	Cahier des charges PIAD Détection de triche dans les jeux en temps réel		
Activité	PIAD de Master 1 d'Informatique	**Type**	CdC
Objet	Cahier des charges PIAD : Détection de triche dans les jeux en temps réel		
Destinataires	M. Pierre-Henri Wuillemin	**Pages**	11

Version	Statut	Auteur	Date	Description
1.0	Création	Larroque Stephen	30/01/2013	Création
1.02	MAJ	Larroque Stephen	31/01/2013	Ajout de quelques avantages dans la description de la solution proposée
1.1	MAJ	Larroque Stephen	15/02/2013	Renommé chapitre 1.2 Problématique → Motivation et ajout dans ch.1.2 de davantage de détails concernant les systèmes implémentés dans [3] et [4]
1.1.2	MAJ	Larroque Stephen	15/02/2013	Ajout de la contrainte de temps dans ch.2.8
1.1.3	MAJ	Larroque Stephen	21/02/2013	Ajout de l'aspect communautaire dans la triche dans ch.1.2

Table des matières

1 OBJECTIF DU PROJET

1.1 Présentation

Ce cahier des charges pour le projet de Détection de triche dans les jeux en temps réel présentera la problématique que tente de résoudre le problème, les objectifs principaux à atteindre, ainsi qu'une collection des fonctionnalités principales, et enfin une vue d'ensemble de leurs implémentations.

1.2 Motivation

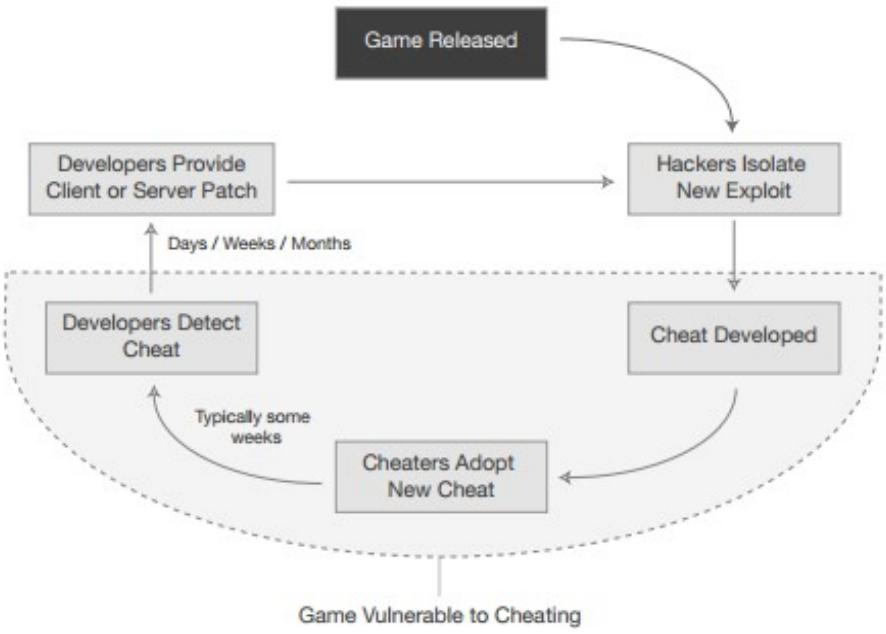
Les jeux en lignes représente une importante part de l'industrie vidéoludique et une importante source de revenus.

Outre les problèmes de gestion du réseau, le problème principal que rencontre les concepteurs de jeux en ligne est la triche.

Les tricheurs sont des joueurs qui utilisent un procédé afin de gagner un avantage décisif et inéquitable avec les autres joueurs, en général de façon non légitime par rapport aux règles du jeu (sauf pour les triches spécifiques au jeu[1] qui restent une zone floue et où la légitimité dépend des conventions éthiques entre joueurs et/ou du concepteur du jeu).

Les méthodes traditionnelles de détection des tricheurs nécessitent de lourdes infrastructures, sont peu efficaces[2], posent des problèmes concernant la vie privée des joueurs[2][4] et ils nécessitent un cycle de maintenance triche-défense réactionnaire par nature et très coûteux en temps et en personnel[2][5].

D'autre part, les tricheurs ont un avantage critique : ils ont un accès total à la machine qu'ils utilisent[4]. Les méthodes traditionnelles de détection des tricheurs, reposant essentiellement sur l'analyse de la machine cliente, sont nécessairement vouées à l'échec sur le long terme[2][4].



1. Illustration: cycle triche-défense, extrait du document [5]

Dans ce cadre, une approche complètement côté serveur semble être la plus appropriée. L'apprentissage machine semble être également un choix judicieux puisqu'il est très facile de générer des données en grand nombre, avec de nombreux critères, alors qu'il est difficile de définir explicitement les comportements de triche par une fonction ou des règles.

Certaines expérimentations antérieures à ce projet ont d'ailleurs été menées dans ce sens et ont été encourageantes[5][6][7], mais d'une part, la qualité, le nombre et les paramètres des jeux de test utilisés dans ces expérimentations ne sont pas suffisant pour permettre une évaluation précise des performances et de la viabilité de ces systèmes[5][6][7], et d'autre part, ces systèmes avaient beaucoup de contraintes notamment sur les types de triche qu'ils pouvaient détecter[5][6], et enfin, les faux positifs[5][6][7] (joueur honnête détecté comme tricheur) qui doivent à tout prix être évités. Enfin, la plupart de ces recherches ont été menées dans le but de réduire les triches par collusion dans les MMORTS (jeux de stratégie)[2][4][8], qui ont une dynamique tout à fait différente des jeux en temps réel à la première personne (FPS), et aucun code source n'est disponible.

D'autres types de systèmes de détection ont été proposé, notamment un système utilisant le langage Event-B afin de définir des règles logiques et procéder à un *model checking* comportemental[3], et un système d'analyse par machine learning du binaire du client de jeu en mémoire et de ses dépendances/hooks[4], automatisant ainsi le travail usuel des développeurs de signatures pour les systèmes anti-triche basé sur des base de signatures.

Ces deux systèmes ont tous deux des approches très intéressantes, mais ils ont néanmoins des lacunes flagrantes : le premier nécessite de définir manuellement des règles comportementales, ce qui est d'une part très difficile, très coûteux en temps humain et enfin peu robuste puisque modélisant uniquement les anomalies et triches connues ; le deuxième système, bien que théoriquement robuste car la détection se fait coté serveur, est en pratique peu fiable car il nécessite que les données coté client lui soient envoyés (ce qui peut être modifié par l'utilisateur de la machine cliente)[4] et ne peut détecter que les systèmes de triches de type « hook » qui s'attache au binaire du jeu, ignorant ainsi totalement les système de triche tierce tels que les *color bots* (triche qui tire automatiquement lorsqu'une couleur définie passe au milieu de l'écran et du curseur du joueur) et qui ne nécessitent aucune attache au binaire du jeu.

Une autre composante importante de ce problème et souvent ignorée est l'aspect communautaire: les tricheurs sont souvent des développeurs expérimentés, et les tricheurs forment de grandes communautés où ils partagent leurs connaissances et expérience. Il est donc vain de vouloir solutionner ce problème avec seulement une équipe restreinte de développeur, et avec un système « one-size-fits-all » (un seul système statique pour tous les problèmes de ce type). La déduction logique est que pour qu'un système anti-triche puisse être efficace sur le long terme, il doive lui aussi permettre une collaboration communautaire à grande échelle.

Il n'y a donc à l'heure actuelle aucune solution efficace pour résoudre le problème de la triche.

Dans le cadre de ce projet, nous nous intéresserons aux systèmes de triche basés sur la technologie : les systèmes de triche traditionnels d'une part (eg : *aimbot, wallhack, autoshoot*) et les *togglers* (tricheur activant leur système de triche de manière intermittente ou à des moments clés) d'autre part, qui sont beaucoup moins étudiés dans la littérature.

1.3 Solution proposée et objectifs principaux

Afin d'aborder tous les problèmes sus-mentionnés, nous posons comme hypothèse que les tricheurs montrent un ***comportement significativement différent*** des joueurs honnêtes. Si c'est le cas, les tricheurs pourraient être identifiables quelque-soit la méthode utilisée pour tricher.

Pour vérifier cette hypothèse, nous nous proposons de réaliser la conception d'un framework complètement opensource et côté serveur pour les systèmes de détection de triche, composé de briques logicielles :

- interchangeables : chaque partie du système (interface jeu, algorithmes, détection) peut être changée aisément ;
- génériques et réutilisables : entre plusieurs jeux, il suffit de générer des données ;
- collaboratives : esprit open source, les paramètres de détection peuvent être partagés et améliorés itérativement dans une communauté à grande échelle.

Ce système pourra fonctionner en mode online (les données sont traitées au vol) ou offline (les données sont collectionnées et traitées plus tard).

D'autre part, les données d'apprentissage collectées n'ont pas besoin d'être identifiées.

Les spécifications de ce système ont de nombreux avantages :

- Côté serveur : totalement transparent pour les utilisateurs, allège les ressources client et serveur puisque le système peut être placé sur un autre serveur, impossible à désactiver ni outrepasser, mise à jour facile par les administrateurs, etc.
- Préemptif: plus de cycle triche-défense, le système de détection de triche détecte les tentatives de triches même si les méthodes de triche ne sont pas encore connues (*zero-day* ou *private*).
- La méthode de triche importe peu, car tant que le comportement induit est significativement différent d'un joueur normal, la triche sera détecté.
- Le(s) développeur(s) peut travailler à son rythme pour étendre et améliorer le système de détection, et même agir en avance, plutôt que de ne pouvoir que seulement réagir après qu'une tentative de triche se soit déroulée.
- Aisé d'étendre le système puisque chaque brique logicielle est interchangeable.
- Générique et réutilisable : l'interface jeu et le système de détection de triche sont découplés, il est donc possible de réutiliser le système entre plusieurs jeux très différents, en n'ayant seulement qu'à programmer une nouvelle interface jeu pour le jeu cible.

- Open source : le système pourra être développé et amélioré par de nombreuses personnes, puisque chacun peut avoir accès au code source librement.
- Collaborative : puisque le système est open source et que les données collectées n'ont pas d'identifiants,il est donc possible de partager les jeux de données d'apprentissage, ou simplement les paramètres résultant de l'apprentissage, entre un groupe d'utilisateurs, ou même sur le web entier.
- Amélioration itérative généralisée : les points précédents pourront donc donner lieu à une amélioration itérative du système par un groupe restreint de développeurs, mais qui profitera à tous les utilisateurs puisque le système est complètement ouvert.

Les détails de l'implémentation de la solution proposée seront discutés dans les chapitres suivants ainsi que dans le plan de développement.

1.4 Contraintes techniques

- Le serveur de jeu étendu (interface avec le système de détection de triche) sera programmé en C/C++ à partir des sources du projet[9].
- Le framework pour les systèmes de détection de triche sera programmé en Python[10] et NumPy[11].
- L'algorithme d'Inférence Probabiliste Générale sera implémenté avec la librairie pyAgrum[12], qui est une interface Python pour la librairie C++ aGrUM[13].

Le système sera développé sous l'OS Windows et exécuté en situation réelle dans un environnement Linux, et sera probablement également compatible avec MacOSX.

2 DESCRIPTION DE LA SOLUTION PROPOSÉE

2.1 Implémentation de la solution proposée

Pour l'implémentation, nous utiliserons le jeu FPS (jeu de tir en temps réel à la première personne) multijoueurs et opensource OpenArena, qui est basé sur Quake 3 Arena et dont le code source est disponible et est bien documenté.

Le framework proposé sera implémenté avec les composants suivants :

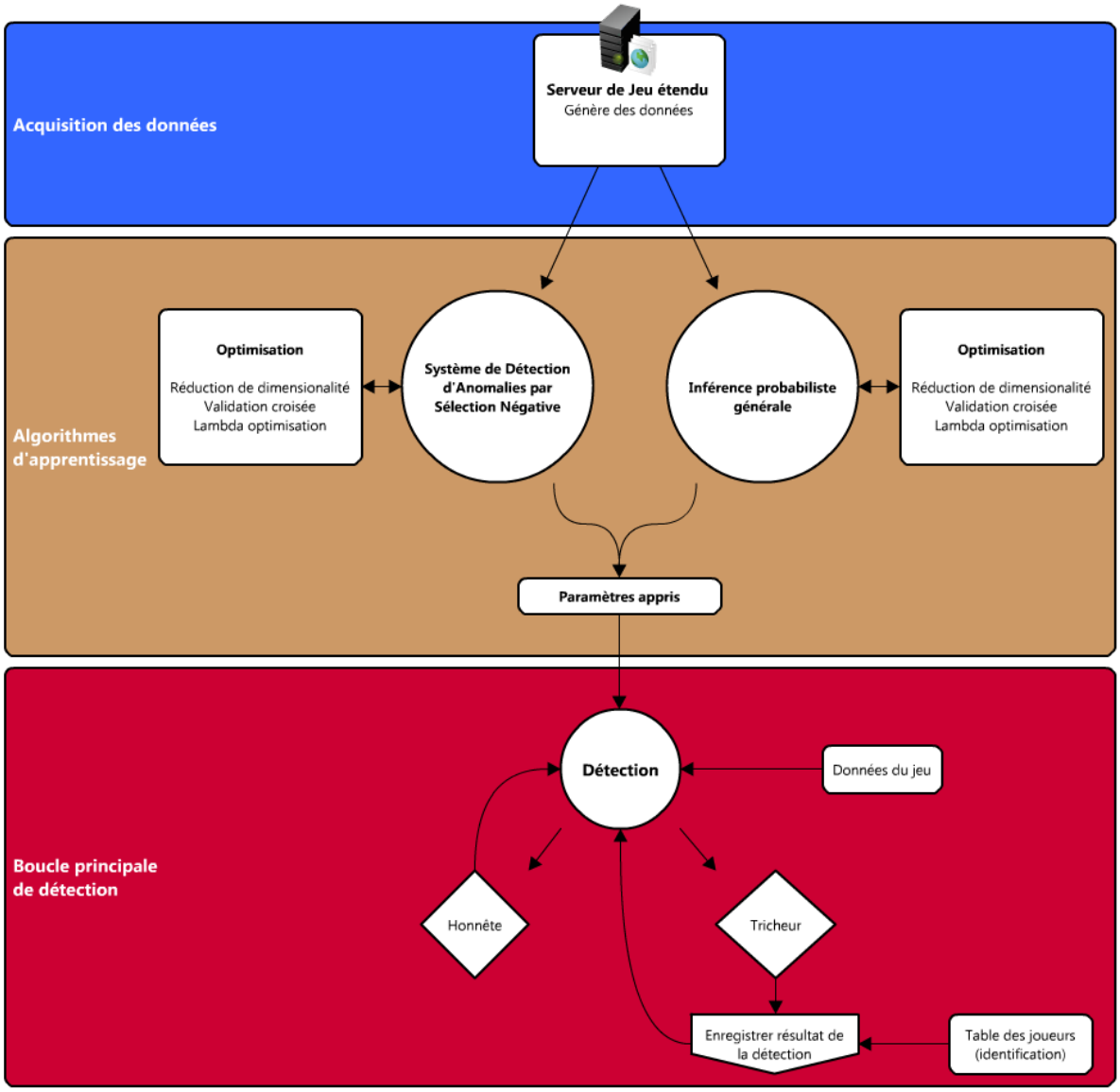
- **Un serveur de jeu OpenArena étendu**, comprenant :
  - l'ajout des critères d'apprentissage/détection,
  - l'interface avec le système de détection de triche (enregistrement des critères dans un fichier log dédié ; génération de la table des joueurs pour identification à la phase de détection),
  - des commandes spéciales pour déclarer un joueur comme honnête ou tricheur (pour l'étiquetage des données).
- **Algorithme1** : un système de détection d'anomalies non supervisé utilisant le paradigme de Système Immunitaire Artificiel avec Sélection Négative et à priori Gaussien uni/multivarié. Une approche similaire a été suggérée dans [5].

- **Algorithme2** : Un algorithme supervisé d'Inférence Probabiliste Générale en utilisant la librairie pyAgrum.
- **Boucle principale de détection** des tricheurs et de génération des rapports de triches.

Optionnellement :

- Un sous-framework d'optimisation :
  - algorithme de réduction de dimensionnalité automatique via la méthode PCA.
  - optimisation automatique des paramètres d'apprentissage des algorithmes par validation croisée (eg : lambda, limitation du taux F1 et de faux positifs, etc..).

Voici un schéma illustrant le fonctionnement du framework proposé :



Nous allons par la suite détailler plusieurs caractéristiques spéciales de notre framework.

2.2 Données de jeu et interframes

Les données de jeu seront composés d'un certain nombre de critères (tels que la position des joueurs, l'arme utilisée, etc... - Plus d'informations dans *Types de critères*).

Plus précisément, les données seront extraites des *interframes*, c'est-à-dire de la différence entre deux *frames* (état du monde dans le jeu) consécutives.

Ces données seront enregistrées dans un fichier texte (ou *log*), et seront utilisées à la fois pour l'apprentissage de nos modèles, et leur évaluation dans le cadre de la détection (biensûr, les données seront différentes).

2.3 Niveau d'abstraction des critères

Dans notre implémentation, nous avons choisi des critères d'apprentissage et de prédiction/détection de ***bas niveau*** (non traités). Éventuellement, des critères de haut niveau (agrégés et pré-traités) pourront être également utilisé si les critères bas niveaux ne sont pas suffisants.

2.4 Types de critères

Pour une efficacité et interopérabilité maximale de notre système, nous avons défini une échelle de quatre types de critères :

IDENTIFIEURS : ce type de critère permet d'identifier des qualités de l'état du monde dans le jeu, mais qui ne sont pas nécessaires à l'apprentissage, mais seulement à la détection. Par exemple : identifieur aléatoire du joueur, identifieur aléatoire de la partie jouée, numéro de frame, timestamp, etc...

Note : ce sont des identifieurs qui renseignent sur l'état du jeu, mais ne permettent pas d'identifier les joueurs (les identifieurs joueurs sont générés aléatoirement pour toute la durée de la connexion au serveur).

Note2 : ce type est le seul type de critère à ne pas être utilisé à l'apprentissage. À la détection, ils ne sont pas donnés en argument aux algorithmes de détection (qui prennent les mêmes données en détection qu'à l'apprentissage), mais par la suite pour implémenter la tolérance temporelle (voir plus bas) et enregistrer la description des événements de triche.

SPÉCIFIQUE HUMAIN : ces critères sont utilisé à l'apprentissage, et sont caractéristiques d'un comportement humain, et qui sont invariables aux règles du jeu (voire au jeu joué). Par exemple : le temps de réaction après avoir vu un adversaire, la fréquence que la mire de visée reste sur l'adversaire, la précision de la visée, etc...

SPÉCIFIQUE AUX RÈGLES DU JEU : ces critères d'apprentissage et détection définissent des paramètres de jeu qui peuvent changer si les règles du jeu changent. Par exemple : La quantité de dégâts infligés avec une arme, numéro de l'arme utilisée, mort conséquente au tir, etc...

SPÉCIFIQUE AUX LIMITES (PHYSIQUES) DU JEU : ces critères définissent des paramètres qui sont théoriquement limités par la physique du jeu. Nous pensons que ce type de critère n'est pas fiable et beaucoup trop variable car dépendant de limites théoriques qui peuvent être (et ont souvent été) dépassé par les joueurs sans aucune triche et de façon totalement légitime (comme les « tricks » et « trickjumping », qui sont des mouvements spéciaux qui permettent de dépasser les limites physiques théoriques du système). Ce type de critère ne sera donc pas utilisé dans notre implémentation. Exemples : vitesse du joueur, acceleration du joueur, etc...

2.5 Temporalité du système

Les concepteurs de ce système ont le sentiment que le manque de temporalité, limitation bien connue des algorithmes qui vont être implémentés, empêchent de prendre en compte des données qui pourraient être bien plus significatives si elles étaient replacées dans leur contexte temporel.

Afin de pallier à ce manque de temporalité, les concepteurs se proposent d'implémenter deux solutions simples :

- Des ***critères accumulateurs*** à l'apprentissage : ce sont des critères qui, au lieu de ne simplement contenir la valeur de l'événement en cours dans l'interframe, gardent en mémoire les événements passés et s'incrémentent.

  Par exemple, au lieu d'avoir un critère *"a tué un joueur"* on peut avoir *"a tué x joueurs d'affilée"* ou encore au lieu de *"a touché un adversaire"*, on aura *"a touché x fois un adversaire d'affilée sans rater"*

- Un ***concept de tolérance temporelle*** à la détection : puisque les algorithmes ne peuvent évaluer que des événements séparés (une interframe à la fois), il serait intéressant de pouvoir faire le lien entre plusieurs détection : si un joueur n'est détecté qu'une seule fois, cela ne devrait pas pour autant le sanctionner car cela peut être un événement rare isolé, mais tout à fait normal.
  À l'inverse, un joueur qui fréquemment procède à des événements qui sont détectés comme de la triche devrait être sanctionné.

  Le concept de tolérance temporelle induit ce comportement en gardant à tout moment une liste des joueurs et l'historique de leurs détections passées, comprenant le taux/probabilité de triche et la date de chaque détection. Ensuite, lors d'une nouvelle détection d'anomalie, le taux de cette détection est pondérée avec les taux des détections passées par rapport à l'intervalle de temps. Si le taux résultant dépasse un certain seuil fixé, alors le joueur est déclaré tricheur. Au final, plus l'intervalle entre les détection est grand, moins le joueur a de chance d'être un tricheur. Au contraire, plus l'intervalle entre les détection est petit, plus le joueur a de chance d'être un tricheur.

Ces deux concepts sont simples à implémenter et peu coûteux en terme de temps de réalisation et de computation.

2.6 Table des joueurs

Afin de permettre le partage opensource des résultats et la protection de la vie privée, les données de jeu ne contiennent pas de données d'identification (les seules données d'identification sont des identifieurs aléatoires qui ne permettent en aucune façon d'inférer l'identité du joueur à partir seulement de ces données).

Néanmoins, s'il est impossible d'inférer l'identité d'un tricheur, il est alors impossible de prendre des décisions.

Pour remédier à cela, le serveur de jeu étendu générera une *table des joueurs*, qui contiendra les données d'identification des joueurs (pseudonyme, adresse IP, date de connection) ainsi que la correspondance entre ces joueurs et leur identifieur aléatoire (*playerid*). Cette table pourra être optionnellement utilisée à la détection pour ajouter des informations supplémentaires dans les résultats.

2.7 Résultat de détection et décision

Le résultat de la détection sera par défaut le simple enregistrement dans un fichier texte de la description d'un événement de triche, avec l'identification du joueur si la table des joueurs est disponible (liste de tous les joueurs et paramètres d'identification, puisque les données d'apprentissage n'en possèdent pas). Un administrateur pourra alors vérifier les événements, consulter les ressources associées (capture d'écran, demo, log de jeu, etc...) afin de prendre une action à l'encontre de ce joueur.

Alternativement, le système pourrait également décider d'une action automatiquement, avec potentiellement une *gratification décalée* (l'action est prise avec un délai plus ou moins long après la détection) afin de détecter plus de tricheurs avant de les sanctionner tous à la fois, mais ceci ne sera pas implémenté dans le cadre de ce projet, la prise de décision étant en dehors du cadre de la détection de triche qui est fixé pour ce projet.

2.8 Contraintes de réalisation

- Le système devra être réalisé selon les paradigmes de la programmation orientée objet, afin que les composants puissent être interchangeables et réutilisables.
- Le système n'a pas de contrainte de performance puisque les analyses pourront être exécutées en offline.
- Le système devra s'efforcer de limiter au maximum le taux de faux positifs avant tout (la possibilité de détecter par erreur un joueur honnête comme étant un tricheur devant être évitée à tout prix).
- Une formalisation théorique et une implémentation fonctionnelle devront être implémentés en l'espace de 4 mois (de Janvier à début Mai).

3 COMPOSITION DU DÉLIVRABLE

À la conclusion de ce projet, les fichiers et documents suivant seront livrés :

- L'exécutable implémentant toutes les fonctionnalités prévues.
- Le code source entièrement commenté.
- La documentation complète du projet, à savoir :
  - Cahier des charges
  - Plan de développement
  - Dossier d'Analyse et de Conception
  - Rapport de tests décrivant les protocoles de test, leurs résultats complets et commentés
  - Rapport final
- Manuel d'installation
- Manuel de l'utilisateur
- Manuel du développeur

4 RÉFÉRENCES

[1] : 'Cheating in Online Video Games', by Savu-Adrian Tolbaru, 15/08/2011
[2] : 'Cheating and Virtual Crimes in Massively Multiplayer Online Games' by Rahul Joshi, Technical Report, RHUL-MA-2008-06, 15 January 2008
[3] : Tian, H., Brooke, P.J. and Bosser, A-G. (2012) 'Behaviour-based cheat detection in multiplayer games with Event-B', Lecture Notes in Computer Science, 7321/2012, Heideberg: Springer, pp.206-220
[4] : Fides: Remote Anomaly-Based Cheat Detection Using Client Emulation, by Edward Kaiser, Wu-chang Feng, Travis Schluessler, November 2009
[5] : Laurens, P.; Paige, R.F.; Brooke, P.J.; Chivers, H.; , "A Novel Approach to the Detection of Cheating in Multiplayer Online Games," Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on , vol., no., pp.97-106, 11-14 July 2007
[6] : Galli, L.; Loiacono, D.; Cardamone, L.; Lanzi, P.L.; , "A cheating detection framework for Unreal Tournament III: A machine learning approach," Computational Intelligence and Games (CIG), 2011 IEEE Conference on , vol., no., pp.266-272, Aug. 31 2011-Sept. 3 2011
[7] :Chapel, L.; Botvich, D.; Malone, D.; , "Probabilistic approaches to cheating detection in online games," Computational Intelligence and Games (CIG), 2010 IEEE Symposium on , vol., no., pp.195-201, 18-21 Aug. 2010
[8] :Mitterhofer, S.; Kruegel, C.; Kirda, E.; Platzer, C.; , "Server-Side Bot Detection in Massively Multiplayer Online Games," Security & Privacy, IEEE , vol.7, no.3, pp.29-36, May-June 2009
[9] :http://ioquake3.org/
[10] :http://www.python.org/
[11] :http://www.numpy.org/
[12] :https://forge.lip6.fr/projects/pyagrum
[13] :https://forge.lip6.fr/projects/agrum