Assignment 3: FlightGear Enhancement Report

April 12, 2024

Group 1

Matthew Susko

Marcus Dipalma

Mark Kennedy

Darcy Mckinnon

Armaan Wander

1.1 Abstract

This report examines our proposed concept to add a voice control feature to the simulated aircrafts of FlightGear. After a short introduction to what the voice control would do, this report will focus on two different modes of implementation that the voice control could potentially use. Next is the SAAM architectural analysis which contains a transparent and clear identification of key stakeholders that are affected by this enhancement, and an analysis of the Non-Functional Requirements (NFRs) that are integral to the stakeholder groups. This is all used to identify which of the two aforementioned modes of implementation will be selected. We then look at the effects of the enhancement, what some select use cases could look like, plans for testing, potential risks and lessons learned.

1.2 Introduction

The integration of a voice control feature into FlightGear would present a new interesting method for players to interact with the simulated environment of FlightGear. This report explores two possible implementations for incorporating voice control functionality into the existing FlightGear architecture. The first option looks at embedding the voice control system into the existing modules, more specifically putting it into the Input Manager Module. This is a nice option as a new module does not have to be created however it does increase complexity. The second option proposes the idea of a Voice Control module. This would be a new separate module which is separated for efficiency, only activating when prompted, however it is more complex to implement.

Continuing is the SAAM analysis of the module aiming to identify all key stakeholders and their NFRs to ultimately decide which of the two options should be chosen. For end users, this implementation of this new feature should feel seamless. A key component of making the implementation of voice control seamless is making it easy to maintain and implement for developers. Proper documentation of such a feature will also be needed as well perhaps for community contributors. As well as easy integration for any flight training school that use flight gear or any other software integrators. This Section having identified all main stakeholders, being; End Users, Developers, Flight Training Schools, FlightGear Community Contributors and software integrators along with all their NFRs, further dives into which of the options should be selected. This section concludes with option 1 being selected.

It’s then examined how the effects of this specific implementation will or could be. Taking a look at the effects on current architecture, maintainability, evolvability, testability and performance. Followed by a much closer look at two specific use cases being; A pilot initiating auto landing sequence and a pilot changing the throttle of an aircraft, both using a voice command. Finishing up the article are some proposed areas that will need testing, and potential risks that could occur with the implementation of the new system, and of course finishing with the conclusion and lessons learned.

2. Possible Implementations of the Voice Control Feature

There are multiple ways that the voice control feature can be implemented within the existing FlightGear architecture. This section outlines two possible implementations.

Option 1: Voice Control is implemented in existing modules

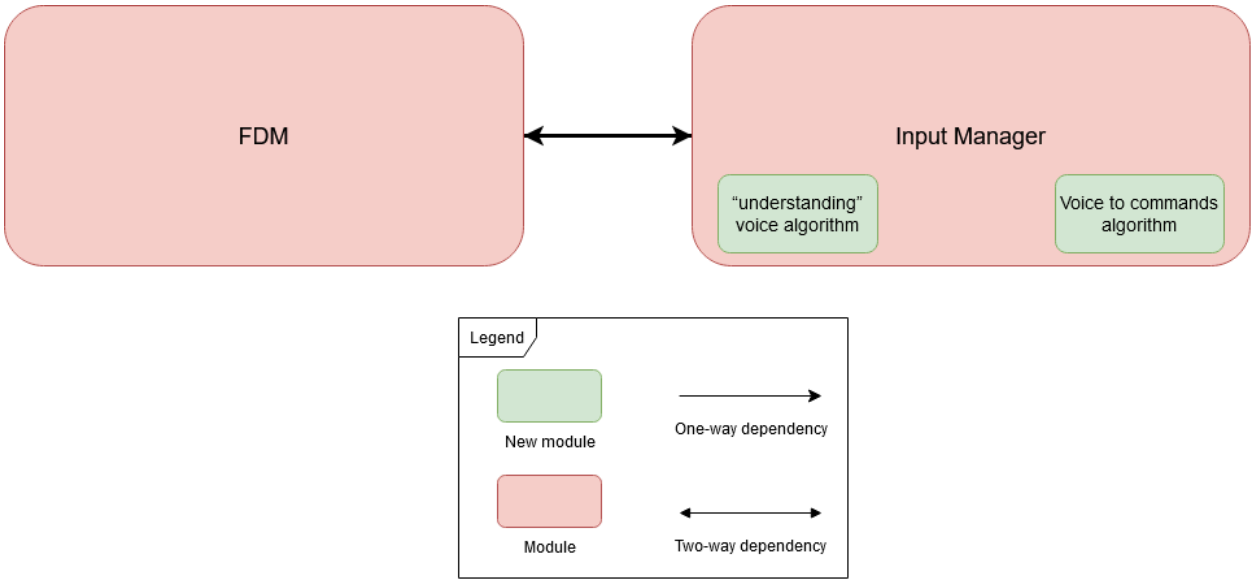

Figure 1. The implementation architecture of the enhancement

If we were to choose to implement the voice control system into the existing modules without a new module, it could be added to the Input Manager module. In this scenario, there could be two algorithms, one for "understanding" the voice commands and converting them into text or some other computer readable format and the other algorithm for converting the text into standard aircraft commands, identical to the ones from the standard keyboard and mouse/controller input. The Input Manager could then handle the inputs just like keyboard and mouse/controller inputs.

The advantage of not creating a new module is that integrating new systems into an existing module avoids the expensive and time-consuming process of implementation and testing. However, implementing the new systems into the Input Manager module could increase its complexity, making it more time-consuming to potentially sort through and understand in the future.

Option 2: Adding a dedicated Voice Control module

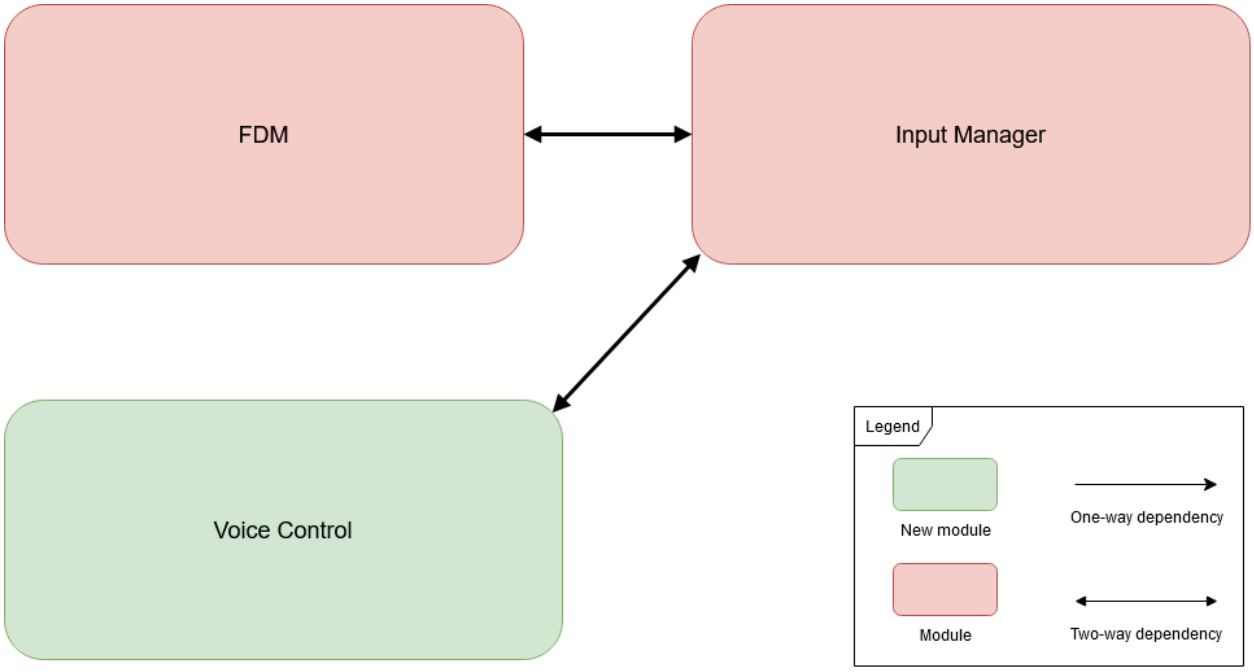

Figure 2. The alternative implementation architecture of the enhancement

If we were to choose to implement the voice control system as its own module, we would implement the same algorithms as discussed in the other option, mainly "understanding" the voice commands and converting the voice commands into standard aircraft commands. The new Voice Control module would have a two-way dependency with the Input Manager, as it would take the raw vocal inputs from the Input Manager and return control commands to be processed by the Input Manager.

The advantage of creating a new module is that the new module would only be used/loaded when needed, thus freeing processing resources for other modules to use. However, implementing the new module can be costly and time-consuming and could increase the overall complexity of the entire system.

3. SAAM Analysis

As we embark on introducing a new voice control feature to FlightGear, it's pivotal to assess its implications on the simulator's ecosystem. This begins with a clear identification of the key stakeholders affected by this enhancement, followed by an analysis of the Non-Functional Requirements (NFRs) that are crucial for each stakeholder group. This systematic approach ensures we address the needs and concerns integral to the feature's success. The primary stakeholders for the voice control feature, alongside their corresponding NFRs, are outlined as follows:

End Users

End Users, comprising both casual aviation enthusiasts and professional pilots, are at the heart of this enhancement. These individuals seek an elevation in realism and immersion through the simulation experience, hoping for a voice control system that simplifies complex controls and is accessible to users with varying abilities. Their primary expectation is a seamless, intuitive interface that responds accurately to voice commands, enhancing the flight simulation without detracting from the overall experience.

Non-Functional Requirements:

Usability - The voice control system must be intuitive and easy to use, with minimal learning curve and cognitive effort

Developers

The FlightGear Development Team, consisting of the software engineers, designers, and testers behind the simulator, plays a crucial role in integrating this new feature. Tasked with embedding voice control into FlightGear's diverse and complex system, their focus is on maintaining high standards of code quality, ensuring cross-platform compatibility, and facilitating future maintenance and scalability. The team aims to achieve a balance between innovation and the practical aspects of implementation, striving for a solution that aligns with the project's long-standing values and architectural principles.

Non-Functional Requirements:

Maintainability - The voice control code should be easy to maintain and update at any point in the future

Implementability - The voice control feature should be possible to implement with few major changes to the overall architecture of the system

Flight Training Schools

Flight training schools, which might use FlightGear as an educational tool, are keenly interested in the reliability and realism offered by voice control. They view this technology as a means to simulate real-world flying conditions more closely, providing students with a valuable learning experience. The expectation here is for a voice control system that can be reliably integrated into their training programs, enhancing the educational value of the simulation.

Non-Functional Requirements:

Integration - Easy integration with existing training protocols and systems.

FlightGear Community Contributors

A vibrant community of volunteers who contribute to FlightGear by developing aircraft, scenery, or other add-ons. Their work enriches the FlightGear ecosystem, and they are crucial in testing and providing feedback on new features.

Non-Functional Requirements:

Documentation - Provide clear documentation on how to make community-contributed content compatible with voice control.

Software Integrators

Entities that integrate FlightGear into larger simulation systems, or use it in combination with other software tools, for various purposes such as research, training, or entertainment.

Non-Functional Requirements:

API Availability - Provide APIs for integrating the voice control system with other software or systems.

Now that we have clearly identified our main stakeholders and their NFRs, we can now analyze how our two implementations -- the existing embedded implementation and new module implementation -- will affect their interests. First, for the end users, comprising both casual aviation enthusiasts and professional pilots, the best approach to implementing the new voice control feature in FlightGear is one that prioritizes usability above all else. Given their primary NFR of usability, ensuring an intuitive, easy-to-use system with minimal learning curve is paramount. An embedded implementation, which seamlessly integrates voice control within the existing simulator interface, could offer a more intuitive user experience by leveraging familiar controls and systems. This approach minimizes cognitive effort and facilitates a seamless introduction of voice commands into the simulation, enhancing realism and immersion without detracting from the overall experience.

The development team, tasked with embedding voice control into FlightGear, must balance high standards of code quality with the practical aspects of implementation. Given their NFRs of maintainability and implementability, the best approach is likely an embedded implementation. This method leverages the existing architecture, ensuring that the voice control feature can be implemented with minimal architectural changes and is easier to maintain and update in the future. This approach aligns with the team's aim to balance innovation with the simulator's long-standing values and architectural principles.

For flight training schools that use FlightGear as an educational tool, reliability and realism are key. Their primary NFR of integration suggests that the new feature should easily fit into existing training protocols without extensive modifications. An embedded implementation, which would integrate voice control more closely with the simulator's existing functions, might offer a smoother transition and better reliability. This approach ensures the voice control system can be reliably integrated into their training programs, enhancing the educational value of the simulation with minimal disruption.

Community contributors, who enrich FlightGear with their developments, have a primary NFR of documentation. To best support this group, whichever implementation is chosen must come with clear, comprehensive documentation on making community-contributed content compatible with voice control. An embedded implementation might require less extensive documentation changes, as it would utilize existing systems and interfaces, making it easier for contributors to understand and adapt their content to work with voice control.

Software integrators, who combine FlightGear with other systems, prioritize API availability. An embedded implementation could potentially offer more straightforward API integration, as it would use existing interfaces and modules within FlightGear. This approach ensures easier access and integration for third-party software, maintaining the flexibility and extensibility that integrators require.

NFRs and Implementation Effects

Stakeholder	**NFR**	**Embedded Implementation Effect**	**New Module Implementation Effect**
End Users	Usability	**Positive:** Enhances usability by integrating voice control seamlessly into the existing interface	**Negative**: Might introduce a learning curve and cognitive effort due to a new, separate system.
Developers	Maintainability	**Positive**: Easier to maintain due to minimal changes in architecture.	**Negative**: More challenging to maintain due to added complexity of a new module.
Developers	Implementability	**Positive**: Aligns with current architecture, simplifying implementation.	**Negative**: Requires significant architectural changes, complicating implementation.
Flight Training Schools	Integration	**Positive**: Simplifies integration into existing protocols and systems.	**Negative**: Potentially complicates integration with additional systems and protocols needed.

FlightGear Community Contributors	Documentation	**Positive**: Less extensive updates required for existing documentation.	**Negative**: Requires comprehensive new documentation to accommodate a separate system.
Software Integrators	API Availability	**Positive**: Maintains or improves API integration with existing systems	**Negative**: May require new APIs or modifications, complicating integration with other systems.

4. Effects of the Enhancement

Having decided on a specific implementation of the Voice Control feature, we will now summarize the possible effects that this implementation will have on the rest of the system. Due to the current Client-Server architecture of FlightGear, it should be possible to add the Voice Control algorithms into the Input Manager module without changing any other the other modules.

Effect on the current architecture

- **Input Manager**: The Voice Control algorithms would be directly implemented into the Input Manager module as a new input option. The new algorithms would handle the analysis of the raw voice input and then the conversion of that input into standard control inputs. Once the voice is converted to standard control inputs, they would be handled like any other control input.
    - Potential new file: flightgear\src\Input\VoiceInput.cxx
    - Potential new file: flightgear\src\Input\VoiceConversion.cxx
    - Potential changes to file: flightgear\src\Input\input.cxx
- **FDM**: The FDM module is the only module with dependencies on the Input Manger module, but there would be no need to change anything within the FDM module. This is because the Input Manager would not need to be sending or receiving any new information formats, so as far as the FDM module is concerned the Input Manager module has not changed.

Effects on maintainability

Maintenance of the system is made slightly more difficult by the addition of voice control, as there will be new files within the Input Manager module which may have dependencies on other pre-existing components, like taking input from the computer's microphone. However, the voice

control feature is not expected to drastically affect the maintainability of the architecture, as it is a relatively small feature with minimal complexity.

Effects on evolvability

The evolvability of the whole system would not be affected, however the evolvability of the Input Manager module could be effected for any changed to pre-existing components or overhauls to the way inputs/controls are processed by the FDM could clash with the controls being output by the voice control algorithms. The Voice Control algorithms, specifically the one that converts voice to controls, would need to be changed to account for the potential change in control format.

Effects on testability

Testing the new system would not be very complicated due to the nature of the new feature. The new feature will only need a few tests to make sure it is working properly, tests such as, making sure the voice input is being taken in properly by the game, making sure the right words are being understood from the raw voice input and finally make sure the voice commands are being accurately converted into standard control commands.

Effects on performance

There would be minimal effect on performance as the systems are not computation or graphics heavy. There could potentially be an issue with performance in the analysis of the raw voice input as this could potentially be a costly activity, however since the FDM module does not have to “wait” for the Input Manager to return an input command like it would for graphical processing, if there were to be an issue processing some vocal input, the FDM would continue with the simulation like normal without waiting for a control input.

5. Use Cases

Use case scenario 1: Pilot initiates auto-landing sequence using a voice command

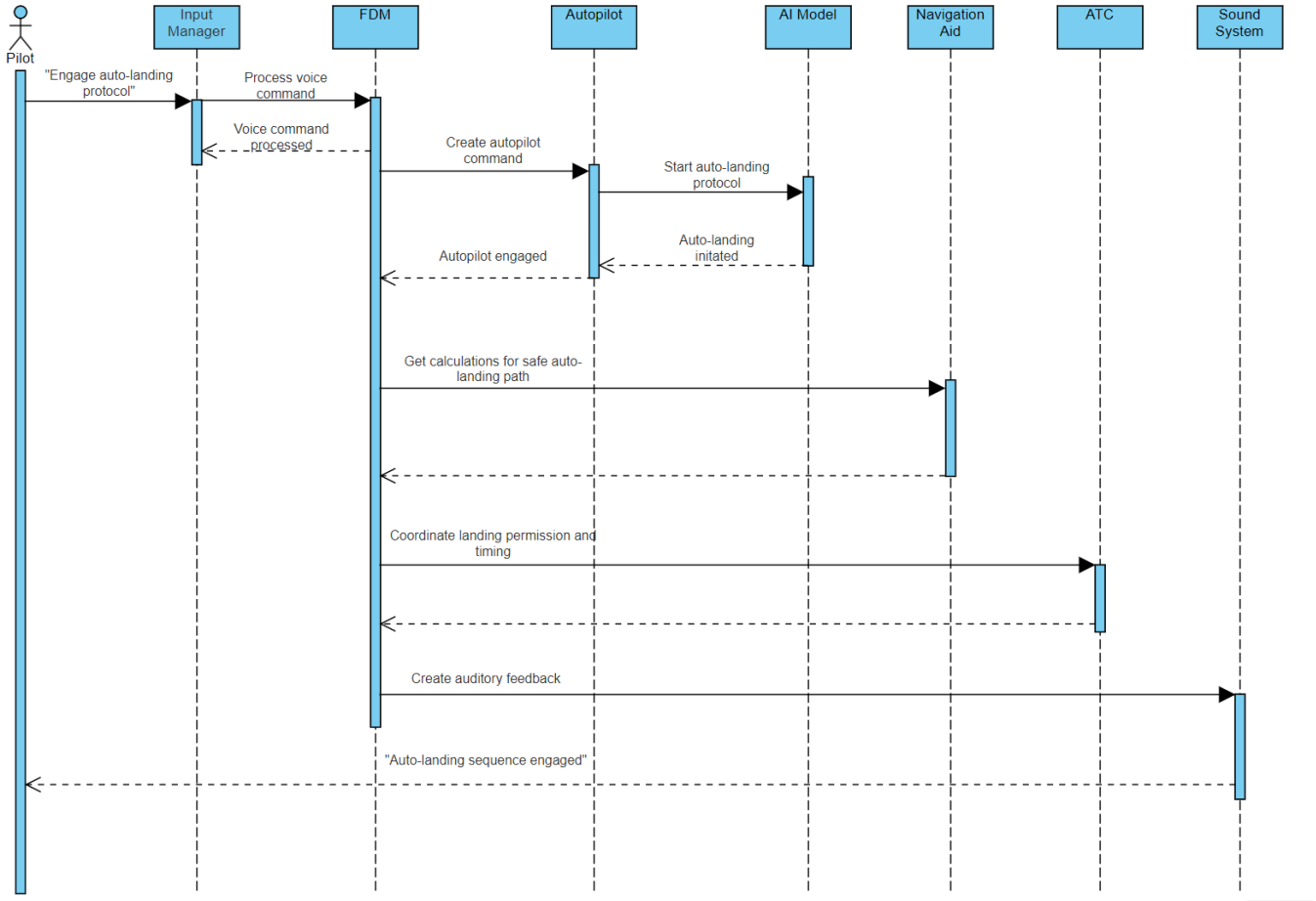

Figure 3. Use case for vocally engaging auto-landing

This sequence diagram illustrates the systematic process activated when a pilot employs a voice command to initiate an auto-landing protocol in the FlightGear simulator. The sequence is initiated when the pilot vocalizes the command "Engage auto-landing protocol," which is promptly received by the Input Manager. This component is responsible for interpreting the command and orchestrating the subsequent series of actions. Upon processing the command, the Input Manager conveys the pilot's request to the Flight Dynamics Model (FDM), which then collaborates with the Autopilot system to engage the landing sequence.

The Autopilot, now armed with directives from the FDM, commences the auto-landing protocol. It calculates the optimal landing trajectory incorporating input from the AI Model for decision-making and the Navigation Aid for navigational data. While the Autopilot is engaged, the ATC module undertakes the task of synchronizing the landing with existing air traffic patterns and provides necessary clearances. Lastly, the Sound System generates an audio cue. This cue serves to confirm the activation of the auto-landing protocol, providing the pilot with real-time auditory feedback.

Use case scenario 2: Pilot changes throttle using voice commands

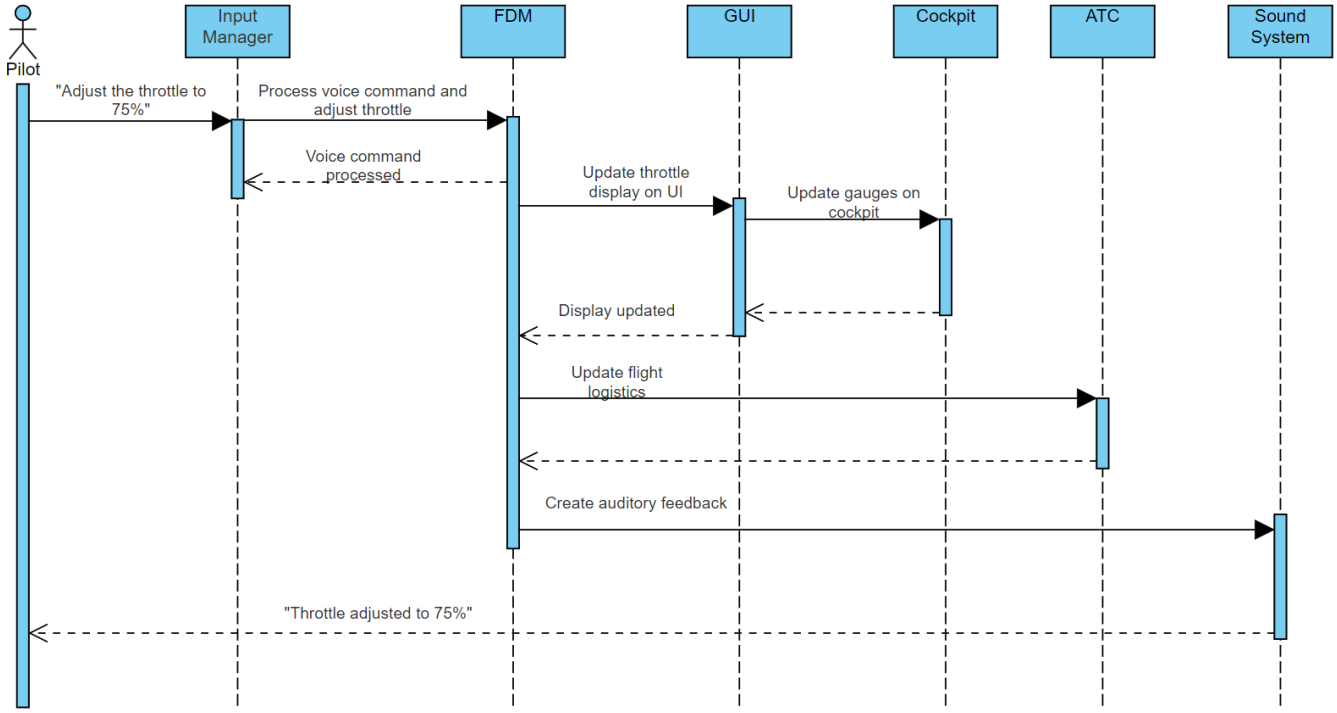

Figure 4. Use case for vocally changing the throttle

This sequence diagram illustrates the operational flow when a pilot inputs a voice command to adjust the throttle in the FlightGear simulator. The sequence is initiated by the pilot's command, "Adjust the throttle to 75%," which is captured by the Input Manager. Tasked with interpreting and implementing the pilot's instructions, the Input Manager processes the voice command and relays the requisite adjustment to the Flight Dynamics Model (FDM). The FDM, responsible for simulating the aircraft's physical response, adjusts the throttle accordingly, resulting in a change to the aircraft's thrust and consequently, its speed.

Following the FDM's response, the Graphical User Interface (GUI) is updated to reflect the new throttle setting, providing the pilot with visual feedback of the throttle gauge's change. The updated display includes the throttle position and potentially other flight logistics affected by the change in power, such as airspeed or climb rate. In parallel, the Cockpit gauges are updated to mirror this change, ensuring that all representations of the aircraft's status are consistent. Additionally, the change in throttle setting prompts an update in the simulator's Air Traffic Control (ATC) module. While not typically necessary for a simple throttle adjustment, this inclusion simulates the requirement to inform ATC of significant operational changes, particularly when they might affect the flight plan or airspace usage. Lastly, the Sound System

creates auditory feedback for the pilot. This auditory cue serves as a confirmation of the command's execution.

6.1 Plans for Testing

We will propose tests that developers can execute to ensure proper implementation of voice controls. Tests can be run to evaluate responsiveness under many use conditions to address the challenge of increased computational complexity and resource demands. Tests will assess resource utilization such as CPU usage, memory usage, frame rate, latency, etc., producing valuable statistics. These stats will give valuable insight on FlightGear's performance across various hardware, making sure that FlightGear can maintain accessibility.

Accuracy testing will also be necessary. These tests will be more manual as the testers will be required to document their intended command, and the resulting command given to the system. These tests will facilitate FlightGear's consistent delivery of performance and responsiveness, enhancing user satisfaction.

6.2 Potential Risks

Adding any features to software can lead to beneficial or detrimental changes of the non-functional requirements. Flightgear's architecture makes it easy for new features to be integrated without drastic changes being done to the existing codebase but there are potential risks when implementing voice control as it interacts with many components.

Impacted Components	**Reasoning**
Air Traffic Control/Airports	Players should be able to communicate with ATC through voice communications.
Navigation Aid	Players should be able to control navigation using voice commands.
Input Manager	Players will need proper sound input devices
Aircraft/Cockpit	Certain cockpit instruments will allow for voice control.
GUI	New GUI will be implemented for voice control
FDM	FDM will be managed by voice control

The current structure of FlightGear already supports third-party API integration so using a well-supported speech recognition API will be relatively streamlined.

Usability comes to mind when adding voice controls as it is important to have the voice recognition be accurate. Users should be able to use voice control without significant latency or resource consumption. If voice controls are inaccurate then there are in-game consequences that may not be the user's fault. The increased computational demands of voice recognition may lead to a performance degradation on less powerful setups, potentially restricting some users who would like to use voice control. Increasing the computational complexity poses a risk of a digital divide and potential financial burdens on users, undermining FlightGear's accessibility appeal. Along with complexity would be hardware changes, which would lead to the same consequences.

Any change to code will cause an increase in difficulty to maintainability and documentation in the long term. Introducing additional code complexity will make the codebase harder to maintain. However, implementing voice control modularity can enhance maintainability by isolating changes to voice control from the rest of the code. Documentation of voice control code will be necessary, resulting in additional documentation.

7. Conclusion and Lessons Learned

Integrating a voice control feature into FlightGear has broadened our understanding of the software's ecosystem and the interactions between new features and existing systems. This process has emphasized the importance of considering both the technical challenges and the user experience in software development. Our journey revealed the importance of stakeholder-centric design, where the identification of key stakeholder needs guided the development process, ensuring that the voice control feature addressed specific requirements and integrated smoothly with the existing system. We learned about the architectural adaptability necessary to accommodate new technologies without disrupting system integrity or performance.

The balance between innovation and practicality was crucial. The development team faced the challenge of integrating an innovative feature while maintaining the practical aspects of software performance and maintenance. This highlighted the need for careful planning and thorough evaluation to ensure the new feature was both implementable and maintainable without causing excessive disruption. Documentation and testing emerged as critical components throughout the integration process. Proper documentation ensures all contributors can effectively interact with the new feature, while comprehensive testing confirms that the feature performs as intended across various scenarios and does not negatively impact the overall system. Performance considerations were also paramount, as addressing potential impacts on system efficiency was crucial in preventing the alienation of users with less powerful hardware. From this experience, it became clear that user experience is paramount; ensuring that the voice control feature was intuitive and responsive directly correlated with user satisfaction and usability. The flexibility offered by considering both an embedded and a modular approach allowed for a more informed decision-making process. Proactively managing potential risks, such as performance degradation

and increased system complexity, was essential in mitigating negative impacts on the user experience and system stability. Engaging with the community early and continuously in the development process helped in refining the feature according to real-world feedback and expectations. Finally, the integration process served as a reminder that technology and user needs are constantly evolving, and staying adaptable is crucial for maintaining relevance and effectiveness in software solutions. This project has not only improved FlightGear's capabilities but also provided us with valuable insights into effective software development practices that will guide future enhancements, ensuring that FlightGear remains a leading flight simulation technology.

References

1. https://wiki.flightgear.org/