UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Egor Lukjanenko

Modular system for text-based interaction with non-player characters in game environments

Bachelor's Thesis (9 ECTS)

Supervisor:
Giacomo Magnifico, MA

Tartu 2025

Modular system for text-based interaction with non-player characters in game environments

Abstract:

This thesis presents the design and implementation of a modular system for text-based dialogue in games, focusing on controlled and predictable interactions. The system is composed of several key modules: input parsing, dialogue flow, personality module, world-state module, and natural language output generation. Dialogue transitions are managed using a Dialogue Flow module, ensuring logical state progression. The Personality Module is used to add personality-consistent instructions, while a structured World State supports fact-based reasoning. Finally, a large language model (LLM) generates responses based on the instructions produced by previous modules. A functional prototype was developed to demonstrate the system in a game-like environment.[1] Evaluation confirms the system's modularity and capacity for coherent and branching dialogue flow, with text-based user input. Future improvements could focus on real-time interaction and enhancing contextual consistency in dynamic scenarios.

Keywords: dialogue systems, game AI, modular architecture, natural language processing

CERCS: P170 Computer science, numerical analysis, systems, control

[1] Source code available at https://github.com/egor123/modular-dialogue-system

Modulaarne süsteem tekstipõhiseks suhtluseks mängukeskkondade mitte-mängijategelastega

Lühikokkuvõte:

Käesolev lõputöö tutvustab modulaarse süsteemi kavandamist ja teostust tekstipõhise dialoogi jaoks mängudes, keskendudes kontrollitud ja ennustatavale suhtlusele. Süsteem koosneb mitmest põhikomponendist: sisendi töötlemine, dialoogivoo juhtimine, isiksuse moodul, maailmaseisundi moodul ning loomuliku keele väljundi genereerimine. Dialoogide üleminekuid juhib dialoogivoo moodul, mis tagab loogilise olekute jada. Isiksuse moodul lisab tegelaskujule vastavaid käitumisjuhiseid ning struktureeritud maailmaseisundi moodul toetab faktipõhist otsustamist. Lõpuks genereerib suur keelemudel (LLM) vastuseid varasemate moodulite koostatud juhiste põhjal. Funktsionaalne prototüüp töötati välja süsteemi demonstreerimiseks mängulaadses keskkonnas.[2] Hindamise käigus kinnitati süsteemi modulaarsust ja suutlikkust luua sidusat ning harunevat dialoogivoo, lähtudes tekstipõhisest kasutajasisendist. Tulevikutöös võiks keskenduda reaalajas suhtlusele ja kontekstitruevuse parandamisele dünaamilistes olukordades.

Võtmesõnad: dialoogisüsteemid, mängu tehisintellekt, modulaarne arhitektuur, loomuliku keele töötlus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

[2]Lähtekood on saadaval aadressil https://github.com/egor123/modular-dialogue-system

Contents

1. Introduction

In modern narrative-driven games, especially role-playing games (RPGs), dialogue systems play a crucial role in facilitating player immersion within the presented narrative and emotionally engaging them. As generative artificial intelligence becomes more capable, there is a growing trend to use this technology to generate fully open-ended and dynamic dialogues [1]. These dynamic forms of dialogue, despite being able to provide a wider array of inputs to the user (such as freeform text or even voice input), usually suffer from unpredictable, inconsistent, or even immersion-breaking behavior[1]. Additionally, non-playable characters (NPCs) controlled by generative AI may lack enough consistency in their course of action to be properly integrated in the game narrative structure. In narrative-heavy games, where consistency and dramatic pacing are essential, such unpredictability can diminish the player's experience.

This thesis argues for an alternative design philosophy: prioritizing *predictability over freedom*. Instead of relying solely on generative AI to drive interactions, this work presents a modular dialogue system where controlled narrative goals guide player interaction; this choice allows to generate instructions and subsequently use generative AI to enrich responses derived from such instructions. This system thus combines the benefits of traditional dialogue methods with the adaptability of generative AI, aiming to balance narrative coherence and adaptability to freeform-text user input.

The motivations and aim behind this work can be summarized as the following research questions:

- How can a dialogue system be designed that allows dynamic player input while maintaining narrative coherence?
- What modular structure supports clear communication between different dialogue logic components?
- To what extent can AI models be used to fill in stylistic or contextual details without violating the narrative goals of the game?

The modular software provided in this thesis focuses on individual NPC interactions in a role-playing or narrative-driven game context. Although the system can be adopted for a far wider range of use cases, the prototype is limited exclusively to text-based input and the implementation of possible speech-to-text synthesis add-ons is beyond the scope of this thesis. Additionally, full natural language understanding, while in the realm of possibilities, is outside the scope of this thesis - as it is limited to predefined state configuration.

Terminology. Within the scope of this work, the term *artificial intelligence (AI)* is used only to indicate generative AIs, in contrast with the general usage within the fields of game development (the assumption of behavioral patterns) and machine learning (algorithms).

Use of AI for formatting. Writefull's model, accessed via Overleaf.com, was used for formatting and grammar correction purposes.

2. Theoretical Background

2.1 Dialogue Systems

Dialogue systems enable computers to interact with users using natural language. In games, dialogue systems function as a mechanism for narrative interaction between the player and the game. They play a crucial role in the story, expressing character personalities and building the world of the game. Depending on the designer's final goals, these systems can range from simple, linear configurations to more complex, adaptive architectures.

According to Brusk and Björk [2], dialogue systems can be categorized into four main types according to their complexity:

- **Finite State-based Systems**: These operate using predefined dialogue trees or graphs, where each state prompts the user with a fixed response, and transitions are dependent on matching the user's input. These systems are context-dependent but offer limited flexibility or reactivity. Example: Finite-State Machine, Behavior Trees.
- **Form-based Systems**: These allow multiple pieces of information (slots) to be gathered from a single user input, and support input provided in varied order. Although slightly more flexible than finite-state systems, they are still fundamentally pattern-matching oriented.
- **Plan-based Systems**: These interpret the user's input in terms of goals and task solutions. They usually used in expert systems, as they purpose is to achieve a consensus between user and the system. Example: Goal Oriented Action Planning, Hierarchical Task Networks.
- **Agent-based Systems**: The most complex type, where the system maintains an autonomous goal-directed agent capable of replanning based on dynamic environments. Example: The Belief-Desire-Intention model.

2.2 Behavior Patterns

The design of intelligent agents in games and interactive systems has long relied on structured patterns for behavior generation. Among the most widely used approaches are finite state machines (FSM), behavior trees (BT), goal-oriented action planning (GOAP), hierarchical task networks (HTN) and belief-desire-intention (BDI). These architectures provide different trade-offs between modularity, reactivity, predictability, and expressive power-qualities that are essential when building behavioral systems. Usually these analogythms are used for in game

characters behaviors, but later in this thesis their applications for dialogue system would be considered.

Finite-state machines (FSM) represent one of the earliest and simplest behavior control models. FSM exists in one of several discrete states. Transitions between states are triggered by external inputs or internal conditions. Each state defines a fixed behavior, and once a transition condition is met, the control jumps to a new state. FSMs are straightforward to implement and reason about, making them ideal for tightly scripted interactions. However, as Iovino et al. [3] point out, FSMs can quickly devolve into monolithic structures that are difficult to extend, debug, or reuse. In dialogue systems, FSMs are typically used to implement non-linear branching conversation trees. They provide deterministic flow, but can become rigid and difficult to maintain in more dynamic scenarios.

Behavior Trees emerged in the game industry as a response to the limitations of FSMs. Instead of distributing logic across states and transitions, BTs organize behavior hierarchically using control flow nodes like sequences, fallbacks, and parallels. Execution begins at the root and proceeds by invoking child nodes based on control flow logic. Each node returns one of the three statuses-success, failure, or running. This design enables modular construction, where behaviors can be reused and composed cleanly. [3] Compared to FSMs, BTs are more visually readable, scalable, and reactive. In dialogue systems, BTs are valuable for managing layered interaction strategies. They can support fallback responses, conditional branching.

When FSM and BT represent the state-based system, **Goal-Oriented Action Planning (GOAP)** is a flexible planning-based model of behavior. GOAP agents are provided with a set of actions, each defined by preconditions and effects, and a desired goal state. The system uses a forward search to generate a sequence of actions that transitions the world from its current state to the goal. As Studiawan et al. [4] demonstrate, this enables agents to react dynamically to changing environments by regenerating plans at runtime. GOAP has been used successfully in tactical and real-time games, offering emergent behavior and adaptability. This model also requires a well-structured world model and action set and may introduce higher computational overhead compared to more deterministic models.

Hierarchical Task Networks (HTN) extend planning further by incorporating hierarchical

decomposition of tasks. In HTN, complex goals are broken down recursively into subtasks using predefined methods. Each method defines a way to accomplish a high-level task through a sequence or set of lower-level actions or sub-goals. Georgievski and Aiello [5] highlight that this allows designers to use domain knowledge directly in the planning process, improving both plan quality and control. Unlike GOAP, which relies purely on action chaining, HTN planning follows a top-down approach guided by the structure of the task network. This makes HTN particularly well-suited for heavy structured gameplay scenarios.

The Belief-Desire-Intention (BDI) model offers a cognitively inspired architecture for agents with autonomy and persistence. Agents maintain explicit representations of their beliefs (information about the world), desires (motivational goals), and intentions (committed plans). Behavior is driven by selecting and executing plans that achieve current desires, while reacting to updates in belief state. As Wadsley and Ryan [6] explain, this makes BDI have great potential to improve AI agent models by adding focus on the character model. Supports individualized motivations, long-term goals, and adaptive story responses. However, implementing BDI requires careful design of the intention management system to avoid irrational behavior or plan changes.

2.3 Natural Language Processing and generative AI

As this thesis heavily relies on working with text values, **Natural Language Processing (NLP)** is crucial for parsing user input, identifying intent, extracting relevant entities, and generating context-aware responses. NLP is a field of artificial intelligence that focuses on enabling computers to interpret, understand, and generate human language in a meaningful way [7]. Using various NLP methods, dialogue systems can engage users in more natural, fluid, and contextually appropriate interactions. The techniques described below will be used for different subsystems.

Summarization is a fundamental task in natural language processing (NLP) that aims to condense long content into a shorter version while preserving the core meaning. It is generally categorized as either extractive or abstractive. Extractive summarization involves selecting and reordering salient sentences from the source text based on statistical and linguistic features [8]. In contrast, abstractive summarization generates new sentences by rephrasing or synthesizing information from the original text, requiring deeper semantic understanding and reasoning. Although extractive methods are lighter, they often suffer from readability and cohesion issues

because of problems such as dangling anaphora. Abstractive methods offer a more human-like, concise and coherent output, but are significantly more complex due to the intricacies of natural language generation [9].

Vector embeddings represent words, phrases, or entire sentences as dense vectors in a high-dimensional space, capturing their semantic relationships. The introduction of the Transformer architecture revolutionized this space by relying entirely on self-attention mechanisms. This design allows models to capture global dependencies regardless of sequence position and enables efficient parallel training. As a result, Transformers have become the basis for models in a wide range of NLP tasks [10].

Sentiment analysis is another fundamental task in natural language processing, which focuses on determining the emotional tone of a piece of text-typically categorized as positive, negative, or neutral. [11].

Another critical aspect of language understanding systems is **named entity extraction**. This covers three main tasks such as Named Entity Recognition (NER), Named Entity Disambiguation (NED), and Named Entity Linking (NEL). These processes identify and classify references to real-world entities (e.g., people, locations), resolve ambiguity, and link them to standard identifiers in knowledge bases. Modern entity extraction systems are increasingly adopting deep learning approaches that integrate these subtasks into unified models, allowing greater accuracy and contextual awareness[7].

Generative artificial intelligence refers to a class of AI systems capable of producing new content, such as text, images, code, or audio. A prominent example of generative AI is the family of **Large Language Models (LLMs)**. LLMs are typically built on Transformer architectures and trained using self-supervised learning objectives, such as next-word prediction. LLMs can handle a variety of tasks by conditioning on different prompts, allowing them to follow instructions, perform reasoning, and even simulate personalities or roles in a dialogue[12]

2.4 Information Storage and Retrieval

Encoding all relevant world knowledge directly into a large language model (LLM) context window is computationally infeasible, and the excess of unnecessary information worsens the

accuracy. For this purpose, the storage and retrieval of information related to the game is important for the current project. This creates the need for an external information storage and retrieval system for structured knowledge.

A prominent approach involves the use of **Knowledge Graphs** (KGs), which offer a structured, computer-processable representation of information. A knowledge graph consists of nodes representing entities (e.g., characters, locations, items) and edges denoting relationships between them (e.g., owns, located_in, interacts_with). [7]

As an alternative, **document-based systems** is a more intuitive model. For retrieval a well-established baseline method, Term Frequency-Inverse Document Frequency (TF-IDF), which scores terms based on how frequently they appear in a specific document relative to their frequency across the entire corpus. This weighting helps to surface terms that are meaningful for the current context while lowering common or generic words [13]. Transformer-based approaches also exist, which extend beyond lexical similarity by leveraging contextual embeddings by avoiding traditional bag-of-words stages entirely, using embeddings and vector-based similarity[14].

Although knowledge graphs could also be constructed from text documents [13], this thesis avoids such a method due to the additional uncertainty introduced during the information extraction and conversion stages.

3. System Architecture

For this thesis, a modular and extensible architecture was designed to manage text-based dialogues. The core idea behind the architecture is to separate the dialogue generation process into well-defined independent stages, where each step is encapsulated within an independent module. This modularity enables greater control, transparency, and as well more realistic and complex behavior.

In order to better manage the system modularity, a shared data structure called *Data Container* is used as a moving block along the processing pipeline, passing through the entirety of it and being incrementally modified in the process. Furhtermore, instead of relying solely on large language models (LLMs) to generate new content from only inputs and conversation context, the system assembles a prompt composed of predefined instructions, facts, and character traits. The LLM is used only to rephrase and contextualize provided information in a way that reflects the NPC's personality and current situation.

The pipeline processing is thus the following: the data container, initialized with the input and interaction history, is first processed to normalize and prepare the input. The Dialogue Flow Module then determines the current conversational state and sets the main instruction. The Personality Module modifies the instruction set to reflect the NPC's traits and behavioral tendencies. Then, the World State Module adds to the container relevant factual context from the game environment. Finally, the fully populated container is forwarded to the Generation Module, which produces a stylized, context-aware NPC response using a large language model.

An overview of the pipeline is illustrated in Figure 1.

As the system was designed, with flexibility and modularity in mind, configuration requires to wrap all models in interface layers to unify their usage across modules. This design decoupled logic from specific underlying technologies or libraries. All modules have base classes, allowing the creation and integration of custom implementations to meet the needs of the designers.

On startup, all systems will be preconfigured and initialized to save time at runtime. All models are initialized, and each module loads its configuration from external JSON or YAML files. These configurations define behavior, parameters, and available transitions, enabling non-programmatic and intuitive control over system logic. Once initialized, all modules are registered within the pipeline. The complete system can then process the input in a fully encapsulated manner: The application submits a single string input and receives a fully generated string response as output.

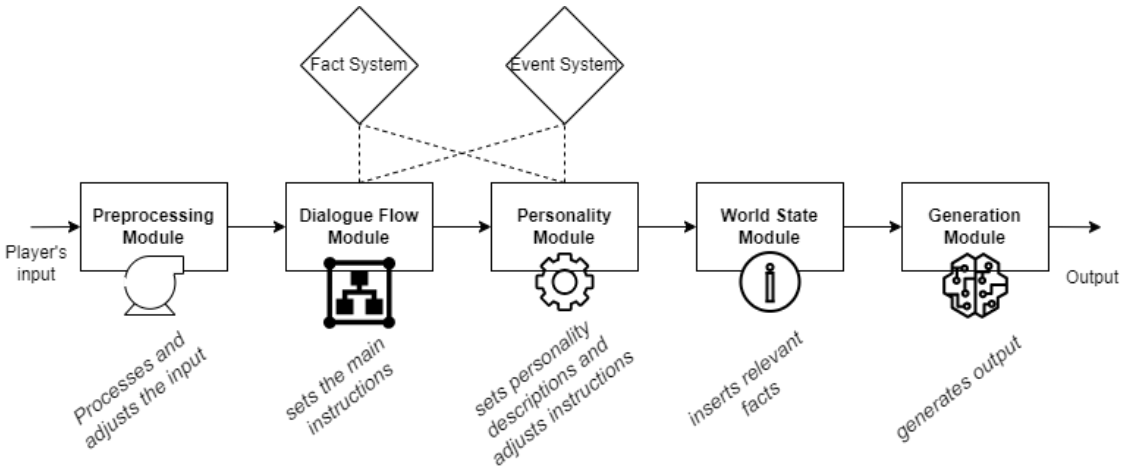

Figure 1. System pipeline overview showing modular flow: input is processed step-by-step through Preprocessing, Dialogue Flow, Personality, World State, and Generation Modules.

3.1 Auxiliary subsystems

Some modules, such as Dialogue Flow and Personality Modules, require logical conditions to trigger transitions or match rules. Those are defined as string expressions (e.g. ”sim(’user rejects to take the quest’) or (sent(’neg’) > 0.8 and fact(’failure’))”) in the configuration files and are parsed and compiled into expression trees using the Shunting Yard algorithm [15], allowing fast and deterministic evaluation at run-time. In addition, all possible vector embeddings are pre-calculated in advance for optimization purposes. In the current prototype, BERTweet is used for sentiment analysis, a RoBERTa model trained on English tweets (finiteautomata/bertweet-base-sentiment-analysis) [11], and sentence transformers (sentence transformers/stsb-roberta-large) model [16] is used to produce vector embeddings. Llama3 [12] is used to create paraphrasings to expand the range of fitting formulations. This system can assess semantic similarity, sentiment analysis, and retrive fact values from the Fact System.

sentence transformers (sentence transformers/stsb-roberta-large) model [16] is used to produce vector embeddings.

To enable dynamic interaction between the game architecture and the dialogue system, the Dialogue Flow and Personality Modules use two subsystems:

- **Fact System**: A centralized key-value dictionary representing the current state of the game. Each key (e.g., `"current_level"`, `"character_mood"`) is associated with a primitive value (e.g., string, Boolean, float). The system supports a callback mechanism

by which other systems can subscribe to changes in specific facts. When a fact is modified, all registered callbacks are automatically triggered. This enables reactive and responsive behavior across systems while maintaining loose coupling.

- **Event System**: A lightweight communication mechanism based on named events. Events can be explicitly triggered either directly by name or via callbacks from the Fact System. The Event System is used to provide mechanism to trigger in-game events and actions and reduce direct dependencies.

As a result, those systems allow us to alter the state of the game and impact the environment.

3.2 Data Container

The Data Container encapsulates all the important context and instructions needed for response generation. It is used to pass information between different modules. This data structure includes the following fields:

- **input**: `str` – the textual input of the player.
- **history**: `list[str]` – the sequence of previous dialogue lines in the current interaction.
- **freedom**: `float (0-1)` – defines the allowed deviation from the current topic; higher values allow off-topic discussion.
- **urgency**: `float (0-1)` – represents the instantaneous response required by the player; higher values prioritize immediate reaction.
- **facts**: `list[str]` – a set of contextual facts relevant to the current interaction, not used as direct instructions.
- **personality**: `list[str]` – traits and stylistic attributes that characterize the NPC's speech and behavior.
- **instructions**: `list[str]` – explicit instructions outlining what the NPC is expected to say.

3.3 Preprocessing Module

The Preprocessing Module serves to refine player input, reducing issues that arise from using raw text directly. For example, longer inputs can negatively affect vector-based similarity calculations; to address this, the module applies abstractive summarization when the amount of input tokens exceeds a predefined threshold.

Abstractive summarization was selected over extractive, as the stylistically rich nature of role-playing dialogue as well as relatively short text size introduce difficulties for extractive methods.

The current prototype uses the large-scale BART model (facebook/bart-large-cnn) for summarization. [17]

3.4 Dialogue Flow Module

The Dialogue Flow Module is responsible for maintaining the current interaction state and setting instructions for following modules. Its primary function is to ensure that the conversation remains coherent and progresses in a controlled manner.

The main implementation utilizes a Finite State Machine (FSM), which determines dialogue states and transition. This approach provides a predictable and user-friendly framework for managing the progression of dialogue, as FSMs are particularly well-suited for common dialogue scenarios where the interaction follows a clear structure.

Upon receiving player input, the module evaluates all transition conditions associated with the current state. If a condition score exceeds threshold, and the margin between the top two scoring conditions—exceeds high enough, a state transition is triggered.

Relevant data such as instructions (`templates`) are then passed to the data container. If a transition occurs, the corresponding in-game events, if set to `on_exit` or `on_enter`, are also triggered.

Figure 2 shows an example FSM configuration used in the system.

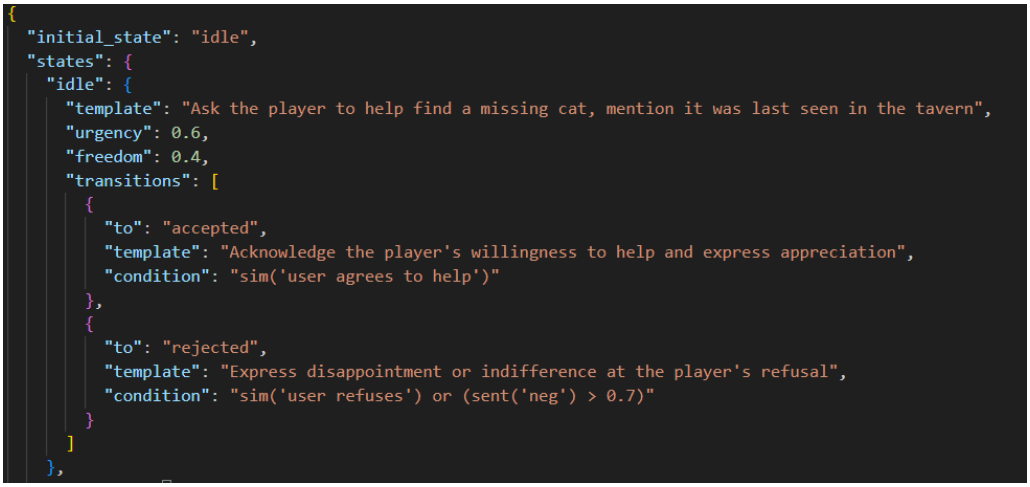

Figure 2. Example of an FSM configuration in JSON format, showing states, transitions and other fields.

An alternative implementation using Goal-Oriented Action Planning (GOAP) is also available. GOAP is appropriate in situations where the exact sequence of actions is less important than achieving a specific conversational goal. However, despite its flexibility, configuring GOAP for longer dialogues can become cumbersome.

3.5 Personality Module

The Personality Module inserts stylistic and behavioral personality descriptions that reflect the NPC's responses, as well as expanding the instructions set and tracking the general behaviour of characters. The current implementation is rule-based: each rule is evaluated against the current dialogue context and player's input, and if a rule is matched, the instruction list is modifid either by inserting a new instruction (`type: "insert"`) or by overwriting the entire set (`type: "overwrite"`). This enables dynamic characterization, allowing traits such as mood, reputation, or trust level to evolve during interaction, as well as comments on defined topics important to the character. Some rules may depend on the `urgency` and `freedom` fields provided by the Dialogue Flow Module, either allowing or restricting additional traits and topics.

Figure 3 shows an example JSON configuration used to define rules and personality expressions.

```
{
    "description": "Shady merchant, who offers quests in the tavern",
    "rules": {
        "apples":
        {
            "template": "I love apples! I'll tell you every fact about them",
            "freedom": 0.6,
            "type": "overwrite",
            "condition": "sim('user ask about apples') > 0.5"
        },
        "tavern":
        {
            "template": "Don't be rude with me!",
            "freedom": 1,
            "type": "insert",
            "condition": "sent('neg') > 0.8"
        }
    }
```

Figure 3. Example JSON configuration of the Personality Module with insert and overwrite rules.

As an alternative, the implementation of the provided Behavior Tree-based personality module offers greater flexibility, modularity, and clearer control flow, making it easier to manage complex behaviors and personality models.

3.6 World State Module

The World State Module is responsible for injecting relevant knowledge into the data container. This contextual information fills gaps not covered by the Dialogue Flow or Personality modules and helps the language model ground its responses in the current state of the game world.

The current module implementation retrieves facts from a knowledge graph and selects those appropriate to the current context from the data container. These facts are not used as direct instructions, but are instead appended as contextual cues to guide the final response. For named entity extraction the *en_core_web_sm* model was used[18].In addition, it inserts the required values in the set of instructions from the Fact System (e.g., ”Hello, ${usr_name}” → ”Hello, Traveler”).

Figure 4 provides an example of a JSON-based configuration used to structure information.

Figure 4. Example configuration of the World State Module for fact selection and tagging.

3.7 Generation Module

The Generation Module is the final stage of the pipeline and is responsible for producing the textual response of the NPC. The module uses assembled data container, which includes instructions, personality traits, contextual facts, and interaction history context.

Instead of generating output from raw prompts, the module instructs a large language model to synthesize a coherent and stylistically appropriate reply based on the structured input. The prompt explicitly asks the model to follow instructions while incorporating personality and contextual cues, resulting in responses that are both character driven and goal driven.

The current implementation uses the LLaMA 3 [12] language model to perform this generation step. Given its performance and open architecture, LLaMA 3 offers both stylistic fluency and sufficient controllability for the dialogue system's needs.

4. Prototype Implementation and Case Study

The implementation of prototypes aims to demonstrate the interaction between modules and assess the performance of the system in handling player inputs in a game-like environment. For demonstration purposes, a sample scenario was created featuring an NPC, a tavern keeper looking for his cat, who interacts with the player. In this scenario, the player has to agree to help and find the cat, which is located on the roof, as a result NPC will thank the player. The goal is to simulate a quest offer and resolution through dynamic and context-aware NPC behavior.

For response generation, the following prompt template is used:

```
You are roleplaying a character with the following personality traits:
**{personality}**.

Context to consider:
- **Known facts**:
{facts}
- **Urgency**: {container.urgency:.2f} → {urgency_desc}

Your task:
- React naturally to the user's input.
- Follow the intent of the instructions below, but rephrase them in a
natural, in-character way.
- Stay grounded in the provided facts and personality.
- Do not invent new actions or topics, only reword what's given.
- Do not add anything new
- Write small response

Conversation history:
{container.history[-self.history_limit:]}

User input:
"{container.input}"

Following intent of instruction below rephrase:
{instructions}

Your short response:
```

Upon system start-up, multiple NLP models are loaded, including: BERTweet (`finiteautomata/bertweet-base-sentiment-analysis`), Sentence-BERT (`stsb-roberta-large`), BART (`facebook/bart-large-cnn`), spaCy (`en_core_web_sm`), LLaMA 3.

Each model is wrapped within a uniform interface. Afterwards, all modules are loaded from external JSON or YAML configuration files and initialized. A sample console log of this process is shown in Figure 5.

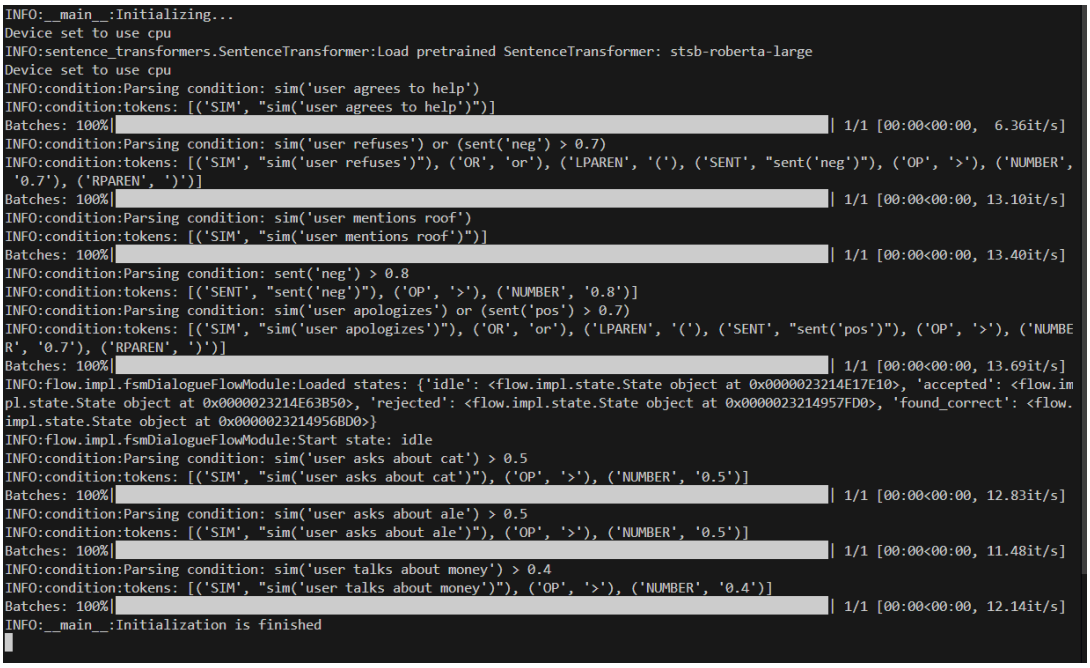

Figure 5. Console output during system startup and model loading

To demonstrate the work of the Personality Module, *"I will help you"* was submitted:

1. **Preprocessing Module** determines that no summarization is needed, as the input does not exceed the length threshold.

2. **Dialogue Flow Module** evaluates transitions from the state *idle*. It scores a transition to *accepted* with a score of 0.6890 and to *rejected* with a score of 0.0761. The system transitions to the *accepted* state, setting default urgency (0.5) and freedom (0). It inserts the transition instruction (*"Acknowledge the player's willingness to help and express appreciation"*) and the new state instruction (*"Suggest possible cat locations such as the roof, fireplace, or near the well; imply cat is clever or sneaky"*). As the "accepted" state starts, the event "quest_started" binded to "on_enter" is raised.

3. **Personality Module** adds the default personality description (*"Gruff tavern keeper who acts tough but has a soft spot for his cat."*). No rule matches are found, so no further personality adjustments are applied.

4. **World State Module** findes entry named *Whiskers* and adds following fact *"Whiskers": { "description": "black cat", "favorite_spots": ["roof", "fireplace", "behind barrels"] }*.

5. **Generation Module** fills the prompt template with the content of the data container and generates the final NPC response using LLaMA 3.

The result of this interaction within the interface is shown in Figure 6.

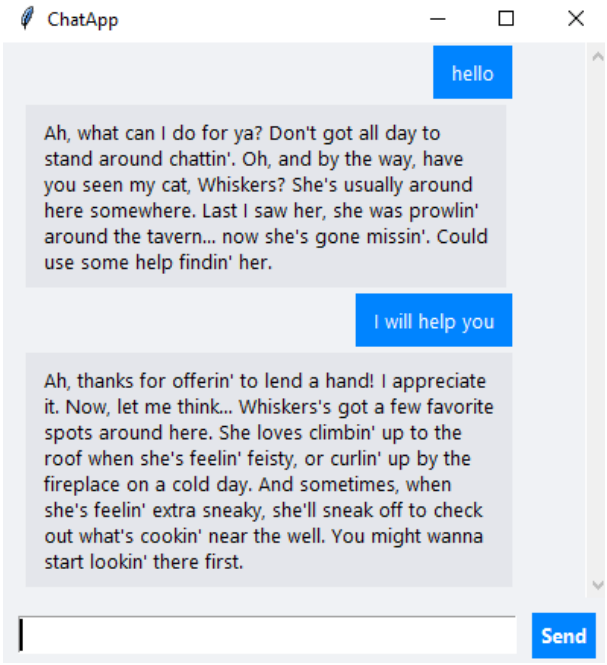

Figure 6. Example chat interaction after player agrees to help find the cat

Upon receiving the user's input, the dialogue pipeline is triggered. For example, inputting *"I will help you"* initiates the following processing sequence:

1. **Preprocessing Module** determines that no summarization is needed, as the input does not exceed the length threshold.

2. **Dialogue Flow Module** evaluates transitions from the state *accepted*. It scores a transition to *found_correct* with a score of 0580 and to *rejected* with a score of 0. No transitions are made and *accepted* state sets urgency as 0.5 and freedom as 0. It inserts the transition instruction (*Suggest possible cat locations such as the roof, fireplace, or near the well; imply cat is clever or sneaky*). No events raised.

3. **Personality Module** adds the default personality description (*"Gruff tavern keeper who acts tough but has a soft spot for his cat."*). Rule *ale* is matched, as it's type is *overwrite* all previous instructions are discarded and new instruction *Boast about the strength or uniqueness of the tavern's homemade ale* is added.
4. **World State Module** does not find related enties.
5. **Generation Module** fills the prompt template with the content of the data container and generates the final NPC response using LLaMA 3.

The result of this interaction within the interface is shown in Figure 7.

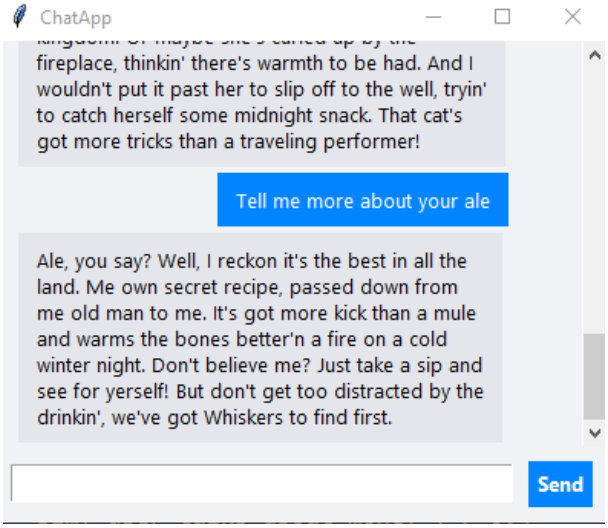

Figure 7. Example chat interaction after ask about personal topic

5. Discussion and Future Work

The modular design of the proposed dialogue system allows for seamless integration into a wide range of game architectures. Its flexibility lies in how individual components can be replaced or extended with different models or libraries without the need for significant structural changes. Configuration is handled through simple formats like JSON or YAML, which provides accessibility to designers. However, effective use of the system requires understanding of behavioral patterns and being able to craft high-quality text-based instructions.

Despite its robustness, the system has inherent limitations. The use of internal behavioral systems can occasionally lead to unintended transitions or the opposite, to a lack of expected transitions, especially in edge cases or ambiguous inputs. Although the system will not fail completely, it may still output inconsistent or suboptimal responses. Additionally, even when provided with precise and explicit instructions, LLMs may sometimes introduce unexpected ”creative” elements, requiring careful prompt engineering to mitigate undesired behavior. The current pipeline flow assumes a turn-based interaction model, limited to one player and one NPC, which constrains more dynamic or multi-agent conversations. Finally, the reliance on multiple AI models makes the system most suitable for more isolated interaction loops – such as chat applications – rather than continuous real-time environments.

5.1 Future Directions

While it currently offers a solid starting point for RPG chat applications, the architecture provides several opportunities for improvement and alternative implementations. For example, the Dialogue Flow Module could benefit from adopting Behavior Trees for a more adaptable yet structured approach, or a Belief-Desire-Intention (BDI) model to simulate highly realistic autonomous behavior. In the Personality Module Hierarchical Task Networks (HTNs) could enable more goal-driven and nuanced character behavior. For the World State Module, alternative data retrieval approaches could be explored, e.g. TF-IDF-based document search or transformer-based semantic similarity.

Furthermore, improvements in pre-processing would help the system better handle vague or ambiguous input. Techniques like named entity recognition could help identify and structure key information, which could then be stored reliably using a form-based subsystem. This would allow facts introduced during the conversation (like player names or affiliations) to be formally

tracked; expanding the system's ability to update its knowledge graph from live dialogue context would also improve long-term consistency. Ultimately, the evaluation and integration of more reliable/domain-specific AI models could benefit the overall system quality.

In terms of new use cases, extending the system with speech recognition and text-to-speech capabilities could enable full voice interaction. It could also be adapted to support predefined input systems, such as classic RPG dialogue choice options, while still using the generation component to provide dynamically styled responses. This would allow designers to enrich background characters and side interactions with minimal additional effort.

6. Conclusions

This thesis introduced and developed a modular system architecture for text-based dialogue in games. A prototype was implemented as a proof of concept, demonstrating the viability of the design [3]. Although the system has some limitations, it achieves a high degree of modularity, allowing developers to substitute and customize modules for specific gameplay needs. Each module plays a focused role within the overall architecture, supporting a balance between narrative control and dynamic, input-driven dialogue.

The process of designing and building the system revealed several challenges, some of which still remain unresolved and indicating possibilities for future work. Many existing NLP methods, while promising in general applications, proved to be not reliable and consistent enough in the stylized and fragmentary context of game dialogues. Inconsistencies in preprocessing and the absence of a form-based fact-tracking subsystem limited the system's ability to fully retain and use contextual information. However, the prototype successfully validated the core design and every obstacle encountered provided a crucial understanding of practical implementation challenges. The modularity of the system is confirmed to be useful in cases requiring both pre-defined structure and adaptive text generation.

Although the current version of the system is not fully production ready, it shows strong potential. With further refinement, particularly in areas like real-time interaction, fact management, and prompt stability, the system could be a valuable tool for modern role-playing games. Incorporating voice functionality would make it well-suited for VR environments, while adapting it for choice-based dialogues could enhance more traditional gaming formats. Beyond games, the architecture could also support regulated yet personalized behavior in domains such as robotics, virtual assistants, or any dialogue-based AI where structure and character are essential.

[3] Source code available at https://github.com/egor123/modular-dialogue-system

References

[1] Buongiorno S., Klinkert L. J., Chawla T., Zhuang Z., and Clark C. PANGeA: Procedural Artificial Narrative using Generative AI for Turn-Based Video Games. 2024. arXiv: 2404.19721 [cs.AI]. https://arxiv.org/abs/2404.19721.

[2] Brusk J. and Björk S. Gameplay Design Patterns for Game Dialogues. *Proceedings of DiGRA 2009 Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Tampere: DiGRA, 2009.

[3] Iovino M., Scukins E., Styrud J., Ögren P., and Smith C. A Survey of Behavior Trees in Robotics and AI. *arXiv preprint arXiv:2005.05842* (2020). https://arxiv.org/abs/2005.05842.

[4] Studiawan R., Hariadi M., and Sumpeno S. Tactical Planning in Space Game using Goal-Oriented Action Planning. *Jurnal Aplikasi Rekayasa Elektrika dan Elektronik (JAREE)* 2.1 (2018). http://jaree.its.ac.id/index.php/jaree/article/view/32.

[5] Georgievski I. and Aiello M. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence* 222 (2015), pp. 124–156. DOI: https://doi.org/10.1016/j.artint.2015.02.002. https://www.sciencedirect.com/science/article/pii/S0004370215000247.

[6] Wadsley T. and Ryan M. A Belief-Desire-Intention Model for Narrative Generation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 9.4 (June 2021), pp. 105–108. DOI: 10.1609/aiide.v9i4.12627. https://ojs.aaai.org/index.php/AIIDE/article/view/12627.

[7] Al-Moslmi T., Gallofré Ocaña M., L. Opdahl A., and Veres C. Named Entity Extraction for Knowledge Graphs: A Literature Overview. *IEEE Access* 8 (2020), pp. 32862–32881. DOI: 10.1109/ACCESS.2020.2973928.

[8] Zhang H., Yu P. S., and Zhang J. A Systematic Survey of Text Summarization: From Statistical Methods to Large Language Models. 2024. arXiv: 2406.11289 [cs.CL]. https://arxiv.org/abs/2406.11289.

[9] Gupta S. and Gupta S. K. Abstractive summarization: An overview of the state of the art. *Expert Systems with Applications* 121 (2019), pp. 49–65. DOI: https://doi.org/10.1016/j.eswa.2018.12.011. https://www.sciencedirect.com/science/article/pii/S0957417418307735.

[10] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., and Polosukhin I. Attention Is All You Need. 2023. arXiv: 1706.03762 [cs.CL]. https://arxiv.org/abs/1706.03762.

[11] Pérez J. M., Giudici J. C., and Luque F. pysentimiento: A Python Toolkit for Sentiment Analysis and SocialNLP tasks. 2021. arXiv: 2106.09462 [cs.CL].

[12] Grattafiori A. et al. The Llama 3 Herd of Models. 2024. arXiv: 2407.21783 [cs.AI]. https://arxiv.org/abs/2407.21783.

[13] Mishra A. and Vishwakarma S. Analysis of TF-IDF Model and its Variant for Document Retrieval. *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*. 2015, pp. 772–776. DOI: 10.1109/CICN.2015.157.

[14] Qadrud-Din J., Rabiou A. B., Walker R., Soni R., Gajek M., Pack G., and Rangaraj A. Transformer Based Language Models for Similar Text Retrieval and Ranking. 2020. arXiv: 2005.04588 [cs.IR]. https://arxiv.org/abs/2005.04588.

[15] Gatto M., Maue J., Mihalák M., and Widmayer P. Shunting for Dummies: An Introductory Algorithmic Survey. Oct. 2009, pp. 310–337. DOI: 10.1007/978-3-642-05465-5_13. https://www.researchgate.net/publication/225576076_Shunting_for_Dummies_An_Introductory_Algorithmic_Survey.

[16] Reimers N. and Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. http://arxiv.org/abs/1908.10084.

[17] Lewis M., Liu Y., Goyal N., Ghazvininejad M., Mohamed A., Levy O., Stoyanov V., and Zettlemoyer L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *CoRR* abs/1910.13461 (2019). arXiv: 1910.13461. http://arxiv.org/abs/1910.13461.

[18] Explosion. en_core_web_sm: spaCy small English model. Version 3.5.0, accessed 2025-05-10. 2023. https://spacy.io/models/en#en_core_web_sm.

License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Egor Lukjanenko,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis *Modular system for text-based interaction with non-player characters in game environments*, supervised by Giacomo Magnifico;
2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;
3. am aware of the fact that the author retains the rights specified in points 1 and 2;
4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Egor Lukjanenko

15/05/2025