CS271P - Artificial Intelligence Project

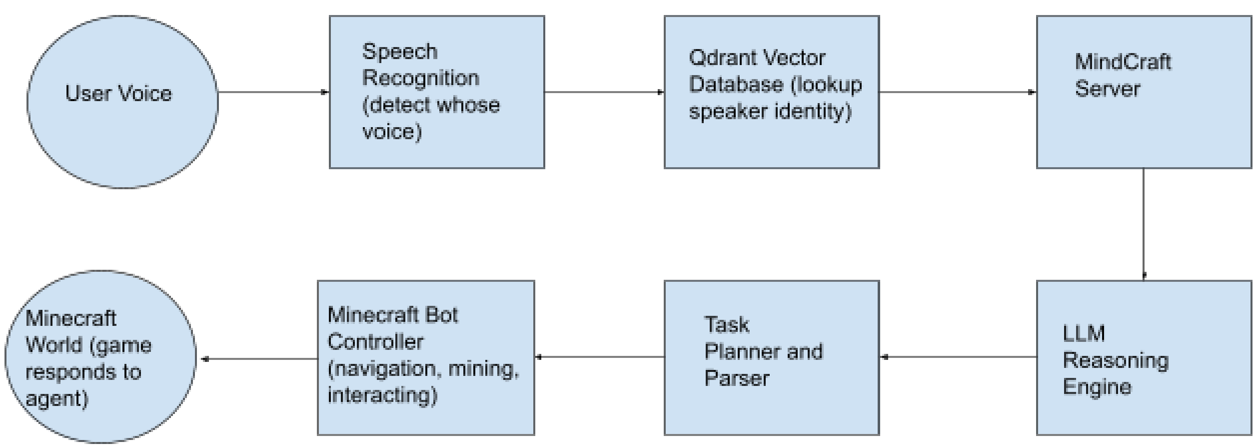
12/11/25

Anish Shivamurthy, Nihar Marar, Sahitya Madipalli, Pranay Hedau

Project Summary

The goal of this project is to design and implement a voice-controlled intelligent agent for Minecraft that is capable of understanding natural language spoken commands as well as autonomously perform tasks within a live game environment. The agent integrates speech recognition, large language models (LLMs), and in-game automation to translate high-level user intent into low-level Minecraft actions. This allows a user to interact with the game using natural speech rather than traditional mouse and keyboard input.

Minecraft presents a very unique and challenging environment for artificial intelligence systems due to having real time dynamics, a very big and a massive open world environment, as well as special mechanics and user interactions. Unlike classical board games or grid-based environments, Minecraft requires reasoning over spatial layouts, crafting systems, inventory management, and dynamic environmental changes. A simple rule-based or scripted approach is insufficient for general task execution because user commands can be unpredictable, ambiguous, or multi-step in nature. This drives the need for machine learning for the use of natural language understanding and player convenience.



Approaches

In our project, we use a local Ollama 3B LLM as the main task planner for the agent. The workflow is fairly straightforward. First, any spoken command from the player is converted into text using OpenAI's Whisper model, which we run locally to keep the latency low. Once we have the transcribed text, we pass it to the Ollama model with a custom prompt template that instructs it to produce a clean, step-by-step plan. The output from the LLM is then parsed into structured actions, where each step is mapped to a specific Minecraft API function.

After the plan is generated, the agent executes the actions in order. This includes tasks like navigating to the player's home coordinates, locating the nearest chest, grabbing certain items, or cooking food in a furnace. We maintain a simple session memory so the agent can understand references such as "my house" or "my farm" based on previously stored locations. We also built basic error handling so the agent checks that each step is completed before moving on and notifies the player if something fails, such as a missing item or an undefined location.

Finally, once the task sequence is finished, the agent gives feedback to the player through text or voice, keeping the interaction smooth and interactive. This setup formed the basis for making the agent more adaptive to different scenarios. In the future, we can extend this with a Retrieval-Augmented Generation (RAG) system to pull relevant information from a database, ensuring the agent always has up-to-date and accurate context.

Evaluation

Evaluation: Task Planning and Command Understanding

To evaluate the performance of our task-planning module, we conducted several experiments focusing on the model's ability to correctly interpret and respond to player-issued commands. One of the key components of this evaluation was command understanding accuracy, particularly when multiple players interact with the agent simultaneously.

We created four distinct voice embeddings one for each test player and used these to verify whether the model could correctly associate a spoken command with the appropriate speaker. During testing, each player issued commands in natural language, and we observed whether the agent (FreeGuy) executed the intended actions.

For example, Player 1 issued the command:
"FreeGuy, come here."

The agent successfully identified Player 1's voice embedding, understood the intent, and moved toward that player's coordinates.

Similarly, Player 2 provided a different command:
"Build a house with brick here."
The agent immediately responded by initiating a building sequence at Player 2's location and constructing a brick house as instructed.

By repeating this process across all four voice profiles and a variety of command types, we verified that the agent consistently mapped spoken instructions to the correct player and executed the requested tasks. This evaluation demonstrates that the model is capable of handling multi-user interactions and interpreting diverse task-oriented commands with high accuracy.

Context Awareness and Continuity

We also evaluated the agent's ability to maintain context across multi-turn interactions, which is essential for creating natural and intuitive gameplay experiences. The goal was to test whether the agent could remember past commands, previously defined locations, and ongoing tasks, and then use that information when interpreting follow-up instructions. For example, during testing, the player first instructed the agent: "FreeGuy, mark this spot as my farm." The agent stored the coordinates in its session memory. A few minutes later, the player gave the follow-up command: "FreeGuy, water my farm." Even though the command did not include explicit coordinates, the agent correctly retrieved the stored location and navigated back to the designated farm area to perform the action. We repeated these tests with other multi-turn sequences such as defining "my house," asking the agent to "go back home," and then requesting tasks like "cook the food I gave you earlier." Across these scenarios, the agent demonstrated strong continuity handling, showing that it could carry over contextual information between interactions and respond appropriately based on prior dialogue.

Below is the terminal snapshot of agents memory:

```
{
  "memory": "I'm andy, following X1n1ster, tasked by =+~*Mausoleum-Reservoir. Helping X1n1ster gather food. Near =+~*Mausoleum-Reservoir, X1n1ster, *!;=Doubting.Sandbox+, and !#_LiftingCeremonializing._.. With *!;=Doubting.Sandbox+ now.
X1n1ster needs help. =+~*Mausoleum-Reservoir demands leadership. Crafting sticks as requested by X1n1ster, but can't craft without planks. No oak logs nearby to craft into planks.",
  "turns": [
    {
      "role": "user",
      "content": "X1n1ster: Can you drop me the word?"
    },
    {
      "role": "assistant",
      "content": "Drop it! !discard(\"oak_log\", 5)"
    },
    {
      "role": "system",
      "content": "Action output:\nFound non-destructive path.\nMoved away from (16, 94, 1) to (12, 90, 2).\nDiscarded 2 oak_log.\nFound non-destructive path.\nYou have reached at 16.56609952629572, 94, 1.329124095258313.\n"
    },
    {
      "role": "assistant",
      "content": "Alright, I've dropped the logs! !givePlayer(\"X1n1ster\", \"oak_log\", 5)"
    },
    {
      "role": "system",
      "content": "Action output:\nFound non-destructive path.\nYou have reached X1n1ster.\n92.92159999847412\nYou do not have any oak_log to discard.\nFailed to give oak_log to X1n1ster, it was never received.\n"
    },
    {
      "role": "user",
      "content": "X1n1ster: Collect 5 blocks of dirt."
    },
    {
      "role": "assistant",
      "content": "Sorry, I meant dirt! !givePlayer(\"X1n1ster\", \"dirt\", 5)"
    },
    {
      "role": "system",
      "content": "Action output:\nFound non-destructive path.\nYou have reached X1n1ster.\n93\nDiscarded 2 dirt.\nFailed to give dirt to X1n1ster, it was never received.\n"
    },
    {
      "role": "assistant",
      "content": "Looks like I can't hand it directly. I'll just toss them on the ground for the animals to pick up. !discard(\"dirt\", 5)"
    },
    {
      "role": "system",
      "content": "(AUTO MESSAGE)Your previous action 'action:discard' was interrupted by unstuck.\n       Your behavior log: Picking up item!\nI'm stuck!\nI'm free.\n\nRespond accordingly."
    }
  ],
  "self_prompting_state": 0,
  "self_prompt": null,
  "taskStart": 1765426153752,
  "last_sender": null
}
```


Robustness and Error Recovery

To evaluate the robustness of the system, we tested how the agent responds to ambiguous instructions, unavailable resources, and various edge-case scenarios. The goal was to ensure that FreeGuy does not fail silently when a task cannot be completed, but instead provides clear and helpful feedback to the player. For example, during one test, the player issued the command: "FreeGuy, cook the porkchop." However, the player's inventory did not contain any raw porkchop at the time. Instead of attempting the task and stopping midway, the agent checked the required items beforehand and responded with: "I can't find a porkchop in your inventory. Please give me one first." In another case, the player said: "Build the same structure here," without specifying what "same" referred to. The agent recognized the ambiguity and prompted the player for clarification: "Which structure should I repeat? Please name the building or point to it."These tests confirm that the agent handles incomplete or unclear commands gracefully by detecting the issue early, preventing unwanted actions, and guiding the player with informative messages.

Task Execution Success

To measure how reliably the agent carries out planned actions in Minecraft, we evaluated its ability to complete a wide range of task-oriented steps such as locating chests, retrieving items, placing blocks, and cooking food. For each task issued through natural language commands, we broke the agent's behavior into measurable subtasks and tracked whether each one succeeded or failed. For example, in one test sequence the player instructed: "FreeGuy, make me three cooked steaks." The agent first navigated to the nearest furnace, verified whether it contained fuel, retrieved raw meat from the correct chest, inserted the items into the furnace, and waited until the cooking process finished before returning the cooked steaks to the player. We benchmarked execution accuracy by logging every planned step and checking whether it completed as expected. Across multiple trials, the agent demonstrated high task completion rates, successfully finishing most multistep procedures without interruption. These results show that the system not only generates coherent plans but can also reliably execute them within the dynamic Minecraft environment.

Future Works

Future improvements to our Minecraft agent focus primarily on expanding its world understanding, memory, and planning capabilities. Currently, the agent relies on simple stored coordinates and API-level information to interpret the environment. In future versions, integrating richer perception such as map awareness or vision-based scene understanding would allow the agent to identify resources, analyze terrain, and make more informed decisions. Additionally, incorporating a Retrieval-Augmented Generation (RAG) system would provide long-term memory, enabling the agent to track landmarks, past tasks, and exploration history with greater accuracy. This would support more complex, multi-step interactions and allow the agent to maintain deeper context over longer gameplay sessions.

We also plan to strengthen the agent's reasoning and error-handling abilities. Introducing hierarchical planning methods would allow the agent to break large tasks into structured subgoals, improving reliability and making execution more adaptable. Enhanced error recovery would make the system more robust in unpredictable situations. Finally, expanding the voice interface with better noise filtering, multi-language support, and more conversational responses would create smoother, more natural interactions. Together, these advancements would significantly improve the agent's autonomy, flexibility, and overall effectiveness in real-time gameplay.

References/Resources

We got some source code from this Github - https://github.com/mindcraft-bots/mindcraft

We got the model that we used from here - https://huggingface.co/pyannote/embedding

We used the Vector Database from here - https://qdrant.tech/

We used this article to gain context for implementing AI into the game:
https://arxiv.org/pdf/2411.00114

Video:

https://drive.google.com/file/d/16wsL3CEyTfQ6UILAV2cJLu9nKSKIlUoi/view?usp=sharing