Emilie Knight

Designing a User Interface for Drone Control and Management

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

14 November 2022

Abstract

Author:	Emilie Knight
Title:	Designing a User Interface for Drone Control and Management
Number of Pages:	37 pages
Date:	14.11.2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Game Applications
Supervisors:	MSC Ivana Kovacevic, 5G Applications Engineer Miikka Mäki-Uuro, Senior Lecturer

The main objective of this thesis was to offer the case company suggestions on how to re-design the UI of an application. This application is called the Ground Control Station 1 (GCS 1) and its purpose is to manage and operate a fleet of drones. It was not possible to merely improve on the already existing GCS 1, so the decision was made to re-program the whole application using Unity. The new application is called the GCS 2. Therefore, the secondary objective of this study was to propose solutions on how to implement the new design in Unity.

To achieve these goals, a survey was conducted to provide feedback on the UI of the GCS 1. After processing the feedback results, research was done on UI/UX design. The most influential source in this study was Steve Krug's book "Don't Make Me Think", as it is often considered a must-read for any aspiring UI designer. Additionally, a mock-up design for the UI of the GCS 2 was created based on the research and the feedback from the survey.

As a result of the survey, the most important areas for improvement in the GCS 1 UI were identified. The respondents requested a more modular and readable UI design. However, the general layout was perceived as being logical. As a result, the UI mock-up of the GCS 2 kept the overall layout but added UI elements that would make the content easier to read. Other major changes included added contrast to the color scheme and increased UI scale.

The outcome of the thesis should provide a good starting point to the case company to implement a better design for the GCS 2. It is recommended that further user testing of the GCS 1 UI be conducted. Additionally, to lessen the sheer amount of information on the GCS 1 UI, a decision should be made on which information should be prioritized.

Keywords: Unity, UI, UX, design, drone, Ground Control Station

Tiivistelmä

Tekijä:	Emilie Knight
Otsikko:	Droonin ohjaus- ja hallintaohjelman käyttöliittymä
Sivumäärä:	37 sivua
Aika:	14.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaajat:	5G Applications Engineer MSC Ivana Kovacevic Lehtori Miikka Mäki-Uuro

Insinöörityön tarkoitus oli antaa suosituksia työn tilanneen yrityksen ohjelman käyttöliittymän suunnitteluun. Ohjelman nimi tulee englanninkielisestä nimestä Ground Control Station 1 (GCS 1), ja ohjelman tarkoituksena on hallita ja ohjata drooneja. Koska yrityksen ei ollut mahdollista vain parantaa GCS 1:n käyttöliittymää, ohjelma päätettiin toteuttaa kokonaan uudelleen käyttäen Unity-pelimoottoria. Uuden ohjelman nimeksi tulisi GCS 2. Tämän työn toinen tavoite oli antaa ehdotuksia siitä, miten uusiksi suunniteltu käyttöliittymä tulisi toteuttaa Unityllä.

Näiden tavoitteiden saavuttamiseksi toteutettiin kysely, jotta saataisiin GCS 1:n käyttöliittymästä palautetta. Kun kyselyn vastaukset oli prosessoitu, haettiin tietoa käyttöliittymän suunnittelusta. Lähde, jolla oli tähän työhön eniten vaikutusta, oli Steve Krugin teos “Don’t Make Me Think”, sillä tätä teosta suositellaan kaikille käyttöliittymäsuunnittelijoille. Lisäksi uuden tiedon ja kyselyn tulosten perusteella luotiin malli GCS 2:n käyttöliittymästä.

Kyselyn avulla tunnistettiin GCS 1:n käyttöliittymän osat, jotka kaipasivat eniten muutosta. Osallistujat toivoivat modulaarisempaa käyttöliittymää, mutta yleinen pohja nähtiin hyväksi. Tästä syystä GCS 2:n malli käyttää samaa pohjaa kuin edeltäjänsä, mutta siihen lisättiin elementtejä, jotka parantaisivat käyttöliittymän luettavuutta. Kontrastin lisääminen väreihin ja tekstin koon suurentaminen kuuluivat myös suurimpiin muutoksiin.

Insinöörityön tuloksena yrityksen pitäisi saada hyvä lähtökohta toteuttaa paremmin suunniteltu käyttöliittymä GCS 2:lle. On suositeltavaa, että GCS 1:stä kerättäisiin enemmän käyttäjäkokemuksia. Lisäksi, jotta tiedon määrää näytöllä saataisiin pienemmäksi, olisi hyvä päättää, mitä tietoa on priorisoitava.

Avainsanat: Unity, käyttöliittymä, suunnittelu, drooni, Ground Control Station

Contents

List of Abbreviations

API:	Application Programming Interface
GCS:	Ground Control Station
gRPC:	Google Remote Procedure Call
IMGUI:	Immediate Mode Graphical User Interface
NDN:	Nokia Drone Networks
UAV:	Unmanned Aerial Vehicle
uGUI:	Unity User Interface
UI:	User Interface

1 Introduction

This thesis explores re-creating an Unmanned Aerial Vehicle (UAV) control and management application for the case company. The application used to manage and fly drones is commonly called a Ground Control Station (GCS) (ArduPilot Dev Team, 2021). The case company's current GCS is facing some technical and design challenges and therefore, prior to this project, it was decided within the team that the original GCS application would undergo re-architecture and re-writing. For the purposes of this study, the old application will be called GCS 1, and the new application, developed in this work, GCS 2. The GCS 2 is the client application installed on the user's machine, for example a laptop, that allows the user to control and manage the drones. It is made with the Unity game engine and is designed to be modular to fulfil different customer demands. In addition to technical difficulties, the User Interface (UI) of the GCS 1 needs re-designing, as some elements of it are facing criticism from current customers and application testers.

The drones themselves allow for different payloads to be attached to them. They are marketed by the case company as being highly customizable. For example, the payload can be swapped with a thermal camera to detect oil leakages, or a sensor to measure air or water quality. This has led to a wide variety of customer demands. This thesis explores how to re-create a UI based on customer feedback and how to take modularity into account in the application's UI design and programming to meet those diverse demands.

The UI elements should be designed so that they can be added or removed from the layout without causing too much damage to the overall visual design. Similarly, the code should reflect that flexibility. Ideally, the project should not require much rewriting when pieces of code matching the UI elements are removed or added. Therefore, this thesis attempts to find ways to write dynamic, easy to maintain and extendable code and UI elements using C# and Unity. It will also determine the most urgent flaws in the GCS 1 with the help of

a survey and make suggestions on how to correct them from a design perspective, as well as how to implement those corrections in Unity.

This thesis has six sections. Section 2 provides an overview of the system architecture of the case company's product, as well as which tools were used in this project. A survey and some background research on UI design were conducted and are discussed in Section 3. In Section 4, suggestions are given on how to improve the on GCS 1 UI design and Unity implementation, and, finally, Section 5 provides a conclusion to this project.

2 Project Description

This section explains the system architecture of the case company's product to provide a better understanding of the product in its entirety to the reader. Additionally, information about which software was used in this project is provided.

2.1 System Architecture Overview

The case company's product is comprised of several hardware and software components that are all connected to the same dedicated 5G network. The drones are controlled using Windows, MacOS or Linux laptops. The drones and the docking stations are connected to the laptops through the Edge server that relays data between the two. The Edge also acts as storage for video streams from the drones. A visual representation of the software architecture is presented in the following diagram:

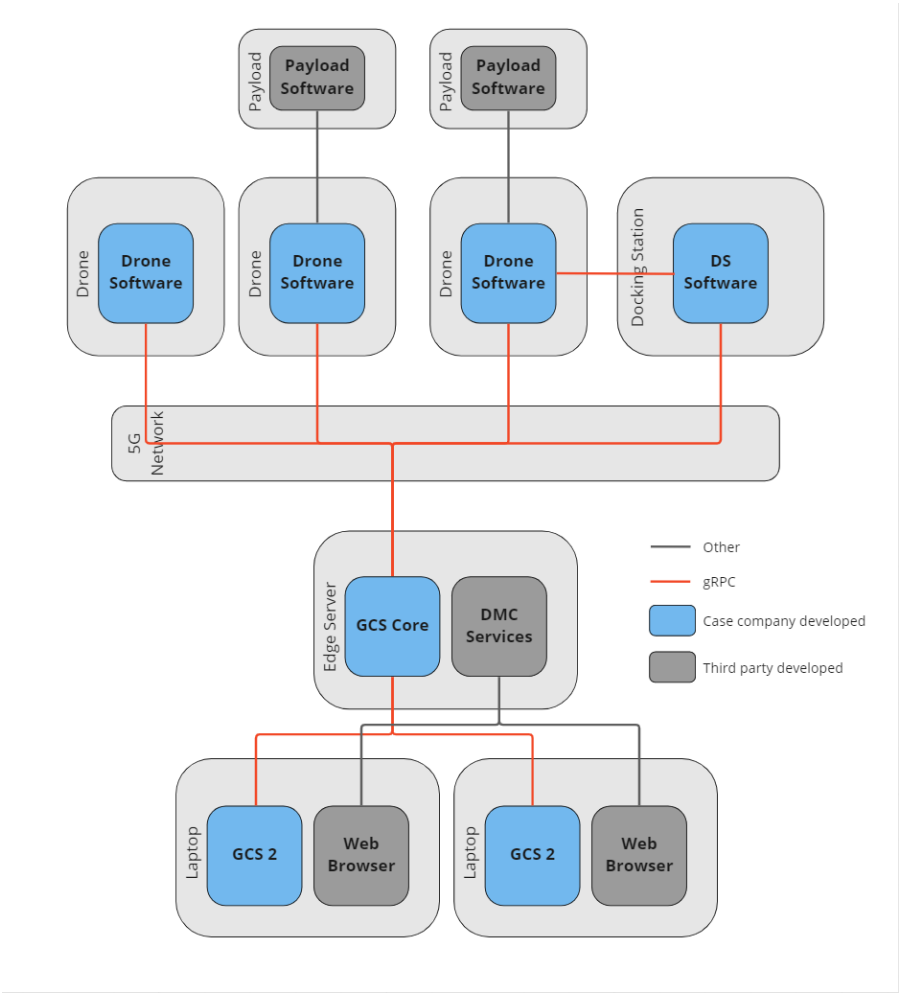

Figure 1. A visual representation of the software architecture of the drone and its related systems.

The software domain is divided into five parts (Figure 1) and each part runs on a different piece of hardware: the end user device, the server, the drone, the payloads, and the docking station. The end user software that is running on the user's device includes a web browser, from which the user can watch video streams provided by the drones, and the GCS 2, which is essentially just the UI, and is the focus of this thesis. Very little non-UI related logic is involved in the GCS 2. The server runs the GCS Core, whose main function is to communicate with the drones using google Remote Procedure Call (gRPC). The Digital Management Center (DMC) services are also situated on the server and are responsible for streaming video from the drones. The drone itself has several pieces of software running on it. These programs' basic function is to allow the

drone to fly safely and to communicate with the docking stations and the payloads. The docking station is where the drone recharges. Payloads are attached to the drones and can vary drastically depending on use case, and therefore the software required to run them is situated on the payloads themselves. A payload can be, for example, a camera or a sensor.

2.2 The Need for Software Re-architecture

There are multiple reasons why a re-architecture of the GCS 1 is needed. It was made using Windows Forms, and it only runs on the Windows operating system. Since Windows tends to force updates and restart the computer, it is not ideal for use cases where the application needs to be running around the clock every day. Linux is a more appropriate platform for this purpose. In addition to Linux, the GCS needs to run on Windows and macOS as well to provide as much flexibility as possible for the end user. Windows Forms is also not well suited for real-time visualization, which makes building a UI with it ineffective, inconvenient, and performance-hungry.

The GCS 1 needs to be split into two parts: the GCS Core, which handles communications to the drones and is situated on the server along with the DMC services, and the GCS 2, which is the UI part of the architecture. This split is necessary for two reasons. Firstly, the GCS Core enables communications with several GCS 2 clients at once in use cases where there are, for example, several drone pilots controlling the same drones. Secondly, the GCS Core includes support for third-party Application Programming Interfaces (APIs). This enables the customer to use their own applications for, for example, analytics, or if they want to develop their own UI. Consequently, the GCS Core uses .NET 6, and the GCS 2 uses Unity. The team working on this product is well versed in C#, and therefore existing knowledge can be utilized to write both applications.

2.3 Unity and .NET

.NET is a framework or tool used to enable developers to build applications using, for example, the C# programming language (Codecademy, 2022). Unity, on the other hand, is a development platform used primarily for making games, whose primary development language is C# as well (Unity Technologies, 2022). Unity is also used in various other industries for building 2D, 3D, VR and AR environments. Therefore, one of Unity's advantages is the number of development options it provides. It enables making builds for several platforms, including all target operating systems for this project (Unity Technologies, 2022). The current use of Unity in this case is limited to 2D, but thanks to its flexibility, Unity leaves many possibilities open for future development, for example if 3D assets are be used. Picking Unity for this project means the case company will be less limited in their future plans, than if they had continued using Windows Forms. Additionally, it provides better rendering performance than Windows Forms, and several third-party assets are available to purchase on the Unity Asset Store to speed up development (Unity Technologies, 2022).

2.4 Unity UI Framework Selection

Unity has three different frameworks for creating UIs: the IMGUI, the UI Toolkit, and the uGUI. The IMGUI is used for creating custom Unity Inspectors but is not recommended by Unity for building runtime UI. The uGUI system is the current method used by most Unity projects. With the uGUI system, the UI is built from GameObjects, and the developer uses the Scene and Game views to position and style the UI elements. (Unity Technologies, 2021b) According to Unity documentation, the uGUI system will be deprecated soon, and is to be replaced with the new UI Toolkit, which will become the standard way of building UIs for both runtime and custom Inspectors, combining the capabilities of the IMGUI and uGUI. The UI Toolkit is based on current web development practices such as stylesheets and UXML, a HTML and XML based markup language, which makes creating UI structures easier. It is also designed to improve performance over multiple platforms. (Unity Technologies, 2021c)

As the UI Toolkit has not yet been fully implemented, it was unclear at the start of this project whether it should be used to build the UI of the GCS 2. Therefore, a comparison between the UI Toolkit and the uGUI is provided in the following figure (Unity Technologies, 2021a).

Figure 2. A comparison between Unity's uGUI and UI Toolkit frameworks

As seen from Figure 2, both systems have positive and negative sides. However, some features weigh more than others when deciding if the switch from uGUI to UI Toolkit should be made and when. there are certain unsupported features in the uGUI that are handy to have and are not currently implemented in the UI Toolkit. The use of serializable UnityEvents probably has the largest impact in terms of available features. UnityEvents are similar to C# Actions, which allow the programmer to define an action that happens as a result of another action, such a user input. Since the UI in this project is rather static, and generally things do not happen unless there is user action, such action-reaction relationships in the code are common and convenient. In the uGUI system, such actions are serializable, which means that certain parameters can be set directly from the editor, rather than in the code. This is especially useful when considering code maintenance. (Pitaksarit, 2021)

There are several features in the UI Toolkit that would be beneficial to the project in future. The most anticipated features of the new system all relate to maintaining the code and reusing assets. For example, global style management of the UI elements, such as colors and fonts, would be a welcome addition (Unity Technologies, 2021c).

Based on the reasoning in the previous paragraphs, it was decided during the writing of this project that development would continue using uGUI, despite concerns that it will be deprecated eventually and no longer supported. The main reasoning behind that decision is that development had already started using the older system and the team’s knowledge of uGUI and C# in general is higher. However, it is probable that the switch to the UI Toolkit framework will have to be done in future. To prepare for this change, a decision was made within the team to increase knowledge about the UI Toolkit while still developing with the uGUI. Consequently, when the switch is made, the development of the UI need not be halted entirely during the learning process, and some of the experience needed to rebuild the UI will already be there.

To design the GCS 2, the first step was to determine what is wrong in the design of the GCS 1. To do that, a survey was conducted among users of the GCS 1. Based on that feedback, some general information was gathered on UI/UX design to acquire the necessary knowledge to fix some of the issues in the GCS 1 UI design. Lastly, suggestions were provided on how to improve on those issues and on implementations for those improvements in Unity.

3 Research and Theoretical Background

A survey was carried out as part of this project. In this section, the survey results are presented, as well as some background research on UI design and implementation.

3.1 Feedback Survey on the GCS 1 UI Design

To help recognize major UI design problems in the GCS 1, a survey was conducted among the current UI users. The purpose of the survey was to gather as many feedback items as possible to get an overall understanding of which features of the GCS 1 UI design could be improved. The survey was comprised of open-ended questions, where the respondents were asked to analyze different aspects of the application and freely give their opinion about how the UI behaves and feels (Appendix 1). It also included ratings to help evaluate how well the GCS 1 UI is implemented. The following diagram gives a general idea of the responses given.

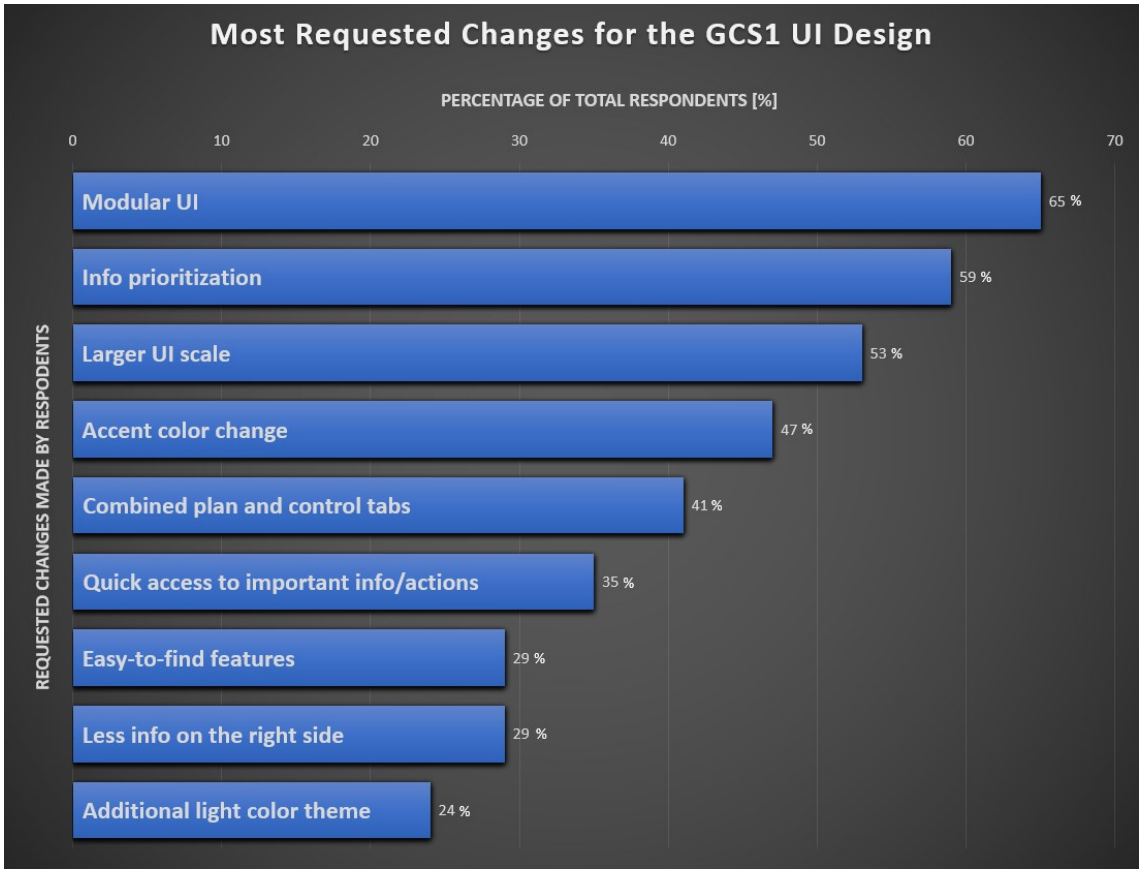

Figure 3. The top nine requested changes for the GCS 1 grouped in general categories

The group of respondents (N = 14) consisted of customers, partners, and project team members. The team consists of engineers from various disciplines, who work on the drones or on the drone systems. The group of respondents includes both first-time and experienced users. The first-time users provide important information on how intuitive the basic functions of the GCS 1 are. Experienced users, on the other hand, have the best knowledge of what to expect from an application with which to control a fleet of drones. They are aware of the more advanced features and have on-field experience as well as experience of repetitive use. The whole product provided by the case company is an industrial grade solution, and therefore, a certain level of specific knowledge is required for advanced users. However, the aim is also to provide an intuitive UI suitable for a simple mission execution for first-time users and

smoothing the learning curve for professional pilot training. Therefore, in the presented analysis, all responses are weighted equally.

To quantify the data from the open-ended questions, each answer was split into separate statements. Duplicates from the same users were not taken into account. Next, statements were grouped together under common categories as seen in Figure 3. Each statement is included in only one group. The “Modular UI” category includes all statements where the respondents wished for a more customizable UI layout by being able to move elements around. The “Information prioritization” category covers any statement requesting to emphasize important information and to reduce irrelevant information in different parts of the UI. The “Larger UI scale” category is self-explanatory: any user who wished for larger UI elements or fonts had their statements included here. Many respondents felt like the use of the accent color was confusing, as the GCS 1 also uses a highlight color. These statements belong in the “Accent color change” group.

The GCS 1 also has “Plan” and “Control” tabs situated at the top of the screen. Those tabs enable different UI elements, but many respondents felt like the layouts behind the tabs still look deceptively similar and wished for them to be unified. These statements were included in the “Combined plan and control tabs” group. Some respondents felt like some information and actions should be visible at all times. Such statements were included in the “Quick access to important information/actions” category. The “Easy-to-find features” category is a combination of statements describing a request to make features more distinguishable from each other, or more visible in general. Many respondents felt like they had to search the screen for the feature they were looking for. Somewhat related to this is the “Less information on the right side” group, which merits its own category due to the sheer number of user wishes to remove information from that specific part of the UI. Lastly, the GCS 1’s dominant color is dark grey. The “Additional light color theme” category encompasses the statements where the users wished for a color scheme that was lighter, as, if used outside, the GCS 1’s dark theme can be difficult to see properly. Figure 3

only represents the largest categories. Other categories are deemed to be out of the scope of this report. Some of them conflict with the overall product vision, while others simply included too few items. This project is most likely to make the largest impact by focusing on the top categories.

Categories with positive feedback were also made but were not included in Figure 3. The positive feedback is good to keep in mind, as those features can then be re-implemented in the GCS 2 similarly to the GCS 1. Most of the positive feedback statements described the icons as informative and praised the use of the yellow and red warning colors for critical information and actions that required immediate attention.

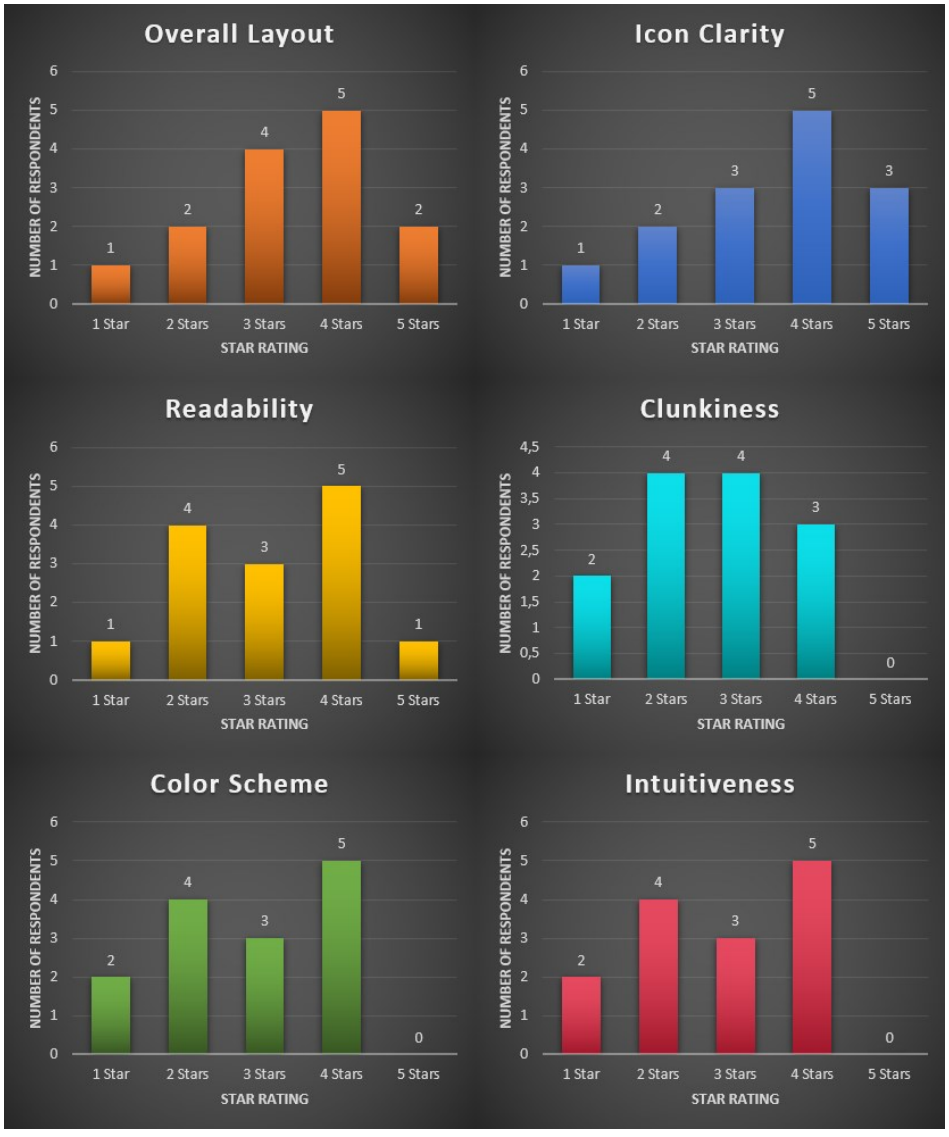

Figure 4. Ratings for the GCS 1. Each chart directly represents a survey question.

In addition to qualitative feedback, some quantitative data was gathered by asking the respondents to rate different aspects of the GCS 1 UI from one to five. Each chart in Figure 4 matches a question in the survey. The average ratings for all categories ranges between 2.62 and 3.50, and the overall rating is 2.98, which are all close to the average between one and five. There are no categories with particularly polarizing opinions, however the “Icon clarity” category gathered the most ratings above four and has the highest average. Nevertheless, all the categories are below the satisfactory rating and therefore all these features require redesigning in GSC 2.

3.2 UI Modularity

The most requested feature from the survey was adding modularity to the UI. In this study, the definition of UI modularity is as follows: it is a way of designing UI elements so that they can be independently rearranged, customized, and reused. In this work, a distinction is made between visual modularity of the UI, and modularity in code. Therefore, going forward, the term “modularity” is used in the visual design context, and “flexibility” is used when referring to code.

Visual modularity of the UI makes designing for different screen sizes easier by making the layout more responsive. The GCS 2 is supposed to primarily run on a laptop or desktop. However, should the decision be made to switch to a mobile platform in future, having a self-arranging modular UI will help in making the transition. Also, by designing the UI as separate modules, the boundaries between those elements look cleaner, thereby addressing the concerns of the users who felt like features were not separated clearly enough from each other. This, in turn, could improve readability and help guide the user\s eye to relevant parts of the UI. (UX studio, 2017) There could be, however, a few pitfalls related to making a modular design. The homogeneity of the design may suffer if modules end up looking too different from each other.

Due to being reusable, flexible code makes the application more sustainable. If one part of the UI needs to be re-designed or re-programmed, other parts will

continue working independently. Maintenance of such code can be done in smaller steps, and it is less likely that the entire layout will break if the elements are as little interconnected as possible. The development is also simplified, as large projects can be broken down into manageable parts, which makes collaboration easier. It follows, that developers new to the project can get easier acquainted with smaller modules one at a time. (Gwentech LLC, 2020) Writing flexible code can also be challenging, as not all modules can be exactly the same. Individual modules will always be a little different from each other, but the code should still take these edge cases into account. While it is tempting to think that the functionality of a class should be flexible enough that it can work in many different cases, the more edge cases are introduced to the system, the more complicated the logic becomes. This problem can, in certain cases, be solved by using abstract classes, for instance (Microsoft, 2022).

3.3 Designing for Scanning

In his book (Krug, 2014, pp. 17-24), Krug emphasizes that a website's UI should be clear and intuitive, because, on average, users spend a short time on the page and expect to find the information they seek in a matter of seconds. People tend to scan UIs, and then decide to click on the first element that looks reasonably adequate for their purpose. Unlike with websites, it is expected that once a customer has purchased the case company's product, they will continue using it for at least a few years. The goal of this product is to provide an application that is versatile and adequate for a long period of time. Thus, it is acceptable to think that users have to spend some time learning to use the GCS 2. The GCS 1 has a steep learning curve, however, which drains resources from the development team, as they must spend time teaching the customers how to use the program. The users want to be able to direct most of their attention towards flying the drones, and they should not have to search for relevant information on the screen. Critical information should be available at a glance.

In the GCS 2, the application's default view should display basic features and information to safely fly the drone in an intuitive way. While more advanced features are not visible in the default view, it should be obvious that they exist should the user need them. With this layout in mind, it is important to evaluate which information and features should be included in the basic layout, and which can go in the advanced view. To achieve that, Krug has a handy list of suggestions: stick to conventions, make the element's purpose crystal clear, break the application into obviously defined features, and remove distracting clutter (Krug, 2014, pp. 21-29).

UI elements that belong to the same feature should be grouped visually, for example by applying a common visual style or collecting them together in a distinct area. Some form of nesting can be used to create sub-groups. Prominence within the groups or between groups can be determined by adjusting the size, color, and placement of the element. It is natural for humans to hierarchize visual elements. The UI should reflect this natural tendency, as if everything looks the same or equally important, the time spent to spot what is relevant increases. (Krug, 2014, p. 34) Web UI/UX designers commonly use grids to organize and hierarchize the layout of webpages. They also help ensure that pages behave as intended in different resolutions. (Designlab, 2022) Figure 5 below shows a few UI designs that use grids to make everything look orderly and logical, and Figure 6 displays designs that do not follow a strict grid layout.

Figure 5. A set of designs that follow a clear grid layout. (Designlab, 2022)

Figure 6. Examples of designs where following a strict grid is less important. (Designlab, 2022)

As can be seen from Figures 5 and 6, placing elements according to a layout helps to keep it orderly and guide the eye according to an established hierarchy. Even in Figure 6, there are grids, although they are broken up. These examples were designed to draw attention to them because they look disorderly. However, even though the pictures may appear chaotic, the texts are still laid out according to a somewhat intact grid, which makes them still readable. In Figure 5, everything is arranged according to grids. The Beethoven concert advert in the top-right seems to make an exception to that by adding a circular UI element to give more visual weight to the text. In fact, this circular element is also a grid, albeit a radial one. (Designlab, 2022)

The second most requested change for the GCS 1 was for better prioritized information. There are several ways to emphasize elements in the UI. The concept of visual weight, where each element on a screen has a certain level of attraction to the eye, can be a helpful one when designing the GCS 2. The more weight an object has, the more it attracts attention from the viewer. One of the

challenges of UI/UX design is that visual weight cannot be measured, it can only be perceived. The core idea for visual weight is that anything that looks exceptional on the screen will stand out. To increase visual weight, an element can be made larger, a different shape, or it can be oriented in an exceptional way (vertical objects have more weight than horizontal ones, for example). Lastly, another way to control visual weight is by managing white spaces, which are the spaces between UI elements that don't contain anything. Elements with more empty space around them tend to stand out. Color is also a powerful tool for adjusting visual weight. (Babich, 2020) The following figure illustrates the use of accent colors.

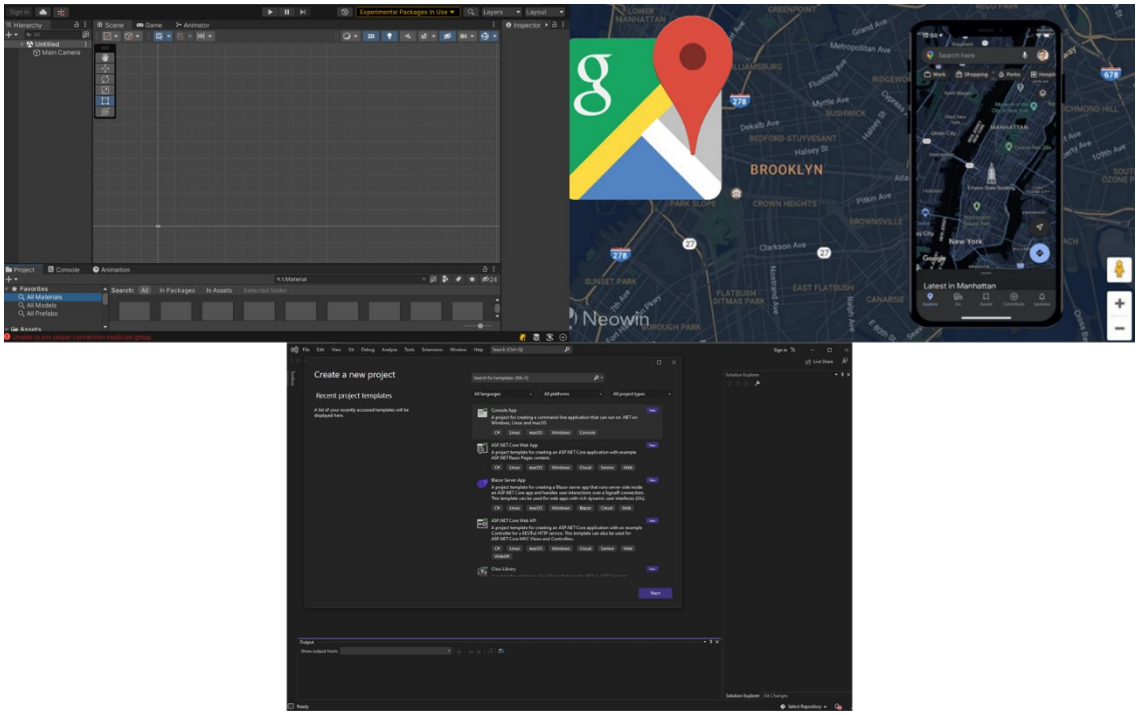

Figure 7. Dark themes of Unity Game Engine, Google Maps (Parker, 2021) , and Microsoft Visual Studio all use an accent color which has a low visual weight.

Colors in the red to yellow spectrum attract more attention than in the blue to green spectrum. And, of course, more saturation means more weight. (Babich, 2020) Often, color themes are picked so that there are background colors that are close to each other in hue, but differ in contrast, and some accent colors that stand out due to being different from the background colors. The GCS 1's

background color palette contains various greys, and the accent color is blue, which seems to be quite popular in dark themes, as can be seen for example in the Unity Editor and in Google maps in Figure 7. Microsoft Visual Studio's accent color is purple, which also has a low visual weight (Figure 7). Accent colors often match brand or product colors, as with Visual Studio. The case company brand color is blue, so its use in the GCS 1 makes sense. In Figure 7, it is apparent that accent colors have mostly been reserved for interactable elements like buttons, tabs, and toggles. Adding contrast and texture are other ways to make colors attract the attention of the user. When using a dark theme, lighter colors stand out. The GCS 1 UI does not employ any textures, but insufficient contrast between the different UI elements was mentioned by a few users in the survey. Those respondents felt like it made the UI hard to read, especially if that lack of contrast involved texts, or if the fonts were too thin, as that can make the letters look darker. Critical information, such as safety features or warnings in the message box are marked in red or yellow in the GCS 1. According to the survey, this was perceived well, and those elements draw attention, as they should.

3.4 UI Design Conventions

Fortunately, to help design the GCS 2, a lot of conventions are already in place in the UX/UI design world. Information is readily available especially when it comes to designing for the web, and while web applications and desktop applications have their differences, most of these conventions generally still apply to both.

One of the issues with the GCS 1, according to the survey, is the overwhelming amount of information on the screen. In addition to being more efficient at displaying that information in a logical manner, the GCS 2 should also prioritize which information is visible at what time. Additionally, it should consider the varied use cases different users might have. Hiding non-relevant information can be a powerful tool to reduce clutter as well as assist visual scanning for

meaningful information. There are several UI elements that can help solve this problem. (Godfrey, 2017) They are illustrated in the figure below.

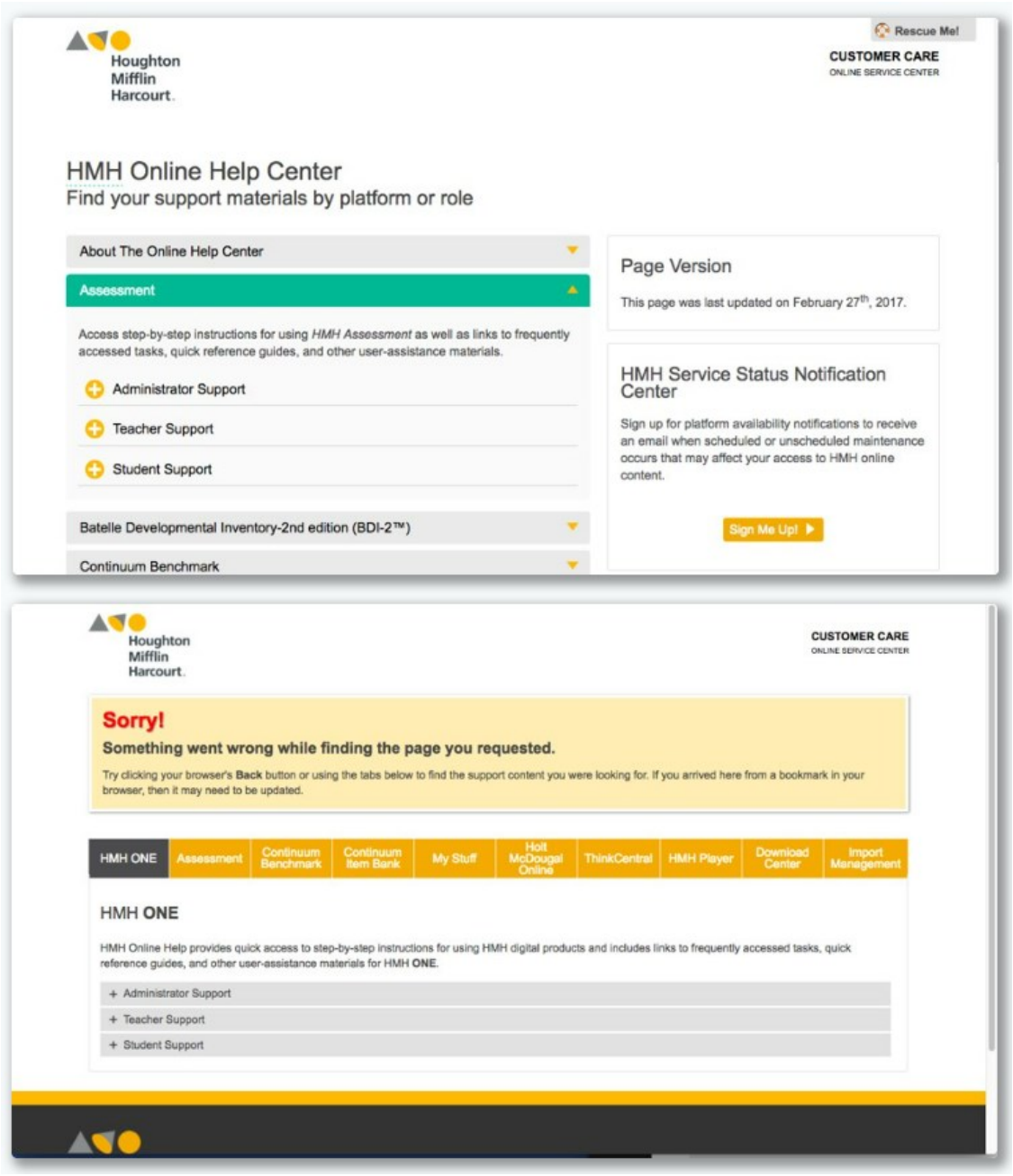

Figure 8. Tabs and collapsible containers can be used to manage how much information is shown at once. (Godfrey, 2017)

Tabs, for instance, are a useful element to categorize information as well as hide it. In recent years, tabs have also started to be replaced by collapsible containers, with plus, minus, or arrow symbols to indicate whether that

container is opened or closed. As seen in Figure 8, the two can be used in tandem to help hierarchize elements and display only the relevant information. (Godfrey, 2017) It should be noted that a plus sign can also have a meaning akin to “create new content”, so if it used to show hidden items, it should consistently do only that task. In addition to information, settings are also often hidden behind menus. In the GCS 1, there are several menu options that need to be hidden. However, they are scattered around the screen, and more importantly, several different menu icons are used. In addition to the top bar menu provided in many desktop applications, there are modern standard menu icons that contain different features. (UX Pickle, 2022) These icons and their explanations are listed in the figure below, although they can sometimes be used interchangeably.

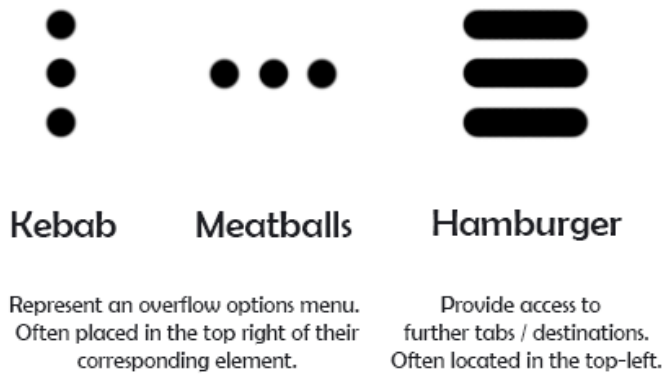

Figure 9. Different menu icons and their meanings.

In UI design, one of the most common conventions is to always minimize the number of clicks required to reach the desired information, as users tend to get frustrated if they must click through many layers of information to achieve their goal. While the “3-Click Rule” comes up often when searching for information on this topic, this convention has its own pitfalls. Minimizing the number of clicks can be useful, but not at the cost of clarity. Rather than only counting the number of clicks required to reach information, UI designers should also consider how much thought process each click takes. In certain cases, lowering

the number of clicks can also mean increasing the number of choices to choose from, and thus possibly adding doubt as to which choice is the right one. An example is provided in the figure below:

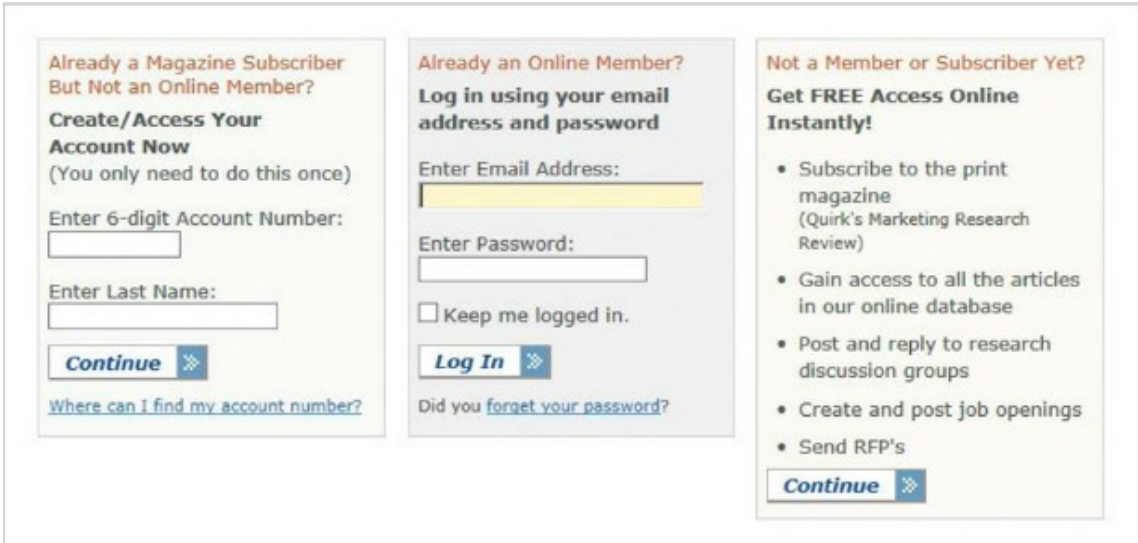

Figure 10. This page presents three options to users before letting them access an online article. (Krug, 2014)

In Figure 10, a user wanting to read an online article is given three choices before they can access the information they are after. At first glance, there is a lot of text to sift through to figure out which choice is the correct one. Next, the user must remember whether they are a Subscriber, an Online Member, or neither one. Lastly, the user will have to go find their account details to see which category they belong in, depending on whether their username is a six-digit number, or an e-mail address. In this case, the article might only be one click away behind the “Log In” and “Continue” buttons, but the thought process for the user so far seems like a long and frustrating one. The New York Times website (Figure 11) provides a clearer path to the required information:

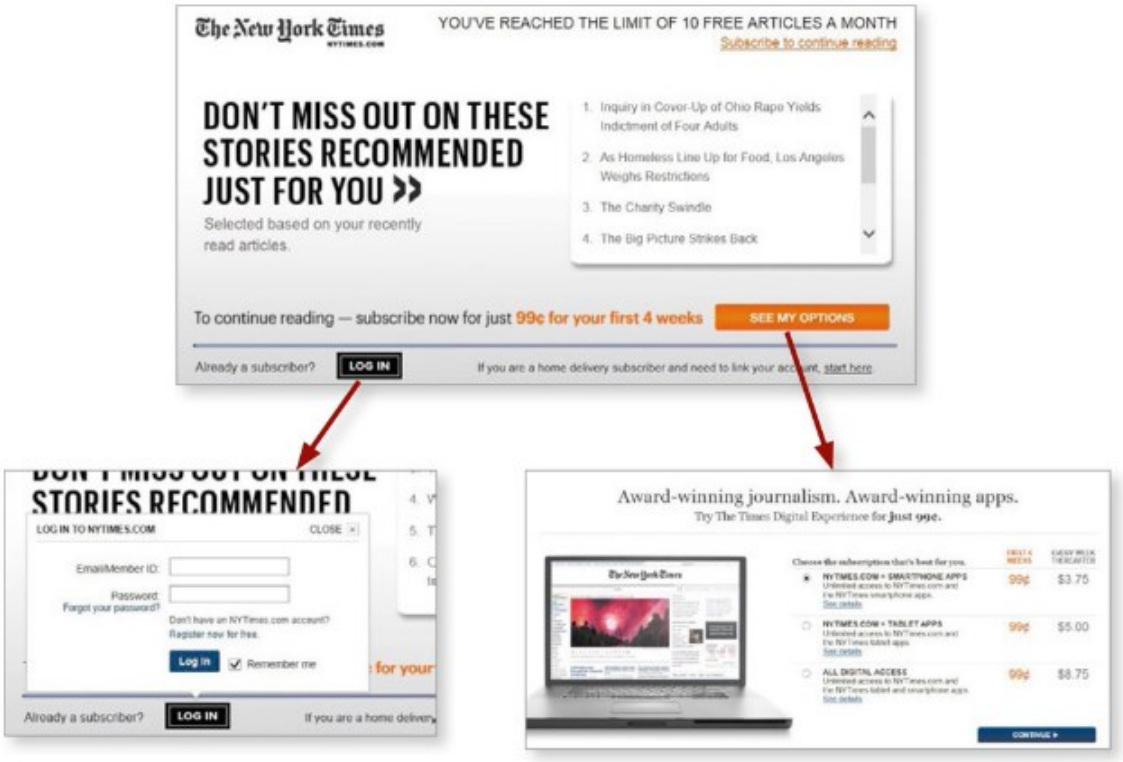

Figure 11. The New York Times website divides user choice into subpages. (Krug, 2014)

In Figure 11, the user is only presented with two choices to begin with: to login if they are already a member, or to subscribe if they are not. Subscribing means further choices to make. Either way, the user must make two or more clicks to achieve their goal, which is to read an article. However, each click brings the user closer to said goal, and the thought process does not leave anything unclear. (Krug, 2014, pp. 43-46)

4 Proposed Solutions

This section provides suggestions on how to solve the biggest issues the GCS 1 has from a UI design point of view as well as recommendations on how to implement the changes in the GCS 2 using Unity. Based on the survey, these are the issues from Figure 3 that can be improved in the scope of this project:

- Modular UI
- Information prioritization
- Larger UI scale
- Accent color change
- Combined “Plan” and “Control” tabs
- Easy-to-find features
- Less information on the right side

Most of the above issues can be corrected by making a few changes to the GCS 1 UI. The only category that was left out from the results to the survey was “Additional light color theme”, as that is a fairly simple problem to fix, and it can be done at any point in the development. It also requires very little programming and was therefore deemed to be outside the scope of this project. Screenshots of the GCS 1 UI are provided in Figure 12 below to provide a comparison target with the new UI design of the GCS 2. Some of the texts have been blurred as per the case company’s request.

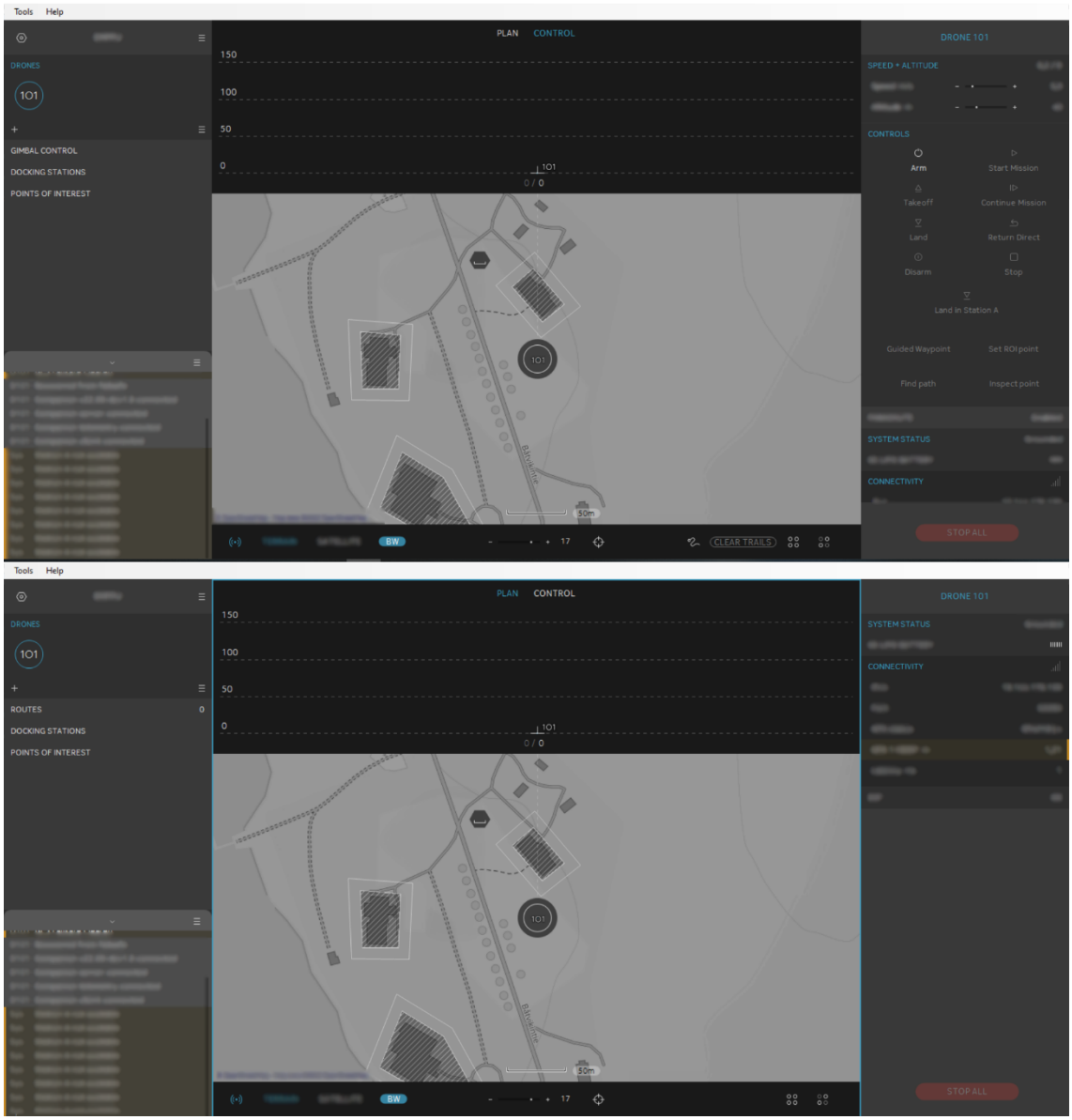

Figure 12. Screenshots of the GCS 1 “Plan” and “Control” views.

A UI mock-up was created using Adobe Photoshop to represent what the UI of the GCS 2 should look like (Figure 13).

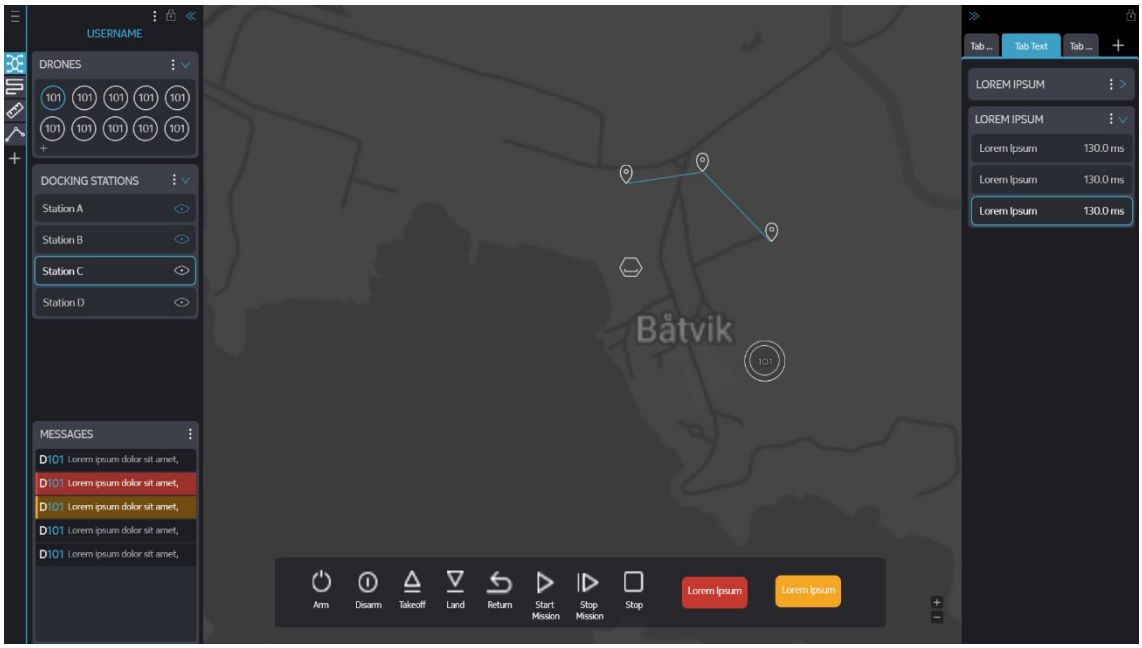

Figure 13. A mock-up for the GCS 2 UI design.

As can be seen from Figures 12 and 13, the general UI layout of the GCS 1 was kept intact. The most apparent changes in the design lie in the change in color to add more contrast, the addition of white space between the different elements to differentiate features from each other, and the addition of a tab system to help the user filter which information is shown at once. It should be noted that some texts or values in Figure 13 have intentionally been left as placeholders and they are not actual information or values from the drones. The reason behind this decision is that additional usage data is needed to determine which information is presumed to be important to be displayed in the basic view, and which information should be part of the advanced view.

4.1 General UI Layout of the GCS 2

In the GCS 1, the “Plan” and “Control” tabs (Figure 12) caused some confusion according to the survey. They are situated at the very top of the screen, and they do not look like tabs at all. They are also situated in an illogical place, as their main function is to change the information available in the panels on the right and left side of the screen. The main source of confusion lies in the fact

that the “Plan” and “Control” views don’t differ much from each other. In the panel on the left, “Gimbal control” is swapped for “Routes”. On the right side, some functionality is added (“Speed + Altitude” and “Controls”), but none is removed. This is a problem, as the main function of a tab is to show a certain set of information that is different from other sets in other tabs.

To solve this problem, the “Plan” and “Control” tabs were entirely removed from the GCS 2 UI design (Figure 13). Instead, the idea behind the panels on either side of the screen has been slightly modified. For the purposes of this project, the panel on the left will be called the Plan panel, and the one on the right will be called the Information Display panel. The Plan panel includes a list of what can be seen on the map, for example docking stations and drones. It also includes a message box. Logically, anything that relates to planning should go in this panel, for example, mission and route planning. The Information Display panel should show relevant information about the elements in the Plan panel. This would include drone information such as telemetry, battery life, speed and orientation, altitude, etc. Information about multiple drones could be displayed at the same time.

The bar across the bottom of the map is called the Quick Access Bar (Figure 13). It contains action commands for the drones as well as emergency action buttons. It could also contain other actions such switching between map modes. In the GCS 1, it is possible to toggle certain information about the drones on the map. A button with this action could also be included in the Quick Access Bar, as well as any action that requires constant availability. The GCS 1 also has a panel at the top visually displaying the altitudes of various drones (although in Figure 12, there is only one drone displayed). This has been entirely omitted from the GCS 2 UI mock-up, as it is not in the scope of this project. It should be noted, however, that it did receive some negative feedback in the survey.

4.2 Project Modularity and Information Prioritization

Writing flexible code can be a daunting task. Ideally, code should be flexible enough, that if some changes are made in some parts of the UI's code, the rest of the UI would still function independently. It can help to think of code as separate modules with limited interaction with each other. One way to achieve this, and to add code security and readability, is to add namespaces to those modules. In the GCS 2, the code is generally divided in three high-level folders: Monobehaviours, Communication, and Domain. The first folder contains all classes that derive from Monobehaviour and can therefore be instantiated as GameObjects within Unity. That code represents the front-end of the application and it is in charge of the UI's behavior and contains little other logic. The Communication section communicates with the rest of the whole product system through the GCS Core. Its role is to pass on commands to the drones using gRPC and receive information from the drones. The code in the Domain folder only acts as a buffer between the Monobehaviours and the Communication parts. By relaying information from one side of the application to the other in this way, only one side has to be updated if small changes are made to either how the UI works, or how the drones communicate to the GCS 2. For example, when the APIs in the GCS Core get updated, only the code in the Communications folder needs to change accordingly, unless entirely new features are added.

Design-wise the UI of the GCS 2 is also separated in several functional parts. Figure 14 below illustrates this concept. The panels on right (outlined in green) contain tabs (outlined in yellow), and tabs include collapsible containers (outlined in red). Dividing the information in this way addresses some of the concerns presented in the "Information prioritization" category in the survey by displaying less at once. While the GCS 1 also has these collapsible containers (Figure 12), according to the survey, it is not clear they are clickable. In the GCS 2, additional panels can be created by clicking and dragging a tab outside of its panel. Doing so then makes a new panel object and nests the tab inside it, as seen in Figure 14.

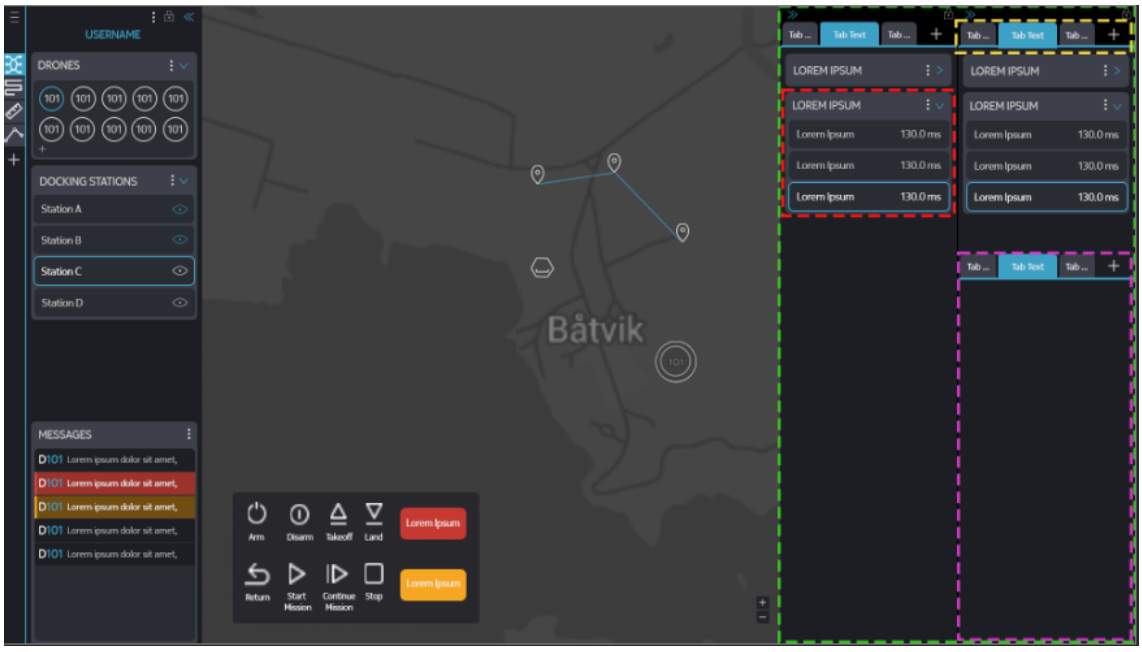

Figure 14. GCS 2 UI mock-up highlighting the modular elements of the design.

Panels can also be divided into several vertical sections (outlined in pink in Figure 14) that house tab groups. The panel on the left has visually different tabs. Both kinds of tabs should not be used in the same UI design, as it would look confusing, but they were combined into one mock-up for the purposes of this project to show two different possible styles. Naturally, tabs can be dragged into any panel or vertical section to go beside other tabs. The sizes of the panels can be adjusted horizontally by dragging the inner edges. Similarly, the vertical height of the sections can be modified.

The lock icons in the top right of the panels can be used to lock/unlock the view. This is needed so that users cannot accidentally move UI elements around while they are flying the drones. One way to do this would be to make sure all elements lock into place when a drone is armed and ready to fly. Creating additional panels reduce the space reserved for the map. Therefore, the number of opened panels should be limited, and they also collapse to the sides using the double arrow icon located in the top left. Additionally, The Quick Access bar no longer fits into the map's space, and therefore needs to change shape. The elements inside the Quick Access Bar need to reorganize

themselves any time the size or shape of the bar changes. Unity's own library provides a handy solution to this problem: the LayoutGroup classes (Figure 15).

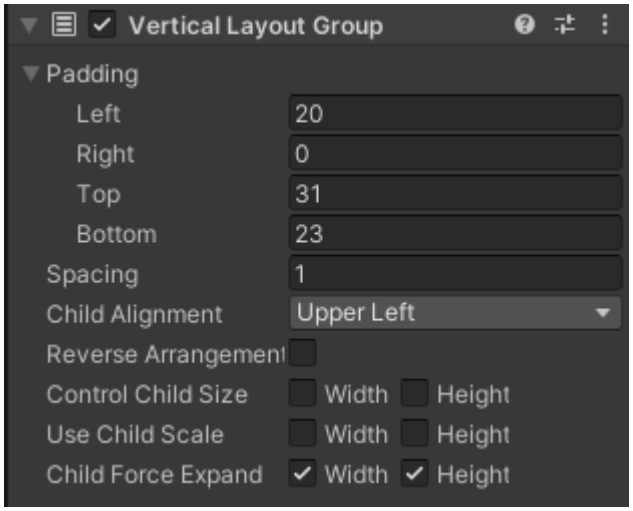

Figure 15. The Vertical Layout Group component of a GameObject in Unity.

As can be seen from Figure 15, the Layout Group component of a GameObject provides a way for developers to control how children of that GameObject align themselves. Vertical and Horizontal Layout Group components have slightly different parameters to adjust, but essentially each child of that GameObject behaves according to the parameters set in this component. If a grid is needed, rather than children stacking next to or on top of each other, a Grid Layout Group can be used.

Dragging and dropping tabs is also simple to implement in Unity. Two classes are responsible for this functionality. The DragDrop class should be attached to the item being dragged. The Unity library provides a handy set of interfaces that handle mouse pointer events. The item that can be dragged could also benefit from the CanvasGroup component, as it allows blocking Raycasts and adjusting parameters such the alpha of the object for added visual effect.

The second class, ItemSlot, should be attached to the slot into which the tab is being dragged. All it needs to do is implement the IDropHandler interface, which sends an event that is fired when the item is "dropped" above the item slot.

Next, to get the desired “snapping into place” effect, the transform of the item should be adjusted to the transform of the slot.

Containers are nested within the tabs and contain the actual information the user seeks. To save space, these containers can be collapsed. The CollapsibleContainer (Listing 1) class oversees operating the containers. Additionally, it works in tandem with the AnimationPlayer class that contains various logic for animations, such as turning the arrow icons on the containers. It also contains a LerpHeight() method, shown below.

```
public IEnumerator LerpHeight(RectTransform valueToLerp, float width, float
startValue, float endValue, float lerpDuration) {
        IsRunning = true;
        float timeElapsed = 0;
        while (timeElapsed < lerpDuration) {
            valueToLerp.sizeDelta = new Vector2(width, Mathf.Lerp(startValue,
endValue, timeElapsed / lerpDuration));
            timeElapsed += Time.deltaTime;
            yield return null;
        }
        valueToLerp.sizeDelta = new Vector2(width, endValue);
        IsRunning = false;
    }
```

Listing 1. The LerpHeight() utility method can be used to animate the height of a GameObject.

The Lerp() method (Listing 1) is provided in Unity’s own library, and incrementally changes a given start value to an end value over time. The developer can therefore control how fast a container should reach a certain height.

The user should be able to do other things on the UI while the animation is playing. Therefore, the LerpHeight() is called as a coroutine in the CollapsibleContainer class, as coroutines execute in an asynchronous manner, thus freeing the main thread. The CollapsibleContainer class has a reference to each child GameObject of this object. When the user clicks on the arrow button or on the container name, the OpenCloseContainer() method is called, which in turn calls the UpdateHeight() method as a coroutine (Listing 2).

```
public void OpenCloseContainer() {
        _isOpen = !_isOpen;
        StartCoroutine(UpdateHeight(true));
    }

public IEnumerator UpdateHeight(bool useLerp) {
        yield return new WaitForEndOfFrame();
        var childrenHeight = 0f;
        foreach (var child in children) {
            var rt = (RectTransform)child.transform;
            childrenHeight += _layoutGroup.spacing + rt.rect.height;
        }
        var startHeight = _rectTransform.rect.height;
        var targetHeight = _isOpen ? childrenHeight + _layoutGroup.padding.top
+ _layoutGroup.padding.bottom : _closedHeight;
        var lerpDuration = childrenHeight / 2000;
        if (useLerp) {
            if (_lerpCoroutine != null)
                StopCoroutine(_lerpCoroutine); // Stopping previous animation
if it is still running
            _lerpCoroutine =
StartCoroutine(_animationPlayer.LerpHeight(_rectTransform,
_rectTransform.rect.x, startHeight, targetHeight, lerpDuration));
            StartCoroutine(ActivateChildren());
        }
        else
            _rectTransform.sizeDelta = new Vector2(_rectTransform.rect.width,
targetHeight);
    }
```

Listing 2. The methods that open and close the collapsible containers.

The total height of the children is taken into account when determining the target height of the container when it has been opened or closed. Since children should be able to be added at runtime, the class also contains the AddChild() and RemoveChild() methods. These animations could also be done using Unity's Animator feature but animating from the code gives more control to the developer what happens, for instance, when children are added or removed.

The layout described above helps with prioritizing which information is shown on the screen. Another feature planned in the GCS 2 UI mock-up is customizable tabs. As users might consider different information as relevant, they can create their own set by clicking on the "+" button situated next to or under the tabs (Figure 13). This will create an empty tab that can then be filled with the required information, either by dragging it from an existing tab, or by right clicking the empty space. This opens a tooltip menu with the option to add information from a list.

The message box of the GCS 1 in the bottom-left corner of the screen (Figure 12) also received some negative feedback concerning which information is visible and when. The problem is that many messages in the box are repeated multiple times and, therefore, the box gets full quickly, forcing the user to scroll through a lot of text to find what they are looking for. To solve this problem, the message box should combine identical messages into one message and only update its time stamp. Generally, the messages should appear in order with the newest messages at the top. However, an exception can be made for warnings and critical errors. Those can be anchored to the top until resolved.

In conclusion, Unity provides developers with generally accessible and convenient tools to build a UI even using the older uGUI. When building a UI in this way, the use of prefabs is a must. Having a modular UI means many elements must be moved, transformed, or instantiated at runtime. Additionally, changing the look or the functionality of the UI is easier when using prefabs, as if a change is made in a prefab, that change can be applied to all instances of that prefab. The use of generic prefabs for all use cases can seem counter-intuitive at first, but Unity provides a solution to edge cases in prefab variants. A variant is a copy of the original prefab that varies from it slightly. When updating a component from the original prefab, for instance, that change can be applied to all or some variants. However, updating a variant will not affect the original prefab. (Unity Technologies, 2022)

4.3 The Use of Colors, Contrast and Shapes in the GCS 2 UI Mock-up

The colors used in the GCS 1 received both positive and negative feedback from the survey. Mostly, respondents liked the dark theme with a blue accent color, as it seemed pleasing to the eye. However, at the same time, many felt like there was not enough contrast between the different elements. Therefore, the GCS 2 UI mock-up (Figure 13) uses a similar theme with adjusted colors. To put it simply, the dark greys are darker, and the light greys are lighter. A very faint blueish hue was added to these background colors to give the UI a slightly warmer, deeper feel. The blue hue also makes the blue accent color feel more

in tune with the background colors. The order in which the background colors are displayed, for example in the containers, has also changed. The darkest and the lightest background colors are now placed side-by-side to increase the contrast. This helps with differentiating the containers more clearly from each other. In the GCS 2 UI mock-up, the containers are spaced further apart from each other and the edges of the panels. As stated in section 3 the use of white space helps guide the user's eye to the important information inside the container. The general scale of text was also made larger, and many texts use the bold style, as very thin text can seem darker than it is. This creates the illusion of higher contrast and helps with readability.

When it comes to the general UI and text scale, perhaps the UI settings could have a scale option. This would require different versions of the UI to be made in Unity, and then loaded when the user changes the settings, the implementation of which is deemed out of the scope of this project. A UI scaling option is common in many applications, especially in games. In fact, the scale and contrast in the UI design should be considered not only for comfort, but also accessibility. For example, Microsoft provides some guidelines for making accessible games (Microsoft, 2022).

The use of warning colors received some praise from the respondents, so similar colors were used in the same places in the GCS 2 mock-up, as shown in Figure 16 below. Red is used emergency situations where immediate action is required by the user and yellow is used when the user needs to be warned of a potential danger.

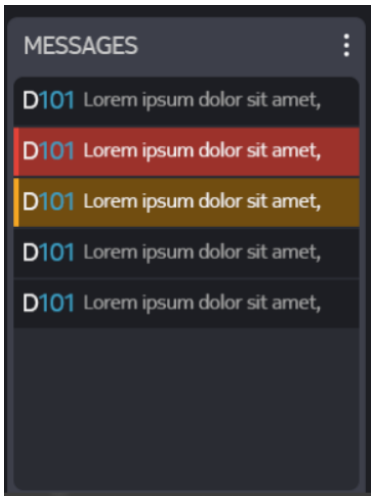

Figure 16. Screenshots of a section of the GCS 2 UI mock-up using warning colors as an example.

These warning colors (Figure 16) could, however, also be used elsewhere in the UI. For example, if a critical situation arises with a drone, both the drone icon on the map and in the drone list in the top-left of the screen (Figure 13) could start blinking in the appropriate warning color.

The blue accent color seemed to cause some confusion among the respondents of the survey. On the one hand, it is looks pleasing, but on the other hand, which element is in the “open”, “highlighted” or “chosen” state is unclear in the GCS 1. The GCS 2 UI mock-up attempts to correct this problem. The following figure shows a section of both UIs side-by-side for comparison.

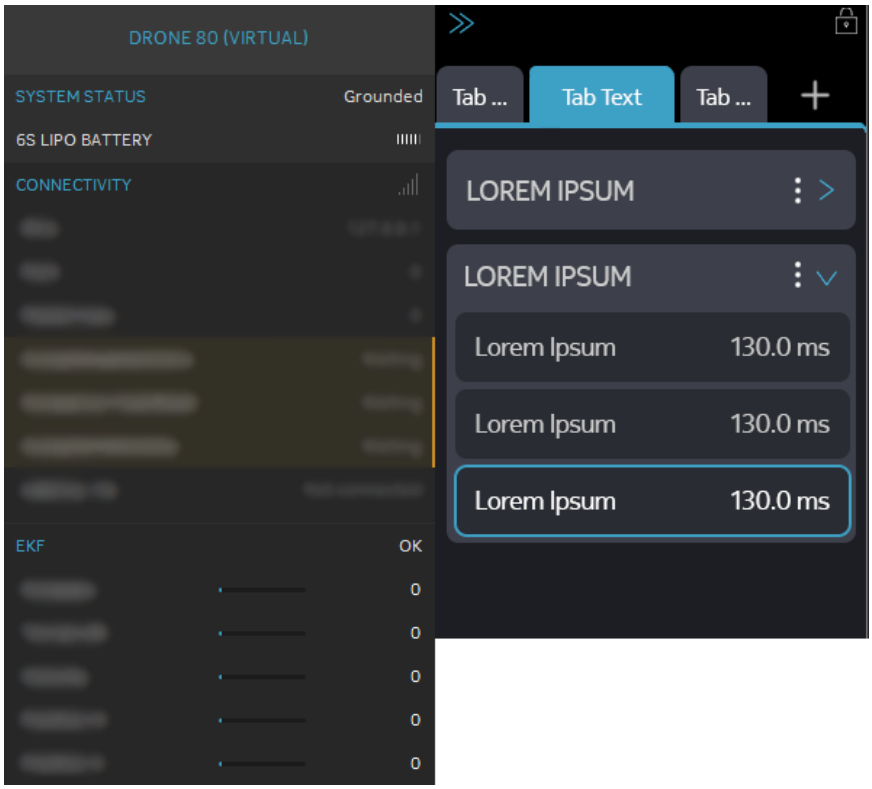

Figure 17. Sections of the GCS 1 UI (left) and the GCS 2 UI mock-up (right) depicting the use of the blue highlight color.

The use of the blue accent color as an indication that a container is open in the GCS 1, or that a value has been clicked on (Figure 17) can cause confusion. Generally, if a UI element is in the “chosen” state, which means that is has been clicked on or opened, it is expected to be highlighted. This means that it should be lighter in color than it’s “non-chosen” counterparts on the screen. For example, the “system status”, “6S lipo battery”, “connectivity”, and “EKF” elements in Figure 17 are all collapsible containers. However, the “6S lipo battery” container looks like it is a text element that has been highlighted from the other text elements, rather than a collapsible container that is currently in the closed state. Two steps were taken to address this problem. The collapsible containers in the GCS 2 UI mock-up were made to look more “clickable” by adding an arrow icon to the right, but also by adjusting the shape and color. Because its rounded edges and its light color, the new container looks like it “pops out” of the background, which makes it seem more interactable. Secondly, the blue highlight color is no longer used in texts, but rather in

elements that around the texts. All texts are now light grey, except if they are highlighted; when clicked on, texts become white. It should be noted that some testing should be done on the use of the blue frame around the last element inside the container list in Figure 17. It is unclear whether this element will now be perceived as highlighted. However, if warning colors are applied to the frame, it could be used to draw attention from the user.

5 Conclusion

Overall, it can be said that the objectives of the study were met. The main focus of this thesis was to provide suggestions on how to improve on the UI design of the GCS1, and the mock-up UI of the GCS 2 provides a good starting point for future development. Unfortunately, not all suggestions provided in section 4 were implemented during this project, but they are straightforward, and therefore the development time or difficulty should not be an issue in future. While this study was useful to gain a basic understanding of which features should be improved in the GCS 2 compared to the GCS 1, some aspects of this project could, in retrospect, have been better planned.

The most problematic area of this study was the survey. Firstly, the number of participants was too low to provide any real statistically relevant information. Another major problem was the lack of meaningful quantitative data. The ratings would have been more useful if they touched on more specific subjects rather than just general categories in the UI design. As for the qualitative data, i.e. the open-ended questions, it was difficult to transform to it into quantitative data. Perhaps the use of a tool specifically designed to analyze freeform answers would have been useful, but it was deemed to be outside the scope of this project. Also, distributing the statements into the different categories seen in Figure 3 proved quite difficult. This is because some statements could have been included in several different categories. In general, however, reading through the feedback was very useful to this study even if the analysis could have been more descriptive.

The request for UI modularity being so popular was a surprise. The GCS 2 UI mock-up provides an example on how to add modularity in multiple ways. However, perhaps not all levels of modularity (panels, tabs and containers) should be implemented as this can add some unnecessary complexity for the user. Perhaps the modularity of the UI will not be as central to this project going forward, if the prioritization of information is done right, as that could reduce the need for the user to be able to filter the information themselves.

As a result of this study, a better understanding of how to design the GCS 2 was achieved. The GCS UI mock-up provides a good basis on which to build the UI of the GCS 2 with the suggestions detailed in section 4. It should be noted, however, that user experience data will have to be gathered once the proposed layout has been implemented. Additionally, there should be further discussion about precisely which information is considered essential enough to appear in the basic layout, and which information should be hidden from view, as the GCS 2 UI mock-up only provides an improvement suggestion on a high level. Further research should also be conducted on competing products on the market and how those products solved the issues mentioned in this study.

References

ArduPilot Dev Team, 2021. *Choosing a Ground Station.* [Online]
Available at: https://ardupilot.org/copter/docs/common-choosing-a-ground-station.html
[Accessed 12 9 2022].

Babich, N., 2020. *11 Ways to Add More Visual Weight to UI Object.* [Online]
Available at: https://uxplanet.org/11-ways-to-add-more-visual-weight-to-ui-object-6aca19c7bc97
[Accessed 19 10 2022].

Codecademy, 2022. *What is .NET?.* [Online]
Available at: https://www.codecademy.com/article/what-is-net
[Accessed 14 9 2022].

Designlab, 2022. *The Grid System: Importance of a Solid UX/UI Layout.* [Online]
Available at: https://designlab.com/blog/grid-systems-history-ux-ui-layout/
[Accessed 19 10 2022].

Godfrey, P., 2017. *Show and Hide Content.* [Online]
Available at: https://www.learningtoo.eu/articles/show-and-hide-content.htm
[Accessed 21 10 2022].

Gwentech LLC, 2020. *The Advantages of Modular Software and Programming.* [Online]
Available at: http://gwentechembedded.com/the-advantages-of-modular-software-and-programming/
[Accessed 18 10 2022].

Krug, S., 2014. *Don't Make Me Think.* Third Edition ed. San Francisco: New Riders.

Microsoft, 2022. *abstract (C# Reference).* [Online]
Available at: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract
[Accessed 18 10 2022].

Microsoft, 2022. *Xbox Accessibility Guidelines V3.1.* [Online]
Available at: https://learn.microsoft.com/en-us/gaming/accessibility/guidelines
[Accessed 26 10 2022].

Parker, S., 2021. *Here's how to enable dark mode for googles maps on iOS..* [Online]
Available at: https://www.neowin.net/news/heres-how-to-enable-dark-mode-for-google-maps-on-ios/
[Accessed 19 10 2022].

Pitaksarit, S., 2021. *UnityEvent Serialization Research.* [Online]
Available at: https://gametorrahod.com/unityevent-serialization-research/
[Accessed 14 9 2022].

Unity Technologies, 2021a. *Comparison of UI Systems in Unity.* [Online]
Available at: https://docs.unity3d.com/Manual/UI-system-compare.html
[Accessed 2 9 2022].

Unity Technologies, 2021b. *Create User Interfaces (UI).* [Online]
Available at: https://docs.unity3d.com/Manual/UIToolkits.html
[Accessed 2 9 2022].

Unity Technologies, 2021c. *UI Toolkit.* [Online]
Available at: https://docs.unity3d.com/Manual/UIElements.html
[Accessed 2 9 2022].

Unity Technologies, 2022. *Coding in C# in Unity for beginners.* [Online]
Available at: https://unity.com/how-to/learning-c-sharp-unity-beginners#:~:text=The%20language%20that's%20used%20in,variables%2C%2

0functions%2C%20and%20classes.
[Accessed 14 9 2022].

Unity Technologies, 2022. *Platform development.* [Online]
Available at: https://docs.unity3d.com/Manual/PlatformSpecific.html
[Accessed 14 9 2022].

Unity Technologies, 2022. *Prefab Variants.* [Online]
Available at: https://docs.unity3d.com/Manual/PrefabVariants.html
[Accessed 04 11 2022].

Unity Technologies, 2022. *The world's leading platform for real-time content creation.* [Online]
Available at: https://unity.com/
[Accessed 14 9 2022].

Unity Technologies, 2022. *Unity Asset Store.* [Online]
Available at: https://assetstore.unity.com/
[Accessed 14 9 2022].

UX Pickle, 2022. *Know your menu: Hamburger vs. Kebab.* [Online]
Available at: https://uxpickle.com/know-your-menu-hamburger-vs-kebab/
[Accessed 21 10 2022].

UX studio, 2017. *Does a Modular Design Approach Future-Proof Your Concept?.* [Online]
Available at: https://uxstudioteam.com/ux-blog/modular-design/
[Accessed 18 10 2022].

The Survey Questions

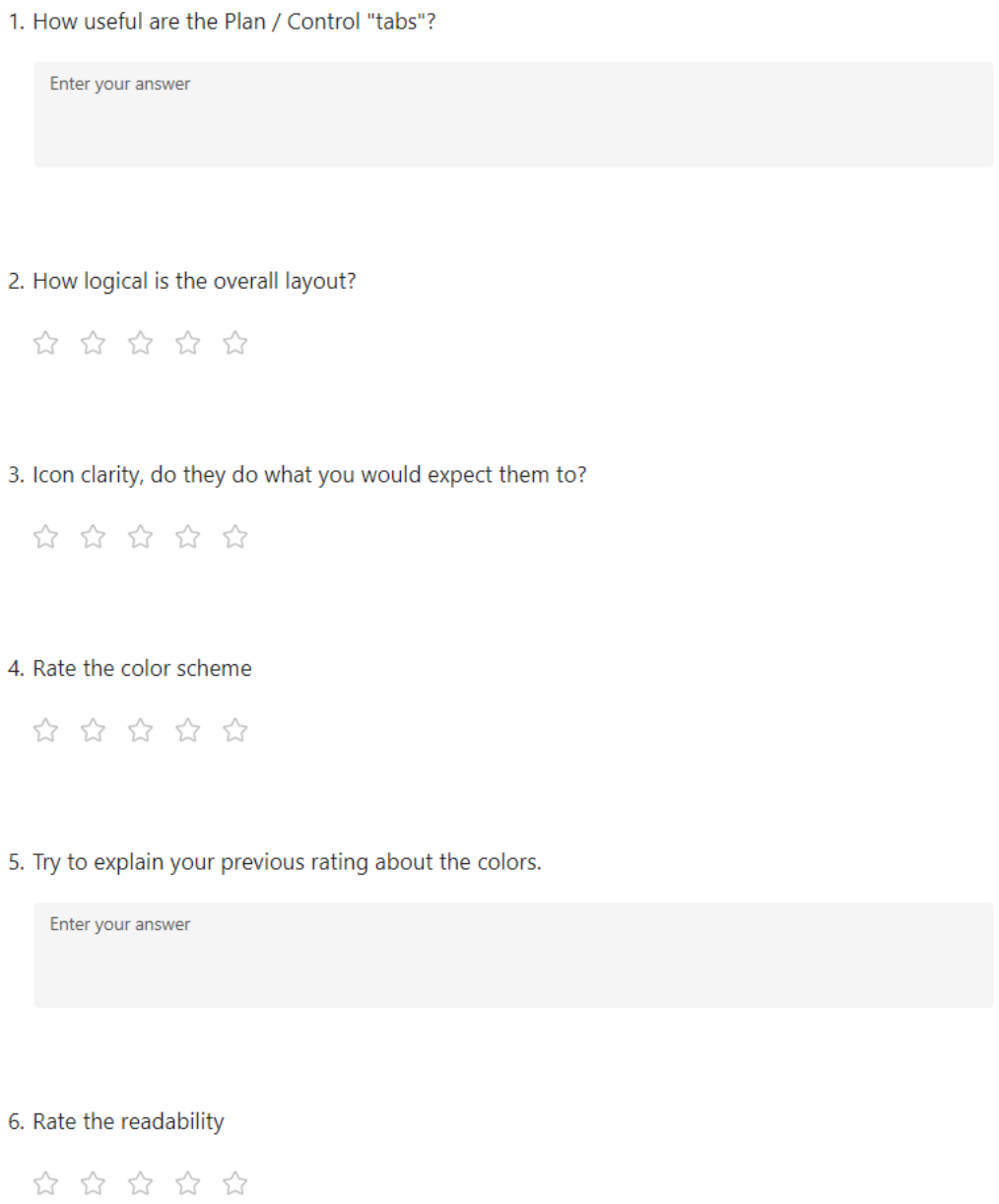

Figure 1. Questions 1-6 in the survey.

7. Rate how clunky/fluid the UI feels (0 stars = feels very clunky, 5 stars = super fluid)

☆ ☆ ☆ ☆ ☆

8. Rate how intuitive the UI feels

☆ ☆ ☆ ☆ ☆

9. Can you put your finger on why you think it is/isn't intuitive?

Enter your answer

10. Do you feel like you have enough information? Or is there too much at once? (Please try to be specific)

Enter your answer

11. How important do you think the UI modularity is? Do you wish you could move things around? (Please try to be specific)

Enter your answer

12. Thank you for answering the survey! Any last thoughts you'd like to share about the UI? Suggestions on improvements?

Enter your answer

Figure 218. Questions 6-12 in the survey.