Supplementary Documentation: Autonomous NPC Framework

Team: PixelMinds

1. Project Summary

This project is a sophisticated framework for creating dynamic, in-character Non-Player Characters (NPCs) using a multi-agent system powered by Autogen. Our system allows an NPC to perceive its game world, recall story lore, understand its environment by reading game code, and respond to the player with structured actions, all while maintaining a consistent personality.

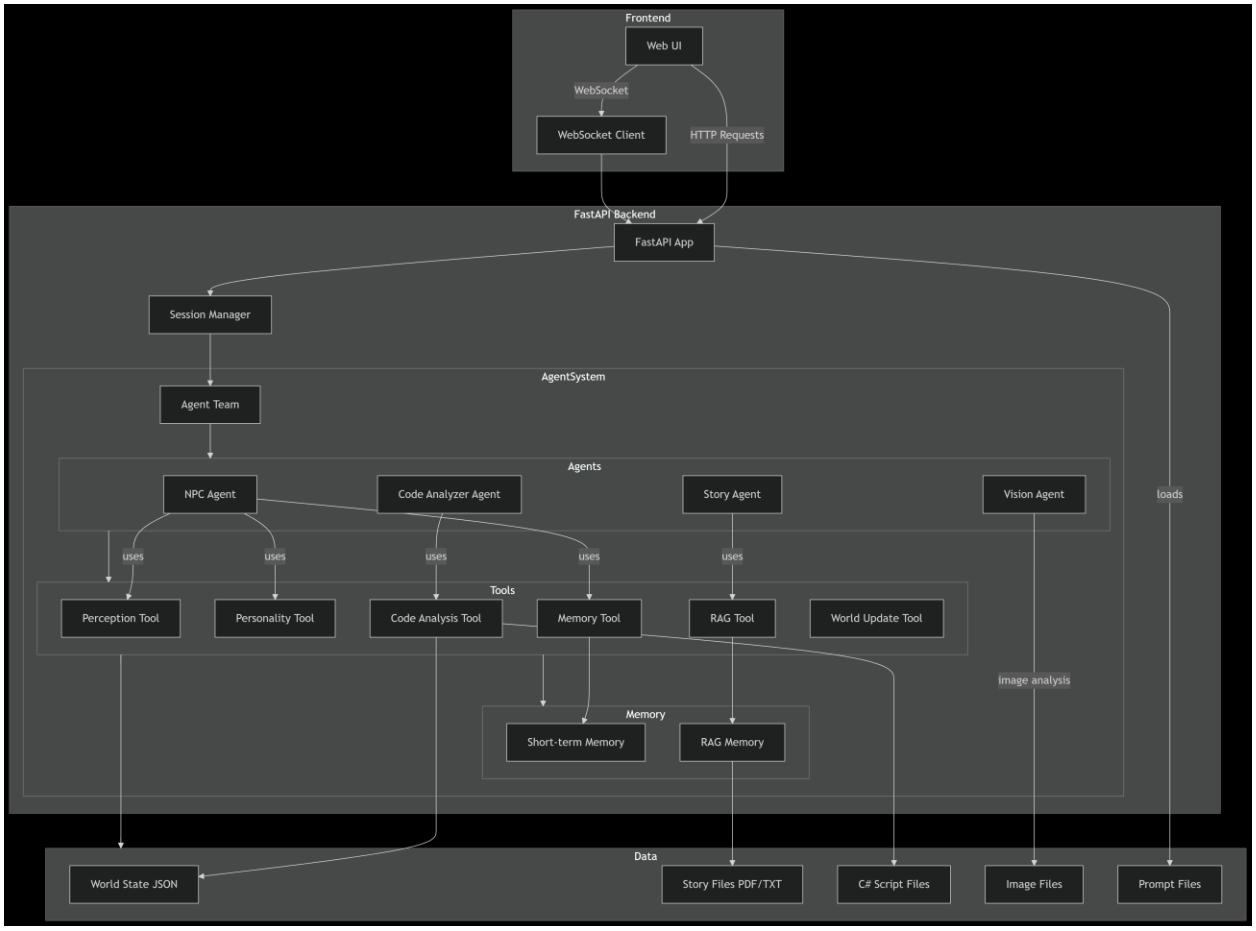
2. Core Architectural Design

Our design philosophy is centered on **specialization and decoupling**. Instead of a single, monolithic AI trying to do everything, we've created a team of specialist agents that collaborate. The NPC's output is a structured JSON object, decoupling its "brain" from the game engine, which can simply execute the provided actions and animations.

High-Level Architecture Flow

The system follows a clear, predictable flow for handling user interaction:
A text-based representation of the flow:

User Input * Group Chat * Selector Agent * Specialist Agent (Story, Code, Vision) * NPC Agent (Synthesizer) * JSON Output



3. Agent Roles and Responsibilities

Our multi-agent team consists of four distinct agents, each with a specific purpose defined by its system prompt.

a. The NPC Agent (`npc_agent`)

- **Role:** The "face" of the operation. This is the only agent that speaks directly to the user.
- **Function:** It synthesizes information provided by other specialist agents, combines it with its own personality (defined in `npc_system_message.txt`), and generates the final, structured JSON response. It manages mood, actions, and animations.
- **Key File:** `prompts/npc_system_message.txt`

b. The Story Agent (`story_agent`)

- **Role:** The Lore Master.
- **Function:** Handles questions related to the game's narrative, background, and lore. It uses a Retrieval-Augmented Generation (RAG) tool (`rag_tool`) connected to a ChromaDB vector database to find relevant information from provided story documents (.pdf, .txt).
- **Key Files:** `core/memory.py`, `prompts/story_system_message.txt`

c. The Code Analyzer Agent (`CodeAnalyzerAgent`)

- **Role:** The World Observer.
- **Function:** This agent has the unique ability to read the game's C# script (`tools.py`) to understand the static layout of the world (e.g., store sections, character lists). It also reads a `world_state.json` file to get the dynamic status of objects (e.g., if a door is open or closed). Its output is pure JSON data for the NPC agent to interpret.
- **Key Files:** `core/tools.py`, `prompts/code_analyzer_system_message.txt`

d. The Vision Agent (`VisionAgent`)

- **Role:** The Eyes.
- **Function:** When a user provides an image (e.g., a screenshot from the game), this agent describes what it sees in a single, concise paragraph. This visual context is then passed to the NPC agent.
- **Key File:** `prompts/vision_agent_system_message.txt`

4. Key Technical Highlights

- **Structured JSON Output:** The `NPCResponse` Pydantic model (`core/config.py`) enforces a strict output format. This makes integration with a game engine (like Unity or Unreal) straightforward, as the engine only needs to parse a predictable JSON structure for actions (`MOVE, PICKUP`), mood changes, and animations

- **Dual Memory System (`core/memory.py`):**
  1. **Short-Term Memory:** A simple list-based memory for tracking the immediate conversation context.
  2. **Long-Term RAG Memory:** A persistent ChromaDB vector store for deep knowledge retrieval about the game's story, allowing the NPC to answer detailed lore questions accurately.
- **Dynamic and Static World State (core/tools.py):**

  The system cleverly separates the world's state into two types:

  - **Static State:** Layout, permanent features, and character definitions are read directly from a C# script, representing the game's hard-coded design.
  - **Dynamic State:** The status of interactable objects (e.g., a door's `open/closed` state) is stored in `world_state.json`. The NPC can not only read this file but also *write* to it via the `update_world_state` tool, allowing its actions to have a persistent effect on the game world.
    Consequently, the next time the *CodeAnalyzerAgent* inspects the environment, it will report the new, altered state.
- **Advanced Prompt Engineering:** Each agent's behavior is meticulously crafted through its system message text file in the `/prompts` directory. The NPC's prompt is particularly complex, defining persona, rules for factual accuracy, and the required JSON output structure with examples.

5. Core Technology: Microsoft Autogen

This project is built on the **Microsoft Autogen** framework. Autogen is an open-source library designed for orchestrating and automating workflows using multiple, conversational LLM agents. We chose Autogen because it provides a robust foundation for defining specialized agents and managing their interactions within a "group chat" paradigm, which was a perfect fit for our architectural goals.

Official Autogen Resources:

- **Official Documentation:** https://microsoft.github.io/autogen/
- **GitHub Repository:** https://github.com/microsoft/autogen
- **Research Paper (on arXiv):** https://arxiv.org/abs/2308.08155