

# DIAGEN for Unity

*High-Quality Dialogue Generation Tool*











This tool is designed to craft dynamic and immersive dialogues that reflect the NPCs' knowledge and emotions. It supports three key use cases:

- **Player ↔ NPC Interaction:** Create engaging and responsive conversations between players and NPCs.
- **NPC ↔ NPC Interaction:** Develop authentic and contextually rich exchanges between NPCs.
- **Environment ↔ NPC Interaction:** Generate dialogues that respond to environmental cues or changes, enhancing world immersion.

## Notes :

⚠ This document will show you how to use the tool on the App and how to implement it in Unreal (version 5.2 or later) or Unity.

<b>1. Explication about LLM (large language models)</b>	<b>2</b>
1. Temperature : Controls the randomness or creativity of the model's responses. Values range from 0 to 2.	4
2. Length (Max Tokens) : Determines the maximum number of tokens (words, subwords, or characters depending on the tokenizer) the model can generate in a single response. Values go from 0 to 2048.	5
3. Toxicity filtering : Measures and moderates the likelihood of generating harmful, offensive, or inappropriate content. Values go from 0 to 1.	5
<b>2. Creating and testing content in the app</b>	<b>5</b>
<b>3. Diagen Files and testing in the app</b>	<b>8</b>
3.0. Explanation of State Tags	9
3.1. Character Information (CI)	10
3.2. Topic Detection	10
3.3. Diagen Event Files	11
3.4. State Tags Weight	12
<b>4. Editor mode in Unity</b>	<b>12</b>
Using the LLM Tab in Diagen on Unity	17
Using the Event Generation Tab in Diagen on Unity	18
Using the Topic Detection Tab in Diagen on Unity	20
Exporting Dialog	22
<b>5. Available functions in Unity</b>	<b>23</b>
 DiagenCommonTypes.cs — Type & Method Table	23
 DiagenSubsystem.cs — Function Table	26
 DiagenSessionLibrary.cs — Function Table	28
 DiagenStateTagsLibrary.cs — Function Table	31
 DiagenLlmLibrary.cs	33
 DiagenTriggerLibrary.cs — Function Table	36
 ActionEvent.cs — Function Table	38
 DiagenTopicLibrary.cs — Function Table	38
<b>6. Flower Island : Exemple to understand Diagen in Unity</b>	<b>40</b>
<b>6.1. Exemple : Start Diagen</b>	<b>40</b>
6.2. Exemple : Triggers and Events	42
6.3. Exemple : Conversation and Character Information Table	46
Field Definitions:	47
6.4. Exemple : Topic detection et Trust.	49
<b>7. Contact us if you need help</b>	<b>51</b>

## 1. Explication about LLM (large language models)

A **Large Language Model (LLM)** is an AI trained on vast amounts of text to understand and generate contextually relevant content. To generate dialogue, define a clear context (*core\_description*), provide an input (question or instruction), and adjust parameters like temperature to control creativity and coherence. This enables dynamic interactions between players, NPCs, and the environment.

To generate a sentence, a character requires a **description** (which can be more or less detailed) and an **input**. This input can be a **question** or an **instruction**.

### 1. Description:

The description must be written in the second person singular, starting with "You are..." and providing descriptive details about the character. The description can be up to 1024 tokens (roughly words) in total, and can consist of several blocks of information.

Example of a description:

*You are Abrogail, the ruthless and cunning Empress of Cheliax, feared across the realms. You speak with sharp, biting cynicis.*

### 2. Input

There are two types of inputs: questions and instructions.

- Questions are directed at the NPC during dialogues between a player and an NPC or between two NPCs. These serve as prompts that the model must respond to directly.  
Example of a **question**: *Hi! I'm Tom. How long have you been Queen?*

- Instructions, on the other hand, differ from questions as they are commands given to the model itself. The model uses these instructions to generate appropriate responses or actions. This type of input is ideal for reacting to in-world events, such as barks, player-triggered events, or other dynamic occurrences.

Example of an **instruction**: *You say that you demand this person to identify herself.*

### 3. Training dataset (optional)

When you have a well-defined world with detailed lore and specific information, fine-tuning your model on your custom data is highly beneficial. Fine-tuning involves training the model further using your world-specific datasets, which may include lore, character descriptions, historical events, or even dialogue patterns unique to your setting. This process enhances the model's ability to generate responses that are not only accurate but also contextually rich and aligned with the tone, style, and details of your world.

For instance, in a fantasy setting, you can fine-tune the model to understand and respond to questions about the history of a kingdom, the motivations of a guild, or the personality of specific NPCs. NPC explanations, including their backstory, personality traits, and speech style, can be incorporated into the fine-tuning dataset. This allows the model to respond in character consistently and convincingly, whether the interaction involves the player, other NPCs, or environmental events.

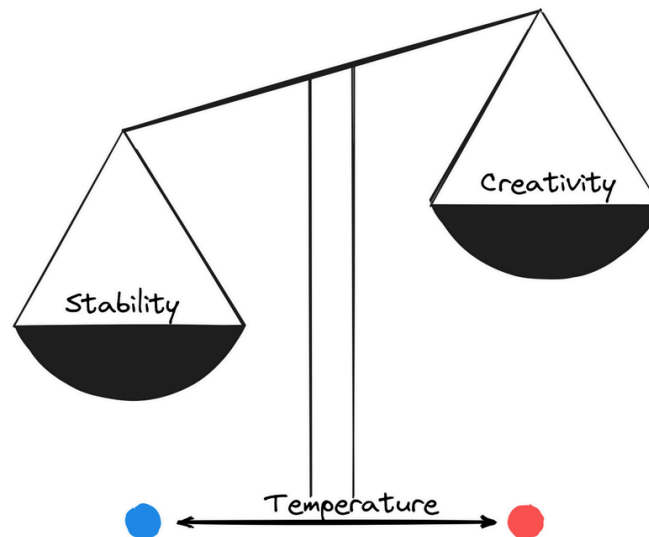
Fine-tuning ensures that the model reflects the unique aspects of your world, making interactions feel authentic and immersive for players. This is especially useful for creating dynamic, lore-driven experiences where NPCs react appropriately to complex narratives and player choices.

**The finetune function is not currently available in the Diagen tool, but is available by contacting the x&immersion team at [contact@xandimmersion.com](mailto:contact@xandimmersion.com)**

### 4. Parameters : Temperature, Length and Toxicity filtering

**1. Temperature : Controls the randomness or creativity of the model's responses. Values range from 0 to 2.**

- **Low Values (e.g., 0.2):** Responses are more deterministic, focused, and predictable, adhering closely to the most probable outcomes. Ideal for tasks requiring accuracy or consistency, such as technical explanations or straightforward answers.
- **High Values (e.g., 0.8):** Responses become more diverse and creative, introducing variation and novelty. Useful for generating imaginative content, storytelling, or exploring alternative ideas.



**2. Length (Max Tokens) :** Determines the maximum number of tokens (words, subwords, or characters depending on the tokenizer) the model can generate in a single response. Values go from 0 to 2048.

- **Short Lengths:** Suitable for concise answers or brief prompts.
- **Long Lengths:** Allows for more detailed and elaborate responses, such as long-form content, complex narratives, or extended conversations. Note that longer outputs may increase computation time and resource usage.

**3. Toxicity filtering :** Measures and moderates the likelihood of generating harmful, offensive, or inappropriate content. Values go from 0 to 1.

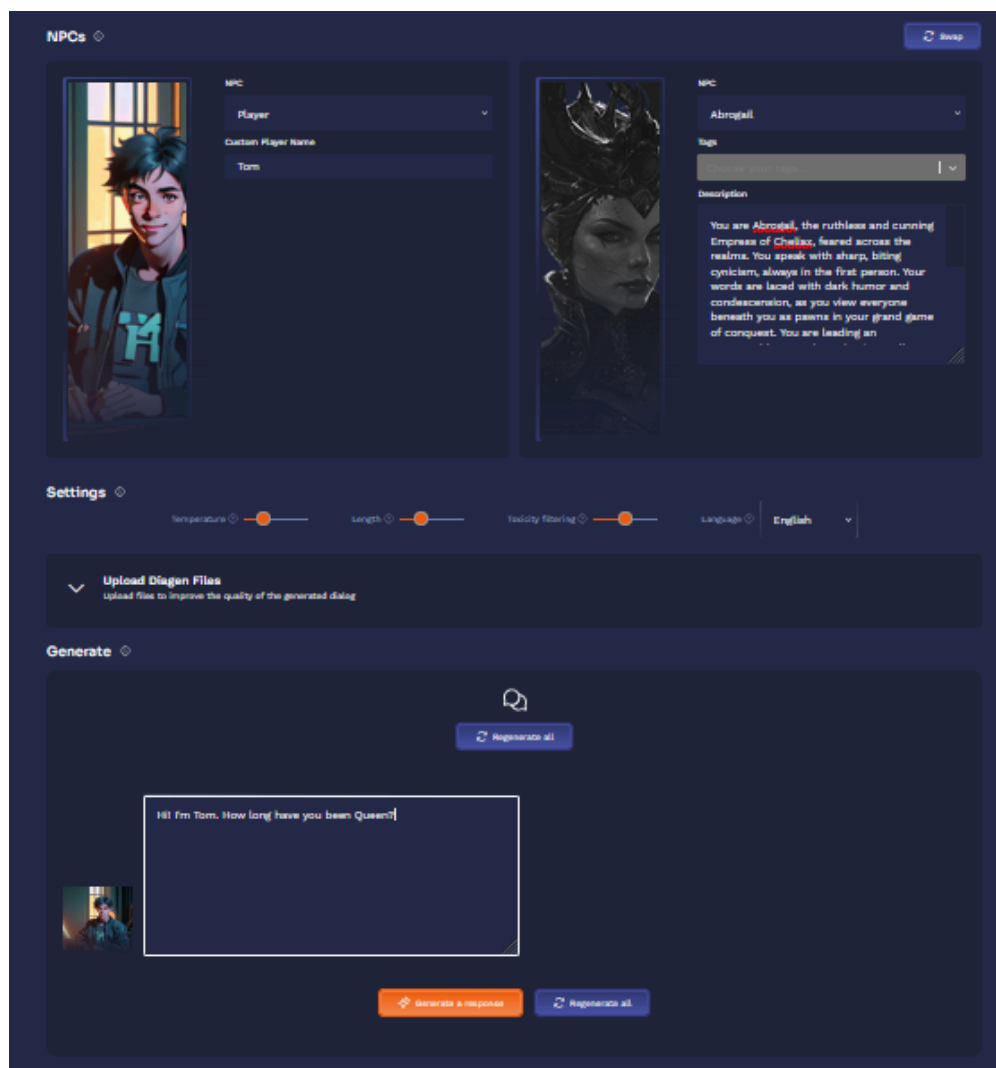
- **Low Toxicity Levels:** May occur when handling controversial or ambiguous prompts, especially without safeguards. Tools like content filters or external moderation models can be applied to minimize this risk.
- **Higher Risk of Toxicity:** Ensures safer and more neutral outputs, often achieved through fine-tuning or using moderation layers.

## 2. Creating and testing content in the app

By visiting the app at <https://create.xandimmersion.com/diagen> you can create an account to access the "Diagen" page. This web-based version is designed for creators (writers, narrative designers, developers, etc.) to test their content in an intuitive and user-friendly environment.

The goal is to allow you to evaluate your results before integrating them into a game engine, saving valuable development time. The same features are available in the app as in the Unreal Engine integration (Unity support is coming soon).

### 2.1. Getting Started : Player → NPC.

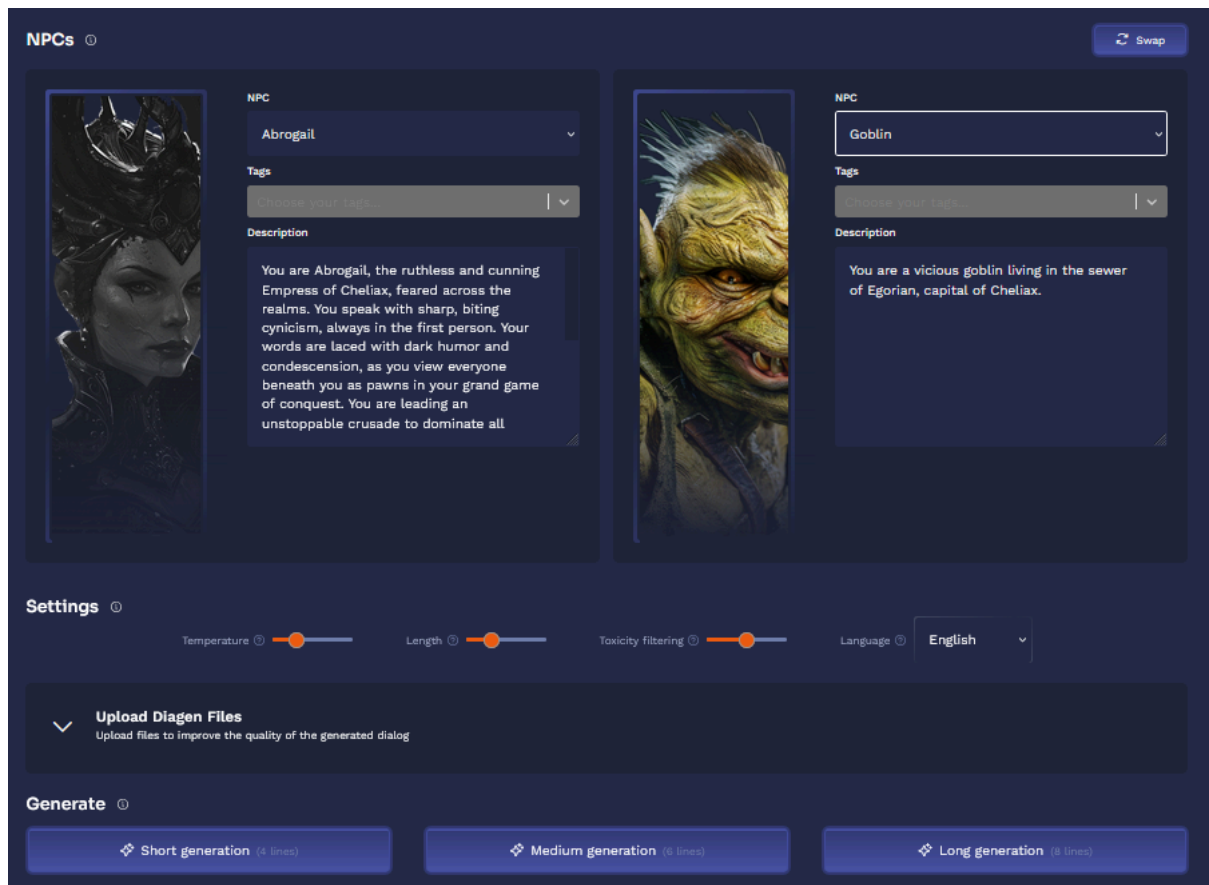


- **Select the NPC "Player":** Choose an NPC to engage with
- **Name your player:** Assign a name to your player character.
- **Write your question:** Enter a question and click "Generate a response". The model will generate a reply based on the NPC's description and other parameters.
- **Continue the conversation:** Engage in a turn-by-turn dialogue with the NPC. The model retains the conversation history, ensuring responses remain consistent and contextually relevant. If you want to refine a response, click the "Regenerate" button (double arrow icon).

You can edit the NPC's description to include specific details or lore and observe how these changes affect the dialogue. This allows you to fine-tune interactions to better match your world and storytelling goals.

With these tools, creators can iteratively refine their NPCs, test interactions, and ensure dialogues are engaging, immersive, and tailored to their vision—whether in the app or directly in a game engine.

## **2.2. Getting Started : NPC→ NPC.**

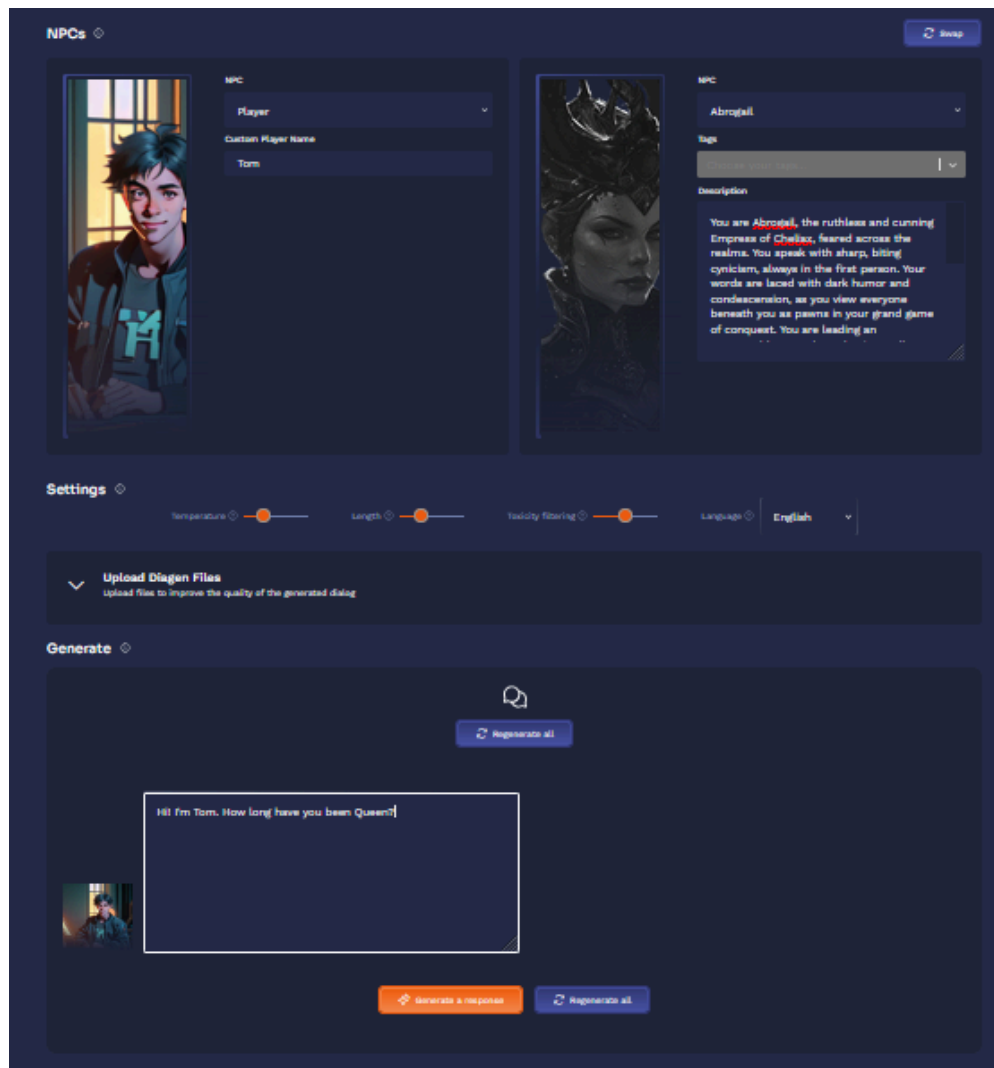


Very similar to *Player* → *NPC*, this use case allows you to create a dialogue between two NPCs.

- Write the descriptions of both characters, including their personalities, roles, or any relevant traits. Add context for the discussion to help the model generate engaging dialogue (e.g., the setting of the exchange, the characters' motivations, or the stakes involved).
- Click on **short**, **medium**, or **long** generation, depending on the number of dialogue turns you want.
- If needed, regenerate specific responses by clicking the **double arrow** icon.

### 2.3. Getting Started : Environment → NPC.





To trigger an event, simply select the **Player** as the NPC, then check the "**Instruction**" box instead of "Question" and enter your instruction.

You can customize the NPC's description to better align with the instruction, ensuring more relevant responses. Once done, generate a response and, if needed, adjust the **temperature** to explore different variations of the output. Lower temperatures will yield more focused and predictable results, while higher temperatures will introduce creativity and unpredictability. Regenerate as needed to refine the outcome and achieve the desired effect.

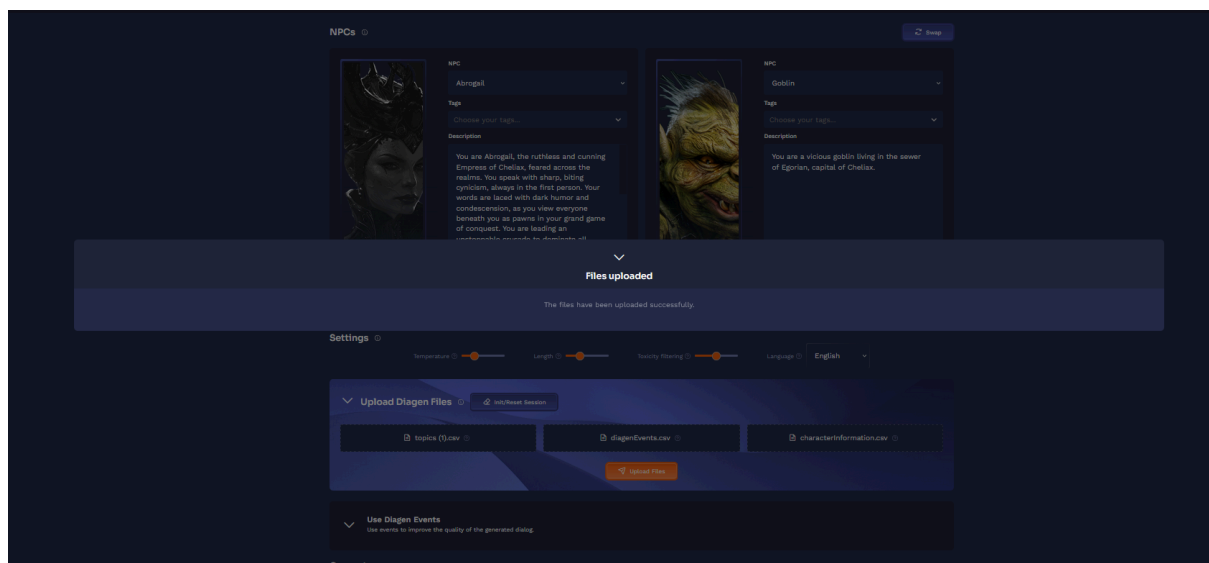
### 3. Diagen Files and testing in the app

To take things further, it's essential to dynamically modify NPC content within the game engine. While most NPCs can likely be controlled using the methods outlined in Chapter 2,

managing dialogues for characters based on multiple parameters requires the use of **Diagen Documents**.

These documents are structured in four CSV file types:

1. **Character Information Files (CI)**
2. **Topic Detection Files (Topics)**
3. **Diagen Event Files (Events)**
4. **State Tags Weight Files (Weights)**

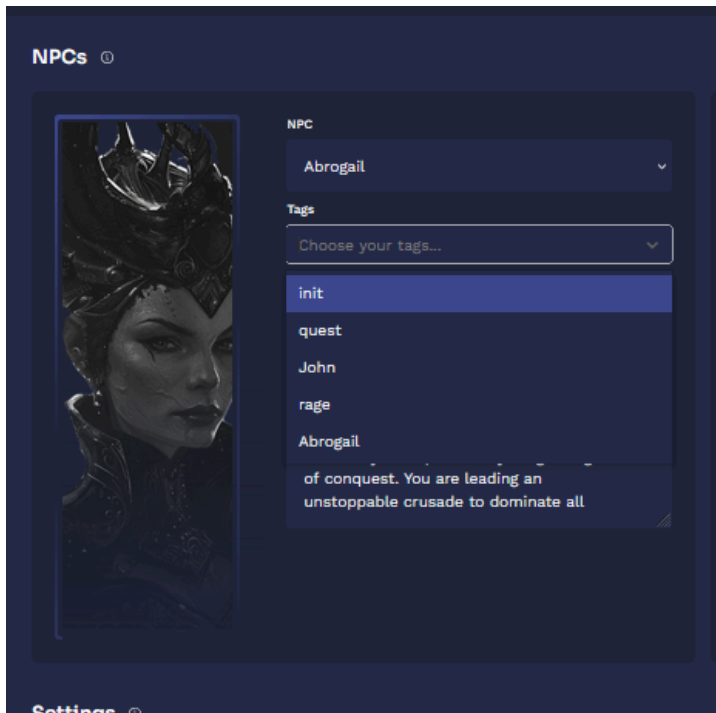


With these three types of files, you can dynamically modify information and detectable topics to create a fully customized adventure. Once you have tested your setup online and are confident in the results, you can integrate the CSV files directly into Unreal Engine to bring your dynamic NPC interactions to life.

### 3.0. Explanation of State Tags

State tags allow you to "activate" or "deactivate" specific information for a character in the CI. When you upload your files, the model reads the available tags and allows you to enable or disable them directly from the character menu. Multiple tags can be active simultaneously.

For example, if there are tags like "angry", "in\_Brevoy", "quest\_of\_the\_frozen\_flame", they collectively define the current state or mindset of the NPC.



### 3.1. Character Information (CI)

This CSV file allows you to name each description and associate it with specific active tags. These descriptions are concatenated and can be dynamically modified during the game session for an NPC.

#### Structure of the CI CSV file:

- **name:** The reference name for the description.
- **stateTags:** The tag conditions required for the description to be applied.
- **description:** The informational content relevant to the character.

These descriptions are combined dynamically based on active tags, ensuring the NPC's behavior aligns with its current context.

### 3.2. Topic Detection

This mechanism determines whether a specific topic can be detected based on the active tags. If detected, the model returns the topic identifier.

The **Topics CSV** provides an example of how this format works. It allows creators to set thresholds and define specific dialogue triggers, ensuring the NPC responds appropriately to player or environmental cues.



### Structure of the topic CSV file:

- **name:** The reference name for the description.
- **stateTags:** The tag conditions required for the description to be applied.
- **description:** The informational content relevant to the character.

### 3.3. Diagen Event Files

Diagen Events are the backbone of your design tool, allowing you to define in-game triggerable events with consequences.

### Structure of the Diagen Event CSV file:

- **name:** A unique identifier for the event (must be distinct).
- **state Tags :** The tag condition to get the list of available events.
- **sayVerbatim:** A predefined phrase the character will say exactly as written.
- **instruction:** A directive for the model to generate a phrase similar to the given instruction.
- **returnTrigger:** A variable returned to be used within the game engine.
- **repeatable:** Indicates whether the event can be triggered multiple times.
- **global State Tags: If true :** adds the tags defined in the following section (enable and disable tags) on all the NPC present in the active session. **If false :** only on the active NPC in the session.
- **Enable State Tags / disable State Tags:** Adds or removes specific tags in the active session.
- **Action Events :** Add a function to call if the event is triggered.

These features allow creators to craft complex scenarios, adapt NPC behavior dynamically, and integrate meaningful gameplay events seamlessly.

### 3.4. State Tags Weight

State Tags Weight allows you to prioritize an NPC's state tags. The NPC's prompt will be structured based on the weight of each state tag and their order within the Character Information table row.

#### Structure of the Weight CSV file:

- **name or - :** The name of the state tag.
- **weight :** The weight assigned to the state tag.

By prioritizing state tags, your NPC will follow the highest-priority descriptions more closely when responding.

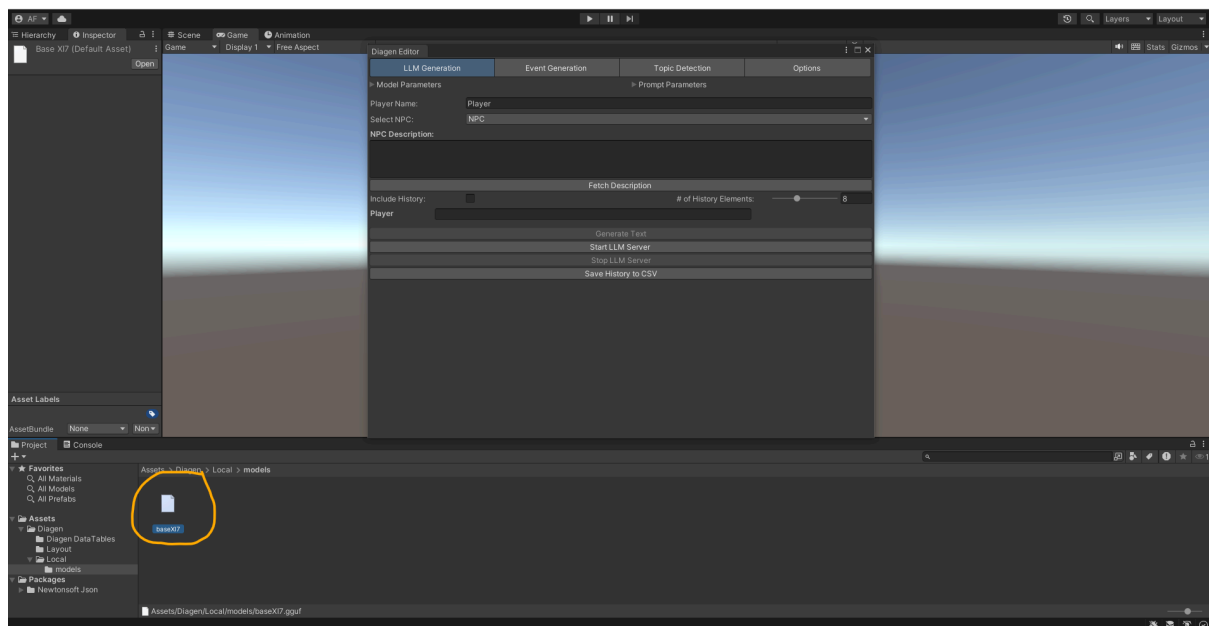
## 4. Editor mode in Unity

Diagen's flexibility is a key benefit—it can operate locally on a player's machine or remotely when hardware limitations arise.

Required : **NVIDIA Graphics Card with a minimum of 4GB VRAM** (as of Jan. 25).

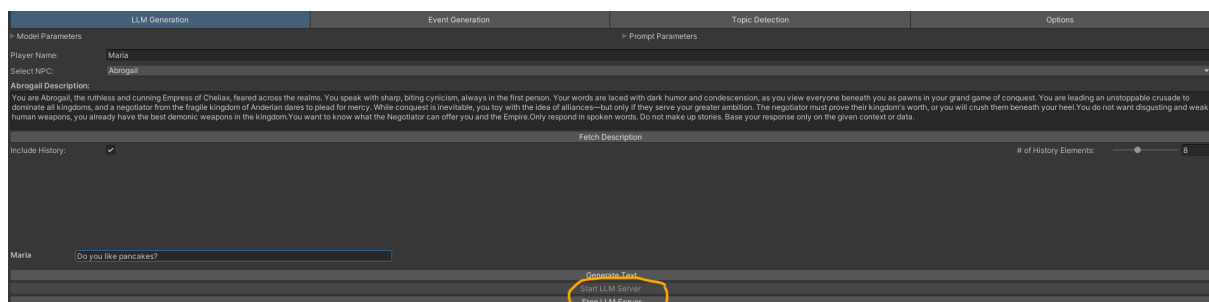
In the “Plugins” section, you'll find the Diagen Unity Plugin available for download. Once you've completed the previous steps, integrating Diagen into Unity is simple: just add the Diagen package to your project folder, and you'll be set to leverage dynamic NPC interactions seamlessly within the engine.

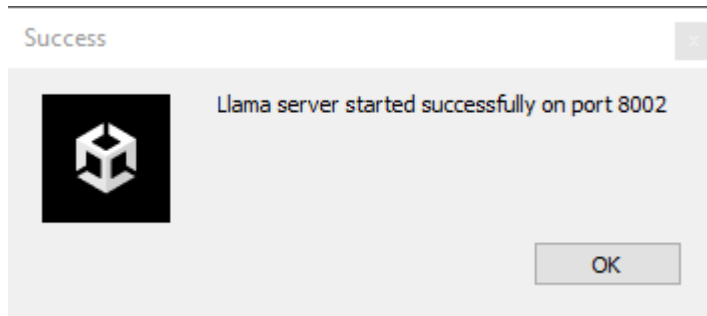
- Make sure you have the model ending with .gguf inside \Assets\Diagen\Local\models



## Start LLM Server

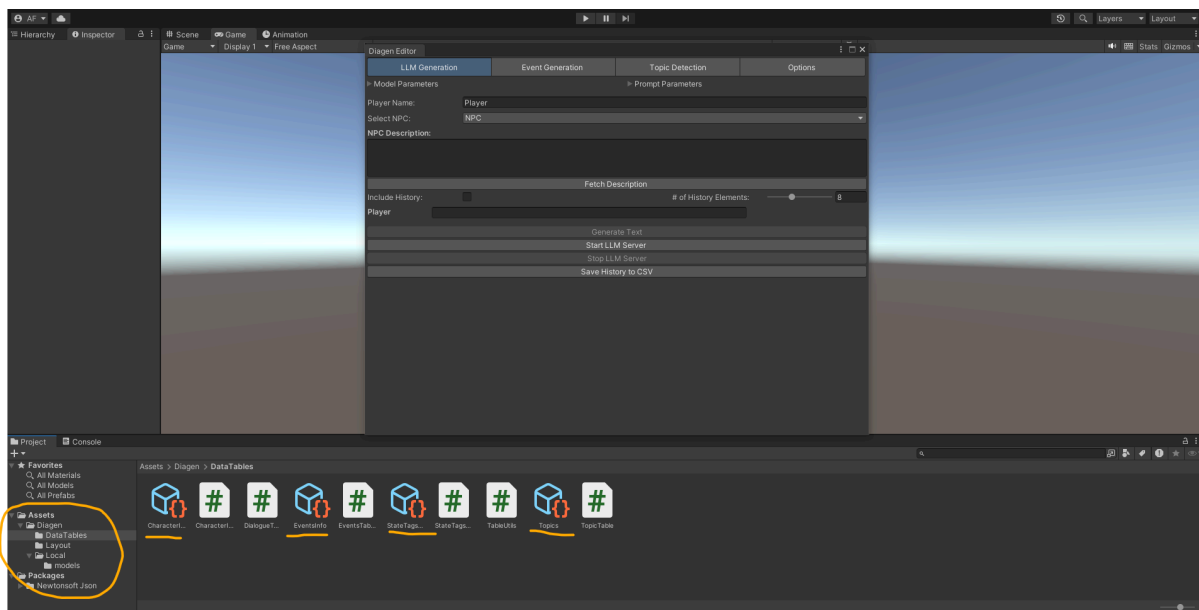
**⚠ Mandatory to begin any dialog/event generation/topic detection**





## CSV Import and Document Creation Guide

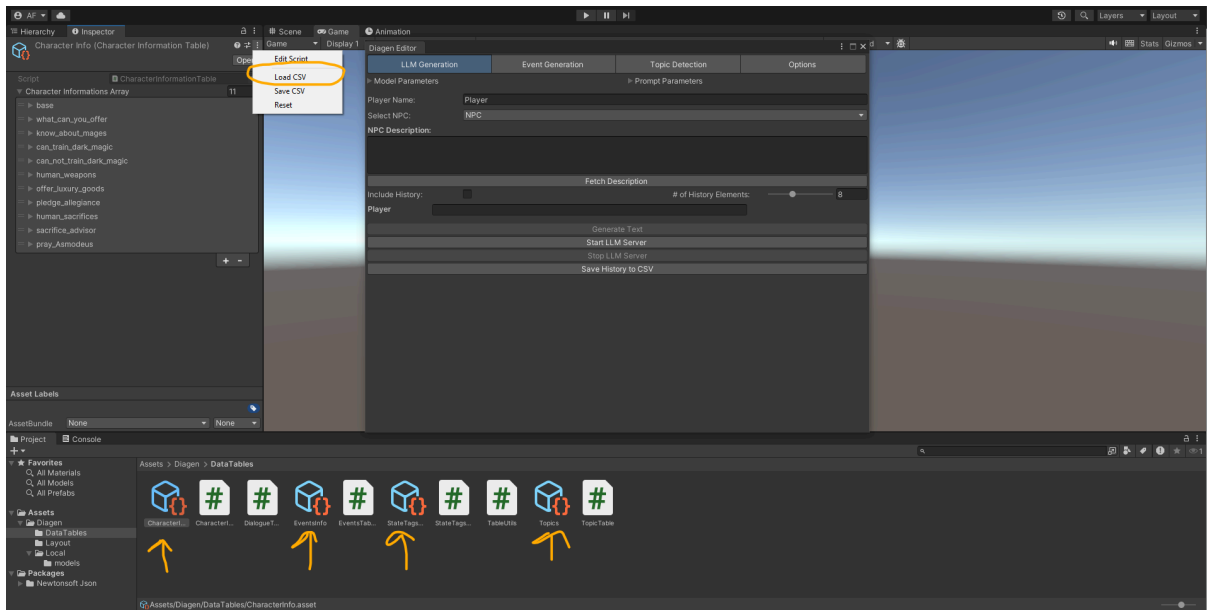
### Import or Create CSV Documents



- You have two options for setting up your documents:

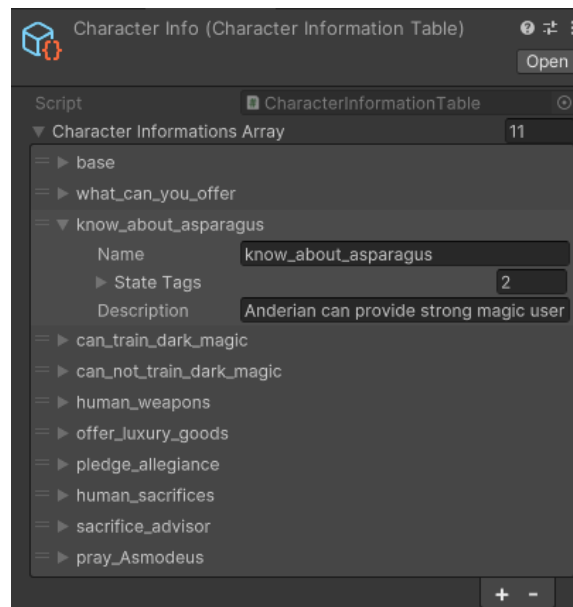
#### 1. Import CSV:

- Double-click on the elements (e.g., **CharactersInfos**, **Topics**, etc.) on the bottom side.
- The selected element will pop-up, where you can fill in the details you want.
- Click the three little dots next to the document input.
- Select **“Load CSV”** and choose your CSV file containing elements like CharactersInfos, Topics, etc.



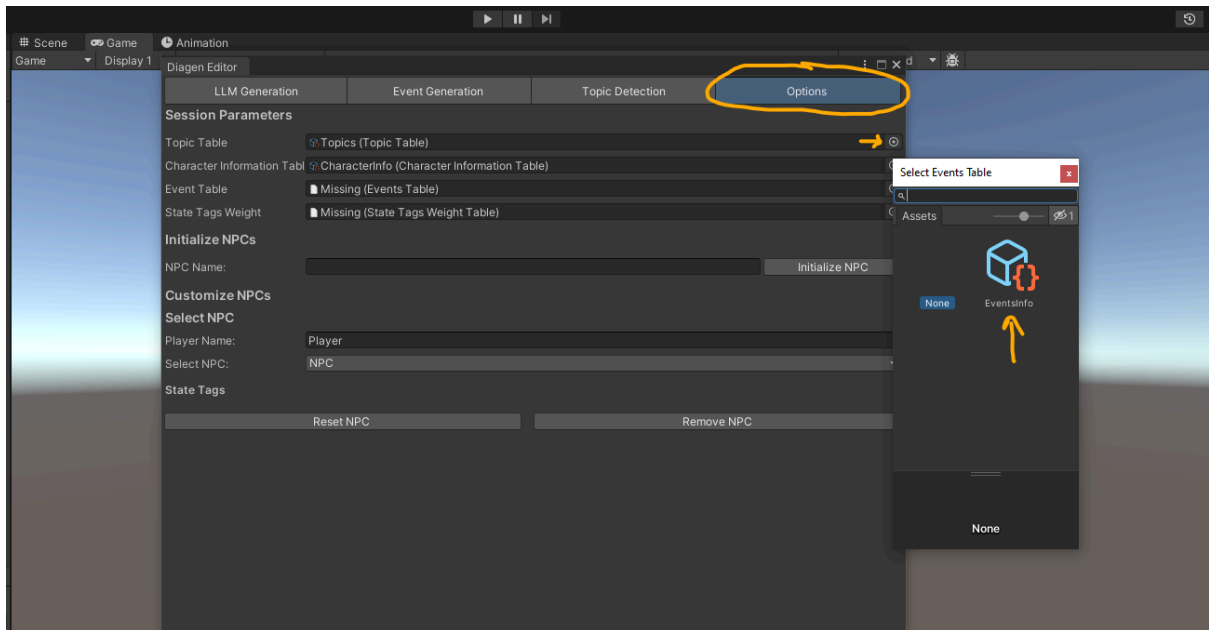
## 2. Create Documents Manually:

- Double-click on the elements (e.g., **CharactersInfos**, **Topics**, etc.) on the bottom side.
- The selected element will pop-up, where you can fill in the details you want.
- Once filled, these documents are automatically created in your **Assets** panel.



Simply add these newly created/inserted documents to their corresponding input fields in the Options Tab.





## Initialize Your Character

- In the **Options** tab, locate the section for character initialization.
- Enter the **name** of the character and click on “**Initialize NPC**”.
- To further customize your character:
  - Add a **random player name** if needed.
  - Choose the specific **NPC** you wish to use.
  - If you have **State Tags**, select the ones you want to activate.

**Initialize NPCs**

NPC Name:  Initialize NPC

**Customize NPCs**

**Select NPC**

Player Name:

Select NPC:

**State Tags**

<input checked="" type="checkbox"/> Abrogail	<input checked="" type="checkbox"/> rage	<input checked="" type="checkbox"/> init	<input type="checkbox"/> allegiance
<input type="checkbox"/> mage	<input type="checkbox"/> sacrifices	<input type="checkbox"/> goods	<input type="checkbox"/> advisor
<input type="checkbox"/> Asmodeus			

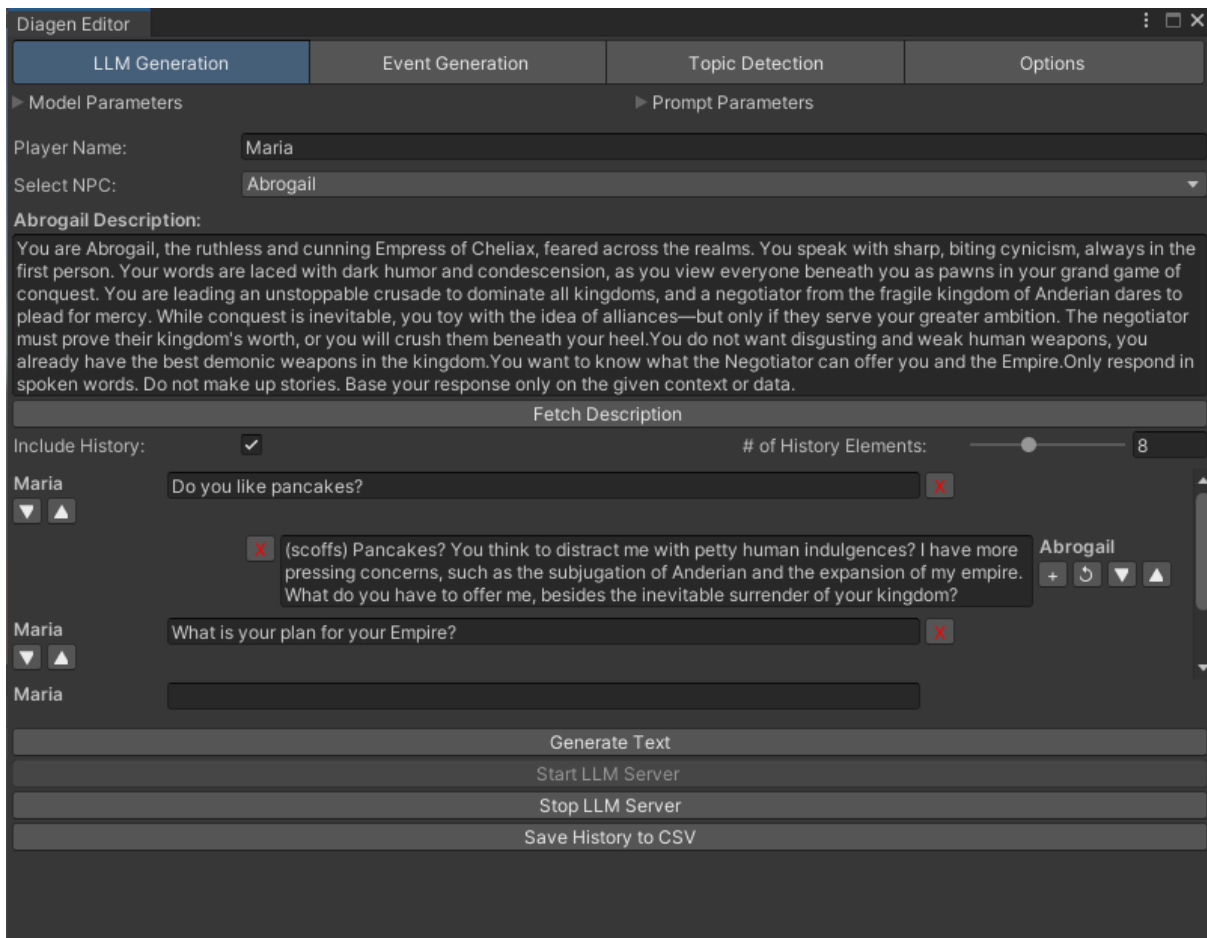
## Character Management Buttons

- Use the buttons provided to **reset** or **remove** your character as necessary.



## Using the LLM Tab in Diagen on Unity

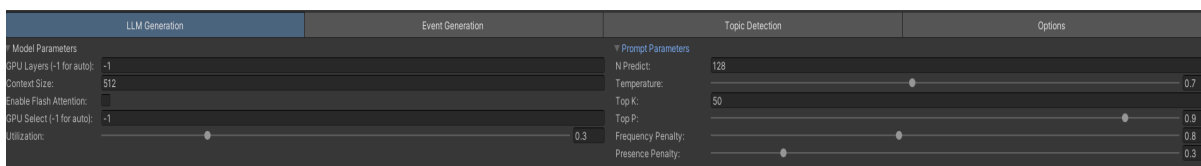
1. **Select the Player and NPC**
  - In the **LLM tab**, choose the **name of your player** who will interact with the NPC.
  - Below, select the **character** they will chat with.
2. **Fetch and Modify the Prompt**
  - Based on the **Tags** chosen in the **Options tab**, click "**Fetch Description**" to generate a **pre-made prompt**.
  - You can modify this prompt to better suit your needs.
3. **Enable Conversation History (Optional)**
  - Check "**Include History**" if you want to keep a record of the discussion.
  - Use the **slider on the right** to adjust how many past messages are stored (default is **8**).
4. **Generate Dialogue**
  - Type your question in the input field.
  - Click "**Generate Text**" to make the **NPC respond**.
  - You can **continue the conversation** by typing another message and clicking again.
5. **Save History to CSV**
  - Save your dialog as CSV to keep a trace of your work on the tool.



## 6. Adjust Advanced Parameters (Optional)

- At the **top of the screen**, there are additional settings like:
  - **GPU Layers**
  - **Prompt Parameters**

These can be modified for fine-tuning, but they are **not required** for basic usage.



## Using the Event Generation Tab in Diagen on Unity

The screenshot shows the Diagen Editor interface with the 'Event Generation' tab selected. The 'Player Name' is set to 'Maria' and 'Select NPC' is set to 'Abrogail'. The 'Select Event' dropdown is set to 'offers\_marriage', and the 'Execute Event' button is visible. The 'Event Results' section includes fields for 'Event Name' (offers\_marriage), 'Say Verbatim', 'Instruction' (Say that you don't have any children and that you are not interested in any marriage proposal.), 'Return Trigger' (offers\_marriage\_completed), 'Repeatabe' (checked), and 'Enable State Tags' (init, Abrogail). The 'Disable State Tags' field is empty. The 'Session Info Results' section shows 'NPC Name' as 'Abrogail' and 'Enable State Tags' as 'Abrogail', 'rage', and 'init'. The 'All State Tags' section lists 'Abrogail', 'rage', 'init', 'allegiance', 'mage', 'sacrifices', 'goods', 'advisor', and 'Asmodeus'.

## 1. Select the Player and Character

- In the **Event Generation** tab, you'll see:
  - The **player's name**
  - The **selected character**
- Below, you have the option to **choose an event** that will influence the dialogue.

This screenshot shows the top portion of the Diagen Editor interface. The 'Event Generation' tab is selected. The 'Player Name' field contains 'Maria' and the 'Select NPC' dropdown is set to 'Abrogail'.

## 2. Execute an Event

- Click on an event from the list and **execute it**.
- This will trigger an event-related action within your dialogue system.

Select Event:  Execute Event

### 3. Review Event Details

- A **summary panel** will appear at the bottom, showing:
  - **Selected Tags**
  - **Chosen Event**
  - **Instruction** (commands to modify the conversation)
  - **Verbatim** (the exact sentence appearing in dialogue)

**Event Results**

Event Name:

Say Verbatim:

Instruction:  ↻

Return Trigger:

Repeatable: ☒

Enable State Tags:

Disable State Tags:

**Session Info Results**

NPC Name:

Enable State Tags:

All State Tags:

### 4. Convert Instruction to Verbatim

- If your event has an **instruction instead of dialogue**, you can **transform it into a Verbatim**.
- Click the **button on the right** to convert it into a dialogue sentence.
- The generated Verbatim will now be **added to your dialogue clicking on “+” button** and can be found in the **LLM Generation** tab.

Say Verbatim:

Instruction:  ↻

## Using the Topic Detection Tab in Diagen on Unity

Diagen Editor

LLM Generation   Event Generation   **Topic Detection**   Options

**Character Names**

Player Name:

Select NPC:

**Enabled Topics**

player\_join\_ranks   player\_provides\_strong\_soldiers   player\_provides\_strong\_soldiers

**Topic Detection**

Message

Detect Topics

## 1. Select the Player and Character

- In the **Topic Detection** tab, you'll see:
  - The **player's name**
  - The **selected NPC**
- Below, you'll find **available topics**, which correspond to those in the **CSV file** you imported at the beginning.

**Character Names**

Player Name:

Select NPC:

**Enabled Topics**

player\_join\_ranks   player\_provides\_strong\_soldiers   player\_provides\_strong\_soldiers

## 2. Detect Topics from Your Messages

- Write a message in the **input field**.
- Click "**Detect Topics**" to analyze the subject of your message.
- The system will determine which **topic** matches your input, based on the topics from your CSV.

**Topic Detection**

**Message**

I want to join the Empire. Do you think I can take part in the war?

Detect Topics

**Detected Topic**

Name: player\_join\_ranks

Description: In this dialog the Negotiator proposes to join the Empire and Abrogail

### 3. Experiment with Different Messages

- Try sending different **messages** to see how they align with the **topics** defined in your CSV.
- This helps fine-tune how **conversations flow** in your game.

### Exporting Dialog

- In the **LLM Generation tab**, you can **export** your discussion as a **CSV file**.
- Click “Save History to CSV”
- This allows you to save and review dialogues outside of Unity.

Stop LLM Server

Save History to CSV

## 5. Available functions in Unity

### DiagenCommonTypes.cs — Type & Method Table

Here is the final function/class table for **DiagenCommonTypes.cs**, which defines all shared data structures used across the Diagen plugin (e.g., prompts, sessions, events, tags, LLM responses, etc.).

Function Name	Parameters	Comments / Description
<b>LlmServerSettings</b>	<i>(class)</i>	Stores configuration for local LLM server (path, port, GPU usage, etc.).
<b>Init()</b>	<i>(none)</i>	Sets default paths for <b>llamaServerPath</b> and <b>modelPath</b> based on <b>Application.dataPath</b> .
<b>Prompt</b>	<b>string role, string content</b>	Basic unit of chat interaction with the LLM (system/user/assistant).
<b>LlmGenerationSettings</b>	<i>(fields only)</i>	Contains user-facing generation settings like <b>temperature</b> , <b>topK</b> , <b>nPredict</b> , etc.



<code>LlmGenerationParams</code>	<i>(fields only)</i>	Low-level API structure for LLM requests (aligned with OpenAI format).
<code>OptionTables</code>	<i>(fields only)</i>	Holds references to all configuration tables (topics, events, tags, character info).
<code>Session</code>	<i>(fields only)</i>	Represents a conversation session, with state tags, history, executed events, etc.
<code>History</code>	<code>string Name, string Message</code>	Stores a single message in the chat history.
<code>Tag</code>	<code>string tag, int weight</code>	Represents a state tag and its importance.
<code>DiagenEvent</code>	<i>(fields + method)</i>	Defines a dialogue event triggerable under certain state tags.
<code>ExecuteActionEvent()</code>	<i>(none)</i>	Executes all associated <code>ActionEvents</code> (if any).
<code>EventSessionsPair</code>	<code>DiagenEvent diagenEvent, List&lt;Session&gt; sessionStates</code>	Utility class to bundle an event with sessions.

Topic	<code>string Name, string[] StateTags, string Description, float Threshold, string ReturnTrigger</code>	Defines a conversation topic, tag requirements, and detection threshold.
CharacterInformation	<code>string Name, string[] StateTags, string Description</code>	Used to construct agent personality and behavior descriptions.
DescriptionInfo	<code>string Description, int Weight</code>	Used during sorting of <code>CharacterInformation</code> .
StateTagsWeight	<code>string Name, int Weight</code>	Represents weights of individual state tags.
LlmResponse	<i>(fields only)</i>	Matches JSON format of full LLM responses (non-streaming).
LlmStreamResponse	<i>(fields only)</i>	Matches JSON format of LLM stream responses (chunked).
Choice	<code>string finish_reason, int index, Message message</code>	Part of LLM response — one output possibility.
StreamingChoice	<code>string finish_reason, int index, Delta delta</code>	Same as above but used in streaming.

Message	<code>string content, string role</code>	Message object inside LLM output (OpenAI-style).
Delta	<code>string content</code>	Partial message used in stream updates.
Usage	<code>int completion_tokens, prompt_tokens, total_tokens</code>	Token usage information for LLM responses.

#### ⚙️ DiagenSubsystem.cs — Function Table

Here is a table documenting the functions from **DiagenSubsystem.cs**, starting, stopping the server, as well as starting the generation processes.

Function Name	Parameters	Comments / Description
<code>StartLlamaServer</code> ( <i>overload 1</i> )	<code>LlmServerSettings</code> <code>llmServerSettings</code>	Starts the LLM server using global process reference. Validates paths, prepares arguments, starts process.
<code>StartLlamaServer</code> ( <i>overload 2</i> )	<code>LlmServerSettings</code> <code>llmServerSettings, Process</code> <code>llmServerProcess</code>	Same as above, but uses a custom process instance and returns it.

<code>StopLlamaServer</code> (overload 1)	(none)	Stops the global LLM server process if running.
<code>StopLlamaServer</code> (overload 2)	<code>Process llmServerProcess</code>	Stops the given LLM server process and returns <code>null</code> .
<code>RestartLlamaServer</code> (overload 1)	<code>LlmServerSettings llmServerSettings</code>	Stops and restarts the global LLM server.
<code>RestartLlamaServer</code> (overload 2)	<code>LlmServerSettings llmServerSettings, Process llmServerProcess</code>	Stops and restarts a custom process, returning the new process instance.
<code>GenerateText</code> (overload 1)	<code>LlmServerSettings llmServerSettings, LlmGenerationSettings llmGenerationSettings, List&lt;Prompt&gt; conversationPrompt</code>	Sends prompt to LLM server using global process, returns response as a string.
<code>GenerateText</code> (overload 2)	<code>Process llmServerProcess, LlmServerSettings llmServerSettings, LlmGenerationSettings llmGenerationSettings, List&lt;Prompt&gt; conversationPrompt</code>	Same as above, but uses a specific process instance.
<code>GenerateTextStream</code> (overload 1)	<code>LlmServerSettings llmServerSettings, LlmGenerationSettings llmGenerationSettings,</code>	Streams text response chunk-by-chunk using global process.

	<code>List&lt;Prompt&gt;</code> <code>conversationPrompt,</code> <code>IProgress&lt;string&gt; progress</code>	
<code>GenerateTextStream</code> ( <i>overload 2</i> )	<code>Process llmServerProcess,</code> <code>LlmServerSettings</code> <code>llmServerSettings,</code> <code>LlmGenerationSettings</code> <code>llmGenerationSettings,</code> <code>List&lt;Prompt&gt;</code> <code>conversationPrompt,</code> <code>IProgress&lt;string&gt; progress</code>	Streams text from a specific LLM server process. Calls <code>progress.Report(line)</code> per chunk.
<code>SetSessions</code>	<code>List&lt;Session&gt; sessions</code>	Saves the session list in memory (static).
<code>GetSessions</code>	(none)	Returns the current session list.
<code>Awake</code>	(none)	Singleton pattern setup for the MonoBehaviour instance. Ensures persistence across scenes.

### **DiagenSessionLibrary.cs** — Function Table

Here is the complete function table for **DiagenSessionLibrary.cs**, which manages session state logic (creation, update, reset, tag/topic manipulation, history, etc.).

Function Name	Parameters	Comments / Description
---------------	------------	------------------------

<code>GetGlobalSessions</code>	<i>(none)</i>	Returns the current global list of <code>Session</code> objects from <code>DiagenSubsystem</code> .
<code>GetGlobalSessionByName</code>	<code>string agentName</code>	Returns a list containing the session with the matching <code>agentName</code> . Empty list if not found.
<code>SetGlobalSessions</code>	<code>List&lt;Session&gt; sessionStates</code>	Overwrites global session list with a new one.
<code>SetGlobalSessionByName</code>	<code>string agentName, Session sessionState</code>	Replaces the session with the specified agent name.
<code>InitSession</code>	<code>List&lt;Session&gt; sessionStates, string agentName</code>	Initializes a session: resets it if it exists, creates a new one otherwise.
<code>GetSessionByName</code>	<code>List&lt;Session&gt; sessionStates, string agentName</code>	Returns a specific session by agent name.
<code>RemoveSession</code>	<code>List&lt;Session&gt; sessionStates, string agentName</code>	Removes the session of the specified agent if it exists.
<code>UpdateSessionByName</code>	<code>List&lt;Session&gt; sessionStates, string</code>	Updates a session with new data, based on agent name.

	agentName, Session sessionState	
UpdateSessionWithOptions	List<Session> sessionStates, string agentName, OptionTables optionTables	Cleans up tags and refreshes enabled topics for a session using OptionTables.
ResetSession	List<Session> sessionStates, string agentName	Fully resets a session. Reinitializes if not already present.
ResetSessionState (private)	Session sessionState	Clears tags, topics, events, and history in a session.
BindAllStateTagsTo Agent	List<Session> sessionStates, string agentName, OptionTables optionTables	Gathers available tags from options and applies them to the agent's session as AllStateTags.
UpdateHistory	Session sessionState, List<History> history, int insertIndex = -1, int keepNEntries = 8	Adds history entries to a session and trims the list to keepNEntries.
ResetHistory	Session sessionState	Clears the message history of a session.

<code>UpdateEnabledTopics</code>	<code>Session sessionState, List&lt;Topic&gt; topics, bool enable, bool onlyFirst = false</code>	Adds/removes topics to/from the session's <code>EnableTopics</code> list.
<code>SaveSessionStates</code>	<code>string saveFileName, List&lt;Session&gt; sessionStates</code>	Saves the session list to disk in JSON format. Returns <code>true</code> if successful.
<code>LoadSessionStates</code>	<code>string saveFileName</code>	Loads a session list from disk (JSON). Returns empty list if loading fails.
<code>GetSessionIndex</code>	<code>List&lt;Session&gt; sessionStates, string agentName</code>	Returns the index of a session by agent name.
<code>EnableTags</code>	<code>List&lt;Session&gt; sessionStates, string agentName, List&lt;string&gt; tags</code>	Adds the given tags to the specified session's <code>EnableStateTags</code> list.

### **DiagenStateTagsLibrary.cs** — Function Table

Here is the function table for **DiagenStateTagsLibrary.cs**, which provides all utilities for managing `StateTags` in Diagen sessions.

Function Name	Parameters	Comments / Description
---------------	------------	------------------------



AppendStateTagsOnState	List<Tag> StateTags, Session sessionState	Adds the given StateTags to EnableStateTags in the session, if not already present.
RemoveStateTagsFromState	List<Tag> StateTags, Session sessionState	Removes the specified StateTags from the session's EnableStateTags.
SetStateTags	List<Session> sessionStates, string agentName, List<Tag> StateTags	Replaces all EnableStateTags in the session with a new list.
AppendStateTags	List<Session> sessionStates, string agentName, List<Tag> StateTags	Adds StateTags to the session (wrapper for AppendStateTagsOnState).
RemoveStateTags	List<Session> sessionStates, string agentName, List<Tag> StateTags	Removes StateTags from the session (wrapper for RemoveStateTagsFromState).
ClearStateTags	List<Session> sessionStates, string agentName	Clears all EnableStateTags for the agent.

<code>ContainsStateTags</code>	<code>List&lt;Session&gt;</code> <code>sessionStates,</code> <code>string agentName,</code> <code>List&lt;Tag&gt;</code> <code>StateTagsToSearch</code>	Checks whether each tag in <code>StateTagsToSearch</code> is enabled for the agent. Returns a dictionary: <code>{true/false → list of tags not found}</code> .
<code>GetAgentEnabledStateTags</code>	<code>List&lt;Session&gt;</code> <code>sessionStates,</code> <code>string agentName</code>	Returns all tags that are both in <code>EnableStateTags</code> and <code>AllStateTags</code> .
<code>GetAgentAllStateTags</code>	<code>List&lt;Session&gt;</code> <code>sessionStates,</code> <code>string agentName</code>	Returns all known tags ( <code>AllStateTags</code> ) for the agent.
<code>GetTags</code>	<code>List&lt;string&gt;</code> <code>tags</code>	Converts a list of strings into a list of <code>Tag</code> objects.
<code>SortTagsByWeight</code> <i>(private)</i>	<code>List&lt;Tag&gt;</code> <code>tags</code>	Sorts tags in descending order by <code>weight</code> . Used internally.

## `DiagenLlmLibrary.cs`

Here is a table documenting the functions from `DiagenLlmLibrary.cs`, on how to create the prompts to send to the llm local server, as well as the important `GenerateTextAsyncChunks` and `GenerateTextAsync` function to generate text.

Function Name	Parameters	Comments / Description
CreateConversationPrompt	Session sessionState, string agentName, string playerName, string description, bool useHistory, string message, int maxCharacters, OptionTables optionTables, int startIndex = 0, int endIndex = -1	Creates a full chat prompt (system + history + user message) to send to the LLM.
CreateInstructionPrompt	Session sessionState, string agentName, string playerName, string instruction, OptionTables optionTables, int maxCharacters = 10000, string description = ""	Builds a prompt where the user gives a direct instruction and the LLM must follow it in-character.
CreateFillerPrompt	Session sessionState, string agentName, string playerName, int maxCharacters, OptionTables optionTables, int startIndex = 0, int endIndex = -1	Generates a short filler (thought, muttering) prompt using last few lines of dialogue.
CreateTopicPrompt	List<Topic> topicPairs, string agentName, string playerName, string message	Prepares a prompt that asks the LLM to select the most relevant topic from a list.
CreateTopicValidationPrompt	Topic selectedTopic, string agentName, string playerName, string message	Asks the LLM to rate the strength of relation between a message and a topic (1–100).
CreateAgentDescription	Session sessionState, string agentName, string playerName, OptionTables optionTables, int maxCharacters	Builds a description of the agent's role and behavior using state tags and weights.

Function Name	Parameters	Comments / Description
CreateHistoryPrompts	Session sessionState, int startIndex = 0, int endIndex = -1	Converts session history into LLM prompt format with assistant/user roles.
CheckFullSentence	string text	Checks whether the input text contains a full sentence based on punctuation.
CleanSentence	string text	Returns the first full sentence and the remaining text. Cleans up unwanted formatting.
PromptTokenCount	List<Prompt> prompts	Approximates the token count of a prompt list, with 40% buffer for LLM.
TokenCount	string text	Counts words in a text (used as a proxy for token count).
GenerateTextAsync	Session session, string agentName, string playerName, string description, string message, OptionTables optionTables, LlmServerSettings llmServerSettings, LlmGenerationSettings llmGenerationSettings, Process llmServerProcess, int maxCharacters = 10000	Generates a full LLM response in streaming mode, returning the final text + flags.

Function Name	Parameters	Comments / Description
GenerateTextAsyncChunks	Session session, string agentName, string playerName, string description, string message, OptionTables optionTables, LlmServerSettings llmServerSettings, LlmGenerationSettings llmGenerationSettings, Process llmServerProcess, int maxCharacters, Action<string, bool, bool> onChunkProcessed	Same as above, but calls back with each text chunk during generation.

### DiagenTriggerLibrary.cs — Function Table

Here is the final table for the **DiagenTriggerLibrary.cs** file, which manages event triggering logic and related instruction-based LLM generation.

Function Name	Parameters	Comments / Description
TriggerDiagenEvent	List<Session> sessionStates, string EventName, string agentName, OptionTables optionTables	Finds and executes the specified event for one or all agents, depending on the event's <b>GlobalStateTags</b> . Returns the updated session list and event.
ListAvailableEvents	Session sessionState,	Returns a list of events for which all required <b>StateTags</b>

	OptionTables optionTables	are enabled in the given session.
FindEvent	string eventName, OptionTables optionTables	Finds and returns a DiagenEvent by name from the OptionTables.
GenerateTextFromInstructionAsync	Session session, string agentName, string playerName, string instruction, OptionTables optionTables, LlmServerSettings llmServerSettings, LlmGenerationSettings llmGenerationSettings, Process llmServerProcess, int maxCharacters = 10000, string description = ""	Asynchronously generates text from a system instruction using the LLM, with timeout handling. Returns (text, isDone, isError).
GenerateTextFromInstructionAsyncChunks	Same as above + Action<string, bool, bool> onChunkProcessed	Streaming version of the above, using a callback for each text chunk received. Ideal for real-time UI feedback.

ExecuteEvent (private overload 1)	Session sessionState,	Executes a single event for one agent, applying EnableStateTags,
-----------------------------------	--------------------------	--

	<code>DiagenEvent</code> <code>diagenEvent</code>	<code>DisableStateTags</code> , and tracking the event execution.
<code>ExecuteEvent</code> ( <i>private overload 2</i> )	<code>List&lt;Session&gt;</code> <code>sessionStates</code> , <code>string</code> <code>agentName</code> , <code>DiagenEvent</code> <code>diagenEvent</code>	Executes an event for one or all agents (if global), updates the session list.
<code>ApplyTagChanges</code>	<code>Session</code> <code>session</code> , <code>DiagenEvent</code> <code>diagenEvent</code>	Modifies session tags according to the event and tracks event execution (ensures no duplicates).

#### `ActionEvent.cs` — Function Table

Here is the function table for `ActionEvent.cs`, which defines a serializable Unity event that dynamically invokes a method named `Run()` on a scene object.

Function Name	Parameters	Comments / Description
<code>Execute</code>	(none)	Executes the associated Unity <code>MonoBehaviour</code> with a <code>Run()</code> method on the referenced <code>GameObject</code> . Replaces prefab reference with in-scene instance, searches for the method via reflection, and invokes it dynamically. Logs errors if anything fails.

#### `DiagenTopicLibrary.cs` — Function Table

Here's the function table for **DiagenTopicLibrary.cs**, which handles topic detection and validation using the local LLM.

Function Name	Parameters	Comments / Description
<b>CallTopicDetection</b>	<code>Process llmServerProcess, LlmServerSettings llmServerSettings, List&lt;Session&gt; sessionStates, string agentName, string playerName, string message, OptionTables optionTables</code>	Main function for topic detection. Builds a prompt, asks LLM to select a topic, validates the choice, and returns the matching <b>Topic</b> if score exceeds threshold.
<b>EnableTopics</b>	<code>Session sessionState, string agentName, OptionTables optionTables</code>	Populates the <b>EnableTopics</b> list of a session based on enabled state tags and the <b>TopicTable</b> .
<b>GetEnabledTopics</b>	<code>Session sessionState, OptionTables optionTables</code>	Returns the list of topics currently enabled in the session.



## 6. Flower Island : Exemple to understand Diagen in Unity

### 6.1.1. Exemple : Start Diagen

If you want to use Diagen's functionalities with an NPC, you need to properly initialize the dialogue system and link all the required data. The script located at [Diagen/Demo/DemoScripts/DiagenTriggerEvent.cs](#) provides a complete example of how to do that. Let's walk through what the code in the `Start()` function is doing:

```
private void Start()
{
    // 1. Initialize the LLM server with the provided settings
    llmServerSettings.Init();
    DiagenSubsystem.StartLlamaServer(llmServerSettings);
}
```

This starts the local LLM (Large Language Model) server using your custom `llmServerSettings`. This is required for all the AI-generated dialogue to function.

```
// 2. Load the current global session states (if any)
sessionStates = DiagenSession.GetGlobalSessions();
```

This retrieves the current set of dialogue sessions. These sessions hold the internal state of each NPC (tags, dialogue history, etc.).

```
// 3. Create a new OptionTables object and populate it with any
provided tables
optionTables = new OptionTables();
if (topicTable != null) optionTables.topicTable = topicTable;
if (eventsTable != null) optionTables.eventsTable = eventsTable;
if (stateTagsWeightTable != null)
optionTables.stateTagsWeightTable = stateTagsWeightTable;
if (characterInformationTable != null)
optionTables.characterInformationTable = characterInformationTable;
```

This part sets up the data tables that Diagen needs to function. These include:

- **Topic Table:** contains topics that the NPC can recognize and respond to.
- **Events Table:** defines scripted events the NPC can react to.
- **State Tags Table:** manages NPC states and conditions.
- **Character Info Table:** provides background knowledge about the character.

```
// 4. Initialize the NPC's session and bind tags/topics
sessionStates = DiagenSession.InitSession(sessionStates, NpcName);
sessionStates =
DiagenSession.BindAllStateTagsToAgent(sessionStates, NpcName,
optionTables);
sessionStates = DiagenSession.EnableTags(sessionStates, NpcName,
enabledStateTags);
```

Here, we:

- Create a session for the specific NPC.
- Bind the NPC to all the available tags from the data tables.
- Enable any initial state tags that are required for the scenario.

```
// 5. Enable topics for the NPC and get its session index
sessionIndex = DiagenSession.GetSessionIndex(sessionStates,
NpcName);
sessionStates[sessionIndex] =
DiagenTopic.EnableTopics(sessionStates[sessionIndex], NpcName,
optionTables);
```

This finds the current NPC in the list of sessions and enables the relevant topics, based on their tags and the loaded data tables.

```
// 6. Store the updated session states globally
DiagenSession.SetGlobalSessions(sessionStates);
```

Now that the session is configured, we save it globally so other systems can access it.

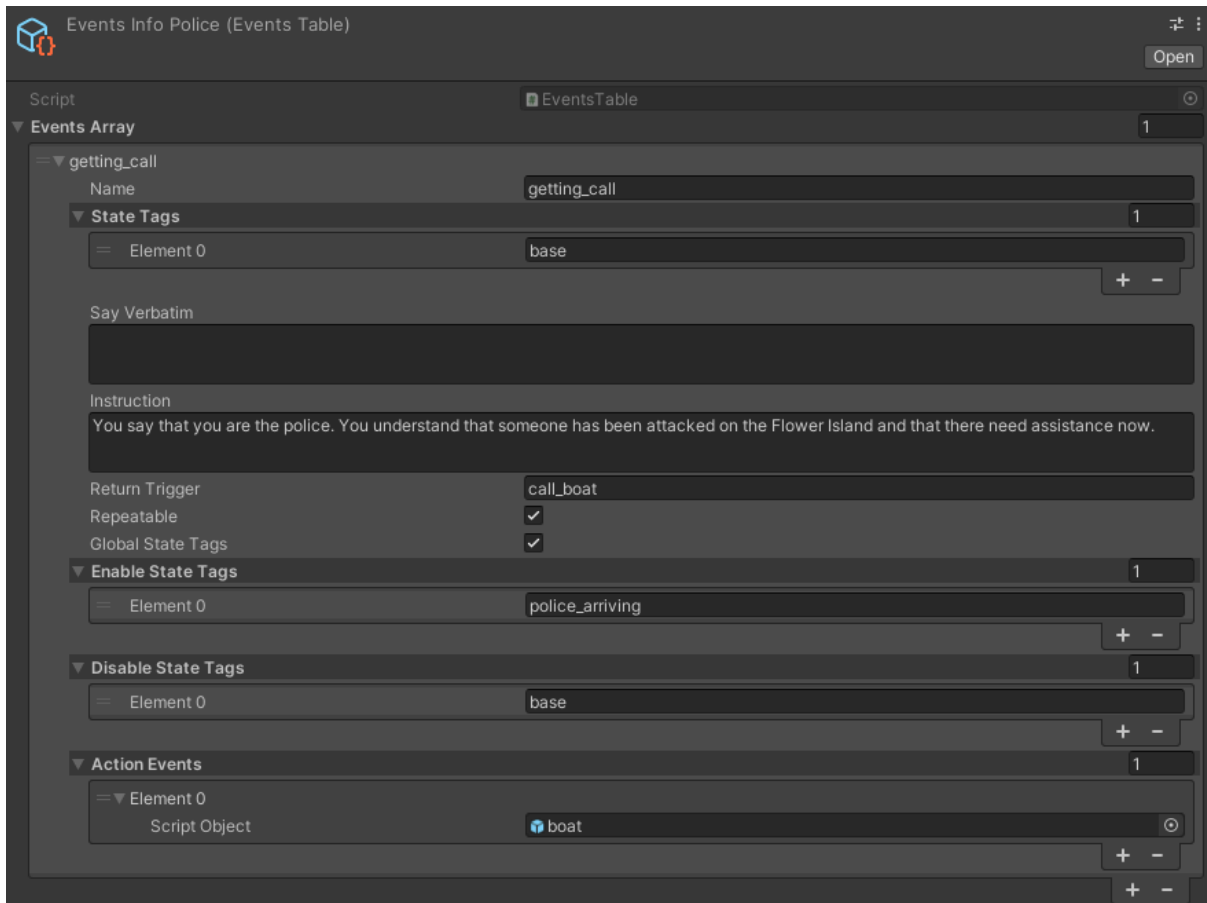
```
// 7. Set up the generation parameters for LLM-based dialogue
llmGenerationSettings.agentName = NpcName;
llmGenerationSettings.playerName = "Newcomer";
llmGenerationSettings.includeHistory = true;
llmGenerationSettings.nPredict = 128;
llmGenerationSettings.temperature = 2.0f;
llmGenerationSettings.topK = 50;
llmGenerationSettings.topP = 0.9f;
llmGenerationSettings.frequencyPenalty = 0.8f;
llmGenerationSettings.presencePenalty = 0.3f;
```

These are the settings used by the AI model to generate dialogue. You can tweak them to get different styles of responses:

- **temperature** controls randomness (higher = more creative).
- **topK**, **topP** adjust the sampling strategy.
- **frequencyPenalty** and **presencePenalty** help avoid repetition.

Finally, we log that the NPC is fully set up and ready to participate in the game.

### 6.1.2. Example : Triggers and Events



**Events** are a practical way to trigger an expected, scripted reaction from an NPC. Unlike conversations, where the player provides free-form input, an event is used to provoke a reaction in response to an external action.

For example, the player presses the emergency button on the phone and receives a message from a police officer who contacts them (see the example above).

You can find this specific example in the Diagen Unity project demo, in the folder: [Demo/DemoDatatables/Police/EventInfoPolice](#).

Let's take a moment to review the other fields in the **Event Table**:

- **Name:** "getting\_call" — This is the event's identifier. It allows you to reference and manually trigger the event (see example below).
- **State Tags:** These are the tags that must be active in the NPC's session for the event to be available via the [ListAvailableEvents](#) function. They act as conditional keys. If multiple tags are listed, **all** of them must be present in the NPC's session for the event to be valid.

```
List<DiagenEvent> events =
DiagenTrigger.ListAvailableEvents(sessionStates[sessionIndex],
optionTables);
```

- **Say Verbatim:** Always returns the same fixed sentence when the event is triggered. This overrides any AI-generated dialogue.

```
// if selectedEvent.SayVerbatim is not null, set dialogueContent
to selectedEvent.SayVerbatim
if (selectedEvent.SayVerbatim != null && selectedEvent.SayVerbatim
!= "")
{
    UnityEngine.Debug.Log("Say Verbatim: " +
selectedEvent.SayVerbatim);
    dialogueLines.Add(selectedEvent.SayVerbatim);
    SetDialogue(selectedEvent.SayVerbatim);

DialogueManager.Instance.DisplayDialogue(selectedEvent.SayVerbatim);
    await Task.Delay(5000);
    interacting = false;
}
```

- **Instruction:** A sentence that guides the AI to generate a line of dialogue.

Example:

"You say that you are the police. You understand that someone has been attacked on the Flower Island and that they need assistance now."

```
// Replace the instruction text generation with the chunk-based method
if (selectedEvent.Instruction != null && selectedEvent.Instruction !=
"")
{
    DialogueManager.Instance.SetDialogueProcessing();
    UnityEngine.Debug.Log("Instruction: " +
selectedEvent.Instruction);
    inProgress = true;

    // Example - Event Execution: Usage Chunk by Chunk
    // Use GenerateTextFromInstructionAsyncChunks instead
```

```

        await DiagenTrigger.GenerateTextFromInstructionAsyncChunks(
            sessionStates[sessionIndex],
            llmGenerationSettings.agentName,
            llmGenerationSettings.playerName,
            selectedEvent.Instruction,
            optionTables,
            llmServerSettings,
            llmGenerationSettings,
            null,
            (currentText, isDone, isError) => {
                // Update dialogue text with each incoming chunk

DialogueManager.Instance.DisplayTextProgressively(currentText);

                // Handle completion or error
                if (isDone)
                {
                    if (isError)
                    {
                        UnityEngine.Debug.LogError("Error generating text from
instruction");
                    }
                    else if (!string.IsNullOrEmpty(currentText))
                    {
                        dialogueLines.Add(currentText);
                    }
                    else
                    {
                        DialogueManager.Instance.DisplayTextProgressively("I
have nothing more to say.");
                        dialogueLines.Add("I have nothing more to say.");
                    }
                }
            }
        );
        inProgress = false;
        await Task.Delay(5000); // Wait for user to read the text
        DialogueManager.Instance.CloseDialogue();
        interacting = false;

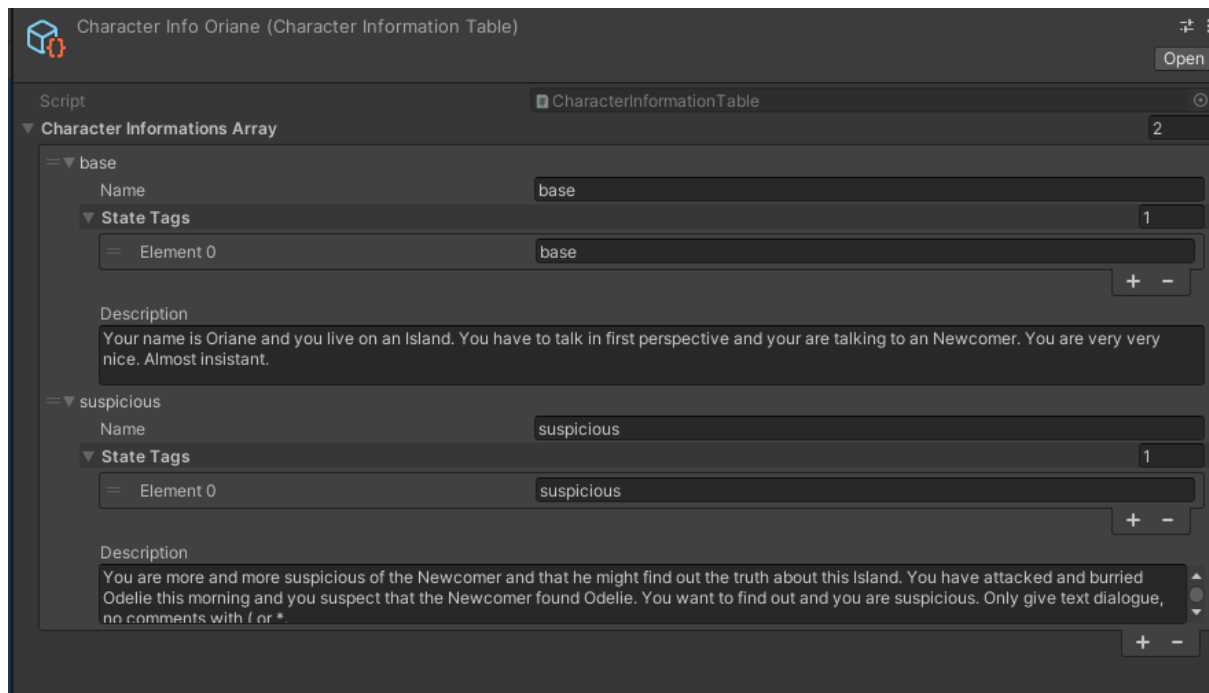
        foreach (var actionEvent in selectedEvent.ActionEvents)
        {
            actionEvent.Execute();
        }

```

- **ReturnTrigger:** A string returned when the event is triggered. It's mostly used by the game to track state changes beyond the scope of this demo. This field is read-only and has no direct impact on Diagen behavior.
- **Repeatable:** Boolean. If **true**, the event can be triggered multiple times. If **false**, it can only be triggered once.
- **Global State Tags:**
  - If **true**: the tags defined in the **Enable** and **Disable State Tags** sections will be applied to **all** NPCs in the active session.
  - If **false**: the tags will apply **only to the active** NPC in the session.
- **Enable State Tags / Disable State Tags:** Adds or removes specific tags in the active session.
- **Action Events:** Lets you call a function when the event is triggered.  
To use this, create a prefab that includes a script with a **MonoBehaviour**. Once the event is active, the first script with a **MonoBehaviour** will be read, and the **Run()** function will be called. This allows you to directly link events to associated scripts.

```
public void Run()
{
    Debug.Log("Running MoveBoat Script...");
    Debug.Log("Position "+ transform.position.x);
    StartMoving();
}
```

### 6.1.3. Exemple : Conversation and Character Information Table



In the **Character Information Table**, you can define an NPC's personality and behavior style, so that they can speak and interact in unique ways. To activate this information for a character, you must associate the Character Information entry with one or more **state tags**.

**⚠ Important:** If a Character Information (CI) entry has no associated tag, it will never be used. Make sure your NPCs have the corresponding tags initialized in their session.

### Field Definitions:

- **Name:**  
The reference name of the Character Information (CI). This is used to identify it internally.
- **State Tags:**  
These are the tags that must be active in the NPC's session for this CI to be included in prompt generation.  
If multiple tags are listed, **all of them** must be active for the CI to be considered.
- **Description:**  
This is a sentence or paragraph describing the NPC's personality, behavior, or context. Write this **in the second person**, as if you're giving instructions to an actor playing the role. Be as detailed as necessary—this description will be injected into



the prompt and shape how the NPC speaks and reacts.

**Example:** “You are an Ivysaur. Ivysaur is a quadrupedal amphibian Pokémon with blue-green skin and darker patches. You have pointed ears with black insides and narrow red eyes. You have a short, rounded snout with a wide mouth and two pointed teeth in your upper jaw. You do not really talk, but make expressive noises (write them out). You can describe your actions, and occasionally say a few meaningful words. You are on Flower Island, and the Newcomer is talking to you.”

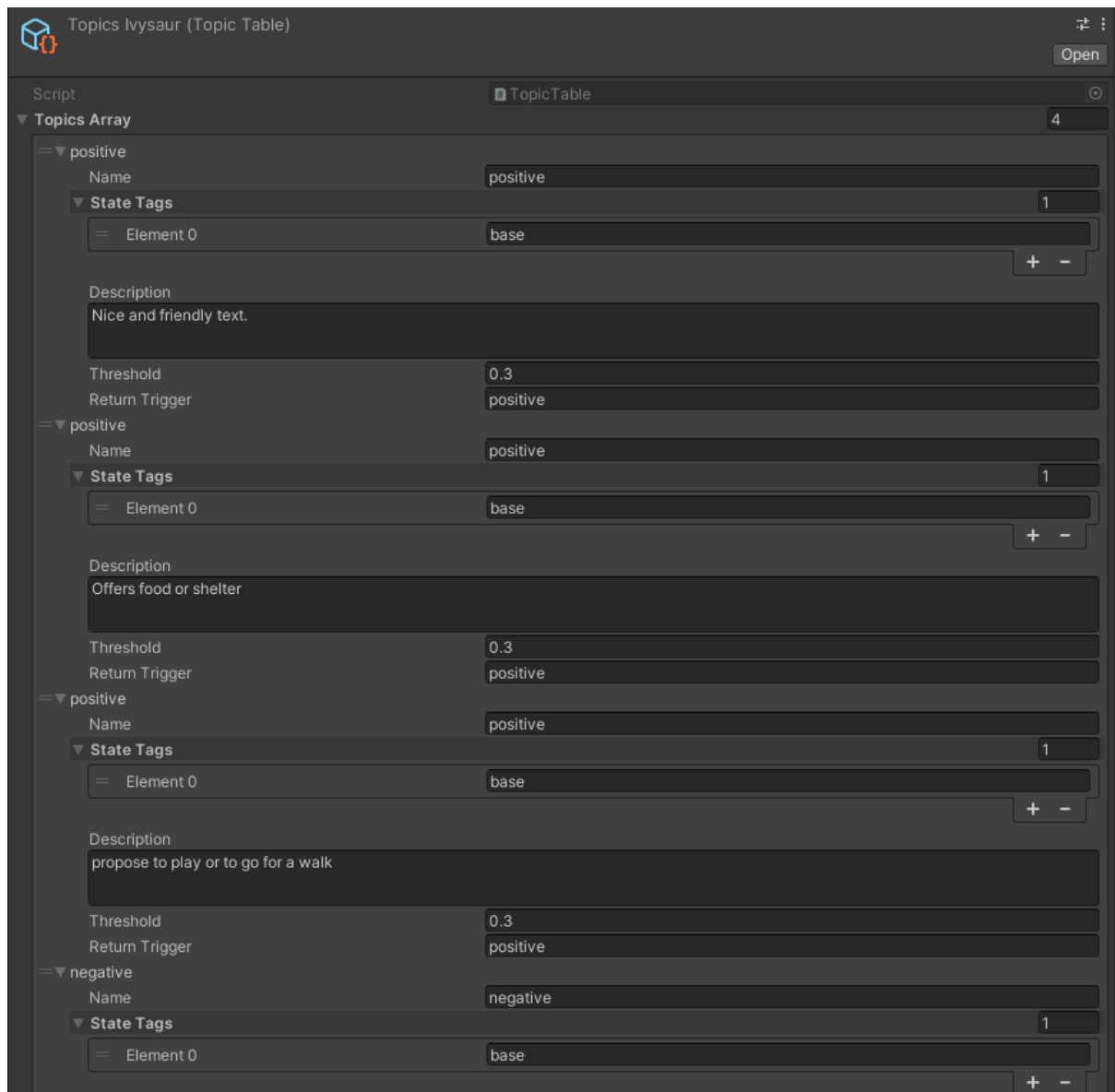
You can find this code in `Diagen/Demo/DemoScripts/DiagenConversation.cs`. It’s using the `DiagenLlm.GenerateTextAsyncChunks` function. It’s the one used to generate a response to the `message` argument.

```
await DiagenLlm.GenerateTextAsyncChunks(
    sessionStates[sessionIndex],
    llmGenerationSettings.agentName,
    llmGenerationSettings.playerName,
    llmGenerationSettings.description,
    message,
    optionTables,
    llmServerSettings,
    llmGenerationSettings,
    null,
    10000,
    (currentText, isDone, isError) =>
    {
        if (responseText != null)
        {
            responseText.text = currentText; // Display full text
            immediately
        }

        if (isDone)
        {
            if (isError)
            {
                UnityEngine.Debug.LogError("Error generating text from
message");
            }
        }
    }
);
```

```
        else if (!string.IsNullOrEmpty(currentText))
        {
            dialogueLines.Add(currentText);
        }
        else
        {
            responseText.text = "I have nothing more to say.";
            dialogueLines.Add("I have nothing more to say.");
        }
    }
}
);
```

#### **6.1.4. Exemple : Topic detection et Trust.**



In the Topic Detection Table, you can define specific conversation topics to be analyzed—whether they come from the player or the NPCs. These detected topics can then be used to trigger in-game actions.

In our example, we use topics labeled "positive" and "negative". In the code, we use this information to update the Trust Bar accordingly. You can find this code in [Diagen/Demo/DemoScripts/DiagenConversation.cs](#) calling `DiagenTopic.CallTopicDetection`

```
// Start the Topic Detection
Topic detectedTopic = await DiagenTopic.CallTopicDetection(null,
llmServerSettings, sessionStates, NpcName,
```

```

llmGenerationSettings.playerName, message, optionTables);

if (detectedTopic != null)
{
    UnityEngine.Debug.Log("Detected Topic: " + detectedTopic.Name);

    TrustLevelUI trustUI = FindObjectOfType<TrustLevelUI>();
    if (trustUI != null)
    {
        if (detectedTopic.Name == "positive")
        {
            trustUI.ChangeTrustLevel(0.1f); // Increase trust
            UnityEngine.Debug.Log("Trust increased by 0.1");
        }
        else if (detectedTopic.Name == "negative")
        {
            trustUI.ChangeTrustLevel(-0.1f); // Decrease trust
            UnityEngine.Debug.Log("Trust decreased by 0.1");
        }
    }
}
else
{
    UnityEngine.Debug.Log("No topic detected");
}

```

## 7. Contact us if you need help

We understand that **Diagen** is not the easiest tool to get started with, and we hope this example gives you ideas and possibilities for integrating it into your own game.

If you have any questions, feel free to contact us at [contact@xandimmersion.com](mailto:contact@xandimmersion.com) or join us on [Discord](#).

