Proceedings of the 21st Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 215–220
DOI: 10.15439/2026F1905
ISSN 2300-5963 ACSIS, Vol. 47

Server-Side Machine Learning for Cheat Detection in Massively Multiplayer Online Role-Playing Games

Konrad Operacz
0009-0005-7704-433X
APIS S.A.
Poland
Email: konrad.operacz@apissa.pl

Hubert Zembrowski
0009-0007-8757-593X
Institute of Computer Science
Cardinal Stefan Wyszynski
University in Warsaw
Warsaw, Poland;
APIS S.A.
Email: hubert.zembrowski@apissa.pl

Robert A. Kłopotek
0000-0001-9783-4914
Institute of Computer Science
Cardinal Stefan Wyszynski
University in Warsaw
Warsaw, Poland
Email: r.klopotek@uksw.edu.pl

***Abstract*—Cheating in massively multiplayer online role-playing games (MMORPGs) degrades gameplay fairness and causes significant economic harm to game operators. Existing anti-cheat mechanisms predominantly operate on the client side, making them susceptible to bypass by modified clients or external automation tools. This paper presents the Cheat Detection System (CDS), a server-side machine learning framework developed as part of an industrial R&D project and validated on a controlled MMORPG testbed. The CDS integrates three complementary algorithms: a one-class Support Vector Machine (SVM) for detecting temporally anomalous player actions, an Affinity Propagation clustering algorithm for identifying unnaturally regular bot movement patterns, and a multilayer perceptron (MLP) acting as a meta-level decision engine. A login anomaly module employs behavioral profiling to detect unauthorized account access. Experimental results demonstrate F-scores ranging from 0.81 to 1.00 across five cheat categories, with a CPU overhead of less than 0.02 percentage points when running alongside 100 concurrent players. The modular architecture allows straightforward extension to new cheat patterns without redeployment of the core system.**

***Index Terms*—cheat detection, MMORPG, one-class SVM, Affinity Propagation, anomaly detection, game security, server-side anti-cheat, machine learning**

I. Introduction

ONLINE games, and MMORPGs in particular, represent a multi-billion-dollar industry in which in-game economies and player progression carry real monetary value [1]. Cheating undermines both the fairness and the commercial viability of these platforms. Common cheat categories include: (1) automation bots that farm experience points or in-game currency without human input; (2) movement exploits that teleport characters to arbitrary positions; (3) auto-healing scripts that trigger protective abilities faster than human reaction time allows; (4) malicious packet injection targeting server stability; and (5) account hijacking through credential compromise. A broader taxonomy surveyed in [8] additionally identifies memory-editing tools (e.g., Cheat Engine [16]) that modify client-side game state at runtime, speed hacks that bypass server-side movement validation by manipulating local timing, and packet editors that craft or replay arbitrary protocol messages to trigger unintended server behavior. Account takeover via credential stuffing (automated replay of credentials leaked from third-party breaches) represents a distinct threat vector that does not require any modification of game client software [17].

Existing countermeasures such as BattleEye and GameGuard operate as kernel-level drivers on the player's machine [2]. While effective against unsophisticated attackers, these client-side solutions share a fundamental weakness: a sufficiently motivated attacker who controls the client hardware and software can enumerate, spoof, or bypass the monitoring process. Server-side detection, by contrast, observes only the stream of actions arriving at the server and is therefore impervious to client-side manipulation.

Prior work on server-side game cheat detection has explored statistical and sequence-based approaches [3], [4], deep learning classifiers trained on multivariate time series [5], and multi-view anomaly tracking [6]. However, industrial deployments must satisfy strict latency and resource constraints: detection logic must not materially increase server CPU load or introduce perceptible delays in game packet processing.

This paper presents the Cheat Detection System (CDS), designed and validated as part of POIR.01.02.00-00-0123/20, an NCBR-funded industrial research project conducted by APIS S.A. (Poland, 2020–2023). The system achieves robust detection across five cheat categories while adding less than 0.02 percentage points of CPU load to the host server. The main contributions of this paper are:

This work was carried out as part of the project "Zestaw innowacyjnych narzędzi wspierających rozwój i utrzymanie gier o tematyce MMORPG" (POIR.01.02.00-00-0123/20), co-funded by the National Centre for Research and Development (NCBR), Poland, under the Intelligent Development Operational Programme 2014–2020.

- A modular server-side cheat detection architecture integrating complementary ML algorithms through a shared action vector pipeline.
- An adaptation of Affinity Propagation to spatial movement fingerprinting in MMORPG worlds.
- A neural meta-decision layer (DecisionMaker) that learns from confirmed enforcement outcomes, enabling the system to improve over time.
- Quantitative evaluation across five cheat categories on a controlled MMORPG server, using ground-truth labels assigned by professional game masters and parallel human-versus-bot capture.

II. Related Work

A. Client-Side Anti-Cheat

Solutions such as Valve Anti-Cheat (VAC), BattleEye, and GameGuard use kernel-mode drivers, memory scanning, and process integrity checks [2]. Their common limitation is susceptibility to virtualization and client modification, as comprehensively surveyed in [8].

B. Sequence-Based Bot Detection

Tao et al. [3] proposed NGUARD, a bot detection framework deployed across multiple NetEase MMORPG productions. NGUARD combines supervised classifiers on session-level features with unsupervised clustering to detect both known and novel bots. Xu et al. [4] extended this work with NGUARD+, an attention-based framework using Bidirectional LSTM and Hierarchical Self-Attention Networks to capture long player behavior sequences. Both frameworks operate on server-side logs, making them architecturally similar to our CDS; however, neither employs spatial movement fingerprinting nor a neural meta-decision layer with enforcement feedback.

C. Deep Learning on Multivariate Time Series

Pinto et al. [5] applied convolutional neural networks to multivariate time series of mouse and keyboard events, achieving 99.2% accuracy for triggerbot and 98.9% for aimbot detection without relying on in-game data. Su et al. [7] used Gaussian Mixture Models on player trajectories for mobile MMORPG bot detection, demonstrating the importance of movement path analysis. BEAT [6] combines game settlement data, click trajectories, and movement trajectories into multi-view representations, using supervised and semi-supervised models. Recent surveys [8] also point to graph neural networks as a promising direction for modeling interactions among multiple players, which we identify as future work.

D. Clustering-Based Approaches

Chung et al. [9] proposed K-means clustering of player behaviour followed by local SVM classifiers trained per cluster, allowing the system to account for different play styles when distinguishing bots from humans. This cluster-then-classify paradigm inspired the spatial segmentation used in our Affinity Propagation module.

E. One-Class SVM for Anomaly Detection

One-class SVM, introduced by Schölkopf et al. [11], has been widely applied to network intrusion detection [12] and user behavior analytics. Alayed et al. [10] demonstrated its effectiveness for aimbot detection from server-side FPS game logs. Our work extends one-class SVM to the timing-anomaly domain in MMORPGs, where the absence of labeled bot data at training time motivates the one-class formulation.

F. Affinity Propagation

Frey and Dueck [13] introduced Affinity Propagation as a message-passing clustering algorithm that does not require specifying the number of clusters a priori. Unlike K-means, it automatically identifies exemplars from the data, making it suitable for the irregular movement patterns observed in MMORPG environments where the number of distinct patrol routes is unknown.

III. System Architecture

The CDS is deployed alongside the game server and communicates with it via ZeroMQ (ZMQ) publish-subscribe messaging [14]. Fig. 1 illustrates the overall architecture.

Each player action is transmitted as a structured message containing an operation code (opcode) followed by action-specific data. The system distinguishes several types of events, including login, movement, position updates, packet transmission, and health-related actions. All player actions are logged with millisecond precision timestamps and unique identifiers. Due to the heterogeneous structure of the data, a non-relational database (MongoDB) is used for storing gameplay events, while login-related data is maintained in a relational database for efficient querying and aggregation. Vectors are normalized before being passed to the detection modules. Detection modules operate asynchronously and do not block game packet processing. The enforcement recommendation (block, alert, or no action) is returned to the game server via a separate ZMQ channel. The CDS supports two complementary operating modes, analogous to the IDS/IPS distinction in network security [15]. In *Detection mode* (used by all modules except WrongPackets), events are analyzed post-hoc: the game server processes each packet immediately and the CDS evaluates the resulting action vector asynchronously without introducing any latency. In *Prevention mode* (used exclusively by the WrongPackets module), the game server holds each incoming packet on a mutex and waits for a binary allow/drop decision from the CDS before forwarding the packet to game logic. This inline blocking is justified only for malicious packet injection, where the cost of processing a single anomalous packet may immediately destabilize the server; the latency overhead is acceptable because the one-class SVM inference for packet feature vectors completes within the game engine's own packet-handling timeout.

IV. Data Representation

Each player interaction is represented as a structured event containing both temporal and contextual attributes. Depending

on the event type, the recorded data may include position coordinates, packet size, theoretical movement speed, health state, or login metadata.

Formally, each event can be defined as:

$$e_i = (t_i, u_i, \mathbf{a}_i)$$

where t_i denotes the timestamp, u_i is the unique player identifier, and $\mathbf{a}_i$ represents the event-specific attribute vector. The system processes multiple event streams (login events, movement and position updates, packet transmission events, and health/healing actions); each stream is transformed into a dedicated feature vector space tailored to a specific anomaly detection task.

V. Detection Algorithms

A central design choice of the CDS is to rely on machine learning rather than fixed rules. While a naive threshold (e.g., "heal within 50 ms") can catch the simplest bots, fixed rules do not generalize across heterogeneous players, character classes, and game zones, and they are trivially evaded by attackers who tune their tools just below the threshold. The one-class and clustering formulations learn the natural variability of legitimate play per module, and their outputs are combined by the DecisionMaker meta-layer (Sec. V-D), which adapts to evolving cheat patterns through enforcement feedback.

A. One-Class SVM for Temporal Anomaly Detection

The AutoHeal, WrongPackets, WrongTeleports, and Logins modules are built on one-class SVM [11] with a Gaussian (RBF) or linear kernel, selected per module by cross-validation on held-out data.

For the AutoHeal module, the feature vector encodes the time intervals between health-loss events and healing activations observed for each player. Human players exhibit variable reaction times (in our data, mean 144.9 ms with a standard deviation of 45 ms) constrained by perception and motor latency. Automated healers react in under 50 ms (mean 28.3 ms). Figure 3 shows the histogram of healing reaction delays. After normalization, the one-class SVM is trained exclusively on human gameplay data. At inference time, vectors that fall outside the learned hypersurface are classified as anomalous.

TABLE I
HEALING REACTION DELAY STATISTICS

Group	N	Mean [ms]	Std [ms]	Var	Median [ms]
Bot	50	28.28	4.50	20.29	28.00
Manual	50	144.90	45.00	2025.36	143.00

Formally, given training set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of human-labeled feature vectors, the one-class SVM finds a function f such that $f(\mathbf{x}) \geq 0$ for normal data and $f(\mathbf{x}) < 0$ for anomalies, minimizing the volume of the decision region while maximizing the margin from the origin in the kernel feature space.

VI. Packet Anomaly Detection

Detection of invalid or malicious packets is based on statistical analysis of packet characteristics, including size and inter-arrival times.

For each packet type, a dedicated One-Class SVM classifier is trained using historical data collected over a predefined time window. The feature vector includes:

- packet size,
- time intervals since previous packets of various types,
- frequency of packet occurrences within a fixed time window.

Time-related features are capped at a maximum threshold (e.g., 15 seconds) to handle missing or infrequent events. Prior to training, all features are normalized.

During inference, incoming packets are evaluated against the trained model. Packets classified as anomalous are excluded from further training to prevent model contamination.

A. Affinity Propagation for Movement Fingerprinting

The AutoExp module addresses exp-bots that patrol fixed routes in the game world. Unlike rule-based detectors that look for repeated quests or fixed grinding spots, our approach targets the spatial *concentration* of movement waypoints, which is robust to the specific patrol logic of a given cheat tool. Affinity Propagation [13] is applied to historical player position vectors $(x, y, \textit{sub_world_id})$ collected over a 7-day sliding window per player.

The algorithm constructs three matrices: a similarity matrix S (Euclidean distance between position pairs, negated), a responsibility matrix R, and an availability matrix A. Iterative message-passing updates R and A until cluster exemplars stabilize. Positions within a maximum radius of 256 game units are considered for clustering to limit computational cost in large open-world environments.

For detection, the CDS examines the last N reported movement waypoints of the player under scrutiny, discarding micro-movements (distance below a threshold) to avoid noise. Three nested clusters of increasing radius are computed. A high ratio of points in the smallest cluster relative to the larger clusters indicates highly concentrated movement around a small set of repeated waypoints - the hallmark of automated patrolling. A concentration ratio exceeding an empirically determined threshold triggers an alert.

B. DecisionMaker: Multilayer Perceptron Meta-Decision Layer

Individual detection modules produce binary signals (anomaly detected / not detected) with associated confidence scores. These signals are noisy: each module has its own false positive rate, and cheaters may partially trigger multiple modules simultaneously.

The DecisionMaker is a multilayer perceptron trained on labeled enforcement outcomes. Its input feature vector is computed per character and contains the proportion of blocking signals raised by each detection module (AutoExp, AutoHeal, WrongPackets, WrongTeleports) over the player's last 100

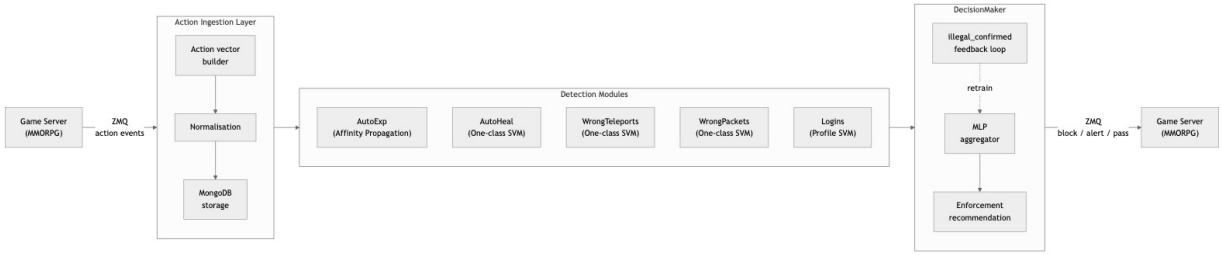


Fig. 1. CDS architecture: Game Server feeds player action events via ZMQ into the Action Ingestion Layer (vector builder, normalization, MongoDB). Five detection modules pass signals to the DecisionMaker MLP, which returns enforcement recommendations via ZMQ.

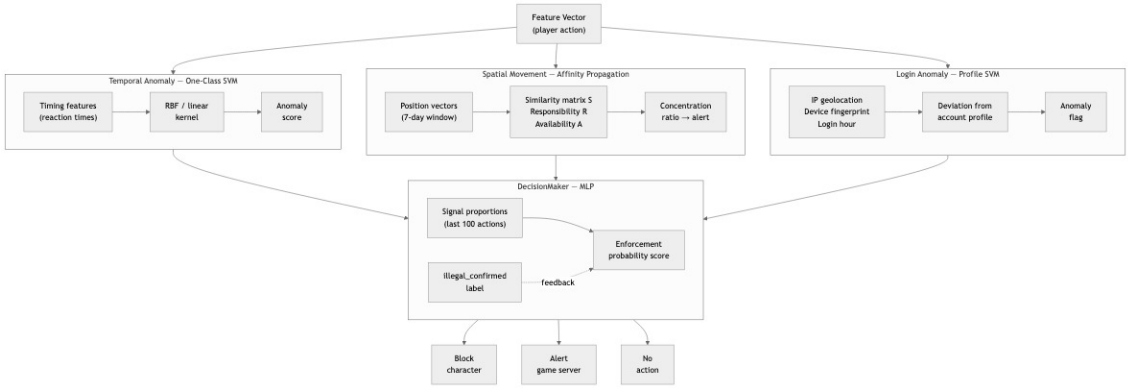


Fig. 2. Detection pipeline: a feature vector is passed simultaneously to three module groups (temporal anomaly SVM, spatial movement Affinity Propagation, and login profile SVM). The DecisionMaker MLP aggregates signals and applies a confirmed-enforcement feedback loop to recommend block, alert, or no action.

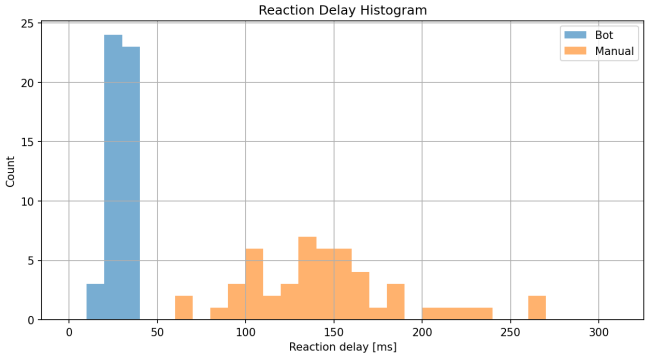


Fig. 3. Human vs auto healing

actions; the target variable is a binary label, *illegal_confirmed*, indicating whether a past enforcement decision was subsequently confirmed as correct by a game master. Because a minimum history of decisions is required, the layer is not activated at the very start of a player's session. The output is a probability score used to recommend enforcement at three levels: no action, notify the game server, or block the character. The feedback loop, in which confirmed enforcement labels flow back into the training set, allows the DecisionMaker to adapt to evolving cheat patterns over time.

C. Login Anomaly Detection

The Logins module builds a per-account behavioral profile from historical login records: typical login hours (encoded as cyclical features), IP geolocation (country, city), and device fingerprint (user-agent hash). At each new login attempt, the one-class SVM evaluates the deviation from the account's profile. Significant deviations trigger an alert to the game server, which may respond with additional authentication steps or character asset protection.

VII. Feature Vector Construction

Each detection module operates on a dedicated feature vector derived from the raw event stream, normalized to $[0, 1]$ prior to training for numerical stability across heterogeneous feature scales. The **AutoExp** vector is built from the player's historical position records $(x, y, \mathit{sub_world_id})$ and the *move_type* identifier; micro-movements below a configurable distance threshold are discarded before Affinity Propagation derives cluster exemplars (Sec. VI-A). The **WrongTeleports** vector captures the spatial plausibility of each movement: source and destination positions, the *theoretical_speed* computed by the game server from the character's attributes, and the *delta_distance* between consecutive reported positions; positions exceeding the maximum displacement feasible for the elapsed interval are anomaly candidates. The **AutoHeal** vector encodes the reaction delay Δt between an HP-loss

event and the subsequent heal, the HP delta, and, crucially, the variance of the health percentage over the last 50 heals; this variance feature means that a bot configured with an artificial delay still differs distributionally from a human, whose timing is intrinsically variable. The **WrongPackets** vector is built per opcode and includes *packet_size*, the elapsed time since the most recent packet of each monitored type (capped at 15 s), and per-type packet frequency in a fixed window; packets flagged as anomalous are excluded from subsequent model updates to prevent contamination.

The **Login** vector is constructed for each game-entry event and enriched with GeoIP information. It comprises: network and geolocation features (`ip_asn`; country, continent and city identifiers; relative login frequencies per IP, ASN, country, continent and city); temporal and behavioral features (the delay between character-server login and game entry, minute of day, per-segment login frequency, and a first-login-in-this-part-of-day indicator); historical consistency features (whether the previous login shared the same IP or ASN, the number of accounts per IP or network interface, and first-occurrence indicators for IP, ASN, interface, computer, city, country and continent); and device-related features (computer and network-interface identifiers and their reuse statistics). Two binary targets are recorded: `hack` (account compromise) and `bot` (automated behavior).

VIII. Experimental Evaluation

A. Experimental Setup and Dataset

The CDS was evaluated on a controlled MMORPG testbed based on a commercial PvE/PvP role-playing title that, for confidentiality and intellectual-property reasons, cannot be named; the title features open-world movement across sub-worlds, combat, healing, and a network protocol with more than 40 packet opcodes. Two complementary data sources were used. First, *production data* drawn from the live operation of the game provided 269,466 real login records; the `hack` and `bot` labels for these records were assigned by professional game masters who actively observed and analyzed player behavior and confirmed violations. Second, *controlled ground-truth* was produced on a test deployment of the same game by running custom cheat tools in parallel with a human-controlled client performing the equivalent task: an auto-walker patrolling four fixed locations (AutoExp), an auto-healer triggering below a health threshold (AutoHeal), a modified client emitting illegal double-jump packets (WrongPackets), and movement manipulation (WrongTeleports). This parallel capture yields exact per-action labels. Each module was trained and tested on disjoint subsets; Table II reports the train/test split sizes. The primary evaluation metric is the F-score, complemented by precision and recall where confusion matrices were recorded.

B. Detection Performance

Table II reports detection performance for all five cheat categories. All modules exceed the project acceptance criterion of F-score ≥ 0.80. For the AutoHeal and Logins modules the confusion matrices were recorded directly: every bot action was flagged and no legitimate action was flagged, giving exact precision and recall of 1.00. For the AutoExp, WrongTeleports, and WrongPackets modules the R&D evaluation logged only the F-score over class-balanced test sets. On a class-balanced set the F-score bounds the symmetric per-class error rate, so the implied error is approximately $1-\mathrm{F}$: about 16% for AutoExp, 19% for WrongTeleports, and 12% for WrongPackets. These non-negligible single-module error rates are precisely why the CDS never auto-bans on an individual module signal: final enforcement is delegated to the DecisionMaker meta-layer, which combines module outputs with confirmed enforcement history and thereby controls the rate of penalizing legitimate players.

TABLE II
Detection Performance by Cheat Category

Module	Cheat type	Train/Test	P	R	F
AutoExp	Exp bot	8k / 4k	–	–	0.84
WrongTeleports	Teleport	8k / 4k	–	–	0.81
AutoHeal	Auto-heal	10k / 1,053	1.00	1.00	1.00
WrongPackets	Injection	8k / 2k	–	–	0.88
Logins	Hijacking	30k / 6k	1.00	1.00	1.00

Precision and recall are reported where confusion matrices were recorded directly. For the remaining modules only the F-score was logged, over class-balanced test sets; the implied symmetric per-class error is approximately $1-\mathrm{F}$ (see text).

TABLE III
Server CPU Overhead (100 Concurrent Players, 10-Minute Test)

Condition	Mean CPU (%)
Without CDS	19.42
With CDS	19.40
Difference	-0.02 p.p. (within noise)

C. CPU Overhead

Table III reports server CPU utilization during a 10-minute test with 100 concurrent players. The measured difference (-0.02 p.p.) lies well within the run-to-run measurement noise of the non-deterministic server and should be read as "no measurable overhead" rather than a genuine decrease. The operationally meaningful figure is the extended integration test below.

In an extended integration test with 1,000 concurrent players and both the CDS and network-monitoring module active, CPU load increased by only 0.94 percentage points compared to the baseline – well within the project specification of < 10 p.p. This positive, clearly-resolved increment confirms that the asynchronous ZMQ-based integration scales without materially burdening the game server.

D. Discussion

The one-class SVM approach is particularly robust for the AutoHeal case because bot reaction time distributions are distinctly separated from human distributions, resulting in a near-perfect decision boundary. Even when a bot is configured to

introduce an artificial delay before healing, the variance of the health-percentage feature over the last 50 heals remains lower than the intrinsic variability of human reactions, preserving separability; a systematic study of adaptive, delay-randomizing bots is left for future work. The Affinity Propagation approach for movement analysis performed well (F-score 0.84) but showed sensitivity to the concentration ratio threshold: a lower threshold reduces false negatives at the cost of more false positives against legitimate players who happen to repeat patrol routes. Threshold calibration should be done per game zone and adjusted after live deployment.

The DecisionMaker MLP requires a minimum of 100 confirmed enforcement decisions before its output is meaningful. This cold-start period should be managed by relying exclusively on individual module thresholds during initial deployment.

IX. Conclusion

This paper presented the Cheat Detection System (CDS), a server-side machine learning framework for cheat detection in MMORPG games. The system combines one-class SVM for temporal anomaly detection, Affinity Propagation for spatial movement fingerprinting, and a multilayer perceptron meta-decision layer. Evaluated on five cheat categories in a controlled MMORPG environment, the system achieved F-scores between 0.81 and 1.00 with negligible CPU overhead. The modular design and ZeroMQ-based integration allow deployment alongside existing game server infrastructures without architectural modifications.

Future work will investigate online learning to allow the CDS to adapt to new cheat patterns without manual retraining, and explore graph neural network approaches for richer behavioral modeling across multiple interacting players.

Acknowledgment

This work was carried out as part of the project "Zestaw innowacyjnych narzędzi wspierających rozwój i utrzymanie gier o tematyce MMORPG" (POIR.01.02.00-00-0123/20), co-funded by the National Centre for Research and Development (NCBR), Poland, under the Intelligent Development Operational Programme 2014–2020.

References

[1] Newzoo, "Global Games Market Report," 2023. [Online]. Available: https://newzoo.com

[2] BattleEye Innovations, "BattleEye Anti-Cheat," product documentation, 2023. [Online]. Available: https://www.battleye.com

[3] J. Tao, J. Xu, L. Gong, Y. Li, C. Fan, and Z. Zhao, "NGUARD: A game bot detection framework for NetEase MMORPGs," in *Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pp. 811–820, 2018. https://doi.org/10.1145/3219819.3219925

[4] J. Xu, Y. Luo, J. Tao, C. Fan, Z. Zhao, and J. Lu, "NGUARD+: An attention-based game bot detection framework via player behavior sequences," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 6, pp. 1–24, 2020. https://doi.org/10.1145/3399711

[5] J. P. Pinto, A. Pimenta, and P. Novais, "Deep learning and multivariate time series for cheat detection in video games," *Mach. Learn.*, vol. 110, pp. 3037–3057, 2021. https://doi.org/10.1007/s10994-021-06055-x

[6] X. Qi, J. Pu, S. Zhao, R. Wu, and J. Tao, "BEAT: Behavior evaluation and anomaly tracking, game bot detection framework in RPG games," in *Proc. 6th Int. Conf. Algorithms, Computing and Artificial Intelligence (ACAI)*, 2023. https://doi.org/10.1145/3639631.3639683

[7] Y. Su, D. Yao, X. Chu, W. Li, J. Bi, S. Zhao, R. Wu, S. Zhang, J. Tao, and H. Deng, "Trajectory-based mobile game bots detection with Gaussian mixture model," in *Proc. Int. Conf. Artificial Neural Networks (ICANN)*, 2022. https://doi.org/10.1007/978-3-031-15934-3_38

[8] M. L. Han, B. I. Kwak and H. K. Kim, "Cheating and detection method in massively multiplayer online role-playing game: Systematic literature review," *IEEE Access*, vol. 10, pp. 49038–49060, 2022. https://doi.org/10.1109/ACCESS.2022.3172110

[9] M. Chung, K. H. Yoo, J. Kim, and H. K. Kim, "Game bot detection approach based on behavior analysis and consideration of various play styles," *ETRI J.*, vol. 35, no. 6, pp. 1058–1067, 2013. https://doi.org/10.4218/etrij.13.2013.0049

[10] H. Alayed, F. Frangoudes, and C. Neuman, "Behavioral-based cheating detection in online first person shooters using machine learning techniques," in *Proc. IEEE Conf. Computational Intelligence and Games (CIG)*, pp. 1–8, 2013. https://doi.org/10.1109/CIG.2013.6633617

[11] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, 2001. https://doi.org/10.1162/089976601750264965

[12] Ye, N. and Chen, Q., "An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems," *Qual. Reliab. Eng. Int.*, vol. 17, no. 2, pp. 105–112, 2001. https://doi.org/10.1002/qre.392

[13] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007. https://doi.org/10.1126/science.1136800

[14] iMatix Corporation, "ZeroMQ: Distributed Messaging," 2023. [Online]. Available: https://zeromq.org

[15] A. S. Ashoor and S. Gore, "Difference between Intrusion Detection System (IDS) and Intrusion Prevention System (IPS)," in *Advances in Network Security and Applications (CNSA 2011)*, Communications in Computer and Information Science, vol. 196, Springer, Berlin, Heidelberg, 2011, pp. 497–501. https://doi.org/10.1007/978-3-642-22540-6_48

[16] E. Heijnen (Dark Byte), "Cheat Engine," open-source memory scanner and debugger, 2019. [Online]. Available: https://www.cheatengine.org

[17] J. Woo and H. K. Kim, "Survey and research direction on online game security," in *Proc. Workshop at SIGGRAPH Asia (WASA 2012)*, Singapore, 2012, pp. 19–25. https://doi.org/10.1145/2425296.2425300