OMI_personality

Contributors

Anthony Burchell, Individual Contributor

Status

Draft Specification

Open Metaverse interoperability Group Stage 1 proposal

Dependencies

Written against the glTF 2.0 spec.

Overview

The OMI_personality extension allows you to specify a personality for a glTF node representing a entity. This extension can be used in virtual worlds, where characters can have unique personalities, and interactions with them can be enhanced by providing additional information about their behavior and dialogue.

Example

To use the "OMI_personality" extension, you must first specify it in the extensionsUsed property of your glTF file.

```
{
    "extensionsUsed": ["OMI_personality"]
}
```

Next, apply the extension to a child node of the glTF file. The node's position and rotation data can be used to determine the location of the spawn point in the scene.

```
{
    "nodes": [
        {
            "name": "cat",
            "extensions": {
                "OMI_personality": {
                    "agent": "cat",
                    "personality": "#agent has a cheerful personality.",
                    "defaultMessage": "nya nya!"
                }
            }
        }
    ]
}
```

In the example above, the "OMI_personality" extension is applied to a node named "cat". The agent property is used to specify the type of agent associated with the node, in this case, it's a cat. The personality property describes the agent's personality, and the defaultMessage property is the message that the agent will send as a default.

Properties

The `defaultMessage` parameter is optional. The `agent` and `personality` options are required and provide context about the avatars name and default personality description.

	Type	Description
agent	`string`	The name of the agent or NPC.
personality	`string`	A default description of the personality of the agent allowing clients to inject that context into language model logic.
defaultMessage	`string`	A default message for this agent to initialize with.

JSON Schema

The OMI_personality extension is defined by the following JSON schema:

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "OMI_personality",
    "description": "An extension for the glTF format that defines a personality for a node and an endpoint whe
    "type": "object",
    "properties": {
        "agent": {
            "type": "string",
            "description": "The name of the agent associated with the node.",
            "maxLength": 128
        },
        "personality": {
            "type": "string",
            "description": "A description of the agent's personality."
        },
        "defaultMessage": {
            "type": "string",
            "description": "The default message that the agent will send on initialization."
        }
    },
    "required": ["agent", "personality"]
}
```

Implementation Details

The OMI_personality extension allows users to inject a unique personality into their virtual representations and adheres to a simple set of properties that aim to be compatible with lots of AI software to come. In the below example implementation, the data for personality is used to combine with the input from the user talking to the NPC. The final prompt is being sent to the OpenAI Davinci model to allow for completion of the agent's response.

```
    // Request coming from three.js frontend that is querying this endpoint making a call to the GPT-3 model.
    const data = await request.json();
    let prompt = data.inputs.personality;
    let prompt = data.Input.personality;

    let finalPrompt = prompt
        .replaceAll('#speaker', data.Input.Speaker)
        .replaceAll('#input', data.Input.Input)
        .replaceAll('#agent', data.Input.Agent)
        .replaceAll('#conversation', data.Input.Conversation)
        .replaceAll('undefined\n','' ).replaceAll('undefined','')
        .slice(-5000)

    const token = authorization.split(' ')[1];
    const postData = {
        prompt: finalPrompt    ,
        max_tokens: 500,
        stop : ["###"],
        temperature: 0.7,
    };

    // Make the first request to the Davinci model
    const davinciResponse = await fetch('https://api.openai.com/v1/engines/text-davinci-003/completions', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${token}`
        },
        body: JSON.stringify(postData)
    });
```

The final prompt in this examples follows a structure of:

```
Agent is cheerful and always willing to help with programming tasks. They are an entity that lives in a virtu
Speaker: Hello agent! Tell me about yourself
Agent:
```

The above example is a final prompt that is sent to OpenAI where the model completes what the "Agent:" would write to complete this conversation factoring in the personality data above the chat log.

In this example, the text is being passed back to the Three.js scene and rendered in the scene using a Text component.