01

はじめに

RPGゲームの道具屋さんの概要や、この教材で学ぶ目標について紹介します。

RPGゲームの道具屋さんを作ろう！ 🎮

UnityでAIと会話するゲームを作る

※ このスライドでは、AIと会話できる道具屋さんを作ります。

今日の目標 🎯

1. AIと会話できる道具屋さんを作る
2. 商品を買ったり、会話したりできる
3. プログラミングの基本を学ぶ
4. C#の基本的な構文を理解する
5. AIとの通信方法を学ぶ
6. JSONデータの処理方法を学ぶ

※ これらの目標を達成することで、AIを使ったゲームの作り方を学びます。

道具屋さんってどんなの？ 🏪

- にゃんすけという猫の商人
- 剣や防具を売っている
- お客さんと会話しながら商品を紹介

※ にゃんすけは猫の種族で、語尾に「にゃ」をつけて話します。 温厚な性格で、お客さんとの会話を楽しみます。

02

プログラムの基本

道具屋さんの会話や仕組み、JSONの扱い方など、プログラムの基礎を学びます。

プログラムの仕組み 🔄

1. プレイヤーがメッセージを送る
2. AIが返事を考える
3. 画面に表示する

※ この流れは、実際の会話と同じように、メッセージのやり取りを繰り返します。 AIは会話の文脈を理解して、適切な返事を返してくれます。

JSONパースの仕方（1/2）🔍

1. 基本的なパース

```
// JSON文字列をデータクラスに変換
var response = JsonUtility.FromJson<CustomGroqResponse>(jsonText);

// データを使用
string message = response.message;
string item = response.buy_item;
```

※ JsonUtility.FromJsonは、JSON文字列を指定したデータクラスのインスタンスに変換します。 変換に失敗した場合はnullが返されます。

JSONパースの仕方（2/2）🔍

2. エラー処理付きのパース

```
try {
    // JSONをパース
    var response = JsonUtility.FromJson<CustomGroqResponse>(jsonText);

    // データの存在確認
    if (response != null && !string.IsNullOrEmpty(response.message)) {
        // 正常なデータの場合の処理
        AppendMessage("AI", response.message);
    } else {
        // データが不正な場合の処理
        Debug.LogError("不正なJSONデータです");
        AppendMessage("システム", "AIからの返事が正しくありません");
    }
} catch (Exception e) {
    // パースに失敗した場合の処理
    Debug.LogError("JSONパースエラー: " + e.Message);
    AppendMessage("システム", "AIの返事を理解できませんでした");
}
```

※ try-catchでエラーを処理し、データの存在確認も行うことで、 安全にJSONデータを扱うことができます。

03

JSONデータの構造

AIとのやりとりに使うJSONデータの形や、データクラスの定義方法を解説します。

JSONデータの構造（1/2）🏗️

1. データクラスの定義

```
[System.Serializable]
public class CustomGroqResponse {
    public string message;    // 会話の内容
    public string buy_item;   // 購入した商品
}
```

※ [System.Serializable]は、JSONに変換できることを示す印です。 UnityのJsonUtilityは、この印がついたクラスだけを変換できます。

JSONデータの構造（2/2）🏗️

2. データクラスの使い方

```
// データクラスのインスタンスを作成
var response = new CustomGroqResponse {
    message = "にゃー！いらっしゃい！",
    buy_item = ""
};

// データクラスをJSONに変換
string jsonText = JsonUtility.ToJson(response);

// JSONをデータクラスに変換
var parsedResponse = JsonUtility.FromJson<CustomGroqResponse>(jsonText);
```

※ データクラスを使うことで、JSONデータを簡単に扱えます。 ToJsonでJSONに変換、FromJsonでJSONからデータクラスに変換できます。

04

AIとの通信

Groq APIを使ってAIと会話する方法や、リクエスト・レスポンスの流れを学びます。

AIからの返事の形式 📝

JSONとは？

- データを整理して保存する形式
- キーと値のペアで情報を表現
- プログラムで扱いやすい形式

例：AIからの返事

```
{
    "message": "にゃー！いらっしゃい！",
    "buy_item": ""
}
```

※ message: 会話の内容 ※ buy_item: 購入した商品（購入していない場合は空文字）

AIとの通信方法 🌐

1. APIキーの取得

```
public string apiKey = "あなたのAPIキー";
```

※ APIキーは、AIサービスを使うためのパスワードのようなものです。 Groqのウェブサイトで取得できます。

05

データ処理

AIからの返事の受け取り方や、会話履歴・購入処理の実装方法を説明します。

リクエストの作成 📤

2. リクエストの作成

```
var data = new ChatRequest {
    messages = chatHistory.ToArray(),
    model = "meta-llama/llama-4-scout-17b-16e-instruct",
    temperature = 1,
    max_completion_tokens = 1024
};
```

※ リクエストには、会話履歴やAIの設定が含まれます。 temperatureは応答のランダム性を、max_completion_tokensは返事の最大文字数を設定します。

AIからの返事の処理（1/2） 📥

1. 返事の受け取り

```
// AIからの返事を文字列で受け取る
string responseText = request.downloadHandler.text;
Debug.Log("AI Response: " + responseText);
```

2. JSONの解析（前半）

```
try {
    // JSONを解析してデータクラスに変換
    var response = JsonUtility.FromJson<CustomGroqResponse>(responseText);

    // 会話内容を表示
    AppendMessage("AI", response.message);
} catch (Exception e) {
    // 解析に失敗した場合
    Debug.LogError("JSON解析エラー: " + e.Message);
    AppendMessage("システム", "AIの返事を理解できませんでした");
}
```

※try-catchでエラーを処理することで、JSONの形式が正しくない場合でもプログラムが停止しません。

AIからの返事の処理（1/2） 📥

2. JSONの解析（後半）

```
} catch (Exception e) {
    // 解析に失敗した場合
    Debug.LogError("JSON解析エラー: " + e.Message);
    AppendMessage("システム", "AIの返事を理解できませんでした");
}
```

※ エラーが発生した場合は、ユーザーに分かりやすいメッセージを表示します。

AIからの返事の処理（2/2）🔄

3. 商品購入の処理

```
// 商品を購入した場合
if (!string.IsNullOrEmpty(response.buy_item)) {
    // 購入リストに追加
    purchasedItems.Add(response.buy_item);

    // メッセージを表示
    AppendMessage("システム",
        response.buy_item + "を購入しました！");
}
```

※ 商品の購入は、AIの返事の中に含まれる 「buy_item」 の値で判断します。

会話履歴の更新 💭

4. 会話履歴の更新

```
// 会話履歴に追加
chatHistory.Add(new ChatMessage {
    role = "assistant",
    content = response.message
});
```

※ 会話履歴を保存することで、AIは前後の文脈を理解できます。 これにより、より自然な会話が可能になります。

会話の例 💭

プレイヤー「こんにちは！」

AIからの返事

```
{
    "message": "にゃー！いらっしゃい！今日は何がお探しにゃ？",
    "buy_item": ""
}
```

※ 会話の流れに応じて、AIは適切なJSONデータを返します。

商品の購入 🛍️

商品リスト

- いい感じの剣：20ルピー
- そこそこ強い防具：50ルピー

購入処理

```
if (!string.IsNullOrEmpty(response.buy_item)) {
    purchasedItems.Add(response.buy_item);
    AppendMessage("システム",
        response.buy_item + "を購入しました！");
}
```

※ 商品の購入は、AIの返事の中に含まれる 「buy_item」 の値で判断します。

06

エラー処理とデバッグ

エラーが起きたときの対処法や、デバッグの基本について学びます。

エラー処理 🚨

1. JSON解析エラー

```
try {
    var response = JsonUtility.FromJson<CustomGroqResponse>(jsonText);
} catch (Exception e) {
    Debug.LogError("JSON解析エラー: " + e.Message);
    AppendMessage("システム", "エラーが発生しました");
}
```

※ エラーが発生した場合でも、 ユーザーに分かりやすいメッセージを表示します。

データ検証 ✅

2. データ検証

```
if (response == null) {
    Debug.LogError("レスポンスがnullです");
    AppendMessage("システム", "AIからの返事が正しくありません");
}
```

※ データの存在確認をすることで、 予期しない形式のJSONが来ても安全に処理できます。

デバッグの方法 🔍

1. ログ出力

```
Debug.Log("AI Response: " + responseText);
Debug.LogError("Error: " + errorMessage);
Debug.LogWarning("Warning: " + warningMessage);
```

2. 条件分岐のデバッグ

```
if (response != null) {
    Debug.Log("レスポンス成功: " + response.message);
} else {
    Debug.Log("レスポンス失敗: null");
}
```

※ デバッグログは、Unityのコンソールウィンドウで 確認できます。問題が発生した時の原因特定に役立ちます。

07

まとめと課題

学んだ内容の振り返りと、チャレンジ課題・参考資料を紹介します。

まとめ 📚

1. Unityでゲームを作る
2. C#の基本的な構文を理解する
3. AIとの通信方法を学ぶ
4. JSONデータの処理方法を学ぶ
5. 会話の文脈を管理する
6. 商品の購入処理を実装する
7. エラー処理を実装する

※ このプログラムは、AIとの会話を通じて ゲームの世界をより豊かにします。 JSONデータの処理は、AIとの通信の重要な部分です。

チャレンジ課題 🎯

1. 新しい商品を追加する
2. 値段を変更する
3. 会話のパターンを増やす
4. 新しい機能を追加する
5. エラー処理を改善する
6. JSONの形式を拡張する

※チャレンジ課題に取り組むことで、プログラミングの理解を深めることができます。

参考資料 📖

- Unity公式サイト
- Groq APIドキュメント
- C#プログラミング入門
- JSONの書き方
- 非同期処理の基礎
- AIとの会話設計ガイド
- JSONパースのベストプラクティス

※ これらの資料を参考にすることで、 より高度な機能を実装できます。

質問タイム！ ❓

何か分からないことはありますか？

※質問は大歓迎です！ 分からないことをそのままにしないでください。

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。