BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

ANOMALY DETECTION IN COMPETITIVE MULTIPLAYER GAMES

by

LAURA GREIGE

B.S., Sorbonne Univesité, 2015
M.S., Sorbonne Université, 2017

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2022

Approved by

First Reader

Peter "Sang" Chin, PhD
Research Professor of Computer Science

Second Reader

Mark Crovella, PhD
Professor of Computer Science

Third Reader

Gianluca Stringhini, PhD
Assistant Professor of Electrical & Computer Engineering

Fourth Reader

George Kollios, PhD
Professor of Computer Science

Acknowledgments

I owe countless depth of gratitude to all the wonderful people without whom this project would not have been reality. First and foremost, I would like to express my sincerest thanks to my advisor, Professor Peter Chin for his endless support and encouragement during every stage of my PhD. I would also like to thank my committee members, Professors Mark Crovella, Gianluca Stringhini, and George Kollios for their time, attention and valuable feedback.

During my PhD, I was fortunate enough to have interned with the EADP Data & AI team at Electronic Arts and would like to extend my deepest gratitude to my team and collaborators, in particular my mentors Meredith Trotter and Fernando De Mesentier Silva as well as my managers Sundeep Naravulla and Alex Sulimanov for their endless support over the last two years.

To my parents and to my sister. I would not be writing this today if it weren't for your unconditional love and constant support of all that I do. Thank you for always encouraging me to follow my dreams and for believing in me.

Finally, to my friends and fellow PhD candidates. You've cheered me on, even through my hardest times and celebrated each of my accomplishments, and I will forever be grateful for our friendship and the memories we share together. Thank you for a wonderful and memorable PhD journey at Boston University.

Laura Greige

PhD Candidate

Computer Science Department

ANOMALY DETECTION IN COMPETITIVE MULTIPLAYER GAMES

LAURA GREIGE

Boston University, Graduate School of Arts and Sciences, 2022

Major Professor: Peter "Sang" Chin, PhD
Research Professor of Computer Science

ABSTRACT

As online video games rise in popularity, there has been a significant increase in fraudulent behavior and malicious activity. Numerous methods have been proposed to automate the identification and detection of such behaviors but most studies focused on situations with perfect prior knowledge of the gaming environment, particularly, in regards to the malicious behaviour being identified. This assumption is often too strong and generally false when it comes to real-world scenarios. For these reasons, it is useful to consider the case of incomplete information and combine techniques from machine learning and solution concepts from game theory that are better suited to tackle such settings, and automate the detection of anomalous behaviors. In this thesis, we focus on two major threats in competitive multiplayer games: intrusion and device compromises, and cheating and exploitation.

The former is a knowledge-based anomaly detection, focused on understanding the technology and strategy being used by the attacker in order to prevent it from occurring. One of the major security concerns in cyber-security are Advanced Persistent Threats (APT). APTs are stealthy and constant computer hacking processes which can compromise systems bypassing traditional security measures in order to

gain access to confidential information held in those systems. In online video games, most APT attacks leverage phishing and target individuals with fake game updates or email scams to gain initial access and steal user data, including but not limited to account credentials and credit card numbers. In our work, we examine the two player game called FlipIt to model covert compromises and stealthy hacking processes in partial observable settings, and show the efficiency of game theory concept solutions and deep reinforcement learning techniques to improve learning and detection in the context of fraud prevention.

The latter defines a behavioral-based anomaly detection. Cheating in online games comes with many consequences for both players and companies; hence, cheating detection and prevention is an important part of developing a commercial online game. However, the task of manually identifying cheaters from the player population is unfeasible to game designers due to the sheer size of the player population and lack of test datasets. In our work, we present a novel approach to detecting cheating in competitive multiplayer games using tools from hybrid intelligence and unsupervised learning, and give proof-of-concept experimental results on real-world datasets.

Contents

List of Tables

List of Figures

List of Abbreviations

The list below must be in alphabetical order as per BU library instructions or it will be returned to you for re-ordering.

ADL	.....	Anti-Defamation League
APT	.....	Adavanced Persistent Threats
BR	.....	Battle Royale
DQN	.....	Deep Q-Networks
FP	.....	False Positives
FPS	.....	First-Person Shooter
GP	.....	General Population
IDS	.....	Intrusion Detection System
IF	.....	Isolation Forests
IoT	.....	Internet of Things
LM	.....	Last Move
MARL	.....	Multi-Agent Reinforcement Learning
ML	.....	Machine Learning
MDP	.....	Markov Decision Process
MMR	.....	Match Making Rating
MO	.....	Manual Outliers
PUBG	.....	PlayerUnknown's Battlegrounds
ReLU	.....	Rectified Linear Unit
RL	.....	Reinforcement Learning
SNA	.....	Social Network Analysis
TTP	.....	Tactics, Techniques and Procedures
XP	.....	Experience Points

Chapter 1

Introduction

1.1 Background

With the rapid growth of digital gaming, there has been an increase in malicious activity in competitive games. Malicious activity can be defined as any cyber-threat, hacker or bad actor that attempts to compromise a system, exploit vulnerabilities or disrupt the gaming user experience. With games integrating more social features and relying more on connectivity, cyber-security threats such as information disclosure and cyber-attacks are more common than ever (Whitney, 2020; Tan, 2021; Owaida, 2021). In particular, web application attacks against the gaming industry grew by 340% in 2020 (Takahashi, 2021), as more people turned to gaming during pandemic lockdowns. Moreover, almost every online game includes some form of voice or text-based chatting. Players continuously interact in real-time to either coordinate or compete in multiplayer online video games, and while the interactivity is essential to game-play dynamics, it also makes the game more open to cheating and other toxic behaviors such as "griefing". Griefing, which can be defined as the act of one player intentionally disrupting another player's game experience for personal pleasure and possibly potential gain, is problematic in games as it ruins the integrity of the game and could potentially impact the relationship between the gaming company and the player community.

Many methods have since been proposed to detect malicious activity in online and competitive multiplayer games (Kunreuther and Heal, 2003; Alpcan and Basar,

2010; Gueye et al., 2012; Yeung and Lui, 2008; Galli et al., 2011). However, most approaches focus on attacks of known types and attempt to detect players using a known cheating technique. With the rise of more complex malicious activity, there has been a critical need for the development of adaptive and defending strategies and more robust detection methods.

In Machine Learning (ML), anomaly detection has been wildly used in many different domains such as fraud and malware detection, data-privacy protection and cyber-physical attacks as it helps identify malicious activity more efficiently than other software-oriented technologies. In particular, anomaly detection techniques aim to identify rare events or observations that are considered suspicious because they differ significantly from standard behaviors or patterns, i.e. data samples that do not conform to an expected behavior. We can categorize anomaly detection techniques into three main classes: supervised, semi-supervised and unsupervised techniques. The choice of the anomaly detection approach essentially depends on the availability of labeled datasets.

- **Supervised anomaly detection.** When labeled dataset is available, the most common technique for anomaly detection is typically a classification method that is trained using a dataset that has been labeled as 'normal' and 'abnormal' and automatically classifies new observations into classes according to their corresponding features. The shortcoming of supervised anomaly detection techniques is the lack of labeled datasets in real-world scenarios, and labelling data manually is an expensive and time consuming task.

- **Semi-supervised anomaly detection.** Semi-supervised techniques take advantage of large amounts of unlabeled data as well as a small number of labeled data for training. Semi-supervised learning represents a more practical setting for anomaly detection as there is a huge number of reliable normal examples available for

models to learn from with a relatively small set of anomalous data in real-world scenarios. However, the assumption that the unlabeled dataset only consists of normal observations may cause inevitable performance degradation when a small number of anomalous data is included.

- **Unsupervised anomaly detection.** With unsupervised learning, models do not learn from labeled dataset. Instead, they learn by finding structure and patterns from input features. As labeled data is relatively rare in real-world scenarios, unsupervised approaches are generally more popular for anomaly detection. With that said, since the anomalies we attempt to detect are generally specific, many outliers detected in a completely unsupervised manner could correspond to noise and may not fit the behavior we are looking to identify.

In this thesis, we analyze two major threats we find in online games and focus in particular on competitive multiplayer games. One of the major security concerns in cyber-security are Advanced Persistent Threats (APT). APTs are stealthy and constant computer hacking processes which can compromise systems bypassing traditional security measures, in order to gain access to confidential information held in that system for example. An APT attack is typically a long-term campaign; it can remain undetected in the target's system or network for an extended period of time and the persistent nature of the threats make it more difficult for the defender to protect against it. Originally used to describe intrusive activities against military organizations, APTs have evolved and are no longer limited to the military domain. As highlighted in several large-scale security breaches (Schwartz and Drew, 2011; Falliere et al., 2011; Villeneuve et al., 2013), APTs are now targeting a wide range of industries and governments, and there has been a critical need for the development of cyber-security technologies and more robust defending strategies in order to mitigate the impacts of these attacks. In online games, most APT attacks leverage phishing to

gain initial access. Phishing is a type of social engineering attack that is commonly used to steal user data, including but not limited to account credentials and credit card numbers. Typically, hackers target individuals with fake game updates or email scams mimicking the gaming company, and trick people into revealing sensitive information to the attacker. Once obtained, this information is then used to conduct fraudulent activity using their identity.

Another threat in competitive gaming is the presence of cheaters. With the increasing popularity of online games, cheating has become a common method for users to gain an unfair advantage over others. Many cheating complaints are generally associated to First-Person Shooter (FPS) games, in particular Battle Royale (BR) games, in which participants compete against one another in solo or team-based matches. Cheating in online games is accomplished in many forms, with the most common one being the use of an external cheating software that provides players with a competitive edge including aimbots, world-hacking and speed-hacks among others. The use of aimbots is a well known technique for cheating in FPS games and it allows users to automatically aim at opponents with extreme efficiency. World-hacking gives players the ability to exploit game bugs and to view more than what was intended by the developer. As an example, world-hacking may allow the user to see beyond the visibility range of game objects, through solid or opaque objects such as structures and walls. Finally, speed-hacks allow players to move and gather resources from a gaming environment faster than other legitimate players. All these cheating tools boost player performance and allow them to gain an unfair advantage that is beyond what normal gameplay would allow.

Cheating in games has many consequences on both players and video gaming companies, and players are quick to express their discontent when they find themselves in the presence of unfair competition. Therefore, leaving cheaters and trolls unpunished

negatively impacts the player base and quickly drives the majority of the community away from the game.

1.2 Challenges

Anomaly detection is one of the most common use cases of ML and it has known a lot of success in finding and identifying outliers to help prevent fraud, adversarial attacks, and network intrusions that can otherwise compromise the gaming company's integrity. Numerous methods have since been proposed to improve the identification and detection of malicious activity in these fields. However, most approaches focus on attacks of known types and well defined cheating techniques. In other words, models are designed to detect specific behaviors based on behavioral patterns from past examples (or a set of legal game rules in the case of cheating techniques), and any behavior that follows those patterns or violates those rules is considered an anomaly. For these reasons, it is important to develop systems that can learn efficiently without relying on any labeled datasets, and to consider adaptive models to automate the detection of threats in competitive multiplayer games. Unsupervised anomaly detection techniques offer a great potential in settings with imperfect and incomplete information; they do not rely on any labeled dataset and can learn to detect unknown anomalies by actively interacting with an environment. Nevertheless, our system must overcome the following major challenges.

Real-world anomaly detection generally relies on some form of human intervention. In online poker for example, participants who feel like they've been cheated against may report their suspicions, and such reports may result in a more in-depth investigation by human experts. As a community's size increases, detecting cheaters places a huge burden on the human experts in terms of responding to cheat complaints. Manually inspecting each report is clearly unfeasible with the active players

numbers in the millions. Moreover, industries cannot solely rely on reports of malicious activity that are brought to their attention by the community, as this could cause many malicious activities to remain undetected.

Anomalous behaviors posing a security threat are not commonly encountered and can take many forms. We typically have no prior-knowledge of the attacker's strategy or behavior and defining defensive strategies against stealthy attacks is non-trivial. In addition, attackers can change their strategies during an attack and anomalous behaviors may evolve due to different external factors. Hence, developing automated detection systems that continuously monitor our environment, check for any intrusion or threat and adapt to any change in behavior is necessary to minimize the possible negative impacts caused by malicious activity.

The evaluation of unsupervised anomaly detection has also been a challenging task in the research community. So far, the performance of unsupervised anomaly detection techniques has often been evaluated by using labeled data sets. In other words, the labels are not used by the algorithms during the training process, but only for evaluating their results. This method is often referred to as external evaluation approach. However, the shortcoming of the external methods is that they are not applicable to real-world problems where labeled data is often rare or not available.

Moreover, while we would like to fully automate the detection of outliers in multiplayer games, human supervision and intervention is still required when using unsupervised learning techniques in the context of cheating detection. When automating the detection, unsupervised and semi-supervised learning techniques may generate a number of False Positives (FP) in the detection. Falsely accusing an innocent player can impact players and the relationship between the developer company and the player community. Hence, it is necessary that we guarantee that no innocent player is incorrectly banned from the game. Overall, for game industries, failing to catch

certain cheaters is still a more desirable outcome then banning a player that has been falsely labeled as a cheater.

Finally, in the absence of labeled data, it becomes harder to define the boundary separating a normal behavior from an anomalous one. We can approximate optimal thresholds depending on different features, based on the distribution of gameplay data points. However, a single threshold value will not accurately separate cheaters from honest players as we usually find overlap between player behaviors in gameplay data. Larger thresholds could produce more FP in our outcome, and while smaller thresholds could generate lower false-positivity rates, they would also allow many players to continue to cheat and remain undetected.

1.3 Thesis Outline

In our work, we present novel approaches that address the above mentioned challenges in competitive multiplayer games with limited and incomplete information.

In Chapter 2, we describe a knowledge-based detection system for network intrusion in games, and show the benefits of combining solution concepts from game theory and tools from machine learning to detect anomalous behaviors and improve learning efficiency in partial-observable settings. Our model is based on understanding the technology and strategy being used by the attacker and developing good defensive mechanisms against stealthy attacks and intrusion. We describe a deep learning model in which agents successfully adapt to different classes of opponents and learn the optimal counter-strategy using reinforcement learning. We apply our model to FlipIt, a two-player security game in which both players, the attacker and the defender, compete for ownership of a shared resource and only receive information on the current state of the game upon making a move. Our model is a deep neural network combined with Q-learning and is trained to maximize the defender's time of ownership of the

resource. Despite the noisy information, our model successfully learns cost-effective counter-strategies outperforming its opponent's strategies and shows the advantages of the use of deep reinforcement learning in game theoretic scenarios. We also extend the work to a larger action-spaced FlipIt with the introduction of a new lower-cost move and generalize the model to multiplayer FlipIt.

When resources such as memory, storage and processing powers are limited, coordination is needed across multiple detection services and an efficient use of resources is necessary to maintain an acceptable level of system performance. In the second part of Chapter 2, we analyze a team-based version of the FlipIt security game, and analyze cooperative strategies. Generally, in partially observable cooperative games, agents tend to maximize their global rewards with joint actions; therefore it is difficult for each agent to deduce their own contribution. When actions are stealthy, assigning a shared global reward directly to each agent may give each agent an inaccurate reward on its contribution to the group, which could cause inefficient learning. For these reasons, we also propose a multi-agent reinforcement learning algorithm, in which a team of agents adapts to different classes of opponents and learn optimal collaborative behavior using reinforcement learning combined with tools from game theory. We apply our model to team-based multiplayer FlipIt, in which both teams of attackers and defenders compete for ownership of a shared resource. Our model is a deep neural network combined with Q-learning, using Shapley Q-value as the critic for each agent. Despite the partial observability throughout the game, our model is able to fairly distribute the global reward reflecting each agent's own contribution to their team's success, in contrast to the shared reward approach, and allows agents to learn collaborative behaviors instead of resorting to dropping out of the game and relying solely on their teammates.

Chapter 3 focuses on behavioral-based detection. With the absence of labeled

datasets, we turn to techniques from unsupervised learning and develop a system where heuristics based on user behavior are used to detect potential cheaters in real-wold competitive multiplayer video games.

As e-sports games become more popular, disruptive and toxic behaviors have become more significant. We find more players that turn to cheating in order to gain an unfair advantage, with the most common cheating techniques being the use of external cheating software and game hacks to improve a player's performance. The task of manually identifying cheaters from the player population is infeasible to game designers due to the sheer size of the player population. Cheating in games has many consequences in the gaming industry and cheaters need to be detected as quickly as possible to minimize any impact. The work we introduce in this chapter is aimed at helping and improving game designers' current workflow with the task of identifying players that are cheating in their game and focus on behaviors that are not necessarily based on the use of external cheating software, nor do they explicitly violate the rules of the game. We propose a system that is based on a player's in-game behavioral patterns, and that detects and highlights the players exhibiting such cheating behaviors in competitive multiplayer games. Our system does not take any action on the identified cheaters as it is important and necessary to be extremely careful with false positives when automating the detection; instead, the set of suspected cheaters is provided to the game designers for further analysis who, in turn, can take enforcement actions against the players, if they find it necessary. The proposed method analyzes the players' social relationships paired with their in-game behavioral patterns and, using tools from graph theory, infers a feature set that allows us to detect and measure the degree of collusion exhibited by each pair of players from opposing teams. We then automate the detection using Isolation Forests, an unsupervised learning technique specialized in detecting outliers, and show the

performance and efficiency of our approach on real-world datasets.

Chapter 4 concludes the thesis with a summary of our work and contributions and discusses future work.

Chapter 2

Anomalous Network Intrusion Detection

2.1 Game Theory in Network Security

2.1.1 Advanced Persistent Threats

With the large-scale growth of the Internet of Things (IoT) in recent years, there has been increasing needs for research in the area of cybersecurity. And with the rise in popularity of online games, and the increasing reliance in connectivity in multiplayer online games, the gaming industry has become a common target for several cyber-attacks, including but not limited to compromised servers, social engineering and identify theft. Cyber-attacks can be defined as any set of actions performed by cybercriminals, bad actors or hackers who try to gain unauthorized access, steal data or cause damage to networks. The attack can be performed by an individual working alone or by a group of cybercriminals using one or more tactics, techniques and procedures (TTPs). These individuals attempt to identify vulnerabilities, problems or weaknesses in computer systems and exploit them to achieve their goals.

Game theory has been commonly used for modeling and solving cybersecurity problems. When payoff matrices are known by all parties, one can solve the game by calculating the Nash equilibrium of the game and by playing one of the corresponding mixed strategies to maximize its gain (or symmetrically, minimize its loss). Nash equilibrium (Nash, 1950) is a solution concept that describes a steady state condition in a game, that is, an optimal outcome where no player has anything to gain by changing only their strategy. The assumption that the payoff is fully known by all

players involved is often too strong to effectively model the type of situations that arise in practice. It is therefore useful to consider the case of incomplete information and apply reinforcement learning methods which are better suited to tackle the problem in these settings. In particular, in this chapter, we examine the two-player game FlipIt (van Dijk et al., 2013) where an attacker and a defender compete over a shared resource and where agents deal with incomplete observability.

The principal motivation for the game FlipIt is the rise of Advanced Persistent Threats (APT) (Schwartz and Drew, 2011; Falliere et al., 2011; Villeneuve et al., 2013). APTs are stealthy and constant computer hacking processes which can compromise a system or a network security and remain undetected for an extended period of time. Such threats include intellectual property theft, host takeover and compromised security keys, caused by network infiltration, typically of large enterprises or governmental networks. We assume that a move by the attacker in FlipIt is a campaign that results in control over essential target resources through a breach of the system, while a move by the defender is a system-wide remediation campaign through defensive precautions. For example, for host takeover, the goal for the attacker is to compromise the device, while the goal for the defender is to keep the device clean through re-installation of critical servers or global password refresh.

In online games, most APT attacks leverage phishing to gain initial access. Phishing is a type of social engineering attack that is often used to steal user data such as login credentials and credit card numbers. Attackers usually pose as a trusted entity (such as the gaming company) and tricks an individual into clicking a malicious link in an email for example, which can lead to the installation of malware or the revealing of sensitive information as part of a ransomware attack.

Defending against APTs is a challenging task due to their stealthy structure, and the uncertainty regarding attacker moves. Traditionally, network security solutions

employ either protective devices such as firewalls or reactive devices such as Intrusion Detection Systems (IDS). However, current IDS are not very sophisticated and they rely on ad-hoc schemes and experimental work. While defenders can attempt to prevent intrusions by patching any known vulnerabilities or by closing potential network backdoors, adaptive strategies are necessary to defend against dynamic attacking strategies. As APTs are stealthy and remain persistent in the system for a long time, it is important that our system detects the intrusion as quickly as possible to minimize any potential damage caused by the attack. Additionally, limited defensive resource availability along with the performance and memory overhead imposed by the defensive mechanism on the system may require resource-efficient detection techniques.

2.1.2 Related Work

Although game theory models have been greatly applied to solve cybersecurity problems, such as intrusion detection and network security (Kunreuther and Heal, 2003; Alpcan and Basar, 2010; Gueye et al., 2012), studies mainly focused on developing defensive mechanisms to defend against well-defined attack schemes. And while this research is important, few studies have focused on situations with imperfect and noisy observations and threats of unknown types. In our work, we examine the two-player game FlipIt, first introduced by by van Dijk et al (van Dijk et al., 2013). FlipIt is the first model that characterizes the persistent and stealthy properties of APTs and is a suitable model to formally represent the strategic interactions between agents while capturing constraints on attacker incentives, defense resource allocations and attack impacts. In their paper, they analyze multiple instances of the game with non-adaptive strategies and show the dominance of certain distributions against stealthy opponents. They also show that the Greedy strategy is dominant over different distributions, such as periodic and exponential distributions, but is not necessarily optimal.

Different variants and extensions of the game have also been analyzed; these include games with additional "insider" players trading information to the attacker for monetary gains (Hu et al., 2015; Feng et al., 2015), games with multiple resources (Laszka et al., 2014) and games with different move types (Pham and Cid, 2012). In all these variants, only non-adaptive strategies have been considered and this limits the analysis of the game framework. Laszka et al. (Laszka et al., 2013a; Laszka et al., 2013b) proposed a study of adaptive strategies in FlipIt, but this was done in a variant of the game where the defender's moves are non-stealthy and non-instantaneous.

Machine Learning (ML) has also been commonly used in different cybersecurity problems such as fraud and malware detection (Buczak and Guven, 2016; Milosevic et al., 2017), data-privacy protection (Xiao et al., 2018) and cyber-physical attacks (Ding et al., 2018). It has allowed the improvement of attacking strategies that can overcome defensive ones, and vice-versa, it has allowed the development of better and more robust defending strategies in order to prevent or minimize the impact of these attacks. Reinforcement Learning (RL) is a particular branch in ML in which an agent interacts with an environment and learns from its own past experience through exploration and exploitation without any prior or with limited knowledge of the environment. RL and the development of deep learning have lead to the introduction of Deep Q-Networks (DQNs) to solve larger and more complex games. DQNs were firstly introduced by Mnih et al. (Mnih et al., 2013) and have since been commonly used for solving games such as backgammon, the game of Go and Atari (Silver et al., 2017; Silver et al., 2016; Mnih et al., 2015). They combine deep learning and Q-learning (Sutton and Barto, 2018; Watkins and Dayan, 1992) and are trained to learn the best action to perform in a particular state in terms of producing the maximum future cumulative reward. Hence, with the ability of modeling autonomous agents that are capable of making optimal sequential decisions,

DQNs represent the perfect model to use in an adversarial environment such as FlipIt. Our work extends the research made in stealthy security games with the introduction of adaptive DQN-based strategies allowing agents to learn a cost-effective schedule for defensive precautions in FlipIt and its variants, all in real-time.

2.1.3 Contributions

Our goal is to learn effectively how often should the defender clean the machines and predict when the attacker will launch its next attack. Hence, the problem can be formulated as finding a cost-effective schedule through reinforcement learning. In our work, we train our model to estimate an opponent's strategy and to learn the best-response to that strategy. Since these estimations highly depend on the information the model gets throughout the game, the challenge comes from the incomplete and imperfect information received on the state of the game. The goal is for the adaptive agents to adjust their strategies based on their observations and good FlipIt strategies will help players implement their optimal cost-effective schedule. In section 2.2, we present previous related studies and provide a description of the game framework as well as its variants. We then describe our approach to address the problem of learning in partial observability and our model architecture (Greige and Chin, 2022a). We demonstrate successful counter-strategies developed by adaptive agents against basic renewal strategies and compare their performance in the original version of FlipIt to one with a larger action-space after introducing a lower-cost move. We also generalize these findings to multiplayer FlipIt where a defender is opposed to multiple attackers.

In section 2.3, we discuss the advantages of coordinating multiple defensive mechanisms to tackle APTs in online games and introduce a cooperative and team-based extension to the multiplayer version of FlipIt (Greige and Chin, 2022b). Generally, in cooperative games, agents tend to maximize their global rewards with joint actions; therefore it is difficult for each agent to deduce their own contribution. When ac-

tions are stealthy, assigning a shared global reward directly to each agent may give each agent an inaccurate reward on its contribution to the group, which could cause inefficient learning. We propose a multi-agent reinforcement learning algorithm, in which a team of agents adapts to different classes of opponents and learns optimal collaborative strategies using reinforcement learning combined with tools from game theory. We apply our model to team-based multiplayer FlipIt, and use concept tools from game theory to efficiently and fairly distribute gains and costs to individuals in a same coalition. Our model is a deep neural network combined with Q-learning, using Shapley Q-value as the critic for each agent.

In section 2.4, we conclude the chapter with a summary of our work, and discuss future work and possible improvements to our model to encourage the ongoing research on cyber-attack detection and mitigation in online video games.

2.2 Adaptive Learning for Multiplayer FlipIt Security Game

2.2.1 FlipIt: The Game of Stealthy Takeover

FlipIt is an infinitely repeated game where the same one-shot stage game is played repeatedly over a number of discrete time periods. At each period of a game, players decide what action to take depending on their respective strategies. They take control of the resource by moving, or by what is called "flipping". Flipping is the only move option available and each player can flip at any time throughout the game. We assume that the defender is the rightful owner of the resource and as such, ties are broken by assigning ownership to the defender. Each player pays a certain move cost for each flip and is rewarded for time in possession of the resource. For our purpose we have used the same reward and flip cost for all players (attackers and defenders, adaptive and non-adaptive), but our environment can be easily generalized in order to experiment with different rewards and costs for both players. Moreover, an interesting aspect in

FlipIt is that contrary to games like Backgammon and Go, agents do not take turn moving. A move can be made at any time throughout the game and therefore a player's score highly depends on its opponent's moves. The final payoff corresponds to the sum of the player's payoffs from each round. Finally, players have incomplete information about the game as they only find out about its current state once they flip. In particular, adaptive agents only receive feedback from the environment upon flipping, which corresponds to their opponent's last move (LM).

Unless stated otherwise, we assume in the remainder of the this chapter that the defender is the initial owner of the resource, as is usually the case in real-world scenarios. The defender is considered to be LM playing a DQN-based strategy against an attacker playing one of the renewal strategies described in the following section.

2.2.2 Markov Decision Process

Our environment is defined as a Markov Decision Process (MDP). At each iteration, agents select an action from the set of possible actions $\mathcal{A}$. In FlipIt, the action space is restrained to two actions: to flip and not to flip. As previously mentioned, agents do not always have a correct perception of the current state of the game. In Figure 2·1, we oppose an LM agent P_1 and a periodic agent P_2 with period $\delta = 3$ and describe an example run with partial observability. When P_1 flips at iteration 9, the only feedback it receives from the environment concerns its opponent's flip at iteration 6. Hence, no information is given regarding the opponent's previous flip at iteration 3 and P_1 is subjected to an incorrect assumption on the time it controlled the resource. Suppose P_1 claims ownership of the resource at iteration 9. Then, P_1's benefit would be equal to the sum of the operational cost of flipping and the reward for being in control of the resource, which is represented by τ_{P_1} in the figure below.

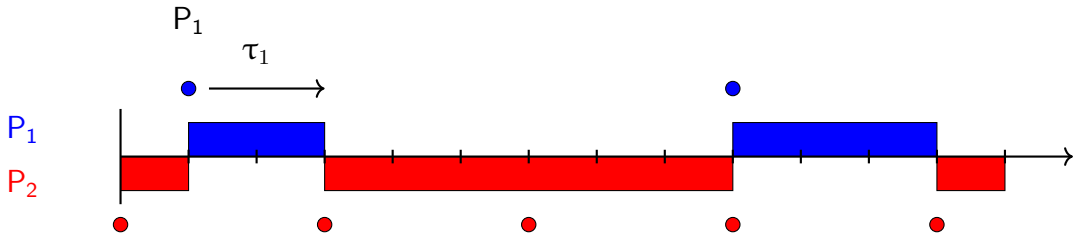


Figure 2·1: Example run with incomplete or imperfect observability in FlipIt.

• **State Space**. Players only receive information regarding the current state of the game once they flip, causing imperfect information throughout the game as previously illustrated. Since players only have partial observability, we consider a discrete state space such that each state indicates the current state of the game, that is, the times elapsed since each player's last known flips.

• **Action Space**. In the original version of FlipIt, the only move option available is to flip. An adaptive agent therefore has two possible actions: to flip or not to flip, and the action spaces are denoted by $\mathcal{A}_d = \{\texttt{idle}, \texttt{flip}\}$ for the defenders and $\mathcal{A}_a = \{\texttt{idle}, \texttt{flip}\}$ for the attackers. In the larger action-spaced extension of FlipIt, we introduce a new move called check. This move allows an agent to check the current state of the game and obtain information regarding its opponents' last known moves, all while paying a lower operational cost than the one of flipping. Just like flipping, an agent can check the state of the game at any time during a game and the defender's action space is therefore denoted by $\mathcal{A}'_d = \{\texttt{idle}, \texttt{flip}, \texttt{check}\}$. The attacker's action space remains unchanged.

• **State Transitions**. Since an agent can move at any time throughout the game, it is possible that two agents involved flip simultaneously and ties are broken by automatically assigning ownership to one of the defenders. At each iteration, the state of the game updates as follows. If any of the defenders flip, the resource is

assigned to the defender. If a set of defenders on the same team flipped at the same time step, ties are broken by randomly assigning the resource to one of the defenders from that set of players. If no defensive player flipped and the attacker flipped, the resource is assigned to the attacker. Finally, if no player flipped, the current owner of the resource remains unchanged. The transition from one state to another therefore only depends on the current state of the game and the actions taken in that state and the state transition function T is defined by $T : S \times \mathcal{A}_d \times \mathcal{A}_a \to \Delta(S)$.

- **Reward System**. We define the immediate reward at each iteration based on the action taken as well as the owner of the resource at the previous iteration.

i) **Operational Costs and Payoff**. Let r_t be the immediate reward received by an agent at time step t. We have,

$$r_t = \begin{cases} 0 & \text{if no play} \\ -C_c & \text{if } a_t = \mathsf{check} \\ \tau \cdot r - C_f & \text{if } a_t = \mathsf{flip} \end{cases} \tag{2.1}$$

where r is the payoff given for owning the resource at one time step, C_c is the operational cost of checking and C_f the operational cost of flipping. τ defines the time elapsed between the agent's last flip move and the time step he last owned the resource previous to its current flip, as described in Figure 2·1.

ii) **Discount Factor**. Let γ be the discount factor. The discount factor determines the importance of future rewards, and in our environment, a correct action at some time step t is not necessarily immediately rewarded. In fact, by having a flip cost higher than a flip reward, an agent is penalized for flipping at the correct moment but is rewarded in future time steps. This is why we set our discount factor γ to be as large as possible, giving more importance to future rewards and forcing our agent to aim for long term high rewards instead of short-term ones.

- **Renewal Strategies**. Three renewal strategy categories have been introduced and analyzed in previous research: periodic strategies, periodic with random phase strategies and exponential strategies. Players following periodic strategies $\mathcal{P}_\delta$ flip every δ iterations, regardless of the current state of the game. Players following a periodic with random phase strategies $\mathcal{P}'_\delta$ initially flip uniformly at random at time step $t \in [0, \delta]$, with subsequent flips occurring periodically every δ iterations. Finally, players following an exponential strategies $\mathcal{P}_\lambda$ are such that the intervals between two flips are distributed according to an exponential, memoryless distribution with rate λ.

2.2.3 Q-Learning Based Model Architecture

Q-learning is a reinforcement learning algorithm in which an agent or a group of agents try to learn the optimal policy from their past experiences and interactions with an environment. These experiences are a sequence of state-action-rewards. In its simplest form, Q-learning is a table of values for each state (row) and action (column) possible in the environment. Given a current state, the algorithm estimates the value in each table cell, corresponding to how good it is to take this action in this particular state. At each iteration, an estimation is repeatedly made in order to improve the estimations. This process continues until the agent arrives to a terminal state in the environment. This becomes quite inefficient when we have a large number or an unknown number of states in an environment such as FlipIt. Therefore in these situations, larger and more complex implementations of Q-learning have been introduced, in particular, Deep Q-Networks (DQN).

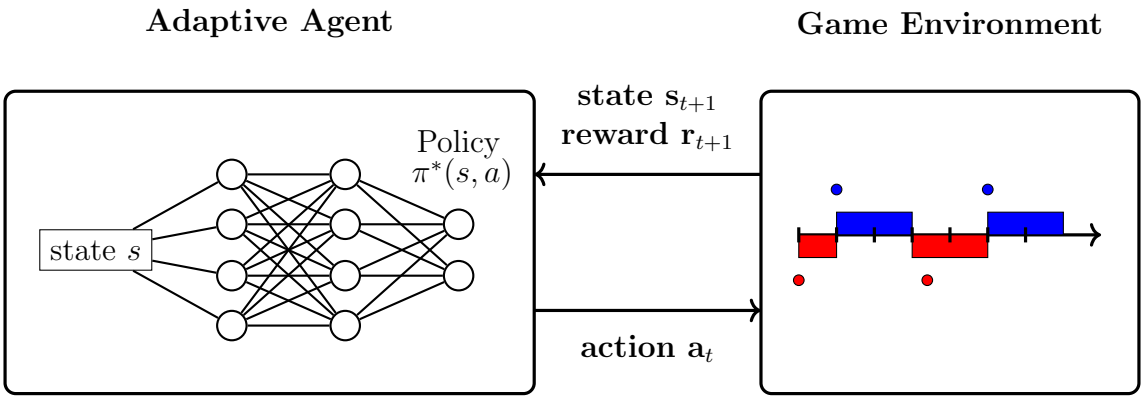


Figure 2·2: Illustration of one DQN iteration in FlipIt.

Deep Q-Networks were firstly introduced by Mnih et al. (Mnih et al., 2013) and have since been commonly used for solving games. DQNs are trained to learn the best action to perform in a particular state in terms of producing the maximum future cumulative reward and map state-action pairs to rewards. Our objective is to train our agent such that its policy converges to the theoretical optimal policy that maximizes the future discounted rewards. In other words, given a state s we want to find the optimal policy π^* that selects action a such that $a = arg\max_a [\ Q_{\pi^*}(s, a)\ ]$ where $Q_{\pi^*}(s, a)$ is the Q-value that corresponds to the overall expected reward, given the state-action pair (s, a). It is defined by,

$$Q_{\pi^*}(s, a) = \mathbb{E}_\pi \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{T-t} r_T \middle|\ s_t = s, a_t = a \right] \tag{2.2}$$

where T is the length of the game. Q-values are updated for each state and action using the following Bellman equation,

$$Q_n(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.3}$$

where $Q_n(s,a)$ and $Q(s,a)$ are the new and current Q-values for the state-action pair (s,a), $R(s,a)$ is the reward received for taking action a at state s, $\max_{a'} Q(s',a')$ is the maximum expected future reward given new state s' and all possible actions from state s', α is the learning rate and γ the discount factor.

Our model architecture consists of 3 fully connected layers with rectified linear unit (ReLU) activation function at each layer. It is trained with Q-learning using the PyTorch framework (Paszke et al., 2017) and optimized using the Adam optimizer (Kingma and Ba, 2014). We use experience replay (Wang et al., 2016) memory to store the history of state transitions and rewards (i.e. experiences) and sample mini-batches from the same experience replay to calculate the Q-values and update our model. The state of the game given as input to the neural network corresponds to the agent's current knowledge on the game, i.e. the time passed since its last move and the time passed since its opponent's last known move. The output corresponds to the Q-values calculated for each action. The learning rate is set to 0.001 while the discount factor is set to 0.99. We value exploration over exploitation and use an ϵ-Greedy algorithm such that at each time step a random action is selected with probability ϵ and the action corresponding to the highest Q-value is selected with probability $1 - \epsilon$. ϵ is initially set to 0.6 and is gradually reduced at each time step as the agent becomes more confident at estimating Q-values.

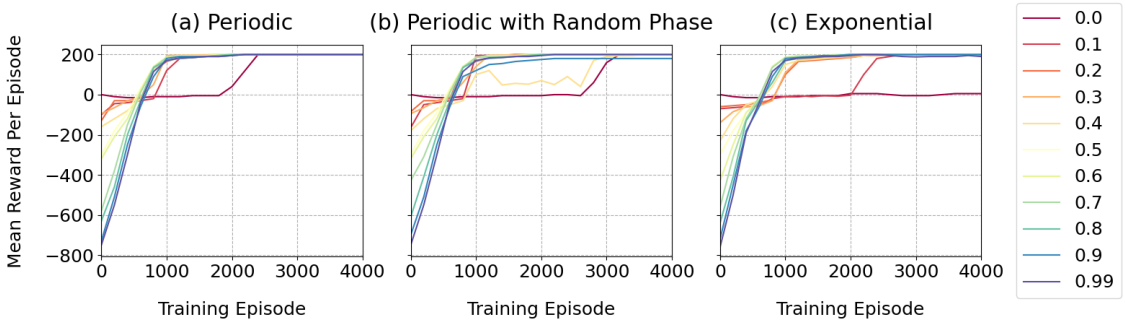


Figure 2·3: Learning overtime averaged over 10 FlipIt simulations against renewal strategies.

We choose 0.6 as it yields the best outcome regardless of the attacker's strategy. In particular, we find that, despite eventually converging to its maximal benefit, higher exploration values can negatively impact learning while lower exploration values can cause a slower convergence.

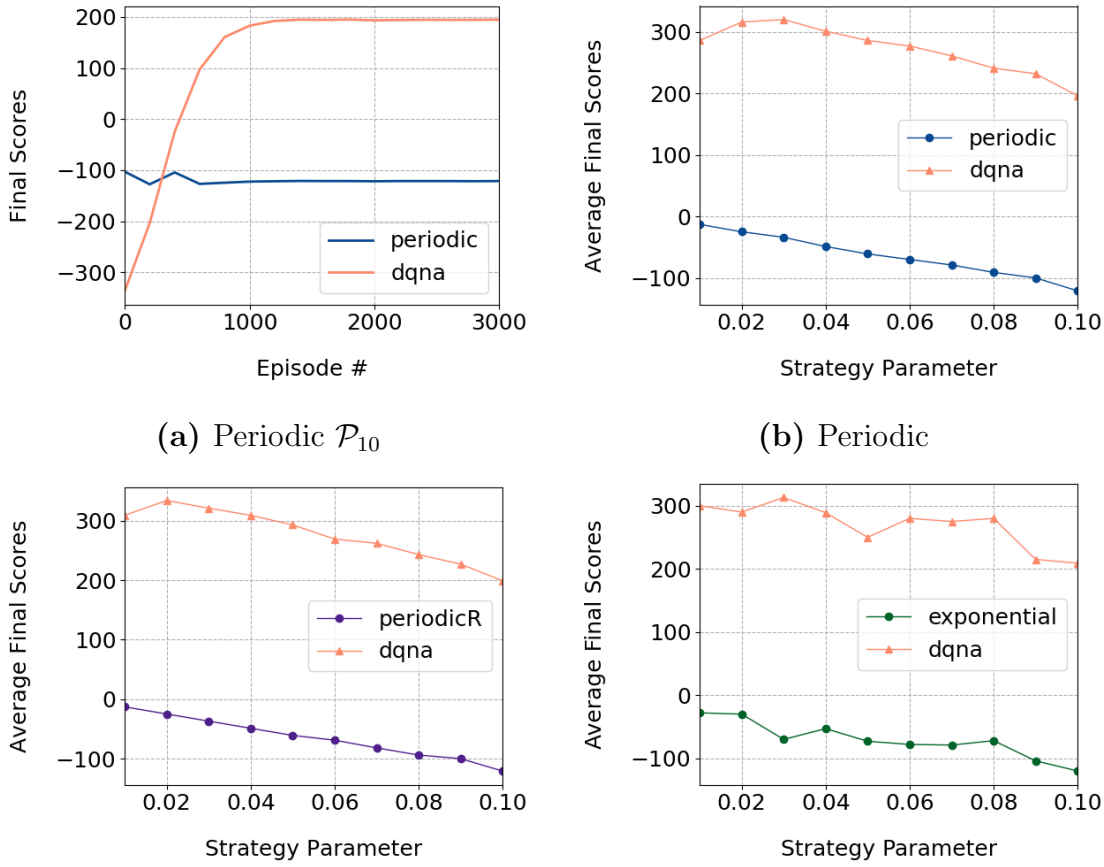
2.2.4 Experimental Results

In what follows, we assume that opponent move rates are such that the expected interval time between two consecutive flips is larger than the flip move cost. We trained our neural network to learn the best counter-strategy to its opponent's such that the reward received for being in control of the resource is set to 1 and the cost of flipping is set to 4. The flip cost is purposely set to a higher value than the reward in order to discourage the defender from flipping at each iteration. The following results can be generalized to any cost value that is greater than the reward.

Renewal Strategies

A renewal process is a process which selects a renewal time from a probability distribution and repeats at each renewal. For our purpose, a renewal is a flip and our renewal process is 1-dimensional with only a time dimension. There are at least two

properties we desire from a good strategy. First, we expect the strategy to have some degree of unpredictability. A predictable strategy will be susceptible to exploitation, in that a malignant or duplicitous opponent can strategically select flips according to the predictable flips of the agent. Second, we expect a good strategy to space its flips efficiently. Intuitively, we can see that near simultaneous flips will waste valuable resources without providing proportionate rewards. In what follows, we examine three basic renewal strategies for the attacker, periodic $\mathcal{P}_\delta$, periodic with a random phase $\mathcal{P}'_\delta$ and exponential $\mathcal{E}_\lambda$, as they present different degrees of predictability and spacing efficiency.



(a) Periodic $\mathcal{P}_{10}$ **(b)** Periodic

(c) PeriodicR **(d)** Exponential

Figure 2·4: FlipIt simulations against renewal strategies over different move rates.

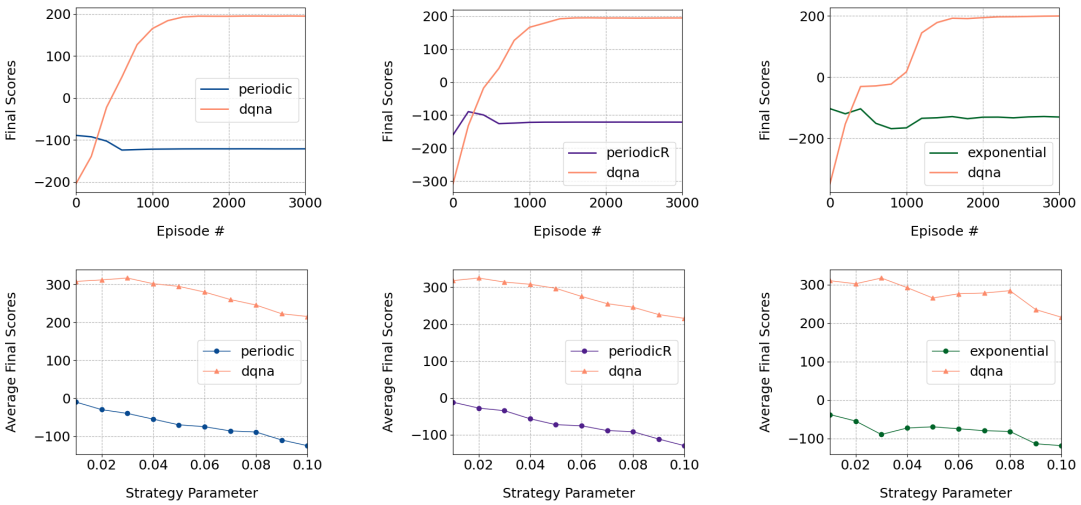
In general, the optimal strategy against any periodic strategy can be found and maximal benefits can be calculated. Since the defender has priority when both players flip simultaneously, the optimal strategy would be to play the same periodic strategy as its opponent's as it maximizes its time of ownership of the resource and reaches maximal benefit. Considering that the cost of flipping is set to 4 and each game is played over 400 iterations, then the theoretical maximal benefit for an adaptive agent playing against a periodic agent with a period $\delta = 10$ would be equal to 200. We oppose an LM adaptive agent to a periodic agent $\mathcal{P}_{10}$ and show that with each game episode, the defender's final score does in fact converge to its theoretical maximal benefit in Figure 2·4a.

In Figures 2·4b, 2·4c and 2·4d, we plot the average final scores after convergence of the defender reward against periodic, periodic with a random phase and exponential strategies, with regards to the opponent strategy parameter. All scores are averaged over 10 runs. In all 3 cases, the defender converges towards its maximal benefit and drives its opponents to negative ones, penalizing them at each action decision. It outperforms all renewal strategies mentioned, regardless of the strategy parameters, and learns the corresponding optimal counter-strategy even against exponential strategies where the spacing between two consecutive flips is random. A more in-depth look into the strategies developed shows that the adaptive agent playing against $\mathcal{P}_\delta$ and $\mathcal{P}'_\delta$ learns a strategy where the distribution of wait intervals concentrates on δ whereas the one playing against $\mathcal{E}_\lambda$ learns a strategy with a wider spread, spacing its flips efficiently throughout the game. We find that the defender's final score decreases as the attacker move rate increases. This can be explained by the fact that a higher strategy parameter suggests flipping more often and causes the defender to also flip more frequently to counter the attacker, thus causing the overall reward to decrease. Moreover, higher move rates cause shorter interval times between two consecutive

flips and this increases the risk of flipping at an incorrect iteration which could penalize the defender; as a matter of fact, in the general case, the worst-case scenario, flipping one iteration before each of its opponents flips, is only one shift away from the optimal strategy, flipping at the same time as the opponents. Despite the decreasing final scores, the defender learns to efficiently counter-attack its opponents, thus maximizing its time of ownership of the resource.

Larger Action-Spaced FlipIt Extension

We compare the defender's performance in the original version of FlipIt with one where the adaptive agent's action space is now $\mathcal{A}'_d = \{\texttt{idle}, \texttt{flip}, \texttt{check}\}$. We set the operational cost for checking the current state of the game to 1 as a way to compensate for the benefit of owning the resource at time step t and we run the same experiments against the renewal strategies in 2-player FlipIt.



(a) Periodic **(b)** PeriodicR **(c)** Exponential

Figure 2·5: Learning overtime in larger action-spaced FlipIt.

In Figure 2·5, the top figures represent the defender's reward per episode while the bottom ones represent the defender's final reward after convergence against each

renewal strategies depending on their move rates, and averaged over 10 runs. With the attacker's final rewards being in the negatives, the defender succeeds in preventing any type of intrusion, regardless of the attacker's strategy. The defender reaches optimal benefits against all renewal strategies, regardless of strategy parameter. Overall, the defender spends less on operational costs with the additional of the lower-cost move and learns just as fast and efficiently. When we cannot afford to take system-wide defensive precautions quite as often, the addition of a lower-cost actions that allow defenders to receive additional feedback from the environment can be beneficial in learning the cost-effective defensive strategy, without having to spend more in terms of move operational costs.

Multiplayer FlipIt

Finally, we extend the game to n players, to illustrate scenarios where a system is under multiple attacks. In multiplayer FlipIt, we assume that one defensive agent is playing against multiple attackers, each attacker working alone or as a group, and competing over a shared resource. The goal for the defender remains the same, and that is, to learn all attacking strategies and develop an efficient counter-strategy. As the state of the game corresponds to the agent's knowledge of the game (i.e. its opponent's last known moves), the state size increases as the number of attackers increase. We examine our model by opposing the adaptive defender to a combination of two opposing agents. Players' final rewards are averaged over 10 runs and are plotted in Figure 2·6, such that darker colors correspond to higher benefits. We illustrate the results obtained in 3-player FlipIt for a clearer visualisation. However, all findings can be extended to $(n > 3)-$players.

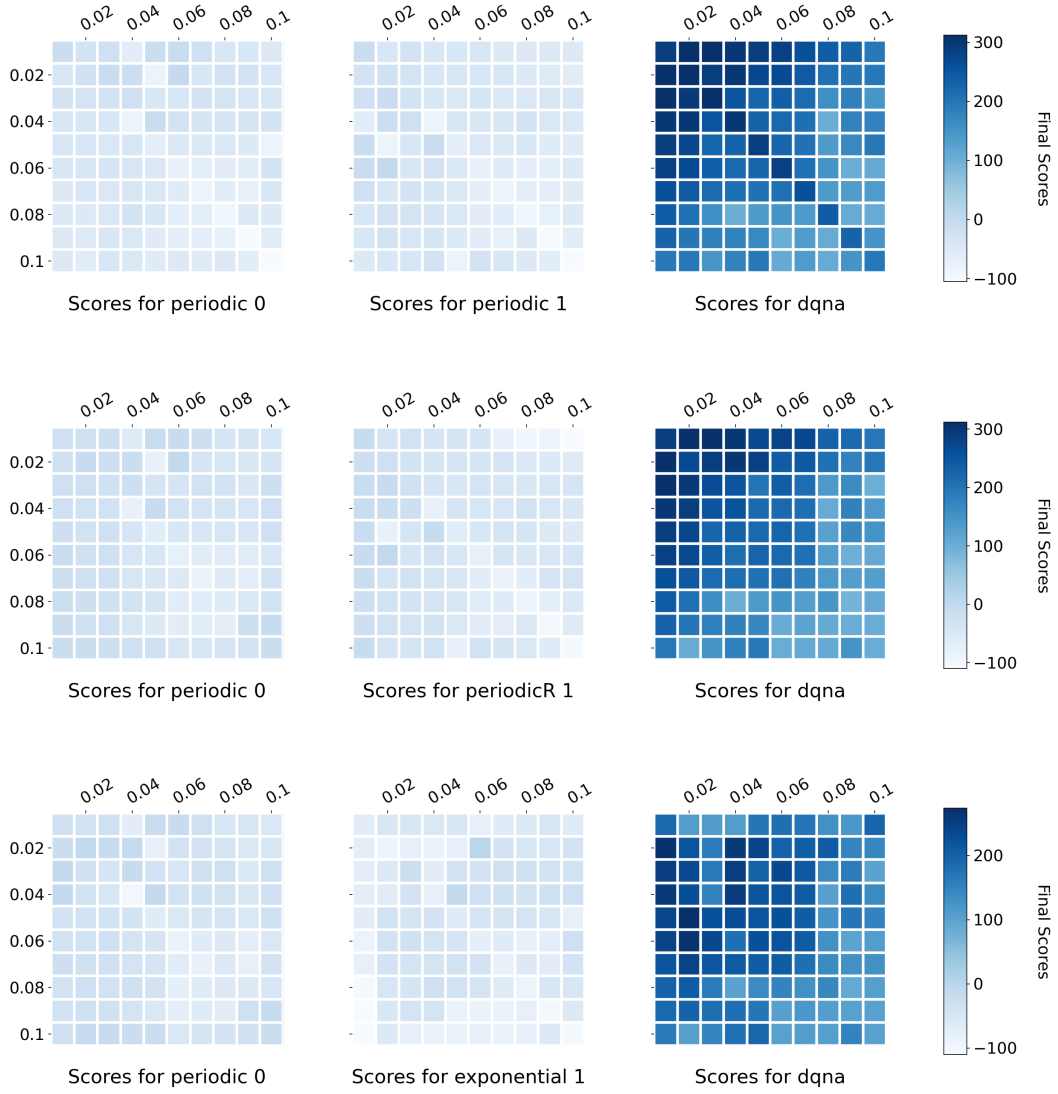


Figure 2·6: 3-Player FlipIt simulations.

As a reminder, we assume that the defender is the rightful owner of the resource and therefore has priority when assigning the resource to a new owner, in the case of simultaneous flips. Consider the case where an adaptive agent plays against two periodic agents with the same move rate δ. Then this would be equivalent to playing against one periodic agent with move rate δ, and we obtain the same results as 2-player FlipIt. Now assume both attackers have different move rates. Hypothetically,

this scenario would be equivalent to playing against an agent such that its strategy is a combination of both periodic agent strategies. An in-depth look at the strategy learned by the defender shows that the agent learns both strategy periods and spaces its flips accordingly. When opposed against a periodic agent $\mathcal{P}_\delta$ and an exponential agent $\mathcal{E}_\lambda$, the defender develops a strategy such that each two flips are efficiently spaced throughout the game, allowing the adaptive agent to converge towards its maximal benefit. As is the case in 2-player FlipIt, the higher the move rates, the smaller the intervals between two consecutive flips are which drives the defender to flip more frequently and causes lower overall final benefit. Nonetheless, the defender develops dominant strategies that yields maximal benefit, regardless of its opponents and opponent move rates.

2.3 Towards a More Resource-Efficient Learning in FlipIt

2.3.1 Cooperative Game Theory and Credit Assignment

In this section, we introduce a team-based extension to multiplayer FlipIt game to study the potential interactions between multiple adaptive defensive agents. This is useful in cases where resources such as memory, storage and processing powers are limited and coordination is needed across multiple detection services to maintain acceptable levels of system performance. Because of the persistent nature of APTs, it is also important for the system to quickly detect intrusion to minimize any potential damage caused by the attack. For these reasons, our goal in cooperative multiplayer FlipIt is to develop quick and resource-efficient defensive strategies.

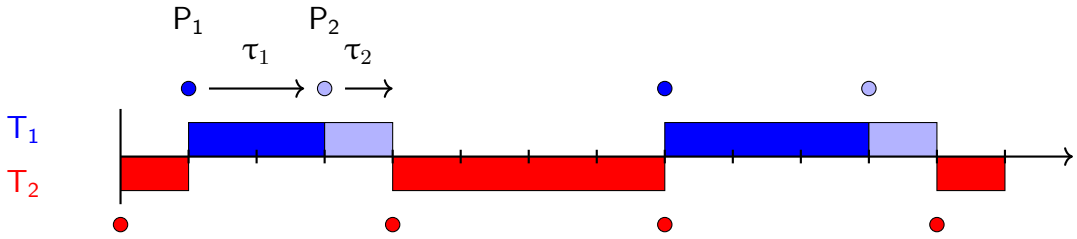
Cooperative game theory is a critical research in multi-agent reinforcement learning (MARL). It addresses the sequential decision-making problem of multiple autonomous agents that operate in an environment, each of which aims to collaboratively optimize a collective long-term payoff return. The main challenge in cooperative

settings is credit assignment, that is, the task of attributing a global shared reward to each individual agent involved in the cooperation. Ideally, in cooperative team-based FlipIt, agents would coordinate their defensive moves and learn a collaborative and resource-efficient defensive strategy against any attacker.

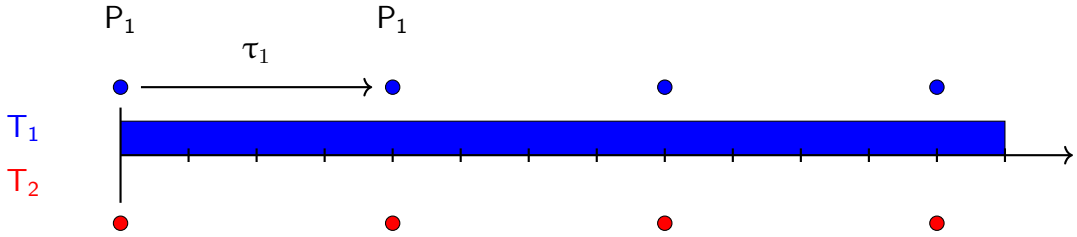
Credit assignment is an important problem studied in the global reward game. Most research on cooperative game theoretical models use the shared reward approach to force cooperation (Lowe et al., 2017; Das et al., 2018; Kim et al., 2019), that is, each agent is assigned a equal share of the global reward directly. However, this approach may give each individual an inaccurate reward regarding its contribution to the group, and hence cause inefficient learning. Similar to most multi-agent cooperation problems, we want to deduce the individual contribution of each defender to the team's success and reward them depending on their contribution. Identifying the contribution of each defender will be beneficial for more effective coordination, which could lead to better defensive mechanisms and in our work, we propose the use of Shapley values (Shapley, 1953) to distribute the global reward to each agent, reflecting accurate representation of the agent's contribution towards the team, and improve learning efficiency using DQNs.

2.3.2 Cooperation in Team-Based Multiplayer FlipIt

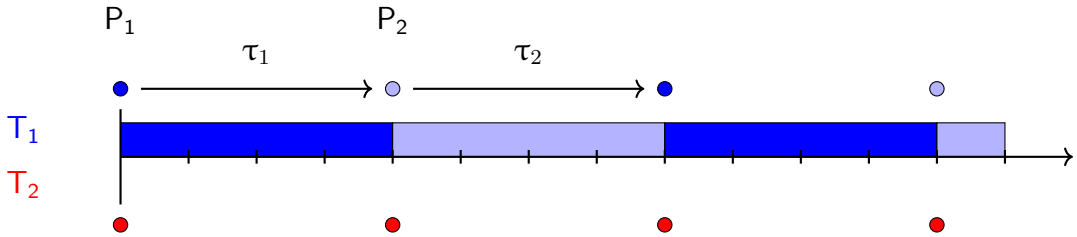
We define team-based multiplayer FlipIt such that two opposing teams of players vie for control of a resource; one team consisting solely of defenders (i.e. adaptive agents) and the other of one or more attackers adopting one of the renewal strategies. We analyze two cases; one where players share their knowledge with there respective teammates, and another where they don't communicate. In the case of knowledge-sharing, this would mean that when agent i flips and receives new information regarding the state of the game, this information is also shared with all its teammates.



(a) Example run



(b) Outcome using the model from (Greige and Chin, 2022a)



(c) Expected optimal outcome

Figure 2·7: Illustration of team-based multiplayer FlipIt opposing team 1 T_1, compromised of two defenders and team 2 T_2 compromised of one attacker.

We assume all adaptive agents to be `last-move` (LM) as defined in (van Dijk et al., 2013; Greige and Chin, 2022a). In other words, agents receive information on the current state of the game upon flipping, that is, they receive information regarding all other players' last known moves. We illustrate an example run with two opposing teams in Figure 2·7: defensive team T_1 in blue, compromised of two players P_1 and P_1, and an attacking team T_2 in red, compromised of one player adopting a periodic strategy with period $\delta = 0.25$. In particular, in Figure 2·7a, when P_1 and P_2 flip respectively at iterations 1 and 3, they receive information regarding T_2's flip at

iteration 0. However, at their next flips at iterations 8 and 11, the only information they receive regards $\mathsf{T_2}$'s flip at iteration 8 (as it is the opponent's most recent flip) and none regarding the flip at iteration 4; hence, players are subjected to incorrect assumptions regarding the opponent's strategy throughout the game. In Figure 2·7b, we illustrate the current outcome when using the approach from (Greige and Chin, 2022a) in team-based multiplayer FlipIt. While player $\mathsf{P_1}$ from team $\mathsf{T_1}$ learns the opponent's strategy and develops an optimal counter-strategy, player $\mathsf{P_2}$ develops what refer to as a "drop-out" strategy, where $\mathsf{P_2}$ remains idle throughout the game and relies solely on its teammate. In Figure 2·7c, we illustrate one possible optimal outcome where players from team $\mathsf{T_1}$ learn a cooperative counter-strategy against the attacker's strategy.

2.3.3 Shapley Q-Network Based Model Architecture

Shapley value was first introduced in 1951 and it is used to fairly attribute rewards to each player involved in a coalition, depending on their contribution to the coalition. In particular, it ensures that each player gains as much (or more) as they would have from acting independently. In other words, suppose we have a cooperative game where a set of players cooperate and collaborate towards a common goal. If we can measure the total payoff of the game, Shapley values capture the marginal contribution of each player to the end result. Let N denote the set of n players in the game. For any $S \subseteq N \setminus \{i\}$, let $\phi_i(S) = (v(S \cup \{i\}) - v(S))$ be the marginal contribution for player i. Given a coalitional game (v, N), the payout amount that player i receives is equal to the following.

$$Sh_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(n - |S| - 1)!}{n!} \cdot \phi_i(S) \tag{2.4}$$

In cooperative team-based FlipIt, adaptive agents cooperate to efficiently learn

the best counter-strategy against attackers and intruders. Our goal is for both DQNs on a same team to work and cooperate together to achieve the same goal, which is to defend the resource from any intrusion or attack. DQNs were introduced to bring the advantages of deep learning to reinforcement learning methods. They are based on a more complex implementation of Q-learning algorithm, in which an agent tries to learn the optimal policy from their past experiences and interactions with an environment. The 'Q' in Q-learning stands for quality, and it refers to the function the algorithm computes, that is, the expected rewards for an action taken in a given state. The agent therefore acts in a way that maximises this Q-value function.

To improve learning efficiency in cooperative team-based FlipIt, we use the Shapley value for reward assignment to each player. According to Equation 2.4 and given the coalitional game (v, N) and state-action pair $(s, a(i)$, the Shapley Q-value of each agent can be written as follows.

$$Q^{\phi_i}(s, a(i)) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(n - |S| - 1)!}{n!} \cdot \phi_i(S) \tag{2.5}$$

such that,

$$\phi_i(S) = Q_{\pi^*}^{S \cup \{i\}}(s, a(i)_{i \in S \cup \{i\}}) - Q_{\pi^*}^{S}(s, a(i)_{i \in S}) \tag{2.6}$$

As in multiplayer FlipIt, Q-values are then updated for each state and action using the Bellman equation described in the previous subsection.

2.3.4 Reward System

While the action and state spaces as well as the state transitions remain unchanged, the reward system differs slightly. Let $r(i)_t$ be the immediate reward received by agent i at time step t. Let $T(i)$ represent agent i's teammate. The reward function

utilized in training is as follows.

$$r(i)_t = \begin{cases} 0 & \text{if } a(i)_t = \textsf{no flip}, \forall i \in [1,2] \\ \tau \cdot r - C_f & \text{if } a(i)_t = \textsf{flip} \\ \tau \cdot r & \text{if } a(T(i))_t = \textsf{flip} \end{cases} \tag{2.7}$$

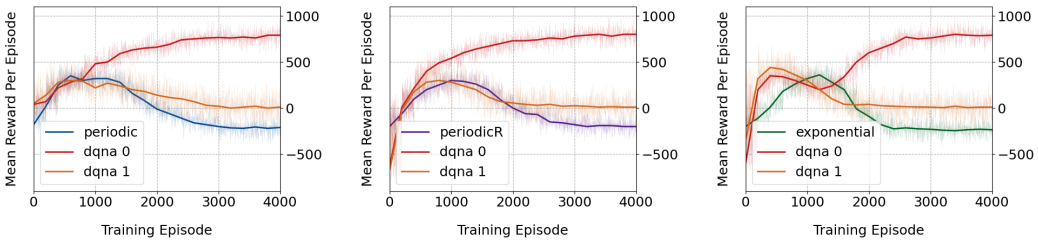
The main difference in reward function (2.7) is that an agent is also rewarded if a teammate regains possession of the resource but does not pay any operational cost for that move. In other words, we take into consideration teammate actions and reward the team for a correct move.

$$r(i)_t^1 = \begin{cases} 0 & \text{if no play} \\ \tau \cdot r - C_f & \text{if } a(i)_t = \textsf{flip} \end{cases} \tag{2.8}$$

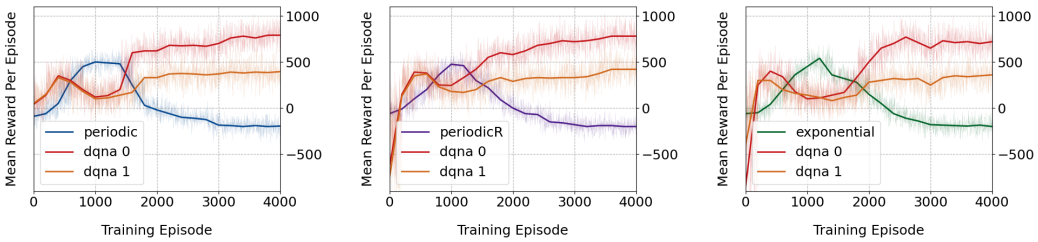
We utilize reward function (2.8) as defined in 2-player FlipIt to represent an individual player's contribution (individual time in posession, minus the operational costs spent) and gather more insights on the player's learnt strategy.

2.3.5 Experimental Results

In what follows, two adaptive agents are trained over 4000 episodes, with each game lasting 1000 steps, against one attacker adopting one of the renewal strategies described earlier. Our model architecture is the same as the one described in section 2.2.3. Learning rate is set to 10^{-4} and the remaining model hyperparameters are left unchanged. In Figure 2·8, we plot the individual rewards per episode for each agent in team-based multiplayer FlipIt.



(a) Individual mean reward for time in possession per episode during training in non-cooperative multiplayer FlipIt.



(b) Individual mean reward for time in possession per episode during training in cooperative multiplayer FlipIt.

Figure 2·8: Individual mean reward per episode during training in team-based multiplayer FlipIt, without knowledge-sharing.

First, we train teams without forcing cooperation, in other words, we update Q-values as defined in non-cooperative FlipIt, following Equation 2.3. In Figure 2·8a, we plot player's individual rewards per episode, that is, an individual player's time in possession, minus it's operational costs. While one player (DQN 0) learns an optimal counter-strategy against the attacker, their teammate (DQN 1) "drops out" of the game. With agents being rewarded in training for optimal moves by teammates, the agent's optimal strategy would be to remain idle throughout the game, and this is regardless of the attacker's strategy. This is exactly the behavior we would like to avoid, and by utilizing Shapley Q-values, we can force cooperation between teammates and learn more resource-efficient cooperative strategies.

In Figure 2·8b, we update Q-values using the Shapley equation as defined in Equation 2.5. Contrary to previous runs, both teammates contribute towards the

task in hand regardless on whether players share observations.

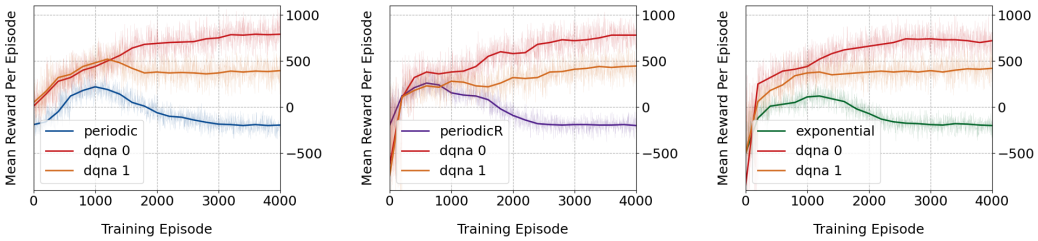


Figure 2·9: Individual mean reward for time in possession per episode during training in cooperative multiplayer FlipIt, with knowledge sharing.

Some environments require explicit communication between agents in order to achieve the best reward and learn more efficiently. We assume that players communicate and share observations, that is, when a defender flips and learns more regarding the current state of the game (e.g. opponent last known moves), this information is also shared with its teammate and their corresponding knowledge base is updated accordingly. With knowledge sharing, players can learn the attacker's strategy faster as they get a more accurate representation of the current state of the game with more frequent flips and are no longer faced to such limited information. In Figure 2·9, we plot the individual rewards per episode for each agent. Both defenders converge to higher final rewards faster and more efficiently than in previous runs. Moreover, attacker final rewards are rarely in the positive range, suggesting that the attacker fails to gain possession of the resource.

2.4 Discussion

Cyber and real-world network security threats such as APTs often present incomplete or imperfect state information. Hence, it is important to develop adaptive defensive mechanisms to deal with the stealthy and persistent properties. We believe our framework is well equipped to handle the noisy information in the game theoretic model

FlipIt and to learn efficient counter-strategies against different classes of opponents, regardless of the number of opponents. Such strategies can be applied to optimally schedule key changes to network security amongst many other potential applications. Furthermore, we extend the game to larger action-spaced FlipIt, in which we introduce a new lower-cost action that allows defenders to obtain useful feedback on the state of the game and avoid unnecessary moves and generalize the game to multiplayer FlipIt.

Any amount of feedback, even limited, received during the game about the attacker's strategy benefits the defenders. Defenders should therefore monitor their systems frequently to gain information about the attacker's strategy and detect potential attacks quickly after take over. Both monitoring and fast detection help defenders schedule moves more efficiently. This results in more control of the resource and less budget spent on moves while increasing the defender's benefit. We generalize these findings to a defender opposed to multiple intruders with different attacking strategies.

When faced with limited resources such as limited storage, memory or processing powers, it is in the interest of the defenders to cooperate and develop balanced resource-allocation strategies. We analyze a cooperative extension of the multiplayer FlipIt security game and show the advantages of cooperative game theory. In particular, we use Shapley values to assign rewards depending on an agent's contribution towards the team's success and force cooperation between adaptive agents. Through continuous interaction with the environment and coordination with teammates, agents achieve team-optimal benefits against different classes of opponents, learning faster and more efficient counter-strategies.

To successfully defend against the stealthy and persistent characteristics of network intrusion and other security threats in online games, companies must consolidate their network security infrastructure and leverage ML-based techniques to en-

sure proactive security control at scale and leverage solution concepts from GT for more resource-efficient defensive mechanisms. We believe that our framework is well equipped to automatically and continuously detect network intrusion that would otherwise blend in with normal activity, and to allow for rapid responses before attackers achieve their objective in partial-observable settings. We generalize our model to deal with multiple intrusions, and defensive mechanisms with limited resources that would need coordination to efficiently monitor the network.

Chapter 3

Bad Actor Detection

3.1 Competition in Online Video Games

3.1.1 Cheating and Exploitation

Competition has always been a major element of games. It has become more relevant in recent years with the rise of esports and the games that are at its core. This becomes even more clear with the grown interest in Battle Royale (BR) games, where dozens of players take part in a match simultaneously, trying to be the last one standing. Some of the highest grossing games since 2017, when PlayerUnknown's Battlegrounds (PUBG) was first made available, are now designed as BRs and games of such genre worth noting include Fortnite, Fall Guys: Ultimate Knockout, Apex Legends and Call of Duty: Warzone. Along with competition, particularly on environments of large popularity, there are usually those who seek to cheat, in order to get an unfair advantage. There are a number of reasons why users would cheat in a game but ultimately, they cheat to increase their chance of winning matches and becoming top-tier players.

Cheaters can generally be classified into two categories; rule-breaking cheaters and "game-breaking" cheaters. The former represents users who cheat and violate the rules of the game, but do it to to further their own status within a game without affecting another player's progression while the latter is defined as the act of one player intentionally disrupting another player's game experience for personal pleasure and possibly potential gain. Allowing cheating to go undisputed impacts the player base

negatively. Players are quick to express their discontent when in the presence of unfair competition, and leaving cheaters unpunished quickly drives the majority of the community away from the game.

3.1.2 Related Work

In order to maintain a fair competitive environment, companies have employed multiple different approaches to detect cheating. A common approach involves allowing players to report users they suspect of cheating, through an in-game interface. The game team then proceeds to review such occasions, to guarantee the validity of the report, and take action against the cheating players if they deem necessary. Common punishments include temporal and permanent ban of players, progress-removals and in-game kicking. Temporary bans and suspensions are generally used when game-rule violations cannot be fully proven while permanent ones are used on players that continuously cheat and ruin the gaming experience for other players. Progress-removal is used in games where points are being used, and players that are found to be cheating will get their score and account reset to the base value that new players are assigned when they first join the game. In-game kicking is considered a milder punishment that is usually used in less severe cases as a way to warn the player in question. Typically, anti-cheat systems decide to kick a player out of a game server as an instant punishment for unfair game play behavior. Some games also provide the community with the option to vote for a particular player to be kicked out of a game, and provides a way to effectively get rid of abusive players without depending on any other anti-cheat methods. However, despite its many benefits, vote kicking may also be used as a tool for griefing by allowing trolls to kick legitimate players out of a game.

When games are the target of common cheating methods, algorithms can also be implemented to help identify cheating. Examples include the use of machine learning techniques (Yeung and Lui, 2008; Galli et al., 2011) and third party anti-cheating

software such as Valve Anti-Cheat and Easy Anti-Cheat (Valve Software, Inc., 2002; Epic Games, Inc., 2006). While these algorithms are effective in identifying blunt, straight-forward cheating behaviors, more specialized techniques need to be developed to deal with less transparent cheating approaches.

3.1.3 Contributions

The solution we describe in this chapter is based on the real-time monitoring and analysis of players' movement and behaviour in the game, and is based on the assumption that players engaged in cheating exhibit behaviour that is significantly distinguishable from normal game play. Our work is aimed at helping game designers with the task of identifying players that are griefing in their game. This type of behavior involves analyzing multiple players in a gaming environment, doesn't involve any "super-human" actions, and is hence harder to identify.

In particular, section 3.2 focuses on collusion in team-based multiplayer games, while section 3.3 focuses on high skilled players playing in low level rankings, otherwise referred to as "smurfs". We present a novel approach to automate the detection of both behaviors (Greige et al., 2022a; Greige et al., 2022b) and give proof-of-concept experimental results on two real-world datasets. Our proposed technique does not take any action on players. Instead, the purpose is to evaluate a player's gameplay history and behavioral pattern, and with our prior knowledge and assumption on these cheating behaviors, we assign anomaly scores to each player indicating the degree of cheating exhibited by the players or their corresponding teams, in team-based settings. The game designers can then anlayze more thoroughly the group of players our technique flags as potential cheaters, to decide whether to take action on them or not. This would be a great improvement in their current workflow, where they either need to manually look through every player, which is unfeasible with the active players numbers in the millions, or only look at players that are brought to

their attention by the community, which would allow many players to continue to cheat and ruin the experience of others and lead to higher churn rate.

In section 3.4, we summarize our work on behavior-based cheating detection, and discuss the value of human intervention, even with detection automation in online video games. We also propose improvements to our detection methods.

3.2 Collusion in Team-Based Multiplayer Games

3.2.1 Background

So far, we have mentioned various methods of cheating in online video games, most of them in the form of software assistance including the use of an external software and bots. One more complex form of cheating is collusion. Collusion in multiplayer games can be described as players coordinating to purposely cooperate, to gain an unfair advantage, in a scenario in which they are posed as adversaries. A representation of such comes in the form of communicating with an adversary to join forces to undermine other opponents.

Since cheating is a strong offense in the context of the game, and one that impacts other players negatively, it is natural that game designers would take strong action against the offenders. Strong actions in this scenario would involve preventing the player from playing the game, either permanently or temporarily. This, in many cases, means blocking an user from accessing an account they invested money, either by purchasing the game or making in-game transactions, and time into. For these reasons, it is necessary to be extremely careful about false positives, there needs to be a guarantee that no user will be banned incorrectly. Overall, failing to catch certain cheaters is a more desirable outcome then to ban a non-cheater.

Collusion detection has been an exciting area of research and has been greatly analyzed in different fields including card games (Mazrooei et al., 2013; Vallve-Guionnet,

2005), online reputation systems and auctions (Liu et al., 2008; Padhi and Mohapatra, 2011), and multiplayer single-winner games (Laasonen et al., 2012; Smed et al., 2007; Knudsen, 2011; Laasonen and Smed, 2013). However, most approaches to detecting collusion use supervised learning with the assumption that large datasets containing both confirmed colluding and normal behaviors are available. In practice, such training sets are not always available, particularly in online video games. Additionally, these models are based on a colluder's past actions and behavior causing potential new forms of collusion to go undetected.

There is no current way to detect colluders in online video games, other than having game designers manually check an individual's gameplay data. Only a handful of colluders have been identified so far (less than 100), meaning there is no or not enough labeled data to be used and by consequence, we cannot rely on supervised learning.

In the next sections, we define cross-team collusion in games based on an individual's social relationships and gameplay data. The game platform allows users to befriend one another, allowing them to stay up to date with their friends' progression, share their own progression and chat about gameplay strategies. Therefore, social relationships are defined by an individual's external connections on the game platform and by its gameplay history, particularly, its in-game teammate and opponent relationships. Using a player's external and in-game social relationships paired with its in-game behavioral patterns, we select a feature set that allows us to differentiate colluding teams from the rest. Hence, the problem comes down to detecting anomalous observations in our datasets. An anomaly is defined as any data point that differs from the norm, in our case, any team behavior and interaction that differ from the rest. Colluding has a negative impact for other, non-colluding, users in online video games and draws bad reputation to video game companies through social media posts

and campaigns. Our goal is to automate the detection of colluding teams in team-based multiplayer online game. With large amounts of data at hand, data mining and AI techniques can help game designers tackle the problem of collusion detection where manual analysis and detection may be impossible. We propose a system that detects and measures the degree of collusion exhibited by all pairs of opponents, based on their behavioral patterns as well as the game designers' knowledge regarding colluding behaviors in team-based multiplayer games. Note that our system does not take any action on the detected outliers. It only identifies a set of suspected colluding teams that will still require further human investigation, giving game designers some prior assumptions on who may be colluding.

3.2.2 Methodology

In what follows, we consider a set of n players and m teams, such that $m < n$. We use real-valued vectors to describe individual player and team features such that one match opposes 20 teams with at least 2 players each, and is described by the combination of feature sets of all teams involved in that match.

Game Framework

Consider a game where teams of two players or more take part in a match simultaneously and compete in an open environment that is known by all players. The goal of the game is to be the last team standing. Teams are eliminated as the game progresses, either by other teams, or by standing outside the safe area. The safe area is the space in the environment where normal conditions apply, standing anywhere outside of it pressures players, as they are eliminated for standing out of the safe area for too long. As the match progresses, the safe area is purposely reduced over time, to force players into a more crowded space, leading to climatic moments at the end of the game, when the safe area is small and teams are forced into conflict. The game

begins with teams starting at different locations. Each team chooses the location they will start in, but their starting position is not known by the other teams. Gameplay revolves around the teams exploring the environment. As they go through the map, they are constantly keeping watch for other players, as learning the position of other teams is crucial for having an advantage over them. Another important element are items that are scattered through the map, collecting them gives the teams an edge, making it easier to eliminate other players. It is important to note that opponents have no way of verbally communicating through the game. Each team is assigned a final rank placement at the end of a match. When all players in a team are eliminated, their placement is equal to the number of teams remaining in the game plus one, with any number of players in that team yet to be eliminated. In other words, if there are n teams in a match, then the first team to be eliminated is ranked n-th, the second team to be eliminated is ranked $(n-1)$-th and so on. Match events logs are captured during gameplay and stored at the end for processing. Events provide information regarding the game state, such as players locations and actions taken throughout the match, as well as information regarding the teams (e.g. teammates, team rank placement, etc).

Data Collection

Our analysis is based on data from team-based tournament modes of an online team-based multiplayer game, including cases of colluders confirmed by the game designers. We use data from 113,060 unique matches and 173,603 unique players that have played over 3 matches over the span of 3 days. We strengthen our findings by using data from 44,097 unique matches and 128,342 unique players that have played over 3 matches over the span of 2 days, at a different time range, that includes a handful of confirmed cases of colluders. Player data is collected in accordance with applicable privacy policies and collected based on players' privacy settings and preferences. In what

follows, when we refer to the game or the environment, we imply that all previously mentioned conditions apply.

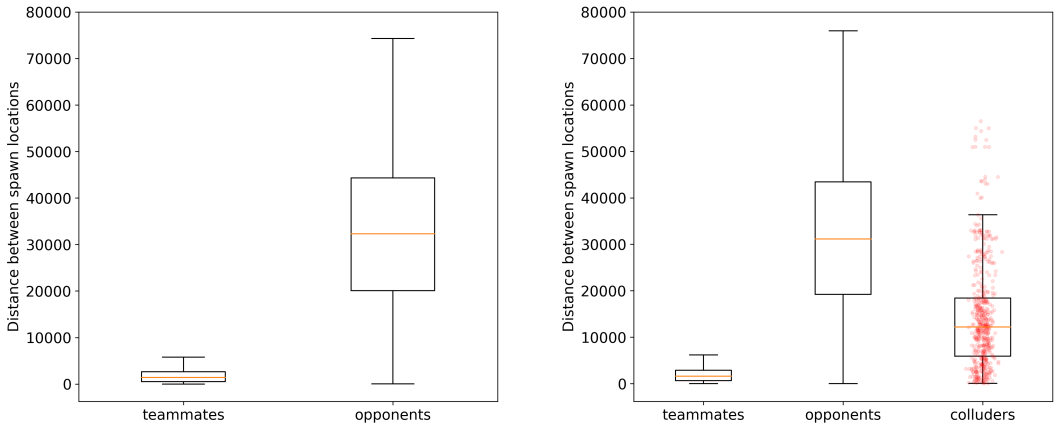
Feature Selection

We construct individual players' feature sets from their in-game attributes, as described in the game logs. We focus on attributes such as match experience (e.g. number of matches a player has participated in, final team rank placement) and play style (e.g. player's starting position in each match played). The choice of features used in our technique comes from a combination of the game designers' expert knowledge on player behaviors and what differs colluding players from non-colluding ones, but also exploration and experimentation with the feature set. Initial investigations into colluding behaviors were initiated by instances of user complaints and social media posts. As an example, video evidences showed a set of players from opposing teams deliberately choosing not to fight each other despite their close proximity, teaming up against other teams to ensure first and second place victories in ranked matches. Hence, when constructing the team feature set, we look at pairwise attributes such as the number of matches both teams participated in, their landing proximity to one another and their final team rank difference at the end of each match played. Using individual player and team features, we selected the following feature set to infer information differentiating colluding teams from the rest: landing proximity, final team rank placement, acquaintance, number of matches and number of consecutive matches.

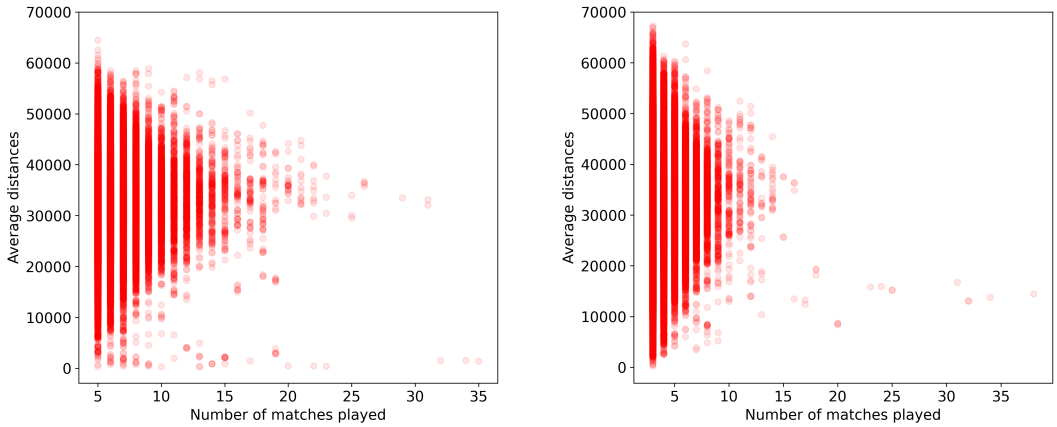
	Dataset 1		Dataset 2	
	Teammates	**Opponents**	**Teammates**	**Opponents**
Number of Pairs	171,794	347,890	95,893	65,744
Acquaintances	469		143	
Average # of matches played	14.4	4.6	9.2	4.5
Maximum # of matches played	217	35	95	38
Average distance	1,886.35	31,433.17	2,115.45	31,694.38
Average rank difference	N/A	5.19	N/A	5.36
Appeared in 3+ consecutive matches	44,229	22,472	23,056	4,196

Table 3.1: Gameplay statistics for Dataset 1 and Dataset 2.

Initial statistics are reported in Table [3.1] for both datasets. As a reminder, both datasets include data from pairs of players that have played over 3 games in the span of 3 days for Dataset 1 and 2 days for Dataset 2. The probability that two teammates appear in the same set of matches is naturally higher than that of two opponents, which explains the significantly higher number of pairs of teammates in both datasets. Moreover, it is highly unlikely to find a pair of players that not only has played over 3 games as teammates but also appeared in at least 3 or more matches as opponents, more so over such a short period of time. However, we find 469 pairs in Dataset 1 and 143 in Dataset 2 such pairs and include these values in the table under acquaintances. As stated in the previous section, we are interested in players' proximity to one another, hence we look at the average distance between each pair of players' starting positions. Values are coherent over both datasets where the average distances between all pairs of teammates are 1,883 for Dataset 1 and 2,125 game units for Dataset 2, and the average distances between all pairs of opponents are 31,928 for Dataset 1 and 31,112 game units for Dataset 2. Similarly, the average rank difference between two pairs of opponents is also coherent over both datasets with a 5.27 average for Dataset 1 and a 5.28 average for Dataset 2.



(a) Player distance distribution over all matches played for each pair of teammates and opponents from Dataset 1 and 2. For Dataset 2, we also add the distance distribution for pairs of confirmed colluders.



(b) Average distance between each pair of opponents from Dataset 1 and 2, averaged over the set of matches both opponents participated in. Each dot represents a pair of opponents.

Figure 3·1: Teammate and opponent distance distribution and distances between opponents.

- **Proximity.** For each pair of teammates and opponents, we calculate the average distance between players' initial positions in the game environment over all matches both players participated in. We plot the distribution of the average distances for all pairs of teammates and for all pairs of opponents from Dataset 1 and 2 in Figure 3·1a. We observe that the majority of teammates were in close proximity at

the start of each match, with some exceptions nearing an average distance of 10,000 game units. In contrast, the average distance between two opponents varies greatly, ranging from near 0 to over 70,000 game units. In Figure 3·1b, we plot the difference between each pair of opponents' starting positions averaged over all matches both players participated in. With over 10 match appearances, certain pairs of opponents were as close as teammates would be.

Colluding teams have an incentive to remain close to one another, which begins by pre-planning map location starting points. Two opponents being in close proximity at the start of a game does not necessarily indicate collusion, but the together with the other features, especially when analysing a pair's proximity averaged over multiple matches, is an important indicator.

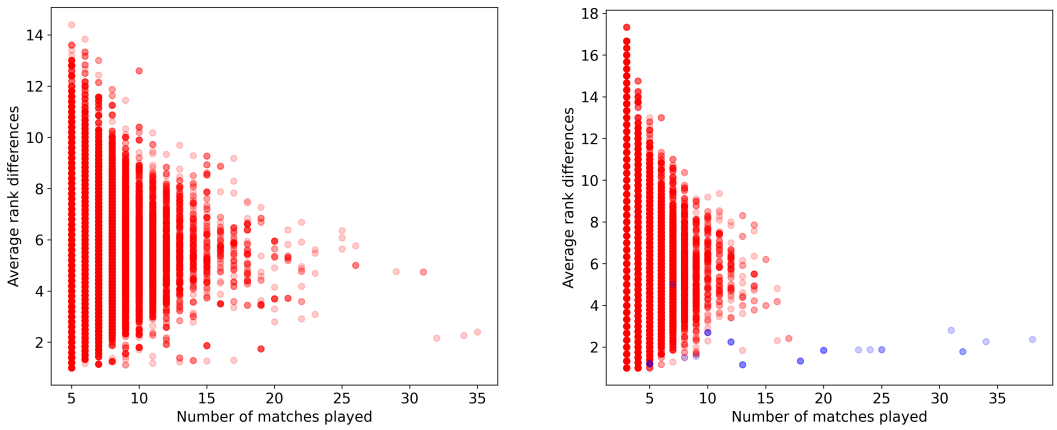


Figure 3·2: Average rank difference for each pair of opponents from Dataset 1 and 2. For Dataset 2, we highlight the pairs of confirmed colluders in blue.

- **Rank Placement.** There are a number of reasons why users would collude in games but ultimately, they collude to increase their chance of winning matches and become top-tier players, joining the highest ranks. These players have an incentive to collaborate with other top-tier players in order to aid each other in keeping their

respective status and tier ranks. For this reason, we believe that colluding teams are more likely to be closely ranked at the end of each match. For each pair of opponents, we look at their team rank placement difference over all matches both players participated in as opponents. In Figure 3·2, we plot these values for each pair of opponents. We also highlight the confirmed cases of colluders from Dataset 2 in blue.

Suppose two random teams finish one rank apart in a single match that involved 20 different teams. Then the probability of this event occurring is equal to $(2 \cdot 19 \cdot 18!)/20! = 0.1$. Therefore, the probability that two teams finish one rank apart k times in n matches would be equal to the following:

$$\binom{n}{k}(0.1)^k(0.9)^{(n-k)} \tag{3.1}$$

Now suppose two random teams finish one rank apart in a single match but also rank in the top 10 spots out of 20 spots. The probability of this event occurring is equal to $(2 \cdot 9 \cdot 18!)/20! = 0.047$. Therefore, the probability that the same two teams finish in the top 10 spots and finishing one rank apart in n different matches is equal to the following:

$$\binom{n}{k}(0.047)^k(0.953)^{(n-k)} \tag{3.2}$$

To give a better idea, suppose both teams finished in the top 10 spots and one rank apart in 3 matches out of 5. Then the probability of this event occurring is equal to 0.00094. Despite the very low probability of this event occurring, we find 6,771 pairs of opponents in Dataset 1 and 26,370 pairs of opponents in Dataset 2 ranked in the top 10 out of 20 teams, with an average rank difference less than or equal to 2 out

of over 110,000 and 290,000 unique pairs of opponents respectively. Note that these values represent pairs of opponents on different teams and not individual teams.

- **Acquaintance.** To coordinate a collusion in a game, we believe that players involved are acquaintances. For this reason, we look at a player's social relationships. Social relationships are defined by the connection between two individuals on the game platform which allows users to befriend one another, and by their gameplay history, particularly, their in-game teammate and opponent relationships.

We find 469 pairs of players that have played both as teammates and as opponents in at least 3 games in Dataset 1 and 143 pairs in Dataset 2. There is no sure way to face a predetermined opponent in the game and players are matched at random with players of similar skill levels. Therefore, it is generally highly unlikely for two players to end up as opponents in multiple matches unless some collusion such as communicating outside the game to coordinate playing at the same time and on the same game server, occurred prior to the matches, increasing their chance of being assigned to the same set of matches.

- **Number of Matches and Consecutive Ones.** Finally, we look at the number of matches and consecutive matches each pair of opponents has participated in. We assume that colluding teams are more likely to have participated in multiple matches with the same teammates and/or opponents. The higher the number of matches two players appeared in, the more likely it is that these players agreed on playing the same set of matches. We are also more interested in players that constantly collude, ruining the user experience for other players. We are not so much concerned about short-term and temporary colluders, as they only temporarily disrupt the experience for the rest of the players. Therefore, in what follows, we only consider pairs of players that have participated in at least 5 matches together for Dataset 1 and in at least 3 matches for Dataset 2. Finally, we also look at the num-

ber of consecutive matches two opposing teams participated in as a higher number of consecutive matches gives more insight on whether appearing in the same set of matches was premeditated.

		Acquaintance	Rank Difference	Max # Consec Games	Proximity	# Matches	Anomaly Score
Dataset 2	A	TRUE	1.18	2	1971.32	11	-0.173
	B	TRUE	1.11	2	984.15	9	-0.166
	C	TRUE	1.45	2	6573.93	11	-0.162
	D	TRUE	3.44	2	18137.61	18	-0.156
	E	TRUE	3.44	2	18389.09	18	-0.156

Table 3.2: Gameplay data for 5 confirmed pairs of colluders from Dataset 2.

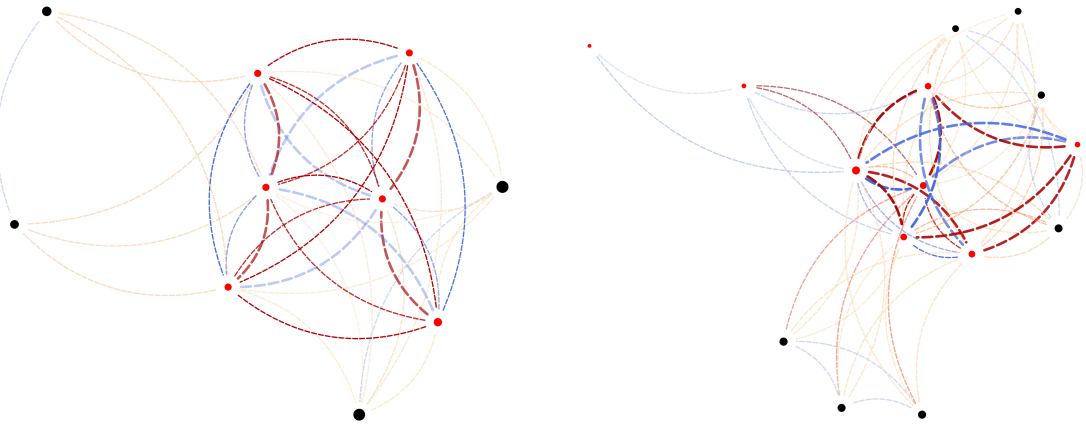
Each individual feature is naturally not enough to determine whether two or more teams have been colluding. As an example, we report the gameplay data for 5 pairs of opponents confirmed to be colluding by the game designers in Table 3.2; column `Acquaintance` is set to TRUE if the pair had previously played as teammates at least once, column `Rank Difference` reports the average difference in rank placements when playing as opponents, column `Max # Consec Games` is the maximum number of consecutive games the pair of players appeared in as opponents, column `Proximity` is the average distance in game units, between players' initial landing positions (with the map area being roughly 10^{10} units^2 and character movement speed 350 units/second) and column `Number of Matches` is the total number of matches the pair played as opponents. Column `Anomaly Score` is the score given to the pair by our technique, such that the lower the score is, the more abnormal the pair's behavior is. Pairs D and E, although relatively close in terms of average final team rank placement and landing proxmity, they do not exhibit the same behavioral pattern as the 3 other pairs. They do however appear as opponents in the same set of matches more frequently than the first 3 pairs. This is why it is important to consider a combination of the different features to better differentiate colluding behaviors from the norm.

Social Network Analysis

In the following sections, we study the relationship between teammates and opponents using social network analysis and explore the efficiency of our model on real datasets while showing its performance on confirmed cases of collusion.

So far, we have analyzed the behavior of pairs of teammates and pairs of opponents with a focus on their gameplay history. We also aim to understand the connections between players from different teams in different matches. Social network analysis (SNA) is one tool that could potentially enhance our understanding of the relations that develop between different teams throughout the matches played, and that could indicate collusive behaviors. SNA uses concepts from graph theory to calculate metrics representing the nodes, the connections or the network itself such as the distance to other individuals in the network or the number of interactions with other individuals. These metrics are important in quantifying interactions between players or teams of players as well as identifying potential clusters in the network. Moreover, network analysis provides a visual model for analyzing and evaluating teammate and opponent relationships.

For clarity purposes, we only plot edges between players that have appeared in over 3 matches together. Each node in the social network represents an individual player and the larger the node size, the more matches that user has played. Teammate relationships are represented by blue edges while opponent relationships are represented by edges of colors ranging from yellow to maroon such that the darker the color of the edge is, the closer the two players are in terms of rank placements averaged over all matches appeared in together. The darker the opacity of an edge is, the higher the number of games the two users have appeared in. Similarly, the thicker the edge is, the higher the number of consecutive games the two users have appeared in.



(a) Detected colluders in Dataset 1 **(b)** Confirmed colluders from Dataset 2

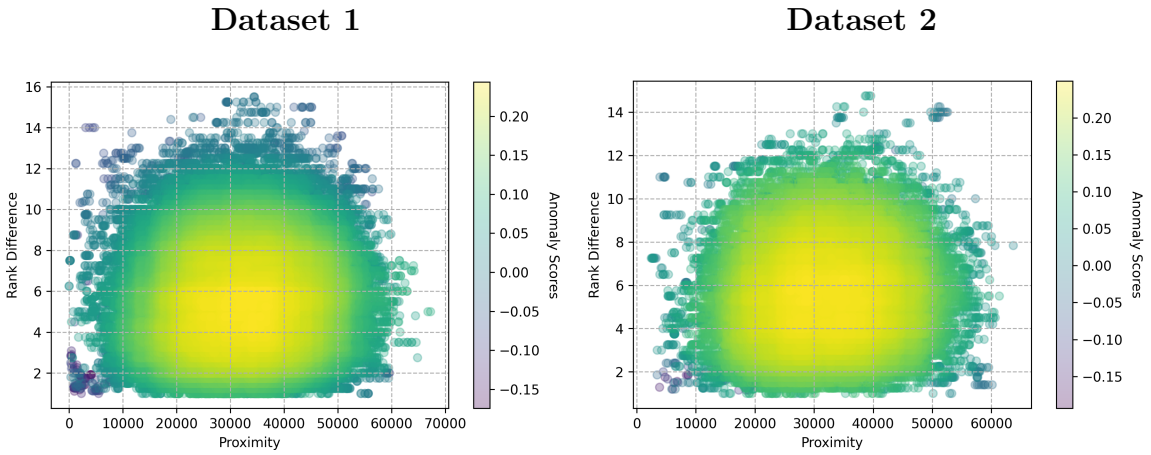
Figure 3·3: Social network illustrations of a set of players from both datasets. Suspected and confirmed colluders are in red.

A number of clusters get formed and two examples are shown in Figure 3·3. Figure 3·3a illustrates an example of a set of suspicious players as defined in the previous section, while Figure 3·3b illustrates an example of a set of confirmed colluders, all represented by red nodes. As stated earlier, the thicker and the darker the edge, the more significant the interaction was between both players. The set of players represented in Figure 3·3 are strongly connected in the social networks, either in terms of close team rank placements (red colored edges) or in terms of game appearance (darker opacity). Each pair of suspected or confirmed colluders has either played as teammates, as opponents or as both, suggesting pre-established acquaintance. The only main difference between both sets of players is that the set of confirmed colluders from Dataset 2 appeared more frequently as teammates as opposed to the suspected colluders. We obtain a number of different clusters that are not shown in this thesis, that exhibit similar behavior.

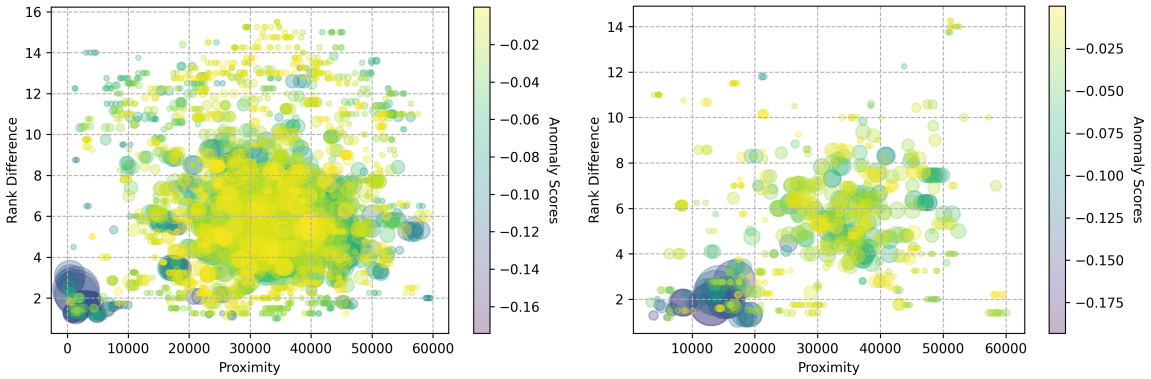
3.2.3 Experimental Results

Automating Detection

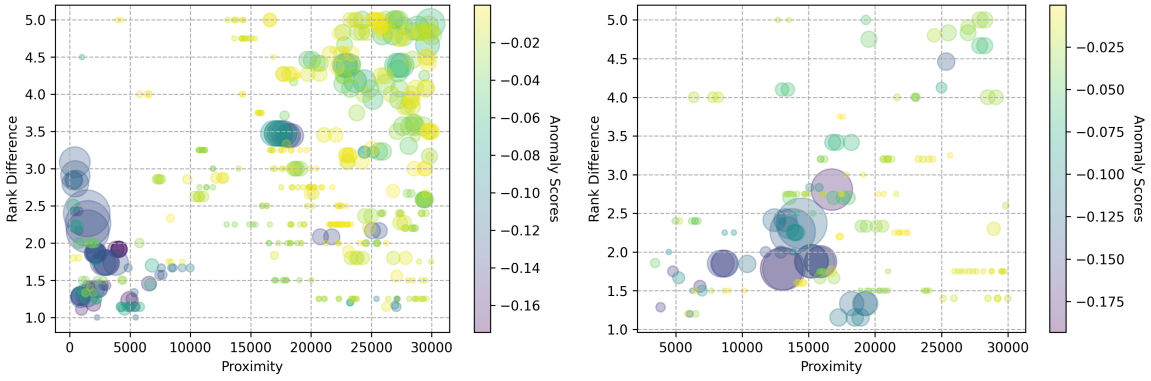
Isolation Forest (Liu et al., 2008; Pedregosa et al., 2011) is an unsupervised learning algorithm used to detect outliers and to identify anomalies instead of normal observations, that is to identify rare events that deviate from the norm. Isolation Forests (IF) are tree-based algorithms built around the theory of decision trees and random forests. IF separates the data into two parts by randomly selecting a feature from the given feature set and then randomly selecting a split value between the minimum and maximum values of the selected feature. The random and recursive partition of data is represented as a tree such that the end of the tree is reached once each data point is isolated. Outliers need fewer random partitions to be isolated from the rest of the dataset, therefore the outliers will be the data points which have a shorter average path length from the root node on the isolation tree. In what follows, we choose 100 estimators with 1000 samples where each base estimator is trained on the features mentioned in the previous section. We do not have an accurate estimation of the number of actual colluders present in our dataset, but, for comparison, game designers have a rough estimate that, in the game, there are 500 unique colluders every month with roughly 2000 user reports of others being suspicious of collusion behavior and Dataset 1 covers a span of 3 days, while Dataset 2 covers a span of 2 days. Given a dataset of players and matches, combined with individual player and team features, each pair of opponents is assigned an anomaly a score indicating the degree of collusion exhibited by the pair's respective teams.



(a) Visual representation of each data point in both datasets.



(b) Visual representation of flagged outlier in both datasets.



(c) Zoom in view of the bottom left corner clusters.

Figure 3·4: Visual representation of opponent rank differences with regards to their average proximity to one another.

In Figure 3·4a, we plot visual representation of each pair of opponent's rank differ-

ence with regards to their average proximity to one another. Each node represents a pair of opponents in the dataset and each pair of opponents is given an anomaly score which indicates the degree of collusion exhibited such that negative scores represent outliers and the lower the score; the darker the node color in the scatter plots, the more abnormal the pair of opponents exhibit.

In Figure 3·4b, we only plot the flagged outliers such that the bigger the size of the node, the more games both opponents appeared in. In the bottom figures, we show a zoomed in version of the scatter plots for the cluster of outliers in the bottom left corner, i.e. pairs of opponents with a lower average rank placement difference and closer proximity. In both datasets, the pairs of opponents that exhibit the highest degree of collusion are clustered in the bottom left corner of the plots as shown in Figure 3·4c, where both the rank difference and proximity are low. We confirm that all suspected colluders mentioned in previous sections as well as the confirmed colluders from Dataset 2 were all identified by the isolation forest and a more detailed look into their behavior is reported in the next subsection. While some pairs of opponents are flagged as outliers, they are not necessarily colluders. In Dataset 2, the isolation forest identifies the pairs with a rank difference higher than 14 as outliers (cf. Figure 3·4b; Dataset 2), and while that is true, they do not exhibit any of the features mentioned earlier a part from being somewhat in close proximity. The biggest feature that differentiates these pairs from the rest is the high average rank difference, which is most likely why they have been identified as outliers. As mentioned in section 3.2.2, it is important to note that the algorithm does not take any action on the detected outliers, as it simply flags potential colluders. Scores are provided to the corresponding game team and designers to facilitate the detection of collusion and further investigation would be required by human experts before taking any action against these players.

Evaluation

Collusion is still a relatively recent behavior emerging in team-based multiplayer games and with the lack of a training set, we can manually verify each outlier flagged through our existing knowledge on colluding behaviors. Prior knowledge characterizes colluding behaviors similar to ones of teammates, i.e. indication of prior established connection by either having played as teammates before or by appearing in the same set of matches as opponents, close proximity to one another and close in terms of final rank placements. Each pair of opponents is assigned an anomaly score which indicates the degree of collusion exhibited by the pair of players, such that negative scores represent outliers. The lower the score, the more abnormal the behavior.

Out of the 288 outliers detected in Dataset 1, 42 were acquaintances. In particular, out of the 20 pairs of opponents with the lowest anomaly scores, 15 of them had previously played as teammates. Moreover, out of the 78 pairs of opponents flagged as outliers that also played over 10 matches, 42.3% of them averaged a rank difference of less than 3 ranks. Looking specifically at all four features mentioned in this section, out of the 288 outliers, 72 pairs of players played over 6 games as opponents, had an average proximity of less than 8000 units, and a rank difference of less than 3 over all games played. 65 of those pairs also appear in the top half of outliers with the lowest anomaly scores. Similarly, out of the 56 outliers detected in Dataset 2, 13 were acquaintances such that 9 of those pairs appear in the top 20 pairs of opponents with the lowest anomaly scores. Although the outliers detected in this dataset did not show as close as an average proximity as the outliers in Dataset 2 (average proximity of all 56 pairs of opponents was 10,000 game units), they did show a stronger indication of prior communication, with 16 pairs of opponents appearing in the same 3 or more consecutive games and 24 of them appearing in over 10 matches as opponents. Finally, 42 out of the 56 outliers were less than 3 rank placements apart, averaged over all

matches played.

		Acquaintance	Rank Difference	Max # Consec Games	Proximity	# Matches	Anomaly Score	Colluders
Dataset 1	A	TRUE	1.18	2	1971.32	11	-0.173	TRUE
	B	TRUE	1.11	2	984.15	9	-0.166	TRUE
	C	TRUE	1.45	2	6573.93	11	-0.162	TRUE
	D	TRUE	3.44	2	18137.61	18	-0.156	FALSE
	E	TRUE	3.44	2	18389.09	18	-0.156	FALSE
Dataset 2	A	TRUE	1.78	3	12982.80	32	-0.184	TRUE
	B	TRUE	1.88	3	15201.66	25	-0.157	TRUE
	C	TRUE	1.875	3	15910.2	24	-0.154	TRUE
	D	TRUE	1.56	2	6825.69	9	-0.149	TRUE
	E	TRUE	1.2	2	6072.28	5	-0.13	TRUE

Table 3.3: Top 5 outliers with the lowest anomaly scores detected in both datasets studied.

We report the gameplay data for the top 5 pairs of opponents exhibiting the highest degree of collusion in both datasets in Table 3.3 (i.e. pairs with the lowest anomaly scores). Column `Colluders` is set to TRUE if the pair of suspected colluders were confirmed to be in fact colluders by the game designers, FALSE otherwise. This is done by manually checking players' gameplay data for each game play during the corresponding date range. We give a more in-depth description of each pair's behavior in what follows.

We confirm with the game designers that 14 out of the 20 most abnormal pairs from Dataset 1 were most likely to be colluding, and 16 out of the first 20 most abnormal pairs from Dataset 2 were all confirmed colluders. According to the game designers, the biggest factors in deciding whether two teams are in fact colluding are the number of consecutive matches and repeated close final rank placements. The probability for this to occur is very low to be a coincidence over a number of consecutive matches or a high number of matches in such a given short time frame. That said, pairs D and E from Dataset 1, although exhibiting a suspicious behavior by appearing in multiple consecutive matches on a more extended date range, could not be confirmed to be colluders due to their disparate placings throughout the game.

When collusion is clearly occurring, game designers spend approximately 3 min-

utes manually checking opposing team's gameplay data. In less obvious cases, game designers need to search through more matches (as was the case with pairs D and E from the previous example), which can significantly increase the time required to detect colluding teams. By isolating the outliers in our dataset, we help game designers reduce the pool of players needed to be manually investigated, focalizing only on the players exhibiting the highest degree of collusion in our dataset.

3.3 Smurfs in Competitive Multiplayer Games

3.3.1 Background

In this section, we focus on high-skilled players in low-level rankings, otherwise known as "smurfs". The term smurf is used to, among other things, identify skilled players that create new accounts to purposely play in a skill bracket that is below their actual level and be matched against novice players to gain a competitive advantage. Matchmaking in online video games prioritizes assigning novice players with low level accounts in the same match. The premise is that the longer you play the game, the more skilled you get and the more levels you gain on your account. However, players have access to the exact same resources during a match, regardless of their account levels. In games where account creation is free, players can easily create new accounts to get matched with lesser skilled players. This affects the gaming experience for novice players, who have to fight in a scenario where they are at an important skill disadvantage. Aggregated metrics show that the number of player reports that indicate smurfing has increased significantly in recent months, becoming one of the most common disruptive behaviors encountered in online video games. The issue of high-skilled players at low levels has been acknowledged in many popular First-Person Shooter (FPS) and Battle Royale (BR) games such as Fortnite, Apex Legends, League of Legends and Dota 2 to name a few.

There are multiple reasons why players would want to smurf and play against lower-skilled players, the most common one being that playing against lower-skilled players generally means easy wins in competitions. For example, some players smurf to boost one's confidence while others do it for practice as it could be a good source of training; more specifically, they create a new account to gain experience and improve their skill set by learning new strategies, without taking the risk of affecting their gameplay statistics and win streaks on their main account. We also find what are considered "ethical smurfs", players that smurf for educational purposes or video content creation. These smurfs typically set themselves a challenge to achieve, such as level up tiers in a record time or reach high ranks using only bad weapons. And while these smurfs are less disruptive in different ways, they still put novice players at a skill disadvantage and have an important impact on player's gaming experience. That said, in this section, we mainly focus on detecting smurfs that abuse the system by constantly creating new accounts to farm lower levels, and disrupt the game environment and gaming experience.

Whether smurfing is considered a bannable offense is up to the discretion of the game designers. With that said, it has become one of the biggest problems in modern gaming across all genres and it is up to video game developers to find methods of detecting and dealing with smurfs. In order to maintain a fair competitive environment, many video gaming companies have decided to take action against players who are found to be smurfing in their games. Fortnite announced they would immediately ban players that are caught smurfing while in Dota 2, for example, players suspected of smurfing are set against each other until the algorithm either bans them or verifies their authenticity. The former however resulted in many innocent players to get banned from the game due to bugs in the game's anti-cheat software. For the most part, the detection of smurfs remains a manual task and while it is hard to completely

eliminate smurfing from occurring in games where account creation is free, it can be detected and appropriate actions can be taken against the players involved.

We propose a novel approach to detect high skilled players in low level matches in competitive multiplayer games based on players' in-game behavioral patterns. Our model is trained to identify average player performance at early account life/levels and detect anomalies in that dataset, defined as players with discrepant high performance in their early stages of the game. Hence, it is necessary to select a set of features that form a good representation of player in-game performance (such as the average number of kills per game, average rank placement in matches, competitive MMR, number of shots hit versus the number of shots fired, etc), and analyze at which level approximately do we observe a clear improvement in a player's skill level. Our goal is to provide game designers insight on early player behavior patterns, in particular, on high skilled low level player behaviors and how they differ from the norm. By detecting smurfs in the early stages of competitive multiplayer games, we can improve beginner players' user experience, which could lead to longer player retention. It is important to note that our system does not take any action on the players and our aim is primarily to improve game designers workflow in detecting smurfs. When automating the detection of smurfs, it is important and necessary to be extremely careful with False Positives (FP) as falsely accusing an innocent player can impact players, and the relationship between the developer company and the player community.

In the following section, we describe the game environment as well as the dataset used in our experiments, followed by an in-depth description of the feature set selected to identify high skilled players in low level rankings. We also mention additional features that were analyzed, but not used in our detection system. In section 3.3.3, we first present initial results on manually selected outliers and generalize our findings after automating the detection. We also improve the automation with additional

features to differentiate smurfing behaviors from ones that leverage external software and show the efficiency of our approach on a real-world dataset that includes over 18,000 unique players.

3.3.2 Methodology

Game Environment

Consider an FPS-type game consisting of 20 players competing against one another in a match. Players experience the game actions through the eyes of a character that is chosen before the match begins, and navigate through an open environment that is known by all players. A player's objective is to defeat all opponents in a match while avoiding being hit, all while remaining the last person standing at the end of the match. Each player is assigned a final rank placement corresponding to the number of players remaining in the game after being eliminated, plus one, similar to BR games.

When a player first registers and creates an account on a game, we assume the player is a novice one and they are assigned the lowest of skill levels to be matched against other beginner players with similar skill level. A player levels up by accumulating experience points (XP) in matches. There are many ways to gain XPs, and this includes accumulating kill assists, kills, placing in the top ranks in the leader board or by winning matches among other things. After reaching a certain number of XPs, the player advances to the next skill level and is matched against players with a similar skill set, and so on.

Data Collection

Our analysis is based on gameplay data from a multiplayer online first-person shooter video game. We use data from 18,264 unique new players and collect their gameplay data over the span of two consecutive weeks after their registration date. Player data is collected in accordance with applicable privacy policies and collected based

on players' privacy settings and preferences.

Detecting smurfs in online video games remains a relatively recent problem, with little to no research on players' behavioral patterns. As a consequence, we do not have access to test datasets. In what follows, we use preconceived knowledge from human experts and explore and experiment with player's gameplay data to infer information on player behavior and on what differentiates a smurf from the rest of the population. We combine these findings with tools from unsupervised learning to automate the detection of smurfs in our dataset. Our goal is to determine whether a high-skilled player in a low level ranking is in fact a smurf and separate them from novice ones.

To do so, we base our analysis on a number of different features that we describe in the following section. In what follows, when we refer to the game or the environment, we imply that all previously mentioned conditions apply.

Feature Selection

There are a certain number of features we are interested in looking into as they give us more insight on a player's skill level. In particular, we look at their average number of kills per match, the average number of matches it takes them to reach the required number of XPs to level up, their average rank placement over the matches played and finally, their shot accuracy.

- **Number of Kills**. Beginner players have little to no skills when playing a game for the first time and eventually become more experienced as they play more matches and get more familiar with the game environment. By analyzing the general trend of beginner players, a novice player averages about 1 kill per match in the first hundreds of games. Hence, any player with over 1 kill per match can be considered a higher skilled player, which gives us important insight on the player's skill set.

• **Matches to Level Up**. We also look at the number of matches it takes a player to achieve a required criteria to level up. As a reminder, a player starts off at the lowest possible level and progressively advances to higher levels as they gain more experience points.

• **Rank Placement**. Another important feature we look at is a player's final rank placement. We expect a novice player to gradually improve its skill set, and hence gradually rank higher in the leader board. Suppose we have 20 players in a match. The probability for a random player to repeatedly rank in the top 5 positions out of 20, n times, is equal to 0.25^n. As n increases, the probability of this event occurring decreases significantly. If we consider that all players in our dataset have the same skill level, then a player has 1 out of 0.25^{20} chance of consecutively finishing in the top 5 rank placements in its first 20 matches. When looking at the average rank placement, we are interested in analyzing players that average a low rank placement as this could be an important indication on their higher skill set.

• **Shot Accuracy (shots hit)**. Finally, we look at a player's shot accuracy. In general, the goal in FPS and battle-royale games is for a player to eliminate all its enemies by defeating them using weapons and other tools found in the game environment. A novice player is more likely to start with an average to low shot accuracy, improving gradually as they gain more experience by playing more games. Players with an almost perfect shot accuracy are either very skilled players opposed to much lower-skilled ones (slower reflexes, movements during the game etc), or cheaters using external tools such as aimbots to improve their shooting accuracy. Hence, we use a player's shot accuracy, which is equal to the number of shots that hit opposing players over all shots fired during a game, averaged over all games played, to separate the higher skilled players and cheaters from the novice ones.

Unused Features

We analyzed additional features that weren't as insightful as the previous ones and are not currently being used in our detection system, but are still worth mentioning to help researchers with future work on high skilled player detection in competitive multiplayer games.

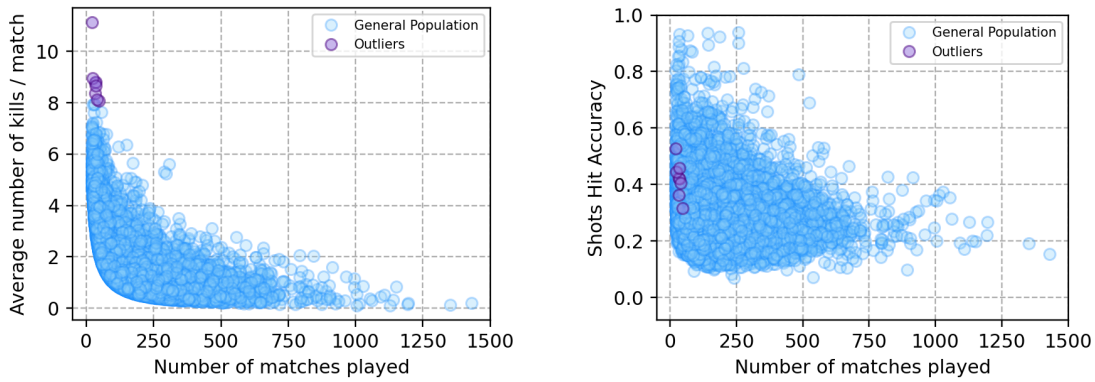
- **Item Categories.** Items are scattered through the game environment and are available for players to collect and use. The purpose of these items is to aid players in eliminating opponents in the match. This includes and is not limited to weapons, shields and survival equipment such as healing items and more. After analyzing suspected smurf behaviors, we notice that on average these players collect more ammunition and health-related items than the average player, whereas the novice players pick up more weapons. This can be explained by the fact that advanced players are more likely to be involved in direct conflicts eliminating opposing players and hence spend more ammunition, increasing their chance of getting injured and their interest in collecting health-related items.

- **Characters Used.** Another important element in FPS games are characters. Players have the freedom to choose from a wide range of characters, all with unique set of skills and abilities. We notice that novice players are more likely to switch characters throughout their matches, whereas higher skilled players are more likely to stick to one or two particular ones, which could be due to habit, experience or personal preferences.

- **Shot Accuracy (fatal shots, head shots).** Finally, when we analyzed a player's shot accuracy, we looked at the number of shots hit (i.e. number of shots that hit an opponent), the number of fatal shots, and the number of head shots. As fatal shots and head shots are more rare and harder to achieve, we did not have

enough information to set a threshold that differentiates a higher skilled player from a novice one.

3.3.3 Experimental Results



(a) Players' average number of kills per match over the total number of matches played. Players with a high average number of kills are highlighted in purple.

(b) Players' average shot accuracy over total number of matches played. Players highlighted in Figure 3·5a are also highlighted in this figure for analysis purposes.

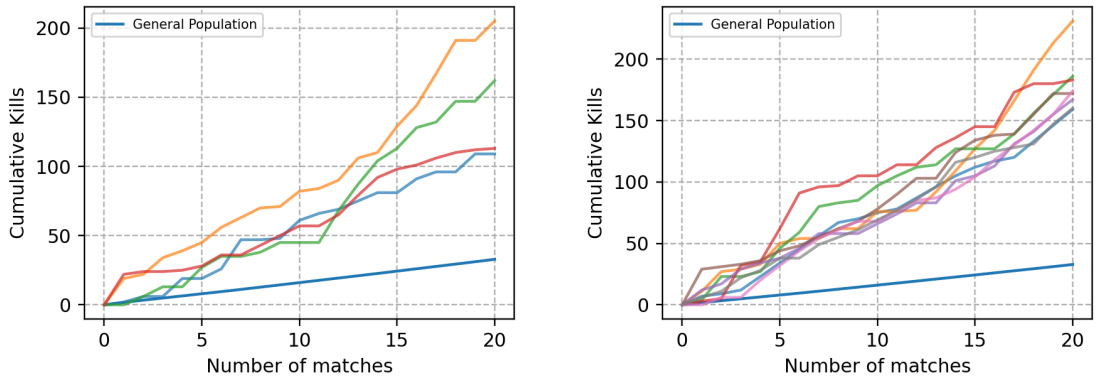
Figure 3·5: Initial analysis of gameplay dataset.

Based on the feature set described earlier, we plot initial results to help with our analysis. In Figure 3·5, we plot some of these features to compare a novice player's skill set level to higher-skilled ones. In particular, we plot players' average number of kills per match for each individual player in Figure 3·5a and players' average shot accuracy over all matches played in Figure 3·5b. We highlight a couple of manually detected outliers in both plots in purple, that is, players that have played less than 100 games with a high average number of kills and high average shot accuracy per match.

	#M to Level Up	Average Kills	Rank Placement	Shot Accuracy
GP	**88.02**	**1.66**	**8.61**	**0.29**
MO	**12.88**	**8.87**	**4.5**	**0.42**
A	19	5.45	9.1	0.34
B	13	10.25	3.45	0.64
C	14	8.1	7.3	0.33
D	16	5.65	7.05	0.46

Table 3.4: Gameplay data for 4 confirmed smurf accounts over the first 20 matches played after registration.

In Table 3.4, we report the average number of matches needed to level up for the general population (**GP**), the manually detected outliers (**MO**) and 4 confirmed smurf accounts (**A**–**D**) obtained from the game designers. We analyze the smurfs' in-game behavioral patterns as well as their match progression as opposed to the general trend. We focus on the four features mentioned in the previous section, that is, the average number of matches needed to level up, the average number of kills per match and their average rank placement and average shot accuracy over the first 20 matches played. Generally, a player averages less than 2 kills per match, as opposed to an average of 7 kills per match for the smurfs. Smurf players also rank higher overall than the general population and have a more precise shot accuracy throughout the first couple of games.



(a) Kill progression for 4 confirmed smurf accounts.

(b) Kill progression for each previously highlighted outlier.

Figure 3·6: Cumulative kill progression of higher-skilled players as opposed to the general population (in navy blue) over the first 20 matches played.

We plot the cumulative number of kills over the first 20 matches for each highlighted outlier in Figure 3·6b and for each confirmed smurf account in 3·6a. For comparison, we also include the general trend of cumulative kills over the first 20 matches for the rest of the players in navy blue, emphasizing the difference in skill levels between smurf players and regular novice players. As a reminder, all players in our dataset are selected from the pool of beginner players.

The outliers highlighted in Figures 3·5a and 3·5b clearly show a faster progression in terms of cumulative kills in the first 20 matches played (cf. Figure 3·6b), reaching over 150 kills after 20 games as opposed to approximately 40 for the general population. From Table 3.4, these outliers level up much faster than the general population, most of them leveling up in less than 13 matches as opposed to approximately 88 matches for a normal player. Similarly, confirmed smurfs level up much faster than the general population and follow the same trend as the manually detected outliers, leveling up in 19 or less matches.

Automating Detection

Unsupervised anomaly detection methods have been commonly used in problems with unlabeled datasets to prevent fraud, adversary attacks and intrusions. It assumes that the majority of data points in the unlabeled dataset are "normal" and identifies observations that don't fit to general patterns considered as normal behaviors; that is, it identifies rare events that differ significantly from normal behaviors. Given the fact that we are looking at a dataset composed of what should only be novice players with little to no skills, we define an outlier as a highly-skilled player significantly exceeding beginner player capabilities. Using our preconceived knowledge on smurf behaviors, combined with the feature set described in the previous section, we use Isolation Forests to detect outliers in our dataset and separate high-skilled players from lower-skilled ones in low rankings.

It is hard to estimate the number of smurfs in our dataset as we have no accurate estimation on the number of new smurfs per month. We can calculate the number of accounts that share one device or use the same IP address, but there is no accurate way to estimate how many of those accounts are actually smurfs as opposed to accounts belonging to one family, or one college campus for example. Out of 18,264 players in our dataset, we assume that 0.01% of them are outliers and possible smurfs, that is, 182 outliers.

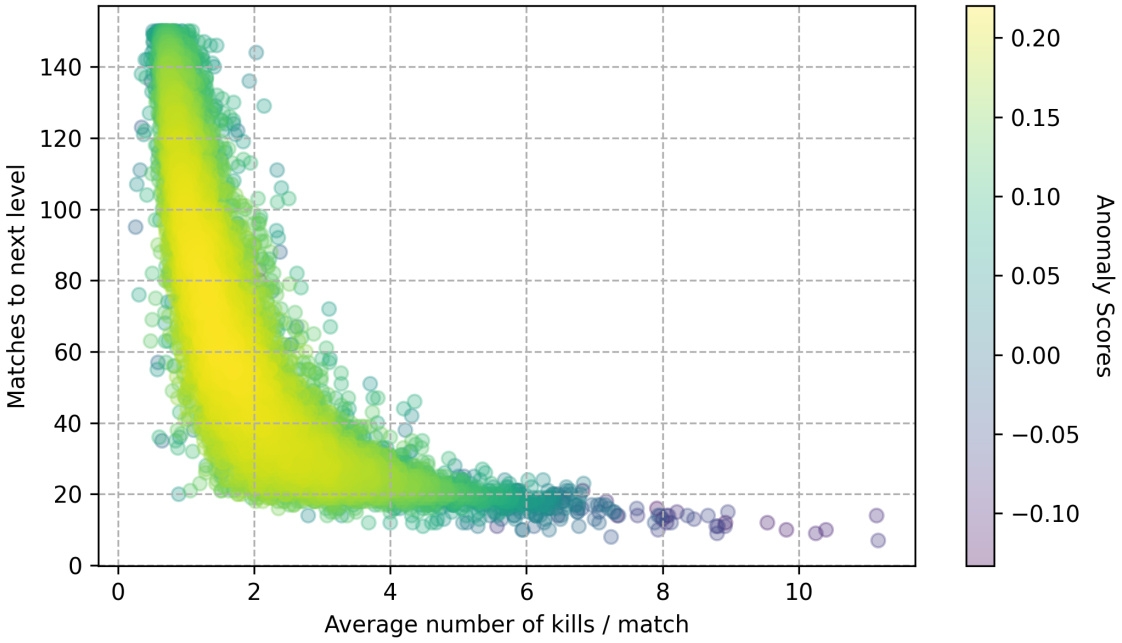


Figure 3·7: Scatter plot representing the anomaly scores assigned to each data point in the gameplay dataset.

In Figures 3·7 and 3·8, we plot data points from our gameplay dataset, such that each data point represents a player's number of matches needed to reach the next game level with regards to the player's number of kills per match, averaged over the first 20 matches played. Each player is assigned an anomaly score by the isolation forest indicating the degree of skill difference exhibited by the player; positive scores (lighter colors) represent normal observations and negative ones (darker colors) represent outliers such that the lower the score, the more abnormal the behavior. In Figure 3·7, we notice that the players that differ most from the standard behavior are the ones in the bottom right corner of the graph, where the number of matches needed to level up tends to be less than 20 matches and the average number of kills per match over the first 20 matches played is at least 7 kills.

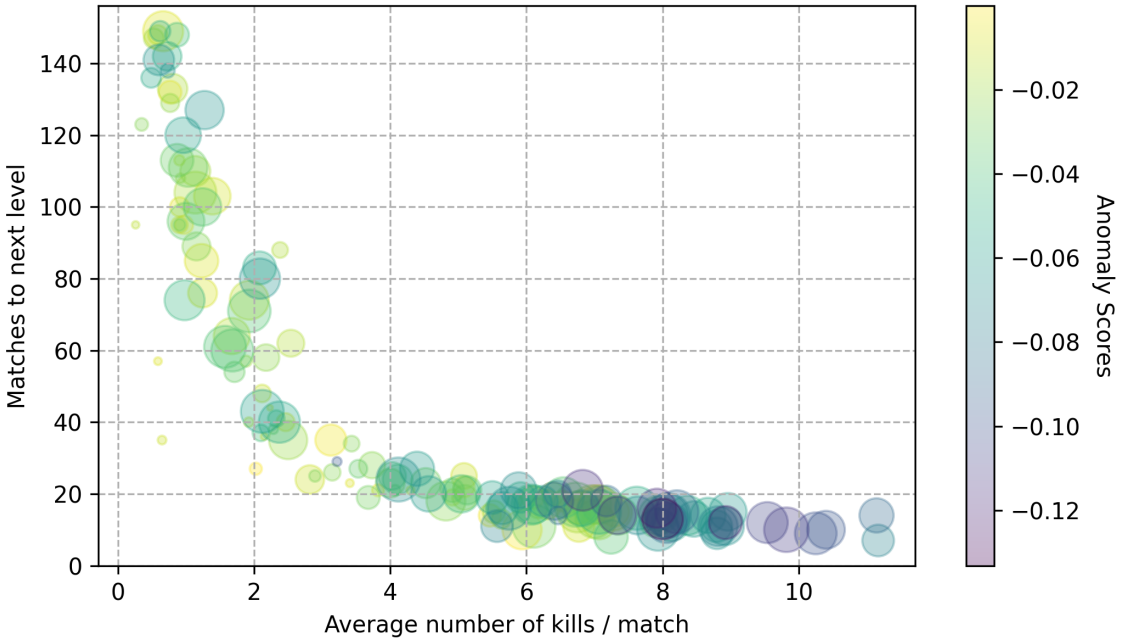


Figure 3·8: Scatter plot representing the anomaly scores assigned to each flagged outliers.

In Figure 3·8, we only plot the outliers detected by the isolation forest, such that the bigger the node size, the higher the player ranked in the first 20 matches played. While it is true that players that need over 60 matches to level up have been flagged as outliers, they are not necessarily smurfs. Since anomaly detection algorithms detect any observation that differs significantly from standard behaviors, our system detects extremely low-skilled players and classifies them as outliers, in particular, players with an abnormally high number of matches played to level up. That said, out of all outliers, the ones assigned the lowest scores and therefore exhibiting the most abnormal players are nevertheless the ones with a significantly higher number of average kills per match, higher shot accuracy and higher rank placement. Further investigation and human intervention is still required to confirm gameplay legitimacy, and we discuss this in following section. Finally, we confirm that the 4 confirmed smurf accounts mentioned earlier were all identified by the isolation forest.

We also remind the reader that no action against any flagged player is taken by our system. Our algorithm solely identifies suspicious players that exhibit typical smurf behaviors and findings are provided to the corresponding game team.

	#M to Level Up	Average Kills	Rank Placement	Shot Accuracy	Duration of Play	Bans/Associations	Cheat Reports	System Label	GD Label
A	13	8	2.61	0.78	0 days 01:59:00	0/0	7	**cheater**	**cheater**
B	16	7.9	3.4	0.75	0 days 23:39:00	4/6	58	**cheater**	**cheater**
C	7	11.2	6.6	0.33	49 days 02:39:00	0/0	17	**smurf**	**unknown**
D	12	8.1	2.7	0.52	240 days 15:38:00	0/5	20	**smurf**	**smurf**
E	18	7.2	7	0.74	0 days 23:36:00	4/6	14	**smurf**	**cheater**
F	14	8.1	1.2	0.32	103 days 04:48:00	0/0	256	**smurf**	**cheater**
G	11	8.9	3.8	0.33	0 days 02:25:00	0/0	0	**smurf**	**unknown**
H	11	8.8	3.1	0.37	28 days 03:43:00	0/8	3	**smurf**	**smurf**
I	10	7.9	2.1	0.33	283 days 07:18:00	0/3	51	**smurf**	**smurf**
J	21	5.9	4.7	0.77	3 days 22:57:00	0/2	1	**smurf**	**smurf**

Table 3.5: Top 10 outliers detected with the lowest anomaly scores, i.e. the most abnormal behavior compared to the rest of the dataset.

Cheaters vs. Smurfs

It is important to note that while most outliers detected by our system show higher skilled behaviors compared to the rest of the population, they might not necessarily smurfs. In fact, some higher-skilled players might be cheaters using external software to boost their performance and gain an advantage beyond what normal gameplay would allow. For this reason, it is important for our system to classify the flagged outliers into legitimate and fraudulent gameplay behaviors, allowing smurfs and cheaters to be dealt with accordingly.

We look at 3 additional features that give us more insight on a player's performance and gameplay history, that is, the number of associated accounts, the duration of play and the number of previous bans and cheat reports.

- **Associated Accounts.** We first look at the number of associated accounts to the suspected player's account. A smurf typically creates multiple fake and secondary accounts to play against lower-skilled players, for fun, for easy wins in competitions, or for personal challenges among other things. Other smurfs create secondary accounts to improve their skills and tactics without risking the statistics and rankings of their

own account. Therefore, the number of associated accounts is a good indication on a player's intention in a match.

- **Duration of Play.** Additionally, smurfs often create these secondary accounts to achieve a particular goal, such as getting the highest possible number of kills in a certain number of matches or leveling up in the least amount of matches. Because they are matched against beginner, lower-skilled players, these goals are usually achieved in a number of hours or days. Once the set goal has been achieved, smurfs abandon the account and potentially create another one to achieve the same or a different goal. Cheaters on the other hand use external software to boost their performance in a match and hence invest more time and effort into their account in the long-run. For this reason, we are interested in the duration of play spent on the flagged account to get a better understanding on the player's intention.

- **Previous Bans + Cheat Reports.** Finally, as mentioned earlier, players are able to report cheating and other forms of disruptive behaviors during a match through an in-game interface to help game designers keep a fair and balanced game experience for users. Players have little to no gameplay history in early stages of the game. However, we focus on a player and any of its associated accounts' cheat report and ban history, to infer on a player's in-game behavior.

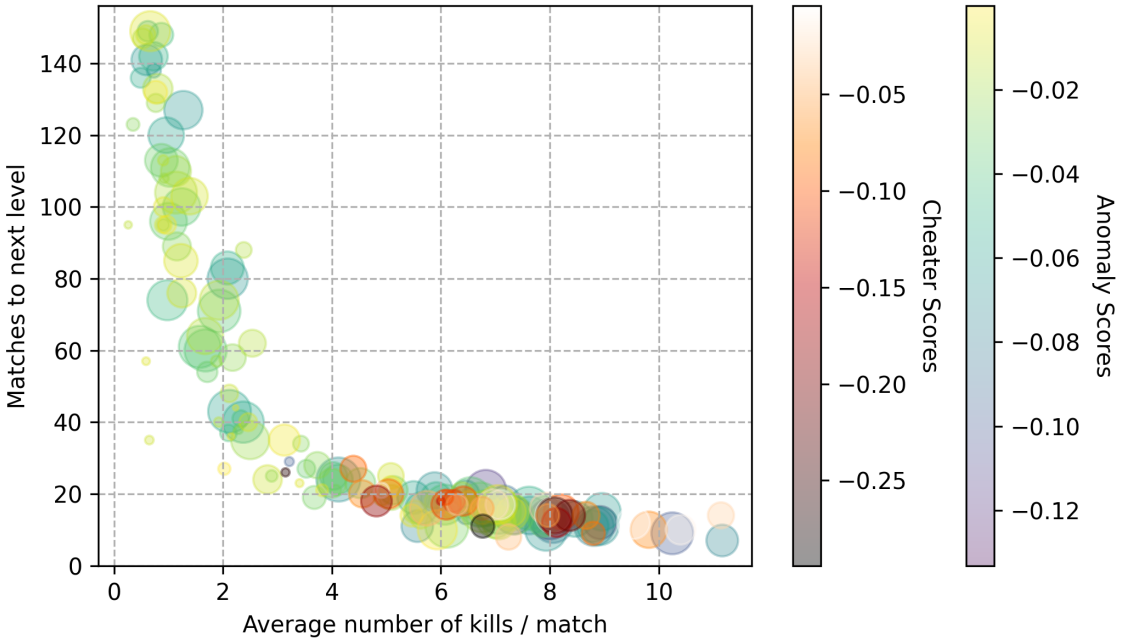


Figure 3·9: Results obtained after running the isolation forest on the gameplay dataset using the additional anti-cheat service features mentioned.

In Figure 3·9, we plot all previously detected outliers and we highlight the ones classified as cheaters in red. The darker the color, the more the player exhibits cheating behaviors. We also included already banned cheaters in our dataset to verify whether they get labeled as such by the system. Whether a player has been banned or not is not used in any part of the system and is only included in the game logs for personal reference.

Evaluation

Detecting smurfs in competitive multiplayer games is still a relatively new problem. Hence, with the lack of test datasets, we turn to game designers to confirm whether the outliers detected are smurfs, cheaters, or neither. In Table 3.5, we include the top 10 outliers with the lowest anomaly scores, that is, the top 10 outliers exhibiting the most abnormal behaviors in the dataset. The first 4 columns correspond to the

features mentioned in section 3.3.2: number of matches needed to level up, number of kills per match, rank placement, and shot accuracy averaged over the first 20 matches played. The following 3 columns correspond to the features used to infer whether a player was legitimately higher skilled, or was more likely using an external cheating software to boost performance: total time between first and last known login, the number of banned associated accounts followed by the total number of associated accounts, and finally the number of cheat reports. The last two columns correspond to the label assigned to the outliers by the system and by the game designers.

Out of the 182 outliers detected in our dataset, 64% of players had over 6 kills per match over the first 20 matches played, 62% of them ranked in the top 5 positions on average, and 87% of them required less than 20 games to level up. Finally, all 182 outliers averaged a shot accuracy of 0.57 in the first 20 matches played. After further investigation by the game design team, we can confirm that 94% of outliers were either cheaters or smurfs, with 65% of them having been confirmed to be smurfs.

When a player uses an external cheating software to boost performance, it is relatively straightforward for game designers to confirm that the player is a cheater. For the rest, it takes game designers approximately 20 to 30 minutes to check and confirm whether a high-skilled player in low level rankings is a smurf and to understand the player's intention (e.g. players that lost access to main account, players with experience playing FPS video games, bad and disruptive intentions). By isolating outliers in our dataset, we significantly reduce the number of players that need to be investigated by prioritizing the most suspicious ones. The most significant features to determine whether the players are players' in-game behavioral patterns as described in the earlier sections, as well as the number of associated accounts which help define a suspicious player's intentions. Finally, outliers flagged by our system that were confirmed to be cheaters by the game designers have been banned from the game.

3.4 Discussion

While it may not be feasible to entirely eliminate cheating from games, it can be detected and the players involved can be punished accordingly. We define game-breaking cheaters as players that intentionally disrupt other players' game experience for personal pleasure and possible potential gains. There are many ways to cheat in online video games, and in this chapter, we present a novel approach to detecting two major cheating behaviors in competitive multiplayer games: collusion in team-based multiplayer games (where collusion happens between teams of players) and high-skill players in low-level rankings of FPS games.

Collusion in multiplayer games can be described as players coordinating to purposely gain an unfair advantage, in a scenario in which they are posed as adversaries. The term "smurf" on the other hand generally refers to skilled players that create new accounts to be matched against novice players and gain a competitive advantage, which puts novice players at a skill disadvantage in early stages of the game. Despite the growing concerns from the gaming community, little work has been done to automate the detection and prevention of cheating in competitive gaming, and collusion and smurfing have become one of the most disruptive behaviors encountered in FPS games such as Fortnite, Apex Legends, League of Legends, and Dota 2 to name a few.

We describe a system that detects outliers in a dataset based on a player's in-game behavioral patterns. Additionally, for colluding behaviors, we use a player's external and in-game social relationships paired with its in-game behavioral patterns, to infer cross-team collusion in games. We automate the detection using Isolation Forests and assign anomaly scores to each player that indicate the degree of cheating exhibited by the player (or its corresponding team, in the case of collusion). Because of the possibility of FP and the on-going discussions on whether common cheating behaviors

such as smurfing should be a bannable offense, we make sure that our system does not take any direct action against suspected cheaters. Whether any action should be in fact taken against suspected cheaters is left entirely to the discrepancy of the game designers. And while human intervention from game design experts is still required to verify the legitimacy of cheaters, our system significantly improves their workflow in detecting cheaters in competitive multiplayer games and gives them more flexibility regarding enforcement actions that could be taken against the players flagged by our system, if deemed necessary.

Chapter 4

Concluding Remarks

4.1 Summary

With the rise in popularity of competitive multiplayer online games, the gaming industry has become a prime target for cyber attacks and malicious activity. Gaming companies are enhancing the stability and security of their network or gaming systems to deal with these threats in an attempt to mitigate the impacts of these activities and improve the gaming user experience. However, most current techniques only detect threats of known types that follow a certain trend or behavior. More robust and adequate techniques need to be developed in order to detect threats without explicit description of the attack and adapt to various types of attacking behaviors and environments. In this thesis, we show how the development of ML-based techniques can contribute to building a more stable and safer network system and mitigate the impacts of malicious activities in competitive multiplayer games. We analyze network intrusion and cheating detection in settings with limited prior-knowledge on attacking strategies and with little to no labeled datasets.

While chapters 2 and 3 focus on different threats, we find that behavioural-based cheat detection is strongly related to intrusion detection. Such systems rely on two techniques to identify malicious activity; the first is a knowledge-based detection (or signature recognition) technique, and we show that advanced technologies based on machine learning and concepts from game theory are well equipped to identify the illicit activities and learn optimal defensive strategies against network intrusion.

The second technique is a behavioral-based detection technique and uses heuristic analysis to identify anomalous behaviour. By incorporating ML-driven solutions, we continuously analyze a player's in-game behavioral patterns, flag suspected cheaters and prevent confirmed cheaters from potentially pursuing fraudulent activities.

4.2 Contributions

The research we present focuses on the application of anomaly detection techniques to identify and detect stealthy persistent threats and automate cheater detection in competitive multiplayer games. The main contributions of this thesis are as follows.

- **Adaptive Defensive Strategies.** As an improvement over the current defensive strategies, we introduce flexible and adaptive strategies to learn cost-effective scheduling of defensive mechanisms to protect against the stealthy characteristics of network intrusion, such as advanced persistent threats. Our model is based on techniques from deep reinforcement learning that are better equipped to deal with the limited information and uncertainty regarding the attacker's moves and is the first adaptive strategy which can play against any opponent with no prior knowledge in the two player security game FlipIt. We extend the work to larger-action spaced FlipIt to improve learning with lower-cost moves and generalize it to multiplayer FlipIt to learn against multiple simultaneous attacks.

- **Resource-Efficient Defensive Strategies.** When resources such as memory, storage and processing powers are limited, it is important to develop cooperative strategies across multiple defensive mechanisms. Cooperation requires that players on a same team compromise, and therefore forgo their individual maximum payoffs. For this reason, we introduce the cooperative game theoretic extension of the FlipIt security game to solve the problem of credit assignment. Under this framework, we

propose a novel approach based on solution concepts from GT to approximate local rewards and fairly distribute the global reward, in contrast to the shared reward approach, reflecting each agent's own contribution 1) to force cooperation between adaptive agents and 2) to improve learning efficiency. With this model, defenders have the ability to develop cooperative defensive strategies exhibiting the fair credit assignment and fast convergence rate in cooperative multiplayer FlipIt, allowing attacks to be detected in a faster and more efficient manner.

- **Generalized Cheater Detection.** As an improvement over the current machine learning approaches that often detect specific behaviors based on pre-defined rules, we present a more generalized anomaly detection system to detect a wider variety of fraudulent behaviors in competitive multiplayer games. Our method is based on analyzing features and heuristics, and detecting informative behavioural characteristics of cheating, rather than detecting the cheating mechanism itself. The proof-of-concept system has been tested on real-world, unlabeled, gameplay datasets and shows significant success in its ability to distinguish between honest players and cheaters. Our method also allows for a single system to be applied to many games and detect a wider range of fraudulent behaviors with minimal re-training and at little cost.

4.3 Future Work

Future work would focus on the continued generalization of our systems to detect more complex behaviors, and the integration of the proposed cheating systems in online games.

- In network intrusion threats, we assume that one player (the defender) is adaptive, and the other (the attacker) is non-adaptive. The problem hence comes down to the defender learning what non-adaptive (renewal) strategy is being used by the

attacker, and playing an optimal counter-strategy. However, strategies employed by the attacker are rarely ever fixed and we would like to extend our work to consider dynamic and non-adaptive attacking strategies besides the class of renewal strategies. Moreover, the current framework presents some limitations to the study of advanced persistent threats. In our work, we assume that once a player makes a move, it occurs instantaneously; this is not generally the case in real-world scenarios as it can take some amount of time for mechanisms to carry out a move. For this reason, we would like to analyze how non-instantaneous moves could potentially affect learning and how adaptive strategies should be developed. We also introduce adaptive defensive strategies based on deep reinforcement learning, and while our model has proven to be better suited than other handcrafted solutions, there is an existing danger in relying on training models solely based on opponent moves. In fact, the opponent may be able to "fake" a strategy that leads our model to learn inaccurate counter-strategies, which the opponent can later exploit. This is commonly referred to as leadability; one solution to prevent leadability is for systems to learn from their mistakes instead of their successes, which would prevent the model from learning from what it thinks are correct moves.

- Our detection systems have great potential in minimizing cheating behaviors in competitive multiplayer games. That said, we would like to continue the research on colluding and smurf behaviors in team-based and competitive multiplayer games to better understand the behaviors exhibited and improve the accuracy of the detection systems. This would allow us to lower FP rates in our detection and to better differentiate behavioral-based cheaters from ones that leverage external cheating software. We would also like to analyze additional features in more depth, such as in-game location for suspected colluders (whether an object such as a wall or building separates two teams that are believed to be close in proximity), and player psychomotor skills

measurements (i.e. movement, coordination, speed) which could give us more insight regarding a player's skill set. Additionally, in our work, we focused on smurfs in the early stages of games, in other words, in beginner game modes. Yet, we find smurfs in different levels of BR games and we would like to extend our work to all ranked mode levels, where players manipulate their gameplay statistics to play on lower-level rankings and be matched against lower-skilled players.

References

Alpcan, T. and Basar, M. (2010). *Network security: A decision and Game-Theoretic approach*. Cambridge University Press.

Buczak, A. L. and Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176.

Das, A., Gervet, T., Romoff, J., Batra, D., Parikh, D., Rabbat, M., and Pineau, J. (2018). Tarmac: Targeted multi-agent communication.

Ding, D., Han, Q.-L., Xiang, Y., Ge, X., and Zhang, X.-M. (2018). A survey on security control and attack detection for industrial cyber-physical systems. *Neurocomputing*, 275:1674–1683.

Epic Games, Inc. (2006). Easy anti-cheat. `https://easy.ac`. (Online; last accessed on 2022-04-29).

Falliere, N., Murchu, L. O., and Chien, E. (2011). W32. stuxnet dossier. *Symantec White Paper*.

Feng, X., Zheng, Z., Hu, P., Cansever, D., and Mohapatra, P. (2015). Stealthy attacks meets insider threats: A three-player game model. In *IEEE Military Communications Conference (MILCOM)*, pages 25–30.

Galli, L., Loiacono, D., Cardamone, L., and Lanzi, P. L. (2011). A cheating detection framework for unreal tournament III: A machine learning approach. In *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, pages 266 – 272.

Greige, L. and Chin, P. (2022a). Deep reinforcement learning for FlipIt security game. In Benito, R. M., Cherifi, C., Cherifi, H., Moro, E., Rocha, L. M., and Sales-Pardo, M., editors, *Complex Networks & Their Applications X*, pages 831–843, Cham. Springer International Publishing.

Greige, L. and Chin, P. (2022b). Shapley Q-value learning for cooperative multiplayer flipit. Submitted.

Greige, L., De Mesentier Silva, F., Lawrence, C., Pierse, C., Shi, J., and Chin, P. (2022a). Detecting high skill players in low level rankings. Submitted.

Greige, L., De Mesentier Silva, F., Meredith Trotter, C. L., Chin, P., and Varadarajan, D. (2022b). Collusion detection in team-based multiplayer games. In Martin, A., Hinkelmann, K., Fill, H.-G., Gerber, A., Lenat, D., Stolle, R., and van Harmelen, F., editors, *Complex Networks & Their Applications X*, Stanford University, Palo Alto, California, USA.

Gueye, A., Marbukh, V., and Walrand, J. C. (2012). Towards a metric for communication network vulnerability to attacks: A game theoretic approach. In *Game Theory for Networks (GameNets)*.

Hu, P., Li, H., Fu, H., Cansever, D., and Mohapatra, P. (2015). Dynamic defense strategy against advanced persistent threat with insiders. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 747–755.

Kim, D., Moon, S., Hostallero, D., Kang, W. J., Lee, T., Son, K., and Yi, Y. (2019). Learning to schedule communication in multi-agent reinforcement learning.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Knudsen, K. (2011). Impact of collusion and coalitions in RISK.

Kunreuther, H. and Heal, G. (2003). Interdependent security. *Journal of Risk and Uncertainty*, 26(2):231–249.

Laasonen, J., Knuutila, T., and Smed, J. (2012). Eliciting collusion features. In *2nd International Workshop on DIstributed SImulation & Online gaming*. ACM.

Laasonen, J. and Smed, J. (2013). Detecting a colluding subset in a simple two-dimensional game.

Laszka, A., Horvath, G., Felegyhazi, M., and Buttyán, L. (2014). Flipthem: Modeling targeted attacks with flipit for multiple resources. In Poovendran, R. and Saad, W., editors, *Decision and Game Theory for Security*, pages 175–194.

Laszka, A., Johnson, B., and Grossklags, J. (2013a). Mitigating covert compromises. In *Proceedings of the 9th International Conference on Web and Internet Economics (WINE)*, volume 8289, pages 319–332.

Laszka, A., Johnson, B., and Grossklags, J. (2013b). Mitigation of targeted and non-targeted covert attacks as a timing game. In *4th International Conference on Decision and Game Theory for Security (GameSec)*, volume 8252, pages 175–191.

Liu, F. T., Ting, K. M., and hua Zhou, Z. (2008). Isolation forest. In *In ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining. IEEE Computer Society*, pages 413–422.

Liu, Y., Yang, Y., and Sun, Y. L. (2008). Detection of collusion behaviors in online reputation systems. In *2008 42nd Asilomar Conference on Signals, Systems and Computers*, pages 1368–1372.

Lowe, R., WU, Y., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Mazrooei, P., Archibald, C., and Bowling, M. (2013). Automating collusion detection in sequential games. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, page 675–682. AAAI Press.

Milosevic, N., Dehghantanha, A., and Choo, K.-K. (2017). Machine learning aided android malware classification. *Computers & Electrical Engineering*, 61:266–274.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nash, J. F. (1950). Equilibrium points in ¡i¿n¡/i¿-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49.

Owaida, A. (2021). Gaming industry under siege from cyberattacks during pandemic. `https://www.welivesecurity.com`. (Online; last accessed on 2022-04-29).

Padhi, S. S. and Mohapatra, P. K. (2011). Detection of collusion in government procurement auctions. *Journal of Purchasing and Supply Management*, 17(4):207 – 221.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *Neural Information Processing Systems*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Édouard Duchesnay (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830.

Pham, V. and Cid, C. (2012). Are we compromised? Modelling security assessment games. In *Decision and Game Theory for Security*, pages 234–247.

Schwartz, N. D. and Drew, C. (2011). Rsa faces angry users after breach. *New York Times*, page B1.

Shapley, L. S. (1953). *A Value for n-Person Games*, pages 307–318. Princeton University Press.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Smed, J., Knuutila, T., and Hakonen, H. (2007). Towards swift and accurate collusion detection. In *8th International Conference on Intelligent Games and Simulation, Game-On 2007*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press.

Takahashi, D. (2021). Akamai: Cyberattacks on gaming grew 340% in pandemic. `http://venturebeat.com`. (Online; last accessed on 2022-04-29).

Tan, A. (2021). Video game industry under relentless cyber attacks. `https://www.computerweekly.com`. (Online; last accessed on 2022-04-29).

Vallve-Guionnet, C. (2005). Finding colluders in card games. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 2, pages 774–775 Vol. 2.

Valve Software, Inc. (2002). Valve anti-cheat (vac). `https://developer.valvesoftware.com`. (Online; last accessed on 2022-04-29).

van Dijk, M., Juels, A., Oprea, A., and Rivest, R. L. (2013). Flipit: The game of "stealthy takeover". *J. Cryptology*, 26(4):655–713.

Villeneuve, N., Bennett, J. T., Moran, N., Haq, T., Scott, M., and Geers, K. (2013). Operation ke3chang: Targeted attacks against ministries of foreign affairs. *Fireeye White Paper*.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Whitney, L. (2020). How cyberattacks are targeting video gamers and companies. `https://www.techrepublic.com`. (Online; last accessed on 2022-04-29).

Xiao, L., Wan, X., Lu, X., Zhang, Y., and Wu, D. (2018). IoT security techniques based on machine learning. *ArXiv*, abs/1801.06275.

Yeung, S. F. and Lui, J. C. S. (2008). Dynamic bayesian approach for detecting cheats in multi-player online games. *Multimedia Systems*, 14(4):221–236.

CURRICULUM VITAE

Laura Greige

Education

Boston University, Boston, MA — *Sep 2017 – May 2022*
Ph.D., Computer Science

Sorbonne Université, Paris, France — *2015 – 2017*
Master's Degree with Honors, Artificial Intelligence

Sorbonne Université, Paris, France — *2012 – 2015*
Bachelor's Degree, Applied Mathematics and Computer Science

Brown University, Providence, RI — *Jul 2014*
Summer Research Student

Professional Experience

Electronic Arts, Redwood City, CA — *Sep 2021 – Mar 2022*
AI Scientist Intern · *Part-time*

Electronic Arts, Redwood City, CA — *May – Aug 2021*
AI Scientist Intern

Electronic Arts, Redwood City, CA — *May – Aug 2020*
AI Scientist Intern

ISIR, Paris, France — *Feb – Jul 2017*
Research Intern

Laboratoire d'Informatique de Paris 6 (LIP6), Paris, France — *Jun – Jul 2016*
Summer Research Intern

Publications

Greige L., Wolf M., Chin P. (2022) **Shapley Q-Value Learning for Team-Based FlipIt.** Submitted.

Greige L., De Mesentier Silva F., Lawrence C., Pierse C., Shi J., Chin P. (2022) **Detecting High Skill Players in Low Level Rankings.** Submitted.

Greige L., De Mesentier Silva F., Trotter M., Lawrence C., Chin P., Varadarajan D. (2022) **Collusion Detection in Team-Based Multiplayer Games.** In: A. Martin, K. Hinkelmann, H.-G. Fill, A. Gerber, D. Lenat, R. Stolle, F. van Harmelen. (eds) Proceedings of the AAAI 2022 Spring Symposium on Machine Learning and Knowledge Engineering for Hybrid Intelligence (AAAI-MAKE 2022), Stanford University, Palo Alto, California, USA, March 2022.

Greige L., Chin P. (2022) **Deep Reinforcement Learning for FlipIt Security Game.** In: Benito R.M., Cherifi C., Cherifi H., Moro E., Rocha L.M., Sales-Pardo M. (eds) Complex Networks & Their Applications X. COMPLEX NETWORKS 2021. Studies in Computational Intelligence, vol 1015. Springer, Cham.

Patents and Patent Applications

Greige L., De Messentier Silva F., Sulimanov A. 2022. Detecting High-Skilled Entities in Low-Level Matches in Online Games. US Patent Application 17/457,194. Patent pending.

Greige L., De Messentier Silva F., Trotter M., Narravula S., Aghdaie N. 2021. Detecting Collusion in Online Games. US Patent Application 17/302,837, filed May 2021. Patent pending.