

PART 3: On AI Agent Integration

For the AI Agent integration I've chosen to utilise it to

1. Refine my word code
2. Improve documentation of my README.md file

Word code refinement

For this I had prompted copilot these exact instructions:

"Here is my word code for an automated pet feeder system. Please review it and suggest improvements or identify potential issues"

I uploaded this document along with this instruction.

Copilots' response:

Thanks for sharing your pseudocode for the automated pet feeder system! You've laid out a clear and logical structure. Here are some suggestions and potential improvements to enhance reliability, maintainability, and functionality:

✅ Strengths

- **Clear logic flow:** The use of conditional checks for feeding time and food availability is well-structured.
- **Sensor integration:** Incorporating both infrared and weight sensors adds robustness.
- **Logging and alerts:** Good use of logs and alerts for staff intervention.

🔧 Suggested Improvements

1. Time Comparison Precision

- **Issue:** `IsItFeedingTime IS TRUE` might be too vague.
- **Suggestion:** Use a time window (e.g., ± 1 minute) to account for clock drift or execution delays.
 - Example: `IF (CurrentTime WITHIN FeedingTime \pm 1 minute)`

2. Sensor Error Handling

- **Issue:** No fallback if sensors fail or give erroneous readings.
- **Suggestion:** Add checks for sensor status or error codes.
 - Example: `IF (InfraredSensorStatus IS ERROR) THEN ALERT STAFF (Sensor Fault)`

3. Redundant Conditions

- `ELSE (NOT (IsItFeedingTime IS TRUE))` is logically redundant after the previous IF and ELSE IF.

- **Suggestion:** Replace with a simple ELSE.

4. Scalability

- **Issue:** Only one feeding time is considered per loop.
- **Suggestion:** Use a list or array of scheduled feeding times and iterate through them.

5. Pet Identification

- **Suggestion:** If feeding multiple pets, include RFID or camera-based identification to personalize feeding.

6. User Interface

- **Suggestion:** Include a basic UI for staff to update feeding schedules, monitor logs, and override system manually.



Additional Features to Consider

- **Remote Monitoring:** Allow staff to receive alerts via mobile app or email.
- **Battery Backup:** Ensure system continues during power outages.
- **Cleaning Reminder:** Periodic alert for cleaning the bowl and dispenser.

Would you like help turning this pseudocode into actual code in a specific programming language like Python or C++?

My insights:

Interacting with copilot and its response has made me realise how even the hardware's can malfunction and have errors. I previously thought that hardware parts, such as the sensors, would always be functional. But now I have realised that even the system can face errors or have issues. This has made me realise that I need to think bigger than just environmental errors or output errors. And look more into the actual hardware, like for example, what if the feeder gets stuck or it needs cleaning? The cleaning part specially made me realise how hardware needs human intervention to work soundly too. I will be carrying this advice with me to my next assignments and projects making sure to highlight the assumed interventions more clearly. Overall, this has showed me I need to add those kinds of upkeep things in my assumptions as well broaden the scope of the possibilities so that my word code and flowcharts can become even more accurate.

Improve documentation of my README.md file

For this I had prompted copilot these exact instructions:

“here's my readme.md file for my automated pet feeder project. can you please review it and give me specific tips on how to improve it. tell me tips on the layout, structure and other things I could have added. as well as tips for future readme.mds”

Along with this I had copied and pasted my readme.md from Github

Copilots' response:

Your [README.md](#) is already quite thorough and well-written—great job! It clearly communicates the purpose, functionality, and context of your automated pet feeder system. That said, here are some **specific suggestions** to improve layout, structure, clarity, and future-proofing for this and future README files:

Structural & Layout Improvements

1. Add a Table of Contents

For long README files, a clickable table of contents improves navigation.

Table of Contents

- [Project Overview](#project-overview)
- [Key Features](#key-features)
- [Core Assumptions](#core-assumptions)
- [Inputs and Outputs](#inputs-and-outputs)
- [Assignment Context](#assignment-context)

2. Use Headings Consistently

Use consistent heading levels (##, ###) to create a visual hierarchy.

- ## Project Overview
- ## Key Features
 - ### Alerts
- ## Inputs and Outputs
 - ### Inputs
 - ### Outputs

This makes the document easier to scan.

3. Bullet Point Formatting

Use consistent bullet formatting. For example, under “Key Features,” each feature could be bolded for emphasis:

- ****Reliable Dry Kibble Dispensing****: Dispenses dry kibble...
 - ****Robust Monitoring Capabilities****: Incorporates advanced monitoring...
-

Content Suggestions

4. Add a Getting Started Section

Even if it's a conceptual project, a “Getting Started” section helps readers understand how to use or simulate the system.

Getting Started

To simulate or implement this system:

1. Review the flowchart and word code in the repository.
2. Set feeding times using the Real-Time Clock module.
3. Ensure sensors are connected and calibrated.
4. Monitor alerts via the display or sound signals.

5. Include a System Diagram

A simple block diagram or flowchart image would visually reinforce your architecture. You can reference it like:

![System Architecture](images/system_diagram.png)

6. Add a Technologies Used Section

Even if it's pseudocode, mention tools or platforms considered.

Technologies Used

- Infrared and weight sensors
- Real-Time Clock (RTC)
- Basic microcontroller (e.g., Arduino or Raspberry Pi)
- Sound and display modules

7. Add Future Improvements

Show that you're thinking ahead.

Future Improvements

- Add battery backup for power outages
- Integrate mobile notifications
- Support for wet food dispensing
- Pet identification via RFID

Tips for Future [README.md](#) Files

- **Start with a one-line project summary** at the top.
- **Use badges** (e.g., build status, license, version) if applicable.
- **Include a License section** if you're sharing publicly.
- **Add a Contributing section** if you want collaborators.

- **Keep it concise**—move detailed documentation to separate files if needed.
-

Would you like help creating a visual system diagram or converting this into a more interactive format like a GitHub project page?

My insights:

This response has helped me get a clearer understanding of how readme.md aswell as GitHub functions. Having little to no knowledge on this platform as well as it being the first time of utilising a readme.md, I found it hard to structure and present my layout in a way that was cohesive to how I had constructed the words doc, I specifically struggled in understanding how to represent different types of text, such as titles, headings, sub headings, and highlighted important information. Due to GitHub having less variety in word and letter customisation the readme.md is harder to read as all types of text look the same. This helps as it enables me to understand how one can work around this issue and still be able to present it well. This includes utilising hash (#) to represent hierarchy, and Asterisk (*) in bullet points for highlighted text.