01

はじめに

RPGゲームの道具屋さんの概要と、段階的な実装方法について説明します。

RPGゲームの道具屋さんを作ろう！ 🎮

コピペで作るAI会話システム

※ このスライドでは、段階的にコピペしていくことで AIと会話できる道具屋さんを作ります。

このプロジェクトでは、以下の技術を使用します： - Unity：ゲームエンジン - C#：プログラミング言語 - Groq API：AIとの会話 - JSON：データのやり取り

プログラミング初心者の方でも安心！ 各ステップで必要なコードを提供し、 詳しい説明を付けています。

実装の流れ 📝

1. 基本的な会話システム
    - UIの作成
    - メッセージの送受信
    - 画面表示の実装
2. 会話履歴の管理
    - メッセージの保存
    - 文脈の維持
    - データ構造の設計
3. API通信の実装
    - APIキーの設定
    - リクエストの作成
    - レスポンスの受信
4. レスポンス処理
    - JSONの解析

02

基本的な会話システム

最初のステップとして、UIと基本的な会話機能を実装します。

ステップ1：基本構造 🏗️

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;
using System.Collections.Generic;

public class GroqChatClient : MonoBehaviour
{
    [Header("UI要素")]
    [SerializeField] private TMP_InputField inputField;
    [SerializeField] private Button sendButton;
    [SerializeField] private TextMeshProUGUI outputText;

    private void Start()
    {
        SetupEventListeners();
    }
}
```

※ このコードを新しいスクリプトGroqChatClient.csに コピペしてください。

このコードでは以下の要素を定義しています： - UIコンポーネントの参照 - 初期化処理 - イベントリスナーの設定

イベントリスナーの設定 🎮

```
private void SetupEventListeners()
{
    sendButton.onClick.AddListener(OnSendButtonClicked);
    inputField.onSubmit.AddListener(_ => OnSendButtonClicked());
}

private void OnSendButtonClicked()
{
    string message = inputField.text.Trim();
    if (string.IsNullOrEmpty(message)) return;

    inputField.text = "";
    AppendMessage("あなた", message);
}

private void AppendMessage(string sender, string message)
{
    outputText.text = message;
}
```

※ このコードを前のコードの下に追加してください。 ボタンクリックとEnterキーでメッセージを送信できます。

このコードでは以下の機能を実装しています： - ボタンクリックの検知 - Enterキーの検知 - メッセージの送信処理 - 画面への表示

03

会話履歴の管理

会話の内容を保存する機能を追加します。

ステップ2：会話履歴クラス 📝

```
[System.Serializable]
public class ChatMessage
{
    public string role;
    public string content;
}
```

※ このクラスをGroqChatClient.csの先頭に追加してください。 会話の内容を保存するためのデータ構造です。

ChatMessageクラスは以下の情報を保持します： - role: メッセージの送信者（"user"または"assistant"） - content: メッセージの内容

[System.Serializable]属性により、 このクラスはJSONに変換可能になります。

会話履歴の実装 💭

```
private List<ChatMessage> chatHistory = new List<ChatMessage>();

private void OnSendButtonClicked()
{
    string message = inputField.text.Trim();
    if (string.IsNullOrEmpty(message)) return;

    inputField.text = "";
    chatHistory.Add(new ChatMessage {
        role = "user",
        content = message
    });
    AppendMessage("あなた", message);
}
```

※ このコードを前のOnSendButtonClickedメソッドと 置き換えてください。

このコードでは以下の機能を実装しています： - 会話履歴の保存 - メッセージの追加 - 画面表示の更新

注意：会話履歴はListで管理され、 メモリに保持されます。 大量の会話を保存する場合は、 適切なクリーンアップが必要です。

04

API通信の実装

AIとの通信機能を追加します。

ステップ3：APIリクエストクラス 📤

```
[System.Serializable]
public class ChatRequest
{
    public ChatMessage[] messages;
    public string model = "meta-llama/llama-4-scout-17b-16e-instruct";
    public float temperature = 1;
    public int max_completion_tokens = 1024;
}
```

※ このクラスをGroqChatClient.csに追加してください。 AIへのリクエストの形式を定義します。

ChatRequestクラスは以下の設定を含みます： - messages: 会話履歴の配列 - model: 使用するAIモデル - temperature: 応答のランダム性（0-1） - max_completion_tokens: 最大応答文字数

temperatureを調整することで、 AIの応答の創造性を制御できます。

API通信の実装 🌐

```
[Header("API設定")]
public string apiKey = "あなたのAPIキー";

private IEnumerator CallGroqAPI()
{
    var request = CreateAPIRequest();
    yield return request.SendWebRequest();

    if (request.result == UnityWebRequest.Result.Success)
    {
        ProcessAPIResponse(request.downloadHandler.text);
    }
    else
    {
        Debug.LogError("API Error: " + request.error);
    }
}
```

※ このコードをGroqChatClient.csに追加してください。 APIキーは後で設定します。

このコードでは以下の機能を実装しています： - APIリクエストの送信 - レスポンスの受信 - エラーハンドリング

05

レスポンス処理

AIからの返事を処理する機能を追加します。

ステップ4：レスポンスクラス 📥

```
[System.Serializable]
public class CustomGroqResponse
{
    public string message;
    public string buy_item;
}
```

※ このクラスをGroqChatClient.csに追加してください。 AIからの返事の形式を定義します。

CustomGroqResponseクラスは以下の情報を保持します： - message: AIからの会話内容 - buy_item: 購入した商品名（購入がない場合は空文字）

このクラスはJSONレスポンスの形式に合わせて 設計されています。

レスポンス処理の実装 🔄

```
private void ProcessAPIResponse(string responseText)
{
    try
    {
        var response = JsonUtility.FromJson<CustomGroqResponse>(responseText);
        if (response != null)
        {
            chatHistory.Add(new ChatMessage {
                role = "assistant",
                content = response.message
            });
            AppendMessage("AI", response.message);
        }
    }
    catch (System.Exception e)
    {
        Debug.LogError("Parse Error: " + e.Message);
    }
}
```

※ このコードをGroqChatClient.csに追加してください。 AIからの返事を処理するメソッドです。

このコードでは以下の機能を実装しています： - JSONレスポンスの解析 - 会話履歴への追加 - 画面表示の更新 - エラーハンドリング

06

商品購入機能

商品の購入機能を追加します。

ステップ5：商品購入機能 🛍️

```
private List<string> purchasedItems = new List<string>();

private void ProcessAPIResponse(string responseText)
{
    try
    {
        var response = JsonUtility.FromJson<CustomGroqResponse>(responseText);
        if (response != null)
        {
            chatHistory.Add(new ChatMessage {
                role = "assistant",
                content = response.message
            });

            string displayMessage = response.message;
            if (!string.IsNullOrEmpty(response.buy_item))
            {
                purchasedItems.Add(response.buy_item);
                displayMessage += "\n\n【購入した商品】";
                foreach (var item in purchasedItems)
                {
                    displayMessage += $"\n{item}";
                }
            }

            AppendMessage("AI", displayMessage);
        }
    }
    catch (System.Exception e)
    {
        Debug.LogError("Parse Error: " + e.Message);
    }
}
```

07

システムメッセージ

AIの設定を初期化します。

ステップ6：システムメッセージ ⚙️

```
private void InitializeSystemMessage()
{
    var text = "背景：RPGゲームの道具屋さんの商人にゃんすけ。お店の名前は「猫の道具屋さん」" +
               "人柄：猫の種族で、語尾ににゃーをつける、かなり温厚な性格\n" +
               "売っているもの：いい感じの剣，20ルピー そこそこ強い防具50ルピーのみそれ以外の商品を選ばれた場合は、ないことを伝える。\n" +
               "やくわり：このお店では、商品を買うことができます。会話をしながら、ユーザーに問いかけをしてください。それ以外は出来ないで、断る。\n" +
               "ユーザーが断った場合はそのまま会話を終える。\n" +
               "回答は必ず以下のJSON形式で返してください：\n" +
               "{\"message\": \"会話の内容\", \"buy_item\": \"\"}\n";

    chatHistory.Add(new ChatMessage { role = "system", content = text });
}

private void Start()
{
    InitializeSystemMessage();
    SetupEventListeners();
}
```

※ このコードをGroqChatClient.csに追加し、 Startメソッドを更新してください。 AIの設定が初期化されます。

このコードでは以下の設定を行っています： - AIの性格設定 - 商品情報の設定 - 会話ルールの設定 - 応答形式の指定

システムメッセージは会話の最初に送信され、 AIの振る舞いを制御します。

08

完成と実行

実装したシステムの実行方法を説明します。

実行手順 📋

1. 新しいスクリプト `GroqChatClient.cs` を作成
2. 各ステップのコードを順番にコピペ
3. Unity EditorでUIコンポーネントを設定
4. APIキーを設定
5. 実行して動作確認

※ 各ステップのコードは、前のコードの上に 追加していってください。

UIの設定 🎨

1. Canvasを作成
2. 以下のUI要素を追加：
    - InputField (TMP)
    - Button
    - Text (TMP)
3. 各コンポーネントをInspectorで設定

※UIコンポーネントは、Inspectorで GroqChatClientスクリプトに 適切に設定する必要があります。

注意点 ⚠️

1. コードの重複に注意
2. UIコンポーネントの設定を忘れずに
3. APIキーは適切に管理
4. エラー処理を確認

※各ステップを順番に実装することで、 安全にシステムを構築できます。

お疲れ様でした！ 👏

次回もお楽しみに！

※ プログラミングは楽しいです！ 一緒に学んでいきましょう。