Game Bot Identification Based on Manifold Learning*

Kuan-Ta Chen[1], Hsing-Kuo Kenneth Pao[2], and Hong-Chung Chang[2]
[1]Institute of Information Science, Academia Sinica
[2]Dept. of Computer Science & Information Engineering, National Taiwan Univ. of Science & Technology
ktchen@iis.sinica.edu.tw, {pao,m9515069}@mail.ntust.edu.tw

ABSTRACT

In recent years, online gaming has become one of the most popular Internet activities, but cheating activity, such as the use of game bots, has increased as a consequence. Generally, the gaming community disapproves of the use of game bots, as bot users obtain unreasonable rewards without corresponding efforts. However, bots are hard to detect because they are designed to simulate human game playing behavior and they follow game rules exactly. Existing detection approaches either disrupt players' gaming experiences, or they assume game bots are run as standalone clients or assigned a specific goal, such as aim bots in FPS games.

In this paper, we propose a manifold learning approach for detecting game bots. It is a general technique that can be applied to any game in which avatars' movement is controlled by the players directly. Through real-life data traces, we show that the trajectories of human players and those of game bots are very different. In addition, although game bots may endeavor to simulate players' decisions, certain human behavior patterns are difficult to mimic because they are AI-hard. Taking Quake 2 as a case study, we evaluate our scheme's performance based on real-life traces. The results show that the scheme can achieve a detection accuracy of 98% or higher on a trace of 700 seconds.

Keywords

Cheating, Classification, Isomap, kNN, Online Games, SVM, Trajectory

1. INTRODUCTION

In recent years, online gaming has become one of the most popular Internet activities. However, as the population of online gamers has increased, game cheating problems, such as the use of *game bots*, have become more serious. Game bots are automated programs with or without artificial intelligence that help players enhance, accelerate, or bypass some routines in the game. For example, in MMORPGs (Massively Multiplayer Online Role Player Games), players can save a great deal of time by using bots to perform repetitive tasks, such as slashing low-level monsters, or fishing in a river to master the avatar's fishing skills. Meanwhile, in FPS (First-Person Shooter) games, users can employ bots to play in place of themselves in order to get high scores and gain a reputation in the community.

Generally, the gaming community disapproves of the use of game bots, as bot users obtain unreasonable rewards without corresponding efforts. However, game bots are hard to detect because *they are designed to simulate human game playing behavior and they follow game rules exactly.* Some bot detection studies [8, 27] propose using CAPTCHA tests during a game to determine whether an avatar is actually controlled by a person. Although this method is effective, it disrupts the game play and degrades players' feelings of immersion in the virtual world [10, 16]. Alternatively, passive detection approaches, such as schemes based on traffic analysis [4] and schemes based on avatars' shooting accuracy in FPS games [28], have been proposed. However, the former approach assumes that a game bot works as a standalone client, while the latter is only suitable for detecting aim bots in shooting games.

In this paper, we propose a general approach for *all genres of games in which players control the avatar's movements directly.* Taking the *avatar's movement trajectory* as the input, we adopt a learning method for bot detection. The rationale is that *the trajectory of an avatar controlled by a human player is hard to simulate.* Players control the movement of avatars based on their knowledge, experience, intuition, and a great deal of environmental information provided in the game. Since human decisions are sophisticated and depend on multitudinous observable and unobservable factors, how to model and simulate realistic movements is still an open question in the AI field.

To exploit the complexity of an avatar's trajectory for bot detection, it is necessary to tackle the high dimensionality of the derived information because the trajectory contains a long series of two-dimensional or three-dimensional coordinates (depending on whether the game is 2D or 3D) over time. To analyze such inputs, we adopt a *manifold*

*This work was supported in part by Taiwan Information Security Center (TWISC), National Science Council under the grants NSC 97-2219-E-001-001 and NSC 97-2219-E-011-006. It was also supported in part by Taiwan E-Learning & Digital Archives Program (TELDAP), National Science Council under the grants NSC 96-3113-H-001-010 and NSC 96-3113-H-001-012.

NetGames'08, Worcester, MA, USA

learning framework that transforms the dataset in the high-dimensional space into a dataset in a low-dimensional space, so that the data representation in the latter space can be utilized for bot detection. The objective is to solve the so-called *curse of dimensionality* that usually occurs in a dataset of high dimensionality. We adopt Quake 2 as our case study because it is a classic and popular FPS game, and many real-life human traces are available on the Internet. Therefore, we can use such traces to validate our proposed scheme.

The contribution of this paper is two-fold. 1) We propose using a manifold learning framework to detect game bots based on avatars' trajectories. It is a general model that can be applied to any game in which avatars' movement is controlled by the players directly. 2) Based on real-life human traces, the performance evaluation results show that the scheme can achieve a detection accuracy of 98% or higher on a trace of 700 seconds. Because it is difficult to simulate human players' logic and determine how they control game characters, we believe this approach has the potential to distinguish between human players and automated programs and thus merits further investigation.

The remainder of this paper is organized as follows. Section 2 contains a review of related work. In Section 3, we introduce our game case study, Quake 2, and describe the game trace collection methodology. In Section 4, we consider the distribution of discriminative features and propose identification schemes based on a manifold learning framework. Section 5 contains an evaluation of our approach's performance. We also analyze the results for traces of different length. Then, in Section 6, we summarize our conclusions.

2. RELATED WORK

In recent years, a number of studies have employed machine learning techniques to detect or simulate game bots in online games. For example, Yeung et al [29] proposed using the a *dynamic Bayesian network* (DBN) to model the aiming accuracy for *aimbot* detection in first-person shooter games. In the DBN, the aiming accuracy depends on whether the player is cheating, whether the player or the target is moving, the aiming direction, and the distance between the player and the target. Since the possibility that a player is cheating is a random variable, the authors modeled it by a Markov chain. The model can detect cheaters with a high degree of accuracy, but it can only be applied to aimbots. Kim et al [13] proposed detecting *auto programs* in MMORPGs [13] based on the window events, which are generated by a player's key strokes, mouse button clicks, and mouse movements. The events are collected during game play and used to compute statistics like the mean and standard deviation of the counts of certain events at regular intervals. Then, various classification schemes, such as the decision tree, the k-NN classifier, the multilayer perceptron network, and the naïve Bayesian classifier, are used to determine whether automated programs are being used. Because of the high regularity exhibited by such programs, the window-event-based approach yields a decent performance irrespective of the classification method used.

Thurau et al [23] attempted to create human-like game agents with machine learning approaches. They classified the behavior of human players into two categories: *perceptions* and *reactions*. The former includes a player's environmental information like the avatar's position and the distance between the avatar and nearby opponents; the latter includes a player's actions, such as the avatar's movement velocity and direction. With the information from both categories the authors created automatic human-like game agents. A number of learning approaches have been exploited in a series of papers by the same research group, including self-organizing maps [24], manifold learning [22], Bayesian networks [25], and waypoint maps [21,23]. Equipped with these learning techniques, the proposed game agents proposed can imitate human behavior very well compared to traditional rule-based game agents. However, the manifold learning approach in [22] only performs a 3-D to 2-D mapping, which is not really a case of the curse of dimensionality. Instead, in this work, based on avatars' movement trajectories, we apply a manifold learning framework with more than 200 original dimensions to detect the use of game bots.

Figure 1: A screen shot of Quake 2.

3. DATA DESCRIPTION

In this section, we describe our case study game, Quake 2, and the procedures we used to collect the game traces.

3.1 Quake 2

Quake 2 is a famous FPS (First-Person Shooter) game that was developed by id Software [2]. As with FPS games generally, a player adopts the role of a particular character and shoots his enemies via the user interface shown in Fig. 1. Multiple players can participate in a game simultaneously, and they can cooperate to complete a mission. However, death-match games, in which each player tries to kill as many other participants as possible, are much more popular. Quake 2 was nominated as "The Best Game Ever" by PC Gamer in 1998, and went on to sell over one million copies [1]. One reason for the game's popularity is that it is easy to customize, and a large number of maps, player models, textures, and sound effects are available on the Internet. The game has been ported to many platforms other than PCs, for example, Nintendo 64, Playstation, Amiga PowerPC, and Xbox 360.

3.2 Human Traces

Quake 2 supports a game-play recording function that keeps track of every action and movement, as well as the status of each character and item, throughout the game. With a recorded trace, one can reconstruct a game and re-

view it from any position and angle desired with VCR-like operations. Players often use this function to assess their performance and combat strategies. Moreover, experienced players are encouraged to publish their game-play traces as teaching materials for novice gamers and thereby build a reputation in the community.

To ensure that our game traces represented the diversity of Quake players, we only used traces that players had contributed voluntarily. The traces were downloaded from the following archive sites: GotFrag Quake[1], Planet Quake[2], Demo Squad[3], and Revilla Quake Site[4]. We restricted the traces to the map *The Edge*, one of the most well-known levels in death-match play. At this level, each player's only goal is to kill as many other players as possible, until the time limit is reached. Because short traces contain little information, we only collected traces longer than 600 seconds.

3.3 Bot Traces

There are many game bots available for Quake 2. For this study, we selected three of the most popular bot programs for trace collection, namely CR BOT 1.14 [15], Eraser Bot 1.01 [7], and ICE Bot 1.0 [12].

To collect the game bot traces, we set up experiments on our own Quake server and ran a number of game bots to fight each other. The experimental setup was as follows:

1. In each game, 2–6 bots were selected at random to fight each other. Each session spanned 20 hours.
2. The game trace was recorded at the server using the `serverrecord` command.
3. The game's catch-the-flag mode was turned off, so the game bots kept fighting each other until the server shut down. The cheating mode was also disabled.
4. The AI levels of CR Bot and Eraser Bot were randomly set from 0 to 9 and 0 to 3 respectively.

We collected 1,306 hours of raw traces in total. Then, from each trace, we took the first 1,000 seconds, the middle 1,000 seconds and another 1,000 seconds near the end to compile our dataset. In total, we collected 143.8 hours of trace data, as shown[5] in Table 1. In CR Bot and Eraser Bot, all human players and bots were active most of time ($\geq 89\%$). There was less activity in ICE Bot because it often remained idle in some places waiting for an opportunity to ambush other players.

4. BOT IDENTIFICATION SCHEMES

We propose using two approaches for bot detection, namely, the kNN algorithm and the support vector machine (SVM) model [11, 26]. To improve the detection accuracy, the approaches can be combined with a dimension reduction (DR) technique called *Isometric Feature Mapping* or Isomap [20], a method that belongs to the category of manifold learning. Considering the machine learning framework for our problem, some labeled data is collected for model training. We treat bot traces as positive samples and the traces from human players as negative samples; thus, a binary classification problem is formed. Specifically, we compile a dataset of M samples $\{(\mathbf{x}_m, y_m), m = 1, \ldots, M\} = \{(x_{m1}, \ldots, x_{mN}; y_m)\}$ for the training process. Each data item is described by N attributes and $y_m \in \{0, 1\}$ is the class information.

Table 1: Trace Summary

	Name	No	Length	Total	Active
1	Human	282	1000 seconds	78.0 hours	89%
2	CR	75	1000 seconds	20.8 hours	89%
3	Eraser	102	1000 seconds	28.3 hours	92%
4	ICE	60	1000 seconds	16.7 hours	67%

Notations. A trace $\mathbf{s}$ is a series of locations in either a 2-D or 3-D space, i.e., $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_t, \ldots, \mathbf{s}_T)$ up to time T. Usually, T represents the effectiveness of the detection technique, or how quickly an alarm should be raised about a bot user. A step in a trace is the vector $\mathbf{s}_{t+1} - \mathbf{s}_t$, and, the Euclidean step size is denoted by $\|\mathbf{s}_{t+1} - \mathbf{s}_t\|$. We then consider the distribution of the step sizes, i.e., the frequency counts of the step sizes after discretization. The counts are collected in B bins as $(f_1, f_2, \ldots, f_B)$, based on the frequencies of the step sizes from 0 to a large number. We assume that frequency counts of step size 0 indicate periods of conversation, rest periods, hiding from intense fire, or waiting for the arrival of opponents. The frequencies then become the input vector for our machine learning framework. Next, we consider several classifiers that can be used for bot detection.

4.1 k Nearest Neighbors

The kNN algorithm is one of the oldest and most intuitive classification methods, and many applications continue to demonstrate its competitive performance compared to other classifiers (e.g. [19]). In kNN, the class label of a new trace is decided by the class labels of the traces surrounding it. One of the keys to the successful application of kNN is the choice of an appropriate metric. In our study, for two data points, i.e., two feature vectors, P and Q, we choose either the Euclidean distance or the Kullback-Leibler divergence [5]

$$d_{kl}(P, Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} ,$$

as the metric. In general, the KL divergence is not symmetric; thus, we would like to choose a symmetric version.

$$D_{kl}(P, Q) = d_{kl}(P, Q) + d_{kl}(Q, P) .$$

We can also use the Euclidean metric as

$$D_e(P, Q)^2 = \sum_x (P(x) - Q(x))^2 ,$$

the distance measure between two data points.

4.2 Support Vector Machines

[1] `http://www.gotfrag.com/quake/home/`
[2] `http://planetquake.gamespy.com/`
[3] `http://q2scene.net/ds/`
[4] `http://www.revilla.nildram.co.uk/demos-full.htm`
[5] We assume that the sections at the beginning, in the middle and near the end of a trace dissimilar and can thus be considered as different samples. In this way, we can create more useful data for the input of our learning scheme, though this preprocessing is not essential for our scheme to work properly.

Support Vector Machines (SVMs) are well suited for solving binary classification problems like bot detection. Theoretically, in a linear case, by selecting the separating hyperplane $\mathbf{w}^T\mathbf{x} + \mathbf{b} = 0$ that maximizes the margin between positive and negative samples, we can obtain a classifier that minimizes the *generalization error* [11,26]. Specifically, finding the optimal classifier is equivalent to minimizing a functional comprised of the training error term and the regularization term as follows:

$$\begin{aligned} \min_{(\mathbf{w},\mathbf{b},\xi)\in R^{N+1+M}} \quad & C\sum_{m=1}^{M}\xi_m + \tfrac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{s.t.} \quad & y_m(\mathbf{w}^T\mathbf{x}_m + \mathbf{b}) + \xi_m \geq 1 \\ & \xi_m \geq 0, \quad \text{for } m = 1, 2, \ldots, M, \end{aligned}$$

where ξ_m denotes the positive slack variables, and C is a positive parameter that controls the balance between the training error and the margin maximization term. In a nonlinear case, the kernel trick [18] can help us find a nonlinear separating surface between positive and negative samples. Several variants of the typical SVM model have been proposed; e.g., the smooth SVM (SSVM), which tries to solve an unconstrained minimization problem instead [14]. In this study, we use SSVM to evaluate our framework. We consider both the linear and the nonlinear versions.

4.3 Dimension Reduction using Isomap

Using kNN or SVM (SSVM) solely can give us the classification output, but the performance may be poor. This may be due to the *curse of dimensionality* (e.g., [3]) which degrades the generalization power, or to the heavy computation required in the training or prediction process. To resolve this problem, we use a feature extraction technique called Isomap to reduce the dimensionality of the original space to a space with lower dimensionality, where the relationships between data points can be identified easily. Isomap, which is a type of manifold learning method [17,20], assumes that the data is lying on a smooth manifold so that each local area can be approximated by a Euclidean space without much loss of information.

The goal of Isomap is to find a mapping from the original space to an intrinsic space in which, *locally*, it tries to maintain the neighborhood relationship between each pair of data points; however, *globally*, a geodesic distance between two points is substituted to describe their dissimilarity. Given the training data in the input space, the Isomap process can be divided into three steps:

1. Construct a relation graph by linking each pair of points that qualify as neighbors. The input is the pairwise Euclidean distances between data points and the output is the relation graph.

2. Use any shortest path algorithm to find the pairwise shortest path on the graph and record the length of that path as the pairwise distance (if the pair of points are not neighbors). The input is the graph from the previous step and the output is a pairwise $M \times M$ distance matrix D.

3. Take the distance matrix D as the input to apply *Multidimensional Scaling* (or MDS) [6] to find the global coordinates of all the data points in a new space. In this step, the dimensionality is reduced by choosing a small number, d, the dimensionality of the new space. We use an $M \times d$ matrix A to record the coordinates in the new space, where the row vector A_m contains the new coordinates of the m-th point.

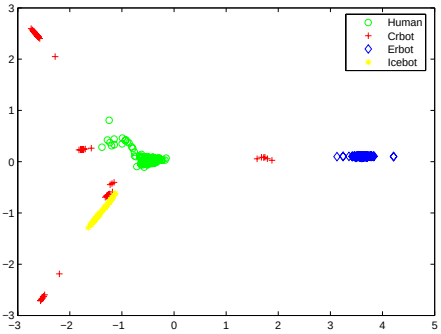


Figure 2: Data representation after dimension reduction by Isomap, where a point represents a trace of a human user (green circle) or from a bot (others). The x- and y-axes are the first and second principal coordinates [6] from Isomap. As the figure shows, the human data and the bot data are well separated. Classification in this space using kNN or SVM (or SSVM) can be performed with a high degree of accuracy.

To summarize the process, Isomap arranges the data points/traces in a low dimensional space so that the dissimilarities between the points can be determined by their Euclidean distances, as shown in Fig. 2. In other words, we can measure the dissimilarity between two points by their Euclidean distance in the low dimensional space; however, we may need to measure the much harder geodesic distance between them in the high dimensional space. In Isomap, the geodesic path (or the shortest path in a discrete case) is used to link each pair of data points. The length of the geodesic path between each pair of points is then calculated and used to describe the relationship between the points. In the last step, MDS is used to find the embedding so that the pairwise relations can be visualized.

Fig. 2 shows the results derived by applying Isomap[6]. The (green) circles indicate the traces of human users, while the others are obtained from several different bots. Among them, CR Bot (the cross symbols) has the highest variance, and the human players have the second highest variance. Meanwhile, the remaining traces, including Eraser Bot and ICE Bot, exhibit relatively low variance. Most importantly, data items with the same labels are clustered together, or in a few groups (as in CR Bot). However, such discriminative results can not be obtained if we use the well-known *Principal Component Analysis* (PCA) [9] method to perform the DR, as shown by the results in Fig. 3. After Isomap finds a low dimensional representation of the data, we can use any classification scheme, e.g., such as the kNN algorithm, to decide the label (either a bot or a human player) for a new trace. In the next section, we evaluate the detection performance of our approach and present the detailed clas-

[6]We only present data in 2-D for visualization purposes. In general, the detection or classification task is executed in the space of intrinsic dimensionality.

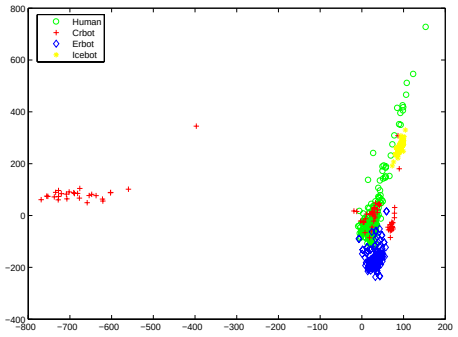


Figure 3: Dimension reduction by PCA, where the x- and y-axes represent the first and second principal components respectively. The points of human users and bots overlap, so they are not as distinguishable from each other as the result obtained by Isomap.

sification scheme.

5. PERFORMANCE EVALUATION

We consider four classification schemes for bot detection: (1) kNN with KL divergence as the metric, which is applied in the original space; (2) SSVM[7], which is also applied in the original space; (3) kNN, which is applied in the low dimensional space derived using Isomap and (4) Isomap followed by SSVM. Note that kNN is a naïve classification method, whereas SSVM is a sophisticated method. Our results demonstrate that, in terms of performance, kNN combined with DR (Isomap) is comparable to the other methods, while not to mention that kNN, compared to SVM (or SSVM) is a very efficient approach in terms of time complexity. To evaluate the performance, we use the detection error rate, which is measured by a ten-fold cross-validation procedure. In other words, the whole dataset is partitioned into ten subsets of more or less equal size, with stratification[8]. Then, nine of the subsets are used for training while the tenth is reserved for testing. The whole procedure is repeated ten times using different partitions for cross-validation to obtain an average result.

To compile the training data, we transform each trace into a distribution of step sizes, as mentioned in Section 4. The distribution is discretized and partitioned into $B = 201$ bins for each trace, with the corresponding probability function values or frequency counts; therefore, a data item is in a 201-dimensional space. Based on the dataset described in Table 1, there are 519 data items, of which 282 are positive samples (human) and 237 are negative samples (bots). To apply the Isomap procedure, the neighboring graph is defined by considering the five closest neighbors of each sample. Then, after the DR to a low dimensional space (of dimensionality 5), we either use an SSVM for classification, or we simply use kNN in the new space for classification with $k_2 = 13$. In the original space, the number of data items to be considered as neighbors should be limited ($k_1 = 5$) due to the possible curse of dimensionality; however, this can be relaxed to a larger number ($k_2 = 13$) in a dimension reduced space. We discuss the performance of each approach below.

[7]As mentioned previously, we adopt SSVM instead of SVM because it achieves a better performance.

[8]That is, the set is divided into several groups that contain similar percentages of positive and negative samples.

Table 2: The results of 10-fold cross-validation, which is repeated 10 times using six different classification methods. The results show the average false positive rates, false negative rates and error rates.

Classification Methods	FP(%) /	FN(%)	Err(%)
(A1) kNN	0.00 /	3.22	(1.45%)
(A2) Linear SSVM	1.07 /	0.95	(1.02%)
(A3) Nonlinear SSVM	1.43 /	0.35	(0.94%)
(B1) DR + kNN	0.00 /	1.87	(0.84%)
(B2) DR + Linear SSVM	0.88 /	16.68	(8.00%)
(B3) DR + Nonlinear SSVM	0.00 /	0.00	(0.00%)

5.1 Error Rates

We use the previously mentioned trace as input for the classification. Table 2 shows the performance of several classifiers. Both kNN and SSVM are applied with and without DR; and we use both the linear and nonlinear versions of SSVM. Most of the classification methods yield error rates[9] of less than 2%, while DR by Isomap and nonlinear SSVM achieve perfect classification results. Overall, the methods that employ Isomap for DR yield better results than the methods that do not use it. Moreover, the methods based on SSVM outperform those based on kNN.

5.2 Using Trajectories of Different Length

Since we want to detect bot users as early as possible, we can analyze the performance when only a partial input trace up to a certain time is given, rather than a whole sequence. As shown in Fig. 4, one method may be superior to another for inputs of different length, but the results are similar to those reported in the previous section:

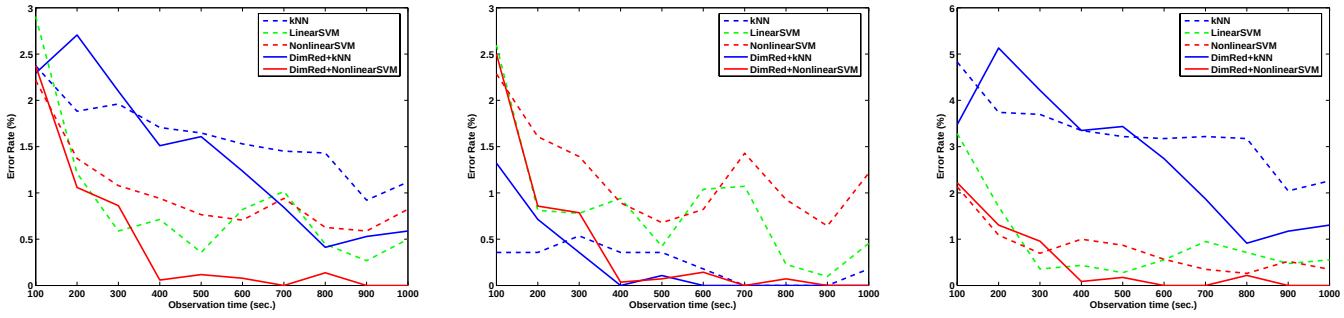
1. The methods that use DR (Isomap) outperform those that do not use it.

2. The methods based on SSVM perform better than those based on kNN, except those with false positive rates. In the latter case, although kNN-based methods often outperform those based on SSVM, the differences are marginal.

6. CONCLUSION

We have proposed a trajectory-based approach for detecting game bots. Specifically, we employ Isomap for dimension reduction, and then use kNN or SSVM to perform supervised classification. The performance evaluation results demonstrate that, based on real-life Quake 2 traces, our approach can achieve a detection accuracy of 98% or higher on a trace of 700 seconds. Because it is difficult to simulate human players' behavior when controlling game characters,

[9]DR combined with linear SSVM is not effective because the decision boundary tends to be nonlinear in a low dimensional space. For ease of visualization, we removed the result of Isomap combined with linear SSVM because it is not comparable to the results of the other methods.



(a) (b) (c)

Figure 4: The error rates with different trajectory observation times, measured in seconds: (a) error rate, (b) false positive rates, and (c) false negative rates. The results are similar to those given in the previous section: 1) the methods combined with Isomap outperforms those without it; and 2) the methods based on SSVM outperform those based on kNN, except those with false positive rates.

we believe that our method has the potential to distinguish between human players and automated programs, and thus merits further investigation.

7. REFERENCES

[1] Id Software: id History. http://www.idsoftware.com/business/history/.

[2] id Software, Inc. http://www.idsoftware.com/.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] K.-T. Chen, J.-W. Jiang, P. Huang, H.-H. Chu, C.-L. Lei, and W.-C. Chen. Identifying MMORPG bots: A traffic analysis approach. In *Proceedings of ACM SIGCHI ACE'06*, Los Angeles, USA, Jun 2006.

[5] T. M. Cover and J. A. Thomas. *Elements of Information Theory (2nd Ed.)*. Wiley-Interscience, July 2006.

[6] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling, Second Edition*. Chapman & Hall/CRC, 2000.

[7] R. R. Feltrin. Eraser Bot 1.01, May 2000. http://downloads.gamezone.com/demos/d9862.htm.

[8] P. Golle and N. Ducheneaut. Preventing bots from playing online games. *Computers in Entertainment*, 3(3):3–3, 2005.

[9] H. Hotelling. Analysis of a complex of statistical variables into principal components. *J. of Educational Psychology*, 24:417–441, 1933.

[10] S. Ila, D. Mizerski, and D. Lam. Comparing the effect of habit in the online game play of australian and indonesian gamers. In *Proceedings of the Australia and New Zealand Marketing Association Conference*, 2003.

[11] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[12] jibe. ICE Bot 1.0, 1998. http://ice.planetquake.gamespy.com/.

[13] H. Kim, S. Hong, and J. Kim. Detection of auto programs for MMORPGs. In *Proceedings of AI 2005: Advances in Artificial Intelligence*, pages 1281–1284, 2005.

[14] Y.-J. Lee and O. L. Mangasarian. Ssvm: A smooth support vector machine for classification. *Comput. Optim. Appl.*, 20(1):5–22, 2001.

[15] M. Malakhov. CR Bot 1.15, May 2000. http://arton.cunst.net/quake/crbot/.

[16] T. P. Novak, D. L. Hoffman, and A. Duhachek. The influence of goal-directed and experiential activities on online flow experiences. *Journal of Consumer Psychology*, 13(1):3–16, 2003.

[17] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[18] B. Schölkopf and A. Smola. *Learning with Kernels Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, USA, 2002.

[19] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 2006.

[20] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[21] C. Thurau and C. Bauckhage. Tactical waypoint maps: Towards imitating tactics in fps games. In M. Merabti, N. Lee, and M. Overmars, editors, *Proc. 3rd International Game Design and Technology Workshop and Conference (GDTW'05)*, pages 140–144, 2005.

[22] C. Thurau and C. Bauckhage. Towards manifold learning for gamebot behavior modeling. In *In Proc. Int. Conf. on Advances in Computer Entertainment Technolog (ACE'05)*, pages 446–449, 2005.

[23] C. Thurau, C. Bauckhage, and G. Sagerer. Learning human-like movement behavior for computer games. In *In Proc. 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB'04)*, pages 315–323. IEEE Press, 2004.

[24] C. Thurau, C. Bauckhauge, and G. Sagerer. Combining self organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In *Proceedings of the GAME-ON03 Conference*, pages 119–123, 2003.

[25] C. Thurau, T. Paczian, and C. Bauckhage. Is bayesian imitation learning the route to believable gamebots? In *In Proc. GAME-ON North America*, pages 3–9, 2005.

[26] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, November 1999.

[27] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

[28] S. Yeung, J. Lui, J. Liu, and J. Yan. Detecting cheaters for multiplayer games: theory, design and implementation. *Proc IEEE CCNC*, 6:1178–1182.

[29] S. F. Yeung, J. C. S. Lui, J. Liu, and J. Yan. Detecting cheaters for multiplayer games: theory, design and implementation. *Consumer Communications and Networking Conference, 2006. 3rd IEEE*, 2:1178–1182, 2006.