Uso de Inteligência Artificial Generativa para Criação de Diálogos e Interações Mais Realistas entre NPCs e Jogadores

Davi Darroit Dutra
Universidade do Vale do Itajaí – Curso de Ciência da Computação, São José - SC
davidutra567@outlook.com

Anita Fernandes
Universidade do Vale do Itajaí – Curso de Ciência da Computação, São José - SC
anita.fernandes@univali.br

Dennis Kerr Coelho
Universidade do Vale do Itajaí – Curso de Ciência da Computação, São José - SC
dennis@univali.br

Ramices dos Santos Silva
Universidade do Vale do Itajaí – Curso de Ciência da Computação, São José - SC
ramices@gmail.com

ABSTRACT

This work proposes the implementation of a dynamic dialogue system between players and NPCs (Non-Playable Characters) using the Gemini 2.5 Pro model from Google AI as the central text generation tool. The objective is to demonstrate how LLMs can be integrated into commercial game engines, such as RPG Maker MZ, to create more natural, contextual, and immersive interactions, without the need for additional training. The proposed system employs hybrid architecture, comprising a JavaScript client plugin and a Node.js backend server, which is responsible for storing conversation context, managing player logins, and protecting the API key. Communication between the game and the model occurs via REST requests, with dialogue history and behavioral instructions sent in real time. The use of prompt engineering ensures narrative consistency and allows for customization of NPC behavior according to their personalities and roles within the story. The system is expected to reduce the limitations of traditional dialogue approaches based on dialog trees or FSMs, providing more expressive and adaptive responses.

KEYWORDS

Generative Artificial Intelligence, Dynamic Dialogue, Non-Playable Characters.

1 Introdução

Em Role Playing Games (RPG), a interação com NPCs (Non-Player Character) costuma ser o principal meio pelo qual o jogador se envolve emocionalmente com o universo do jogo e compreende seus objetivos dentro da experiência lúdica. No entanto, essas interações costumam ser limitadas por abordagens tradicionais de design narrativo. O comportamento e os diálogos dos NPCs geralmente são definidos por sistemas determinísticos, como as FSMs (Finite State Machine) e árvores de diálogo.

As FSMs modelam diferentes estados que um NPC pode assumir, como por exemplo patrulhar, entrar em alerta, atacar ou retornar ao estado inicial, bem como os gatilhos que determinam a transição entre eles [1]. Já as Árvores de Diálogo organizam as falas em fluxos pré-programados, como respostas limitadas e previsíveis. Embora funcionais, essas abordagens comprometem a naturalidade das interações e frequentemente revelam a artificialidade do personagem [2, 3].

Com os avanços no campo da IA Generativa, surge a possibilidade de superar essas limitações. Os LLM (Large Language Models), demonstram capacidade expressiva para compreender a linguagem natural, gerar respostas inéditas e adaptar o diálogo ao contexto [4]. Esses modelos são capazes de processar informações sobre a personalidade do NPC, o histórico de interações, o estado emocional, eventos recentes do jogo e o *lore* (conhecimento interno do mundo fictício). Essa integração de fatores permite criar diálogos mais autênticos e imprevisíveis, que se aproximam da comunicação humana [5,6].

O uso da IA Generativa no design de NPCs não se restringe apenas à geração de texto. O conceito de Engenharia de Prompt torna-se essencial para orientar o modelo de linguagem relativo ao papel do personagem, seu tom de voz, o que ele sabe e como deve se expressar [7]. No entanto, sua aplicação também traz desafios significativos. Entre eles estão a necessidade de curadoria constante, o risco de geração do conteúdo inadequado ou incoerente, e a preocupação com a qualidade autoral da narrativa [8]. Ainda assim, estudos recentes sugerem que os benefícios superam as limitações, quando há um planejamento adequado. Iarovoi [4], por exemplo, investigou como a IA afeta a imersão do jogador, com resultados positivos em termos de engajamento. Da mesma forma, Park et al [6], destacam o potencial dos agentes generativos para simular comportamentos sociais complexos, criando mundos virtuais mais vivos e realistas.

Dentro deste contexto, esse trabalho visa utilizar a IA Generativa como ferramenta central para o desenvolvimento de NPCs mais realistas e dinâmicos em games. A proposta consiste em explorar como LLM podem ser aplicados na criação de diálogos interativos que se adaptem em tempo real ao comportamento do jogador, integrando aspectos como memória, personalidade e consistência narrativa.

Além disso, o estudo busca demonstrar que a combinação entre técnicas de engenharia de prompt, curadoria de conteúdo e integração contextual podem superar as limitações dos sistemas determinísticos tradicionais, oferecendo uma experiência imersiva e próxima da comunicação humana.

2 Abordagem Proposta

A abordagem proposta parte de um plugin desenvolvido para o motor RPG Maker MZ, que se comunica com um servidor backend desenvolvido em Node.js. Esse servidor é responsável por intermediar a comunicação entre o jogo e a API do modelo Gemini 2.5 Pro, da Google AI. Quando o jogador inicia uma conversa com um NPC, o texto é enviado ao servidor via requisição REST. O backend processa a solicitação, aplica técnicas de engenharia de prompt para moldar o comportamento da IA conforme a persona e o papel narrativo do NPC, e então envia o prompt final para o modelo generativo. A resposta retornada pelo Gemini é armazenada junto ao histórico da sessão e enviada de volta ao game para exibição em tempo real. Além disso, o sistema implementa persistência de contexto por jogador, garantindo que cada sessão de jogo seja personalizada e mantenha coerência narrativa mesmo após o encerramento da aplicação. O jogador faz o login no início da partida, permitindo que o backend associe seu histórico de diálogo e variáveis de contexto ao seu perfil. Assim, NPCs podem "lembrar" de interações passadas e adaptar suas respostas em conversas futuras, simulando uma memória de longo prazo. O sistema é dividido e três camadas principais: camada cliente, camada servidor e camada de IA.

A camada cliente é responsável pela interface com o jogador. O plugin intercepta o início das conversas com os NPCs e envia mensagens para o backend por meio de requisições HTTP POST. As respostas da IA são recebidas e exibidas em uma janela de diálogo customizada, preservando o estilo visual do jogo. A camada servidor atua como intermediária entre o jogo e o modelo Gemini. Além de gerenciar as requisições, o servidor é responsável por: (i) armazenar contextos e perfis de jogadores em um banco de dados; (ii) aplicar engenharia de prompt, combinando a persona do NPC, histórico e parâmetros narrativos; (iii) controlar os limites de uso da API e protegera a chave de autenticação; e (iv) registrar os logs de conversas e monitorar desempenho. A camada de IA refere-se ao modelo generativo que interpreta o prompt composto pelo backend e gera as respostas do NPC. O sistema não realiza treinamento adicional (*fine-tuning*). Toda personalização é feita por meio de prompts e contexto persistente. Essa abordagem modular permite flexibilidade, segurança e escalabilidade. O backend pode ser adaptado para diferentes modelos de IA ou ampliado com novas funcionalidades semânticas, personalização por personagens e curadoria automatizada de conteúdo.

O modelo de IA Generativa utilizado neste projeto é o Gemini 2.5 Pro, da Google AI. Essa escola foi motivada por uma combinação de fatores técnicos e práticos, incluindo o acesso gratuito a API, a facilidade de integração com aplicações em JavaScript/Node.js, e a capacidade do modelo de compreender e gerar texto em múltiplos idiomas, incluindo o português. O Gemini é um LLM baseado na arquitetura Transformer [9], projetado para executar tarefas complexas de compreensão, raciocínio e geração textual em tempo real. Sua arquitetura multimodal permite lidar com diferentes tipos de entrada como texto, imagem e código, embora, neste projeto, apenas a modalidade texto seja utilizada.

A integração entre o jogo e o modelo ocorre por meio de requisições REST enviadas pelo plugin do RPG Maker RZ para um servidor de backend desenvolvido em Node.js, que atua como intermediário entre o cliente (jogo) e a API da Google AI. Esse servidor é responsável por: (i) armazenar o contexto das conversas de cada jogador, permitindo que os diálogos mantenham coerência entre sessões distintas; (ii) gerenciar logins e perfis de usuário, vinculando o histórico de interações a cada conta; (iii) proteger a chave de API do Gemini, evitando sua exposição direta no código do jogo; e (iv) implementar a engenharia de prompt, adaptando dinamicamente as instruções enviadas à IA de acordo com o perfil e o papel do NPC. Desta forma, a IA não é treinada localmente, mas orientada por prompts estruturados que definem o comportamento, o estilo de fala e o nível de conhecimento do personagem. Essa abordagem possibilita personalizar a experiência sem necessidade de reprocessamento do modelo, reduzindo custos de complexidade técnica.

O plugin GeminiNPC_Chat.js, desenvolvido em JavaScript tem como objetivo permitir a comunicação direta entre o jogador e os NPCs, dentro do ambiente do RPG Maker MZ, utilizando o modelo Gemini 2.5 Pro da Google AI. Esse módulo representa a principal ponte entre o jogo e a camada de IA Generativa, sendo responsável por capturar as mensagens dos jogadores, enviar essas informações para a IA e exibir as respostas dentro do próprio jogo de forma fluida e natural. A implementação do plugin segue a estrutura padrão do RPG Maker, integrando ao seu sistema de eventos e variáveis globais. A arquitetura foi projetada e modo modular, permitindo futuras expansões e integração com um servidor Node.js, que cuidará da persistência de contexto e da proteção da chave de API. O fluxo básico do sistema proposto ocorre em cinco etapas principais: (i) o jogador interage com o NPC no mapa; (ii) o plugin cria uma janela de diálogo flutuante sobre a tela do jogo; (iii) a mensagem do jogador é capturada e enviada ao modelo Gemini; (iv) a IA processa e envia sua resposta de volta; e (v) o texto é exibido no jogo, e o histórico é armazenado em variáveis internas. O registro dos parâmetros principais ocorre logo no início do código, permitindo a personalização dentro do próprio editor do RPG Maker.

Quando o jogador inicia uma conversa com um NPC, o evento chama as funções que dão início ao fluxo de funcionamento do chat (Figura 1).

```
PluginManager.registerCommand(pluginName, 'startChat', async function (args) {
    this.setWaitMode('geminiChat');
    currentChatArgs = args;

    customChoicePos = {
        x: args.choiceWindowX,
        y: args.choiceWindowY,
        z: args.choiceWindowZ
    };

    createStreamingWindow();
    lockPlayerMovement();
    runChatLoop();
});
```

Figura 1: Fluxo de funcionamento do chat.

Este trecho de código é o ponto de partida de toda a lógica de funcionamento do chat. Ele cria a janela do chat, bloqueia o movimento do jogador e do NPC e chama a função runChatLoop(), que gerencia todo o ciclo de conversa. Dentro do loop, o sistema apresenta as opções de fala ("Say", "Leave"), depois disso ele captura a mensagem digitada pelo jogador via *input dialog*, exibe a

fala no log, e por fim envia o conteúdo à IA por meio da função generateAndStreamResponse().

A interface do chat é criada dinamicamente em HTML, dentro do jogo, garantindo compatibilidade visual com a tela do RPG Maker. O método createStreamingWindow() define a janela de diálogo e o log de mensagens. A cada nova fala, a função addMessageToLog() adiciona o texto à janela de forma incremental, criando uma sensação de conversa contínua. O NPC tem seu movimento bloqueado durante a interação. Quando o chat termina, a função cleanAfterStream() remove a interface e restaura o comportamento original do evento (Figura 2). A engenharia de prompt é crucial para o comportamento do NPC. A função buildApiRequestBody() organiza todos os elementos que serão enviados ao modelo Gemini, incluindo: o histórico de conversas; o papel do NPC e suas instruções comportamentais; o ponto de vista narrativo (POV); exemplo de pergunta e resposta; e o textual atual do jogador (Figura 3).

```
function createStreamingWindow() {
    removeElementById('geminiChatWindow');
    const chatWindow = document.createElement('div');
    chatWindow.id = 'geminiChatWindow';
    Object.assign(chatWindow.style, {
        display: 'flex', flexDirection: 'column',
        position: 'fixed', zIndex: "100",
        left: '0', bottom: '0',
        width: '100%',
        height: params.chatWindowHeight,
        boxSizing: 'border-box', color: 'white', fontSize: '20px',
```

Figura 2: Definição da Interface.

```
function buildApiRequestBody(args, playerInput, interactionType) {
    const historyVarId = Number(args.CustomHistoryVarId) || globalHistoryVarId;
    let history = $gameVariables.value(historyVarId) || [];
    const historySize = Number(args.historySize) * 2;
    if (historySize > 0 && history.length > historySize) history = history.slice(-historySize);
    else if (historySize === 0) history = [];
    const npcName = processCtl(args.characterName) || 'the character';
    const localPOV = args.povOverride === 'Default' ? defaultPOV : args.povOverride;
    let povInstruction = `--- Core Rules ---\n1. YOUR ROLE: You are portraying a character named "${npcName}
    switch(localPOV) {
        case 'First Person': povInstruction += `**Current POV: First Person.**\n- Write from your perspectiv
        case 'Second Person': povInstruction += `**Current POV: Second Person.**\n- You are a narrator talki
        case 'Third Person': povInstruction += `**Current POV: Third Person.**\n- You are a narrator. Descri
    }
```

Figura 3: Função buildApiRequestBody().

A função generateAndStreamResponse() envia a mensagem formatada ao modelo da Google AI por meio de requisições REST. O corpo da requisição inclui um prompt estruturado, enquanto a resposta é recebida e streaming, ou seja, em partes contínuas. É nessa função também, que será inserida a chave API para que aconteça a conexão com a API do Gemini, conforme o avanço do código do projeto, a chave de API será acessada pelo RPG Maker de outra maneira, mais segura, permitindo assim, que a chave não esteja aparente no código. O código processa essas partes com o conteúdo processStream(), reconstruindo a resposta completa e exibindo-a progressivamente na tela (Figura 4).

```
async function generateAndStreamResponse(playerInput, interactionType) {
    if (!currentChatArgs) return;
    const apiKey = 'AIzaSyAaO6I3PnvbUpgGt0lMg7BuF--OzRWyBUY';
    const url = `https://generativelanguage.googleapis.com/v1beta/models/${params['GoogleAI_Model']}:streamGenerateContent?key=${apiKey}`;
    const placeholderId = `gemini-placeholder-` + Date.now();
    addMessageToLog('system', `<i>pensando...</i>`, placeholderId);
    const requestBody = buildApiRequestBody(currentChatArgs, playerInput, interactionType);
    try {
        const response = await fetch(url, { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(requestBody) });
        if (!response.ok) { throw new Error(`API Error ${response.status}: ${await response.text()}`); }
        const fullResponseText = await processStream(response.body, placeholderId);
        saveToVariables(currentChatArgs, playerInput, fullResponseText);
    } catch (error) {
        console.error('Google AI Fetch Error:', error);
        updateMessageInLog(placeholderId, 'ai', `<span style="color:orange;">${formatError(error)}</span>`);
    } finally {
        showContinueButton();
    }
}
```

Figura 4: Comunicação com a API.

3 Considerações Preliminares

Esta pesquisa está em desenvolvimento, sendo a primeira etapa focada em gerar funções para desenvolvimento dos diálogos. As próximas etapas do projeto estarão concentradas na evolução do protótipo desenvolvido, com foco em aprimorar tanto a infraestrutura técnica quanto os aspectos narrativos e interativos.

As próximas etapas do projeto focarão em: (i) implementação completa do servidor backend em Node.js, com rotas REST para login, salvamento de contexto e envio de mensagens; (ii) integração do sistema com um banco de dados MongoDB, para armazenamento persistente dos históricos de conversa e perfis de jogador; e (iii) refinamento da engenharia de prompt, incluindo instruções de persona, contexto narrativo e variáveis dinâmicas de comportamento dos NPCs.

Também será conduzida a fase de testes e validação do sistema. Nessa etapa, serão realizados: testes de desempenho e latência de resposta do modelo Gemini; avaliações de coerência narrativa e consistência de contexto; e testes com usuários para avaliar imersão, naturalidade e engajamento nas interações com NPCs.

REFERÊNCIAS

[1] Felipe Rolim, 2023. *LLMs e IA Generativa em Jogos: Uma Análise de Técnicas e Aplicações*. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação), Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2023. Acessado em 23 de setembro de 2025. Disponível em: https://pantheon.ufrj.br/bitstream/11422/22275/3/FMRolim.pdf.

[2] Pietro S. Ghiringhelli, 2023. *O uso de Inteligências Artificiais Generativas em Jogos Digitais*. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação), Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023. Acessado em 30 de setembro de 2025. Disponível em: https://www.linux.ime.usp.br/~ghiringhelli/mac0499/monografia.pdf.

[3] Mit Technology Review Brasil, 2024. *Como a IA generativa pode reinventar o que significa jogar*. MIT Technology Review Brasil, 29 jul. 2024. Acessado em 05 de outubro de 2025. Disponível em: https://mittechreview.com.br/ia-generativa-e-jogos/.

[4] Dimitri Iarovoi, Richard Hebblewhite, Phoey Lee Teh, 2024. AI's Influence on Non-Player Character Dialogue and Gameplay Experience. In: Arai, K. (eds) Intelligent Computing. SAI 2024. *Lecture Notes in Networks and Systems*, vol 1016. Springer, Cham. DOI: https://doi.org/10.1007/978-3-031-62281-6_6

[5] Steph Buongiorno; Lawrence J. Klinkert; Tanishq Chawla; Zixin Zhuang; Corey Clark. 2024. PANGeA: Procedural Artificial Narrative using Generative AI for Turn-Based Video Games. *ArXiv preprint* arXiv:2404.19721. Acessado em 04 de outubro de 2025. Disponível em: https://arxiv.org/pdf/2404.19721

[6] Joon Sung Park; Joseph C. O´Brien; Carrie J. Cai; Meredith R. Morris; Percy Liang; Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. *ArXiv preprint* arXiv:2304.03442, 2023. Acessado em 04 de outubro de 2025. Disponível em: https://arxiv.org/pdf/2304.03442.

[7] Xavier Amatriain. 2024. Prompt Design and Engineering: Introduction and Advanced Methods. 2024. *ArXiv*. https://arxiv.org/abs/2401.14423.

[8] Unite.AI Além dos scripts: o futuro dos NPCs de videogame com IA generativa. *Unite.AI*, 2025. Acessado em 09 de agosto de 2025. Disponível em: https://www.unite.ai/pt/beyond-scripts-the-future-ofvideo-game-npcs-with-generative-ai

[9] Tianyang Lin, Yuxin Wang, Xiangyang Liu, Xipeng Qiu, 2022. A survey of transformers. AI Open, Volume 3, 2022, Pages 111-132. DOI: https://doi.org/10.1016/j.aiopen.2022.10.001