Bot Detection in Online Games

Phou Lee
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
leex5047@morris.umn.edu

ABSTRACT

Prevention and detection of automated bots is an important part of multiplayer online games. Botting is a form of cheating that goes against game developers' term of service. Specifically in massively multiplayer online games, players can bot to gain unfair benefits such as virtual resources or to automatically battle with monsters. This forces game developers to continuously create new countermeasures against botting. In this paper we will review the fundamentals of botting and the current techniques used to detect it. We will discuss the development of the following two efficient bot detection techniques: Waypoint with Path Segments and Party Play Analysis. We will then summarize our discussion of bot detection and the importance of it.

Keywords

Bot detection, Online gaming, Algorithm, Server-side, MMOG

1. INTRODUCTION

The popularity of online gaming has increased greatly in the last decade. Players enjoy online gaming because it introduces a play style where they can compete with other players. There are multiple online games. The most popular games at the moment are *World of Warcraft* and *League of Legends*, both games have attracted over a million players. With the support of the game community, online gaming has become a very profitable industry for game developers. However, the community has also attracted a form of cheating known as automated botting that is used for illegitimate gains.

The definition of *automated botting*, or simply known as botting, is described as the use of a bot to play the game in substitution of the player [8]. Botting is against most game developers' term of service. A bot is a software-based program that runs a script composed of game actions. Inputting the actions like how peripherals do to the game, the program can play it in place of the player. Usually the actions are for game tasks that can easily be managed by the player, but they do not want to commit their time to perform them.

Massively Multiplayer Online Games (MMOGs), online games that can sustain a large number of players, are the primary targets for botting. There are a few types of MMOGs but the most common types are *Role-Playing Game* (RPG) and *First Person Shooter* (FPS). Both of these type function like most games do. Players are represented as a virtual objects in the game world, and they control the object with peripherals such as a mouse or a keyboard. The game developer introduces game contents for players to have fun and to compete with each other. For competition wise, players would want to obtain the best items or rank. To obtain them though requires the player to perform time consuming tasks. These tasks, however, could become repetitive and boring causing players to lose interest in it. This introduces the use of botting to perform repetitive tasks.

We can categorized the benefits of botting into two groups: *score* and *asset*. Score is defined as the ranking or the measurement of a player's skill in the game. FPS online games fall heavily into this category where skills are very important. However, a player can use a bot to obtain a good score without having the necessary skills. An example would be the measurement of a player's shooting accuracy. A bot program known as an aimbot would be able to obtain a perfect or near perfect shooting accuracy. The second category, asset, is when botting is used to obtain virtual resources. Some MMOGs have implementations of a virtual economy that allow players to trade or to gather resources in the game. These resources are used by players to increase the value of their assets. The common example that revolves around asset are RPGs with crafting materials. Crafting materials can be gather in the game, and the task is extensively repetitive. Bots though, could be programmed to gather the materials with little effort on the player's part. These two benefits of botting can severely harm the enjoyment in a MMOG or its economy. If botting cannot be prevented by game developers, the end result could be legitimate players leaving the game, and the developers losing revenue.

This paper will review bot detection techniques as well as describe two efficient approaches to detect bots. Section 2 defines terminology, and it will also discusses the types of bot detection technique and their efficiency. Section 3 describes a movement analysis approach that uses waypoints with path segments, and in Section 4 we review a party play analysis approach. Section 5 summarize our approaches with current detection approaches.

UMM CSci Senior Seminar Conference, December 2013 Morris, MN.

2. BACKGROUND INFORMATION

This section will review the terminology necessary for bot detection techniques. We will discuss the different types of bot detection and their efficiency.

2.1 Terminology

The following terminologies are widely used and are defined in the interest of bot detection. These terminologies may vary for each online game.

- *Online Game*: An online game is always running and is played by using the game's client. Clients are downloaded by players and allow them to send information to the game. The game's role is to accept the information from its clients and verifies that the information is legitimate. It then sends back information to the clients allowing updates to their user interfaces.
- *Log*: Logs are archived information from activities relating to the game. They consist of all the game's information and all the information from clients. Logs can only be accessed by game developers and are not accessible to the public. Bot detectors would primarily focus on the client informations from logs. An example of a client's information in a log would look like <Timestamp, Coordinate, Action> [5].
- *Trace/Tracing*: A trace is similar to a log except it does not record server information. A trace is created from programs that record a client while it is running and can obtain the client information. Like logs, the information from the client are similar. They are used by both developers and players for debugging and reviewing.
- *Timestamp*: The real time taken when information is sent by the client or the game.
- *Action*: An action is an interaction the player takes in the game. Actions can be anything the player does. Actions can include but are not limited to: login, logout, battle, trade with other players, chat [5, 7]. An example of actions in a FPS game would be: turning inside the game, aiming, shooting, and being blocked by virtual objects.
- *Coordinate*: A Coordinate is a set of numbers identifying where a player is located in the game, and a movement is a set of coordinates. Coordinates are important because an action requires the game to verify the player's location before it can happen. An example is that a three dimensional game would have three dimensional coordinates, and the game must verify an attack's length before it can occur.

2.2 Types of Bot Detection

Bot detection techniques are classified into the three following types [10].

- *Network-side*: The majority of network-side detection monitors the traffic of information going and coming to the game. This type of detection constantly runs in real time to make sure the information from clients are legitimate. An example of a network-side approach is that bots and players have different traffic patterns. A detection that checks the traffic for time interval patterns is able to determine if a client is botting.
- *Client-side*: Client-side detections seek to detect players who tries to use third-party programs that will affect game client. If it detects a third-party program it will report the client to the game, and the game can block the client until further notice. This detection attempts to run silently in the background when the client is running.
- *Server-side*: Server-side detections happen entirely at the game's location. This detection type seeks to find bot patterns from passed client information. Client information can be obtained from logs or traces. Once it determines enough information from a client, it can determine if the client is botting.

Out of the three detection types, our focus will on server-side detection because it is more efficient [13]. Efficiency is defined as the least cost and the least effort for game developers if the detection type is implemented in a real environment.

Network-side detectors monitor between clients and the game. There does not seem to be any advantage to this as it is possible to monitor all activities from the server-side. In addition, to monitor all the traffic coming and going requires a network-side detector to run as long as the game is active. Since the game is online and always active, this would add extra computational power to the cost.

Client-side detectors are the most common defense because it is easy for game developers to implement. The detection program is set to start up when the player starts their client. This detection type runs on the player's computer so developers do not have to worry about computational power. This way when a player attempts to start a bot program, it can tell the game to block the client. Client-side detectors are usually set to only report known bot programs and can not recognize new ones. This allows new bot programs to run undetected until developers update their detection program. If developers decide to allow their client-side defense to search for new bot programs, they can definitely detect better. However, this will cause problems to players when programs are mistakenly considered as bot programs. Given that players are the most important part of the game, it is assumed that developers would not risk losing them. Also, for developers to troubleshoot incompatibility issues would be a time consuming task.

Server-side detectors would be the best choice. Detection only happens on the game side so problems that occur would not affect the clients. This detection type aims to find patterns from information that has already been record by logs and traces. It will not be able to prevent botting the instant a bot program starts but by using archived information to determine botting, it would cost less computational power. Simply put, unlike network-side detectors, it would not have to run constantly. It will run when there is enough information to classify a client as a bot or a player. Lastly, server-side detectors are hidden from clients allowing the assurance that bot programs will have a hard time trying to counter them.

3. WAYPOINT WITH PATH SEGMENTS

In this section we will review a movement analysis approach for bot detection. This server-side approach focuses on one important feature of bots: they all have significantly

larger movement repetitions over time in comparison to players. Bot programs have a limited number of specific movements before they reach the end of the program. They will then start from the beginning of the program and repeat the process for a long time. Players though, do not have specific movements and have a high variance in their movement. As such, the authors use the feature to develop measurements on movement repetitions from traces and logs. Then after distinguishing a reasonable threshold on the measurements, they can classify which trace or log belongs to a bot.

Mitterhoffer et al. describe two measurements, suffix array with longest common prefix and average path segment, for classifying bots in their algorithm [6]. We will review the measurement of the average path segment approach. The approach will attempt to create waypoints out of meaningful clusters of coordinates. These clusters are locations that are most traversed.

3.1 Algorithm

The algorithm takes a trace as the input and simply graphs it in a N-Euclidean space, where N is the dimension of the trace. Each coordinate of the trace can be treated like a vertex. They will have connections, or edges, with the previous and next timestamped coordinate in the trace. For example, coordinate x_1 is connected to x_2, and coordinate x_2 is connected to x_1 and x_3. The result after all the connections is a line segment from the first to the last timestamp coordinates.

The line segment of the trace will have a lot of unnecessary coordinates that gives little information about repetitious movement. These points may be when the player is not moving, turning, or moves very slowly, and they are commonly clustered together in the line segment at close timestamps. The Ramer-Douglas-Peucker line simplification algorithm (RPD) is used to reduce the unnecessary coordinates in the line segment [6]. The RDP algorithm takes a line segment and can produce a new line segment with considerably less points [9]. The RPD algorithm removes some coordinate points of the line segment but what is left are the important coordinate points. The RPD simplifies the line segment, but more importantly it removes the cluster of points at close timestamps. The simplified line segment will still consist of clusters, however these are cluster of points at very different timestamps and are important for finding repetition. Figure 1 shows an example of a simplified line segment [6].

From the simplified line segment, the algorithm creates *waypoints* from its clusters. Clusters are found by using a custom clustering algorithm [6]. If clusters overlap, only the bigger clusters are created into waypoints. A waypoint is the diameter that encloses a cluster of points. Players and bots will create waypoints while they explore the game. Bots though, will stop exploring the game due to their repetitious movement and come back to traverse old waypoints. This means that bots will have a limited number of waypoints while a player continues to create new waypoints.

A path segment is the path from one waypoint to another waypoint, and the average path segment is the measurement for classifying bots. The authors decided to measure the average path segment instead of average waypoints because it was found to be more robust [6]. The average path segment is calculated as the total path segments traversed divided by the distinct number of path segments. After a certain time,

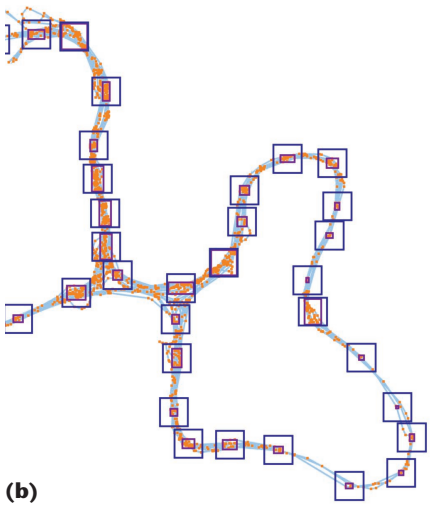


Figure 1: A graph of a simplified line segment taken from [6]. Clusters are found with a custom clustering algorithm. Each square represents a waypoint and between each waypoint is a path segment.

bots will stop making new waypoints and they will traverse the same path. This increase the total paths traversed while distinct waypoints stay the same. This means the measurement will increase for bots and stay low for players. They then can then set a threshold on the measurement. The threshold is set at a number that we know players will not likely pass to but bots can easily.

3.2 Performance Evaluation

The performance was evaluated with the online game World of Warcraft (WoW), a MMOG developed by Blizzard Entertainment. A private WoW server was set up so traces could be collected. Traces were collected from ten players and two botting programs. A four hour trace was collected from each player. The players consist of beginners who have never play the game before to very experience players. They were told to focus on battling and gathering resources so their movements may resemble bots. The bots consist of the free ZoloFighter Bot and the Glider Bot [6]. Two different traces were collected from each of the bots, and each trace consisted of a different movements provided by the bot program.

In total, fourteen traces were collected and used their algorithm. Figure 2 shows the results of the average path segments as time increases. The results showed that as time passes, the average path segments of bots increased. All four traces for the bots continued to increase. Players on the other hand stayed below the average path segment measurement of two. The threshold for measurement was set at five, which detected all four of the traces within 12 to 60 minutes.

4. PARTY PLAY ANALYSIS

Party play is essentially when players party together in a game to take on game content. Partying is often common and it is an important part of RPGs. In RPGs, parts of the game require partying tackle game content. Parties can also ideally take on content that a single player can. Players

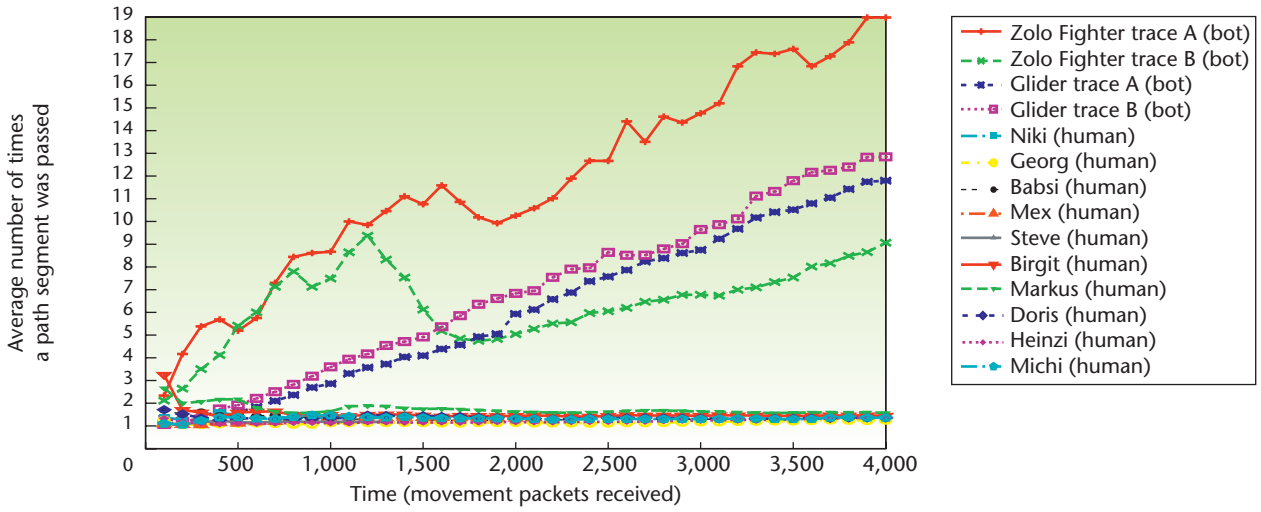


Figure 2: The average path segments for their performance evaluation. As assumed, the average way points of bots continue to increase as time increases. Zolo Fighter Bot B had a sharp drop at 1200 time packets because a player took over until at 1700 time packets. [6].

usually party to social, to have fun, and to compete in hard content for high rewards. Bot programs take partying as an advantage and use it to gain benefits. Developed by Ah Reum Kang et al. this section reviews a party play analysis approach that attempts to distinguish bots from players through parties [4].

This approach considers the feature that bots will always attempt to gain benefits, and they will use partying to their advantage. In comparison to players, bots compose parties for an entirely different goal. They will always party together to reap benefits. The benefit that bots are interested in would be assets as described in Section 1. The major assets are items, gold, and experience points. A RPG's resources can be gathered all around the game's maps. The majority of resources are guarded by monsters that must be defeated before they can be gathered. When a monster is slain and a resource is gathered by a player, they will receive experience points, gold, item, and the resource. If this event is taken in a form of a party, all the assets will have to be divided between the party. As such, it is better for a single player to obtain assets by themselves. For bots though, the majority of them will be programmed to do only a specific sequence of tasks; either to kill monsters or to gather resources. Being programmed to do multiple tasks may be too hard or inefficient for bots. Hence it is easier for bots with different tasks to party up and gain assets together.

The algorithm of this approach will focus on the actions taken in parties. Using logs provided by the game, it attempts to create rules to classify bots. The algorithm will go as follow:

1. Distinguish outlier parties from the party log and use the outlier parties to find potential bots from them

2. Find the significant actions from rankings of players and rankings of potential bots

3. Create rules to classify bots base on the significantly actions

4.1 Algorithm

Knowing some difference between bot and player parties, the authors assume the following:

1. The duration of bot parties is larger than normal parties.

2. Bots will most likely be in a party of two; one that gathers and the other kills monsters.

These assumptions will be used to denote parties, whom are outlier parties, as potential bot parties [4]. To start, the algorithm will find outlier parties from party logs acquired from the game. Party logs should have duration of parties and party actions recorded. Party actions are actions taken while partying. Finding the outlier parties is an easy task, since they figure that players have short term goals, they will disband their parties when their goals are met. Bots though will party for a large time doing repetitive tasks. The algorithm simply graphs the cumulative distribution function (CDF) of party duration, and find a reasonable point in time where player parties will likely disband. Figure 3 gives an example of a CDF. From the figure it shows that most parties do not last longer than four to five hours, which can be the cut off for separating outlier parties from regular parties.

After distinguishing outlier parties, they want to find out which outlier party is more likely to be bot parties. Using assumption two from above, they figure that having more than two bots in a party is inefficient and having a bot party of one is unnecessary [4]. Figure 4 shows two pie graphs of the numbers in parties at different time. This figure shows that long duration parties consist of two and it follows with their assumption. The players within these parties are denoted as potential bots.

To obtain rules to classify bots, the algorithm will attempt to find significant actions in both the actions from the potential bots and the regular players. A significant action is an action that changes as time increases. A ranking for their actions is created for each group. The ranking is a list that defines which actions occurs more. The algorithm will look

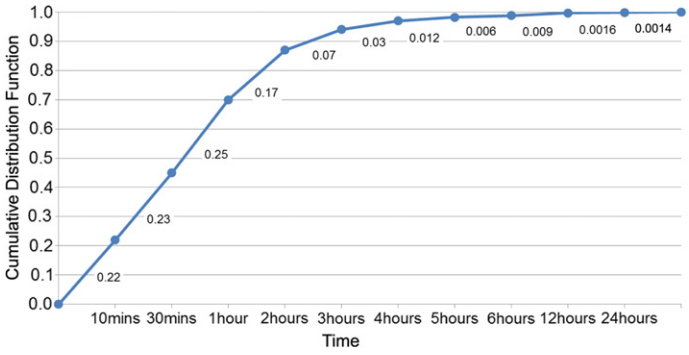


Figure 3: A CDF graph from their performance evaluation [4].

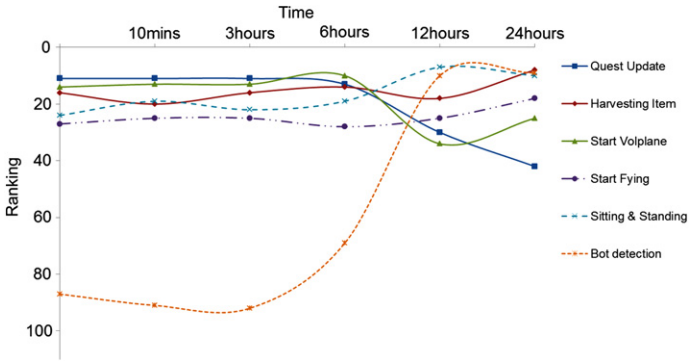
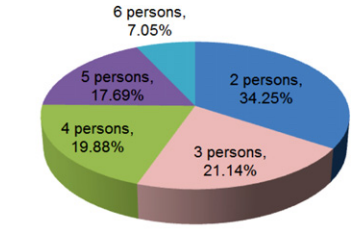


Figure 5: The rank representation of the top significant actions from potential bots. It is unsure what the bot detection action is from the authors [4].

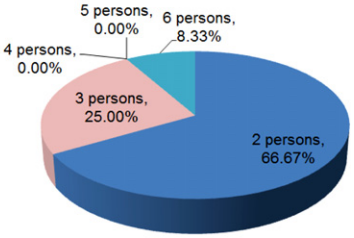
for the most difference in terms of rank change as time increases for regular actions and for potential bot actions [4]. An example is if questing was on the top ten actions taken by regular players at three hours while it is above the fifty rank at 12 hours, it would be considered as a significant action.

The algorithm takes the top significant actions and sets a threshold for each of them to create a rule base. Thresholds are set depending on how important the action is to the game, and how it appears on the ranking. Actions may vary for each game so a reasonable threshold should be set for each action. Using this rule base, party logs can be taken from the game and run all the players through it to classify bots.

4.2 Performance Evaluation



(c) t+3h



(f) t+24h

Figure 4: Pie graphs at three hours and twenty-four hours. At twenty-four hours two player parties were the majority [4].

The performance for this approach was tested in the MMOG game Aion [4].The game developer, NCsoft, Inc. permitted the use of their logs for this performance evaluation, with the addition of also verifying the accuracy of their approach. Seven days worth of party logs were obtained, and 63,092 parties were extracted from them. As seen in Figure 4, these are the graphical representation of the party logs [4]. As assumed, when party duration increases, party of two is shown to be the majority. The significant actions for regular players and for potential bots are shown in Figure 5 and Figure 6. The ranking of each shows the top significantly actions obtained from each of the group. From these data, classifiers were created as seen in Table 1. Classifiers were set at reasonable thresholds depending on an action's rank and the importance of it to the game. The rule base shows that bots took advantage of sitting still to regenerate health points before going back to battle. The quest completion became a classifier because bots have no need to participate in quests from the game. Taking these classifiers and applying them to 52,377 players they were able to classify 49 bots.

To evaluate their accuracy, NCsoft, Inc. compared the classified bots to their internal detection tool's data, and also attempt to confirm the authors' findings [4]. Details of

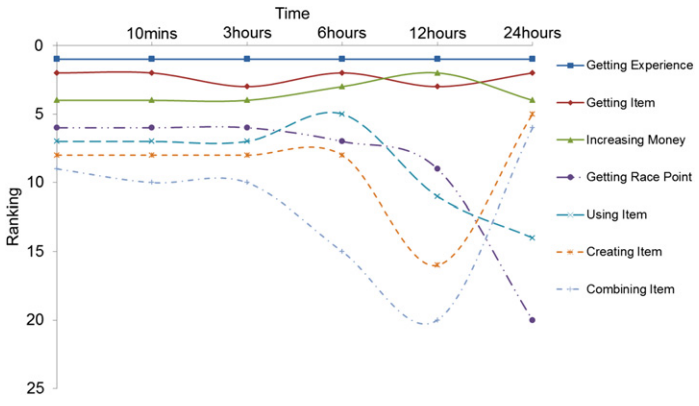


Fig. 7. The most frequent logs by normal user.

Figure 6: The rank representation of the top significant actions from players. [4].

Rules
Getting experience log >= 34%
Getting race point log <= 1.69%
Standing and sitting log <= top 10 rank
Using item log <= 1.19%
Quest completion log <= 0.16%
Start volplane log >= top 34 rank
Party member = 2 and party duration >= 600

Table 1: Bot rule base for classifying [4]. Volplane is a flying ability unique to the game Aion.

how their internal detection tool is unknown because NCsoft, Inc. did not permit it. Out of the 49 bots detected, 26 were detected by the internal tool. From the 23 bots undetected, NCsoft were able to verify that 21 were bots and 2 were false positives. In all, the algorithm had a 95.92% accuracy for detecting bots. More importantly, this algorithm was able to detect bots that the internal tool did not. This shows that this approach is a potential tool to help increase detection of bots.

5. CONCLUSION

Bot detection is important part of online gaming and is a potential tool for preserving fairness in a game. An online game's main purpose is to introduces fun and competitive contents for players. Without a defense technique to protect the fairness of a game though, bots can literally alter the economy of a game or take away the fun and competitiveness for players. Games would lose their purpose causing legitimate players to leave the game. This is an issue for game developers whose revenue comes from their players. This paper focused on the fundamentals of botting and bot detection in online gaming. We reviewed two different types of server-side detection approaches. Server-side bot detections were the main focus in this paper because they can be implemented with the least cost.

Server-side detection are starting to become more implemented by game developers. This type of detection is important because developers do not include anything on the client side thus players problems do not occur. Additionally, server-side approaches aims at finding features that distinguishes bots from players. The server-side approaches reviewed in this paper were a movement analysis approach and a party-play analysis approach. We showed that bots can be efficiently detected by these approach and that they have the potential to help bot detection in online games. There are more interesting approaches that has also been developed that we did not review but are worth mentioning. Mishima et al. introduces an approach focusing on specific battling actions [5]. Chow et al. introduces an approach that integrates CAPTCHAs into minigames to remove immersion [1].

As of now, the most common type of bot defenses are implemented as client-side types. For game developers, client-side defensive approaches essentially take the least effort to implement. Common approaches usually revolve around the game's detection or prevention method that is required to be started when the client runs. These detection methods either make sure bot programs are not being run or they are interactive detections within the game. Examples of bot defenses that detect bot programs is WoW's Warden [2]. An example of interactive detection is the use of CAPTCHAs to check if a client is botting [3]. These detection approaches run on the player's hardware and cost relatively low in computational power for developers. However, the problems that arise from these type of detections decreased their efficiency. For defense approaches like the Warden, there are always incompatibility issues with programs that are not bot programs. For interactive detections like CAPTCHAs, they interrupt players in the game too much to be worth implementing. These problems drive up the cost for game developers in terms of customer support and troubleshooting.

6. REFERENCES

[1] Y.-W. Chow, W. Susilo, and H.-Y. Zhou. Captcha challenges for massively multiplayer online games: Mini-game captchas. In *Cyberworlds (CW), 2010 International Conference on*, pages 254–261, 2010.

[2] S. Gianvecchio, Z. Wu, M. Xie, and H. Wang. Battle of botcraft: Fighting bots in online games with human observational proofs. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 256–268, New York, NY, USA, 2009. ACM.

[3] P. Golle and N. Ducheneaut. Preventing bots from playing online games. *Comput. Entertain.*, 3(3):3–3, July 2005.

[4] A. R. Kang, J. Woo, J. Park, and H. K. Kim. Online game bot detection based on party-play log analysis. *Computers & Mathematics with Applications*, 65(9):1384–1395, 2013.

[5] Y. Mishima, K. Fukuda, and H. Esaki. An analysis of players and bots behaviors in MMORPG. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 870–876, 2013.

[6] S. Mitterhofer, C. Kruegel, E. Kirda, and C. Platzer. Server-side bot detection in massively multiplayer online games. *Security Privacy, IEEE*, 7(3):29–36, 2009.

[7] S.-H. Park, J.-H. Lee, H.-W. Jung, and S.-W. Bang. Game behavior pattern modeling for game bots detection in mmorpg. In *Proceedings of the 4th International Conference on Uniquitous Information Management and Communication*, ICUIMC '10, pages 33:1–33:5, New York, NY, USA, 2010. ACM.

[8] M. van Kesteren, J. Langevoort, and F. Grootjen. A step in the right direction: Bot detection in MMORPGs using movement analysis. In *Proceedings of the 21st Belgian-Dutch Conference on Artificial Intelligence*, 2009.

[9] Wikipedia. Ramer-douglas-peucker algorithm — wikipedia, the free encyclopedia, 2013. [Online; accessed 31-October-2013].

[10] J. Woo and H. K. Kim. Survey and research direction on online game security. In *Proceedings of the Workshop at SIGGRAPH Asia*, WASA '12, pages 19–25, New York, NY, USA, 2012. ACM.