case, the game would be responsible for ensuring that the NPC is approaching the player at an appropriate time so as not to create a strange opening.

It is important to note that the knowledge base and the episodic memory module illustrated within the conversation handler are derived from the NPC's knowledge that is provided by the game. The knowledge base stores the NPC's knowledge about facts while the episodic memory module stores the NPC's knowledge about events.

5.2 Knowledge Base

The knowledge base is responsible for storing the knowledge an NPC has about its world. These facts could be permanent (e.g., "*Ottawa is the capital of Canada*") or temporary (e.g., "*It is raining*"). The knowledge base is dynamic, so as the game progresses, the NPC is able to learn new facts. These new facts can be provided directly by the game or by the player.

5.3 Greeting Handler

The greeting handler is responsible for selecting the appropriate opening greeting/closing salutation for various situations. As mentioned in Chapter 4, there are three main factors

to consider when choosing a greeting: social distance, relative power of the listener, and the absolute ranking of the imposition. The greeting handler must calculate the required level of politeness based on these factors and then select the greeting that most closely matches the necessary politeness level. The greeting handler must also ensure that the greeting makes sense for the situation. For example, "*Good morning*" should not be selected if it is the evening.

5.4 Question Answering System

The question answering system (QA system) is responsible for answering player queries. Its goal is to offer responses in the same way a human would. Thus its replies must be as natural-sounding as possible. The question answering system can be divided into two main parts: a question parser and an information retrieval system. These parts can be further divided into more sub-modules, as seen in Figure 5.2. Sections 5.4.1 and 5.4.2 elaborate on these components further.

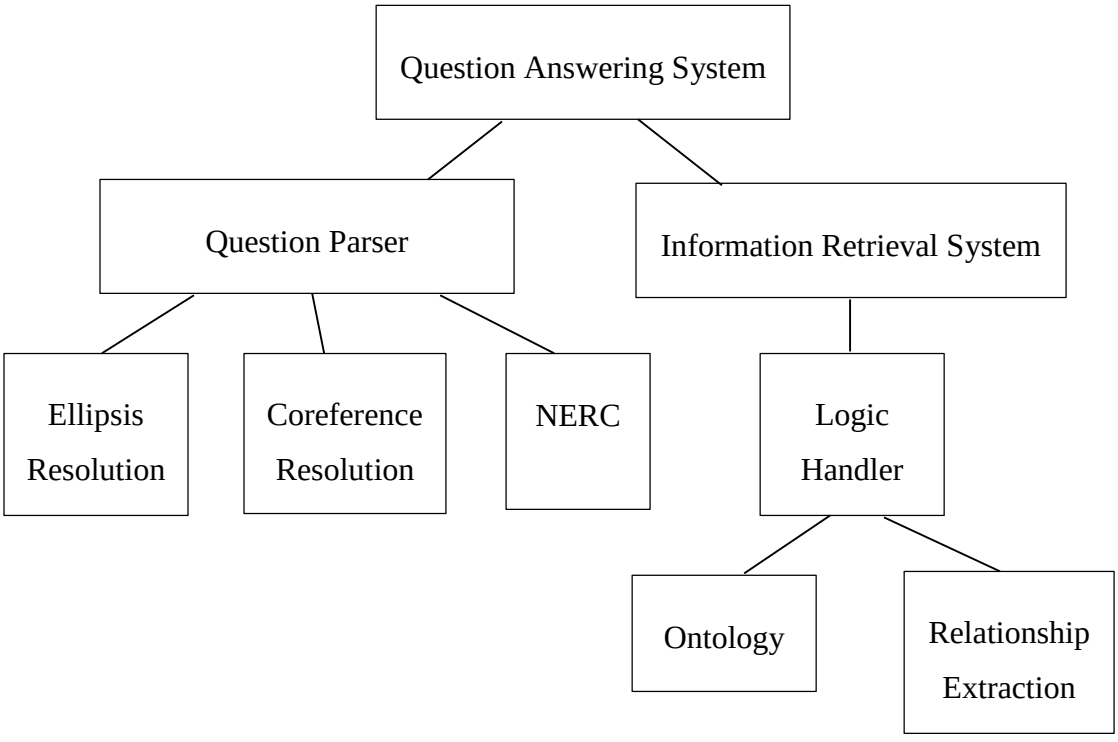

Figure 5.2 Question Answering System represented as a tree

5.4.1 Question Parser

The question parser is responsible for determining both what the player is asking and what sort of reply is expected. In Chapter 4, two main types of questions were discussed: "W6" questions and yes/no questions, so the system must classify which of these types the player is asking. Also as previously mentioned, the answer to a "W6" question must make sense for the type of question asked, so it is important to be able to determine the appropriate type of reply.

Sometimes, what the player is asking or what the appropriate reply should be is not immediately obvious. In these situations, extra processing is required to determine what the player's query was and how best to answer it. The following sections describe sub-modules the question parser will need for these special scenarios.

5.4.1.1 Ellipsis Resolution

As mentioned in Chapter 3, ellipsis occurs when the player inputs a sentence lacking a verb phrase (there are other forms of ellipsis, but for the purpose of this thesis, this definition is the one that will be used). Usually, this type of input will occur as a follow-up question. Thus to resolve ellipsis, the system needs to examine previous questions the player asked to fill in the missing verb phrase. In order to do this, the system must construct a parse tree of the previous questions to isolate their verb phrases.

5.4.1.2 Coreference Resolution

To avoid repetition, the player may input questions that include coreferences such as anaphora. Thus the question parser requires a sub-module capable of resolving these coreferences. Like with ellipsis resolution, it may be necessary to examine previous questions if the context is not obvious from the most recent input alone.

5.4.1.3 Named Entity Recognition and Classification

Since anaphora is often captured by the use of personal pronouns, named entity recognition is required in conjunction with coreference resolution to resolve instances of

words like “he” and “she”. Furthermore, NERC may also be needed to determine the appropriate answer to certain types of “Who” and “Where” questions.

5.4.2 Information Retrieval

The information retrieval module is responsible for finding the answer to the player’s query from a knowledge base of facts. It needs to search through the knowledge base and assign non-negative weights to the different facts for how relevant they are to the actual query. Only those facts whose weights are higher than a certain threshold (the threshold can be selected via experimentation) are considered to be potential answers to the query. The weighting that a fact receives depends heavily on the rarity of its words (other factors can also affect the weight, such as the order of the words). For example, common words like “as” or “the” would not add a lot of weight to a fact because they are used quite often in English sentences. Thus, they would not greatly influence whether or not a fact is chosen. However, nouns or verbs that were also present in the original query may cause a fact to be given more weight because only a few facts are likely to contain them.

5.4.2.1 Logic Handler

Not all facts are necessarily stated explicitly in the knowledge base. Instead, some facts may need to be derived. Thus, a logic module is required. For example, if the knowledge base stores as a fact, “*Tim is Jane’s father*”, but the player queries, “*Who is Tim’s daughter*?”, the system may not be able to explicitly find the fact. However, if the system also stores as a fact, “*Jane is a female*”, then it has all the information it needs to answer the player’s query. It just needs to logically conclude based on some rules that if Tim is Jane’s father and Jane is a female, then Jane is Tim’s daughter.

5.4.2.1.1 Ontology

As mentioned in the previous subsection, the information retrieval system should be able to logically conclude new facts based on facts it already knows. What was not mentioned was where the rules needed to make such new derivations come from. These rules can be extracted from an ontology. An ontology is “a description […] of the concepts and relationships that can exist for an agent or a community of agents” [33].

1. A father is a male and is a parent
2. X is the parent of Y is equivalent to Y is the child of X
3. A daughter is a female and is a child

Figure 5.3 An example of how an ontology works

Figure 5.3 illustrates the rules needed to conclude that Jane is Tim's daughter. Because of Rule 1, the logic handler could deduce that Tim is Jane's parent. Then, because of Rule 2, it could deduce that Jane must be Tim's child. Finally, due to Rule 3, it could conclude that Jane is Tim's daughter, which answers the player's original query.

5.4.2.1.2 Relationship Extraction

Before the logic handler can use an ontology to explore relationships, however, it must first have a way of extracting the initial relationships directly from the facts in the knowledge base. The system cannot assume that the facts in the knowledge base are already in relational form because some facts may have been provided by the player during a conversation. Consequently, it is assumed that all facts are stored as natural-language sentences. For example, for the fact, "*Tim is Jane's father*", the system needs to extract the binary relation *father-of(Tim,Jane)*, as shown in Figure 5.4. For the fact, "*Jane is a female*", it needs to extract the relation *instance-of(Jane,female)*, as shown in Figure 5.5. Once it is explicitly aware of these relations, then it can use the ontology to deduce that Jane is Tim's daughter.

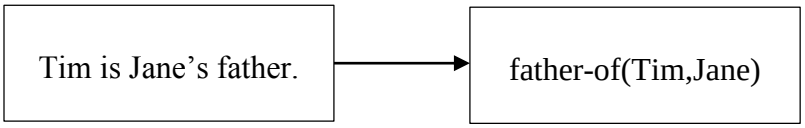

Figure 5.4 First example of extracting a relationship from a fact.

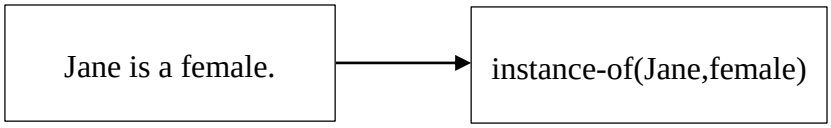

Figure 5.5 Second example of extracting a relationship from a fact.

5.5 Sentiment Analyzer

The sentiment analyzer is responsible for determining the player's opinions about different entities. It must be able to give a score between -1 and +1 of how strong the sentiment is. It must also be able to identify the entity being discussed, as well as the specific aspect of the entity if one is mentioned.

The purpose of having this ability is so the NPC can respond in a sensible manner. As mentioned in Chapter 4, if the player compliments the NPC, it should act in a friendly way. However, if the player insults the NPC, it should act in a more hostile way. Furthermore, if the player complains about an object, then the NPC should try to recommend solutions to remedy the problem. Finally, the NPC may also tell the player whether it agrees or disagrees with the opinion.

5.6 Topic Handler

The topic handler is responsible for determining what topic the NPC should discuss with the player. It can either be a continuation of the current topic, or the topic handler can propose a new topic. During the small talk stage, the topic handler must only select neutral subjects to help build rapport with the player. During the core stage, however, it is allowed to select other types of topics that may be more polarized (e.g., asking which side of an in-game war the player supports).

New topics can be introduced either by having the NPC ask the player a question or remember an event. It is preferable to avoid introducing a new topic by simply reciting a fact from the knowledge base because the effect may be jarring and could ruin the flow of the conversation. Instead, stating facts is only appropriate when continuing a discussion about a particular topic or when answering a query.

5.7 Episodic Memory

The episodic memory module is responsible for recalling episodic memories based on topics discussed during the conversation. Important keywords mentioned during the dialogue trigger the activation of certain memories. When the topic handler decides that the NPC should recall an event, the most recently activated memory is retrieved.

As illustrated in Figure 5.6, an NPC's episodic memories are stratified into three separate memory pools: immediate, short term, and long term. Activated memories are initially searched for in the immediate pool. If none can be found, then they are searched for in the short term pool. If no activated memories can be found in the short term pool either, then they are searched for in the long term pool. Once a memory is retrieved, it is moved from its current pool (i.e., short-term or long-term pool) to the immediate pool, where it can be easily accessed again later.

Like with the knowledge base, episodic memories can be created ahead of time as well as added dynamically during gameplay. When a memory is created, it is assigned a weight to represent how important it is. This assignment can either be given manually or estimated automatically. If no memory pool is manually specified, then the memory is automatically placed into the immediate pool.

Over time, memories either get deleted or moved to the next memory pool (i.e., memories in the immediate pool get moved to the short-term pool, and memories in the short-term pool get moved to the long-term pool), depending on how much weight they carry (if a memory has a low weight, it gets deleted). This deletion of memories is meant to simulate how people can forget mundane events after a while. Furthermore, memories in the long-term pool can be corrupted over time (although, these cannot be deleted). That is, certain keywords are swapped with other keywords that are ontologically related. Keywords, in this sense, are any words in the memory's description that are important to its meaning (e.g., nouns or verbs). The purpose of the corruption is to simulate how people can remember events incorrectly after a long period of time [32].

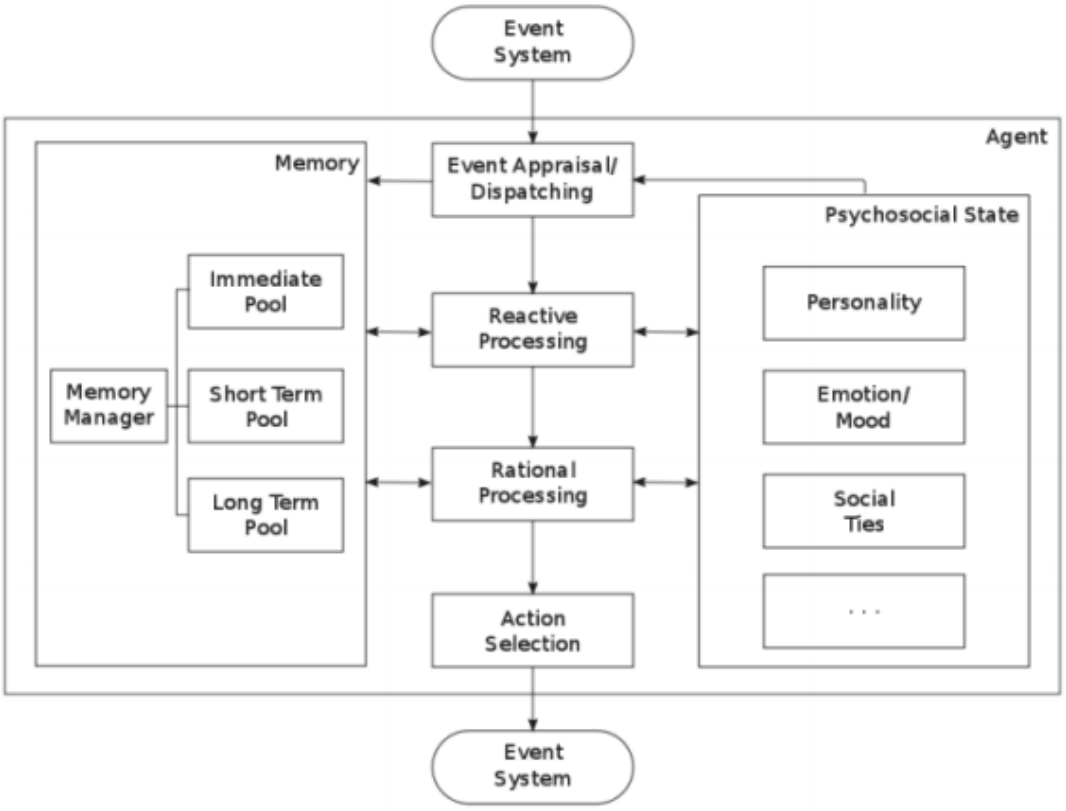

Figure 5.6 Structure of Episodic Memory Module [32]

5.8 Summary

This chapter discussed an object-oriented architecture for the conversation model from Chapter 4. The architecture described in this chapter is an innovative way of using current research in Artificial Intelligence in a novel application. It illustrates how to create a conversation system that could be used in multiple video games. The primary module is the conversation handler, which is the only module the video game interacts with. The conversation handler delegates certain tasks to other modules and ensures that the conversation flows smoothly through the four main stages. The knowledge base is responsible for storing facts the NPC knows about its world. It can be updated dynamically either by the game or the player. The greeting handler is responsible for choosing the appropriate greeting based on the sociological factors that affect politeness, as well as the current situation (e.g., time of day). The question answering system is responsible for answering player queries. It needs to parse the question, and then use

logic to find the answer from the knowledge base. The sentiment analyzer is responsible for determining the player's opinions about entities so that the NPC can offer a sensible response. It gives a score between -1 and +1 and identifies the aspect of the entity that the opinion is about. The topic handler is responsible for determining the topic the NPC should bring up based on how the conversation is progressing. The NPC can either continue with the current topic, ask a question about a new topic, or remember an event. The episodic memory module is responsible for recalling memories about events to add a storytelling element to the conversation. The event themes it recalls are partially decided by the topic handler. These modules work together to ensure a seamless dialogue and to make the NPC appear more autonomous.

Chapter 6

6 Implementation

This chapter discusses an implementation for the framework presented in Chapter 5. The conversation system is written primarily in Java because it is a commonly used object-oriented language in Artificial Intelligence research. However, some parts of the system were written in JavaScript to allow a human to easily edit certain portions of the program without needing to recompile the code, and some other parts were written in Python in order to use a third-party Python library for sentiment analysis. Although not every element from the framework in Figure 5.1 has been fully implemented, this system is still powerful enough that one could engage in a simple dialogue with an NPC. Furthermore, while the system was only designed to handle keyboard input, a video game could easily use a speech-to-text library to allow the player to speak to an NPC rather than type to it. Furthermore, the game could output an NPC's reply as audio by using a speech synthesizer.

6.1 Greeting

The greeting handler is implemented primarily via two JavaScript files. The first file is responsible for opening greetings while the second one is responsible for closing salutations. The Java part of the program is responsible for deciding which of the two files to run and when to run them. In order to bridge the gap between the Java and JavaScript code (i.e. facilitate information passing between the two languages), the program uses the Mozilla Rhino JavaScript engine for Java.

The reason why the greeting handler is implemented in JavaScript rather than Java is because it is designed to be easily editable by a human. Unlike Java which needs to be recompiled whenever changes to the code are made, JavaScript is a scripting language that is interpreted. Consequently, it is quite simple to modify the greeting handler to generate alternate greetings for a variety of situations. This feature is useful because greetings are strongly tied to certain social characteristics as discussed in Chapter 4, so it allows NPCs to act in a way that is canon with their game world.

The default implementation of the greeting handler creates a list of possible greetings depending on the situation. In this implementation, the situations considered are the player's gender, the time of day, and the NPC's social factors. For example, "*Good morning*" would only be said if the time is between 5:00 AM and 12:00 PM. Furthermore, "*Ahoy*" would only be said if the required level of politeness was quite low. Part of the information that the greeting handler needs to decide what is appropriate and what is not comes from the Java part of the program. The conversation handler accepts as input a Java map containing some useful information about the world, the player, and the NPC's social characteristics (e.g., "social distance" → "0.5"; "player's gender" → "female"). This map is passed to the greeting handler, so it can use facts such as the social distance between the NPC and the player, the relative power the player has over the NPC, the imposition of the NPC's greeting, and the player's gender when creating a list of acceptable greetings. The greeting that is returned is randomly selected from this final list.

In order to calculate the required level of politeness, the greeting handler calculates the sum of the NPC's social characteristics (i.e. social distance + relative power + imposition). If the sum is less than or equal to 1.0, a low politeness level is required. If the sum is greater than 1.0 but less than or equal to 2.0, a medium politeness level is required. If the sum is greater than 2.0, a high politeness level is required.

6.2 Small Talk

Like the greeting handler, the small talk portion of the program is implemented primarily in JavaScript. Due to the nature of small talk, it makes sense that this part should be designed to be easily editable by a human as well. In the current implementation, there is no topic handler that explicitly determines acceptable topics for small talk (more information is given in Section 6.4.8). Instead, the topics are hardcoded in the JavaScript file.

In the default implementation, three topics are provided: how the player is feeling, the weather, and an arbitrary compliment on an aspect of the player's appearance. For each of these topics, a few different sentences to express the same meaning are included to

avoid predictability. For example, when asking how the player is feeling, the NPC could ask any of the following: "*How are you?"*, "*How are things?*", "*How's life?*", "*How's it going?*" These topics were chosen because they are neutral and relatively common; however, there is no reason why this particular set must be used. In fact, with some knowledge of JavaScript programming, it is quite easy for a person to modify the small talk module to include other topics. Due to time constraints, social factors were not taken into consideration when choosing the wording for small talk. However, they could be readily included since the small talk module has access to the same Java map as the greeting handler.

To avoid repetition, the small talk module keeps track of topics it has already discussed. Due to the fact that the scope of the JavaScript program ends once it returns control to the Java program, it stores covered topics in a Java list where it can access them when it is run again. Thus, the small talk module is guaranteed to return a new topic each time the player says something for the entire duration of the conversation.

The small talk module has access to player responses, so it can give reasonable replies for certain situations. For example, if the NPC had asked, "*How are you?*", and the player had responded, "*Fine, thank you*", then the NPC may reply with, "*That's good to hear!*" However, if the player had responded, "*I'm not well*", then the NPC may reply with, "*I'm sorry to hear that*". The choice of how the NPC should respond to various player input is rule-based.

Once the small talk module has exhausted its topics, it returns a default message to the player indicating the need to switch to the core of the conversation. An example of such a message could be, "*So is there anything that I can help you with?* \t" Note that it also ends its message with the tab symbol, \t, to invisibly tell the conversation handler that it needs to switch the stage to the core.

In the event that the player would like to bypass small talk altogether (e.g., the player just wants to get some information quickly), he or she can ask a question at any time. Once a question has been posed, the conversation handler switches to the core stage immediately, and the NPC will attempt to answer the query.

6.3 Core

Due to the complexity of a conversation's core, its implementation went through various stages to reach the state it is now. This section will discuss previous implementations as well as the current implementation, and it will highlight the significant changes that were made.

6.3.1 Implementation 1

The original implementation of the core revolved around using a MySQL database to store facts. Some examples of tables included the main NPC table, the NPC relationship table, and the NPC attribute table. Tables 6.1, 6.2, and 6.3 illustrate the structures of these three tables. For this particular implementation, only questions were acceptable input; statements were ignored.

ID	First Name	Last Name	Date of Birth	Gender
1	John	Smith	1945-07-02	Male
2	Katherine	Smith	1987-09-05	Female

Table 6.1 Example of main NPC table in MySQL database

FirstName1	LastName1	RelationshipType	FirstName2	LastName2
John	Smith	Daughter	Katherine	Smith
Katherine	Smith	Father	John	Smith

Table 6.2 Example of NPC relationship table in MySQL database

FirstName	LastName	Attribute
John	Smith	Handsome
John	Smith	Intelligent

Table 6.3 Example of NPC attribute table in MySQL database

To find the answer to the player's query, the English question had to be converted into SQL. To accomplish this task, regular expression matching was used. A text file contained a series of regular expressions followed by their SQL translations. Figure 6.1 is an example of an actual rule from this file. The line that begins with a hash symbol is a comment indicating the pattern being matched. The line with just a numeral represents the ranking of the rule relative to other rules (a higher number represents a higher rank). The third line is the actual regular expression to be matched, and the fourth line is the SQL code to replace it with. The RuleBank class read Rule objects from this text file and stored them for later use.

Using the Stanford Core NLP library, player input was first converted into the following format: lemma/part of speech/named entity. Then the rules from the rule bank were used to see if the input matched any of the regular expressions. If it did, it was converted into an SQL query, and the result (if there was one) was returned. If it did not match, then a message was returned indicating that the NPC did not understand the question.

```
# who is <FIRST_NAME> <LAST_NAME>'s <RELATIONSHIP_TYPE>?
14
.*who/WP/O be/VB.?/O (.+)/NNP/PERSON (.+)/NNP/PERSON 's?/POS/O
(.+)/NN/O.*
SELECT npc.FirstName, npc.LastName FROM relationships INNER JOIN npc
ON npc.FirstName=relationships.FirstName2 AND
npc.LastName=relationships.LastName2 WHERE
relationships.RelationshipType='$3' AND relationships.FirstName1='$1'
AND relationships.LastName1='$2';
```

Figure 6.1 Example of regular expression rule to convert an English question to an SQL query

6.3.1.1 Issues

Using a database to answer questions proved to be quite difficult in practice. Often, SQL queries for seemingly simple English questions would become quite complex due to the need to join tables to find the answer. For example, the question, "*Who is Katherine Smith's father?*" would be translated into the complicated SQL seen in Figure 6.1. Furthermore, these complicated queries occasionally caused some noticeable delay in retrieving the answer to a question.

6.3.2 Implementation 2

The second implementation of the core revolved around using a Prolog database to store facts. The Prolog database was structured similarly to the original MySQL database (see Figure 6.2), except that it also included rules to derive new facts (see Figure 6.3). Having inference rules allowed for a larger variety of possible queries without making them as complicated as in MySQL.

```
%%%%%%%%%%%%%%%%%%%%%% NPC Table %%%%%%%%%%%%%%%%%%%%%%%%
%   ID,     First Name, Last Name,  Gender,     DOB (YYYY,MM,DD)
npc(npc1,   'Jim',      'Parsons',  'male',     date(1960,02,08)).
npc(npc2,   'Susan',    'Parsons',  'female',   date(1985,10,26)).
npc(npc3,   'Roy',      'Parsons',  'male',     date(1987,05,16)).
npc(npc4,   'June',     'Smith',    'female',   date(1995,12,03)).
npc(npc5,   'Angela',   'Smith',    'female',   date(1990,07,13)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 6.2 Example of NPC table in Prolog database

```
father(X,Y) :- parent(X,Y), male(X).
mother(X,Y) :- parent(X,Y), female(X).
brother(X,Y) :- sibling(X,Y), male(X).
sister(X,Y) :- sibling(X,Y), female(X).
uncle(X,Y) :- brother(X,W), parent(W,Y).
aunt(X,Y) :- sister(X,W), parent(W,Y).
```

Figure 6.3 Example of some inference rules for Prolog database

Just like with the MySQL implementation, the Prolog implementation required English questions to be converted into queries. Once again, regular expression matching was used to accomplish this task. The text file containing the translation rules was structured almost identically as before, except instead of MySQL queries, it contained Prolog queries. Figure 6.4 shows an example of how the text file was restructured. Note that the Prolog query is much simpler than its MySQL counterpart.

```
# who is <FIRST_NAME> <LAST_NAME>'s? <RELATIONSHIP_TYPE>?
14
.*who/WP/O be/VB.?/O (.+)/NNP/PERSON (.+)/NNP/PERSON 's?/POS/O
(.+)/NN/O.*
npc(ID1,'$1','$2',_,_), $3(ID2,ID1), npc(ID2,FirstName,LastName,_,_).
```

Figure 6.4 Example of regular expression rule to convert an English question to a Prolog query

6.3.2.1 Issues

While the Prolog implementation was certainly an improvement over the MySQL one, it still suffered many of the same major problems. Firstly, creating rules for translating English questions into Prolog queries was quite time-consuming since it had to be done manually. Secondly, it was very difficult to anticipate different types of questions that the player might ask. Finally, converting the result of a Prolog query into an English response was quite challenging.

6.3.3 Current Implementation

The current implementation of the core revolves around using text files as the knowledge base. Each line of a text file contains a fact written as an English sentence. Furthermore, special meta-data is added to the end of a fact to indicate the types of questions it answers. Figure 6.5 is an example of such a knowledge base.

```
Jane is in Paris. <who> <where>
Jane is Tim and Mary's daughter. <who>
Tim and Mary are Jane's parents. <who>
Tim is Jane's father. <who>
Mary is Jane's mother. <who>
Jane likes to eat apple pie. <who> <what> <does>
Jane likes Billy. <who> <does>
Billy is Jane's boyfriend. <who>
Jane is Billy's girlfriend. <who>
```

Figure 6.5 Example of knowledge base stored in a text file

To find facts based on player queries, the Apache Lucene library [34] is used. It has high-quality search engine capabilities, making it a useful tool for information retrieval. Each fact from the knowledge base is added to Lucene as a document. Afterwards, the player's question is used to query Lucene, where it then returns the highest-ranked document (fact) that best matches the query. To ensure it returns reasonable results, the conversation system requires that it must have a confidence level of at least 0.8 (as computed by Lucene using the default settings); otherwise, the NPC will indicate that it does not know the answer.

This implementation works exceedingly better than the previous two because the player has more flexibility in how to word questions. While there are still instances where a question that should have an answer returns nothing, this event occurs much less frequently. Furthermore, it is now trivial for the player to give information to the NPC. Any time the player makes a statement, it gets stored in a special text file of player facts (if a player fact is the result of the player answering a question, then the original question is stored as meta-data; for other types of facts that the player volunteers, there is currently no way to add extra meta-data). Then, if the player queries the NPC on something he or she had previously said, the NPC can usually respond correctly.

6.3.3.1 NPC Asks Questions

Because it is so simple to store information that the player gives the NPC, it makes sense to allow the NPC to ask the player questions. At this time, however, this feature is implemented in a rudimentary way due to the lack of a topic handler (see Section 6.4.8 for more details), so the NPC currently retrieves a list of questions to ask the player from a text file.

6.3.3.2 Sentiment Analysis

To determine the player's sentiment, the sentiment analysis tool from the TextBlob library [35] was used. Because it is a Python library, a custom Python file that calls this library is run as an external process. The console output of the Python file is treated as input to the Java program. The TextBlob library only has the capability to provide the overall sentiment of the input and a subjectivity score; it is unable to identify the entity that the sentiment is about (see Section 6.4.7 for more details). Consequently, its usefulness is limited.

6.3.3.3 Episodic Memory

To add a storytelling element to the conversation, the episodic memory module described in Kope, Rose, and Katchabaw [32] was used. This module allows for memories to be stratified in three different memory pools: immediate, short-term, and long-term. It also ages the memories and moves them to the next pool if their weight is high enough. Furthermore, it corrupts long-term memories after a while by swapping out keywords with other words that are categorically related.

When the player provides input, each word is sent to the episodic memory system as a prompting keyword. If any memory becomes activated as a result of this prompting, then it gets returned.

6.3.3.4 Issues

The biggest issue with this implementation is that it cannot makes inferences. If a fact is not explicitly stated in the knowledge base, then the NPC cannot answer the player's

question. With a logic handler, then this problem would be solved. However, due to time constraints, one was not implemented (see Section 6.4.6 for more details).

It should also be noted that the system assumes there are no typos or mistakes in the player's input. If typos are present, it will not be able to find the information that the player was looking for. Originally, a spellchecking library was used to assist in such a situation, but it would occasionally misclassify a correctly typed word as incorrect and then change it to something else. Consequently, it was removed from the conversation system.

6.4 Unimplemented Features

Due to constraints on time and other resources, not all the features mentioned in Chapter 5 were included. This section discusses the features that were either omitted or only partially implemented.

6.4.1 Ellipsis Resolution

With more time, ellipsis resolution would have been implemented as follows:

- Store the previous question that the player asked as well as its parse tree.
- Use an NLP library to create a parse tree for the current query. If it is missing a verb phrase, then add the verb phrase of the most recent question. Store this modified question and its parse tree.
- Run the modified query.

6.4.2 Coreference Resolution

Since coreference resolution is already a large area of research, an NLP library would have been used to perform this task. This feature was omitted because a library could not be found that could do it quickly enough for a video game. Stanford Core NLP [36] was initially used, but it proved quite slow.

6.4.3 Named Entity Recognition and Classification

Like with coreference resolution, Stanford Core NLP was initially used, but it was too slow. A faster library that could run in real-time could not be found.

6.4.4 Ontology

Given enough time, an ontology would have been created using OWL [37], the Web Ontology Language. Specifically, a library like Apache Jena [38] would have been used to access the ontology since it has a built-in reasoner for OWL/lite, which can be used to make inferences about relations.

6.4.5 Relationship Extraction

Since relationship extraction is its own area of research, this project would have required the use of a freely available library rather than create a new implementation. However, it proved quite difficult to find such a library, so relationship extraction was omitted. Primarily, a library was needed that could find binary relations in sentences.

6.4.6 Logic Handler

Since the logic handler depends on using both an ontology and a relationship extraction tool, it was not implemented. Had these two necessary features been available, then it would have performed the following steps if an answer to the original query could not be found:

- Use the relationship extraction tool to find relationships in the query.
- Use the ontology to find equivalent relationships (e.g., "*A is the child of B*" is equivalent to "*B is the parent of A*"). Also use the ontology to find super-classes/sub-classes (e.g., "*A father is a parent*").
- Search through the knowledge base for these equivalent relationships or super-classes/sub-classes. If something is found, then it is likely the answer to the player's question. If nothing is found, then the NPC probably does not know the answer.

6.4.7 Sentiment Analyzer

The sentiment analyzer was only partially implemented due to the decision to use the TextBlob library [35]. TextBlob's sentiment analysis tool can assign a score for how subjective a sentence is and for the sentence's overall sentiment; however, it is unable to indicate the entity the sentiment is about. Thus, its utility is fairly limited. With more time, a better sentiment analysis tool would have been used instead.

6.4.8 Topic Handler

Had it been created, the topic handler would have read the following items from a JSON file and stored them in memory: general topics to discuss, different utterances within each topic (including potential replies to some player statements), how emotionally charged the topics are, and how important the topics are for the game's storyline. Using this information, it would have been responsible for sending neutral topics to the small talk module. It would have also been responsible for deciding when to switch topics (e.g., if there are no more utterances to say for a topic, or if a particular topic is important for the plot of the game), and determining if the player has changed topics. The two main reasons why JSON was selected is because it can be edited by a human, and it is trivial to convert it to a Java object by using third-party Java libraries.

6.5 Minecraft Implementation

In order to test the conversation system, it was added as a mod to the popular game Minecraft [39]. The reason why Minecraft was chosen over other games is because it is written entirely in Java, making it quite easy to integrate the conversation system into it. Furthermore, this game already has a large modding community, so there is some documentation available to assist in the modding process.

Plate 6.1 Screenshot of conversation system added to Minecraft. © 2014 Mojang. Used with permission.

In Minecraft, there are special NPCs called villagers which are human characters. In the original game, they are not much smarter than barn animals and have no conversation abilities. Consequently, they were the best characters to test the system on. When the human player walks up to a villager, the conversation mod allows him or her to open a chat window by pressing the C key on the keyboard. In this chat window, the player can type to the NPC and receive a response (see Plate 6.1).

Minecraft uses an event-driven implementation. Thus, artificial intelligence modules are added to the villagers as a series of tasks (e.g., there is an AI module for swimming, trading with the player, going indoors, etc.). In order to incorporate the conversation system into Minecraft, a special AI module was created called EntityAIConversation. This module runs whenever the villager is watching the player at a close distance. While running, it checks another module called mod_Conversation to see if the player has submitted input to the NPC. If the answer is yes, it fetches the input and sends it to the conversation system. Then it sends the NPC's output back to mod_Conversation, where the chat GUI is instructed to write the output to the screen.

6.6 Summary

This chapter discussed a partial implementation for the framework mentioned in Chapter 5. This implementation was written mainly in Java, with some parts written in JavaScript and Python. The greeting handler and small talk module were written in JavaScript to make them easy for a human to edit and customize them. The core was written primarily in Java and included the following features: the player can ask the NPC questions, the NPC can ask the player questions, the NPC can recall episodic memories, the player can give the NPC information via statements, and the overall sentiment of the player's input can be recognized. The following features were omitted from this implementation due to time constraints: ellipsis resolution, coreference resolution, named-entity recognition and classification, an ontology, relationship extraction, the logic handler, and the topic handler. Sentiment analysis was only partially included due to TextBlob's inability to identify the entity that the sentiment is about. In order to test the conversation system in an actual game, a Minecraft mod was created to allow the player to converse with villager NPCs.

Chapter 7

7 Testing and Results

This chapter discusses the methods that were used to test the performance of the conversation system described in Chapter 6. Although this system is only a partial implementation of the architecture described in Chapter 5, it is robust enough to engage in simple dialogue with an NPC. The testing methods examined how well individual modules perform on their own, how well the modules work together in the overall flow of the conversation, and the computational performance impact during actual gameplay in Minecraft. The following sections go into more detail about how each part is going to be tested.

7.1 Greeting

The conversation system should greet the player in an appropriate way. Appropriate, in this sense, means that the greeting matches the required level of politeness and makes sense for the situation. Sections 7.1.1 and 7.1.2 elaborate on how these two aspects of a greeting were tested.

7.1.1 Required Politeness Level

The conversation system should be able to greet the player with the desired level of politeness. For example, the NPC should not greet the player with, “Hey there” if the politeness level is supposed to be high. The social factors affecting politeness – social distance, relative power of listener over speaker, and imposition – are passed to the greeting handler via a Java map. The greeting handler calculates the required level of politeness from these factors and then selects appropriate greetings. The following sets of values were used to test if the politeness is what it should be:

- Social Distance: 0.1; Relative Power: 0; Imposition: 0 → Low politeness
- Social Distance: 0.3; Relative Power: 0.6; Imposition: 0.2 → Medium politeness
- Social Distance: 0.9; Relative Power: 0.5; Imposition: 0.7 → High politeness

In this implementation, there are three possible levels of politeness: low, medium, and high. Some examples of greetings with a low politeness level are, "*Hey there*" and "*Ahoy*". Furthermore, some examples of greetings with a medium politeness level are, "*Salutations*" and "*Hi*". Finally, some examples of greetings with a high politeness level are, "*Hello*", "*How do you do*", and "*Greetings*". Sections 7.1.1.1, 7.1.1.2, and 7.1.1.3 illustrate actual program output for a required politeness level of low, medium, and high respectively.

7.1.1.1 Low Politeness

```
Player Input: Hello.
NPC Response: Hey there. How are you?
```

Figure 7.1 Output 1 of NPC greeting player with low politeness

```
Player Input: Hello.
NPC Response: Ahoy. How's life?
```

Figure 7.2 Output 2 of NPC greeting player with low politeness

7.1.1.2 Medium Politeness

```
Player Input: Hello.
NPC Response: Salutations. How's life?
```

Figure 7.3 Output 1 of NPC greeting player with medium politeness

```
Player Input: Hello.
NPC Response: Hi. Did you get a haircut?
Your hair looks really nice!
```

Figure 7.4 Output 2 of NPC greeting player with medium politeness

7.1.1.3 High Politeness

```
Player Input: Hello.
NPC Response: Hello. How are you?
```

Figure 7.5 Output 1 of NPC greeting player with high politeness

```
Player Input: Hello.
NPC Response: How do you do. How's life?
```

Figure 7.6 Output 2 of NPC greeting player with high politeness

```
Player Input: Hello.
NPC Response: Greetings. Can you believe
all this rain?
```

Figure 7.7 Output 3 of NPC greeting player with high politeness

Note that in Figure 7.7, the NPC makes a reference to there being a lot of rain. While this portion of the NPC's utterance is small talk rather than a greeting, it is still worth mentioning that the NPC is making this statement because the small talk module was provided input indicating that it is raining. Had it received input indicating some other kind of weather, it would have never made this comment.

7.1.2 Situational Factors

The conversation system must consider the context when selecting a greeting. For example, it should not say, "*Hello, sir*," if the player identifies as a female in the game. Moreover, it should not say, "*Good morning*," if it supposed to be night time. Thus, to ensure the system is taking these situational factors into consideration, the following tests were performed:

- Indicate to the system that the player is a male; then indicate the player is female
- Adjust the in-game time

Greetings that are situation-insensitive such as, "*Hello*" were ignored. Only greetings that assume a particular situation (whether the assumption is correct or not) were tested for appropriateness.

In this implementation, only two situations were considered: gender and time of day. Section 7.1.2.1 illustrates output where the NPC had decided to include the player's gender information in its greeting. It chose the correct gender based on information passed to the greeting handler. Section 7.1.2.2 shows output where the NPC had decided to select a greeting based on the time of day. It correctly chose the greeting based on the player's computer clock. Figure 7.12 is special because it demonstrates the NPC choosing a greeting based on both politeness level and time of day.

7.1.2.1 Gender

```
Player Input: Hello.
NPC Response: How do you do, m'lady. I
really like your shoes.
```

Figure 7.8 Output of NPC greeting a female player

```
Player Input: Hello.
NPC Response: Greetings, gentle sir. How
are things?
```

Figure 7.9 Output of NPC greeting a male player

7.1.2.2 Time of Day

```
Player Input: Hello.
NPC Response: Good morning. That's a nice
dagger you have there.
```

Figure 7.10 Output of NPC greeting player in the morning

```
Player Input: Hello.
NPC Response: Good afternoon. I heard
there's going to be a storm.
```

Figure 7.11 Output of NPC greeting player in the afternoon

```
Player Input: Hello.
NPC Response: Good day. How are you?
```

Figure 7.12 Output of NPC formally greeting player in the afternoon

```
Player Input: Hello.
NPC Response: Good evening. Did you get a
haircut? Your hair looks really nice!
```

Figure 7.13 Output of NPC greeting player in the evening

7.2 Small Talk

Since the small talk stage of the conversation is when the player has the opportunity to build rapport with the NPC, it is important that the NPC appears as autonomous and believable as possible. In order to accomplish this goal, the NPC must avoid repeating itself and must give appropriate replies to the player. Sections 7.2.1 and 7.2.2 go into more detail about how the small talk module were tested.

7.2.1 Avoid Repetition

When engaging in small talk, the system should avoid repeating topics it has already discussed. For example, if the system had already asked how the player was, it should not ask again. Since the small talk module is executed each time the player provides input for the small talk stage of the conversation, it must store information about what topics it has mentioned outside of the JavaScript program. In this implementation, a Java list object stores previous topics and sends that information to the JavaScript program. Aside from the fact that the NPC would lose credibility as an intelligent agent if it repeats itself unnecessarily, if the small talk module is not aware of what topics it has already mentioned then it cannot know when to signal to the main Java program that it needs to

switch to the core stage. Thus, it is important that this feature works properly. To test that it is working as expected, the following steps were done:

- Engage in small talk with the system
- Make note if it repeats any topics
- Continue conversing until it exhausts its topics and see if it switches to the core stage.

Figure 7.14 illustrates actual output from this short interaction. The order of topics is randomly chosen, and within each topic, there are multiple ways of expressing the same meaning. Consequently, Figure 7.14 is an example of only one possible exchange that could have occurred. In this particular case, the order of topics was as follows: the weather, a compliment, and asking how the player is feeling. Once the small talk module recognized that it ran out of topics, it prompted the player to switch to the core stage.

```
NPC Response: Good day. I heard there's
going to be a storm.
Player Input: That's not good. I don't
like storms.
NPC Response: That's a nice dagger you
have there.
Player Input: Thank you.
NPC Response: You're welcome! How are
you?
Player Input: I'm pretty good.
NPC Response: I'm glad. So is there
anything that I can help you with?
```

Figure 7.14 Output of NPC engaging in small talk with player

7.2.2 Appropriate Replies to Player Responses

The system must be able to respond appropriately to what the player says. In particular, it should be able to acknowledge when the player has said something in reply to an NPC's question or statement. For example, if the system gives the player a compliment and the player says, "*Thank you*," then the system should acknowledge the player by replying with, "*You're welcome*." The following types of interactions were observed and tested:

- If the system compliments the player, respond with
    - A message of thanks (e.g., "*Thanks*"/"*Thank you*")
    - A neutral message (e.g., "*OK*")
- If the system asks how the player is, respond with
    - Something indicating that the player is doing well (e.g., "*I'm fine, thanks*")
    - Something indicating that the player is not doing well (e.g., "*I'm not well*")
    - Something unrelated

In Section 7.2.2.1, there are examples of actual output where the system detects whether or not the player thanked the NPC. In Section 7.2.2.2, there are examples of output where the NPC responds based on whether the player is doing well or not, or if the player chose to ignore the question.

7.2.2.1 Compliment

```
NPC Response: That's good. I really like
your shoes.
Player Input: Thanks so much. That's very
kind of you to say.
NPC Response: You're welcome! I heard
there's going to be a storm.
```

Figure 7.15 Output of NPC acknowledging thanks from the player

```
NPC Response: I'm glad. I really like
your shoes.
Player Input: OK.
NPC Response: Can you believe all this
rain?
```

Figure 7.16 Output of NPC moving forth with the dialogue when no thanks were given

7.2.2.2 Asking How the Player Is

```
NPC Response: Good afternoon. How's it
going?
Player Input: It's going wonderfully,
thanks.
NPC Response: That's good to hear! I
heard there's going to be a storm.
```

Figure 7.17 Output of NPC acknowledging that player is doing well

```
NPC Response: How's life?
Player Input: Terrible, unfortunately.
NPC Response: Oh no! Did you get a
haircut? Your hair looks really nice!
```

Figure 7.18 Output of NPC acknowledging that player is not doing well

```
NPC Response: How do you do, gentle sir.
How's it going?
Player Input: I can see a tree.
NPC Response: Did you get a haircut? Your
hair looks really nice!
```

Figure 7.19 Output of NPC moving forth with the conversation when the player gives no indication of wellbeing

7.3 Core

During the core of the conversation, the system must be able to accomplish several tasks to maintain the illusion of intelligence. These tasks include recognizing when the player asks a question or makes a statement, answering questions appropriately, understanding contractions, responding suitably to statements the player may make, asking the player questions, remembering the answer the player gave to a question, and recalling events related to the topic being discussed. The following sections elaborate on how each of these tasks were tested.

7.3.1 Recognize Question vs Statement

The conversation system must be able to differentiate between the player asking a question and making a statement. It would be trivial to assume that if a sentence ends with a question mark then it is a question, and if it ends with a period then it is a statement. However, this assumption could potentially be wrong. For example, the player may accidentally omit punctuation, meaning the system could confuse the question, "Who are you," for a statement. Thus, the system was tested using the following types of input to see how well it performs:

- "W6" question ending in a question mark (e.g., "*Who are you?*")
- "W6" question ending in a period (e.g., "*Who are you.*")
- "W6" question not ending with any punctuation (e.g., "*Who are you*")
- Yes/No question ending in a question mark (e.g., "*Is Jane in Paris?*")
- Yes/No question ending in a period (e.g., "*Is Jane in Paris.*")
- Yes/No question not ending with any punctuation (e.g., "*Is Jane in Paris*")
- "Did you know…" style questions (e.g., "*Did you know that Jane is in Paris?*")
- Statement ending in a period (e.g., "*Paris is in France.*")
- Statement not ending with any punctuation (e.g., "*Paris is in France*")
- Statement converted into a question (e.g., "*Paris is in France?*")
- Imperative statements (i.e. commands) requesting information (e.g., "*Tell me where Jane is.*")

Figures 7.20 to 7.30 illustrate the different tests that were performed to see how well the system could detect questions vs statements. In all cases, the system performed quite well.

```
Player Input: Who are you?
NPC Response: I am Edmond Dantes. I am
the Count of Monte Cristo.
```

Figure 7.20 Recognizing a "W6" question ending in a question mark

```
Player Input: Who are you.
NPC Response: I am Edmond Dantes. I am
the Count of Monte Cristo.
```

Figure 7.21 Recognizing a "W6" question ending in a period

```
Player Input: Who are you
NPC Response: I am Edmond Dantes. I am
the Count of Monte Cristo.
```

Figure 7.22 Recognizing a "W6" question ending with no punctuation

```
Player Input: Is Jane in Paris?
NPC Response: Yes.
```

Figure 7.23 Recognizing a Yes/No question ending in a question mark

```
Player Input: Is Jane in Paris.
NPC Response: Yes.
```

Figure 7.24 Recognizing a Yes/No question ending in a period

```
Player Input: Is Jane in Paris
NPC Response: Yes.
```

Figure 7.25 Recognizing a Yes/No question ending with no punctuation

```
Player Input: Did you know that Jane is
in Paris?
NPC Response: Yes, I did.
```

Figure 7.26 Recognizing a "Did you know..." question

```
Player Input: Paris is in France.
NPC Response: That's good to know.
```

Figure 7.27 Recognizing a statement ending in a period

```
Player Input: Paris is in France
NPC Response: That's good to know.
```

Figure 7.28 Recognizing a statement ending with no punctuation

```
Player Input: Paris is in France?
NPC Response: Yes.
```

Figure 7.29 Recognizing a statement converted into a question

```
Player Input: Tell me where Jane is.
NPC Response: Jane is in Paris.
```

Figure 7.30 Recognizing an imperative statement requesting information

7.3.2 Answer Questions Appropriately

When the player asks the NPC a question, the answer should be appropriate for the type of question asked. For example, a question asking who someone is should not be answered with where that person is at the moment. To test how well the system gives appropriate answers, the following types of input were given:

- "W6" questions
- Yes/No questions
- "Did you know…" type questions
- Statements converted into questions
- Imperative statements requesting information

Figures 7.31 to 7.41 illustrate the results of these tests. Overall, the system performed remarkably well. The only noticeable issue is that the system responds by using full sentences quoted directly from the knowledge base. While this is generally suitable when being formal, it would sound somewhat unnatural when engaging in casual conversation. Ideally, it should be able to replace nouns with pronouns and only respond with the clause that contains the relevant answer (e.g. to answer the question in Figure 7.34, the NPC could have responded with, "*Because I got a promotion.*").

```
Player Input: Who is Danielle Smith?
NPC Response: Danielle Smith is the vice-
chancellor of the House.
```

Figure 7.31 NPC answering a "Who" question

```
Player Input: What does Sam sell?
NPC Response: Sam the blacksmith sells
swords and other weapons in Cambridge.
```

Figure 7.32 NPC answering a "What" question

```
Player Input: Where is Danielle Smith?
NPC Response: Danielle Smith is visiting
her aunt in Essex County.
```

Figure 7.33 NPC answering a "Where" question

```
Player Input: Why are you happy?
NPC Response: I am happy because I got a
promotion.
```

Figure 7.34 NPC answering a "Why" question

```
Player Input: When will Jane be back?
NPC Response: Jane will be back at 9:00
PM.
```

Figure 7.35 NPC answering a "When" question

```
Player Input: How do I find the pot of
gold?
NPC Response: To find the pot of gold,
you need to look for the rainbow.
```

Figure 7.36 NPC answering a "How" question

```
Player Input: Is Jane not seeing Billy
anymore?
NPC Response: No. Jane is seeing Billy.
```

Figure 7.37 NPC answering a Yes/No question where the answer is no

```
Player Input: Is Jane seeing Billy?
NPC Response: Yes.
```

Figure 7.38 NPC answering a Yes/No question where the answer is yes

```
Player Input: Did you know that Jane
isn't seeing Billy anymore?
NPC Response: Actually, that's not true.
Jane is seeing Billy.
```

Figure 7.39 NPC answering a "Did you know..." question

```
Player Input: Jane is in Paris?
NPC Response: Yes.
```

Figure 7.40 NPC answering a statement converted into a Yes/No question

```
Player Input: Tell me what Jane likes to
eat.
NPC Response: Jane likes to eat apple
pie.
```

Figure 7.41 NPC answering an imperative statement requesting information

7.3.3 Contractions

The player may potentially decide to use contractions in either questions or statements. Since contractions are grammatically correct, the system should be able to handle them. Consequently, one would expect that two questions that differ only by their use of contractions would yield the same response from the NPC. Thus, the following types of input were used to test the system:

- Question with no contractions (e.g., "*Who are you?*")
- Question with contractions (e.g., "*Who're you?*")

Figures 7.42 to 7.45 show that the system is able to recognize what the contracted words mean and then respond with the expected answer.

```
Player Input: Who are you?
NPC Response: I am Edmond Dantes. I am
the Count of Monte Cristo.
```

Figure 7.42 NPC answering a question without contractions.

```
Player Input: Who're you?
NPC Response: I am Edmond Dantes. I am
the Count of Monte Cristo.
```

Figure 7.43 NPC answering the same question with contractions.

```
Player Input: Where is Jane?
NPC Response: Jane is in Paris.
```

Figure 7.44 NPC answering another question without contractions.

```
Player Input: Where's Jane?
NPC Response: Jane is in Paris.
```

Figure 7.45 NPC answering the other question with contractions.

7.3.4 Respond to Player Statements

The system should be able to respond to different types of statements the player may make. In particular, the system should demonstrate that it can recognize the player's sentiment based on how it replies. Consequently, it was tested under the following conditions:

- Player makes a neutral statement (e.g., "*I'm going to the movies.*")
- Player makes a positive statement about something that is not the NPC (e.g., "*I love rock music!*")
- Player makes a positive statement about the NPC (e.g., "*You're really interesting to talk to.*")
- Player makes a negative statement about something that is not the NPC (e.g., "*I hate eating fish.*")
- Player makes a negative statement about the NPC (e.g., "*You're stupid.*")

In general, it can classify these types of statements correctly, as seen in Figures 7.46 to 7.50. However, it will occasionally incorrectly interpret a statement as being a compliment/insult due to the presence of the words "*you*", "*you're*", or "*your*", as illustrated in Figures 7.51 and 7.52. With a more robust sentiment analysis tool capable of detecting the entity that the sentiment is about, this phenomenon would not occur.

```
Player Input: I'm going to the movies.
NPC Response: That's good to know.
```

Figure 7.46 NPC responding to a neutral statement

```
Player Input: I love rock music!
NPC Response: It seems to make you pretty
happy!
```

Figure 7.47 NPC responding to a positive statement

```
Player Input: You're really interesting
to talk to!
NPC Response: Thank you for the
compliment!
```

Figure 7.48 NPC responding to a compliment

```
Player Input: I hate eating fish.
NPC Response: You have such a low opinion
about that.
```

Figure 7.49 NPC responding to a negative statement

```
Player Input: You're stupid.
NPC Response: Right back at you, jerk.
```

Figure 7.50 NPC responding to an insult

```
Player Input: Your friend said that Mary
is stupid.
NPC Response: Right back at you, jerk.
```

Figure 7.51 NPC incorrectly interpreting a statement as an insult

```
Player Input: You said that I am very
pretty.
NPC Response: Thank you for the
compliment!
```

Figure 7.52 NPC incorrectly interpreting a statement as a compliment

7.3.5 Ask Questions

The system should have the capability of asking the player questions. However, it should not repeat a question it had previously asked in the conversation unless the player did not answer (see the next section for more details). Thus, to test that questions are not being repeated unnecessarily, a conversation with the system was carried on until all of the NPC's questions have been answered.

Figure 7.53 illustrates a conversation with the system to test whether it would repeat itself. Portions of the conversation where the system did not ask any questions were omitted. As demonstrated, the system asked each question only once. Although not explicitly shown here, it did not ask any more questions once it finished its list.

```
NPC Response: I am Edmond Dantes. I am
the Count of Monte Cristo. What kind of
music do you like?
Player Input: I like rock music.
…
NPC Response: Jane is in Paris. Do you
have plans for the weekend?
Player Input: I'm going to the movies.
NPC Response: That's good to know. What's
your favourite movie?
Player Input: The Mummy.
…
NPC Response: Yes.  Do you like this
city?
Player Input: Yes.
…
NPC Response: That's good to know. What
was the last book you read?
Player Input: The Phantom of the Opera.
…
```

Figure 7.53 NPC asking the player questions without repeating what it asks

7.3.5.1 Repeat Question If Player Did Not Answer

The system may ask the player a question, but instead of answering, the player may decide to ask about something else instead. Since the original question was not answered, the system should attempt to ask it again at a later point in the conversation. To test if this feature works properly, the following steps were performed:

- When the system asks a question, respond with a different question
- Continue conversing with the system for several minutes until it has exhausted other questions it was going to ask
- See if the original question is asked again and then answer it

Figure 7.54 illustrates a conversation where the player did not answer the first question. Instead, she asked a question of her own. Once the system asked its other questions, it asked the original question again. Once again, portions of the conversation where the system did not ask a question were removed.

```
NPC Response: Jane is in Paris. What kind
of music do you like?
Player Input: Where is Danielle?
…
NPC Response: I am Edmond Dantes. I am
the Count of Monte Cristo. Do you have
plans for the weekend?
Player Input: Yes, I'm going to the
movies.
…
NPC Response: Jane is seeing Billy.
What's your favourite movie?
Player Input: The Mummy.
…
NPC Response: I don't know. Do you like
this city?
Player Input: Yes.
NPC Response: That's good to know. What
was the last book you read?
Player Input: The Phantom of the Opera.
NPC Response: That's good to know. What
kind of music do you like?
Player Input: Rock music.
```

Figure 7.54 NPC repeating a question the player did not initially answer

7.3.6 Remember Answer to Question

The system should have the capability of storing answers that the player provides to the NPC's questions. Furthermore, if the player queries the system about an answer, the system should be able to retrieve it. In order to test how well this works, the following steps were done:

- When the NPC asks a question, answer it (e.g., "*What's your favourite movie?*" → "*Casablanca*")
- Ask NPC about the question. (e.g., "*What's my favourite movie again?*")
- See if NPC's answer is correct

Figure 7.55 demonstrates how the system can store and retrieve player responses.

```
NPC Response: That's good to know. What
kind of music do you like?
Player Input: Rock music.
NPC Response: That's good to know.
Player Input: What kind of music do I
like again?
NPC Response: Rock music.
```

Figure 7.55 NPC remembering the player's answer to a question

7.3.7 Remember Events Related to Topic

To add a storytelling element to the conversation, the NPC should be able to recall episodic memories. Moreover, these memories should be related to whatever is being discussed at the time. If no memory can be found that is related in some way to something previously mentioned, then the NPC should avoid mentioning a memory at all. In order to test this aspect of the system, the following steps were performed:

- Ask questions or make statements related to some theme (e.g., movies)
- When the NPC recalls an event, see how well it relates to what was being discussed

Figure 7.56 illustrates the NPC answering the player's question and then remembering an event related to movies. The sentence, "*I went to the movies with my friends*," is not from the knowledge base. Instead, it is an episodic memory that the NPC was recalling.

```
Player Input: Are you coming to the
movies?
NPC Response: I don't know. I went to the
movies with my friends.
```

Figure 7.56 NPC remembering an event related to the theme of "movies"

7.4 End of Conversation

The system should be able to recognize that the player wants to end the conversation without requiring the player to manually exit (e.g., by clicking the close button). Instead, it should be able to notice the use of end greetings and take that as a cue that the conversation is over. Furthermore, the system should respond with an end greeting, ensuring once again that the desired politeness level is used. In order to test if this works as it should, the following was done:

- Use different values for the different social factors
- Use words/sentences that contain end greetings
- Observe if the NPC responds with an end greeting and uses the correct level of politeness

Sections 7.4.1 and 7.4.2 demonstrate that the system is able to recognize several different cues that the conversation should end. In particular, it is even able to recognize more complex cues like those seen in Figures 7.58 and 7.62.

7.4.1 Required Politeness Level

For this implementation, "*See you*" and "*So long*" are considered to be informal, "*Bye*" is considered to be of medium formality, and "*Goodbye*" and "*Farewell*" are considered to be quite formal. Sections 7.4.1.1, 7.4.1.2, and 7.4.1.3 illustrate actual output when the required politeness is low, medium, and high respectively.

7.4.1.1 Low Politeness

```
Player Input: See you later.
NPC Response: See you.
```

Figure 7.57 NPC saying goodbye to the player with a low politeness level

```
Player Input: Anyways, I have to go. Bye
now!
NPC Response: So long.
```

Figure 7.58 NPC saying goodbye to the player with a low politeness level

7.4.1.2 Medium Politeness

```
Player Input: Farewell.
NPC Response: Bye.
```

Figure 7.59 NPC saying goodbye to the player with a medium politeness level

7.4.1.3 High Politeness

```
Player Input: Bye.
NPC Response: Goodbye.
```

Figure 7.60 NPC saying goodbye to the player with a high politeness level

```
Player Input: See you.
NPC Response: Farewell.
```

Figure 7.61 NPC saying goodbye to the player with a high politeness level

7.4.2 Situation

Section 7.4.2.1 illustrates the NPC bidding farewell to the player by using an end greeting that can only be used at night. At any other time of the day, this particular greeting would never be chosen.

7.4.2.1 Night-time

```
Player Input: It's late now. Goodnight.
NPC Response: Goodnight.
```

Figure 7.62 NPC saying goodbye to the player in a situation-appropriate way

7.5 Overall Conversation Flow

Although the individual modules may work well on their own, it is equally as important that they work well together. The conversation should flow smoothly, so there should not be any choppiness when it is transitioning from one stage to another. In particular, the transition that has the greatest chance of seeming overly abrupt is from the small talk stage to the core stage. Section 7.5.1 discusses in more detail what should be done to avoid this sudden switch and how to test that it is working properly.

7.5.1 Transition from Small Talk to Core

The system must be able to transition smoothly from the small talk stage to the core; it should not be obvious to the player that these two stages are separate from each other. The following are two ways of accomplishing a seamless transition, and both were tested to ensure they work properly:

- The player is not interested in finishing the small talk and asks the NPC a question immediately. In this case, the NPC should proceed to answer the player's question instead of ignoring it.

- The small talk module exhausts its topics, so it gracefully requests the player to assist in choosing a core topic (e.g., "*So is there anything I can help you with?*")

Section 7.5.1.1 demonstrates a conversation where the player opts to ask a question before the small talk naturally concludes. The system recognizes the player's desire for an answer and switches to the core stage. Then, it answers the question and attempts to continue the conversation. When the player informs the NPC that she must leave, the conversation ends.

Section 7.5.1.2, on the other hand, illustrates a conversation where the player does not interrupt the small talk. Here, the small talk continues until the system no longer has any topics remaining. In order to gracefully switch to the core stage, the system asks the player if it can help with anything. The dialogue continues for a few more exchanges before the player decides to end it.

Overall, both conversations worked well as a whole. However, one noticeable flaw was when the player asked, “Do you know where Jane is?” in Figure 7.64. While the system responded correctly that it did know, it also attempted to change the topic before answering the question. The first reason that this event occurred is because of the lack of a topic handler to signal when it is appropriate to change topics. The second reason is that the system classified the query as a yes/no question. While at first glance this classification would appear to be correct, the player’s query was implicitly a request for information; that is, it can be understood to mean, “Where is Jane?” Consequently, this example shows a limitation of the system due to it not being fully implemented.

7.5.1.1 Player Interrupts Small Talk

```
Player Input: Hello.
NPC Response: Good afternoon. How's life?
Player Input: Fine, thanks.
NPC Response: I'm glad. That's a nice
dagger you have there.
Player Input: Anyways, where is Jane?
NPC Response: Jane is in Paris. What kind
of music do you like?
Player Input: I actually have to go now.
Bye.
NPC Response: Bye.
```

Figure 7.63 Conversation where player interrupts small talk to ask a question

7.5.1.2 Small Talk is Exhausted

```
Player Input: Hi.
NPC Response: Hi. How's it going?
Player Input: Pretty good.
NPC Response: That's good to hear! Did
you get a haircut? Your hair looks really
nice!
Player Input: Thank you.
NPC Response: You're welcome! Can you
believe all this rain?
Player Input: It's been raining cats and
dogs.
NPC Response: So is there anything that I
can help you with?
Player Input: Do you know where Jane is?
NPC Response: Yes.  What kind of music do
you like?
Player Input: Well, where is Jane?
NPC Response: Jane is in Paris. Do you
have plans for the weekend?
Player Input: Why do you care?
NPC Response: I care because I'm curious.
Player Input: Why are you curious?
NPC Response: I don't know why I am
curious. I just am.
Player Input: Oh, OK. Anyways, see you.
NPC Response: Bye.
```

Figure 7.64 Conversation where small talk naturally ends and gracefully transitions to the core

7.6 Accuracy

Based on the above tests, the conversation system gave appropriate output approximately 96% of the time. Given that the system has been iteratively improved until it gave reasonable results for all the test cases discussed, this result is not surprising. However, this result may potentially be overly optimistic due to having in-depth knowledge of how the system works while testing it. Consequently, player testing is still required to test the system in a more formal and rigorous fashion. Furthermore, only a limited amount of facts, topics, and episodic memories were used when testing, so it is unclear how the accuracy would be affected when using a larger amount. It is highly likely that as the number of facts, topics, and episodic memories increase, more robust NLP will be required to maintain this level of accuracy.

7.7 Performance in Minecraft

Since the conversation system is designed to be used within a video game, it must not negatively affect performance too much. Otherwise, the game would become unplayable due to lag, which would take away from the immersive experience. Consequently, the system was tested within Minecraft to see how well it performs from a computational standpoint. The following aspects were examined:

- Does the game crash due to lack of memory?
- Is the computer running noticeably more slowly?
- Is the game lagging?

The system was tested on a laptop with specifications higher than the minimum required to run Minecraft. The laptop's specifications are as follows:

- Operating System: Windows 8.1 Pro, 64-bit
- Processor: Intel Core i7-3537U with clock speed of 2.49 GHz
- Memory: 6 GB (5.89 GB usable)
- GPU: NVidia GeForce GT 740M

In order to gauge how well the conversation system worked within Minecraft, the regular version and the modded version were played three times each. To ensure consistency between the two versions, the same seed was used to generate a level with a village nearby. The modded version of the game took several seconds longer to initially load compared to the regular version. There are various possible reasons why the initial loading time increased, such as having to process and attach episodic memories to each of the villagers, but these factors could be optimized in the future for better performance. During actual gameplay, however, the player character could walk around the village with a similar-looking frame rate in both versions of the game, so there was no noticeable lag when real-time performance mattered most.

During an actual conversation with the system, some lag was noticed whenever the NPC attempted to recall an episodic memory. It is not clear why this occurred because neither

CPU usage nor memory usage appeared to change significantly. However, the rest of the conversation progressed at a reasonable speed.

From a computational standpoint, the conversation mod did not appear to have a significant impact on resources because the computer could still run with no noticeable slowdown or visible change in frame rate. As can be seen in Table 7.1, on average, regular Minecraft used 33% of the CPU and 575 MB of memory. The modded version used only 7% more CPU and 25 MB more memory. This increase in CPU usage occurred mainly when the player was engaging in dialogue with the NPC; otherwise, it was fairly similar to the regular version's CPU usage. Thus, the majority of the computational resources that were used are required for the Minecraft game itself to run and not for the conversation mod. Nevertheless, opportunities for optimizations of the implementation will be explored.

	CPU Usage	**Memory Usage**
Regular Minecraft	33%	575 MB
Modded Minecraft	40%	600 MB

Table 7.1 Comparing the typical use of computational resources of regular Minecraft and the modded version

7.8 Discussion of Testing Techniques

Because the nature of this thesis' research is still fairly new, there is no standard set of tests from the literature that could be used. Instead, the tests mentioned above were chosen because they test the required functionality of the conversation system. While all attempts were made to design a comprehensive set of tests, it is certainly possible that other tests could be required in the future. Since the English language allows for virtually an infinite number of possible utterances, it is nearly impossible to be exhaustive in testing. Thus, it is only ever possible to test a sample of possible utterances. Ideally, there

should be extensive player testing to determine how well the system performs in a real-world setting; however, that is outside the scope of this thesis and is left for future work.

7.9 Summary

This chapter discussed the methods that were used to test the partial implementation of the conversation system mentioned in Chapter 6. Each component was initially tested separately. The greeting handler was tested for how well it chooses a greeting depending on the required politeness level and situational factors such as time of day or the player's gender. The small talk handler was tested to ensure that it does not repeat topics and that it gives appropriate responses to player replies. The core was tested for how well it recognizes when the player asks a question or makes a statement, answers questions appropriately, understands contractions, responds suitably to statements the player may make, asks the player questions, remembers the answer the player gave to a question, and recalls events related to the topic being discussed. At the end of the conversation, the system was tested to see if it recognizes that the player would like to conclude the dialogue, and if it responds with an end greeting that uses the desired politeness level. Once each component of the system had been thoroughly tested, the overall flow of the conversation was scrutinized for any kind of sudden shifts. Finally, in order to test how well the system runs in an actual video game, its computational performance was examined within the context of Minecraft.

Overall, the system performed as expected. It was accurate approximately 96% of the time. However, player testing is still required for more formal and rigorous testing. In terms of computational performance, the system had a small resource footprint compared to the original Minecraft game.

Chapter 8

8 Discussion and Conclusion

This chapter discusses the results shown in Chapter 7, as well as concluding the entire thesis. It also comments on features that could be added in the future to make the conversation system even more powerful.

8.1 Discussion

Chapter 7 demonstrated the results of testing the conversation system. Despite being a partial implementation, the system performed quite well at various tasks. The individual modules appeared to work well both on their own and together.

The different modules were tested by seeing how well they performed during their particular stage of the conversation. During the greeting stage, the system was able to adjust its greeting based on the required level of politeness and situational factors such as time of day and gender. During the small talk stage, it was able to successfully engage in simple small talk. Although the small-talk module is limited due to the lack of a topic handler, it was able to avoid repeating itself and could acknowledge certain types of player responses such as thanks and indications of how the player is feeling. During the core stage, the system was able to perform numerous tasks reasonably well. It was able to differentiate between questions and statements, handle contractions, answer different kinds of questions, respond to the player's sentiment, ask questions without repeating itself unnecessarily, and remember events. However, due to the lack of a full sentiment-analysis tool, the system needed to guess when the player was complimenting or insulting it, which would yield false positives in certain circumstances. During the end greeting stage, the system could detect when the player wanted to end the conversation, and it would output an end greeting with the correct level of politeness. In general, the testing of the distinct stages of the conversation showed promising results.

Overall, the system did a very good job at linking the different stages together in a seamless way. The transition between the small talk stage and the core stage was most at risk for seeming sudden, but in the end, it actually flowed quite well. There were points

during the core of the conversation, however, where the system would attempt to change topics too soon. This phenomenon could not be avoided in the present implementation because there is no topic handler to decide an appropriate time to change topics. Thus, it is currently random when the system will ask a question. Nevertheless, the conversation system still performed surprisingly well given its limitations and was able to engage in a simple dialogue with the player.

In order to ensure that dialogues sound less artificial in the future, more computational linguistics will be required. However, as mentioned in Chapter 6, many necessary features were unimplemented because of the lack of libraries that could perform them in real-time. In a video game, it is of utmost importance that performance is in real-time; otherwise, it breaks the suspension of disbelief. Until either faster NLP algorithms are developed or computers become fast enough to run the current ones in real-time, it will be difficult to incorporate more NLP into the system without sacrificing speed.

In terms of computational performance, the system runs fairly well and has a relatively low resource footprint. It successfully ran within Minecraft without causing any crashes or noticeable computer slowdown. The only time any lag was noticed was when the NPC tried to recall episodic memories, so that particular module will likely need to be better optimized for real-time performance.

8.2 Contributions

This thesis has made the following contributions:

- It proposes a method of designing a conversation system for video games capable of more realistic, unscripted dialogue
- It provides a proof-of-concept implementation demonstrating how such a system could work
- It illustrates how to use current research results in Artificial Intelligence in a novel application

- It integrates this conversation system into a commercial video game (Minecraft) and conducts experiments to both test fitness to task and to demonstrate real-time performance

8.3 Conclusion

The current state-of-the-art techniques used to simulate dialogue in video games are quite outdated. After a while, non-player characters cease to appear autonomous, which can break the immersive experience. This thesis has demonstrated an approach to allow for dynamic dialogue that can still fit within a game's world. By using modern techniques in Artificial Intelligence, a conversation system was developed that can allow a human player to interact with non-player characters using natural-language input. Although the current capabilities of the system are limited, they are enough to allow NPCs to engage in simple dialogue: NPCs are able to greet the player, engage in small talk, answer questions, ask questions, remember what the player said, and remember events related to the topic being discussed. Furthermore, the conversation system is a proof-of-concept for a more advanced way of interacting with NPCs. Instead of being forced to rely on scripted techniques, this thesis shows that it is possible for human players to converse with an NPC in a similar manner as they would with another human being. Consequently, NPCs are able to display greater autonomy and better maintain the illusion of intelligence.

8.4 Future Work

As mentioned in Chapter 6, the conversation system is only a partial implementation of the architecture described in Chapter 5. It currently lacks the following features: ellipsis resolution, coreference resolution, named-entity recognition and classification, an ontology, relationship extraction, full sentiment analysis, the logic handler, and the topic handler. As a consequence, it is highly limited in its capabilities and is especially poor at deducing facts and introducing topics at appropriate times. Most of these missing features were excluded from the current implementation due to a lack of time. As part of future work, these features should be added to make the conversation system more robust. Moreover, this thesis did not test for scalability; consequently, it is unclear how well it

would perform with a large amount of memories, a bigger knowledge base, or more NPCs. Likewise, it is unclear how well the system would perform once the missing features have been added. The conversation system should also be added to other games than just Minecraft to further test performance impact and ensure there is no lagging during gameplay. Furthermore, there should be player testing since the end goal is for actual players to be able to use the system to converse with NPCs. Finally, this thesis did not examine how individual factors such as personality can affect conversations. In its current form, the conversation system will attempt to truthfully answer any question the player may ask. However, it would be an interesting addition if the system could either refuse to answer or lie about an answer if it is in the NPC's nature not to trust the human player. It would also be interesting for the wording of an NPC's response to vary based on factors such as its current emotions (e.g., certain responses could be tagged with the emotion they convey and would only be uttered if the NPC was feeling that emotion), how it feels about the player, and its education level. One possible way to add a personality dynamic would be to use the approach discussed in Ochs, Sabouret, and Corruble [15].

References

[1] Entertainment Software Association of Canada, "Essential Facts About the Canadian Video Game Industry 2013.," Entertainment Software Association of Canada Research Report, 2013.

[2] Entertainment Software Association, "Essential Facts About the Computer and Video Game Industry," Entertainment Software Association Research Report, 2014 .

[3] E. Adams, Fundamentals of Game Design, New Riders Publishing, 2013, pp. 15, 237.

[4] C. Bailey, J. You, G. Acton, A. Rankin and M. Katchabaw, "Believability Through Psychosocial Behaviour: Creating Bots That Are More Engaging And Entertaining," in *Believable Bots: Can Computers Play Like People?*, Springer, 2012, pp. 29-64.

[5] B. Reynolds, "How AI Enables Designers," Gamasutra, 6 November 2006. [Online]. Available: http://gamasutra.com/php-bin/news_index.php?story=11577.

[6] I. Millington and J. Funge, Artificial Intelligence for Games, Elsevier, 2009.

[7] J. Forgette, *Reinforcement Learning with Motivations for Realistic Agents*, Master Thesis, University of Western Ontario, 2013.

[8] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2010.

[9] K. Wheeler, *Representing Game Dialogue as Expressions in First-Order Logic*, Master Thesis, University of Western Ontario, 2013.

[10] A. Holmgren, "Saying, doing and making: teaching CMM theory," *Human Systems: The Journal of Systemic Consultation & Management,* vol. 15, no. 2, pp. 89-100, 2004.

[11] P. Cheng, "Waiting for Something to Happen: Narratives, Interactivity and Agency and the Video Game Cut-scene," in *Digital Games Research Association (DiGRA)*, Riverside, CA, 2007.

[12] Gamasutra, "Death to the Non-Interactive Cutscene, Say Top Game Writers," 12 March 2012. [Online]. Available: http://www.gamasutra.com/view/news/165491/Death_to_the_noninteractive_cutscene_say_top_game_writers.php.

[13] S. Rogers, Level Up! The Guide to Great Video Game Design, Wiley, 2014.

[14] E. Knudsen, "An Attempt to Dissect Quick Time Events," Gamasutra, 24 May 2014. [Online]. Available: http://www.gamasutra.com/blogs/EskeKnudsen/20140524/218458/An_attempt_to_dissect_quick_time_events.php.

[15] M. Ochs, N. Sabouret and V. Corruble, "Modeling the Dynamics of Non-Player Characters' Social Relations in Video Games," in *Association for the Advancement of Artificial Intelligence*, Chicago, 2008.

[16] Wikimedia Commons, 16 November 2010. [Online]. Available: http://en.wikipedia.org/wiki/File:Dialog_tree_example.svg. [Accessed 18 November 2013].

[17] C. Rich and C. L. Sidner, "Using Collaborative Discourse Theory to Partially Automate Dialogue Tree Authoring," in *International Conference on Intelligent Virtual Agents*, Santa Cruz, 2012.

[18] D. Kaplan, *Can Novamente's virtual dogs make a Nintendog look dumb?,* Venture Beat, 2008.

[19] M. Mateas and A. Stern, "Writing façade: A case study in procedural authorship," in *Second Person: Role-Playing and Story in Games and Playable Media*, Cambridge, MA, MIT Press, 2007, pp. 183-208.

[20] E. Joseph, "Bot Colony – a Video Game Featuring Intelligent Language-Based Interaction with the Characters," in *8th International Conference on Natural Language Processing*, Kanazawa, Japan, 2012.

[21] A. Freed, "Branching Conversation Systems and the Working Writer, Part 1: Introduction," Gamasutra, 2 September 2014. [Online]. Available: http://www.gamasutra.com/blogs/AlexanderFreed/20140902/224609/Branching_Conversation_Systems_and_the_Working_Writer_Part_1_Introduction.php.

[22] S. Sarawagi, "Information Extraction," *Foundations and Trends in Databases,* vol. 1, no. 3, pp. 261-377, 2008.

[23] O. Etzioni, M. Banko, S. Soderland and D. S. Weld, "Open Information Extraction from the Web," *Communications of the ACM,* vol. 51, no. 12, pp. 68-74, December 2008.

[24] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," in *Named Entities: Recognition, classification and use*, John Benjamins Publishing Co, 2009, pp. 3-27.

[25] M. Poesio, S. Paolo Ponzetto and Y. Versley, "Computational Models of Anaphora Resolution: A Survey," 2010.

[26] N. Bach and S. Badaskar, "A Review of Relation Extraction," *Literature review for Language and Statistics II*, 2007.

[27] B. Liu and L. Zhang, "A Survey of Opinion Mining and Sentiment Analysis," in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds., New York, Springer, 2012, pp. 415-463.

[28] S. Quarteroni and S. Manandhar, "Designing an Interactive Open-Domain Question," *Natural Language Engineering*, 2008.

[29] W. Li, "The Functions and Use of Greetings," *Canadian Social Science*, vol. 6, no. 4, 2010.

[30] P. Brown and S. Levinson, Politeness: Some Universals in Language Usage, Cambridge University Press, 1987.

[31] T. Bickmore and J. Cassell, "Small Talk and Conversational Storytelling In Embodied Conversational Interface Agents," in *AAAI*, 1999.

[32] A. Kope, C. Rose and M. Katchabaw, "Modeling Autobiographical Memory for Believable Agents," in *AIIDE*, Boston, 2013.

[33] T. Gruber, "What is an Ontology?," Stanford University, 2001. [Online]. Available: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.

[34] Apache, "Apache Lucene," [Online]. Available: http://lucene.apache.org/.

[35] S. Loria, "TextBlob," [Online]. Available: http://textblob.readthedocs.org/en/dev/.

[36] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014.

[37] W3C, "OWL Web Ontology Language," [Online]. Available: http://www.w3.org/TR/owl-ref/.

[38] Apache, "Apache Jena," [Online]. Available: https://jena.apache.org/.

[39] Mojang, "Minecraft," [Online]. Available: https://minecraft.net/ .

Curriculum Vitae

Name: Caroline Rose

Post-secondary Education and Degrees:

The University of Western Ontario
London, Ontario, Canada
2013-2014 M.Sc. in Computer Science candidate
Expected graduation: June 2015
Supervisor: Michael Katchabaw

The University of Western Ontario
London, Ontario, Canada
2009-2013 Honors B.Sc. in Computer Science
Minor in Game Development

Honours and Awards:

Natural Sciences and Engineering Research Council (NSERC)
CGS M
2013-2014

The University of Western Ontario Gold Medal
2013

Natural Sciences and Engineering Research Council (NSERC)
USRA
2012-2013

UWO In-Course Scholarships
2011-2012

Edna Jeffery Scholarship
2011

Dean's Honor List
2010-2013

Related Work Experience

Teaching Assistant
The University of Western Ontario
2013-2014

Publications:

A. Kope, C. Rose and M. Katchabaw, "Modeling Autobiographical Memory for Believable Agents," in *AIIDE*, 2013.