



# Detecting In-Game Toxicity via Bullet Hole Patterns Using Image Recognition

Bachelor's thesis

Degree Programme in Computer Applications

Spring 2024

Onni-Petteri Rantanen

Computer Applications

Tekijä Onni-Petteri Rantanen

Työn nimi Luodinreikien luomien toksisten viestien havaitseminen kuvantunnistuksen avulla

Ohjaaja Tommi Saksa

Tiivistelmä

Vuosi 2024

Tämän opinnäytetyön tavoitteena oli kehittää järjestelmä, joka pystyy havaitsemaan luodinrei'illä luotuja toksisia viestejä ammutapeleissa, erityisesti Counter-Strike 2:ssa. Projektissa yritettiin ratkaista haaste, joka liittyy pelaajien luoman sopimattoman tekstin tunnistamiseen pelin ympäristössä, jossa perinteisten moderointijärjestelmät eivät ole riittäviä. Tutkimuksessa keskityttiin tehokkaimpien kuvankäsittelytekniikoiden ja optisen tekstintunnistuksen (Optical Character Recognition, OCR) parametrien määrittämiseen, jotta luodinrei'istä muodostetut tekstit voitaisiin tunnistaa tarkasti. Työssä tarkasteltiin myös laajemmin pelien sisäistä toksisuutta ja sen vaikutuksia pelaajayhteisöihin. Tämä projekti sai alkunsa tekijän henkilökohtaisista kokemuksista pelien sisäisestä toksisuudesta, mikä motivoi kehittämään uudenlaisen ratkaisun tähän ongelmaan.

Tutkimusongelmaa lähestyttiin käytännönläheisesti ja kokeellisesti. Ensin käsiteltiin kuvantunnistukseen ja OCR-tekнологiaan liittyviä keskeisiä käsitteitä, minkä jälkeen keskityttiin erilaisten kuvankäsittelytekniikoiden, kuten Gaussian Blur -suodatuksen, Bilateral Filtering -suodatuksen ja Contour Detection -menetelmän kehittämiseen ja testaamiseen. Näitä tekniikoita testattiin iteratiivisesti niiden vaikutusten arvioimiseksi erilaisissa pelitilanteissa. Data analysoitiin vertaamalla näiden menetelmien tehokkuutta, jotta voitiin tunnistaa luotettavimmat lähestymistavat.

Tutkimus osoitti, että yksittäinen kuvankäsittelytekniikka ei ole universaalisti tehokas kaikissa tilanteissa. OCR:än onnistuminen riippuu suuresti kontekstista, mikä vaatii mukautuvaa kuvankäsittelyä ja huolellisesti valittuja OCR-parametreja. Tulokset viittaavat siihen, että kuvankäsittelytekniikoiden ja OCR-asetusten valinnan automatisointi voisi merkittävästi parantaa järjestelmän käytettävyyttä. Tämä projekti luo pohjan jatkotutkimukselle, jossa pyritään kehittämään entistä kehittyneempiä ja mukautuvampia järjestelmiä pelien sisäisen toksisuuden torjumiseksi.

Avainsanat Pelin toksisuus, kuvantunnistus, kirjainten tunnistus, kuvan esikäsittely, ensimmäisen persoonan ammutapelit

Sivut 42 sivua ja liitteitä 1 sivu

---

The purpose of this thesis was to develop a system capable of detecting toxic messages conveyed through bullet hole patterns in first-person shooter games, with a focus on Counter-Strike 2. The thesis sought to address the challenge of identifying inappropriate text created by players within the game environment, which goes undetected by traditional moderation systems. The research questions centered around determining the most effective image preprocessing techniques and Optical Character Recognition (OCR) parameters to accurately detect and recognize text formed by bullet holes. The broader issue of in-game toxicity, including various subtle and overt behaviors, was also explored. This thesis was commissioned as a personal project motivated by the author's experiences with in-game toxicity.

The research problem was approached through a practical, experimental methodology. The central concepts related to image recognition and OCR were explained first. The thesis then discusses the development and testing of various image preprocessing techniques, including Gaussian Blurring, Bilateral Filtering, and Contour Detection. The primary research method involved iterative testing of these techniques to evaluate their impact on the accuracy of the Tesseract OCR engine in recognizing text under different conditions. Data was analyzed by comparing the effectiveness of these methods in various game scenarios to identify the most reliable approaches.

Research demonstrates that no single preprocessing technique is universally effective for all scenarios. Instead, the success of OCR depends on the specific case, requiring adaptive preprocessing and carefully selected OCR parameters. The analysis indicates that automating the selection of preprocessing techniques and OCR settings could significantly enhance the system's use. This project lays the groundwork for further exploration into more sophisticated and adaptable detection systems for combating in-game toxicity.

**Keywords** In-Game Toxicity, Image Recognition, Optical Character Recognition, Image Preprocessing, First-Person Shooter Games

**Pages** 42 pages and appendices 1 page

## Glossary

OCR	Optical Character Recognition
PSM	Page Segmentation Mode
CS:2	Counter Strike 2
FPS	First Person Shooter
RGB	Red, Green, Blue
BGR	Blue, Green, Red
HSV	Hue, Saturation, Value

# Content

1	Introduction .....	1
2	Toxicity in online games .....	2
2.1	Why online games are toxic .....	2
2.2	How toxicity in video games is detected .....	3
2.2.1	Player reporting system .....	4
2.2.2	Automated Detection Algorithms .....	7
2.2.3	Human moderation and review .....	10
2.2.4	Progressive discipline and education .....	11
2.2.5	Account restrictions .....	11
2.3	Undetected toxicity .....	12
2.3.1	Team sabotage and friendly fire .....	13
2.3.2	“Teabagging” and disrespectful gestures .....	14
2.4	Toxicity in Images .....	15
3	Image recognition .....	17
3.1	Optical Character Recognition .....	17
3.2	Application in FPS games .....	18
4	Image preprocessing methods .....	20
4.1	Gaussian blur & Bilateral filter .....	20
4.2	Colour space conversion: Saturation and Grayscale .....	21
4.3	Edge Detection .....	22
4.4	Contour detection .....	22
5	Implementation .....	23
5.1	Image preprocessing .....	23
5.2	PyTesseract .....	31
6	Results .....	33
7	Summary .....	36
8	References .....	38

## Figures

Figure 1. A screenshot of a Reddit post where a member of the Overwatch Subreddit is complaining about toxicity. ....	4
Figure 2. Reporting a player in Overwatch 2. ....	5
Figure 3. A message the player receives if a user they reported gets punished. ....	6
Figure 4. Screenshot of a Reddit post where a member of the Valorant Subreddit is happy with receiving feedback for reporting.....	7
Figure 5. Voice chat may be recorded message a user receives after joining a voice channel in Overwatch 2. ....	8
Figure 6. Example of a possible blacklist. ....	9
Figure 7. CS:2 Warning after shooting a teammate.....	9
Figure 8. Screenshot of an X post by @Blizzardcs showcasing how the Blizzard customer support account answers users questions about getting punished.....	10
Figure 9. The Competitive Play Qualification Challenge.....	12
Figure 10. Reporting a player for Gameplay Sabotage in Overwatch 2. ....	13
Figure 11. Hello World written with bullet holes in the wall. ....	15
Figure 12. List of Tesseract configurations.....	18
Figure 13. Hello World written with bullet holes in CS:2. ....	24
Figure 14. Hello World with bullet hole edges detected.....	26
Figure 15. Hello World with contours detected. ....	28
Figure 16. Contours being too far apart.....	29
Figure 17. Contour centroids connected by a line. ....	30

Figure 18. An example why connecting contours might not be a good idea. ....	31
Figure 19. Hi written on a brick wall background. ....	34
Figure 20. "Cat" written on a very complex background texture. ....	35

## Codes

Code 1. Applying Gaussian Blurring. ....	24
Code 2. Applying Bilateral Filtering. ....	25
Code 3. Changing the saturation of the image. ....	25
Code 4. Applying Grayscale. ....	25
Code 5. Applying Canny Edge detection. ....	25
Code 6. Finding each contour and drawing a line around them. ....	27
Code 7. Calculating each contours moment. ....	29
Code 8. Drawing a line through centroids of each contour in a certain distance. ....	30
Code 9. Running the PyTesseract OCR engine. ....	31
Code 10. Recognized text from Figure 13 with optimal parameters. ....	33
Code 11. Recognized text from Figure 13 with wrong parameters. ....	33
Code 12. Recognized text from the OCR engine for Figure 19. ....	34

## **Appendices**

Appendix 1. Material management plan



# 1 Introduction

Toxicity is a persistent issue in online gaming. Players often resort to various means in order to engage in harmful behaviour such as verbal abuse and harassment. Gaming platforms have implemented lots of measures to combat toxicity. Some of these measures are chat filters and reporting systems, but players find loopholes around these filters. A significant loophole in first-person shooter (FPS) games is shooting bullet holes in walls. Players can spell inappropriate words to their teammates with this method, and it completely evades detection and moderation.

By taking advantage of image recognition technology this thesis tackles the issue of said loophole. The solution aims to automatically detect and mitigate instances of in-game toxicity by analyzing screenshots that are captured during gameplay.

This thesis will go through the design, implementation and evaluation of the system, examining its effectiveness. Furthermore, it will discuss the technical challenges, and potential implications associated with deploying a system like this in online gaming environments.

Through the development and analysis of the system, this research aims to answer the following questions:

- How do different combinations of image preprocessing techniques affect the accuracy of text detection
- What are all the bad behaviours that are present in online games?
- Would companies be able to use this solution or is there more to innovate?

## 2 Toxicity in online games

Online communities provide opportunities for social interaction and immersive experiences. However, they are increasingly plagued by issues of toxicity, which includes behaviors such as harassment, verbal abuse, cheating, and the use of hate speech (Aguerri et al., 2023). This toxicity not only diminishes the enjoyment that gaming should provide but also poses a significant threat to game companies' revenues. Toxic environments can discourage new players from joining or continuing with a game, leading to a decline in player engagement and ultimately impacting the financial success of these games. (Beres et al., 2021)

The efforts to combat these issues involve a combination of technological solutions, community guidelines, and proactive involvement from both developers and players. Addressing toxicity is not only about managing negative behaviour but also about building a culture that promotes positive social interaction and makes the gaming experience better for all participants.

### 2.1 Why online games are toxic

Since toxicity has become deeply ingrained and expected within online culture, people often do not even bat an eye when someone says something mean to them online. If these interactions were compared to ones in real life, most people would be left shocked over some of the mean things that end up being said (Aguerri et al., 2023). This is because of one of the greatest drivers of toxic behaviour in the digital world; the lack of identity (Leetaru, 2019). The contrast between online interactions and real-world social dynamics highlights a significant difference in toxicity, with individuals often behaving civil in face-to-face encounters. Establishing identity is a critical factor in moderating online toxicity. Since online platforms often allow users to stay completely anonymous, people can say whatever comes to their mind in the moment, without thinking of consequences. In general, players who enjoy being anonymous in games engage in more game-influenced grief-play than those who do not (Chen et al., 2009, p. 343)

Within the context of video games, banter and toxicity have become intertwined and normalized parts of player interaction. Trash talk, taunting and playful teasing are usually included in the gaming experience, especially in competitive games. Due to the competitive nature of these games, such behaviour can escalate into more toxic forms. Research has shown that toxicity in online gaming is often perceived as an unavoidable or even acceptable aspect of the experience, with players frequently viewing such interactions as mere banter. (Beres et al., 2021) The design features of many multiplayer online games, which emphasize intense interpersonal communication and cooperation, contribute to this normalization, making toxic behaviour more pervasive. (Shen, 2013) According to the Anti-Defamation League (ADL, 2021), five out of six adults aged 18-45

experienced harassment in online multiplayer games in 2021. This number continues to grow each year as more people engage in online gaming.

In contrast to real-life sports settings, where banter is often light-hearted and maintained by camaraderie among players, the anonymous and competitive nature of online gaming typically leads to heightened intensity in toxic interactions. The lack of face-to-face interaction and nonverbal signals in online environments diminishes empathy and makes hostile behavior more likely. Nonverbal cues, such as facial expressions and vocal tone, are essential components of effective communication. In their absence, players are more prone to misunderstand each other's intentions, leading to heightened aggression and conflict. (Nogai, 2024) Since this behavior is so common in online games, players often view it as part of the culture and readily dismiss their actions without guilt. Additionally, the competitive nature of these games, combined with the lack of real-life consequences, encourages players to engage in more aggressive and toxic behaviors.

## **2.2 How toxicity in video games is detected**

Game developers make great efforts in order to reduce ingame toxicity. A notable example of a game that has been fighting toxicity for a long time is Overwatch 2, a team-based first-person shooter (FPS) developed by Blizzard Entertainment. The topic of toxicity has been brought up often by the community members of the Overwatch franchise. People have been discussing the problem, on various web forums such as shown in Figure 1.

Figure 1. A screenshot of a Reddit post where a member of the Overwatch Subreddit is complaining about toxicity. Retrieved from Reddit (u/Local\_Failure, 2021)

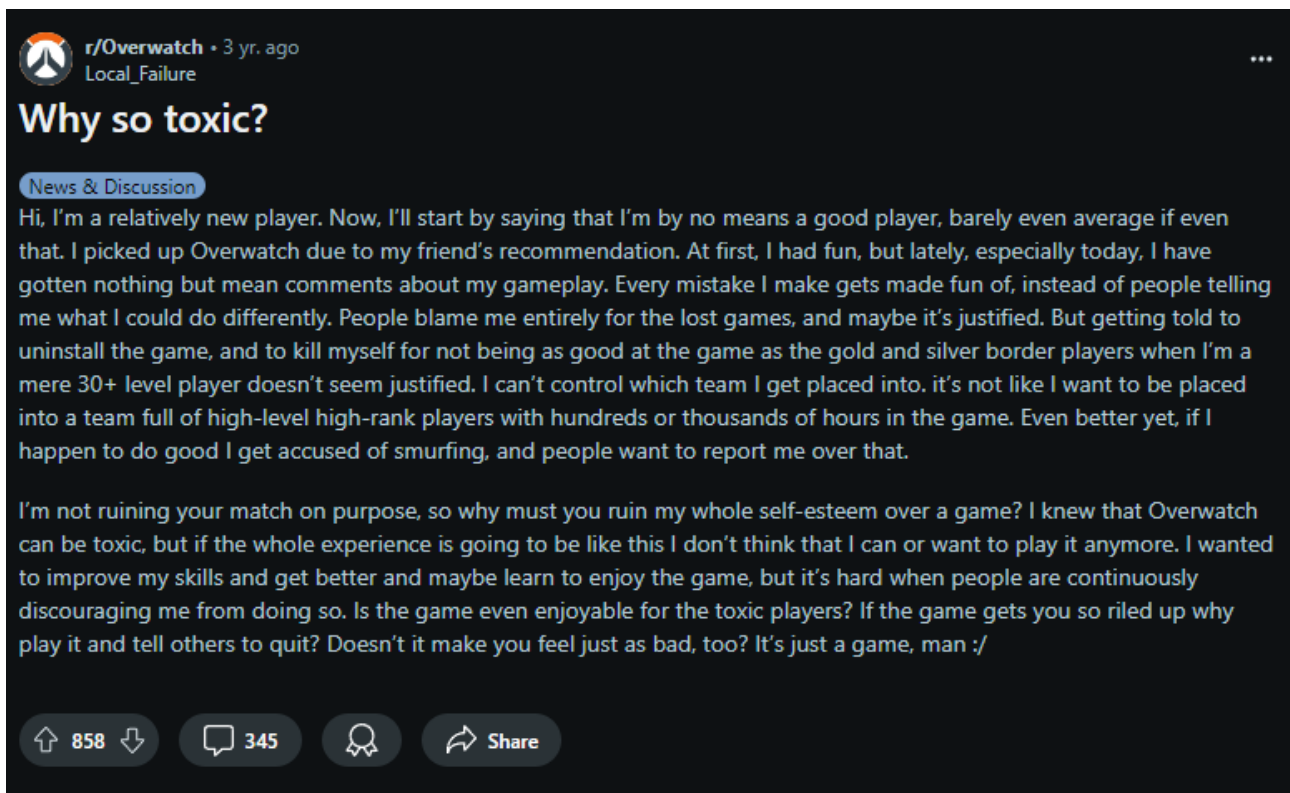


Figure 1 shows a member of the Overwatch community on Reddit expressing their frustration with the toxic environment in Overwatch 2, noting that it is damaging their self-esteem and making it difficult for them to enjoy or even improve in the game. This and many other instances are examples of Overwatch players wondering why the community has been so toxic for years now. Blizzard have heard these complaints and have been trying to fight against toxicity in the game through a combination of player reporting, automated algorithms, human moderation, and other constant development in the field of fighting online. After years of work, the developers of Overwatch made the game a great example of how to fight against toxic behaviour by deploying various different methods of detection.

### 2.2.1 Player reporting system

A big role of Overwatch 2's approach to detecting toxicity is its player reporting system. This system encourages users to report instances of inappropriate behaviour encountered during gameplay. Players can report others for a variety of reasons, such as abusive language,

harassment, and cheating. Figure 2 is an example of an essential report system in order to fight toxicity.

Figure 2. Reporting a player in Overwatch 2.

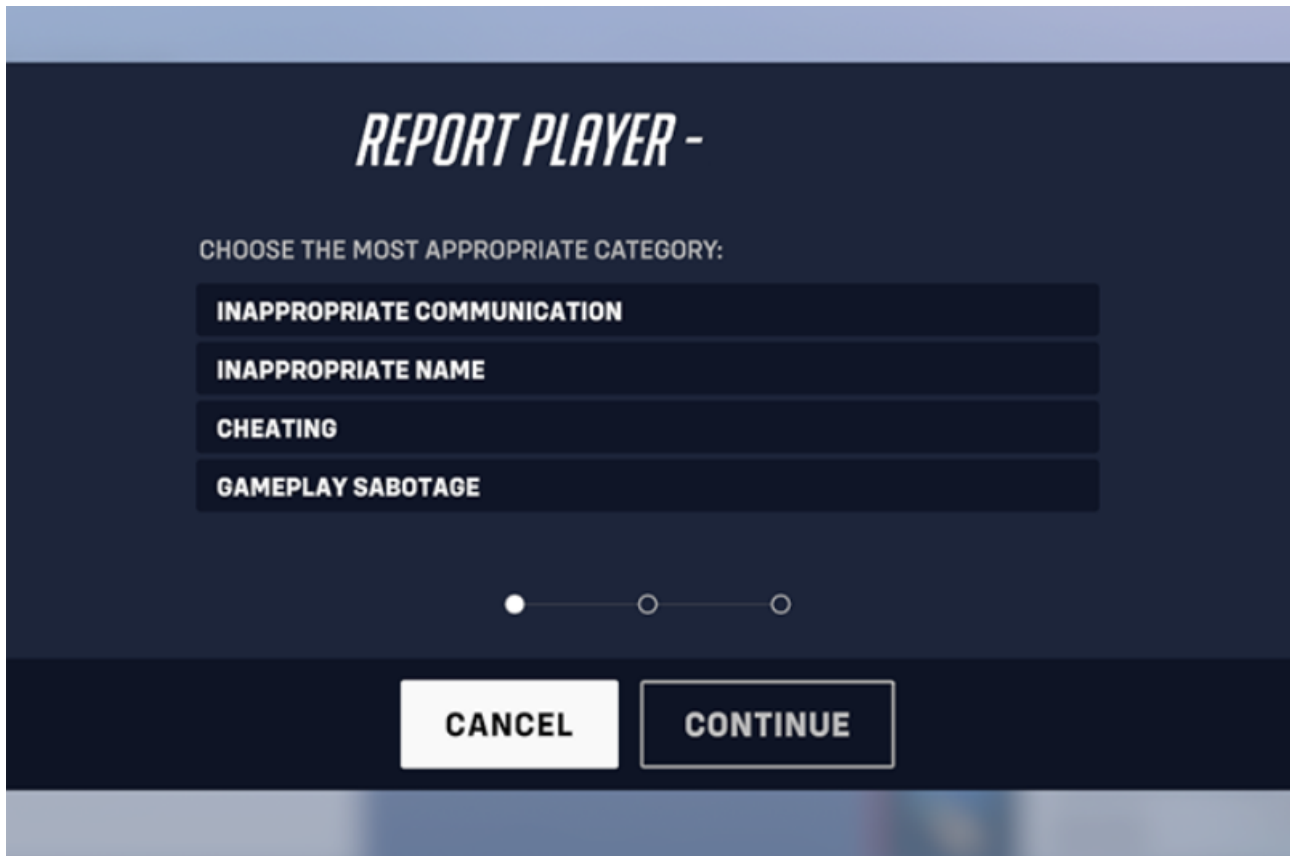
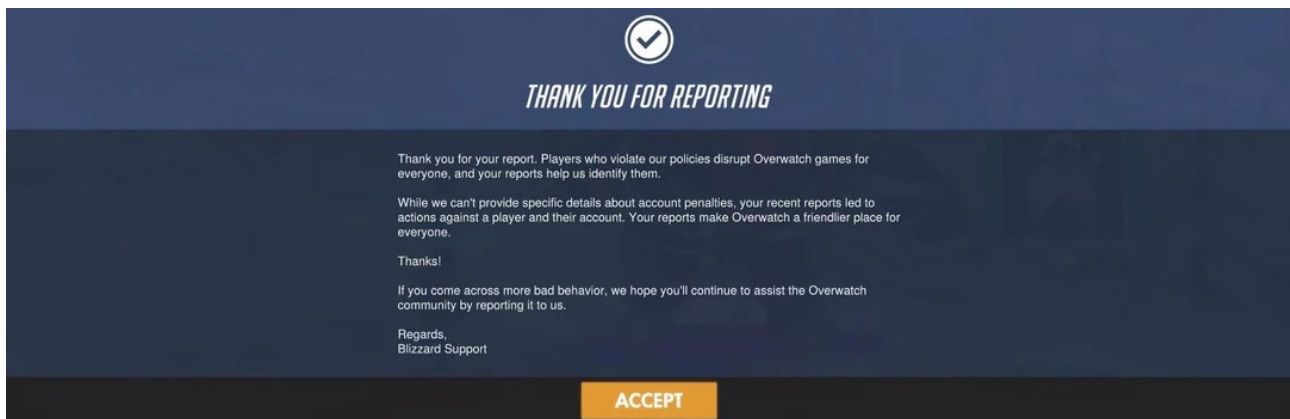
The image shows the 'REPORT PLAYER' screen from the game Overwatch 2. The title 'REPORT PLAYER -' is at the top in a stylized font. Below it, the instruction 'CHOOSE THE MOST APPROPRIATE CATEGORY:' is displayed. There are four selectable options: 'INAPPROPRIATE COMMUNICATION', 'INAPPROPRIATE NAME', 'CHEATING', and 'GAMEPLAY SABOTAGE'. A progress indicator with three circles is shown below the options, with the first circle filled. At the bottom, there are two buttons: 'CANCEL' and 'CONTINUE'.

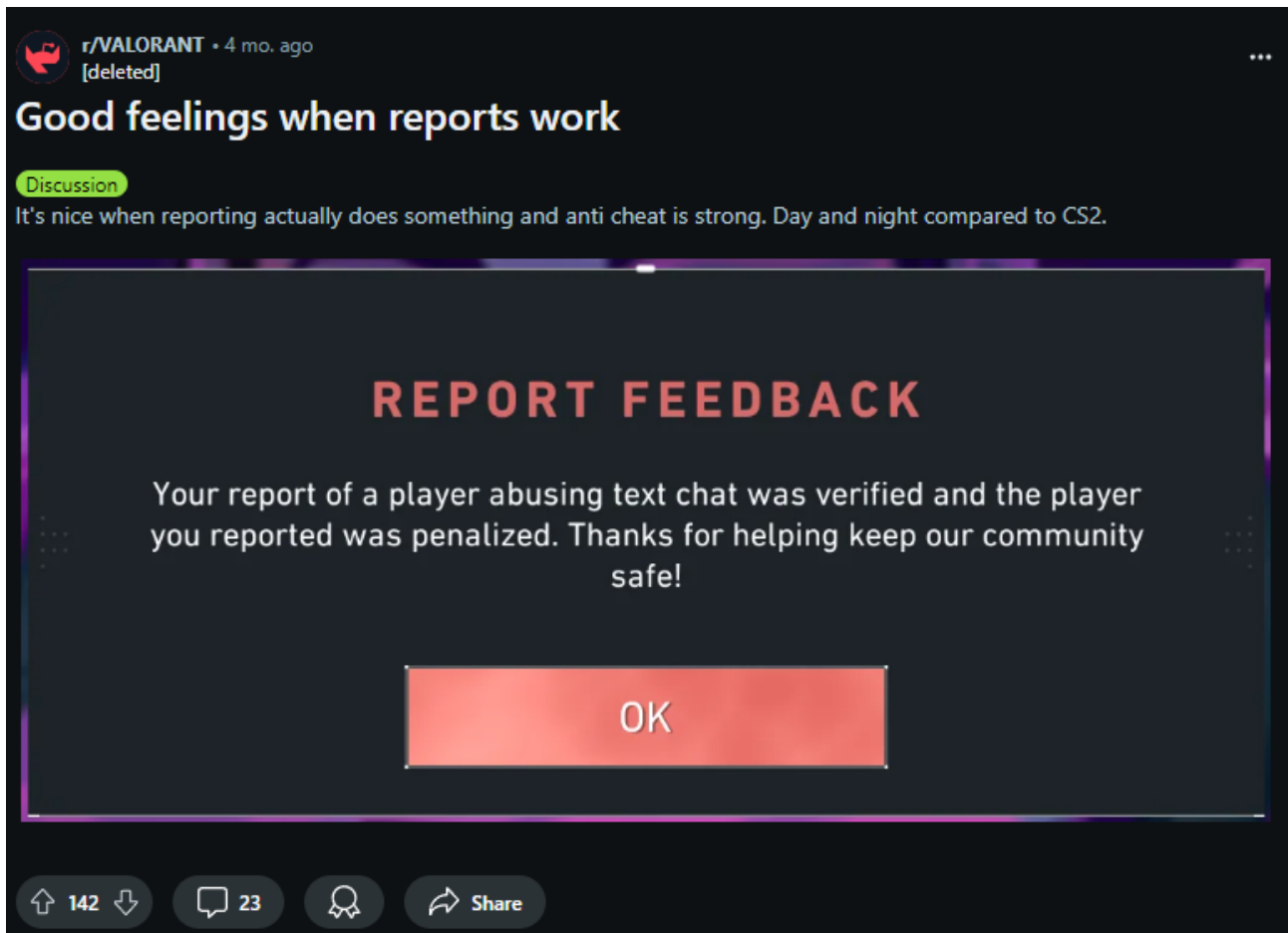
Figure 2 shows all the possible options a player can be reported player for, in Overwatch 2. Players have many categories that they can classify a report as. This allows the user to accurately describe what happened, so the moderation team can later assess the situation and make appropriate decisions. If the reported player eventually gets punished, the user gets the message shown in Figure 3 that thanks them for reporting.

Figure 3. A message the player receives if a user they reported gets punished.



Receiving these messages makes the user feel heard, and the user feels like their report matters. As noted by Marina, a researcher interviewed by the ADL, feedback is integral to systems of reward and punishment because these systems are tied to reports from players. If players do not feel their reports matter, they will not report abuse, directly affecting whether in-game warnings and other tools are used (ADL, 2023). Other games such as Valorant also participate in thanking their users for reporting. Figure 4 shows the thank you message Valorant players receive.

Figure 4. Screenshot of a Reddit post where a member of the Valorant Subreddit is happy with receiving feedback for reporting (Reddit, 2024).



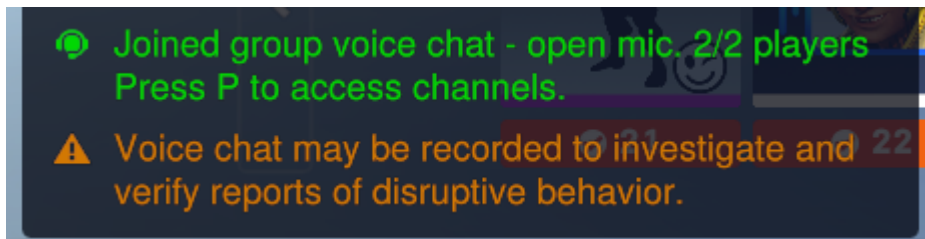
The message shown in Figure 4 is what the user receives when a player they reported gets punished in Valorant. RIOT also clarifies what they got punished for, so the user gets an extra understanding of which player it could have been. In the case of Figure 4 it was a player abusing the text chat. The Reddit user in Figure 4 is very happy to know that their reports are actually making an impact, and it is a logical assumption to make that they will report again in the future, since the player feels heard.

### 2.2.2 Automated Detection Algorithms

In addition to allowing players to manually report offenders, Overwatch 2 employs automated detection algorithms to analyse in-game data and identify patterns indicative of toxic behaviour.

The algorithms deployed have the ability to monitor chat logs and record in-game voice chats for further analysis. Figure 5 shows the warning message a user receives upon joining a voice chat in Overwatch 2.

Figure 5. Voice chat may be recorded message a user receives after joining a voice channel in Overwatch 2.



Audio transcription allows Blizzard to collect temporary voice chat audio recordings of players, and transcribe it through text to speech programs (Blizzard, 2023). In addition to this, Blizzard has also implemented machine learning algorithms to better recognize instances of toxicity (Dunn, 2020). These technologies analyze text and speech in real-time, effectively flagging and filtering some of the offensive content before it can impact other players. Automated systems often employ keyword filtering and blacklisting techniques to identify and intercept basic known toxic phrases or words. Figure 6 shows what an example blacklist could look like.



Figure 6. Example of a possible blacklist.

```

1 2glc
2 2 girls 1 cup
3 acrotomophilia
4 alabama hot pocket
5 alaskan pipeline
6 anal
7 anilingus
8 anus
9 apeshit
10 arsehole
11 ass
12 asshole
13 assmunch
14 auto erotic
15 autoerotic
16 babeland
17 baby batter
18 baby juice
19 ball gag
20 ball gravy
21 ball kicking
22 ball licking
23 ball sack
24 ball sucking
25 bangbros
26 bangbus
27 bareback
28 barely legal
29 barenaked
30 bastard
31 bastardo
32 bastinado
33 bbw
34 bdsm
35 beaner
36 beaners
37 beaver cleaver
38 beaver lips
39 bestiality
40 bestiality

```

By maintaining a database of offensive terms and expressions as shown in Figure 6, these automated systems can match incoming chat messages or voice transcripts against the words in the blacklist, triggering automated actions. Most commonly this means punishing the player. Another example of an automated detection system can be seen in Counter-Strike 2 (CS:2). CS:2 is a fast-paced, team-based tactical shooter video game developed by Valve. One of the games mechanics allows the player to hurt their own teammates. In order to deter toxic behaviour, Valve placed an algorithm that automatically detects if teammates have been harmed. After harming a teammate in CS:2, players receive the warning shown in Figure 7.

Figure 7. CS:2 Warning after shooting a teammate.



After receiving the warning, subsequent offenses will lead to more severe penalties, including

being kicked from the game. The types of automated systems games have vary depending on the mechanics of each game. Each system is as important as the other for maintaining integrity and fairness in the game environment. While automated detection systems help identify potential instances of toxicity, they may also produce false positives. This highlights the need for human oversight and intervention.

### 2.2.3 Human moderation and review

While automated systems are adept at handling clear-cut cases of misconduct, such as the use of prohibited language or obvious instances of harassment, there are cases where these algorithms are not the most optimal. Due to the subtleties and complexities of human interactions, human moderators are a core part of handling in-game toxicity. Blizzard has not explicitly confirmed the exact role of the human moderation team for Overwatch 2. However, some specifics have been revealed. According to Blizzard, while handling a case of bad voice chat behaviour no one listens to the temporary audio, which is quickly deleted after being automatically transcribed by AI. This means that any human element in the moderating process may need to purely rely on the transcribed audio being accurate. (Pearson, 2023) If a user feels like they have been punished unjustly, they can submit an appeal to the Customer Support team, and this team of human moderators will review the reports that were made against the user's account. (Blizzard, 2024a) Figure 8 shows an example of a Blizzard customer support team member replying to an inquiry about an accounts status.

Figure 8. Screenshot of an X post by @Blizzardcs showcasing how the Blizzard customer support account answers users questions about getting punished. (Blizzard CS, 2024).



The user who the response in Figure 8 was directed to, believed they were unjustly punished, due to being mass reported by trolls. This allowed the moderation team to give clarification on why the users account was suspended. If the user was actually falsely punished, they would never be able to get their account back if there was no team to look into this. While automated systems take the

workload off humans, they are still needed for situations where punishments might be unnecessary.

#### **2.2.4 Progressive discipline and education**

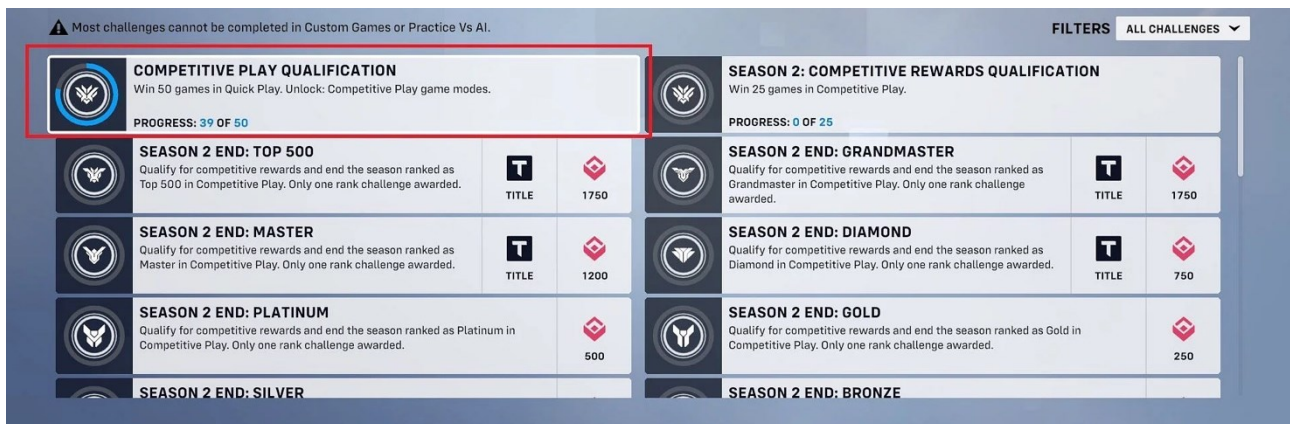
Overwatch 2 employs a system of progressive discipline to manage and mitigate player toxicity, aligning punishment with the severity and recurrence of player infractions. The progressive discipline system starts with an automatic warning, if the user's account has been reported several times for violating in-game rules (Blizzard, 2024a). This initial step serves as a corrective prompt to encourage players to reflect on their behaviour without immediately resorting to harsh penalties. If the behaviour persists, the system escalates the severity of penalties. The severity of the punishment increases every time it is added to the user's account. These penalties begin with silences, but repeated violation of chat behavior can result in suspensions up to account closures. (Blizzard, 2024b) This approach is grounded in the principle that penalties should not only punish but also deter future misconduct.

If an account is banned from Competitive Play during three separate competitive seasons, it may be permanently banned from Competitive Play (Blizzard, 2024c). This severe consequence is reserved for those who consistently disrupt the gaming environment. Additionally, Overwatch 2 incorporates educational components into its disciplinary measures. Players who receive penalties are often provided with explanations for the punishment, detailing what behaviour violated the guidelines and why it is unacceptable.

#### **2.2.5 Account restrictions**

More often than not, the same bad actors will try to break terms of service repeatedly. Due to this, games often put account restrictions in place. In the launch of Overwatch 2, Blizzard initially implemented phone number verification for creating new accounts as a strategy to combat toxicity and smurfing (Good, 2022); the practice where experienced players create new accounts to gain an unfair advantage over less skilled players (Gank, 2023). However, this measure was met with significant backlash from the community, particularly from players using prepaid phones who found themselves unable to participate (McWerthor, 2022). In response to these complaints and the unintended exclusion of legitimate players, Blizzard decided to remove the phone number verification requirement (Kain, 2022). Despite this change, smurfing remains a contentious issue within the game, as it often leads to imbalanced matches and a frustrating experience for both new and experienced players. As shown in Figure 9, in order to combat smurfing, Overwatch 2 forces new accounts to 50 casual games in order to unlock the competitive game mode.

Figure 9. The Competitive Play Qualification Challenge.



A new user has to win 50 games of casual play to complete the Competitive Play Qualification challenge in Overwatch 2. Completing the challenge allows the user to participate in competitive play. This gives new players time to prepare for the higher expectations that come with competitive (Blizzard, 2022), while also making it a lot harder for offenders to constantly create new smurf accounts, due to the effort needed to unlock the competitive game mode.

## 2.3 Undetected toxicity

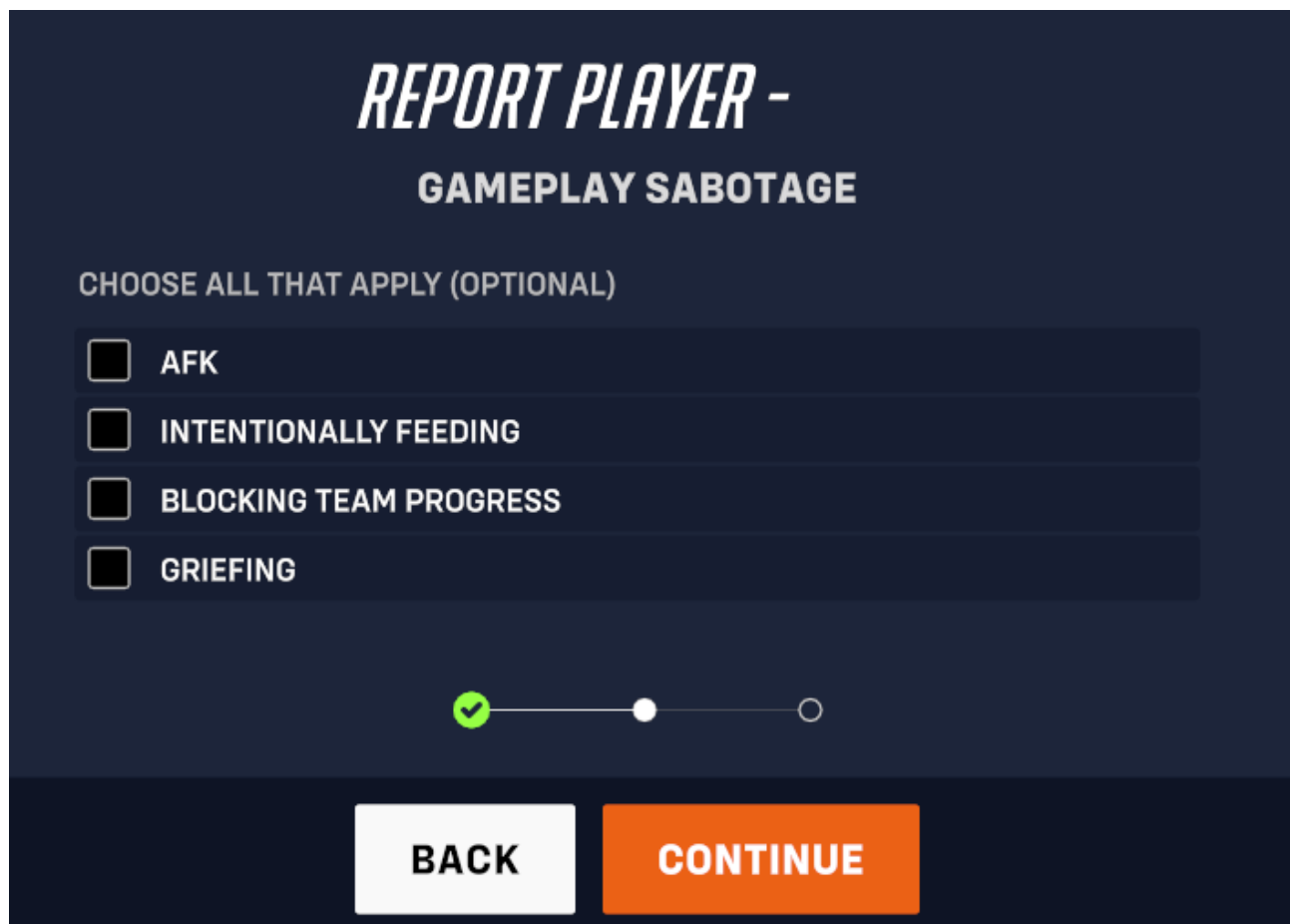
Undetected toxicity in online multiplayer games refers to subtle behaviors that go undetected in the traditional moderation and automated detection systems. Unlike blatant forms of harassment, these toxic actions are involved in the manipulation of game mechanics or take part in very nuanced social interactions, making them difficult to identify and report.

For developers, tackling this type of toxicity presents a unique challenge. It requires more than just technical expertise; it demands an intimate understanding of how players interact with the game and each other. If developers only focus on the overall picture of toxicity detection, some forms go completely undetected (Meyer, 2020, p. 55). This involves engaging directly with the game as regular players to uncover the evolving behaviors of players that can undermine the gaming experience. Active involvement with the community, continuous observation, and a willingness to adapt are essential for developers to effectively address these undetected forms of toxicity. This chapter explores some of the different forms of undetected toxicity.

### 2.3.1 Team sabotage and friendly fire

Team sabotage in online multiplayer games is a form of toxicity that significantly undermines the principles of teamwork and cooperation. This behaviour can manifest in various forms, such as intentionally blocking teammates' movements, being AFK (Away from keyboard), or causing accidental damage through friendly fire. (KIDAS, 2024) The impact and nature of these actions largely depend on each game's specific mechanics, which dictate the possibilities and limitations of player interactions. In Overwatch 2, several specific types of team sabotage are recognized and can be reported by players, as shown in Figure 10.

Figure 10. Reporting a player for Gameplay Sabotage in Overwatch 2.



**REPORT PLAYER -**  
**GAMEPLAY SABOTAGE**

CHOOSE ALL THAT APPLY (OPTIONAL)

- ☐ AFK
- ☐ INTENTIONALLY FEEDING
- ☐ BLOCKING TEAM PROGRESS
- ☐ GRIEFING

Progress bar: [Green checkmark icon] — [White dot] — [Empty circle]

**BACK** **CONTINUE**

The actions listed in Figure 10 fall under the broader category of “gameplay sabotage” in Overwatch 2. Being AFK hinders a team’s performance, as players who are AFK are not actively participating in the game. Players who deliberately get themselves killed to give the opposing team an advantage are “intentionally feeding”. Blocking team progress, is almost similar to intentionally feeding, essentially blocking team progress in a game where there is a broader objective in order

to win. Players who engage in actions specifically designed to frustrate or harass their teammates are called griefers. All of these actions can significantly disrupt the gameplay of everyone else in the game.

In some games, such as CS:2, friendly fire is enabled, adding a layer of realism to player interactions. This feature provides an option for players inclined towards toxic behaviour to sabotage their team by intentionally shooting allies. However, Overwatch 2 takes a different approach by completely removing friendly fire from its gameplay mechanics. This decision by Blizzard is designed to foster a more controlled and teamwork-oriented environment, ensuring that players cannot harm their teammates.

The contrasting approaches of CS:2 and Overwatch 2 to friendly fire and team dynamics illustrate the broader spectrum of design philosophies in the gaming community. These differences highlight how game developers can influence player behaviour and team interaction through careful adjustments to game mechanics. Both games' approaches have their benefits and challenges, and they cater to different player preferences and competitive styles.

### **2.3.2 “Teabagging” and disrespectful gestures**

In online multiplayer games, disrespectful gestures or taunts towards opponents can intensify the culture of toxicity. A notable example of this is “teabagging”. Teabagging is a controversial practice where a player’s avatar repeatedly crouches over a defeated player’s “body” in order to simulate rubbing his or her genitals over the avatar’s body. While often dismissed as harmless or playful banter, these actions convey a sense of disrespect. (Myers, 2017, p. 4)

Despite being controversial, teabagging continues to be a common sight in many games, which can contribute to a hostile environment, particularly for new or less skilled players. This behaviour can discourage player participation and fosters a community where toxicity is normalized.

However, some games do not mechanically allow teabagging such as CS:2. In CS:2 players are not allowed to repeatedly crouch, this removes teabagging from the picture completely. This decision by the developers serves many purposes, such as not allowing crouch peeking but also removes the problem of teabagging. Teabagging is not the only disrespectful gesture, there are numerous different kinds of things players can do to let out their anger. Another good example is harming teammates in CS:2 as mentioned before in chapter 5.2.2. The possible gestures really depend on each game's mechanics.

Major tournament organizers have begun to target this behaviour as well. RIOT, a prominent esports organizer has implemented policies that punish players for actions considered disrespectful, such as shooting an opponent's dead body, or teabagging during tournaments (Jiang, 2022). These measures are part of a broader effort to cultivate a more positive and inclusive gaming community.

## 2.4 Toxicity in Images

Some games like Skribbl.io use images as a primary form of gameplay and communication. Identifying toxic content within these images involves aligning the visual image feature with textual analysis to understand the context of the drawing. This poses a challenge beyond simply identifying harmful images, for in such games, freehand drawings can be fluid and it can be difficult to draw a structured image from a mouse or a touchscreen, hence a higher amount of image interpretation is required. (Collaboratory Against Hate, 2023)

This same issue can be found in FPS games, even though images are usually not the primary form of communication. Shooting walls to create messages with bullet hole patterns is a way to convey toxic messages, while bypassing the detection systems. Such actions take advantage of environmental mechanics in game, in this case bullet holes. These bullet holes are visible to all players and can remain for on the environment for the duration of a match, or a short moment, depending on the games mechanics. Figure 11 shows an example of this mechanic.

Figure 11. Hello World written with bullet holes in the wall.



The subtlety of this form of misconduct lies in its indirect nature. Since it does not use the game's

chat systems, in which messages are directly monitored and moderated. Instead, as Figure 11 shows, it uses a game feature in a way that is unintended by developers. This method allows players to bypass traditional safeguards, allowing toxicity to go undetected solely due to the games mechanics.



### 3 Image recognition

Computer vision is a rapidly advancing field within artificial intelligence that enables computers to interpret and understand digital images. As defined by Baheti (2022), it involves the application of algorithms and techniques to analyze visual data. A critical subset of this field is image recognition, where, according to Yasar (2023), computers use machine vision technologies along with cameras and artificial intelligence to analyze and interpret images. This process allows for the identification of objects, patterns, or text within those images.

The primary objective of image recognition, as highlighted by Ramachandran (2023), is to allow machines to perceive and interpret visual data similarly to humans. This capability extends beyond simple image recognition; it includes classifying and analyzing objects within an image or video sequence.

This technology finds application in a long list of sectors, enhancing both efficiency and effectiveness. In the fields of security, image recognition is used to enhance surveillance systems, enabling the automatic detection of suspicious activities or individuals. The eCommerce sector benefits from this technology through improved product recommendation systems and visual search capabilities, which enhance the shopping experience. Additionally, in healthcare, image recognition supports diagnostic processes, such as analyzing medical images to identify diseases early, which can significantly improve patient outcomes. (Comidor, 2022)

#### 3.1 Optical Character Recognition

This project will utilize Optical Character Recognition (OCR) to detect words created by bullet holes written within the game environment. Character recognition is the technique to detect, identify and segment the characters present in images (Nanda, Goswami, 2023). It works by comparing the text, character by character, with a database of possible characters (Smith, 2023). This ensures that each character identified by the OCR system matches its closest counterpart in the predefined character set, allowing for accurate text extraction from given images.

The OCR engine chosen for this project is Tesseract. Tesseract is an open source OCR engine maintained by Google. (Ibrahimovic, 2023) It represents a powerful tool for character recognition in images. Tesseract is capable of accurately detecting and extracting text from images, even in challenging and varied environments. Through training on vast datasets and continuous refinement, Tesseract achieves impressive levels of accuracy in recognizing text across different languages, fonts, and styles. Since there are many different types of texts to extract from images,

using Tesseract's page segmentation modes (PSMs) listed in Figure 12, helps achieve the most optimal results.

Figure 12. List of Tesseract configurations

```
$ tesseract --help-psm
Page segmentation modes:
 0 Orientation and script detection (OSD) only.
 1 Automatic page segmentation with OSD.
 2 Automatic page segmentation, but no OSD, or OCR. (not implemented)
 3 Fully automatic page segmentation, but no OSD. (Default)
 4 Assume a single column of text of variable sizes.
 5 Assume a single uniform block of vertically aligned text.
 6 Assume a single uniform block of text.
 7 Treat the image as a single text line.
 8 Treat the image as a single word.
 9 Treat the image as a single word in a circle.
10 Treat the image as a single character.
11 Sparse text. Find as much text as possible in no particular order.
12 Sparse text with OSD.
13 Raw line. Treat the image as a single text line,
    bypassing hacks that are Tesseract-specific.
```

Each PSM that Figure 12 displays is designed for specific text layouts and structures, enhancing OCR accuracy by aligning the recognition process with the characteristics of text. For example, PSMs like PSM 3 and PSM 4 are ideal for documents with predictable layouts, automatically segmenting pages into text blocks, while PSM 7 and PSM 8 focus on single lines or words, ensuring high accuracy by reducing segmentation errors. By selecting the appropriate PSM for a given text, Tesseract effectively recognizes text leading to more reliable outcomes.

Tesseract is available as a Python library called PyTesseract. The Python version will be used, since Python is one of the widely used programming languages used for this purpose (Khandelwal, 2023). Python has various libraries that will help with image preprocessing, since that will be necessary for the workflow of this project. In addition to this, Python has a large and active community, which makes it easier to discover possible problems and answers that may come up during the development process.

### 3.2 Application in FPS games

Optical Character Recognition technology has a range of applications in FPS games, especially in monitoring in-game communications. By automatically capturing screenshots of in-game

environments during gameplay, the environment can be analyzed using an OCR engine. After analyzing the text, any inappropriate texts can be automatically detected.

Additionally, OCR engines can be used to enhance player interfaces by converting text found within the game environment into readable data that can be processed or displayed back to the player. This could include extracting and displaying text from in-game signs or documents that players interact with, providing a more immersive and interactive gaming experience.

## 4 Image preprocessing methods

Image preprocessing is a set of techniques employed to enhance the quality and extract relevant information from digital images before they are further analyzed and processed (Microblink, n.d.). This necessity stems from the difficulty associated with the ambiguity of visual information in images. The ambiguity of visual data often requires preprocessing to enhance the interpretability of images before they are analyzed by computational systems. This is particularly important in tasks where visual patterns need to be isolated from complex backgrounds, such as recognizing characters from bullet hole patterns in gaming environments (Jain et al., 1995). In the context of this project, recognizing characters from bullet hole patterns in the walls is impossible without highlighting the bullet holes themselves from the chaotic backgrounds that are typically found in game environments. The human eye can easily see the patterns and shapes as characters, but a computer will analyze the whole photo and not realize that it is looking at characters.

By using different image preprocessing techniques, the key features of the target image can be highlighted from the surroundings of the image. The preprocessed image allows the Optical Character Recognition (OCR) system to focus on characters spelled out by bullet holes, instead of all the other things happening in the background of the image. Effective preprocessing is crucial for OCR systems, as it significantly impacts the accuracy and efficiency of character recognition (Smith, 2007, p. 2). This enhances the accuracy and efficiency of the character recognition algorithm and makes it a viable solution to use in complex environments, such as those found in video games.

### 4.1 Gaussian blur & Bilateral filter

The Gaussian Blur is a smoothing filter, which is used to blur images and remove detail and noise (Cadena, et al. 2017). Noise often occurs due to various factors like compression artifacts, lighting variations or just general graphical rendering issues. In computer vision in general, reducing noise is often a critical pre-processing step because noise can interfere with edge detection and other feature extraction algorithms. The basic idea behind Gaussian blurring is to replace each pixel value with a weighted average of its neighbouring pixels (Vignesh, 2023). By smoothing the image, edges appear less jagged, and less noise is carried forward into the further phases of image preprocessing.

Bilateral filtering is an edge-preserving smoothing technique on an image (Almira, et al. 2016). The key idea is to consider the intensity difference between pixels, which makes it a step more complex

than Gaussian blurring. This dual filtering approach ensures that edges are preserved while smoothing the rest of the image, because sharp intensity changes are less likely to be averaged out. The bilateral filter thus maintains sharp edges while effectively reducing noise in flat regions, making it highly beneficial for pre-processing images. This makes it a very good method to use for detecting bullet holes, since each bullet hole has edges that have to be detected.

## 4.2 Colour space conversion: Saturation and Grayscale

Colour spaces are different methods of organizing colours in an image. Common colour spaces include RGB (Red, Green, Blue), BGR (Blue, Green, Red), and HSV (Hue, Saturation, Value). The HSV colour space separates image luminance from colour information, which is useful in image preprocessing, because it mimics the way humans perceive colour (Scaler Topics, n.d.). Converting an image to HSV from RGB or BGR allows manipulation based on brightness and colour saturation, which can be more intuitive than working directly in RGB or BGR for image preprocessing tasks. Since the conversion to HSV allows for a more effective adjustment of the saturation component, it can enhance certain features in an image by making them more or less prominent. (Python Geeks, n.d.) In the context of this project, increasing the saturation can help distinguish bullet holes from the background better.

Grayscale plays a pivotal role in image processing. It is particularly useful for preparing images for edge detection and OCR. Grayscale images are composed of 256 unique colours, where a pixel intensity of 0 represents the black colour and pixel intensity of 255 represents the white colour. All the other 254 values in between are the different shades of gray (Kundu, 2022). This step is important, because it reduces the amount of information the computer needs to process, focusing solely on intensity rather than the distractions of colour.

It might seem contradictory to first increase saturation and then convert an image to grayscale, but this works well in the project. By boosting saturation, the distinctiveness in colour is emphasized, which, when converted to grayscale, translates to a clearer distinction in intensity. This contrast enhancement is crucial for isolating features of interest, which in this case are bullet holes. By converting an colour image to grayscale, the computational burden during the subsequent processing steps is significantly reduced. Each pixel in a grayscale image requires only one value to represent its intensity, as opposed to three values in the RGB colour space. This reduction not only speeds up the processing time but also minimizes the computational resources required. Additionally, a lot of image processing algorithms, such as edge detection, are optimized for single-channel data which makes grayscale images ideal.

### 4.3 Edge Detection

Edge detection is an image processing technique used to identify the boundaries of objects within images. It is particularly useful for analyzing structures within an image, such as identifying shapes, contours, or text regions. For this project, I will be implementing edge detection using the Canny Edge Detection algorithm. The algorithm is meticulously designed to achieve optimal detection with high accuracy, precise localization of edges, and minimal response (Canny, 1986).

The Canny Edge Detection algorithm calculates the gradient of the image using the Sobel operator, a technique for detecting edges by computing the intensity gradient of the image (OpenCV, n.d.-a). This operator helps determine the magnitude and direction of the steepest increase in intensity, which indicates potential edges. The gradient information is then refined through a process known as Non-Maximum Suppression, which thins out edges by retaining only the maximal gradient values in the calculated direction, thereby eliminating weaker and potential false edges (Jain et al., 1995). The final stages involve applying double thresholding to distinguish true edges from weaker ones, followed by edge tracking through hysteresis, which helps confirm edge continuity (Canny, 1986). Through this sophisticated methodology, the Canny algorithm ensures that the edges detected are not only accurate but also sharply defined, making it highly effective for edge detection.

### 4.4 Contour detection

Contour detection is a technique in image preprocessing used to identify and map the outlines or boundaries of objects within an image (OpenCV, n.d.-b). This process effectively translates a group of connected white pixels against a background of black pixels into a coherent, continuous curve corresponding to the edges of a detected object. The technique is particularly useful because contours are used as the basis for further shape analysis and object recognition. Each contour is stored as a vector of points, and these contours can be manipulated to perform more complex analysis.

Additionally, drawing lines between contours can significantly aid in visualizing relationships and structures within an image, which is especially useful for turning bullet holes into recognizable characters. This helps the OCR engine to figure out which contours are connected to each other to form one whole character (Smith, 2007).

## 5 Implementation

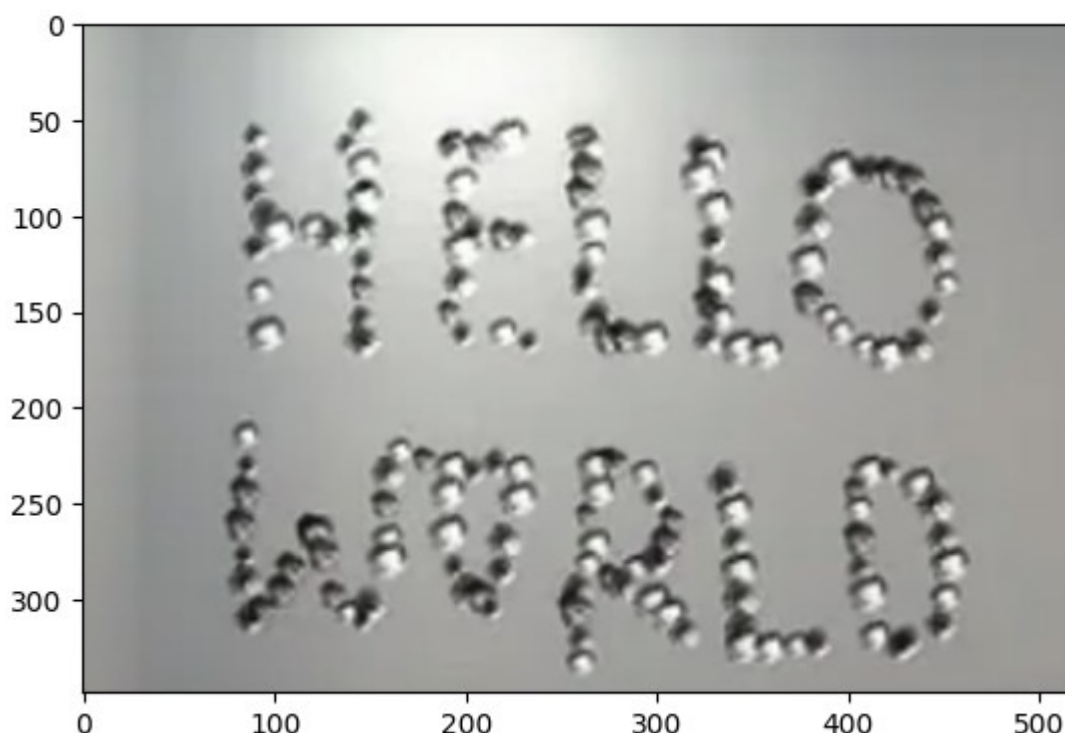
Detecting and recognizing text formed by bullet hole patterns in a game environment presents unique challenges that distinguish it from more conventional OCR tasks. In traditional OCR, text is typically static and well-defined, with consistent fonts and minimal background interference. However, in a gaming environment, the text is created through variable bullet densities and spacings, resulting in irregular patterns that are far from uniform. The complexity of the background, often featuring intricate textures, lighting variations, and other in-game elements further complicates the task, as these factors can easily obscure or distort the text.

These quirks make it difficult to develop a universally effective solution that can accurately detect and recognize text across all scenarios encountered in the game. This implementation aims to test what kinds of solutions work the best, and if there is a consistent “best” solution.

### 5.1 Image preprocessing

Throughout development of this project, I utilized test images drawn in game by shooting walls and creating words with bullet holes. Different kinds of words were used, and different ways of writing the words were used to test accuracies, and to try to mimic real life scenarios for text recognition. Figure 13 serves as a great default picture, since the text is very clear, and the background is not disruptive.

Figure 13. Hello World written with bullet holes in CS:2.



As mentioned, Figure 13 will be the default photo and the same preprocessing methods applied to Figure 13 will be applied to all other test photos, unless they really do not fit. In that case, different methods will be used. The preprocessing workflow begins with the application of a Gaussian Blur, which is crucial for reducing image noise, which is typical to appear in screenshots directly taken from games. This noise can obscure the edges needed for accurate character recognition later on. Code 1 shows the simple application of Gaussian Blurring to the test photo.

Code 1. Applying Gaussian Blurring.

```
image = cv2.imread("helloworldcsgooffice.png")
blurred = cv2.GaussianBlur(image, (5, 5), 0)
```

The OpenCV library for Python will mostly be used for image preprocessing. The library has a function that automatically applies the Gaussian blur for the desired image. In Code 1, a Gaussian blur with a 5x5 kernel is applied to the original image, effectively diminishing image noise and smoothing pixel values across the board.

In addition to the Gaussian blurring, Bilateral filtering will be added, for further refining the image clarity, especially around bullet holes. Code 2 calls the `bilateralFilter` function from the OpenCV library.



**Code 2. Applying Bilateral Filtering.**

```
bilateral = cv2.bilateralFilter(blurred, 70, 75, 75)
```

This step enhances edge definition by filtering out noise while preserving edges that are critical for later stages of image processing. After testing the parameters, the best results were found with Diameter = 70, sigmaColor = 75 and sigmaSpace = 75. The next phase of preprocessing is shown in Code 3. The phase involves adjusting the color properties of the image to better distinguish the bullet holes from the surroundings.

**Code 3. Changing the saturation of the image.**

```
hsv_image = cv2.cvtColor(bilateral, cv2.COLOR_BGR2HSV)
sat = 1.5
hsv_image[:, :, 1] = np.clip(hsv_image[:, :, 1] * sat, 0, 244)
saturated_image = cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)
```

The first line converts the image from the BGR color space to the HSV color space. The HSV model separates color (Hue and Saturation) from luminance (Value). In the third line the saturation channel is multiplied by 1.5 (sat). The np.clip function from the NumPy library for Python is used to limit the resulting saturation values to a range between 0 and 244. This prevents any pixel values from going out of the allowable range, which usually creates distortions in the image. After multiplying the saturation, the cv2.cvtColor function is used to turn the image back to the BGR color space.

Following the color enhancement, the image is converted to grayscale in Code 4. It might seem counter-intuitive to grayscale the target image, after increasing the saturation levels, but since the image was saturated, the key features will be highlighted after grayscaling. Grayscaling simplifies the image and reduces the computational complexity by limiting it to a single-channel grayscale image.

**Code 4. Applying Grayscale.**

```
gray = cv2.cvtColor(saturated_image, cv2.COLOR_BGR2GRAY)
```

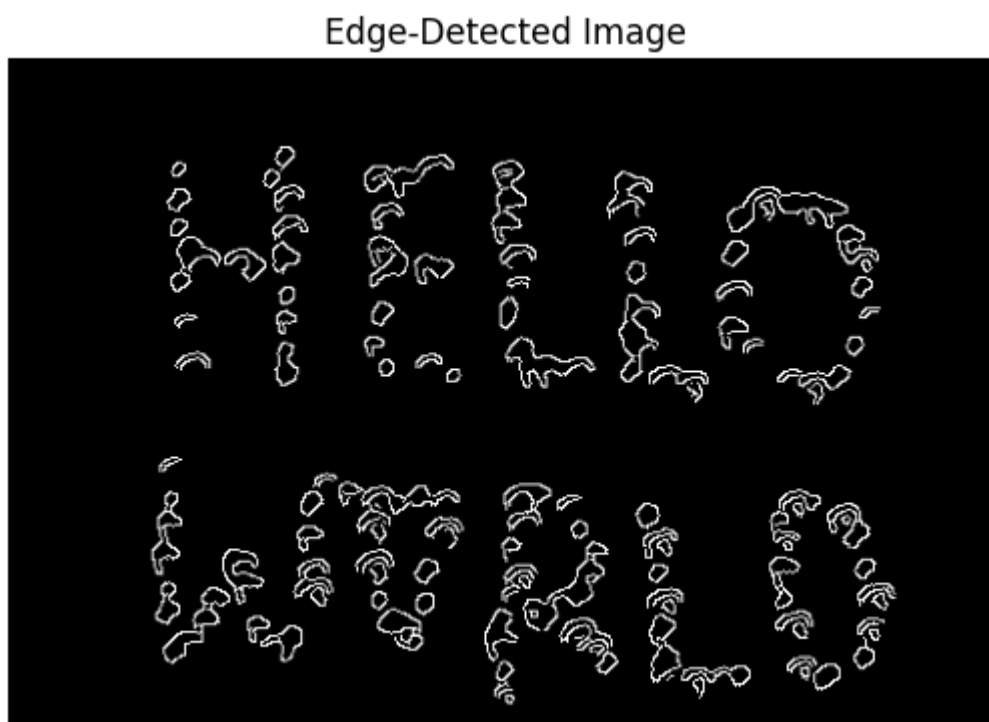
The COLOR\_BGR2GRAY function from the OpenCV can be used to add grayscaling to the target image. After grayscaling the image Code 5 implements the Canny edge detection algorithm to detect edges for each bullet hole. The Canny edge detection algorithm relies on gradients of intensity to detect edges. This is why grayscaling was applied, since the intensities are easier for the algorithm to detect in a grayscaled environment.

**Code 5. Applying Canny Edge detection.**

```
edges = cv2.Canny(gray, 50, 150)
```

Using the prebuilt Canny function from the OpenCV library. The 2 parameters are intensity thresholds. The first threshold is set at 50. This means that if a pixel gradient is lower than 50 the pixel is not accepted as an edge. On the other hand if the pixel is above the second threshold (150) it is accepted as an edge. If the pixel gradient is inbetween those two values it will only be accepted as an edge if it is connected to a pixel that is above the upper threshold, so considered as an edge. Figure 14 shows the test image with the edges of the bullet holes detected.

Figure 14. Hello World with bullet hole edges detected.



As Figure 14 shows, the image preprocessing has worked quite well, but there are odd holes inside characters. It is smart to assume this will be an issue with most cases, because as shown in Figure 13, the bullet holes were very densely shot. This does not happen in a realistic situation, so something has to be done. The holes will ruin the character recognition progress, since it will not recognize which bullet holes belong to which character.

To fix this issue, countour detection will be used. The difference between edges and countours is that countours represent a more “global” interpretation of object boundaries. Contours can be thought of as lines that join all the points along the boundary of an object. That being said, Code 6 will draw lines connecting the centroids of the contours, in order to make the characters more structured.

**Code 6. Finding each contour and drawing a line around them.**

```
contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        M = cv2.moments(contour)

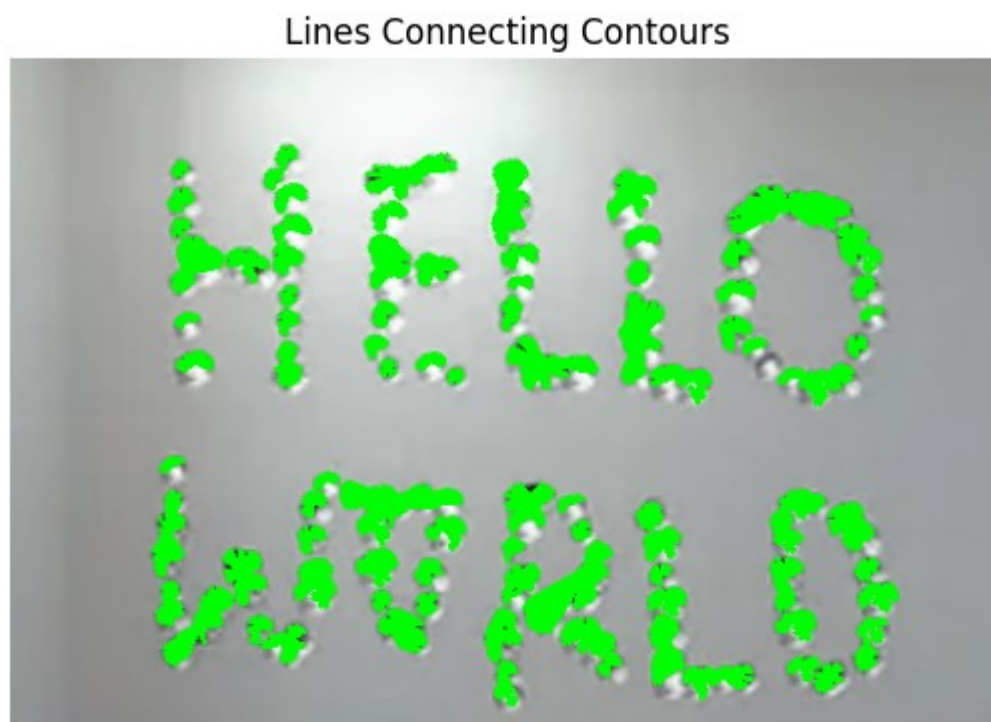
        if M["m00"] != 0:
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])

            for point in contour:
                x, y = point[0]
                cv2.line(image, (cx, cy), (x, y), (0, 255, 0), 2)
```

In the first line, the built in function in OpenCV is used to find the contours in the image where the Canny edge detection algorithm was performed. By using `cv2.RETR_EXTERNAL` the function retrieves only the external countours which is useful for getting only the outermost boundaries. The for loop processes each contour. First it starts by iterating through the contours `cv2.findContours` found. The function `cv2.moments` calculates statistical properties or moments of the contour, which are used to describe the shape and orientation of the contour. The if statement finds the centroid with calculations using the moments that the function previously found.

For each point in the contour, a line is drawn from the centroid to the point. The idea of this is to visually connect the center of the contours to its boundary, highlighting the geometric relationship. This hopefully makes the bullet holes look more connected, and helps the OCR's accuracy when detecting words. Figure 15 displays the result of contour detection.

Figure 15. Hello World with contours detected.



The contour detection works very well on the original photo. As seen, it adds chunk to each bullet hole, and connects some of the very close bullet holes together. This works well on images where the bullet holes are equally spaced out such as shown in Figure 15. As mentioned before, this is not the case always. If this method is used on an image where the bullet holes are spaced further apart it will not work well with the OCR, because each hole is far away. Figure 16 is an example of the bullet holes being further apart from each other.

Figure 16. Contours being too far apart.



As figure 16 shows, the contours are too far apart, so they do not form together into a nice connected character. This would ruin the OCR process later on, since the application would not know which bullet hole belongs to which character. Code 7 uses a new method to connect the bullet holes together.

Code 7. Calculating each contours moment.

```
centers = []
for contour in contours:
    M = cv2.moments(contour)
    if M["m00"] != 0:
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])
        centers.append((cx, cy))
```

Code 7 begins by initializing an empty list 'centers' to store the centroids of each contour. After this, it iterates through each contour. For each contour, it calculates the moments using 'cv2.moments()'. These so called moments are a statistical summary that help describe the shape's characteristics. If the zeroth moment ('M["00"]') is not zero, which represents the contour area, it calculates the centroid coordinates ('cx, cy') using the first moments. These centroids represent the geometric center of each contour. After finding the geometric center of each contour, they can be connected together as done in Code 8.

Code 8. Drawing a line through centroids of each contour in a certain distance.

```
distance_threshold = 50

for i, center1 in enumerate(centers):
    for j, center2 in enumerate(centers):
        if i != j:
            distance = np.sqrt((center1[0] - center2[0])**2 + (center1[1] -
center2[1])**2)
            if distance < distance_threshold:
                cv2.line(image, center1, center2, (0, 255, 0), 2)
```

After storing all centroids, Code 8 sets a distance threshold to define how close two centroids need to be to consider connecting them together. It then iterates through each pair of centroids. The lines are simply drawn by using 'cv2.line()'. Figure 17 shows the result of drawing lines through the contours.

Figure 17. Contour centroids connected by a line.



As shown in Figure 17, it looks a lot more connected and the OCR could potentially recognize the "Hi" from this, since it is mostly correctly connected. This method only works well when the bullet holes are not that densely shot. Figure 18 shows this method used on the default "Hello World" photo.

Figure 18. An example why connecting contours might not be a good idea.



Figure 18. shows why this method can be bad. Since the bullet holes are so densely connected, the contours already form clear characters before the centroids are connected. This just creates a big mess when everything is connected. This proves that different preprocessing methods, have to be used for different instances. Not every user will write text on walls the same way, and it is impossible to create one correct solution.

## 5.2 PyTesseract

Working with PyTesseract itself is very simple, once the image preprocessing is done. All that is needed is to import the PyTesseract Python library, and call Tesseract from its folder location and to input the right configurations for each specific instance of running it. Code 9 runs Tesseract.

Code 9. Running the PyTesseract OCR engine.

```
pytesseract.pytesseract.tesseract_cmd = #tesseract location
config = '--oem 1 -l eng'
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
text = pytesseract.image_to_string(image_rgb)

print("Recognized Text:")
print(text)
```

The first line in Code 9 specifies the file path to the Tesseract executable on my system. After this, the configuration string sets the OCR engine mode and language for Tesseract. '--oem 1' tells

Tesseract to use the LSTM-based OCR, which is generally more accurate for modern text recognition tasks. '-l eng' sets the language to English, optimizing the recognition for English text. Following this, the preprocessed image has to be turned back into the RGB colorspace, since PyTesseract requires it to function. This conversion ensures that the color data is correctly interpreted during text recognition. After this the text the OCR extracts from the image is printed.



## 6 Results

The results this program gives varies on many factors such as, bullet hole frequency, letter clarity, the background and letter spacing. These variables significantly impact the accuracy and the PSMs used for the OCR engine. In instances where the background is clear and does not have extra textures that could mislead the edge detection algorithm, the OCR engine performs optimally. The letters in Figure 18 are clearly spaced, and each bullet holes is proximate to its neighbours, allowing for the OCR to recognize the text easily. All image preprocessing techniques mentioned in this thesis except for connecting contours were used for Figure 13. Using these preprocessing steps, the 3rd and 8th PSMs for Tesseract worked the best. Code 10 shows the text Tesseract recognized.

Code 10. Recognized text from Figure 13 with optimal parameters.

```
Recognized Text:  
HELLO  
WiRLG
```

The OCR engine successfully detected the word “HELLO”, but struggled with the second word, resulting in a slight inaccuracy, but this is the best it performed. Code 11, shows an example result got with Tesseract PSMs other than 3 and 8.

Code 11. Recognized text from Figure 13 with wrong parameters.

```
Recognized Text:  
Pirie  
LV BEG
```

When given Tesseract was given any other PSM other than 3 or 8, it gave the output of “Pirie LV BEG”, which is nowhere near the target “Hello World”. This really proves the importance of having the right PSM for while performing an OCR task with Tesseract. Figure 19 has a background that does not mess with the edge detection algorithm. This makes it the “perfect” situation, and why it was chosen as the default image. Unfortunately, as mentioned before, in a realistic game situation, players can shoot anywhere in the game environment, so it is important to test how this program performs in more challenging environments. Figure 19 is the default image of Figures 16 & 17.

Figure 19. Hi written on a brick wall background.



Figure 19 shows the word “Hi” formed with bullets holes on a brick wall. The background consists of a brick texture. This could theoretically disrupt the edge and contour detection algorithms, due to the gaps in the bricks. However, the bullet holes are evenly spaced. This makes the text in the image easy to detect, but brick texture could possibly mess up the results. After testing, using all image preprocessing methods mentioned in this thesis provided the best results for Figure 19. Contour detection performed very well on this image, because it consists of only one short word unlike Figure 13, which consists of two long words. The 6th PSM for Tesseract worked the best with this image. Code 12 shows the result Tesseract output.

Code 12. Recognized text from the OCR engine for Figure 19.

```
Recognized Text:  
Hi
```

The OCR recognized the word “Hi” without any difficulty. This means that the image preprocessing functioned well with isolating the bullet holes from the background tiles in Figure 19. Figure 20 will be an even harder background.

Figure 20. "Cat" written on a very complex background texture.



Running edge detection on Figure 20 ruins the whole process, because there are too many lines that can be confused with edges. The chosen image preprocessing technique did not work well with this photo, and the OCR did not give any output with any PSM. All 3 of different cases, needed some different approach. The default photo Figure 13 worked well with just basic contour detection, and ended up being quite accurate, even though it was 2 words long. Figure 19 needed connecting contours to work properly, since the holes were too far apart, and Figure 20 wont work with the current techniques. This really proves that there is no universal answer for detecting characters from bullet holes. Each case has to be treated as its own, and automating finding the right parameters for each situation would be necessary.

## 7 Summary

This thesis introduces a novel approach to detecting toxic messages conveyed through bullet hole patterns in FPS games using image recognition methods. The project was inspired by personal experiences with such toxic behavior during gameplay, which highlighted the need for an effective detection system.

The work explored various image preprocessing techniques to enhance the accuracy of Optical Character Recognition (OCR) when detecting text formed by bullet holes. Each technique was tested individually to assess its impact on OCR performance. The findings revealed that no single preprocessing method is universally effective. Instead, the success of text detection depends on the specific scenario, the preprocessing techniques applied, and the parameters configured for the OCR engine.

Beyond the technical exploration, the thesis also assessed the broader context of toxic behaviors in online games, ranging from overt verbal abuse to more subtle forms of misconduct.

Understanding the spectrum of in-game toxicity is crucial for game developers seeking to foster a positive community environment. This research underscores the importance of developers staying connected with their player base to identify and mitigate emerging toxic behaviors effectively.

Through the development of this project, a deep understanding of image recognition techniques and their practical applications was gained. Testing various methods to determine their effectiveness was both challenging and enlightening. Additionally, reflecting on the broader issue of online gaming toxicity prompted a reevaluation of personal online behavior.

Looking ahead, several improvements could be made to enhance this project. One promising direction is the development of an AI-powered algorithm capable of dynamically determining the most effective image preprocessing techniques and OCR parameters for each detection instance. By employing machine learning, the system could analyze initial images and automatically select the optimal preprocessing methods, making it more versatile and effective in a variety of environments.

Another area for improvement is the contour connection algorithm, which could be refined to make the text formed by bullet holes significantly clearer, thereby improving the accuracy of Tesseract OCR. Once the text is detected, running it through a grammar corrector could further enhance the accuracy by correcting basic spelling errors and suggesting stylistic improvements.

To ensure the system's practicality, it is crucial that it does not interfere with gameplay by consuming excessive computational resources. A server-side implementation, combined with the use of lighter libraries such as PIL instead of OpenCV, could reduce the load and make the system suitable for real-time application. Additionally, the program should only activate under certain conditions, such as when walls are being shot for an extended period, to avoid unnecessary processing.

## 8 References

- ADL. (2021). *Hate is No Game: Harassment and Positive Social Experiences in Online Games* 2021. ADL. <https://www.adl.org/resources/report/hate-no-game-harassment-and-positive-social-experiences-online-games-2021>
- ADL. (2023). *Caught in a Vicious Cycle: Obstacles and Opportunities for Trust and Safety Teams in the Games Industry*. ADL. <https://www.adl.org/resources/report/caught-vicious-cycle-obstacles-and-opportunities-trust-and-safety-teams-games>
- Aguerri, J., Santisteban, M., & Miró-Llinares, F. (2023). *The Enemy Hates Best? Toxicity in League of Legends and Its Content Moderation Implications*. <https://link.springer.com/article/10.1007/s10610-023-09541-1>
- Almira, G. A., Harsono, T., Sigit, R., Bimantara, I. G. N. T. B., & Michael Saputra. (2016). *Performance Analysis of Gaussian and Bilateral Filter in Case of Determination the Fetal Length*. <https://riyanto.lecturer.pens.ac.id/course/performance%20analysis%20of%20gaussian%20and%20bilateral%20filter%20in%20case%20of%20determination%20%20the%20fetal%20length.pdf>
- Baheti. (2022). *Image Recognition: Definition, Algorithms & Uses*. <https://www.v7labs.com/blog/image-recognition-guide>
- Beres, N., Frommel, J., Reid, E., Mandryk, R., & Klarkowski, M. (2021). *Don't You Know That You're Toxic: Normalization of Toxicity in Online Gaming*. <https://harvest.usask.ca/items/97ed91c2-9cf2-4db8-b20b-b38e4f3261bc>
- Blizzard. (2023, April 18). *DEFENSE MATRIX - DRIVING POSITIVITY IN OVERWATCH*. Blizzard Entertainment. <https://overwatch.blizzard.com/en-us/news/23934873/>
- Blizzard. (2024a). *Received a Warning Notice in Overwatch 2*. Blizzard Entertainment. Retrieved 19 August 2024 from <https://us.battle.net/support/en/article/146535>
- Blizzard. (2024b). *Appeal Silence in Overwatch 2*. Blizzard Entertainment. Retrieved 20 August 2024 from <https://us.battle.net/support/en/article/33885>
- Blizzard. (2024c). *Overwatch 2 Competitive Play Ban*. Blizzard Entertainment. Retrieved 17 June 2024 from <https://eu.battle.net/support/en/article/125759>

Blizzard. (n.d.). *Overwatch 2 Voice Chat Recording*. Blizzard Entertainment. Retrieved 19 April 2024 from [https://us.battle.net/support/en/article/320424?accountId=363766570&flow\\_type=login\\_challenge&flowTrackingId=843ed398-72ea-4b9e-a836-f37bae4a669c](https://us.battle.net/support/en/article/320424?accountId=363766570&flow_type=login_challenge&flowTrackingId=843ed398-72ea-4b9e-a836-f37bae4a669c)

Blizzard CS. [@BlizzardCS]. (2024, May 19). You seem to have a history, which is why you have the suspension you have, [REDACTED]. While it doesn't seem entirely negative, there does appear to be some toxicity. ^JH [X Post]. X. <https://x.com/BlizzardCS/status/1791964148999377037>

Cadena, L., Zotin, A., Cadena, F., Korneeva, A., Legalov, A., & Morales, B. (2017, July 5-7). *Noise Reduction Techniques for Processing of Medical Images*. [https://www.iaeng.org/publication/WCE2017/WCE2017\\_pp496-500.pdf](https://www.iaeng.org/publication/WCE2017/WCE2017_pp496-500.pdf)

Canny, J. (1986). *A Computational Approach to Edge Detection* (pp. 679-698). ResearchGate. [https://www.researchgate.net/publication/224377985\\_A\\_Computational\\_Approach\\_To\\_Edge\\_Detection](https://www.researchgate.net/publication/224377985_A_Computational_Approach_To_Edge_Detection)

Chen, V., Duh, H., & Ng, C. (2009). *Players who play to make others cry: The influence of anonymity and immersion*. [https://www.researchgate.net/publication/220982527\\_Players\\_who\\_play\\_to\\_make\\_others\\_cry\\_The\\_influence\\_of\\_anonymity\\_and\\_immersion](https://www.researchgate.net/publication/220982527_Players_who_play_to_make_others_cry_The_influence_of_anonymity_and_immersion)

Collaboratory Against Hate. (2023, December 14). *TOXICITY DETECTION IN ONLINE GAMES*. Collaboratory Against Hate. <https://www.collabagainsthate.org/news/toxicity-detection-in-online-games>

Comidor. (2022, March 14). *6 Use Cases of Image Recognition in our Daily Lives*. Comidor. <https://www.comidor.com/knowledge-base/machine-learning/image-recognition-use-cases/>

Gank. (2023, April 6). *What is Smurfing Gaming and Why is It Controversial?* Gank. <https://ganknow.com/blog/smurfing-gaming/>

Good, O. (2022, November 17). *Overwatch 2 phone number requirement now accepts prepaid plans, letting more players in*. Polygon. <https://www.polygon.com/23465287/overwatch-2-sms-connect-account-registration-requirement-prepaid-phones>

Ibrahimovic, S. (2023, October 31). *Our search for the best OCR tool in 2023, and what we found*. Source. <https://source.opennews.org/articles/our-search-best-ocr-tool->



[2023/#:~:text=Tesseract%20is%20a%20free%20and,maintained%20by%20Google%20since%202006.](#)

Jain, A., Kasturi, R., & Schunck, B. G. (1995). *Machine Vision*. McGraw-Hill.

<https://cse.usf.edu/~r1k/MachineVisionBook/MachineVision.pdf>

Jiang, S. (2022, July 27). *Riot Suspends Valorant Pros Over Teabagging Controversy*. Kotaku.

<https://kotaku.com/valorant-riot-teabagging-esports-galorant-dawn-risorah-1849338854>

Khandelwal, N. (2023, August 25). *Image Processing in Python: Algorithms, Tools, and Methods You Should Know*. Neptune.ai. [https://neptune.ai/blog/image-processing-](https://neptune.ai/blog/image-processing-python#:~:text=Python%20is%20one%20of%20the,and%20get%20the%20desired%20output)

[python#:~:text=Python%20is%20one%20of%20the,and%20get%20the%20desired%20output](#)

KIDAS. (2024). Toxic Gaming Behaviour: Sabotaging. KIDAS. [https://getkidas.com/toxic-gaming-](https://getkidas.com/toxic-gaming-behavior-)

[sabotaging/#:~:text=Sabotaging%20in%20video%20games%2C%20also,intent%20to%20lose%20the%20game.](#)

Kundu, R. (2022, August 3). *Image Processing: Techniques, Types & Applications [2023]*. V7

Labs. <https://www.v7labs.com/blog/image-processing-guide>

Leetaru, K. (2019, July 20). *Why Is The Digital World More Toxic Than The Physical One?* Forbes.

<https://www.forbes.com/sites/kalevleetaru/2019/07/20/why-is-the-digital-world-more-toxic-than-the-physical-one/>

Microblink. (n.d.) *Image Pre-processing*. Retrieved 14 May 2024 from

<https://microblink.com/resources/glossary/image-pre-processing/>

Meyer, R. (2020). *Exploring Toxic Behaviour in Online Multiplayer Video Games*.

[https://etheses.whiterose.ac.uk/30580/1/Meyer\\_206059909\\_CorrectedThesisClean.pdf](https://etheses.whiterose.ac.uk/30580/1/Meyer_206059909_CorrectedThesisClean.pdf)

Myers, B. (2017). *Friends With Benefits: Plausible Optimism and the Practice of Teabagging in Video Games*.

<https://journals.sagepub.com/doi/abs/10.1177/1555412017732855?journalCode=gaca>

Nanda, P., Goswami, L. (2023). *Image processing application in character recognition*.

<https://www.sciencedirect.com/science/article/abs/pii/S2214785321027991>



Nogai, A. (2024). *THE PSYCHOLOGY BEHIND TOXIC BEHAVIOUR IN ONLINE GAMING*. Bytes of Balance. <https://www.bytesofbalance.com/blog/the-psychology-behind-toxic-behavior-in-online-gaming>

OpenCV. (n.d.-a). *Canny Edge Detection*. Retrieved 10 May 2024 from [https://docs.opencv.org/3.4/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html)

OpenCV. (n.d.-b). *Contours : Getting Started*. [https://docs.opencv.org/3.4/d4/d73/tutorial\\_py\\_contours\\_begin.html](https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html)

Pearson, R. (2023). *Blizzard Entertainment Kicks Off Audio Transcriptions For Voice Chat Reports In 'Overwatch 2'*. Bounding Into Comics. <https://boundingintocomics.com/2023/01/05/blizzard-entertainment-kicks-off-audio-transcriptions-for-voice-chat-reports-in-overwatch-2/>

Python Geeks. (n.d.). *Color Spaces and Conversion in OpenCV*. <https://pythongeeks.org/color-spaces-and-conversion-in-opencv/>

Ramachandran, S. (2023, April 6). *What Is Image Recognition?* Nanonets. <https://nanonets.com/blog/image-recognition/>

Reddit. (2024). *Good feelings when reports work*. [Online forum post]. Reddit. [https://www.reddit.com/r/VALORANT/comments/1cf4mrt/good\\_feelings\\_when\\_reports\\_work/](https://www.reddit.com/r/VALORANT/comments/1cf4mrt/good_feelings_when_reports_work/)

Scaler. (n.d.). *Color Spaces in Image Processing*. Scaler Topics. <https://www.scaler.com/topics/color-spaces-in-image-processing/>

Shen, C. (2013). *Network patterns and social architecture in Massively Multiplayer Online Games: Mapping the social world of EveryQuest II*. <https://journals.sagepub.com/doi/10.1177/1461444813489507>

Smith, C. (2023, December 5). *What Is OCR (Optical Character Recognition) Technology?* Forbes. <https://www.forbes.com/sites/technology/article/what-is-ocr-technology/>

Smith, R. (2007). *An Overview of the Tesseract OCR Engine*. Google Inc. <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf>

u/Local\_Failure. (2021) *Why so toxic?* [Online forum post]. Reddit. [https://www.reddit.com/r/Overwatch/comments/qao62l/why\\_so\\_toxic/](https://www.reddit.com/r/Overwatch/comments/qao62l/why_so_toxic/)

Vignesh, V. (2023). *Gaussian Blurring - A Gentle Introduction*. Medium.

<https://pub.towardsai.net/gaussian-blurring-a-gentle-introduction-e34aca1d9bbd>

Yasar, K. (2023, March). *What is image recognition?* TechTarget.

<https://www.techtarget.com/searchenterpriseai/definition/image-recognition>

**Appendix 1: Material management plan**

During the development project, a diary of my progress is kept, in which technical information about the project is collected. This information is analyzed for the thesis. The diary is stored on drive C and E of the author's computer. The author's computer is password protected. The diary is regularly backed up on Github, which only the author has access to as well. The diary is kept at station C and Github for at least one year after the completion of the thesis.

I will also add documentation to my code, so it will be easier to come back to and possibly continue development.

