UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ DE MACEDO GAETA

BLINK WELL: BLINK MONITORING SOFTWARE FOR DRY EYE TREATMENT

RIO DE JANEIRO
2024

ANDRÉ DE MACEDO GAETA

BLINK WELL: BLINK MONITORING SOFTWARE FOR DRY EYE TREATMENT

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Silvana Rossetto

RIO DE JANEIRO
2024

CIP - Catalogação na Publicação

```
         Gaeta, André de Macedo
G129b       Blink Well: Blink monitoring software for dry
         eye treatment / André de Macedo Gaeta. -- Rio de
         Janeiro, 2024.
            41 f.

            Orientador: Silvana Rossetto.
            Trabalho de conclusão de curso (graduação) -
         Universidade Federal do Rio de Janeiro, Instituto
         de Computação, Bacharel em Ciência da Computação,
         2024.

            1. Doença do olho seco. 2. Desenvolvimento de
         software. I. Rossetto, Silvana, orient. II. Título.
```

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

ANDRÉ DE MACEDO GAETA

BLINK WELL: BLINK MONITORING SOFTWARE FOR DRY EYE TREATMENT

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 14 de março de 2024

BANCA EXAMINADORA:

Dedicated to all who struggle with dry eyes.

RESUMO

A doença do olho seco é uma condição crônica cada vez mais relevante, que afeta muitas pessoas com sintomas como fadiga ocular, ardência e distúrbios na visão. Apesar de grandes esforços de pesquisadores, a doença continua forte e muitas pessoas não encontram alívio nas opções de tratamento existentes, o que tem suscitado a necessidade de intervenções inovadoras. Sua correlação com o tempo que usuários passam em telas digitais tem sido comprovada por diversos estudos, e a causa raiz do problema reside no fato que telas estimulam hábitos inadequados de piscar. Baixa frequência de piscadas e alta presença de piscadas incompletas são os principais agressores. Este trabalho tem como objetivo contribuir para esse campo em crescimento, documentando o desenvolvimento de um aplicativo assistencial especificamente projetado para auxiliar pessoas com olhos secos. O software proposto aplica métodos existentes de detecção de piscadas para monitorar a frequência de piscadas do usuário e seu tempo de tela. Essas estatísticas podem então ser utilizadas para promover hábitos de piscar mais saudáveis, lembrando o usuário de piscar e fazer pausas através de notificações. Com o aumento da frequência de piscadas e pausas regulares, o aplicativo busca aliviar os sintomas de olho seco e impedir a progressão da doença, representando assim uma alternativa em direção à melhoria do bem-estar de pessoas que sofrem com a doença do olho seco. O software foi desenvolvido usando Python como linguagem de programação e usufrui de bibliotecas como PyQt, para a interface gráfica, e MediaPipe, para detecção de pontos faciais. O código é aberto e está disponível para download via GitHub.

Palavras-chave: doença do olho seco; disfunção da glândula meibomiana; detecção de piscadas; MediaPipe; PyQt.

ABSTRACT

Dry eye disease is an increasingly relevant chronic condition that heavily soils many people's lives with eye fatigue, burning, and vision disturbances. Despite researcher's best efforts, the disease remains strong and many people don't find relief from the existing treatment options, which has sparked the need for innovative interventions. It's correlation with screen time has been substantiated by a plethora of studies, and the root cause of the problem lies in the associated stimulation of unhealthy blinking habits. Low blink rate and a high presence of incomplete blinks are the main aggressors. This work aims to contribute to this growing field by documenting the development of an assistive application specifically designed to assist individuals with dry eyes. The proposed software applies existing methods for blink detection in order to monitor the user's blink rate and screen time. These statistics can then be used to promote healthier blinking habits by reminding the user to blink and take breaks through notifications. By increasing blink frequency and incorporating regular breaks, the application seeks to alleviate dry eye symptoms and prevent the progression of the disease, thus representing an alternative towards improving the well-being of individuals suffering from dry eye disease. The software was developed using Python as a programming language and uses libraries such as PyQt, for the graphical interface, and MediaPipe, for facial landmark detection. It is open source and available for download on GitHub.

Keywords: dry eye disease; meibomian gland disfunction; blink detection; MediaPipe; PyQt.

LIST OF FIGURES

LIST OF CODES

LIST OF ABBREVIATIONS AND ACRONYMS

DED	Dry eye disease
MGD	Meibomian Gland Disfunction
SPK	Superficial Punctate Keratitis

CONTENTS

1 INTRODUCTION

Dry eye syndrome, also known as dry eye disease (or DED, for short), is a chronic condition whose both cause and effect are often governed by inflammation. Patients commonly report ocular discomfort and fatigue, redness, burning, pain and blurry vision. Due to its multifactorial nature and manifestation that is very similar to other ocular problems, DED can be difficult to properly diagnose and therefore treat.

Recent studies have shown the correlation between the occurrence of DED and the time spent staring at screens (AL-MOHTASEB et al., 2021; JENA; PALO; NANDA, 2022; KAUR et al., 2021). Excessive use of electronic devices can not only exacerbate the symptoms of dry eye, but also contribute to the manifestation of Meibomian Gland Dysfunction (CREMERS et al., 2021), a condition visible in approximately 86% of patients with dry eye (LEMP et al., 2012). This fact can be explained by the reduced blink rate and increased incomplete blink ratio associated with digital devices. When performing tasks that require greater focus, as is common when staring at screens, there is a high reduction in the blink rate of users, almost six times lower when compared to the average observed during a conversation, for example (BENTIVOGLIO et al., 1997).

Tasks that require quick and real-time responses, such as video games, can have an even greater impact and promote a drastic increase in the number of incomplete blinks (CARDONA et al., 2011). These are defined as blinks that do not result in complete eyelid closure, and do not cause the release of essential oils for a good quality tear, which aggravates eye dryness. There are studies that promote the detection of incomplete blinks (NOUSIAS et al., 2022; FOGELTON; BENESOVA, 2018), but it is a much more complex problem whose solution is still not seen as viable enough for use in general software. Therefore, the scope of this work is limited to the use of algorithms focused on simple blink detection, without completeness classification.

Fortunately, an individual's blink rate can be controlled and improved. Exercises designed to help with dry eyes have already been developed and have shown extremely satisfactory results (KIM et al., 2021). Similarly, the use of software designed to promote a healthy blink rate has been studied and has been concluded to be a viable option to help combat the disease (CHRISTIANTO; ADI, 2022). There are also similar proposals aimed at promoting eye health with a focus on adjacent problems, such as Computer Vision Syndrome (NEGREYROS; NEIRA; MURRAY, 2022).

This work's main objective is the creation of an assistive software that utilizes a blink detection solution to help the user achieve healthier blink habits while using the computer. Additionally, the software also tracks screen time in order to prompt the user to take frequent breaks.

Further explanations and additional information on DED will be explored in chapter 2. Related research and similar software implementations are mentioned in chapter 3. Following, chapter 4 documents the design of the blink detection algorithm itself. Then, in chapter 5 there's thorough detailing of the development of the software in question. Finally, chapter 6 provides an overview of what has been achieved and possible ways to expand this work in the future.

2 DRY EYE DISEASE

Dry eye disease, also known as *keratoconjunctivitis sicca*, is a common ocular disorder that affects millions of individuals worldwide. It is characterized by a deficiency in tear production or excessive evaporation of tears, resulting in an unstable tear film and ocular surface damage. DED is a multifactorial condition with a complex pathophysiology, making its diagnosis and management challenging. This chapter aims to provide an in-depth understanding of the disease and to expose it's link with bad blinking habits from exacerbated usage of digital devices. The following sections will detail the relevant points regarding the classification of different types of dry eye, the most common complications related to the disease, how screen time plays a role in the matter and how to mitigate it's effects.

2.1 DRY EYE CLASSIFICATION

Dry eye disease can be classified into two main types: evaporative dry eye and aqueous deficient dry eye. While both types share common symptoms of dryness, discomfort, and visual disturbances, they have different underlying causes and mechanisms. A novel method of classification could include a third type of dry eye whose problem is in the mucin layer, but this is currently not widely accepted as a common cause of dry eye (TSUBOTA et al., 2019).

Evaporative dry eye is the most common form of DED and is primarily caused by increased tear evaporation due to a deficient lipid layer in the tear film. Meibomian gland dysfunction (MGD) is a key contributor to evaporative dry eye. MGD can result from factors such as meibomian gland obstruction, altered meibum composition, and glandular inflammation. Prolonged screen use can exacerbate evaporative dry eye by reducing blinking frequency and incomplete blinking, leading to meibomian gland dysfunction and subsequent tear film instability.

Aqueous deficient dry eye occurs when there is a decreased production or availability of the watery component of tears, also known as the aqueous layer. This can be due to various causes, including autoimmune diseases (such as Sjögren's syndrome), lacrimal gland dysfunction, systemic diseases, hormonal changes, or certain medications.

Unlike evaporative dry eye, which can be influenced by screen use, aqueous deficient dry eye is less directly affected by screen time. This is because the underlying causes of aqueous deficient dry eye are often related to the aforementioned systemic conditions and autoimmune disorders. However, it is important to note that despite the primary cause being aqueous deficiency, a significant proportion of patients with aqueous deficient dry eye also have coexisting meibomian gland dysfunction (WANG et al., 2019). Therefore, even in cases of aqueous deficient dry eye, optimizing blink habits and implementing healthy visual habits during screen use can still be beneficial.

By promoting regular blinking and implementing visual breaks, individuals with both aqueous deficient and evaporative dry eye can help improve the ocular surface environment, reduce tear evaporation, and alleviate symptoms associated with dry eye.

2.2 COMMON COMPLICATIONS

Dry eye disease, if left untreated, can lead to long-term and potentially permanent complications and damage to the ocular surface. This subsection aims to elucidate the most common and dangerous complications that are associated with DED. It should then be more clear how the progression of the disease can and should be avoided, as well as the possible mitigation techniques that can be employed in a digital environment.

2.2.1 Meibomian gland dropout and atrophy

Meibomian glands play a vital role in producing and secreting the lipid component of tears, which helps to maintain tear film stability and prevent excessive tear evaporation. In DED, chronic inflammation, microbial growth, tear film instability, and other factors can lead to dysfunction of the meibomian glands. This dysfunction, if left untreated, can progress to meibomian gland dropout and atrophy (GEERLING et al., 2011).

When meibomian glands become dysfunctional, they may undergo structural changes and exhibit reduced secretory activity. Over time, the glands may degenerate, leading to glandular dropout, where the number of functional glands decreases. As a result, the quantity and quality of meibum produced are compromised, leading to further instability of the tear film and increased susceptibility to the aforementioned dry eye symptoms.

Meibomian gland dropout and atrophy can have severe consequences for ocular health. The loss of meibomian glands reduces the availability of the lipid layer in the tear film, which is crucial for preventing tear evaporation and maintaining the integrity of the ocular surface (BADIAN et al., 2021). Without an adequate lipid layer, tears evaporate more quickly, leading to increased tear osmolarity and therefore hyperosmolarity-induced ocular surface damage, which will be discussed in the next section.

The absence or reduced functionality of meibomian glands can result in ongoing tear film instability, reduced tear production, and increased tear evaporation. The ocular surface becomes more susceptible to friction and mechanical stress, leading to epithelial damage, corneal staining, and the formation of dry spots or erosions. The compromised ocular surface may also become vulnerable to infections and delayed wound healing.

Moreover, the chronic inflammation associated with DED can perpetuate the vicious cycle of meibomian gland dysfunction and ocular surface damage (BAUDOUIN et al., 2016). Inflammatory mediators released during the inflammatory response can further compromise the structure and function of the remaining meibomian glands, exacerbating the progression of DED.

2.2.2 Superficial punctate keratitis

Superficial punctate keratits (or SPK, for short) is a condition that affects the outermost layer of the cornea, called the epithelium. In SPK, small, punctate erosions or defects develop on the surface of the cornea. These erosions appear as tiny, irregularly shaped areas of disrupted cells, resembling spots or dots. SPK can occur in one or both eyes and is typically associated with DED, although other factors can contribute to its development as well. The underlying cause is believed to be a combination of inadequate tear film production, tear film instability, hyperosmolarity and ocular surface inflammation. Untreated SPK can lead to corneal scarring, particularly in cases of chronic or severe SPK, which can cause visual distortion and reduced visual acuity (ICHIHASHI et al., 2015).

2.3 ASSESSING SCREEN TIME

In recent years, the widespread use of digital devices such as computers, smartphones, and tablets has raised concerns about their potential impact on ocular health. Many individuals spend prolonged periods staring at screens, which can contribute to the development or exacerbation of DED, particularly in cases involving MGD, which is a significant underlying cause of DED (AL-MOHTASEB et al., 2021). The combination of reduced blinking rate, tear film instability, and increased visual demand contributes to the pathogenesis of MGD in individuals who frequently use screens.

When using digital devices, individuals tend to blink less frequently and incompletely. Blinking often is essential for spreading tears across the ocular surface and maintaining tear film stability. Insufficient blinking can lead to tear film instability and increased tear evaporation, contributing to dry eye symptoms. Prolonged screen use is also associated with incomplete blinking, leading to stagnation of meibum, the lipid component of tears produced by the meibomian glands (CARDONA et al., 2011). Proper blinking plays a fundamental role in stimulating the secretion of meibum. Each blink helps express meibum from the glands, preventing stagnation and obstruction. The accumulated meibum becomes thickened and may clog the glandular orifices, resulting in obstructive MGD. This obstruction can lead to glandular distention, atrophy, and altered glandular structure, impairing the normal secretion of meibum and compromising the stability of the tear film.

Additionally, not blinking enough while suffering from DED can exacerbate the ocular surface damage done to the cornea. Insufficient blinking, as often seen during prolonged screen use or concentrated visual tasks, can lead to tear film instability. Reduced blinking frequency or incomplete blinks can result in uneven tear distribution and increased tear evaporation, contributing to tear film disruption and ocular surface dryness (BADIAN et al., 2021). Tear film instability and resulting dryness can make the corneal epithelium more vulnerable to damage, potentially leading to the development of SPK. Also, the eyelids may not adequately cover the cornea during the interblink period. This exposes the cornea to environmental factors, such as air currents and particulate matter, which can exacerbate ocular surface damage and increase the risk of SPK.

2.4 MITIGATING THE PROBLEM

As explored in the previous section, it's clear that adhering to healthier blink habits, as well as taking frequent screen breaks, can be an invaluable tool in mitigating DED symptoms, as well as avoiding it's progression and potential complications. This is the main fact that drives the demand for an assistive software to help DED sufferers.

Another very helpful procedure that helps improve eye health is what is known as the 20/20/20 rule, which refers to the practice of takings regular breaks by adhering to the following terms: every 20 minutes looking at a screen, you should look at something at least 20 feet away (approximately 6 meters) for 20 seconds. This is a well known procedure and several studies prove it's effectiveness (TALENS-ESTARELLES et al., 2023). It helps reduce the eye strain and fatigue caused by staring at screens for long periods and prevents the eyes from becoming dry or irritated. Staring at objects at close distance for long periods of time can cause eye strain due to overworking the muscles responsible for close range focus, so taking breaks that force you to look far away helps those muscles relax to a more neutral position. By following this rule, the eyes get a chance to rest, blink and refocus, which can help prevent eye strain and other related problems, especially for people who spend extended periods looking at screens.

3 RELATED WORKS

Several studies and implementations have already been conducted and were of utmost importance to the confection of this work. This chapter strives to present the works that were most valuable and how they were synthesized and used as inspiration for the development of this work.

3.1 BLINK DETECTION RESEARCH

Several methods have already been proposed with the aim of creating a high-precision blink detection algorithm (FOGELTON; BENESOVA, 2016; NOUSIAS et al., 2022; SOUKUPOVA; CECH, 2016). Many rely on very complex machine learning models trained specifically for blink detection, which does indeed result in high accuracy blink detection. The method proposed by (SOUKUPOVA; CECH, 2016) is what has been implemented in this work, utilizing a facial landmark detection model, which is then analyzed by an eye aspect ratio analysis algorithm. Further information on the implementation of this method will be documented in section 4.1. Unlike the other methods, it puts the majority of the burden of detecting blinks on a general purpose facial landmark detection model, which means there's no need to use a large data set specific to blinking to train a complex machine learning model from scratch. Therefore, it's implementation is much simpler, at the expense of a very marginal decrease in accuracy.

3.2 SIMILARLY PURPOSED SOFTWARE

As seen in the previous section, there are a variety of published works related to blink detection. However, even though the technology is already mature enough for accessible solutions to exist for the general public, the study of techniques to make computer use healthy for people suffering from Dry Eye Syndrome has had a difficult time leaving the theoretical scientific field. This can be evidenced by the high quantity of related published articles, while there are very few instances of applications that use such algorithms to help patients maintain healthy blinking habits. Two relevant mentions are EyeBlink (Blinking Matters, 2016) and the macOS exclusive SightKick (SightKick.ai, 2022), which were made precisely for this purpose. However, they have certain limitations such as operational system incompatibilities, lack of customization options to better suit the user's preferences, blink reminders with slow and unresponsive behavior, and lack of clarity in the way the user receives notifications from the application. Additionally, they are proprietary pieces of closed source software, which can create insecurities to the public as to how your privacy is being managed, especially given the sensitive nature of data from webcam recordings. As such, there's a visible demand for the development of a more accessible piece of open source software that can create a better user experience and help solve the problem more effectively. However, these pieces of software proved themselves as extremely valuable tools in the development of this work, used extensively to detect points of potential improvement, and established a working proof of concept for the idea of managing dry eye through blink detection algorithms.

4 BLINK DETECTION

In order to monitor the users and assist them in maintaining healthy blink habits, an algorithm to detect blinks is required. The proposed method relies on a machine learning model to analyse frames from a camera's feed. This chapter details the methods employed to achieve such a feat while keeping it accessible to the average consumer.

4.1 METHODOLOGY

For accessibility reasons, it's extremely important that the proposed software does not require any external devices that are not already commonly used by the average person. Therefore, it's approach for blink detection is based around the use of a well positioned webcam, which is already included in most modern laptops and extremely accessible for desktop computers.

The blink detection algorithm aims to analyse real-time webcam data in order to detect whenever the user has performed a blink. This event is then propagated to the GUI layer of the application in order to calculate blink rate, along with other statistics, and finally interact with the user. This process will be properly explored in the next chapter.

Among the many researched methods to detect blinks, the approach of choice is to utilize a facial landmark detection model coupled with an eye aspect ratio analysis algorithm that was proposed in (SOUKUPOVA; CECH, 2016), as previously discussed in section 3.1, which has being shown to be a very viable method that delivers extremely good accuracy despite it's staggering simplicity. It was chosen over the other possible methods for having it's basis on facial landmark detection, which gives it implicit flexibility when considering extreme facial angles and inadequate camera placement, which is really important for real world software viability. Additionally, having the machine learning section focus solely on facial landmark detection also creates many implementation possibilities that rely on already trained high fidelity models provided by public libraries. This way, it's effectiveness can be further improved by pairing it with newer facial landmark detection frameworks. This work is therefore not concerned with the implementation of a brand new facial landmark detection solution and will rely on a third party library instead.

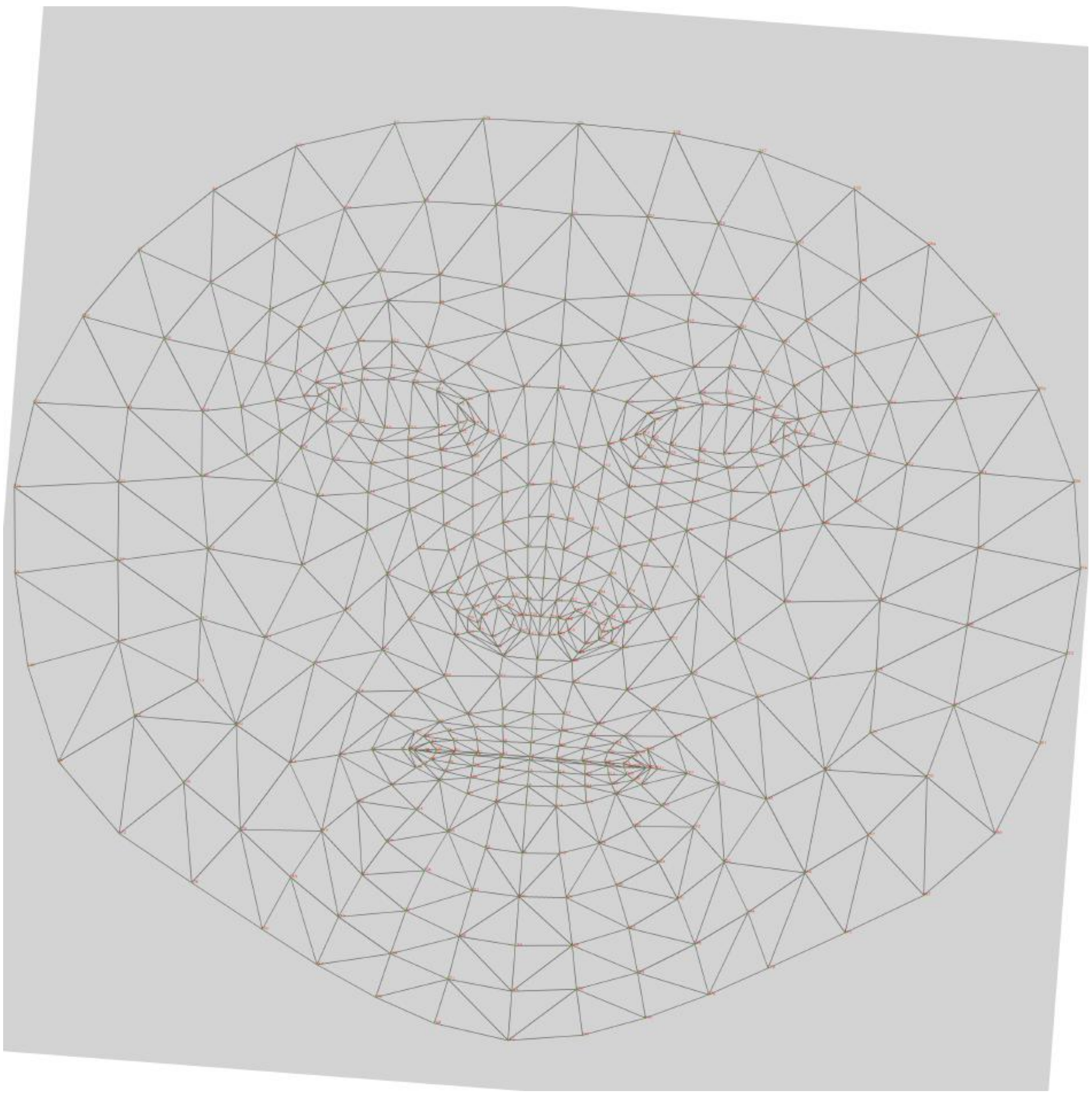
Among the frameworks for detecting facial landmarks, the one used in this work is the one provided by MediaPipe Solutions (Google, 2019), as it proved itself the most accurate and easy to manipulate. It features a suite of machine learning models for common computer vision tasks, with most of it's complexity abstracted away to keep it's usage accessible. Under the hood, it utilizes state-of-the-art technology to power high accuracy models, while benefiting from improved performance due to hardware acceleration. The

MediaPipe Face Mesh solution detects hundreds of facial landmarks in real-time, which can easily be used to extract eye information to drive the blink detection algorithm based on eye aspect ratio.

4.2 EXTRACTING EYE DATA

MediaPipe Face Mesh provides 468 face landmarks that maps the topology of one's face. After identifying the landmarks that represent the inner contour of the eye, it's simple to calculate it's aspect ratio. More specifically, the relevant landmarks are the one's that represent the extremities of the eye (top, bottom, left, right).

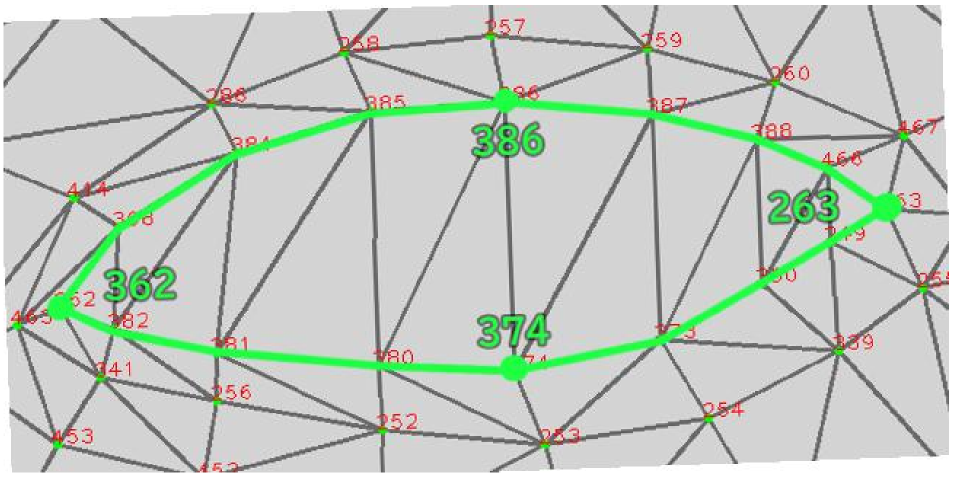
Figure 1 – Annotated Face Landmarks



Fonte: https://github.com/google/mediapipe

An image with annotated indexes for all face landmarks is provided in their GitHub repository (Figure 1), so the relevant indexes can be manually checked and hard-coded in the program (Figure 2). The framework itself provides the possibility of getting the position of a landmark by index.

Figure 2 – Eye Landmarks



Highlighted are the relevant landmark indexes of the left eye's extremities

4.3 EYE ASPECT RATIO

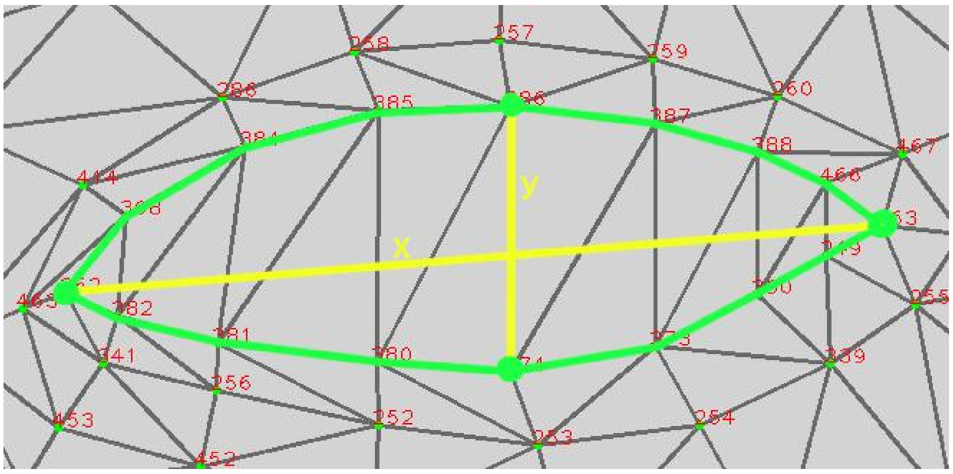
Having the positions of each extremity, the next step in the algorithm proposed by (SOUKUPOVA; CECH, 2016) is to calculate the Euclidian distance between the left and right points, as well as the top and bottom ones, for each eye, highlighted in Figure 3. Then, the eye aspect ratio (EAR) is calculated as the ratio between the vertical distance and horizontal distance (4.1). The overall aspect ratio that will be considered is the maximum between both eyes (4.2), as that provides the algorithm with more consistency throughout a variety of scenarios that one can expect to happen during computer use, such as briefly looking sideways or scratching their face. This way an accidental obstruction of one eye doesn't trick the algorithm into detecting a false positive blink.

Based on careful testing, a threshold is then determined to classify whether or not the eye is closed. If the aspect ratio is below that threshold, the algorithm will consider it to be closed. If it's above the threshold, it will be considered open. This value is exposed to the user in the software's settings to better support a varied facial topology spectrum. This will be further discussed in the following chapter.

$$EAR_{eye} = \frac{||p_{top} - p_{bottom}||}{||p_{left} - p_{right}||} \tag{4.1}$$

$$EAR = \max(EAR_{left}, EAR_{right}) \tag{4.2}$$

Figure 3 – Eye Aspect Ratio



x = horizontal length between extremities, y = vertical length between extremities

4.4 TEMPORAL CLASSIFICATION

After being able to determined if a given frame represents closed or open eyes, the next step is to devise a method to detect when a blink happens. Simply considering a closed eye frame as a blink frame isn't a sound approach, since a blink can consist of multiple frames with closed eyes. The edge case being a person who closes their eyes and keeps them closed; they're most definitely not blinking every frame after the eye closes. So it's adamant to only consider a single blink in a sequence of frames where a blink happens, which can have multiple frames with closed eyes. Additionally, basing the blink detection algorithm across a sequence of frames can help reduce the amount of false positives resulted from noise in the EAR thresholding, such as wrongly classified single isolated frames.

A support vector machine (SVM, for short) approach, as proposed in (SOUKUPOVA; CECH, 2016) is a very elegant solution to the problem, resulting in highly accurate classification. However, it comes with additional overhead as it introduces the need to process another classification pass, since it works as a separate machine learning model that will need to evaluate every frame. Since the proposed software aims to run in the background while the user performs other tasks in their computer, it is extremely important to reduce the unnecessary performance costs, even more so for code that is meant to run every frame. Additionally, this methodology requires the processing of a high quantity of frames in order to detect blinks, which results in a delay of a couple of frames between the moment the user blinks and the moment the algorithm detects it. It is of utmost importance for this to be minimized in order to provide a responsive user experience, improving the blink reminder response time in comparison to the aforementioned software (Blinking Matters, 2016). Therefore, the proposed solution diverges from the original article in this aspect by not utilizing said SVM approach, resulting in a slightly less accurate, but significantly simpler and faster algorithm.

In the method devised for this work, the temporal aspect of blink detection is engendered by determining two thresholds concerning the amount of consecutive frames with eyes opened and closed to detect the blink start and blink finish moments, respectively. A code snippet that counts these consecutive frames and compares them with the aforementioned thresholds is found in Code 1. This way, the exact frame of a blink is easily detected and the fact that additional frames with closed eyes don't incur new blinks is accounted for. After empirical testing, it was determined that accumulating a total of 2 frames for both threshold values produced the best results. This way, blinks are detected only a single frame after the first frame in which the eye is considered closed is detected, resulting in only a very short delay between the user's ground truth blink instant and the one detected by the algorithm.

Code 1 – Temporal Blink Detection

```
if eye_is_closed:
    frames_with_eyes_open = 0
    frames_with_eyes_closed += 1
    if frames_with_eyes_closed == BLINK_FRAME_THRESHOLD and not
       is_blinking:
        is_blinking = True
        # Blink Start Detected
else:
    frames_with_eyes_closed = 0
    frames_with_eyes_open += 1
    if frames_with_eyes_open == UNBLINK_FRAME_THRESHOLD and is_blinking:
        is_blinking = False
        # Blink Finish Detected
```

5 SOFTWARE DEVELOPMENT

Having the blink detection algorithm properly implemented, the next step is to wrap it in a piece of software that will process those blinks to drive the interaction with the user. This chapter aims to document the features and scope of such a software, detailing the technologies chosen to assist it's development, the overall architecture and philosophies of the code base, all the features present in the application, as well as the testing process and the ways in which the project was packaged and published. The source code is openly available in (André Gaeta, 2024).

5.1 LANGUAGE AND FRAMEWORK

Due to the expensive calculations performed every frame in the blink detection algorithm, expanding the program to support portable devices, such as phones and tablets, would be a challenging task for a real world product due to their lower processing power, battery consumption, and heating problems. However, it remains solidly within budget for the majority of modern personal computers. Therefore, the application limits itself to the realm of desktop development. Unlike chapter 4, this implementation will be designed from scratch, without relying on other articles and without outright outsourcing part of the work to a third party library. However, third party tools will be used in order to assist it's creation and guarantee a user interface with high quality standards.

Since the blink detection algorithm is written exclusively in Python, it's a reasonable decision to use that very same language to build the rest of the application. While it would be possible to build the GUI in another language, it wouldn't be worth the effort when factoring in the associated additional development overhead. By keeping everything written Python, it's easier to connect the different pieces of the application that will need to talk, and it greatly simplifies the packaging process. The framework of choice for handling the GUI is PyQt (Riverbank Computing, 1998), which is a Python binding for the very popular C++ framework Qt (Qt Group, 1995). While PyQt development isn't as simple as other frameworks, it delivers the highest amount of customization for creating a polished user interface, while featuring a suite of tools to facilitate common GUI tasks.

Applications based on Qt also have the benefit of having access to QtDesigner (Qt Group, 1996), which is a popular and powerful tool for designing user interfaces. It provides numerous benefits for programmers who want to create intuitive and user-friendly UIs, the main draw being that it allows developers to create UIs visually, without having to write the visual aspects of the application in code. This can save a significant amount of time and effort compared to designing UIs manually. Additionally, it generates code that can be easily integrated into a PyQt program, saving development time. Furthermore, Qt Designer supports drag-and-drop functionality, which makes it easy to add buttons, labels, text boxes, and other widgets to the UI. This way, programmers can quickly create and test different UI designs until they find the one that suits their software best, in a visual and intuitive way. It's a great choice to streamline the UI design process and result in a more efficient and visually appealing user interface, and better separates the functional aspects of the program from the graphical design.

As will be seen in section 5.3, the program interacts with the user by using several notification popups and overlays. PyQt also helps creating windows that function extremely well for those purposes, as they can contain transparency, always be displayed on top and, most importantly, have click-through behavior. This allows the user to continue interacting with his foreground applications without having to worry they will click the application's GUI and end up minimizing or switching the operation system's focus to something else. This is a big problem observed in one of the aforementioned applications that provide similar blink detection functionality (Blinking Matters, 2016). Additionally, the application itself can be easily run in the background in a low key way, by limiting itself to the operational system's tray and only opening it's main window when the user desires.

5.2 EVENT-DRIVEN PROGRAMMING PARADIGM

PyQt employs an event based programming architecture, where event handlers are triggered in response to user actions or other events. This way, the programmer isn't meant to write code that explicitly controls the program's flow. When processing a heavy computation, for example, it can block the loop and cause the application to be unresponsive. Instead, the framework provides an easy to use abstraction to run code in a separate thread. This ensures that the event loop remains responsive and stable to user input while the task is running in the background.

The framework contains it's own internal event loop to keep the GUI alive indefinitely. This isn't accessible to the programmer, however. This is of utmost importance, since the blink detection algorithm performs a loop of it's own. Since we can't embed it inside the GUI core loop, we instead define a new thread to house the algorithm, which also helps keep the GUI responsive in the process, since the calculations can be heavy.

In PyQt's event-driven architecture, signals and slots are used to implement communication between objects and to trigger actions in response to events. A signal is emitted by an object when a particular event occurs, and a slot is a function or method that is called in response to that signal. The framework ensures the slots are being fired without collision. This way, signal and slot mechanism is particularly useful in multithreaded applications, where multiple threads need to communicate and synchronize with each other in a safe and efficient manner. It ensures that events can be propagated across threads in a completely thread-safe manner to avoid data corruption, race conditions, and other concurrency-related issues. All this is achieved without the programmer having to use locking mechanisms such as mutexes or semaphores.

It is worth noting that this approach is widely used in modern GUI design. By leveraging the event-driven programming paradigm, programmers can develop GUI applications that are responsive and intuitive to user input. The use of signals and slots in PyQt provides a powerful mechanism for communication between different parts of the application, allowing for complex interactions and workflows. Therefore, when implementing the blink detection algorithm using PyQt, the use of threads and signals and slots is a highly beneficial choice, as it allows for a more efficient and robust solution.

Additionally, events play a crucial role in software development and are highly important to maintain good coding practices. By using an event-driven approach, developers can write more modular, reusable, and flexible code. This approach, highly analogous to the observer pattern, promotes the separation of concerns, enabling different components to interact and communicate with each other without tightly coupling them. This then leads to better code organization, easier maintenance, and increased scalability.

Adhering to this system, the blink detection code is executed in a separate thread, and propagates thread-safe events that will analyzed by other systems to calculate statistics like blink rate and inter-blink interval. These will ultimately serve as the driving data to notify the user in the GUI thread. All while maintaining a clean and scalable software architecture.

5.3 FEATURES

The final software includes several features, each developed with a specific goal in mind. These will be discussed in detail in their respective subsections, including a general understanding of the methods utilized and their inner workings.

5.3.1 Blink rate calculation

The core of the application revolves around tracking the user's blink rate. This is achieved by maintaining a list of blink entries that consist of their precise manifestation time, which is updated every frame to remove any entries older than however long the blink rate is meant to consider. The implementation can be seen in Code 2, where the oldest entry's elapsed time is compared against the predetermined maximum duration and is removed in case it has expired. The blink rate is then recalculated using the amount of remaining blink entries in the list.

Code 2 – Blink Rate Calculation

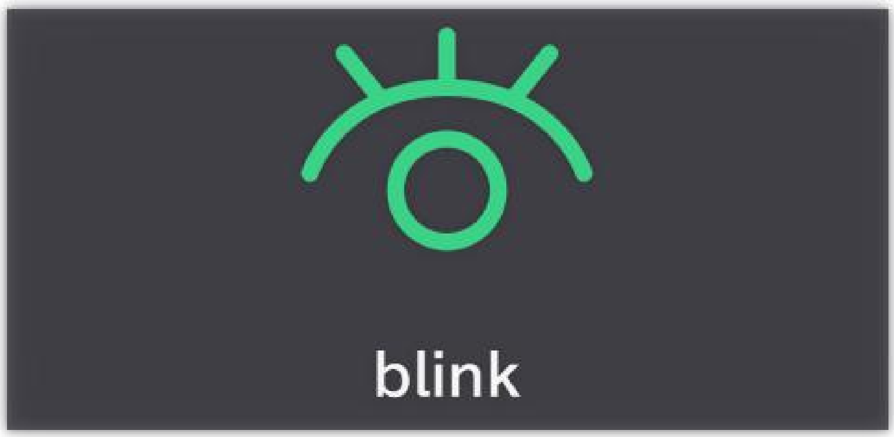
```
# Called when a blink signal is emitted
def on_blink_start(self):
    self.blink_entries_recent.append(time.time())

# Called every frame
def update_blink_rate(self):
    len_blinks_recent = len(self.blink_entries_recent)
    elapsed_time = min(time.time() - self.start_time,
       RECENT_BLINKS_DURATION)

    if len_blinks_recent > 0 and self.blink_entries_recent[0] < time.
       time() - RECENT_BLINKS_DURATION:
        self.blink_entries_recent.pop(0)

    blink_rate_recent = len_blinks_recent * 60 / elapsed_time

    self.app.update_blink_rate_signal.emit(blink_rate_recent)
```

5.3.2 Blink reminder

After having a way to calculate the user's blink rate and the duration since the user's last blink, the application uses those statistics in order to drive a notification system that will remind the user to blink. In order to keep the software customizable and accessible, it is important to provide a "target blink rate"setting to the users, allowing them to control how high or low they want their blink rate to be. Calculating the best moment to notify the user is a very important task, but there's no easy answer to what would be the correct approach. It is absolutely key to consider the blink rate the user has been maintaining recently in addition to the user's target blink rate.

A small algorithm has been devised in order to achieve a very flexible system that takes the user's current blink rate into account, which can be seen in Code 3. The method works by calculating the ideal interval between blinks by projecting the current blink rate into the future, and calculating the blink rate the user would have to maintain in order to achieve it's target blink rate. A variable concerning the rigidness of the system is introduced, controlling how much the blink rate will be projected into the future. More rigidness will result in lower weight for the user's current blink rate, while less rigidness will allow it to more heavily influence the calculated value.

Figure 4 – Blink Reminder Popup



Code 3 – Blink Notification Delay Calculation

```
def calculate_notification_delay(self):
    target_blink_rate = self.thread.get_setting("target_blink_rate")
    blink_reminder_rigidness = self.thread.get_setting("
       blink_reminder_rigidness") / 50

    recent_blink_count = len(self.blink_entries_recent)
    elapsed_time = self.get_elapsed_time()
    projected_blink_count = target_blink_rate / 60 * (1 +
       blink_reminder_rigidness) * RECENT_BLINKS_DURATION
    remaining_blinks = projected_blink_count - recent_blink_count
    remaining_time = (1 + blink_reminder_rigidness) *
       RECENT_BLINKS_DURATION - elapsed_time

    target_rate = remaining_blinks / remaining_time
    target_rate = max(target_rate, target_blink_rate / 60 / 2, 10 / 60)
    target_rate = min(target_rate, target_blink_rate / 60 * 2, 1)
    return 1 / target_rate
```

The way the application notifies the user to blink is via a simple popup that fades away as soon as a blink is detected (Figure 4). A simple blinking animation is displayed in order to further draw the user's attention and to add polish. This popup is designed to be big enough that it won't be missed, but small enough that it won't feel aggressive and disruptive. It's meant to be displayed in the center of the screen, but as will be seen in subsection 5.3.7, this can be changed to fit the user's preference.

5.3.3 Tracking screen time

As seen in section 2.4, implementing the 20/20/20 rule as an additional feature is an easy and effective way to further reduce the strain on the user's eyes and maintain good eye health, especially for people who spend extended periods looking at screens.

Timer applications are very simple to make and plentiful on the market, so this may seem like an unnecessary inclusion. However, they require manual maintenance from the user and often turn out to be a huge nuisance. They can also be very difficult make effective use of, since manual setup means it's prone to human error. Several times a person will be interrupted while using the computer, making random pauses for a variety of reasons. There's a need for the user to remember to stop, resume, and reset the timer when appropriate.

By embedding this functionality into an application that already uses the user's webcam, it's easy to improve it into a smart timer that will take care of all these problems. By detecting and tracking when a person is away from the computer and when they return, such a timer can automatically handle all the previously mentioned actions that are commonly handled by manual maintenance. This comes at essentially zero cost, as the software already relies on face detection for blink detection. The next subsections will further explain how tracking screen time will be utilized in the application.

5.3.4 Break reminder

In order to adhere to the aforementioned 20/20/20 rule, a break reminder is showcased as a popup in order to notify the user to take a break (Figure 5). The default value is indeed 20 minutes, but this is also configurable in the user's settings. The timer calculation itself is based on very simple and generic timer behaviour and therefore doesn't warrant particular discussion pertaining it's implementation.

The reminder stays on screen, displaying the user's elapsed screen time, until the user selects one of the possible courses of action: start, snooze or skip. Start initiates the break time, which is covered in the Break Time Subsection. Snooze will hide the window a notify the user again after a configurable amount of time has elapsed. Skip will ignore this reminder and will only notify the user again after the interval between breaks has passed again.

5.3.5 Break time

When the user accepts taking a break, a break time popup appears, as shown in Figure 6. The user may also at any time take the opportunity to leave the computer to take a voluntary break. This will be automatically detected by the program and trigger a break time, by taking into account the amount of time that has passed since a face was last detected by the facial landmark detection framework. The popup shows some

Figure 5 – Break Reminder Popup

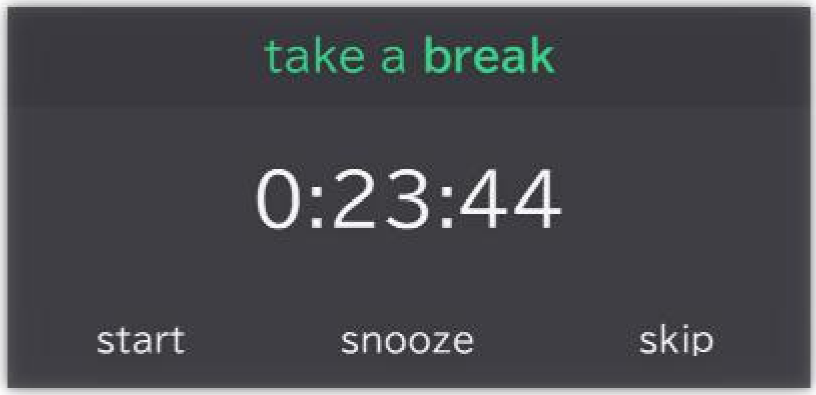
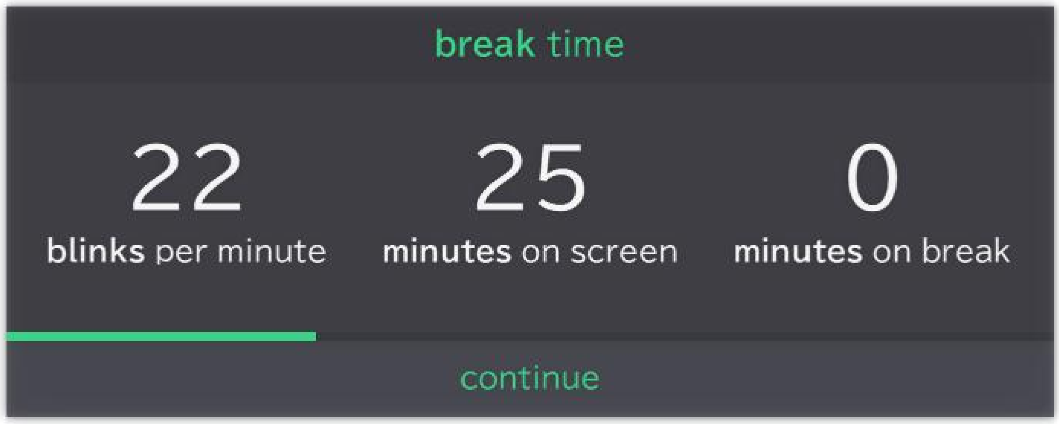


Figure 6 – Break Time Window

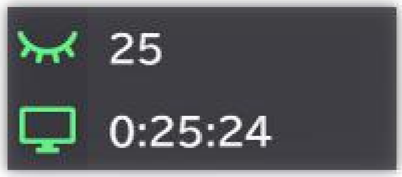


statistics from the last session, such as the overall blink rate and session duration. It also displays how long the user stays on break, which can be a useful information when the user returns. A progress bar with fill as time passes, allowing the user to unlock the screen and resume his computer use as soon as the break time has elapsed. The default value is 20 seconds to follow the 20/20/20 rule, but can also be extended in the user settings in order to force the user into longer break periods.

5.3.6 Stats overlay

It's important to provide the gathered information to the user whenever he demands, so he can easily keep track of his blink rate and potentially plan his timing for an upcoming break. A very elegant way to display to accomplish that is using an overlay window, which is always succinctly displayed to the user in the corner of the screen. It's visuals can be

Figure 7 – Stats Overlay



seen in Figure 7.

5.3.7 Settings

As often mentioned throughout this work, there are multiple variables that the user can tweak to his preference. These are all aggregated in the settings tab of the main window, providing easy access to all possible customizations, shown in Figure 8. The user can also disable particular features he may not want. Additionally, there's a button to allow the user to customize the position of every window by moving it around.

PyQt has a very helpful feature to save and retrieve user preferences, such as window size, position, and other settings. This functionality is achieved by using the QSettings class, which provides a platform-independent way to store and retrieve application settings. It works by creating a persistent store on the user's computer, where key-value pairs are stored in the system registry. These values can be accessed and updated by the application whenever necessary. This was used to great effect to easily allows users to customize the application to their preferences, while leaving the associated code simple and clean.

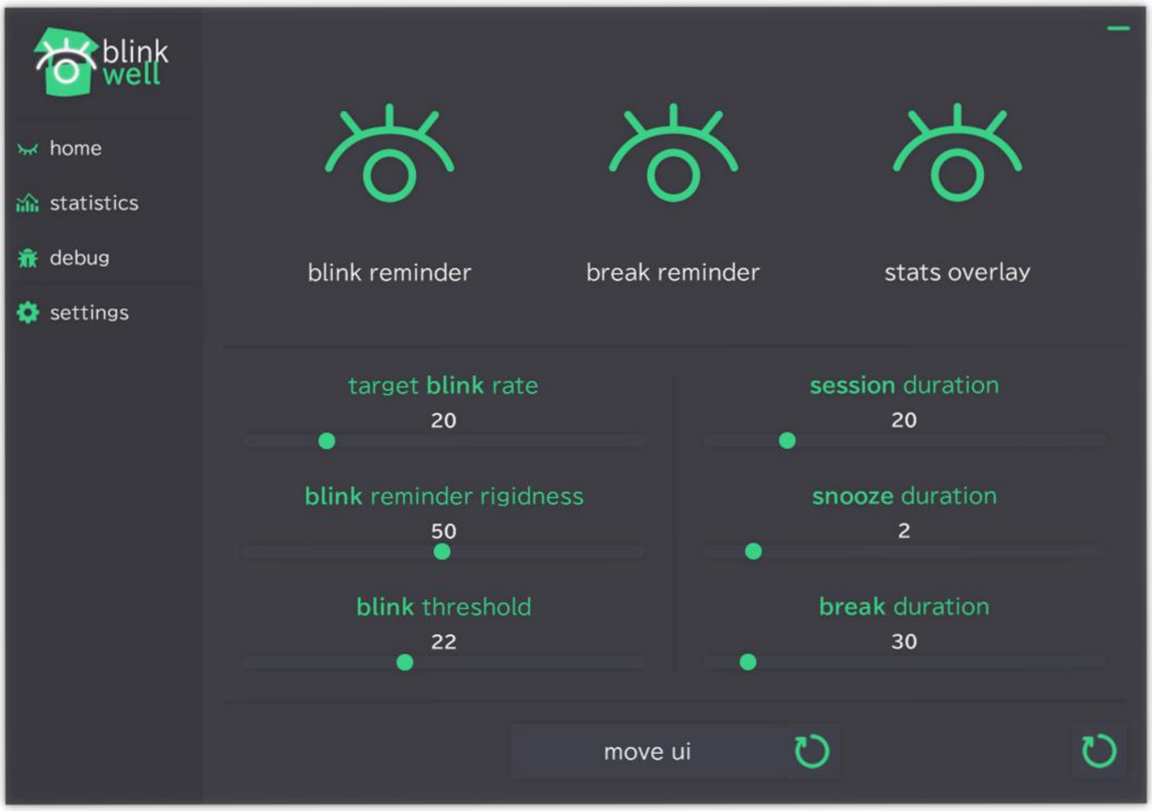
5.3.8 Sound effects

In order to better draw the user's attention, a random sound effect from a predefined pool is played when the user is prompted to take a break. Similarly, there's sound when the break time window unlocks after the minimum break duration has elapsed, making it easy for the user to be notified in case they have their eyes closed or staring out the window, for example.

5.4 TESTING

After the core of the application was finished, it was very important to assure it's functionality worked as intended. Since all features have their roots on the user's blink frequency, the key point to ascertain it's efficacy was to roughly measure it's accuracy when compared to ground truth numbers. As such, the testees were instructed to blink according to the beat of a metronome. Tweaking the tempo of the metronome allows the test to be performed under multiple different blink rates. This way, the blink frequency

Figure 8 – Settings Window



target blink rate 20
blink reminder rigidness 50
blink threshold 22
session duration 20
snooze duration 2
break duration 30

reported by the software can be directly compared against the beat frequency of the metronome, providing a reasonable confirmation of it's accuracy. Tests were made at 10, 15, 20, 25, and 30 beats per minute, values chosen for being reasonable expectations when viewed as blinks per minute. They were performed for one minute, for convenience, and repeated 3 times each. The results confirmed the algorithms efficacy, with the reported blink rates being within an insignificant margin of error of the expected value.

However, given the laborious nature of the test, it was only performed by two participants, using different machines and webcams. In order to further confirm the matter, a survey was openly shared with prospective Blink Well users. Asking simple qualitative questions to measure their level of satisfaction, the scope of the survey was to gather a slightly higher amount of anonymous data that could be answered quickly and without technical details by anyone who chose to contribute. Additionally, the feedback provided in the survey helps understand the potential points of improvement for future versions of the software. A total of 4 participants submitted their feedback, and their accounts were invaluable in order to access the software's public reception. The accounts were very positive, and are exposed at length in Appendix A, along with all the questions that the participants were asked.

5.5 PACKAGING

PyInstaller is a well-known Python library that enables developers to bundle Python programs into self-contained executable files that can operate on computers that lack a Python installation. This feature can be advantageous when distributing Python applications to users who lack familiarity with Python or lack the necessary Python libraries. It allows the average user to launch the application like they would any other professional published software, making it accessible to the public without programming knowledge. It is also an excellent choice for developers who want to publish accessible software due to its compatibility with multiple operating systems. It allows developers to distribute their Python programs across various operating systems, including Windows, macOS, and Linux.

6 CONCLUSION

This work aimed to explore the relationship between screen time and DED and propose a potential solution through the development of a desktop application. The software offers a proactive approach to managing DED by monitoring the user with the usage of a webcam. It offers a promising solution to mitigate the effects of screen time on ocular health. By monitoring users' screen time and implementing reminders for healthy blinking habits, the application aims to enforce regular blinking, which helps in maintaining tear film stability and reducing dry eye symptoms, as well as promoting frequent breaks in order to mitigate eye strain and fatigue.

The desktop application presented in this work demonstrates the potential for technology to play a significant role in managing DED and promoting healthy blinking habits. Ultimately, this research contributes to the ongoing efforts to address the rising prevalence of DED and improve the quality of life for affected individuals.

Certain features could be implemented in the future to further expand on the software's capabilities, such as generating user statistics over long periods of time, for instance. This data could be used by individuals to drive new conclusions about their condition, such as identifying possible triggers and seemingly unrelated habits that might impact their condition. User feedback and input will also be invaluable in identifying areas for improvement and tailoring the application to meet individual needs, as well as adding any additional functionality that might be beneficial to the users.

In regards to it's impact, further research and development are necessary to refine and properly access the software's efficacy. Long-term studies should be conducted to evaluate it's effectiveness in reducing DED symptoms and improving ocular comfort.

REFERENCES

AL-MOHTASEB, Z. et al. The relationship between dry eye disease and digital screen use. **Clinical Ophthalmology**, v. 15, n. 0, p. 3811–3820, 2021.

André Gaeta. **Blink Well**. 2024. https://github.com/andregaeta/BlinkWell.

BADIAN, R. et al. Meibomian gland dysfunction is highly prevalent among first-time visitors at a norwegian dry eye specialist clinic. **Scientific Reports**, v. 11, 12 2021.

BAUDOUIN, C. et al. Revisiting the vicious circle of dry eye disease: a focus on the pathophysiology of meibomian gland dysfunction. **British Journal of Ophthalmology**, BMJ Publishing Group Ltd, v. 100, n. 3, p. 300–306, 2016. ISSN 0007-1161. Disponível em: https://bjo.bmj.com/content/100/3/300.

BENTIVOGLIO, A. R. et al. Analysis of blink rate patterns in normal subjects. **Movement disorders: official journal of the Movement Disorder Society**, v. 12, n. 6, p. 1028–1034, 1997.

Blinking Matters. **Eye Blink**. 2016. https://www.blinkingmatters.com/. Accessed: 2023-20-07.

CARDONA, G. et al. Blink rate, blink amplitude, and tear film integrity during dynamic visual display terminal tasks. **Taylor & Francis**, v. 36, n. 3, p. 190–197, 2011.

CHRISTIANTO, F.; ADI, N. P. The effectiveness of blinking therapy in dry eye disease: An evidence-based case report. **Indonesian Journal of Community and Occupational Medicine**, v. 2, n. 1, p. 66–72, 2022.

CREMERS, S. L. et al. New indicator of children's excessive electronic screen use and factors in meibomian gland atrophy. **American Journal of Ophthalmology**, v. 229, n. 0, p. 63–70, 2021.

FOGELTON, A.; BENESOVA, W. Eye blink detection based on motion vectors analysis. **Computer Vision and Image Understanding**, v. 148, n. 0, p. 23–33, 2016.

FOGELTON, A.; BENESOVA, W. Eye blink completeness detection. **Computer Vision and Image Understanding**, v. 176–177, n. 0, p. 78–85, 2018.

GEERLING, G. et al. The international workshop on meibomian gland dysfunction: Report of the subcommittee on management and treatment of meibomian gland dysfunction. **Investigative ophthalmology visual science**, v. 52, p. 2050–64, 03 2011.

Google. **MediaPipe**. 2019. https://developers.google.com/mediapipe. Accessed: 2023-20-07.

ICHIHASHI, Y. et al. Short break-up time type dry eye has potential ocular surface abnormalities. **Taiwan Journal of Ophthalmology**, v. 5, 05 2015.

JENA, S.; PALO, G. D.; NANDA, P. K. An observational study on association of various risk factors with meibomian gland dysfunction. **IOSR Journal of Dental and Medical Sciences**, v. 21, n. 8, p. 52–55, 2022.

KAUR, P. et al. Association of risk factors with severity of meibomian gland dysfunction. **Ophthalmology Journal**, v. 6, n. 0, p. 76–82, 2021.

KIM, A. et al. Therapeutic benefits of blinking exercises in dry eye disease. **Contact Lens and Anterior Eye**, v. 44, n. 3, p. 101329, 2021.

LEMP, M. et al. Distribution of aqueous-deficient and evaporative dry eye in a clinic-based patient cohort. **Cornea: The Journal of Cornea and External Disease**, v. 31, n. 5, p. 472–478, 2012.

NEGREYROS, H.; NEIRA, M.; MURRAY, V. Real-time eye blinking detection for reducing the effects caused by computer vision syndrome. **IEEE Andescon**, p. 1–6, 2022.

NOUSIAS, G. et al. Eye blink completeness detection. **IEEE Journal of Biomedical and Health Informatics**, v. 26, n. 7, p. 3284–3293, 2022.

Qt Group. **Qt**. 1995. https://www.qt.io/. Accessed: 2023-20-07.

Qt Group. **Qt Designer**. 1996. https://doc.qt.io/qt-6/qtdesigner-manual.html. Accessed: 2023-20-07.

Riverbank Computing. **PyQt**. 1998. https://riverbankcomputing.com/software/pyqt/intro. Accessed: 2023-20-07.

SightKick.ai. **SightKick**. 2022. https://www.sightkick.ai/. Accessed: 2023-20-07.

SOUKUPOVA, T.; CECH, J. Real-time eye blink detection using facial landmarks. **21st Computer Vision Winter Workshop**, Eslovênia, 2016.

TALENS-ESTARELLES, C. et al. The effects of breaks on digital eye strain, dry eye and binocular vision: Testing the 20-20-20 rule. **Contact Lens and Anterior Eye**, v. 46, n. 2, p. 101744, 2023.

TSUBOTA, K. et al. A new perspective on dry eye classification: Proposal by the asia dry eye society. **Eye Contact Lens: Science Clinical Practice**, v. 46 Suppl 1, p. 1, 08 2019.

WANG, Y. et al. Clinical analysis: Aqueous-deficient and meibomian gland dysfunction in patients with primary sjogren's syndrome. **Frontiers in Medicine**, v. 6, p. 291, 12 2019.

APPENDIX A – BLINK WELL USER FEEDBACK SURVEY.

This appendix illustrates all the 10 questions that were asked in the user feedback survey, as well as the answers given by the 4 contributors. Questions generally range from 1 (labeled as "mediocre") to 5 (labeled as "great").

Figure 9 – Survey Question 1

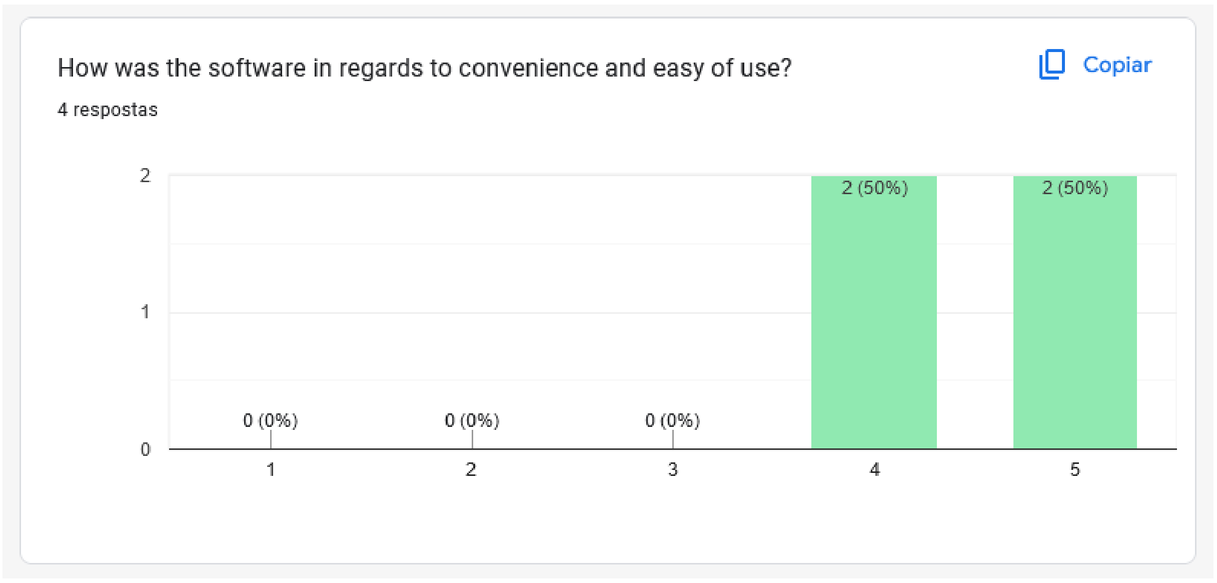


Figure 10 – Survey Question 2

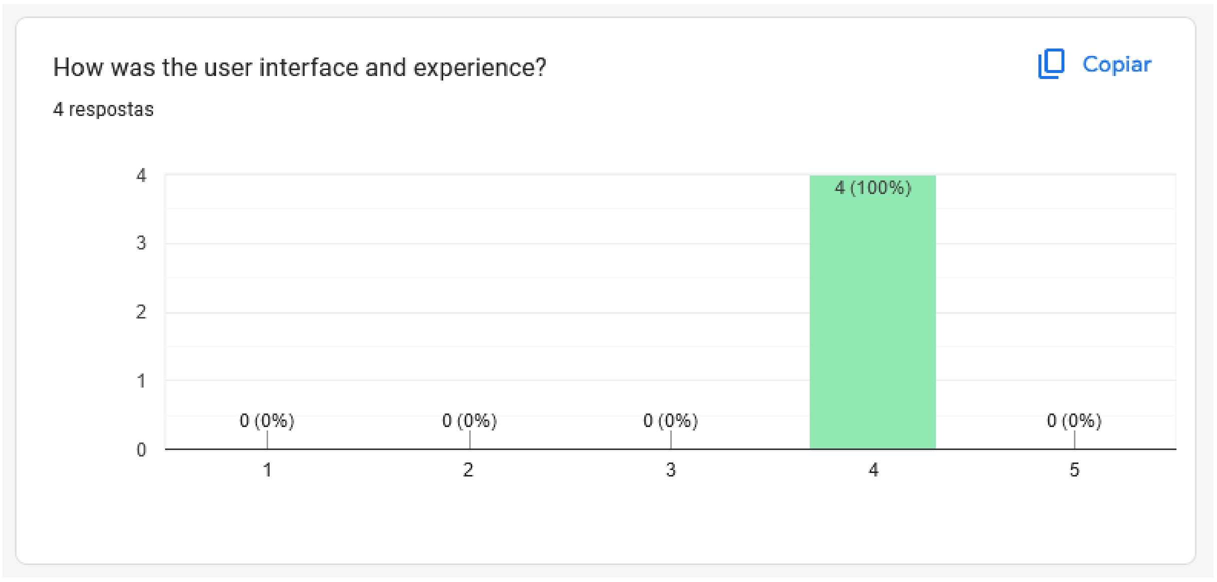
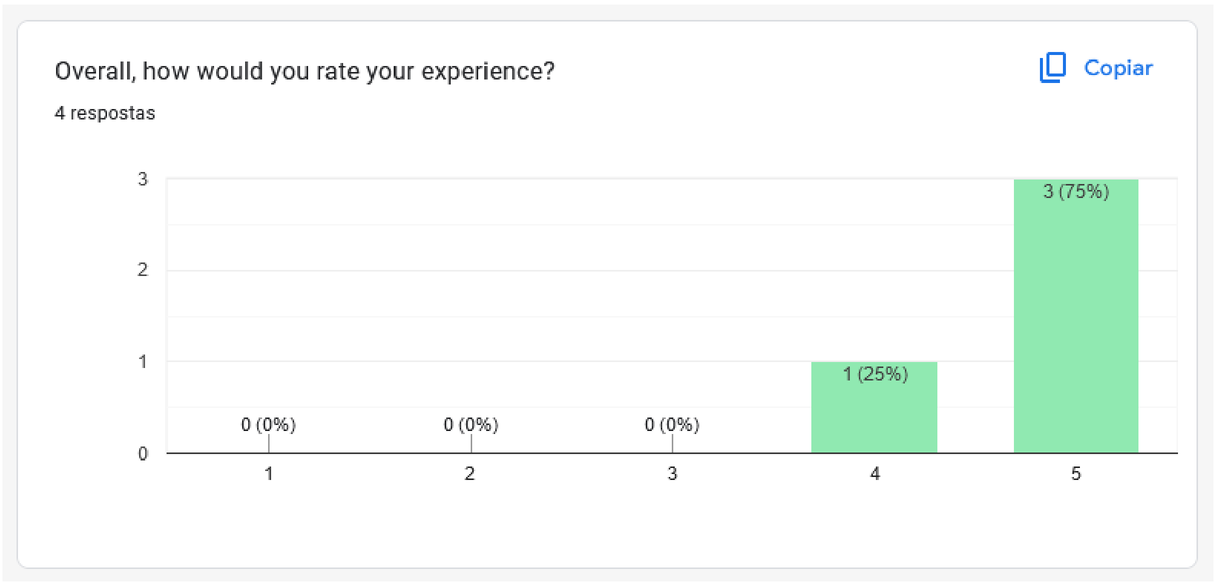
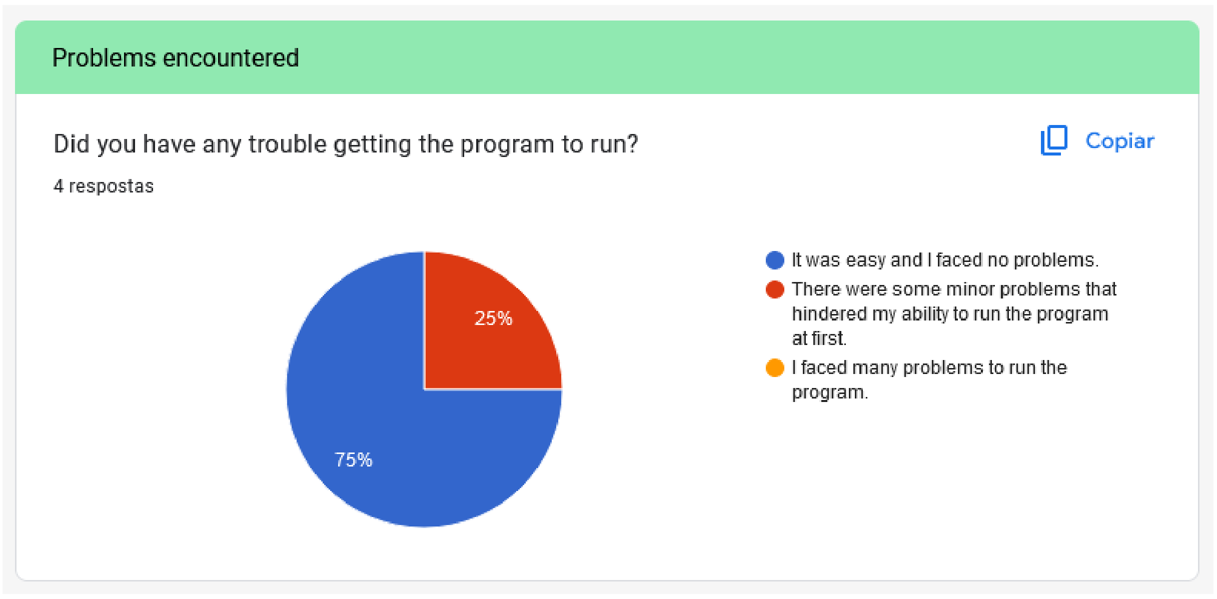


Figure 11 – Survey Question 3


Overall, how would you rate your experience?

4 respostas

Figure 12 – Survey Question 4

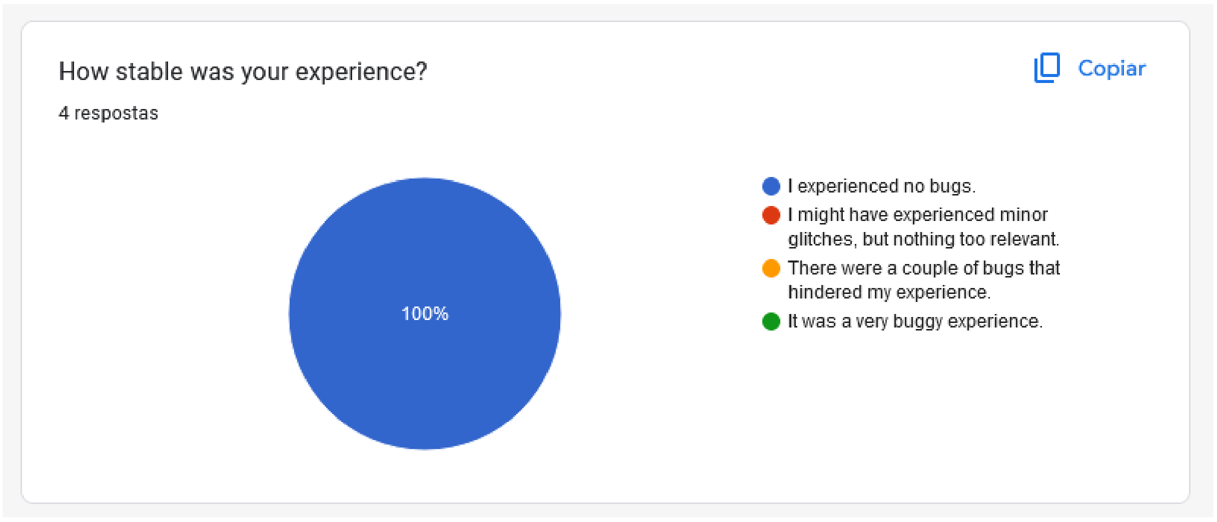

Problems encountered

Did you have any trouble getting the program to run?

4 respostas

Copiar
25%
75%
● It was easy and I faced no problems.
● There were some minor problems that hindered my ability to run the program at first.
● I faced many problems to run the program.

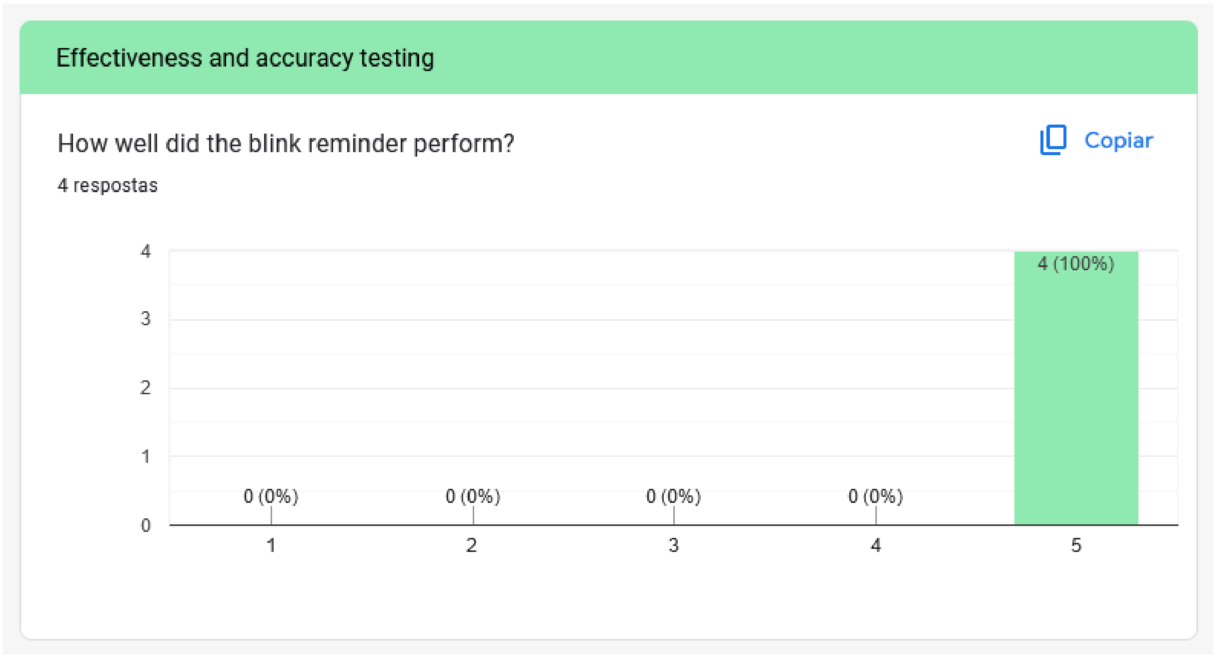
Figure 13 – Survey Question 5


How stable was your experience?

4 respostas

Copiar
100%
● I experienced no bugs.
● I might have experienced minor glitches, but nothing too relevant.
● There were a couple of bugs that hindered my experience.
● It was a very buggy experience.

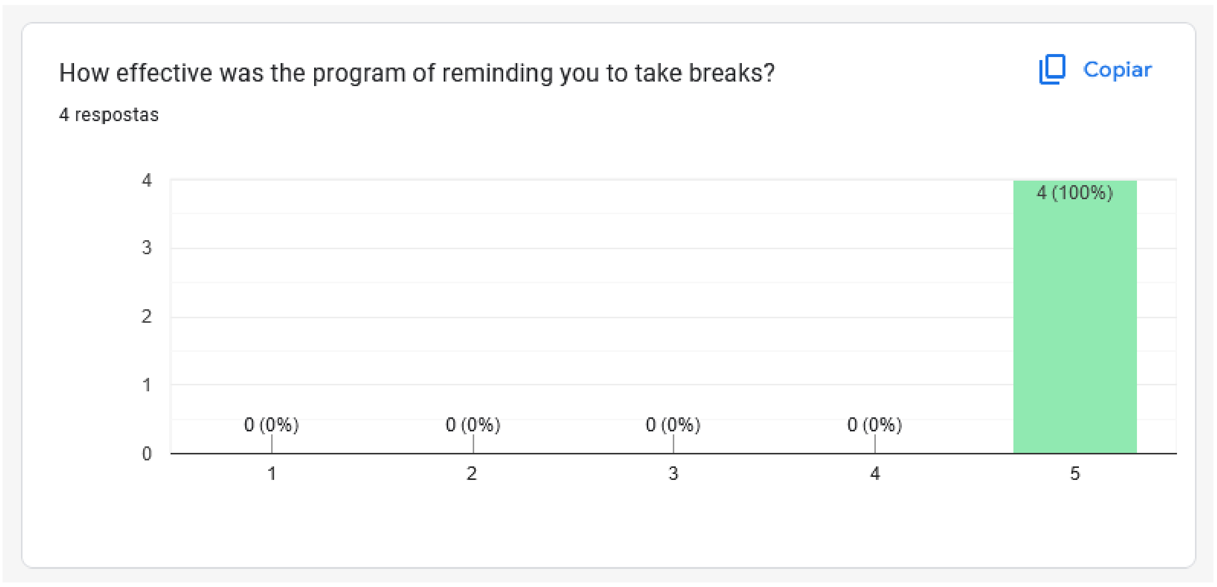
Figure 14 – Survey Question 6


Effectiveness and accuracy testing

How well did the blink reminder perform?

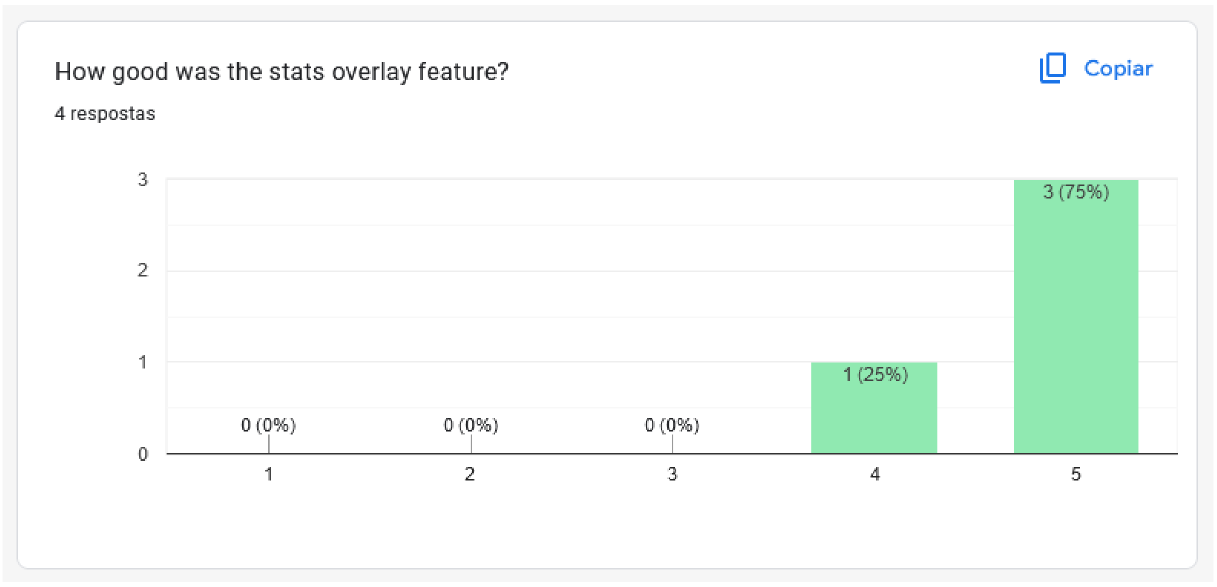
4 respostas

Figure 15 – Survey Question 7


How effective was the program of reminding you to take breaks?

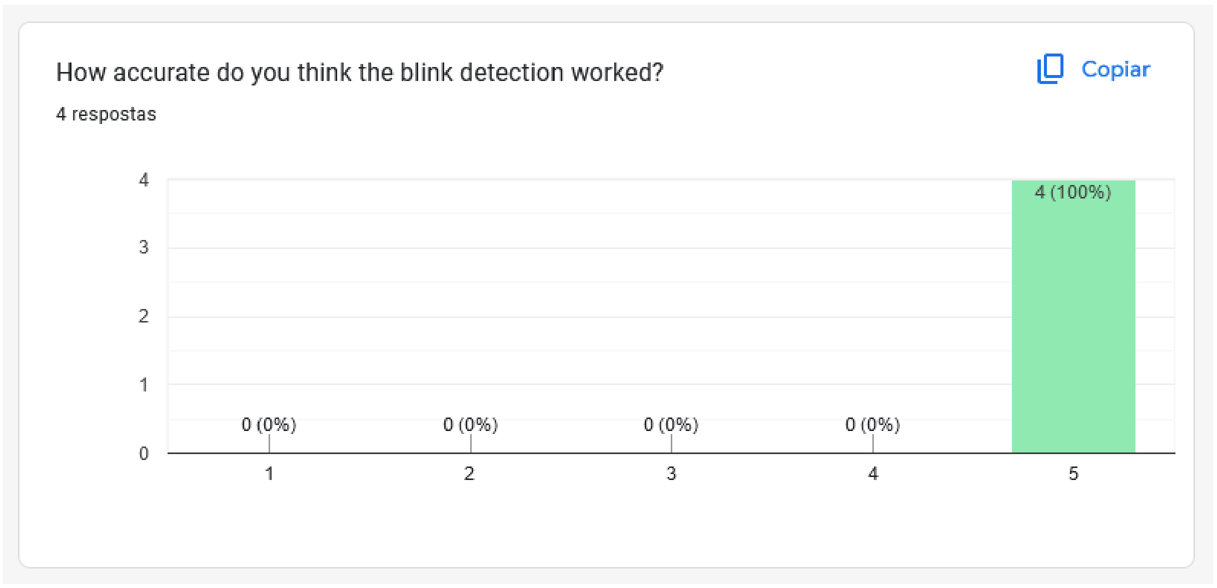
4 respostas

Figure 16 – Survey Question 8


How good was the stats overlay feature?

4 respostas

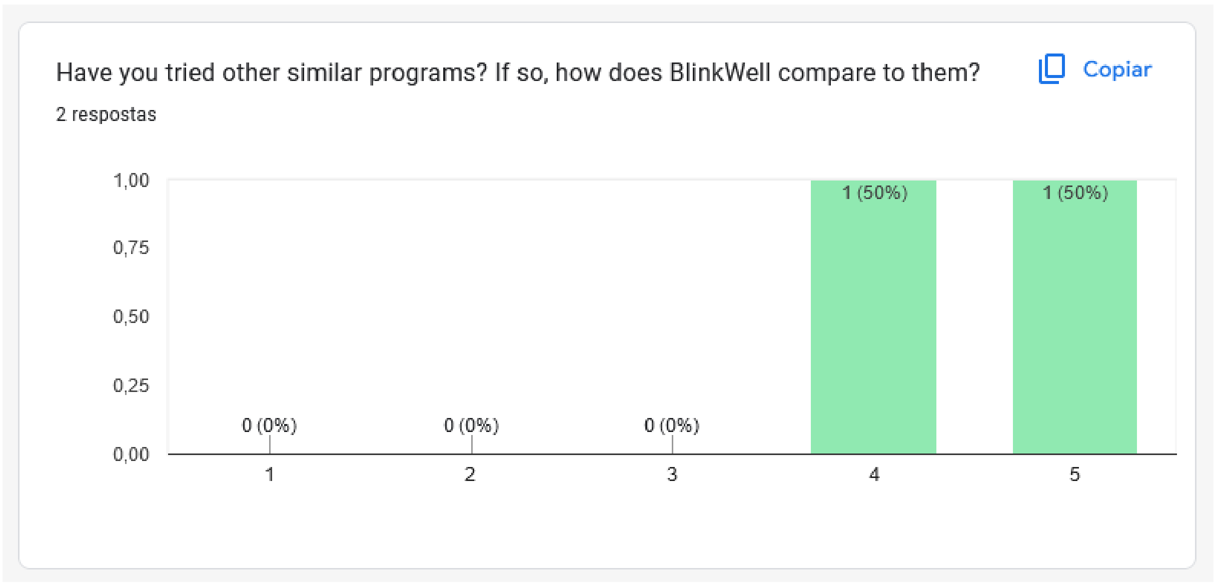
Figure 17 – Survey Question 9



How accurate do you think the blink detection worked?

4 respostas

Figure 18 – Survey Question 10



Have you tried other similar programs? If so, how does BlinkWell compare to them?

2 respostas