Detection on auto clickers in mobile games*

Shing Ki Wong and Siu Ming Yiu†
The University of Hong Kong, Pokfulam, Hong Kong
{skwong, smyiu}@cs.hku.hk

Received: August 9, 2019; Accepted: September 10, 2019; Published: September 30, 2019

Abstract

Mobile games are becoming more and more popular nowadays. According to Statista [6], the gaming revenue rises rapidly from 17.6 billion to 40.6 billion from 2013 to 2017. It gradually becomes a big market in the game industry. Game companies are putting more and more focus and resources on their mobile game development. On the other hand, with such a rapid growth of the number of mobile games developed everyday, cheating software developers are also taking advantage of it by implementing and selling cheating software for money. Some famous mobile games (e.g. FIFA Mobile [2], NBA Live Mobile [7], Madden NFL Mobile [5]) include a live market in game for users to trade game items online. This feature is convenient and provides users a unique kind of interactive gaming experience. However, cheaters can take advantage of this feature by implementing auto clickers to automatically buy and bid items in game 24 hours a day effortlessly. In this paper, we are going to investigate this kind of game cheat and propose a detection methodology to determine whether auto clicking behaviour exists. We analyze the touch input behaviour of the client by calculating the clicking dispersion of the touch inputs and the average number of clicks performed per second. Experiment results show that there is a significant difference between the behaviour of auto clickers and human users in term of their clicking positions and frequencies. Our method is easy to implement and the memory consumption is reasonable and practical.

Keywords: security in mobile games, auto clickers, user behavior, detection of auto clickers

1 Introduction

With the emerging increase in popularity of mobile games in recent years, game companies are putting more and more resources in mobile game developement due to its high return in profit. The high penetration of mobile devices among people all over the world provides good motivations for game developers to focus more on mobile games instead of console games. With a more lenient developing environment and computing power restriction, developing mobile game applications requires fewer work and effort compared to console game developement. Even with lower graphics, simpler logics and fewer game features, mobile game applications can still provide a "good enough" and handy experience for leisure gamers to entertain themselves whenever they want. Such convenience in playing time and location helps to build up a huge user base in mobile gaming. Nevertheless, at the same time, such large pool of user base also attracts a number of game bot developers to implement and sell cheating programs for money. The characteristic of these cheating programs varies and are game dependent. For example, in First Person Shooting (FPS) games, aimbot helps players to automatically aim at targets for effortless kills without moving the crosshair manually. In Massively Multiplayer Online Role-Playing Game (MMORPG), auto

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 10(3):65-80, Sep. 2019
DOI:*10.22667/JOWUA.2019.09.30.045*

*The preliminary version of this paper is presented at the 2019 WISA Workshop. (`https://www.manuscriptlink.com/society/kiisc/conference/wisa2019/workshop`)

†Corresponding author: The University of Hong Kong, Pokfulam, Hong Kong, Tel: +852-28578242

clicker helps players to automatically perform repeatable actions and tasks without the presence of an actual human player. These cheating programs introduce huge unfairness between adversary and legitimate players, causing disrupts among the game community and as a result, destroying the game reputation and shortening the life of the game.

To avoid these bad consequences, game companies take various actions to fight against these cheating behaviours to prevent their game from being taken advantage by the cheating programs and, mostly importantly, to maintain the game revenue. For example, repeated Turing test such as CAPTCHA (Complete Automated Public Turing Test to Tell Computers and Humans Apart) is used to distinguish between human and programs. It filters out botting behaviours effectively by requesting players for inputs periodically. However, the interruptions would also degrade the gaming experience of the players. For action and racing games which seamless gameplay is required for best gaming experience, the problem is even more severe because even a small delay in gameplay could change the whole result of the game. Therefore, such approach is only appropriate in leisure or strategic games such as MMORPG.

Apart from validation during gameplay, suspicious program scanning by the system is another effective measurement taken by many game companies. Scanning software such as Valve Anti-Cheat (VAC) [10] is being installed together with the game during game installation. It tries to scan for malicious programs in the client's computer that may result in cheating. It also checks for abnormal memory and hardware behaviour and collects relevant information for further analysis. These collected data would be sent to the company's server for investigation in potential cheating. Malicious players' account will be banned upon successful cheating identification. These scanning software gives high accuracy in cheating detection while it sacrifices the system resources during scanning. Furthermore, it is possible for cheaters to bypassing the scanning process by modifying the system privilege since the software is located at the client's computer. Recently, there are 'cheating machines' that work in the middle of the link between the client and the server, so that no malicious programs can be detected in the client side.

Cheating in mobile games is much harder compared to cheating in computer games because performing privilege escalation in mobile operation system is more difficult. To gain privilege in mobile devices, users have to undergo unofficial operations (i.e. unlocking bootloader, rooting in Android System or jailbreaking in IOS) which may void the device's warranty. What's more, the limited computing power and software support in mobile devices is another obstacle against the development of cheating programs. In spite of this, once the required privileges are obtained in sacrifice of the system security, it is possible to develop applications to provide game content modification (e.g. GameGuardian [3]) for cheating.

Instead of running cheating programs in mobile devices, it is more common to cheat in mobile games with the help of computers. To facilitate mobile application development, emulators are used as virtual mobile machines so that developers without real mobile devices can still test their applications in computers. In recent years, many emulators (e.g. iPadian [4] for IOS and Nox App Player [8] for Android) are developed for gaming purpose, providing a better gaming experience to players by offering them to a way to play mobile game on their computers. Players can have better control using keywords and joysticks instead of the touchscreen on mobile devices during gaming. Although these emulators provides great convenience to gamers, they gives game bots developers alternative chances on game cheat development. Through the emulator in their computers, they can access and modifiy the game data much easier. The wide range of functionalities provided by computers also give cheaters a lot more ways to cheat in mobile games that run on emulators. One of the common approaches is to use auto clickers (i.e. mouse macro) which involves automated mouse instructions and keyboard inputs that are indistinguishable from human inputs to cheat 'legitimately' in game. With the automation program well set, cheaters can play the game automatically without any effort, causing huge unfairness to other legitimate players.

As mentioned above, although there are researches about behavioural analysis of players' data in computer games, these studies put their focus on popular computers games such as first-person shooter

(FPS) games and massively multiplayer online role-playing games (MMORPG). Very few of them are applicable to mobile games, which are of high popularity nowadays. In this paper, we focus on mobile games involving gameplays with repeatable tasks which can be completed by automation programs. By analyzing and clustering touch inputs from the client, we propose a behavioural detection method to tell whether the inputs are generated by automation programs or not. We look at the spread of the clusters by calculating the Euclidean Distance between the data points and their clusters. The methodology is highly adjustable from game to game and is light in memory consumption.

The rest of the paper is presented as follow. Section II introduces some related work in behavioural analysis-based detection method on game cheats and provides background information about our work. Section III introduces some common types of cheat currently being used in the game industry. Section IV proposes the idea and mechanism of our detection method. Section V presents and explains the experiment results. Section VI points out the limitation of our methodology and suggests possible future improvement. Section VII gives conclusion in our work.

2 Related work

Detection of game cheats has long been a concern in game industry. Game publishers adopted many measures to combat the usage of game cheats in their games. Cheaters, on the other hand, continuously looking for loopholes to take advantage in games. Traditional methods such as anti-cheating rootkits (e.g. VAC and nProtect) which scan for malicious program in clients' computers are direct and intuitive, but on the other hand, technically vulnerable in a way that cheaters can edit the registry values to escape from the detection from the rootkits. Repeated Turing test (e.g. CAPTCHA) degrades the gaming experience as previously mentioned. Although several researches [22] [14] are conducted to minimize the interruption on gaming experience by merging CAPTCHA with in-game elements, the method is found to be imperfect. [23] studied cyborgs, which are semi-bot programs assisted by human, and showed that CAPTCHA fails in detection with such bot programs.

Since the abovementioned method could not solve the problem effectively, researchers look for alternate methods to detect cheating in game from a different perspective. They adopted behavioural analysis-based detection methodologies to analyse the behaviour of the players from the input they directly given out when playing the games. Such kind of input is difficult to modify, especially in online games which the values have to be sent to the server for validation. [19] focused on game log records for database rollback by game companies and proposed a two-stage bot-detection methodology. [21] proposed a bot-detection strategy by making use of the social connections of avatars in social graphs. [11] proposed a set of features to identify gold farms in MMOGs by using data from an MMORPG EverQuest II. [20] analysed the movement behaviour between human and bot in MMORPG to make highly successful classification between the two. [18] proposed an automated server side detection approach based on character activity in game by extracting waypoints and detecting repeated paths. [16] analysed the window event sequences produced by game players to detect auto programs in MMORPGs. [12] proposed a behavioural approach by grouping players with similar behaviours together and perform bot detections to each group individually. [25] focused on aimbots and study the differences between the behaviours of honest players and aimbot users. [13] proposed a human observational proofs (HOPs) to capture game bots by collecting and analysing user input actions. [24] uses the Dynamic Bayesian Network (DBN) approach for cheat detection by representing game states with a DBN. [17] proposed identified features on how players kill and get killed and develop an aimbot detector to detect cheating in FPS games. [15] proposed a user behaviour analysis framework for online game bot detection by studying players' behaviours from their party-play records. These studies focus on popular types of computers games such as first-person shooter (FPS) games and massively multiplayer online role-playing games (MMORPG).

However, there are very few studies focusing on mobile games, which are also very popular nowadays.

3 Common types of cheat in video games

Nowadays, cheating in computer games is heterogeneous attributed to the wide range of functionalities provided by computers. In this section, we give a summary of some common ways of cheating in computer games.

3.1 Memory editing

Memory editing involves searching and modifying on corresponding memory locations of in-game attributes (e.g. increasing the health point of the character) during runtime, usually by trial and error. The modification is simple and intuitive once the corresponding attributes are being correctly located. Such method is vulnerable to detections and can be easily detected if the modification is too abnormal which breaks the game logic.

3.2 Code injection

This method is similar to memory editing. Instead of modifiying the in-game attribute, it edits the execution code to change the behaviour of the game by allowing it to run in favour of the cheater (e.g. repeated generation on in-game currency). Similar to memory editing, code injection may break the execution logic and thus can be easily detected.

3.3 Game acceleration

Game acceleration allows players to perform actions in game much faster than normal players do with the help of an accelerated clock in the system. It accelerates corresponding processes in the client application to shorten its execution time to get earlier responses. For example, the moving speed of the character can be significantly increased with the help of the acceleration. The accelerated clock can even help to reveal in-game data scheduled to be released in the future in advance. Such accelerated behaviours can be identified by implementing server-side checking mechanism on the client's data (e.g. physics check for abnormal moving speed of the character).

3.4 Resending or modifying synchronization data

This kind of cheat acts like the man-in-the-middle attack by intercepting and modifying the data between the client and the server. For example, a skill execution command of a game character sending towards the server can be intercepted and replicated so that the skill can be performed repeatedly ignoring the constraint set up in the game. To indentify such kind of cheating, authentication protocols could be implemented on the server side to verify the legitimacy of the data provided by the client.

3.5 Auto clickers and mouse macro

It is the most typical cheating method used by common players due to its simpilcity. It involves the use of scripts or automation software to generate automatic input instructions. For example, cheaters can make use of auto clicker programs with pre-written mouse macros to play the game on behalf of themselves automatically in multiple machines to earn game items and currencies indefinitely. Detecting such cheating behaviour is much more difficult compared to the abovementioned methods as it does not

Table 1: Comparison between state-of-art approach with our methodology.

State-of-art approach	Our method
Applicable only to a particular game model	Universal
Involve modification of game elements	Only involve tracking of touch inputs
Relatively complex implementation	Easy to implement
Hinder game development	No constraint on game development
May cause impact on user gaming experience	No effect on gaming experience

break any game logic or bypass any authentication process. It acts like normal players by simply entering inputs through the game interface so it is very hard to identify such cheating behaviour.

4 Current state-of-art approaches

In this paper, we aim at finding a detection methodology to effectively identify cheating behaviours with auto clickers and mouse macros. We look at games which contains static UIs where cheaters can easily set up auto clickers to perform their desired operations automatically by manipulating between game interfaces. Since the location of the UI controls are fixed, cheaters can easily locate the controls by search corresponding pixel colours on the screen when creating the mouse macros. Games which are vulnerable to scripting usually contain 'bidding houses' or 'live markets' that allows players to trade game items in game 24/7. As mentioned above, such cheating behaviour is hard to detect and currently there is no direct solution that can solve the problem.

To alleviate the situation, game companies try to introduce additional game features and modifications to reduce the effectiveness of the auto-bots. For example, in FIFA Mobile, a football themed mobile game produced by EA Sports [1], an auto pricing feature is introduced in game so that the system will try to provide suggested prices to each item going to be put on sale. To efficiently generate in-game currency in the game, players can buy low and sell high with the game items through the live market. Such buying and selling process is repetitive and tiring so cheaters would try to program mouse macros to help them to do the bidding operations. The introduction of the auto pricing feature establish a 'nearly perfect' market which everyone can sell their items with optimized prices. As a result, it minimizes the profit gained by bidding and reselling so that the auto clickers would become useless. Such kind of measure can doubtlessly combat the use of auto clickers, but on the other hand it would also affect innocent players, which is undesirable.

What's more, auto clickers work well in any game that contains static user interface. In general, any game which contains gameplay elements that require players to perform repeated tasks are vulnerable to the abuse of auto clickers. Malicious players can 'grind' for rewards effortlessly by programming scripts ans marcos to perform tasks automatically. The approach of introducing new game features to fight against auto clickers cannot solve the problem entirely and may cause side effect to the game community. Our behavioural detection method, on the other hand, tackles the core part of the problem directly. Table 2 give a comparison between the state-of-art approach with our detection method.

5 Proposed method

In this section, we propose a behavioural analysis-based cheat detection methodology which differentiate between auto-bots and normal players by identifying robotic touch inputs behaviours. We look at games containing live markets or auction houses which allows players to trade game items online in live. We

indicate the differences of the touch input behaviour between auto clickers and human users, and then we propose two detection features which analyse the characteristics of the user interface in the games.

5.1 Touch input analysis

There are two common types of UI inputs in mobile games, namely touch inputs and keyboard inputs. Touch inputs are tapping or dragging gestures made by users on the screen of the mobile device. They are the most common inputs made during playtime, usually occurs when the player is interacting with the user interface (e.g. navigating among menus) or during gameplay (e.g. moving a character or shooting at enemies). Keyboard inputs are text inputs made by the player through the virtual keyboard (or hardware keyboard if it exists) of the mobile device when typing (e.g. online chatting with other players or setting items prices when selling). Such kind of input usually happen outside gameplay so we assume that only touch inputs exist during our analysis. This assumption coincide with the bahaviour of auto clickers which only perform clicking operations on the screen without any typing actions.

5.2 Behaviour between auto clickers and normal players

A good auto clicker program has to be efficient and accurate. It should efficiently complete actions or tasks with the least possible number of instructions in a given amount of time. Therefore, the average number of clicks per given amount of time for auto clickers should be much more compared to that given by human users. It should accurately click on the desired locations to avoid any unnecessary trigger that could possibly break the automation. The clicking locations of the auto clickers are usually hard-coded on a particular pixel on the screen to avoid mis-clicking that may trigger unrelated operations that would interrupt the automated process. Some of the state-of-the-art auto clickers are more 'human-like' which try to introduce random offsets to the clicking locations to make them harder to be detected. However, such offsets cannot be too large to avoid mis-clicking. Therefore, we expect the clicking positions made by auto clickers will be more concentrated and clustered compared to that by human users.

5.3 Characteristics of the user interface

Figure 1 and 2 show the user interfaces of FIFA Mobile and NBA Live Mobile, two famous mobile games produced by EA Sports. The items listed for sale in the market are located at the centre of the screen, with timers counting down to their expiration time. Above the selling items, there are some navigation buttons (e.g. search, sell) for various operations. After clicking on a player item, a bidding window will be popped up as shown in figure 3 and 4. Users can then place bids or buy the item buy pressing the corresponding buttons on the screen. Notice that the positions of the buttons and selling items are static so cheaters can hard code the respecitve coordinates positions in the auto clicker.

5.4 Detection features

In out analysis, we proposed two detection features, which focus on clicking frequency and position distribution, to differentiate between the clicking behaviour of human users and auto-clickers.

5.4.1 Dispersion of touch inputs

It is expected that auto clickers would always click on the same position for a particular button on the user interface, while human players will press randomly within the button area. Such difference in behaviour helps in our differentiation. To quantify the difference, we keep track of the clicking positions of auto clickers and human players separately and plot the corresponding coordinates in graphs. We then do a

Figure 1: Market interface of FIFA Mobile

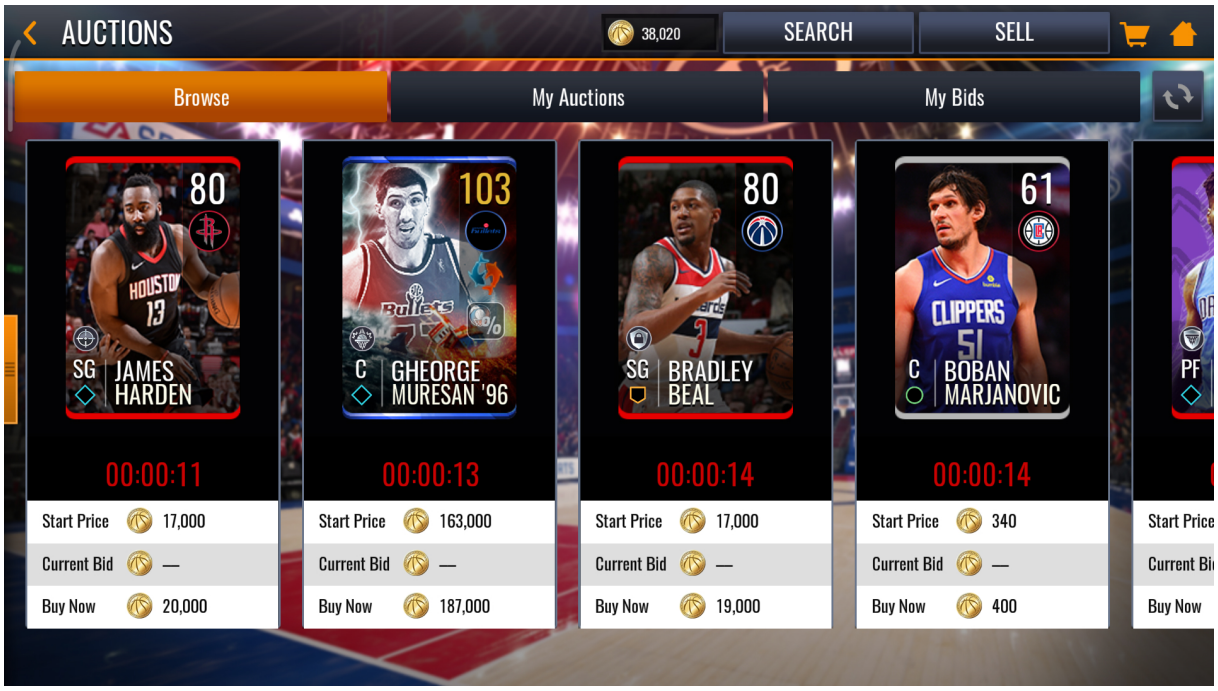

Figure 2: Market interface of NBA Live Mobile

k-mean clustering of the data points after determining the number of clusters using the Elbow method. Each cluster in the graphs represent a certain input area (e.g. a button or a game item) of the game UI. We calculate the Average Euclidian Distance of the data points from their corresponding centroids of the clusters and normalize it by dividing it by the screen resolution of our device to remove the effect on

Figure 3: Bidding page of FIFA Mobile

Figure 4: Bidding page of NBA Live Mobile

resolution differences in different devices. The Adjusted Euclidian Distance ($d_{adj.}$) can then be obtained by the formula

$$d_{adj.} = \frac{AverageEuclidianDistance}{\sqrt{ScreenResolution}}. \quad (1)$$

It is expected that human inputs would give a much larger distance than that by auto clickers.

5.4.2 Number of clicks per second

To optimize the performance, auto clickers would try its best to react to every action (e.g. bidding, selling, navigating between menus etc.) with its quickest response in order to maintain its advantage over human players. Therefore, it usually neglects human response time and physical movement time (i.e. hand movement of human) by performing teleporting moves and thus would generate a larger number of clicks per second. By keeping track on the number of clicks on screen and divide it by the time elapsed in seconds, we can calculate the average number of clicks per second. It is expected the number of clicking operations made by auto clickers would exceed that by human users by a significant amount.

6 Evaluation

We use FIFA Mobile and NBA Live Mobile to be the games in our experiment. Each of the two games contain a live market or auction house which allows users to trade their player items online. Users can search for desired items with filters (e.g. name, team, bid price) by getting rid of items that they don't want. Each player item has a starting price and buy now price. To obtain a player item from the market, users can either buy the player straightly with the buy now price or bid it from the starting price. Buying a cheap player item through buy now price is difficult, as there are many other users hunting for the same item at the same time when it first pops up in the market. Bidding a cheap player item is time consuming, as the user have to undergo a "bidding war" with others by continuously placing bids on the same player item many times to make sure they are the highest bidder. Both approaches are tedious and time consuming so by making use of an auto clicker, players can eliminate such frustration and trade items effortlessly.

6.1 Experiment setup

To simulate the cheating behaviours, we run the two games in Nox App Player, an Android emulator designed for gaming. We use Pulover's Macro Creator [9] to develop an auto clicker which can automatically bid and buy player items in the market through the emulator in our computer. Our self-implemented auto clicker keeps bidding on the player items listed on the screen from left to right. After placing bids on all the items, the auto clicker will click the refresh button at the top of the screen to load a new list of player items for bidding. Such process will be looped infinitely. We set an offset of 15 pixels for each clicking position to mimic the random behaviour of human users. We also filters out players which are not worth buying (i.e. overpriced low value players) and by running our auto clicker with the pre-set filter, only items that are profitable for reselling are available on the list.

Before the experiment, we show and explain to participants how our auto clicker works in practice and let them observe the bidding behaviour. We then request them to mimic the bidding behaviour of the auto clicker when they are bidding on their own during the experiment. The experiment is divided in to two parts. In the first part, the game will be run on the emulator on computer and participants need to bid on items by clicking on the emulator with mouse. In the second part, the game will be run on a mobile phone and participant have to bid on items with their hands. We use Python to perform mouse logging on the computer and use the *getevent()* command through the ADB platform tools to obtain the touch events through the kernel of our rooted android device. After retrieving the touch input coordinates, we plot the results in graphs and carry out k-mean clustering to cluster the data points using sklearn in Python after determining the optimal number of clusters of the graphs by the Elbow method. We then calculate the

Table 2: Summary of detection features in FIFA Mobile

Input Type	Adjusted E.D.	Clicks per second
Auto clicker	0.0150	1.1993
Human (Computer)	0.0289	0.4566
Human (Mobile)	0.0255	0.4854

Table 3: Summary of detection features in NBA Live Mobile

Input Type	Adjusted E.D.	Clicks per second
Auto clicker	0.0165	1.1056
Human (Computer)	0.0306	0.5278
Human (Mobile)	0.0308	0.5291

average and adjusted Euclidian Distance of the data points from their centroids. Within each of the same dataset, we also record the duration of the bidding process and calculate the average number of clicks per second for analysis.

6.2 Results

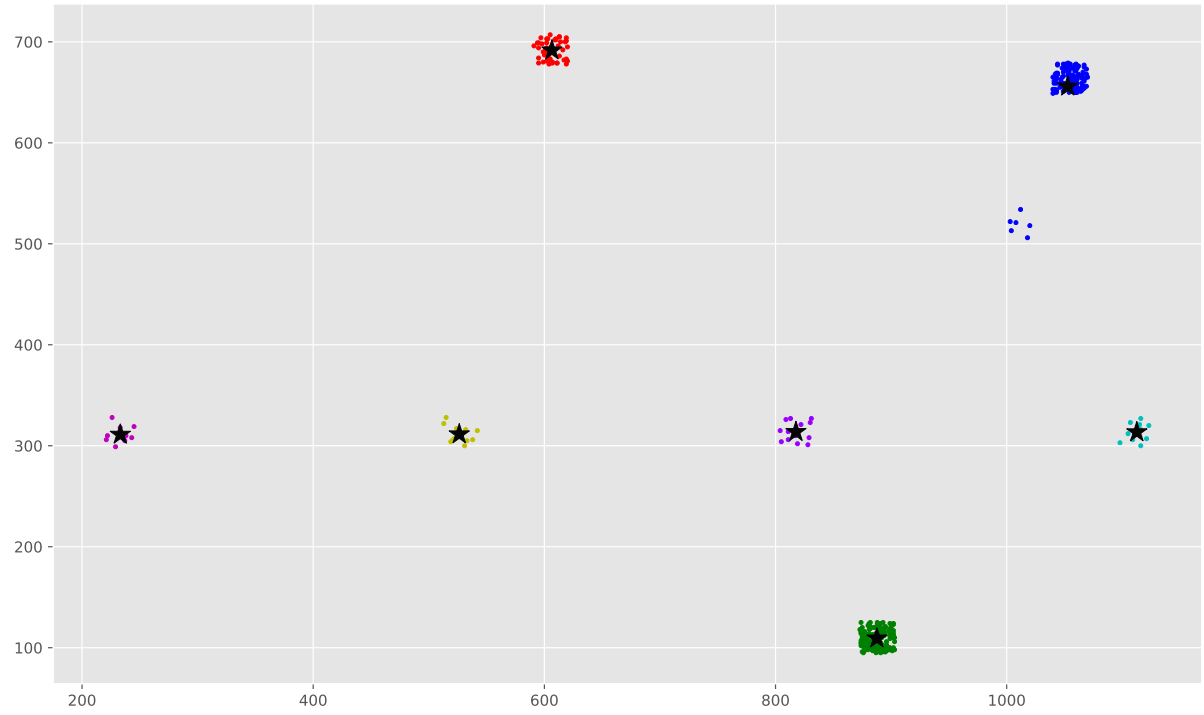

Figure 5: Touch input distribution of auto clicker for FIFA Mobile

Figure 5, 6 and 7 visualize the distribution of the touch input of our auto clicker and participants in FIFA Mobile and 8, 9 and 10 show the distribution of the touch input in NBA Live Mobile. Each colour represents a cluster and their corresponding centroids positions are indicated by the stars. We can see that the inputs are more disperse for human users in figure 6, 7, 9 and 10. The degree of dispersion of each cluster varies, depending on the actual size and shape of the clickable area in the UI corresponding

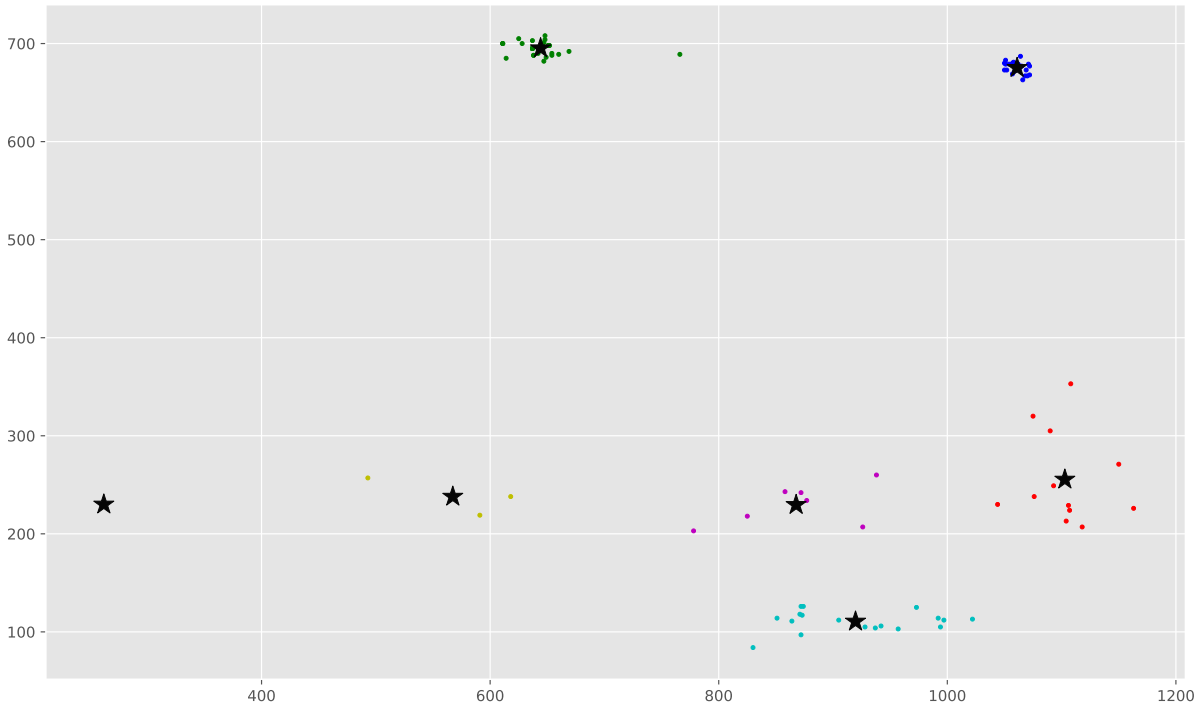

Figure 6: Touch input distribution of human user using emulator for FIFA Mobile

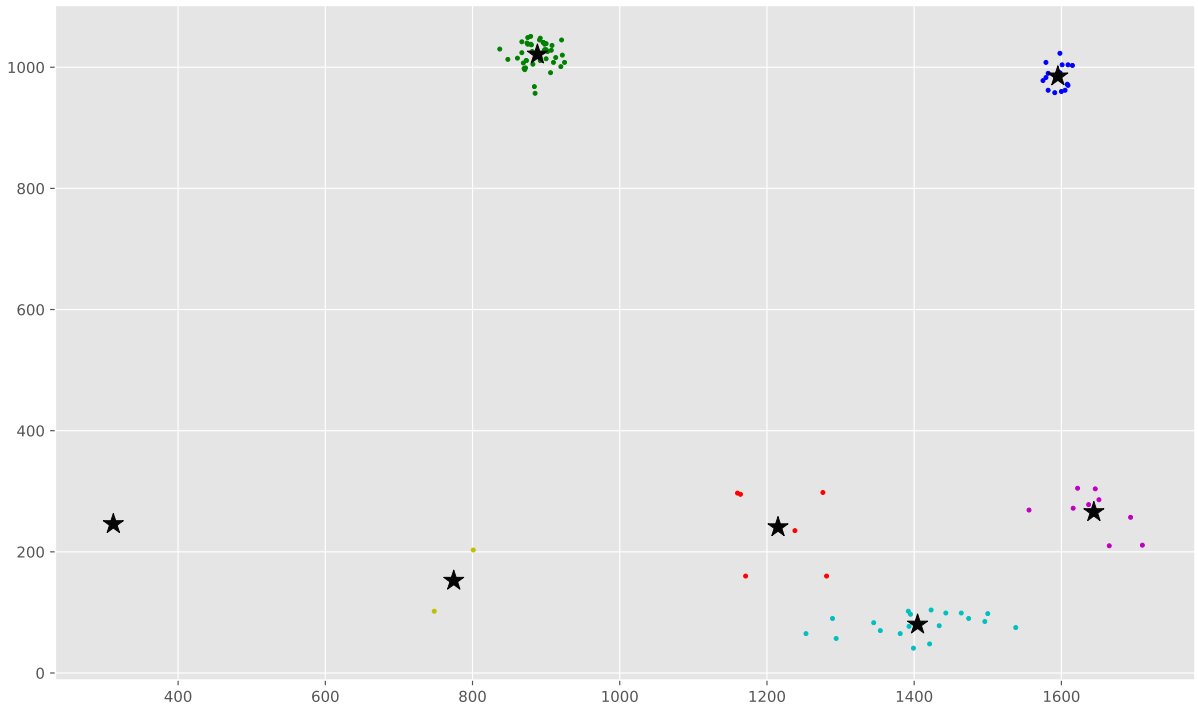

Figure 7: Touch input distribution of human user using mobile phone for FIFA Mobile

to the cluster. Table 2 and 3 summarizes the results of our experiment. We can see that the dispersion of

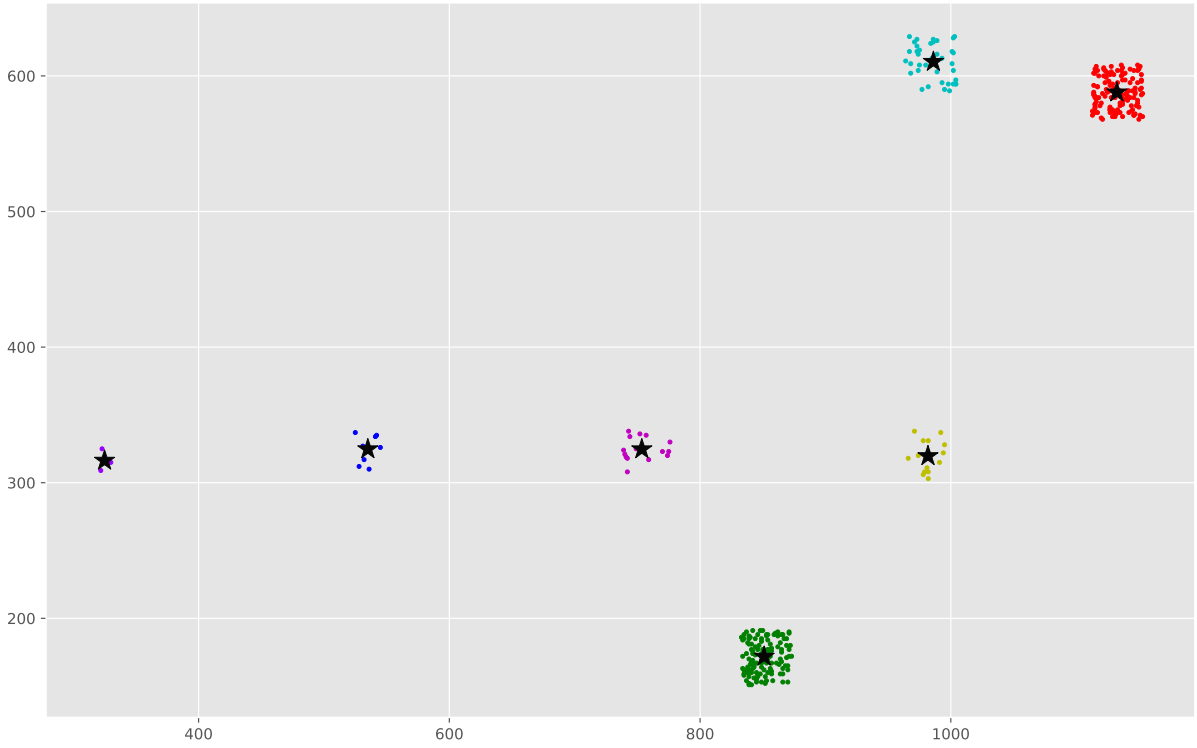

Figure 8: Touch input distribution of auto clicker for NBA Live Mobile

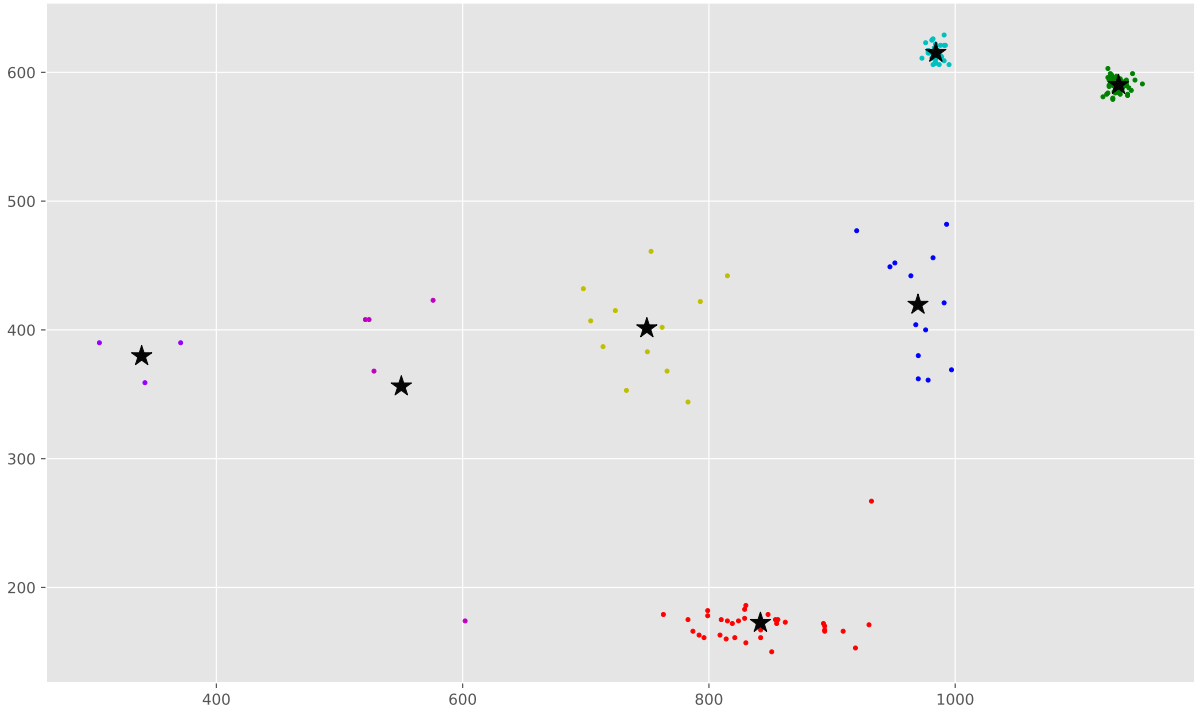

Figure 9: Touch input distribution of human user using emulator for NBA Live Mobile

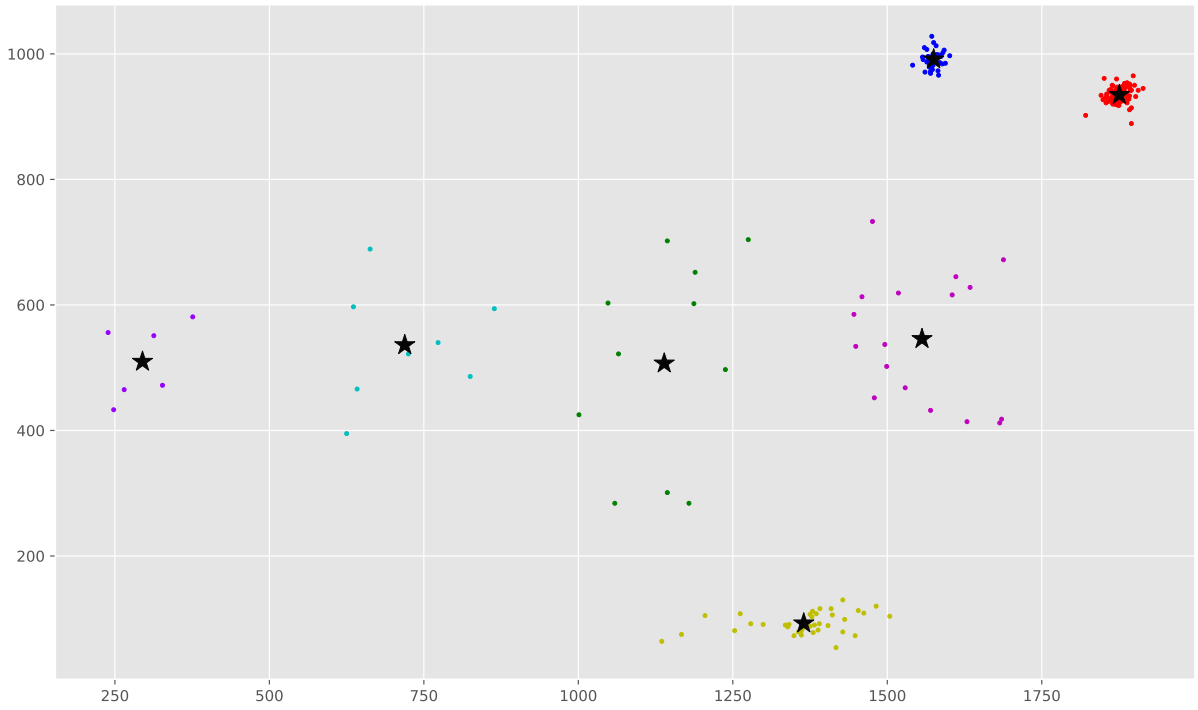

Figure 10: Touch input distribution of human user using mobile phone for NBA Live Mobile

touch inputs (Adjusted E.D.) for human users is significantly higher than that of our auto clicker (0.0150 to 0.0289 and 0.0255 in FIFA Mobile, 0.0165 to 0.0306 and 0.0308 in NBA Live Mobile), which agrees with our expectation. The number of clicks per second of our auto clicker is more than double (1.1993 to 0.4566 and 0.4854 in FIFA Mobile, 1.1056 to 0.5278 and 0.5291 in NBA Live Mobile) compared to human inputs due to the ignorance of reaction and movement time in auto clickers.

6.3 Practicability

In our experiment, we retrieve the input coordinates through third-party programs (i.e. mouse logging with Python and kernel command in Android) since it is not possible for us to access and modify the game code directly. In real implementation, the work is in fact much easier. In Android OS, developers can implement the *OnTouchListener()* in Java to obtain the touch input coordinates. In IOS, developers can get the coordinates in Objective C using *locationInView()* in UITouch. The detection threshold on determining whether cheating occurs is game dependent but it always follows our detection logic (i.e. lower Adjusted E.D. and higher clicks per second for auto clickers). The memory consumption on storing the coordinates for detection is minimal. Assuming there is one click per second at most in average according to the result obtained in table 2. Each coordinate consumes 2 bytes and with the timestamp together for calculation on clicks per second (6 bytes altogether for each click, can be even smaller by storing only the starting and ending timestamps), about 0.5MB is required for logging on each user daily so the memory and network burden on the server is negligible and practical.

7 Limitations and future work

As our methodology highly depends on the clicking behaviour of the touch input, modifying the clicking behaviour of the auto clicker to make it more human-like can possibly escape from our detection. Cheater can sacrifice the performance of the auto clicker by lowering down the clicking frequency to reduce the number of clicks per second, and even creating 'artificial errors' by clicking in random positions to increase the Adjusted Euclidean Distance. Therefore, more detection features should be implemented to avoid such problem.

Since our detection method depends highly on the clicking behaviour of the touch inputs, cheaters can fine tune such behaviour by adjusting the clicking frequency and positions on the auto clickers accordingly to sacrifice its effectiveness in order to escape from the detection (i.e. slowing down clicks to reduce average clicks per second and increasing random offsets in clicking positions to increase Adjusted E.D.). More detection features should be implemented to avoid such problems and we are going to put it as our future work. Researches will also be done on investigating whether our methodology can be extended to efficiently detect auto clicking in more game types.

8 Conclusions

In this paper, we proposed a detection methodology to detect auto clickers in mobile games by analysing the touch inputs of client users. We cluster the touch inputs and calculate the average Euclidian distance of the coordinates with their clusters and divide it by the screen resolution to obtain the adjusted Euclidian distance. We also calculate the number of clicks per second by taking average of the number of clicks performed in a certain time period. We expect that auto clickers would give a lower adjusted E.D. and a higher number of clicks per second. Experiment results shows that our auto clicker gives a higher adjusted E.D. to human inputs (0.0150 to 0.0289 and 0.0255 in FIFA Mobile, 0.0165 to 0.0306 and 0.0308 in NBA Live Mobile) and a doubled number of clicks per second (1.1993 to 0.4566 and 0.4854 in FIFA Mobile, 1.1056 to 0.5278 and 0.5291 in NBA Live Mobile), which agrees with our expectation. The implementation of our method is simple in programming means and the network and memory consumption are negligible with around 0.5MB per user daily.

Acknowledgement

This project is partially supported by the CRF grant (C1008-16G) of the Government of HKSAR.

References

[1] "EA Sports," https://www.easports.com [Online; accessed on September 2, 2019].

[2] "FIFA mobile," https://www.ea.com/games/fifa/fifa-mobile [Online; accessed on September 2, 2019].

[3] "GameGuardian," https://gameguardian.net [Online; accessed on September 2, 2019].

[4] "iPadian," https://ipadian.net [Online; accessed on September 2, 2019].

[5] "Madden NFL Mobile," https://www.ea.com/games/madden-nfl/madden-nfl-mobile [Online; accessed on September 2, 2019].

[6] "Mobile games revenue worldwide from 2013 to 2017 (in billion u.s. dollars)," https://www.statista.com/statistics/536433/mobile-games-revenue-worldwide [Online; accessed on September 2, 2019].

[7] "NBA Live Mobile."

[8] "Nox App Player," https://www.bignox.com [Online; accessed on September 2, 2019].

[9] "Pulover's Macro Creator – The Complete Automation Tool."

[10] "Valve Anti-Cheat System (VAC)," https://support.steampowered.com/kb/7849-RADZ-6869/valve-anti-cheat-system-vac [Online; accessed on September 2, 2019].

[11] M. A. Ahmad, B. Keegan, J. Srivastava, and N. C. Dmitri Williams, "Mining for Gold Farmers: Automatic Detection of Deviant Players in MMOGs," in *Proc. of the 12th IEEE International Conference on Computational Science and Engineering (CSE'09), Vancouver, British Columbia, Canada*. IEEE, August 2009, pp. 340–345.

[12] Y. Chung, C. yong Park, N. ri Kim, H. Cho, T. Yoon, H. Lee, and J.-H. Lee, "A Behavior Analysis-Based Game Bot Detection Approach Considering Various Play Styles," *Etri Journal*, vol. 35, no. 6, pp. 1058–1067, December 2013.

[13] S. Gianvecchio, Z. Wu, M. Xie, , and H. Wang, "Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs," in *Proc. of the 16th ACM Conference on Computer and Communications Security (CCS'09), Chicago, Illinois, USA*. ACM, November 2009, pp. 256–268.

[14] P. Golle and N. Ducheneaut, "Preventing bots from playing online games," *Computers in Entertainment*, vol. 3, no. 3, pp. 3–3, July 2005.

[15] A. R. Kang, J. Woo, J. Park, and H. K. Kim, "Online game bot detection based on party-play log analysis," *Computers and Mathematics with Applications*, vol. 65, no. 9, pp. 1384–1395, May 2013.

[16] H. Kim, S. Hong, and J. Kim, "Detection of Auto Programs for MMORPGs," in *Proc. of the 2005 Australasian Joint Conferenc on Artificial Intelligence (AI'05), Sydney, Australia*, ser. Lecture Notes in Computer Science, vol. 3809. Springer, Berlin, Heidelberg, December 2005, pp. 1281–1284.

[17] D. Liu, X. Gao, M. Zhang, H. Wang, and A. Stavrou, "Detecting Passive Cheats in Online Games via Performance-Skillfulness Inconsistency," in *Proc. of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17), Denver, Colorado, USA*. IEEE, June 2017, pp. 615–626.

[18] S. Mitterhofer, C. Kruegel, E. Kirda, and C. Platzer, "Server-Side Bot Detection in Massively Multiplayer Online Games," *IEEE Security & Privacy*, vol. 7, no. 3, pp. 29–36, May 2009.

[19] R. Thawonmas, Y. Kashifuji, and K.-T. Chen, "Detection of MMORPG bots based on behavior analysis," in *Proc. of the 2008 International Conference on Advances in Computer Entertainment Technology (ACE'08), Yokohama, Japan*. ACM, December 2008, pp. 91–94.

[20] M. van Kesteren, J. Langevoort, and F. Grootjen, "A step in the right direction: Bot detection in MMORPGs using movement analysis," http://wwwis.win.tue.nl/bnaic2009/papers/junk/bnaic2009_submission_95.pdf [Online; accessed on September 2, 2019], 2009.

[21] M. Varvello and G. M. Voelker, "Second life: a social network of humans and bots," in *Proc. of the 20th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'10), Amsterdem, The Netherlands*. ACM, June 2010, pp. 9–14.

[22] R. V. Yampolskiy and V. Govindaraju, "Embedded noninteractive continuous bot detection," *Computers in Entertainment*, vol. 5, no. 4, pp. 7:1–7:11, January 2007.

[23] J. Yan, "Bot, Cyborg and Automated Turing Test," in *Proc. of the 2006 International Workshop on Security Protocols (Security Protocols'06), Brno, Czech Republic*, ser. Lecture Notes in Computer Science, vol. 5087. Springer, Berlin, Heidelberg, April 2006, pp. 190–197.

[24] S. Yeung, J. Lui, J. Liu, and J. Yan, "Detecting Cheaters for Multiplayer Games: Theory, Design and Implementation," in *Proc, of the 3rd IEEE Consumer Communications and Networking Conference (CCNC'06), Las Vegas, Nevada, USA*. IEEE, January 2006, pp. 1178–1182.

[25] S.-Y. Yu, N. Hammerla, J. Yan, and P. Andras, "Aimbot Detection in Online FPS Games Using a Heuristic Method Based on Distribution Comparison Matrix," in *Proc. of the 2012 International Conference on Neural Information Processing (ICONIP'12), Siem Reap, Cambodia*, ser. Lecture Notes in Computer Science, vol. 7667. Springer, Berlin, Heidelberg, December 2012, pp. 654–661.

Author Biography

Shing Ki Wong received his Bachelor degree from Kore University of Enna, Italy. From young age he has developed a passion for computer science and electronics, and now he is continuing his studies to obtain the Master's degree in Computer Engineering at University of Catania, Italy.

Siu Ming Yiu received the Bachelor Degree in Telecommunication Engineer from University of Catania in 2006. He also received the Master Degree in Telecommunication Engineering from University of Catania in 2010. Currently, he is a Ph.D. student at Kore University of Enna. His current research interests include ITS, driverless vehicles and network architecture.