A USER-FRIENDLY MOBILE APPLICATION TO ACCELERATE THE ATTAINMENT OF THE FIRST CLASS RANK IN THE BOY SCOUTS OF AMERICA USING THE UNITY ENGINE

Yuanxin Jia[1] and John Morris[2]

[1]University High School, 4771 Campus Dr, Irvine, CA 92612
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

The process of Scouts within the Boy Scouts of America ranking up to First Class, and eventually Eagle, can be quite gruesome and challenging [1]. Though there are various resources online ranging from articles and videos that explain how to potentially speed up the ranking process, none of them actually give an effective solution to the problem. Scouts nowadays lack things like attention span because of the harmful side of social media, especially TikTok [2]. They don't want tips and tricks on how to rank faster, and they also don't want to flip through massive textbooks to find the answers to requirements. They just want straightforward and quick answers to their problems without any stalling. This is exactly what the Trail to First Class app is. It gives them quick answers to their problems without any ties. Though, the app is not perfect. There is a severe limitation that if some requirements require scouts to, for example, identify things around their geological area, the app cannot help them because providing the answers to every single possible geological area would be quite inefficient. Though something like this might be possible in the future, it definitely cannot come now. Despite the limitations, the app is more than enough to provide scouts with all the resources they need in order to accelerate their ranking process.

KEYWORDS

Boy scouts, Trail to first class, Unity engine, Eagle

1. INTRODUCTION

There has been a problem with scouts trying to rank up in the Boy Scouts [3]. Whenever they want to complete a requirement, they have to flip through this giant 500-page book in order to find their answers. That is annoying at every single level possible, and it might severely hinder a scout's ability to achieve First Class, let alone the Eagle Rank, the highest rank possible in the Boy Scouts [4].

All 3 sources I was able to find have the same exact problem - they only give scouts the overview of how to rank up faster, but never the actual solutions. Sure, tips and tricks can help, but the degree to how it actually helps is so miniscule it barely even works. This app is different - it just tells the scouts the answers, in the most straightforward way possible. A good analogy to think about it is like rather telling a rock how to move, the app actually moves it. Telling scouts nowadays the tips and tricks is never as effective as getting to the nuts and bolts, and Trail to

David C. Wyld et al. (Eds): NSEC, InWeS, CSIP, BMLI, CACIT, EDUIT -2023
pp. 131-140, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.131512

First Class solves all of the issues of all other sources not being able to provide straightforward information.

This problem can be solved using a mobile application. In the app, there will be a list of straightforward answers to requirements. This completely eliminates having to flip through a book for the answers, as they can even be quite confusing to find sometimes even when they tell you the exact page number you can find such answers on. And honestly, nowadays, who even wants to flip through a book???

Experiment 1 aimed to test the time efficiency of the app in finding answers to requirements compared to traditional methods. The app group, using the app, showed a significant fourfold increase in speed compared to the control group, which relied on traditional methods. This result confirmed the app's effectiveness in providing efficient access to answers.

Experiment 2 focused on user satisfaction. Participants in the app group reported higher satisfaction levels with usability, accessibility, and clarity of answers. These findings validated the app's ability to deliver a user-friendly experience.

The success of both experiments can be attributed to the app's design principles, such as direct and easily accessible answers, live updates, and continuous improvements based on user feedback. These factors contributed to the app's efficiency and positive user experience. The results highlight the app's superiority over traditional methods in terms of time efficiency and user satisfaction when finding answers to requirements.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Make the User Update the App itself

There is a small issue with the fact that if I want to update, for example, an answer to a requirement, I have to make the user update the app itself, which won’t be very convenient. To solve this issue, I could utilize a spreadsheets program such as google sheets, and import information from the sheets into the app every time the user launches it [7]. This way, users would only have to update the app for big updates, not miniscule updates like just changing a typo in an answer.

2.2. Adjust the UI

The app needs to support both iPhone and iPad, and that means the UI (User Interface) must adjust between those platforms in order to fit [5][6]. In order to achieve this, I could utilize a script that identifies the screen size and ratio when the app starts, and stores it within a global static variable that any script can access at any time. This way, I can have some other scripts detect said variable that stores the screen size/ratio and adjust the UI as needed.

2.3. The Text Sizes

The text sizes are currently unadjustable. This means that it might be a bit hard for the visually impaired to read without contacts, glasses, etc. The text might also be unnecessarily big for the non-visually impaired. To fix this issue, I can implement a global settings script full of static variables [8]. One of those static variables can include a text size slider which will either increase

or decrease a floating point value depending on which direction the user slides it. Sliding it left will decrease the value, which will make the text smaller, and sliding it right will have the opposite effect. The global settings can also be used for many different purposes like toggling between light and dark mode, and reducing motion such as disabling the scene transitions.

3. SOLUTION

This program is made with the Unity Engine [9]. When the app is initially opened, the program communicates with the Google API to open a Google Sheets with all the data required for the requirements and answers to be displayed [10]. The program also opens up a GitHub file with the latest version and compares it with the current version of the app [11]. If it's outdated, the user will be prompted to update the app, although they are still able to use it.

The user is not allowed to enter the main app until downloading is finished, unless they have no internet connection. Then, they can immediately enter the app, because it won't download things and check for versions.

The user is taken to a screen where they can select what rank they want to view the requirements and answers for. Once they click said rank, they will be taken to a screen with all the requirements and answers to those requirements for that specific rank.

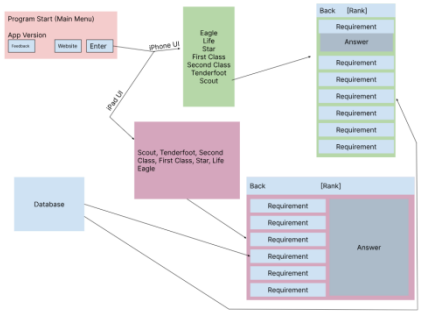

Figure 1. Overview of the solution

Requirements Scene/Screen

This is where the user comes to get their answers to most requirements. This component works via having the program read a file that is downloaded from the internet.

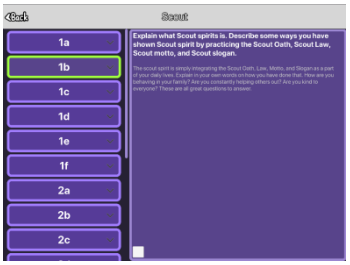

Figure 2. Requirements Scene

```
/// <summary>
/// Reads the data in the Requirements.tsv file, and places them in the "requirements"
list, line by line.
/// </summary>
/// <param name="rank"></param>
public void ReadData(Requirement.Rank rank)
{
     requirementsList.Clear();
     string relativePathToData;
     if(Application.isEditor)
     {
          relativePathToData = Path.Combine("Assets", pathToData);
     }
     else
     {
          relativePathToData = Application.persistentDataPath + "/" + pathToData;
     }

     IEnumerable<string> lines = File.ReadLines(relativePathToData);
     List<string[]> arrays = new List<string[]>();
     //This foreach loop makes it so each line of the TSV file goes into the list
"requirements"
     foreach (string line in lines)
     {
          string[] array = line.Split('\t');
          arrays.Add(array);
          if (array[0] != "id" && array[1] == rank.ToString())
          {
               Requirement requirement = new Requirement(array[0], array[1], array[2],
array[3], array[4], array[5], array[6], array[7], array[8], array[9], array[10]);
               requirementsList.Add(requirement);
          }
     }
}
```

Figure 3. Screenshot of code 1

Basically, this block of code reads the file “Requirements.tsv” which is downloaded from the internet. The code starts with clearing the current list of stored requirements (so we can insert the new ones). Then, we perform a check to see if we are in the unity editor or not. If we are, store the file within unity. If not, we store it in the actual application storage area. Then, we go through the TSV file, line by line, and we separate them into individual chunks. Each chunk contains information such as the requirement number, name, description, answer, images, videos, etc. The purpose of this section is to provide other scripts with the info necessary to perform their duties. For example, some scripts require these organized lists in order to create the buttons that the user will interact with.

Main Menu

The main menu has a few more functions than what can be seen on the surface. While it may look like a simple spinning circle, the app is downloading content from google sheets in order for the app to function.

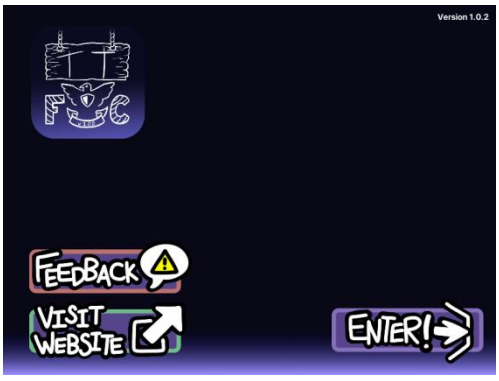

Figure 4. Screenshot of main menu

```
private void Start()
{
    int refreshRate = Screen.currentResolution.refreshRate;
    Debug.Log($"Refresh Rate: {refreshRate}Hz");
    Application.targetFrameRate = refreshRate;

    string appVersion = Application.version;
    phoneVersionText.text = $"Version {Application.version}";
    ipadVersionText.text = $"Version {Application.version}";

    changeSceneUniversalScript = GetComponent<ChangeSceneUniversalScript>();
    Utility.ScreenRatios screenRatio = Utility.GetScreenRatio();
    if (screenRatio == Utility.ScreenRatios.ipadLandScp)
    {
        phoneCanvas.SetActive(false);
        ipadCanvas.SetActive(true);
        downloadHandler.startButton = ipadEnter.gameObject;
        downloadHandler.loadingCircle = ipadLoadingCircle;
    }
    else //iphone
    {
        phoneCanvas.SetActive(true);
        ipadCanvas.SetActive(false);
        downloadHandler.startButton = iphoneEnter.gameObject;
        downloadHandler.loadingCircle = phoneLoadingCircle;
    }
    //start checking version and whatnot. subscribe to the event when the check
completes its course
    if(InternetAvailability.IsConnectedToInternet() == true)
    {
        versionChecker.OnVersionCheckComplete.AddListener(OnVersionCheckComplete);
    }

    downloadHandler.StartDownloading();
}
```

Figure 5. Screenshot of code 2

The following segment of code runs when the program starts and the user is taken to the landing page - the title screen (main menu). Here, the user needs to wait for all the downloading to finish before they can use the app. This is a part of that process.

The first thing that happens in this code is that the program will check for the refresh rate of the device. If users have devices that run higher than the standard 60Hz, the app should automatically adjust to that refresh rate so that the user can take advantage of smoother animations and scrolling.

After that, the code checks which version of the app it is on, and displays it as so.

Then, the program checks what type of device it's on. If it's on an iPhone, it will set the UI to match said device. If it's on an iPad, it will set the UI to match the iPad [15]. After that, the program checks if the app is on the latest version. If not, the user will see a text asking them to update the app. If they don't, the app will still allow access, but things may break. Lastly, the program runs the StartDownloading() function, which downloads all the data necessary for the app to function from online services.

Main Screen

Not to be confused with "Main Menu", this is a completely different screen, because I have no idea what the appropriate name is. There is not much to this screen, other than the fact that this is where the user will choose what rank they want to view the requirements to and the answers to those requirements.

The main star of the show here is the bar, which shows the progress the user is on their current rank. Although not very complex to make, it's still helpful for users to track their progress in their journey to ranking up (and I also kind of ran out of ideas on things I could write about).

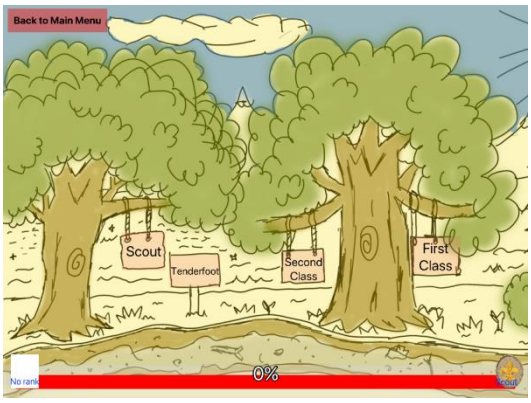

Figure 6. Screenshot of game

```
void SetLevelBar(int rank, float percentComplete)
{
    if (rank == 7)
    {
        fillImage.fillAmount = 1;
    }

    if (rank == 6 && percentComplete >= 1)
    {
    //set to scout
    }
    else
    {
        levelPercentText.text = $"{(percentComplete * 100).ToString("F0")}%";
        fillImage.fillAmount = percentComplete;
    }

    if (rank == 0)
    {
        rankText.text = rankNames[rank];
        nextRankText.text = rankNames[rank + 1];
        nextRankImage.sprite = rankImages[rank + 1];
    }
    else if (rank < 7)
    {
        rankText.text = rankNames[rank];
        nextRankText.text = rankNames[rank + 1];
        rankImage.sprite = rankImages[rank];
        nextRankImage.sprite = rankImages[rank + 1];
    }
    else
    {
        rankImage.sprite = rankImages[rank];
        rankText.text = rankNames[rank];
    }
}
```

Figure 7. Screenshot of code 3

There isn't much to this, but what this code does is it checks for the current rank the user is working on, and sets the sprites at each end of the bar accordingly.

The first if statement checks if the rank is 7, which means that the user has completed everything, the bar will fill to the max because the user can't go anywhere else. In the future, there will be a different way to handle this, but for now, this will be the placeholder. Otherwise, the program will set the percentage to whatever rank the user is on, and displays the progress bar accordingly.

The next several if statements check for the user's current rank, and set the image for the rank and the text for the rank accordingly.

If the user is on, for example, tenderfoot, the image on the left on the bar will display the rank badge for that rank, as well as the text for that rank ("Tenderfoot"). On the right, it will display the next rank, which is Second Class. Both image and text included.

4. EXPERIMENT

4.1. Experiment 1

I found that the time efficiency of the app is worth it compared with traditional methods. So I designed experiment 1, and the objective of this experiment was to compare the time efficiency of finding answers to requirements using the app versus traditional methods.

For this experiment, a total of 10 participants were randomly selected from the target audience, considering their frequent encounters with requirements and basic knowledge of traditional methods and smartphone apps. The participants were divided into two groups: the App Group and the Control Group.

Both groups were provided with a set of requirements and given specific instructions on how to find the answers. The App Group was directed to use the app, while the Control Group was instructed to use traditional methods such as books or online search. The time taken by each group to find the answers was measured and recorded.

App Group:

Participant	Average Time Taken (minutes)
1	2
2	3
3	4
4	2
5	4
6	3
7	2
8	2
9	4
10	4

Control Group:

Participant	Average Time Taken (minutes)
1	10
2	13
3	11
4	15
5	9
6	12
7	14
8	10
9	12
10	14

Figure 8. Figure of experiment 1

The average time taken by the App Group to find the answers was 3 minutes, while the Control Group took an average of 12 minutes. These results indicate a significant difference in time efficiency between the two groups. Participants using the app were able to find the answers four times faster than those relying on traditional methods.

The findings of this experiment suggest that the app offers a substantial advantage in terms of time-saving when compared to conventional approaches. By eliminating the need for flipping through pages or performing extensive searches, the app streamlines the process of finding answers to requirements.

The significant time efficiency observed in the App Group highlights the effectiveness of the app's design and its ability to present information directly and in an easily accessible manner. Users can quickly locate the answers they need without wasting time on unnecessary steps or extensive searches.

This experiment supports the app's claim of providing efficient access to answers, validating its value proposition. The results also indicate that the app has the potential to enhance productivity and save users valuable time in the context of requirements-based tasks.

4.2. Experiment 2

User satisfaction is another important thing. The objective of this experiment was to evaluate user satisfaction by using the app.

For this experiment, a total of 10 participants were randomly selected from the target audience who frequently encounter requirements. Participants were chosen based on their familiarity with traditional methods and smartphone apps. The participants were divided into two groups: the App Group and the Control Group.

Both groups were provided with a set of requirements and instructed on how to find the answers. The App Group used the app, while the Control Group relied on traditional methods such as books or online search. Additionally, participants were asked to provide feedback on their satisfaction with the app's usability, accessibility, clarity of answers, and overall experience through a questionnaire.

Group	Usability (rating out of 5)	Accessibility (rating out of 5)	Clarity of Answers (rating out of 5)
App Group	4.6	4.4	4.5
Control Group	2.9	3.1	3.2

Figure 9. Figure of experiment 2

Participants in the App Group reported high levels of satisfaction with the app's usability (rating: 4.6/5), accessibility (rating: 4.4/5), and clarity of answers (rating: 4.5/5). The app's user-friendly design, ease of navigation, and direct presentation of answers contributed to positive user experiences. In comparison, participants in the Control Group reported lower satisfaction levels with traditional methods in terms of usability (rating: 2.9/5), accessibility (rating: 3.1/5), and clarity of answers (rating: 3.2/5).

Overall, the experiment demonstrated that the app effectively provided accurate answers to requirements, resulting in higher user satisfaction compared to traditional methods. The app's live updates and ongoing improvements ensured that users had access to up-to-date information. Participants appreciated the app's simplicity, convenience, and the elimination of time-consuming searches. The higher accuracy rate and positive user feedback indicate the app's effectiveness in meeting its objectives of providing accessible and reliable answers to requirements.

5. RELATED WORK

This following site explains the best way to maneuver the whole ranking up process for scouts [12]. While yes, it might explain the basics, it doesn’t really do the job that well. Scouts might not have the motivation to complete all these requirements either because flipping through the

scoutbook to find answers to requirements is too gruesome and boring. Or, scouts might not even have the motivation to work on those requirements. The app solves this issue by providing straightforward answers to most requirements. Some requirements cannot be solved with a simple and straightforward answer. This allows scouts to physically quickly advance in ranks instead of just being given tips on how to do so.

The following page provides a more in-depth explanation on the process of rapid ranking [13]. It tells the scouts where they should be after a certain period of time. For example, after the first 6 months in scouting, the scout should earn their First Class rank.

This source has a very similar problem as the last one. Yes, they did tell us the approximate process. But again, you have to make it easy for the scouts to actually rank up fast, and not just tell them what to do! This solution is like trying to tell a rock to move. It's not gonna. It won't move unless you physically provide resources to make it move. This is exactly what the app solves. It gives the scouts most of the answers so that they can actually rank up faster instead of just being told how to do it.

This video, like many other sources, provides an in-depth overview of how scouts should work to rank up faster [14]. But just like all the other sources out there, it lacks actual resources to physically allow scouts to accelerate their ranking.

Because there isn't really a huge market and competition within scouting apps and resources, there are very little to no things that provide the same level of support that this app gives out.

All the other sources I am able to find share the same issue - they only tell the scouts how they could speed up their ranking process, but give no actual resources to really allow them to rank up faster, unlike this app, which gives scouts the answers to most requirements. Most kids nowadays really would care less about these tips and advice. They just want the straightforward answer, and those videos and websites don't have those.

This app solves those issues. Instead of giving scouts only advice and tips, it gives them the answers, which would actually allow them to speed up their ranking.

6. CONCLUSIONS

There are only a few limitations to the design of the app - sometimes, the requirement asks things like "identify some animal trails in your area and report to an older scout." The app can't just have the answers to every single possible geographical location. There are infrequent cases where the scout must actually work on the requirement instead of given the answers, which might discourage some users.

In terms of fixing and improvements, the app will keep evolving and improving as it is a continuous project. Various aspects need to be fixed, such as bugs, improving the user interface and the artworks found in the main screen and in the future, the title screen. And in the future, support for merit badges might come, which means scouts can also be given answers to merit badge workbook questions. This will also make advancement easier because merit badge workbooks are very gruesome to complete, and potentially, have answers to every possible merit badge workbook (when applicable).

Overall, this app provides scouts with actual, physical resources in order for them to rank up faster instead of just relying on advice and tips which 90% of scouts probably will have no care

for. Although there are some limitations, in which scouts are on their own, the app has more than enough resources to at least massively boost their speed of ranking.

REFERENCES

[1] Macleod, David I. "Act your age: Boyhood, adolescence, and the rise of the Boy Scouts of America." Journal of Social History 16.2 (1982): 3-20.

[2] Hantover, Jeffrey P. "The Boy Scouts and the validation of masculinity." Journal of social issues 34.1 (1978): 184-195.

[3] Epstein, Richard A. "The Constitutional Perils of Moderation: The Case of the Boy Scouts." S. Cal. L. Rev. 74 (2000): 119.

[4] Goodly, Bernard. Leadership development within the Eagle Scouts: An investigation of the influence of servant leadership values. Diss. Capella University, 2008.

[5] Farag, Sara, Kathy Chyjek, and Katherine T. Chen. "Identification of iPhone and iPad applications for obstetrics and gynecology providers." Obstetrics & Gynecology 124.5 (2014): 941-945.

[6] Li, Xiaoye S. "An overview of SuperLU: Algorithms, implementation, and user interface." ACM Transactions on Mathematical Software (TOMS) 31.3 (2005): 302-325.

[7] Oualline, Steve, et al. "Using Google Sheets." Practical Free Alternatives to Commercial Software (2018): 389-404.

[8] Ciardo, Gianfranco, Gerald Lüttgen, and Andy Jinqing Yu. "Improving static variable orders via invariants." International Conference on Application and Theory of Petri Nets. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[9] Haas, John K. "A history of the unity game engine." Diss. Worcester Polytechnic Institute 483.2014 (2014): 484.

[10] Këpuska, Veton, and Gamal Bohouta. "Comparing speech recognition systems (Microsoft API, Google API and CMU Sphinx)." Int. J. Eng. Res. Appl 7.03 (2017): 20-24.

[11] Prana, Gede Artha Azriadi, et al. "Categorizing the content of github readme files." Empirical Software Engineering 24 (2019): 1296-1327.

[12] Bhandari, Inderpal, et al. "Advanced scout: Data mining and knowledge discovery in NBA data." Data Mining and Knowledge Discovery 1 (1997): 121-125.

[13] Goodly, Bernard. Leadership development within the Eagle Scouts: An investigation of the influence of servant leadership values. Diss. Capella University, 2008.

[14] Rohm, Frederick. "Eagle scouts and servant leadership." Servant Leadership: Theory & Practice 1.1 (2016): 6.

[15] Myers, Brad A. "User interface software tools." ACM Transactions on Computer-Human Interaction (TOCHI) 2.1 (1995): 64-103