**AI-Powered Text-to-Image Drawing Agent Using MS Paint**

**1. Introduction**

The goal of this project was to develop an AI-powered agent capable of interpreting user-provided text or voice commands and generating corresponding visual drawings automatically in Microsoft Paint. Unlike traditional text-to-image models trained on large datasets, this system focuses on real-time automation, combining Natural Language Understanding and GUI automation to simulate intelligent drawing behavior.

Users can issue commands such as:

* “Draw a house”
* “Add a tree next to the house”
* “Add sun and person”
* “Clear canvas”

The system then opens or focuses MS Paint, interprets the command, and uses mouse-based control to draw objects dynamically on the canvas.

**2. Approach**

**2.1 Architecture Overview**

The project consists of three main layers:

1. **Frontend Layer (User Interface)**
   * Built with HTML, CSS, and JavaScript.
   * Provides a visually appealing and interactive web interface where users can type commands.
   * Displays real-time status logs and progress indicators.
2. **Backend Layer (Flask Application)**
   * Handles command requests from the frontend.
   * Passes user inputs to the drawing engine (draw\_bot.py) for execution.
   * Returns logs or error messages to the frontend for display.
3. **Automation & Drawing Engine (AI Agent)**
   * Implemented in draw\_bot.py using PyAutoGUI, PyGetWindow, and subprocess modules.
   * Responsible for controlling MS Paint—opening it if closed, focusing the window, and drawing shapes using cursor movements and drag operations.
   * Interprets the semantics of text prompts and executes corresponding drawing routines.

**3. System Workflow**

1. The user types a natural language prompt on the web interface.
2. The Flask backend receives the text and calls process\_command() from draw\_bot.py.
3. The function interprets the text using keyword-based logic (e.g., “house”, “tree”, “sun”).
4. The MS Paint window is opened or brought to focus automatically.
5. The respective drawing function (e.g., draw\_modern\_house, draw\_tree\_near\_house, draw\_sun) is executed.
6. PyAutoGUI simulates real mouse actions to draw the object on the Paint canvas.
7. The backend sends a response message back to the frontend indicating success or errors.

**4. Technologies Used**

|  |  |
| --- | --- |
| Component | Technology |
| Frontend | HTML5, CSS3, JavaScript |
| Backend | Python Flask |
| Automation | PyAutoGUI, PyGetWindow |
| Process Control | subprocess module |
| Development Tools | Visual Studio Code, MS Paint |

**5. Dataset**

Unlike standard deep learning-based text-to-image systems, this project did not rely on an external dataset. Instead, it used hand-coded procedural drawing instructions that simulate an AI model’s response. Each drawing module corresponds to an object type, such as:

* draw\_modern\_house()
* draw\_tree\_near\_house()
* draw\_sun()
* draw\_person\_front()

**6. Implementation Details**

**6.1 Flask Backend (app.py)**

* Provides endpoints for the main page (/) and command execution (/command).
* Receives text inputs and forwards them to process\_command().

**6.2 Drawing Engine (draw\_bot.py)**

* Handles Paint launching and window focusing.
* Provides shape-drawing functions for different objects.
* Contains intelligent routines to maintain context — for example, drawing a tree *next to* an already drawn house.

**6.3 Frontend Interface (index.html)**

* Features a clean, gradient-based UI.
* Includes animated command buttons and real-time log feedback.
* Displays a responsive layout suitable for desktop or mobile.

**7. Results**

The system successfully automated MS Paint to generate structured drawings based on text prompts.

The following examples were executed successfully:

* “Draw a modern house” → Generates a structured, architectural-style house (shown in Fig 1).
* “Add a tree next to house” → Adds a professional tree on the right.
* “Add sun” and “Add person” → Incrementally enhances the same scene.
* “Draw scene” → Creates a full composite (house, tree, person, sun) (shown in Fig 2).

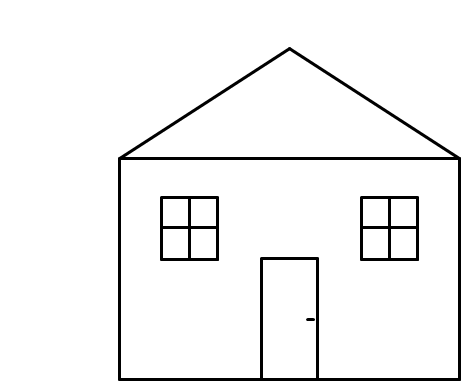


Figure 1: Image Drawn by AI for command Home

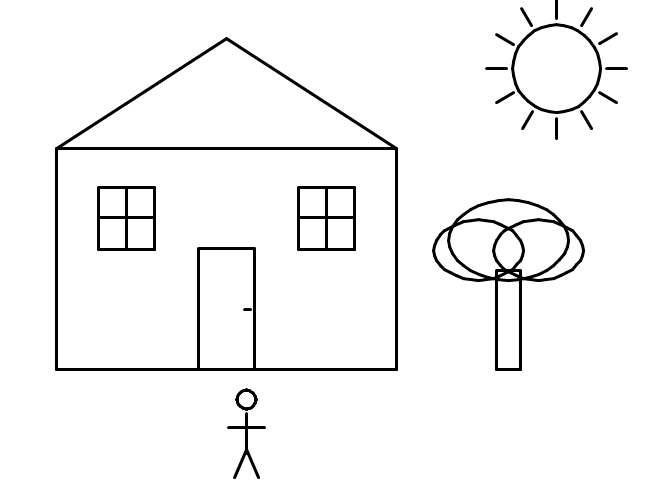


Fig 2: Image Drawn by AI when Draw Full Scene Command Receives

* The tree may not look like tree until it gets coloured. The coloured image is shown in Fig 3.

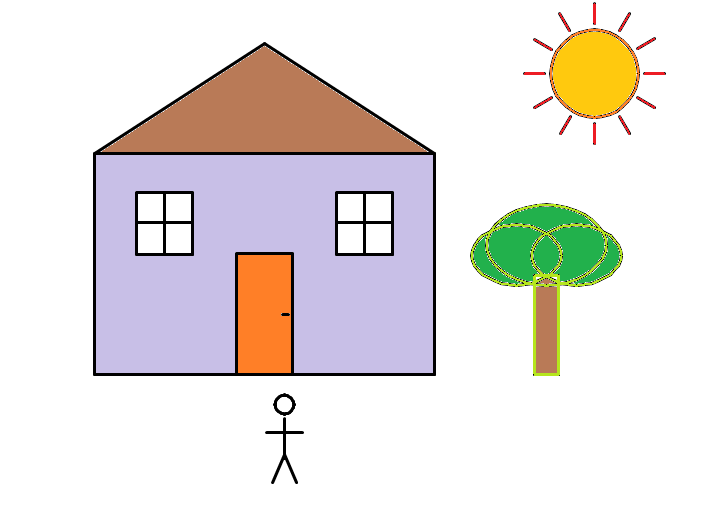


Fig 3: After Colors added

* Even though it is not added by AI, It is possible to do that.
* As, I already added the code part for Clicking the cursor at the right place, the only thing needed is taking the info’s of the location of Tabs.

**8. Challenges Faced**

|  |  |  |
| --- | --- | --- |
| Challenge | Description | Solution |
| Paint Window Detection | Different Windows versions use varied Paint titles (“Paint”, “Untitled - Paint”, etc.). | Implemented title-matching logic and multiple fallback launch methods. |
| Mouse Synchronization | PyAutoGUI requires precise timing between actions. | Added controlled sleep intervals and coordinate calibration. |
| Natural Language Understanding | No pretrained model used to interpret commands. | Used rule-based keyword mapping; future work can integrate LLMs like GPT or T5. |
| Screen Resolution Dependency | Drawing coordinates may vary across systems. | Calibrated relative positions and normalized window sizes. |
| MS Paint Focus Loss | Paint occasionally loses focus during automation. | Added click-focus and Alt+Tab recovery logic. |

**9. Future Enhancements**

* Integration of pretrained NLP models (e.g., T5, BLIP, or CLIP) to map free-form text into structured drawing instructions.
* Addition of voice-based command recognition using SpeechRecognition API.
* Implementation of undo/redo and intelligent correction features.
* Support for more complex shapes, textures, and perspective drawings.
* Deployment as a cross-platform web or desktop app with AI-generated sketches.

**10. Conclusion**

This project successfully demonstrates an innovative approach to bridging text understanding and visual creation through GUI automation. Even without deep learning models, the system achieves intelligent drawing behavior by mimicking how an AI agent would interact with a digital drawing tool.

The result showcases an interactive, real-time, and modular system that can serve as a foundation for future AI-driven art and design automation projects.