Tutorials and Tools

Christiaan Verhoef

2024-10-08

Table of contents

Preface

In the world of logistics and serious game development, we often find ourselves juggling complexity and creativity. Here's where OpenWebUI comes in—our trusty tool to bring order and insight to that delightful chaos.

We're not going to bother with installation today (you're clever enough to have that sorted!). Instead, let's jump straight into what matters most: **How to use OpenWebUI to get things done**.

This book is your guide to mastering OpenWebUI, built step by step, with a focus on practical usage. Whether you're optimizing logistics routes or crafting engaging in-game interactions, you'll find ways to streamline your processes, automate tasks, and make your work both efficient and fun.

Ready? Breathe, get comfortable, and let's embark on this journey together. It's going to be an exciting ride.

---

1 Introduction

Welcome to the world of OpenWebUI, where creativity meets efficiency! Whether you're a **Serious Game Developer** or a **Logistics Researcher**, OpenWebUI is your new secret weapon. It allows you to build smart agents that automate complex tasks, freeing you to focus on innovation and strategy.

But this isn't just about creating bots. It's about crafting intelligent assistants that help streamline your workflow, optimize processes, and reduce the repetitive, mundane tasks that often bog us down. Imagine a team of digital helpers at your disposal, handling everything from resource management in your games to route optimization in logistics.

1.0.1 Who Is This Book For?

This book is designed for **Serious Game Developers** and **Logistics and Supply Chain Researchers**—fields that demand innovative, flexible solutions. Whether you're designing interactive NPCs for your latest game or testing complex logistics scenarios, OpenWebUI offers a set of tools to make your job easier and more efficient.

Here's what you'll learn:

- How to **set up and configure your agents**.
- How to **create custom workflows** for both game development and logistics.
- How to **test and refine your agents** to make them more responsive and capable.
- How to **integrate OpenWebUI** into your daily processes for seamless automation.

1.0.2 Why OpenWebUI?

OpenWebUI is a modular, flexible platform that grows with your needs. Whether you're working on a small game prototype or managing a massive supply chain, this tool adapts to your projects. The best part? You don't need to be a coding expert to get started. OpenWebUI makes it easy for you to spend less time wrestling with the tech and more time doing what matters most: creating, optimizing, and innovating.

Reminder: This book focuses solely on usage. You won't find long installation guides here—we assume you're ready to dive straight into the good stuff!

Let’s get started. By the end of this book, you’ll have the skills to build agents that work for you, leaving you more time for creativity and strategic thinking. Ready? Let’s dive in!

See Knuth (1984) for additional discussion of literate programming.

2 The Interface: Your Dashboard of Possibilities

When you launch OpenWebUI, the interface is designed to give you full control over the interaction between your agents and their tasks. The layout consists of three main panels: the **left panel** for managing active sessions and templates, the **center panel** for live agent interaction, and the **right panel** for adjusting chat controls and system parameters.

2.0.1 The Left Panel: Your Workspace and Navigation Hub

The **left panel** organizes your active and archived agent interactions. This section allows you to quickly move between different agent conversations and manage templates.

- **Active Sessions**: All active agent conversations are listed here. Click on a session to open and interact with the agent in real time.
- **Archived Conversations**: A space to revisit old conversations with agents, useful for refining agent performance based on previous tasks.
- **Templates**: Pre-configured agents that you can quickly launch, such as a logistics optimizer or NPC for games.

2.0.1.1 Example: Managing Sessions

1. Click on **Active Sessions** to view any agents you have running. For instance, if you have an agent performing a logistics task, you'll find its current progress and can issue commands directly.
2. To review a past interaction, click on **Archived Conversations** and load up the history of any agent's previous task.
3. If you want to quickly create an agent, click **Templates**, select **Logistics Optimizer**, and watch as the agent template is loaded and ready for interaction.

2.0.2 The Right Panel: Adjusting Chat Controls and System Parameters

The **right panel** is where you can modify the behavior of your agents in real time by setting the **system prompt**, adjusting various **advanced parameters**, and fine-tuning other controls such as temperature or token limits. These settings directly influence how your agent behaves and responds during conversations.

Here's a breakdown of what each option does:

2.1 System Prompt

The **System Prompt** is the initial instruction or context that guides the agent's overall behavior. It sets the tone for the agent's interactions and is one of the most critical settings.

- **Example**: `"You are a logistics optimization agent. Your goal is to minimize delivery time by adjusting routes based on real-time traffic data."`

  Effect: This ensures the agent's responses stay focused on optimizing routes. Changing the system prompt to include more detailed instructions, such as considering fuel efficiency or avoiding toll roads, will shift the agent's decision-making process.

2.2 Temperature

Temperature controls the randomness or creativity of the agent's responses. It affects how likely the agent is to deviate from predictable answers.

- **Range**: `0.0` to `1.0`
  - A **lower value** (closer to `0.0`) makes the agent more deterministic, meaning it will choose the most predictable and straightforward responses.
  - A **higher value** (closer to `1.0`) increases the variability in responses, allowing the agent to explore less obvious possibilities.
- **Example**:
  - At `0.2`, the agent will give highly predictable and consistent answers. Ideal for tasks like logistics optimization where precision is key.
  - At `0.8`, the agent will provide more creative, varied responses, useful when testing dynamic NPC dialogue in games or brainstorming creative solutions.

Effect: Higher temperatures introduce more randomness. For a logistics agent, using a high temperature could lead to more unconventional route suggestions, which might not always be optimal. Lowering the temperature ensures the agent sticks to tried-and-tested routes.

2.3 Top K

Top K limits how many options the agent considers at each step of its response generation. It controls how wide the agent's "vocabulary" is during interactions.

- **Range**: `1` to infinity (higher values mean more possible words to choose from)
  - A **lower value** (e.g., `10`) restricts the agent to a smaller, more focused set of choices.
  - A **higher value** (e.g., `50`) allows the agent to consider a broader range of possible next words or actions.
- **Example**:
  - Set **Top K** to `10` for a logistics agent. This ensures the agent sticks to the most reliable, often-used routes.
  - Set **Top K** to `100` for an NPC in a game to allow for more varied dialogue options, creating a more dynamic conversation.

Effect: Lower **Top K** values make the agent more predictable by narrowing its choices. In logistics, a low **Top K** ensures the agent only considers the most efficient routes. Higher **Top K** values allow the agent to explore more creative options, useful in contexts where variety is desirable.

2.4 Top P

Top P (nucleus sampling) controls how the agent selects from the top `P%` of most likely responses. This method can reduce reliance on overly common responses.

- **Range**: `0.0` to `1.0`
  - A **lower value** (e.g., `0.1`) means the agent will only select from the top 10% of the most likely responses.
  - A **higher value** (e.g., `0.9`) allows the agent to consider a much broader range of responses.

- **Example**:
  - **Top P** of `0.1` forces the agent to choose the safest, most common responses. Great for logistics where efficiency and reliability are key.
  - **Top P** of `0.8` allows for more variability, letting the agent experiment with routes or responses outside the norm.

Effect: Lower **Top P** values restrict the agent to the safest options, which works well for practical applications like logistics. In creative tasks, like generating dialogue for NPCs, a higher **Top P** value might produce more interesting and varied results.

2.5 Frequency Penalty

Frequency Penalty controls how much the agent penalizes words or phrases that it has already used. This prevents repetitive responses and encourages more varied output.

- **Range**: `0.0` to `2.0`
  - A **higher value** (e.g., `1.5`) makes the agent less likely to repeat words, encouraging it to find new ways of responding.
  - A **lower value** (e.g., `0.2`) allows the agent to repeat words more freely.
- **Example**:
  - Set **Frequency Penalty** to `1.2` for an NPC agent to avoid repetitive dialogue in-game conversations.
  - Set **Frequency Penalty** to `0.5` for a logistics agent, so it can repeat key terms like route names or instructions if necessary for clarity.

Effect: A higher penalty is useful in creative tasks to avoid repetitive answers, like generating dynamic NPC dialogue. In logistics or similar applications, setting a lower penalty ensures that important details (like road names) are repeated when necessary.

2.6 Max Tokens

Max Tokens limits how long the agent's responses can be by setting a cap on the number of tokens (which typically represent words or parts of words).

- **Range**: Any positive integer
  - A **lower value** (e.g., `50`) ensures the agent's responses are short and concise.
  - A **higher value** (e.g., `200`) allows for more detailed and elaborate responses.
- **Example**:
  - Set **Max Tokens** to `100` for logistics tasks to ensure the agent provides concise but detailed route recommendations.
  - Set **Max Tokens** to `150` for NPC agents in games to allow for more detailed explanations or conversations.

Effect: A lower **Max Tokens** value ensures responses remain short and focused, useful for tasks where brevity is important, like logistics. Higher values allow for longer, more elaborate explanations, great for creative writing or character dialogue.

2.7 Mirostat (Eta and Tau)

Mirostat is an advanced control mechanism that adjusts creativity in real-time as the agent generates text.

- **Eta**: Controls how aggressive the adjustment is.
  - Higher values mean more aggressive changes in creativity.
- **Tau**: The target perplexity, or how unpredictable the agent should be.
- **Example**:
  - Set **Mirostat Eta** to `1.0` and **Mirostat Tau** to `5` for a logistics agent to keep its responses stable and predictable.
  - Increase **Tau** for an NPC agent in a game to make dialogue more varied and unpredictable.

Effect: **Mirostat** helps fine-tune the unpredictability in long conversations. In logistics, using a low **Tau** ensures the agent stays consistent over longer interactions, while a higher **Tau** in a game allows for more dynamic and evolving conversations.

2.8 Use of Advanced Parameters in Specific Scenarios

2.8.1 Logistics Use Case:

- **System Prompt**: `"Optimize delivery routes with minimal delays, considering traffic and package urgency."`
- **Temperature**: Set to `0.4` for stable and predictable results.
- **Top K**: Set to `20` to limit route options to the most common choices.
- **Max Tokens**: Set to `100` to keep the responses concise.

2.8.2 Game NPC Use Case:

- **System Prompt**: `"You are a game character. Respond to player questions with wit and humor."`
- **Temperature**: Set to `0.8` for more creative and spontaneous replies.
- **Frequency Penalty**: Increase to `1.5` to avoid repetition in dialogue.
- **Max Tokens**: Set to `150` for more in-depth interactions.

2.8.2.1 Example: Fine-Tuning a Logistics Agent

Let's say you have an agent designed to optimize delivery routes, and you want to adjust how it prioritizes efficiency.

1. **System Prompt**: In the **System Prompt** field, type:
   - `"Prioritize optimizing delivery routes based on shortest time while considering real-time traffic delays."`
2. **Temperature**: Set the **Temperature** to `0.5`. This will make the agent's responses balanced—not too random but still adaptable to complex routing decisions.
3. **Top K & Top P**: Set **Top K** to `50` and **Top P** to `0.9` to allow the agent to consider a wide range of potential routes while still being focused on the most likely options.
4. **Max Tokens**: Set **Max Tokens** to `150` to keep the responses concise but still detailed enough for logistics decisions.
5. **Frequency Penalty**: Increase the **Frequency Penalty** to `0.6` to ensure the agent avoids repeating similar route suggestions over and over.

2.8.3 The Center: Interacting with Agents in Real-Time

The **center panel** is where you have direct conversations with your agents. This panel is where the agent's responses appear, and you can issue new commands or monitor task progress.

2.8.3.1 Example: Running Your Configured Agent

After adjusting the settings on the right panel, it's time to see the results in the center panel.

1. In the **center panel**, type:
   `"Optimize delivery routes for trucks in the downtown area, considering current traffic conditions."`
2. Press **Enter** and observe the agent's response, which should consider your new system prompt and settings.
3. Simulate a traffic delay by typing:
   `"There is a traffic jam on Main Street. Recalculate delivery routes."`
4. The agent will adjust the routes in real-time, providing alternative suggestions based on the adjusted parameters like **Temperature** and **Top K**.

3 Crafting Your First Agent: Getting Started with Bots

The **Workspace** in OpenWebUI is designed to provide a structured environment for interacting with AI models, managing content, and customizing workflows. It is composed of five main tabs, each catering to different functionalities:

3.1 1. Models

The **Models** tab is where you can create, modify, and manage AI models.

You can create, edit, and fine-tune models, including those from external sources like Ollama or OpenAI. It supports functionalities like tagging, cloning, sharing, and exporting model files. You can also attach tools and documents to enhance the model's capabilities, allowing integration with Retrieval Augmented Generation (RAG) for document-based queries

It supports: - Model creation, editing, and fine-tuning - Integration with external APIs (Ollama, OpenAI) - Attaching documents and tools to models - Model tagging, cloning, and sharing - Adjustable model parameters (e.g., temperature, seed)

3.2 2. Prompts

The **Prompts** tab allows for the management and customization of predefined prompts, making interactions with models more efficient. You can set custom system prompts or load prompt presets, making it easy to engage the model in a structured way. This tab may also support prompt variables like {{CURRENT_DATE}}, making interactions dynamic This includes: - Creating and storing prompt presets - Utilizing dynamic variables in prompts (e.g., date, user name) - Setting up system prompts for specific interactions

3.3 3. Documents

The **Documents** tab is primarily used for integrating knowledge sources with models. The Documents tab integrates with RAG, enabling models to use document content for more informed responses. You can upload and organize documents that the model references during conversations, improving the quality of interactions based on specific content You can: - Upload documents for reference in conversations - Use document-based queries via the RAG (Retrieval Augmented Generation) feature - Organize and manage document sources for quick access

3.4 4. Tools

The **Tools** tab provides access to various extensions and utilities that enhance the model's capabilities. The Tools tab allows you to assign various tools to your models. These tools can range from image generation engines to custom Python code, depending on the task. For example, a Python code editor could be integrated here to extend the model's functionality with custom code execution.

This includes: - Image generation, text processing, or code execution tools - Custom-built or third-party tools that can be integrated with your model workflows

3.5 5. Functions

The **Functions** tab allows you to define specific pipelines and actions that models can perform. Functions allow you to define specific actions or pipelines for the models. This can include things like filters, pipes, and even custom user-defined functions to control the flow of data through the model. Functions like valves can be used to manage configuration options or user controls dynamically.

It is useful for: - Creating custom workflows using functions like filters or pipelines - Setting up configurations for model actions based on user interactions

---

3.6 Next Steps:

Let's now break down each section, starting with **Models**, to explore how this tab functions in more depth.

4 Step-by-Step Guide: Using the Models Tab in OpenWebUI

The **Models** tab in the Workspace provides a structured environment for managing AI models, with exclusive integration to models from the Ollama Library. This guide will walk you through the process step by step, explaining how each feature works, along with examples to help you follow along.

4.0.1 a. Model Creation and Selection

4.0.1.1 Step-by-Step:

1. **Open the Models Tab**:
    - In the Workspace, you'll find a tab labeled **Models**. Click on it to begin managing your AI models.
2. **Choose a Model from the Ollama Library**:
    - Once inside, you will see a list of pre-trained models from the Ollama Library. Each model has been optimized for specific tasks such as language understanding, decision-making, and data analysis.
    - **Click** on the model you want to use. For a logistics-related project, choose a model that can handle data well. For a game development project, select a model that can generate dynamic conversations or behaviors.
3. **Loading the Model**:
    - After selecting the model, it will load into your workspace, ready for use.

llama3.2:1b llama3.2:3b llama3.1:8b llama3.1:70b llama3.1:405b gemma2:2b gemma2:9b gemma2:27b qwen2.5:0.5b qwen2.5:1.5b qwen2.5:3b qwen2.5:7b qwen2.5:14b qwen2.5:32b qwen2.5:72b phi3.5:3.8b nemotron-mini mistral-small:22b deepseek-coder-v2:16b deepseek-coder-v2:236b command-r:35b codegemma

4.0.1.2 Examples:

- **Logistics Example**: If you're optimizing delivery routes or analyzing supply chain data, you might select a model like `Llama 2`, which is excellent at handling structured data and making predictions based on historical delivery times.
- **Game Development Example**: When developing an NPC for a role-playing game, you might choose a conversational model from the Ollama Library that can dynamically respond to player choices, simulating realistic conversations in the game.

4.0.2 b. Model Fine-Tuning

4.0.2.1 Step-by-Step:

1. **Adjust Model Settings**:
    - Once your model is loaded, you can fine-tune its performance by adjusting key parameters.
2. **Parameters to Adjust**:
    - **Temperature**: This setting controls how creative or random the model's responses are.
        - Set the temperature **lower (0.2-0.3)** for more predictable and precise responses (e.g., for logistics tasks).
        - Set the temperature **higher (0.7-0.9)** for more creative, dynamic answers (e.g., for game characters' dialogues).
    - **Context Length**: This defines how much prior information the model remembers when generating responses. In scenarios where the model needs to handle ongoing conversations or long-term data, increase this value.
    - **Frequency Penalty**: This adjusts how often words are repeated in the model's responses. A high frequency penalty ensures concise outputs, useful for reports and logistics.

4.0.2.2 Examples:

- **Logistics Example**: If you are predicting how delivery times vary with weather, you might set the **Context Length** higher to ensure the model remembers recent weather data and adjusts its predictions accordingly.

- **Game Development Example**: By adjusting the **Temperature**, you can make NPC dialogues more unpredictable, making the interactions feel more immersive for players. A higher temperature will make NPCs respond in a variety of ways, while a lower temperature will keep conversations focused and repetitive.

4.0.3 c. Model Attachments (Documents and Tools)

4.0.3.1 Step-by-Step:

1. **Attach Documents**:
   - You can upload documents like route maps, inventory data, or game scripts that the model will reference when generating responses.
2. **Attach Tools**:
   - Attach external tools, such as a Python script or an image generation tool, that enhance the model's functionality. For instance, if your model is processing large amounts of data, you might integrate a script that automates part of the process.

4.0.3.2 Examples:

- **Logistics Example**: If you're managing global supply chains, you could upload historical weather data, shipment schedules, and other logistical information. The model can then reference this data and predict how changes (e.g., fuel prices or weather conditions) will affect delivery times.
- **Game Development Example**: You can upload a storyline document with character backstories to the model. When players interact with NPCs, the model can use this document to provide responses that are consistent with the game's narrative, ensuring the characters remain true to their roles.

4.0.4 d. Model Cloning and Tagging

4.0.4.1 Step-by-Step:

1. **Cloning a Model**:
    - If you want to experiment with different configurations, you can clone a model. Cloning keeps the original model intact while creating a copy that you can modify. This is useful when testing out new settings or attaching different tools.
2. **Tagging Models**:
    - Use tags to organize your models. For instance, tag models based on the projects they are used for (e.g., "Logistics Optimization" or "NPC Conversations").

4.0.4.2 Examples:

- **Logistics Example**: You might create separate cloned models for each mode of transportation (e.g., truck, ship, and airplane). This allows you to fine-tune each one for specific conditions, like shipping route preferences or cost factors.
- **Game Development Example**: Clone an NPC dialogue model and adapt it for different environments. For example, one cloned model could handle desert environments, and another could generate dialogue for underwater levels. Tagging these models makes it easy to switch between them during development.

4.0.5 e. Model Presets

4.0.5.1 Step-by-Step:

1. **Save Model Configurations**:
    - After fine-tuning a model and attaching relevant documents or tools, you can save this configuration as a preset. This allows you to quickly reload the same setup later.
2. **Switch Between Presets**:
    - Presets can be helpful if you frequently switch between tasks, such as moving between analyzing inventory data and optimizing delivery routes.

4.0.5.2 Examples:

- **Logistics Example**: Save a preset for **warehouse inventory analysis** where the model focuses on stock levels and reordering times. You could also create another preset for **route optimization** with the model optimized for predicting delivery delays based on historical traffic patterns.
- **Game Development Example**: Save presets for different types of NPC interactions. One preset could focus on **merchant NPCs** that trade with the player, while another preset could focus on **quest-giving NPCs**, referencing different storyline documents for varied conversations.

4.0.6 f. Ollama Model Management

4.0.6.1 Step-by-Step:

1. **Download Models from Ollama**:
   - You can easily download models from the Ollama Library. These models are optimized for various tasks, from decision-making to conversational simulations.
2. **Upload GGUF Models**:
   - For advanced users, the Workspace allows you to upload GGUF models, which are customized models tailored to specific use cases, like logistics planning or game simulation.
3. **Monitor Model Performance**:
   - Keep an eye on how your models perform. The Workspace provides feedback on response times and error rates, helping you fine-tune their performance further.

4.0.6.2 Examples:

- **Logistics Example**: Download a decision-making model from Ollama that can predict the most cost-effective shipping routes based on real-time fuel prices and delivery demands.
- **Game Development Example**: Download a language model that specializes in generating NPC conversations. Then upload custom character documents that help the model create unique, contextually relevant dialogues based on player actions.

5 Refining Agent Behavior: Making Them Smarter

- Adjusting parameters for more effective performance
  - Teaching your agent new tricks: Customization
  - Troubleshooting common agent issues
  - Using feedback loops to improve agent responses

Once you've created a basic agent in OpenWebUI using the **Workbench Tool**, the next step is to refine its behavior to make it smarter and more adaptable. This involves adjusting the agent's responses, setting up triggers, and fine-tuning how it interprets and acts on inputs.

Let's walk through the process of refining your agent's behavior step by step, with a focus on improving its decision-making capabilities and adaptability.

5.1 Step 1: Set a Clear System Prompt

The **system prompt** is where the agent's core instructions are defined. This is what tells the agent how to behave overall, so getting this right is the first step in improving its intelligence. To make your agent smarter, you need to set detailed instructions that account for the nuances of the tasks you expect it to handle.

- **Refinement Tip**: As your agent's role becomes more complex, expand the system prompt to cover more specific conditions or priorities. For example, if your agent is managing multiple tasks (like optimizing routes while managing driver schedules), add this layered instruction in the system prompt.

5.2 Step 2: Fine-Tune Advanced Parameters

In the **Advanced Parameters** section of the **right panel**, you can fine-tune the agent's behavior. This allows you to control the agent's creativity, consistency, and response predictability, which are key for making the agent smarter.

- **Temperature**: Lower the temperature if you want the agent to give more focused and consistent responses. Raise it slightly if you want the agent to explore creative solutions when faced with ambiguity.
- **Top K & Top P**: These parameters affect how broad or narrow the agent's decision-making scope is. Narrow these values (reduce **Top K** and **Top P**) to make the agent more deterministic and focused on the most reliable responses. Adjust them upwards to encourage more creative, exploratory answers.

5.3 Step 3: Establish Behavioral Rules

Smart agents need clear rules on how to react to specific situations. The **behavior rules** are like a set of conditions and responses that you can define to make the agent adaptive to various scenarios.

- **Triggers**: Set up triggers for certain inputs or external factors. For example, if you're working on a logistics agent, a trigger could be "if traffic delay exceeds 30 minutes, recalibrate the route."
- **Actions**: Pair triggers with actions that the agent should take once certain conditions are met. This can include recalculating data, fetching additional information, or issuing a response. For an NPC in a game, you could add an emotional reaction based on player interaction.

5.4 Step 4: Leverage Feedback Loops

To make your agent continuously improve, set up **feedback loops**. A feedback loop allows the agent to learn from its actions and adjust its behavior dynamically. For instance, after receiving input from a user, the agent can ask clarifying questions, ensuring its next action aligns with the desired goal.

- **Iterative Learning**: Make the agent evaluate its past responses. For instance, you could introduce a rule that allows the agent to reevaluate whether its initial response was optimal and suggest improvements on the go.

5.5 Step 5: Incorporate External Knowledge and Tools

Making your agent smarter involves giving it access to external data or tools. If your agent needs to perform more complex tasks like analyzing traffic data or fetching up-to-date pricing information, integrate external tools or documents via the **Tools Workspace** or **Documents Workspace**.

- **Documents**: If the agent needs access to specific data sets or files, upload these documents to the **Documents Workspace** and link them to the agent.
- **Toolkits**: For example, linking APIs like real-time traffic data for a logistics agent or databases for a research agent will give it additional context, enabling it to make smarter decisions based on fresh, real-world inputs.

5.6 Step 6: Test and Refine

Finally, constantly **test** your agent's responses to different scenarios to ensure that it's acting intelligently. You can simulate real-world inputs or dynamic game interactions to evaluate how well the agent handles complex tasks.

- **Iterative Testing**: Each time you make adjustments to the agent's behavior rules, system prompt, or advanced parameters, run new tests. Pay attention to how the agent adapts to various situations, and keep refining its settings until it performs optimally.

5.6.1 Final Thoughts

Refining your agent's behavior is an iterative process that requires ongoing adjustment and testing. By setting clear system prompts, fine-tuning advanced parameters, establishing behavior rules, leveraging feedback loops, and incorporating external tools, you'll develop an intelligent, adaptable agent capable of handling complex tasks effectively.

6 Testing Agents: Trial and Error for Perfection

- Setting up test cases for logistics scenarios
  - Running in-game simulations with NPC bots
  - Gathering feedback: How to observe agent behavior
  - Tweaking based on test results: Iterative improvements

7 Testing Agents: Trial and Error for Perfection

- Setting up test cases for logistics scenarios
- Running in-game simulations with NPC bots
- Gathering feedback: How to observe agent behavior
- Tweaking based on test results: Iterative improvements

After you've configured and refined your agent in OpenWebUI, it's time to put it to the test. Testing is an essential part of ensuring that your agent behaves as expected and delivers the desired results. This is where you identify areas of improvement and fine-tune its performance. It's all about iteration—using trial and error to perfect your agent's behavior over time.

7.1 Step 1: Define Clear Test Scenarios

Before diving into testing, it's crucial to define what scenarios you want your agent to handle. These should reflect real-world use cases where the agent will be deployed. Define different conditions and interactions that your agent might face during its operation.

- **Logistics Agent Example**: If your agent is managing delivery routes, you could test how it handles sudden traffic delays or requests for optimizing multiple delivery points at once.
- **Game NPC Example**: If you're working with an NPC agent, define scenarios like how it reacts when a player takes aggressive actions or asks certain questions.

By outlining your scenarios first, you have a structured approach to evaluate the agent's behavior.

7.2 Step 2: Run Live Interactions

Once your test scenarios are ready, it's time to start running **live interactions** with your agent. These real-time interactions will show how well your agent responds to commands and conditions based on the system prompts, rules, and parameters you've configured.

1. Open the **center panel** in OpenWebUI to begin interacting with your agent.
2. Issue a command or input relevant to your test scenario. For example:
    - `"Optimize the fastest route considering a 20-minute traffic delay on Route A."`
    - `"Respond to the player's hostile question with a defensive answer."`
3. Watch how your agent processes the input and generates a response. Pay close attention to how closely the response aligns with your expectations.

Pro Tip: Record the agent's responses for each scenario. This will help you track its behavior across multiple test sessions.

7.3 Step 3: Adjust and Rerun Tests

It's unlikely that your agent will perform perfectly on the first try. This is where trial and error come into play. Based on the agent's performance, you'll need to revisit its configuration and make tweaks to improve its behavior.

- **Refine System Prompts**: If the agent's response is off-track, consider revising the system prompt to give clearer or more focused instructions.
- **Modify Parameters**: Adjust parameters like **temperature**, **Top K**, and **frequency penalty** to fine-tune the agent's creativity, response consistency, and overall behavior.

After making adjustments, run the tests again to see how the agent's behavior has improved. Repeat this process as often as necessary to get closer to the desired result.

7.4 Step 4: Simulate Edge Cases

Beyond standard test scenarios, it's important to simulate edge cases—situations that might be rare but still possible in the real-world use of your agent. These cases will test the limits of the agent's adaptability.

- **Logistics Edge Case**: Test how the agent responds to a sudden closure of all major routes, forcing it to find a highly unconventional route.
- **NPC Edge Case**: Simulate an extreme or unusual player interaction, like repeated aggressive behavior or a complex multi-step question.

By running these more complex or unexpected tests, you'll uncover potential weaknesses in your agent's logic or adaptability.

7.5 Step 5: Gather Feedback and Iterate

Testing doesn't end with a single round of adjustments. Use feedback loops from your testing sessions to continually improve your agent. If you're testing in a collaborative environment, gather feedback from colleagues or users to get different perspectives on the agent's performance.

- **Feedback Questions**:
  - Was the agent's response clear and aligned with the task?
  - Did the agent handle unexpected scenarios gracefully?
  - Were the responses consistent and reliable, or too random?

7.6 Step 6: Monitor Long-Term Performance

Finally, once your agent is deployed in a live setting, it's important to monitor its long-term performance. This includes checking how well the agent handles real-world data and conditions over extended periods.

- **Automated Testing**: Consider setting up automated test cases that run at regular intervals to ensure that your agent continues to perform optimally. These tests can track changes in external conditions (such as updates in the logistics network or game dynamics) and help you proactively address any issues.

7.6.1 Final Thoughts

Testing your agent is not a one-time task but a continuous process of refinement. By using structured testing methods, adjusting settings through trial and error, and monitoring performance over time, you can create a smarter, more responsive agent. The key to perfection is iteration—fine-tuning the agent until it performs consistently and intelligently in real-world scenarios.

8 Integrating OpenWebUI into Your Workflow

- Embedding agents into serious games
- Applying OpenWebUI to supply chain and logistics research
- Workflow automation: Using bots to handle repetitive tasks
- Real-world use cases: From theory to practice

9 Real-World Examples and Case Studies

- Optimizing game dialogues with NPC agents
  - Using agents to simulate supply chain disruptions
  - Logistics route optimization: Step-by-step case study
  - Applying OpenWebUI in academic research

10 Scaling and Maintenance: Keeping Your Agents Up-to-Date

- Scaling agents for larger projects
  - Monitoring agent performance over time
  - Regular updates and maintenance for optimal performance
  - How to retire outdated agents

11 Final Thoughts and Future Potential

- The evolving role of AI in logistics and game development
  - Expanding beyond the basics: What's next for you?
  - How to stay updated on new OpenWebUI features
  - Encouragement to keep experimenting and learning

12 Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.