Designing Open Dialogue Systems for LLM-Based NPCs

Kieran McKee
DigiPen Institute of Technology
9931 Willows Rd NE
Redmond, WA
kieranmckee7@gmail.com

Joshua D. Savage, Vanessa Hemovich PhD.
DigiPen Institute of Technology
9931 Willows Rd NE
Redmond, WA
joshua.savage@digipen.edu, vhemovich@gmail.com

ABSTRACT

This paper aims to propose a new way of hosting diegetic conversations between a player character and a non-player character (NPC) in the context of interactive entertainment (video games). Instead of a preset list of dialogue options for the user to choose from, the user is allowed to create their own dialogue for the player character, and a large language model (LLM) is used to create an NPC's response. By formatting this system correctly and managing both the unpredictable input of the player and the non-deterministic output of the LLM, an NPC can act in character according to a pre-written script and can dynamically change behavior during the interaction. This paper also goes into depth about using the low-level logical reasoning capabilities of LLMs to change game states, and the potential ways to structure, store and use semantic qualitative data using LLMs.

Keywords

Natural language processing, artificial intelligence, narrative systems, algorithms, communication, outbound sales game

INTRODUCTION

The creation of natural-feeling dialogues between players and non-player characters (NPCs) in games is an undertaking that has long been hampered by the limitations of computer technology (Crawford, 2003). In many cases, these limitations have driven games to follow dialogue conventions that have more similarity to cinema and television than to plays or prose (Smith, 2002). In order to map the interactive element of games onto dialogue in a manageable and scalable manner, designers such as Schell (2008) advocate for the imposition of constraints on the options available to the player. This paper presents a method for using large language models (LLMs) to create broader, more dynamic dialogue possibilities for the players while still allowing

Proceedings of DiGRA 2026

designers to shape the experience through the imposition of constraints not only on the player agent, but also on the computer-controlled agent that is utilizing the LLM.

A dialogue system, in the context of interactive simulation (video games), is how a player communicates with the game system through NPCs (non-player characters) within the game space. Typically, this system of interaction includes the following set of steps:

1: The player is given a list of pre-written dialogue options to choose from, each representing a line of dialogue to be said by the PC (player character) to the NPC.

2: Once a dialogue option is chosen, the NPC responds to the dialogue by replying to the PC, using a pre-written response to the chosen option.

3: Either the player is given more dialogue options to choose from to continue the conversation, or the conversation ends.

This represents a common way to format a conversation between the player and the NPC in a narrative setting and allows for the designer to control every aspect of the interaction, including all possible options for the player's input as well as the NPC's pre-written responses to all inputs. Dialogue systems like this have been present in games for decades, with the most recognizable examples being from narrative-heavy game series like *Fallout* (Bethesda, 1997), *Mass Effect* (BioWare, 2007), and *Disco Elysium* (Kurvitz et al. 2024).

The gameplay purpose of allowing the player to interact in a controlled conversation with an NPC is to feed information to the player in a way that lets them feel present within the game space (Veloso et al 2021). The information given through interactive narrative also has a non-diegetic purpose, which can be to give players direction, a goal, or to assist them with navigating the task the game has given them. By having pre-written NPC reactions to the player's choice of dialogue, it creates the illusion of the player character building a relationship with an NPC, while also subtly influencing the player's ability to interact with the game system.

This paper aims to propose a new way of hosting these player-NPC conversations, to be used to the same effect as traditional dialogue systems, but instead allows players to input their own dialogue instead of being presented with a list of pre-set options. This method is called an open dialogue system, that uses a large language model (LLM) as an agent in an agent-based model to process the text input of the player to create and display a response from the NPC. It will become very apparent throughout this paper that the application of this open dialogue method--and the behavior of the language model agent--heavily relies on human design.

HISTORY OF TEXT PARSING IN GAMES

The challenge of providing the player with meaningful agency is one that has long persisted in games. Dialogue in games was initially command-based, driven by text parsers limiting the options that players had for communicating with characters in games and, by extension, the designers who created them.

Dialogue in games was limited by the constraints of technology at the time. LLMs were impractical in the 1970s and 1980s, a dream relegated to the realm of science fiction. Early computer games such as Will Crowther and Don Woods's *Colossal Cave*

Adventure (1975) and *Zork* (Infocom, 1977) relied entirely on text, both for communicating the game world to the player and for receiving information from the player into the system. These games create the impression of freedom, allowing the player to input any response, but this freedom is an illusion, as the program only accepts commands recognized by the text parser (Barton et al, 2019). The complexity of text parsers were, in turn, limited not only by the skills of the programmer and the processing power of machines, but also by the storage capacity of digital media, which, even for titles distributed on single IBM-compatible 5.25-inch or 3.5-inch floppy disks, was less than 1.5 Megabytes (Fuhrig et al, 2016).

As games developed, the constraints of text parsing drove innovations to streamline interaction with games: games began to use keyboard elements such as arrow keys for movement, and mapped common commands to keys—with extreme cases, like Richard Garriott's *Ultima* (1981), mapping a command to every single letter of the keyboard, using each letter as a mnemonic that could reasonably represent that command, such as [G] for "Get", though some, such as [K] for "Klimb", did not map precisely to standard English (Adams et al., 1989). Graphical adventure games, meanwhile, retained simple text parsing for command interactions in a number of titles by Sierra On-Line such as *King's Quest* (Williams, 1984) and *Space Quest* (Crowe et al, 1986), while the computer Role-Playing Game series *Ultima* continued to use text parsing for dialogue conversations through *Ultima VI: The False Prophet* (Origin Systems, 1990).

Allowing the player more freedom in carrying on conversations with in-game characters also requires plotting out every allowable conversation in-game. This quickly adds to the amount of text that must be created, as evidenced by the leap in dialogue size between Origin Systems' *Ultima V: Warriors of Destiny* (1988) *and Ultima VI: The False Prophet*, where the average size of NPC dialogue in *Ultima VI* was six times that of the largest conversation in *Ultima V* (and the largest conversation in *Ultima VI* being about 30 times the largest in *Ultima V*). (Addams, 1990)

Commands in these early games are often not complete sentences, but rather words or phrases, such as "GET [noun]" or "NORTH". Those with experience using computers at the time may have understood the need for this truncated syntax, as it shares similarities with early programming languages such as BASIC or operating systems such as MS-DOS. But while text parsing was common in early computer games, early game consoles, with their limited input devices, could not easily allow for free-text entry. The creators of *Dragon Quest* (Enix, 1986) realized that, while the idiosyncratic syntax of early text-parsers made sense to computer users, it would not translate as well to the audience of console owners, and so implemented menu-based commands and dialogue (Kohler, 2005). In these systems, commands are not freely inputted by the player, but rather are selected from a predetermined list. While this avoids requiring the player to remember (or guess at) acceptable inputs, it also can create the perception of constraining player freedom by limiting possible interactions. It is important to note, however, that this limitation may in many cases be merely a perception rather than reality, as an open-text parser may only accept that same limited number of inputs as valid.

This practice of limiting user input to a selection from a list of commands was adopted by computer graphic adventure games beginning with *Maniac Mansion* (LucasArts, 1987) and computer role-playing games such as *Ultima VII: The Black Gate* (Origin Systems, 1992), though some games, such as Bob Bates's *Eric the Unready* (1992),

experimented with a hybrid system of presenting both a list of relevant commands and a free-text parser, with creative use of the latter occasionally being rewarded with hidden scenes or dialogue. Natural-language processing in software remained generally constrained to experimental software, such as Creative Labs' *Dr. Sbaitso* (1991), an artificial-intelligence 'psychologist' program comprised entirely of a single open-ended conversation with the titular character, who would parse the user's text inputs and respond with text that was then read aloud using voice synthesis as a means of showcasing Creative Labs' *SOUND BLASTER* sound card technology (Creative Labs, 1990). The limiting of dialogue choices to a curated set of predetermined options, however, became the norm, a practice that avoided immersion-breaking moments where inputted text did not match what the parser recognized, streamlined the user experience, and created a narrative convention that proved scalable as game dialogue increased in size and was, increasingly, accompanied by pre-recorded audio.

The evolution of conventions in video- and computer-game dialogue, then, can be reasonably considered to be a result of concerns of usability, scalability, and constraints. The potential application of LLMs to game dialogue, however, offers the potential for realizing the original promise of the open-input text parsing systems of early games like *Colossal Cave Adventure* and *Zork*. With constraints deliberately set on the LLM, a multitude of emergent play patterns could develop within the framework set by the designer, without placing immersion-breaking constraints upon the player's expression or interaction with the system.

REASONING CAPABILITIES OF LLMS

The emergent capabilities of large language models being studied have proven that LLMs are capable of step-by step reasoning (Zhao et al, 2023) and causal reasoning with reliable accuracy (Li et al, 2023) By automating the process of posing simple reasoning tasks to an LLM and parsing the generated response, a system can be created that changes depending on non-quantifiable values, such as meaning, logic, context, or other semantic information.

For example, by prompting an LLM like *ChatGPT* (OpenAI, 2022) to analyze a written argument between two human agents, it could reliably produce an output text that describes which player won the argument. This is remarkable, as the conclusion reached by the LLM is not dependent on victory points, or other numerical values that would traditionally be used to determine the win condition of a game system. It begs the question: how can this new type of qualitative data analysis be applied in video games, and what new systems and game design principles emerge?

It should be noted that LLMs do not always produce perfect results. The possibility of an LLM producing incorrect output is what has largely prevented its use in important societal tasks (health care, education, law) but the stakes are much lower for its use in interactive entertainment (Godfrey, 2021) For the purposes of video game application, the models and methods defined by this paper are functional given a large language model's ability to produce reliable, coherent written answers to pre-written questions, and its ability to accurately complete reasoning tasks. The example functions provided in this paper use abbreviated examples of actual prompts, excluding the instructions to the LLM on how to format the output. All example functions return a "null" output if the generated response from the LLM is incoherent, or it is unable to complete the reasoning task.

DIALOGUE LOOP MODEL

The foundation of an open dialogue system is dependent on the following basic dialogue model (Figure 1), which attempts to describe the states of interaction between a human agent (player) and a large language model agent such as *ChatGPT.* The player is given an interface with which to interact with the LLM agent, such as a text box. The player inputs a line of dialogue which is added to a list of system messages referred to as a context window. The LLM agent then generates a response to the player's input, based on the context window containing all messages previously added to the conversation. The player receives this generated feedback and responds by giving another input into the system.

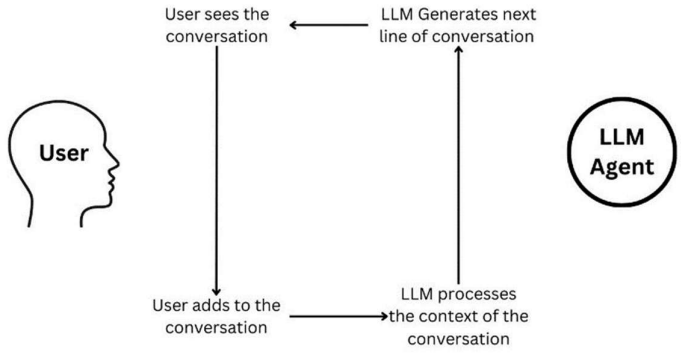

Figure 1: A simplified model of the interaction between a user and an LLM agent, shown as a loop.

In this exchange of information, the LLM agent's output is dependent on the contents of the context window, and LLMs are non-deterministic by nature (Ouyang et al, 2023). If the player is allowed to input any series of characters or words into the context window (which is the defining purpose of an open dialogue system) for the LLM agent to accept, the generated response is as unpredictable as the player's input. Therefore, both the player's input into the system and the LLM agent's output are non-deterministic.

> *A. In an open dialogue loop system, the contents of both the player's input and the LLM agent's input to the system are non-deterministic.*

However, this does not mean that the LLM's response is uncontrollable. While the addition of non-deterministic messages into the context window means that the generated response is unpredictable, there can be additional messages added to the context window that are deterministic and control the behavior of the LLM agent, such as prewritten explicit instructions, directions, or rules.

For example, say a player is allowed to input a message into the context window, in their own words. This message is unpredictable to the designer. But before the agent is called upon to generate a response, the system also adds a pre-written message to the context window with the contents: "Speak like a pirate." The generated output of the LLM is still non-deterministic, but every response to the non-deterministic input will be generated according to the pre-written message. For every player that

interacts with this system, the exact words they receive as feedback will be non-deterministic but always be in the form of a pirate's reply to their query.

B. An LLM agent's input to the system is non-deterministic but predictable when given a deterministic set of instructions, referred to in this paper as a directing prompt.

A directing prompt can either be in the form of a simple prompt or in-context prompt that governs the LLM's output (Lee et al, 2022). A simple directing prompt (or establishing prompt) could be added to the context window before the conversation begins in order to establish what the conversation is about, between whom, and any requests for formatting generated responses. Any directing prompts added afterwards throughout the conversation would be in the context established by this first prompt. Although the player will observe the effects of the directing prompts on the generated output of the LLM, it is not necessary to make the contents of directing prompts observable to the player. It is advisable for user experience purposes to only display the messages in the context window that pertain to the player's interaction, such as the messages sent by the player character or the NPC. To do so means that there are two different versions of the recorded dialogue: what is visible to the player and what is visible to the LLM agent.

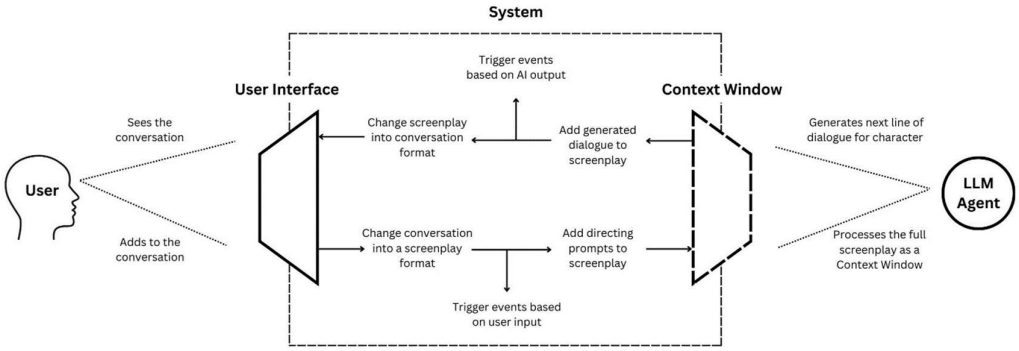

Figure 2: An agent-based dialogue loop model of the interaction between a user and an LLM agent, with a system in place to add directing prompts to the context window to influence the conversation.

By having a list of pre-written prompts to be injected into the context window at specific turns (Figure 2), it is possible to dynamically change how the NPC acts from the perspective of the player. This allows the LLM agent to process new information that was not given by the player through dialogue, and new information that it did not have access to earlier in the conversation. In addition, directing prompts could have an effect outside of the interaction, such as triggering in-game events or sequences. Compared to dialogue systems that rely solely on a standalone LLM to interact with the player, this system is more conducive to design, more controllable by the designer, and more accessible to designers who are not familiar with machine learning. This system is also able to have an observable record of all the context being used to

generate responses via the context window, as well as all the information displayed to the player via the conversation to be used for playtesting data.

CONVERSATION SCRIPTING

A way to format this system would be to have the contents of the context window be formatted as a screenplay describing a scene with dialogue between two characters: the player character and the NPC. In this scenario, only the dialogue portions of the screenplay would be visible to the player. When the player inputs a message into the conversation, the message is formatted and added to the screenplay as a line of dialogue spoken by the player character. Directing prompts are then added to the screenplay as action lines or descriptions, to give context to the LLM agent who then generates the NPC's lines for the continuation of the screenplay. This allows the LLM agent to have a top-down perspective of the entirety of the interaction as described by the designer, and for directing prompts to dynamically change how the NPC acts within the scene. Upon iteration of the screenplay, the dynamically generated lines of dialogue for the NPC are then formatted to display to the player. The player receives this as feedback, adds another line of dialogue as a response, and the loop continues.

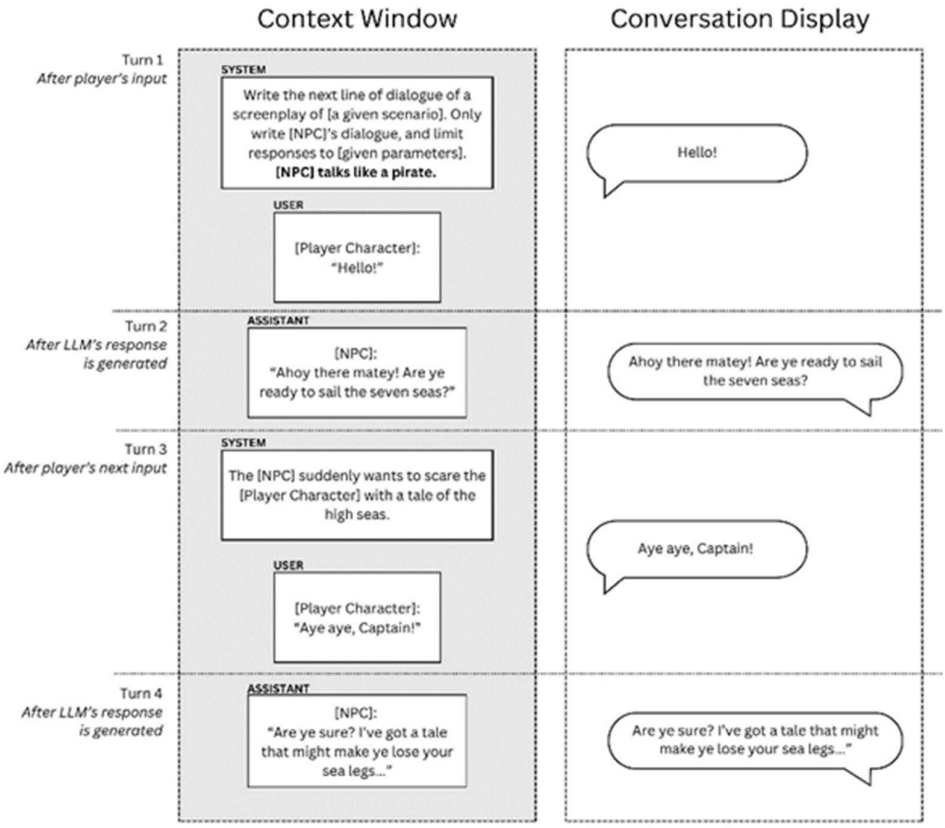

Figure 3: An example of the dynamic contents of a context window (visible to LLM agent) and how the content is displayed to the player.

PROGRAMMING WITH QUALITATIVE DATA

Language models can also be used to analyze the interaction between the player and the NPC in new and interesting ways, other than providing dialogue. For example, a large language model can be given the context window of a conversation in conjunction with a prompt that instructs it to make judgements about what has conspired during the conversation. The generated output can be stored as a variable to change states within the larger game system.

The ultimate result of this analysis function is a generated string of semantic information and logical reasoning provided by the LLM, and since the results are influenced by human factors, they can be treated as analogous to qualitative data types found in human survey design, such as ordinal, nominal or dichotomous data. Using the context window and a sufficient prompt, the predictable semantic output generated by the LLM (in the form of a string of characters) can then be parsed into quantitative data types--such as integers, substrings and booleans--to change game states. The following section provides example qualitative data types that can be used in programming these game systems.

Ordinal Data

In the context of qualitative programming, an ordinal is a qualitative data type that describes the subjective degree of an aspect of the content of the context window, on a numerical Likert scale.

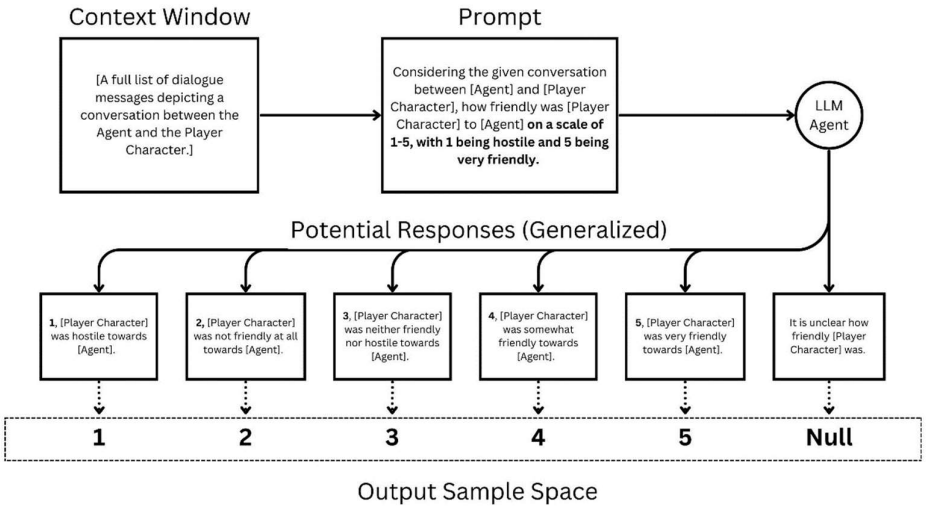

Figure 4: An example diagram of how a program can use an LLM agent's reasoning capability to analyze and store **ordinal** data.

When a question is prompted to the language model that asks it to rate on a Likert scale an element of a context window, the output is described as ordinal data. The string that is generated can describe the degree to which the context window information relates to the scale provided. For example, by providing a prompt that

asks the LLM agent to rate the attitude of a character towards another on a scale from 1-5, the resulting ordinal data can be parsed into an integer and stored within the system (Figure 4). If the generated response is not able to be parsed, the results of the analysis can be described as null.

Dichotomous Data

In the context of qualitative programming, a dichotomy is a qualitative data type that can store the result of a question posed to the language model agent that only has two valid predictable outcomes: true and false.

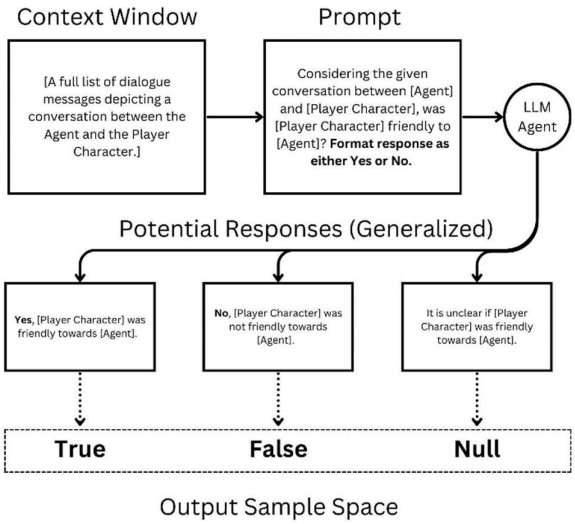

Figure 5: An example diagram of how a program can use an LLM agent's reasoning capability to analyze and store **dichotomous** data.

When a question is prompted to the LLM agent that asks it to choose between two states based on the context window, its output is predictably dichotomous. The string of words that is generated can determine if part of the context is true or false and can narrow the non-deterministic information provided in the context window into a boolean statement. This output can then be parsed into a literal boolean data type and stored within the system (Figure. 5). If the output cannot be parsed, then the data can be described as null.

Nominal Data

In the context of qualitative programming, a nominal is non-deterministic qualitative data type and can store the meaning, intent, purpose or other non-quantifiable nuanced information that is present within the context window and can be described by a list of characters (a string). Nominals are used to identify and store semantic information, and can be re-evaluated later to change game states.

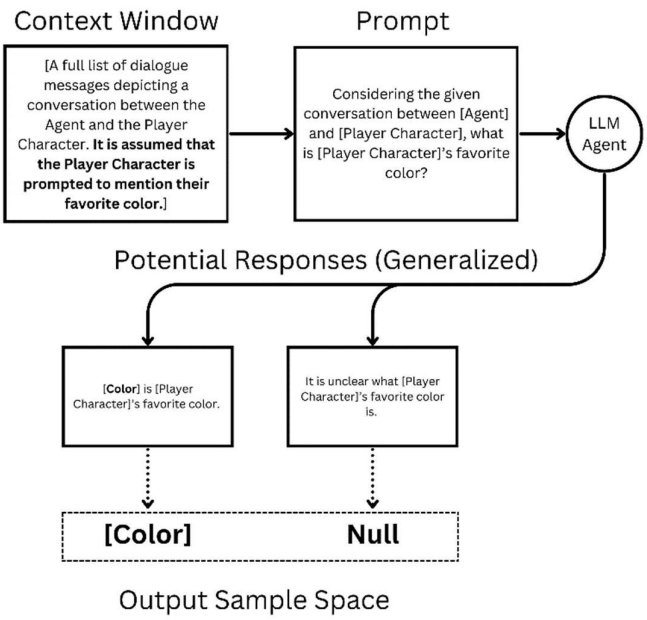

Figure 6: An example diagram of how a program can use an LLM agent's reasoning capability to analyze and store **nominal** data.

Any data that can be described by the player or language model that cannot be quantified by the system is described as nominal data. Nominal data could be any specific topic, category, or set of words with a meaning or purpose behind them. For example, if a language model was prompted to identify the player character's favorite color based on a context window predicted to contain a conversation discussing that topic, the output generated would contain that color as a literal string data type. Although the data is stored in a string, more information is stored within the data than just the list of characters that a string would typically describe. This data describes an attribute of the analyzed context that otherwise cannot be quantitatively measured and can only be inferred, in this instance the player character's favorite color. The meaning and purpose behind the string of characters is what sets nominal data apart from typical string literals. However, nominal data that is stored in string literals can still be used to change game states. Using the previous example, the player's favorite color nominal could be analyzed by another LLM agent to create and store an NPC's diegetic opinion about the player character. This opinion could be added as a directing prompt in a future interaction and affect how the NPC interacts with the player. If the output generated to assess nominal data cannot be parsed or is incoherent, then the data can be described as null.

Potential of Qualitative Programming

Although the previous examples of qualitative analysis were shown to describe characteristics of the player, a language model can be used to analyze any qualitative data in the context window, whether that data is dynamically generated or not. It could potentially analyze the actions and the generated dialogue of the NPC to choose deterministic game actions that the NPC could take. It could potentially analyze the

effects characters have on the simulated environment to be used to change the space that the interaction takes place in. These could be reversed as well, so that previous actions or environmental changes determine what is generated within the dialogue. It could combine dynamically generated context--like dialogue--with pre-written context, such as a character's written backstory or description, to compare character actions with narrative when considering how a character should act or react in a given scenario. The complex emergent behaviors of actionable characters in a diegetic environment could lead to interesting and potentially uniquely immersive experiences for players.

OPEN DIALOGUE IN PRACTICE

To better demonstrate the systems that could be made using qualitative programming, the following section will be dedicated to exposing open dialogue systems that have been created as experimental projects. Please note that the following game, *Outbound Sales* (Anonymous, 2025), is a small student project (created by the authors of this paper) meant to validate the potential of open dialogue systems and has not been viewed by a wide audience. However, the authors of this paper can attest to the functionality and consistency of the systems used within. A link to a playable demo can be found in the references section of this paper.

Conversation Scripting in *Outbound Sales*

As an example of how qualitative programming and conversation scripting can be used for traditional text parsing and creating text-based adventure games, the following section will describe how *Outbound Sales* uses an open dialogue system that creates dialogue for predetermined characters and changes game states based on qualitative data.

States of a sales call in Outbound Sales:

Outbound Sales is a conversational game in which the player acts as a telemarketer and is tasked with selling a variety of fictional items to one of a variety of NPCs in order to make a profit. The main gameplay revolves around the player hosting a fictional phone call with the NPC in a style similar to telemarketing or 'cold calling' and having an open conversation with them within the game context. After the conversation ends, qualitative variables are used to parse the deciding outcomes of the call, such as whether the NPC has decided to buy the product the player is selling, and the price that the player has decided to sell the item for.

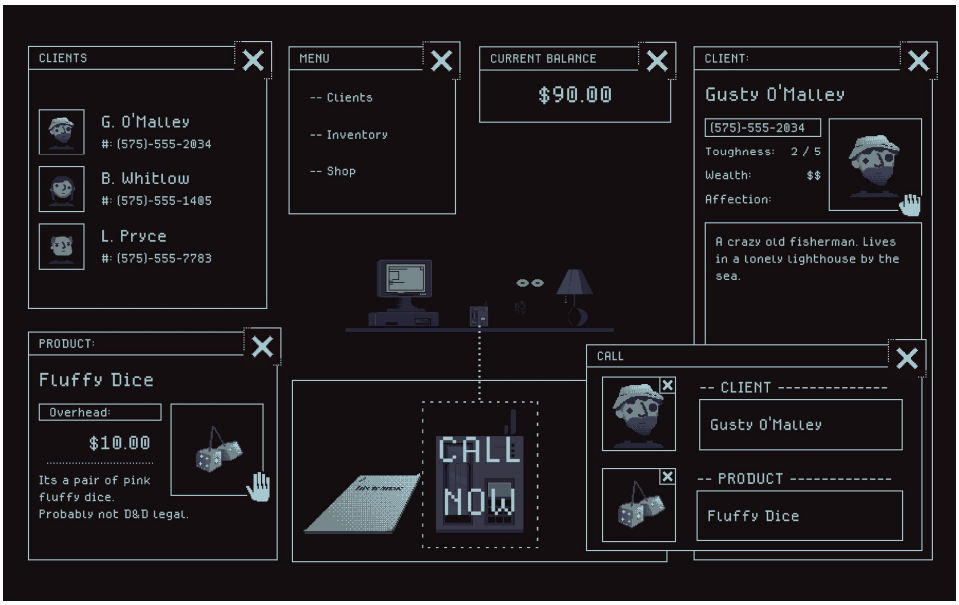

Figure 7.a: Players in *Outbound Sales* must select an item and an NPC from the in-game menu before beginning a conversation.

Figure 7.b: The LLM agent generates dialogue for the NPC, in the context of a sales call.

Figure 7.c: The player is given as much time as needed to type their own dialogue into a text box to respond to the NPC.

Figure 7.d: The player's written dialogue is then animated in a speech bubble. This is to provide the player with feedback that their dialogue is being conveyed diegetically, and to allow time for the LLM agent to generate the next response.

Figure 7.e: Over the course of the conversation, every line of LLM-generated dialogue considers the player's dialogue as well as the pre-written goals of the designer. Before the conversation ends, the NPC will communicate if the sales pitch has convinced them.

Figure 7.f: After the sales call ends, the conversation is parsed for specific semantic data (namely, did the NPC agree to purchase the item and for how much) and accurately displays the end result of the sale.

The open dialogue system in *Outbound Sales* involves two stages, beginning with a conversation and ending with an evaluation of the conversation. The first stage involves hosting a dialogue between the player and an LLM, both acting as agents in a Dialogue Loop (Figure 1). The player types their dialogue into a text box, and the LLM agent generates the NPC's response to the dialogue based on pre-written context and directing prompts:

Turn of Conversation **n**	**Example Directing Prompt** (to be added to the LLM's context window before generating a dialogue response)	**Design Philosophy**
0	"A funny, crazy old sea captain named Gusty receives a phone call. Since not many people call him, he assumes at first it's a prank caller. The sea captain speaks in short, brunt sentences."	This line is meant to define the character's personality and greater diegetic context before the first line of dialogue is generated. This will set the tone of the conversation.
1	"The sea captain now gets the feeling that the caller is someone trying to sell him something. He wants to antagonize the caller."	This is to create some tension with the player and grab their attention by challenging them, while also realigning the conversation towards being a sales call.
2	"Something that the caller says reminds the sea captain of a humorous tale of his time at sea, and he becomes much more animated in his speech."	By leading the NPC's generated dialogue to incorporate what the player character has previously said, this prompt is meant create feedback for the player to understand that the NPC is hearing their argument, and the directed change in the NPC's tone results in a dynamic change in the character's attitude.
3	"The sea captain is close to reaching a decision on whether or not he will buy what the telemarketer is selling, and is hesitant about being scammed by the telemarketer. He wants to know the price."	This prompt's effect is to remind the player that they should clearly state the price of the item, and to invite closing arguments.
4	"The sea captain decides if the telemarketer was convincing enough, and lets them know whether or not he will buy the product. He'll want to say goodbye and end the conversation afterwards."	This ensures that the next line of dialogue will create definitive feedback on the success of the sale and denote the end of the conversation before the diegetic call ends.

Table 1: This table describes an example sequence of directing prompts that are added to the LLM's context window in *Outbound Sales* before dialogue is generated. Each prompt is specifically designed to have an effect on the conversation that either creates narrative tension, hints at character development, or provides feedback to the player about the current state of the sales call.

As the directing prompts are injected into the conversation before a line of NPC dialogue is generated, the behavior of the NPC can change dramatically. This can be used to fulfill the original purpose of traditional NPC dialogue by manipulating the LLM agent to provide the player with information about the state of the game and to create intentional spikes in narrative tension. This allows each conversation to be designed intentionally before the conversation happens, to create a variety of tones, feelings, and specific effects that distinguish one conversation from another.

Qualitative Programming in *Outbound Sales*

Over the course of the conversation, both the player and the LLM agent will take turns writing dialogue. To reiterate: since the LLM's generated dialogue is dependent on the player's input, and we are unable to predict the player's input into the open dialogue system, both the generated text and the input text can be treated as non-deterministic. There is no guarantee that any specific words are being used by either the player or the LLM agent, and therefore traditional text parsing systems like those seen in *Colossal Cave Adventure* and *Zork* cannot reliably be used to change game states based on the contents of the conversation. As a simple example, if a traditional parsing program were to attempt to definitively parse the character string 'hello' from the conversation to judge if the player was greeting the NPC, the text parsing program would be unable reliably to handle every possible way the player could choose to greet the NPC. Additionally, the traditional text parsing system would be unable to parse if the word 'hello' was said in the context of an actual greeting directed at the NPC, as seen in the sentence: "My friend once said hello to me."

However, by leveraging an LLM agent's ability to make subjective observations, we can parse the meaning of the words present within the dialogue. In *Outbound Sales*, qualitative programming is used to determine the results of every sales call, and trigger game state changes accordingly. Three qualitative variables are used to determine the outcome of the conversation:

Variable 1: “Did [NPC] agree to buy the item that [PLAYER] was selling?”

Data type: Dichotomy

Input: [NPC], [PLAYER]

Context: Transcript of Conversation

Output: True, False, or Unclear

Variable 2: “Does the item that [PLAYER] is selling vaguely resemble [ITEM]?”

Data type: Dichotomy

Input: [PLAYER], [ITEM]

Context: Transcript of Conversation

Output: True, False, or Unclear

Variable 3: “What was the selling price that [NPC] agreed to?”

Data type: Nominal

Input: [NPC]

Context: Transcript of Conversation

Output: either a nominal integer or Unclear

Before every diegetic sales call begins, the player is prompted to select an item from their inventory to sell and an NPC client to sell the item to. The player is also prompted with some game rules as to what needs to happen during the conversation for the sale to be registered as a success:

1. The client must verbally agree to purchase the item.
2. The player must only sell the item they selected from their inventory beforehand.
3. The selling price of the item must be clearly stated. The player decides how much the price is and is encouraged to make as much money as possible.

After the sales call ends, the transcript of the conversation is used as context for the qualitative functions to evaluate. If any of the qualitative variables result in False or Unclear, the game can change game states to provide feedback to the player as to the success of the sale and specifically which rule they failed to follow. If all the qualitative variables return True and the price of the object is clearly defined within the transcript of the conversation, the game can provide feedback as to the success of the sale and award the player with the money they have earned to be used in the game's shop after the call has ended.

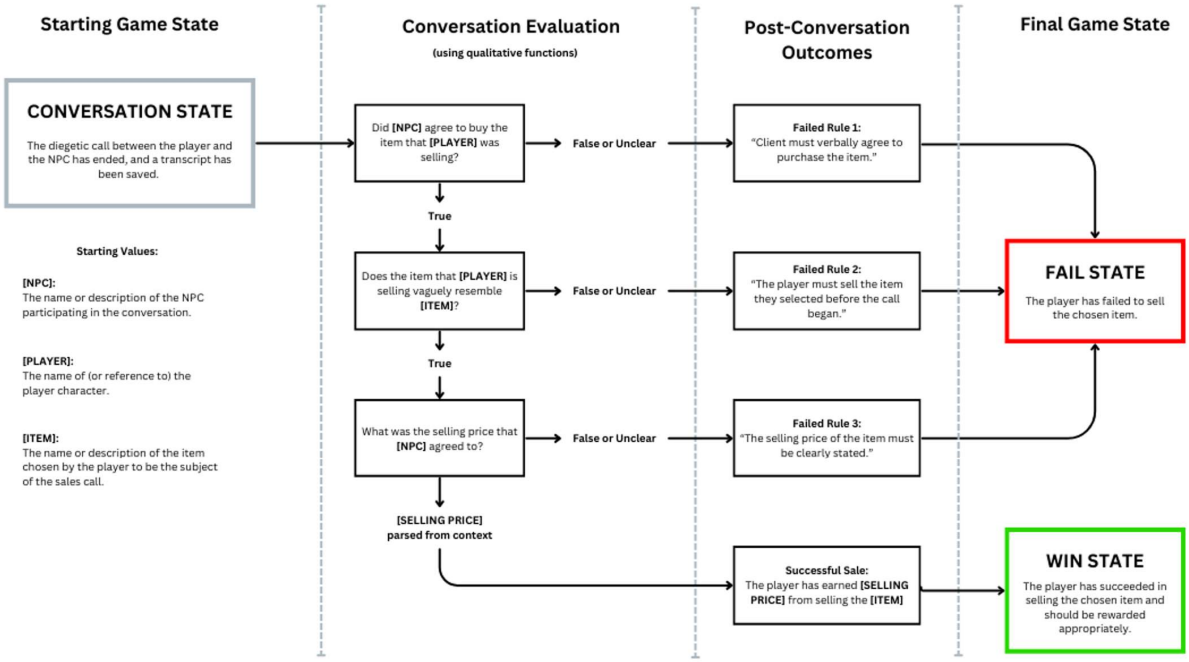

Figure 7.f: *Outbound Sales* uses several qualitative analysis functions to determine the outcome of each sales call. This flow chart diagram shows how the subjective observations of an LLM is used change game states.

CONCLUSION

These systems, which were impossible before the invention of LLMs, can be used to create a whole new genre of games with an entirely new feel: conversational games. In *Outbound Sales,* the player's choice of words is important, but they are not the deciding factor of convincing the NPC to buy the product. The words of the player are merely an interface through which the player communicates their very own logic, attitude, sarcasm, emotion, and reasoning, all the unquantifiable things that make up human interaction. However, communicating these concepts to an empty room would not be a game. Just as important as the player's voice, the way an LLM's output interprets attitude, emotion, and reasoning *in the context of being a human listener* is what determines what the game is like. For example, if the player sets the price of an item too high, an ordinary NPC would back off, leading to bartering and negotiations. But if the NPC's character is cartoonishly rich, the NPC might be offended if the price was set too low. An item beloved by a certain type of NPC might be hated by another, but if the player is convincing enough and appeals to those exclusively-human emotions the right way, they could close even the most impossible of sales.

Qualitative data and scripted conversations allow for games that will have players arguing, deceiving, and reasoning with a computer. And although an open dialogue system uses an LLM to generate dialogue, the feeling of these games is determined

by the intent of the designers and the context of the larger narrative. To use LLM text generation as a funnel through which to channel and process the inexplicable aspects of human communication begs the question: what could be possible now that was never possible before?

REFERENCES

Adams, R. R. III, Albert, D., Garriott, R., et al. (1989). *Ultima Trilogy I·II·III Player Reference Guide*. Londonberry, NH, USA: Origin Systems.

Addams, S. 1990. *The Official Book of Ultima*. Greensboro, NC, USA: COMPUTE! Books.

Anonymous Author (Author of this Article). 2025. *Outbound Sales*. PC Game. Anonymous Institute.

Bates, B. 1993. *Eric the Unready*. Computer Game. Chantilly, VA, USA: Legend Entertainment.

Barton, M. and Stacks, S. 2019. *Dungeons & Desktops: The History of Computer Role-Playing Games* (2nd edition). Boca Raton, FL, USA: CRC Press.

BioWare. 2007. *Mass Effect*. Video Game. Microsoft Game Studios.

Crawford, C. 2003. "Interactive Storytelling." In *The Video Game Theory Reader* edited by M. J. P. Wolf and B. Perron, 259–273. New York, NY, USA: Routledge.

Creative Labs, Inc. 1989. *SOUND BLASTER*. Hardware/Software Package. Snata Clara, CA, USA: Creative Labs, Inc.

Creative Labs, Inc. 1990. *SOUND BLASTER User Reference Manual*. Santa Clara, CA, USA: Creative Labs, Inc.

Creative Labs, Inc. 1991. *Dr. Sbaitso*. Software. Santa Clara, CA, USA: Creative Labs, Inc.

Crowe, M. and Murphy, S. 1986. *Space Quest*. Computer Game. Sierra On-Line.

Crowther, W. and Woods, C. 1976. *Colossal Cave Adventure*. Computer Game. Will Crowther and Don Woods.

Fuhrig, L. S. 2016. "5.25" Floppies: All Is Not Lost." *Smithsonian Institution Archives*, 22 Mar. siarchives.si.edu/blog/floppies-all-not-lost.

Garriott, R. 1981. *Ultima*. Computer Game. California Pacific Computer Company.

Gilbert, R. and Winnick, G. 1987. *Maniac Mansion*. Computer Game. Lucasfilm Games.

Godfrey, T. 2021. "Domain-Specific Modelling Languages for Participatory Agent-Based Modelling in Healthcare," *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 654–659. Fukuoka, Japan. doi: 10.1109/MODELS-C53483.2021.00105.

Han, L. and Tang, H. 2002. “Designing of Prompts for Hate Speech Recognition with In-Context Learning,” *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, 319–320. Las Vegas, NV, USA.

Horii, Y. 1986. *Dragon Quest*. Video Game. Enix.

Infocom. 1980. *Zork*. Computer Game. Infocom.

Interplay Productions. 1997. *Fallout*. Computer Game. Interplay Productions.

Kohler, C. (2005). *Power Up: How Japanese Video Games Gave the World an Extra Life*. Indianapolis, IN, USA: Pearson Education.

Lee, M., Liang, P., Yang, Q. 2022. “Designing a human-ai collaborative writing dataset for exploring language model capabilities.” *Proceedings of the 2022 CHI conference on human factors in computing systems.* 1–19. doi: https://doi.org/10.48550/arXiv.2201.06796

OpenAI. 2022. *ChatGPT*. Online Software. OpenAI.

Origin Systems. 1988. *Ultima V: Warriors of Destiny*. Computer Game. Origin Systems.

Origin Systems. 1990. *Ultima VI: The False Prophet*. Computer Game. Origin Systems.

Origin Systems. 1992. *Ultima VII: The Black Gate*. Computer Game. Origin Systems.

S. -H. Li, G. Zhou, Z. -B. Li, J. -C. Lu and N. -B. Huang. 2023. "The Causal Reasoning Ability of Open Large Language Model: A Comprehensive and Exemplary Functional Testing*," 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, 240-249. Chiang Mai, Thailand. doi: 10.1109/QRS60937.2023.00032.

Schell, J. 2008. *The Art of Game Design: A Book of Lenses*. Burlington, MA, USA: Elsevier.

Shuyin Ouyang et al. 2023. "LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation," *arXiv,* doi: https://doi.org/10.48550/arXiv.2308.02828

Smith, G. M. 2002. “Computer Games Have Words, Too: Dialogue Conventions in Final Fantasy VII.” *Game Studies*. 2 (2). https://gamestudies.org/0202/smith/

Veloso, C. and Prada, R. 2021. “Validating the plot of Interactive Narrative games.” *2021 IEEE Conference on Games (CoG),* 1–8. Copenhagen, Denmark. doi: 10.1109/CoG52621.2021.9618897.

Zhao, Wayne Xin, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min et al. 2023. "A survey of large language models." arXiv preprint arXiv:2303.18223 1, no. 2.

Williams, R. 1984. *King’s Quest*. Computer Game. Sierra On-Line.

ZA/UM. 2019. *Disco Elysium*. Computer Game. ZA/UM.

APPENDIX

Link to play *Outbound Sales:*

Outbound Sales

http://bit.ly/3Ku66cn